第二部分：如何运用UML建模

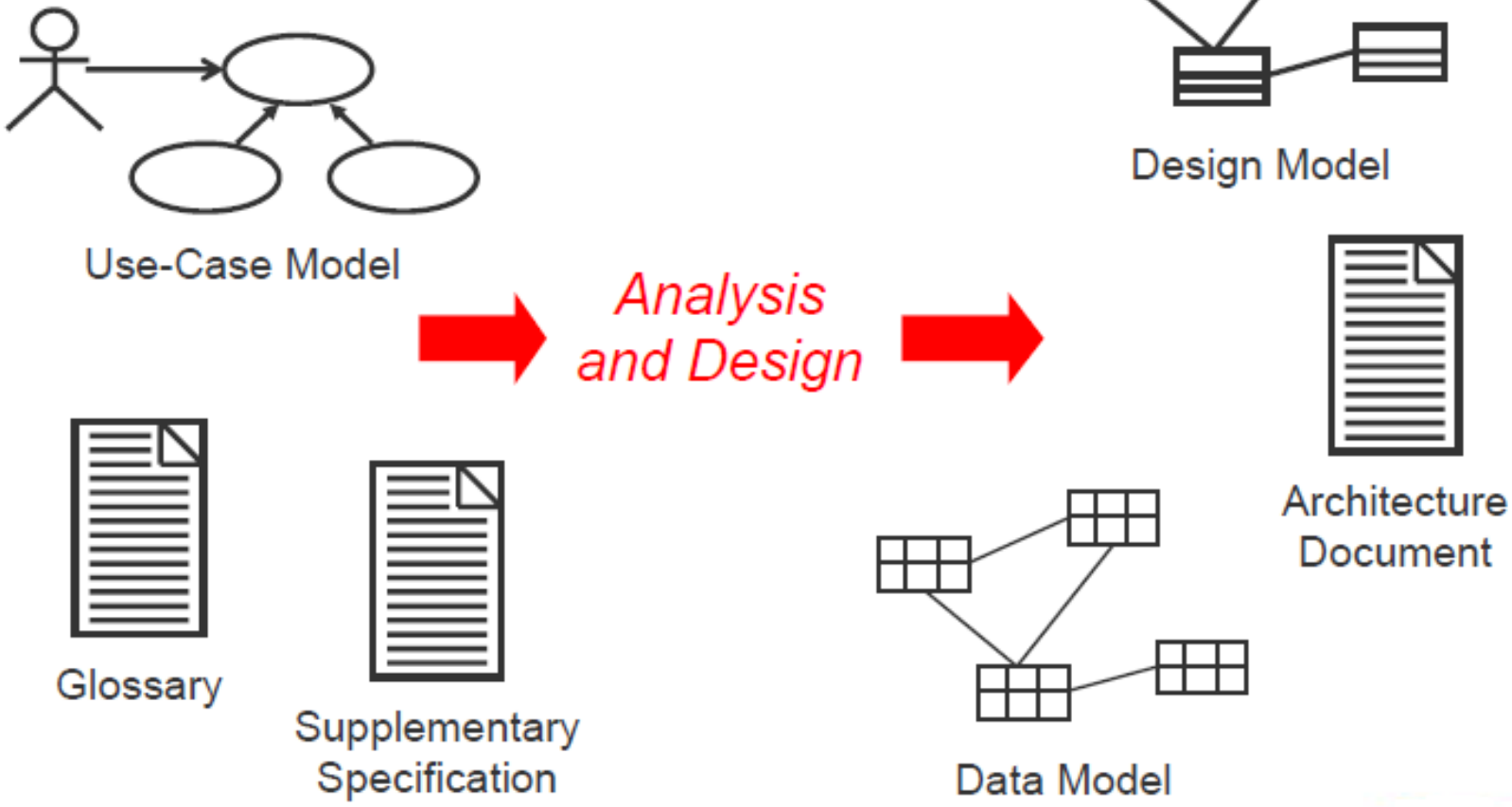
第八章 分析

提纲

- 从OOA到OOD
- 分析活动
  - 构架分析
  - 分析包
  - 用例分析
- 小结

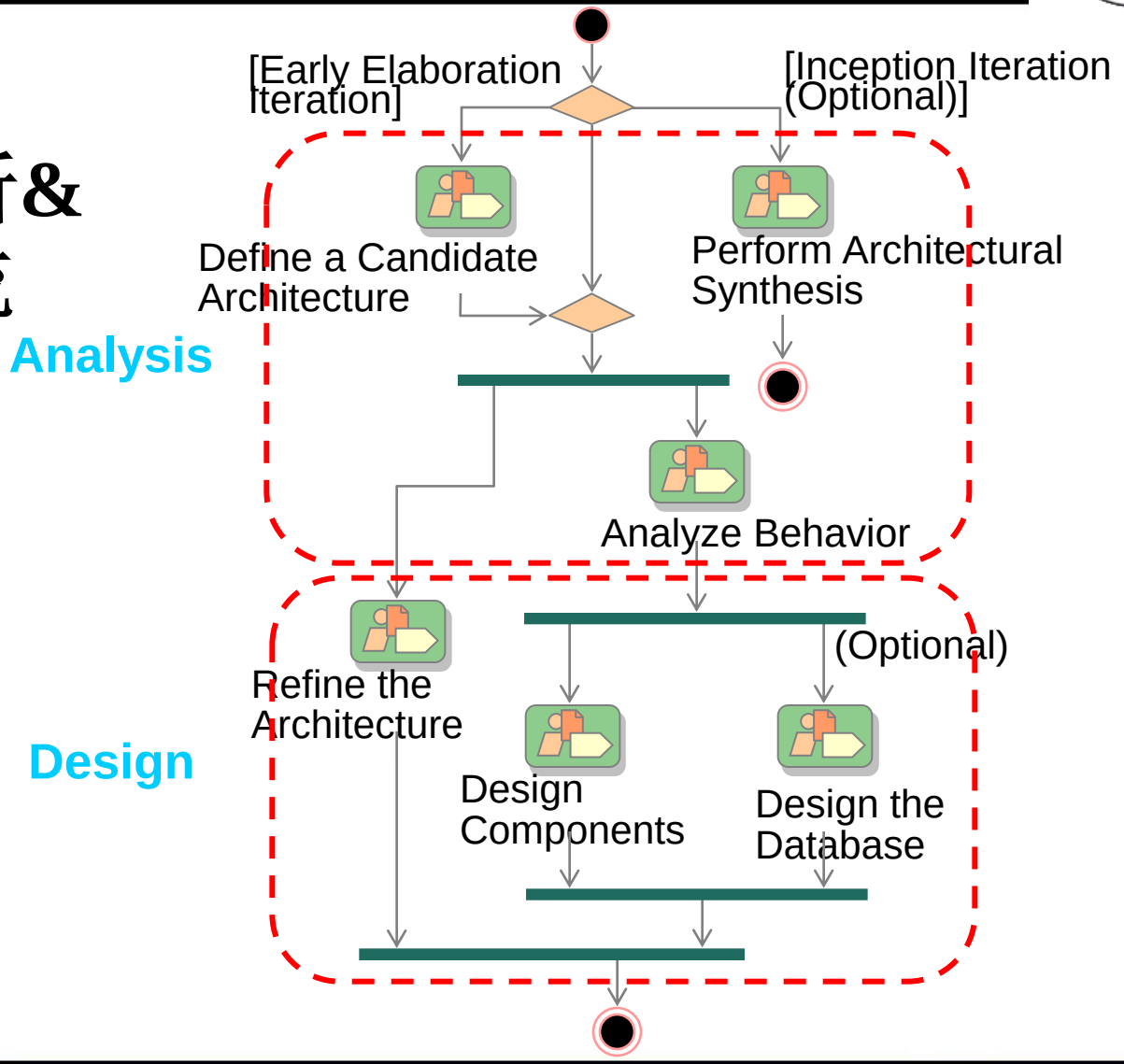
从OOA到OOD

- 从OOA到OOD



从OOA到OOD

- **RUP的分析&设计工作流**

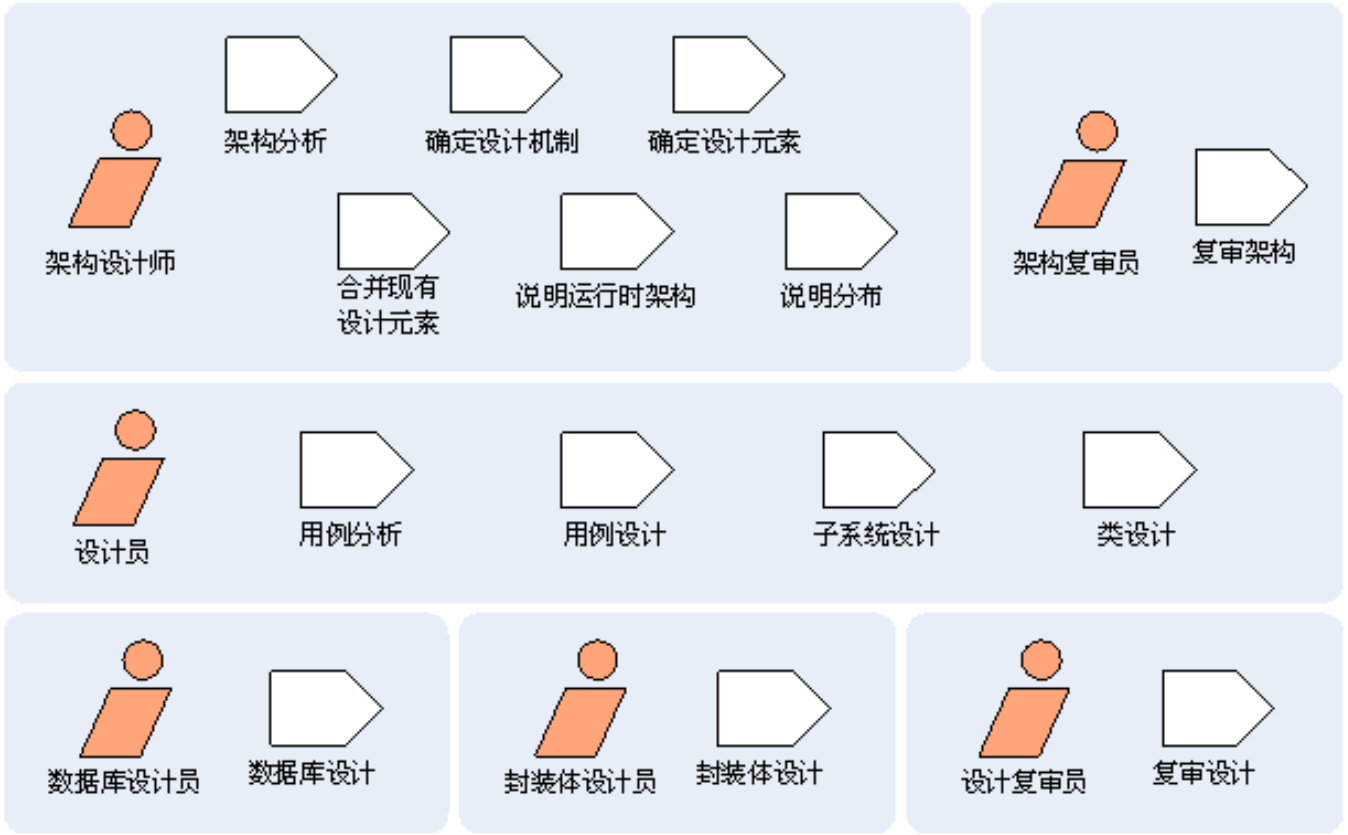


从OOA到OOD

- 分析&设计工作流
  - 目的
    - 将业务需求转换为未来系统的设计
    - 逐步开发强壮的系统构架
    - 使设计适合于实施环境，为提高性能而进行设计
  - 与其他工作流的关系
    - 业务建模流程为系统提供组织环境
    - 需求流程为其提供主要的输入
    - 其输出是实施流程的输入

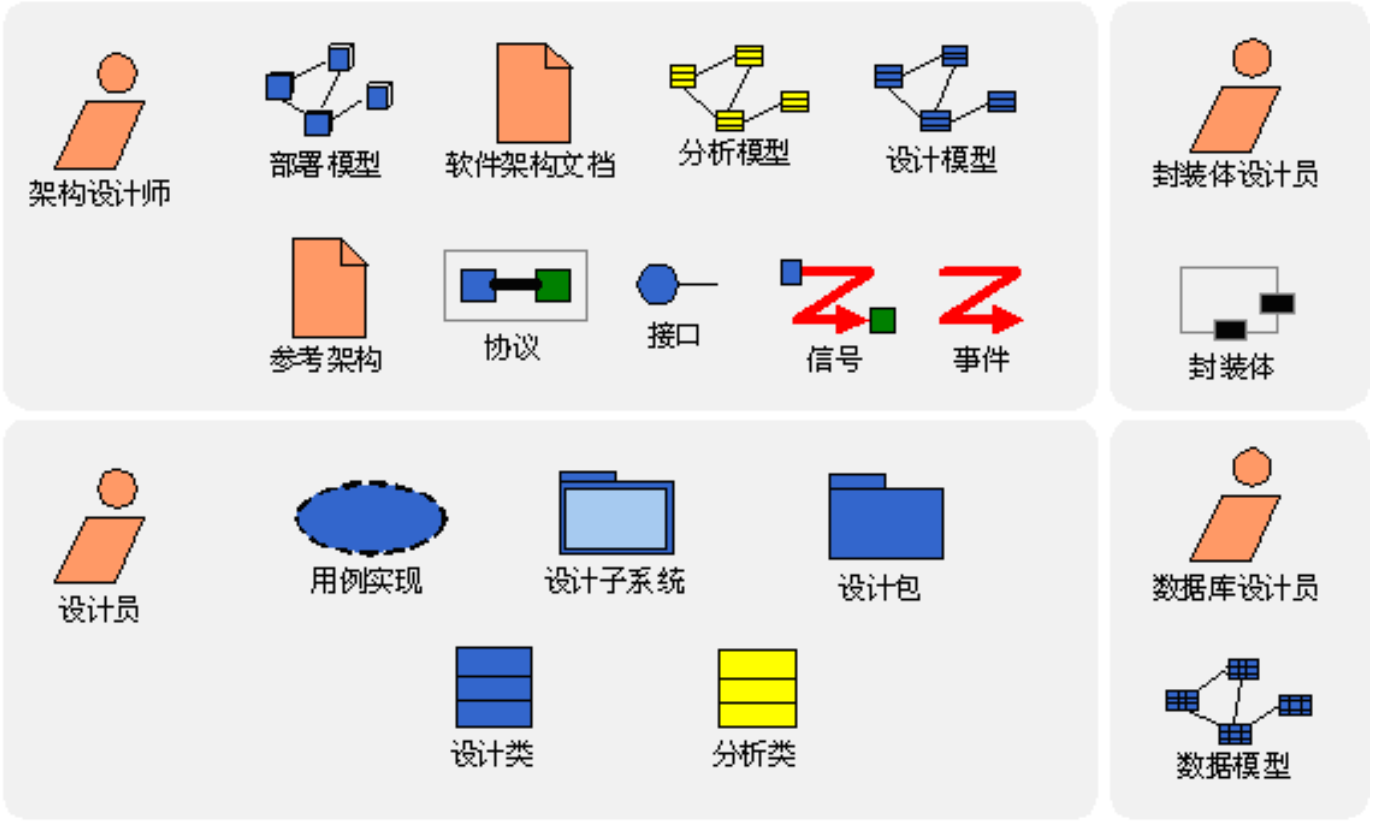
从OOA到OOD

- 分析&设计工作流
  - 角色与活动



从OOA到OOD

- 分析&设计工作流
  - 工件



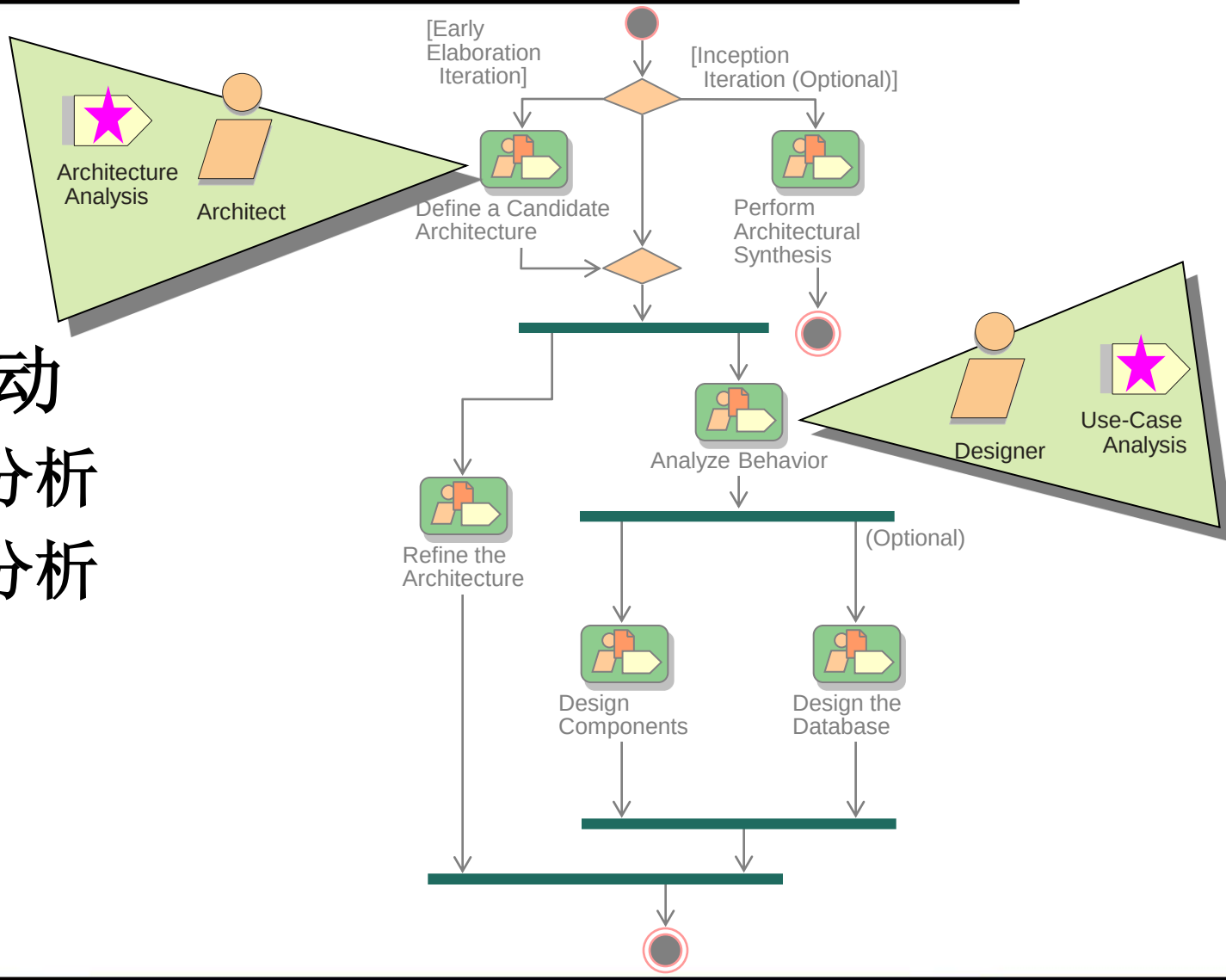
从OOA到OOD

- 分析VS设计

Analysis	Design
注重理解问题	注重解决方案
理想化设计	操作和属性
行为	性能
系统结构	接近实际代码
功能性需求	对象生命周期
小模型	非功能性需求
	大模型

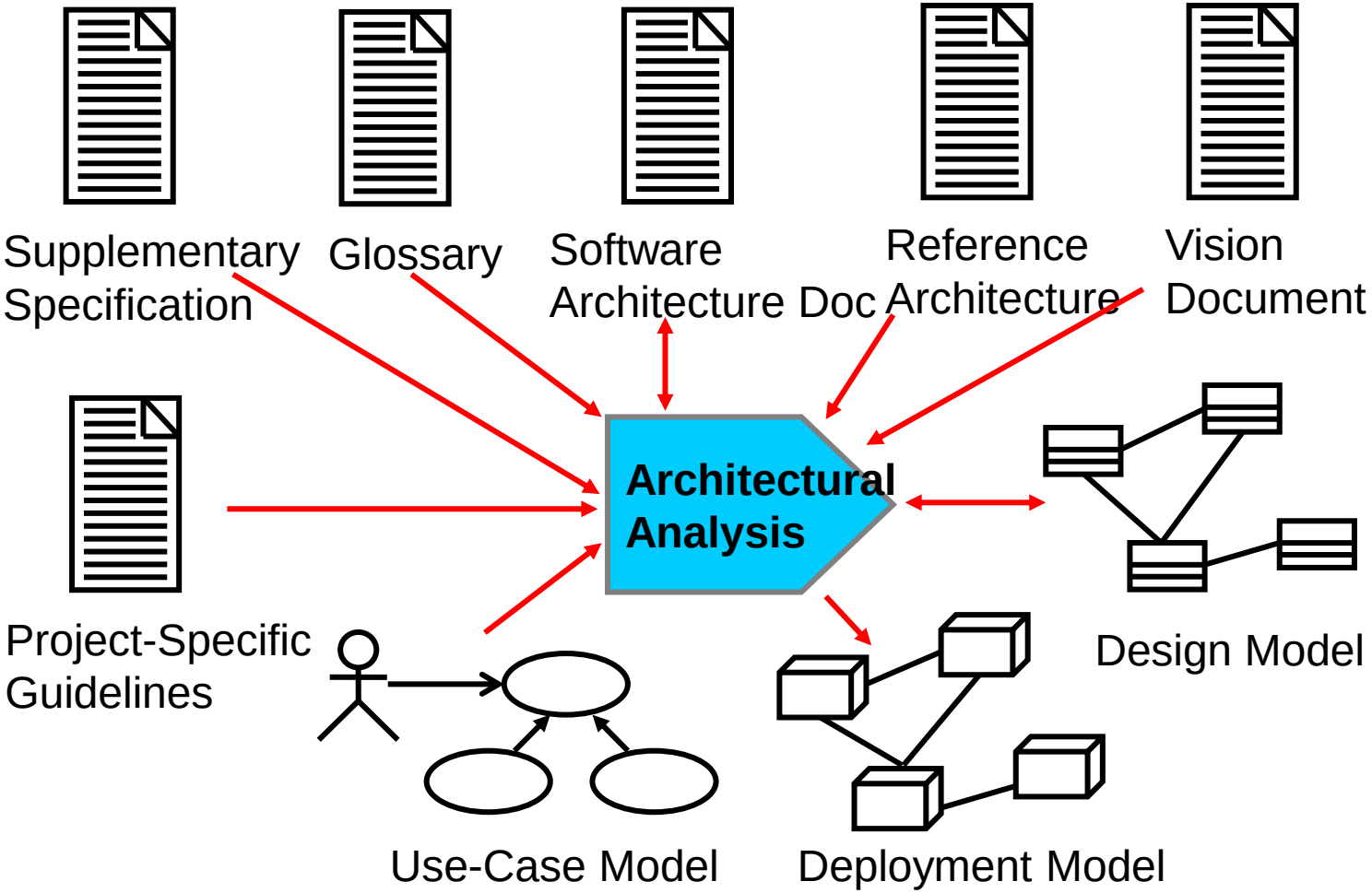
分析活动

- 分析活动
  - 构架分析
  - 用例分析



构架分析

- 构架分析

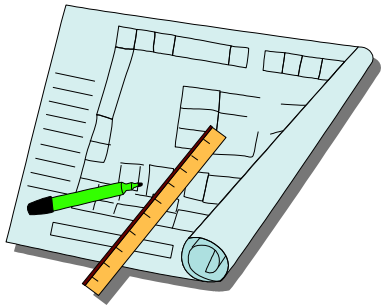

Supplementary Specification
Glossary
Software Architecture Doc
Reference Architecture
Vision Document
Architectural Analysis
Project-Specific Guidelines
Design Model
Use-Case Model
Deployment Model

构架分析

- 构架分析的步骤
  - 定义模型的高层组织结构
    - 分析/设计模式
    - 框架
  - 标识分析机制
  - 标识关键抽象
  - 创建用例实现

构架分析

- 体系结构模式
  - 描述一个软件系统的基本组织结构概要
  - 常见模式
    - **Layers**
    - **Model-view-controller (M-V-C)**
    - **Pipes and filters**
    - **Blackboard**



架构风格示例：分层模型

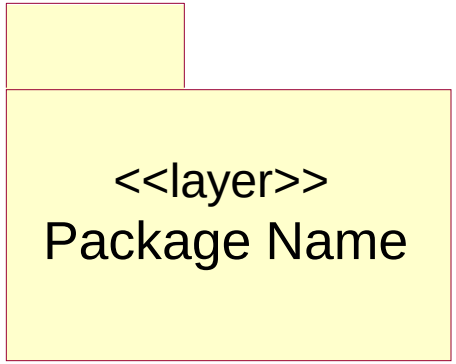
Application	Layer 7	Provides miscellaneous protocols for common activities
Presentation	Layer 6	Structure information and attaches semantics
Session	Layer 5	Provides dialog control and synchronization facilities
Transport	Layer 4	Breaks messages into packets and guarantees delivery
Network	Layer 3	Selects a route from send to receiver
Data Link	Layer 2	Detects and corrects errors in bit sequences
Physical	Layer 1	Transmits bits: velocity, bit-code, connection, etc.

