IBM Software Group

SOA 开发第二步 – 设计SOA架构

IBM 软件部 李傲雷

e business on demand.

SOA 开发三步曲

第一步 业务分析员

业务目标
业务模型
业务需求

- 分析并记录现有的业务流程及其概念
- 优化改进业务流程

Rational. software
Rational Requisite Pro
Rational Software Modeler

第二步 架构设计师

服务模型
软件架构
企业架构

- 精练服务架构
- 设计服务
- 设计组件

Rational. software
Rational Software Architect

集成开发员

服务流程模型
服务组装模型

- 实现业务流程和组合应用
- 定义服务

WebSphere software
Websphere Integration Developer

第三步 开发人员

组件框架
组件代码

- 生成组件框架
- 编码实现组件

Rational. software
Rational Application Developer

资源组合　共享资产　公共流程

Rational. software
Rational Portfolio Manager　Rational ClearCase
Rational Unified Process　Rational ClearQuest

内容

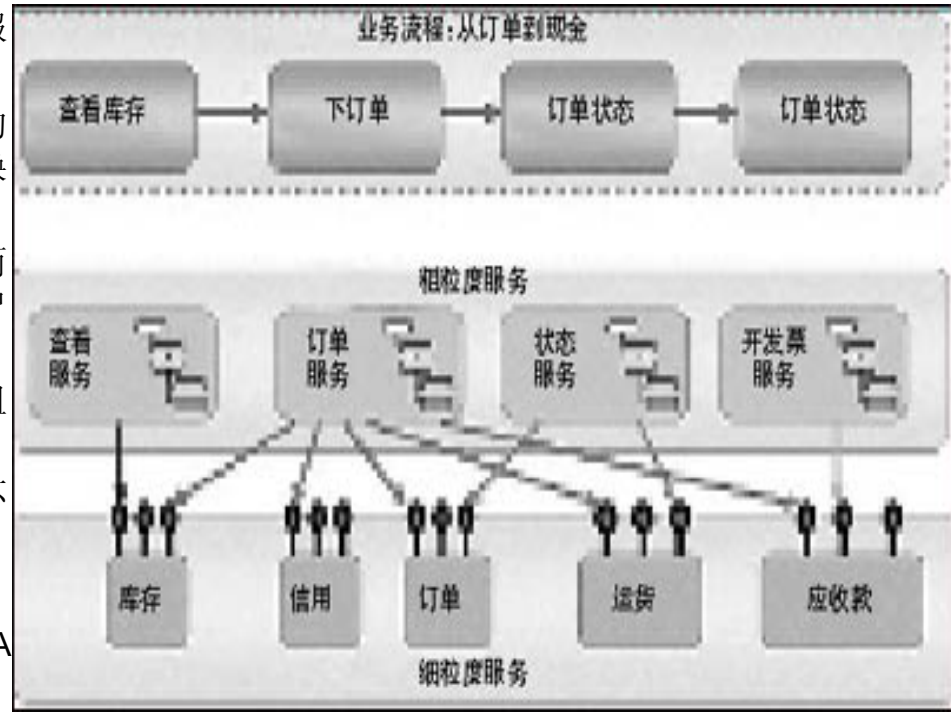
- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

SOA与架构设计师

- SOA对架构设计师的能力要求
  - 共通：专业知识、领域知识、管理知识、沟通能力、善于快速学习、洞察力、责任心……
  - 特别：深刻理解SOA的业务敏捷性、服务识别的能力、粒度的把握、组合与划分的技巧……
- SOA对架构设计师的职责要求
  - 充分理解商业需求和非功能性服务水平
  - 识别并降低各种技术风险
  - 可用性、安全性、可重用性、可扩展性、可管理性、可维护性、可靠性、性能、容错、美学等全方位的思考和决策
- SOA对架构设计师的任务要求
  - 体系结构整体决策
  - 构建系统解决方案
  - 基于服务构建分析模型与设计模型
  - 子系统设计

SOA与架构设计师

- **SOA架构设计师需注意的关键问题**
  - 着手体系结构设计时必须全面考虑所有可能的集成需求
  - 必须注意业务需求和可以提供的服务之间的动态关系
  - 控制服务粒度
    - 基于服务的功能及发送和接收的数据数量来定义服务粒度，如细粒度服务、粗粒度服务或组合服务
    - 细粒度服务执行了最小的功能，发送和接收少量的数据；粗粒度服务执行了较大的业务功能，并交换了更多的数据
    - 细粒度服务是供粗粒度服务或组合服务使用的，而不是由终端应用直接使用的，即粗粒度服务的用户不能直接调用他所使用的细粒度服务
    - 组合服务可以使用粗粒度服务和细粒度服务进行组装
    - 粗粒度服务可能使用多个细粒度服务，因此它们不能提供粒度级的安全和访问控制
  - 设计无状态的服务
    - 服务不应该依赖于其他服务的上下文和状态，SOA架构中的服务应该是无状态的服务
  - 基于 SOA 的系统并不排除使用面向对象的设计来构建单个服务，但是其整体设计却是面向服务的



SOA与架构设计师

- **企业IT系统向SOA架构转型，IBM能为架构师提供哪些咨询和帮助？**
  - IBM在全球设立了四个SOA设计中心，帮助客户进行SOA的流程设计以及实施
  - 在中国北京、上海的两个SOA设计中心将帮助重要合作伙伴和客户进行SOA架构设计、SOA评估

- **SOA架构设计师应该如何开始？**

  SOA不可能一蹴而就，需要分阶段、循序渐进、逐步实现：
  - 首先从梳理某一个关键业务入手，实施独立的Web服务
  - 接下来以SOA为思路，对具体的整合对象（如一项业务），按照建模、实现、装配、部署、管理5个阶段实现整合
  - 由小及大，逐渐在企业业务中进行整合扩散
  - 随着整合的业务逐渐增多，企业的IT整合计算平台会一步步完善，形成整个企业的IT转型，最终通过全面整合实现随需应变的企业IT架构

SOA与架构设计师

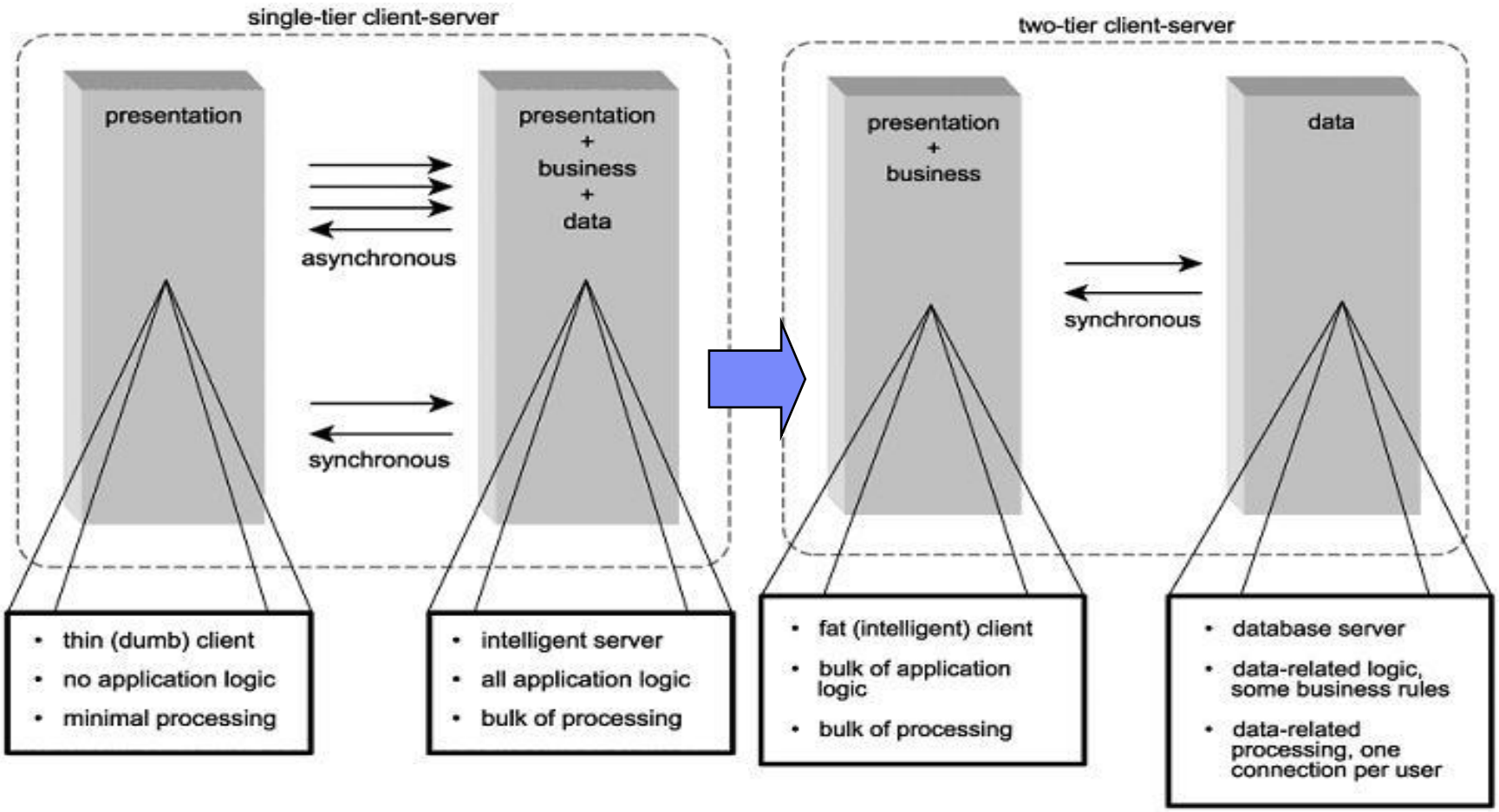
- SOA成熟度级别模型

级别	名称	目标或方法	Web服务的作用
4	随需应变的业务转型	*对已有业务模型或者将要部署的新业务模型的全面转型*	▪ 对整个价值网络建立基于标准的连接 ▪ 服务、产品、合作伙伴和过程流可以随市场情况变化，并且服务是随时可用的
3	企业范围的 IT 转型	*一个架构的实现能够覆盖整个企业中的所有业务职能*	▪ 建立服务的发布和使用——通过自描述和动态合并——在单一的面向服务的体系架构中
2	面向业务功能集成的服务	*针对一个业务目标集成来自企业内部和外部的跨多个应用的服务*	▪ 对于基本的交易，通过 Web 服务集成合作伙伴、部门或渠道 ▪ 集成合作伙伴或者部门的私有网络
1	实现独立的Web 服务	*从已有或者新应用中包含的任务创建服务*	▪ 将已有的功能作为Web 服务来展示，从当前的基础架构中释放价值

