软件体系结构复习笔记（第二版）

根据复习课增加了部分内容。
本人不保证此复习笔记中任何内容的正确性和严谨性。
zhc
2018 年 12 月 22 日，星期六，冬至

考试时间：2018.12.26 2:00 PM-4:00 PM
考试地点：A1-306

注：这里“架构”同“构架”，代表 architecture。structure 翻译成结构。

1. 软件构架的定义：软件系统构架是该系统需要被推理出来的结构，由软件元素、软件结构间的关系以及它们的属性组成。

英文原文：The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

2. 三种结构：模块结构（module structure）、组件-连接件结构（component-and-connector structure, C&C structure）和分配结构（allocation structure）。

A. 模块结构

α 分解结构（decomposition structure）

元素类型（element type）：模块

关系（relations）：作为其子模块（is a submodule of）

用途（useful for）：资源分配以及项目结构和计划；信息隐藏、封装（encapsulation）；配置控制（configuration control）

受影响的质量属性（quality attributes affected）：可修改性

β 使用结构（use structure）

元素类型：模块

关系：使用（uses），比如需要其正确出现（presence）

用途：工程子集（engineering subsets）、工程扩展（engineering extensions）

受影响的质量属性：可分解性（“subsetability”）、可扩展性（extensibility）

γ 层结构（layer structure）

元素类型：层（layer）

关系：需要其正确出现（requires the correct presence of）、使用其服务（uses the services of）、为其提供抽象（provides abstraction to）

用途：增量式开发（incremental development）、在“虚拟机”上实现系统（implementing systems on the top of “virtual machines”）

受影响的质量属性：可移植性（portability）

δ 类/泛化结构（class/generalization structure）

元素类型：类、对象

关系：是其实例（is an instance of）、共享其访问方法（shares access methods of）

用途：用于面向对象设计的系统（in object-oriented design systems）、提取出共同点（factoring out commonality）；计划功能扩展（planning extensions of functionality）

受影响的质量属性：可修改性、可扩展性

ε 数据模型（data model）

元素类型：数据实体（data entity）

关系：一对一（one-to-one）、一对多（one-to-many）、多对一、多对多、泛化（generalizes）、特化（specializes）

用途：影响一致性和性能的工程的全局数据结构（engineering global data structures for consistency and performance）

受影响的质量属性：可修改性、性能

B. 组件-连接件结构

α 服务结构（service structure）

元素类型：服务（service）、企业服务总线（enterprise service bus, ESB）、注册表（registry）、其他

关系：与其并发运行（runs concurrently with）、可能与其并发运行（may run concurrently with）、互斥（excludes）、先于其运行（precedes）、其他

用途：调度分析（scheduling analysis）、性能分析（performance analysis）

受影响的质量属性：互操作性、性能

β 并发结构（concurrency structure）

元素类型：进程（processes）、线程（threads）

关系：可以并行（can run in parallel）

用途：分辨资源竞争或者线程分支、连接、创建或终止所在的位置（identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed）

受影响的质量属性：性能、可获得性

C. 分配结构

α 部署结构（deployment structure）

元素类型：组件（components）、硬件元素（hardware elements）

关系：被分配给（allocated to）、迁移到（migrates to）

用途：性能、可获得性和安全性分析

受影响的质量属性：性能、可获得性、安全性

β 实现结构（implementation structure）

元素类型：模块、文件结构（file structure）

关系：被存储在（stored in）

用途：配置控制（configuration control）、整合（integration）、测试活动（test

activity）

受影响的质量属性：开发效率（development efficiency）

γ 工作分配结构（work assignment structure）

元素类型：模块、组织单元（organizational units）

关系：被指派给（assigned to）

用途：项目管理（project management）、专门知识和可用资源的最佳使用（best use of expertise and available resources）、共同点管理（management of commonality）

受影响的质量属性：开发效率

参考：

[1] 1.2 架构的结构和视图 https://www.jianshu.com/p/566d99243f68

3. 架构模式：

常用的模块类型的模式：

A. 分层模式（layered pattern）

常用的组件-连接件类型的模式：

B. 数据共享模式（仓库模式，shared-data pattern，repository pattern）：这种模式由创建、储存和访问持久化数据的组件和连接件组成。仓库通常是一个（商用的）数据库。连接件是数据管理协议，例如 SQL。

C. 服务器-客户机模式（C/S 模式，client-server pattern）

常用的分配模式：

D. 多层模式（multi-tier pattern）：描述如何分配在一个不同的、被通讯媒介所连接的软硬件子集中的系统的组件（英文原文：describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium）。这种模式特化了一般的部署结构。

E. 能力中心和平台（competence center and platform）：一种特化了一个软件系统工作分配架构的模式。在能力中心，根据某个工作站内技术或领域的专长，工作被分配到不同工作站。例如，用户界面设计由易用性领域的技术人员所在的工作站来完成。在平台上，一个工作站负责开发一条产品线上的可复用核心资产，同时其他工作站开发使用这个核心资产的应用。

4. 架构的重要性：

A. 架构影响系统的驱动质量属性（driving quality attribute）。

B. 架构决策允许你随着系统发展（evolve）而推理出（reason about）和管理变更。

C. 对架构的分析使对系统质量的早期预测得以实现。

D. 被归档的架构促进（enhance）了涉众（stakeholder）间的沟通。

E. 架构是最早产生的设计决策的载体，所以它是也最基础的、最难以改变的设计决策的载体。（英文原文：The architecture is a carrier of the earliest and hence most fundamental, hardest-to-change design decisions.）

F. 架构定义了一组关于随后的实现的约束（constraint）。

G. 架构决定（dictate）了一个组织的结构，反之亦然。

H. 架构提供了进化式原型（evolutionary prototyping）的基础。

I. 架构是允许设计师和项目经理估计成本和制定进度表（reason about cost and schedule）

的关键制品（key artifact）。

J. 架构可以被设计成一个可转换的（transferable）、可复用的模型，构成产品线的心脏（heart）。

K. 基于架构的开发注重组件（component）的组合（assembly）而不是简单地关注它们的创建。

L. 通过限制设计选择（或者翻译成自由，alternative），架构引导（channel）开发者的创造力，以减少设计和系统的复杂度。

M. 架构可以是训练团队新成员的基础。

5. 软件架构的四个背景

A. 技术（technical）：软件架构在系统中扮演着什么技术角色？

最重要的技术背景因素是：

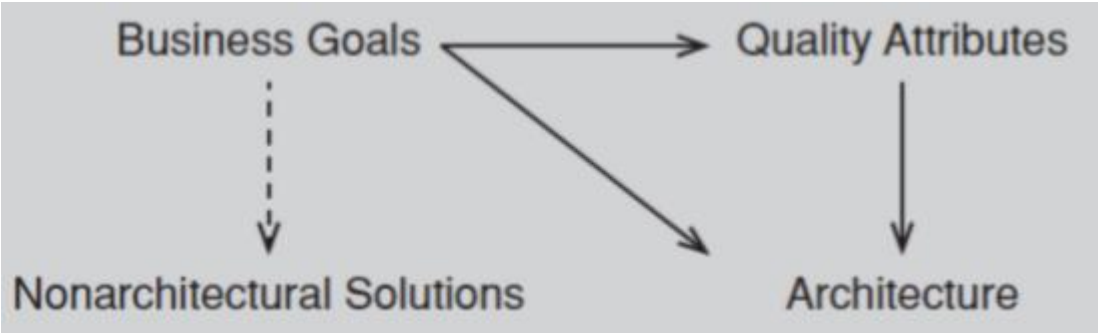
α 体系结构可以帮助实现的质量属性集。

β 体系结构的当前技术环境。

B. 项目生命周期（project life cycle）：软件架构和软件开发生命周期的其他阶段有什么关系？

有四种主要的软件开发过程（process）：瀑布（waterfall）、迭代（iterative）、敏捷（agile）和模型驱动开发（model-driven development）。

C. 业务（business）：软件架构的存在（presence）如何影响组织的业务环境。



D. 专业性（professional）：软件设计师在组织或开发项目中扮演什么角色？

架构师不仅需要知道架构，还要学会各种技巧，比如商业谈判和沟通。

E. 涉众：分析师（analyst）、架构师（architect）、业务经理（business manager）、一致性检查员（conformance checker）、客户（customer）、数据库管理员（database administrator）、部署人员（deployer）、设计师（designer）、评估员（evaluator）、实现人员（implementer）、集成人员（integrator）、维护人员（maintainer）、网络管理员（network administrator）、产品线经理（product-line manager）、项目经理（project manager）、外部系统代表（representative of external systems，负责管理与我们的系统互动的外部系统的管理员）、系统工程师（system engineer）、测试员（tester）和用户（user）

F. 什么影响架构？

α 需求影响架构。

β 是业务和社会影响的结果，也是技术的影响。

γ 体系结构的存在反过来随后影响未来体系结构的技术、业务和社会环境。

δ 特别是，体系结构的每一个背景都在影响着架构师和体系结构。

G. 架构影响什么？

各个背景，形成闭环。

6. 架构师的设计决策

A. 分配职责：

α 确定重要的职责，包括基本的系统功能、架构基础设施和要满足的质量属性。

β 确定如何将这些职责分配给非运行时和运行时元素（即模块、组件和连接件）。

B. 协调模型：

α 确定必须协调或禁止协调的系统要素。

β 确定协调的属性，如及时性、并发性、完整性、正确性和一致性。

γ 选择实现这些属性的通信机制。

C. 数据模型：

α 选择主要的数据抽象、它们的操作及其属性。这包括确定数据项是如何创建、初始化、访问、持久化、操作、翻译和销毁的。

β 数据一致性解释所需的元数据。

γ 数据组织。这包括确定数据是否将保存在关系数据库、对象集合或两者中。

D. 资源管理：

α 确定必须管理的资源并确定每个限制。

β 确定哪个系统元素管理哪个资源。

γ 确定资源共享的方式和在争用时采用的仲裁策略。

δ 确定饱和度对不同资源的影响。

E. 架构元素之间的映射：

α 将模块和运行时元素映射到彼此，即从每个模块创建的运行时元素；包含每个运行时元素的代码的模块。

β 将运行时元素分配给处理器。

γ 将数据模型中的项分配给数据存储。

δ 将模块和运行时元素映射到交付单元。

F. 绑定时间决策：

α 对于职责分配来说，您可以通过参数化构建脚本的方式来对模块进行构建时选择。

β 对于协调模型，您可以设计运行时协商的协议。

γ 对于资源管理，您可以设计一个系统来接受运行时插入的新外围设备。

δ 对于技术的选择，你可以建立一个智能手机的应用程序商店，自动下载应用程序的适当版本。

G. 技术的选择：

α 决定哪些技术可用于实现其他类别的决策。

β 确定是否支持这一技术的工具（IDE、模拟器、测试工具等）是足够的。

γ 确定技术的内部熟悉程度和外部支支持程度（例如课程、教程、示例、承包商的可获得性）。

δ 确定选择技术的副作用，如所需的协调模型或受约束的资源管理机会。

ε 确定一项新技术是否与现有的技术栈兼容。

7. 质量属性

A. 简单的概念场景（sample concrete scenario）：刺激源（source）→刺激（stimulus）→制品（artifact），环境（environment）→响应（response）→响应度量（response measure）

注：要换成具体的。

B. 可获得性（availability）：系统掩盖或修复错误的能力，使得一定时间内系统的不可用时间小于特定值。

通用场景（general scenario）：

刺激源	（内部或外部的）人、硬件、软件、物理基础设施（physical infrastructure）和物理环境（physical environment）
刺激	（错误，fault）疏忽（omission）、崩溃（crash）、时间错误（incorrect timing）和不正确的响应
制品	处理器（processor）、沟通渠道（communication channel）、持久化储存和过程（process）
环境	正常操作、启动（startup）、关闭（shutdown）、修复模式（repair mode）、降级操作（degraded operation）和重载操作（overloaded operation）
响应	阻止错误（fault）变成失败（更大的错误，failure）。 检测错误： α 把错误记录到日志中 β 通知相关的（appropriate）实体（人或系统） 修复错误： α 使事件的刺激源无法导致错误（disable source of events causing the fault） β 修复时使其暂时不可用（unavailable） γ 修复或屏蔽错误/失败或容许其导致的损坏 δ 修复时在降级模式下操作
响应度量	系统可用的最小时间或时间间隔 可用时间百分比（availability percentage），例如 99.999% 错误检测时间 错误修复时间 系统处于降级模式的时间或时间间隔 系统阻止或无失败处理（handle without failing）某类错误的比例（proportion，比如 99%）或速度（rate，比如最高 100 个每秒）

战术（tactic）：

部分战术解释：

α 状态监测：比如校验和。

β 热备份（主动冗余，active redundancy）：备份处于联机状态，当前应用系统通过高

速通信线路将数据实时传送到备份系统，保持备份系统与当前应用系统数据的同步。一旦发生灾难，不用追补或只需追补很少的孤立数据，备份系统可快速接替生产系统运行，恢复营业。

γ 暖备份（被动冗余，passive redundancy）：将备份系统已安装配置成与当前使用的系统相同或相似的系统和网络运行环境，安装了应用系统业务定期备份数据。一旦发生灾难，直接使用定期备份数据，手工逐笔或自动批量追补孤立数据或将终端用户通过通讯线路切换到备份系统，恢复业务运行。

δ 冷备份（备份，spare）：备份系统未安装或未配置成与当前使用的系统相同或相似的运行环境，应用系统数据没有及时装入备份系统。一旦发生灾难，需安装配置所需的运行环境，用数据备份介质（磁带或光盘）恢复应用数据，手工逐笔或自动批量追补孤立数据，将终端用户通过通讯线路切换到备份系统，恢复业务运行。

