Requirements Modelling

WALLY, DID YOU FINISH THE ANALYSIS FOR TOMORROW?
NO.
I'M WAITING UNTIL THE LAST MINUTE SO YOU WON'T HAVE TIME TO ASK FOR UNNECESSARY CHANGES.
I'M A STEP AHEAD OF HIM — THE ANALYSIS ITSELF IS UNNECESSARY.
scottadams@aol.com
www.dilbert.com
© 1999 United Feature Syndicate, Inc.

统一建模方法

Unified Modelling Language

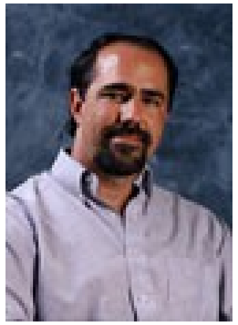
清华大学软件学院 刘璘

What is the UML ？

- UML stands for **Unified Modeling Language**.
- The UML is a language for
  - visualizing （可视化）
  - specifying （详述）
  - constructing （构造）
  - documenting （文档化）

  the artifacts of a software-intensive system.

UML is the de facto standard notation for software design and analysis…


Booch

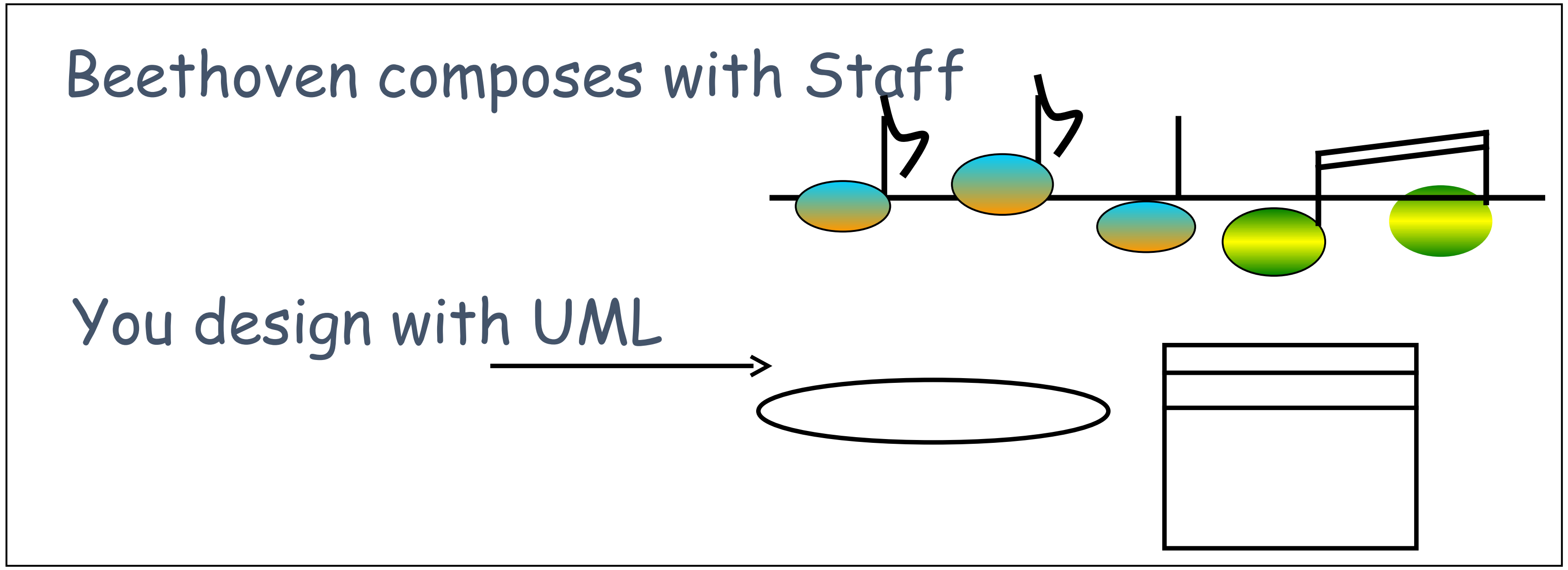
Rumbaugh

Jacobson

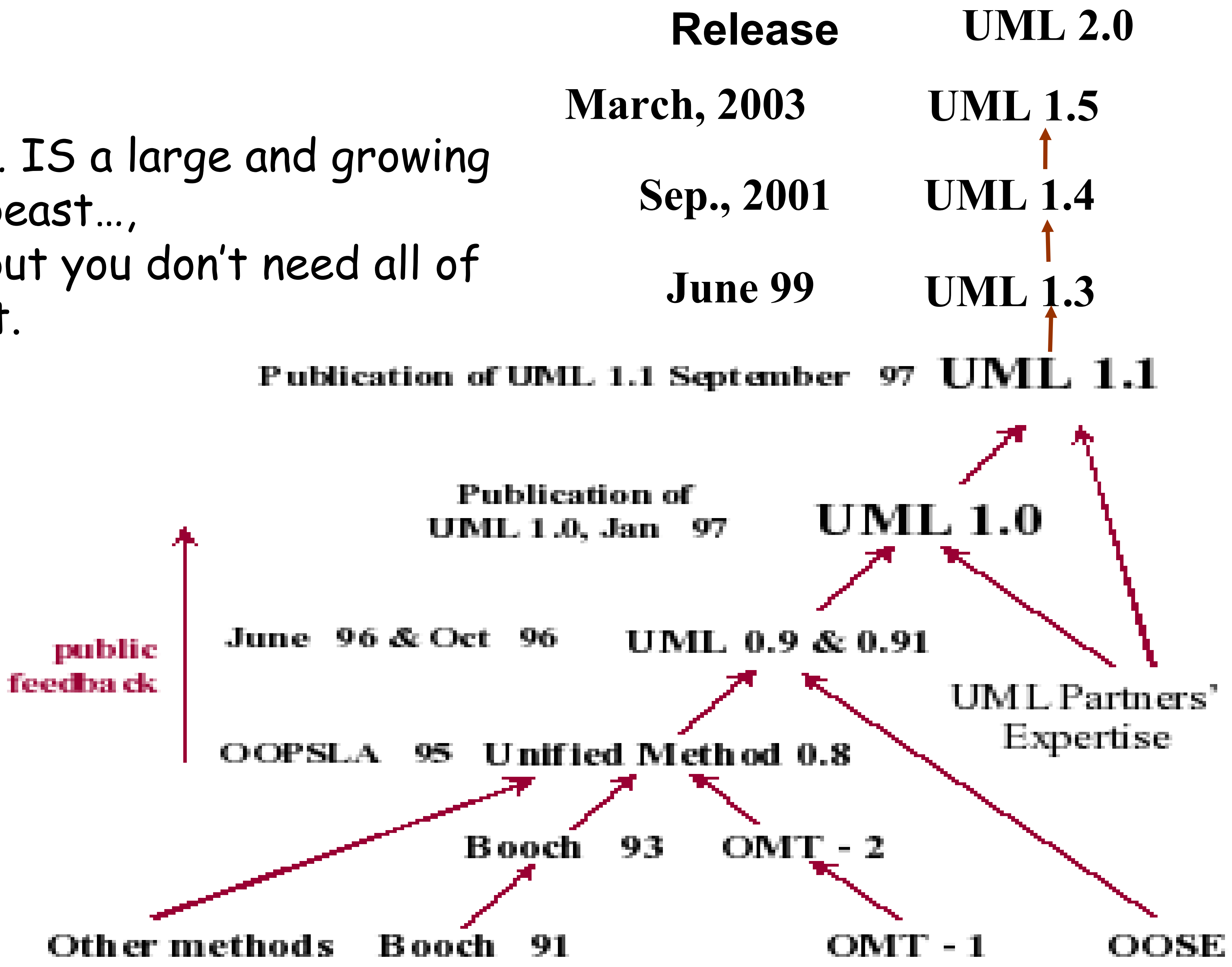
UNIFIED MODELING LANGUAGE

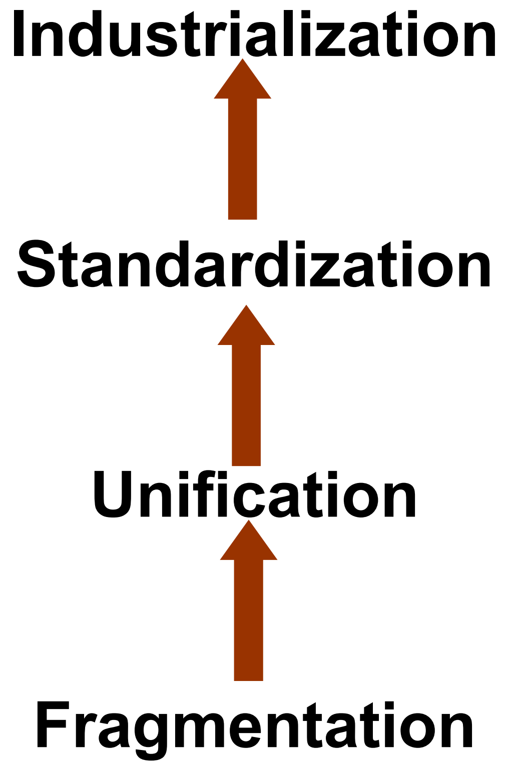
Relationships to programming languages

- Java， C++ are programming languages to implement a system.
- UML is a modeling language to describe a system or its design.
- Some CASE tools can generate Java， C++ programs based on UML models.

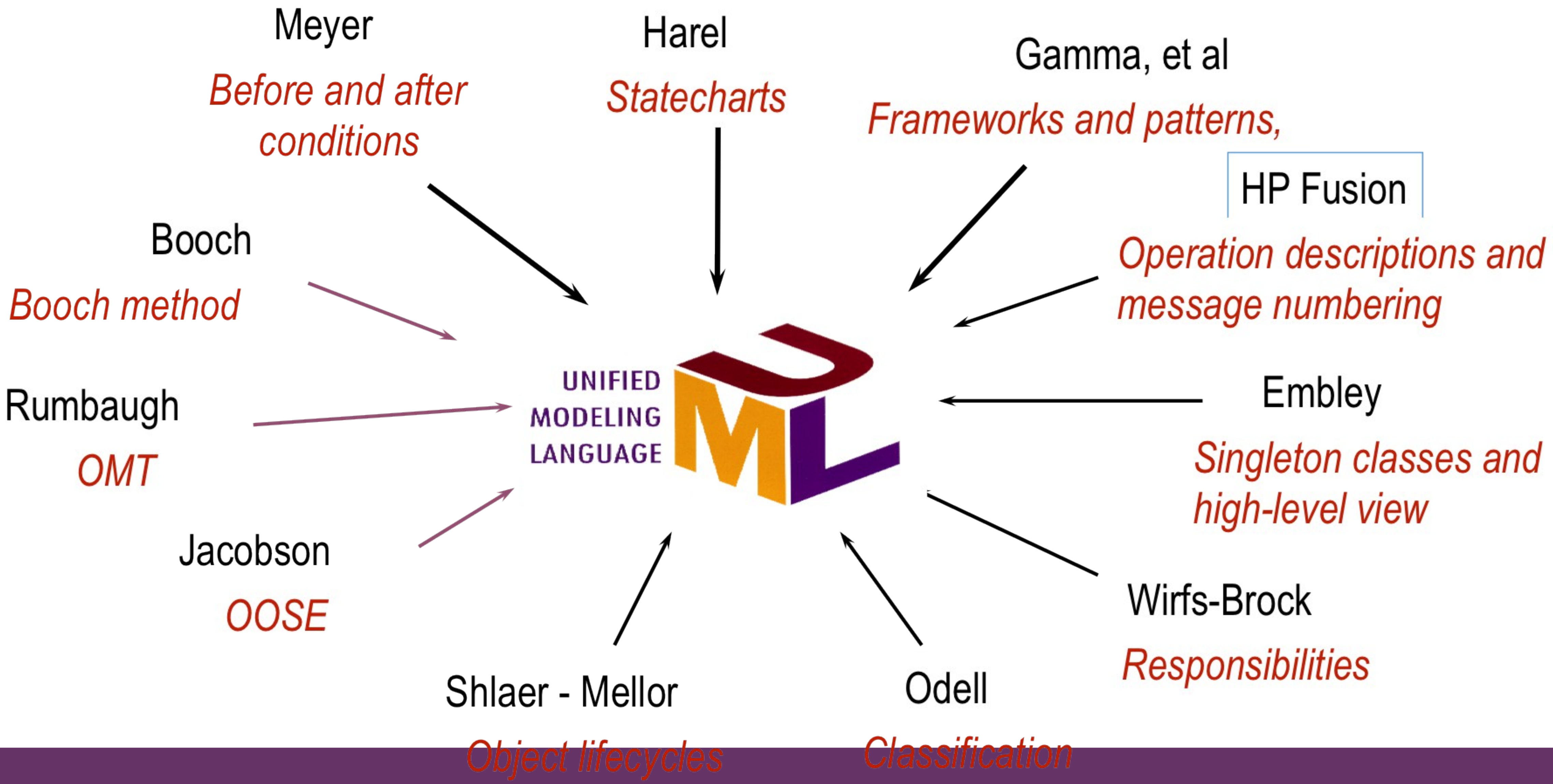


… IS a large and growing beast…,
but you don't need all of it.





Contributions to the UML



UML constructs

- 基本构造块 (basic building blocks)
  - 事物 （ things ）
    - 结构事物 （ structural things ）
      - class, interface, collaboration, use case, active class, component, node
    - 行为事物 （ behavioral things ）
      - interaction, state machine
    - 分组事物（ grouping things ）
      - package
    - 注释事物 （ annotation things ）
      - note
  - 关系 （ relationships ）
    - 依赖 （ dependency ）
    - 关联 （ association ）
    - 泛化 （ generalization ）
    - 实现 （ realization ）
  - 图 （ diagrams ）
- 规则 (rules)
- 公共机制 (common mechanisms)

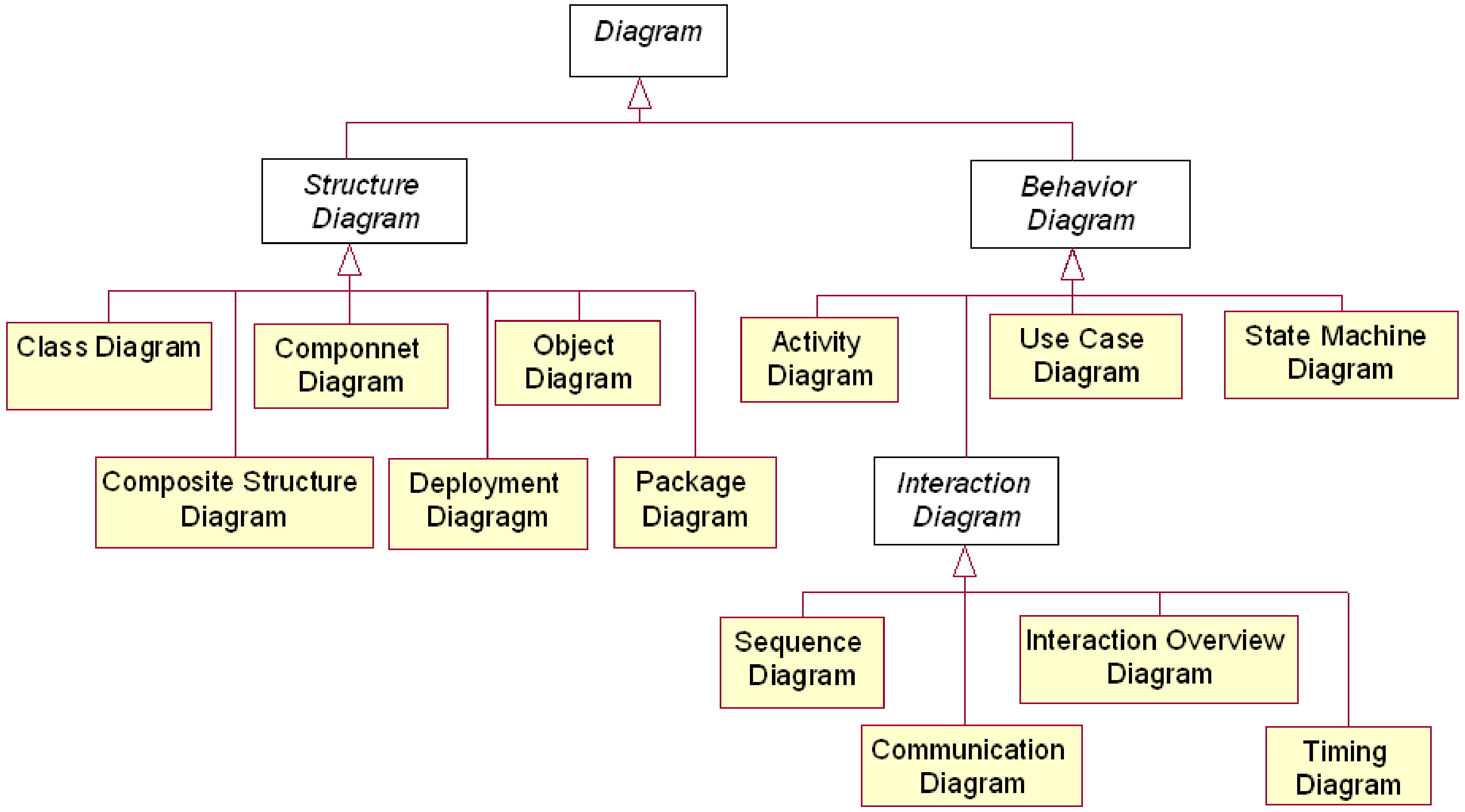
… UML Diagrams

- UML was conceived as a language for modeling software. Since this includes requirements, UML supports world modeling (...at least to some extent).

- UML offers a variety of diagrammatic notations for modeling static and dynamic aspects of an application.

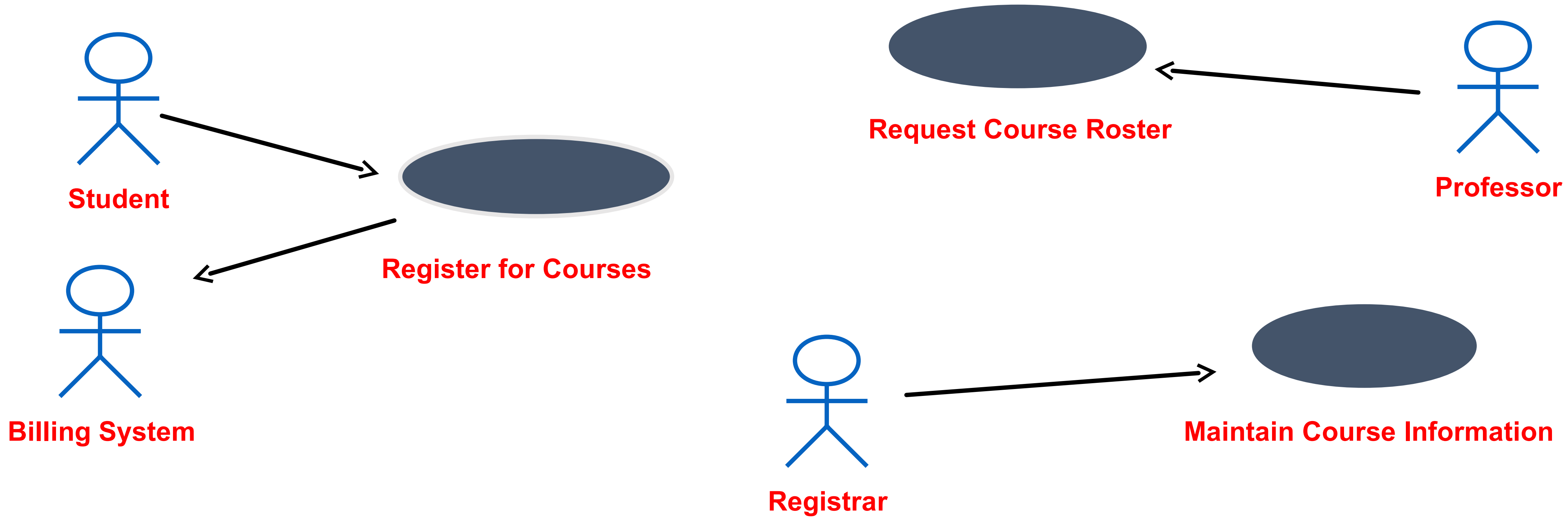
- The list of notations includes use case diagrams, class diagrams, interaction diagrams – describe sequences of events, package diagrams, activity diagrams, state diagrams, … more...

Classification of Diagrams in the UML 2.0



Use-Case Diagram

- A use-case diagram is created to visualize the interaction of your system with the outside world.



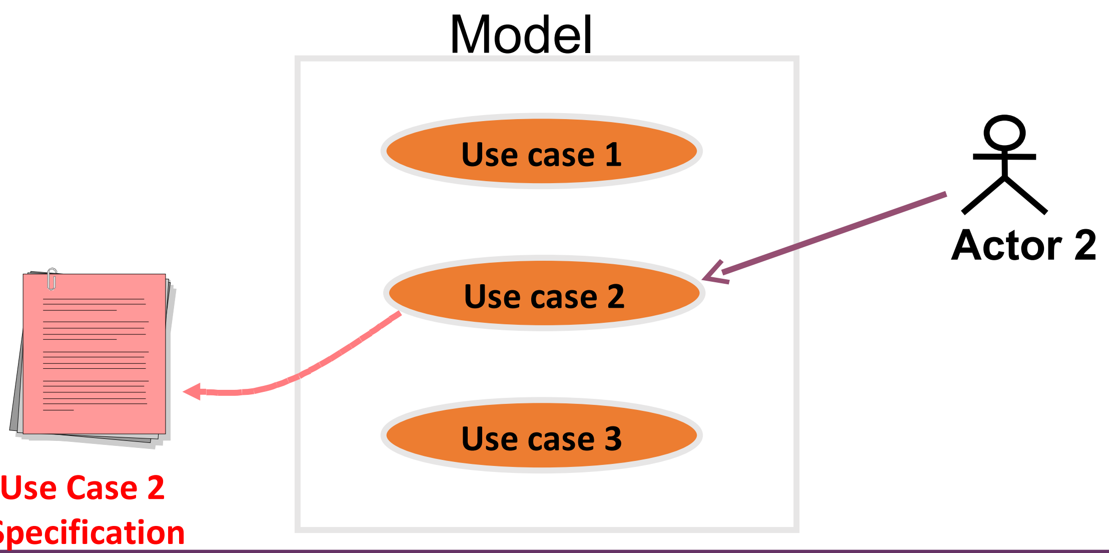
情景驱动的需求方法 – 用例建模

Unified Modelling Language

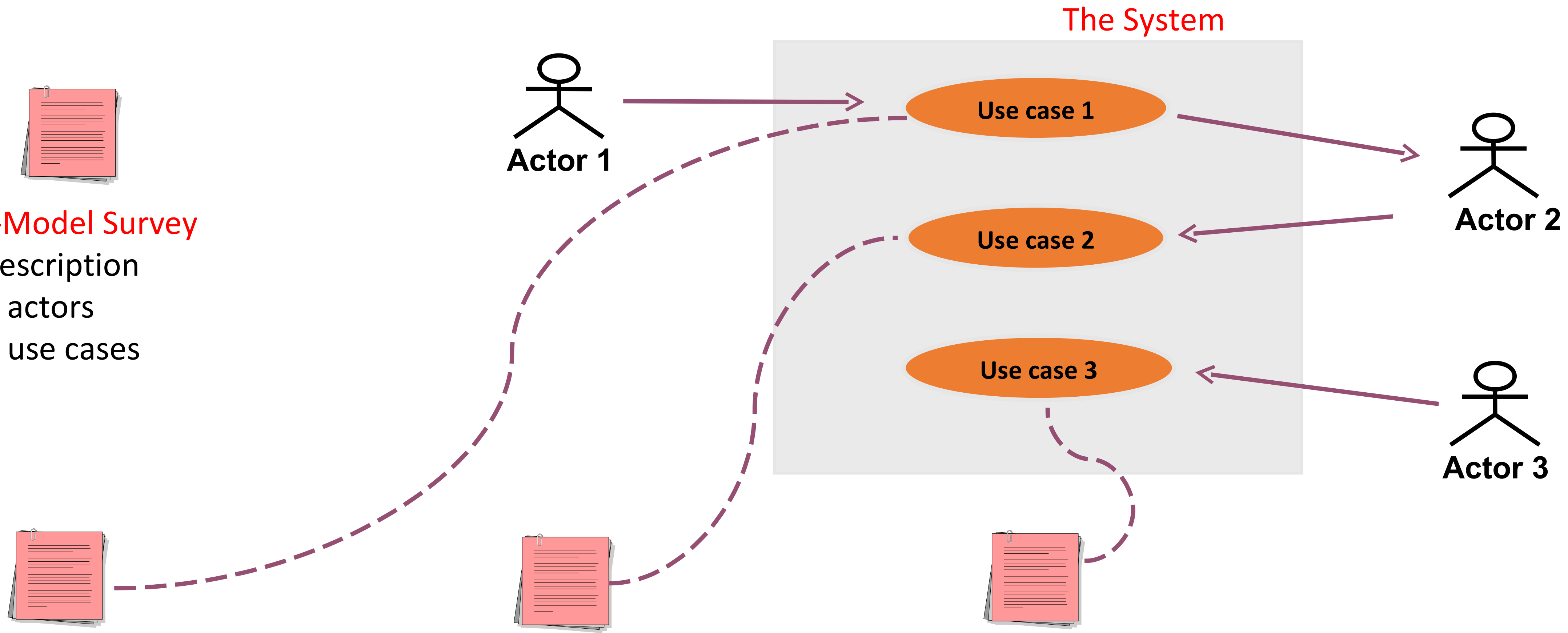
清华大学软件学院 刘璘

What Is Use-Case Modeling?

