Lecture 9. 系统分析 (System analysis)
(用例分析 Use Case Analysis) I

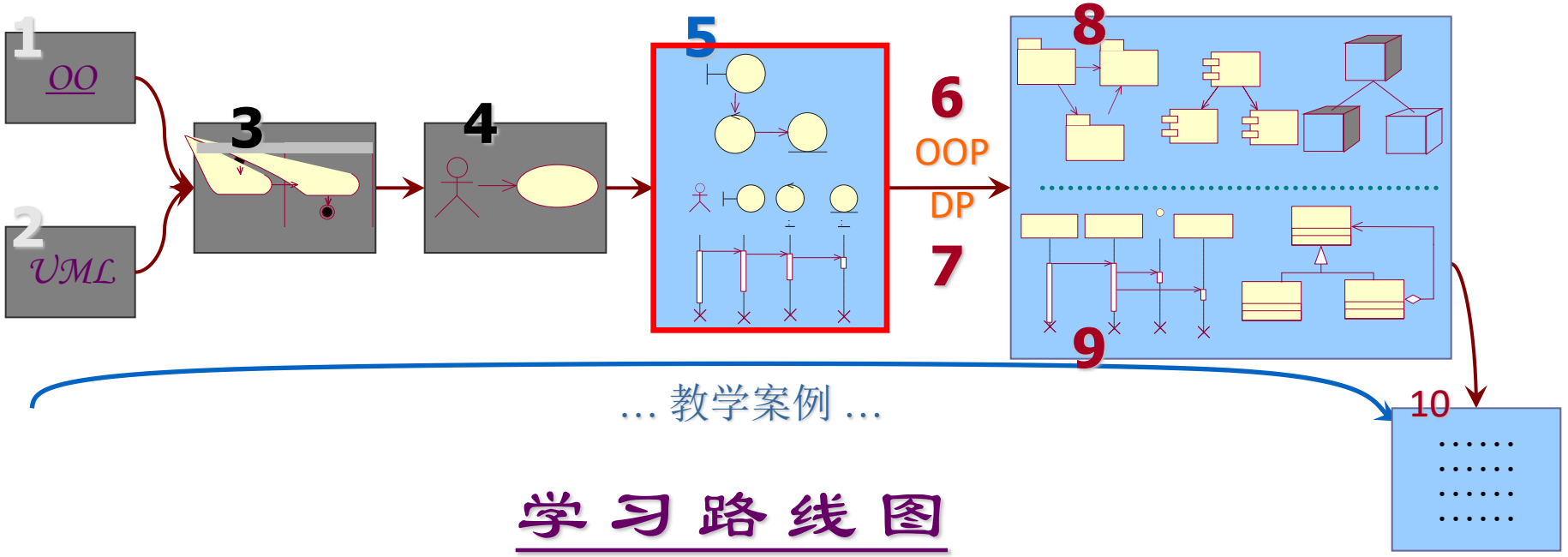
Object Oriented Modeling Technology

面向对象建模技术

Professor: Yushan Sun

Fall 2022

学习路线图

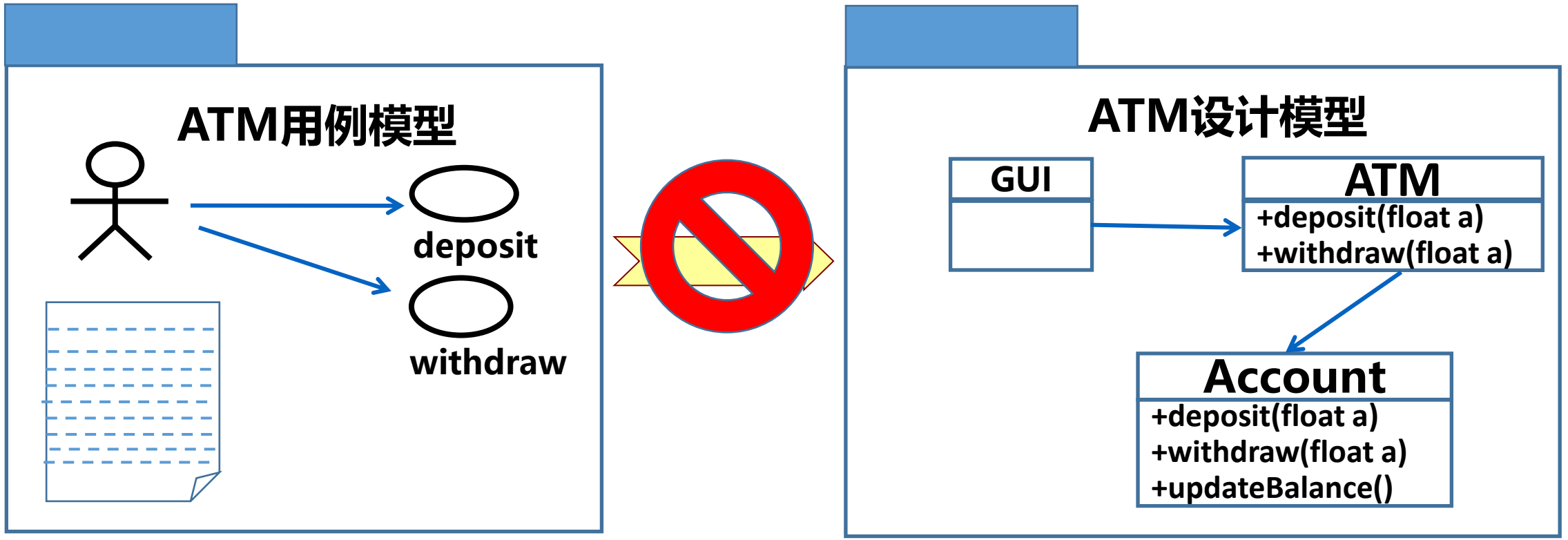


内容安排

- 理解分析
- 从用例开始分析
- 架构分析
- 构造用例实现
  - 完善用例文档
  - 从用例行为中别分析类

理解分析

美好的愿望：

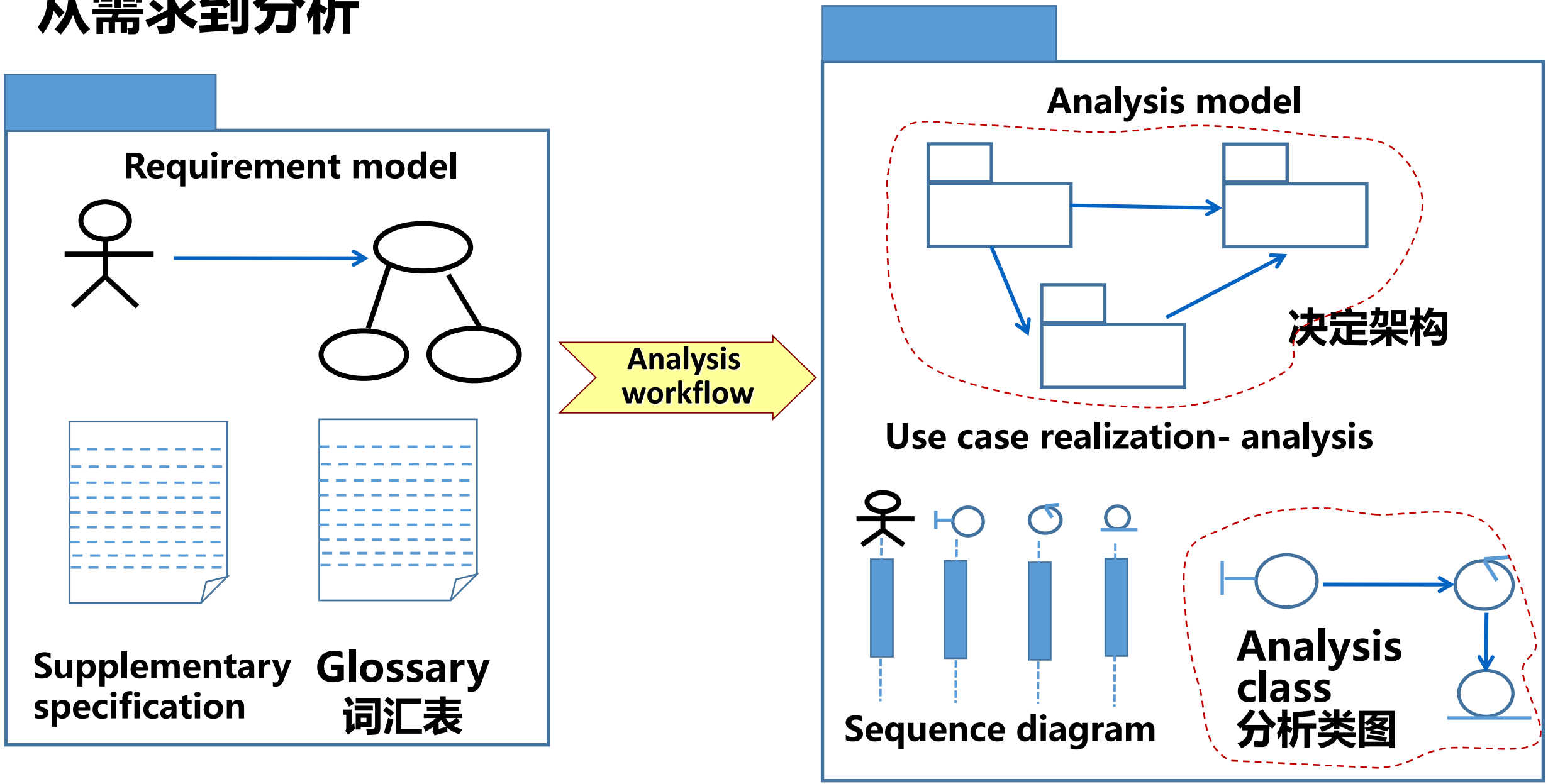


- 希望：是否能从用例模型开始，进行设计，得到类图，然后编写程序，然后项目结束。
- 回答：这样是行不通的，必须有分析阶段。

系统分析（用例分析）

- **分析：为了满足需求模型中所描述的功能，系统内部应该有什么样的业务核心机制(需要几个对象互相协作呢？)**
- **分析的目标是开发一系列模型，以描述软件核心成分，从而满足客户定义的需求：称为分析模型**
  - **包括两个层次：架构分析和用例分析**
  - **包括两类模型：静态结构(包图、类图)和动态交互(顺序图、通信图)**
  - **模型元素按照架构来组织，各类视图按照用例实现(协作)来组织**

从需求到分析


Requirement model
Supplementary specification
Glossary
词汇表
Analysis workflow
Analysis model
决定架构
Use case realization- analysis
Sequence diagram
Analysis class
分析类图

分析与需求的关系

- 需求模型以用户的角度描述系统所要实现的目标，而分析模型表示在系统内部应该提供哪些核心业务元素和关系，以实现需求模型提出的目标（例如，要有哪些对象(类)的配合才能完成需求中所提出的功能）
- 用例模型被“翻译”成分析模型(用例描述➔顺序图，用例描述中的重要名词被翻译成类名及其属性)
- 与需求一样，分析还是在问题域中(分析类反映业务情况)
- 分析与需求捕获在很大程度上重叠，常常相辅相成，为了澄清和找出任何遗漏或歪曲的需求，常常需要修改用例文档.