ε 软件升级：如微软的补丁、更新包。

ζ 忽略错误行为：比如用访问控制名单过滤拒绝服务攻击（DoS）。

η 降级：只维持最重要的系统功能，关闭不重要的，比如 Windows 的安全模式。

θ 影子：在“影子模式”下做某个操作，如果运行良好就确认这些操作，否则恢复到原来的状态，比如 Windows 安装补丁时先设还原点，失败了就恢复到这个还原点。

ι 状态再同步：指热备份和暖备份要求所恢复的组件在重新提供服务前更新其状态。在热备份中指的是备份系统接受与活动系统一样的输入；在暖备份中指的是备份系统接受的周期性更新。

κ 逐步重启：先重启对系统服务冲击少的组件，再将重启的范围扩大到对服务冲击较大的，最后重启所有组件。

λ 不间断转发：来自于路由器设计。路由器有控制平面和数据平面组成。当控制平面重启时，数据平面继续工作，转发数据包。

μ 从服务中除去：将出错的系统组件暂时移到停止服务状态，以避免（mitigate）潜在系统错误的发生。

ν 预测模型：维持系统的一些指标，以维持正常状态。

ξ 异常预防：比如智能指针和包装类。智能指针在引用计数为零时自动释放资源，避免异常。

ο 增加能力集：在组件中增加处理一些原来被认为是异常情况的能力。

参考：

[1] 热备份、温备份、冷备份(Hot/Warm/Cold Backup)
https://blog.csdn.net/u014558484/article/details/52017089

[2] 状态再同步
http://dict.youdao.com/w/%E7%8A%B6%E6%80%81%E5%86%8D%E5%90%8C%E6%AD%A5/

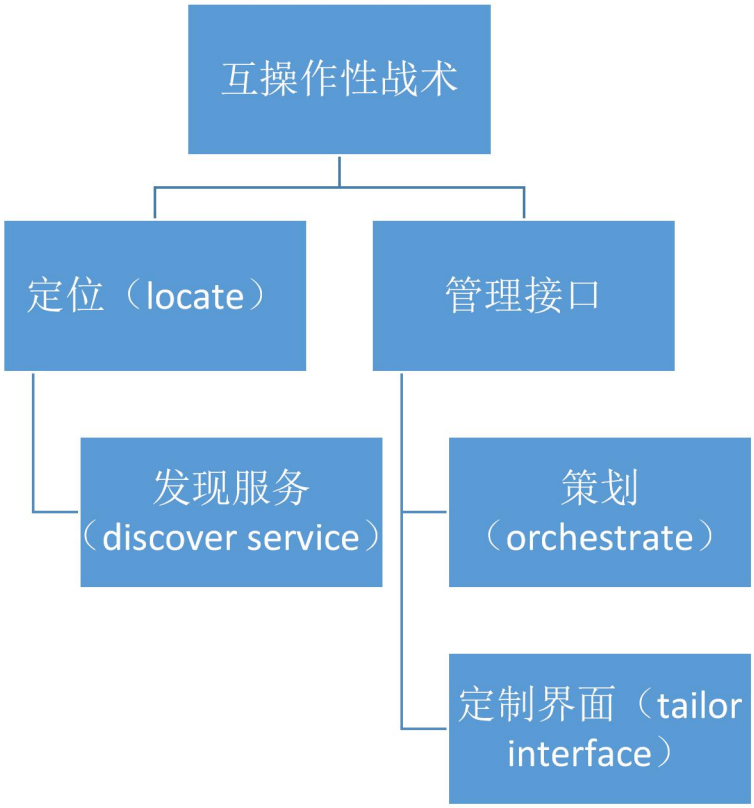
C. 互操作性（interoperability）：两个或以上的系统，在一定的背景（context）下，可以通过接口有用地交换有意义的信息。

通用场景：

刺激源	一个系统向另一个系统提出一个互操作请求（request）
刺激	一个系统间交换信息的请求
制品	想互操作的系统
环境	想互操作的系统在运行时发现对方或运行前就知晓对方的存在
响应	互操作请求引发（result in）信息交换。信息被接收方依照语法地和语义上（syntactically and semantically）理解。或者（alternatively），请求被拒绝且相应

	的（appropriate）实体被通知。无论哪种情况（in either case），请求都会被记录到日志中。
响应度量	被正确处理的信息交换的百分比或被正确拒绝的信息交换的百分比

战术：



部分战术解释：

α 发现服务：在一个已知的目录服务中定位一个服务。

β 策划：使用一个控制机制去协调、管理和排序对特定服务的调用。

γ 定制界面：从一个界面增删功能。

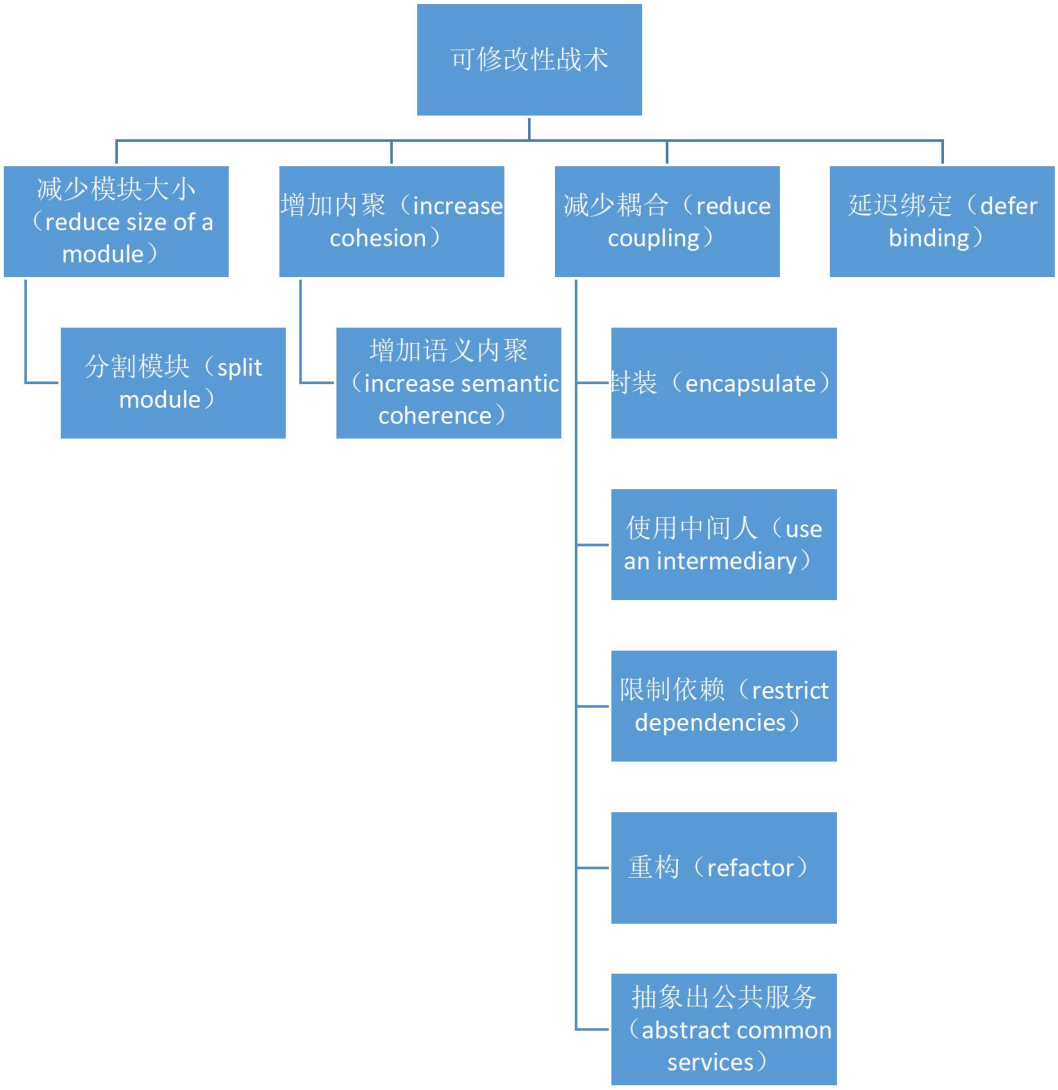
D. 可修改性（modifiability）

通用场景：

刺激源	终端用户（end user）、开发者和系统管理员
刺激	一个增删改功能或改变质量属性、能力或技术的指示
制品	代码、数据、接口、组件、资源和配置
环境	运行时、编译时、新建时（build time）、初始化时（initiation time）和设计时
响应	以下的一个或多个： α 做出修改 β 测试修改 γ 部署修改
响应度量	以下项目的代价： α 受影响制品的数量、尺寸和复杂度 β 效果 γ 日程时间（calendar time）

	δ 钱，包括直接开支（direct outlay）或机会成本（opportunity cost） ε 这次修改对其他功能或质量属性的影响范围 ζ 新缺陷的引入

战术：



部分战术解释：

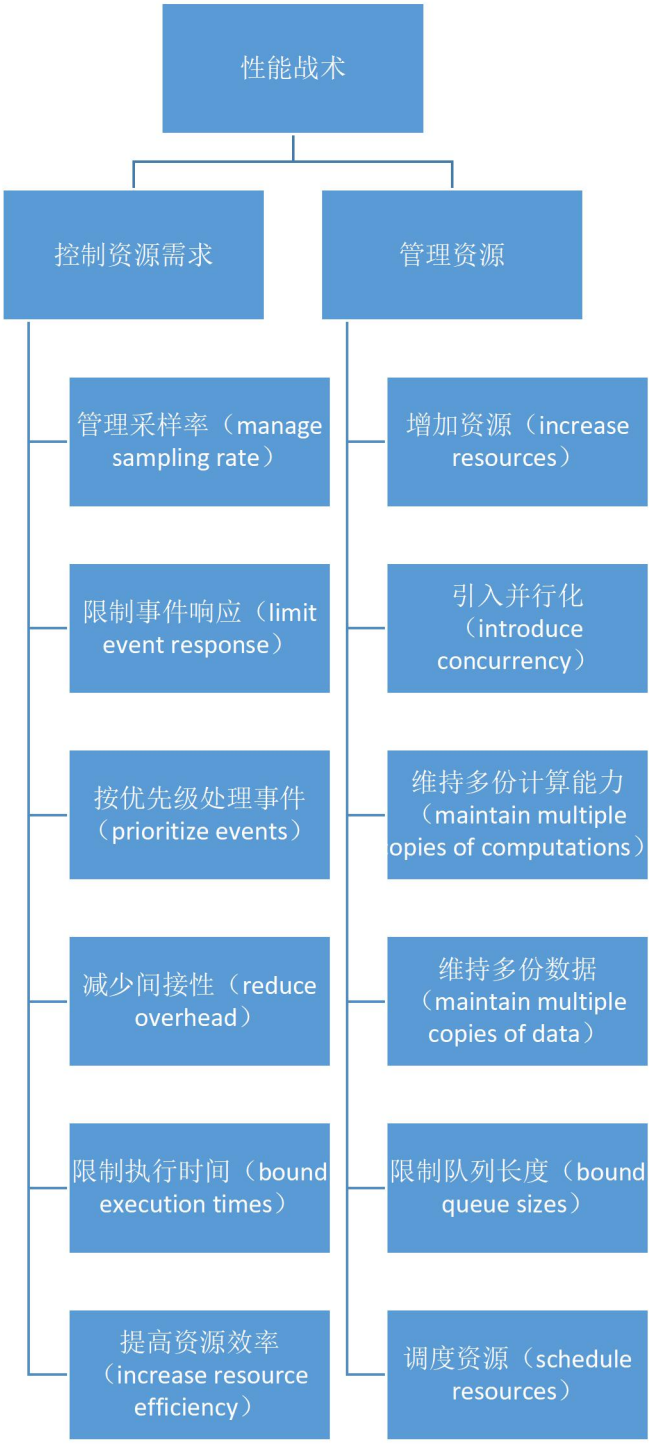
α 增加语义内聚：若模块的 A 和 B 功能不服务于同一目的，该模块就应该被分成两个。

E. 性能（performance）：软件系统满足时间需求的能力。

通用场景：

刺激源	内部或外部系统
刺激	一个周期（periodic）、偶发（sporadic）或随机（stochastic）的事件的到达
制品	系统或系统中的一个或多个组件
环境	操作模式：正常、紧急、最大负荷（peak load）和超负荷
响应	处理事件和改变服务级别
响应度量	延迟（latency）、截止时间、吞吐量、偏差（jitter）和失误率（miss rate）