- Links stakeholder needs to software requirements.
- Defines clear boundaries of a system.
- Captures and communicates the desired behavior of the system.
- Identifies who or what interacts with the system.
- Validates/verifies requirements.
- Is a planning instrument.



A Use-Case Model is Mostly Text



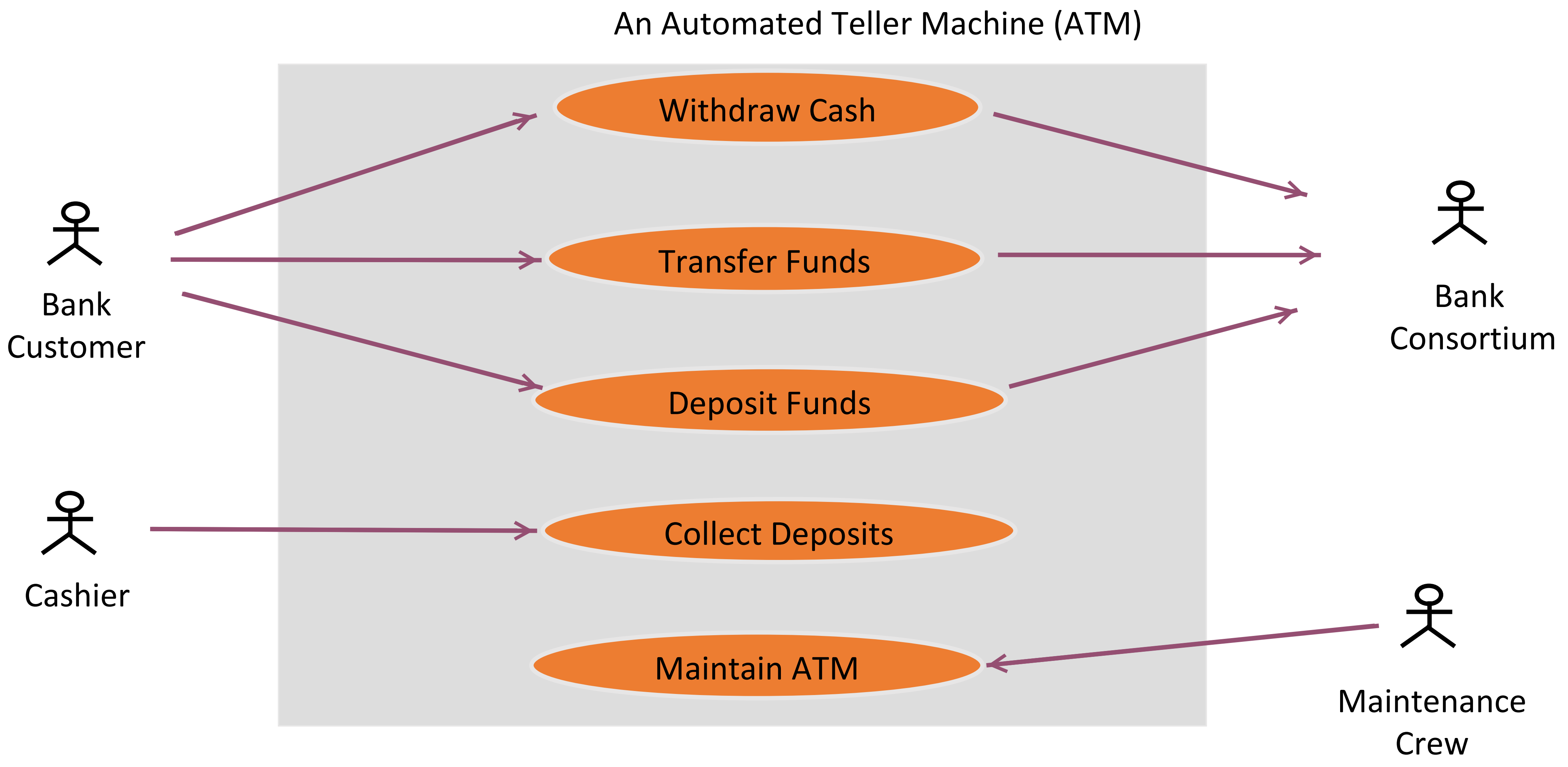
Use-Case-Model Survey
- survey description
- list of all actors
- list of all use cases

Use-Case 1 Spec
- brief description
- flow of events

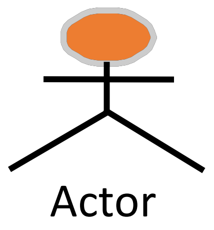
Use-Case 2 Spec
- brief description
- flow of events

Use-Case 3 Spec
- brief description
- flow of events

Use-Case Diagram

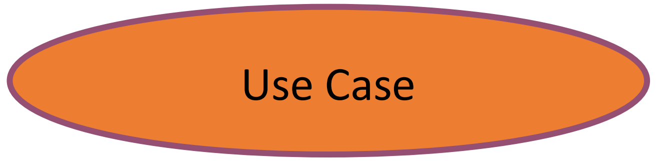


Major Use-Case Modeling Elements



Actor

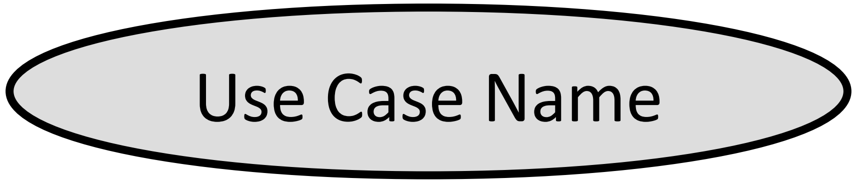
Someone/something outside the system, acting in a role that interacts with the system



Use case

Represents something of value that the system does for its actors

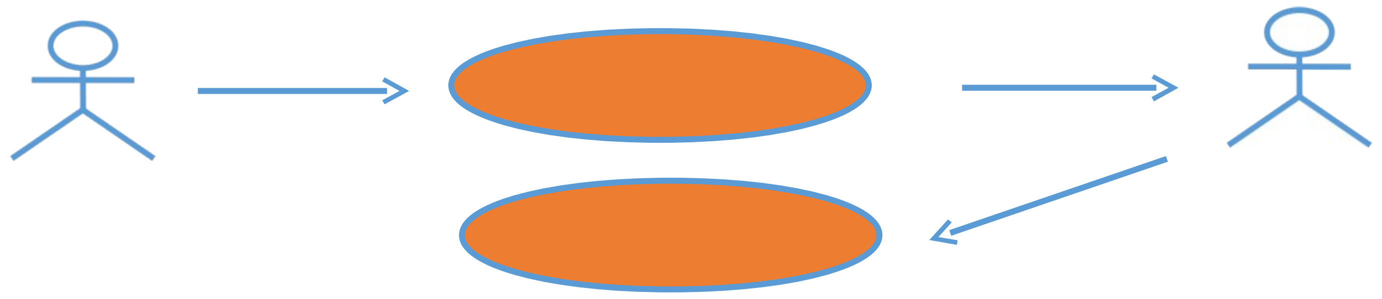
What Is a Use Case?



A use case defines a sequence of actions
performed by a system
that yields an observable result of value
to an actor.

Use Cases Contain Software Requirements

- Each use case
  - Describes actions the system takes to deliver something of value to an actor.
  - Shows the system functionality an actor uses.
  - Models a dialog between the system and actors.
  - Is a complete and meaningful flow of events from the perspective of a particular actor.

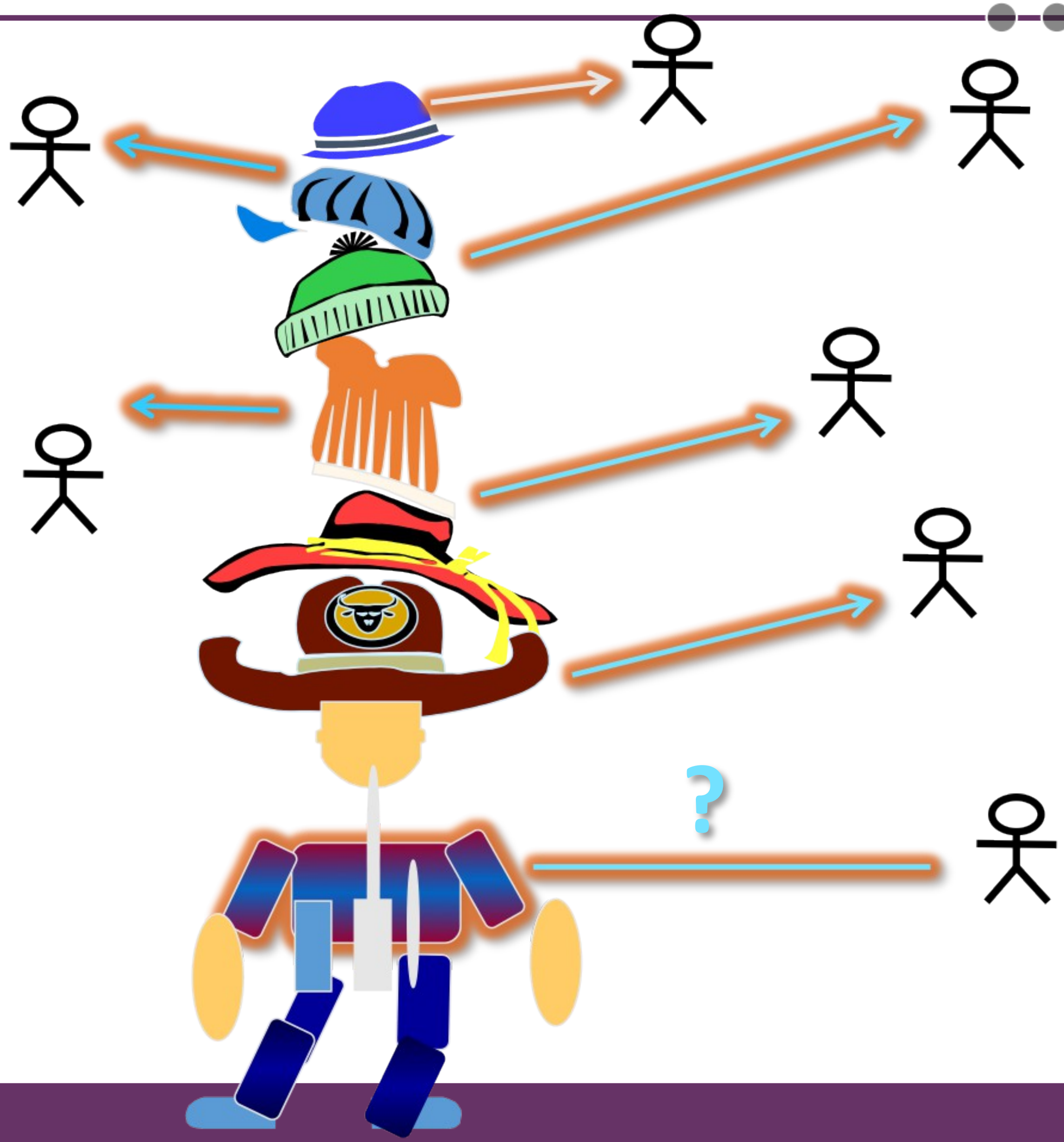


Benefits of Use Cases

- Give context for requirements.
  - Put system requirements in logical sequences.
  - Illustrate why the system is needed.
  - Help verify that all requirements are captured.
- Are easy to understand.
  - Use terminology that customers and users understand.
  - Tell concrete stories of system use.
  - Verify stakeholder understanding.
- Facilitate agreement with customers.
- Facilitate reuse: test, documentation, and design.

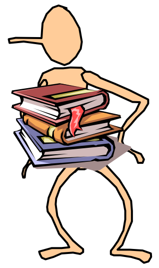
Define Actors: Focus on the Roles

- An actor represents a role that a human, hardware device, or another system can play in relation to the system.
- Actor names should clearly denote the actor's role.



Actors and Roles

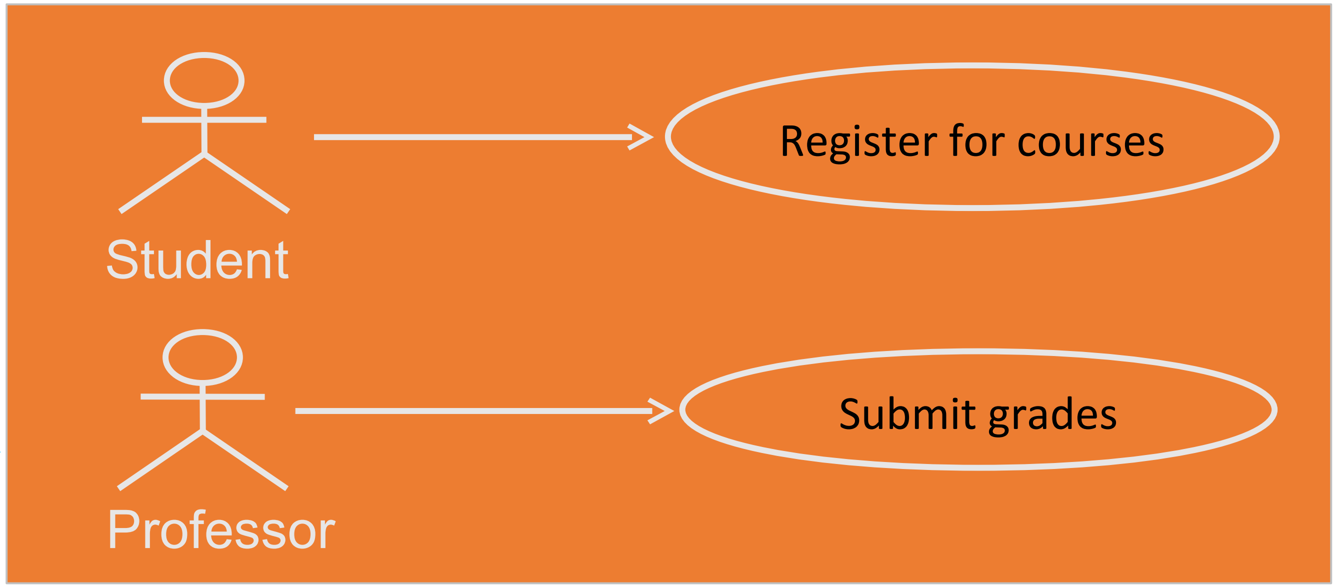
Charlie: Is employed as a math professor and is an economics undergraduate.



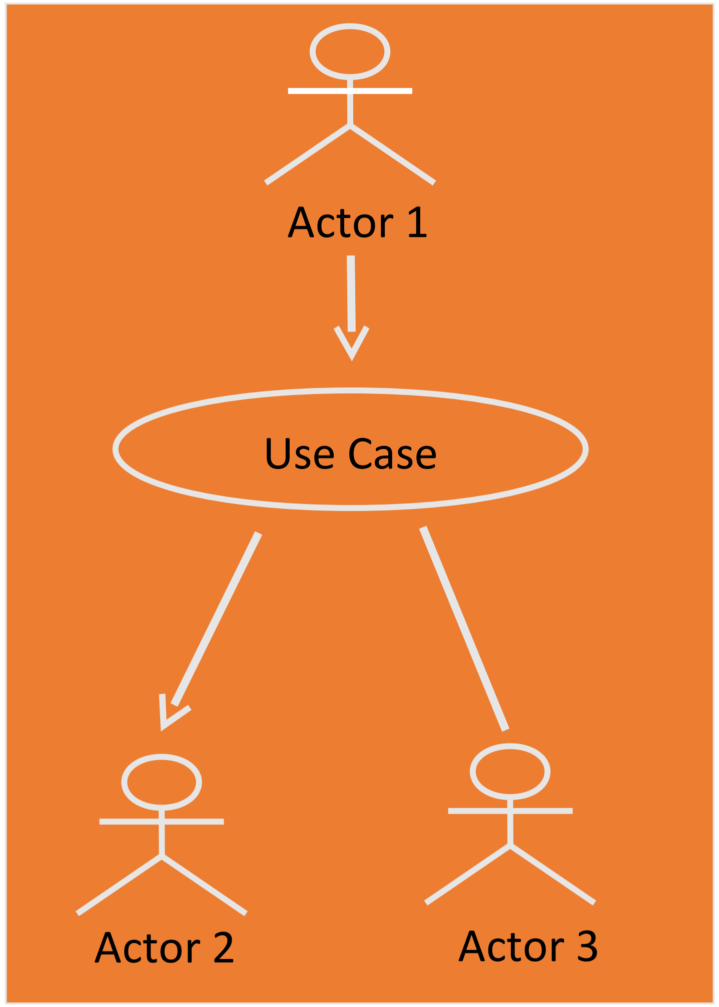
Jodie: Is a science undergraduate.

Charlie and Jodie both act as a Student.

Charlie also acts as a Professor.

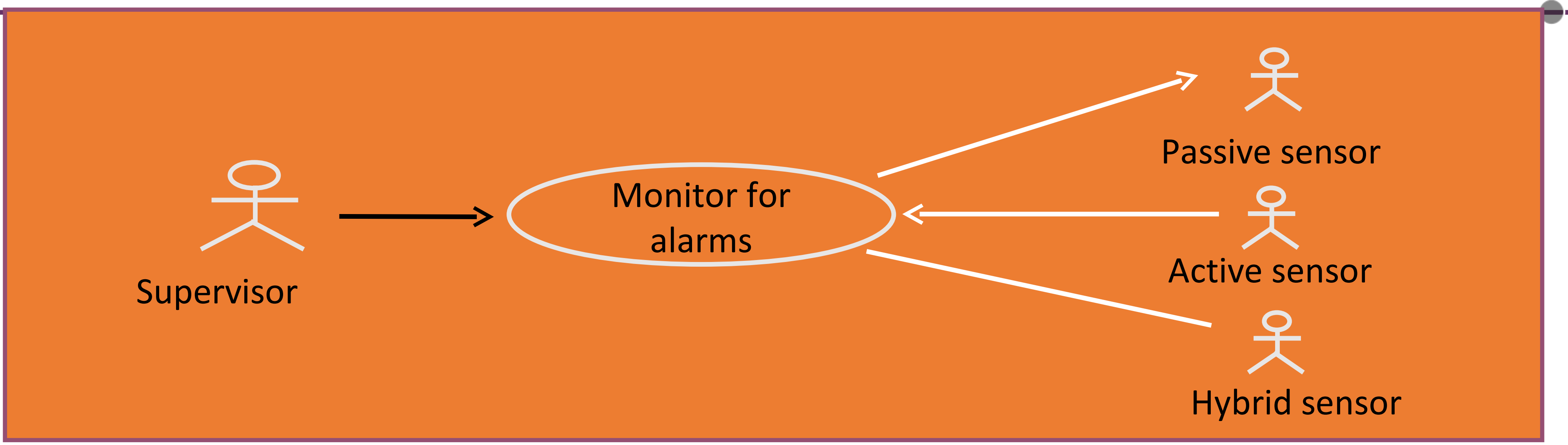


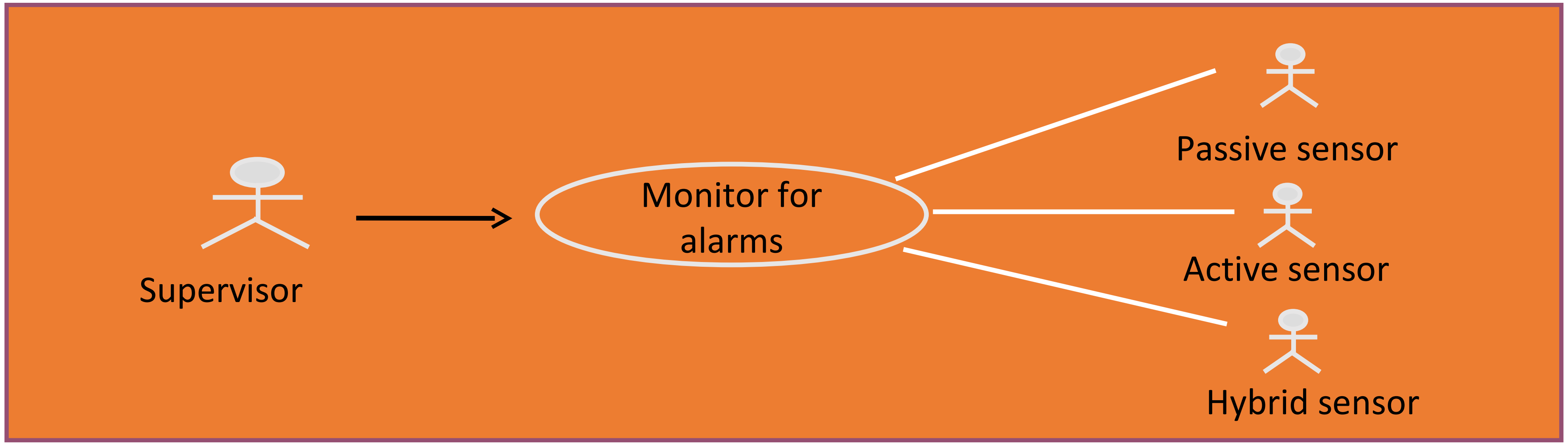
Communicates-Association



- A channel of communication between an actor and a use case.
- A line is used to represent a communicates-association.
    - An arrowhead indicates who initiates each interaction.
    - No arrowhead indicates either end **can** initiate each interaction.

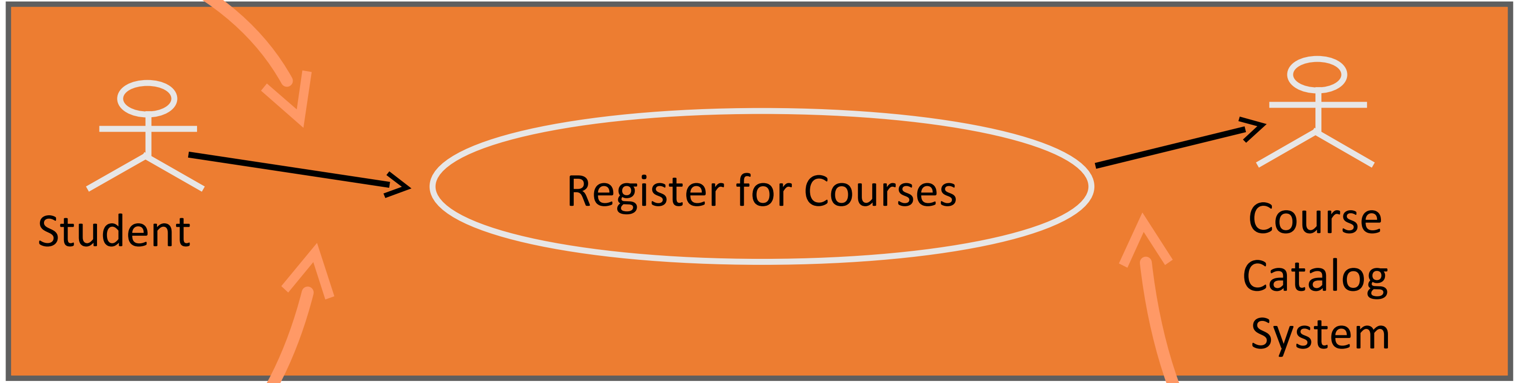
Arrowhead Conventions





Each Communicates-Association Is a Whole Dialog

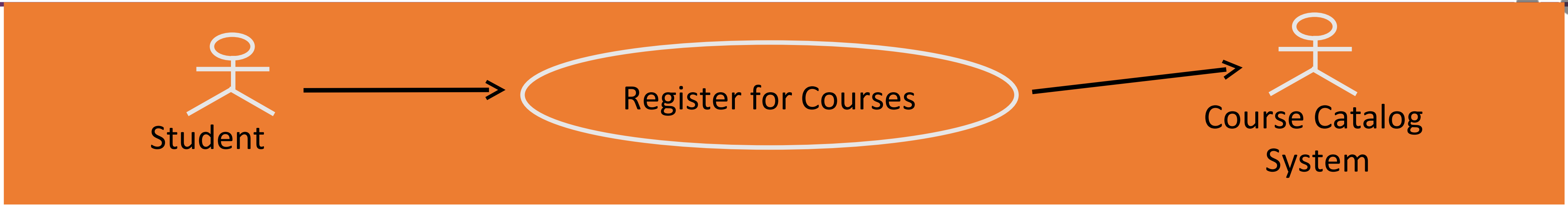
Student logs on to system.
System approves log on.
Student requests course info.



System displays course list.
Student select courses.
System displays approved schedule.

System transmits request.
Course Catalog returns course info.