- **一些具体的分析原则：**
  - **分析模型使用业务语言**
  - **分析类和关系等应该是业务中明确存在的**
  - **分析是对需求模型的重新表述，是以理想化的方式(例如，使用顺序图来表达系统中某些对象的交互以实现用例模型中的功能)来实现用例行为，不考虑技术实现**
  - **分析侧重于系统主要部分，关注核心的业务场景；**
  - **对支撑性行为、非功能需求等不做深入的分析**

Return

从用例开始分析

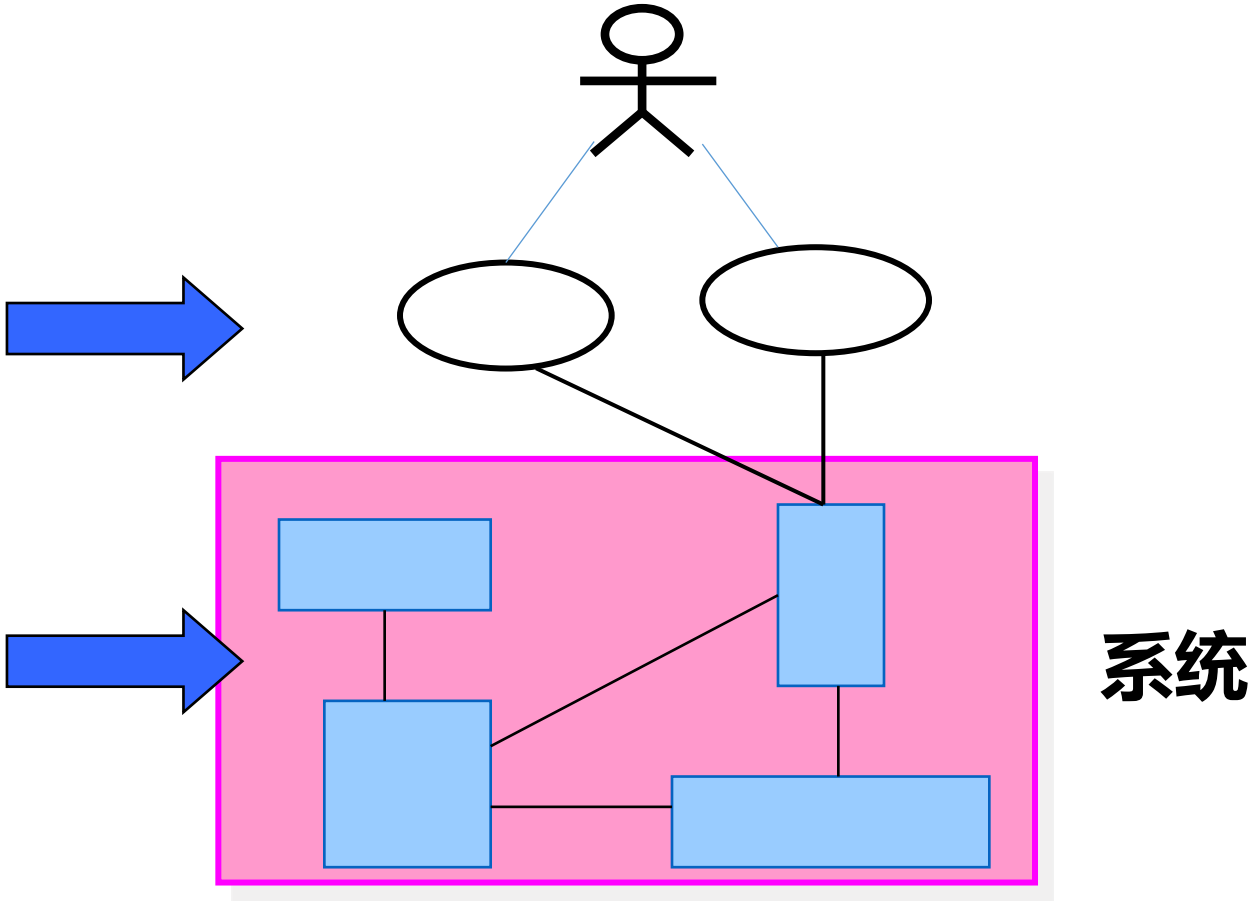
如何开始分析?

从 用 例 开 始!