战术：



部分战术解释：

α 管理采样率：比如降低音频的采样率。

β 限制事件响应：当满足一定条件时才去处理事件。

γ 维持多份数据：例如缓存。

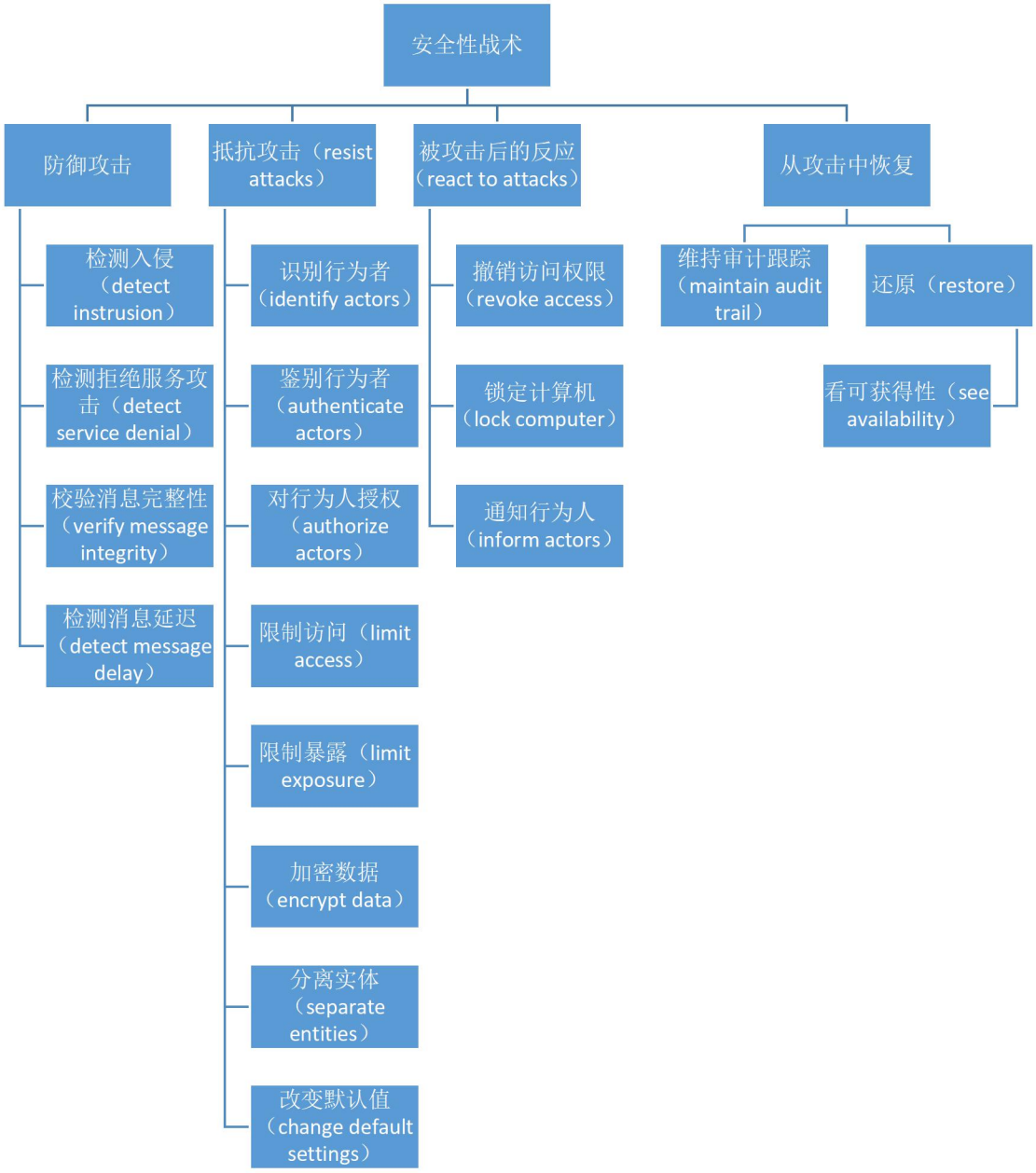
δ 限制队列长度：队列太长时丢掉部分事件。

F. 安全性（security）：系统保护数据免受未授权访问和能够被授权访问的能力，包括机密性（confidentiality）、完整性（integrity）和可获得性（availability）。

通用场景：

刺激源	未被确认或已被确认（无论是否正确）身份的人或系统。一个人类攻击者可能在组织的内部或外部。
刺激	未授权的尝试行为，其目的是显示数据、改变或删除数据、访问系统服务、改变系统行为或者降低可获得性。
制品	系统服务、系统中的数据、系统中的组件或资源以及系统生成或消费的数据
环境	在线或离线的、联网或不联网的、防火墙保护或对网络开放的以及完全可用、部分可用或不可用的系统
响应	执行相应事务以达成以下目的： α 数据或服务免受未授权访问。 β 数据或服务免受未授权操纵。 γ 事务方（party to a transaction）保证被辨明身份。 δ 事务方不能拒绝它们的参加（involvement）。 ε 数据、资源和系统服务在合法目的下（for legitimate use）可用。 系统通过以下手段跟踪： α 记录访问或修改。 β 记录访问数据、资源或服务的尝试。 γ 当一个明显的攻击发生时，通知相关实体（人或系统）。
响应度量	以下的一个或多个： α 一个系统折中（compromise）了多少，当特定组件或数据被折中时 β 检测攻击所需时间 γ 能防御多少攻击 δ 被成功攻击需要多少时间恢复 ε 面对特定攻击，多少数据是脆弱的

战术：



部分战术解释：

α 检测拒绝服务攻击：检测输入流量是否符合已知的拒绝服务攻击（DoS）的模式。

β 校验消息完整性：比如校验和以及散列。

γ 检测消息延迟：消息延迟可能暗示有潜在的中间人攻击。

δ 识别行为者：如用户名、访问码、IP 地址、协议、端口等。

ε 鉴别行为者：如密码、一次性密码、数字证书和生物特征（biometric identification）。

ζ 对行为人授权：常通过对用户分类来实现。

η 限制访问：比如内存保护、屏蔽某些主机、关闭端口或拒绝协议。

θ 限制暴露：将攻击面（可能被攻击的元素）最小化。

ι 分离实体：比如物理隔离不同网络的服务器、使用虚拟机、把敏感数据和不敏感数据分开等。总之，把系统分成多个部分。

κ 撤销使用权限：必要时甚至可以撤销合法用户为合法使用的权限。

λ 锁定计算机：在一定时间内，限制每台电脑用同一个用户的错误登陆次数。

参考：

[1] 拒绝服务攻击

https://baike.baidu.com/item/%E6%8B%92%E7%BB%9D%E6%9C%8D%E5%8A%A1%E6%94%BB%E5%87%BB

[2] 中间人攻击

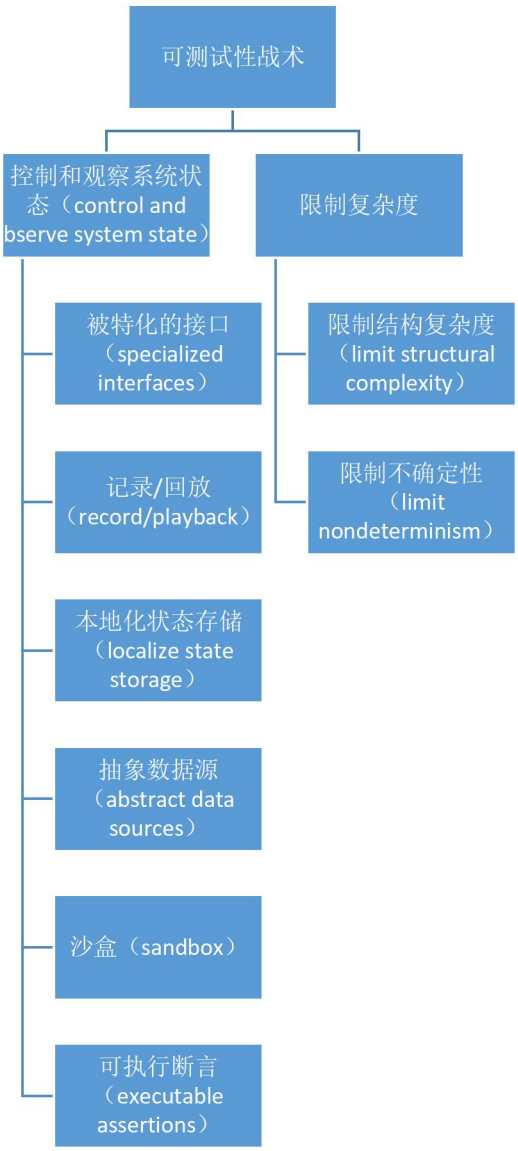
https://baike.baidu.com/item/%E4%B8%AD%E9%97%B4%E4%BA%BA%E6%94%BB%E5%87%BB

G. 可测试性（testability）：软件可以被证明有错误的容易程度。

通用场景：

刺激源	单元测试员（unit tester）、集成测试员（integration tester）、系统测试员（system tester）、验收测试员（acceptance tester）以及终端用户，无论是手工测试还是运用自动化测试工具（automated testing tool）
刺激	由于一段代码增量的完成而执行的一系列测试。代码增量可以是一个类、层或服务，子系统的完全整合，整个系统的完全实现，或者是系统交付给客户。
制品	被测试的部分系统
环境	设计时、开发时、编译时、整合时、部署时和运行时
响应	以下的一个或多个：执行测试套件并捕获结果、捕获导致错误的活动、控制和监视系统状态
响应度量	以下的一个或多个：成功找到一个或一类错误、覆盖了一定百分比的状态空间、下次测试揭示错误的概率（probability of fault being revealed by the next test）、测试执行时间、检测错误的效果、测试中最长依赖链的长度、准备测试环境的时长以及风险暴露的减量（损失大小乘损失概率，即损失期望）

战术：



部分战术解释：

α 被特化的接口：特化的测试接口，可以让测试员控制或捕获组件中的变量值，无论是在测试还是正常执行。

β 记录/回放：系统状态难以重新生成，所以把状态记录下来，多次测试。

γ 本地化状态存储：用于分布式系统的状态。如果调试分布式系统时，涉及多个组件，如果有不在本地的，难以获得其状态。所以要本地化其状态。

δ 沙盒：新建一个易于恢复的系统实例用于测试。常用虚拟机。

ε 可执行断言：在程序特定位置中插入代码，判断系统是否满足特定条件（如变量的范围等），从而确定其是否正常工作。

参考：

[1] 沙盒 https://baike.baidu.com/item/%E6%B2%99%E7%9B%92/3769297

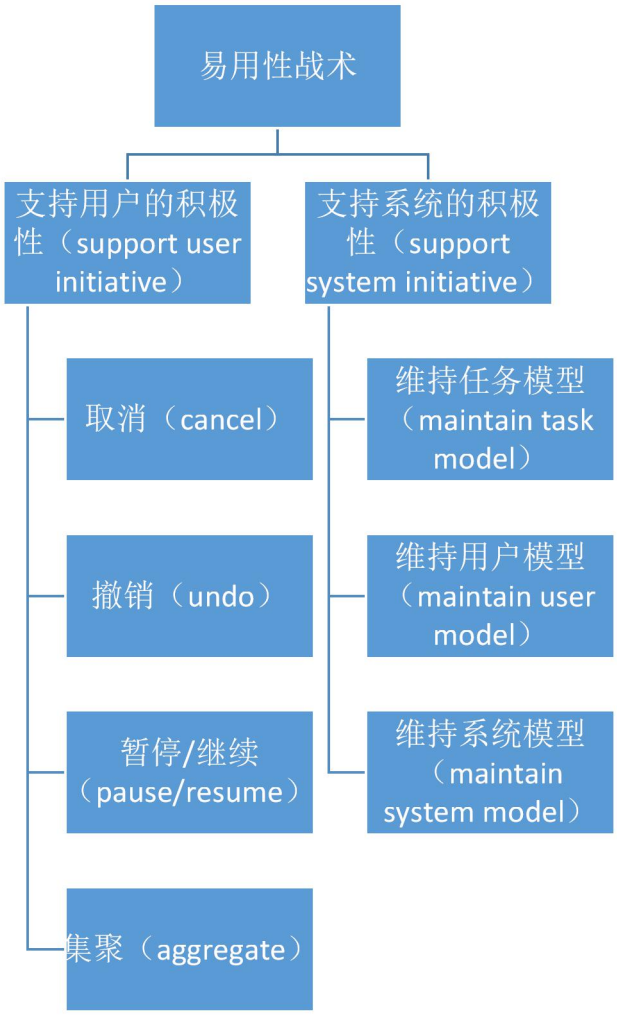
H. 易用性（usability）：指用户完成一项任务的容易程度和系统所提供用户支持的种类。

通用场景：

刺激源	终端用户，可能是一个被特化的角色
刺激	终端用户尝试高效地使用系统、学会使用系统、最小化错误（error）的冲击、适

	应系统或者配置系统
制品	系统或系统中专门与用户交互的部分
环境	运行时或配置时
响应	系统应该提供用户所需的特性或预料到用户的需求
响应度量	以下一个或多个：任务时间、错误数、任务完成数、用户满意度、用户知识获取、操作成功率或出错后的时间或数据损失

战术：



部分战术解释：

α 取消：用户下令取消时，系统要接受取消命令，终止被取消的命令，等等。

β 集聚：即允许批量（针对多个对象的）操作。

γ 维持任务模型：确定背景（语境）来得知用户意图，从而帮助用户。例如，知道英文句子首字母要大写，从而自动纠正用户句首的小写字母。

I. 其他质量属性：易变性（variability）、可移植性（portability）、开发分布性（development distributability）、可伸缩性（scalability）、部署性（deployability）、移动性（mobility）、可监控性（monitorability）、（物理实体上的）安全性（safety）。

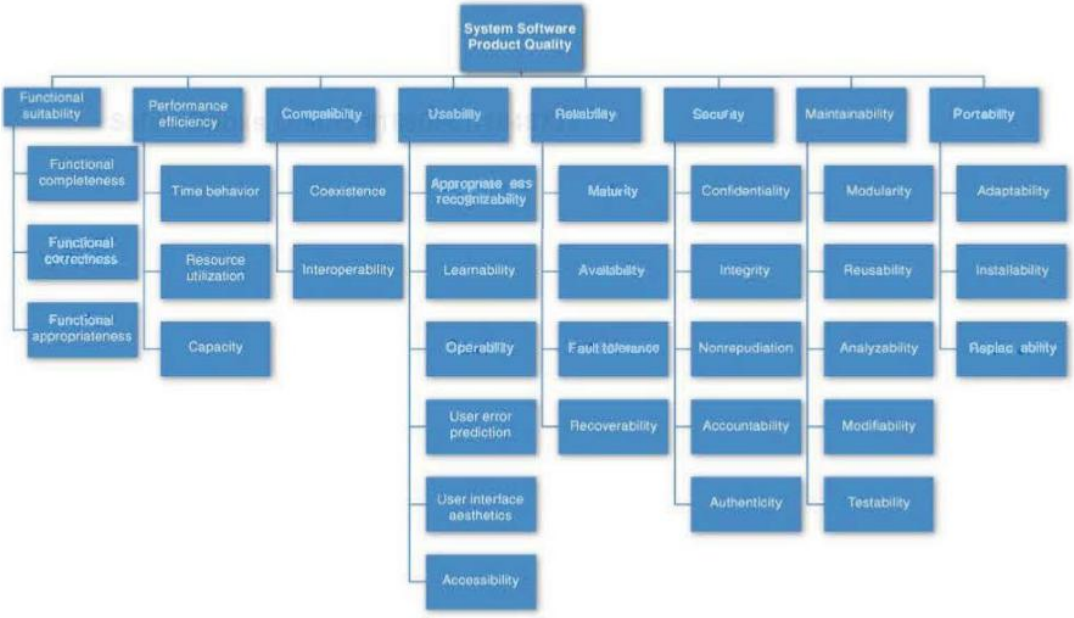


Figure 12.1. The ISO/IEC FCD 25010 product quality standard

J. X 质量属性（X-ability）

指未列出的的质量属性。

如何添加新质量属性：

- α 为新质量属性捕获场景
- β 为新质量属性收集设计方法（design approach）
- γ 为新质量属性建模
- δ 为新质量属性收集一组战术
- ε 为新质量属性构建设计检查表（design checklist）