分层的策略

- 抽象的层次
  - 将相同抽象层次的元素组织在一起
- 分离关注点
  - 将相似的事物组织在一起
  - 将不同的事物进行分离
  - **Application vs. domain model elements**
- 灵活性
  - 松耦合
  - 致力于封装变化
  - 对于用户界面、业务规则和所持有的数据，保持具有较高的变化潜力的支持。

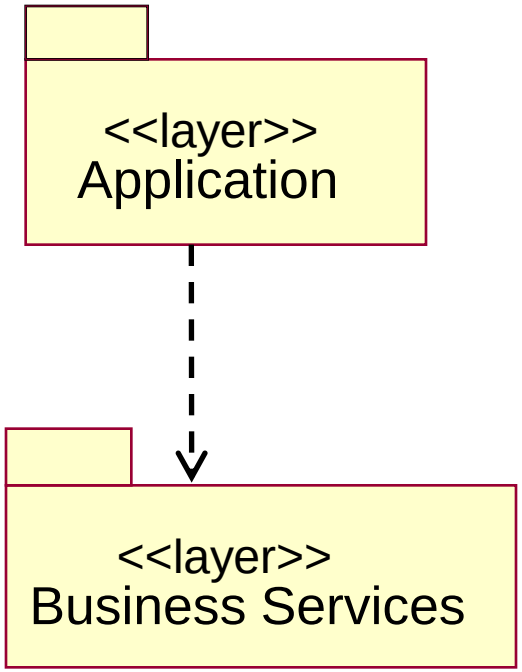
建模体系结构的层次

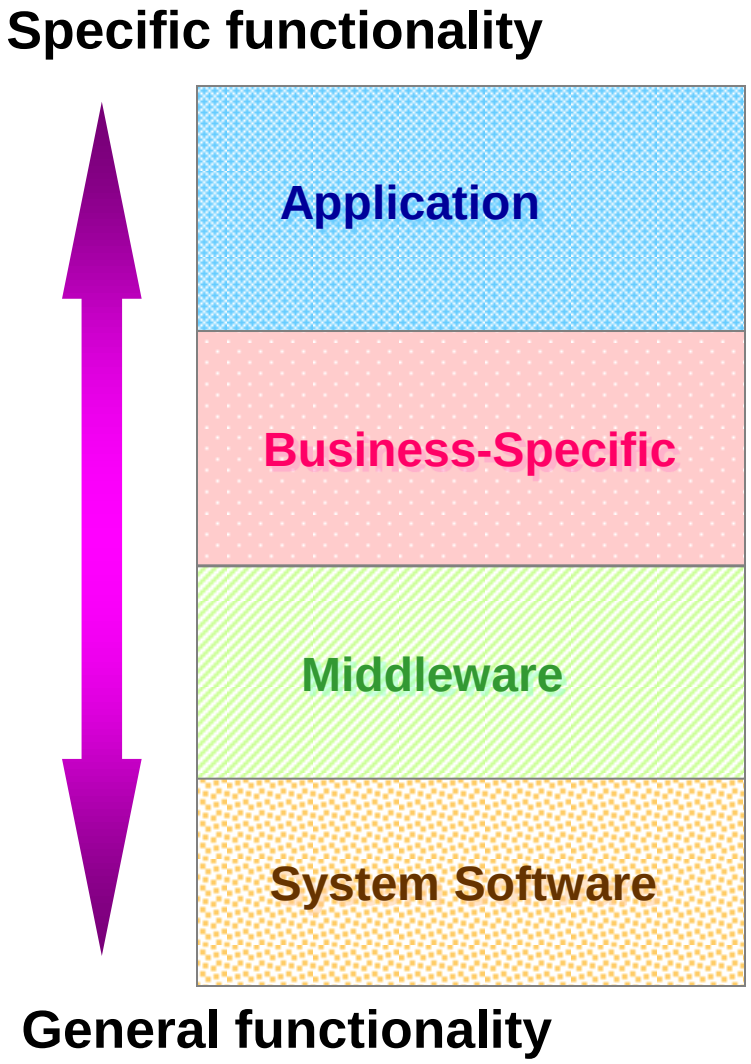
- 体系结构(架构)的层次，可以用包的构造型来建模
- **〈〈layer〉〉 stereotype**



构架分析

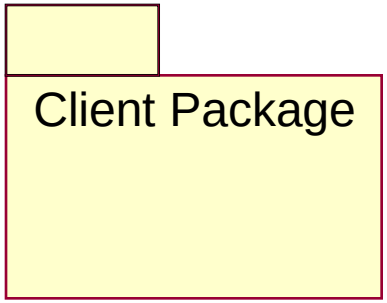
- 体系结构模式举例
  - 典型的分层模式





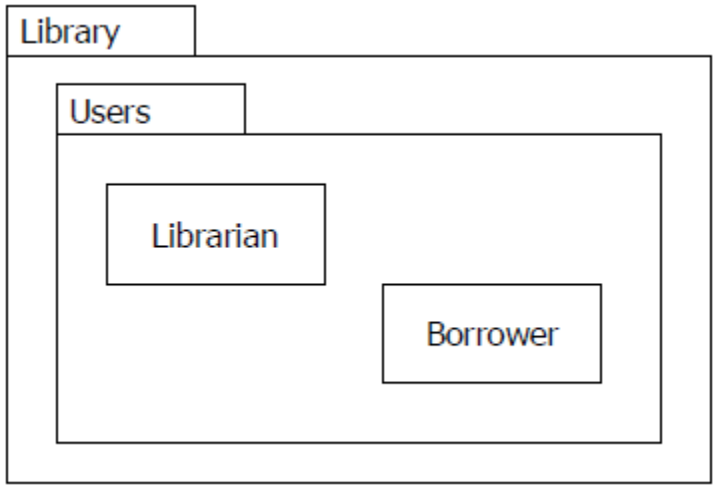
分析包

- 什么是包 (Package)
  - UML的分组事物
  - 将元素和图组织到组中的通用目的机制
    - 用于组织开发中的模型
    - 为并行工作和配置管理提供单元
  - 逻辑分组机制



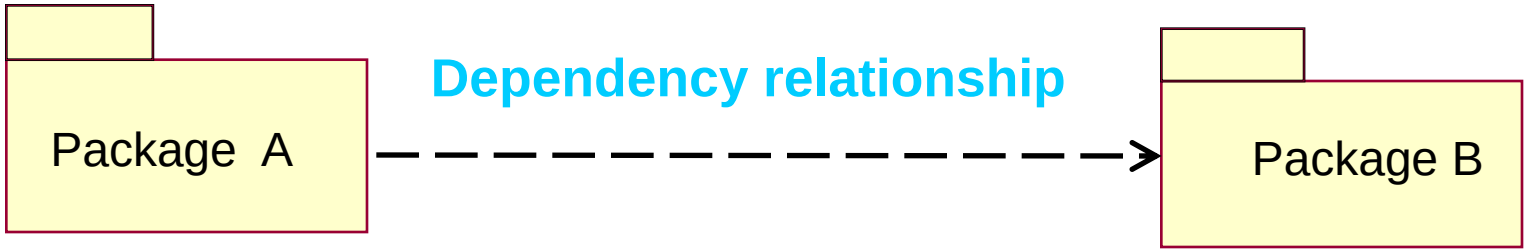
分析包

- 包的命名空间
  - 命名空间
    - 创建了一个边界
    - 边界内的元素名称唯一
    - 可以嵌套
  - 受限名称
    - 被命名空间外的元素引用时的名称
    - 元素的路径名称　Library :: Users :: Librarian



分析包

- 包依赖
  - 包可以通过依赖关系联系到别的包



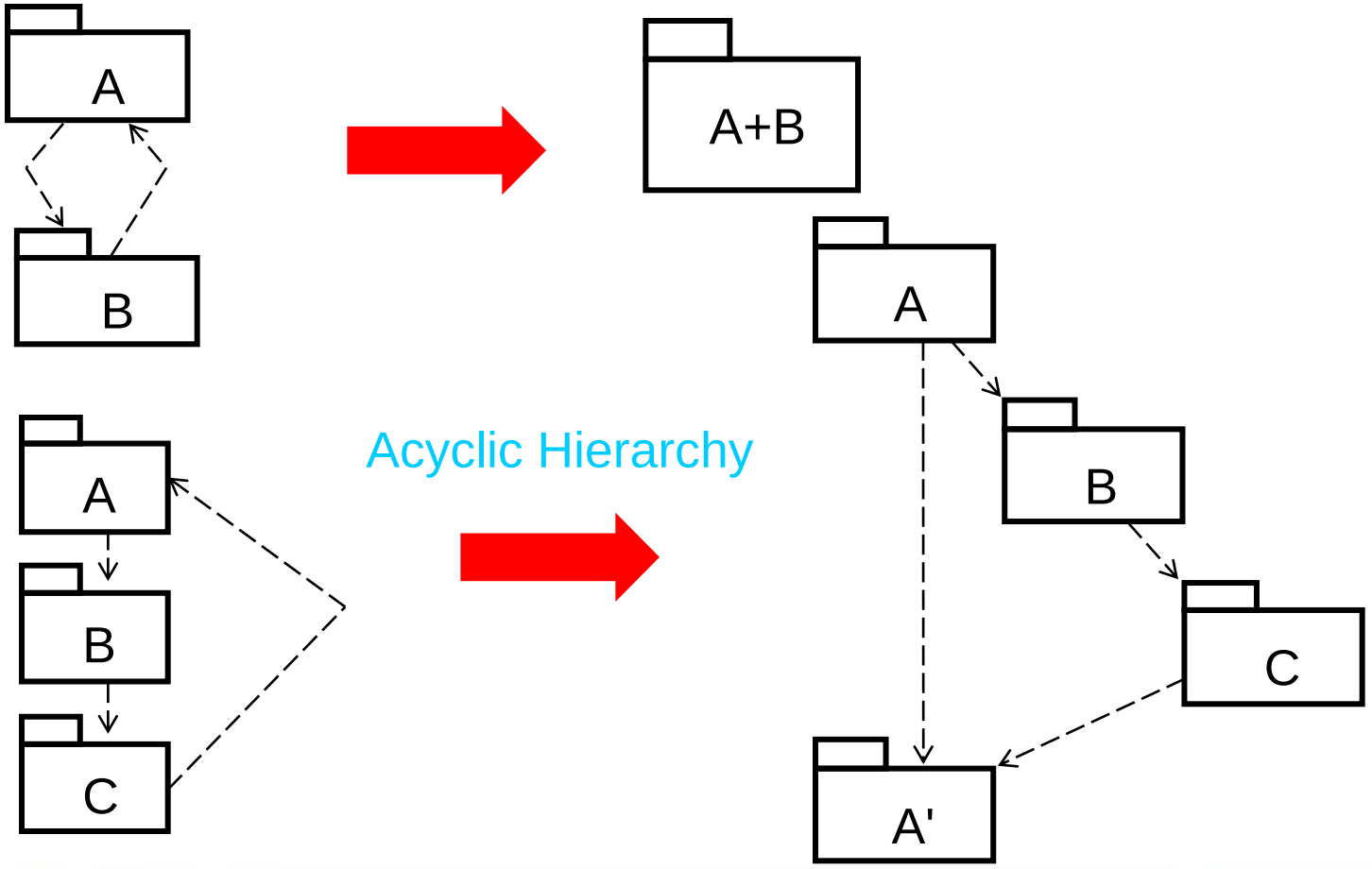
  - 包依赖表示
    - 对包B的改变会影响到包B
    - 不能独立地重用包A，因为它依赖于包B

分析包

- 找出分析包
  - 识别具有很强语义联系的建模元素的分组
  - 分析包可以包含用例、分析类、用例实现
  - 良好包结构的关键
    - 包内高内聚
    - 包间低耦合
  - 原则
    - 避免深层嵌套
    - 避免循环依赖

分析包

- 避免包间的循环依赖



体系结构定义

Mary Shaw, CMU
Grady Booch,
Philippe Kruchten,
Rich Reitman
Kurt Bittner, Rational

- 软件的体系结构包含关于软件系统组织的有意义的定义配置
  - 组成系统的结构元素和接口；
  - 元素中协作的特定行为；
  - 大的子系统中这些结构和行为的组合；
  - 体系结构风格支配了组织；

体系结构定义 (continued)

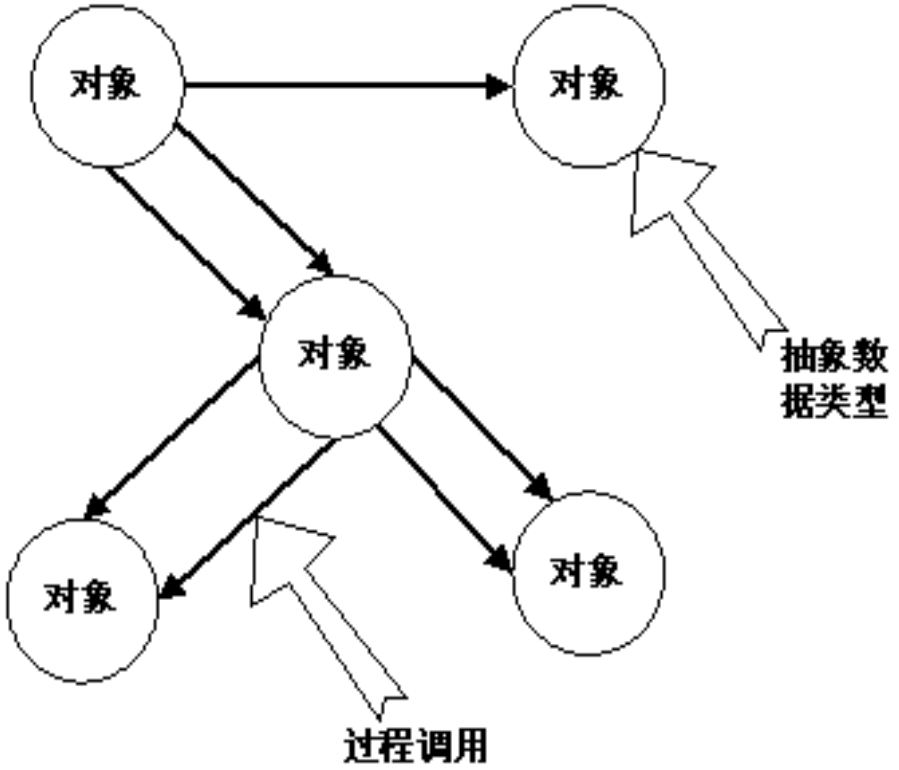
Mary Shaw, CMU
Grady Booch,
Philippe Kruchten,
Rich Reitman
Kurt Bittner, Rational

- 软件体系结构也涉及：
  - 用法
  - 功能
  - 性能
  - 伸缩能力
  - 重用
  - 内容广泛性
  - 经济上和技术上的冲突和平衡
  - 美学的观点

常见的体系结构风格

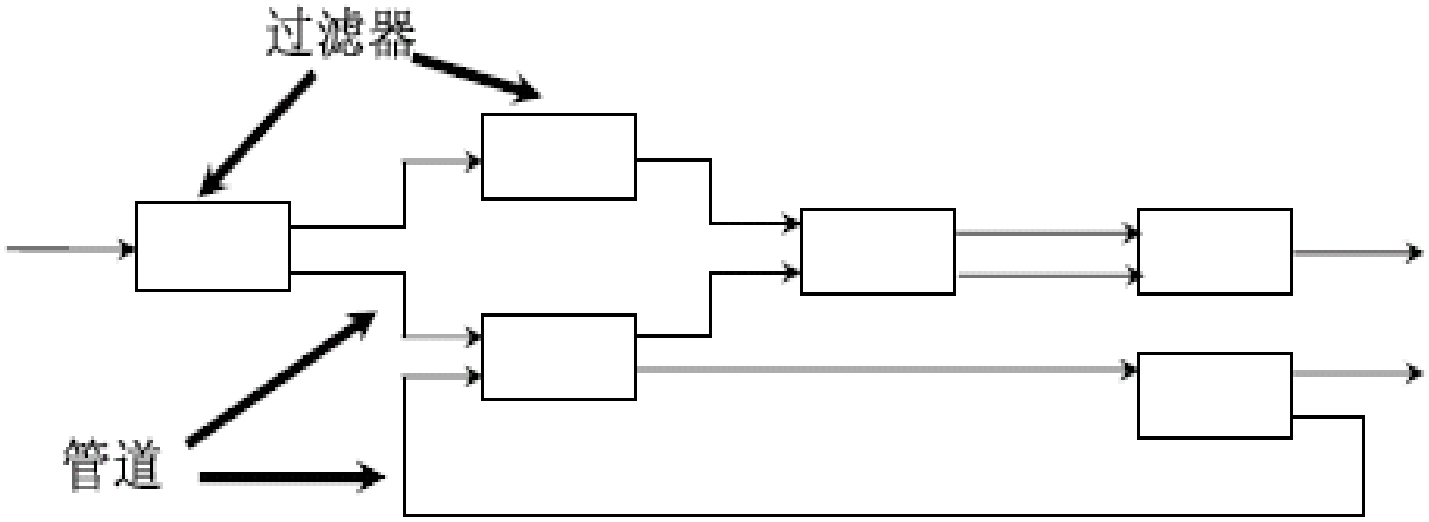
- 数据抽象和面向对象风格(ADT)
- 管道——过滤器(Pipe-and-Filter)
- 基于事件的隐式调用(Event-based, Implicit Invocation)
- 层次 (Layer)
- 代理(Broker)
- 黑板(Blackboard)
- MVC(Model-View-Controller)
- ……

数据抽象和面向对象风格(ADT)



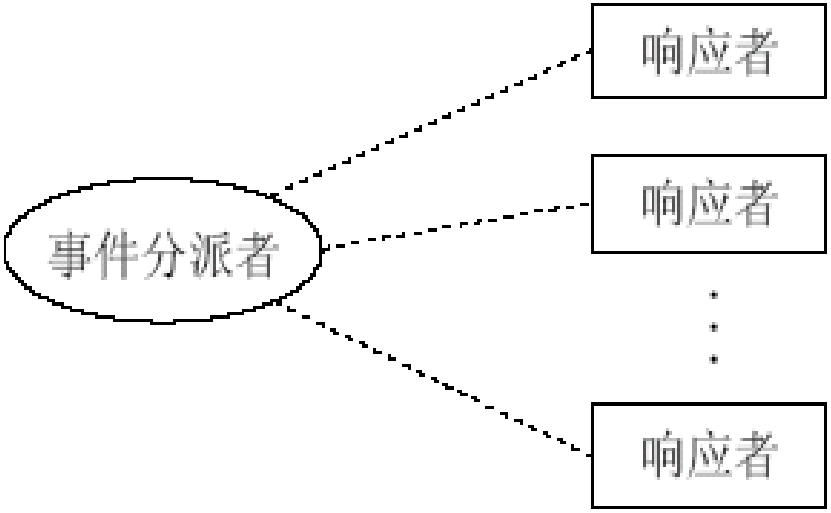
- 优点：(1)对象相对其他对象隐藏它的表示；(2)将一些数据存取操作的问题分解成一些交互的代理程序的集合。
- 缺点：(1)必须通过对象的标识进行交互；(2)必须修改所有显式调用它的其他对象，并消除由此带来的一些副作用。

管道和过滤(Pipe-and-Filter)



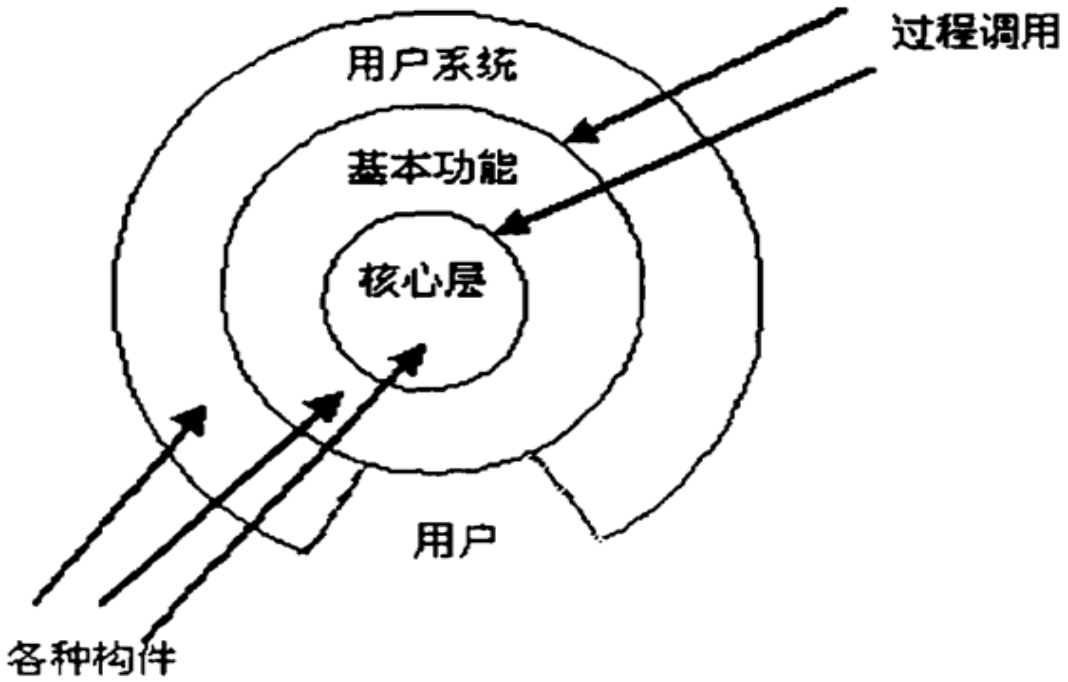
- 优点：(1) 结构简单；(2) 系统易于维护和增强；(3) 支持复用；(4) 各过滤器可以并发运行。
- 缺点：(1) 通常不适合交互式的应用；(2)通常导致进程成为批处理的结构；(3)因为在数据传输上没有通用的标准，每个过滤器都增加了解析和合成数据的工作。

Event-Based Implicit Invocation



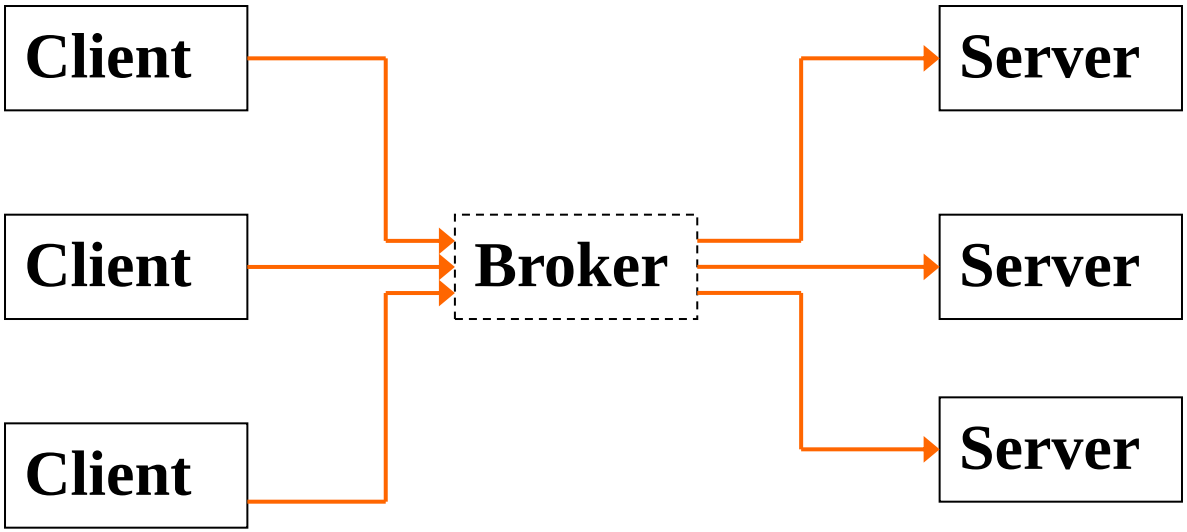
- 优点：(1) 支持复用，构件通过登记它所感兴趣的事件被引入系统；(2) 便于系统演化，构件可以容易地升级或更换。
- 缺点：(1) 系统行为难以控制，发出事件的构件放弃了对系统的控制，因此不能确定系统中有无或有多少其它构件对该事件感兴趣，系统的行为不能依赖于特定的处理顺序。(2) 同事件关联的会有少量的数据，但有些情况下需要通过共享区传递数据，这时就涉及到全局效率和资源管理问题。