分析模型与用例模型

用例模型：表达系统功能。描述系统外部功能；描述参与者与系统进行交互的情况。

分析模型：表达内部机理。表达系统内部有多少个对象互相协作，以完成用例模型所涉及的功能。



系统

从用例开始定义迭代

- 迭代开发是现代软件开发的主流，而迭代的基础就是用例
- 从用例开始分析基本思路，用例分级：根据
  - 风险、
  - 重要性以及
  - 项目组

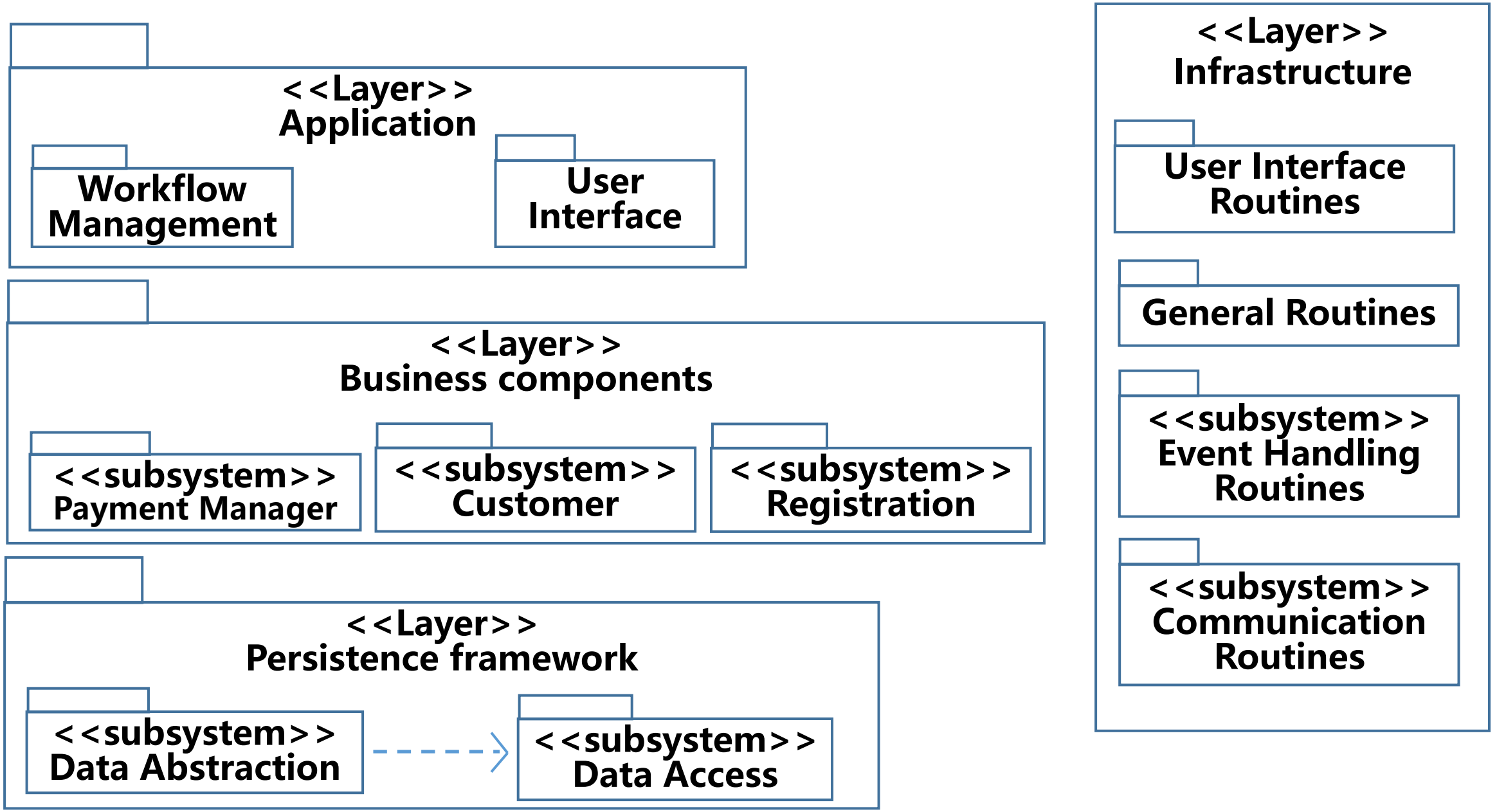
  的能力确定用例以及用例相关路径的优先级

利用早期迭代建立软件架构

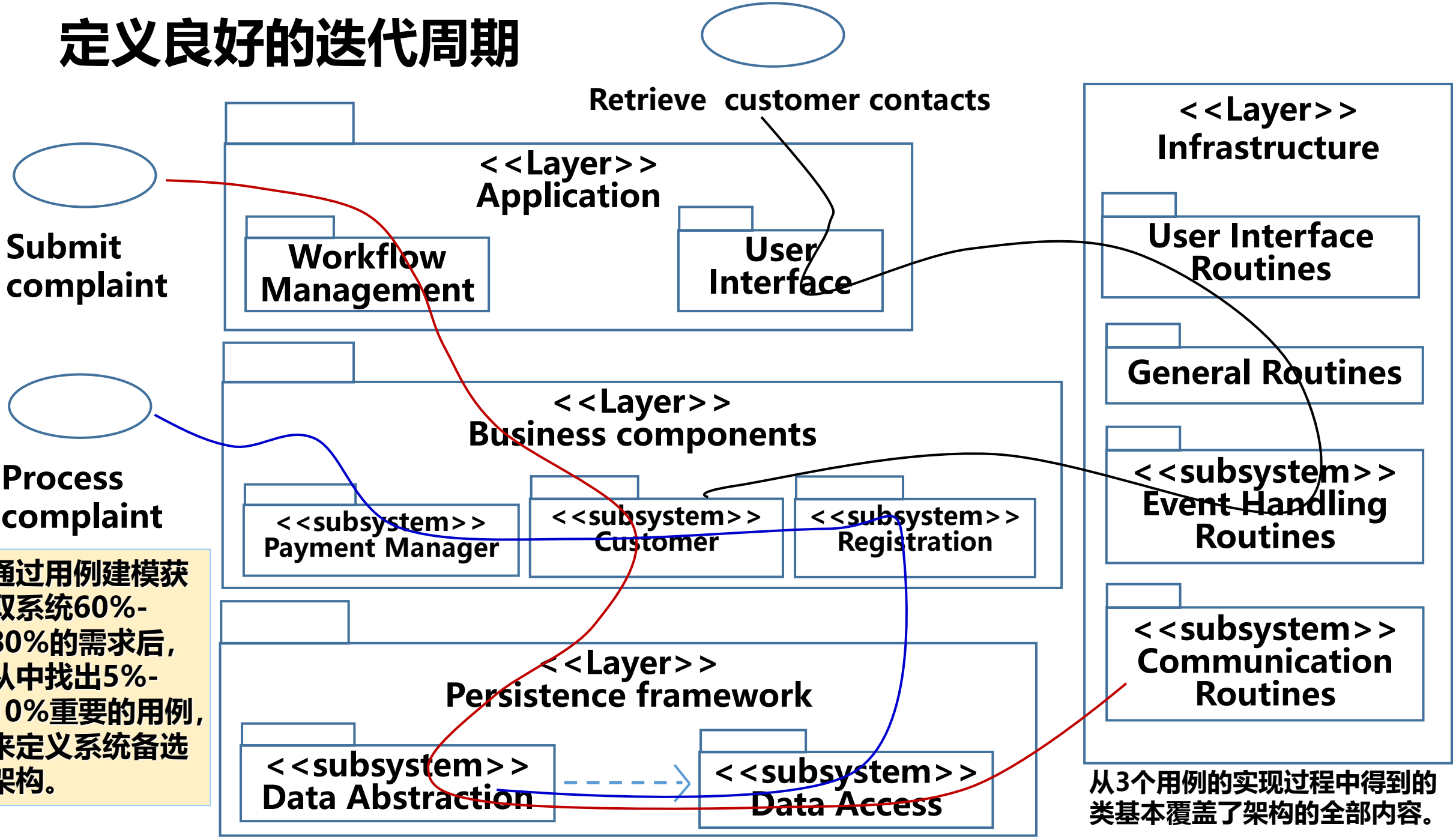
- 早期的迭代非常重要，其目的是建立和证明软件的核心架构；该架构将决定系统是否能够顺利地推进后续的迭代开发。

- 早期迭代关注的重点（架构）
  - 核心业务的主要部分 (将产生分析类图)
  - 系统架构有重要影响 （将产生架构图，即软件体系结构图）
  - 影响系统性能等其他关键非功能需求的部分
  - 存在高风险的部分，如新技术、新产品部分

Jacobson的例子：选取几个关键的用例来定义系统备选架构



定义良好的迭代周期



通过用例建模获取系统60%-80%的需求后，从中找出5%-10%重要的用例，来定义系统备选架构。

从3个用例的实现过程中得到的类基本覆盖了架构的全部内容。

分析阶段的用例—用例实现

- 用例实现是分析/设计模型中系统用例的表达式
  - 在所使用的建模工具中(如Rational Rose, StarUML)，使用构造型<<use-case realization>>
  - 用例实现描述了对象间的协作以完成用例目标
  - 用例实现将用例模型中的用例和设计(分析)模型中的类和关系连接在一起
  - 用例实现说明了每个用例必须用那些类来实现
- 用例实现提供了从分析和设计到需求的可跟踪性