8. 架构模式（architectural pattern）

什么是模式：a.是一个在实践中不断使用的设计决策包； b.允许质量属性重用； c.定义了一类架构。

模式建立了背景（context）、问题（problem）和解决方案（solution）间的关系。

关于模块的模式（module pattern）：

A. 分层模式（layered pattern）

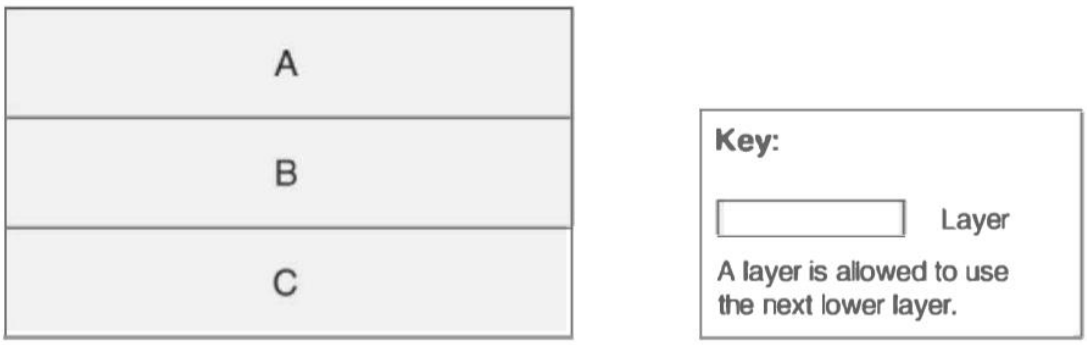


Figure 13.1. Stack-of-boxes notation for layered designs

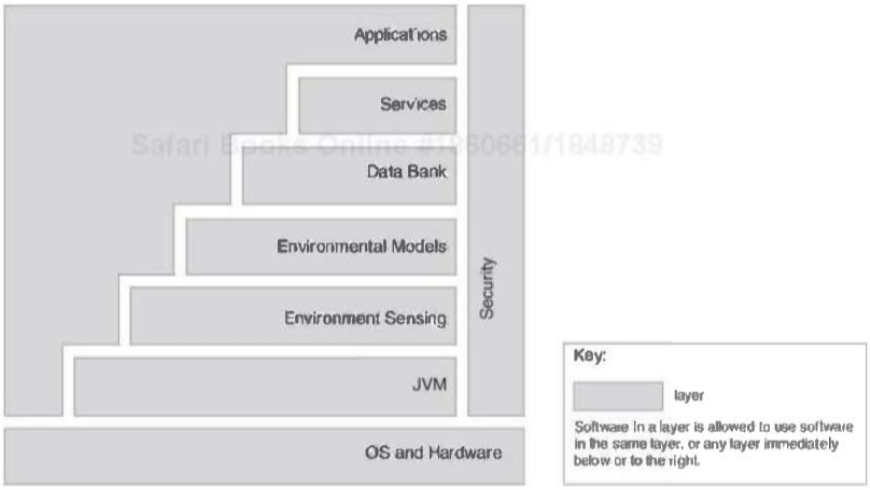


Figure 13.2. A simple layer diagram, with a simple key answering the uses question

解决方案：

概览（overview）	分层模式定义了层（提供紧密结合的一系列服务的模块分组，groupings of modules that offer a cohesive set of services）和层与层之间单向调用的关系（a unidirectional allow-to-use relation among the layers）。这个模式通常以堆箱子在其他箱子上面代表层在其它层上面的方式展示。（The pattern is usually shown graphically by stacking boxes representing layers on the top of each other.）
元素（element）	层，一种模块。对层的描述应该定义这层包含什么模块以及对层所提供的紧密结合的一系列服务的刻画（characterization）。
关系（relation）	允许使用，这是一种更一般的依赖关系的特殊形式（specialization）。设计应该定义层的使用规则（比如“层可以使用比它低级的所有层”或“层只能使用比它低一级的层”）和所有被允许的异常。
约束（constraint）	α 软件的每个部分都会被分配到一个层。 β 至少有 2 个层。 γ 只能上层用下层，下层不能用上层。
弱点（weakness）	α 添加层会增加系统的前期（up-front）成本和复杂性。 β 层会造成性能损失。

关于组件-连接件的模式（component-and-connector pattern）：

B. 经纪人模式（broker pattern）

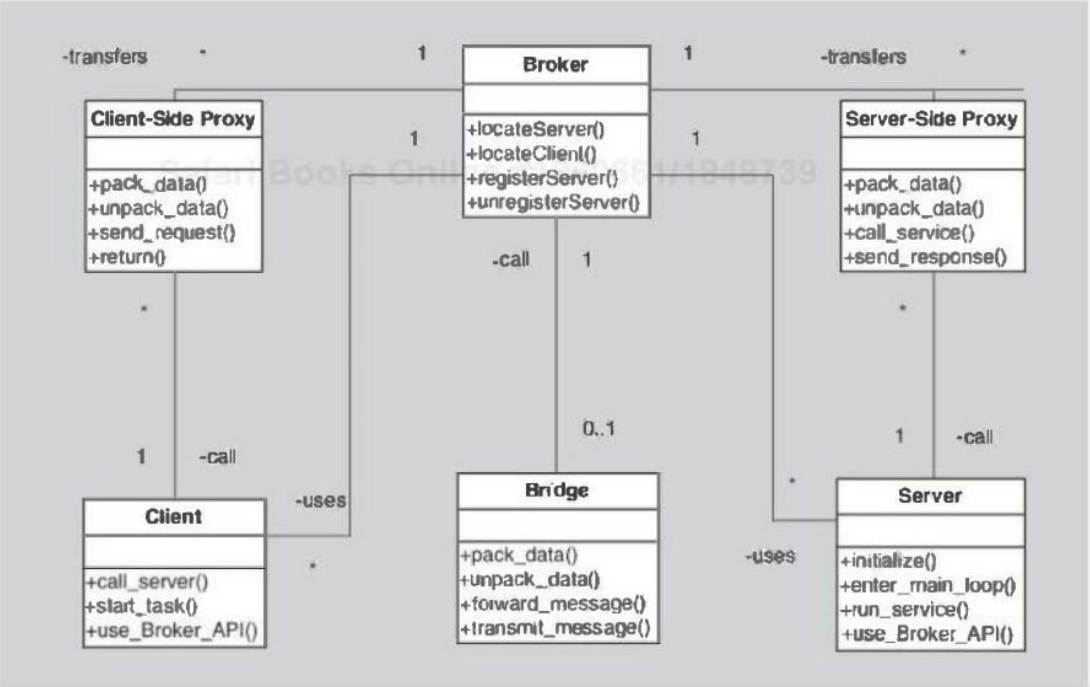


Figure 13.6. The broker pattern

解决方案：

概览	经纪人模式定义一个叫经纪人的运行时组件，作为连接许多服务器和客户机的媒介。
元素	客户机（client），服务请求人 服务器（server），服务提供者 经纪人（broker），一个交互媒介，负责定位相应的服务器来满足一个客户机的请求，转发请求到服务器，以及返回结果给客户机 客户机代理（client-side proxy），一个交互媒介，负责管理与经纪人真实的通讯，包括整理（marshal）、发送和逆整理（unmarshal）消息 服务器代理（server-side proxy），一个交互媒介，负责管理与经纪人真实的通讯，包括整理（marshal）、发送和逆整理（unmarshal）消息
关系	依附关系（attachment relation）使客户机（可以加上客户机代理）和服务器（可以加上服务器代理）与经纪人有联系。
约束	客户端和服务器都只能接触到代理。
弱点	α 代理在客户端和服务器之间添加了一层间接寻址，会导致延迟，可能是通信瓶颈。 β 代理可以是单点故障。 γ 增加了前期（up-front）的复杂性。 δ 代理可能是安全攻击的目标。 ε 代理可能很难测试。

C. 模型-视图-控制器模式（model-view-controller pattern, MVC pattern）

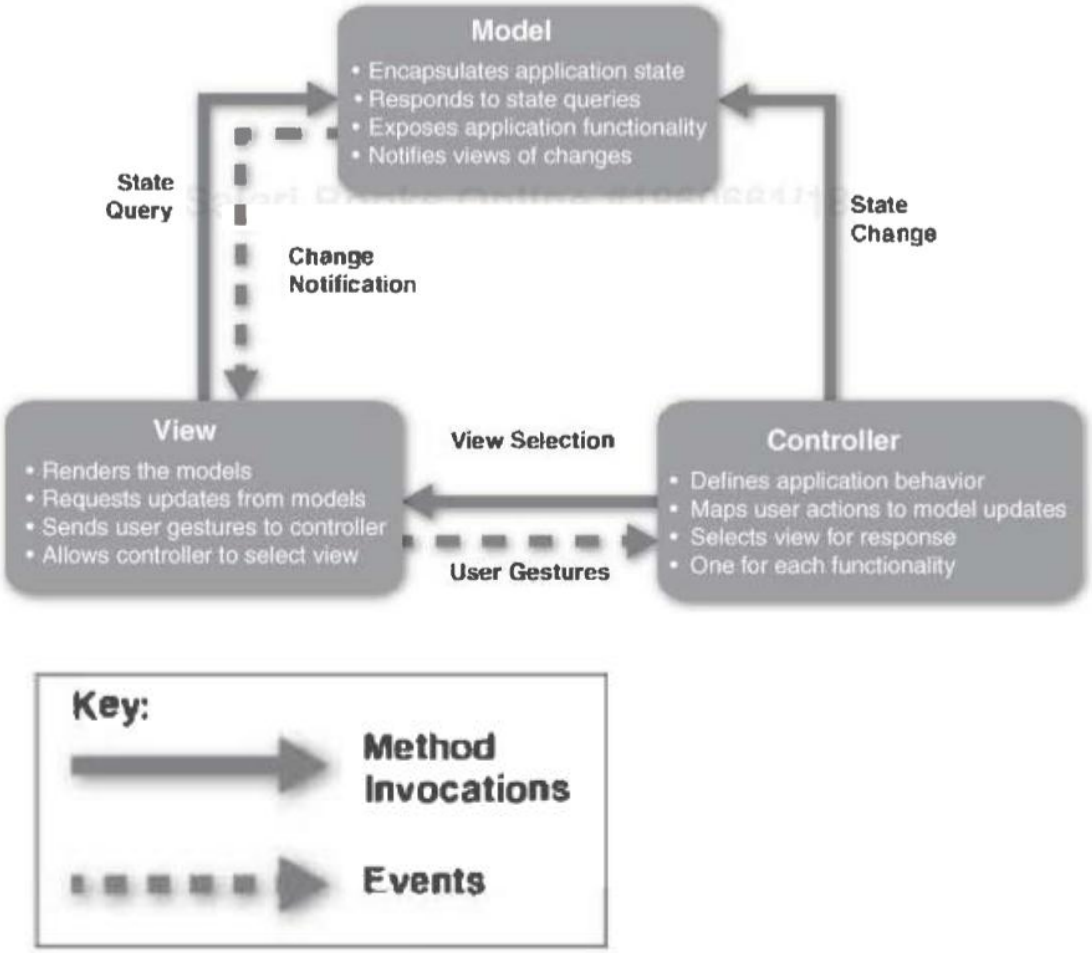


Figure 13.7. The model-view-controller pattern

解决方案：

概览	MVC 模式把系统功能分成 3 个组件：模型、视图以及模型和视图间的控制器。
元素	模型（model），代表应用数据或状态，并包含（或提供接口给）应用逻辑。 视图（view），用户接口组件，将模型展示给用户（produce a representation of the model for the user）或允许某些形式的输入，或两者兼有。 控制器（controller），管理模型和视图间的互动，将用户操作转化成模型或视图的转换。
关系	通知关系，连接模型、视图和控制器的实例，通知相关状态改变的元素。
约束	α 三者至少每者有一个实例。 β 模型不能直接和控制器交互。
弱点	α 对于一一些简单的用户界面，不值得用这么复杂的东西。 β MVC 抽象可能不适合某些用户界面的工具包。