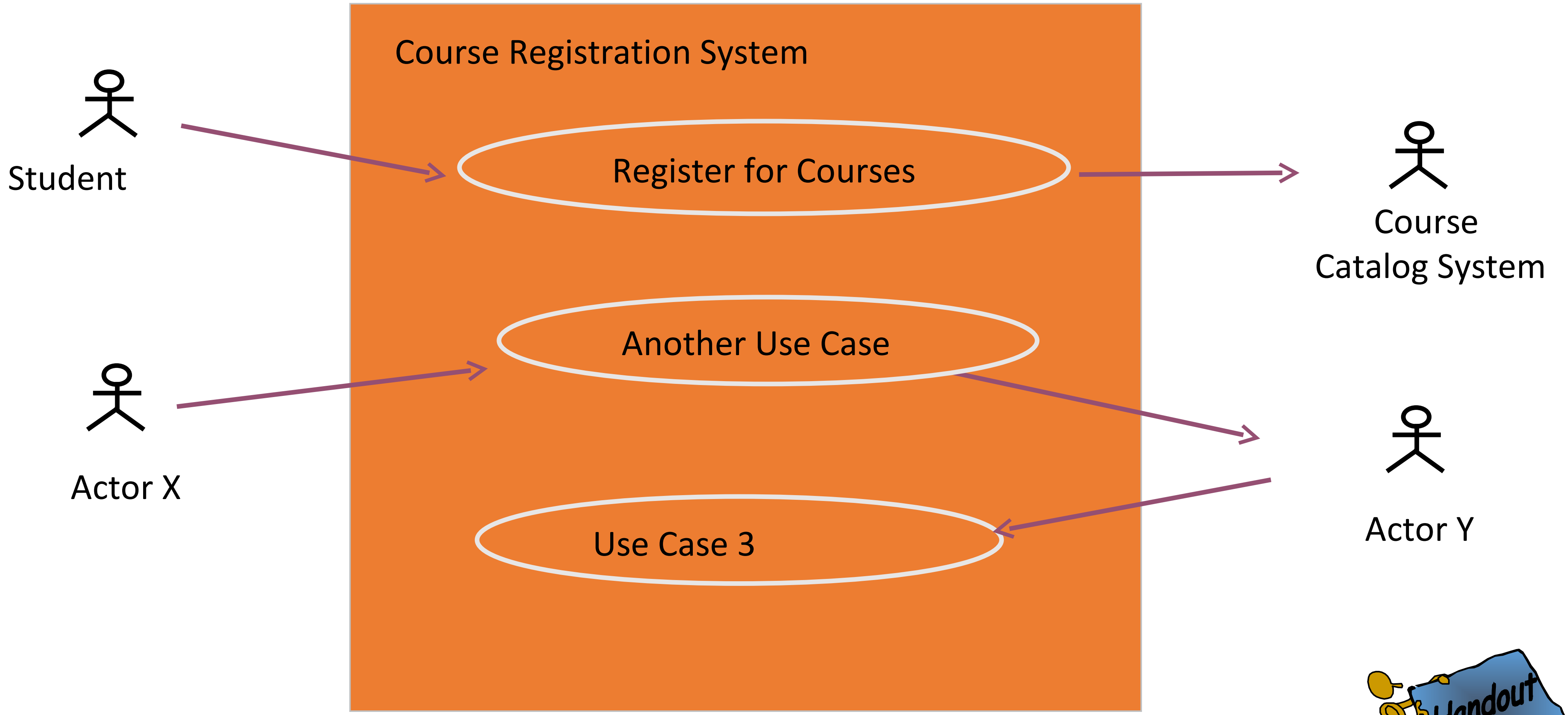
A Scenario Is a Use-Case Instance



<u>Scenario 1</u>
Log on to system.
Approve log on.
Enter subject in search.
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

<u>Scenario 2</u>
Log on to system.
Approve log on.
Enter subject in search.
Invalid subject.
Re-enter subject.
Get course list.
Display course list.
Select courses.
Confirm availability.
Display final schedule.

Example: Online Course Registration System



Course Registration
System Artifacts

How Should I Name a Use Case?

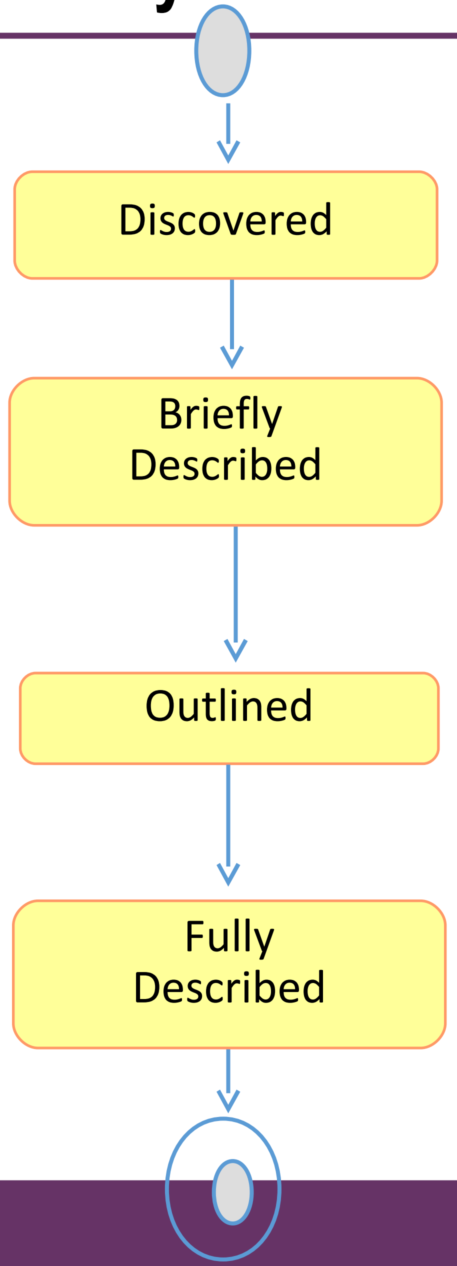
- Indicate the value or goal of the actor.
- Use the active form; begin with a verb.
- Imagine a to-do list.
- Examples of variations
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration
  - Use Registration System

Which variations show the value to the actor? Which do not?
Which would you choose as the use-case name? Why?

Steps for Creating a Use-Case Model

- Find actors and use cases.
  - Identify and briefly describe actors.
  - Identify and briefly describe use cases.
- Write the use cases.
  - Outline all use cases.
  - Prioritize the use-case flows.
  - Detail the flows in order of priority.

Lifecycle of a Use Case



Brief description: This use case allows a Registrar to close the registration process. Course offerings that do not have enough students are cancelled. The Billing System is notified for each student in each course offering that is not cancelled, so the student can be billed for the course offering.

Close Registration *Outline*
-Flow of events
    -Step-by-Step

Close Registration *Use-Case Specification*
-Detailed Flow of Events
Special Requirements
-Pre/Post Conditions

简洁用例的例子 - Briefly Described Use Cases

- 简洁用例：

  简洁的一段摘要，主要是成功场景

- 处理销售：

  客户带着要购买的货物到达收款处，出纳员使用 POS 系统记录每一个购买的货物。系统提供总价和详细条目。客户输入付款信息供系统验证并记录。系统更新库存，客户得到收条并带着货物离开。

临时用例的例子 - Outlined Use Cases Example

- 临时用例：
  - 非正式、随意的格式
  - 非正式段落，覆盖各种场景

退货处理

主要成功场景：

客户带着要退的货物到达收款处，出纳员使用 POS 系统记录每一个要退货的货物 ,...

候选场景：

若信用验证失败，通知客户并要求使用其他付款方法

若系统检测到与外界计税系统通信失败 , ...

详细用例的例子 - Fully Described Use Cases Example

Use Case Description

Name: Place Order

Precondition: A valid user has logged into the system.

Description:

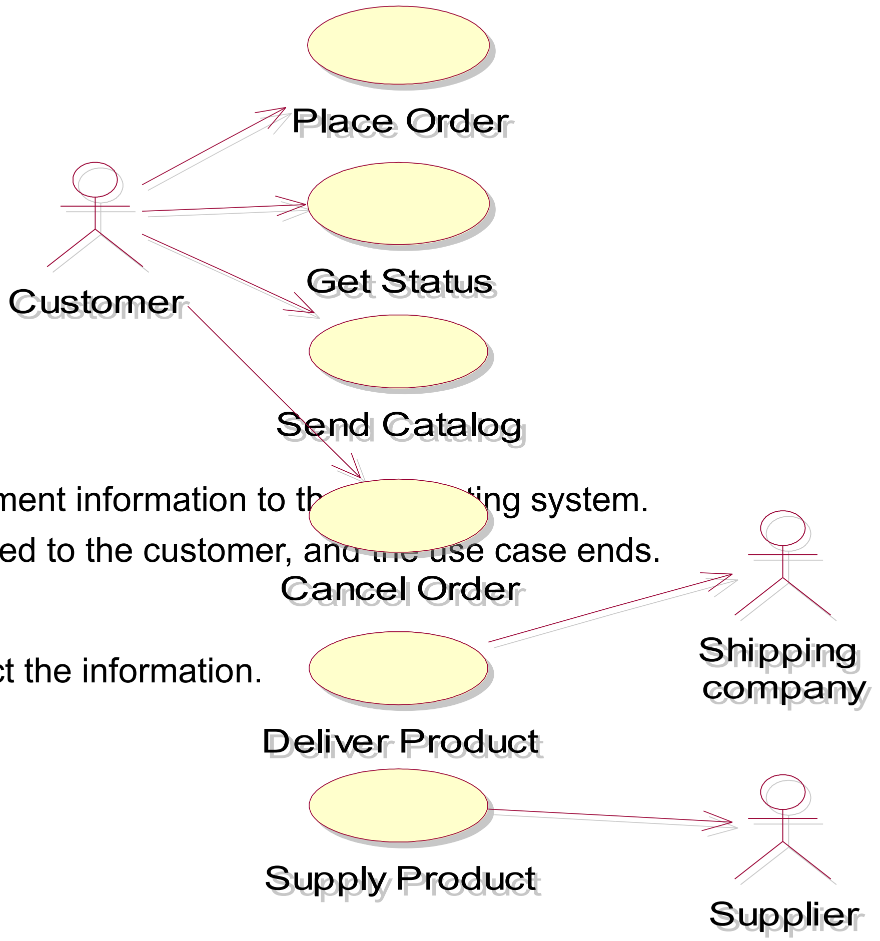
1. The use case starts when the customer selects Place Order.
2. The customer enters his or her name and address.
3. If the customer enters only the zip code, the system will supply the city & state.
4. The customer will enter product codes for the desired products.
5. The system will supply a product description and price for each item.
6. The system will keep a running total of items ordered as they are entered.
7. The customer will enter credit card payment information.
8. The customer will select Submit.
9. The system will verify the information, save the order as pending, and forward payment information to th ing system.
10. When payment is confirmed, the order is marked Confirmed, an order ID is returned to the customer, and the use case ends.

Exceptions:

In step 9, if any information is incorrect, the system will prompt the customer to correct the information.

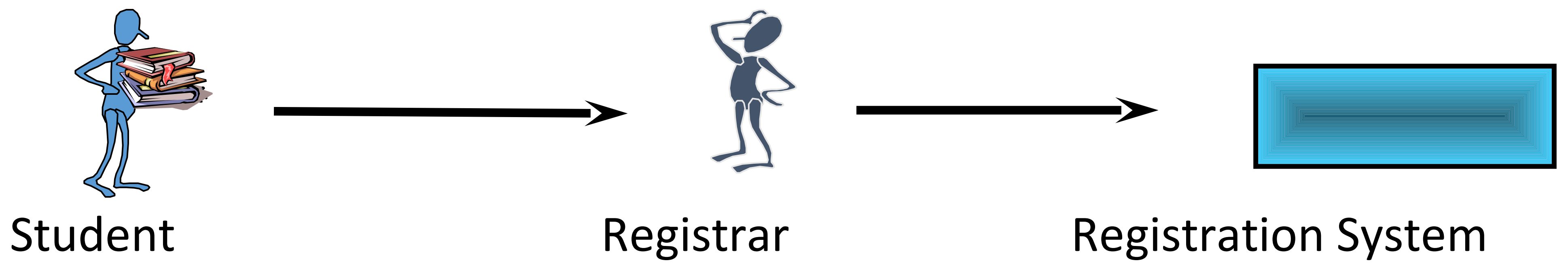
Postcondition: The order has been saved in the system and marked confirmed.

Use Case Diagram

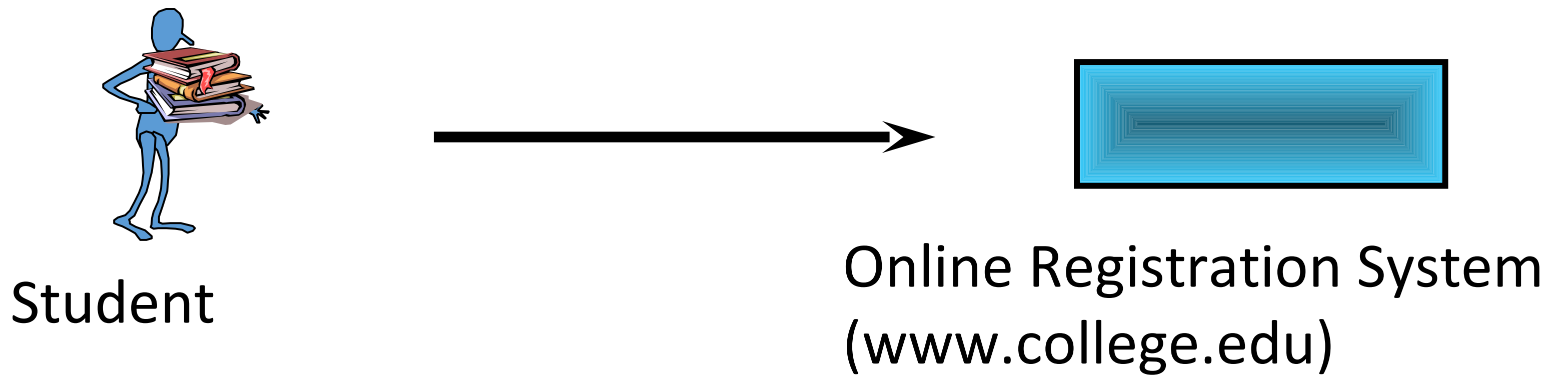


Find Actors

Who is pressing the keys (interacting with the system)?



The student never touches the system; the registrar operates it.
Or, are you building an Internet application?



Identify Actors

- Who/what uses the system?
- Who/what gets information from this system?
- Who/what provides information to the system?
- Where in the company is the system used?
- Who/what supports and maintains the system?
- What other systems use this system?

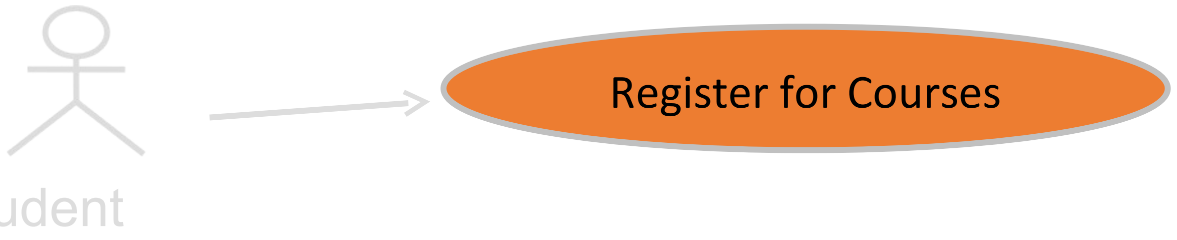
Description of an Actor

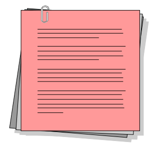
Text

Name Student

Brief description A person who signs up for a course.

Relationships with use cases





Checkpoints for Actors

- Have you found all the actors? Have you accounted for and modeled all roles in the system's environment?
- Is each actor involved with at least one use case?
- Can you name at least two people who would be able to perform as a particular actor?
- Do any actors play similar roles in relation to the system? If so, merge them into a single actor.

Class Exercise -> Assignment 1

Please spend 5 minutes writing down what use cases should be included in an ATM Banking Service System. Also give a brief description for a use case named "ATM Cash Withdrawal".

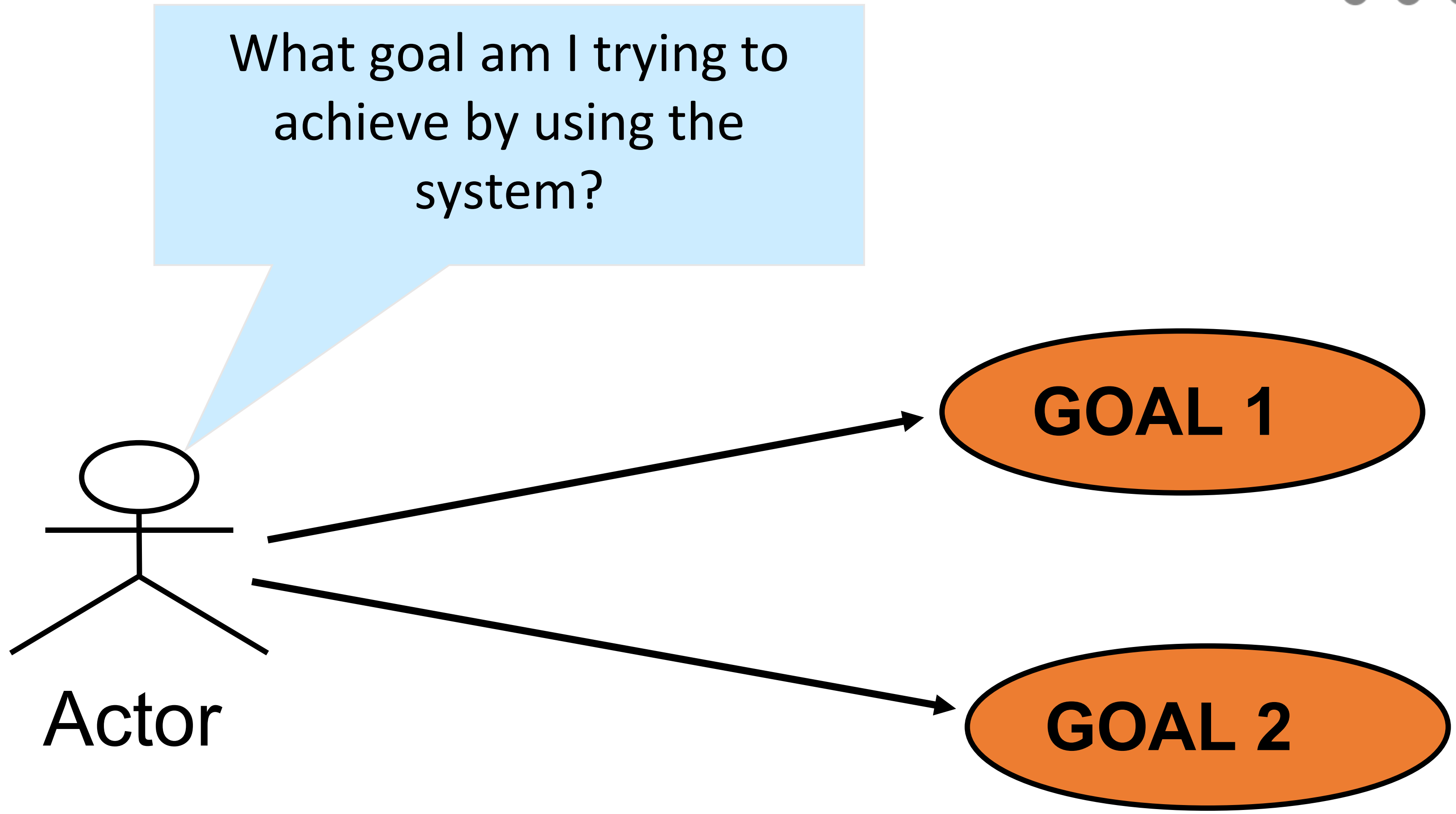
Self Evaluate your use cases with the following scheme:

- the use cases cover the major requirements for the system? (40%)
  - the inclusion, extension, and generalization relationships are used correctly
- each of your use description covers a major functional requirements?(40%)
  - Main successful scenario(15%)
  - major failure scenarios(15%);
  - pre/post conditions , and others(10)%
- using a well-formed format and style; (20%)
  - the name of actors, use cases, and descriptions are written in well-chosen phrases
  - Easy to read and understand, do not include too much unnecessary details

干系人 (Stakeholder)? 主、次参与者 (Actor)? 系统? 无关?

- ATM
- 顾客
- 银行卡
- 银行
- 机器面板
- 银行董事
- 打印机
- 服务维护人员
- 银行中央计算机系统
- 银行职员
- 银行抢劫犯

Find Use Cases

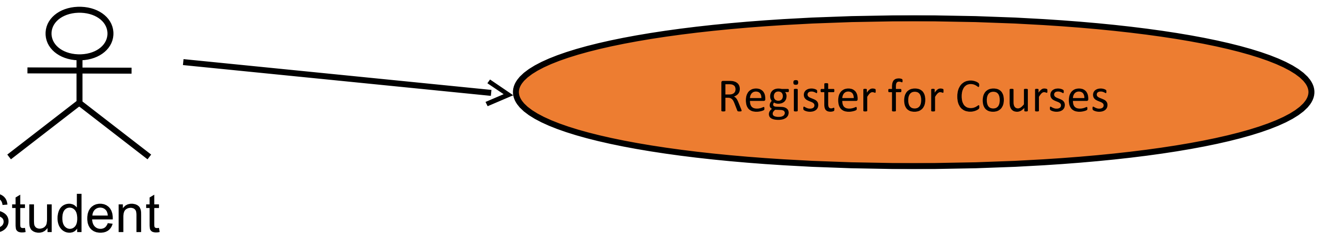


Identify Use Cases

- What are the goals of each actor?
  - Why does the actor want to use the system?
  - Will the actor create, store, change, remove, or read data in the system? If so, **why**?
  - Will the actor need to inform the system about external events or changes?
  - Will the actor need to be informed about certain occurrences in the system?
- Does the system supply the business with all of the correct behavior?

Description of a Use Case

Text description of a use case.

Name	Register for Courses
Brief description	The student selects the courses they wish to attend to the next semester. A schedule of primary and alternate courses is produced.
Relationships with actors	

Checkpoints for Use Cases

- The use-case model presents the behavior of the system; it is easy to understand what the system does by reviewing the model.
- All use cases have been identified; the use cases collectively account for all required behavior.
- All features map to at least one use case.
- The use-case model contains no superfluous behavior; all use cases can be justified by tracing them back to a functional requirement.
- All **CRUD** use cases have been removed.
  - **C**reate, **R**etrieve, **U**pdate, **D**elete

Use Case 图的建立步骤

(1) 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；

(2) 确定每一个参与者所期望的系统行为；

(3) 把这些系统行为命名为 Use Case ；

(4) 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；

(5) 编制每一个 Use Case 的脚本；

(6) 绘制 Use Case 图；

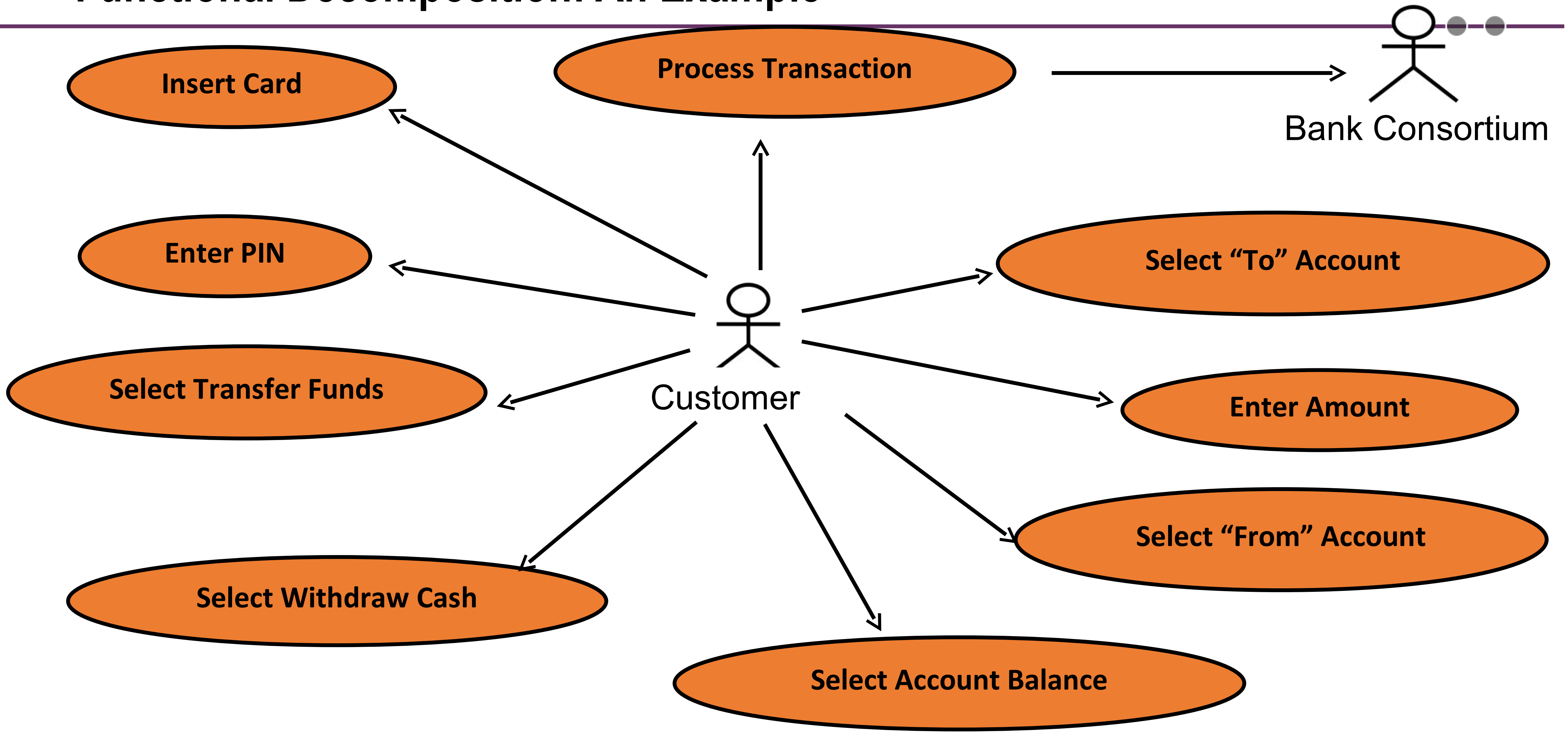
(7) 区分主事件流和异常情况的事件流，可以把表示异常情况的事件流作为单独的 Use Case 处理；

(8) 细化 Use Case 图，解决 Use Case 间的重复与冲突问题。

Functional Decomposition

- Is breaking down a problem into small, isolated parts.
  - The parts work together to provide the functionality of the system.
    - Often do not make sense in isolation.
- Use cases:
  - Are NOT functional decomposition.
  - Keep the functionality together to describe a complete use of the system.
  - Provide context.

