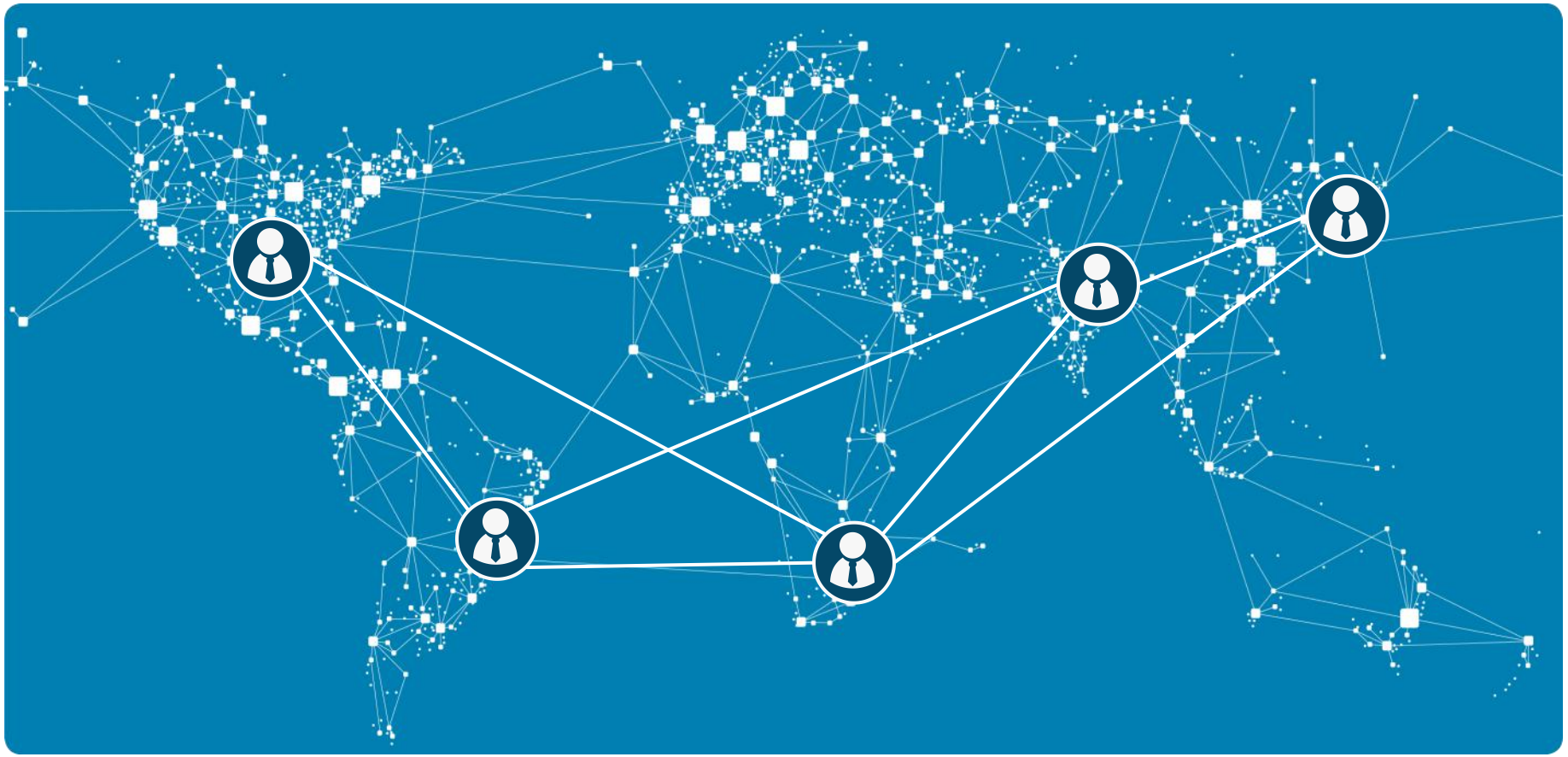


用例建模

用例建模

用例建模概念

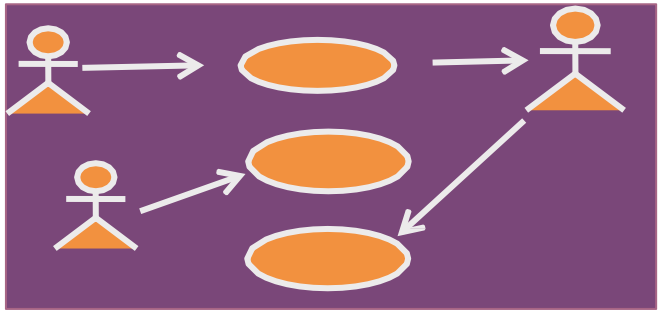
用例建模过程

用例建模精讲

建模工具介绍

用例建模概念

用例在需求管理过程中的作用



分析问题

理解干系人的需求

定义系统

用例 规约
(提纲)

管理领域
（Manage the Scope）

完善系统

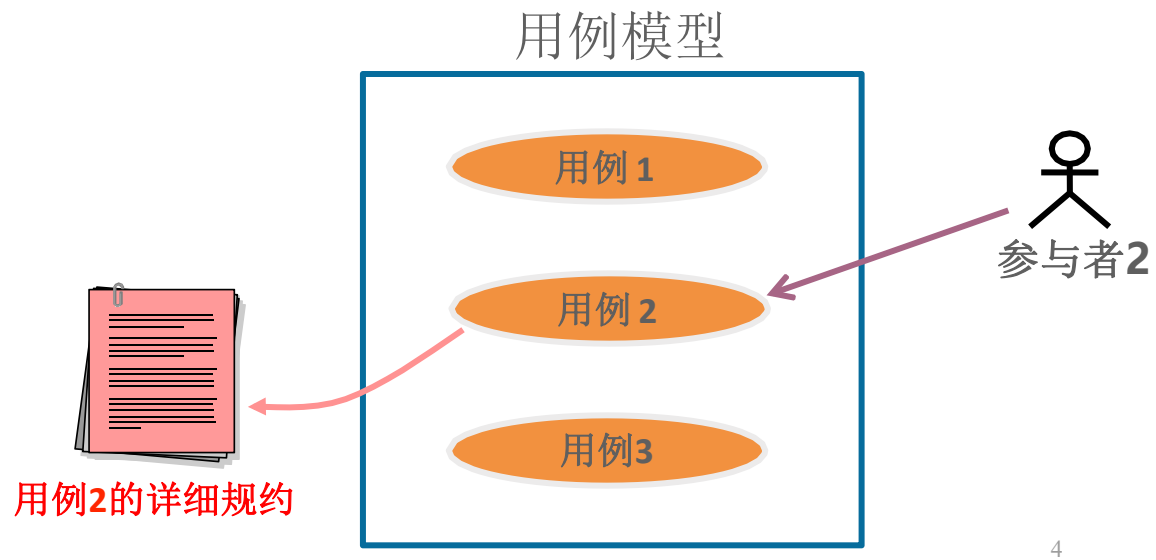
管理变化的需求

用例 规约
(详细)