Layered Systems



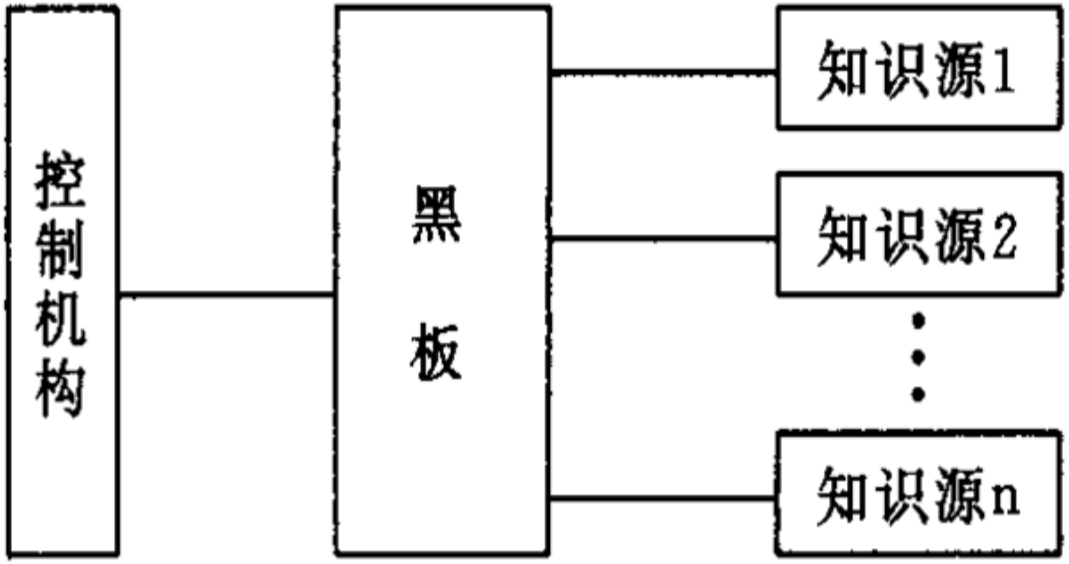
- 优点：(1)支持基于抽象程度递增的系统设计，使得设计者可以把一个复杂系统按递增的步骤分解开。(2)支持功能扩展，每一层至多和相邻的层次交互。(3)支持复用，只要服务接口定义不变，不同的实现可以交换使用。
- 缺点：并不是每个系统都可以很容易的划分层次。

代理(Broker)



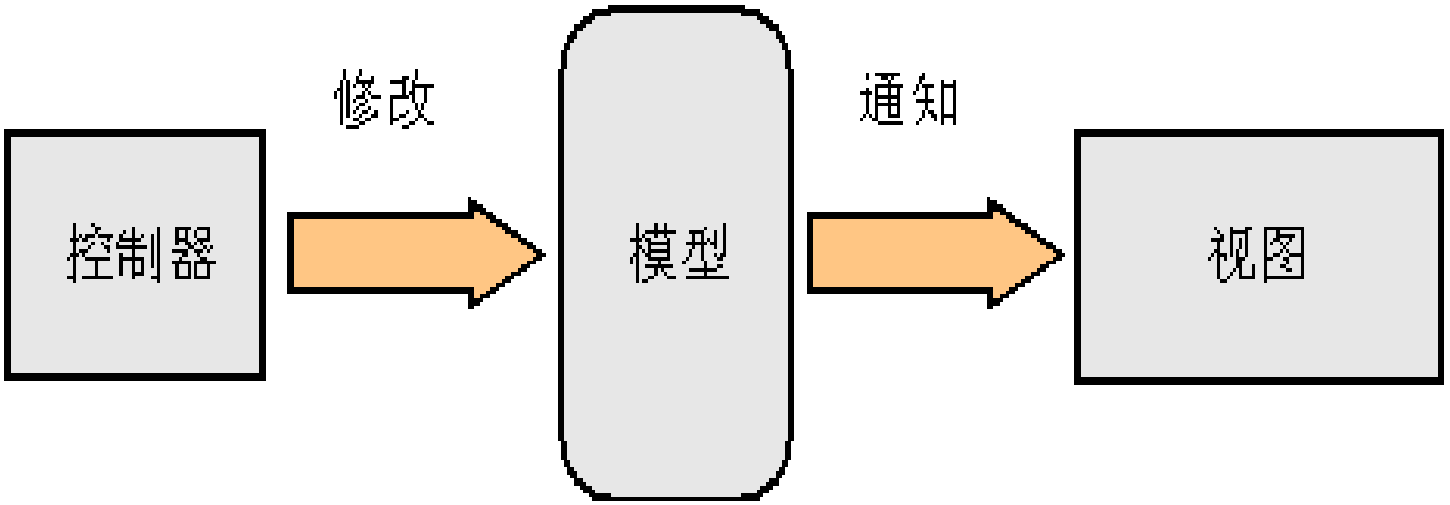
- 优点：(1)定位透明性；(2)组件的可变性和可扩展性；(3)代理系统的可移植性；(4)不同代理系统之间的互操作性；(5)可重用性。
- 缺点：(1)效率受限；(2)容错性较差。

黑板(Blackboard)



- 优点：(1)可更改性和可维护行；(2)可重用的知识源；(3)支持容错性和健壮性。
- 缺点：(1)测试困难；(2)不能保证有好的求解方案；(3)难以建立一个好的控制策略；(4)缺少对并行机制的支持。

MVC (Model–View–Controller)



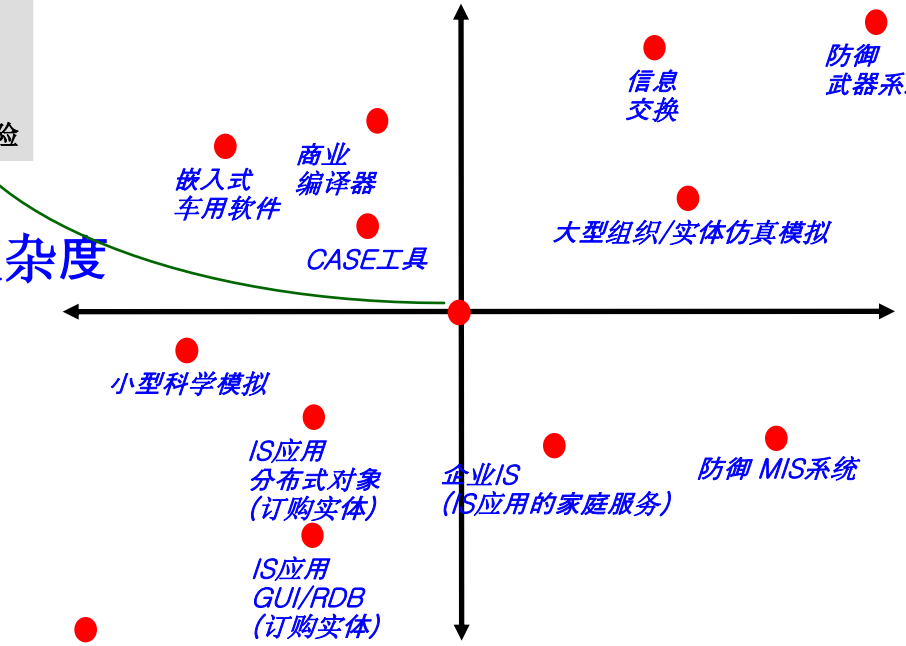
- 优点：(1)可以为一个模型在运行时同时建立和使用多个视图；(2)视图与控制器的可接插性，允许更换视图和控制器对象；(3)模型的可移植性。
- 缺点：(1)增加了系统结构和实现的复杂性；(2)视图与控制器间的过于紧密的连接，这样就妨碍了他们的独立重用。

软件复杂性的度数

更高的技术复杂性
- 嵌入式，实时的，分布式的，不可出错的
- 定制的，空前的，可复用的
- 高性能的

一个比较中等的项目
- 5–10 人
- 10–15 个月的开发周期
- 3–5 个外部界面
- 一些不可知的事情 & 风险

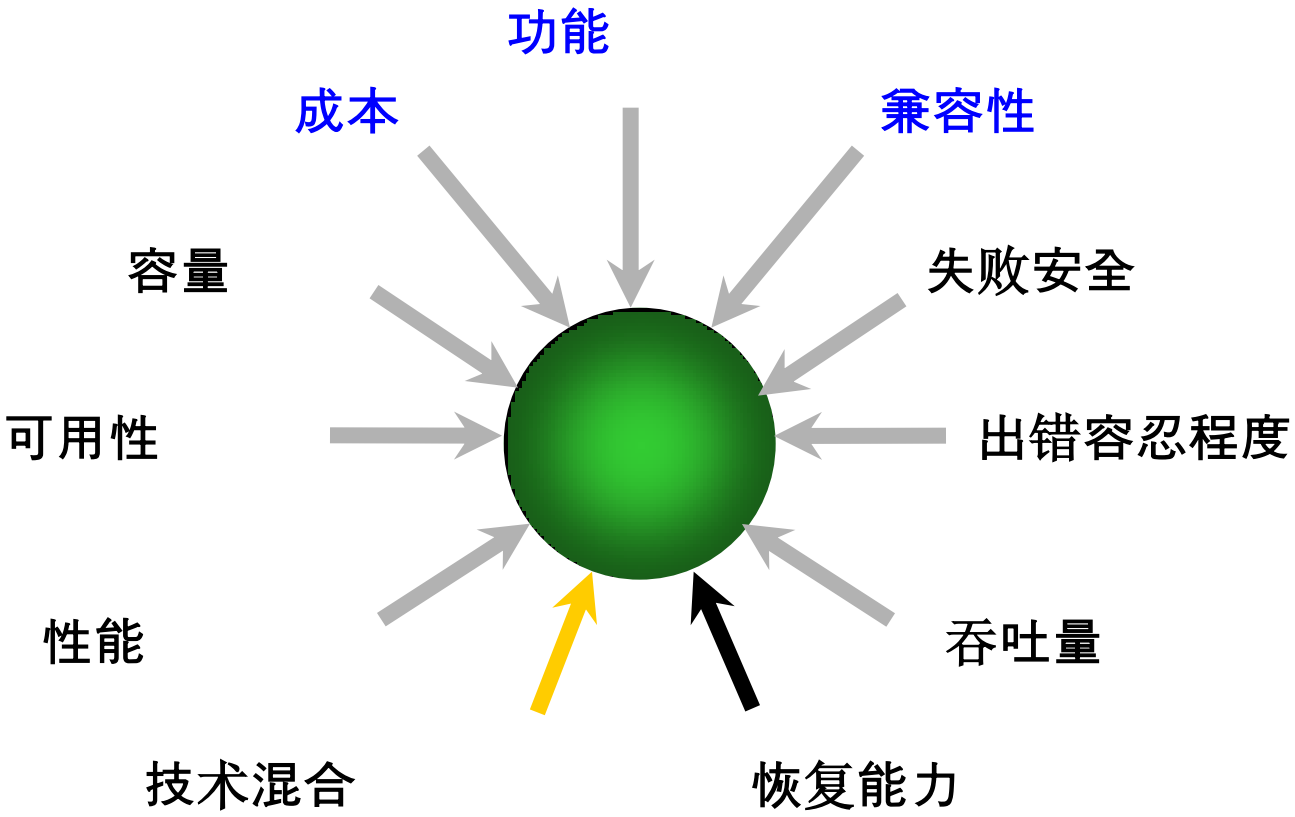
较低的管理复杂度
- 小范围
- 非正式的
- 简单的资金运作
- “产品”



较高管理复杂度
- 大范围
- 合同契约性
- 多数人控制的
- “项目”

低技术复杂度
- 大部分是4GL，或基于组件技术的
- 应用反向工程
- 交互性能

软件中的影响因素



20年之后的挑战不是速度、成本和性能，而是复杂度的问题了。

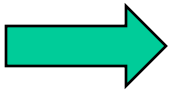
Bill Raduchel, Sun微系统公司策略执行总裁

复杂度是我们的敌人，是我们的目标，我们要消灭它。

不同的投资者，不同的观点

- 体系结构有许多不同兴趣者的不同看法
  - 最终用户
  - 客户
  - 项目管理者
  - 系统工程师
  - 开发者
  - 构建者
  - 维护人员
  - 其他开发人员
- 多维实体
- 更多的投资者

多个观点，多个蓝图



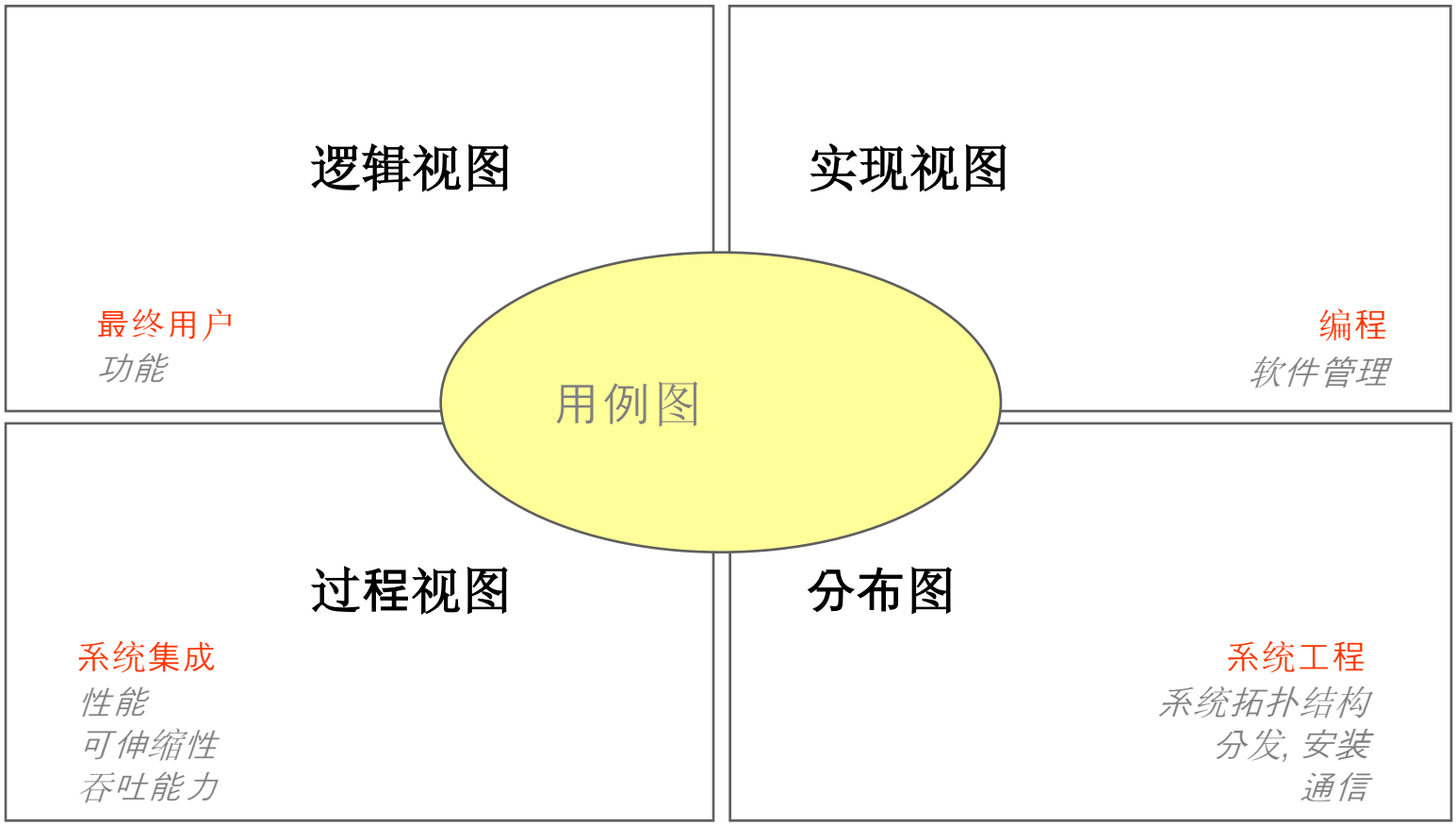
体系结构重要的元素

- 不是所有的设计都是体系结构
- 主要的“商业”类
- 重要的商业机制
- 处理器和处理过程
- 层和子系统
- 结构视图 = 模型切片

好的体系结构的特征

- 可伸缩性的
- 简单
- 亲切的
- 关系清楚明了
- 职责分布明确
- 效益和技术平衡

描绘系统体系结构



概念模型

物理模型

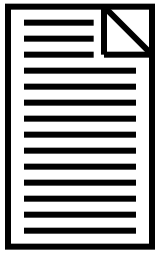
需要多少视图?

- 适合需要的简单模型
- 不是所有的系统都需要所有的视图
  - 单处理器: 不用分布图
  - 简单处理过程: 不用过程视图
  - 很小的程序: 不用实现视图
- 增加视图:
  - 数据视图, 安全视图

什么是架构机制？

Required Functionality

Implementation Environment



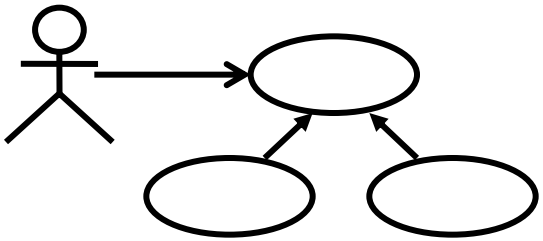
Supplementary Specification

"realized by client classes using"

Mechanisms

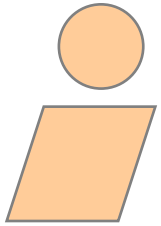
"constrained by"

COTS Products
Databases
IPC Technology, etc.



Use-Case Model

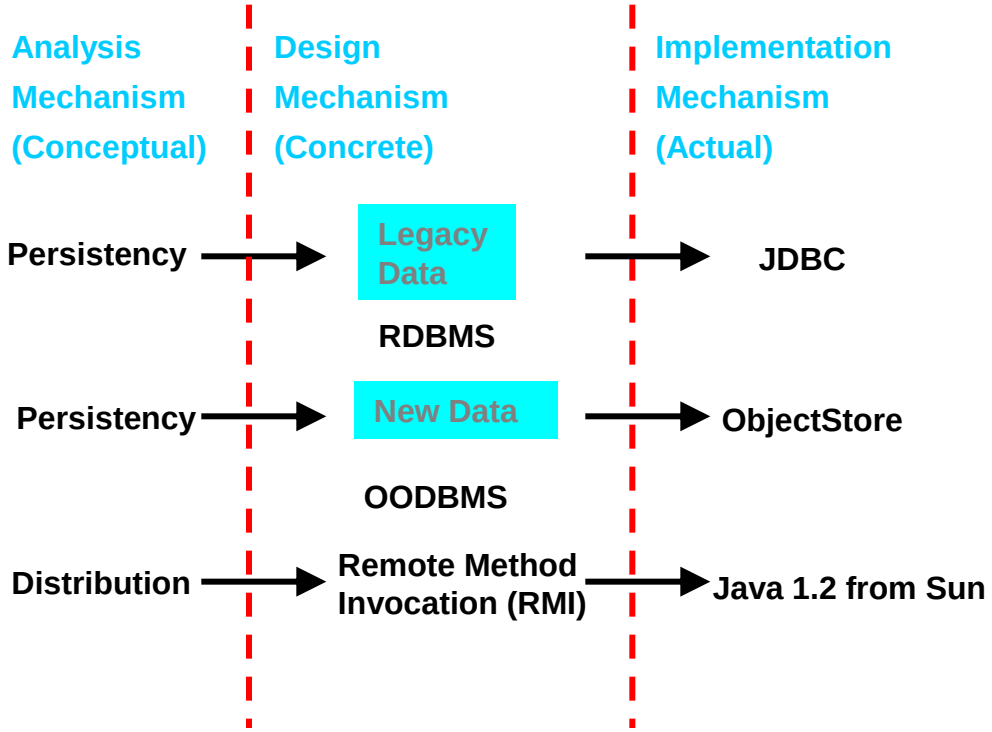
"responsible for"



Architect

架构机制：三种机制

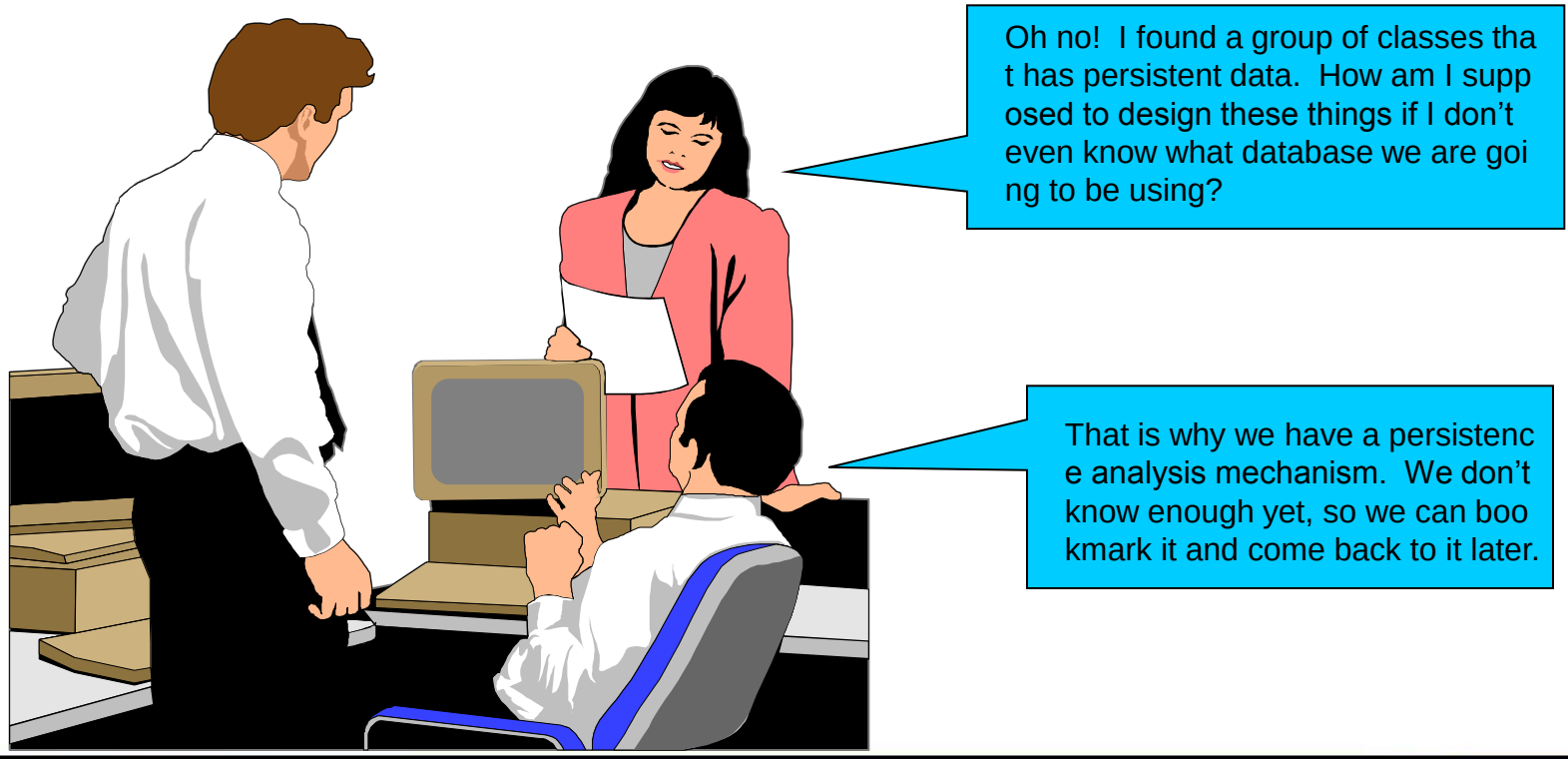
- **Architectural Mechanism Categories**
  - Analysis mechanisms (conceptual)
  - Design mechanisms (concrete)
  - Implementation mechanisms (actual)



构架分析——分析机制

- 分析机制

  Analysis mechanisms are used during analysis to reduce the complexity of analysis and to improve its consistency by providing designers with a shorthand representation for complex behavior.