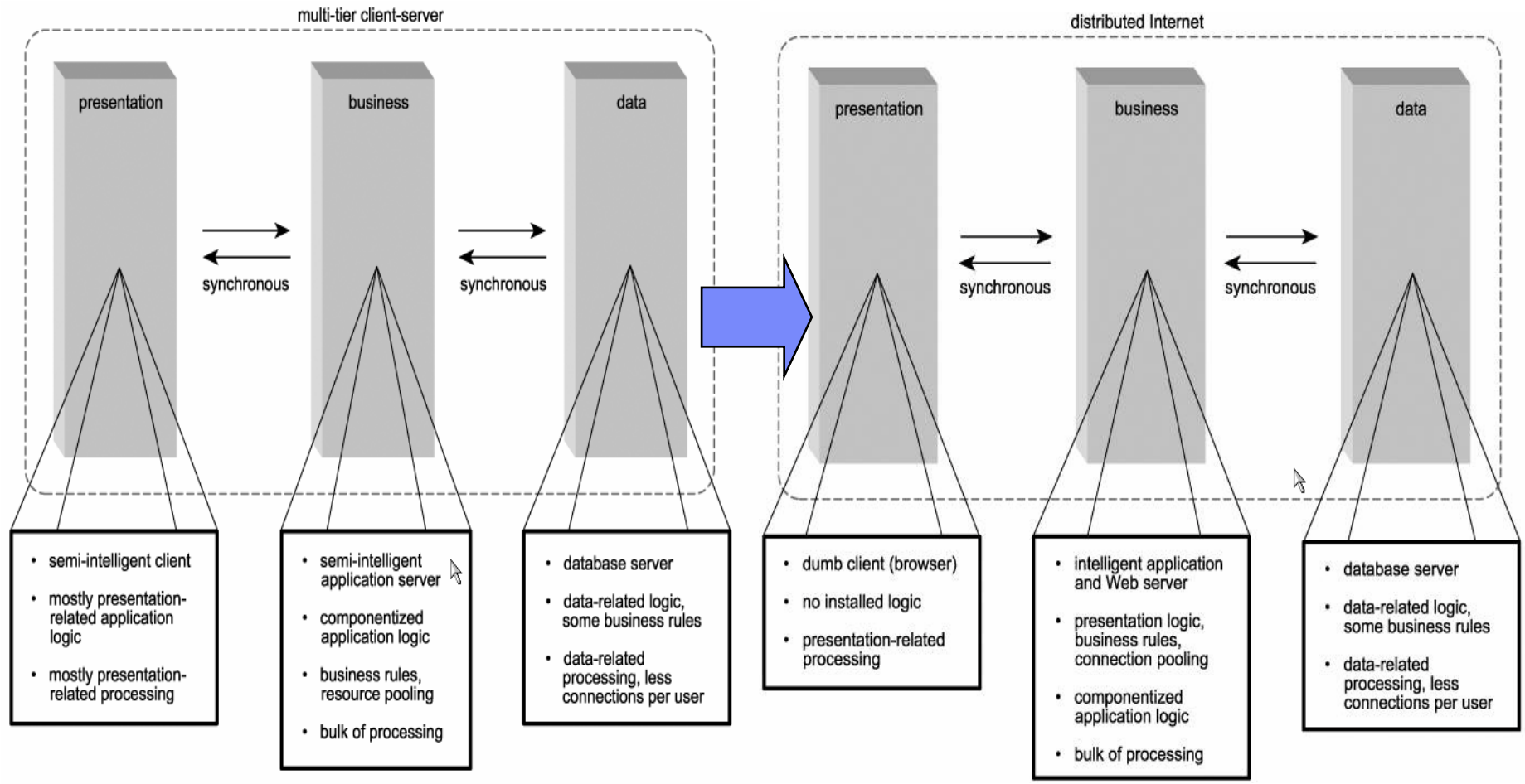
内容

- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

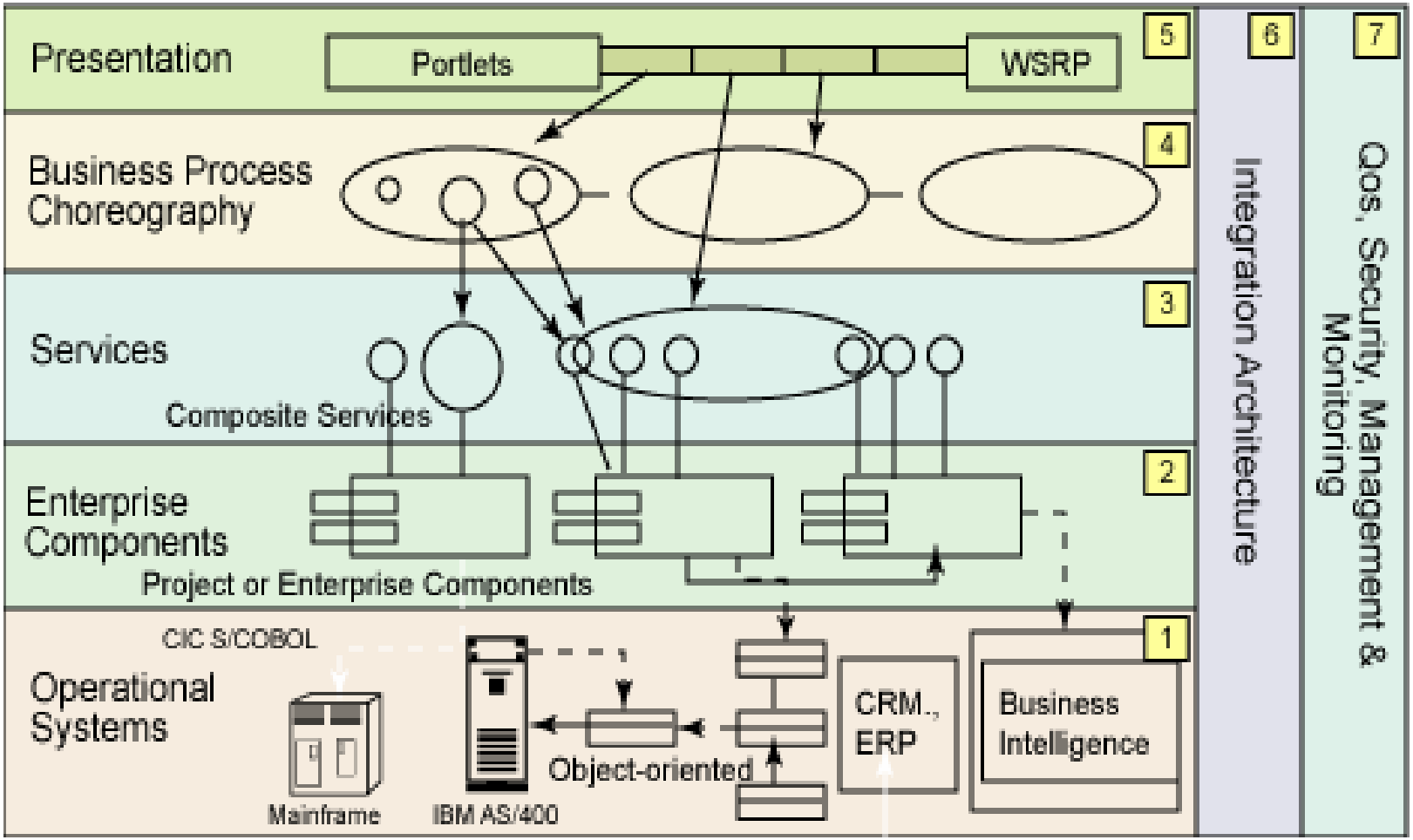
SOA和其它几种架构的比较—C/S架构



SOA和其它几种架构的比较—N-层架构



SOA和其它几种架构的比较－SOA



SOA和其它几种架构的比较－SOA架构的实例

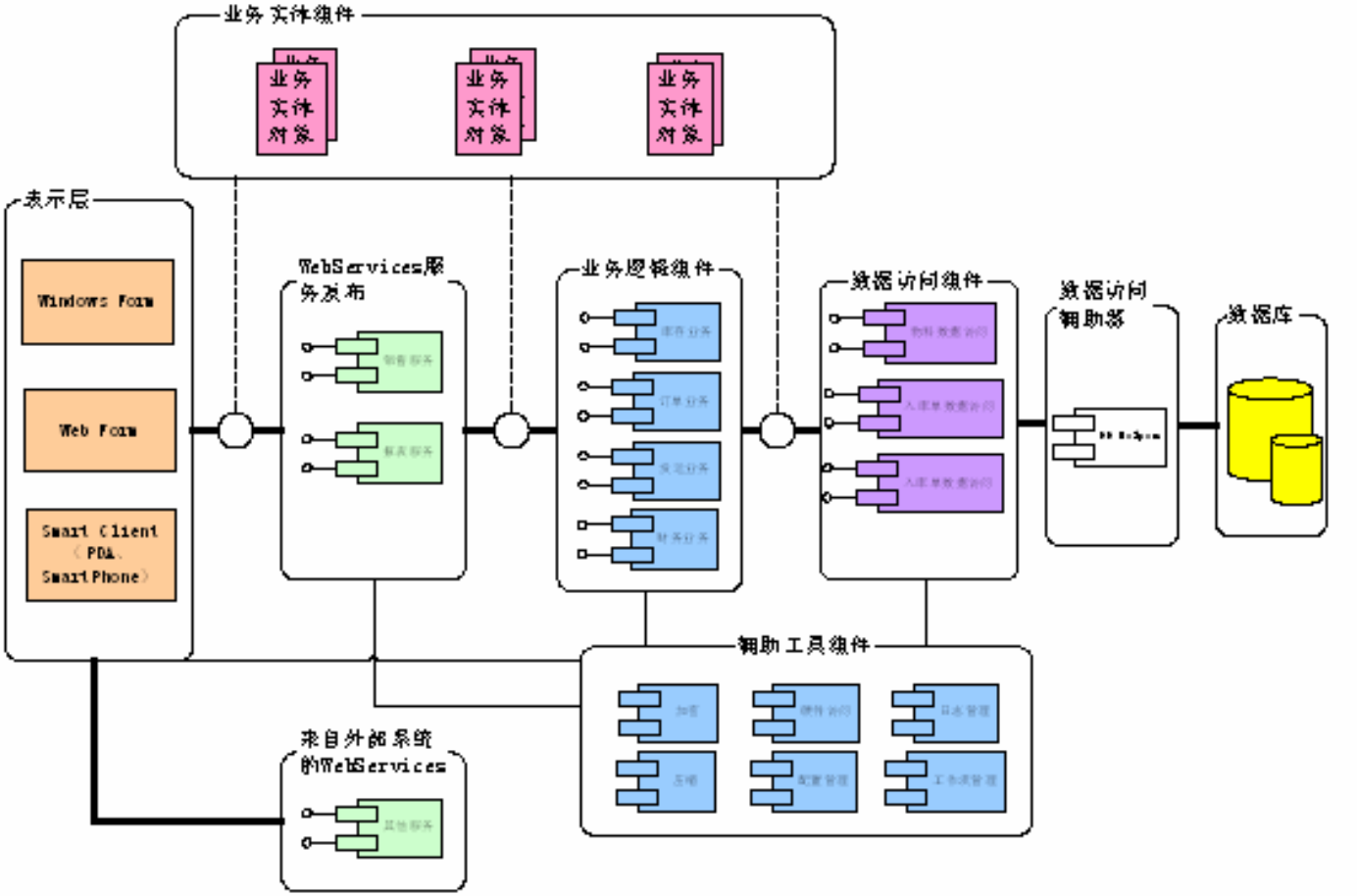
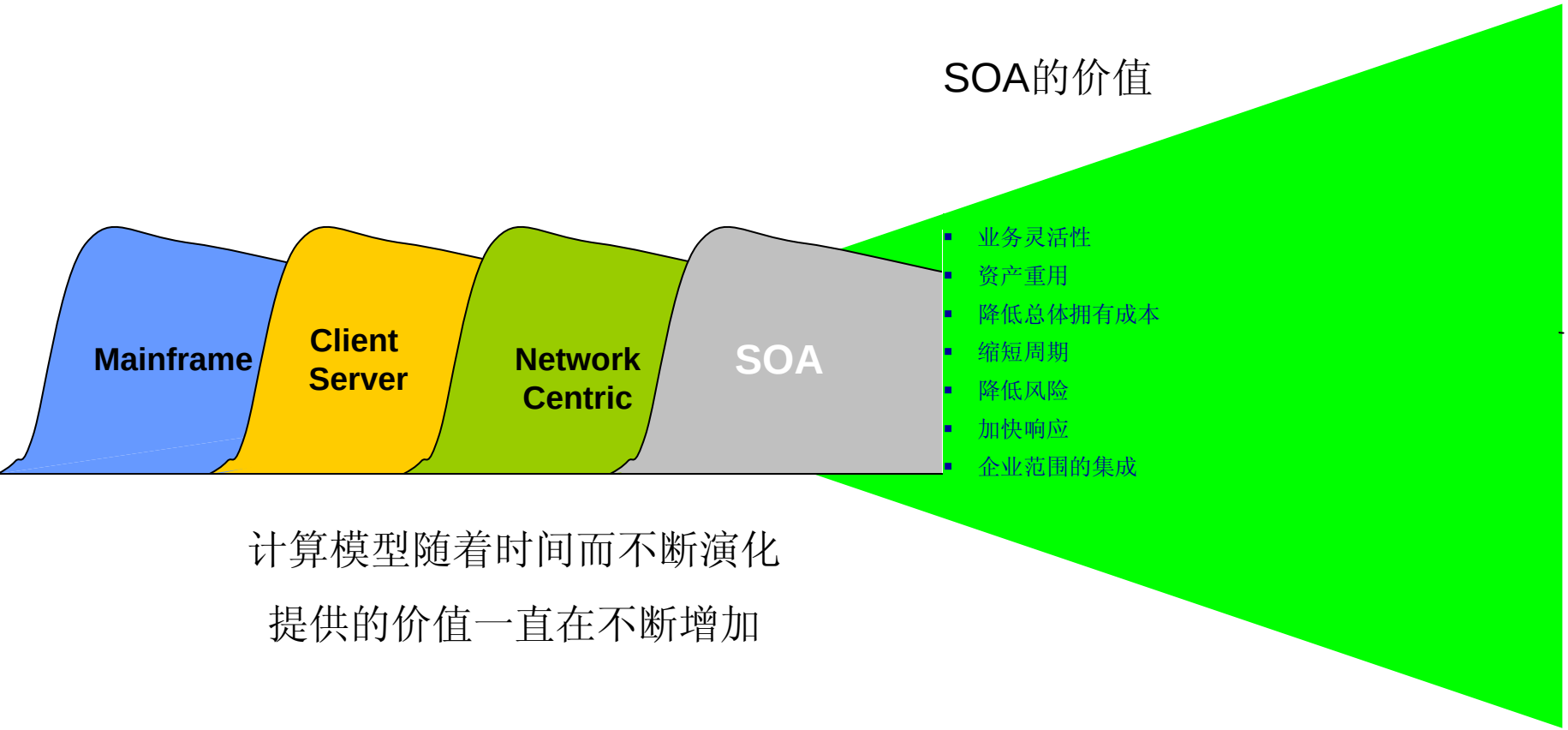