D. 管道-过滤器模式（pipe-and-filter pattern）

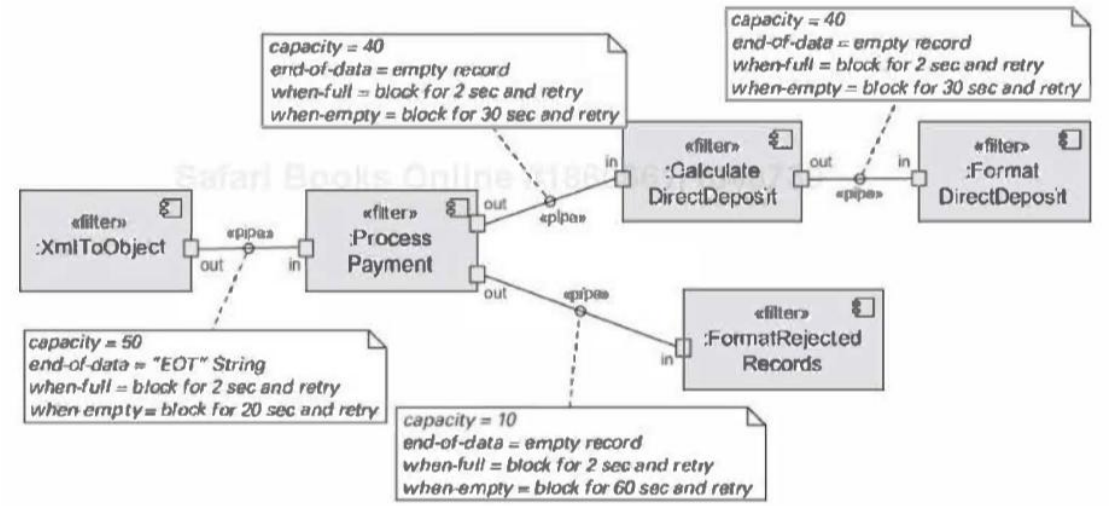
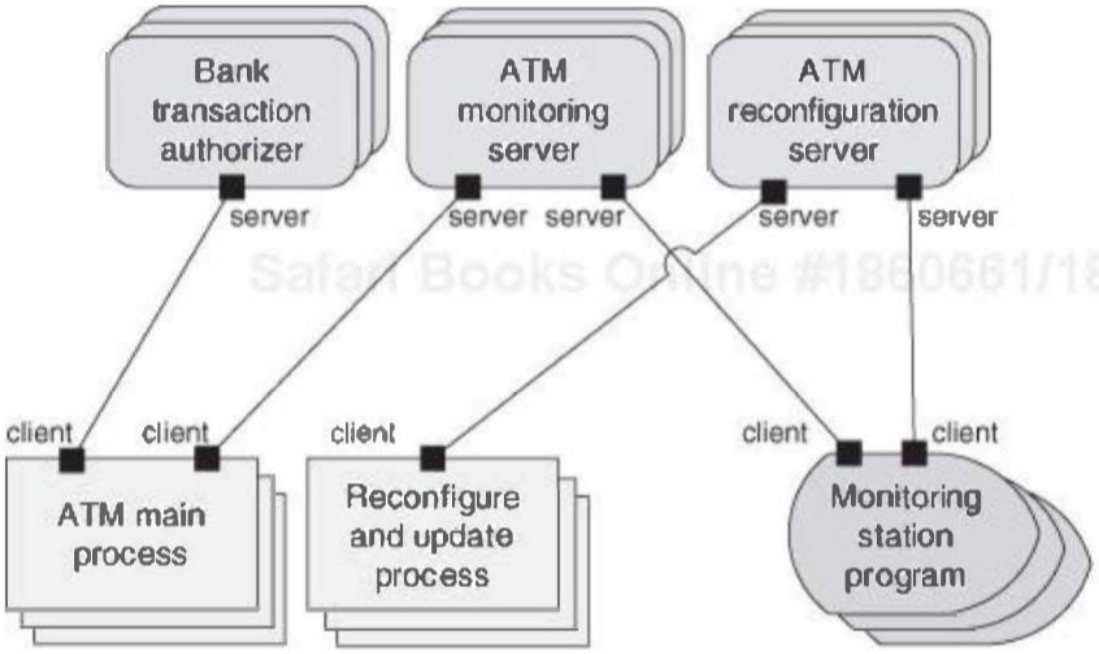


Figure 13.8. A UML diagram of a pipe-and-filter-based system

解决方案：

概览	数据从系统的外部输入，通过被管道连接的过滤器的一系列转换后输出到系统外部。
元素	过滤器，转化数据。 管道，在不同过滤器间传输数据。
关系	过滤器的输出和管道的输入的相互依附关系。
约束	α 过滤器输出端口连到管道输入端口。 β 管道和过滤器之间的数据类型必须吻合。
弱点	α 对交互系统来说不是典型的好选择。 β 大量独立的过滤器会增加许多计算花费。 γ 可能不适用于长时间运行的计算（long-running computations）。

E. 服务器-客户机模式（client-server pattern）



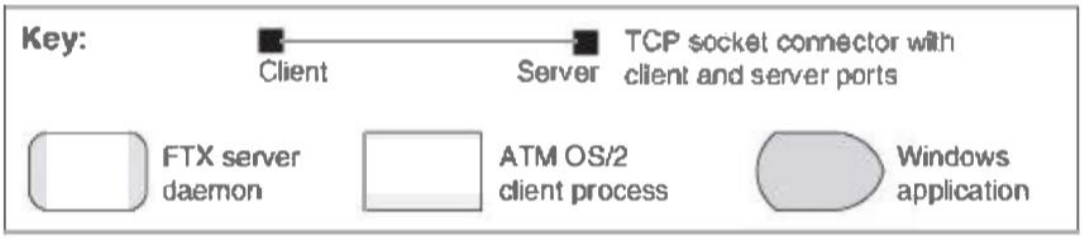
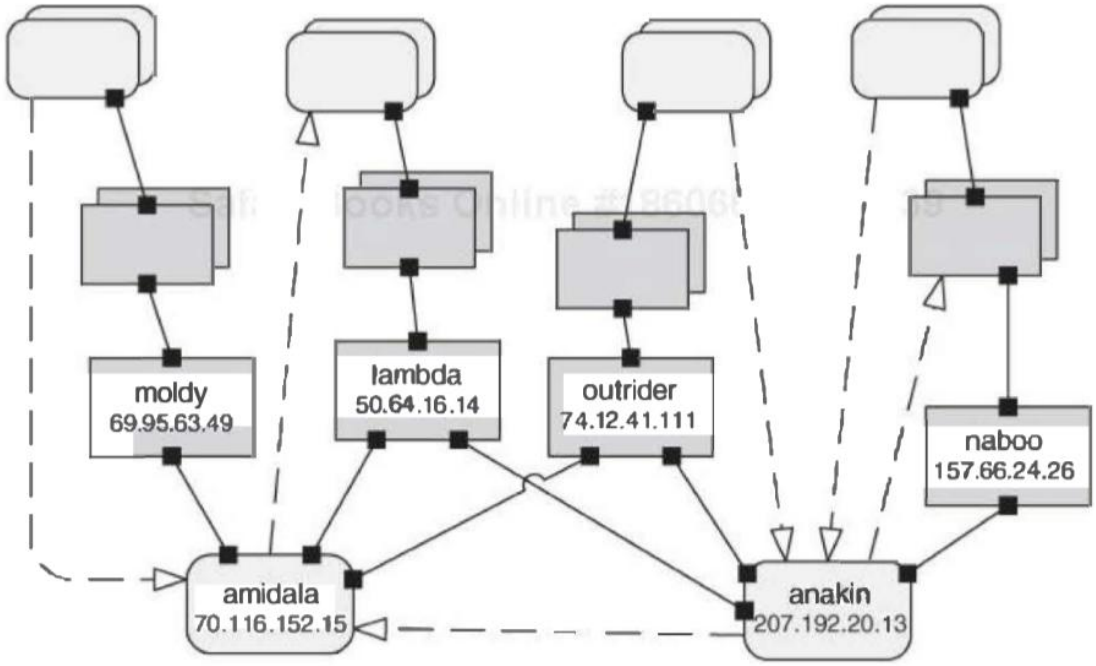


Figure 13.9. The client-server architecture of an ATM banking system

解决方案：

概览	客户机发起与服务器的互动，要求（invoke）所需的服务给服务器并等待其请求的结果。
元素	客户机 服务器 请求/应答连接件（request/reply connector），一个使用请求/应答协议的数据连接件
关系	服务器和客户机的依附关系
约束	α 客户机通过请求/应答连接件（request/reply connector）与服务器相连。β 服务器也可以是别的服务器的客户机。
弱点	α 服务器可能造成性能瓶颈。 β 服务器可能发生单点故障。 γ 关于把功能放到哪里（在客户机或服务器中）的决策通常是复杂的，并且在系统生成后更改成本很高。

F. 对等模式（peer-to-peer pattern, P2P pattern）



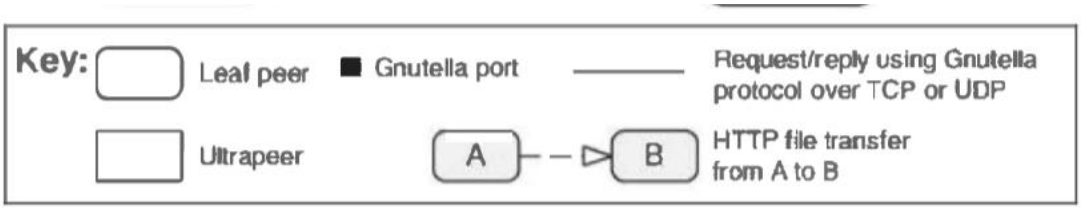


Figure 13.10. A peer-to-peer view of a Gnutella network using an informal C&C notation. For brevity(简洁), only a few peers are identified. Each of the identified leaf peers uploads and downloads files directly from other peers.

解决方案：

概览	对等节点（peer）相互合作，在网络上请求和提供服务，从而完成计算。
元素	对等节点（peer），网络节点上的独立组件。 请求/应答连接件，用于连接对等网络，搜索其他对等节点，并请求其他对等节点的服务。
关系	对等节点及其连接件间的关系。
约束	α 一个给定的 peer 所允许连接的 attachments 是受限数量的。 β 搜索一个 peer 所用跳数也是受限的。 γ 哪些 peer 知道哪些 peer。 δ 有些 P2P 网络是用星型拓扑结构组织的，其中的对等节点只连接到 supernodes（比如采用 NAT 的局域网）。
弱点	α 管理安全性、数据一致性、数据/服务可获得性、备份和恢复都比较复杂。 β 小型对等系统可能无法始终如一地实现质量目标，如性能和可获得性。

G. 面向服务的架构模式（service-oriented architecture pattern）

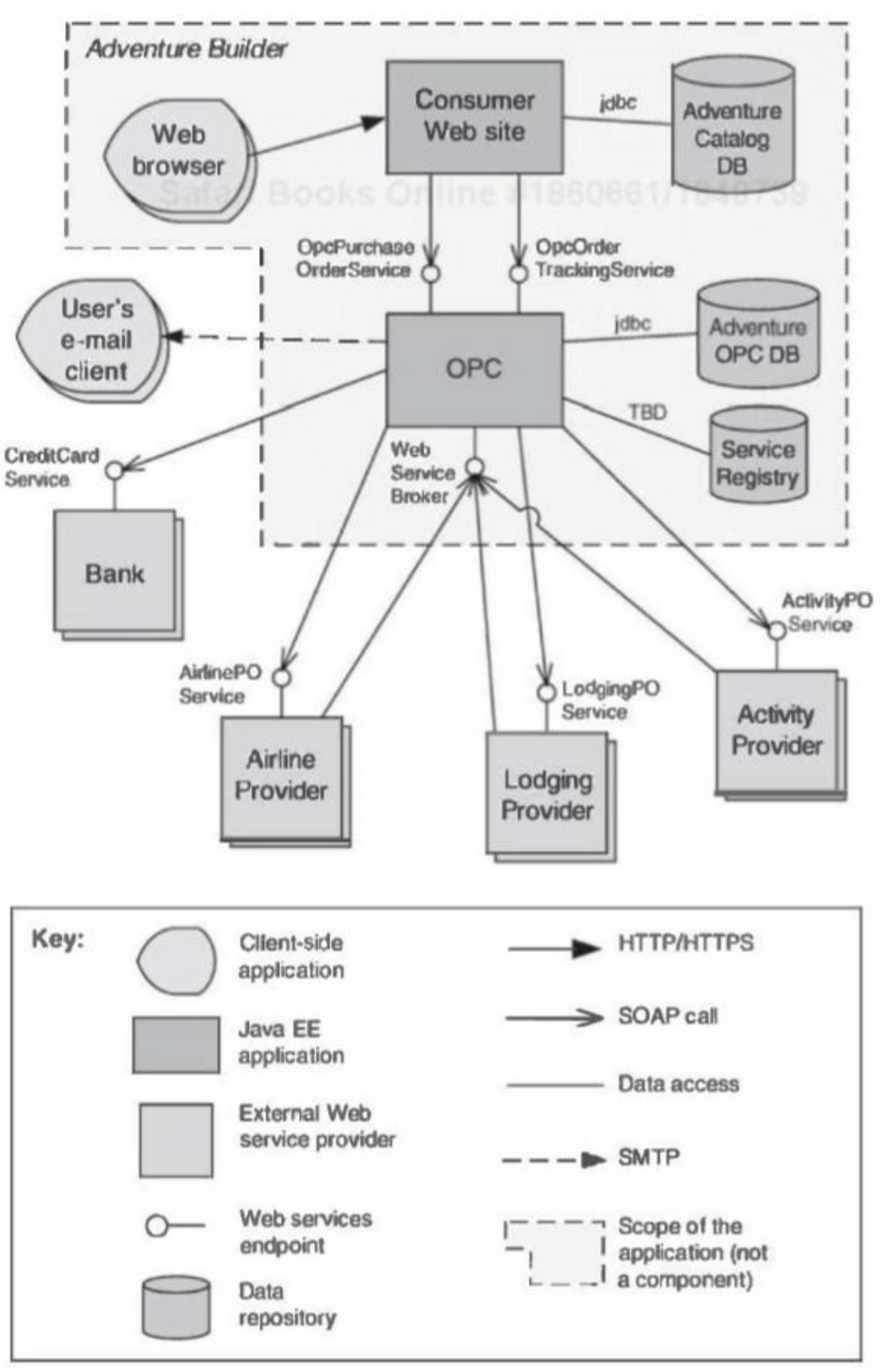


Figure 13.11. Diagram of the SOA view for the Adventure Builder system. OPC stands for "Order Processing Center."