创建用例实现

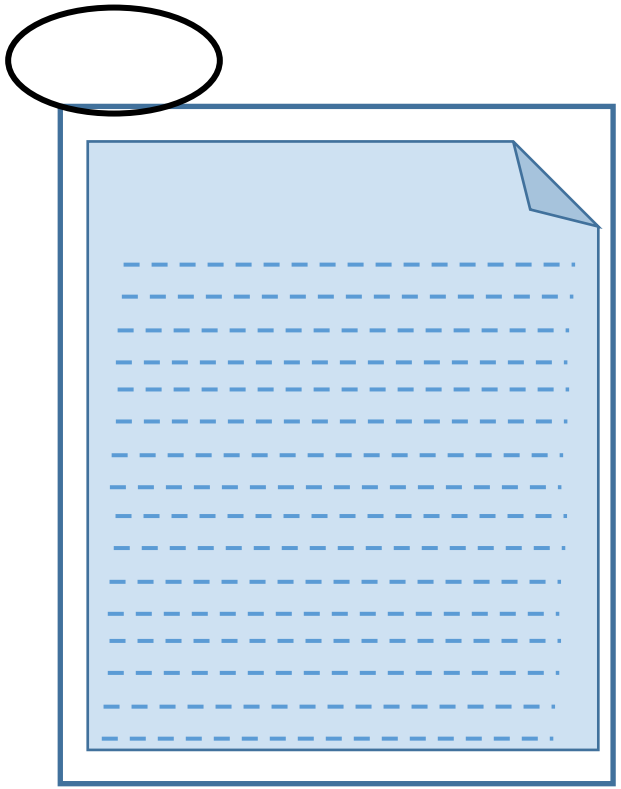
Use case model

Analysis (design) model

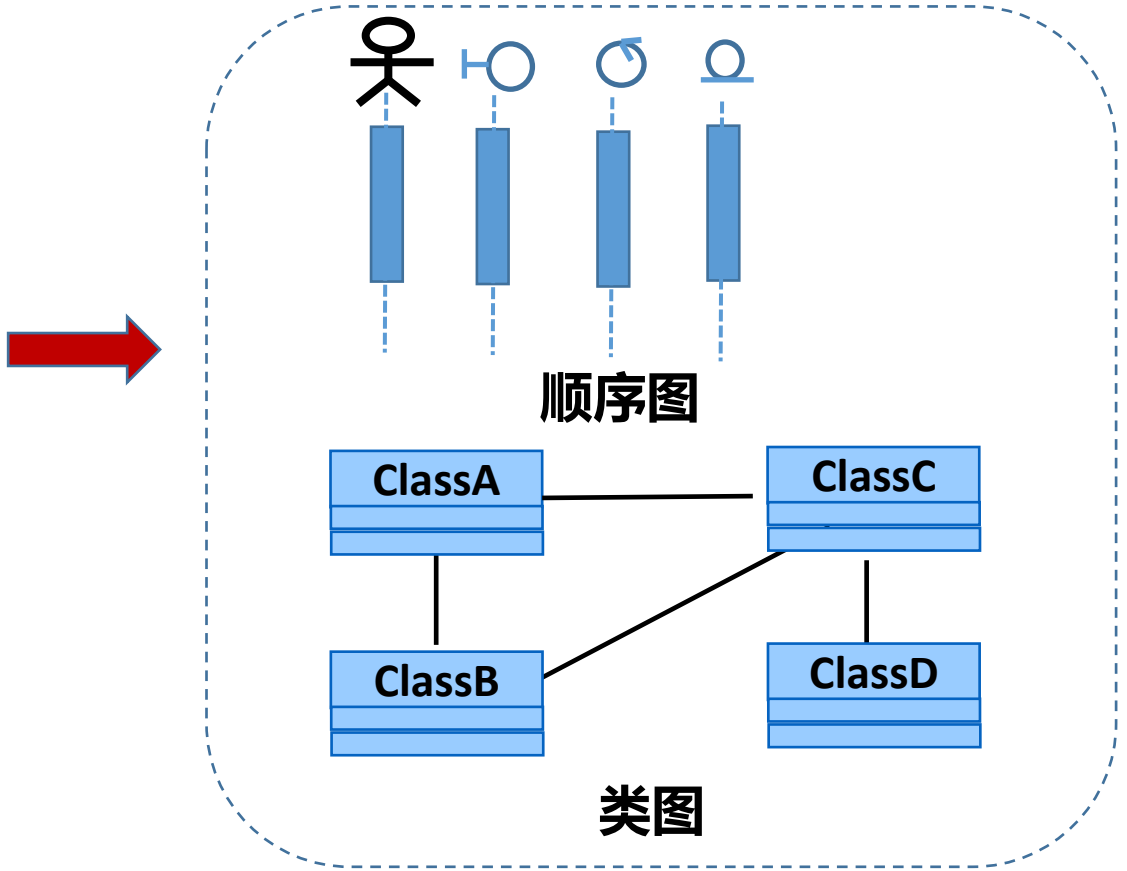
Use case

Use case realization

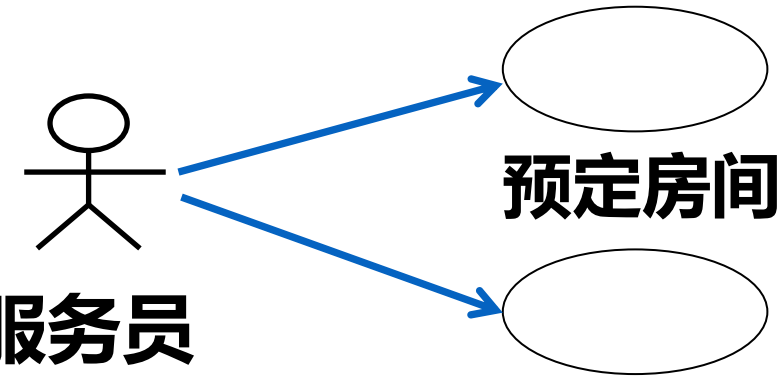
用例实现

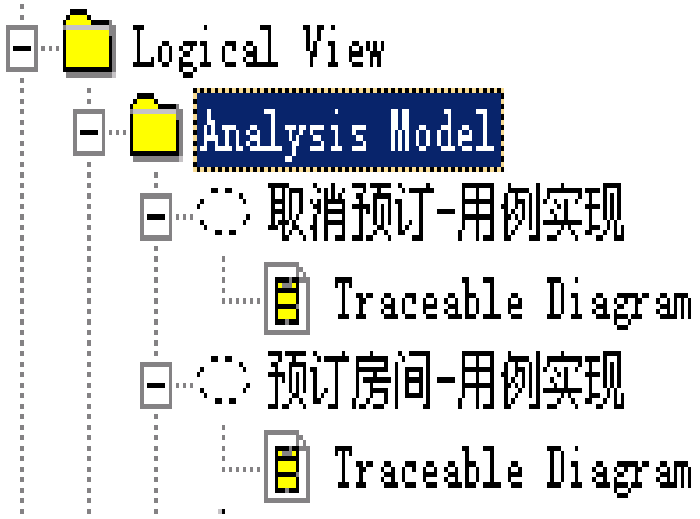


用例描述



实例-旅店预订系统迭代1和用例实现





考虑以上两个用例的实现

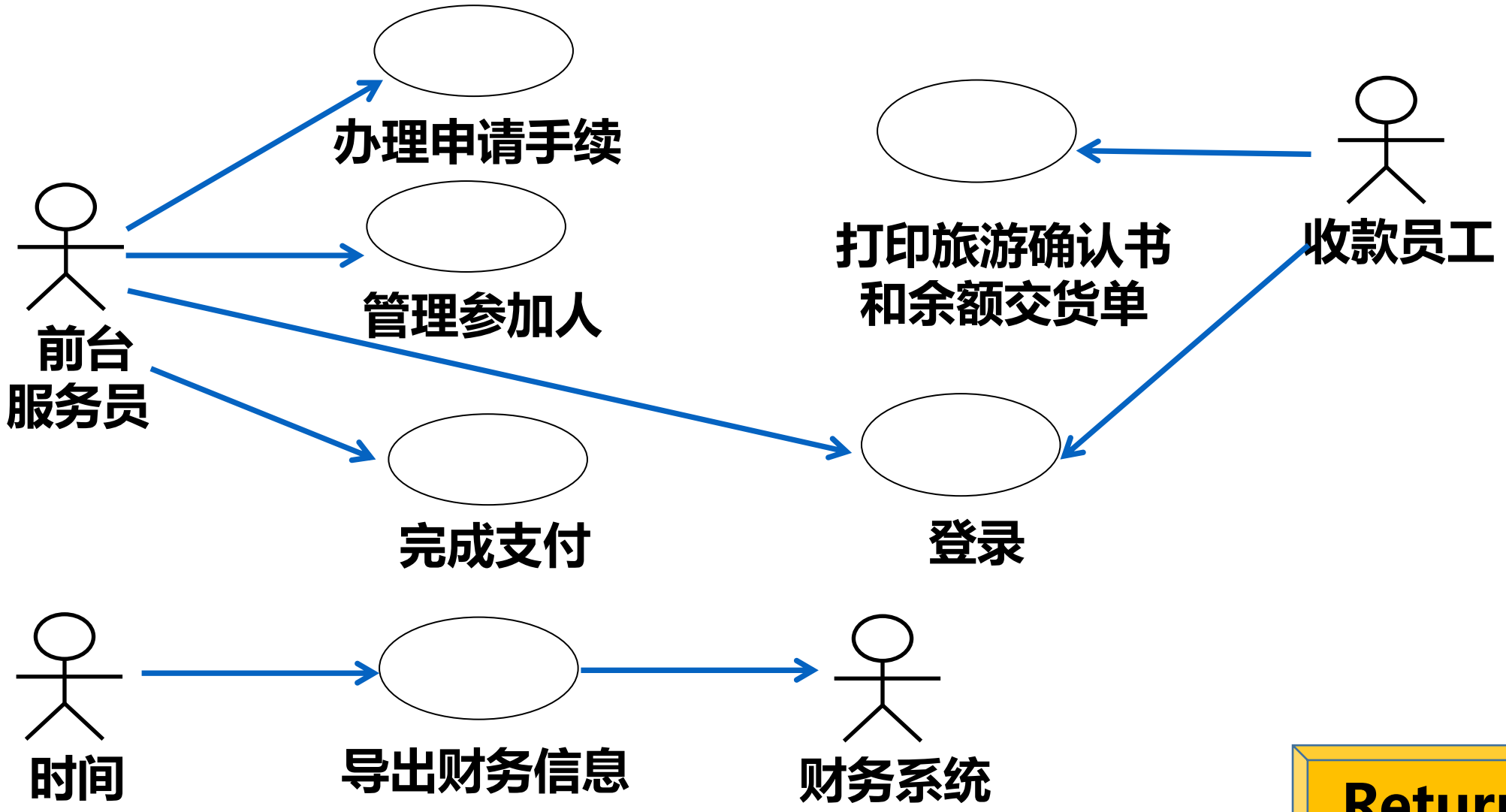
预定房间
From use-case model

预定房间-用例实现

取消预定
From use-case model

取消预订-用例实现

实例-旅游申请系统迭代1



Return

架构分析

架构分析

- 架构分析的过程就是定义系统高层组织结构和核心架构机制的过程
  1. 定义系统的备选架构来描述系统的高层组织结构—备选架构(选取层次结构，Client-server, MVC等)
  2. 提取系统的关键抽象以揭示系统必须能够处理的核心概念—关键抽象(领域模型图)
  3. 创建用例实现来启动用例分析—用例实现

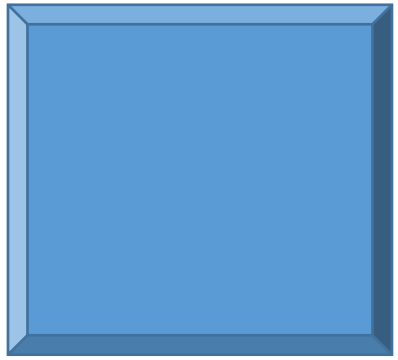
1.定义备选架构

- 定义备选架构
  - 架构的初始草图
  - 初步定义系统的分层与组织
  - 初步定义一组在架构方面具有重要意义的元素，以作为分析的基础
  - 初步定义一组分析机制
  - 定义要在当前迭代中处理的用例实现

备选架构模式

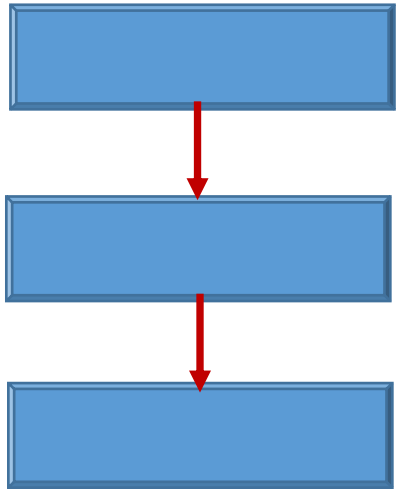
- 架构模式(软件体系结构，架构)表示了对软件系统的一个基础结构组织形式。它提供了一套预定义子系统，详细说明它们的职责，并且包括组织它们之间的规则和指南。
- 经典的架构有
  - 分层架构(层次架构)
  - 模型-视图-控制器(M-V-C)
  - 客户端-服务器架构
  - …

分层架构简介



早期将软件设计为一块

分层



三层层次结构

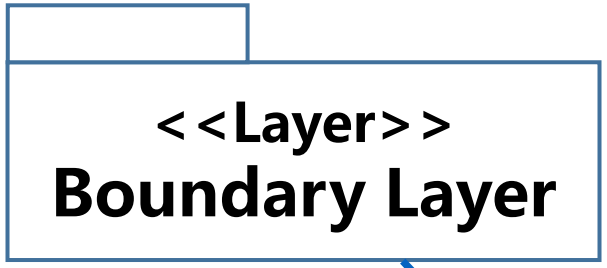
GUI

应用层：包含应用逻辑

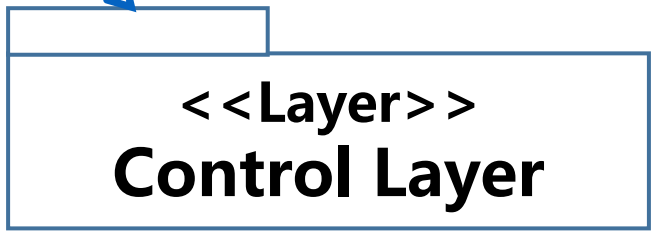
持久层：将数据存到数据库中

- **分层架构动机：**将用户图形界面，应用逻辑单独分离出来，数据库访问模块(构件)单独分离出来。优点：
  - **可扩展性好**：可以针对一个层进行修改，而不影响其它的层；可以对一层增加功能，而不影响其它的层
  - **可复用**：可以复用GUI，应用层，数据库访问层
  - **改善性能**：将各个层次分配到各个不同的物理计算节点。这样可以改善系统性能
  - **并行开发**：将**不同层开发任务**交给不同的**开发组**，并行开发，提高效率

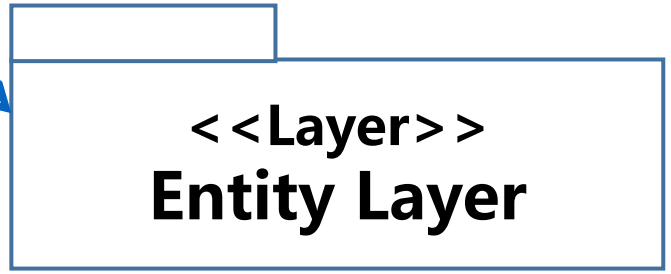
B-C-E三层架构简介



边界层：负责与参与者和外部设备进行交互



控制层：负责协调用例内部的处理流程



实体层：负责存储和访问系统内部实体信息。

- 本项目采用B-C-E三层架构作为备选架构

B-C-E三层架构解析

- 以构造型<<layer>>表示系统不同层次
- 以B-C-E三层划分系统三类处理逻辑
  - 边界层(Boundary)负责系统与参与者之间的交互，包含GUI或其它的接口
  - 控制层(Control)处理系统的控制逻辑，包括控制类，根据用户所发的请求类型，调用实体层的相关功能
  - 实体层(Entity)管理系统使用的信息，包括核心业务类，提供核心业务逻辑
- 层之间存在依赖关系：**边界层➔控制层➔实体层**

2. 关键抽象

- 关键抽象是一个通常在需求上被揭示的概念，系统必须能够对其处理
  - 来源于业务，体现了业务的核心价值，即业务需要处理哪些信息；这些信息所构成的实体即可作为初步的实体分析类
  - 领域专家建立系统的初始关键抽象候选集合
  - 再结合业务对象模型、需求和词汇表等业务文档资料进行补充和完善
  - 通过一个或多个类图来展示关键抽象，并为其编写简要的说明
  - 这里，可以引入之前建立的领域模型图

旅游申请系统中的关键抽象(领域模型，可以用领域模型图代替)