图 1 SOA 架构下的软件体系结构图

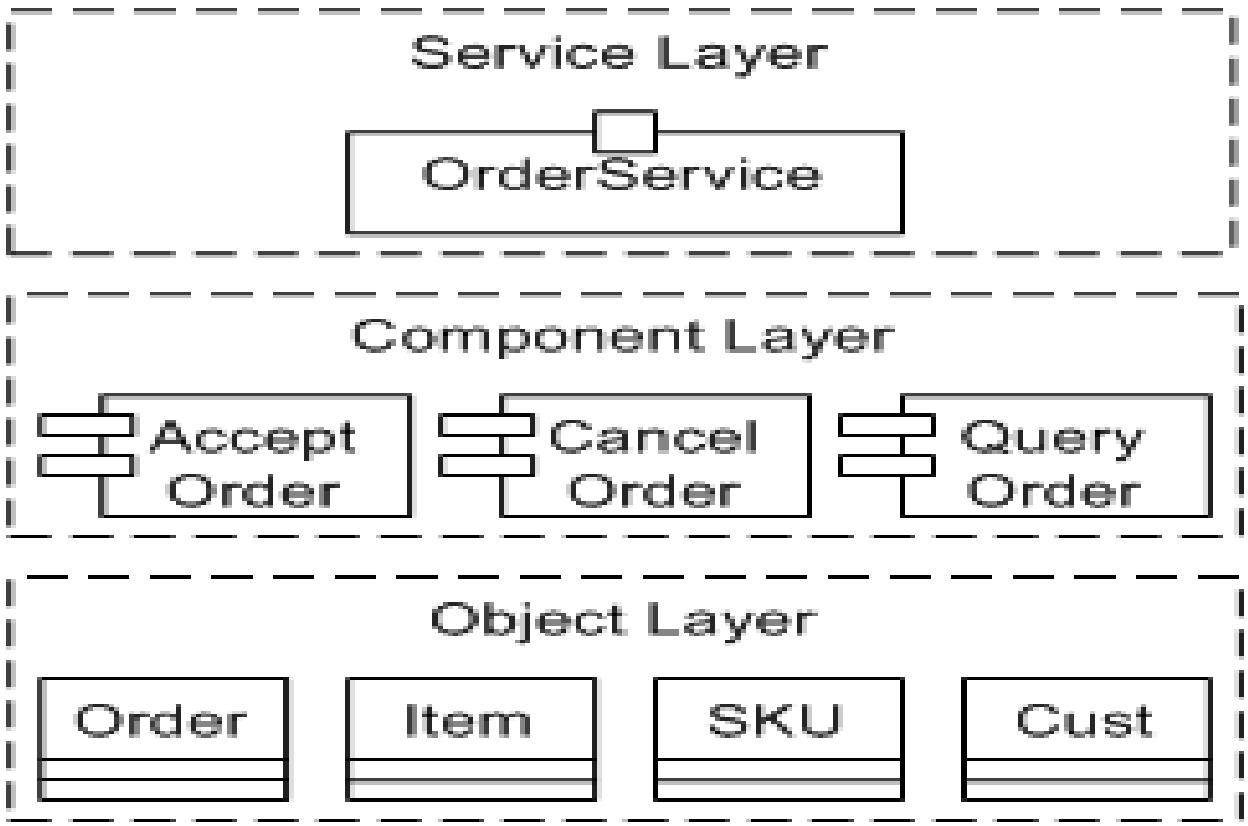
SOA和其它几种架构的比较



计算模型随着时间而不断演化

提供的价值一直在不断增加

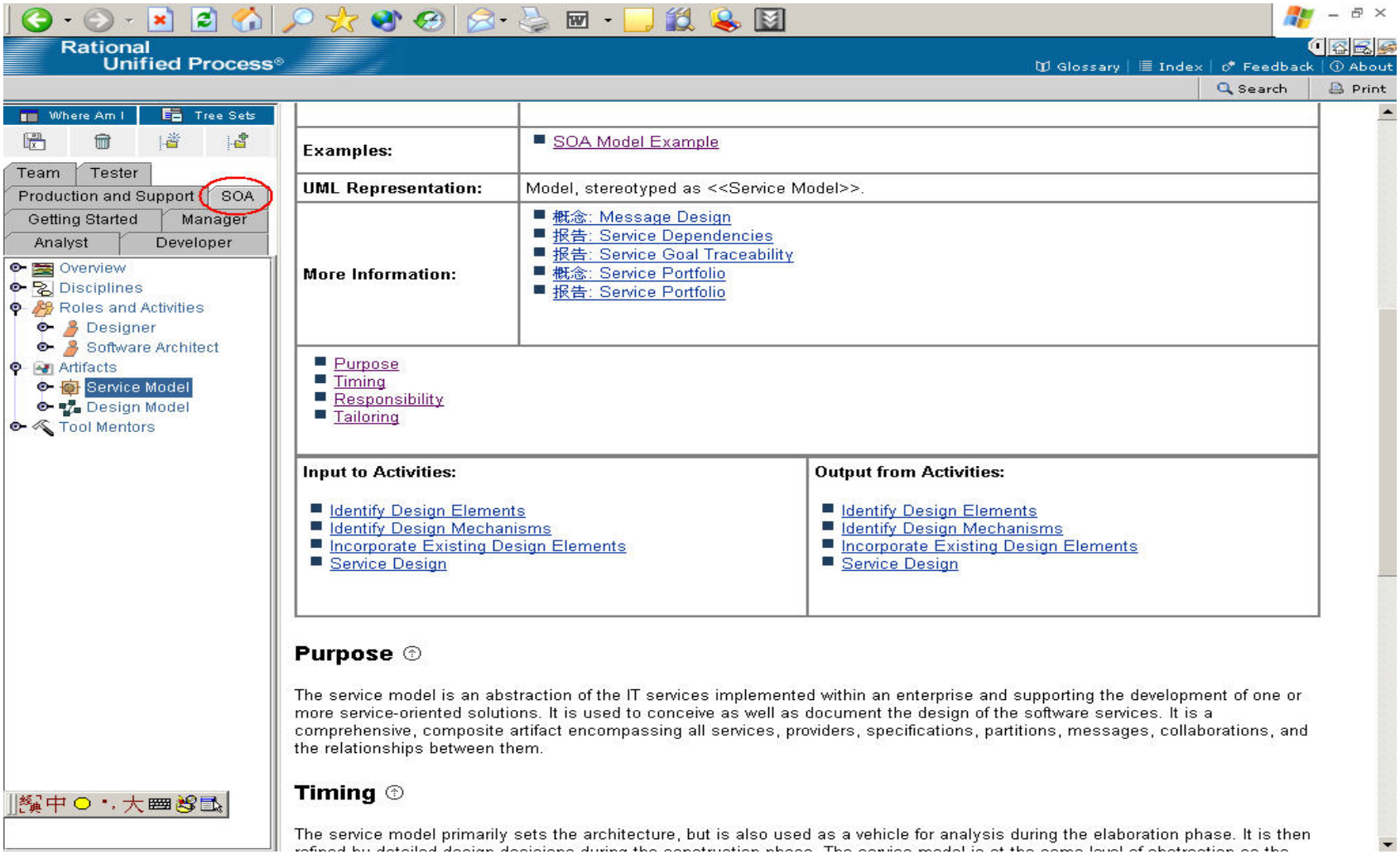
SOA与对象技术和组件技术



内容

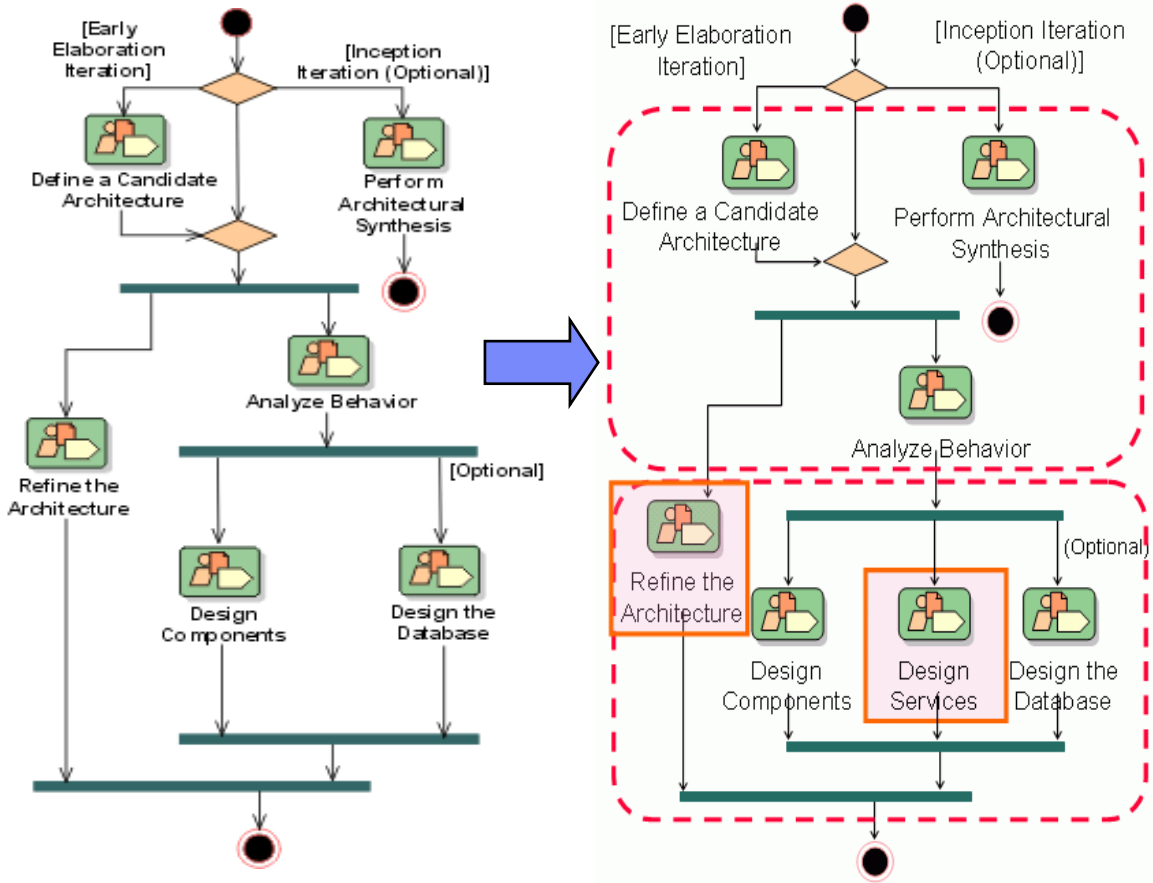
- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

RUP Plug-in for SOA



RUP Plug-in for SOA

- RUP Plug-in for SOA为分析与设计引入了两个关键流程：
  - 识别服务：精练架构工作流的一部分
  - 设计服务
- RUP Plug-in for SOA使用UML Profile for Software Services建模服务

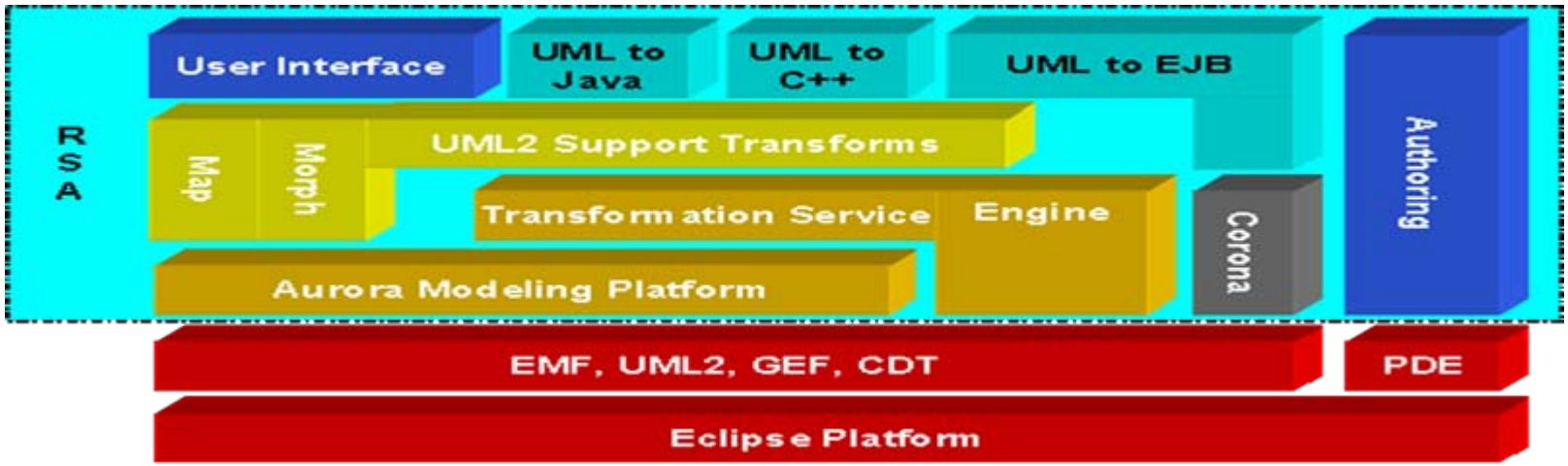


内容

- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

RSA平台功能概览

- IBM Rational Software Architect（RSA）是一套设计与开发工具，它构建在开放的、可扩展的Eclipse3.0平台之上，实现了多项行业最新标准，提供了灵活的插件扩展机制
- IBM在RSA中缺省提供了UML到Java、UML到C++、UML到EJB的模型转化实现，其整个实现架构基于Eclipse 3.0和Eclipse Model Framework(EMF)、UML2、Graphical Editing Framework(GEF)和插件开发环境(PDE)
- 借助UML2.0技术实现了多种软件开发模式，可以帮助开发团队创建更加强壮的软件架构
- RSA作为IBM Rational业务驱动软件开发平台的核心构件，提供了与需求管理工具、测试工具、配置和变更管理工具和项目管理工具的完美集成，从而真正实现了企业内部的核心软件开发流程、开发平台和软件生产线