构架分析——分析机制

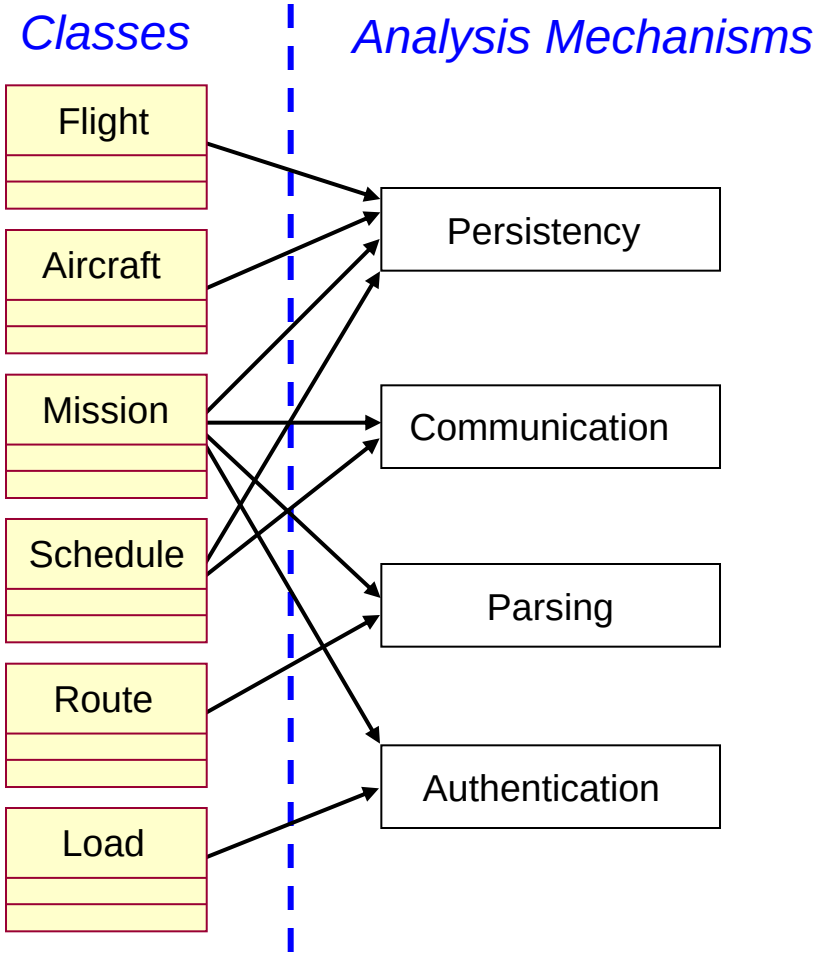
- 分析机制
  - 持久性
  - 通信 (进程间通信 、远程过程调用)
  - 消息传递
  - 分发
  - 事务管理
  - 进程控制和同步
  - 信息交换
  - 格式转换
  - 安全
  - 错误检测 / 处理 /报告
  - 冗余
  - 接口提供

构架分析——分析机制

- 分析机制
  - 持久机制
    - 粒度、数量、持续时间、存取机制、存取频率、可靠性
  - 进程间通信机制
    - 响应时间、同步、消息大小、协议
  - 安全机制
    - 数据大小、用户数量、安全规则、权限
  - ……

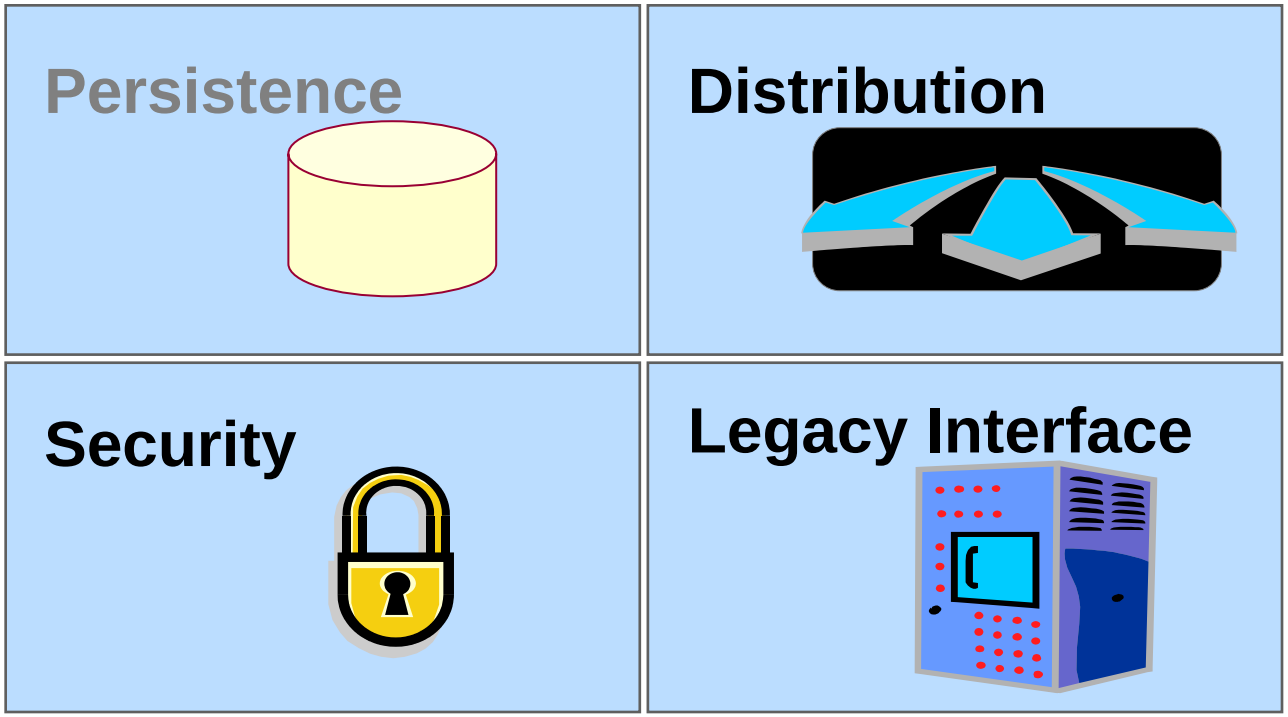
构架分析——分析机制

- 描述分析机制
  - 将分析机制收集为一张列表
  - 绘制类到分析机制的映射关系
  - 识别分析机制的特征
  - 使用协作建模



构架分析——分析机制

- 分析机制举例
  - 课程注册系统的分析机制



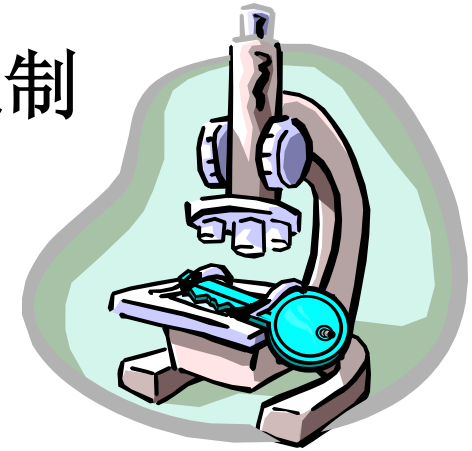
构架分析——标识关键抽象

- 标识关键抽象 (Key abstractions)
  - 关键抽象

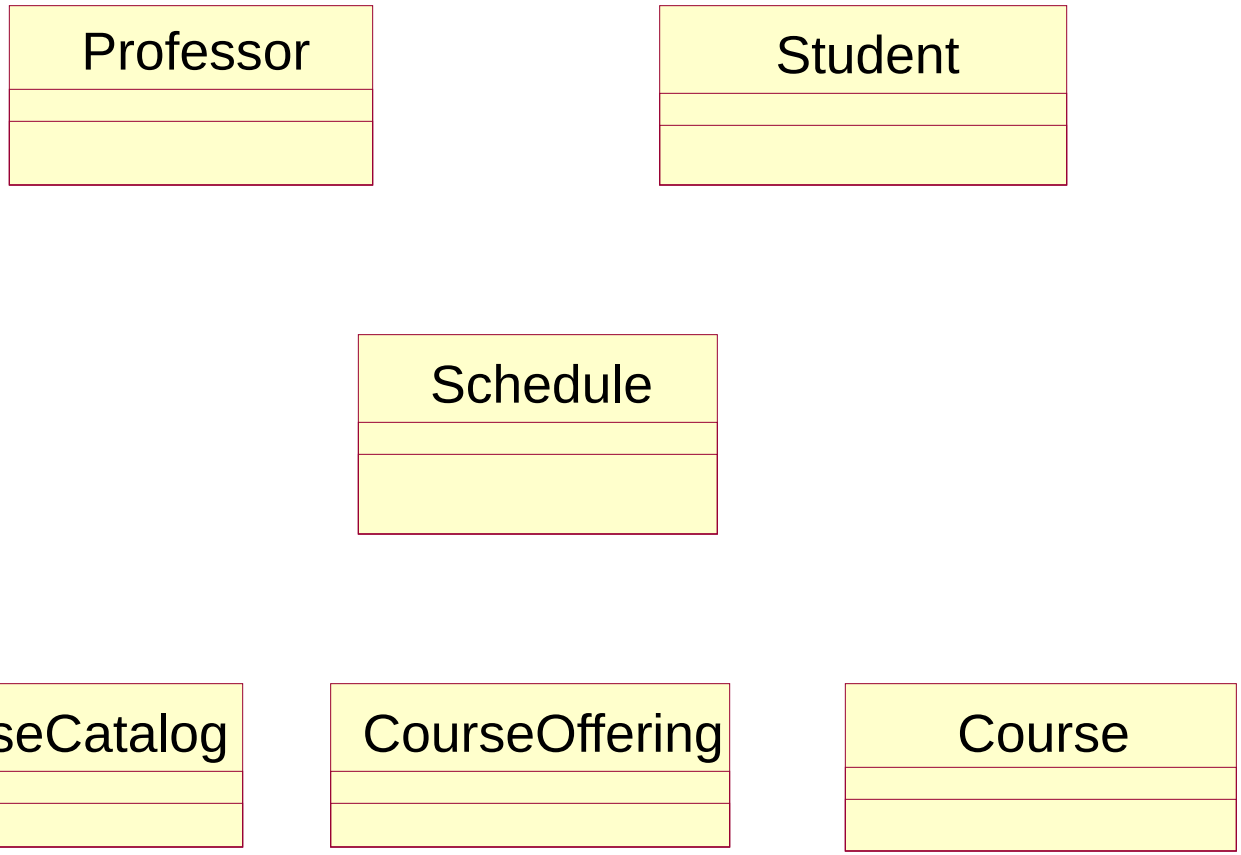
    A key abstraction is a concept, normally uncovered in requirements, that the system must be able to handle.
  - 来源
    - 领域知识
    - 需求
    - 词汇表
    - 领域模型、业务模型……

构架分析——标识关键抽象

- 定义关键抽象的步骤
  - 定义分析类
  - 在类图上建模分析类和类间的关系
    - 关系反映了抽象间的语义联系
    - 而不用于支持实现或通信
  - 将分析类映射到必要的分析机制



Example: Key Abstractions



构架分析——创建用例实现

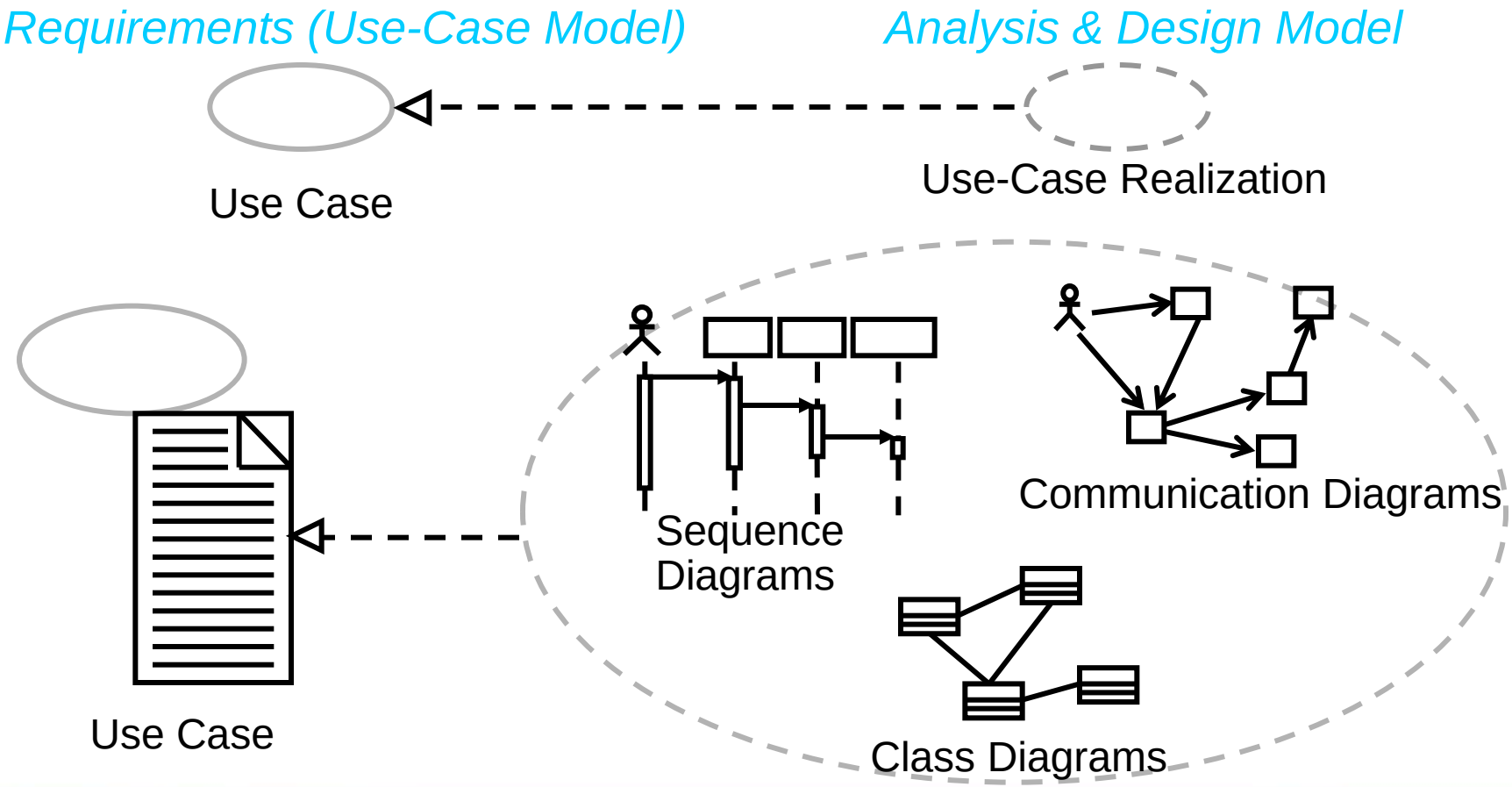
- 创建用例实现
  - 用例实现
    - 用例的构造型<<**use case realization**>>
    - 由一组类及类间的交互构成
    - 实现了用例所说明的行为

Place Order　　Place Order Realization

  - 提供了从分析&设计到需求的可追踪性

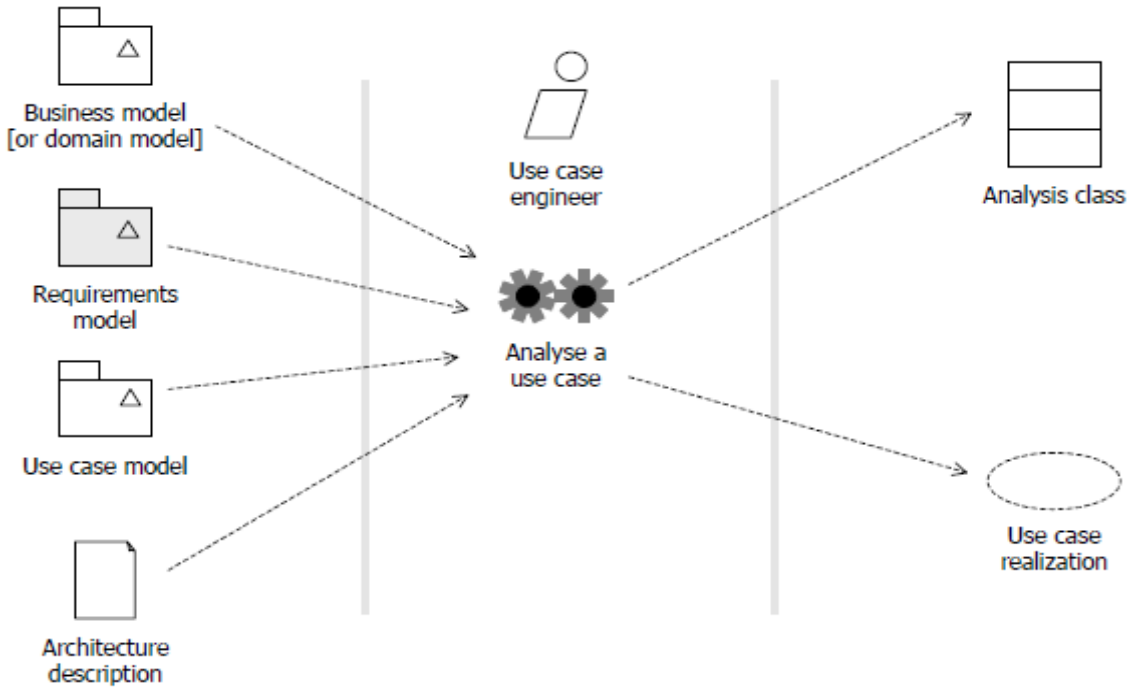
构架分析——创建用例实现

- 用例实现

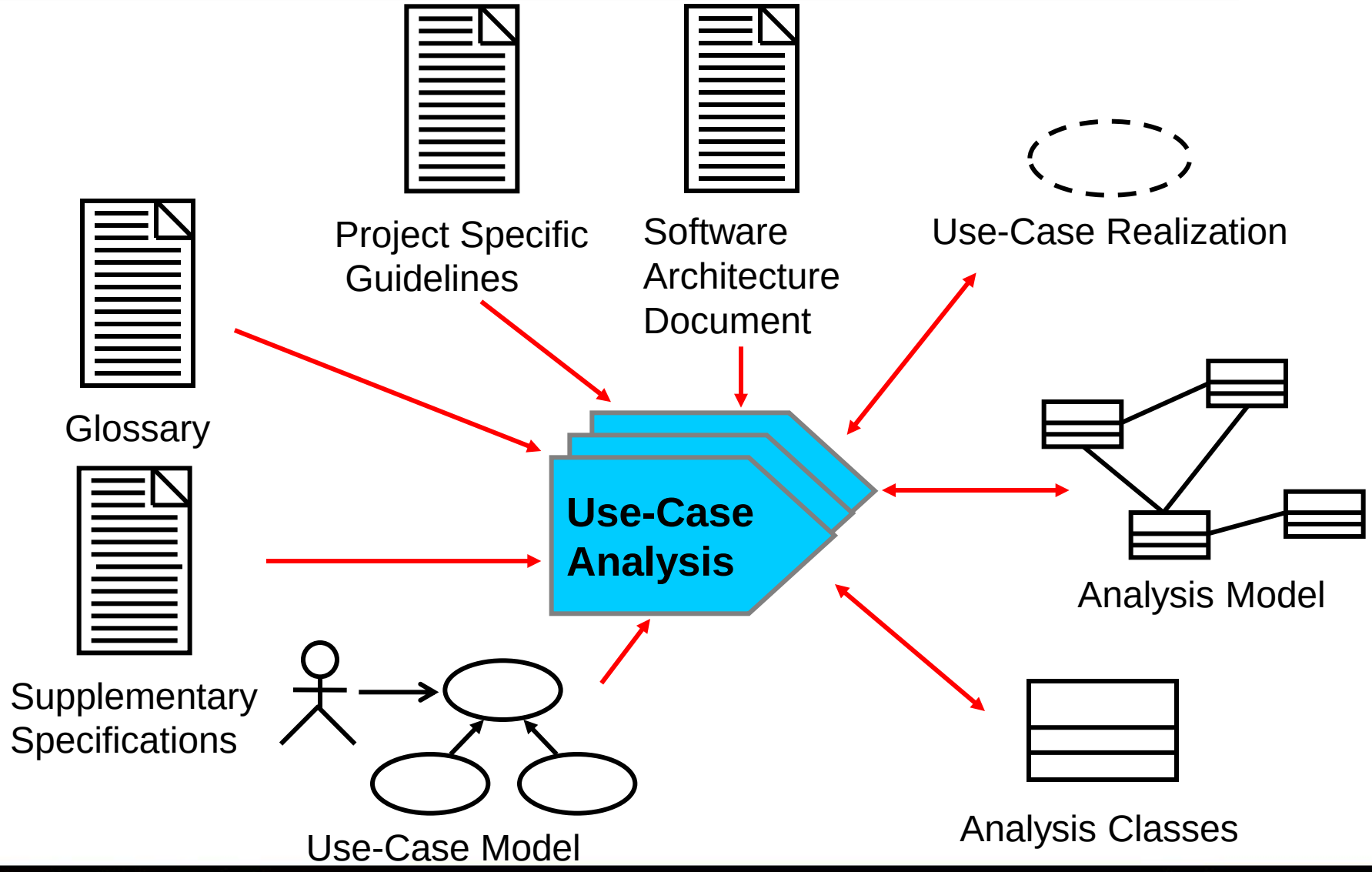


用例分析

- 用例分析
  - 角色
  - 输入工件
  - 输出工件



用例分析

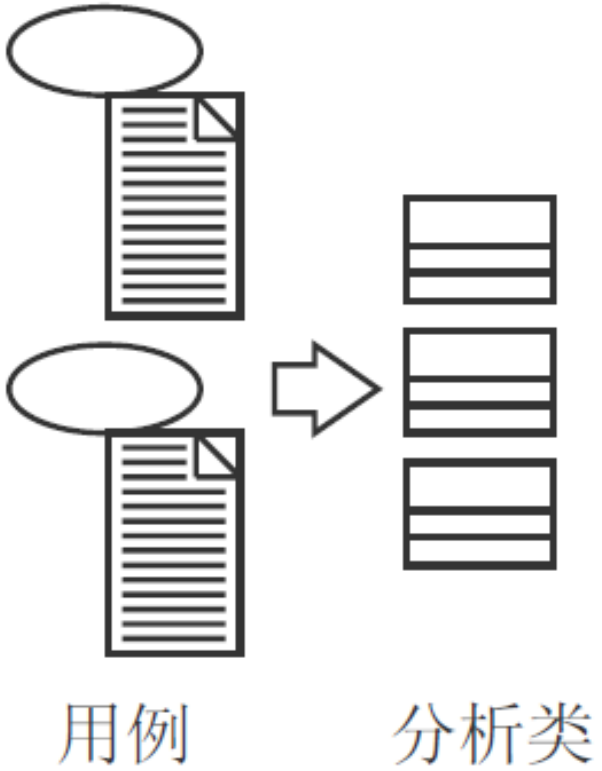

Project Specific Guidelines
Software Architecture Document
Use-Case Realization
Glossary
Use-Case Analysis
Analysis Model
Supplementary Specifications
Use-Case Model
Analysis Classes

用例分析

- 用例实现（分析）
  - 组成元素
    - 分析类图、交互图
    - 特殊要求、用例精化
  - 展示分析类的实例如何交互以实现系统功能
  - 目标
    - 找出分析类的交互以实现用例行为
    - 找出类实例间的消息以实现特定行为

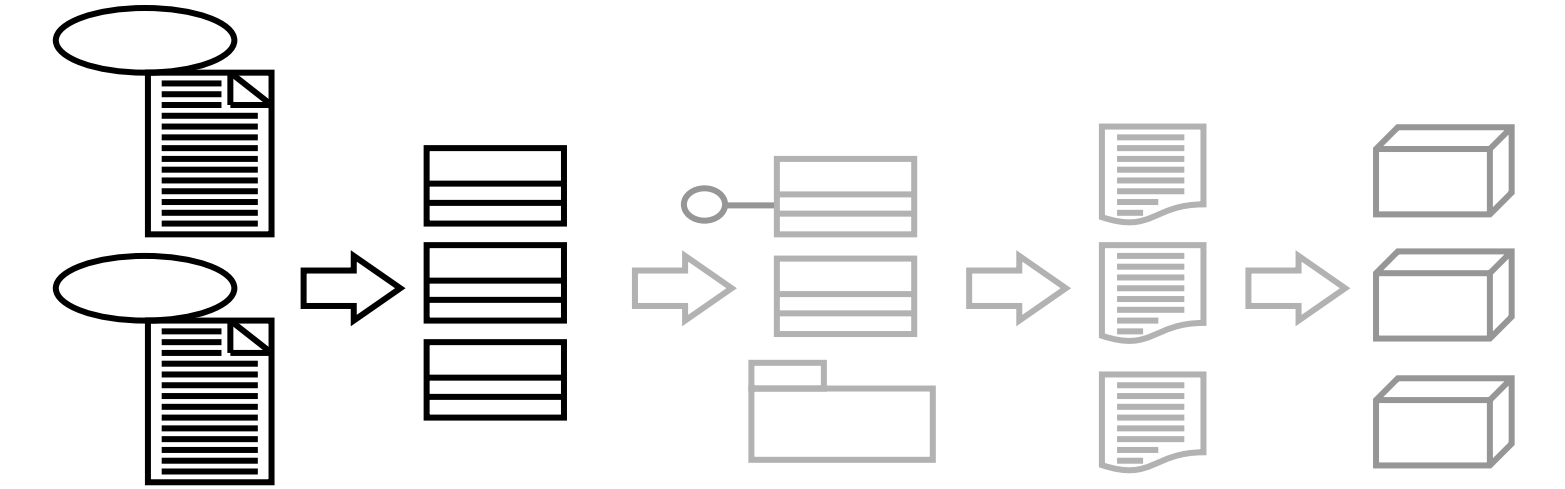
用例分析

- 用例分析的步骤
  - 补充用例说明
  - 对每一个用例实现
    - 找出分析类
    - 将用例行为分发给类
  - 对每个分析类
    - 描述职责
    - 描述属性和关系
  - 统一分析类