关键抽象	含义	相关概念
路线	包含具体旅游安排或介绍旅游路线	办理申请手续及相关的取消、支付等后续业务
旅游团	各个旅游路线具体组团，各个线路可组织多个不同的旅游团	路线
申请	参加人提交的旅游团申请表	旅游团
支付明细	与申请相关的定金、旅费、退费等费用支付信息	申请
参加人	某申请的参加人员信息，分责任人和普通人员两种角色；具体人员还需要区分大人和小孩	申请
联系人	参加人指定的联系人信息	参加人

3. 创建用例实现(下一节PPT讲)

Return

构造用例实现

构造用例实现

用例实现和用例是两个不同的概念，它们关注的重点不同

- 用例(描述)面向用户描述功能
- 用例实现则面向分析设计人员描述软件的内部结构

- 构造用例实现是分析最核心的工作
  - 获得实现用例行为所必须的分析类
  - 利用这些分析类来描述其实现逻辑

构造用例实现

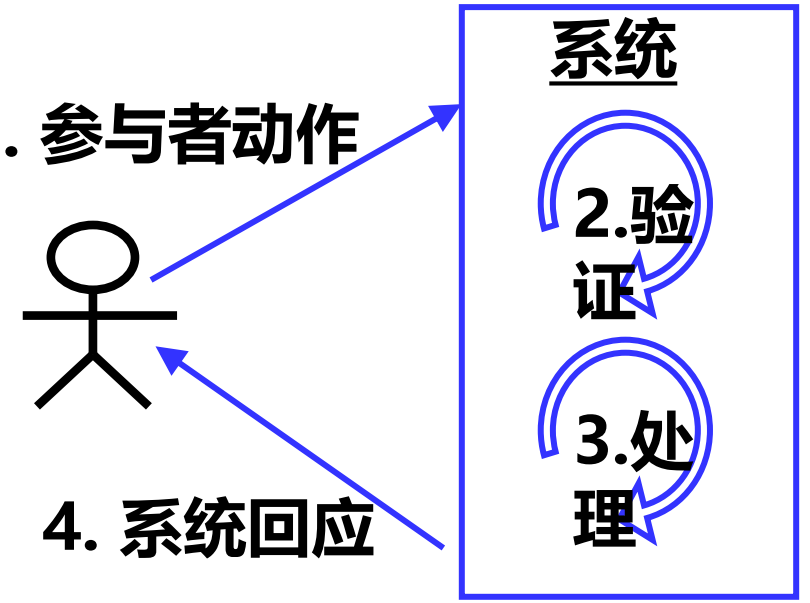
- 针对每一个用例实现，应该完成以下工作：
  - 1. 完善用例文档
  - 2. 识别分析类
    - 边界类、控制类和实体类
  - 3. 分析交互
    - 将用例行为分配给类(给类增加方法)
  - 4. 完成参与类类图
    - 参与用例实现的类的类图
  - 5. 处理用例关系

Return

完善用例文档

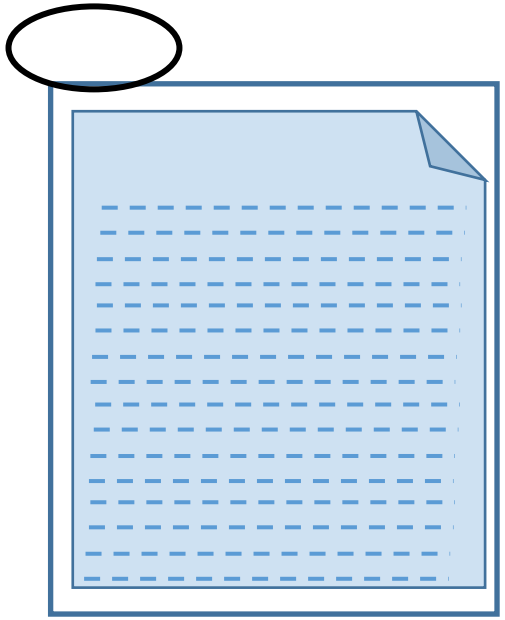
1.完善用例文档

- 需求阶段的用例文档是从用户角度看待用户问题，侧重描述交互的1、4步
- 分析阶段则需要从系统角度看待用户问题，重点关注交互的2、3步：即系统如何响应用户请求；



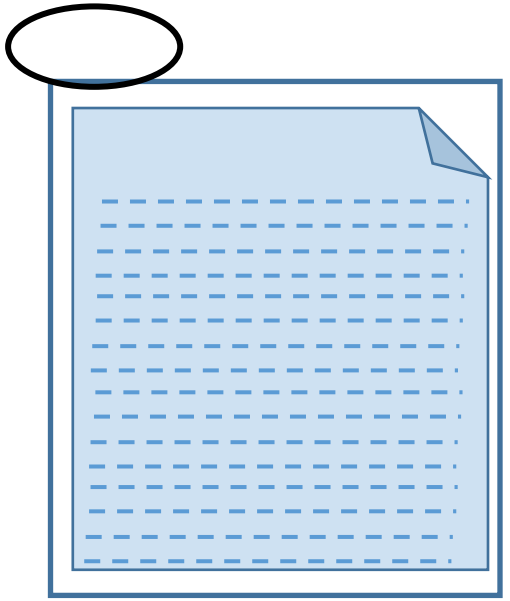
- 该阶段可对需求阶段用例文档中系统的处理流程进一步细化(仅对复杂的，难以理解的细节细化即可)
  - 获取理解系统必要的内部行为所需的其他信息：从“黑盒”到“白盒”的过渡
  - 有些情况下，可能找到一些不正确或不易理解的需求；此时，原始的用例事件流也需要进行更新

示例：对用例”办理申请手续”，基本事件流，完善用例文档



Use case

●系统计算并显示旅行费用的总额和申请订金金额



Use case

●系统根据旅游团价格和参加人情况计算费用总额以及订金金额，……

Return

从用例行为中识别分析类

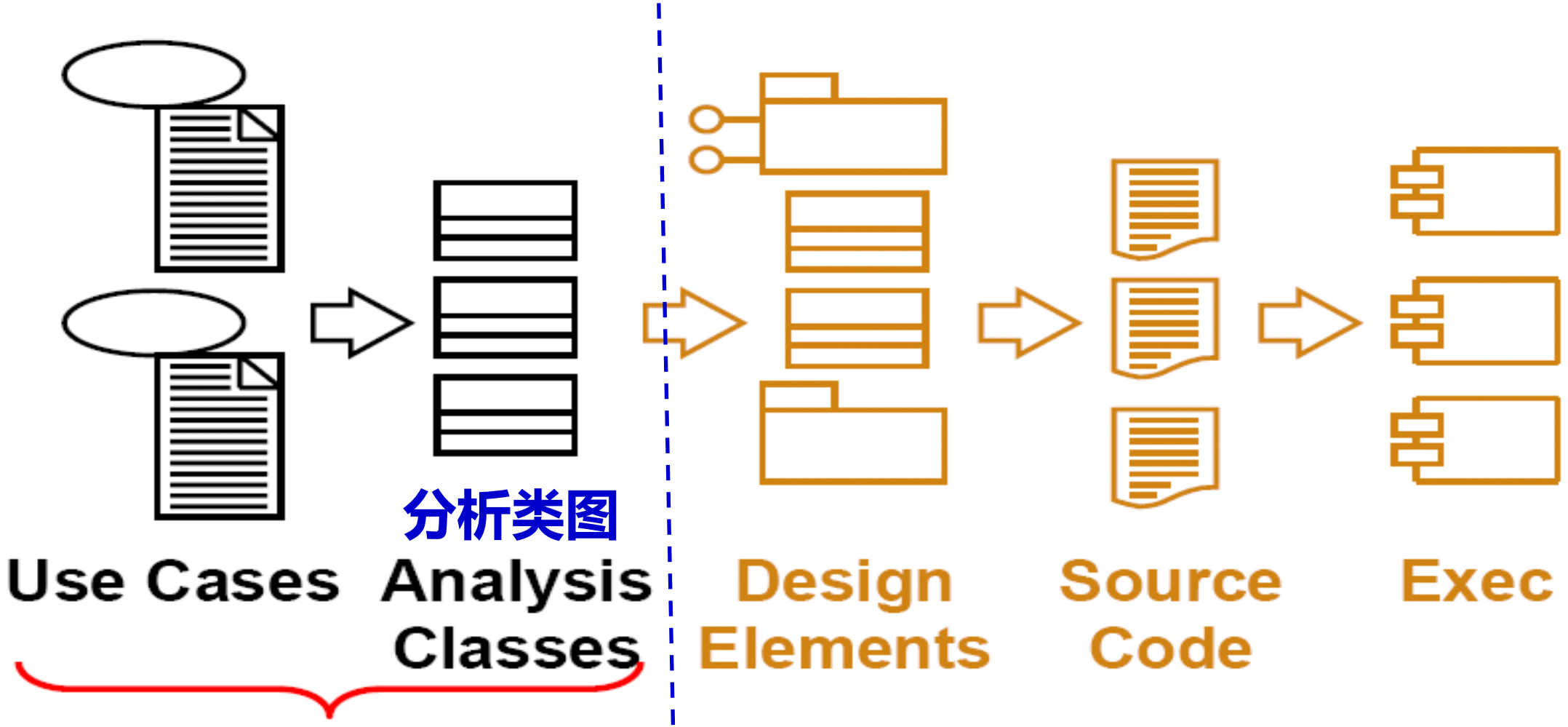
从无到有的跨越

This is the most creative job!