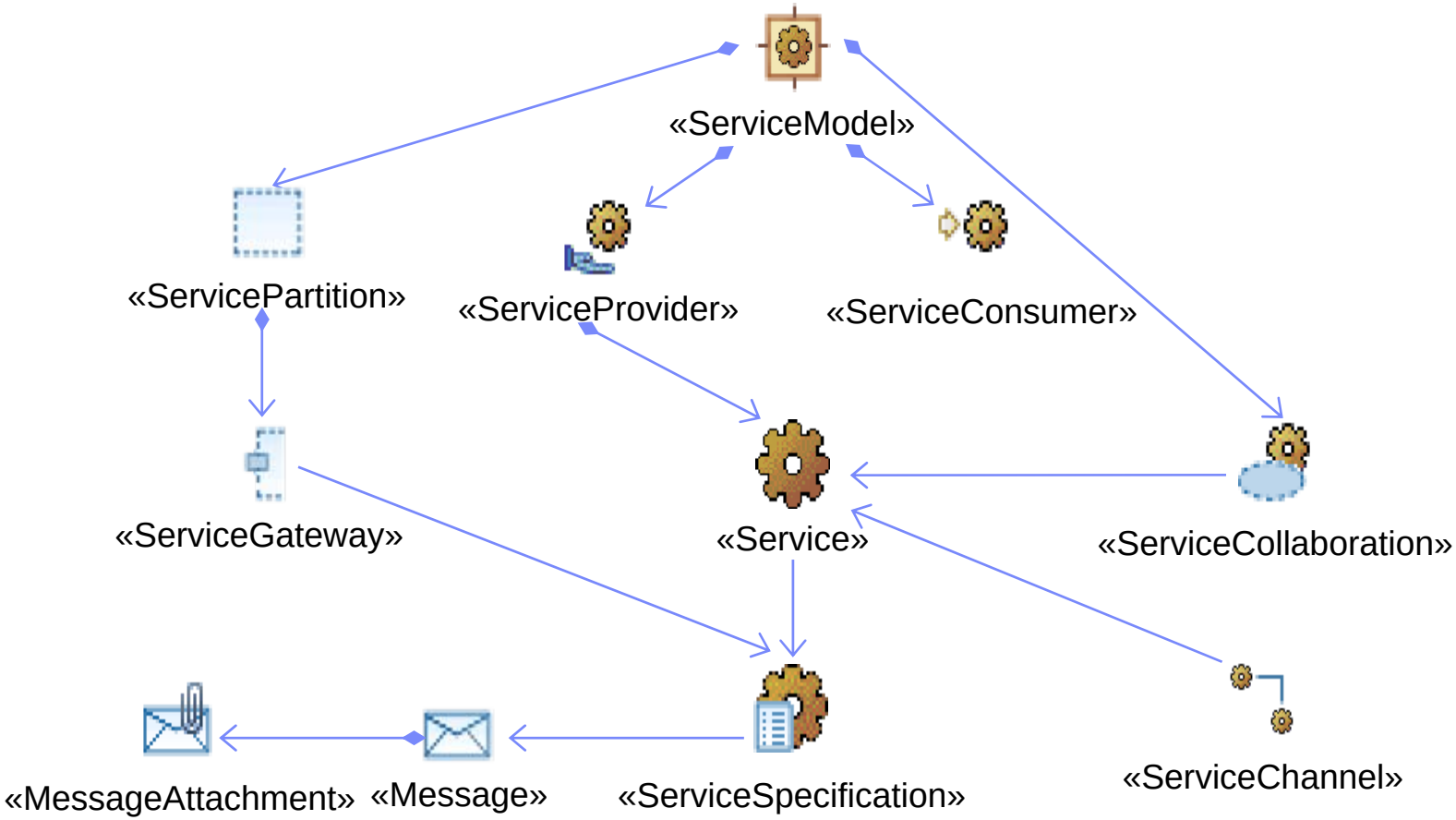
内容

- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

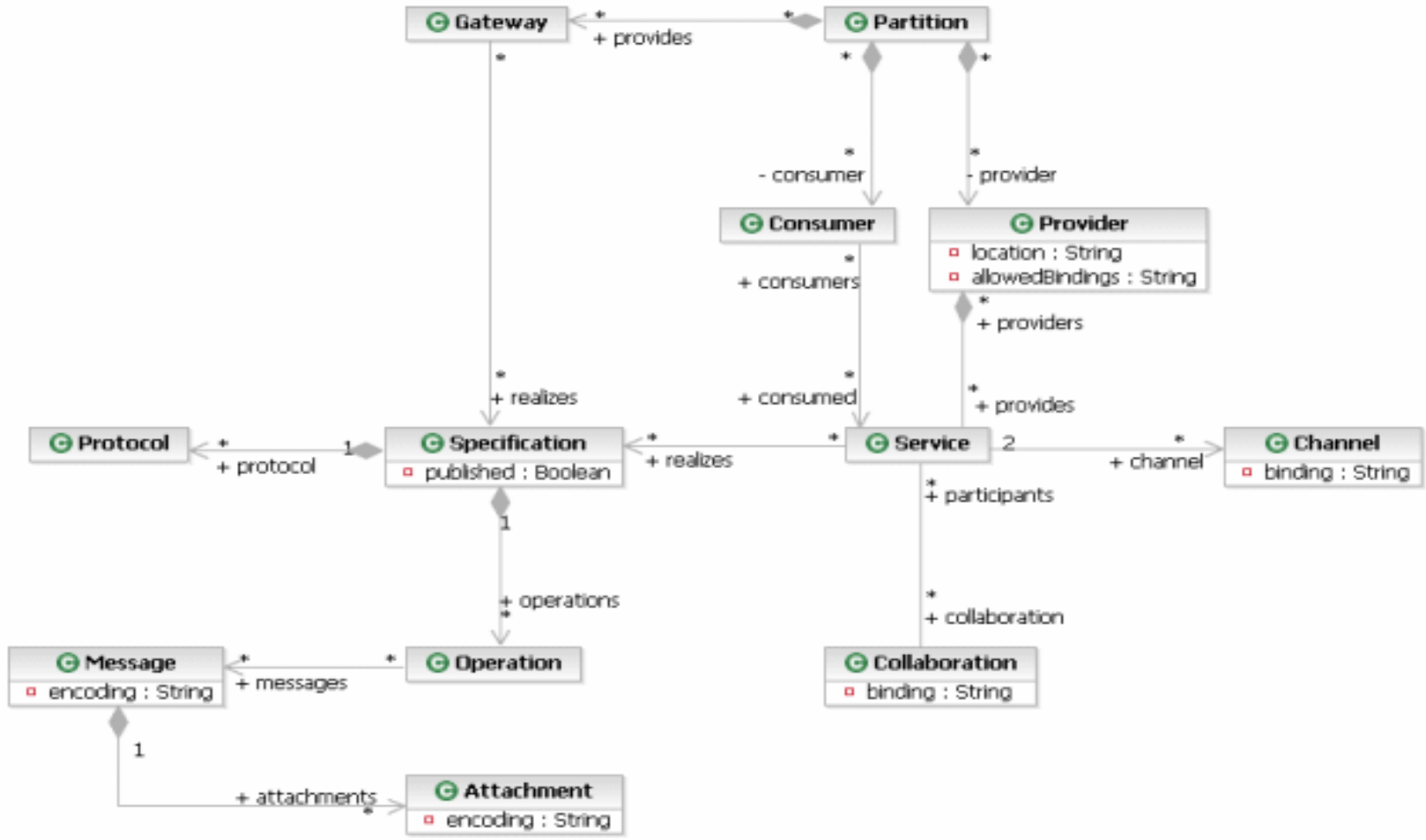
Software Services的UML概要文件(Profile)

- 作为一个**RSA Plug-In**，目的是为描述**Service**提供一个共通语言
  - 用于为Service、SOA和Service-oriented Solutions建模
  - 适合复杂场景(scenario)建模，帮助人们解决在开发Service-oriented Solutions过程中所关心的问题
  - 包括开发周期内的很多活动并且为不同的涉众提供了视图
  - 包括下列文件：
    - SoftwareServices.epx – Software Services的UML概要文件的具体实现
    - Service Design Model.emx – 一个模板，用于创建新模型
    - Retail Example Service Model.emx – 一个例子，演示概要文件的用法
- 下载网址
  - http://www-128.ibm.com/developerworks/rational/library/05/510_svc/
- 安装
  - 需将RSA升级至6.0.0.1以上
  - 用IBM Rational Product Updater 工具安装

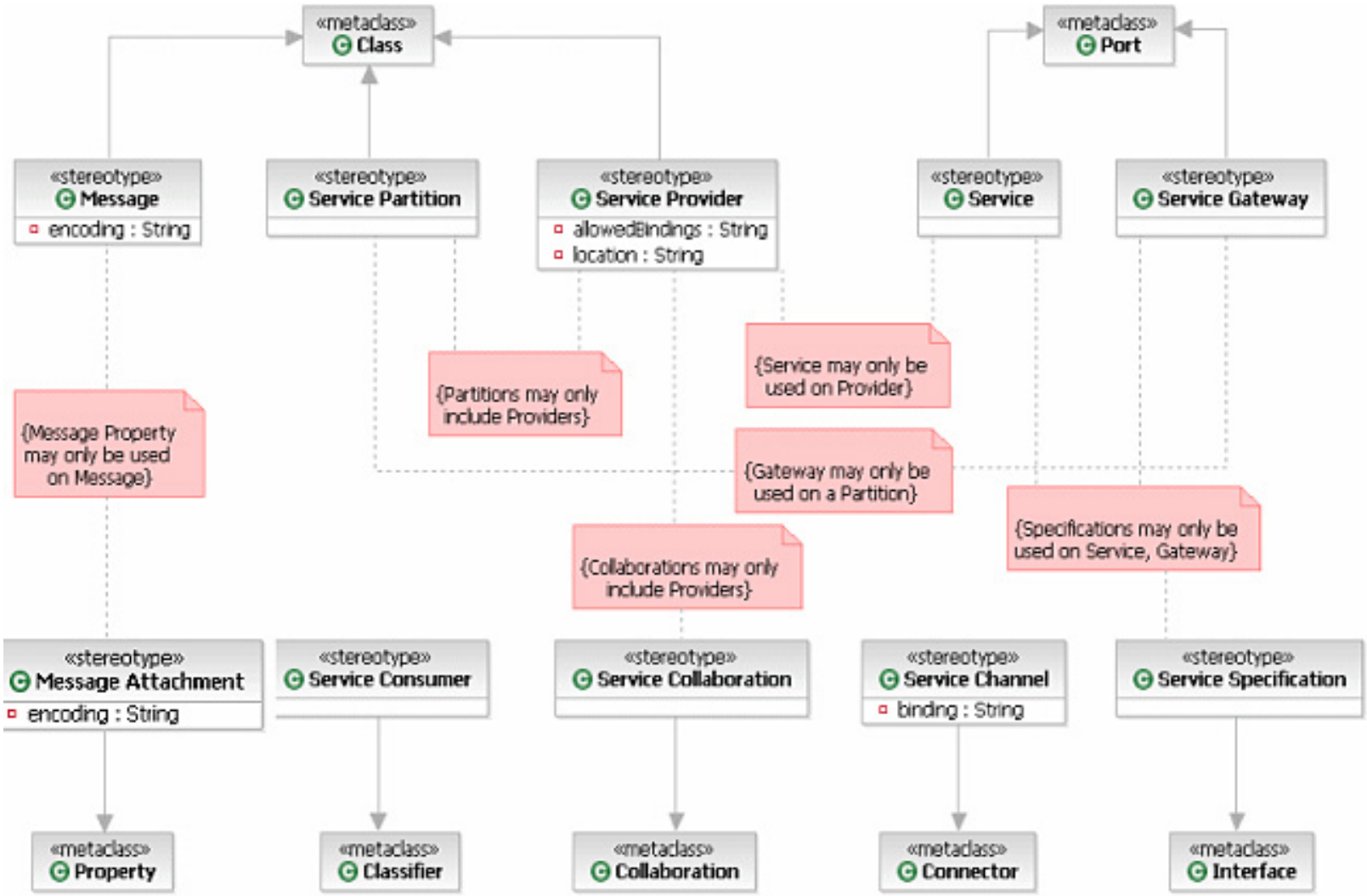
Software Services的UML概要文件(Profile)



Software Services的UML概要文件(Profile)



Software Services的UML概要文件(Profile)



Software Services的UML概要文件(Profile)

■ Service 

继承自	端口（Port）
语义	●Service模型元素提供Service交互作用的一个端点，而这些交互作用由Service Specification定义
属性	无
约束	●应该只在<<Service Provider>>中使用 ●应该以<<Service Specification>>为类型

Software Services的UML概要文件(Profile)

■ Service Specification 

继承自	接口（Interface）
语义	●对接口的使用表示Service提供的一个操作集合，一个Service可能实现多个接口 ●规定一个Service的结构和行为，可以看成是控制服务存取或者使用的一组规则 ●可以将一个协议状态机或者UML2.0 Collaboration附加到Service Specification上来表示Service Specification上操作调用的顺序 ●Service Specification只能提供公开的操作，而且每个操作应当只消耗至多一条消息、产生至多一条消息
属性	●Published：Boolean 表示Service是否被发布到一个Service存储库中，这和UML中提供的公有/私有的属性是不同的概念
约束	●所有的操作都应当被标记为公有的 ●只能作为<<Service>> 或 <<Service Gateway>>的类型

Software Services的UML概要文件(Profile)

■ Service Partition 

继承自	类（Class）
语义	● 一个Service Partition代表系统的某个逻辑或者物理的边界 ● 建模Service Partition是可选的但是有用的 ● 一个Service Partition的符号也比较严格，如果一个Service Partition表示它和其他所有Service Partition的全部通信必须直接通过指定的Gateway，那么它被称为是一个严格的(Strict) Service Partition。
属性	无
约束	● 任何自己的部分是<<Service Provider>>

Software Services的UML概要文件(Profile)

■ Service Collaboration 

继承自	协作（Collaboration）
语义	●Service Collaboration代表Service之间通信，这些Service通常被封装成一个新的Service，新的Service的实现是一组现存Service的简单的协作 ●Service Collaboration是一种指定一个实现的Service作为一个其他Service的协作的方式 ●在定义Service Specification阶段，通过Service Collaboration理解Service所承担的职责；在Design Service阶段，通过Service Collaboration表达Service在整个业务流程中是如何互相协作的
属性	无
约束	无

Software Services的UML概要文件(Profile)

■ Service Provider 

继承自	类（Class）
语义	●Service Provider是一个提供一个或者多个Service的元素 ●一个Service Provider拥有一个属性能够捕捉他的位置信息 ●Service Provider可能不直接暴露任何属性和操作 ●只有公开的Port可被提供，Service Specification中会有所记录
属性	●Location：String Service Provider的位置，它可以被用作创建端点的名字。 ●AllowedBindings：String 表示允许的平台绑定机制，一个Service Channel可能被用在连接Service中
约束	●任何自己的部分应当是 <<Service>>

Software Services的UML概要文件(Profile)

■ Service Consumer 

继承自	分类器（Classifier）
语义	●任何分类器(类、组件等等)可能既充当Service的Consumer，也包括其他的Service
属性	无
约束	无

Software Services的UML概要文件(Profile)

■ Service Channel 

继承自	连接器（Connector）
语义	●一个Service Channel表示两个Service之间的通讯路径 ●交互作用可能出现在一个Service Channel，但Service Channel并不表示任何特殊的交互作用 ●在Web Service世界里，每个Service表示与之相联系的绑定从而一个客户可以访问它
属性	●Binding：String 表示在 Web 服务描述语言（WSDL）规范中产生服务绑定的平台绑定机制，例如 SOAP-RPC、SOAP-Doc、HTTP-Get等等
约束	无

Software Services的UML概要文件(Profile)

■ Service Gateway 

继承自	端口（Port）
语义	●Service Gateway看起来像一个Service，但它却只是对Service Partition可用的，而对Service Provider是不可用的 ●一个Service Gateway充当一个代理服务，可以使用它协调协议或者表示一个Service Partition上的可以利用的接口 ●如果只有某些Service对于Service Partition之外是可用的，则使用Service Gateway来提供这些服务
属性	无
约束	●应当只用于<<Service Partition>> ●应该以<<Service Specification>>为类型

Software Services的UML概要文件(Profile)

■ Message 

继承自	类（Class）
语义	●Message是实际数据对Service和Consumer有意义一个容器 ●Message应采取值传递方式，其中不应定义任何行为，但是它可能有属性和与其他类的关联(很可能是某个领域模型)
属性	●Encoding：String 代表了用于产生消息模式的平台编码机制，如SOAP-literal、SOAP-rpc、ASN.1等等
约束	●不应有任何自己的操作 ●不应有任何自己的行为

Software Services的UML概要文件(Profile)

■ Message Attachment

继承自	属性（Property）
语义	●表示Message的某些组件是该Message的一个附件(相对于消息的直接部分本身) ●在高级的设计活动中一般不太可能经常使用这个原型，但是对于许多过程，区分Attachment数据和内嵌的Message数据还是很重要的
属性	●Encoding：String 代表了用于产生消息模式的平台编码机制，如SOAP-RPC、Doc-Literal、ASN.11等等
约束	●只能用于<<Message>>的属性