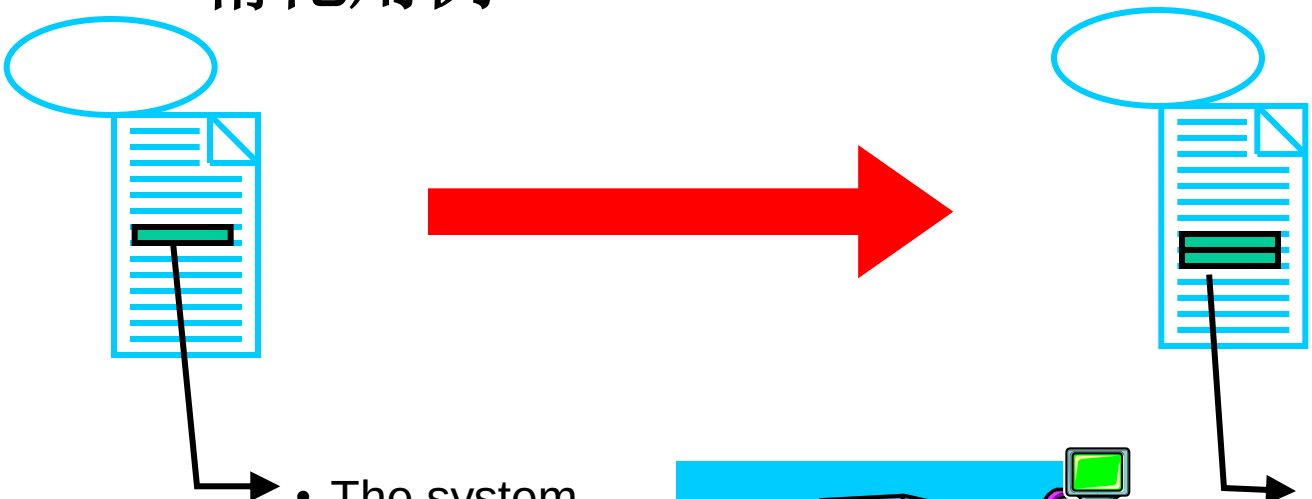
用例分析

- 用例分析
  - 以找出并定义分析类为核心

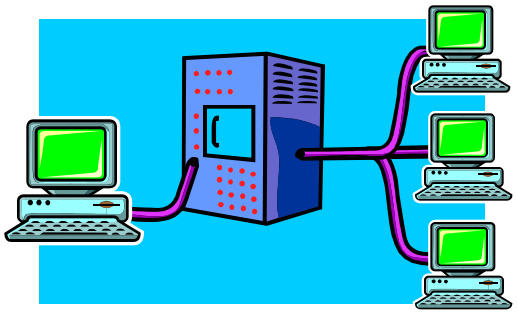


用例分析

- 补充用例说明
  - 精化用例



- The system displays a list of course offerings.



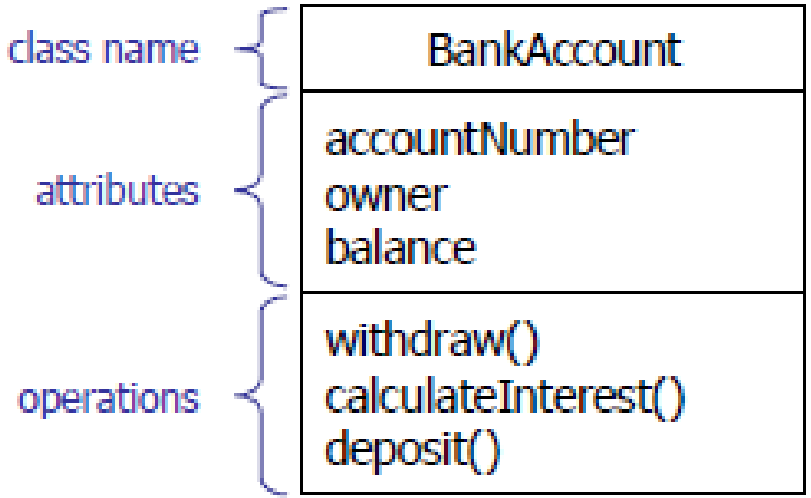
- The system retrieves and displays a list of current course offerings from the course catalog legacy database.

用例分析

- 分析类
  - 特征
    - 代表问题域中的简洁抽象
    - 应该映射到真实世界的业务概念
  - 属性
    - 高级层次的属性集合
    - 为设计类捕获候选属性准备了条件
  - 操作
    - 说明类必须提供的关键服务
    - 一个分析级操作常分解为多种设计级操作

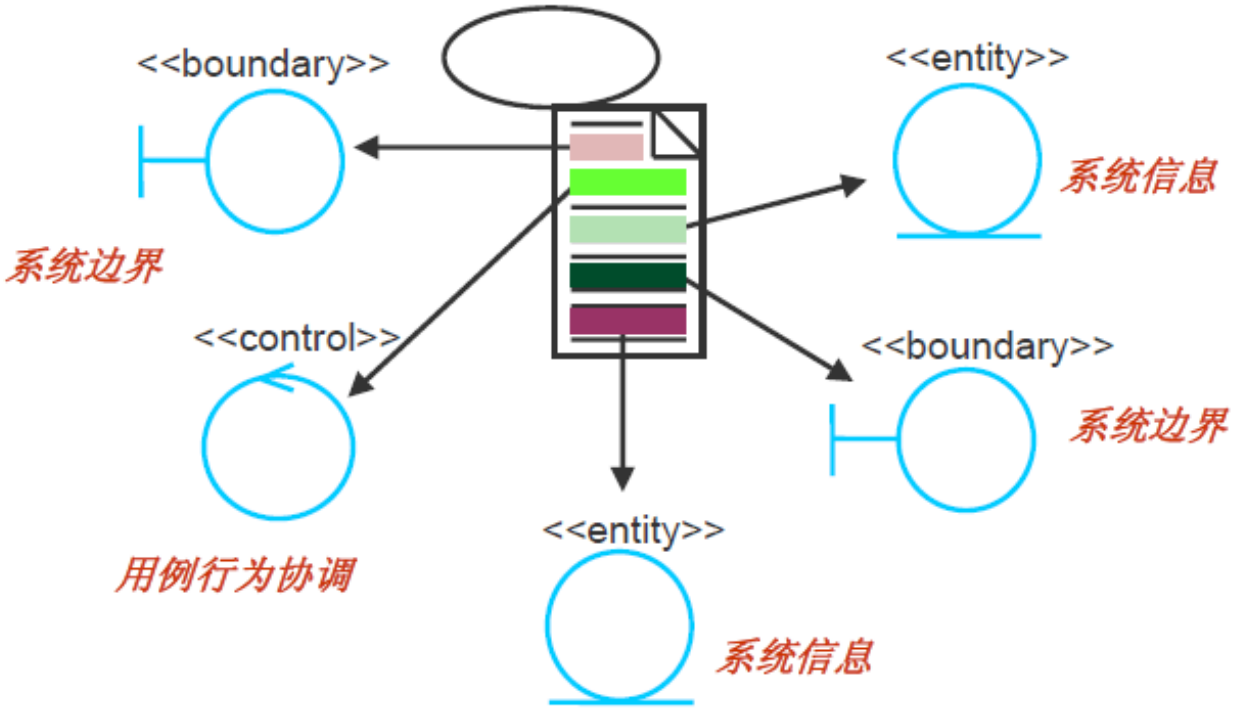
用例分析

- 分析类
  - 分析类的最小形式
    - 名称：强制
    - 属性：名称强制，类型可选
    - 操作：对类职责的高层陈述
    - 可视性：不显示
    - 构造型：可选
    - 标记值：可选



用例分析

- 分析类
  - UML中分析类的构造型



用例分析

- 找出分析类
  - 使用名词/动词分析找出类
    - 收集相关信息
      - 补充的需求规格说明
      - 用例
      - 项目词汇表
      - 其他文档
    - 分析信息
      - 名词、名词短语　　类或属性
      - 动词、动词短语　　操作

用例分析

- 确定潜在的类
  - 陈述中的名词和名词短语，以不同形式展现
    - 外部实体（如其它系统、设备、人员），他们生产或消费计算机系统所使用的信息；
    - 物体（如报告、显示、信函、信号），它们是问题域的一部分；
    - 发生的事情或事件（如，性能改变或完成一组机器人移动动作），它们出现在系统运行的环境中；
    - 角色（如管理者、工程师、销售员），他们由与系统交互的人扮演；
    - 组织单位（如，部门、小组、小队），他们与一个应用有关；
    - 场所（如制造场所、装载码头），它们建立问题和系统所有功能的环境；
    - 构造物（如四轮交通工具、计算机），它们定义一类对象，或者定义对象的相关类。

用例分析

- 通过回答下列问题识别潜在的类
  - 是否有要储存、转换、分析或处理的信息？
  - 是否有外部系统？
  - 是否有模式（pattern）、类库和构件等？
  - 是否有系统必须处理的设备？
  - 是否有组织部分（organizational parts）？
  - 业务中的执行者扮演什么角色？这些角色可以看作类，如客户、操作员等。

用例分析

- 筛选并确定最终的类
  - 1) 保留的信息：
    - 仅当必须记住有关潜在对象的信息，系统才能运作时，则该潜在对象在分析阶段是有用的；
  - 2) 需要的服务：
    - 潜在对象必须拥有一组可标识的操作，它们可以按某种方式修改对象属性的值；
  - 3) 多个属性：
    - 在分析阶段，关注点应该是“较大的”信息（仅具有单个属性的对象在设计时可能有用，但在分析阶段，最好把它表示为另一对象的属性）；

用例分析

- 筛选并确定最终的类
  - 4）公共属性：
    - 可以为潜在的对象定义一组属性，这些属性适用于该对象所有发生的事情；
  - 5) 公共操作：
    - 可以为潜在的对象定义一组操作，这些操作适用于该对象所有发生的事情；
  - 6) 必要的需求：
    - 出现在问题空间中的外部实体以及对系统的任何解决方案的实施都是必要的生产或消费信息，它们几乎总是定义为需求模型中的对象。

用例分析

- 标识属性
  - 类的稳定特性
  - “在当前问题范围内，什么数据项完整定义了该对象”？
- 定义操作
  - 以某种方式操纵数据的操作（如，增加、删除、重新格式化、选择）；
  - 完成某种计算的操作；
  - 为控制事件的发生而监控对象的操作。

用例分析

- 发现类的例子
- 小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计

用例分析

- 发现类的例子
- 小王是一个爱书之人，家里各类书籍已过千册，而平时又时常有朋友外借，因此需要一个个人图书管理系统。该系统应该能够将书籍的基本信息按计算机类、非计算机类分别建档，实现按书名、作者、类别、出版社等关键字的组合查询功能。在使用该系统录入新书籍时系统会自动按规则生成书号，可以修改信息，但一经创建就不允许删除。该系统还应该能够对书籍的外借情况进行记录，可对外借情况列表打印。另外，还希望能够对书籍的购买金额、册数按特定时间周期进行统计

筛选备选类

- “小王”、“人”、“家里”很明显是系统外的概念，无须对其建模;
- 而“个人图书管理系统”、“系统”指的就是将要开发的系统，即系统本身，也无须对其进行建模;
- 很明显“书籍”是一个很重要的类，而“书名”、“作者”、“类别”、“出版社”、“书号”则都是用来描述书籍的基本信息的，因此应该作为“书籍”类的属性处理，而“规则”是指书号的生成规则，而书号则是书籍的一个属性，因此“规则”可以作为编写“书籍”类构造函数的指南。
- “基本信息”则是书名、作者、类别等描述书籍的基本信息统称，“关键字”则是代表其中之一，因此无需对其建模;
- “功能”、“新书籍”、“信息”、“记录”都是在描述需求时使用到的一些相关词语，并不是问题域的本质，因此先可以将其淘汰掉;

筛选修选类

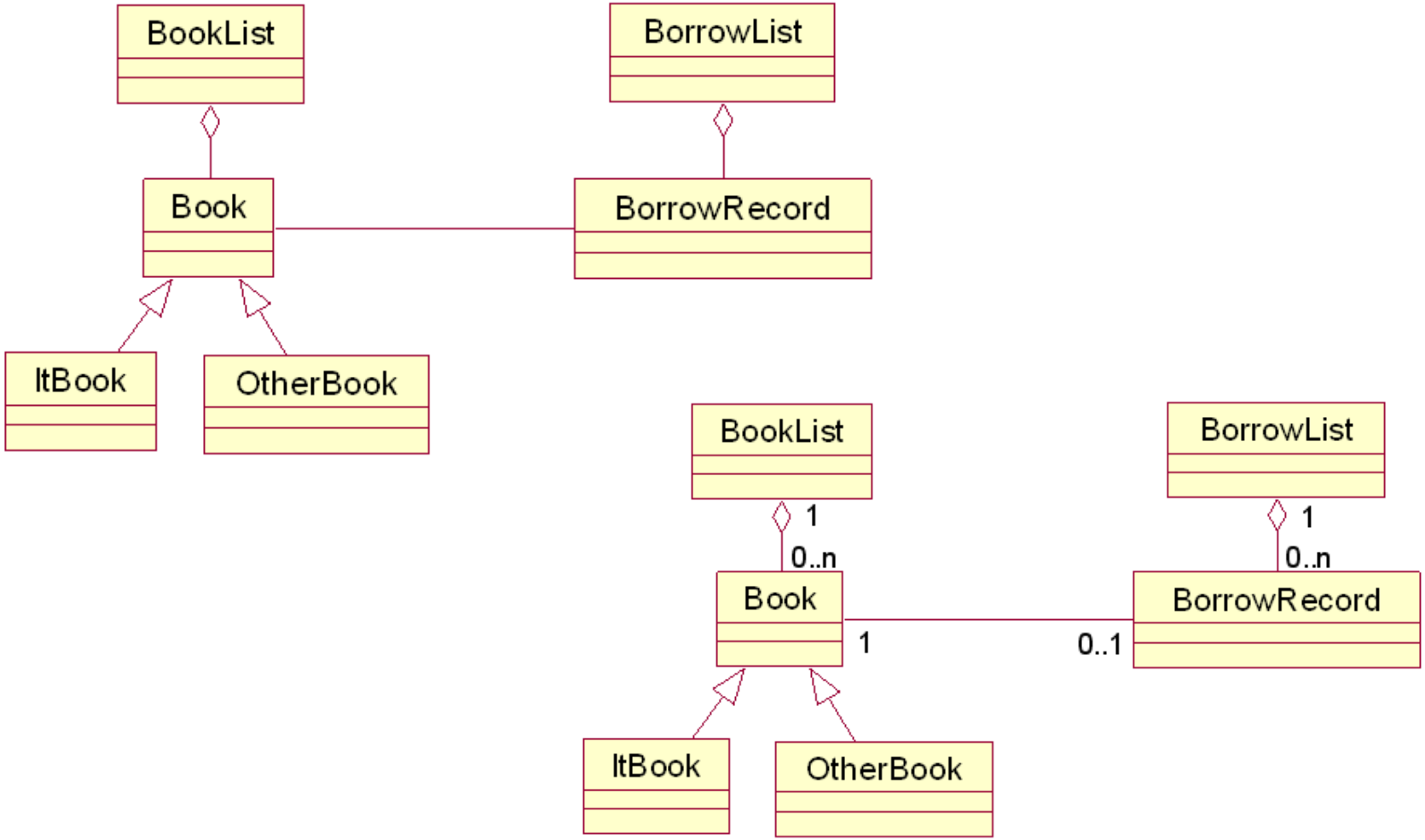
- “计算机类”、“非计算机类”是该系统中图书的两大分类，因此应该对其建模，并改名为“计算机类书籍”和“非计算机类书籍”，以减少歧义；
- “外借情况”则是用来表示一次借阅行为，应该成为一个候选类，多个外借情况将组成“外借情况列表”，而外借情况中一个很重要的角色是“朋友”—借阅主体。虽然到本系统中并不需要建立“朋友”的资料库，但考虑到可能会需要列出某个朋友的借阅情况，因此还是将其列为候选类。为了能够更好地表述，将“外借情况”改名为“借阅记录”，而将“外借情况列表”改名为“借阅记录列表”；
- “购买金额”、“册数”都是统计的结果，都是一个数字，因此不用将其建模，而“特定时限”则是统计的范围，也无需将其建模；不过从这里的分析中，我们可以发现，在该需求描述中隐藏着一个关键类—书籍列表，也就是执行统计的主体。

得到候选类

书籍	计算机类书籍	非计算机类书籍
借阅记录	借阅记录列表	书籍列表

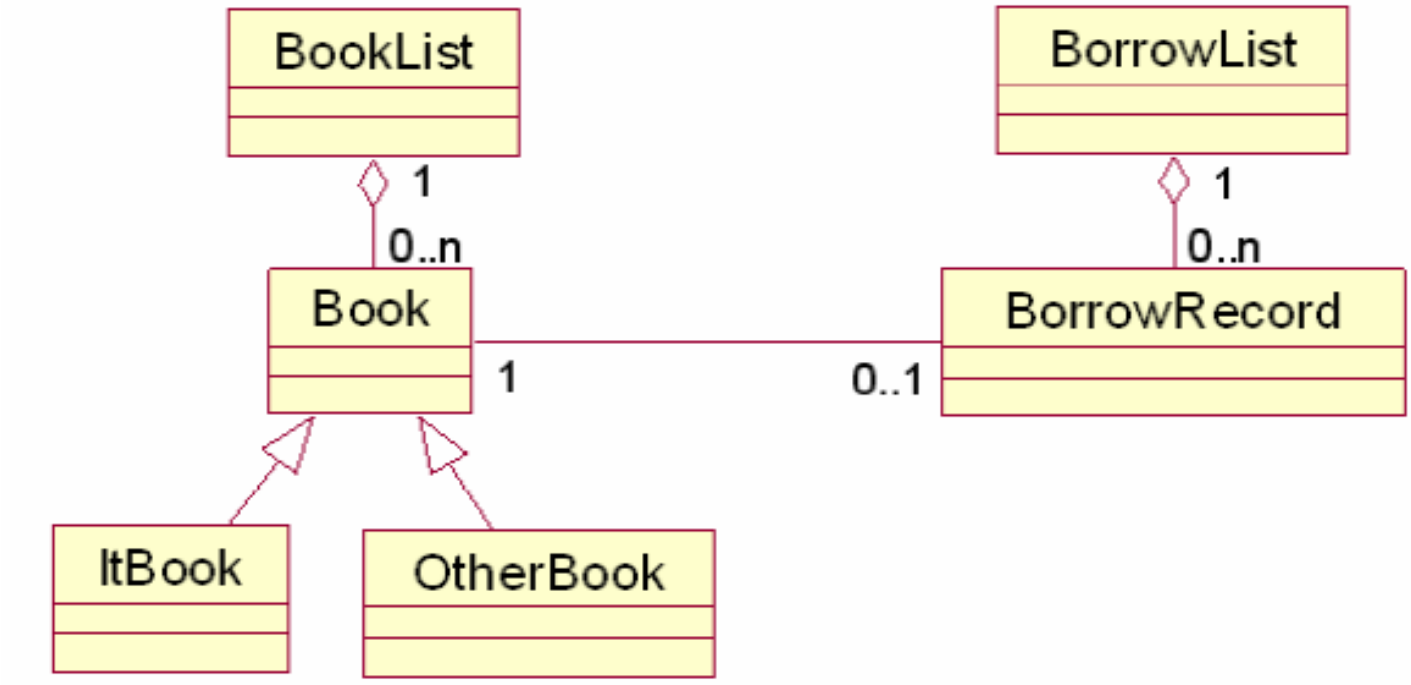
- 在使用“名词动词法”寻找类的时候，很多团队会在此耗费大量的时间，特别是对于中大型项目，这样很容易迷失方向。其实在此主要的目的是对问题领域建立概要的了解，无需太过咬文嚼字

关联分析，建模，多重性分析，再建模



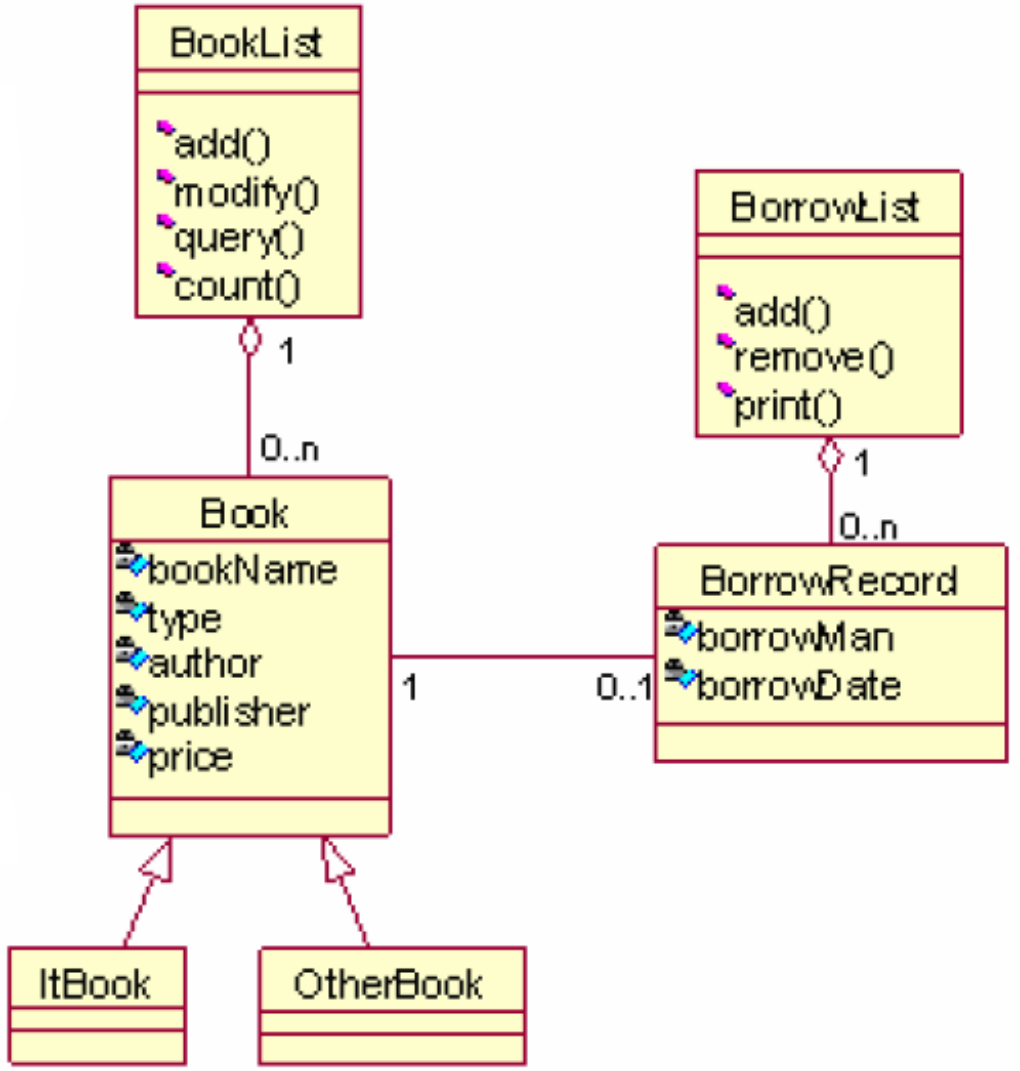
用例分析

- 发现类的例子


BookList
BorrowList
1
0..n
1
0..n
Book
BorrowRecord
1
0..1
ItBook
OtherBook

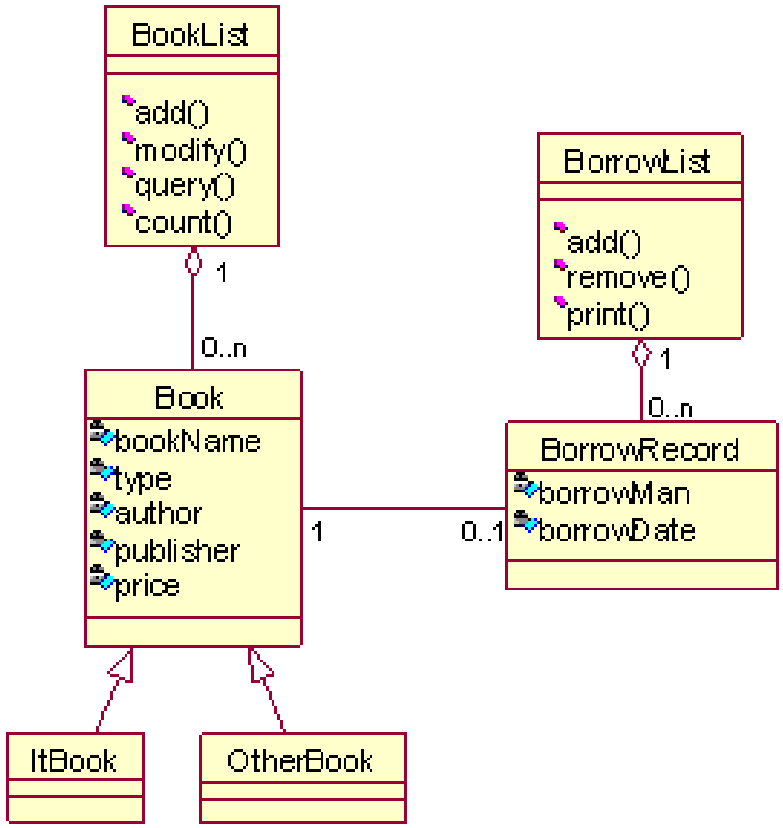
用例分析

- 发现类的例子


BookList
add()
modify()
query()
count()
1
0..n
BorrowList
add()
remove()
print()
1
0..n
Book
bookName
type
author
publisher
price
1
0..1
BorrowRecord
borrowMan
borrowDate
ItBook
OtherBook