2. 从用例行为中识别分析类

- 在对象技术中，一个用例的全部行为都是由相应的类来完成的
- 这些行为必须被分配到类中
  - 分析阶段就是对这个过程的第一次尝试
  - 这是一个从“无”到“有”的跨越

分析类：达成目标的第一步

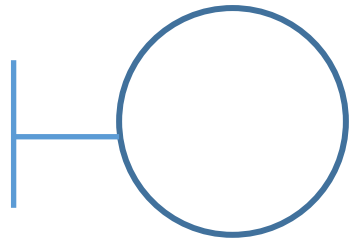


什么是分析类

- 分析类代表了“系统中必须具备职责和行为的事物”的早期概念模型
- 分析类处理主要的功能需求，模型化问题域对象
- 根据备选BCE架构定义三类分析类
  - **边界类**：系统及其参与者的边界
  - **控制类**：系统的控制逻辑
  - **实体类**：系统使用的信息

边界类

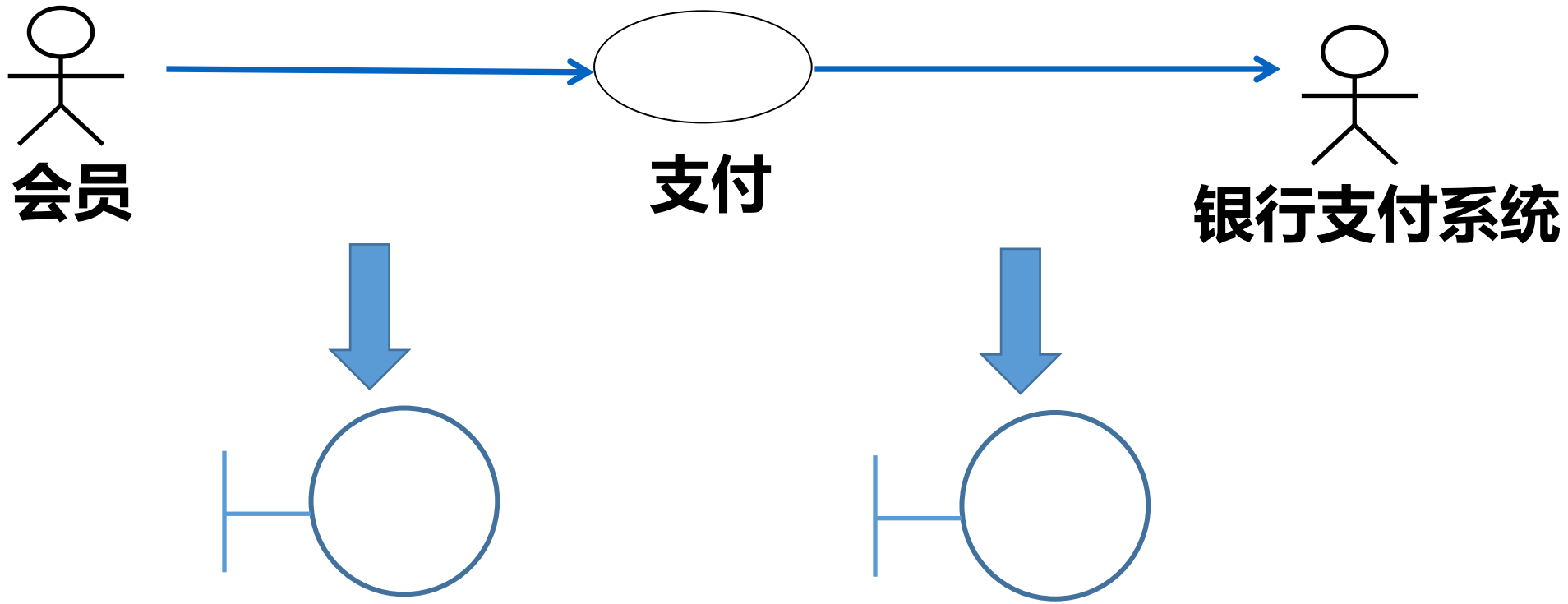
- 边界类表示系统与参与者之间的边界
  - 代表系统与环境的交互
  - 是接口和外部事物的中间体
  - 构造型<<boundary>>
- 两类边界类
  - 用户界面类(分为用户图形界面，无图形界面)
  - 系统和设备接口类



边界类记号

示例：识别边界类

- 每对参与者/用例定义一个边界类



会员支付界面类

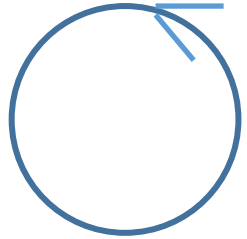
生成用户图形界面，以便会员操作此界面，与系统交互。

银行支付系统接口类

类中包含一些方法，而在这些方法代码中，调用银行支付系统接口。

控制类

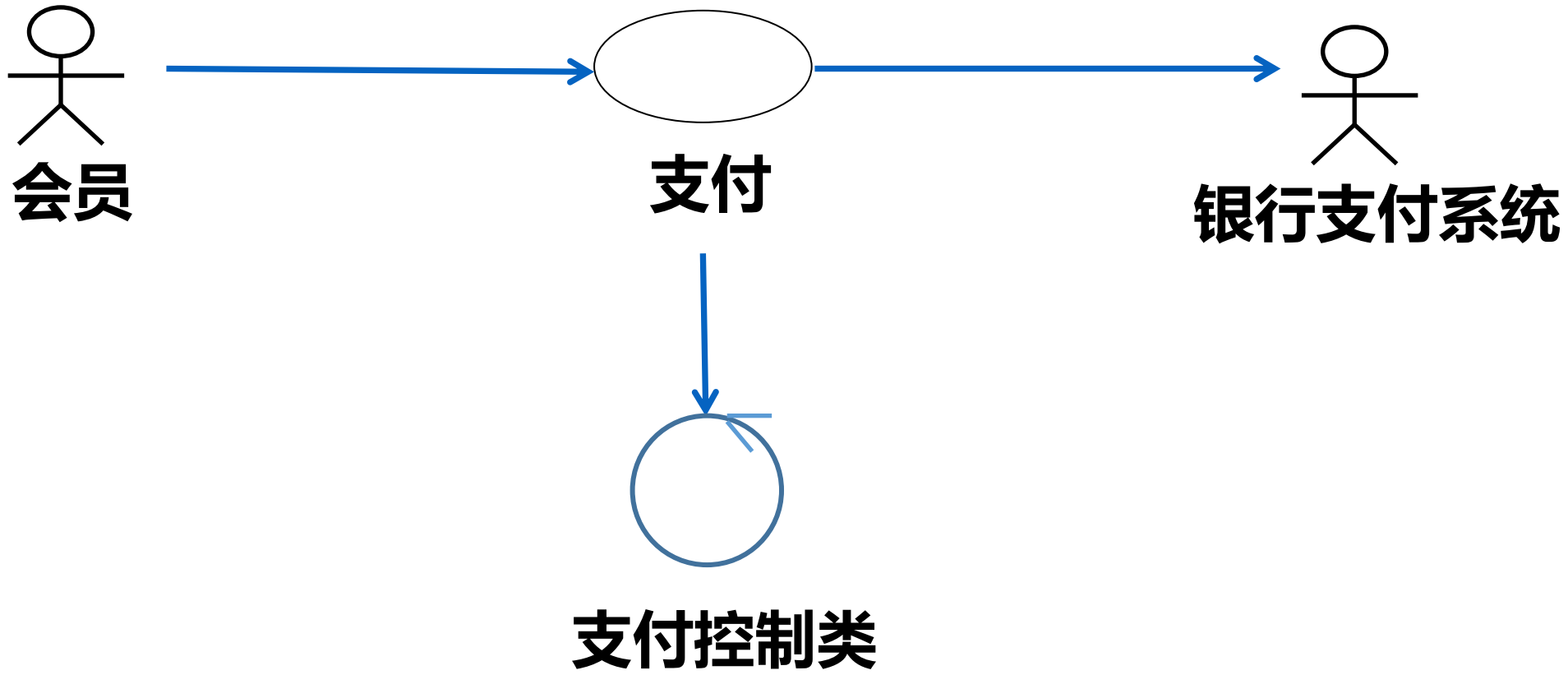
- 控制类表示系统的控制逻辑
  - 系统行为的协调器
  - 构造型<<control>>
- 识别控制类
  - 在系统开发早期，为每个用例定义一个控制类，负责该用例的控制逻辑
  - 针对复杂用例，可为备选路径分别定义不同控制类



控制类记号

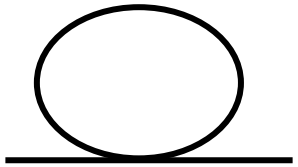
示例：识别控制类

- 通常，每个用例定义一个控制类
  - 随着分析的继续，一个复杂用例的控制类可以发展为多个



实体类

- 实体代表了待开发系统的核心概念，来自于对业务中的实体对象的归纳和抽象（例如银行业务系统中的Account类）
  - 实体类的主要职责是存储和管理系统内部的信息
  - 实体类中所封装的数据往往都是永久的，都是应该存储到数据库之中的。



实体类记号

- 使用构造型<<entity>>
- 可以从以下文件中找到实体类
  - 用例事件流(需求)、
  - 业务模型(业务建模)、
  - 词汇表(需求)

识别实体类

a) 关键抽象得到的类是实体类

b) 在分析用例事件流中的名词和名词短语中找出系统所需的实体对象。

- 对这些名词、名词短语进行筛选，抽取出系统对象，并抽象成类
- 综合考虑在系统中的意义、作用和职责，对于所识别的类进行命名

指南：名词筛选法识别实体类

- 名词筛选法识别实体类的基本思路：
  - 将用例事件流作为输入，找出名词或名词性短语，得到实体类的初始候选列表
  - 合并那些含义相同的名词(如：购物车，购物筐，购物柜)
  - 删除那些系统不需要处理的名词（食品，发型，气质，等等与系统无关的名词）
  - 删除作为参与者的名词（？）
  - 删除与实现相关的名词（例如，数据库，堆栈，队列等）
  - 删除那些作为其它实体类属性的名词(如：颜色，年龄，性别，身份证号码等)
  - 对剩余的名词，综合考虑它在当前用例以及整个系统中的含义、作用以及职责，并基于此确定合适的名字，作为初始实体类存在

例子：在旅游申请系统的“办理申请手续”基本事件流用例描述中筛选出实体类

1. 用下划线标注事件流中的名词
   ① 用例起始于旅客需要办理申请手续
   ② 前台服务员录入要申请的旅游团旅行路线代码和出发日期
   ③ 系统查询要申请的旅游团信息(A-1)
   ④ 系统显示查询到的旅游团和相关路线信息(D-1),(A-2, A-3)
   ⑤ 前台服务员录入本次申请信息(D-2)
   ⑥ 系统计算并显示旅行费用的总额和申请定金金额
   ⑦ 申请责任人缴纳订金，前台服务员录入订金信息，提交本次申请信息
   ⑧ 系统保存该申请信息（A-4），用例结束
2. 合并同义词：旅游团和旅游团信息合并为旅游团
3. 删除系统不需要处理的名词：用例，申请手续，系统
4. 删除参与者：前台服务员
5. 删除与实现相关的名词：无
6. 删除可作为属性的名词：路线代码作为路线类的属性被删除，出发日期作为旅游团的属性被删除； 旅行费用总额，申请订金金额可以作为类申请信息类的属性；可以额外定义一个支付明细的实体类。
7. 剩余名词：旅客，旅游团，相关路线信息，申请信息，支付明细；修改后：旅客➔参加人，相关路线信息➔路线，申请信息➔申请。最终确定实体类：参加人，旅游团，路线，申请和支付明细。