Functional Decomposition: An Example



Avoid Functional Decomposition

Symptoms

- Very small use cases
- Too many use cases
- Uses cases with no result of value
- Names with low-level operations
  - “Operation” + “object”
  - “Function” + “data”
  - Example: “Insert Card”
- Difficulty understanding the overall model

Corrective Actions

- Search for larger context

  “Why are you building this system?”
- Put yourself in user’s role

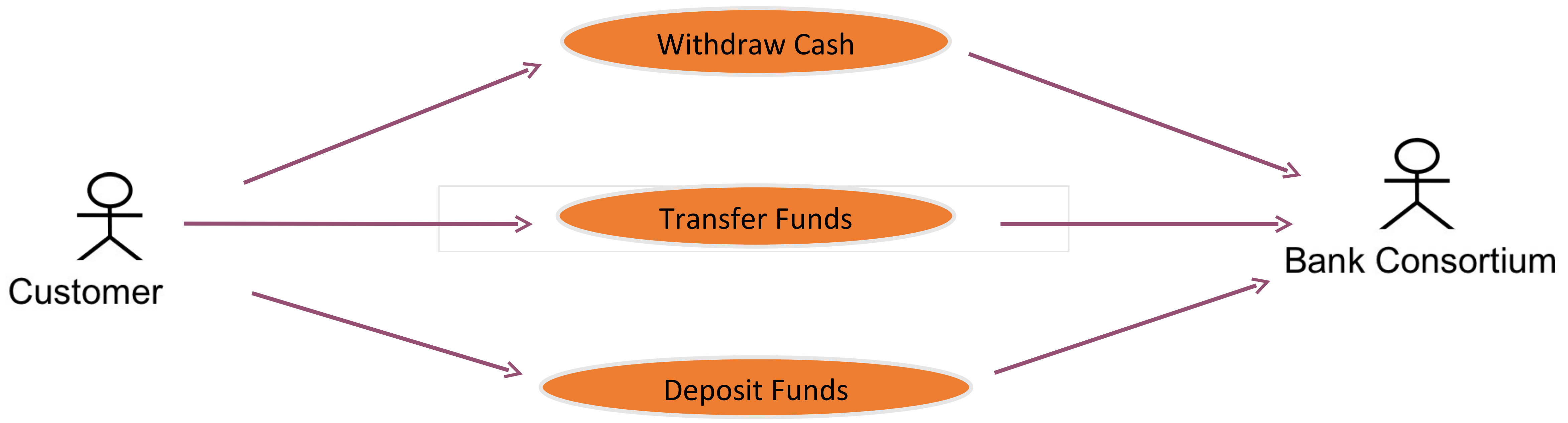
  “What does the user want to achieve?”

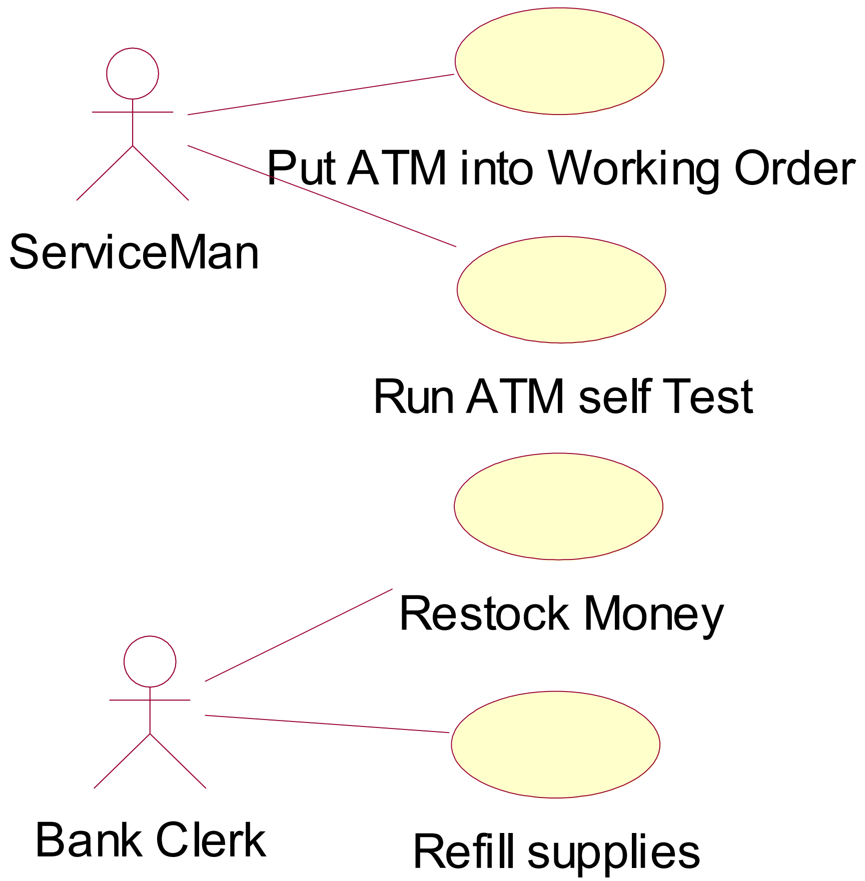
  “Whose goal does this use case satisfy?”

  “What value does this use case add?”

  “What is the story behind this use case?”

Functional Decomposition: A Corrected Example




ServiceMan
Put ATM into Working Order
Run ATM self Test
Restock Money
Bank Clerk
Refill supplies

主成功流程

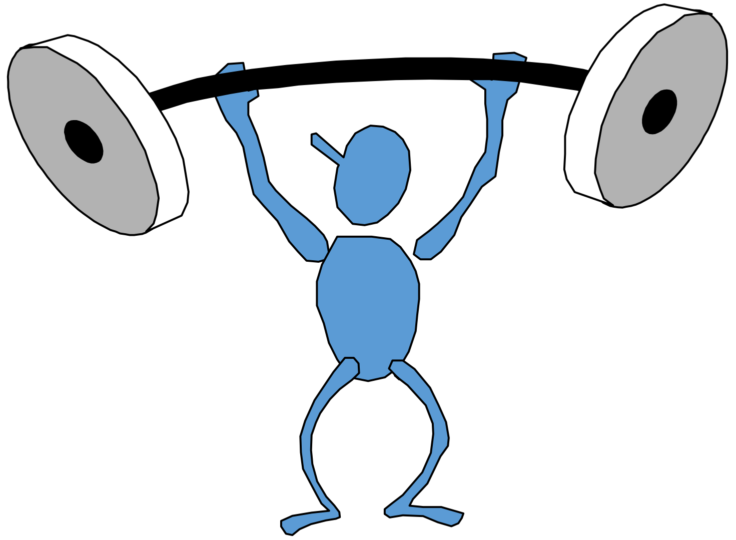
1. 顾客插卡
2. ATM 读卡
3. ATM 提示选择语言
4. 顾客选择英语
5. ATM 提示输入密码
6. 顾客输入密码
7. ATM 显示待选服务
8. 顾客选择取款
9. ATM 提示输入取款数额，必为 50 的倍数
10. 顾客输入数值
11. ATM 通知银行中央系统客户的取款数额
12. 银行中央系统接受请求，并通知 ATM 新的账户余额
13. ATM 输出现金
14. ATM 询问顾客是否要收据
15. 顾客回答需要收据
16. ATM 输出并打印收据
17. ATM 记录该项交易

例外情况:

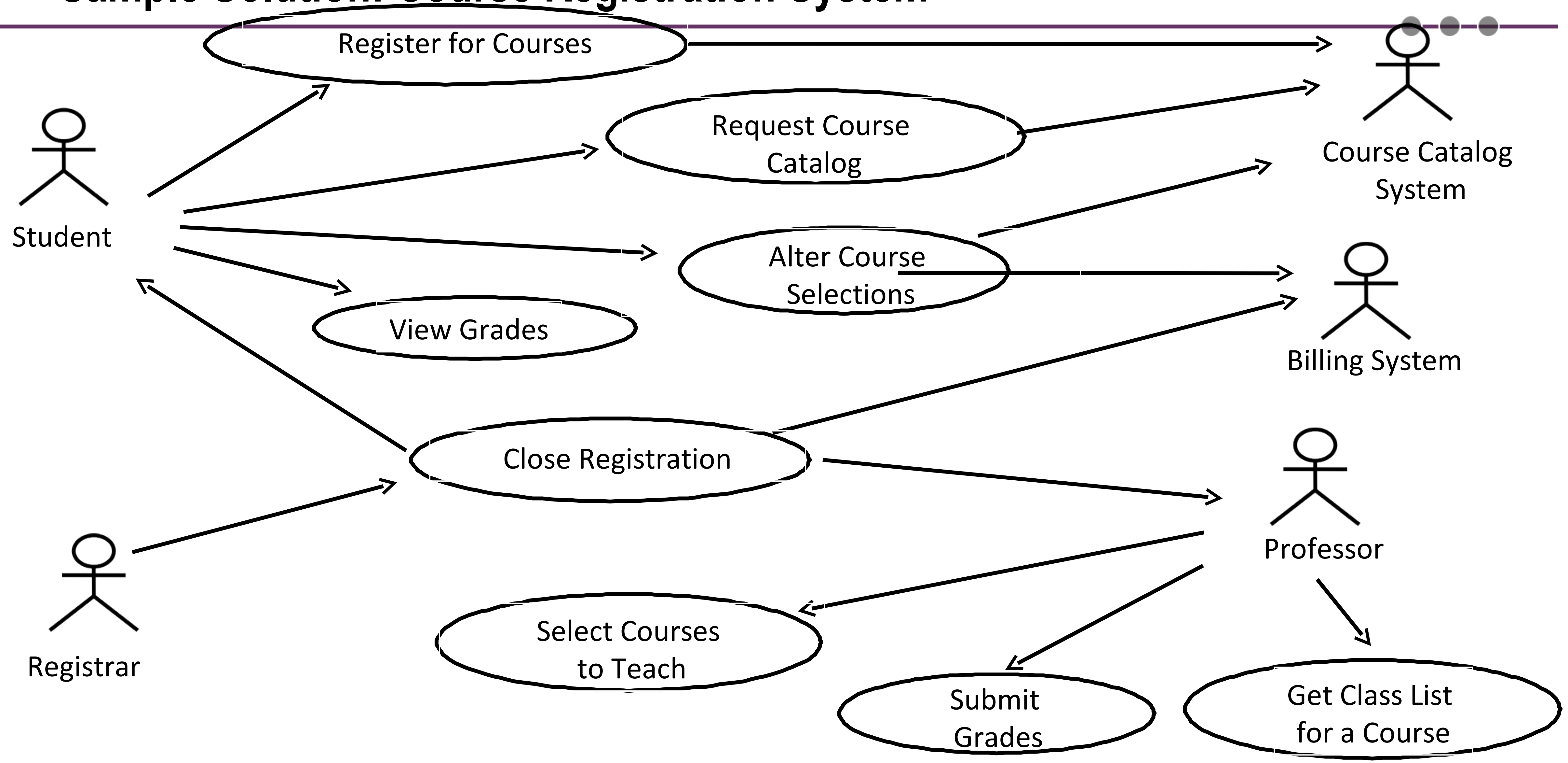
• Card reader broken or card scratched	读卡异常
• Card for an ineligible bank	非本机接受卡种
• Incorrect PIN	错误密码
• Customer does not enter PIN in time	输入密码超时
• ATM is down	机器停机
• Host computer is down, or network is down	主机坏或网络断
• Insufficient money in account	账户余额不足
• Customer does not enter amount in time	输入数额超时
• Not a multiple of $50	数额不合要求
• Amount requested is too large	数额过大
• Network or host goes down during transaction	交易期间系统坏
• Insufficient cash in dispenser	ATM 现金量不足
• Cash gets jammed during dispensing	出闭口夹纸
• Receipt paper runs out, or gets jammed	收据打印纸用完
• Customer does not take the money from the dispenser	顾客忘取现金

Exercise 3.1: Identify Actors and Use Cases

- Identify the actors who interact with the Course Registration System.
- Identify use cases for the system.
- Sketch a use-case diagram.
  - Refer to use-case and actor checkpoint slides.

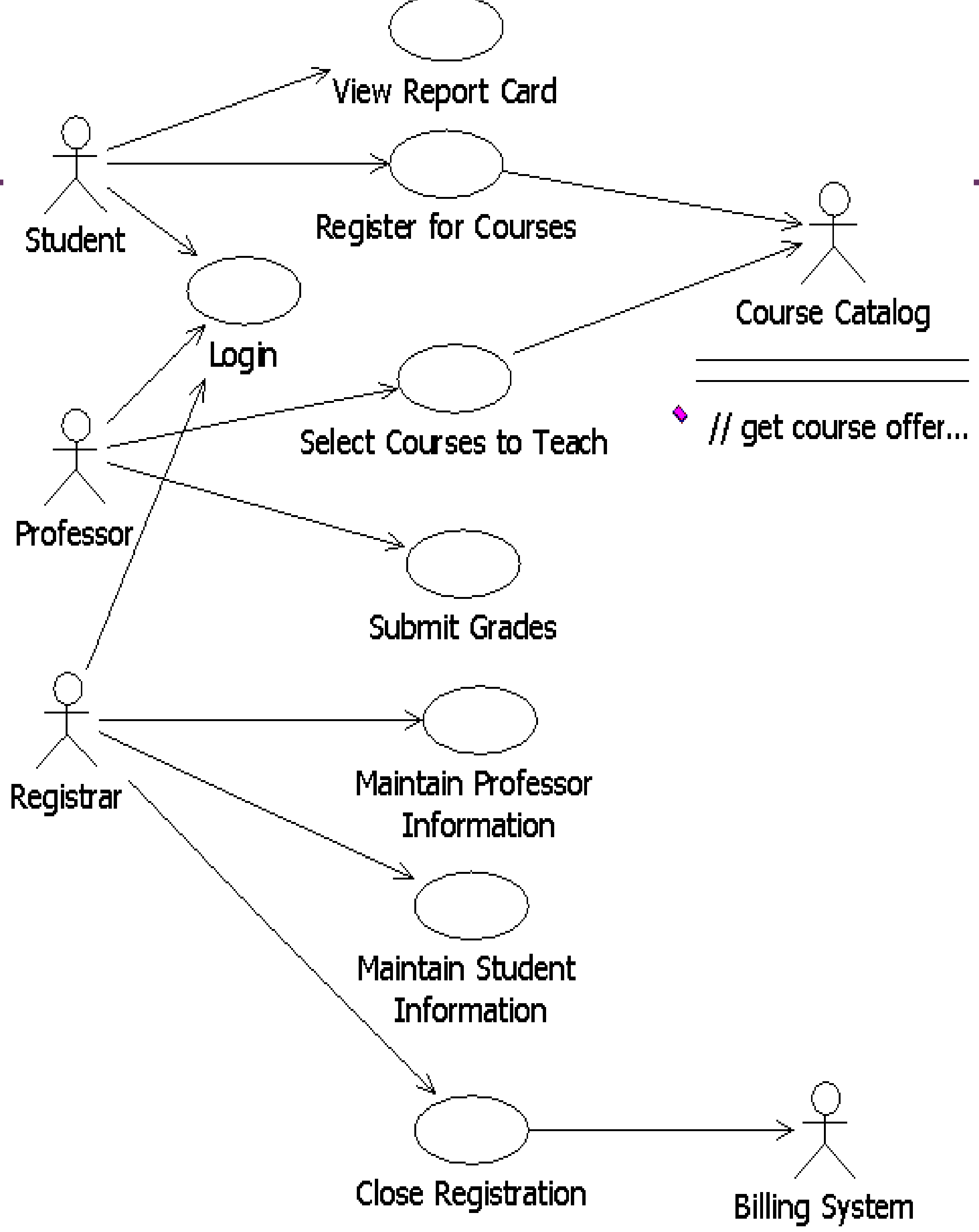


Sample Solution: Course Registration System

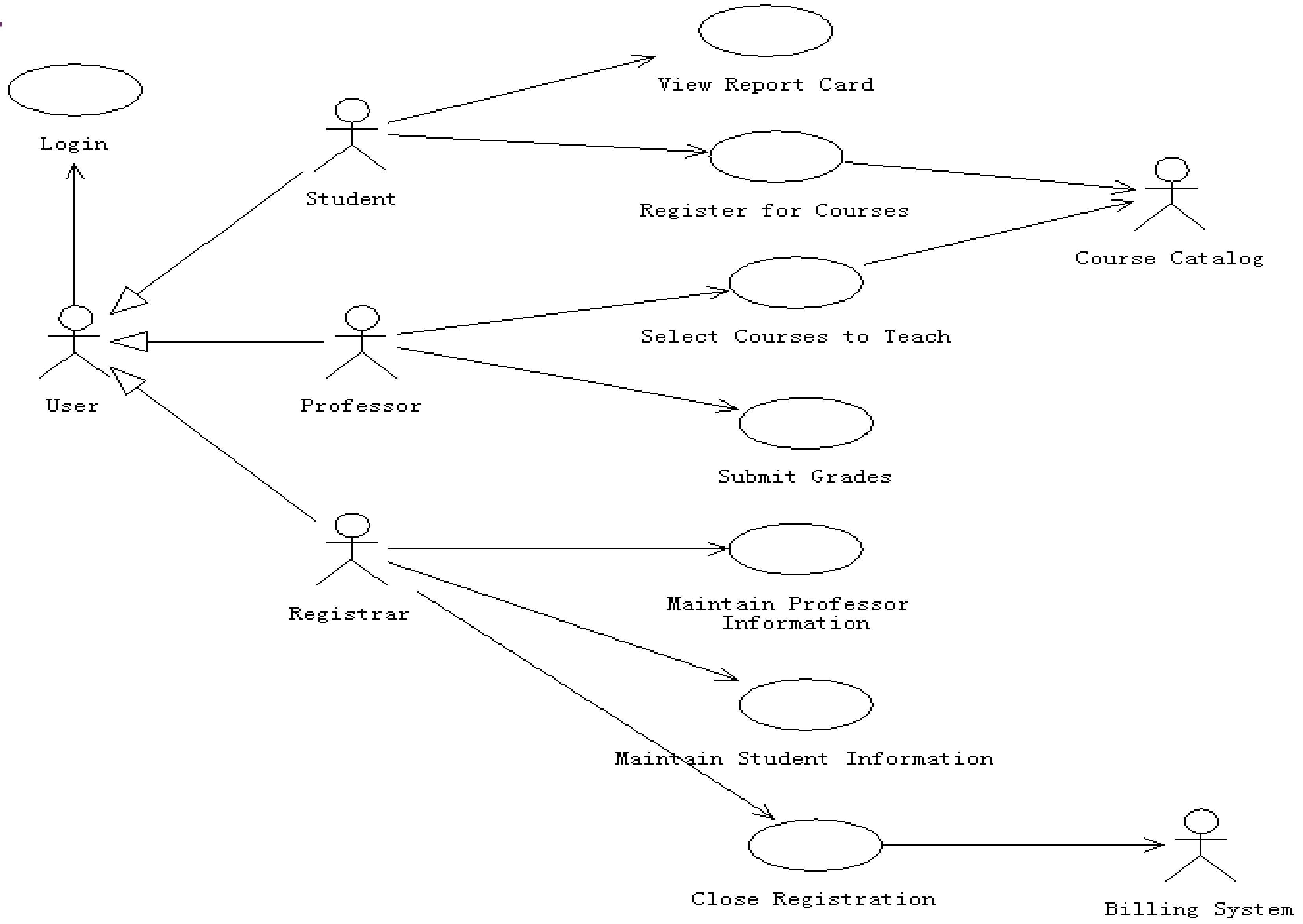


Use Case 图说明

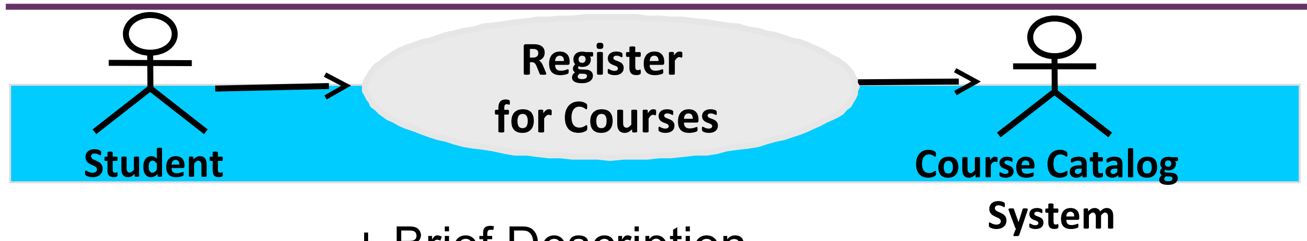
共五个 actor，八个 use case 。



比较：RUP 2002 中的例子的 Use Case 图

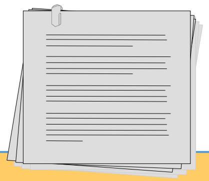


Evolve the Use Case: Diagram→ Outline→ Detail



✓

+ Brief Description



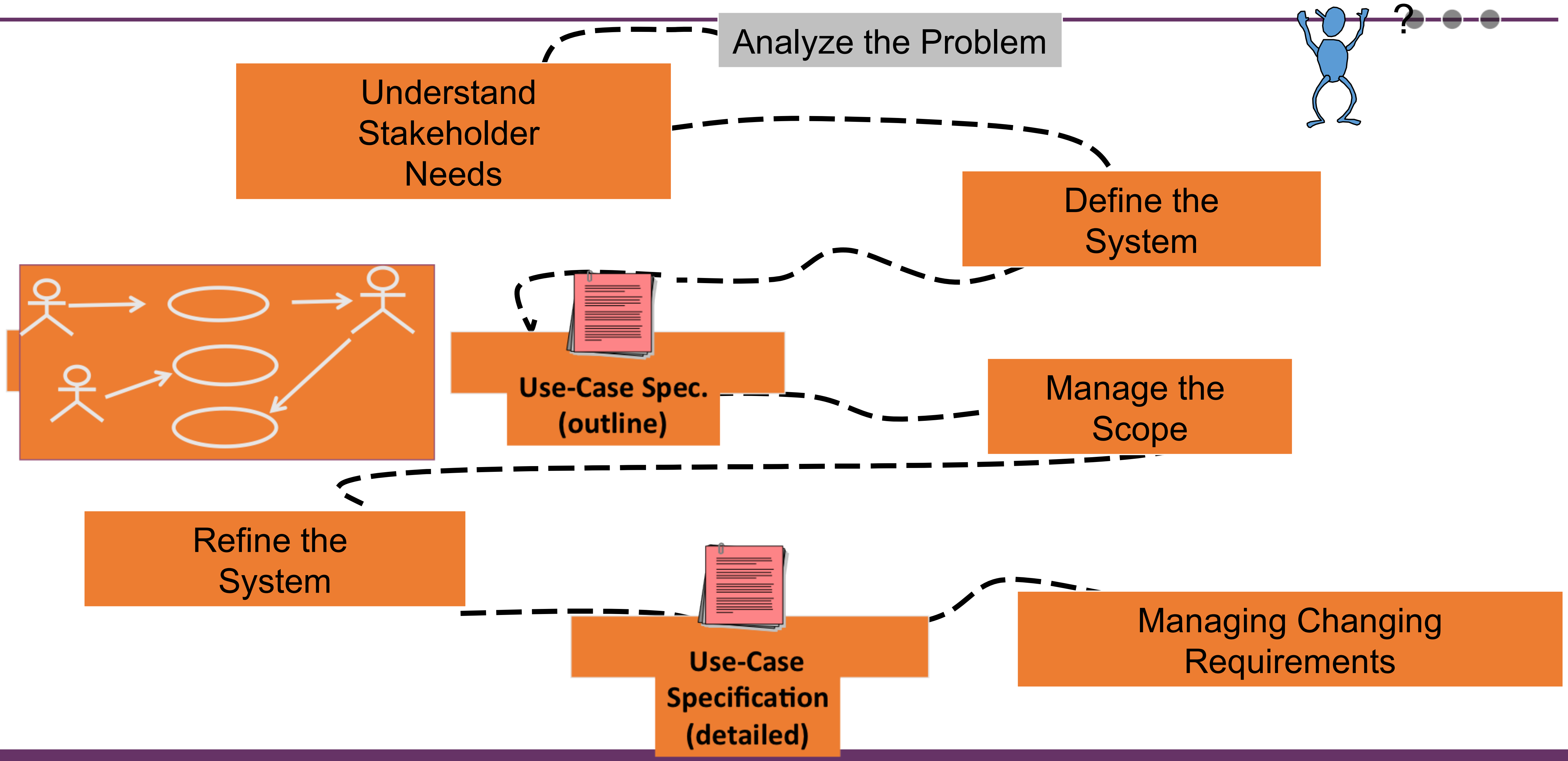
Register for Courses Outline

+ Flow of events outlined
  - High-level steps

Register for Courses Use-Case Specification

+ Flow of events detailed
  - Step-by-step
+ Special Requirements
+ Pre/Post Conditions

Where Do Use Cases Fit into the RM Process?



Review

1. What are the benefits of use-case modeling?
2. What is included in a use-case model?
3. How do you identify actors and use cases?
4. What is functional decomposition?
5. Why do we want to avoid functional decomposition?
6. What are some questions you can ask to test the quality of your use-case model?