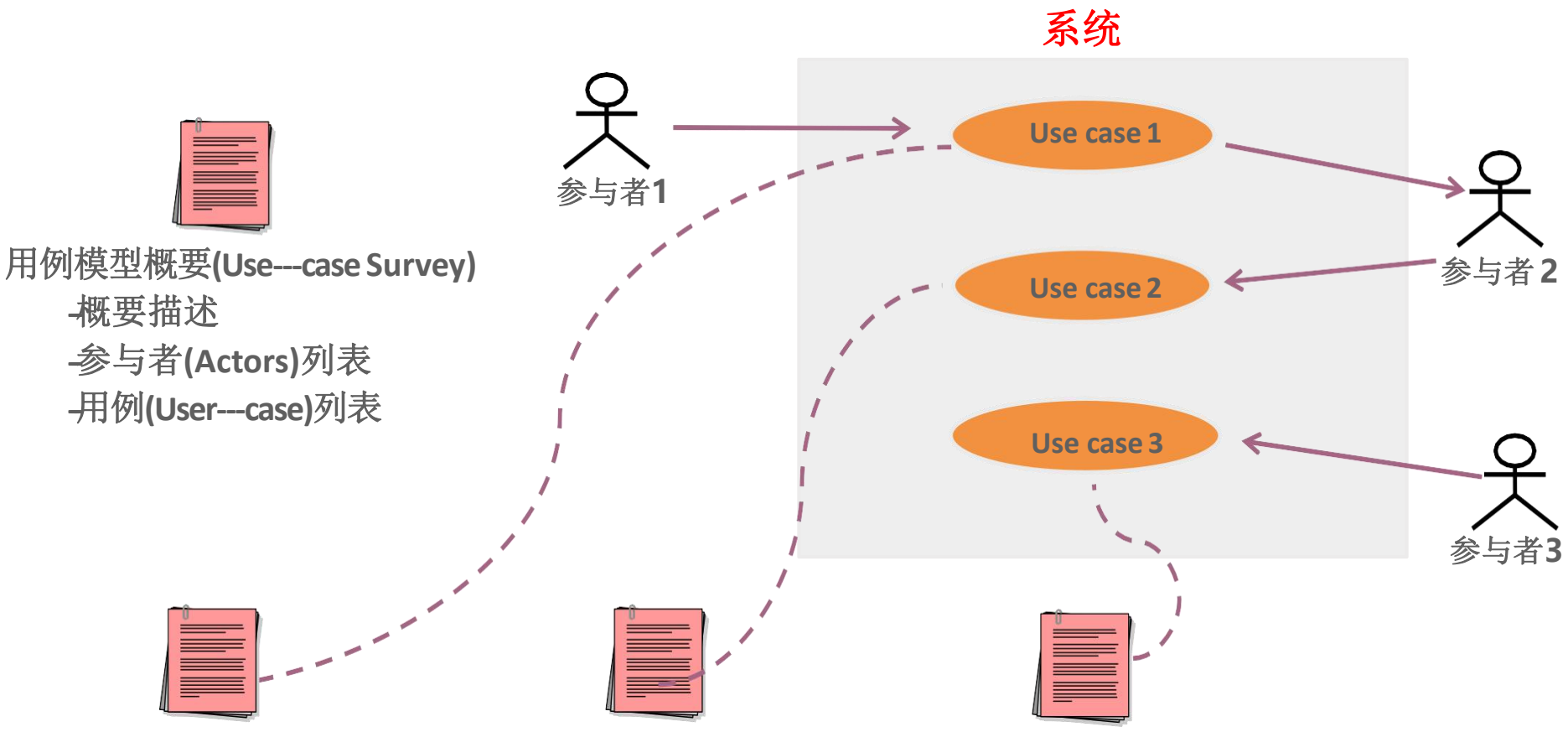
为什么需要用例建模--描述系统的功能性需求

- 关联干系人需求以及软件需求
- 确认与系统交互的人或对象（参与者）
- 定义系统的边界
- 捕捉和传达系统的理想行为（用例）

- 验证或确认需求
- 规划工具



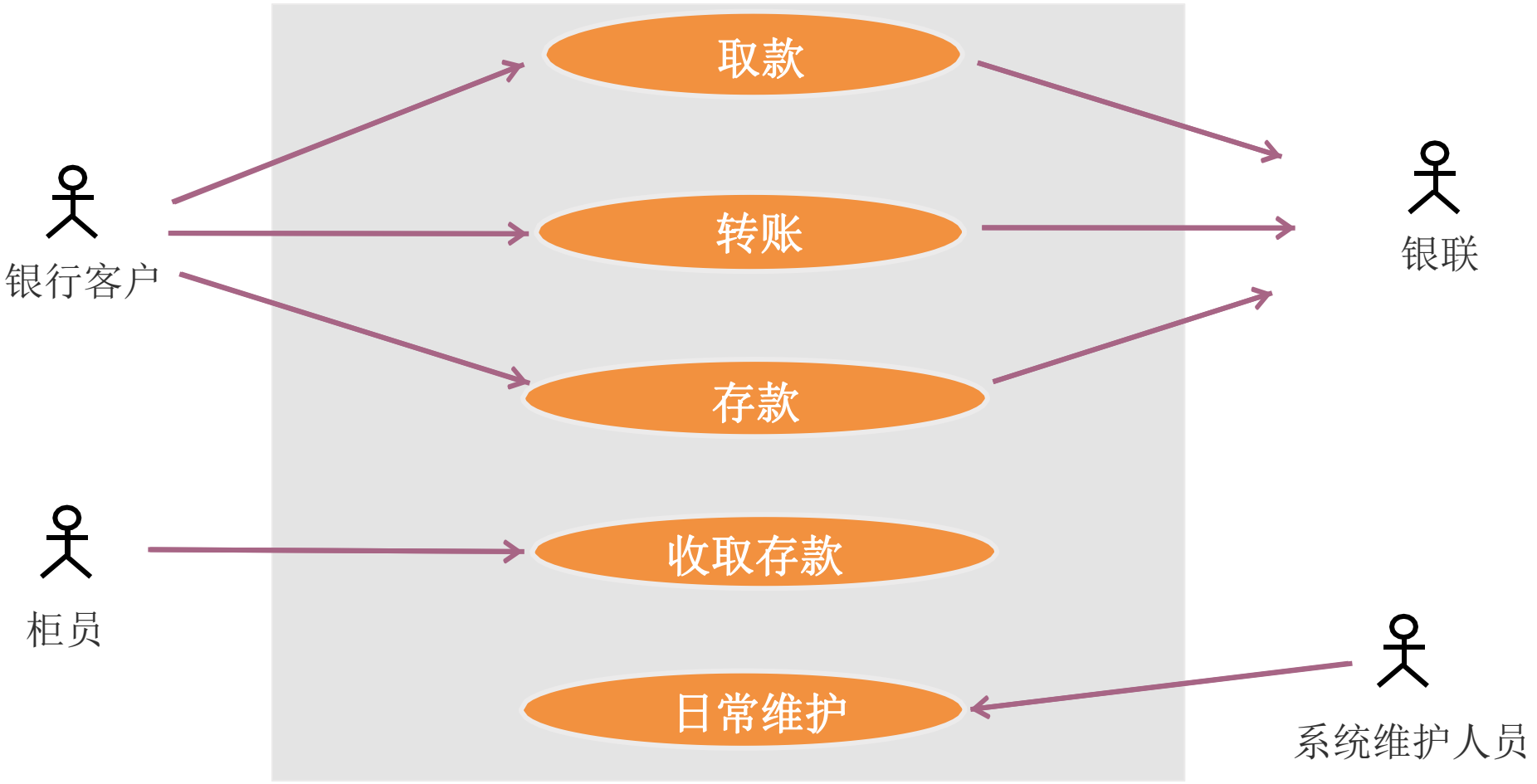
用例模型的表示--文本描述



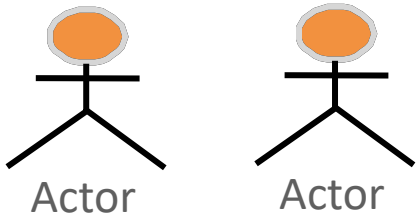
用例模型的表示--用例图

- ATM涉及哪些业务？
- ATM会与哪些系统或对象进行交互？
- 不同对象、系统是如何和ATM进行交互的？

用例模型的表示--用例图



用例图的主要元素



参与者（*Actor*）

与系统交互的人或外部系统

用例（*Use case*）

系统为参与者提供的有价值的服务功能

关联（*Association*）

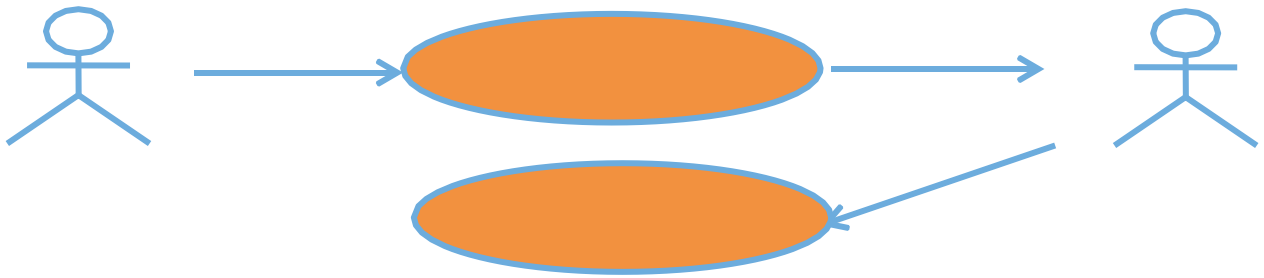
用例图中用例与参与者之间的交互关系

什么是用例

一个用例　定义**系统**的一系列**行为**，通过此可为**参与者**提供**有价值**且**可观测**的结果。

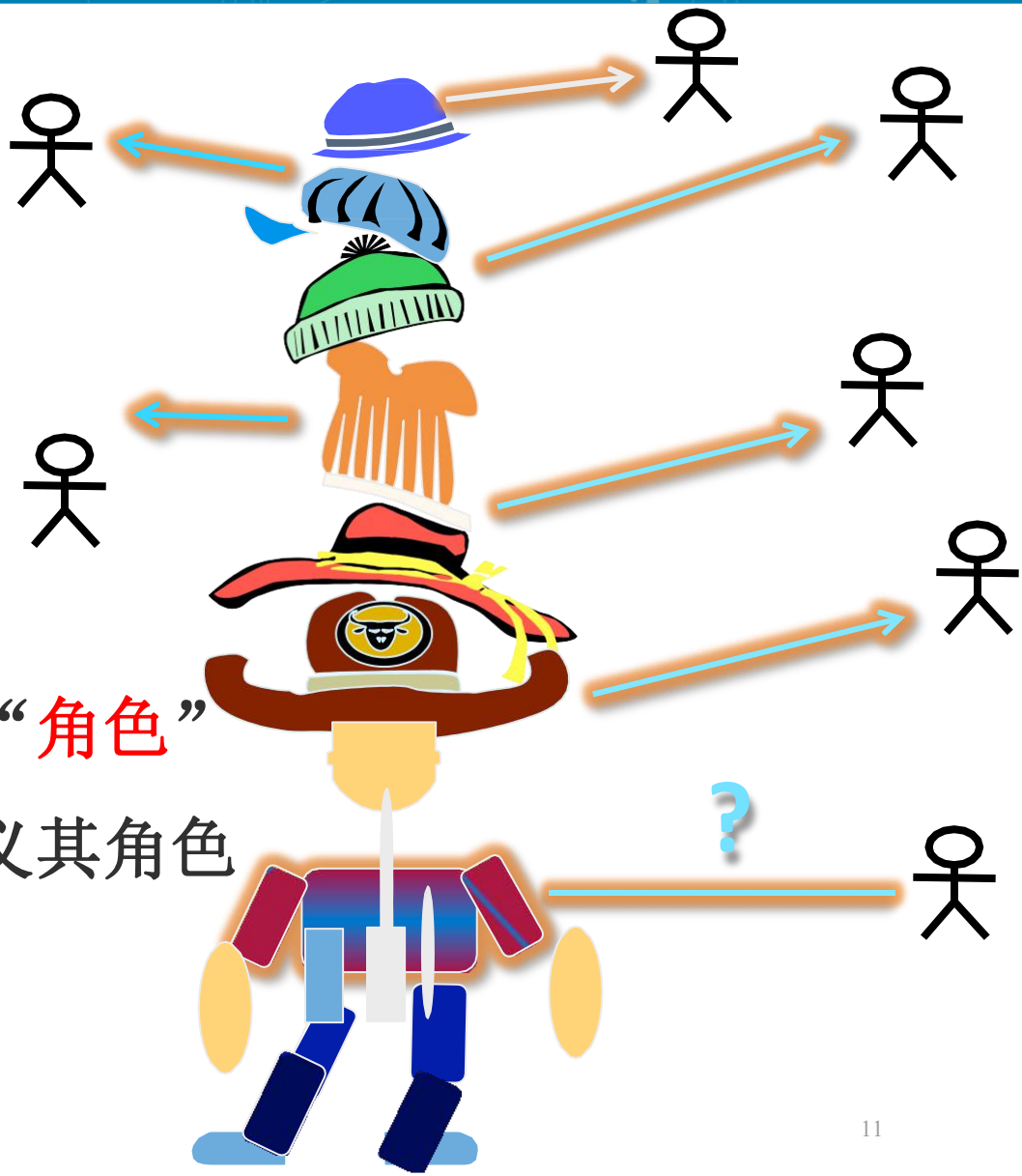
用例包含软件系统需求

- 用例
  - 定义一个参与者要用到的系统功能
  - 描述系统为实现参与者价值所开展的行为序列
  - 对参与者与系统之间的交互活动进行建模
  - 从特定的用户角度出发，是完整的，实现特定用户价值的事件流