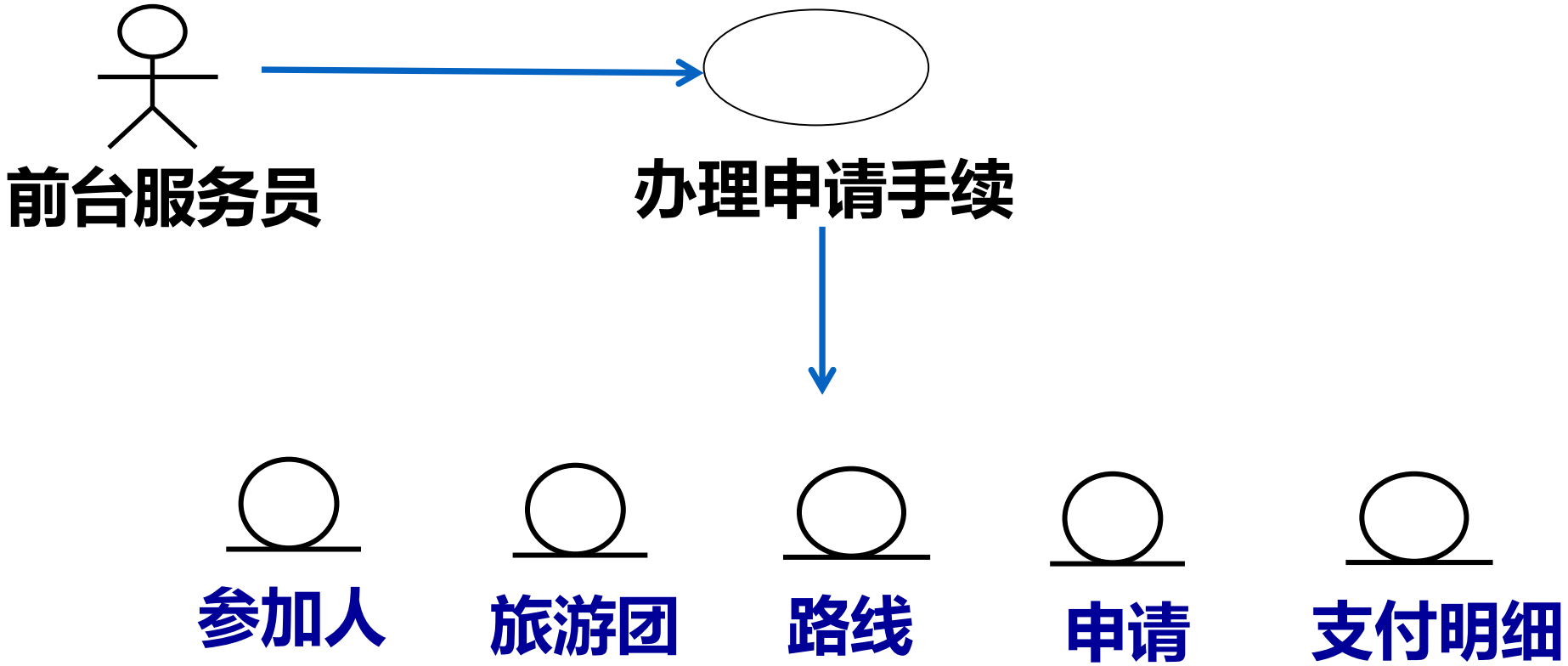
识别实体类总结

- 在实际应用中，依赖于类似项目的经验和对业务及系统的理解（或领域专家意见），**建立领域模型图，**来获取系统关键抽象，作为**初始的实体类**，再辅以名词筛选法补充完善实体类

- 此外，还有其他实体类的来源
  - 系统原始需求书/问题描述
  - 该领域相关文献、专家意见或个人知识
  - 过去的类似系统
- 后续职责分配中可能识别一些新的实体类(**经常发生**)
- 实体类的命名要用该领域中最经常使用的名称