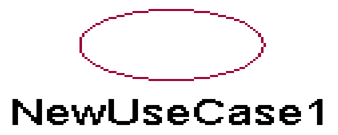
寻找用例的方法

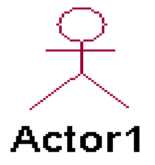
- 和用户交互。
- 基本策略：把自己当作 actor，与设想中的系统进行交互。
  考虑:
  - 系统交互的目的是什么?
  - 需要向系统输入什么信息?
  - 希望由系统进行什么处理并从它得到何种结果?
- 确定 Use Case 和确定 actor 不能截然分开。

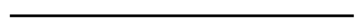
- Jacobson 提出以下问题:
  - 参与者的主要任务是什么?
  - 参与者要了解系统的什么信息？需要修改系统的什么信息?
  - 参与者是否需要把系统外部的变化通知系统?
  - 参与者是否希望系统把异常情况的变化通知自己?

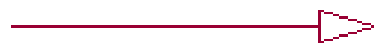
- 寻找用例时需要注意的问题:
  - 不要一开始就去捕捉所有的细节。
  - 全面地认识和定义每一个 use case ，要点是以穷举的方式考虑每一个 actor 与系统的交互情况。

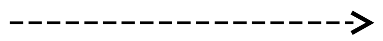
用例图中的一些主要图标

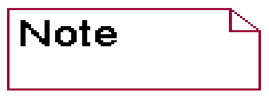




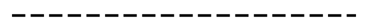

Association
关联

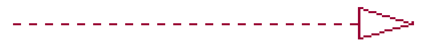

Generalization
泛化

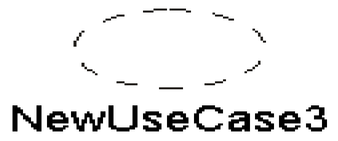

Dependency
依赖



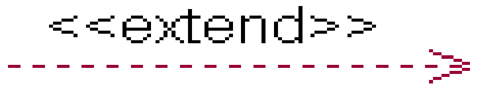
注释


注释连接

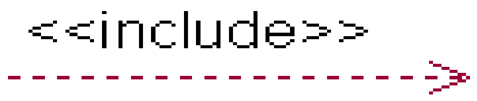

Realize



use-case realization



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。

When to use Includes？ 何时使用包含关系？

- You have a piece of behavior that is similar across many use cases （多个用例有共享行为）
- Break this out as a separate use-case and let the other ones “include” it （为共享行为单独创建用例）
- Examples include
  - Valuation (估价)
  - Check for proper authorization （权限检查）

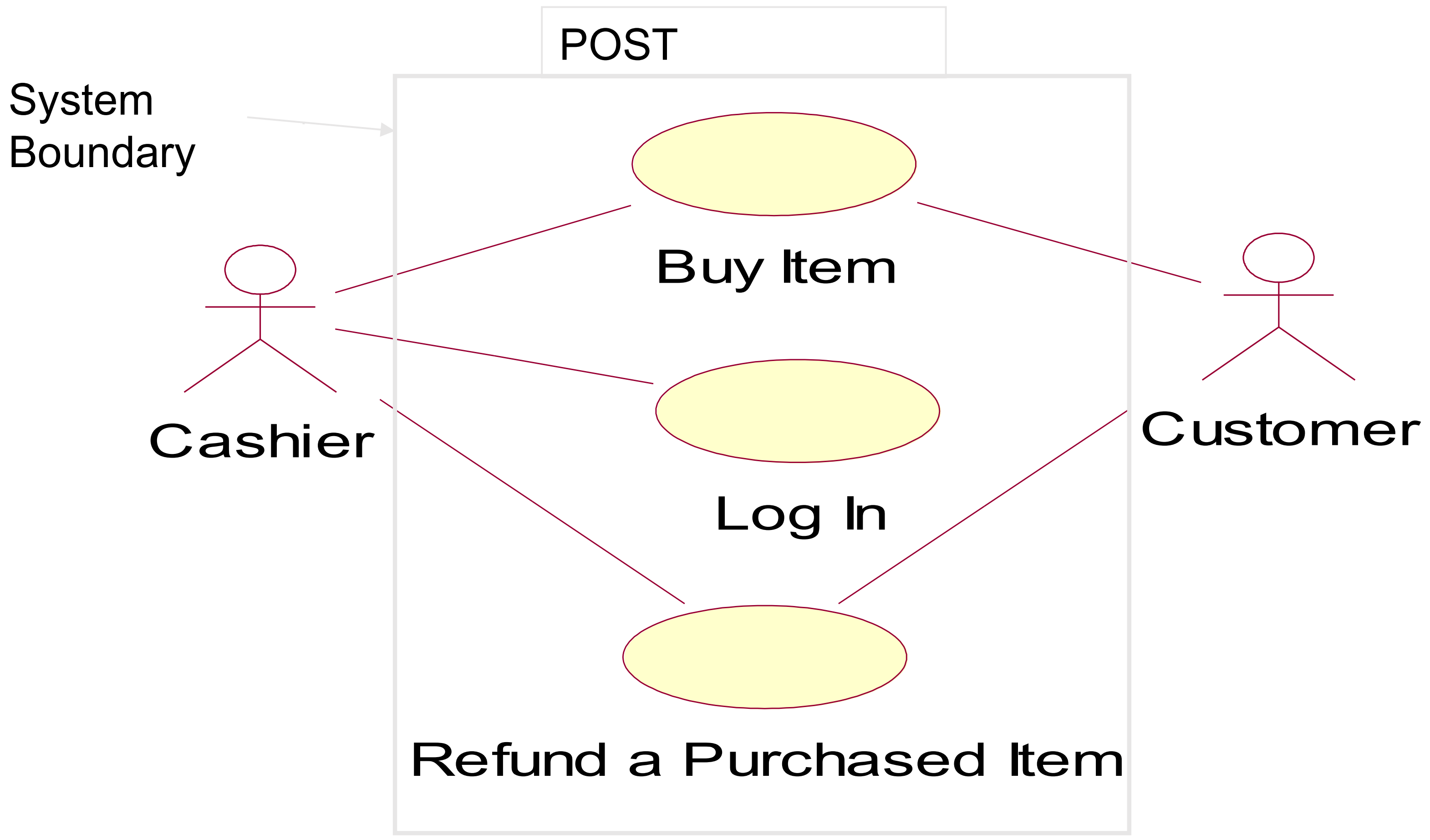
When to use Extends？ 何时使用扩展关系？

- A use case is similar to another one but does a little bit more （一个用例与另外一个用例近似，只有少许额外的活动）
- Put the normal behavior in one use-case and the exceptional behavior somewhere else （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）

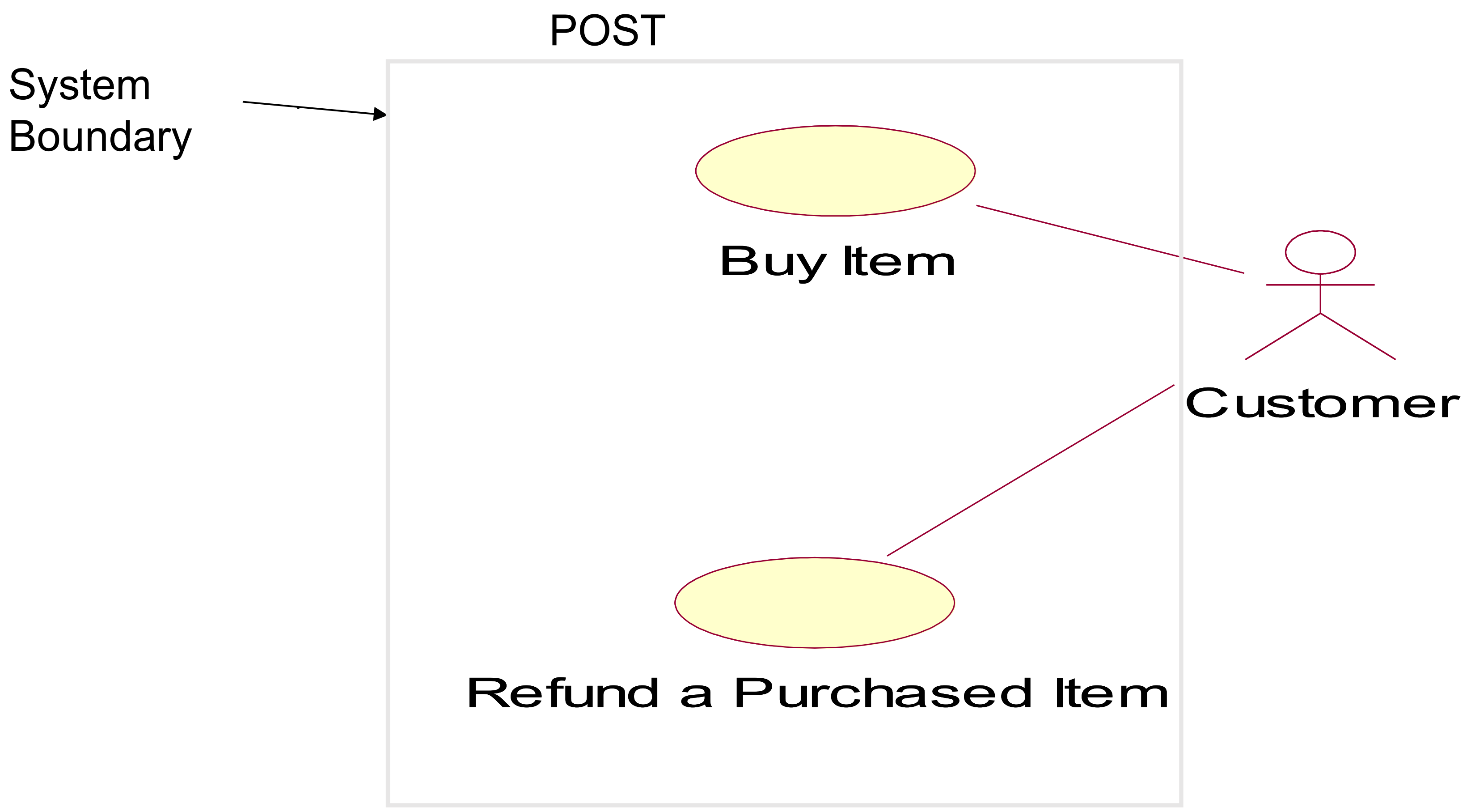
Setting the system boundary 设定系统边界

- 系统边界会对用例以及 Actor 的定义有所影响
- 画出以下系统的用例图 （3 分钟）：
  - 记录销售及付款情况的计算机系统
  - 用于零售店
  - 包括硬件设备，如计算机、条码扫描装置
  - 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

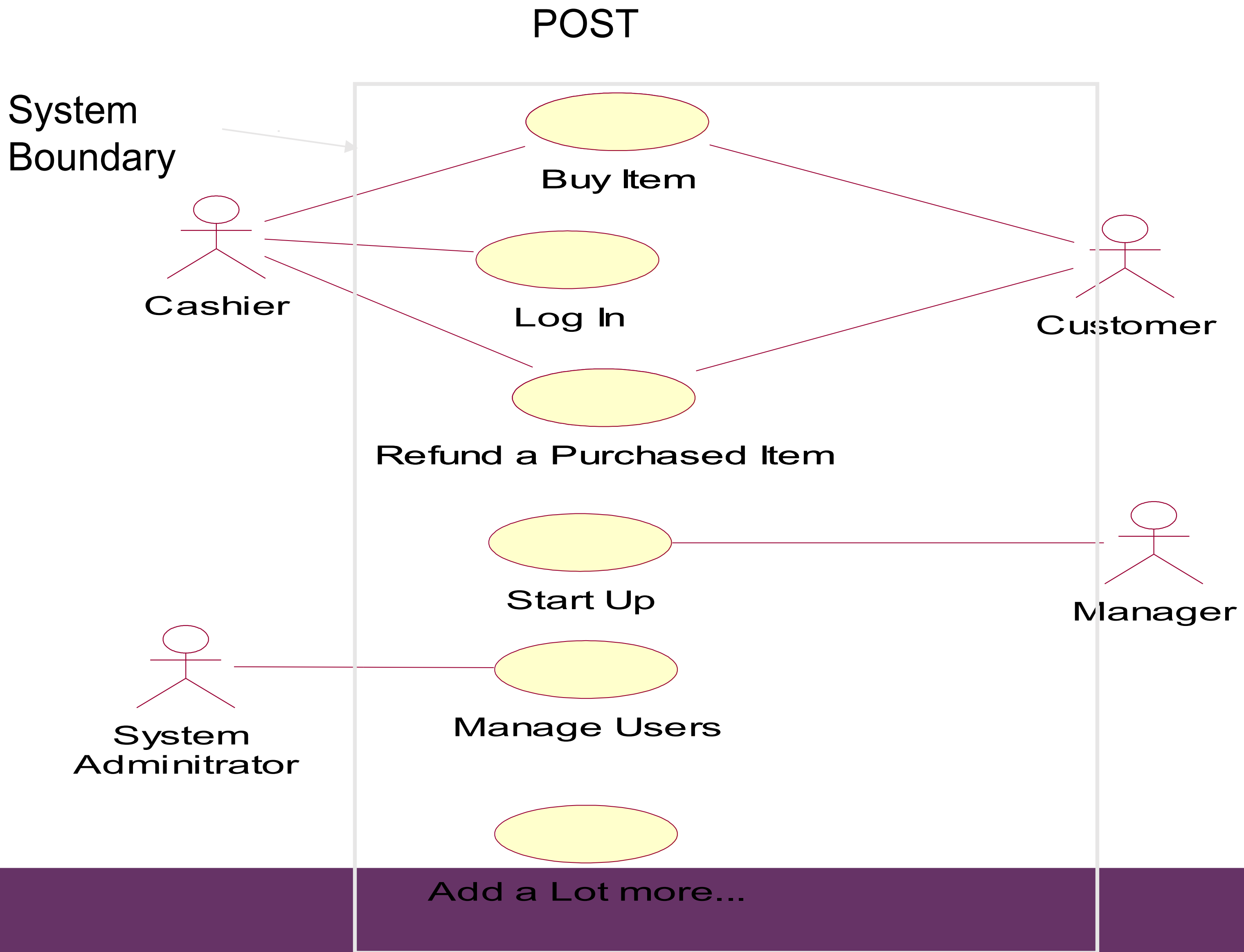
System Boundary (系统边界定义之一)



System Boundary (系统边界定义之二)



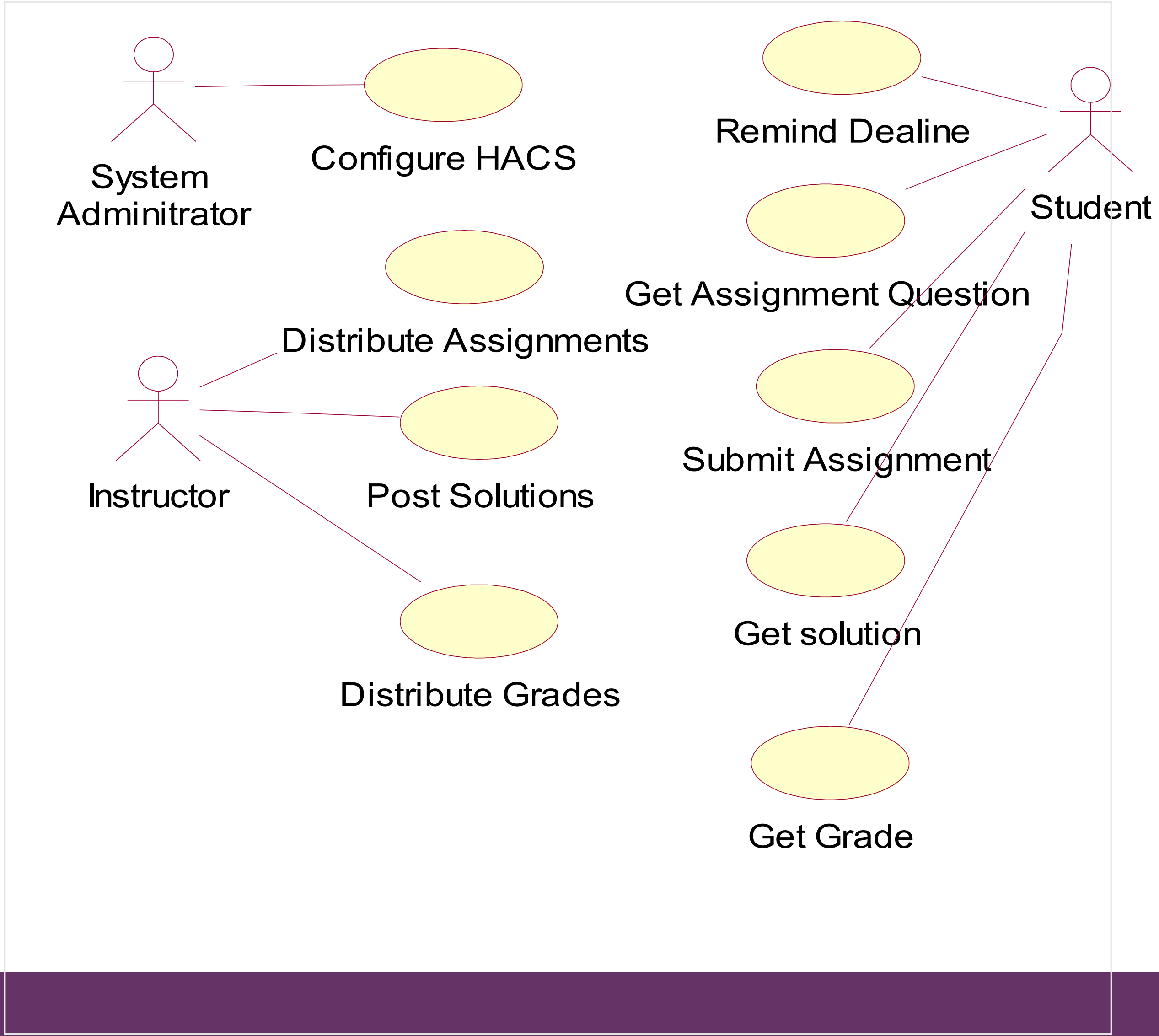
System Boundary（系统边界定义之三）



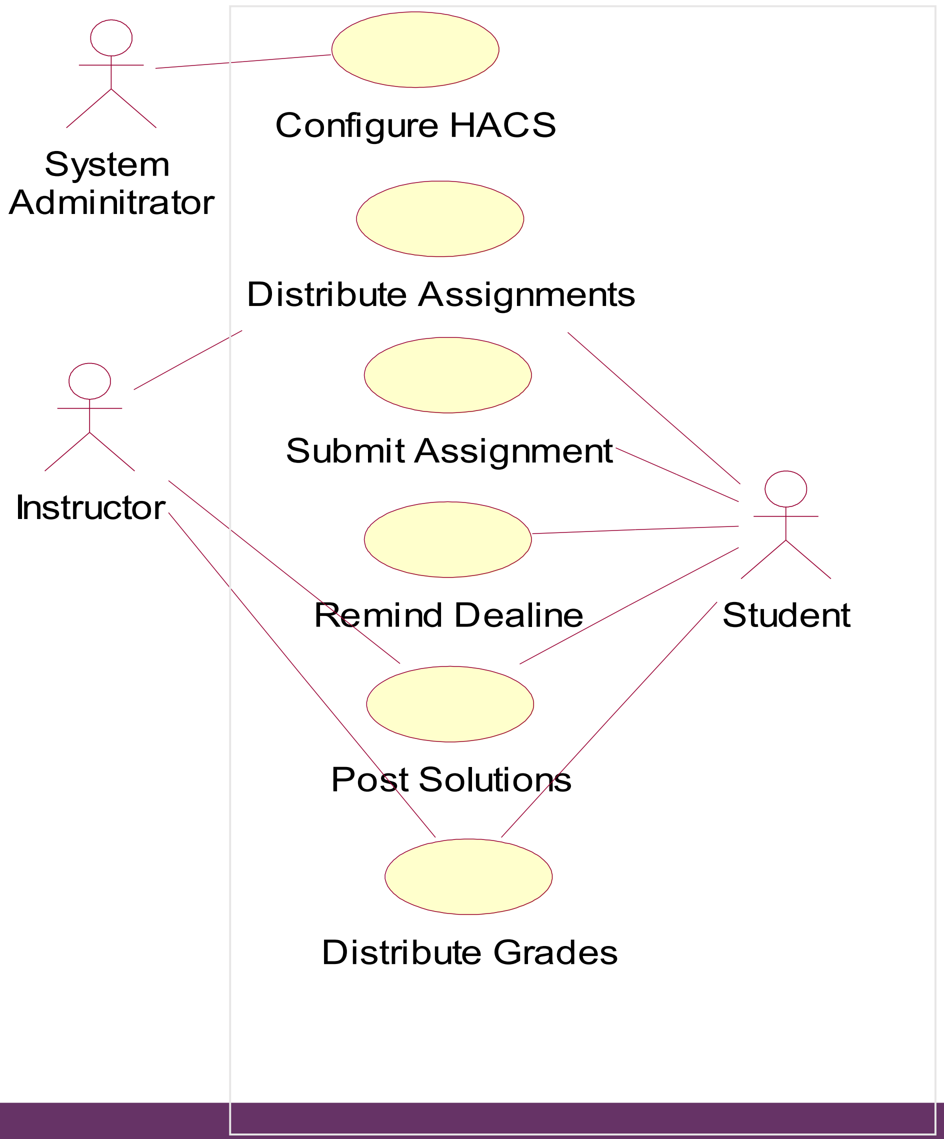
Another Small Exercise for you

- We want to build a homework assignment distribution and collection system (HACS).
- HACS will be used by the instructor to distribute the homework assignments, review the students′ solutions, distribute suggested solution, and distribute student grades on each assignment.
- HACS shall also help the students by automatically distributing the assignments to the students, provide a facility where the students can submit their solutions, remind the students when an assignment is almost due, remind the student when an assignment is overdue.

用例图一


System Adminitrator
Configure HACS
Remind Dealine
Student
Get Assignment Question
Distribute Assignments
Instructor
Post Solutions
Submit Assignment
Get solution
Distribute Grades
Get Grade

用例图二


Configure HACS
System Adminitrator
Distribute Assignments
Submit Assignment
Instructor
Remind Dealine
Student
Post Solutions
Distribute Grades

思考题

- 用 Use Case 获取需求的方法是否有什么缺陷，还有什么地方需要改进?
（提示：是否对所有的应用领域都适用？使用的方便性？……）

总结

1. Use case 在软件开发中的作用，一些基本概念： use case ， actor ， scenario 等
2. actor 之间的泛化关系
3. use case 之间的泛化，包含，扩展关系及其区别
4. use case 的应用
5. 完成大作业的用例图

说明:

- 参与者 Course Catalog 和 Billing System 为外部系统。
- 在分析阶段，参与者 Course Catalog 有方法:
  - // get course offerings
- 课程注册系统对 Login， Register for Courses 用例的分析和设计较完整，包括了 Use Case Model， Analysis Model， Design Model， Process Model 和 Deployment Model； 对 Close Registration 用例只进行了分析，包括了 Use Case Model 和 Analysis Model； 对于剩下的用例，只提供了 Use Case Model 。
- 有八个文件分别对各个 Use case 进行说明。

Usecase Description

Usecase name	Register VolBank
Usecase Description	This usecase is for volunteers to register one's own information.
User	Volunteer
Pre-condition	Website is available.
Post-condition	The web-server stores the detail of the volunteer for match.
Basic flow	1. Open the website. 2. Register one's information include skills, time and address. 3. Summit.
Extended flow	4. Edit one's information again.
Usecase generalized	Record skill, record time, record address.
Usecase included	Null
Usecase extended	Null

注册课程 (Register for Courses 用例的描述)

1. 用例名称：注册课程

1.1 简要描述

这个用例允许学生注册本学期需要学习的课程。在学期开始的课程 “ add/drop” 阶段，学生也可以修改或删除所选择的课程。课程目录系统提供了本学期开设的所有课程列表。

1.2. 事件流程

1.2.1 基本流程

当学生希望注册课程，或想改变他的课程计划 (Schedule) 时，用例开始执行。

- 1. 系统要求学生选择要执行的操作 (创建计划，修改计划，或删除计划).
- 2. 一旦学生提供了系统要求的信息，以下子流程中的某一个将被执行 .

如果学生选择的是 “ Create a Schedule”，则“创建计划”子流程将被执行。
如果学生选择的是 “ Update a Schedule”，则“修改计划”子流程将被执行。
如果学生选择的是 “ Delete a Schedule”，则“删除计划”子流程将被执行。

1.2.1.1 创建计划

1. 系统从课程目录系统中检索出有效的课程列表并显示 .
2. 学生从有效课程列表中选择 4 门主选课和 2 门备选课 .
3. 当学生完成选择 , 系统将为这个学生创建一个“计划” ，这个计划包含了学生所选的课程 .
4. 执行“提交计划”子流程

1.2.1.2 修改计划

1. 系统检索并显示学生当前的计划
2. 系统从课程目录系统中检索出有效的课程列表并显示 .
3. 学生可以通过增加或删除课程来修改所选课程． 学生从有效课程列表中选择增加的课程， 学生也可以从当前的计划中选择任何想要删除的课程 .
4. 当学生完成选择 , 系统将修改这个学生的“计划” .
5. 执行“提交计划”子流程

1.2.1.3 删除计划

1. 系统检索并显示学生当前的计划
2. 系统提示学生确认这次删除
3. 学生确认这次删除
4. 系统删除计划．如果这个计划中包含“已注册” (enrolled in) 的 course offering，则这个学生要从 course offering 中删除．

1.2.1.4 提交计划

1. 对于计划中所选的每门课程，如果还没标记为“已注册”，则系统将验证学生满足先修条件，且课程处于“open”状态，且计划中没有冲突，则系统将把学生加到所选的 course offering 中，计划中所选的课程标记为“已注册”
2. 计划被保存在系统中．

1.2.2 可选流程

1.2.2.1 保存计划

在任何情况下，学生可以选择保存计划而不是提交计划．在这种情况下，“提交计划”这一步被下面步骤代替：

1. 计划中没有被标记为“已注册”的课程被标记为“选择” (selected).
2. 计划被保存在系统中．

1.2.2.2 先修条件不满足或课程满员或计划冲突

如果在“提交计划”子流程中，系统检测出学生没有满足先修条件，或学生所选的课程已满，或存在计划冲突，则系统显示错误消息．学生可以选择其它课程 (用例继续)，或保存计划 (和“保存计划”子流程一样)，或取消本次操作，如果是取消操作，则用例基本流程重新开始 .