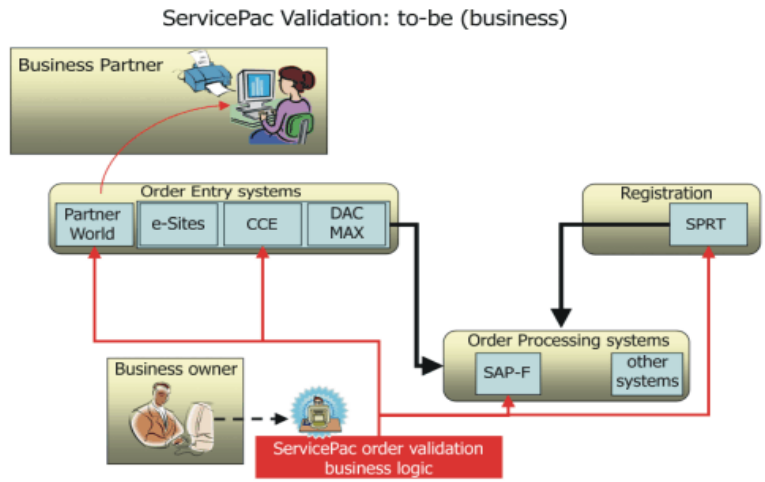
内容

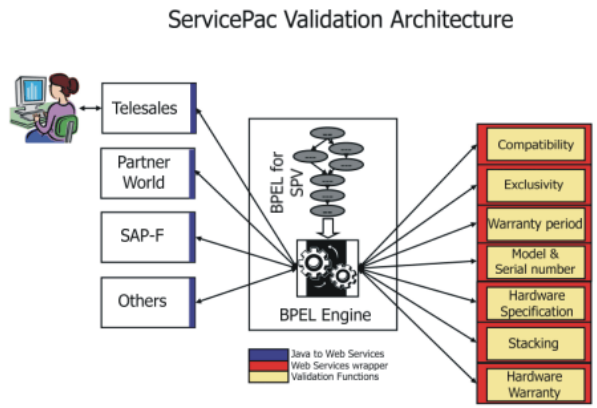
- SOA与架构设计师
- SOA和其它架构的比较
- RUP Plug-in for SOA
- RSA平台功能概览
- Service概要文件
- 开始SOA架构设计

开始SOA 架构设计

一、基于IBM ServicePac的Order Validation业务演示SOA架构设计

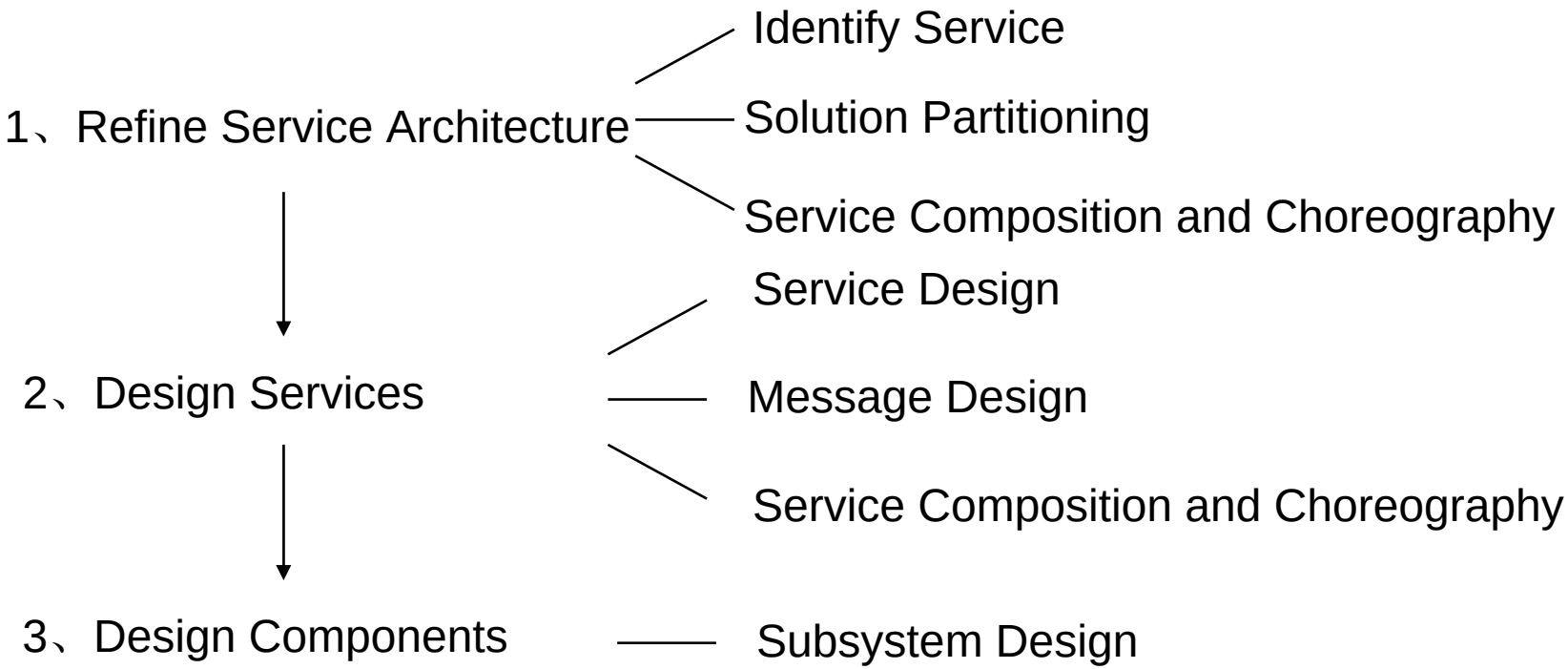
- IBM ServicePac背景
  - IBM公司向用户提供的全球专业服务产品，包括台式机、服务器、专业工作站、便携机等的维修和服务
  - 服务通过IBM业务合作伙伴提供，内容包括保修期内升级、保修期外合约服务、单机安装服务等等
  - 业务处理复杂，形式多种多样，还在不断增加新的服务项目
  - 系统需要不断的升级和与旧系统集成
- 采用SOA重构系统原因
  - 通过升级ServicePac业务的IT支撑系统，达到业务处理能力和速度连续不断的提升
  - 降低开发、维护和运营成本
  - 适应业务流程重组和变革的需要





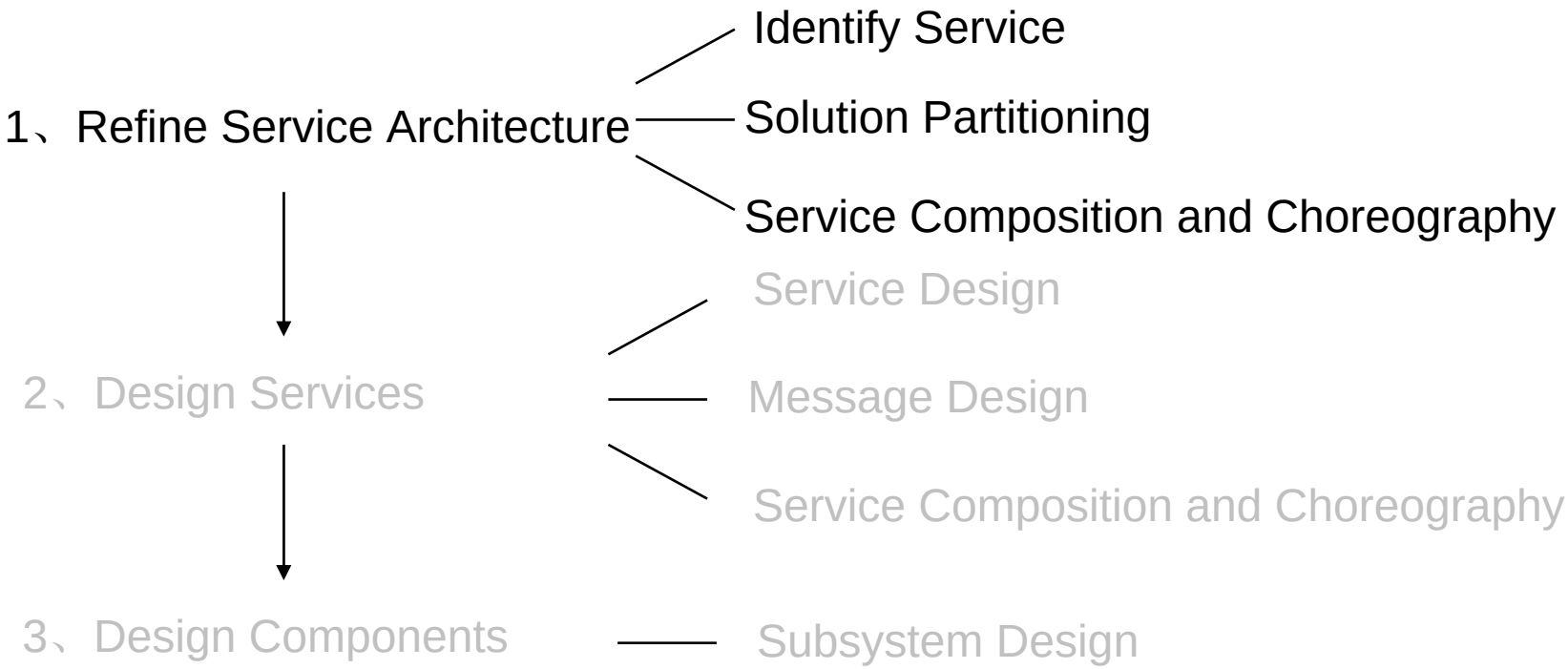
开始SOA 架构设计

二、设计步骤



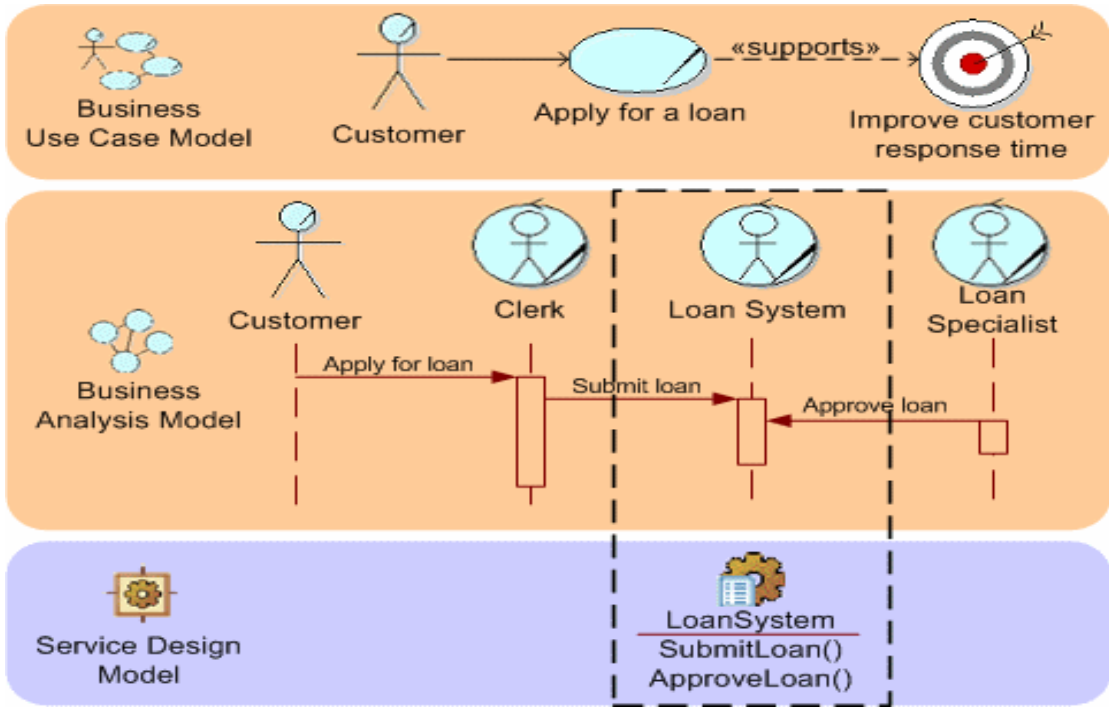
开始SOA 架构设计

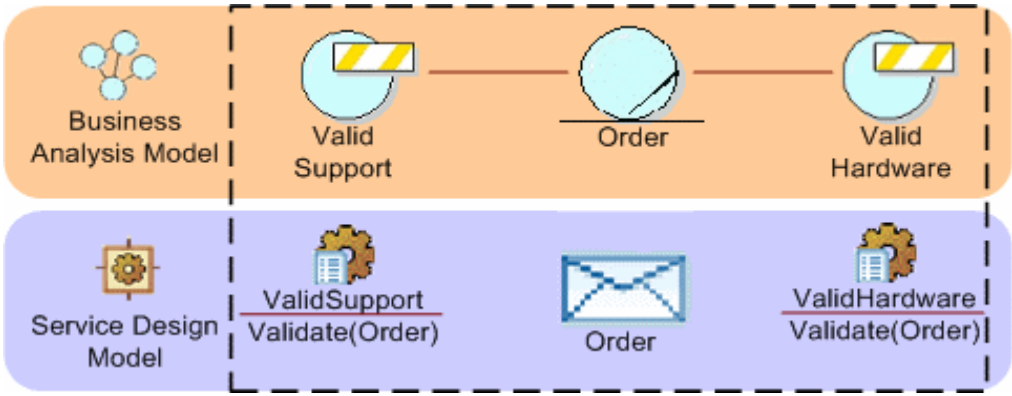
二、设计步骤

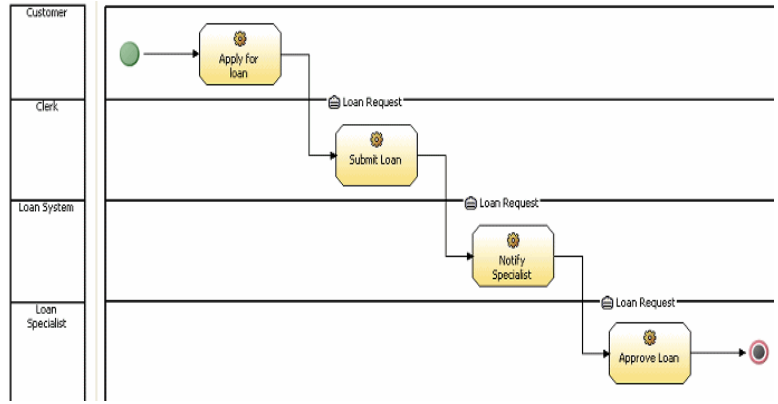


Identify Service

- 活动: Identify Services
  - 自上向下，业务流程驱动
  - 自上向下，用例驱动
  - 数据驱动
  - 业务规则驱动
  - 自底向上，通过现有的资产产生服务