解决方案：

概览	网络中，一系列相互合作的组件通过提供或（并）消费服务来完成计算。计算过程通常用工作流语言描述。
元素	组件： α 服务提供者 β 服务消费者 γ 企业服务总线（enterprise service bus, ESB），为服务提供者和消费者间的消息提供路由和传输功能 δ 服务注册处（registry of services），供服务提供者注册服务以及消费者寻找服务 ε 协调服务器（orchestration server），基于业务处理和工作流语言来协调服务提供者和消费者的互动 连接件： α SOAP 连接件（SOAP connector），使用 SOAP 协议来进行万维网（web）服务间同步通信，往往基于 HTTP 协议 β REST 连接件，依赖 HTTP 协议的基本的请求/应答操作 γ 异步消息连接件（asynchronous messaging connector），其使用一个消息系统来提供点对点（point-to-point）或发布-订阅（publish-subscribe）异步消息交换。
关系	Attachment of the different kinds of components available to the respective connectors
约束	服务使用者连接到服务提供商，但可以使用中间组件（如 ESB、注册表、业务流程服务器）。
弱点	α 基于面向服务架构的系统通常是复杂的。 β 你不能控制独立服务的发展。 γ 与中间件相关的性能开销和服务可能是性能瓶颈，通常不提供性能保证。

H. 发布-订阅模式（publish-subscribe pattern）

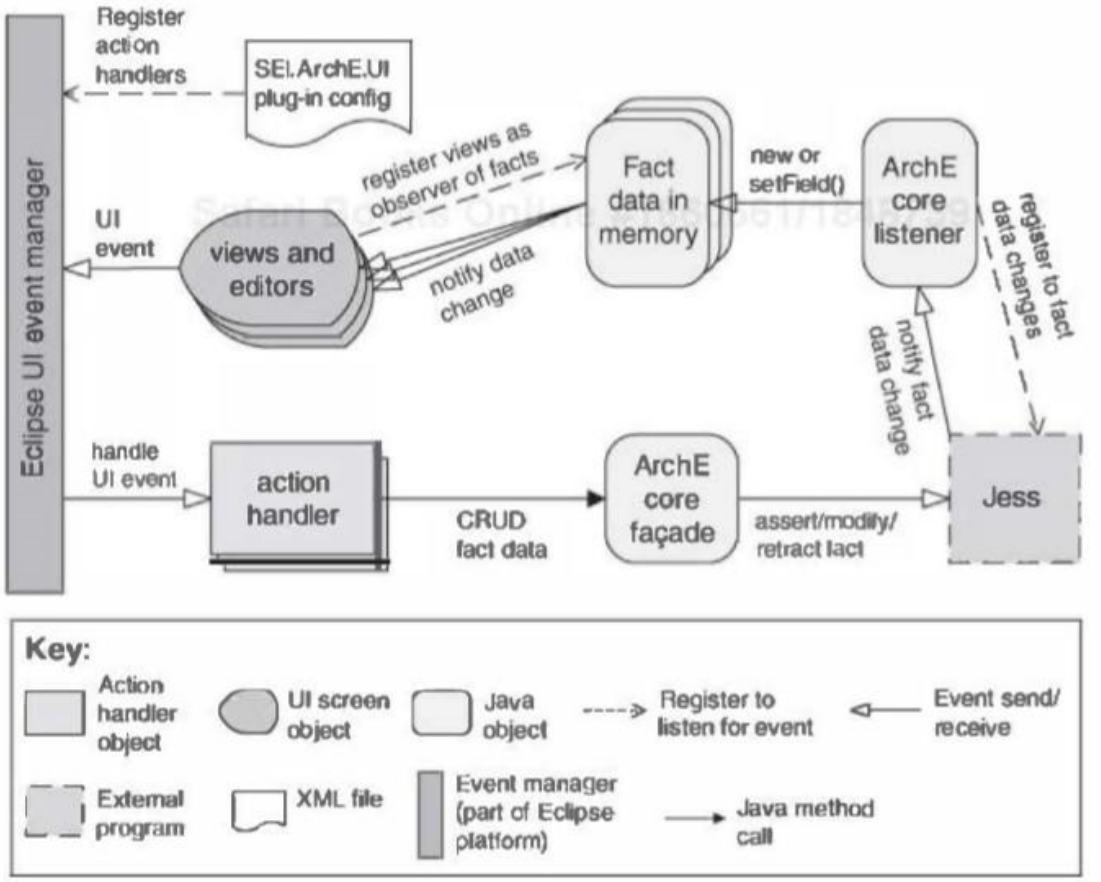


Figure 13.12. A typical publish-subscribe pattern realization

解决方案：

概览	组件发布和订阅事件。当一个组件发布事件时，连接件基础设施（connector infrastructure）将事件转发给所有注册的订阅者。
元素	所有 C&C 组件至少有一个发布或订阅端口。关注（concern）包括被分布或订阅的事件以及事件的粒度。
关系	The attachment relation associates components with the publish-subscribe connector by prescribing which components announce events and which components are registered to receive events.
约束	所有组件都连接到一个事件分发服务器，可以将其视为总线连接器（publish-subscribe connector）。
弱点	α 通常会增加延迟，并对消息传递时间的可伸缩性和可预测性产生负面影响。 β 减少了对消息排序的控制，并且不保证传递消息。

I. 数据共享模式（shared-data pattern）

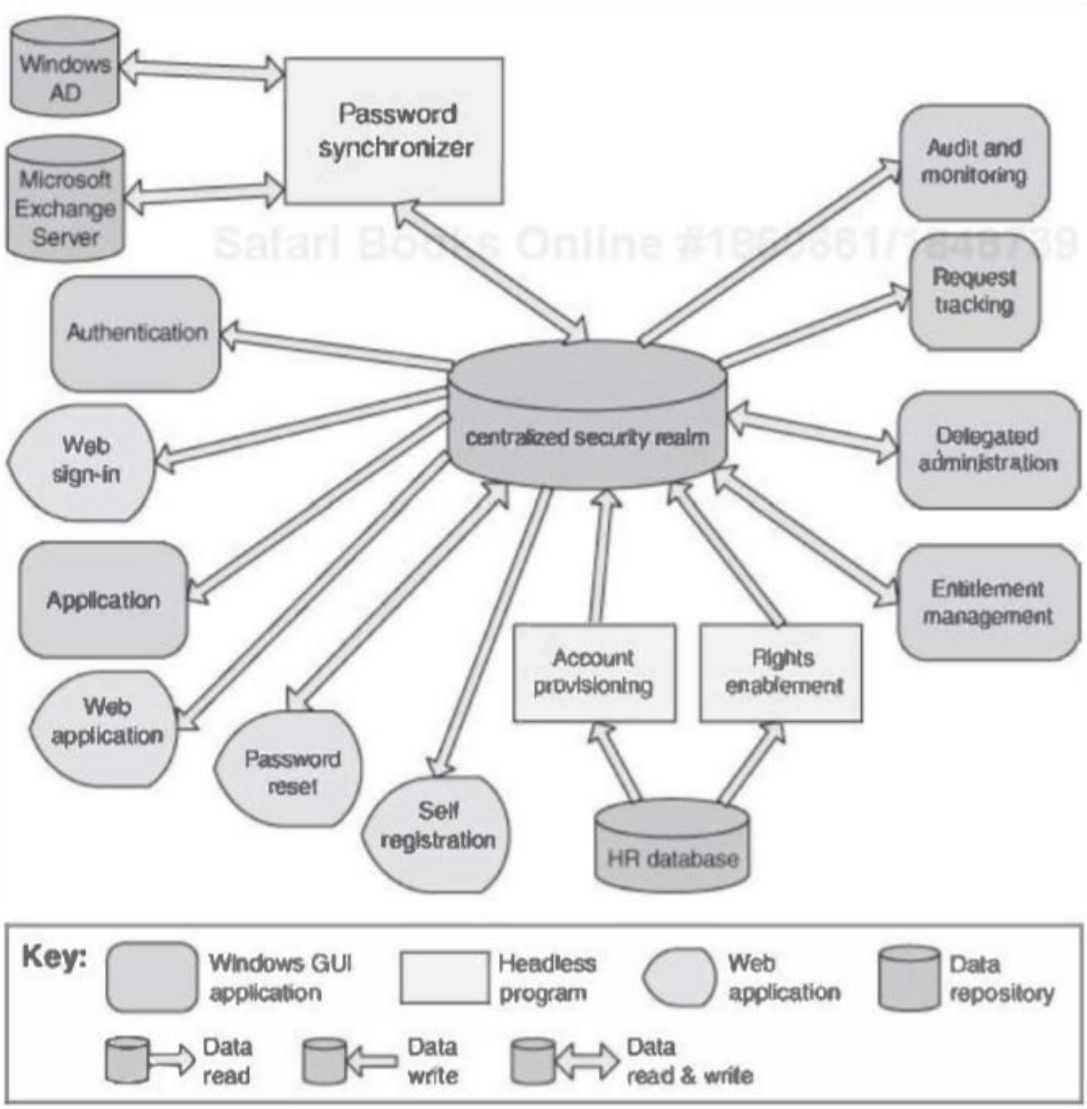


Figure 13.13. The shared-data diagram of an enterprise access management system

解决方案：

概览	数据访问者（data accessors）间的通信都通过一个共享数据存储中心（shared-data store）进行。控制权被数据访问者或数据存储中心所有（Control may be initiated by the data accessors or the data store）。数据被永久存储在数据存储中心。
元素	共享数据中心（shared-data store），储存了所存储的数据类型、数据面向性能的属性（data performance-oriented property）、数据分布和已授权用户的数量 数据访问者组件（data accessor component） 数据读写连接件（data reading and writing connector），一个重要的选择是连接件是否是事务的，读写语言、协议和语义学（an important choice here is whether the connector is transactional or not, as well as the read/write language, protocols, and semantics）
关系	Attachment relation determines which data accessors are connected to which data store(s).
约束	数据访问器仅与数据存储区交互。
弱点	α 共享数据存储中心可能是性能瓶颈。 β 共享数据存储中心可能会发生生单点故障。

	γ 数据的生产者和消费者可能是紧密耦合的。

分配模式（allocation pattern）：

J. 映射-规约模式（map-reduce pattern）

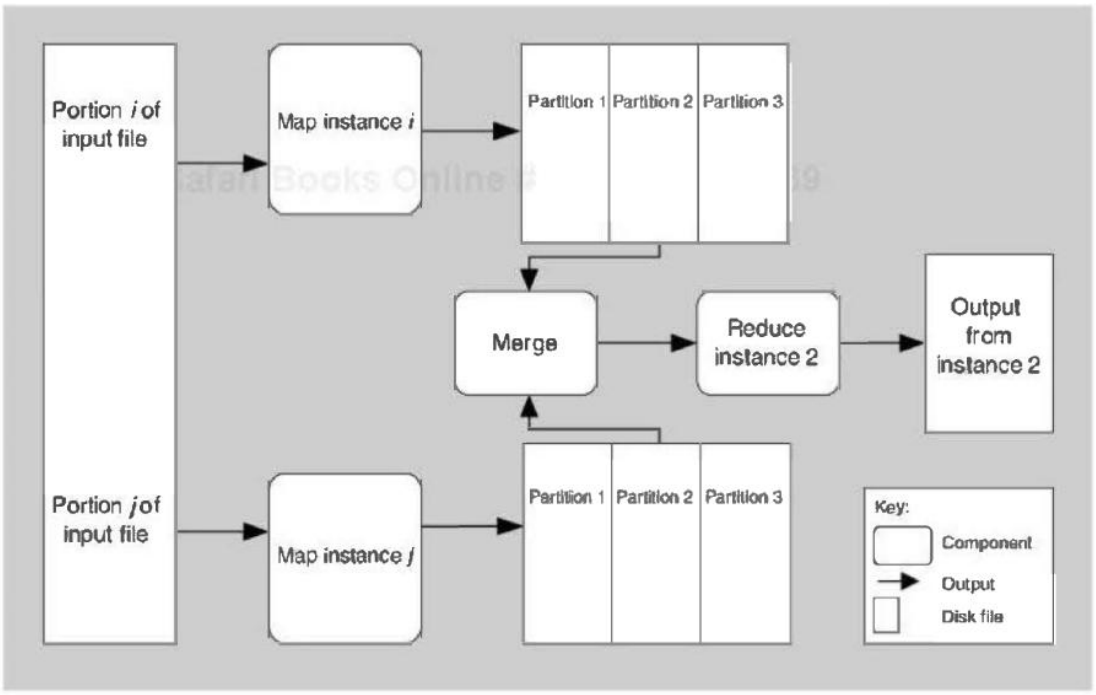


Figure 13.14. A component-and-connector view of map-reduce showing how the data processed by map is partitioned and subsequently processed by reduce

解决方案：

概览	映射-规约模式提供了一个并行分析大型数据集的框架。并行化可以做到低延迟和高可获得性。映射过程执行分析（analysis）中的提取（extract）和变换（transform）部分的操作，而规约过程执行结果的加载（loading of the results）。
元素	映射（map）是部署在多处理器（processor）上的函数，执行分析的提取和变换部分（a function with multiple instances deployed across multiple, processors that performs the extract and transformation portions of the analysis）。 规约（reduce）是部署在单或多处理器上的函数，执行加载部分（a function that may be deployed as a single instance or as multiple instances across processors to perform the load portion of extract-transform-load）。 基础设施（infrastructure）是负责部署映射和规约实例、照看它们的数据，以及检测错误并从中恢复的框架。
关系	一个映射或规约实例及其所安装在的处理器的部署关系（deploy on）
约束	α 要分析的数据必须作为一组文件存在。 β 映射函数是无状态的，不相互通信。 γ map-reduce 实例之间的唯一通信是从 map 实例中以键值对的形式发出的数据。
弱点	α 只适合分析大型数据集。 β 如果不能将数据集划分为类似大小的子集，则并行性的优点会丢失。 γ 需要多次 reduce 的操作的协调是非常复杂的。