参与者的定义：关注角色

- 与系统交互的人
- 与系统交互的硬件组件
- 或者其他的外部系统



- 关注的重点是所承担的“角色”
- 参与者的名字要明确定义其角色

参与者定义与角色划分

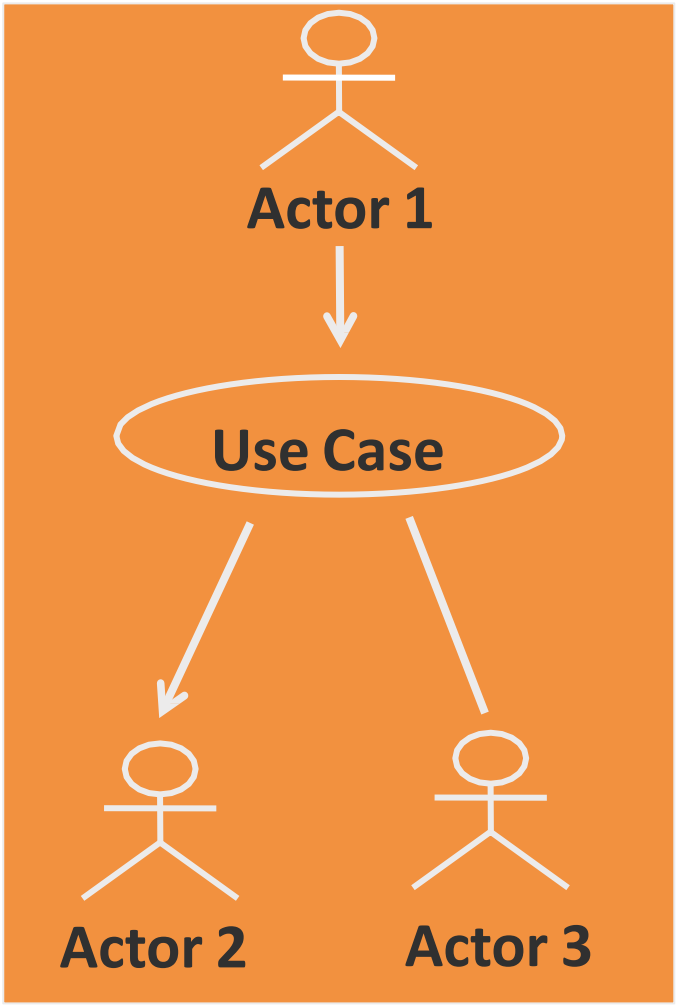
张跃：数学系的教授
经管学院的博士生

李玫：软件学院本科生

张跃和李玫都具有学生角色

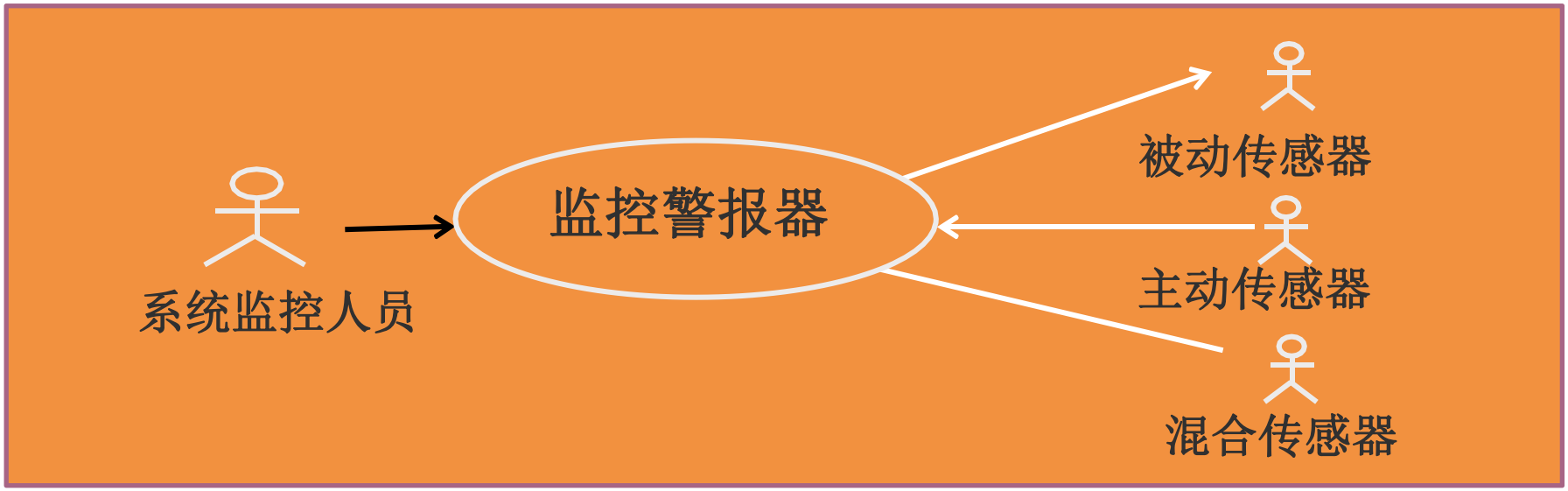
张跃同时也具有教授的角色

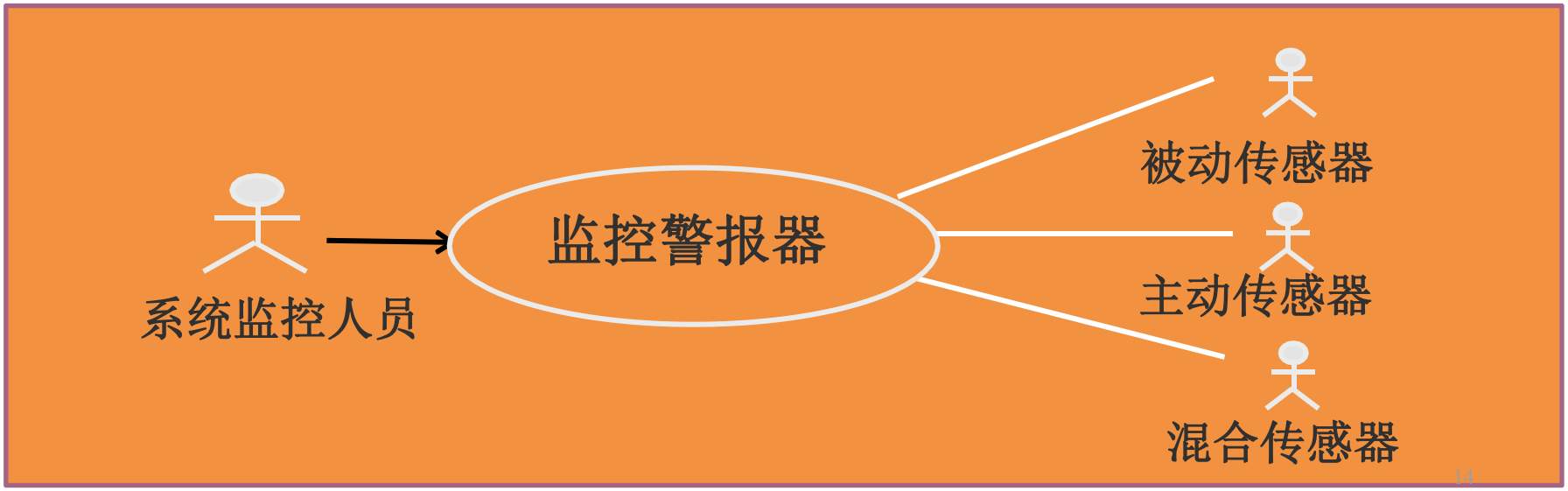
交互--关联（Association）



- 参与者与用例之间的交互通道
- 用一条直线表示交互——关联
  - 有箭头的关联指出是谁发起的交互
  - 没有箭头则表明双方都可以发起交互

交互--关联（Association）





每一个交互---关联代表一个完整的对话

学生登录到系统
系统验证登录过程
学生请求课程信息

系统显示课程列表
学生选择课程
系统将选择的课程纳入课程表

系统传输请求
课程目录系统返回课程的信息

场景（Scenario）是用例的实例

场景1

登录系统

认证登录
输入主题词进行查询
获取课程列表
显示课程列表
选择课程
确认课程可选
显示最终的课程表

场景2

登录系统

认证登录
输入主题词进行查询
无效主题词
再次输入课程
获取课程列表
显示课程列表
选择课程
确认课程可选
显示最终的课程表

用例建模过程

构建用例模型的步骤

- 第一步：找到所有的参与者和用例
  - 识别出参与者并做简单的描述
  - 识别出用例并做简单的介绍

- 第二步：编写用例
  - 列出用例
  - 给用例事件流程划分重要等级
  - 按照重要程度排序详细描述事件流程

寻找参与者

- 谁/什么使用系统？
- 谁/什么从系统中获取信息？
- 谁/什么向系统提供信息？
- 公司的哪个部门会使用系统？
- 谁/什么负责系统的维护？
- 还有哪些其他系统会使用系统？

识别参与者--是谁与系统进行交互？

学生

教务人员

中心选课系统

学生并不直接操作选课系统；是教务人员进行操作。
或者，构建一个基于浏览器的在线应用？

学生

在线选课系统

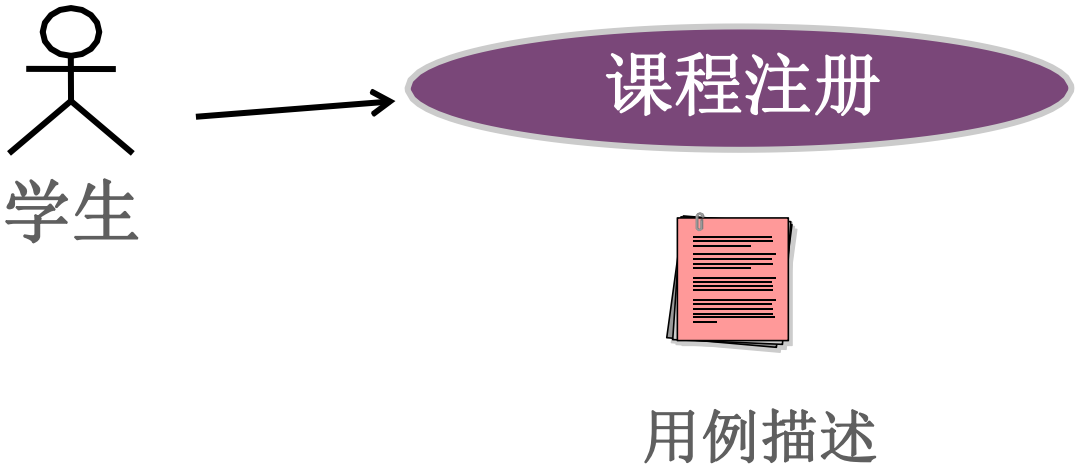
(course.whut.edu.cn)

参与者的描述

名称　　学生

简要描述　　注册课程的用户

和用例之间的关系

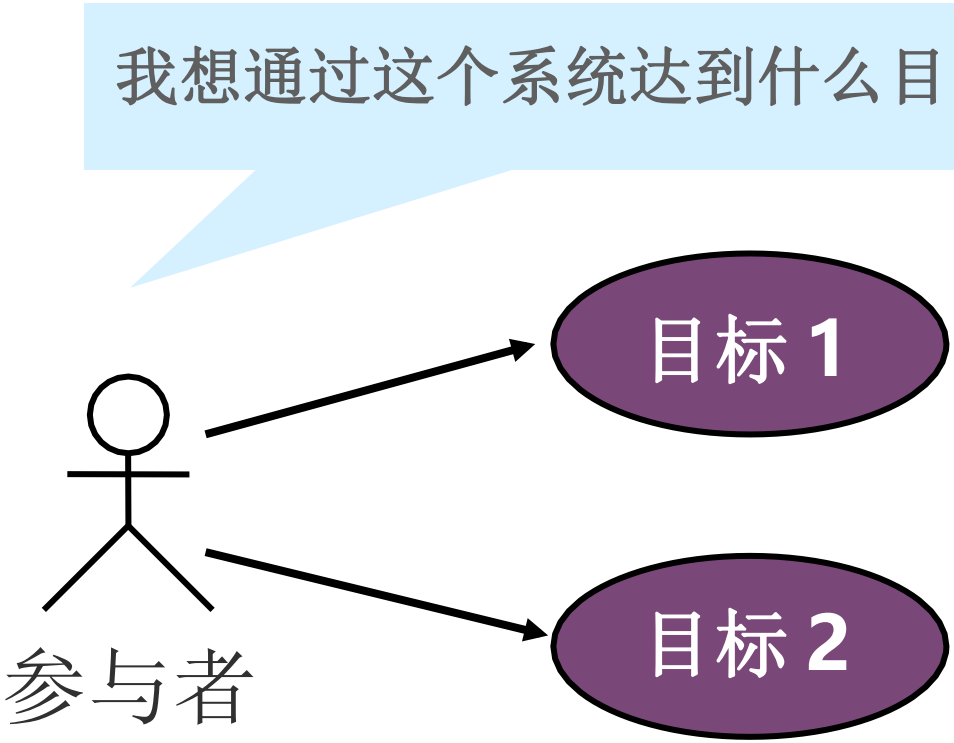


参与者建模的检查项

- 是否找全所有的参与者？是否对系统环境中所有的角色进行了描述和建模？
- 每个参与者是否至少与一个用例发生了交互？
- 是否可以为每一个角色找到至少两个实例？
- 不同参与者与系统的交互是否一致，扮演的角色是否相似？如果有，则应该要合并这些参与者作为同一种角色