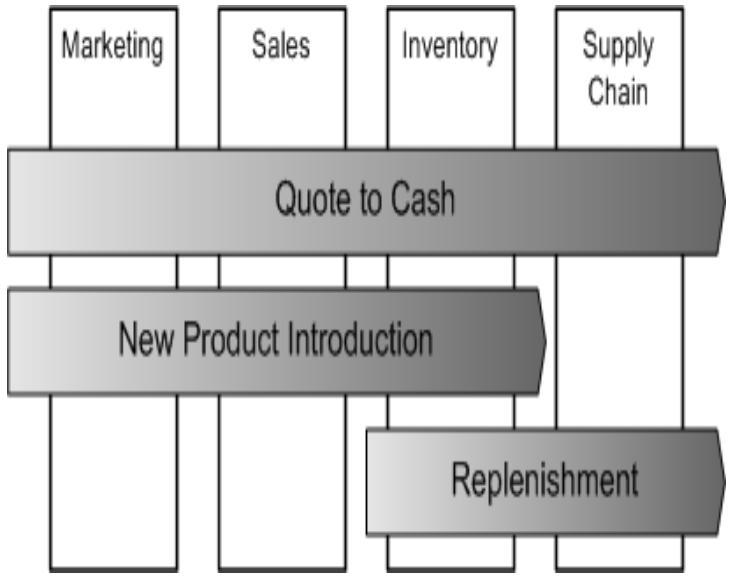


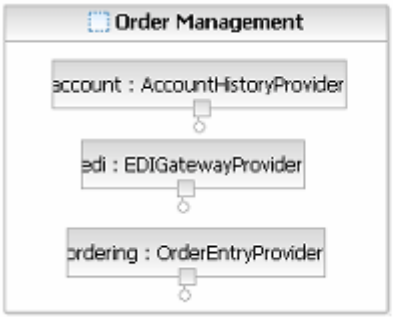


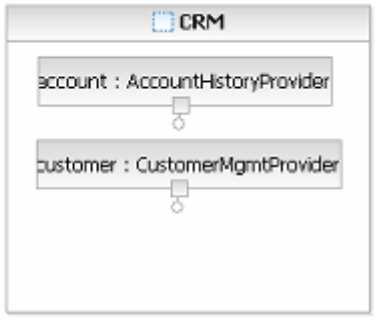


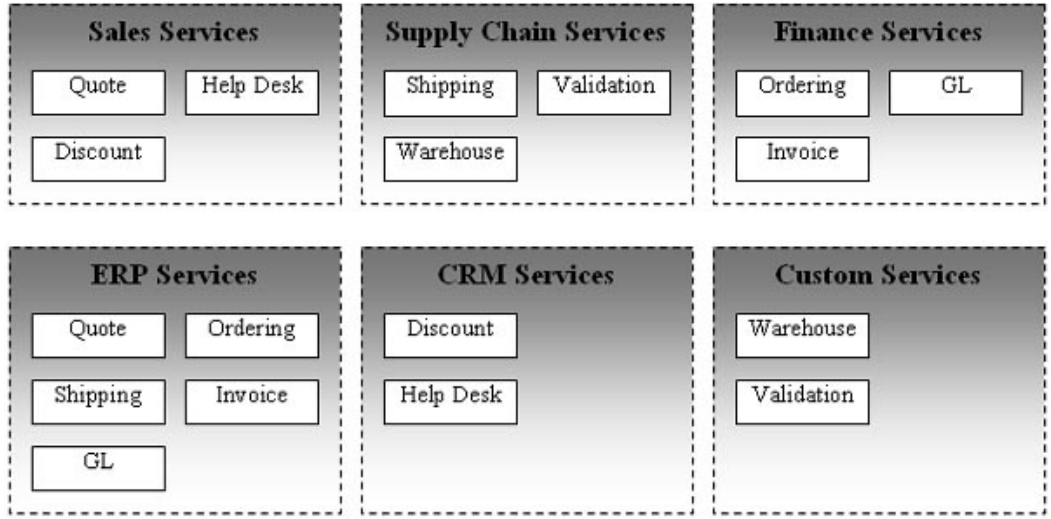
Solution Partitioning

- 活动: Solution Partitioning
  - 架构师通过Solution Partitioning来对架构中比较关注的地方进行深入研究
    - 如右图中，架构师描绘出Service在企业不同部门的应用情况



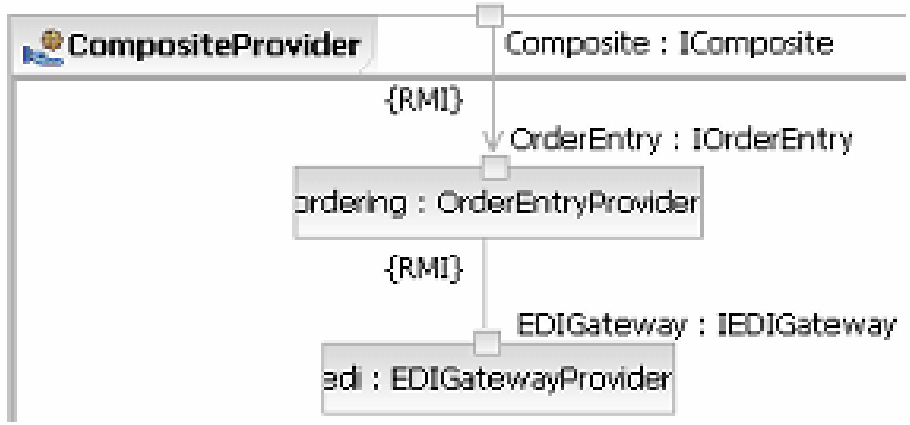




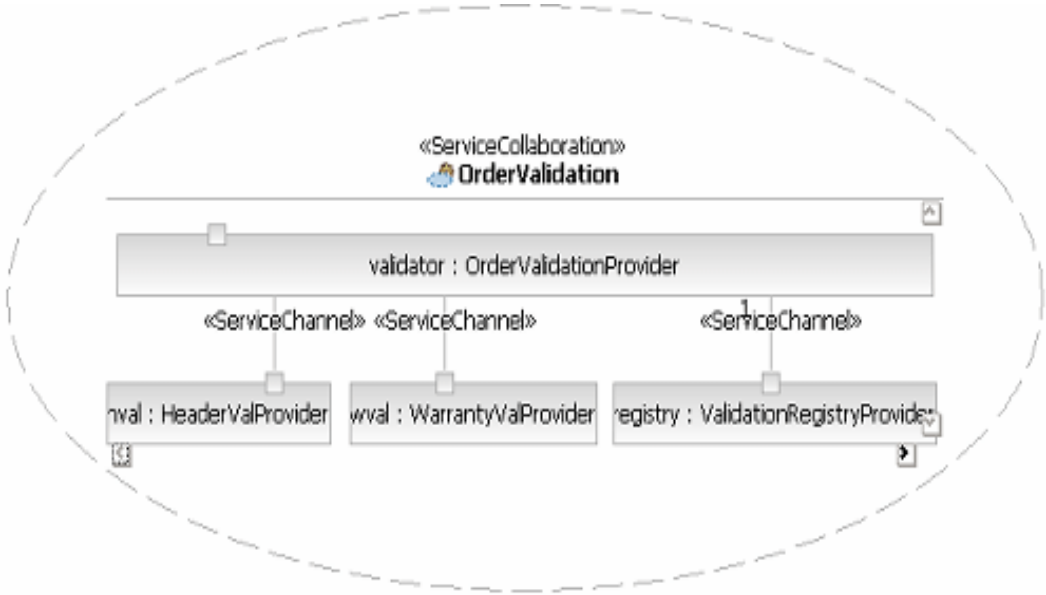


Service Composition and Choreography

- 活动: Service Composition and Choreography
  - 如果一个ServiceProvider内部是由多个Service组成，可以通过Service Composition来描绘组成关系