1.2.2.3 没有找到计划

如果在“修改计划”或“删除计划”子流程中，系统不能检索到学生的计划，则系统显示错误信息。学生确认该错误，用例基本流程重新开始 .

1.2.2.4 课程目录系统不可用

如果系统不能和课程目录系统通讯，系统将向学生显示错误信息，学生确认该错误，用例终止 .

1.2.2.5 课程注册结束

如果在用例开始的时候 , 系统检测到本学期的课程注册已结束，系统将向学生显示信息，用例终止．学生在本学期的课程注册结束后就不能再注册课程了 .

1.2.2.6 取消删除

如果在“删除计划”子流程中，学生决定不删除计划了，则删除被取消，用例基本流程重新开始 .

1.3 特殊需求

无

1.4 前置条件

开始这个用例之前学生必须已登录到系统

1.5 后置条件

如果用例成功结束，则会创建，或修改，或删除学生的计划，否则系统的状态不变.

1.6 扩展点

无

Title: <u>Successful meeting scheduled using messaging option</u>

Participants: Alice (initiator, not attending); Bob, Carlo, Daphne (attendees)

情景举例

Action	Goals satisfied	Obstacles / Problems
Alice requests meeting, specifying participants, timeframe	Meeting requested; Attendee list obtained	What if selected timeframe is infeasible?
AS sends participant requests to Bob, Carlo and Daphne	?	Did we miss a goal?
Bob reads message	Participants informed	Can't detect when messages are read; what happens if Bob reads the message but doesn't reply?
Carlo reads message		
Daphne reads message		
Bob replies with preferences	Attendees preferences known	What if the preferences are mutually exclusive? Should we allow some to be higher priority?
Carlo replies with preferences		
Daphne replies with preferences		
AS schedules meeting	Room availability determined; room booked	
AS notifies Alice, Bob, Carlo, Daphne of time and location	Meeting announced; Attendance Confirmed (?)	How do we know if they've all read the announcement? What if the schedule is no longer convenient for one of them?

用例补充讲议

Relationships between Use Case

- Use Case 除了和参与者有**关联 (association)** 外， Use Case 之间也存在着一定的关系 (relationship) 。包括： **泛化 (generalization) 关系**、**包含 (include) 关系**、**扩展 (extend) 关系**等。
- 也可以利用 UML 的扩展机制自定义 Use Case 间的关系。
- **relationship(关系)**, **association(关联)**, **generalization(泛化)**, **dependency(依赖)** 的区别。
  - association, generalization, dependency 都属于 relationship 。
  - include ， extend 属于 dependency 。

Generalization(泛化关系)

- 泛化 (generalization) 代表一般与特殊的关系。 use case 间的泛化关系与类之间的泛化关系（继承关系）类似。

- use case generalization is a taxonomic relationship between a use case (the child) and the use case (the parent) that describes the characteristics the child shares with other use cases that have the same parent.

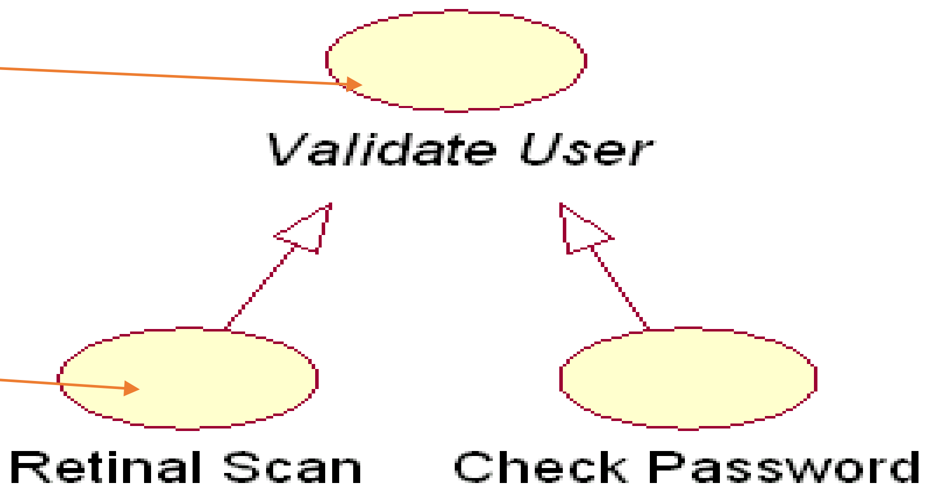
Generalization example :

Use case behavior for Parent Check Password:

1. Obtain password from master database
2. Ask user for password
3. User supplies password
4. Check password against user entry

parent use case

child use case



Use case behavior for child Retinal Scan

1. Obtain retinal signature from master database
2. Scan user's retina and obtain signature
3. Compare master signature against scanned signature

More about Generalization

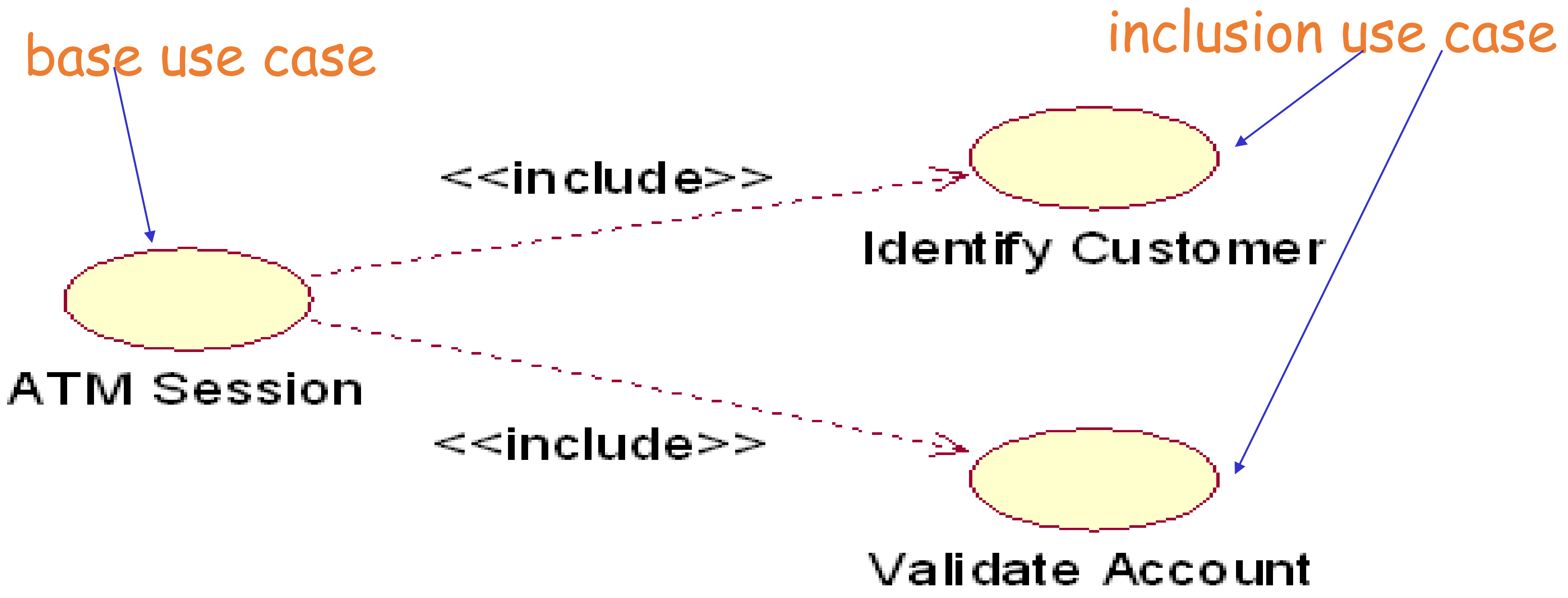
- **child use case** inherits the behavior and meaning of the **parent use case**;
- The **child** may add to or override the behavior and meaning of the **parent** use case;
- The **child** may be substituted any place the **parent** appears.

include (包含关系)

- 包含 (Include) 关系是指一个基本 Use Case 的行为包含了另一个 Use Case 的行为。

- A relationship from a base use case to an inclusion use case, specifying how the behavior defined for the inclusion use case can be inserted into the behavior defined for the base use case.

- 包含关系的例子：


base use case
inclusion use case
<<include>>
Identify Customer
ATM Session
<<include>>
Validate Account

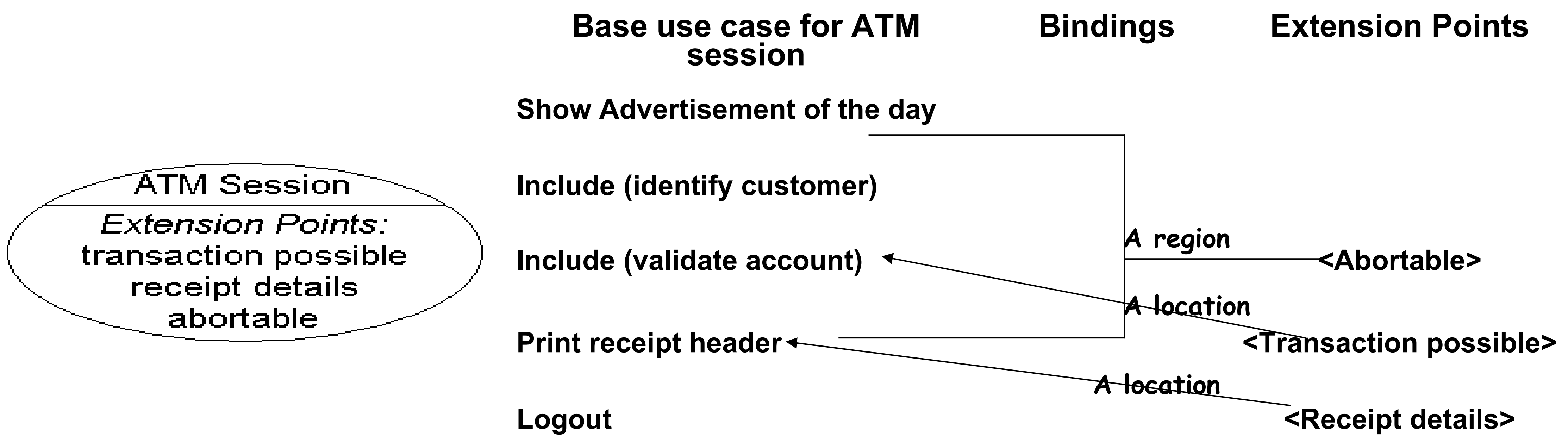
extend (扩展关系)

- 扩展 (extend) 关系的基本含义与泛化关系类似，但是对于扩展 Use Case 有更多的规则限制，即基本 Use Case 必须声明若干 “扩展点” (extension point)，而扩展 Use Case 只能在这些扩展点上增加新的行为。

- A relationship from an extension use case to a base use case, specifying how the behavior defined for the extension use case can be inserted into the behavior defined for the base use case.

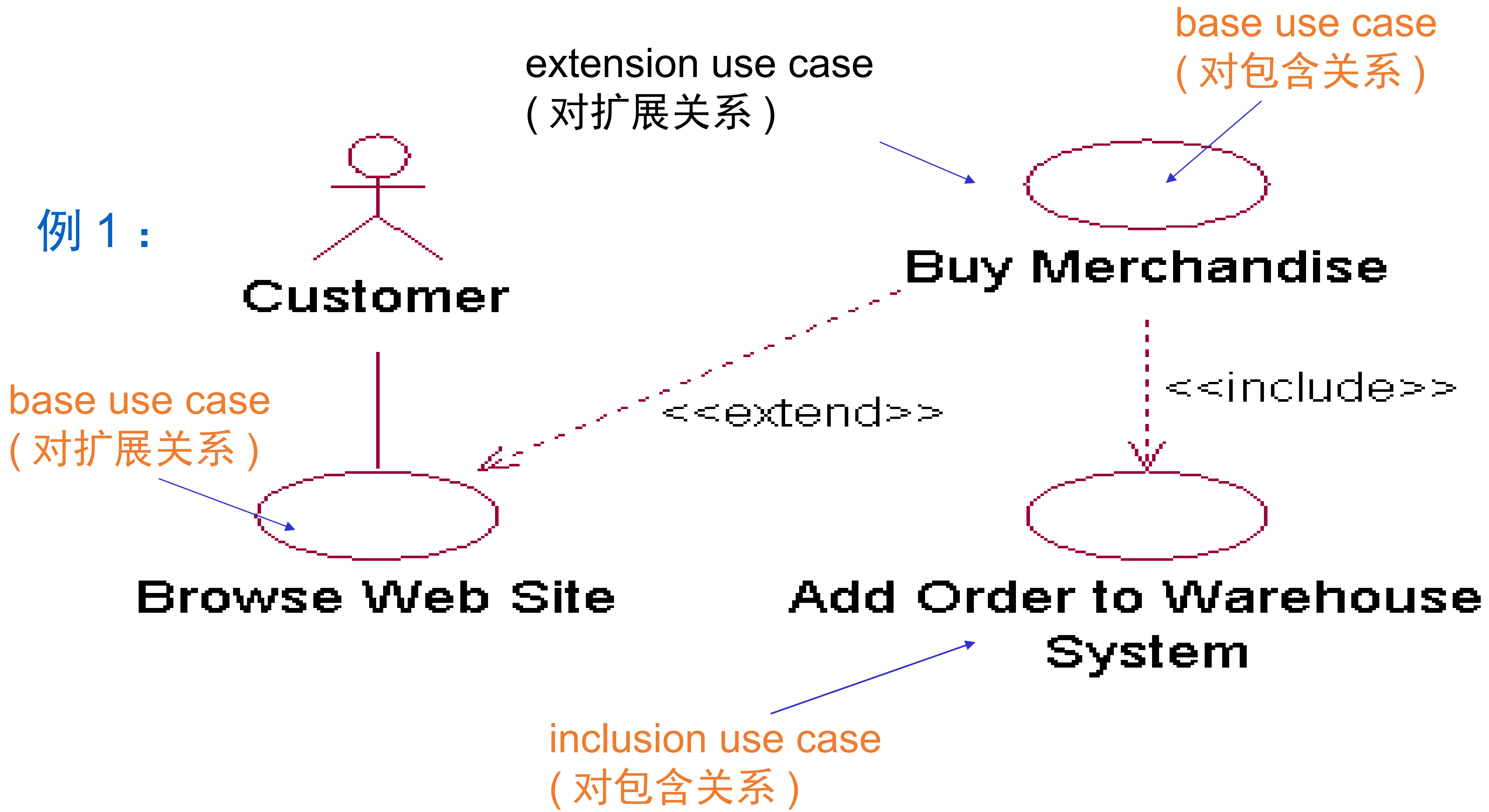
extension point(扩展点)

- An **extension point** is a named marker that references a location or set of locations within the behavioral sequence for a use case, at which additional behavior can be inserted.

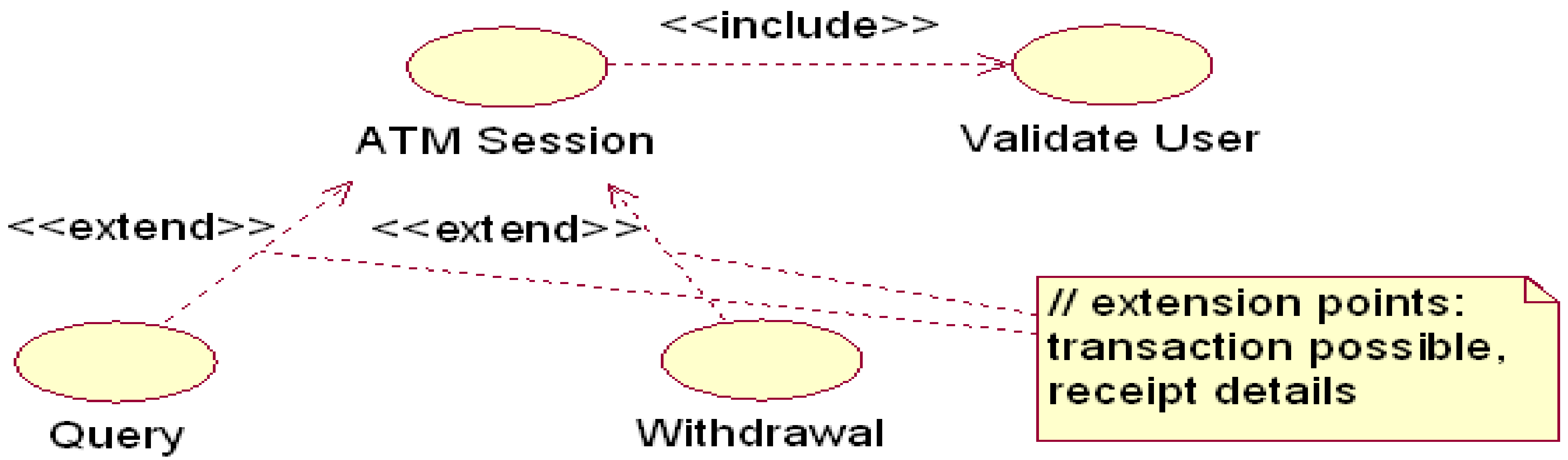


扩展关系和包含关系的例子：

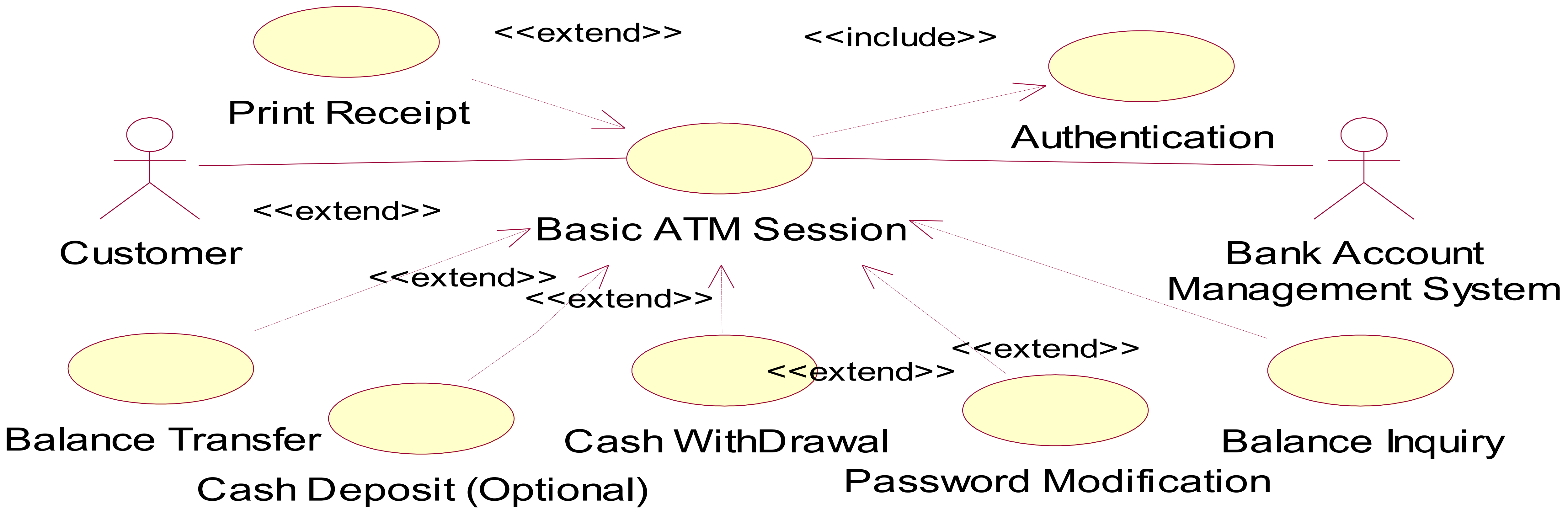
例 1：



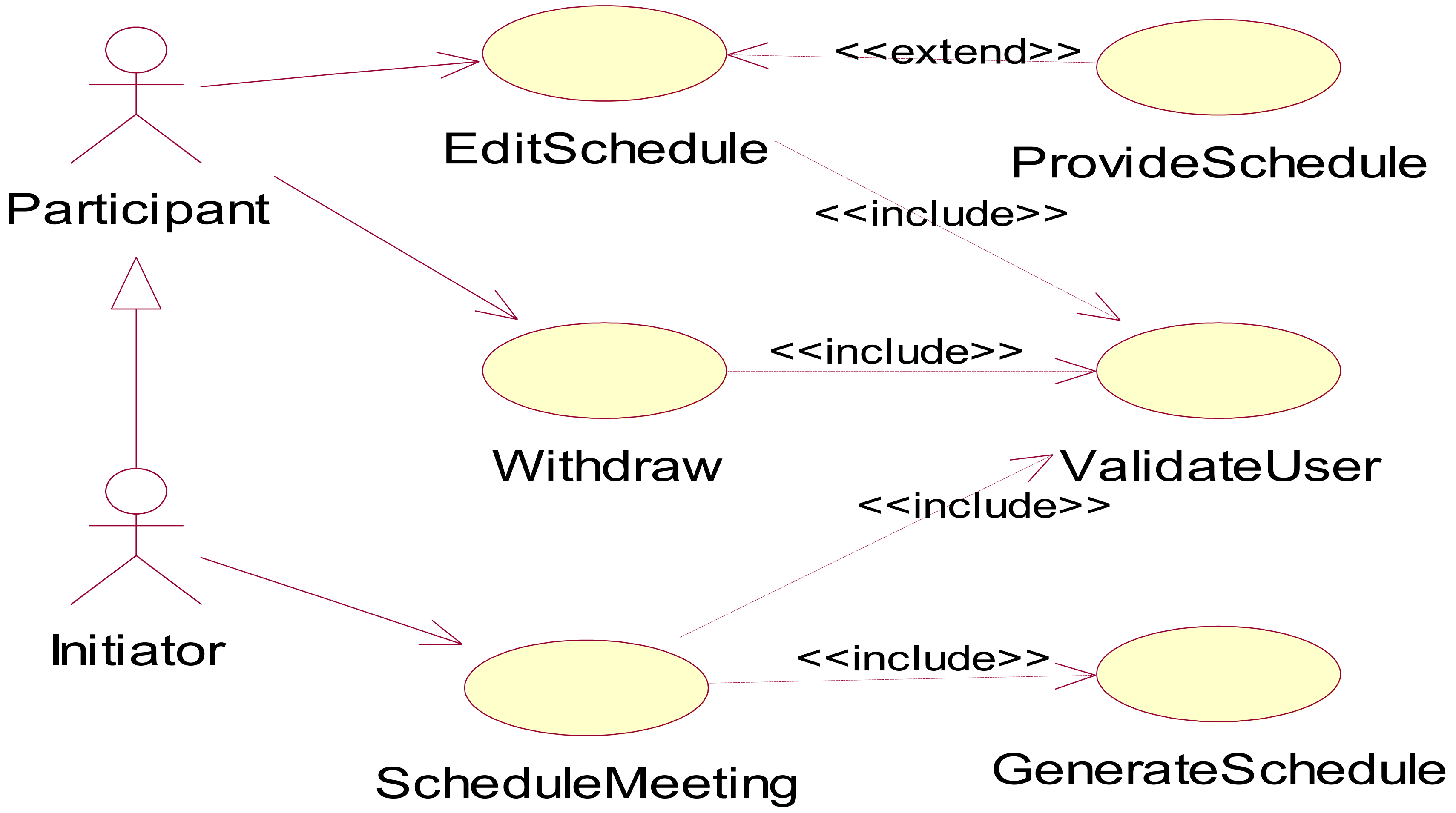
例 2：



<<include>>
ATM Session
Validate User
<<extend>>
<<extend>>
Query
Withdrawal
// extension points:
transaction possible,
receipt details


Print Receipt
<<extend>>
<<include>>
Authentication
Customer
Basic ATM Session
Bank Account Management System
<<extend>>
<<extend>>
<<extend>>
<<extend>>
<<extend>>
Balance Transfer
Cash Deposit (Optional)
Cash WithDrawal
Password Modification
Balance Inquiry

用例图实例二：日程安排系统



Class Exercise 1　问题描述

- 我们在为一家邮购公司开发订单处理软件，从供应商那里购买产品，再销售。
- 这家公司每年发布两次产品目录，并将其邮递给客户和其他感兴趣的人。
- 客户以提交商品列表并向邮购公司付费的方式购买商品。邮购公司填写帐单并把商品运送到客户的地址。
- 订单处理软件记录从收到订单直道商品被运送给客户的整个过程。
- 邮购公司将提供快捷的服务，以最快，最有效的方法来运送客户订购的产品。
- 客户可以退货，要求重新进货，但有时要付费。

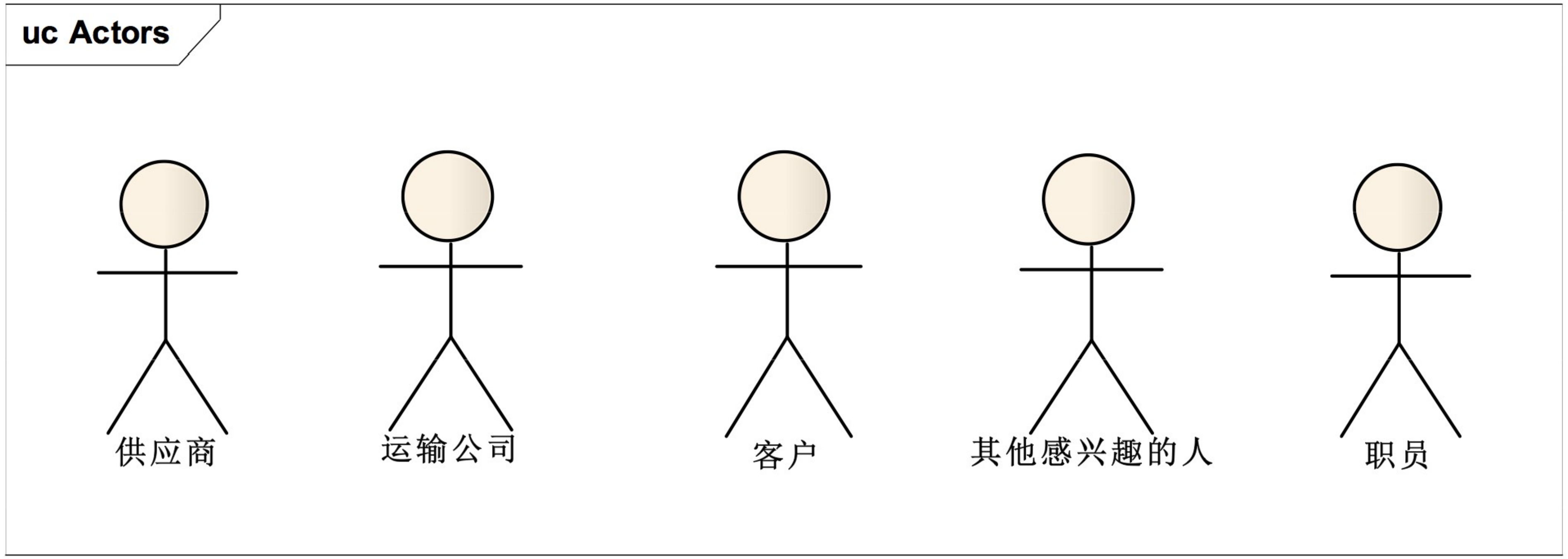
确定执行者— Actor

- 谁使用这个系统?
- 谁安装这个系统?
- 谁启动这个系统?
- 谁维护系统?
- 谁关闭系统?
- 哪些其他系统使用这个系统?
- 谁从这个系统获取信息?
- 谁为这个系统提供信息?
- 是否有事情自动在预计的时间发生?