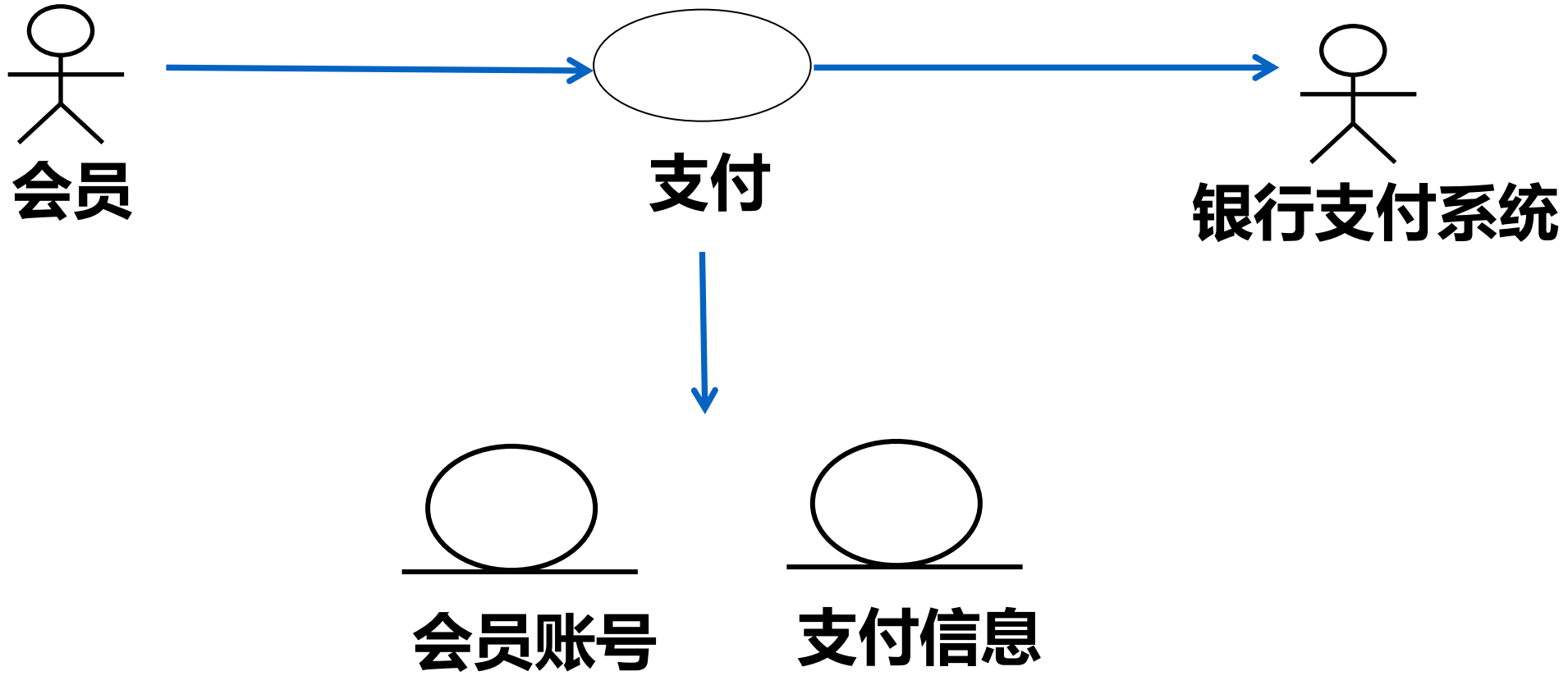
示例：候选实体类

- “办理申请手续”用例基本路径中描述可以得到如下的候选实体类



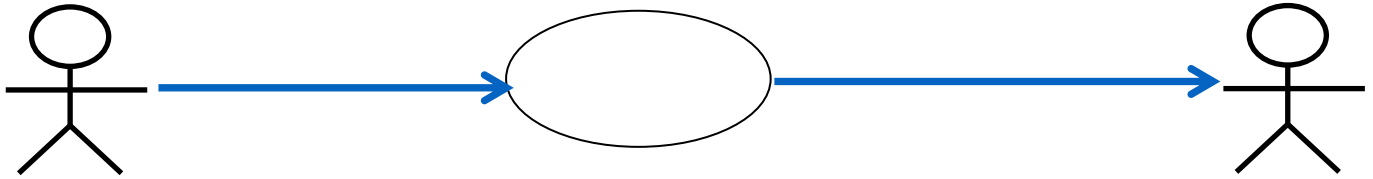
示例：候选实体类

- “支付”用例基本路径中描述可以得到如下的候选实体类

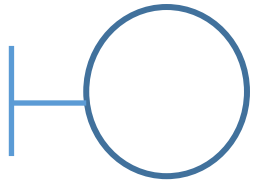


示例：总结：分析类

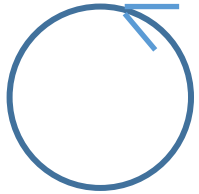
用例模型



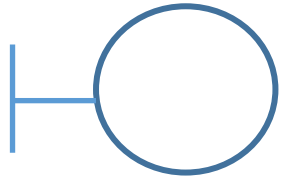
分析模型



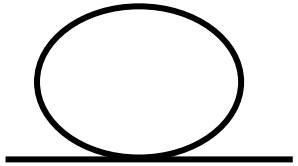
会员支付界面类



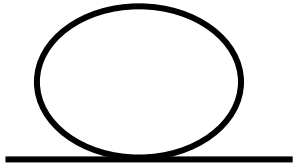
支付控制类



银行支付系统接口类

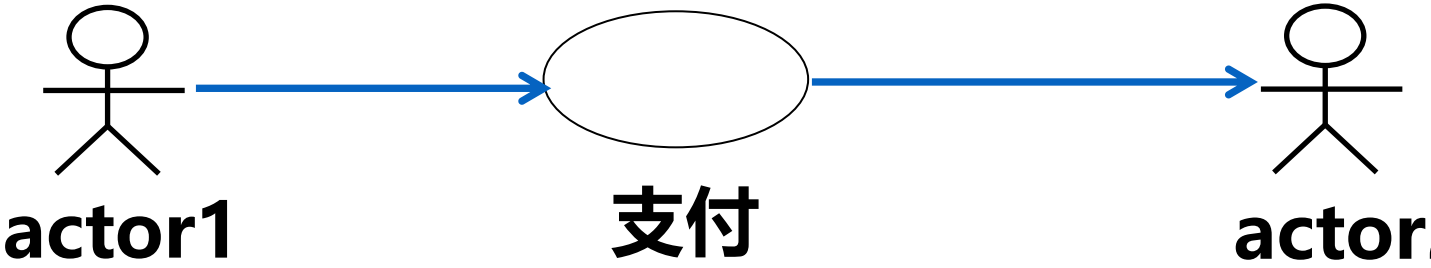


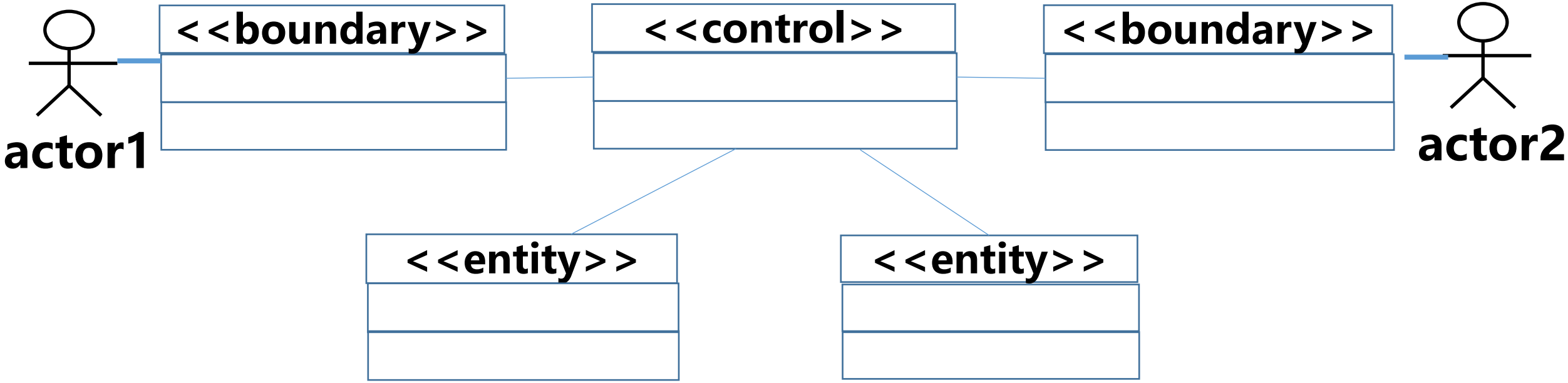
支付信息



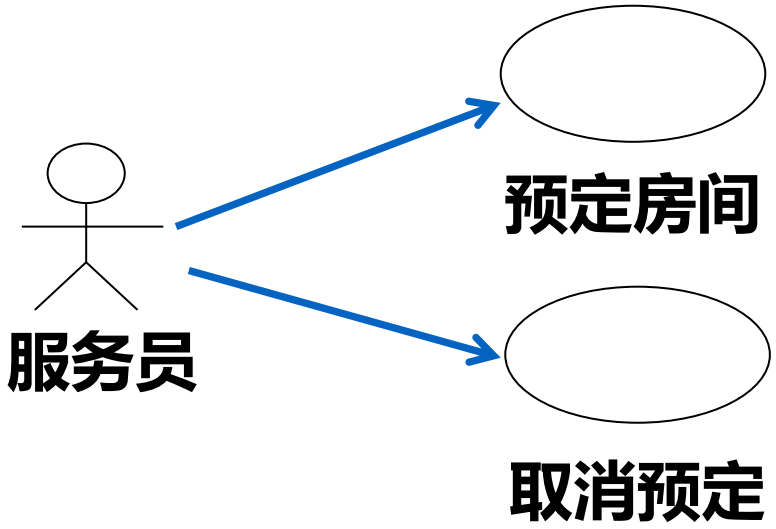
会员账号

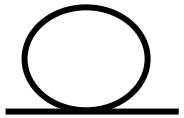
总结：从用例中识别分析类（每个用例都需要一个控制类）

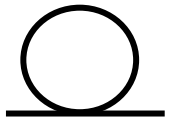


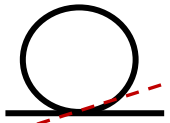


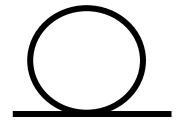
总结-旅店预订系统中识别分析类

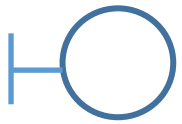


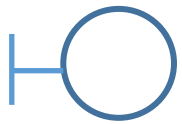

Customer

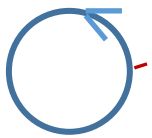

Room

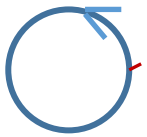

Reservation

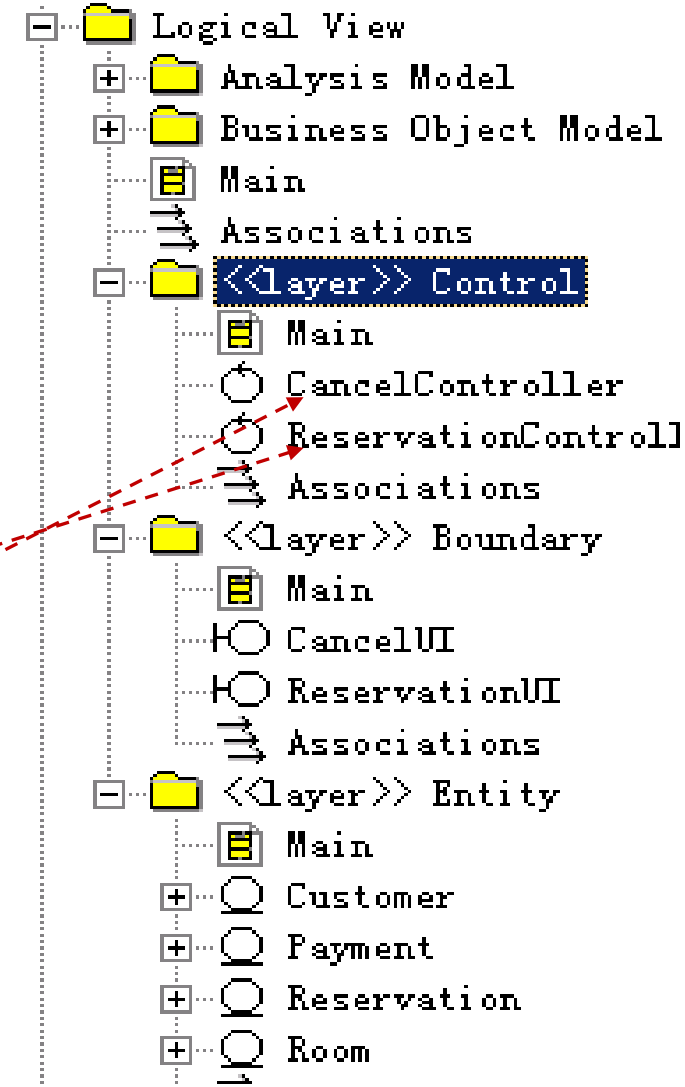

Payment


ReservationUI

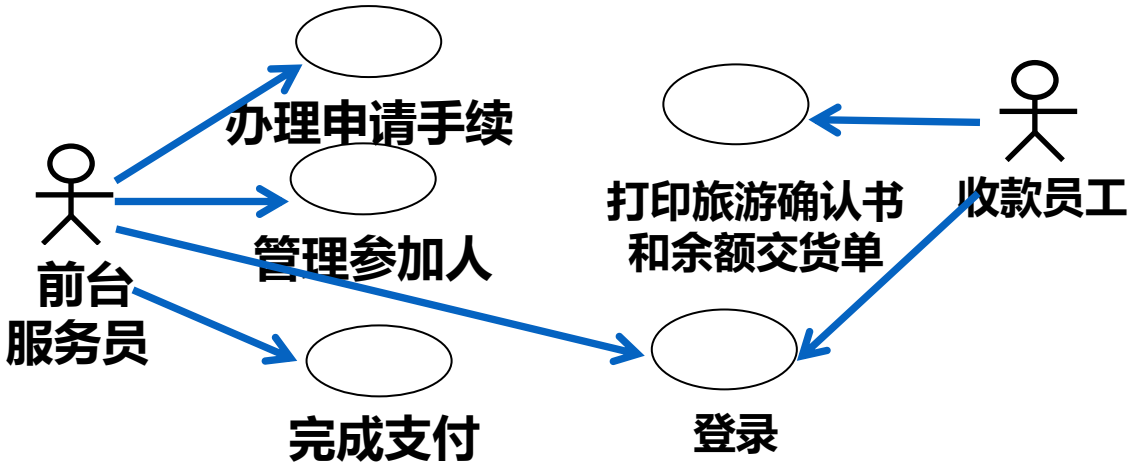

CancelUI

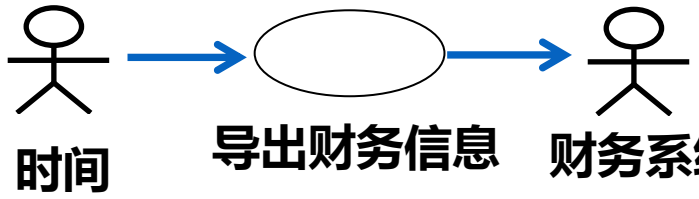

Reservation control

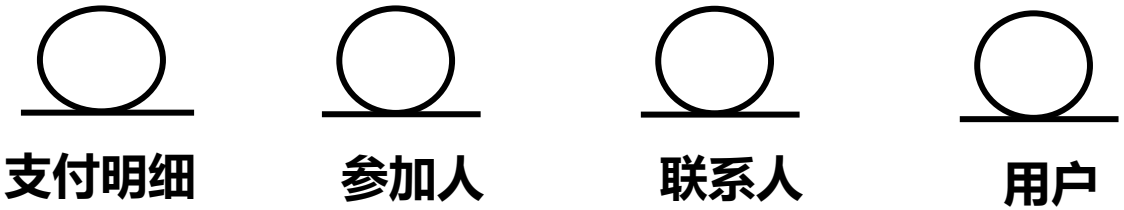

Cancel Control

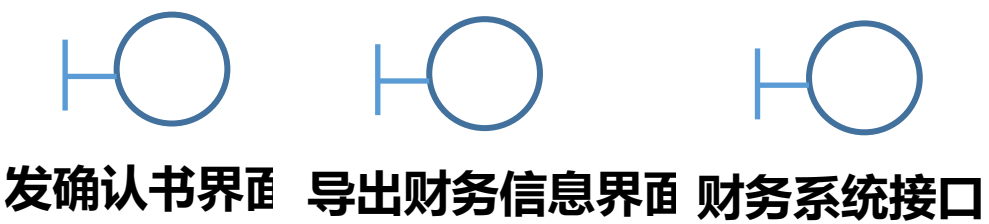


总结-旅游申请系统中的分析类









Return