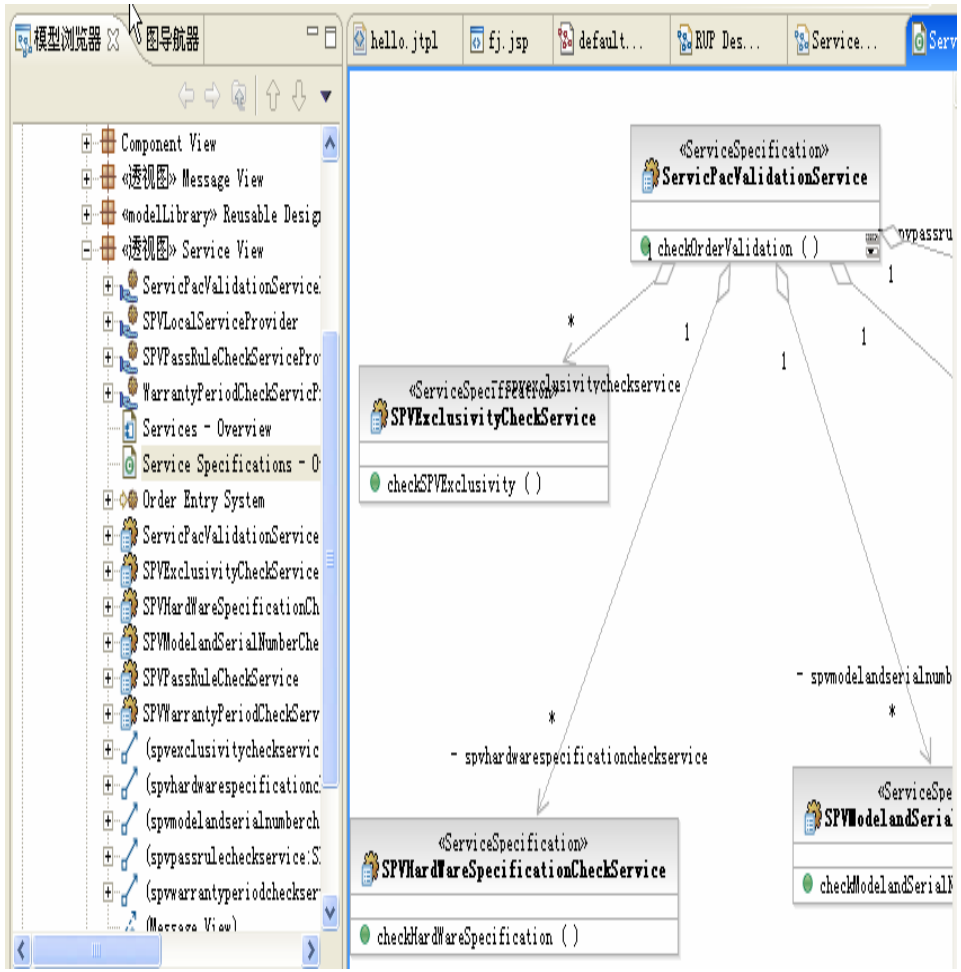


  - 如果要了解Service之间的协作关系，或者了解Service是如何一起实现整个业务流程的，可以通过Service Collaboration描绘



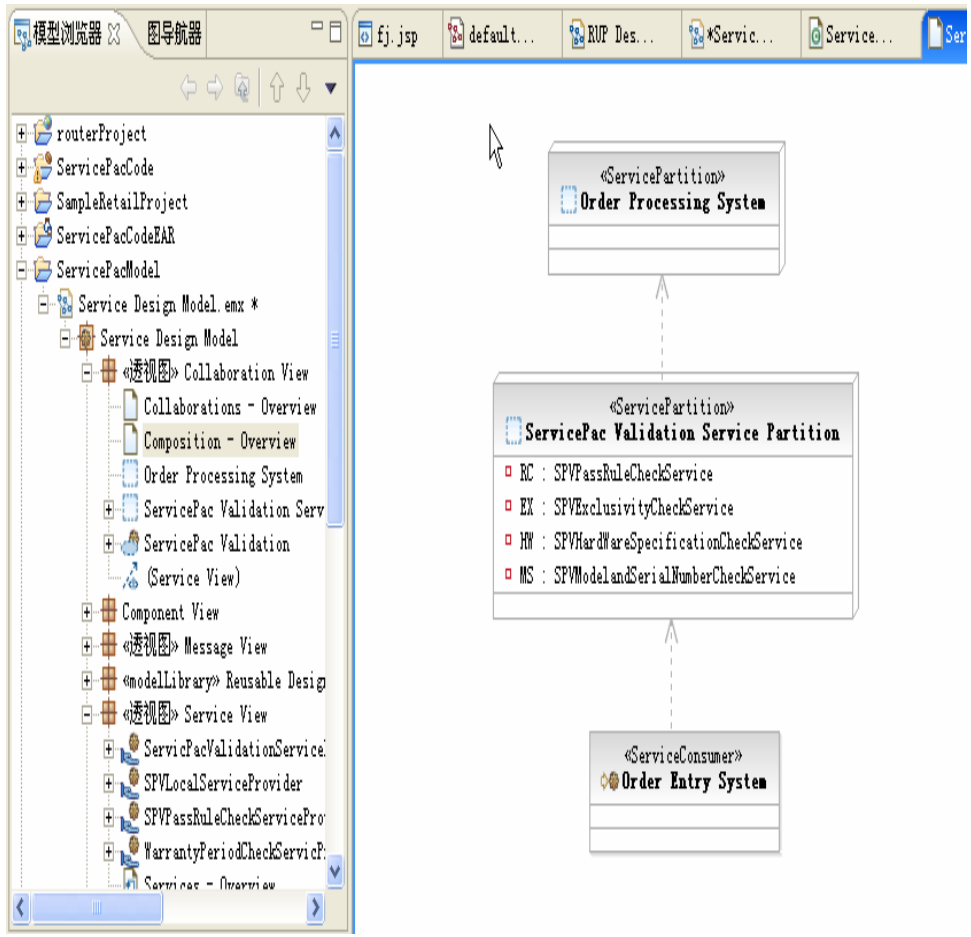
Demo1 - Service Specification

根据在业务需求阶段找出的Service，定义其Service Specification



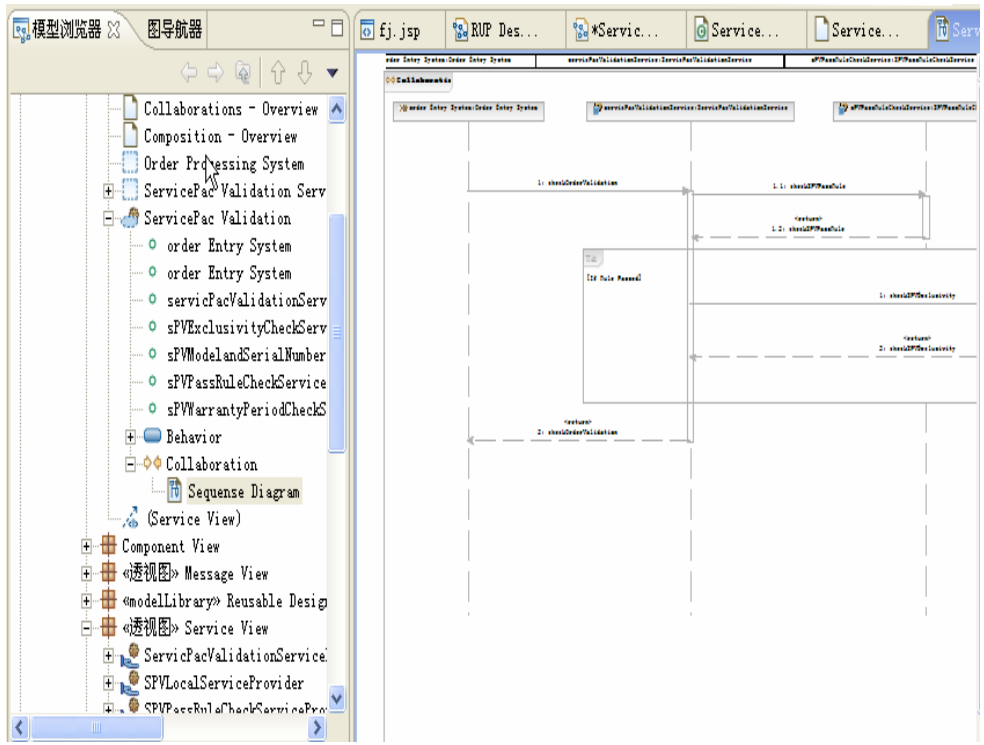
Demo2 - Service Partition

通过Service Partition描述Validation Service系统与其它系统的关系



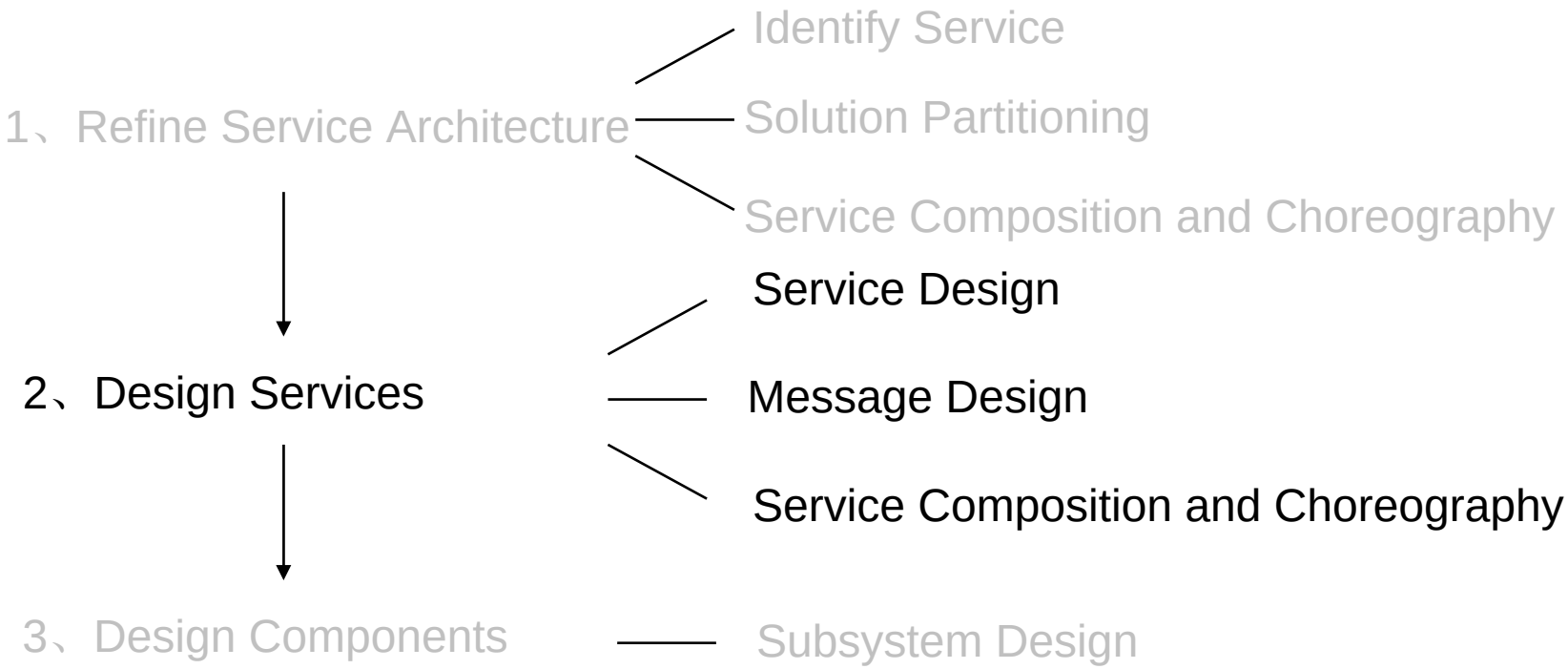
Demo3 - Service Collaboration in Identify Service

通过Service Collaboration描述Service Rule Validation场景，从而找出Service的职责



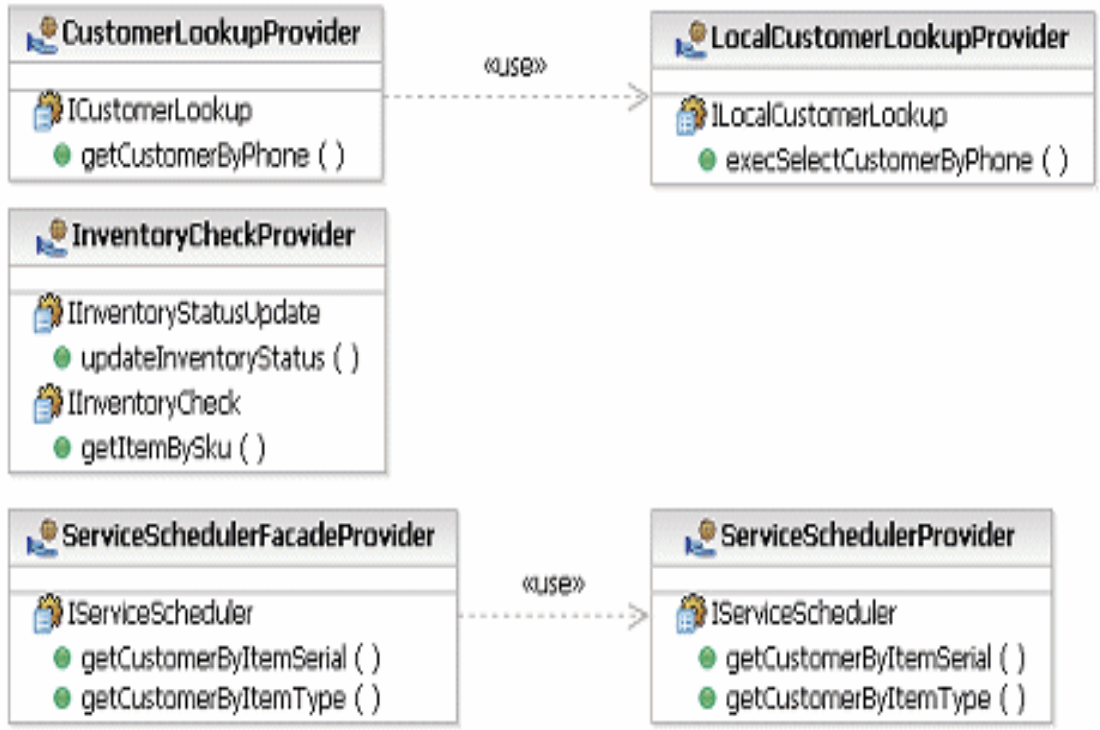
开始SOA 架构设计

二、设计步骤



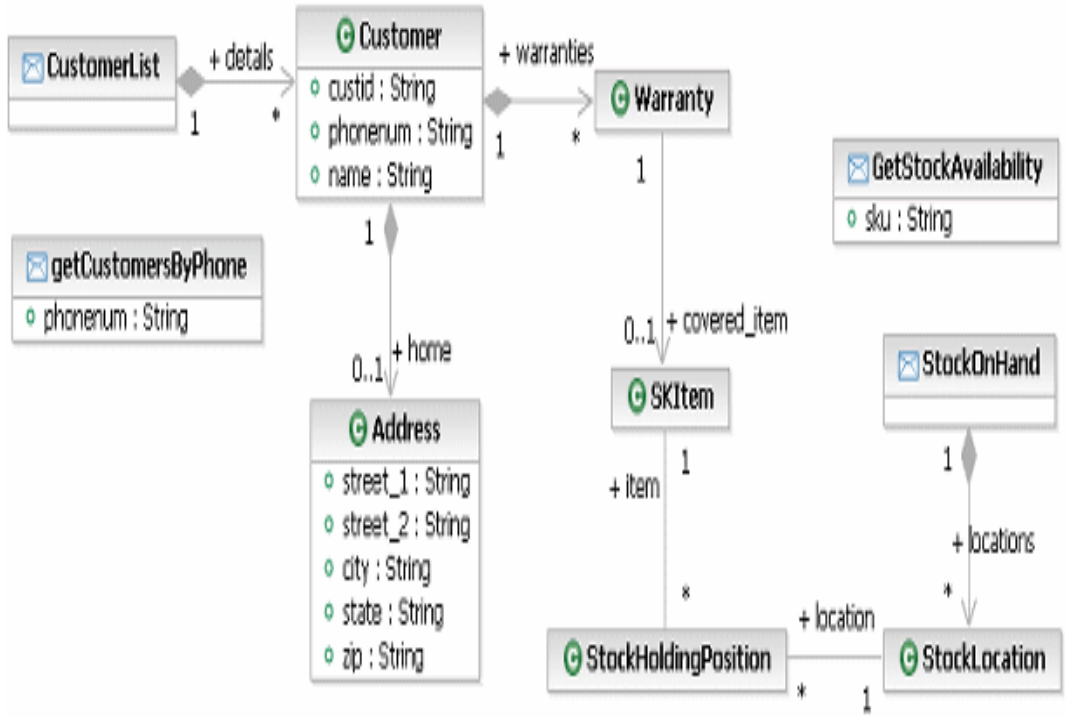
Service Design

- 活动: Service Design
  - 在这一步通常要考虑如何具体的实现之前定义的Service Specification
    - 根据实际部署的情况定义Service Provider
    - 合并相应的Service Specification



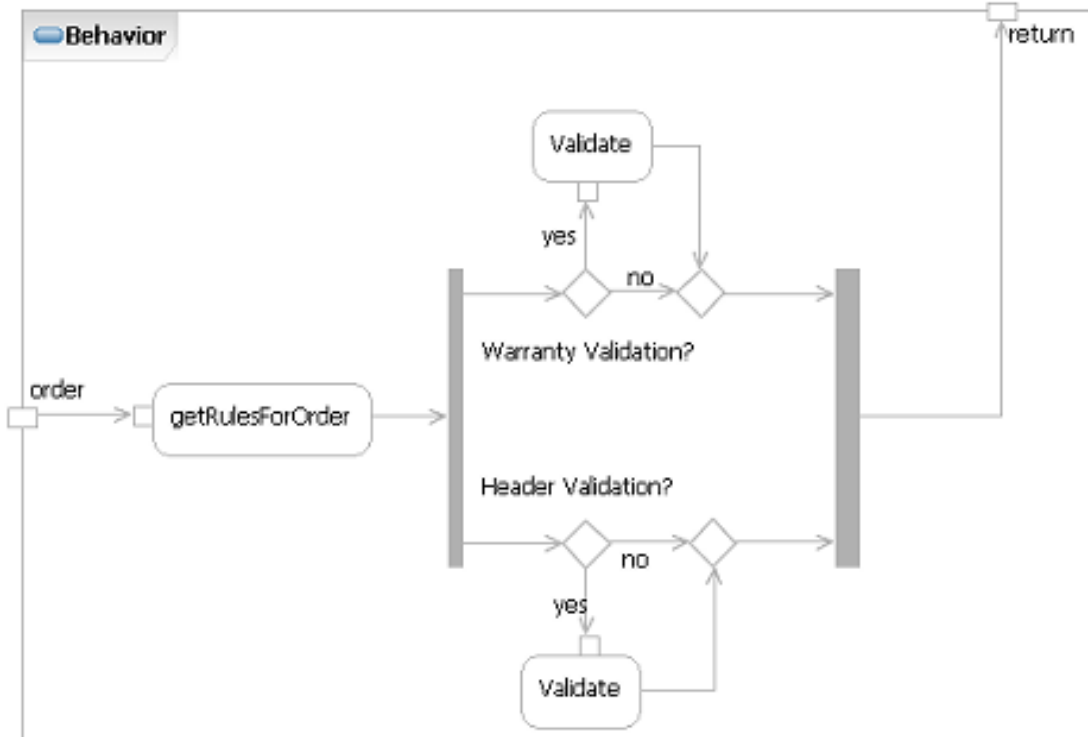
Message Design

- 活动: Message Design
  - 首先定义领域的实体模型，然后通过Message定义实体类之间的数据传递



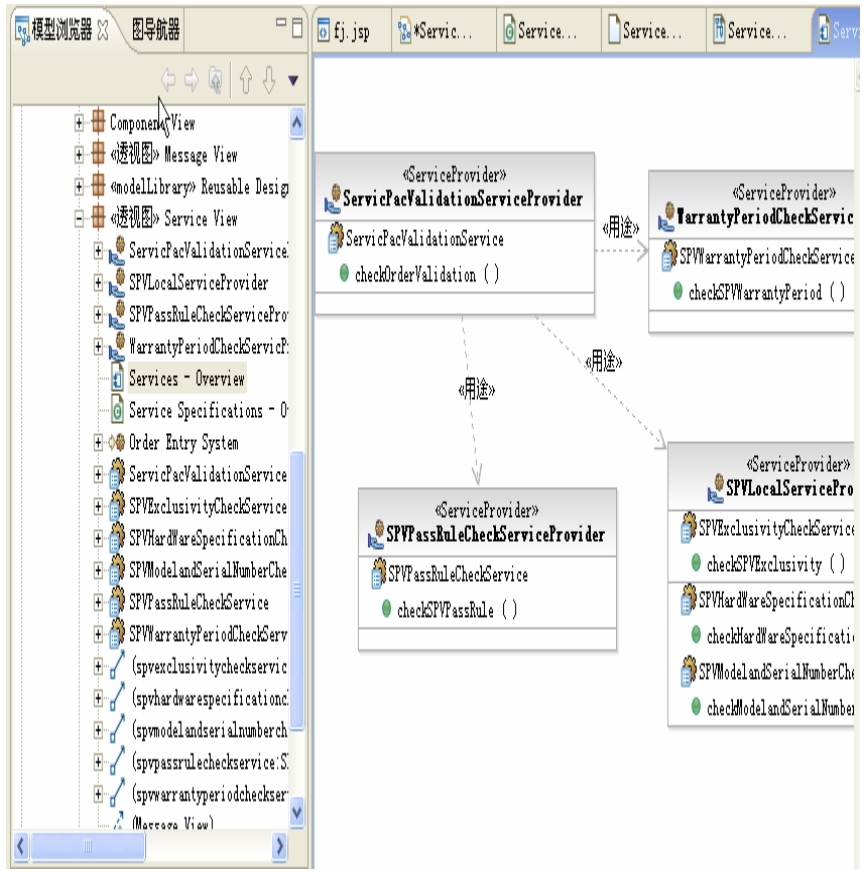
Service Composition and Choreography

- 活动: Service Composition and Choreography
  - 通过Service Collaboration描绘Service是如何一起实现整个业务流程的