干系人 (Stakeholder)?
主、次参与者 (Actor)? 系统? 无关?

- 我们（开发人员）
- 邮购公司
- 供应商
- 产品
- 产品目录
- 客户
- 特快专递
- 小红马
- 其他感兴趣的人
- 商品列表
- 运输公司
- 订单处理软件
- 邮购公司职员
- Mary Smith （邮购公司客户）
- Mary Smith （邮购公司职员）
- 记帐系统
- 库存系统

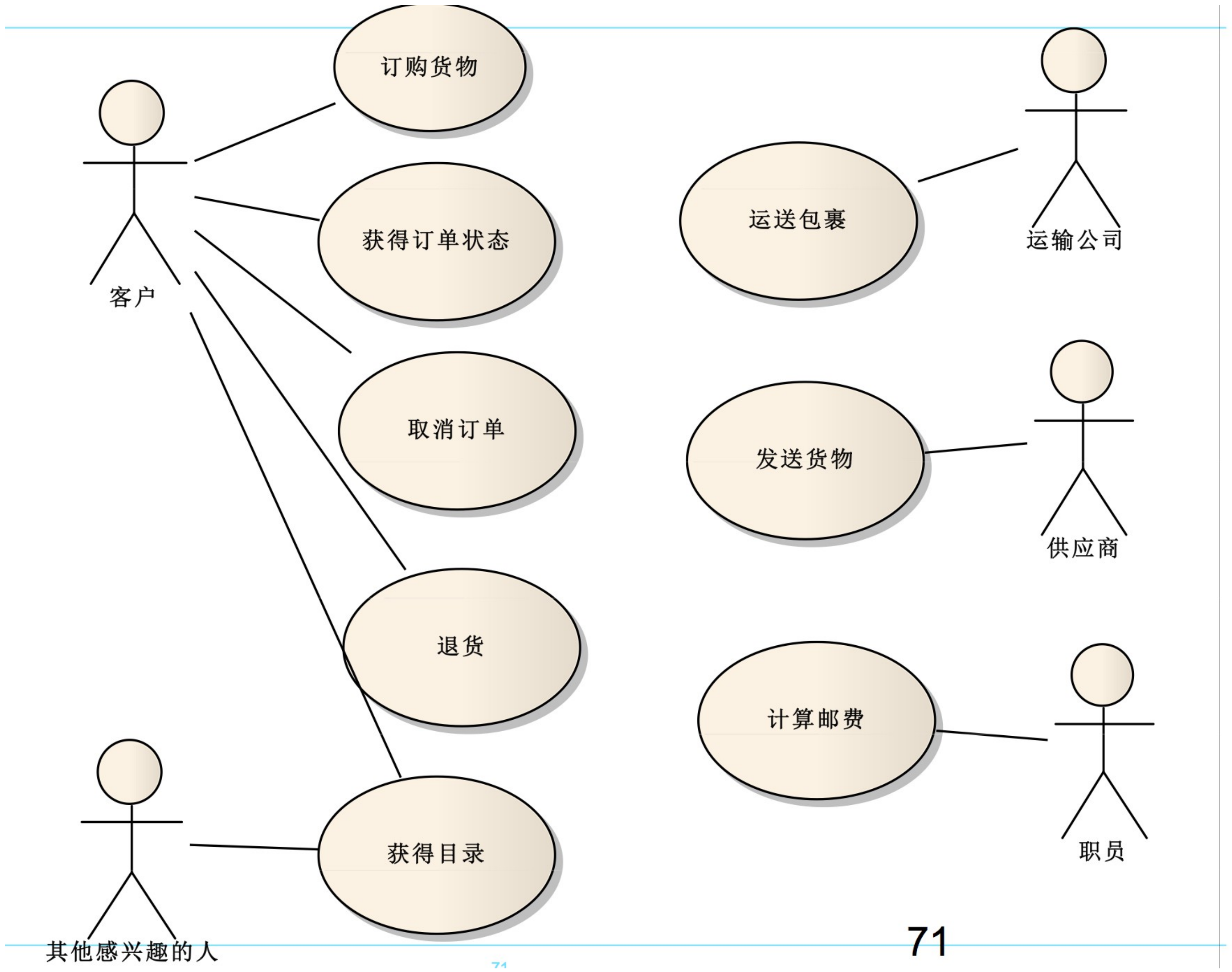
参与者



确定用例

- 执行者希望系统提供什么样的功能?
- 系统存储信息吗?
- 执行者将要创建、读取、更新或删除什么信息?
- 系统是否需要把自身内部状态的变化通知给执行者？系统必须知道哪些外部的事件？执行者将怎样通知系统这些事件?
- 其他需要考虑的用例包括启动、关闭、诊断、安装、维护、培训和改变商业过程。

用例


订购货物
客户
获得订单状态
取消订单
退货
获得目录
其他感兴趣的人
运送包裹
运输公司
发送货物
供应商
计算邮费
职员
71

撰写用例描述

UC01：订购货物用例

。前置条件：一个合法的用户已经登录到这个系统

事件流：

1. 当客户选择订购货物时，用例开始；
2. 客户输入想要购买的商品代码；
3. 系统逐项列出产品描述和价格；
4. 系统保存已经订购的产品清单；
5. 客户输入信用卡支付信息；
6. 客户选择提交；
7. 系统检验输入的信息；
   a. 如果提交的信息正确，保存订单，向记帐系统转发支付信息；
   b. 如果客户提交的信息不正确，系统提示用户修改；
8. 当支付确认后，订单就被标记上已确认，同时返回给客户一个订单 ID，用例结束。
   a. 如果支付没有被确认，系统提示客户改正支付信息或取消；
   b. 如果客户选择修改信息，就回到第 5 步；
   c. 如果选择取消，用例结束。

后置条件：如果订单未被取消，则被系统保存起来，并被标记为已确认

找出可选路径的方法

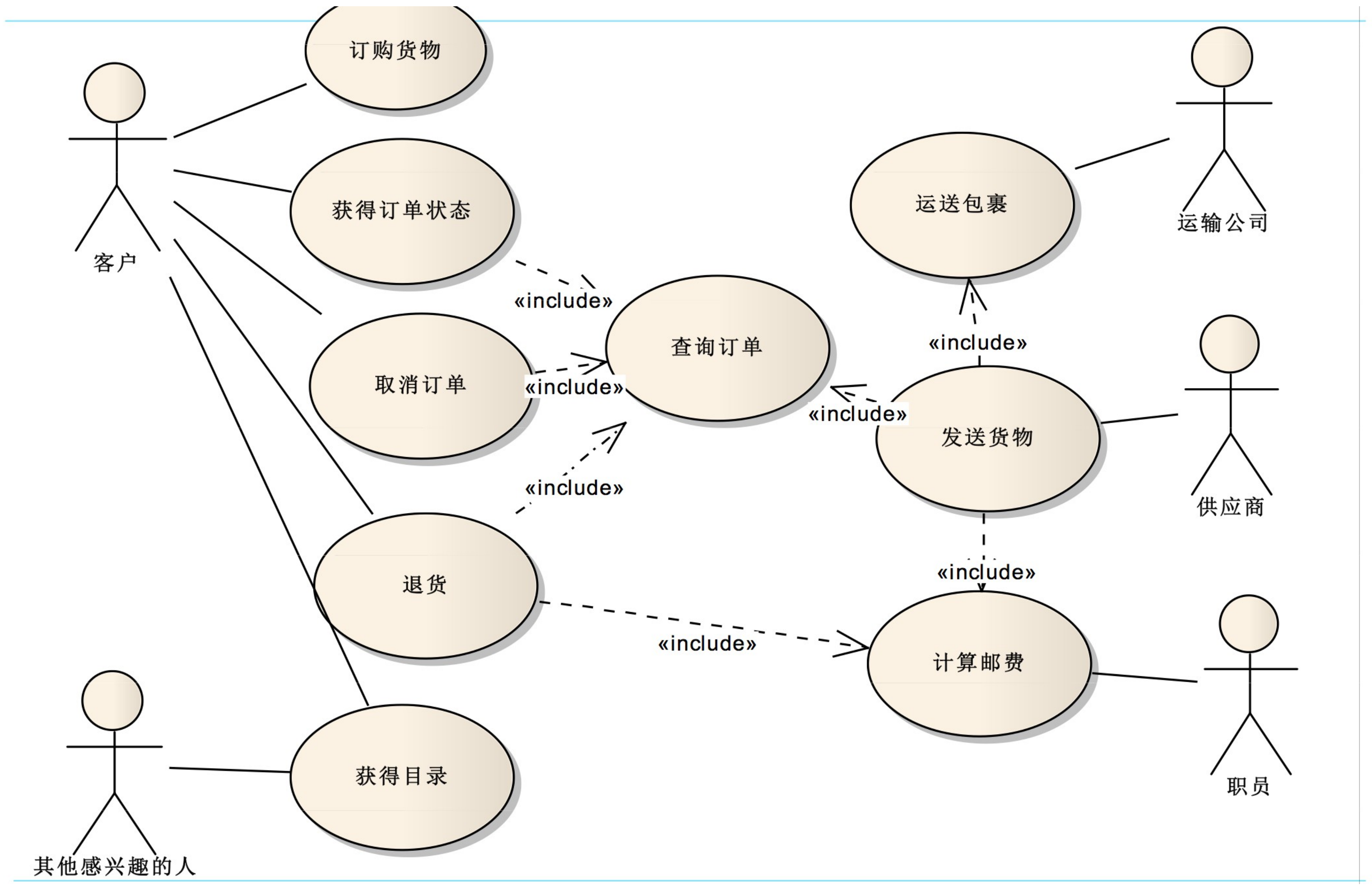
- 沿着基本路径一条一条的找，并且考虑:
    - 在这个点上还可以执行别的活动吗？
    - 在这个点上有没有什么可能出错的?
    - 有什么随时可能发生的行为吗?
- 或者，用以下大类去发现可选路径。例如:
    - 参与者退出应用程序;
    - 参与者取消指定操作;
    - 参与者请求帮助;
    - 参与者提供了“坏数据”；
    - 参与者提供了不完整数据;
    - 参与者选择了一个执行用例的可选方法;
    - 系统崩溃;
    - 系统不可用。

可选路径

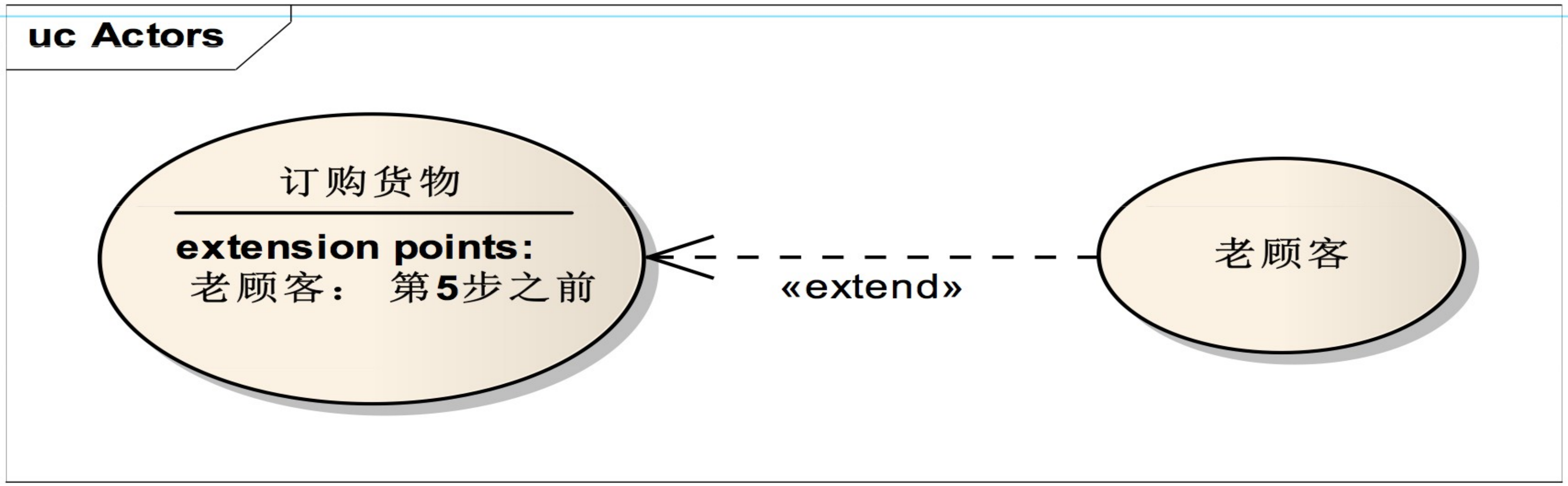
- 1. 不正确的商品代码；
- 2. 不正确的信用卡支付信息；
- 3. 填写订单过程中，用户选择取消；
- 4. 用户选择修改订单信息。

- 要用到搜索订单的用例有：
- 1. 取消订单；
- 2. 查询订单状态；
- 3. 退货。

修改后的用例


订购货物
客户
获得订单状态
«include»
查询订单
取消订单
«include»
«include»
退货
«include»
获得目录
其他感兴趣的人
运送包裹
运输公司
«include»
发送货物
供应商
«include»
计算邮费
职员

扩展用例及扩展点

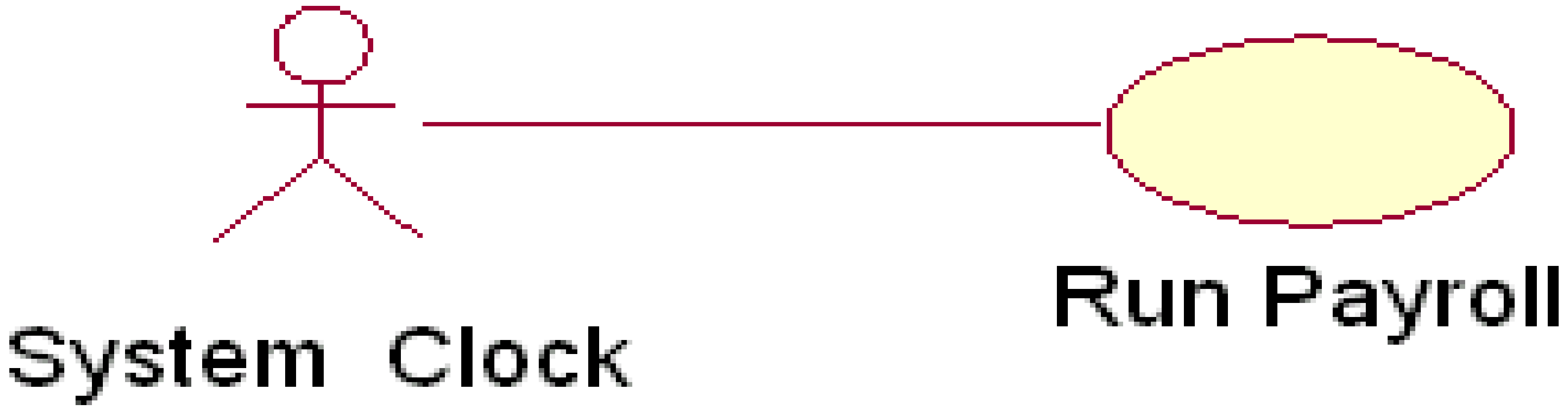


UC012： 老顾客打折用例
事件流:
- 顾客选择为老顾客时，用例开始;
- 系统提示客户输入顾客卡号＿；
- 顾客输入卡号后，系统验证卡号合法时;
- 系统为顾客计算打折信息。

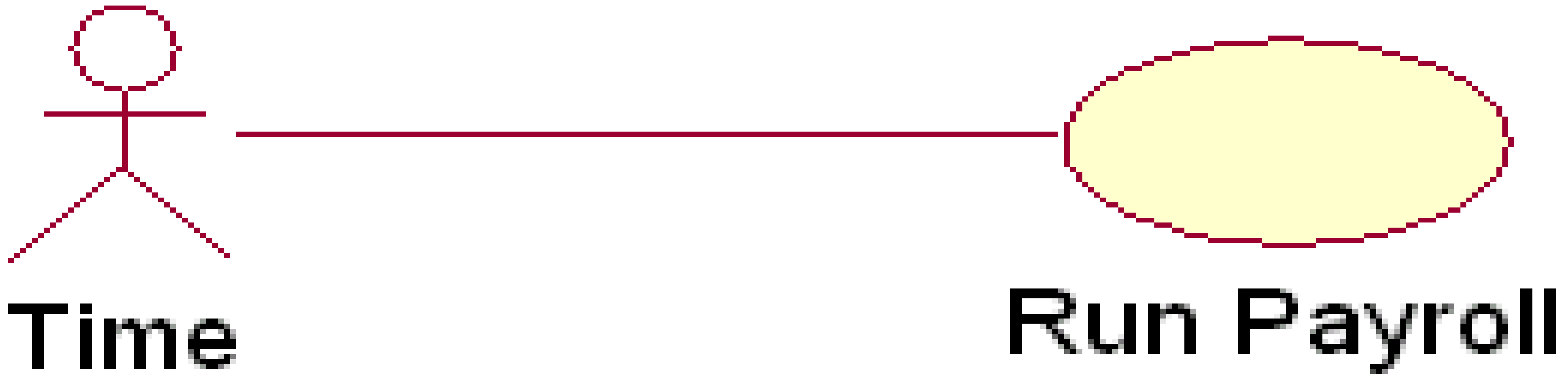
用例分析的例子:

- 假设要开发一个工资支付系统，其中有这样的需求:
  - 每个星期五以及每个月的最后一个工作日，系统自动生成一份员工工资册。

- 分析 1：考虑下面的用例图:

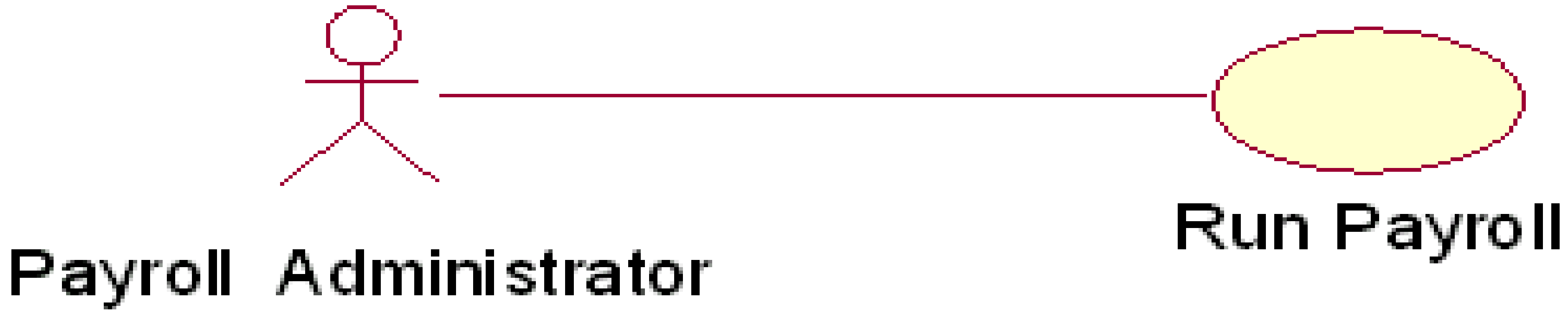


- 分析 2： System Clock 是设计时的问题。考虑改用下面的用例



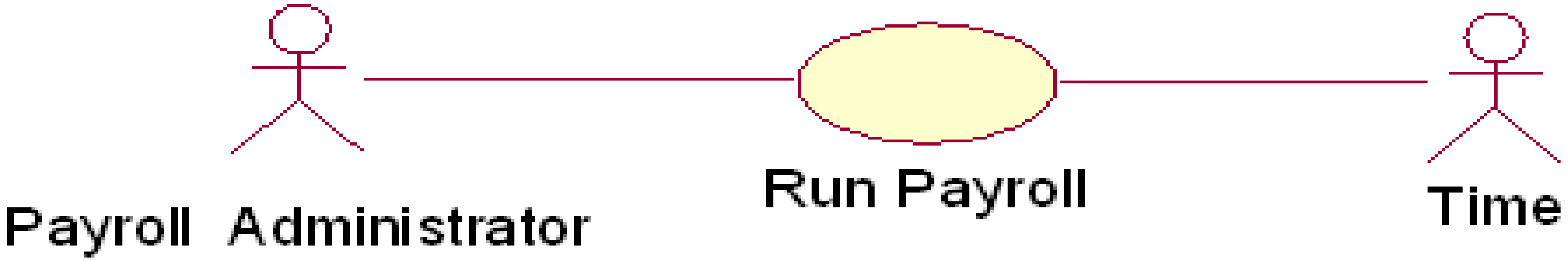
- Time 是否是真正的参与者?

- 分析 3： 系统的目标是什么？ 谁真正需要这个功能?
  - 可能真正的参与者是工资册管理员 (Payroll Administrator)



• 分析 4：如何处理需求中的“自动生成一份员工工资册”的问题？

- 考虑将 Time 作为次要参与者。



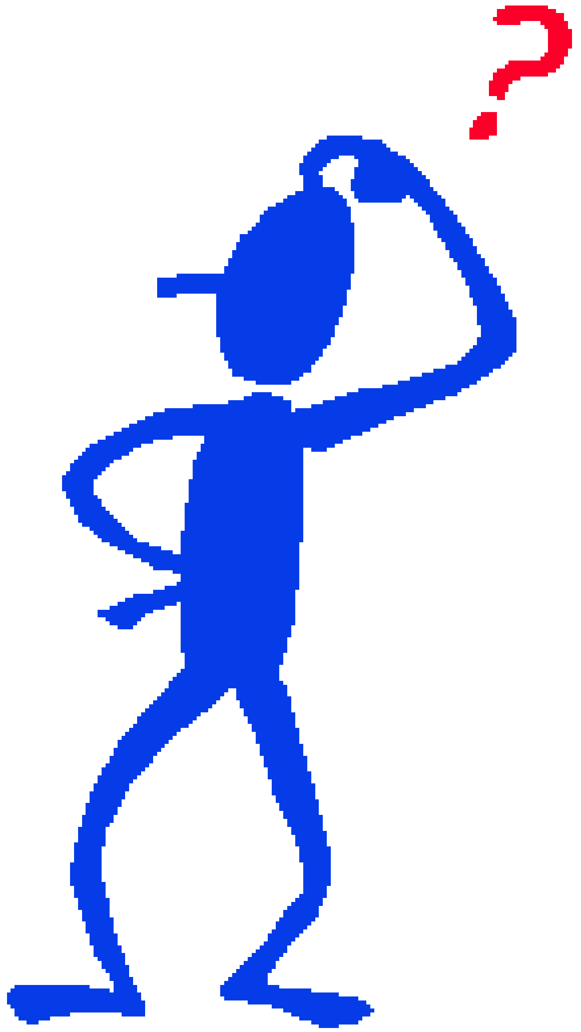
- 如果有必要在用例图中指明哪个用例是依赖于时间的，则这种方法很有效。
- 分析 3 也是可行的方法，用例图中不指明哪个用例是依赖于时间的，但在用例描述中说明。

常见问题分析

1. Use case 的粒度问题，即对于一个系统的 Use Case 图，所包含的用例数目问题。

- 这是很多人争论的重点。例如， Ivar Jacobson 说，对一个十人年的项目，他需要二十个用例。而在一个相同规模的项目中， Martin Fowler* 则用了一百多个用例。

* 《 UML distilled 》一书的作者



2. 许多应用中需要对系统访问进行控制，应该如何处理登录问题比较好?