寻找用例

基本策略：把自己当作actor，与设想中的系统进行交互。



注意：确定Use Case和确定actor不能截然分开

识别用例

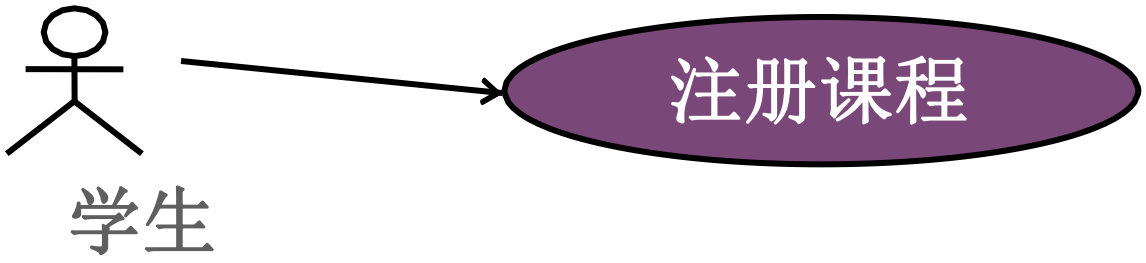
- 每个参与者的目标是什么？
  - 为什么参与者要使用这个系统？
  - 参与者是否需要对系统中数据进行创建，存储，更改，删除或者读取的操作？为什么？
  - 参与者是否需要将外部事件或发生的改变告知系统？
  - 参与者是否需要知道系统内部发生的事件或改变？
- 系统是否能够应对业务中所有的正确行为与操作？

用例的描述

名称　　注册课程

简要描述　　学生选择下个学期想上的课程。生成必修课和选修课的课表信息。

与参与者的关系

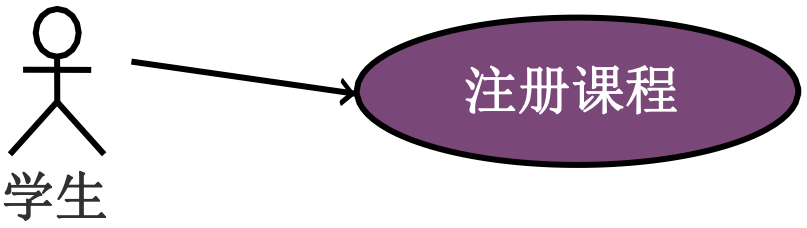


用例的命名

- 表明参与者的目标或者作用
- 使用主动语态：用动词起始
- 设计一系列操作流程(to-do list)
- 几种表达：
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration

默认的判别规则

将参与者的名称与用例的名称连成句子，检查是否有实际的意义



哪种表达形式可以表现出参与者的意义或价值？哪些不可以？
你会选择哪个作为你的用例名称？为什么？

用例建模过程中的检查项

- 用例建模是为了表示系统的行为。通过模型可以很容易理解系统进行的操作
- 应该识别出所有的用例，用来表达所有的需求。
- 系统的任何一个特性都可以找到对应的用例
- 用例模型并不包含多余的行为；所有的用例可以追溯到系统的功能性需求作为验证。
- 去掉所有的**CRUD** 类的用例

  创建(Create), 查找(**R**etrieve), 更新(**U**pdate), 删除(**D**elete)

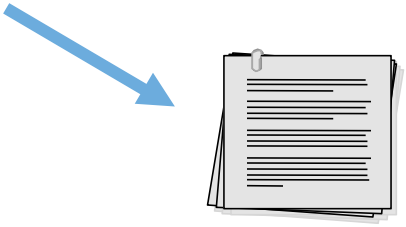
构建用例模型的步骤

- 第一步：找到所有的参与者和用例
  - 识别出参与者并做简单的描述
  - 识别出用例并做简单的介绍

- 第二步：编写用例
  - 找出用例
  - 给用例事件流程划分重要等级
  - 按照重要程度排序详细描述事件流程

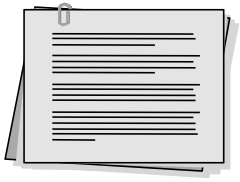
用例建模的过程：用例图->用例提纲->用例详细规约

+ 用例简单描述



注册课程用例提纲

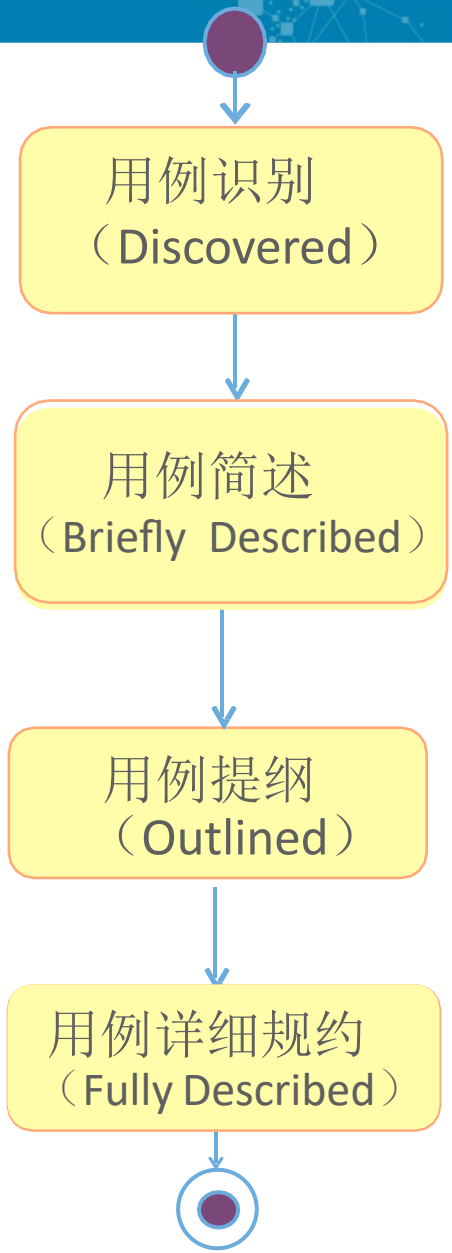
+ 粗略列出事件流程
  - 大体步骤



注册课程用例的详细规约

+ 列出详细的事件流程
  - 按步骤（详细）
+ 特殊的规约说明
+ 前置/后置条件

用例的全生命周期



下订单

简述: 客户带着要购买的货物到收款处，收银员使用POS机扫描记录每一种预购买的货物。系统计算总价并打印清单。客户付款，系统验证并保存销售记录。系统更新库存，客户得到收条并带着货物离开。

下订单（概述）
– 事件流
– …

下订单（用例规约）
--- 详细的事件流
--- 特殊的需求
--- 前置/后置条件

用例简述的例子

- **用例简述:**

一段简洁的摘要，主要描述用例的成功场景

- **下订单:**

客户带着要购买的货物到收款处，收银员使用POS机扫描记录每一种预购买的货物。系统计算总价并打印清单。客户付款，系统验证并保存销售记录。系统更新库存，客户得到收条并带着货物离开。

用例概述（用例提纲）的例子

- 用例概述：
  - 非正式、随意的格式
  - 覆盖各种场景

候选场景1:

1~4 同主成功场景

5 付款验证失败

6 通知客户并要求使用其他付款方法

下订单

主成功场景:

1客户带着需要购买的货物到达收款处

2出纳员使用POS系统记录每一个要购买的货物

3系统计算总价，告知客户价格

4客户付款

5系统确认收款，保存销售记录

6系统更新库存

7客户得到收条后带着货物离开

候选场景2:

1 同主场景

2 若系统检测到与POS系统通信失败

3 通知客户系统故障

详细用例规约的例子

用例名称: 下订单 （Place Order）

前置条件: 用户通过身份认证登录系统

描述:

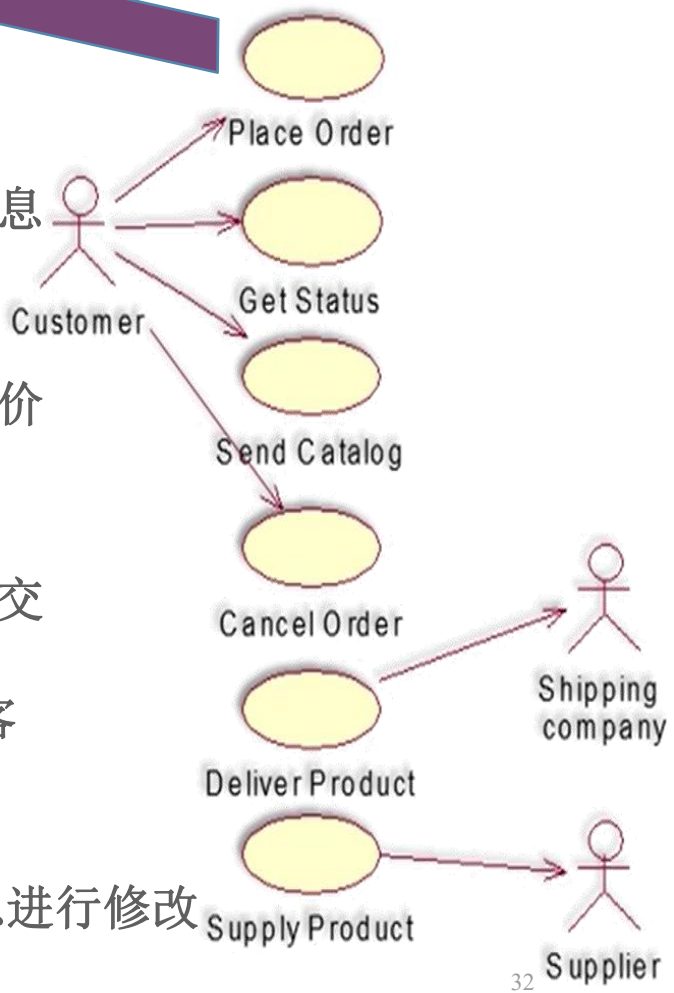
1. 当顾客选择“下订单”时，进入该用例流程
2. 顾客输入姓名和地址信息
3. 如果顾客仅输入了邮政编码，系统会提供州和城市信息
4. 顾客输入代购买的物品编码
5. 系统显示每个产品的描述信息和价格信息
6. 系统将持续记录顾客输入的所有商品信息和相应的总价
7. 顾客输入信用卡付账信息
8. 顾客选择提交(Submit)
9. 系统确认信息，保存待付款订单信息，将账单信息提交给账务系统
10. 确认付账成功后，系统标记账单为完成状态，向顾客显示账单ID信息，用例结束

异常情况:

第9步中，如果信息不正确，系统将提示顾客对相应信息进行修改

后置条件: 系统保存订单并且标记为已确认。

用例图



总结：Use Case模型的建立步骤

⑴ 找出系统外部的参与者和外部系统，确定系统的边界和范围；

⑵ 确定每一个参与者所期望的系统行为；

⑶ 把这些系统行为命名为Use Case；

⑷ 使用泛化、包含、扩展等关系处理系统行为的公共或变更部分；

⑸ 编制每一个Use Case的脚本；

⑹ 绘制Use Case图；

⑺ 区分主事件流和异常情况的事件流，可以把表示异常情况的事件流作为单独的Use Case处理；

⑻ 细化Use Case图，解决Use Case间的重复与冲突问题。

设定系统边界

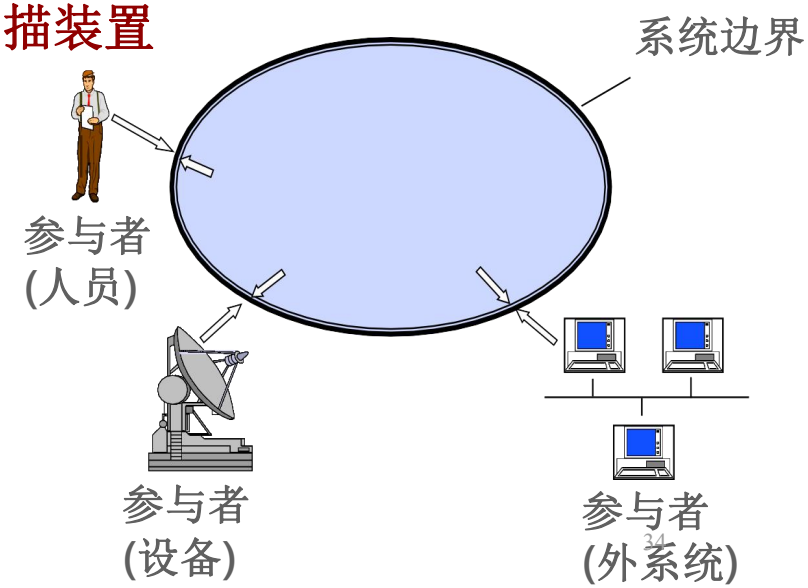
- 系统边界:

  一个系统所包含的所有系统成分与系统以外各种事物的分界线

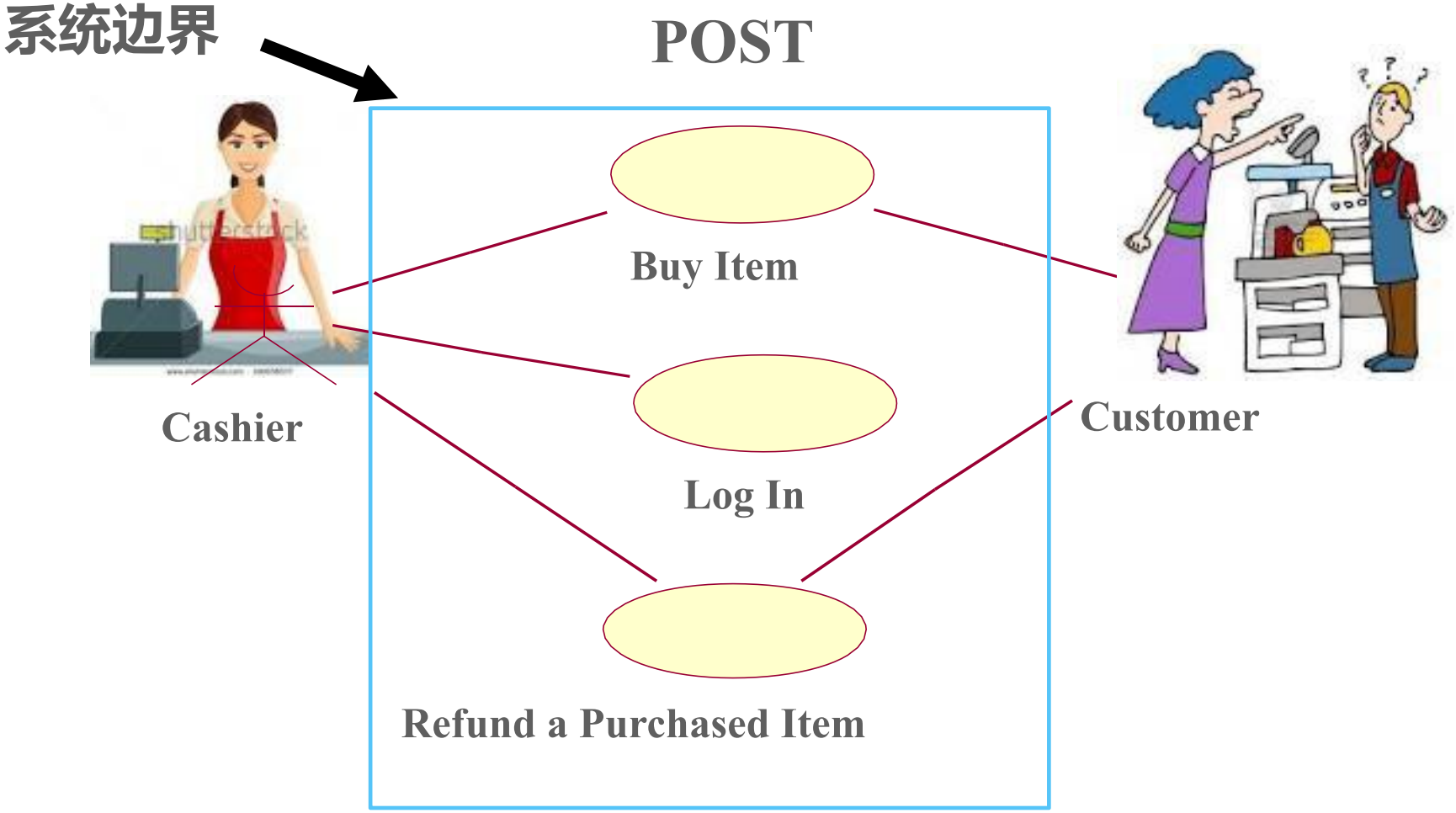
- 系统边界会对用例以及Actor的定义有所影响

考虑用于零售店销售管理的系统的用例图：

- 记录销售及付款情况的软硬件集成系统
  - 包括硬件设备，如计算机、条码扫描装置
  - 包括运行在系统上的软件
- 系统目标包括：
  - 自动收款
  - 快速准确的销售情况统计及分析
  - 自动的库存管理