参考：

[1] MapReduce https://baike.baidu.com/item/MapReduce

K. 多层模式（multi-tier pattern）

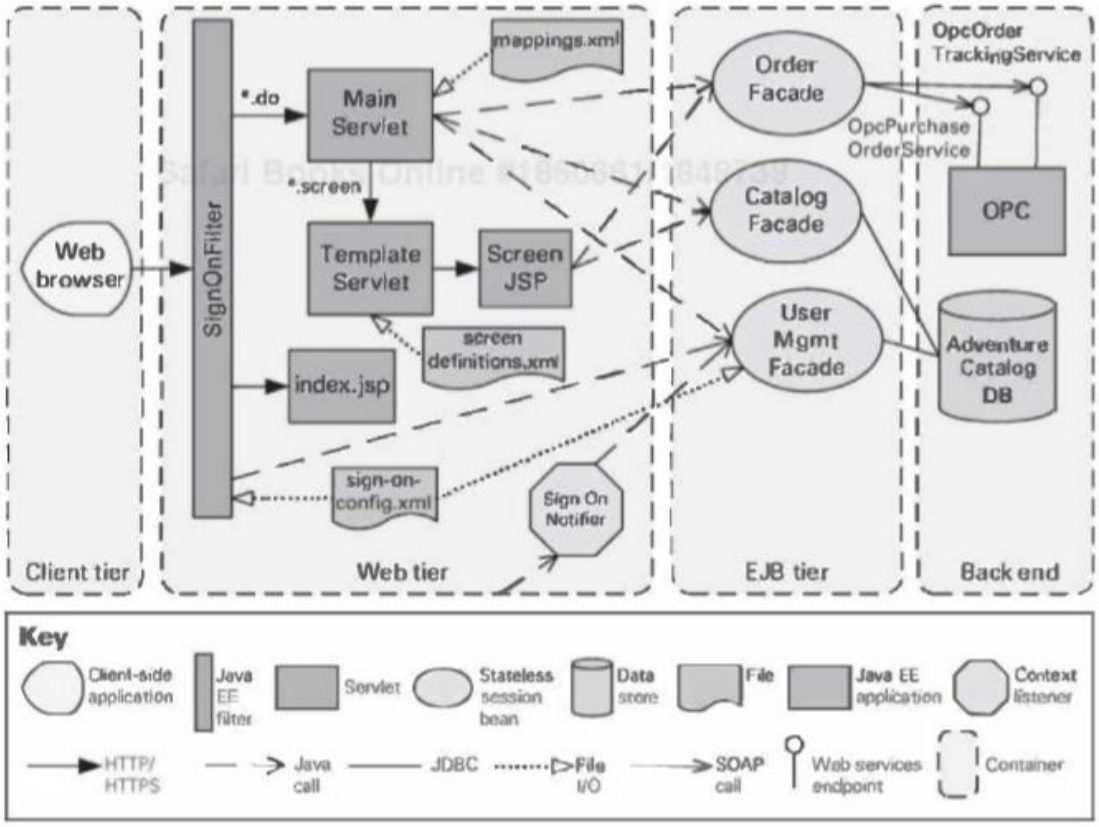


Figure 13.15. A multi-tier view of the Consumer Website Java EE application, which is part of the Adventure Builder system

解决方案：

概览	许多系统的执行结构组织成组件的逻辑分组。每一组被称为层（tier）。组件基于一系列标准，如组件类型、共享相同的执行环境或有相同的运行时目的，分组形成层。
元素	层（tier），软件组件的逻辑分组 层可以以常用的计算平台为基础组建。在这种情况下，这些平台也是该模式的元素。
关系	属于（is part of），对层与组件 与通信（communication with），表示层及其包含的组件的交互（interact） 被分配给（allocated to），层与计算平台
约束	单个软件组件属于一个层，不能一对多。
弱点	增加大量前期（up-front）成本和复杂性。

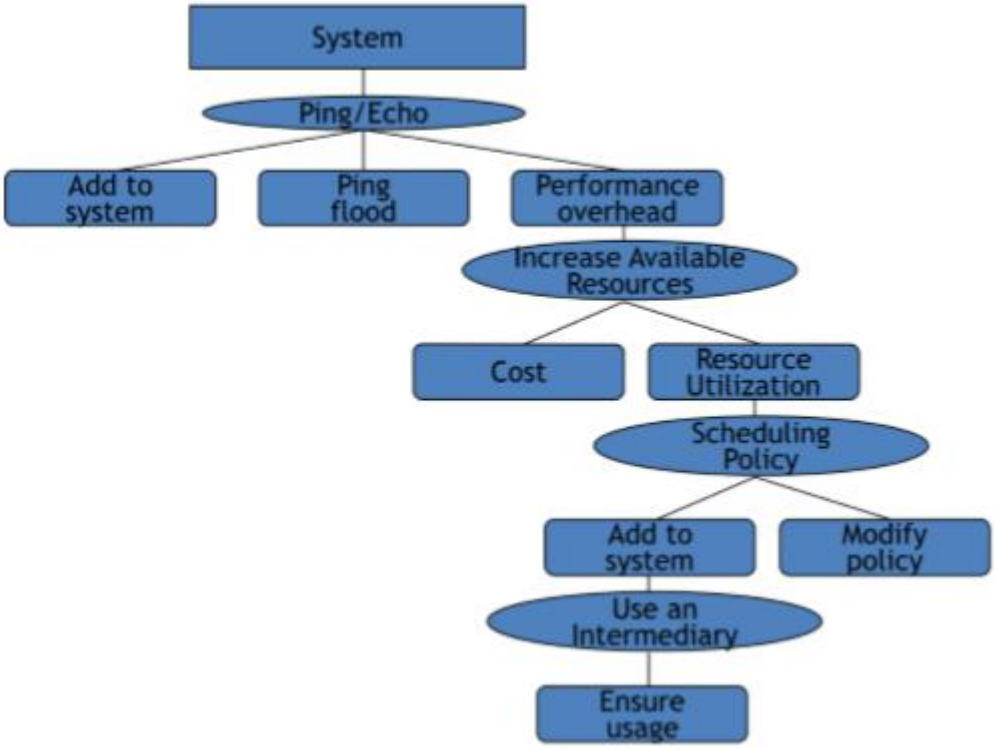
9. 模式与战术

A. 模式和战术之间的关系：模式由战术组成。如果一个模式是一个分子，那么战术就是一个原子。

B. 模式是对症下药，但是有可能对其他质量属性不利。这个时候就要补点战术，来解决一些模式的缺点。

C. 每个战术都有优点和副作用。

D. 战术调用树：



如何保证 usage？使用严格限制交流路径的战术。

它的副作用又是什么？如何确保中介的性能开销不过分？

这样又回到了最初的性能开销问题，开始递归吧！非常尴尬。

如何结束这个过程？ 每次使用战术都会带来新的问题。每解决一个新的问题都会导致新战术的加入。

我们是在一个无限的进程？最终，每种战术的副作用变得小到足以忽略，递归停止。

10. 敏捷项目中的架构

A. 敏捷宣言

四个核心价值：

- α 个体和互动高于流程和工具
- β 工作的软件高于详尽的文档
- γ 客户合作高于合同谈判
- δ 响应变化高于遵循计划

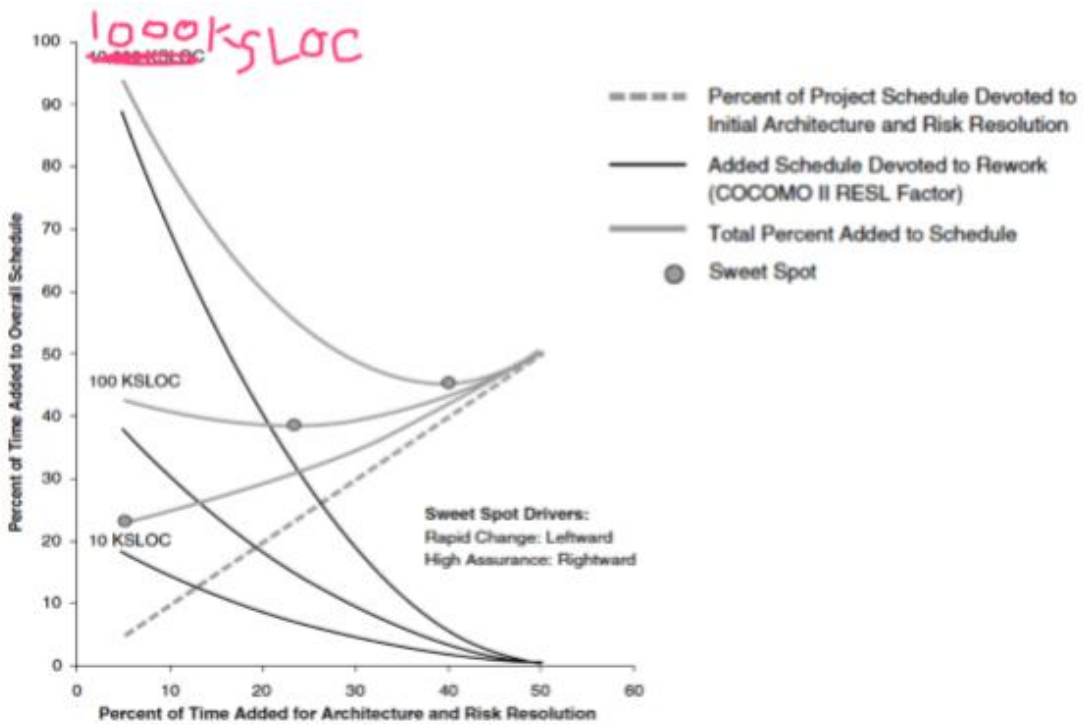
十二条原则：

- α 通过早期和连续型的高价值工作交付满足“客户”。
- β 大工作分成可以迅速完成的较小组成部门。
- γ 识别最好的工作是从自我组织的团队中出现的，
- δ 为积极员工提供他们需要的环境和支持，并相信他们可以完成工作。
- ε 创建可以改善可持续工作的流程。
- ζ 维持完整工作的不变的步调。
- η 欢迎改变的需求，即使是在项目后期。
- θ 在项目期间每天与项目团队和业务所有者开会。
- ι 在定期修正期，让团队反映如何能高效，然后进行相应地行为调整。
- κ 通过完成的工作量计量工作进度。

λ 不断地追求完善。

μ 利用调整获得竞争优势。

B. 甜点图



项目额外时间百分比与用于架构和解决风险的时间百分比的关系

参考：

[1] 敏捷宣言
https://baike.baidu.com/item/%E6%95%8F%E6%8D%B7%E5%AE%A3%E8%A8%80

11. 架构和需求

A. 架构关键需求（architecturally significant requirement, ASR）

α 是一种会对架构产生深远影响的需求。

β 有很高的业务价值。

B. 收集架构关键需求的方法

α 从需求文档收集（gathering ASRs from requirement documents）

β 通过与涉众的座谈会收集（gathering ASRs by interviewing stakeholders）：质量属性研讨会（QAW）

γ 通过理解商业目标来收集架构关键需求（gathering ASRs by understanding the business goals）

C. 质量属性研讨会（quality attribute workshop, QAW）：

在软件架构完成之前，QAW 是一种方便的、以干系人为中心的方法，用于生成、划分优先级以及完善质量属性场景。

相关步骤：

α QAW 展示和介绍：QAW 的主持人描述了 QAW 的动机，并解释了方法的每一步。

β 业务/任务介绍：代表系统背后的业务关注的利益干系人介绍了系统的业务环境、广泛的功能要求、约束和已知的质量属性要求。将在以后的步骤中细化的质量属性主要来自于这一步骤中提出的业务/任务需求。

γ 架构方案展示：架构师将展示系统架构计划。这让干系人了解当前的架构思想。

δ 架构驱动程序的标识：主持人将共享他们在步骤 2（β）和 3（γ）中组装的关键架构驱动程序的列表，并要求干系人进行澄清、添加、删除和更正。这样做的目的是在一个包含总体需求、业务驱动因素、约束和质量属性的架构驱动程序的列表上达成共识。

ε 关于场景的头脑风暴：每一个干系人都表示他或她对该系统的关注的情况。主持人确保每个场景都有明确的刺激和反应。协调员确保在步骤 4（δ）中列出的每个体系结构驱动程序至少存在一个代表方案。

ζ 场景整合：相似的场景得到整合。整合有助于防止重复投票。

η 场景划分优先级。

θ 场景细化（完善）：改善和阐述了顶级方案。主持人帮助干系人将场景放在六部分的场景形式中，即源-刺激-人工制品-环境-响应-响应度量。

D. 谱系属性启发方法（Pedigreed Attribute eLicitation Method, PALM）：用于生成效用树。

α PALM 概述展示；

β 商业驱动演示；

γ 架构驱动演示；

δ 商业目标的启发；

ε 从商业目标中识别潜在的质量属性；

ζ 对现有质量属性驱动程序的谱系进行分配；

η 总结。

E. 表格形式的效用树示例

Quality Attribute	Attribute Refinement	ASR
Performance	Transaction response time	A user updates a patient's account in response to a change-of-address notification while the system is under peak load, and the transaction completes in less than 0.75 second. (H,M)
		A user updates a patient's account in response to a change-of-address notification while the system is under double the peak load, and the transaction completes in less than 4 seconds. (L,M)
	Throughput	At peak load, the system is able to complete 150 normalized transactions per second. (M,M)
Usability	Proficiency training	A new hire with two or more years' experience in the business becomes proficient in Nightingale's core functions in less than 1 week. (M,L)
		A user in a particular context asks for help, and the system provides help for that context, within 3 seconds. (H,M)
	Normal operations	A hospital payment officer initiates a payment plan for a patient while interacting with that patient and completes the process without the system introducing delays. (M,M)
Configurability	User-defined changes	A hospital increases the fee for a particular service. The configuration team makes the change in 1 working day; no source code needs to change. (H,L)
Maintainability	Routine changes	A maintainer encounters search- and response-time deficiencies, fixes the bug, and distributes the bug fix with no more than 3 person-days of effort. (H,M)
		A reporting requirement requires a change to the report-generating metadata. Change is made in 4 person-hours of effort. (M,L)
	Upgrades to commercial components	The database vendor releases a new version that must be installed in less than 3 person-weeks. (H,M)
Extensibility	Adding new product	A product that tracks blood bank donors is created within 2 person-months. (M,M)
Security	Confidentiality	A physical therapist is allowed to see that part of a patient's record dealing with orthopedic treatment but not other parts nor any financial information. (H,M)
	Integrity	The system resists unauthorized intrusion and reports the intrusion attempt to authorities within 90 seconds. (H,M)
Availability	No downtime	The database vendor releases new software, which is hot-swapped into place, with no downtime. (H,L)
		The system supports 24/7 web-based account access by patients. (L,L)