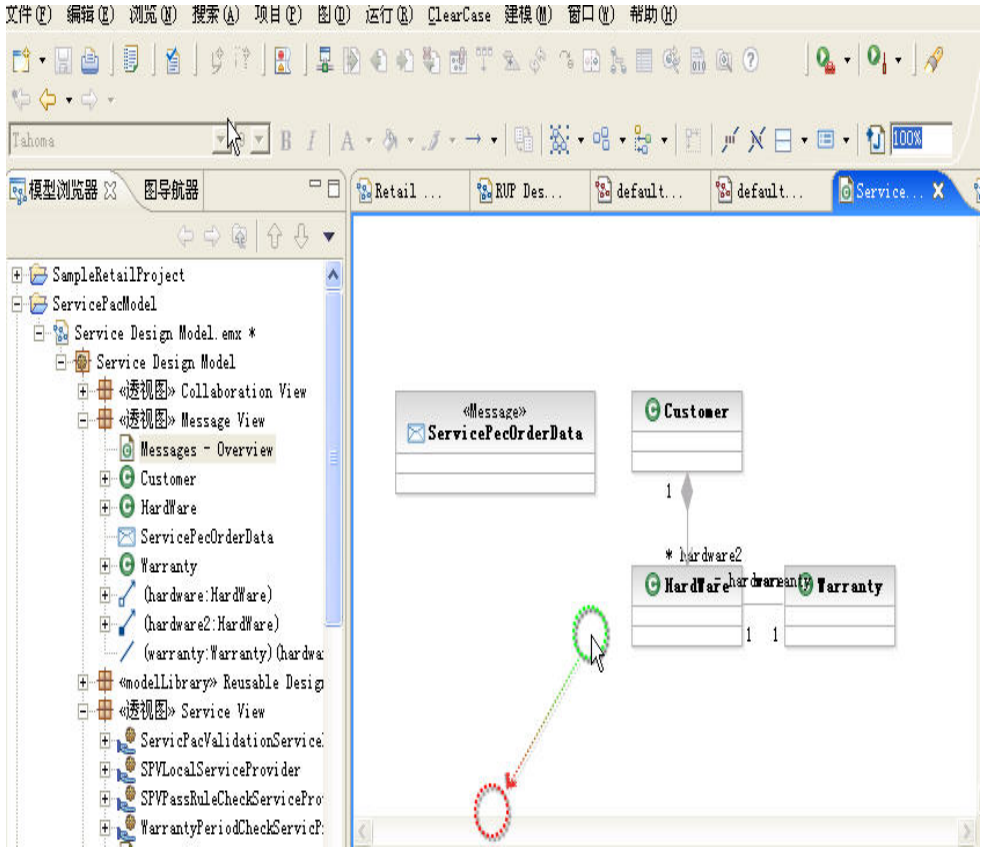
Demo4 - Service Provider

把第三方系统提供的Service由单独的ServiceProvider提供，而local Service由一个localServiceProvider提供



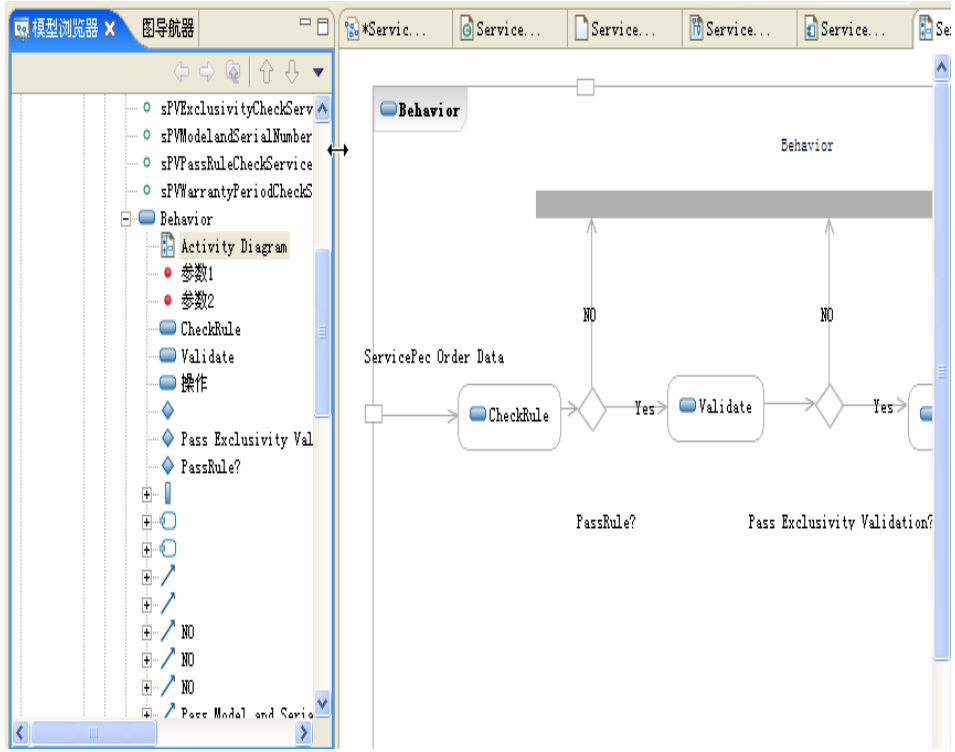
Demo5 – Message Design

首先定义领域的实体模型，然后通过Message定义实体类之间的数据传递



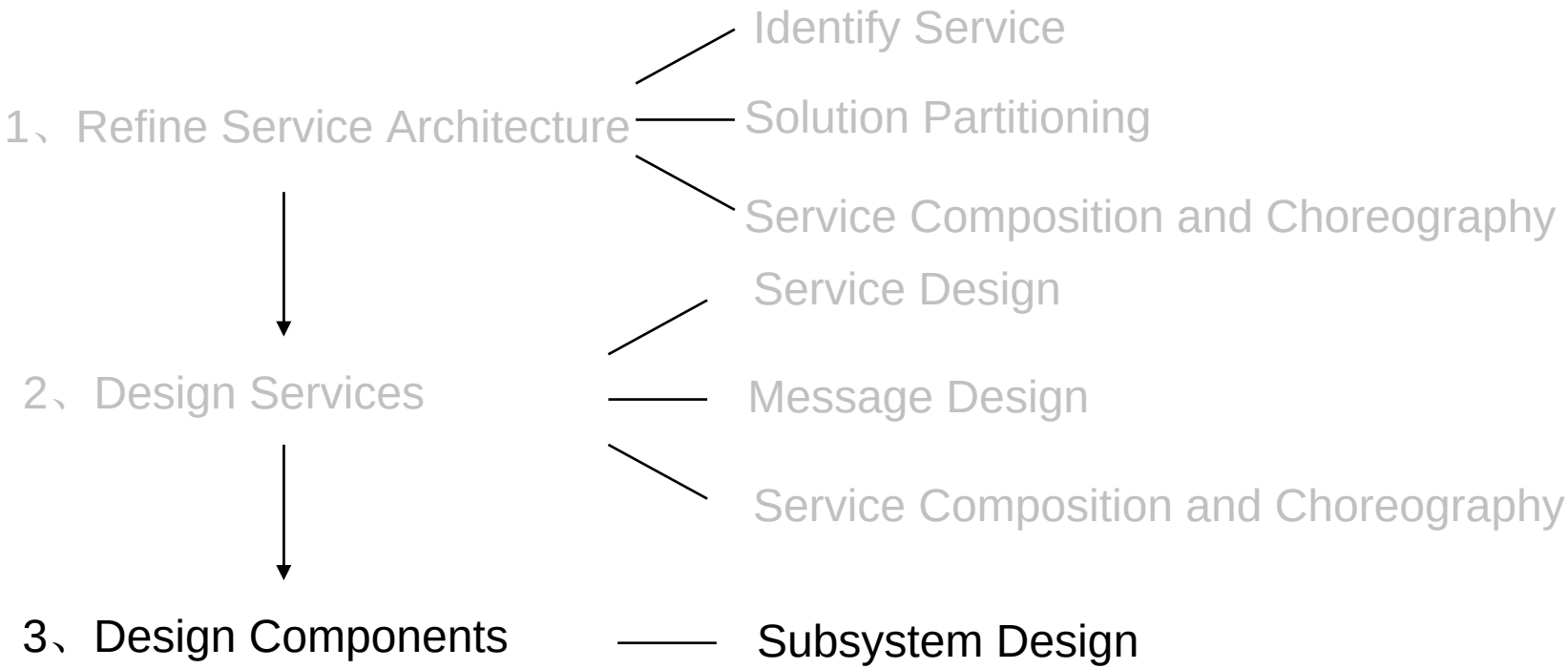
Demo6 - Service Collaboration in Design Service

通过业务流程将定义好的ServiceProvider串起来，以复审ServiceProvider的职责是否正确，以及为转变为BPEL流程打下基础



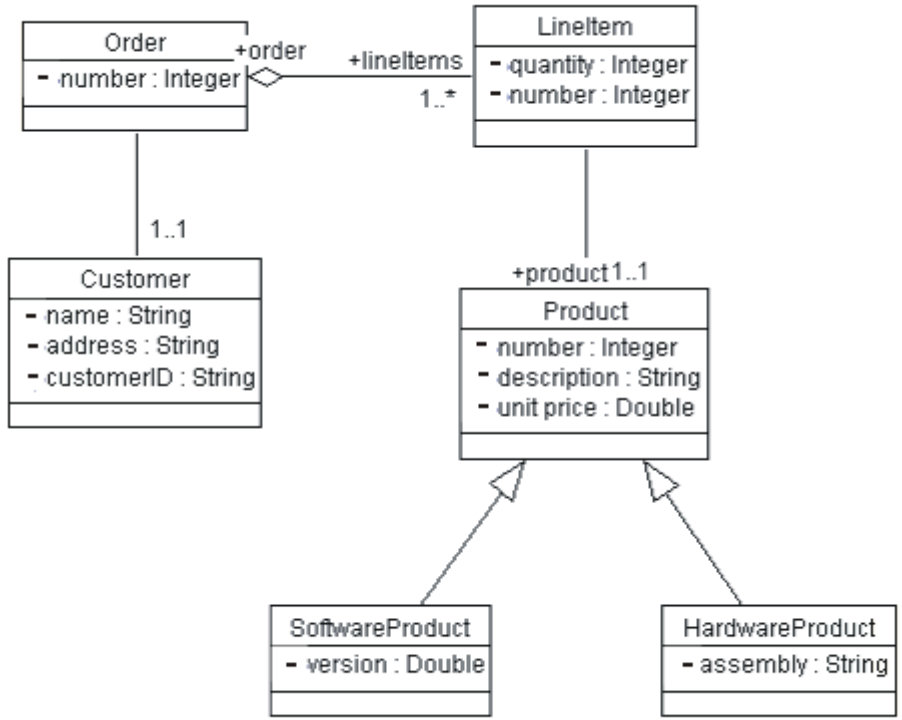
开始SOA 架构设计

二、设计步骤



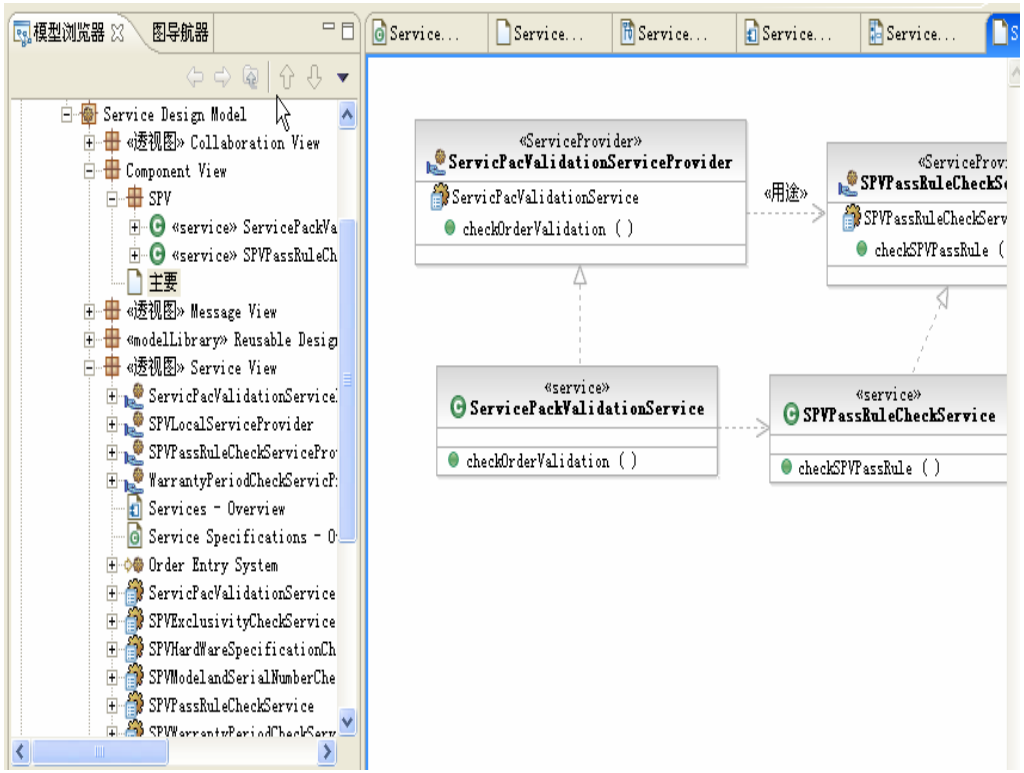
Subsystem Design

在此步具体设计实现ServiceProvider的设计类



Demo7 - Subsystem Design

在Demo里面每个ServiceProvider都由相应的Session Bean来实现




Thank
You