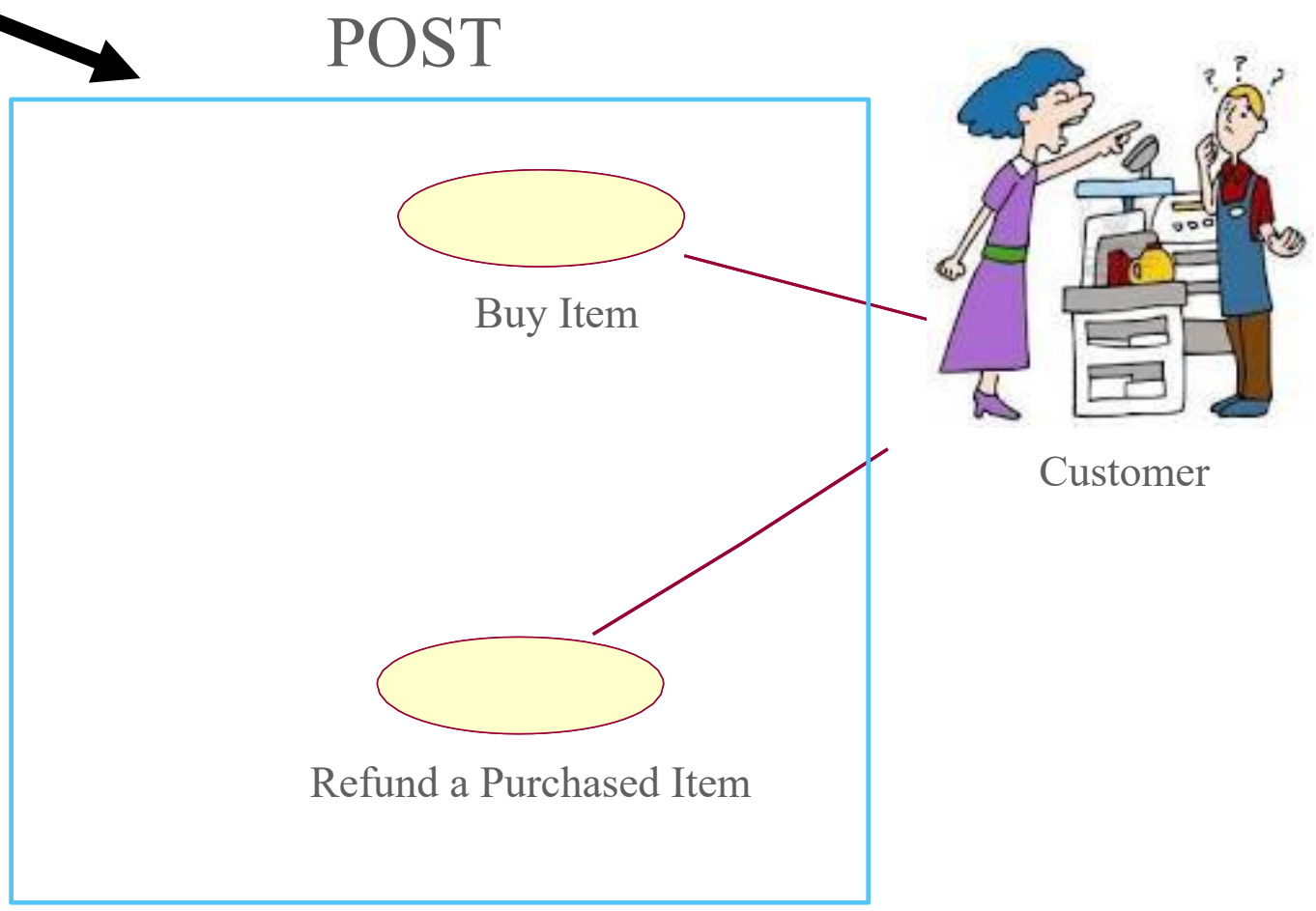


系统边界定义之一

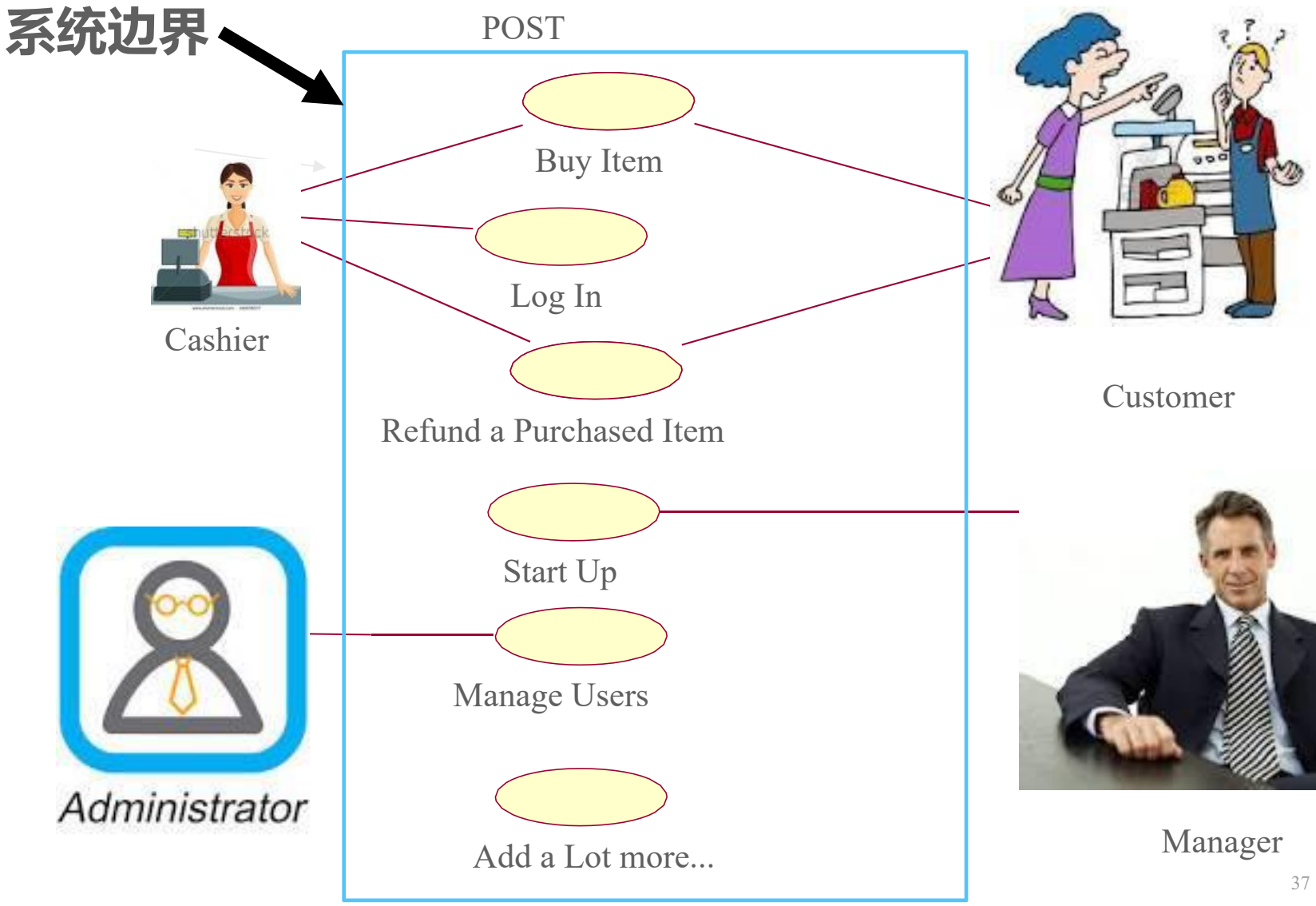

系统边界
POST
Buy Item
Log In
Refund a Purchased Item
Cashier
Customer

系统边界定义之二

系统边界



系统边界定义之三

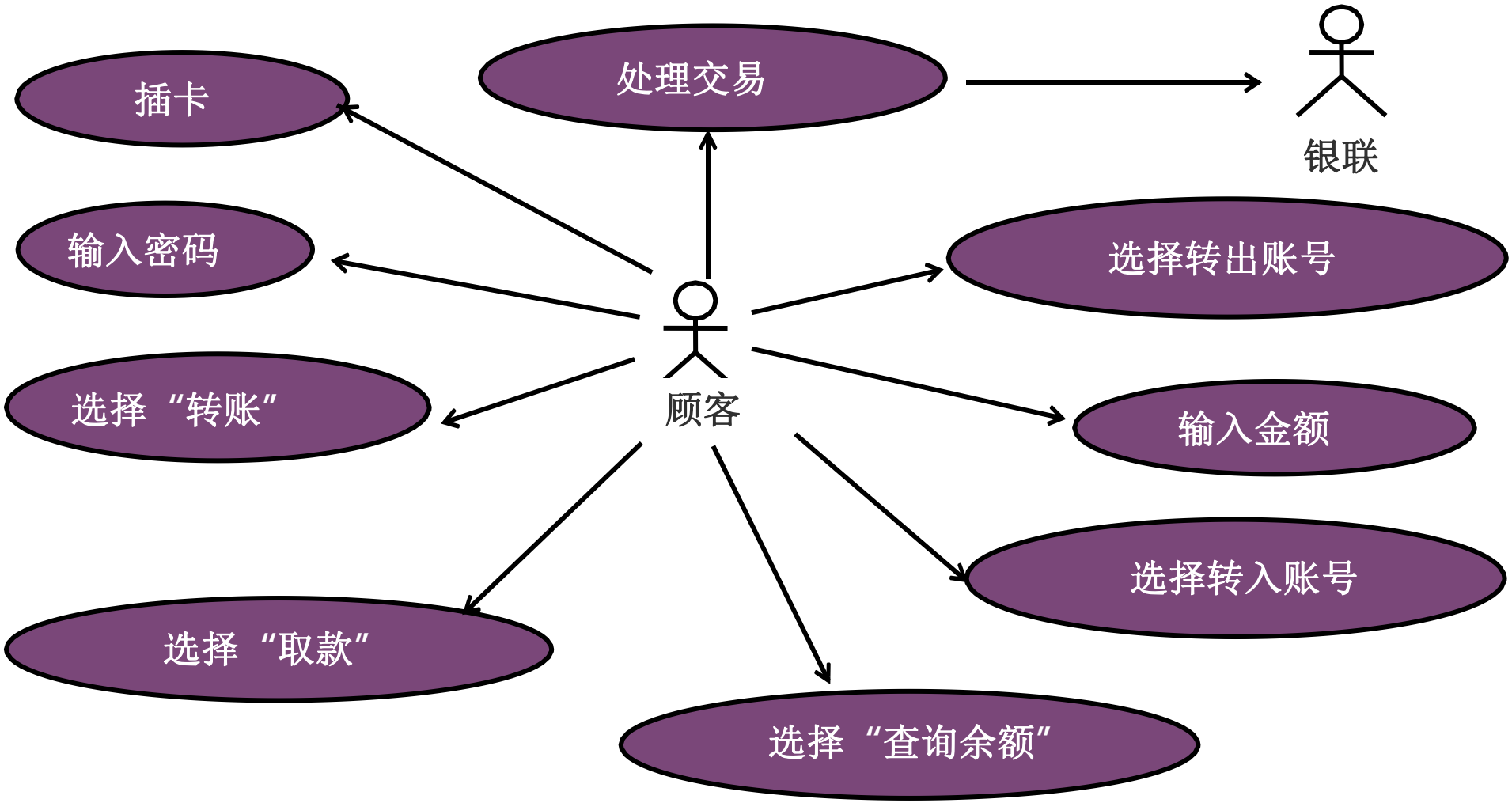

系统边界
POST
Buy Item
Log In
Refund a Purchased Item
Start Up
Manage Users
Add a Lot more...
Cashier
Customer
Administrator
Manager

不要把用例定义成功能分解

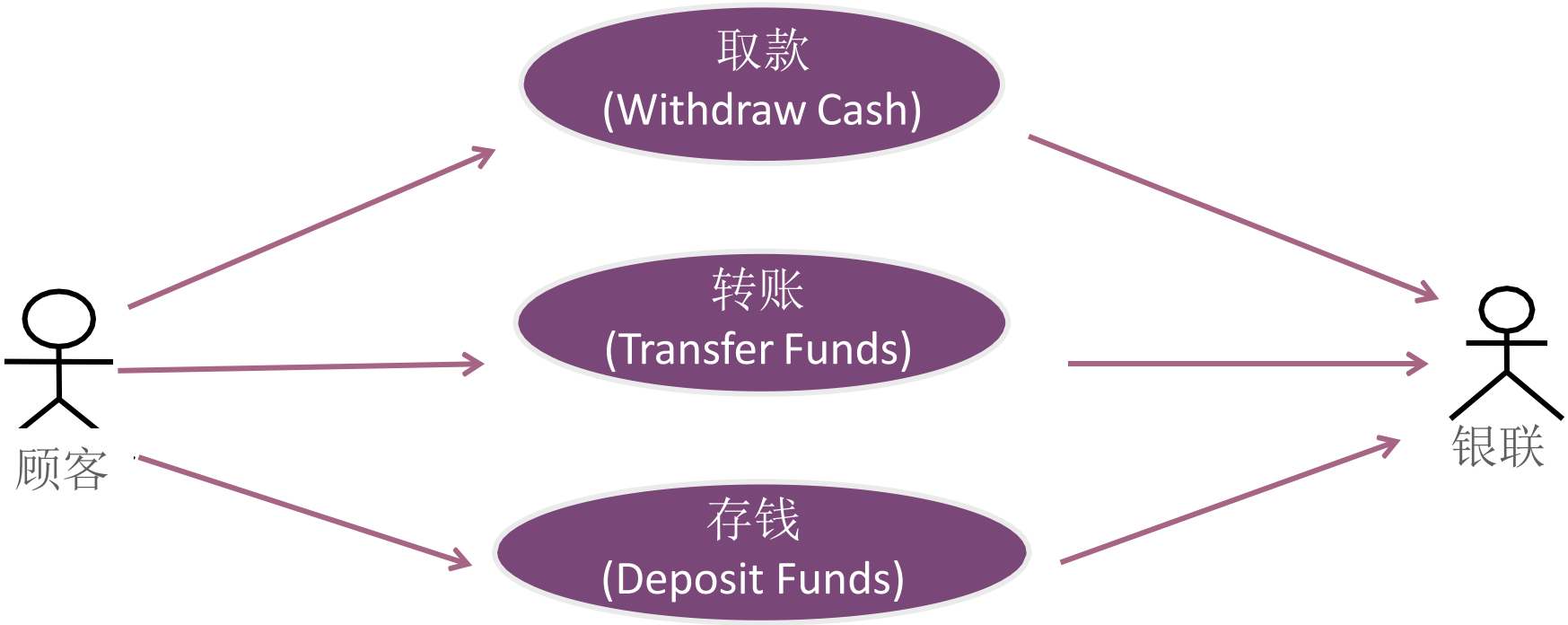
- 功能分解：将问题分解为粒度小，独立的部分。
  - 不同的模块协同工作，体现系统的功能。
  - 通常，一些功能分解并没有实际的意义。

- 用例：
  - **不是**功能分解的过程！
  - 综合所有功能一起描述系统如何使用。
  - 需要包含语境信息。

功能分解：一个例子



走出功能分解：正确的用例建模



如何避免功能性分解

问题现象

- 非常细小的用例
- 用例过多
- 没有实际价值的用例
- 通过底层操作进行命名
  - “操作”+“对象”
  - “功能”+“数据”
  - 例如：“插入卡片”

修改思路：

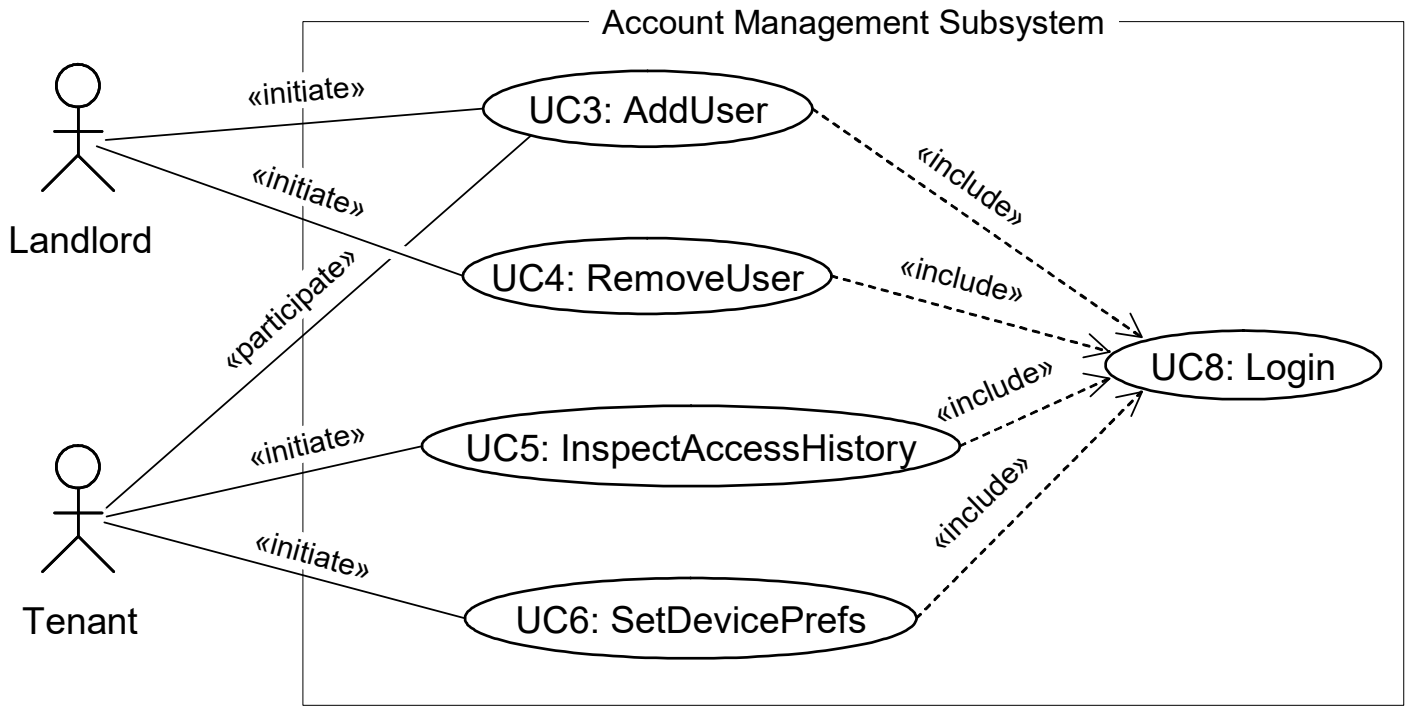
- 寻找更大的应用场景

  “为什么要构建这个系统？”