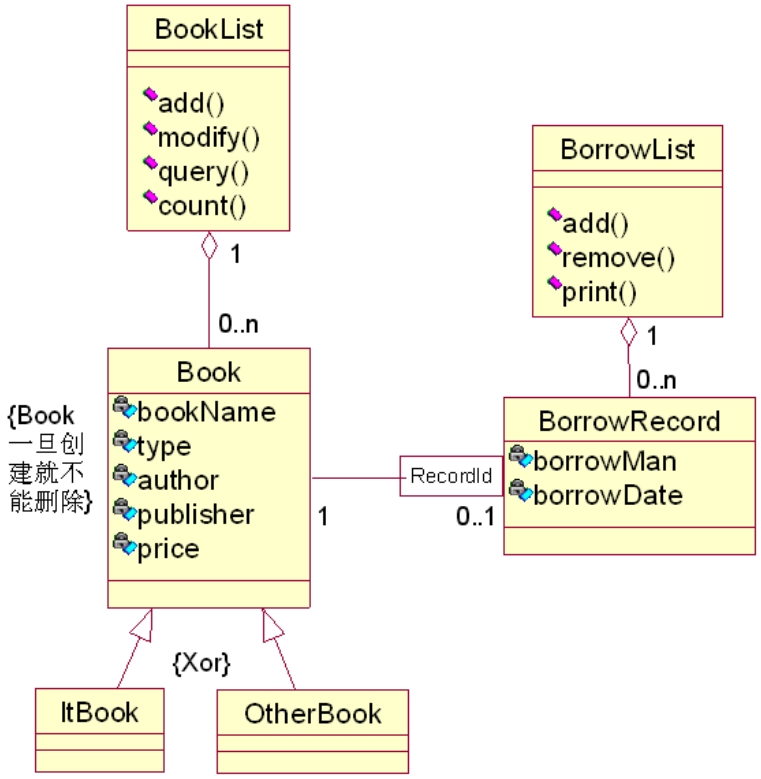
职责分析

- 书籍类：从需求描述中，可找到书名、类别、作者、出版社；同时从统计的需要中，可得知“定价”也是一个关键的成员变量。
- 书籍列表类：书籍列表就是全部的藏书列表，其主要的成员方法是新增、修改、查询（按关键字查询）、统计（按特定时限统计册数与金额）。
- 借阅记录类：借阅人（朋友）、借阅时间。
- 借阅记录列表类：主要职责就是添加记录（借出）、删除记录（归还）以及打印借阅记录



限定与修改

- 导航性分析：Book与BookList之间、BorrowRecord和BorrowList之间是组合关系均无需添加方向描述，而Book与BorrowRecord之间则是双方关联，也无需添加
- 约束：Book对象创建后就不能够被删除只能被修改，因此在Book类边上加上用自由文本写的约束；一本书要么属于计算机类，要么属于非计算机类，因此在ItBook和OtherBook间加了 “{Xor}”约束
- 限定符：一本书只有一册，因此只能够被借一次，因此对于一本Book而言只能有一个RecordId与其对应



用例分析

- 找出分析类
  - 使用CRC分析找出类
    - CRC分析：C(class)
      R(responsibility
      C(collaborator)

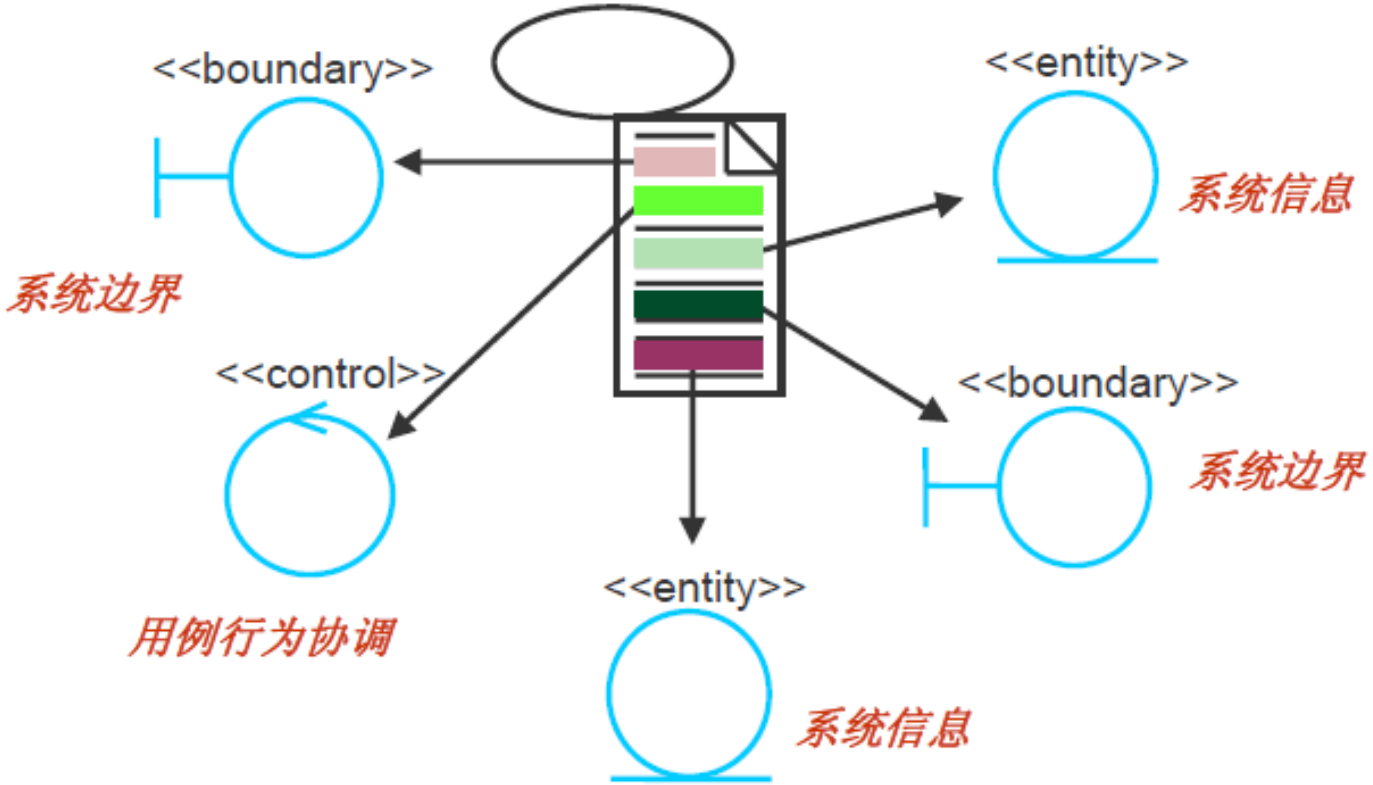
Class name: BankAccount	
Responsibilities: Maintain balance	Collaborators: Bank

    - 分析过程

脑力风暴——收集信息　　分析信息

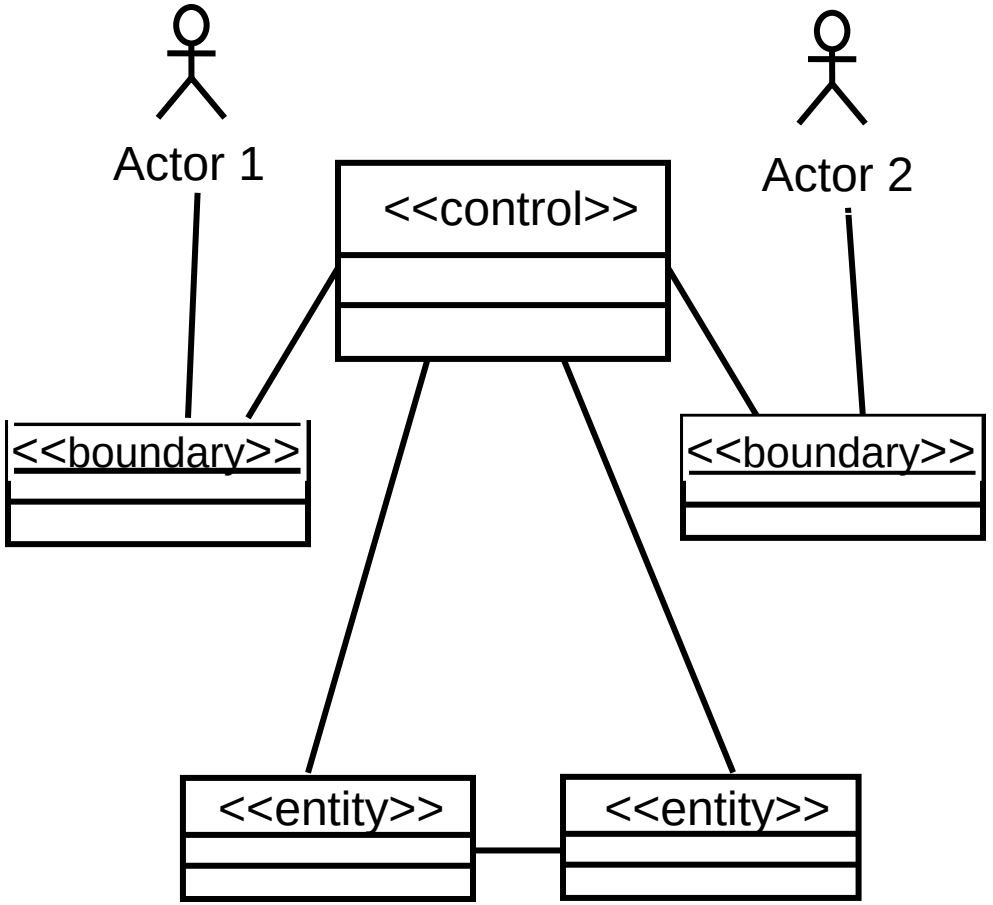
用例分析

- 找出分析类
  - 采用构造型找出类



用例分析

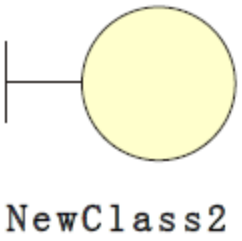
- 分析类
  - 边界类 (boundary)
    - 图标
    - 同外部参与者通信
  - 实体类 (entity)
    - 图标
    - 建模事物的信息
  - 控制类 (control)
    - 图标
    - 协调系统行为

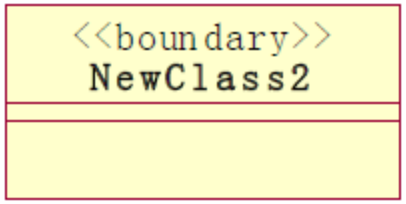


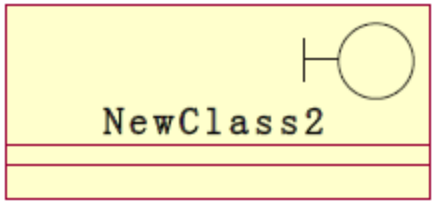
用例分析

- 边界类
  - 建模系统与环境的交互
  - 系统边界与系统外部某物的调解者
  - 类型
    - 用户界面类
    - 系统接口类
    - 设备接口类

参与者	暗示
代表人	用户界面类
代表系统	系统接口类
代表设备	设备接口类

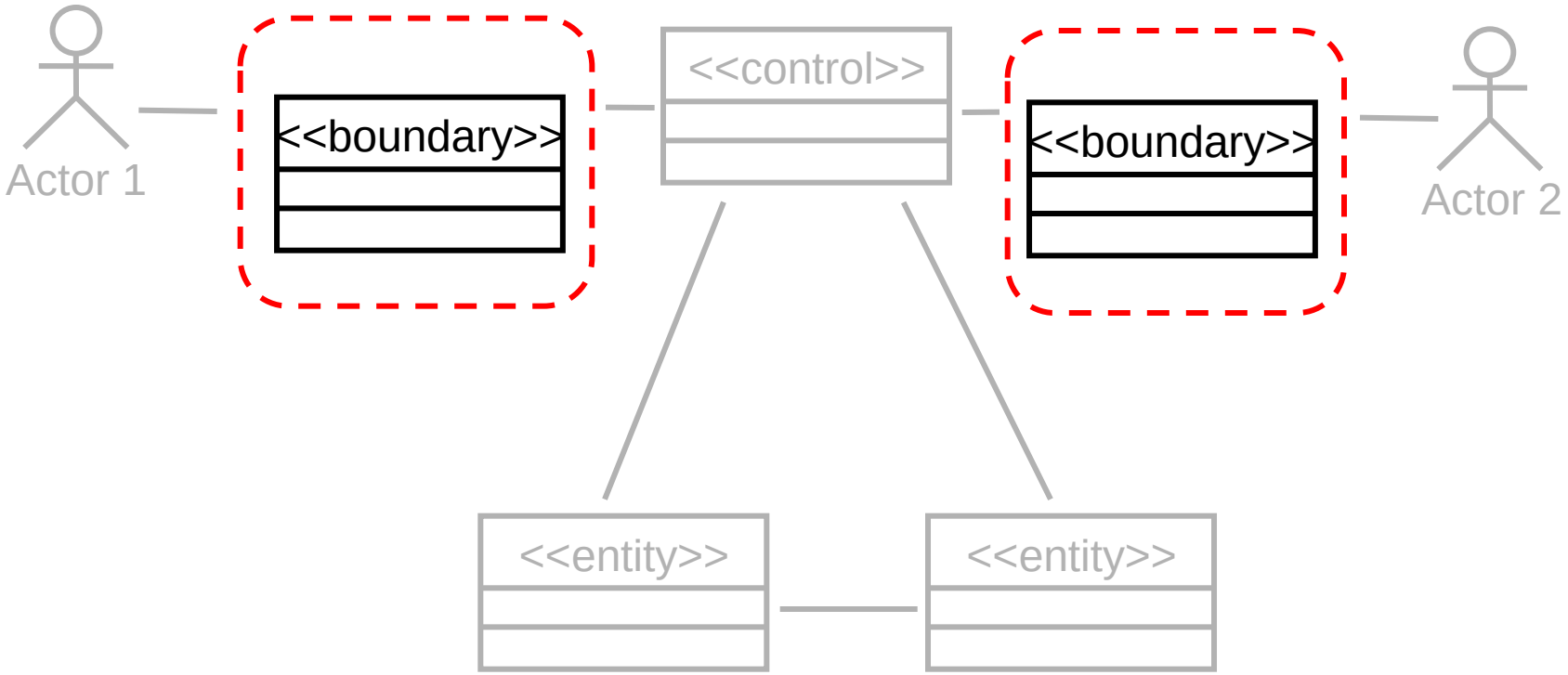






用例分析

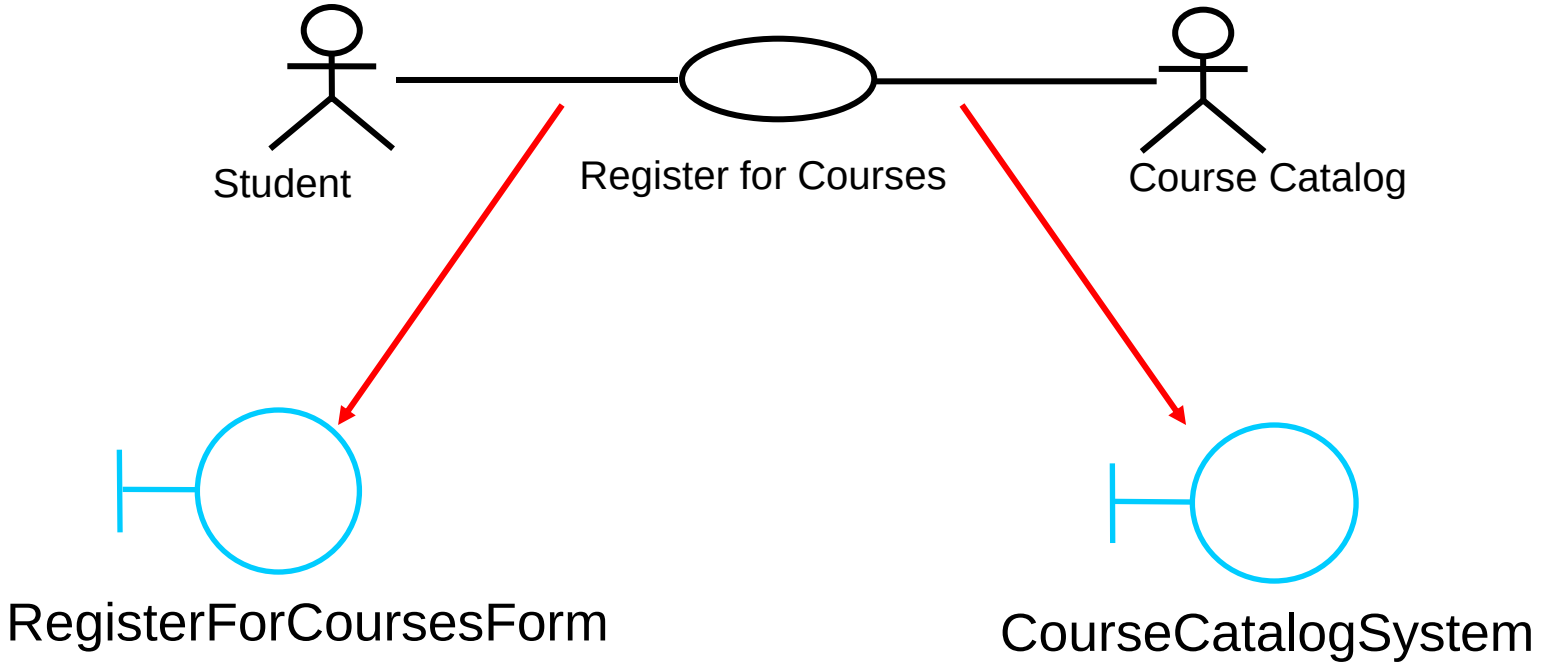
- 边界类



Model interaction between the system and its environment

用例分析

- 边界类
  - 每个参与者/用例对都对应一个边界类
  - “注册课程”用例



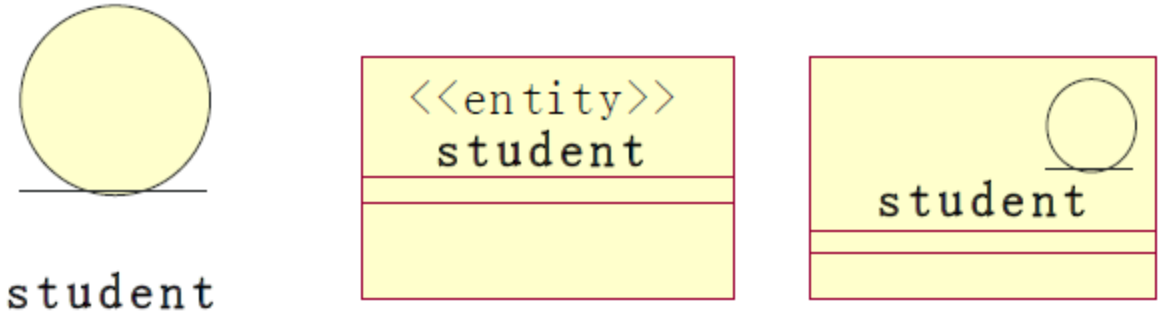
用例分析

- 边界类建模原则
  - 用户界面类
    - 关注于向用户呈现哪些信息
    - 无需关注系统UI的细节
  - 系统和设备接口类
    - 关注于哪些协议必须被规定
    - 无需关注协议的具体实现

Concentrate on the responsibilities, not the details!