（业务价值，架构冲击）

H=high, M=medium, L=low

12. 设计策略

A. 分解（decomposition）

B. 设计 ASR（designing to ASR）

C. 生成并测试（generate and test）

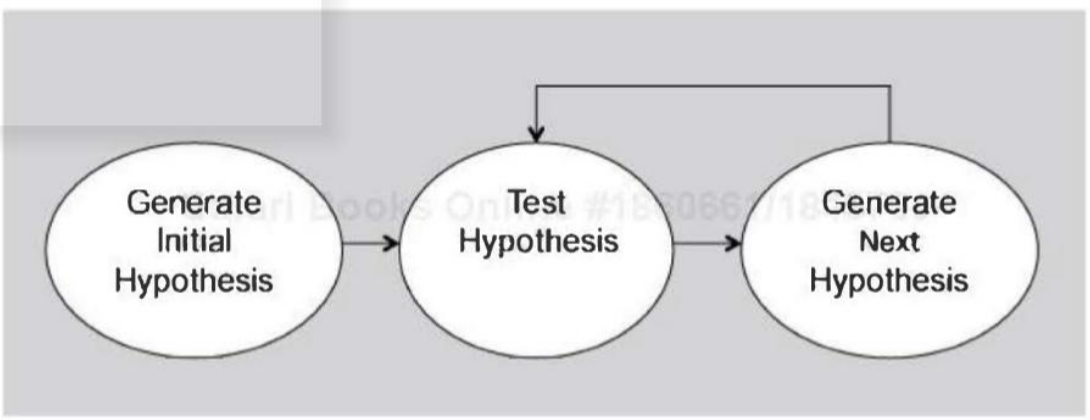


Figure 17.1. The generate-and-test process of architecture design

（Hypothesis n. 假说）

α 把当前设计看作假说。

β 测试，看是否满足需求。

γ 如果不满足，生成新假说（设计）。

13. 属性驱动设计（attribute-driven design, ADD）

A. ADD 是一个迭代的过程，每次迭代：

α 选择系统之一部分去设计；

β 整理那部分系统之所有之 ASRs；

γ 产生并测试那个部分之设计。

ADD 不产生完整的设计，不为容器（即输出）产生 API 或签名。

B. 输入

需求（requirement）：功能上（functional）、质量（quality）和约束（constraint）

一份背景描述（a context description）：系统设计的范围（what are the boundary of the system being designed）和所设计的系统所与之交互的外部系统、设备、用户和环境条件（what are the external systems, devices, users and environment conditions with which the system being designed must interact）

C. 输出

架构元素及其之间的关系（architectural elements ad their relationship）：元素职责（responsibility of elements）、互动（interactions）和元素间的信息流（information flow among the elements）

D. ADD 的步骤（一次迭代的步骤，跟 A 部分一样，但更加严谨和正式）

α 选择系统的一个元素（element）来设计。

β 识别所选元素的 ASR。

γ 为所选元素生成设计方案（design solution）。

δ 清点残留（未满足的）需求并选择下次迭代的输入（inventory remaining requirements and select the input for the next iteration）。

ε 重复步骤 1-4（α到δ）直到所有 ASR 都被满足。

14. 视图

A. 架构三视图：模块视图（module view）、组件-连接件视图（component-and-connector view, C&C view）和分配视图（allocation view）。

B. 此外还有质量视图（quality view）。

15. 架构评估

A. 三种形式：

α 设计过程中由设计者进行评估。

β 在设计过程中由对等方进行评估。

γ 在设计了架构后，由外部分析。

B. 架构权衡分析方法（architecture tradeoff analysis method, ATAM）：

α 评估团队：а 是这个项目外的人； б 3 到 5 人，一人分饰多角； в 公认公正的局外人。

ATAM 评估团队角色：

角色	职责（这栏随便看看）
团队领导人（team leader）	建立评估，与客户协调，确保满足客户的需求；建立评估合同；形成评估团队；看到（see）最终报告的制作和交付（尽管可以委托编写）
评估领导人（evaluation leader）	运行评估；促进场景的启发；管理场景选择/优先级处理；促进对架构的场景评估；促进现场分析
场景抄写员（scenario scribe）	在白板上记录场景；捕捉参会者商定每个场景时的措辞，直到停止讨论，然后确切场景被抓获
会议记录抄写员（proceedings scribe）	在手提电脑或工作站上以电子形式记录会议：原始场景、每个场景的诱导问题（往往在关于场景的讨论中丢失）以及每个场景应用到架构时的解决方案；也要生成所采用场景的清单并交给与会者
提问者（questioner）	提出建筑兴趣的问题，通常与其所具有的专门知识的质量属性有关

β ATAM 的输出：

а 一份架构的简明介绍；

б 业务目标的表述；

в 划分了优先级的，以质量属性场景来表示的质量属性需求；

г 一组风险与非风险（点）；

д 一组风险主题；

е 将架构决策映射到质量需求；

ё 一组确定的敏感点和权衡点。

γ ATAM 的阶段：

阶段	活动	参与者	一般持续时间
0	合作与准备：后勤、规划、招聘干系人、组建团队	评估团队领导人和关键项目决策者	按要求非正式地进行，也许几周
1	评估：步骤 1 到 6	评估团队和项目决策者	1-2 天以及随后间断的 2-3 周
2	评估：步骤 7 到 9	评估团队、项目决策者和干系人	2 天
3	跟进：报告的生成和交付，过程改进	评估团队和评估的客户	1 周

步骤 1：展示 ATAM

步骤 2：展示业务驱动

展示以下内容：

а 系统最重要的功能；

б 任何与科技、管理、经济、政治有关的约束；

в 与项目相关的业务目标和背景；

г 主要的干系人；

д ASRs。

步骤 3：展示架构

步骤 4：标识架构方法（identify architectural approaches）

步骤 5：产生效用树

关键词：quality attribute goals

步骤 6：分析架构方法（Analyze Architectural Approaches）

а 风险：心跳频率会影响系统检测故障组件的时间。某些分配将导致此响应产生不可接受的值。

б 敏感点：同步数据库客户端的数量会影响数据库每秒能处理的事务数量；

в 权衡点：心跳频率决定了检测到故障的时间。更高的频率会导致更好的可获得性，但会消耗更多的处理时间和通信带宽（可能会降低性能）。

Scenario #: A12	Scenario: Detect and recover from HW failure of main switch.			
Attribute(s)	Availability			
Environment	Normal operations			
Stimulus	One of the CPUs fails			
Response	0.999999 availability of switch			
Architectural decisions	Sensitivity	Tradeoff	Risk	Nonrisk
Backup CPU(s)	S2		R8	
No backup data channel	S3	T3	R9	
Watchdog	S4			N12
Heartbeat	S5			N13
Failover routing	S6			N14

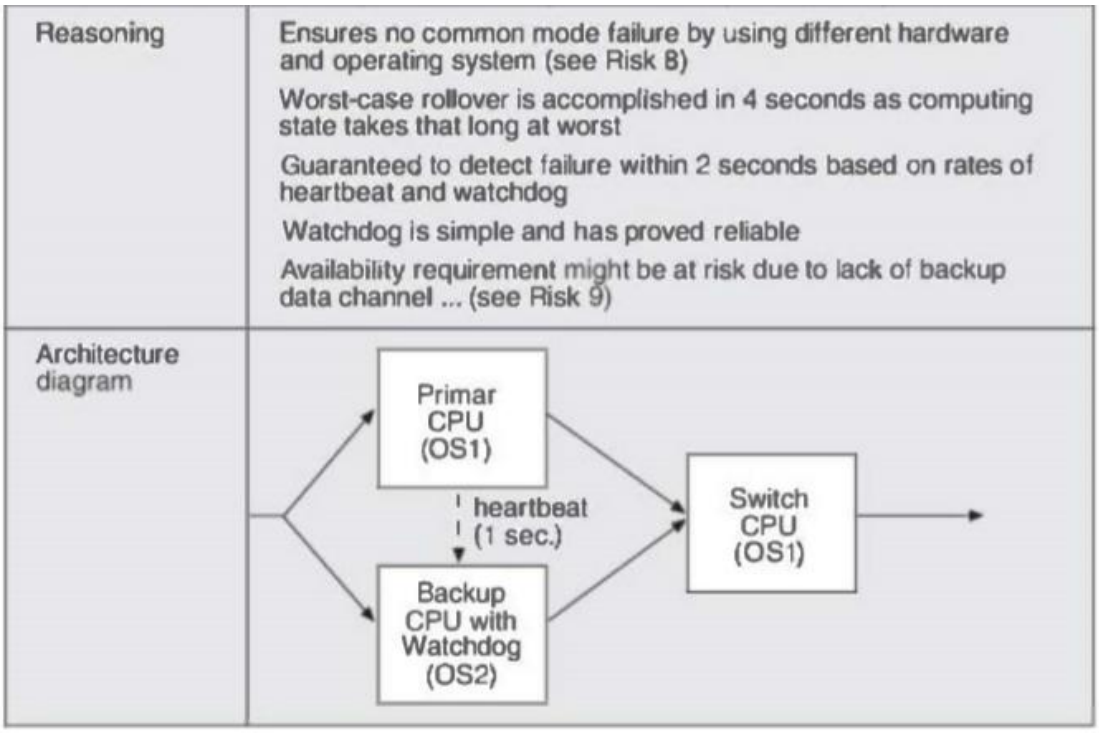


Figure 21.1. Example of architecture approach analysis (adapted from [Clements 01b])

步骤 7：头脑风暴和给场景划分优先级（Brainstorm and Prioritize Scenarios）

步骤 8：分析架构方法

与步骤 6 类似，但是使用最高级的、最新生成的场景。

步骤 9：展示结果

C. 轻量级架构评估（lightweight architecture evaluation）

一个 ATAM 过程通常需要 20-30 个人日，只适用于大型昂贵的项目。所以发展出了轻量级架构评估，一般只需要一天甚至半天，只涉及组织内部的人员。这个评估方法不产生最终报告，而是由抄写员负责收集结果。

只有组织内部成员来评估可能得出不客观的结果，缺乏创新和讨论。但是这个评估方法快速廉价，所以可以被快速部署，无论项目是否需要关于架构质量保证的合理性检查（sanity check）。

Table 21.4. A Typical Agenda for Lightweight Architecture Evaluation

Step	Time Allotted	Notes
1: Present the ATAM	0 hrs	The participants are familiar with the process. This step may be omitted.
2: Present Business Drivers	0.25 hrs	The participants are expected to understand the system and its business goals and their priorities. Fifteen minutes is allocated for a brief review to ensure that these are fresh in everyone's mind and that there are no surprises.
3: Present Architecture	0.5 hrs	Again, all participants are expected to be familiar with the system and so a brief overview of the architecture, using at least module and C&C views, is presented and 1 to 2 scenarios are traced through these views.
4: Identify Architectural Approaches	0.25 hrs	The architecture approaches for specific quality attribute concerns are identified by the architect. This may be done as a portion of step 3.
5: Generate Quality Attribute Utility Tree	Variable 0.5 hrs – 1.5 hrs	Scenarios might exist: part of previous evals, part of design, part of requirements elicitation. If you've got 'em, use 'em and make them into a tree. Half hour. Otherwise, it will take longer. A utility tree should already exist; the team reviews the existing tree and updates it, if needed, with new scenarios, new response goals, or new scenario priorities and risk assessments.
6: Analyze Architectural Approaches	2–3 hrs	This step—mapping the highly ranked scenarios onto the architecture—consumes the bulk of the time and can be expanded or contracted as needed.
7: Brainstorm and Prioritize Scenarios	0 hrs	This step can be omitted as the assembled (internal) stakeholders are expected to contribute scenarios expressing their concerns in step 5.
8: Analyze Architectural Approaches	0 hrs	This step is also omitted, since all analysis is done in step 6.
9: Present Results	0.5 hrs	At the end of an evaluation, the team reviews the existing and newly discovered risks, nonrisks, sensitivities, and tradeoffs and discusses whether any new risk themes have arisen.
TOTAL	**4–6 hrs**	

16. 云
    A. 特点
        - α 按需自助；
        - β 无处不在的网络访问；
        - γ 资源池；
        - δ 位置独立；
        - ε 资源消费者不需要关心资源的位置；

ζ 快速弹性；

η 测量服务：可以监视，控制和报告资源使用情况，以便服务的使用者仅针对他们使用的内容进行计费；

θ 多租户：应用程序和资源可以在不知情的多个消费者之间共享。

B. 基本服务模型： 软件即服务（SaaS）、平台即服务（PaaS）、基础架构即服务（IaaS）

C. 部署模型：私有云、公共云、社区云、混合云

D. 基本机制：虚拟机监控机制、虚拟机、文件系统、网络（Network）

E. HDFS (Hadoop Distributed File System)

F. 云为具有不同特征的应用程序提供了一个新的平台。架构师需要知道云如何运作，并要特别注意： 安全、性能、可获得性。

G. CAP 定理：在一个分布式系统中，一致性（consistency）、可获得性（availability）和分区容错性（partition tolerance）三者不可兼得。

α 一致性（C）：在分布式系统中的所有数据备份，在同一时刻是否同样的值。（等同于所有节点访问同一份最新的数据副本）

β 可获得性（A）：在集群中一部分节点故障后，集群整体是否还能响应客户端的读写请求。（对数据更新具备高可获得性）

γ 分区容错性（P）：以实际效果而言，分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性，就意味着发生了分区的情况，必须就当前操作在 C 和 A 之间做出选择。

参考：

[1] CAP 原则 https://baike.baidu.com/item/CAP%E5%8E%9F%E5%88%99

致谢

除了以上列出的参考资料外，本笔记还参考了周长鑫师兄的复习资料、林连南老师的复习课内容、石望华提纲以及 Len Bass、Paul Clements 和 Rick Kazman 的《软件构架实践》（课本）。