- 从一个用户的角度出发
  - “用户希望达到什么目的？”
  - “这个用例可以满足谁的目标？”
  - “这个用例的意义是什么？有什么价值？”
  - “这个用例背后的用户故事是什么？”

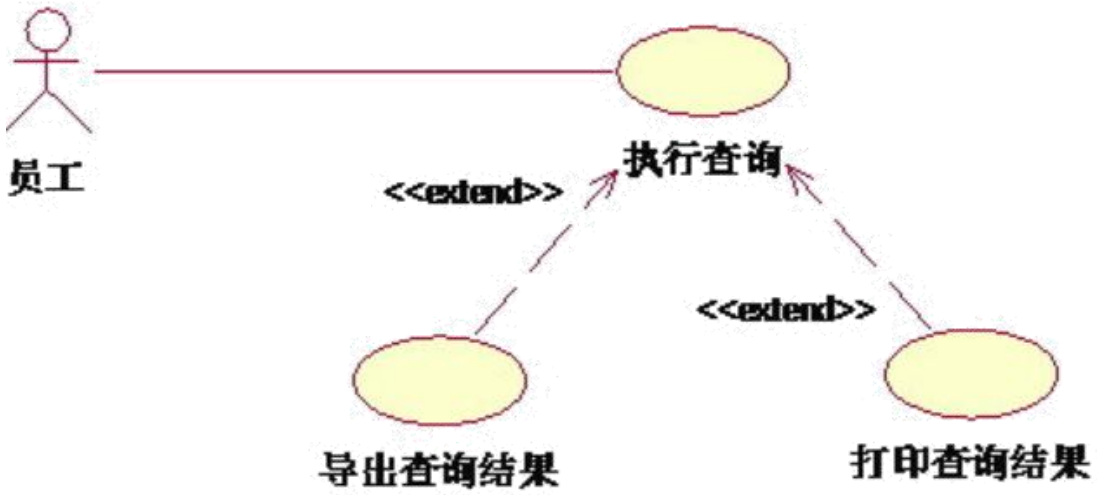
何时使用包含关系

- 当多个用例有共享行为时，使用包含关系
- 为共享行为单独创建用例，被相关用例“包含”

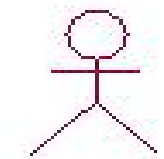


何时使用扩展关系

- 一个用例与另外一个用例近似，只有少许额外的活动
- 将代表普遍或基本行为的情况定义为一个用例
- 将特殊的、例外的部分定义为扩展用例



用例图中的主要图标



注释

注释连接

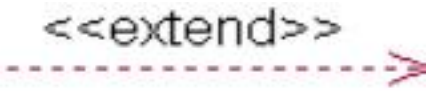
Realize

Association
关联

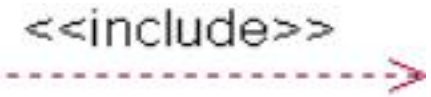
use-case realization

Generalization
泛化

Dependency
依赖



extend use case

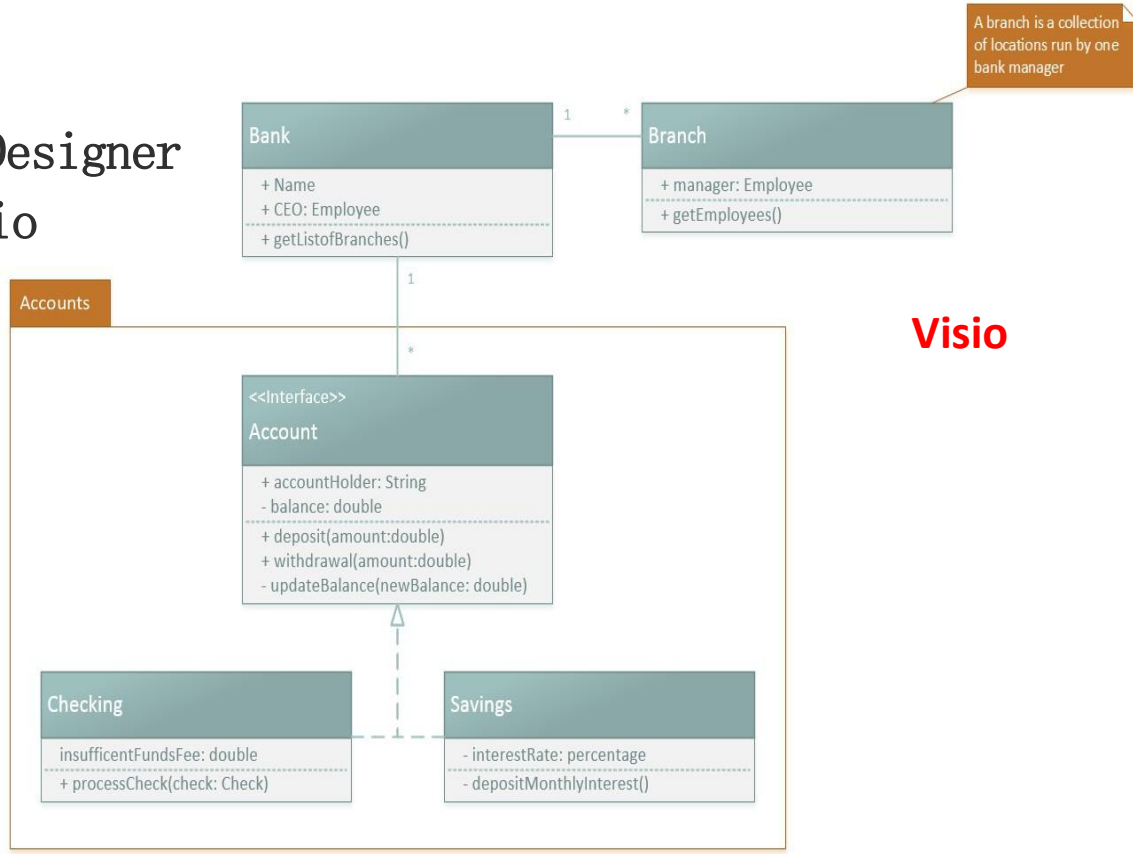


include use case

说明：UML中不使用颜色来作为图形语义的区分标记。

建模工具介绍

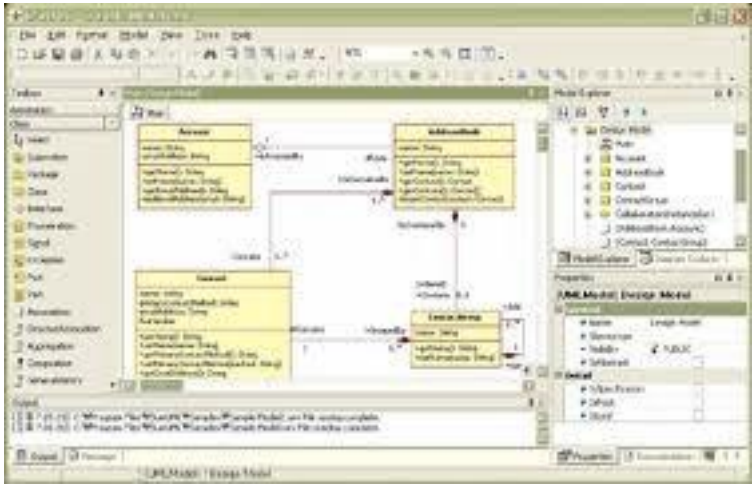
- 可视化模型表达
  - UML模型
  - Web模型，例如Azure
  - 数据库模型，例如Power Designer
  - 用户自定义模型，例如Visio



Visio

- 画图工具

StarUML



- 辅助开发流程中的项目管理

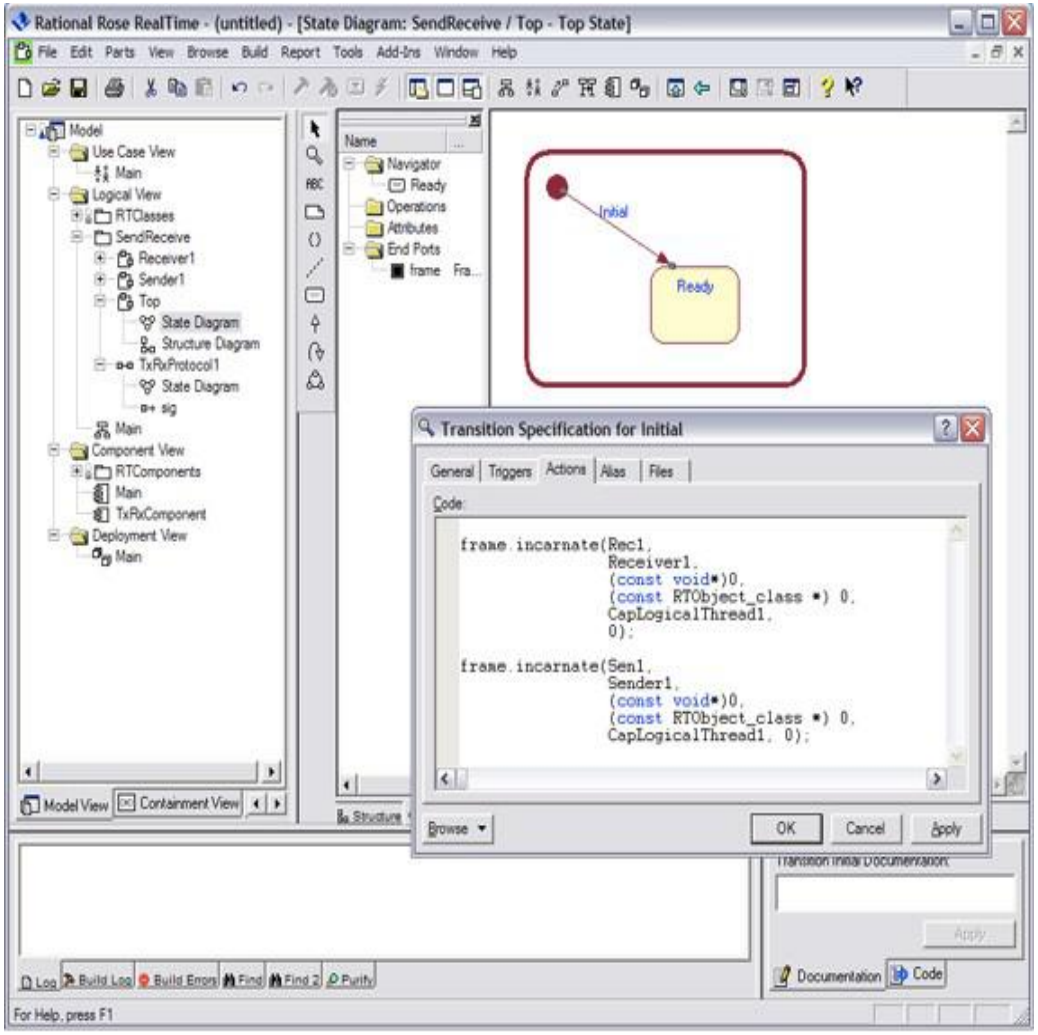
常见系统建模工具

- Caliber (version 11.4.2)， Borland (Micro Focus)
  - 应用于需求管理、建模可视化
- inteGREATE (version 8.7.13)， dDev Technologies
  - 应用于需求管理，需求开发，建模可视化
- VersionOne (version 15.0.9)， VersionOne
  - 应用于需求管理，敏捷开发
- StarUML (version 5.6.5)， MKLab
  - 应用于UML建模