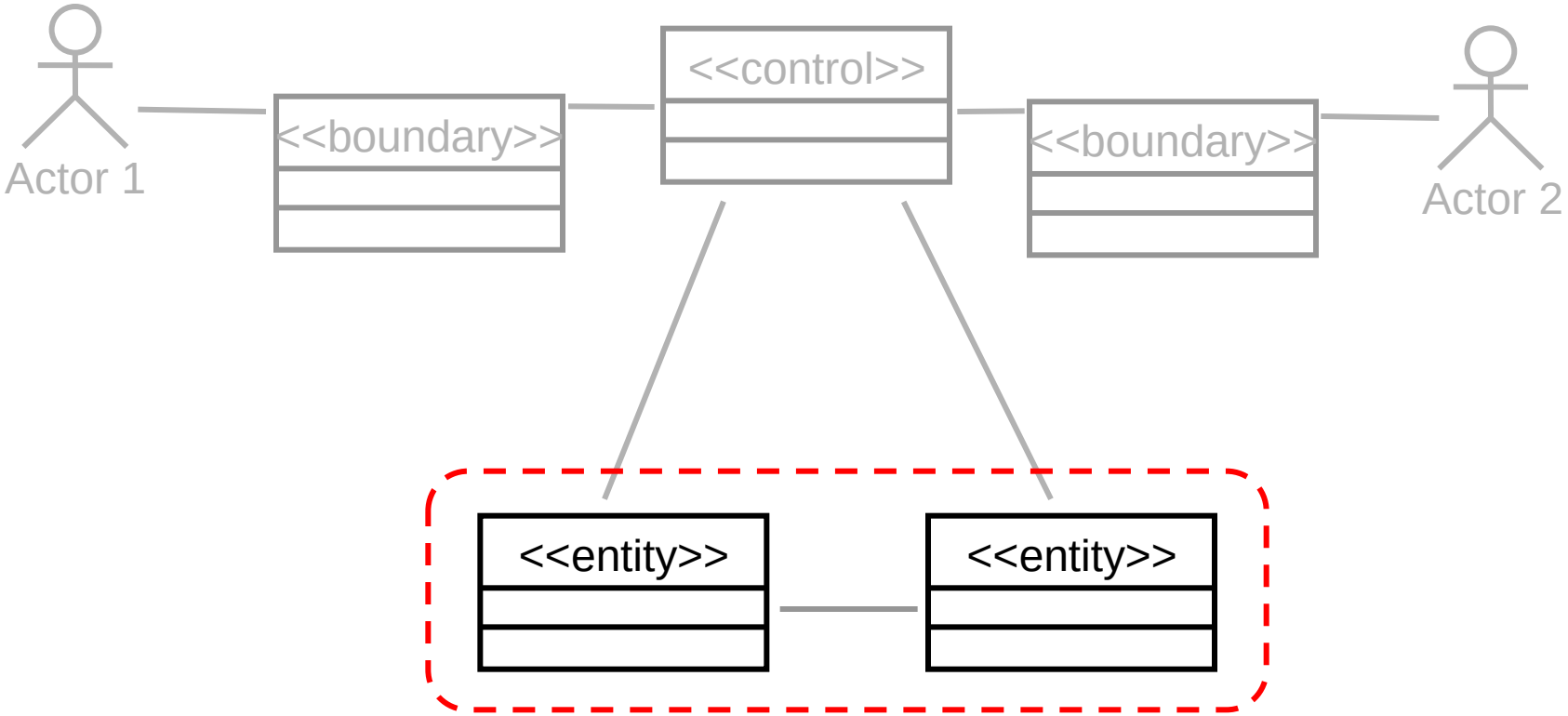
用例分析

- 实体类
  - 代表待开发系统中的关键概念
  - 提供了理解系统的另一个视角
  - 用于存储和管理系统中的信息
    - 保存永久信息
    - 最终可能映射到数据库中的表和字段



用例分析

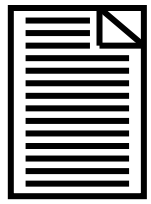
- 实体类



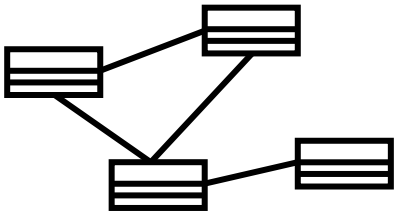
Store and manage information in the system

用例分析

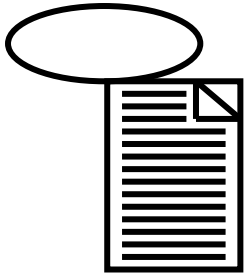
- 实体类
  - 实体类通常来自于
    - 词汇表
    - 业务领域模型
    - 用例事件流
    - 关键抽象（识别于体系结构分析阶段）



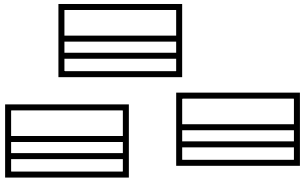
Glossary



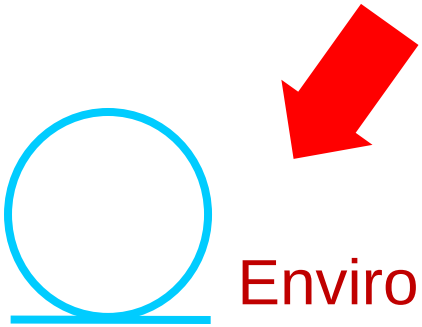
Business-Domain Model



Use Case



Architectural Analysis Abstractions



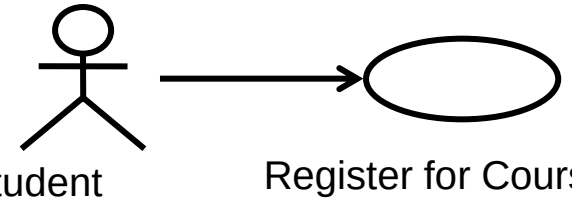
Environment independent

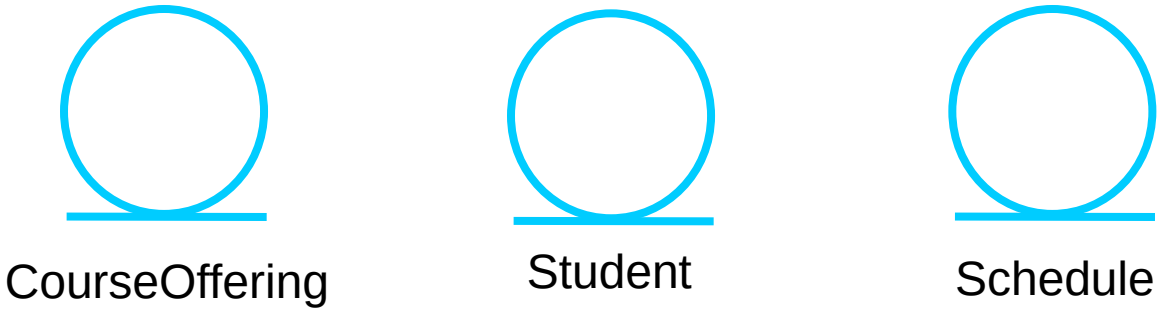
用例分析

- 实体类
  - 以用例的事件流为输入寻找实体类
  - 步骤
    - 画出事件流中的名词（短语）作为候选者
    - 移除冗余或含糊的候选者
    - 移除系统外的参与者
    - 移除实现相关的构造
    - 移除属性
    - 移除操作

用例分析

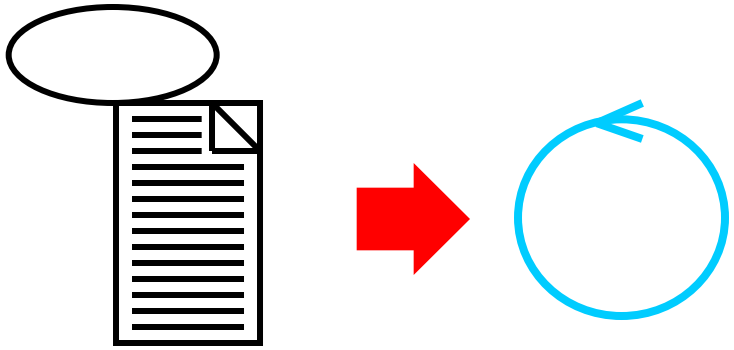
- 实体类
  - “注册课程(创建课程计划)”用例中的候选类





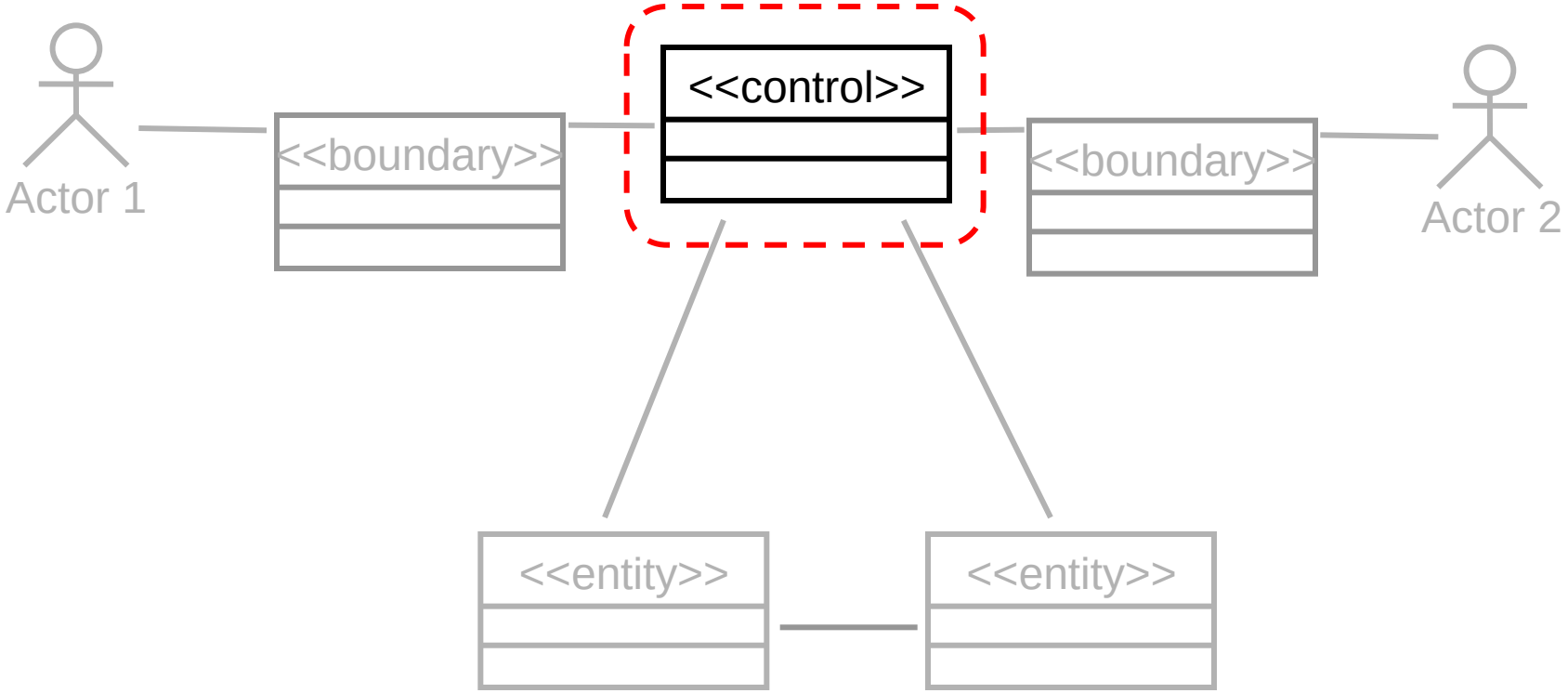
用例分析

- 控制类
  - 用例行为的协调者
    - 协调其他类工作
    - 控制总体逻辑流程
  - 通常情况下一个用例对应一个控制类



用例分析

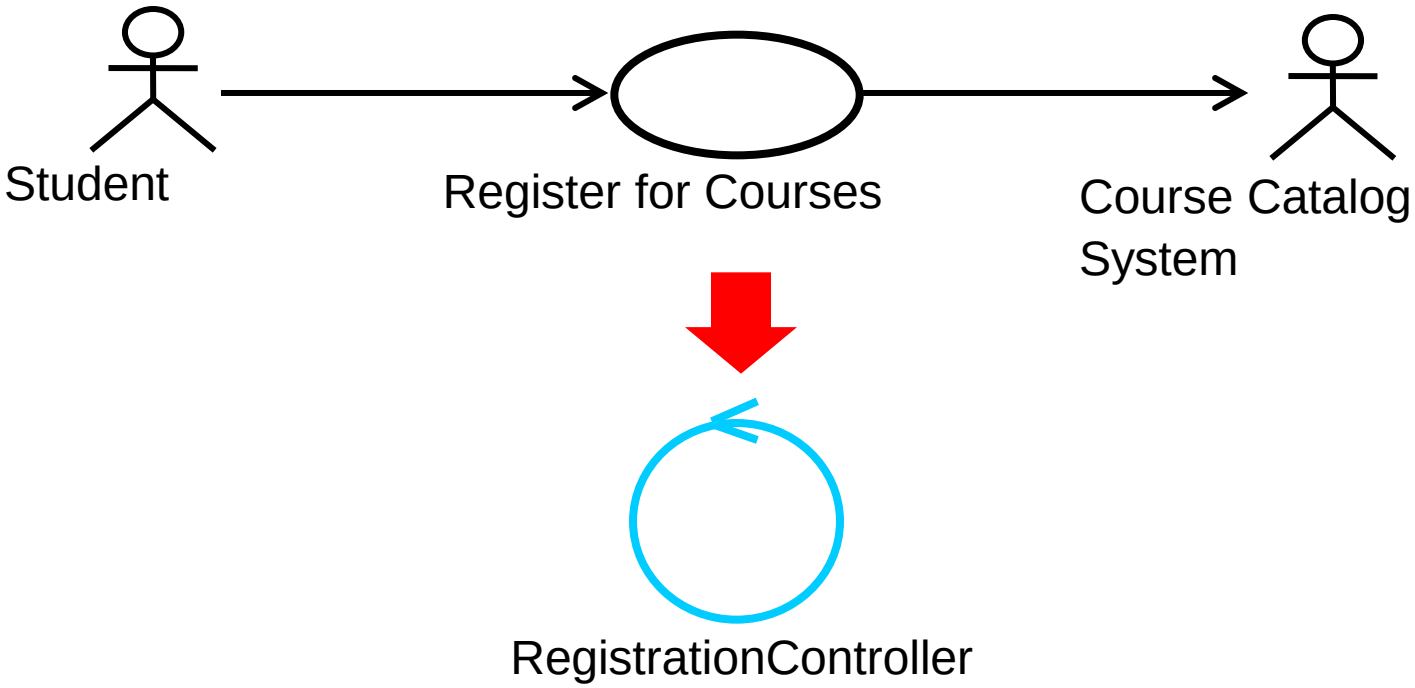
- 控制类



Use-case dependent. Environment independent.

用例分析

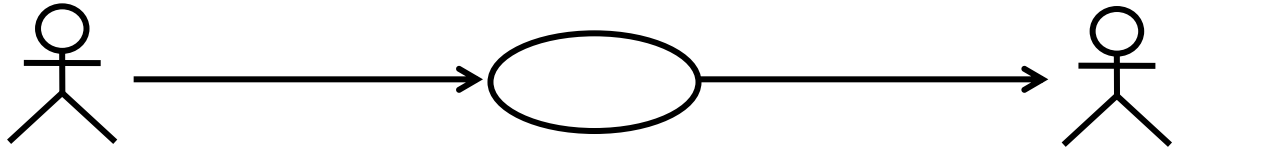
- 控制类
  - “注册课程”用例



用例分析

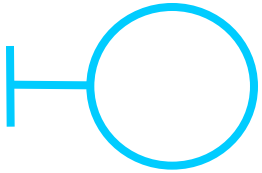
- 边界类、控制类、实体类间的关系
  - 参与者只能与边界类之间交互
  - 实体类从边界类接受信息
  - 实体类为边界类提供信息
  - 控制类操纵实体类和边界类

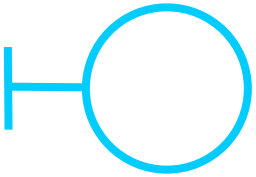
用例分析



Use-Case Model

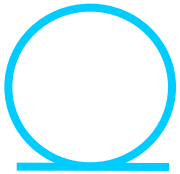
Analysis Model

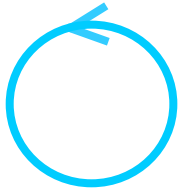

RegisterForCoursesForm


CourseCatalogSystem


Student

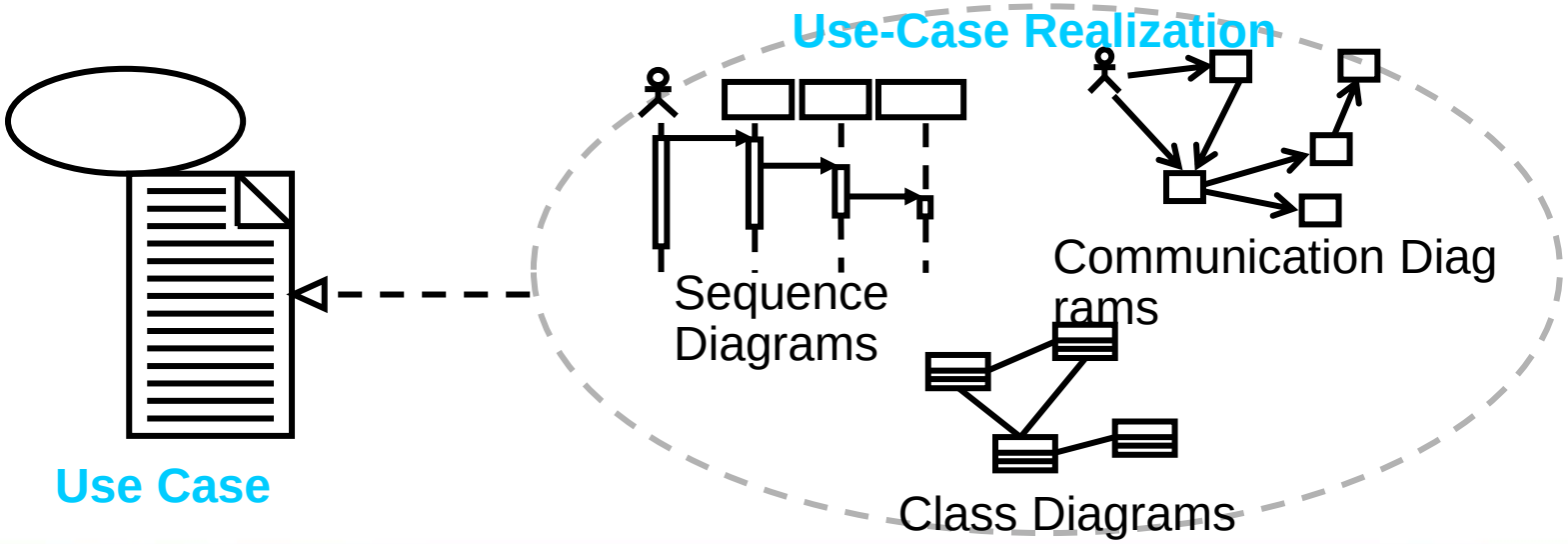

Schedule


CourseOffering


RegistrationController

用例分析

- 将用例行为分发给类
  - 对于每个用例的事件流
    - 识别分析类
    - 将用例职责分配给分析类
    - 使用交互图建模分析类间的交互



用例分析

- 将用例行为分发给类
  - 以分析类的构造型为指导
  - 边界类
    - 涉及与参与者交互的行为
  - 实体类
    - 涉及被抽象封装的数据的行为
  - 控制类
    - 特定于用例的行为
    - 特定于重要事件流某一部分的行为

用例分析

- 将用例行为分发给类

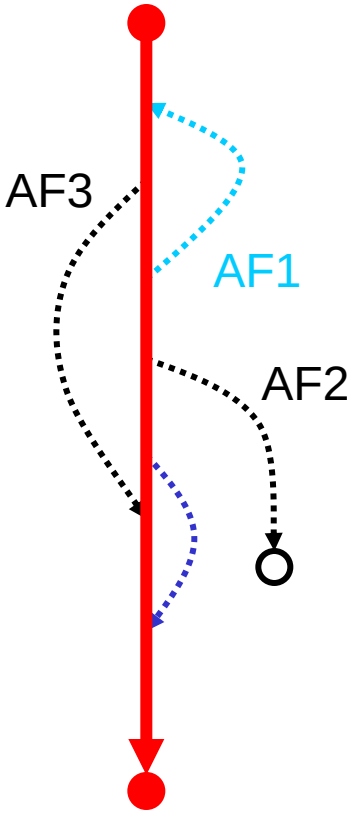
  Who has the data needed to perform the responsibility?

  - 一个类
    - 将数据相关的职责分配给它
  - 多个类
    - 分配给一个类，向另一个加一条关系
    - 创建负责此数据的新类，在向需执行职责的类添加关系
    - 分配给控制类

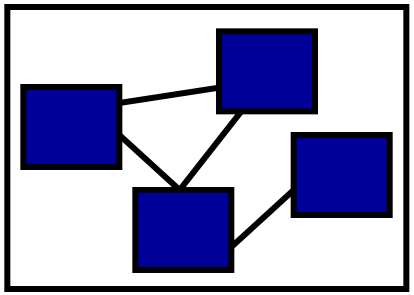
用例分析

One Interaction Diagram Is Not Good Enough !

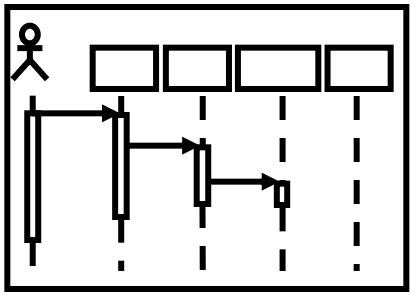
Basic Flow



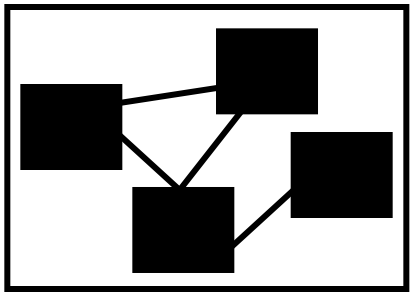
Alternate Flow 1



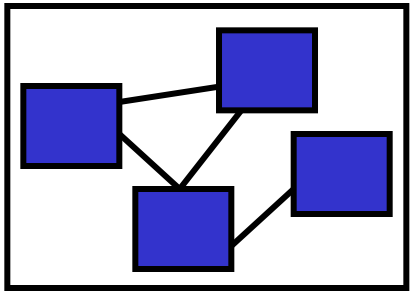
Alternate Flow 2



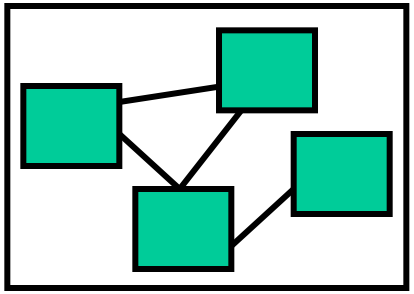
Alternate Flow 3



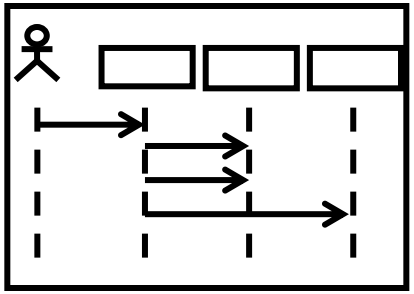
Alternate Flow 4



Alternate Flow 5



Alternate Flow n



用例分析

- 交互图
  - 顺序图VS通信图

Communication Diagrams	Sequence Diagrams
– Show relationships in addition to interactions – Better for visualizing patterns of collaboration – Better for visualizing all of the effects on a given object – Easier to use for brainstorming sessions	– Show the explicit sequence of messages – Better for visualizing overall flow – Better for real-time specifications and for complex scenarios

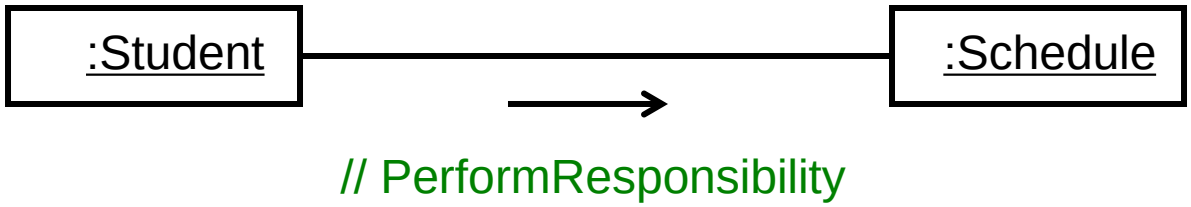
用例分析

- 对每个分析类描述职责 (Responsibilities)
  - 职责： a statement of something an object can be asked to provide
  - 分类
    - 对象能够执行的行为
    - 对象持有并能向其它对象提供的知识
  - 来源：交互图中的消息

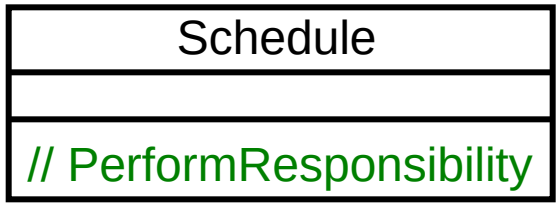
用例分析

- 对每个分析类描述职责

Interaction Diagram

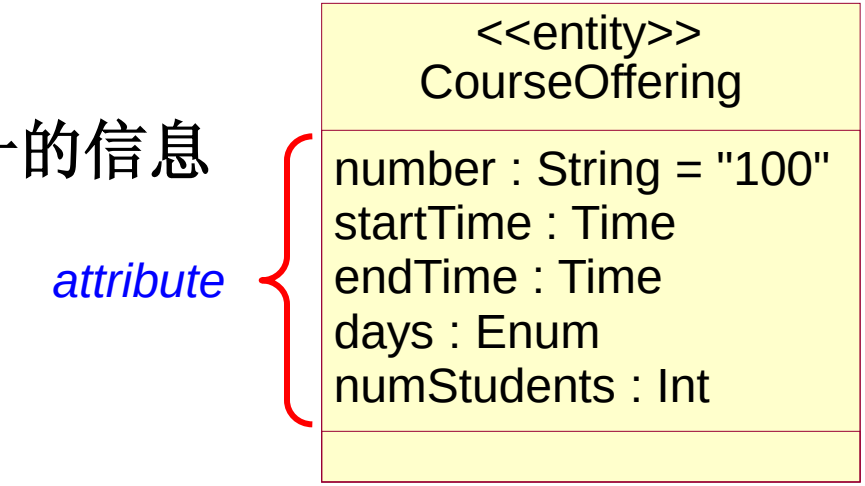


Class Diagram



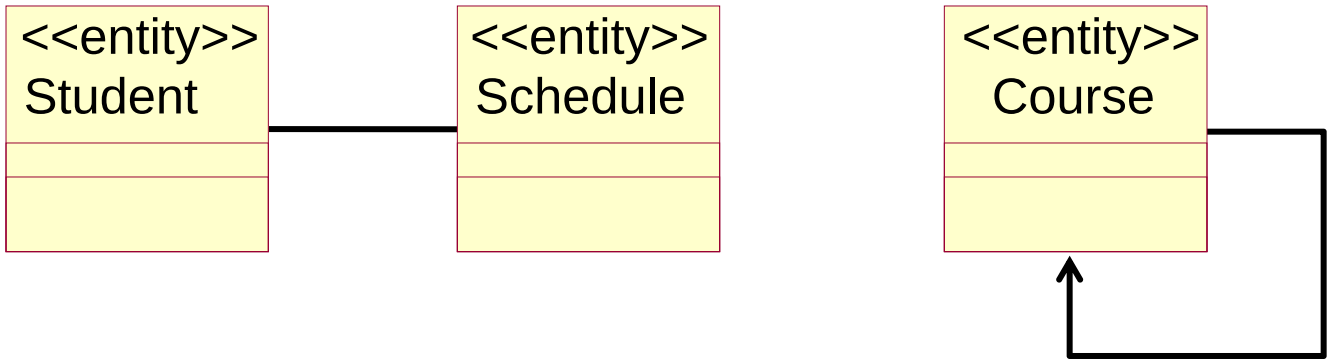
用例分析

- 描述属性 (Attributes)
  - 所识别类的性质/特性
  - 由所识别类保管的信息
  - 通常是未变成类的名词
    - 保存重要值的信息
    - 对一个对象来说唯一的信息
    - 没有行为的信息