- 有四种处理登录用例的方式:

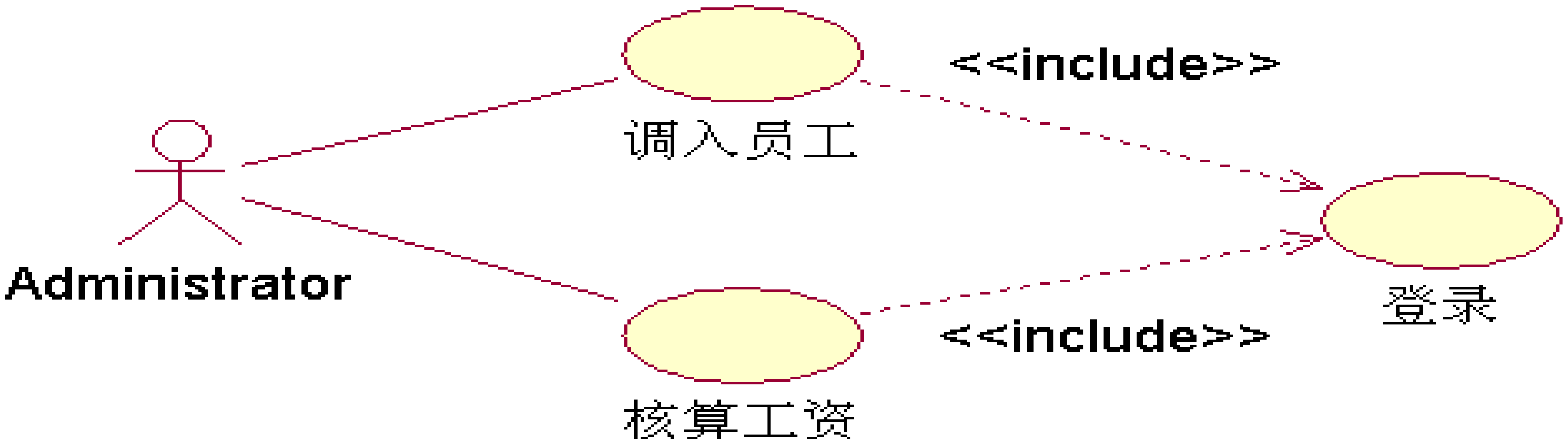
(1) 其它用例包含登录;

(2) 其它用例扩展登录;

(3) 登录独立于其它用例;

(4) 登录包含其它用例。

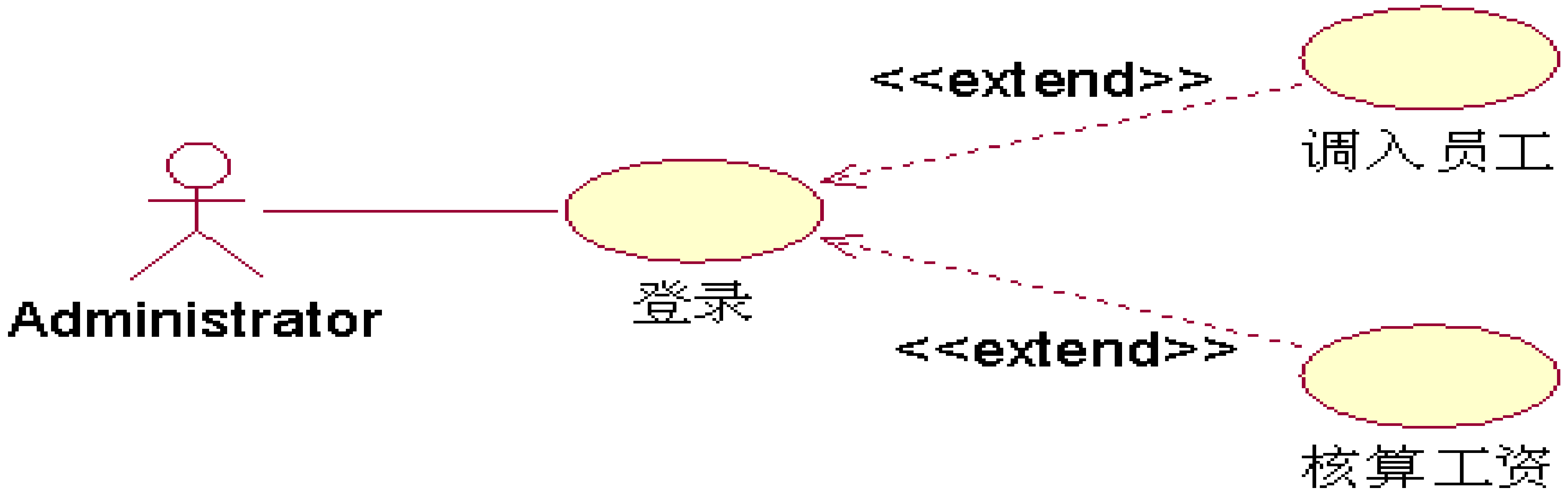
(1) 其它用例包含登录



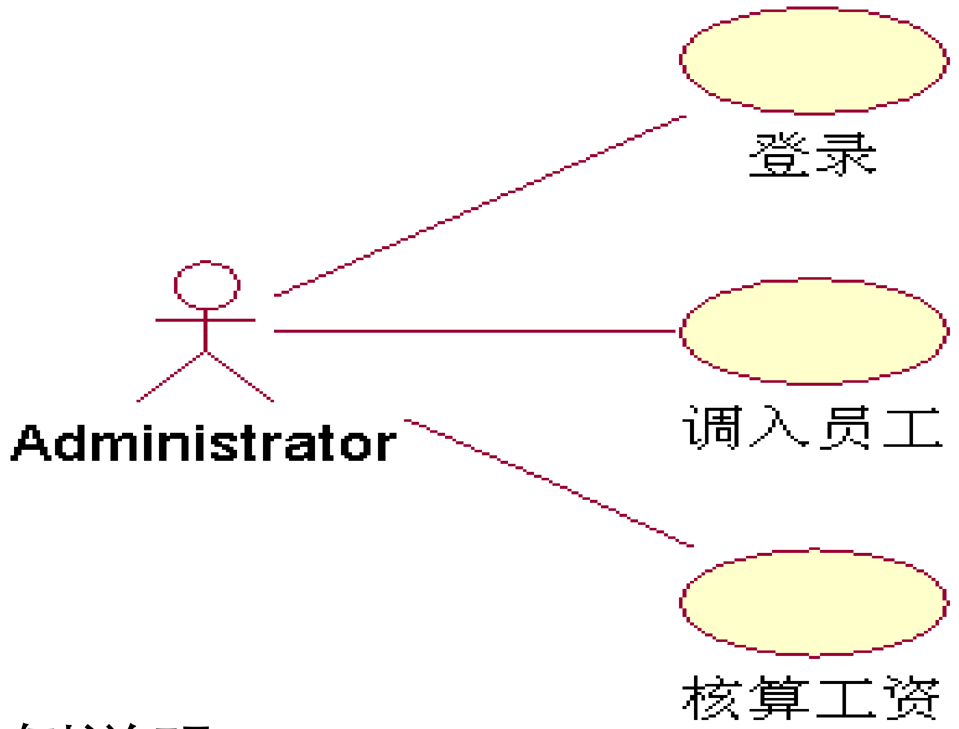
用例说明：……

这种处理方式的特点：

(2) 其它用例扩展登录



(3) 登录独立于其它用例。



用例说明：……

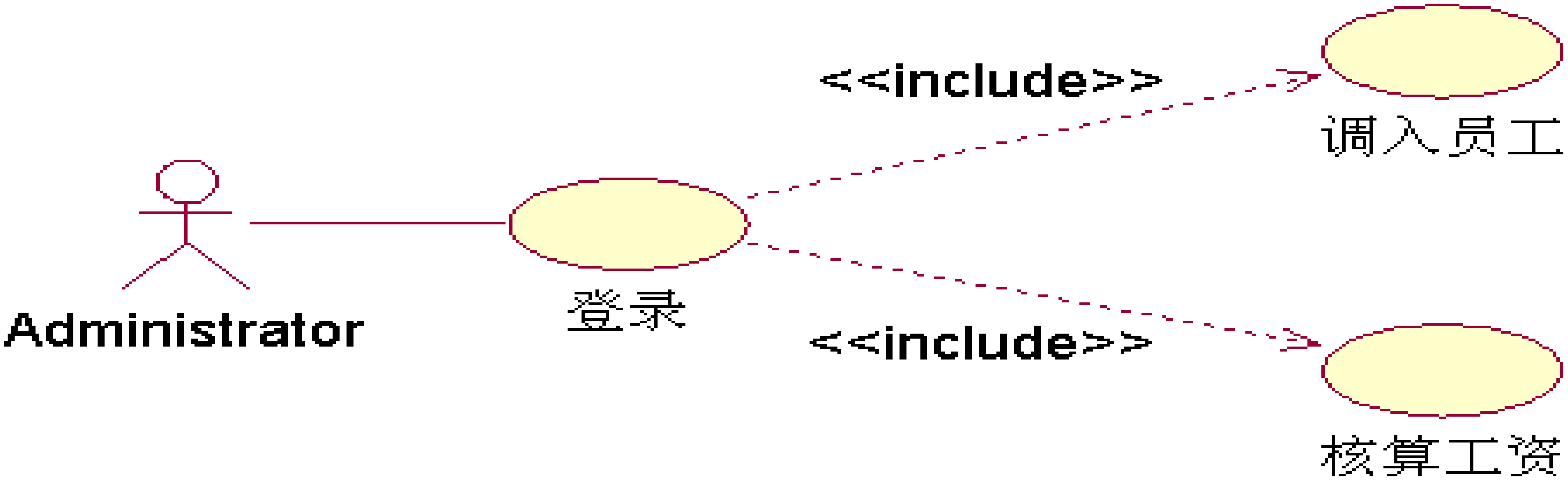
这种处理方式的特点：

- 登录用例说明：
  1. 当超级用户启动应用时用例开始
  2. 系统提示超级用户输入用户名和密码
  3. 超级用户输入用户名和密码
  4. 系统验证其是否为有效超级用户
  5. 用例结束

- 调入员工用例说明：

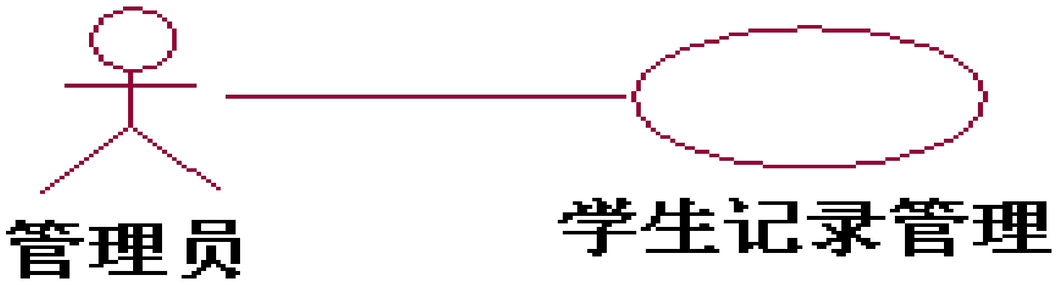
前置条件：一个有效超级用户登录了系统

(4) 登录包含其它用例

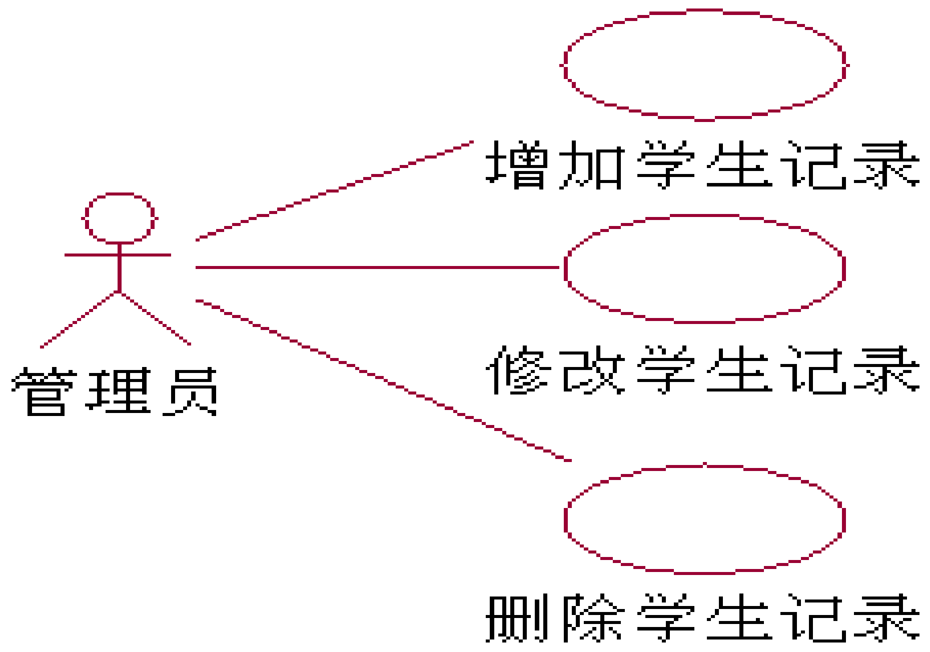


存在的问题：

3. 假设有这样需求：学生档案管理中，用户经常需要做三件事：增加一条学生记录、修改一条学生记录，删除一条学生记录。如果要画出 use case 图，以下 2 种方法哪种更合适?

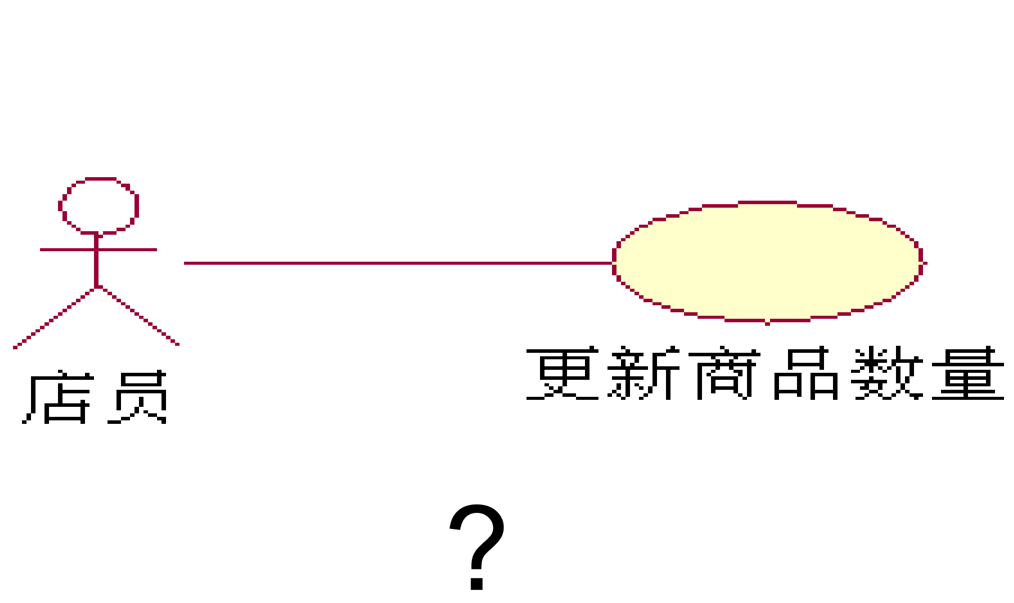


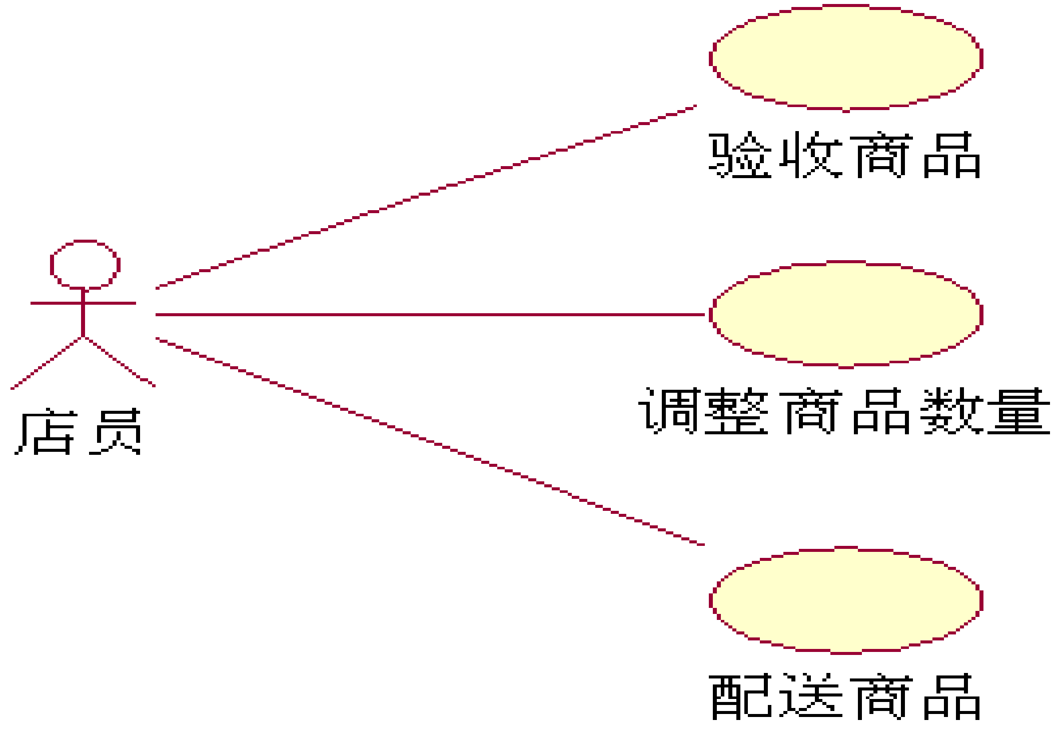
方法 1：再分成 3 个脚本，分别画 3 个交互图。脚本 1 增加学生记录；脚本 2 修改学生记录；脚本 3 删除学生记录。



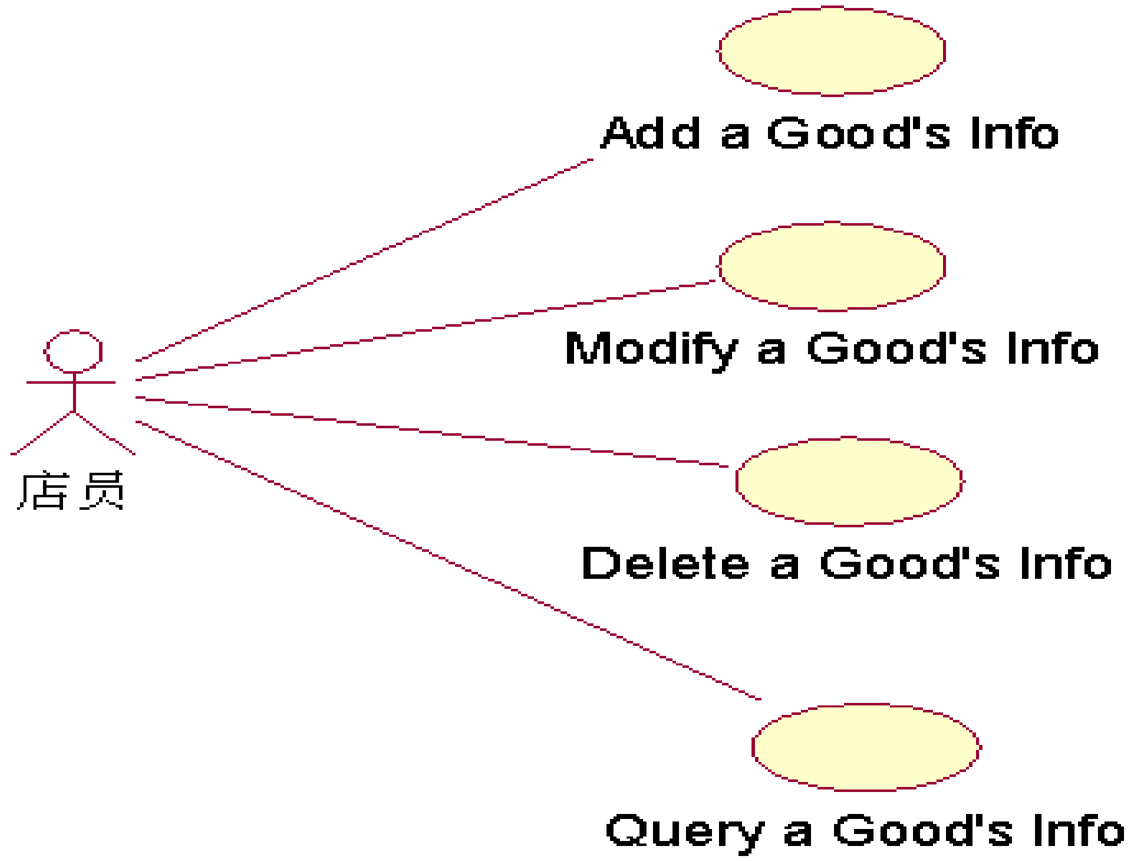
方法 2：每个 use case 画一个交互图。

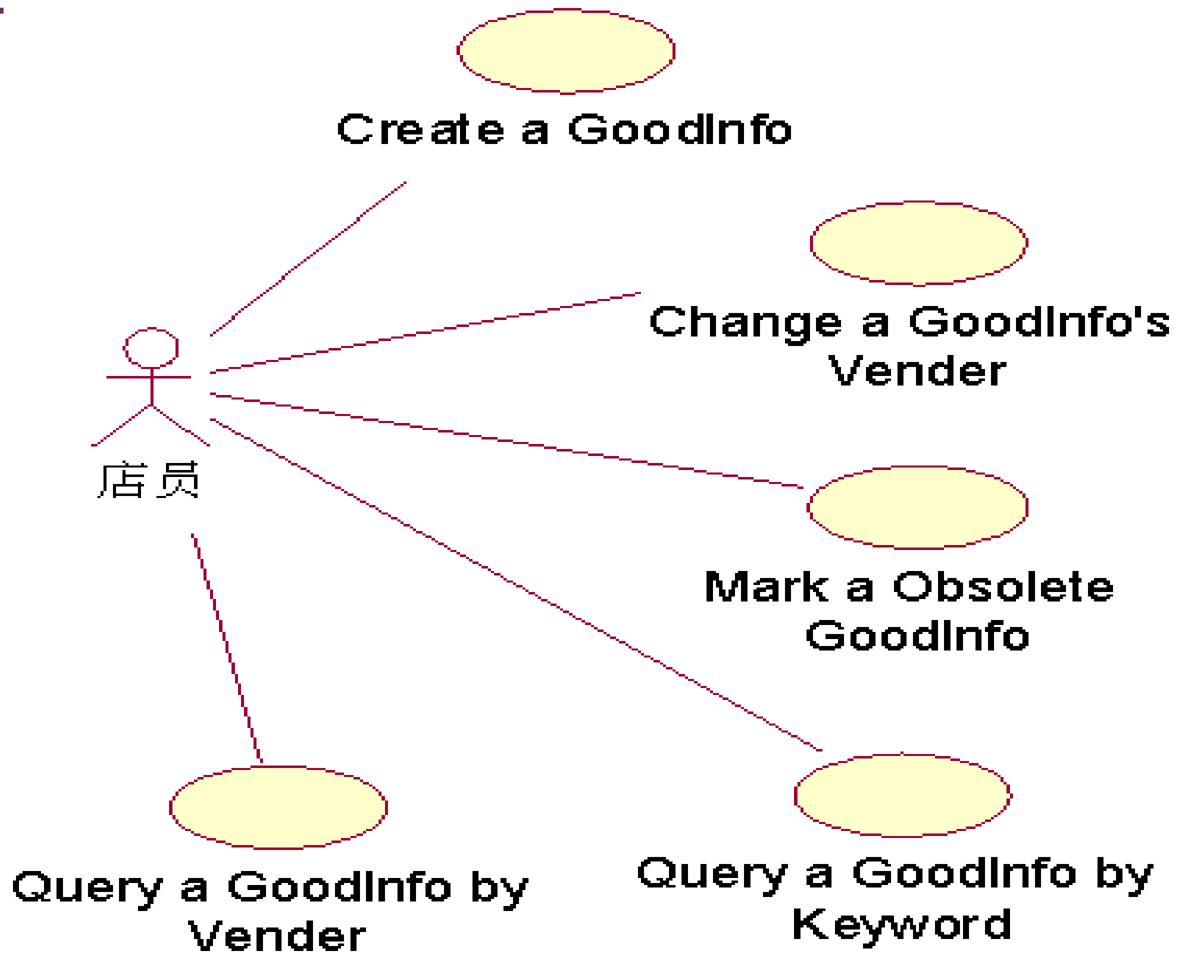
- **Create, Retrieve, Update, Delete 类型用例的处理：**
  - 采用 CRUD 四个用例还是一个用例？
  - 从用户需求的角度考虑而不是从数据处理的角度考虑。
- 例 2:





- 例 3:


Add a Good's Info
Modify a Good's Info
Delete a Good's Info
Query a Good's Info
店员


Create a GoodInfo
Change a GoodInfo's Vender
Mark a Obsolete GoodInfo
Query a GoodInfo by Vender
Query a GoodInfo by Keyword
店员

?