资源链接：http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools
资源链接：http://makingofsoftware.com/resources/list-of-rm-tools

常用系统建模工具(UML 2.0)

- IBM Rational Rose
  - Rational是IBM设计的集成设计、建模和开发软件应用程序套件。Rose是其中的可视化建模工具
  - 适用于应用程序开发、数据建模、Web Service设计、业务建模、组件建模等
  - 支持模型与代码之间的转化，逆向工程

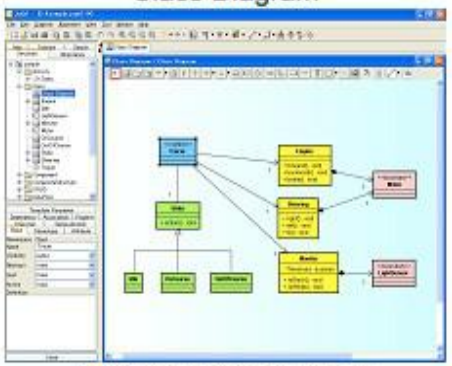


资源链接：http://www.ibm.com/developerworks/cn/rational/uml

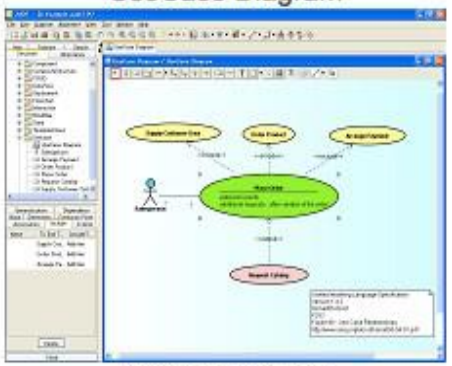
常用系统建模工具(UML 2.0)

- **JUDE**
  - 轻量级的UML建模工具
  - 支持项目团队中的协同工作
  - 文档转换
  - 全面支持UML1.4，支持UML2.0 中部分模型

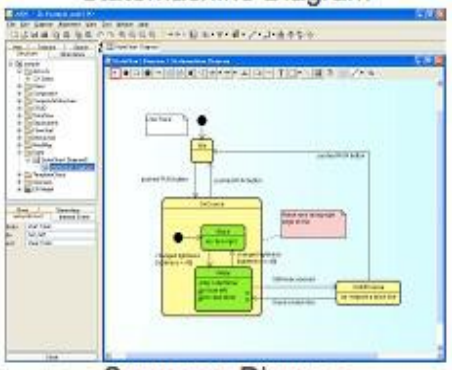
Class Diagram



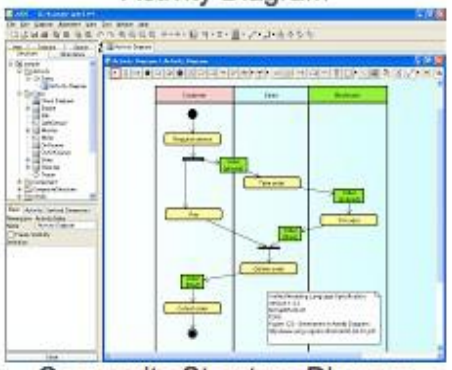
UseCase Diagram



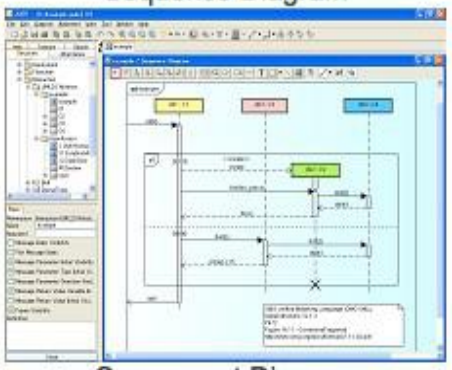
Statemachine Diagram



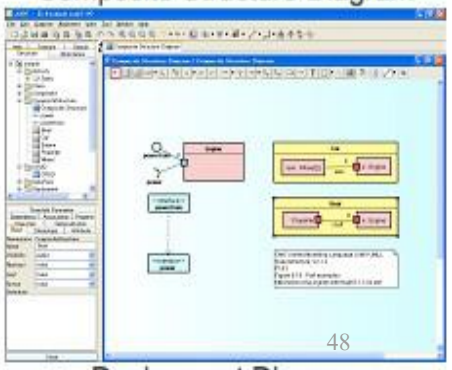
Activity Diagram



Sequence Diagram



Composite Structure Diagram

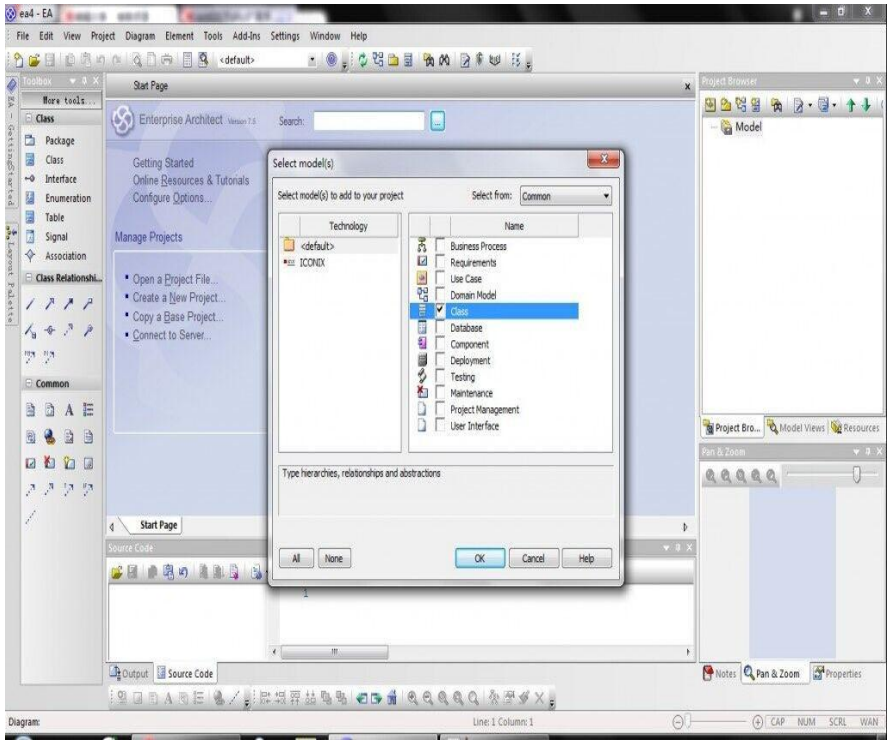


Component Diagram

Deployment Diagram

常用系统建模工具(UML 2.0)

- **Enterprise Architect（EA）**
  - 全功能的，基于**UML2.0**的可视化工具
  - 用于设计、编写、构建并管理以目标为导向的软件系统。
  - 为整个团队提供系统开发不同阶段所需的信息建模，服务于不同身份的成员
  - 提供从需求分析、软件设计一直到执行和部署整个过程的全面可跟踪性
  - 支持多种语言的前向或逆向生成代码工程



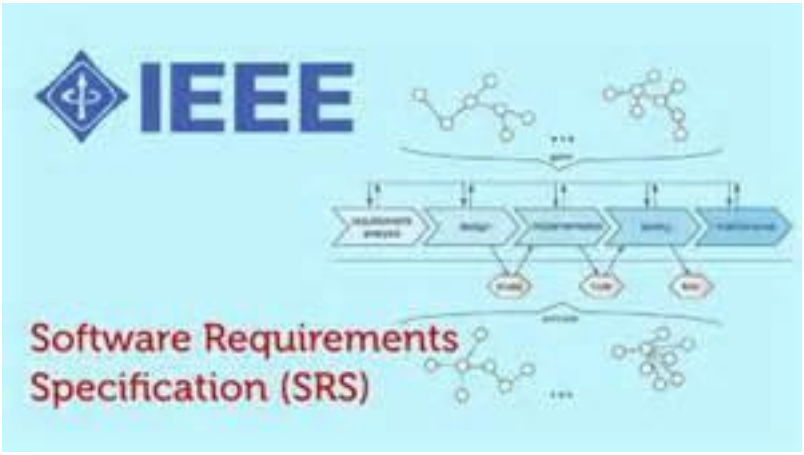
资源链接：http://www.softwarechn.com/SparxSystems/sparxsystems_index.htm

需求规格说明（Software Requirements Specification SRS）

- 是具有一定法律效力的合同文档
- 清楚地描述软件在什么情况下，需要做什么，以及不能做什么
- 以输入/输出、输入到输出之间的转换方式来描述功能性需求
- 描述经过干系人磋商达成共识的非功能性需求
- 一般参考需求定义模板，覆盖标准模板中定义的所有条目
- 作为后续的软件评估依据和变更的基准

软件需求规格说明SRS的风格

- 描述性的自然语言文本
  - 用户故事
- 从用例模型产生
  - 用例模型与需求转化可看成可逆的过程
  - 如果需求模型以用例的形式表示，我们可以逆向生成需求的完整集合



- 从需求数据库中生成
  - 商业需求数据库有内置的功能来生成经过筛选的需求规格说明
  - 从产品线需求规格数据库中生成特定产品的需求规格说明
- 从混合模型中生成
  - 特征模型和用例模型

以上两种常常通过模板完成的

用户手册作为SRS

- 撰写用户手册作为一种性价比高的一箭双雕的方法，同时获得SRS和用户手册
- 用户手册作为SRS对于和用户交互的系统是比较有效的，这样系统由交互驱动
- 好的手册描述了所有用例的所有场景
  - Fred Brooks的《人月神话》中描述了将用户手册作为需求规格说明书的做法

- 但是…用户手册并没有描述
  - 非功能性需求
  - 不和用户交互的功能性需求
    - 比如：函数计算、过滤器或者翻译工具的时候

用户手册大纲

- 介绍
  - 产品总览及基本原理
  - 术语和基本特征
  - 展示格式与报表格式的总结
  - 手册的大纲
- 开始
  - 开始指令
  - 帮助模式
  - 样例运行
- 操作模式:
  - 命令行/对话框/报告
- 高级特性
- 命令语法和系统选项

IEEE-830 SRS模板大纲

Source: *Adapted from IEEE-STD-830-1993* ***See also****, Blum 1992, p160*

- 介绍
- 术语表
- 用户需求规格说明
- 系统结构
- 系统需求规格说明
- 系统模型
- 系统的演化
- 附录
- 索引