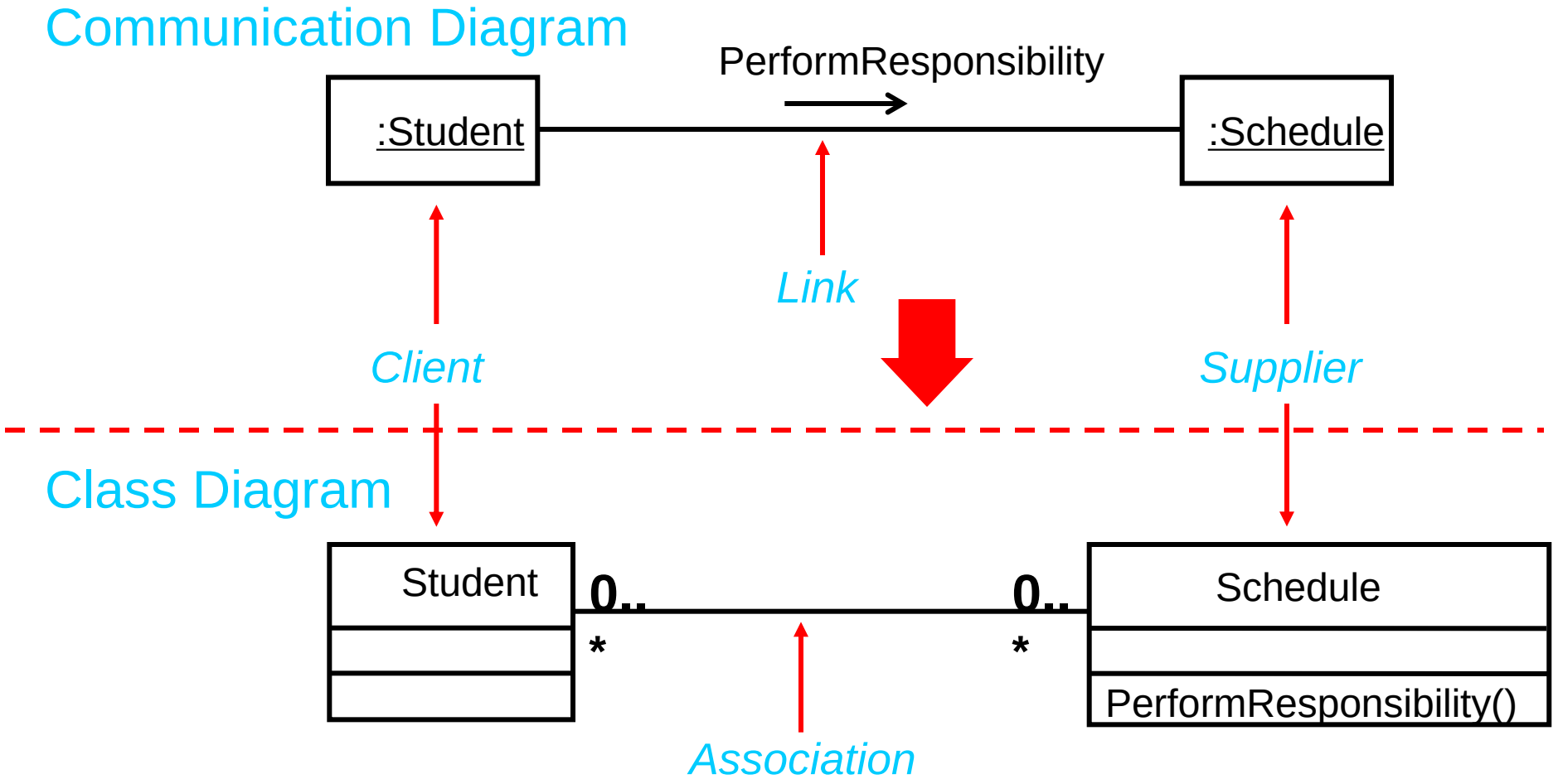
用例分析

- 描述关联 (Associations)
  - 关联
    - 指明两个或多个类元实例间存在联系的语义关系
    - 不同类的对象间的结构关系



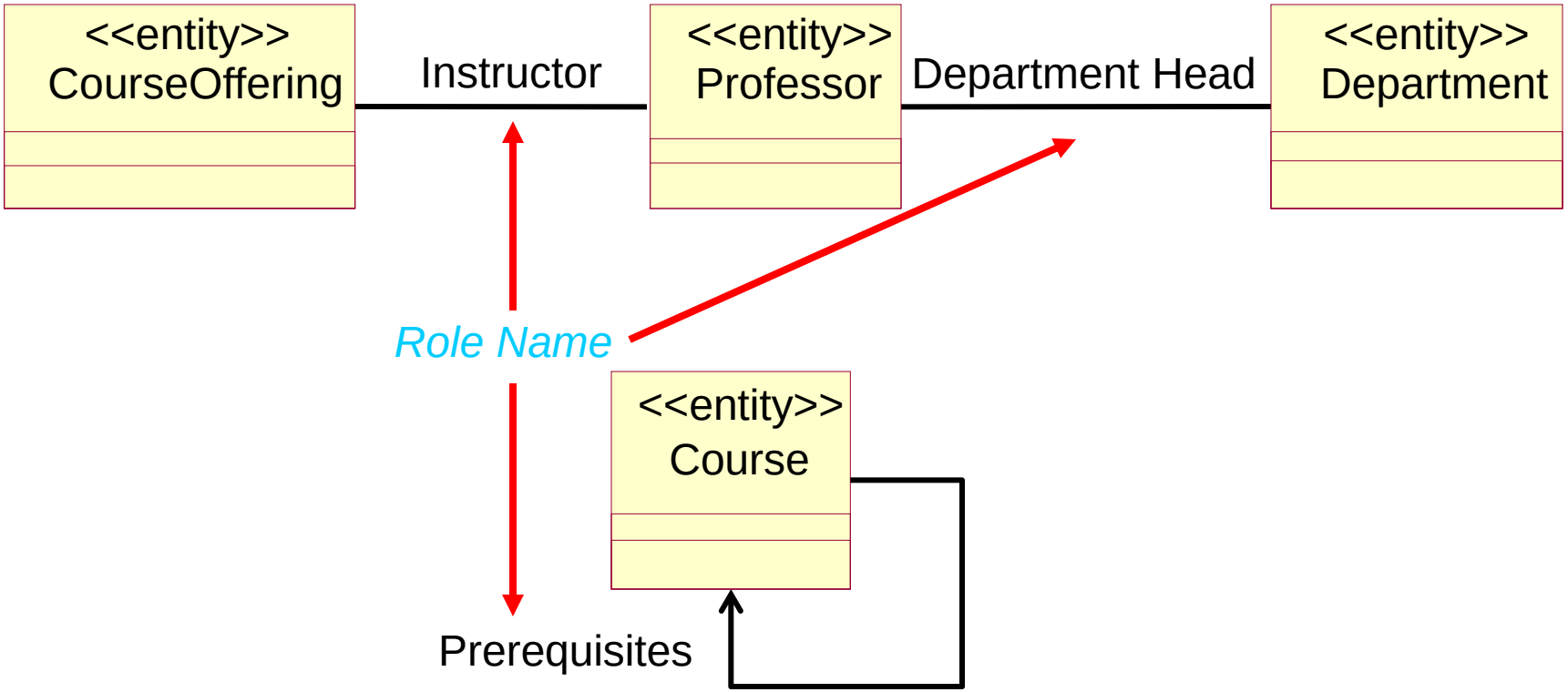
用例分析

- 寻找关联


Communication Diagram
PerformResponsibility
:Student
:Schedule
Link
Client
Supplier
Class Diagram
Student
0..
0..
*
*
Schedule
PerformResponsibility()
Association

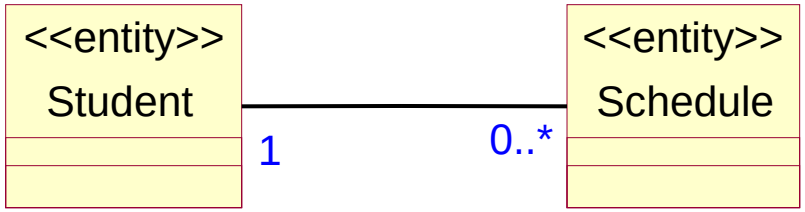
用例分析

- 描述关联 (Associations)
  - 角色：类中关联中扮演的“脸孔”



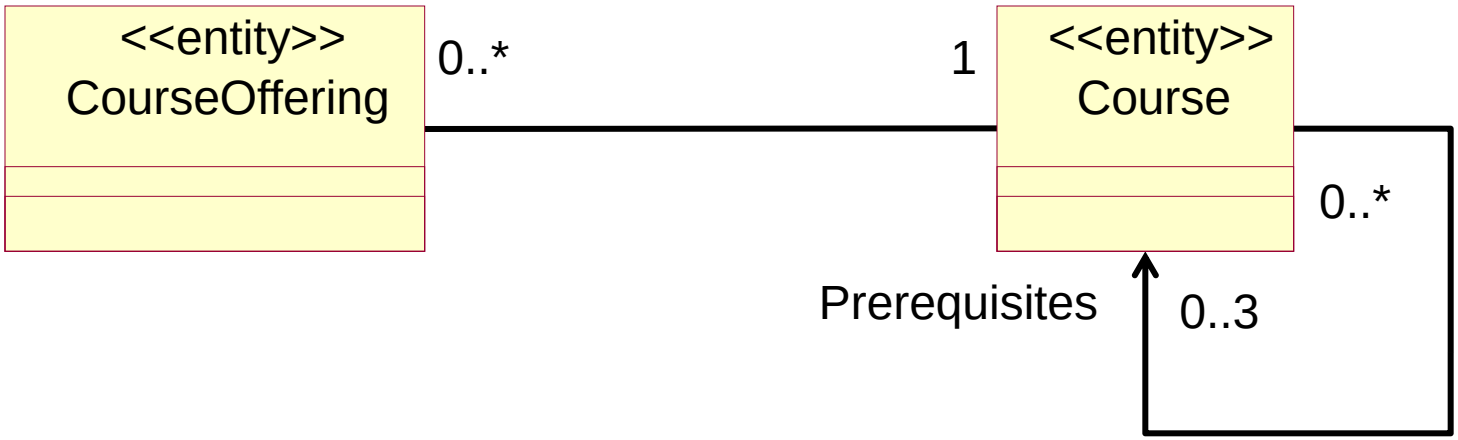
用例分析

- 描述关联 (Associations)
  - 多重性
    - 关联是强制的还是可选的？
    - 一个实例能够被链接到另一个实例的最大/小数目



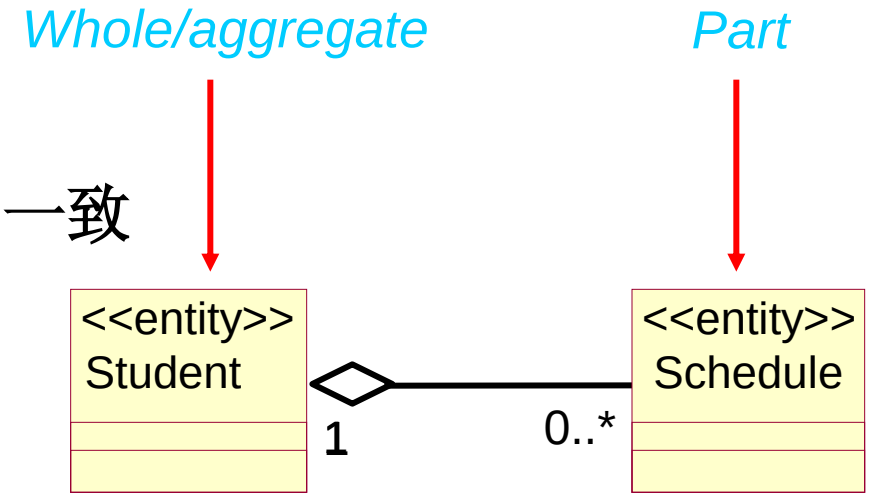
What Does Multiplicity Mean?

- Multiplicity answers two questions:
  - Is the association mandatory or optional?
  - What is the minimum and maximum number of instances tha t can be linked to one instance?



用例分析

- 描述关联 (Associations)
  - 聚集 (Aggregation)
    - 建模部分与整体的关系
    - 部分与整体的生命周期不一致
  - 组合 (Composition)
    - 加强的拥有关系
    - 部分与整体的生命周期一致



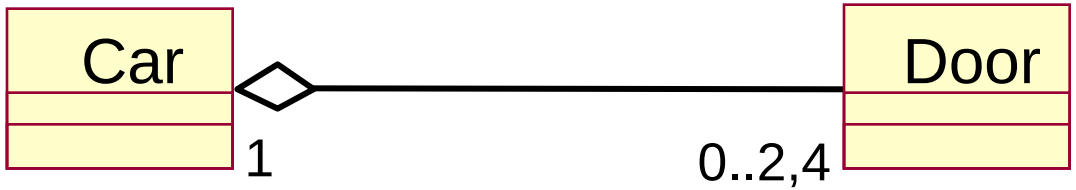
用例分析

- 描述关联 (Associations)
  - 什么时候用到聚集？
    - 两个类的实例间存在紧密的“整体-部分”关系时，使用聚集
    - 若两个对象间存在链接，且经常被独立地考虑，采用关联

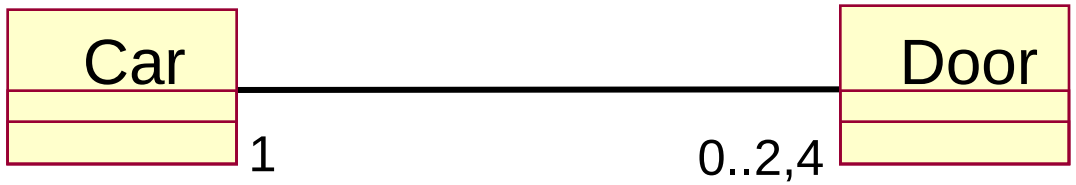
When in doubt, use association.

Association or Aggregation?

- If two objects are tightly bound by a whole-part rel ationship
  - The relationship is an aggregation.



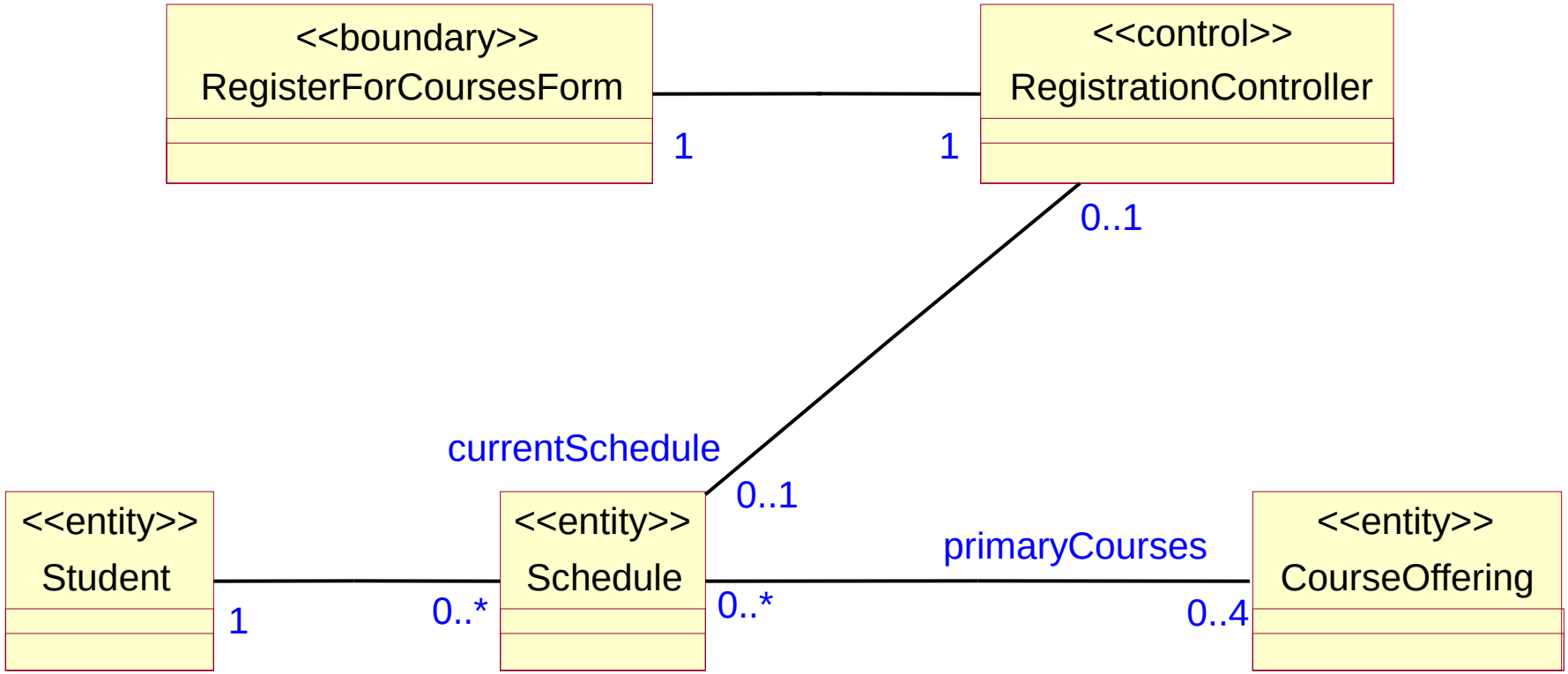
- If two objects are usually considered as independe nt, although they are often linked
  - The relationship is an association.



When in doubt, use association.

用例分析

- 描述关联

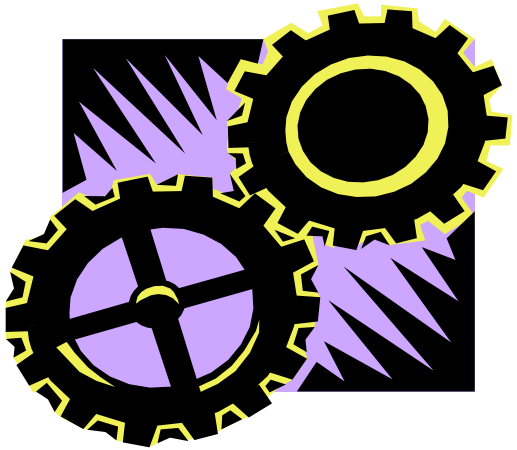

<<boundary>>
RegisterForCoursesForm
<<control>>
RegistrationController
1
1
0..1
currentSchedule
0..1
<<entity>>
Student
<<entity>>
Schedule
<<entity>>
CourseOffering
primaryCourses
1
0..*
0..*
0..4

Review: Why Use Analysis Mechanisms?

Analysis mechanisms are used during analysis to reduce the complexity of analysis and to improve its consistency by providing designers with a shorthand representation for complex behavior.

Describing Analysis Mechanisms

- **Collect all analysis mechanisms in a list**
- **Draw a map of the client classes to the analysis mechan isms**
- **Identify characteristics of the analysis mechanisms**



Example: Describing Analysis Mechanisms

- **Analysis class to analysis mechanism map**

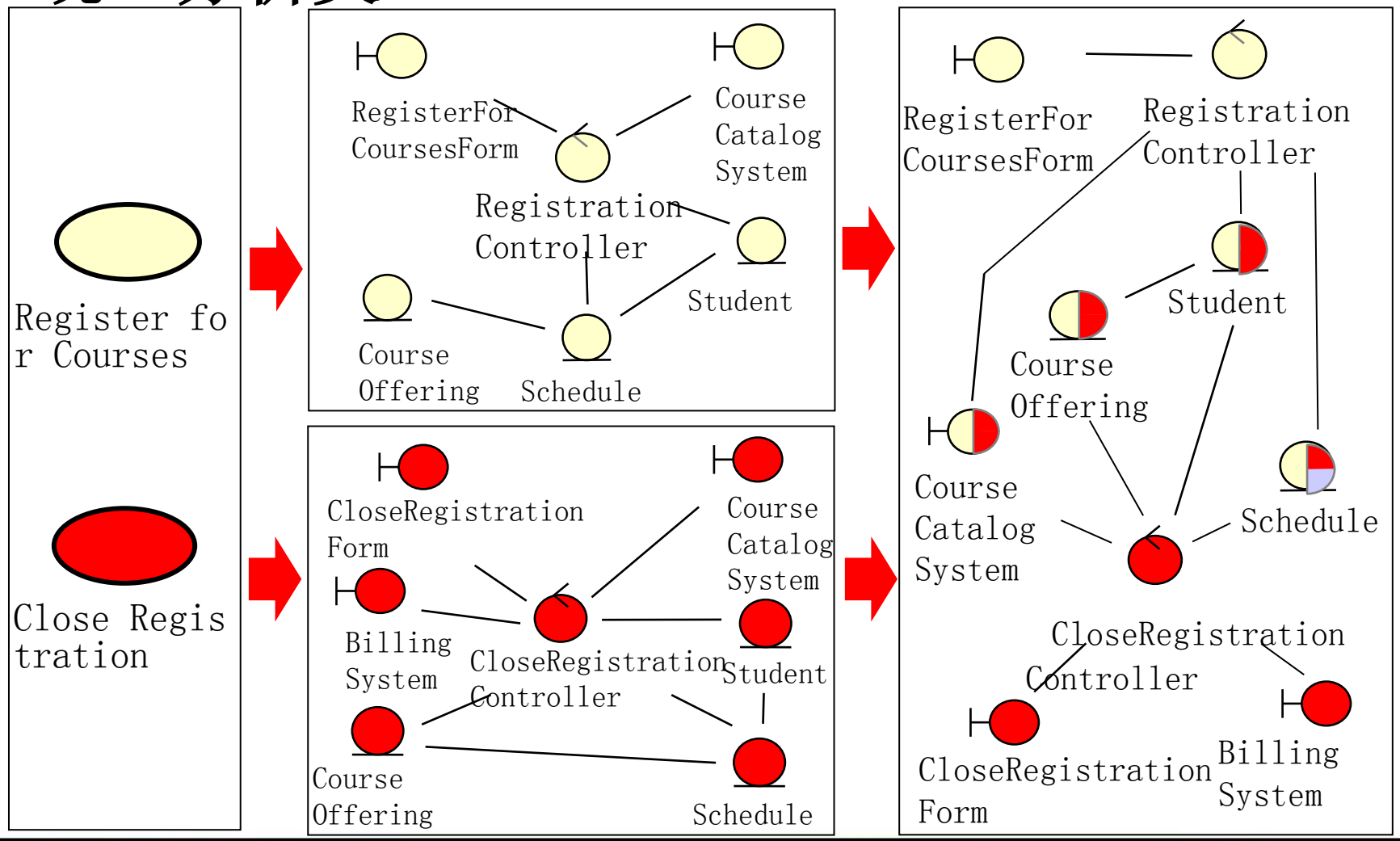
Analysis Class	Analysis Mechanism(s)
Student	Persistency, Security
Schedule	Persistency, Security
CourseOffering	Persistency, Legacy Interface
Course	Persistency, Legacy Interface
RegistrationController	Distribution

Example: Describing Analysis Mechanisms (continued)

- **Analysis mechanism characteristics**
- **Persistency for Schedule class:**
  - **Granularity: 1 to 10 Kbytes per product**
  - **Volume: up to 2,000 schedules**
  - **Access frequency**
    - **Create: 500 per day**
    - **Read: 2,000 access per hour**
    - **Update: 1,000 per day**
    - **Delete: 50 per day**
  - **Other characteristics**

用例分析

- 统一分析类


Register for Courses
Close Registration
RegisterFor CoursesForm
Course Catalog System
Registration Controller
Student
Course Offering
Schedule
CloseRegistration Form
Billing System
CloseRegistration Controller
Course Catalog System
Student
Course Offering
Schedule
RegisterFor CoursesForm
Registration Controller
Student
Course Offering
Course Catalog System
Schedule
CloseRegistration Controller
CloseRegistration Form
Billing System

小结

- 重点
  - 用例分析活动
  - 用例实现（分析）
  - 三种带构造性的分析类
  - 由用例模型生成分析类及交互图