UML (Unified Modeling Language)

2 Use Cases

Software Institute of
Nanjing University

Use cases

- Use cases are a technique for capturing the functional requirements of a system.
  - Use cases work by describing the typical interactions between the users of a system and the system itself, providing a narrative of how a system is used.

1、需求分析

- 软件产品建造的第一步：需求分析
  - 根据用户对产品的功能的期望，提取出产品外部功能的描述。
- 用例视图：
  - 支持产品外部功能描述的视图。
  - 从软件产品的使用者的角度而不是开发者的角度，描述用户对待开发的产品的需求分析产品所需的功能和动态行为。

1、需求分析（cont’）

- 使用者软件产品的功能的考察
  - 功能的设置的合理性
  - 运行效率
  - 功能使用的方便程度
- 这就是软件产品的外部特性
  - 软件系统通过其边界呈现给用户的特性
  - 外部特性通常是动态的
  - 通过外部特性，软件系统的使用价值得到了体现
  - 呈现在软件系统的边界上的外部特征是由软件系统的内部实现决定的

1、需求分析（cont’）

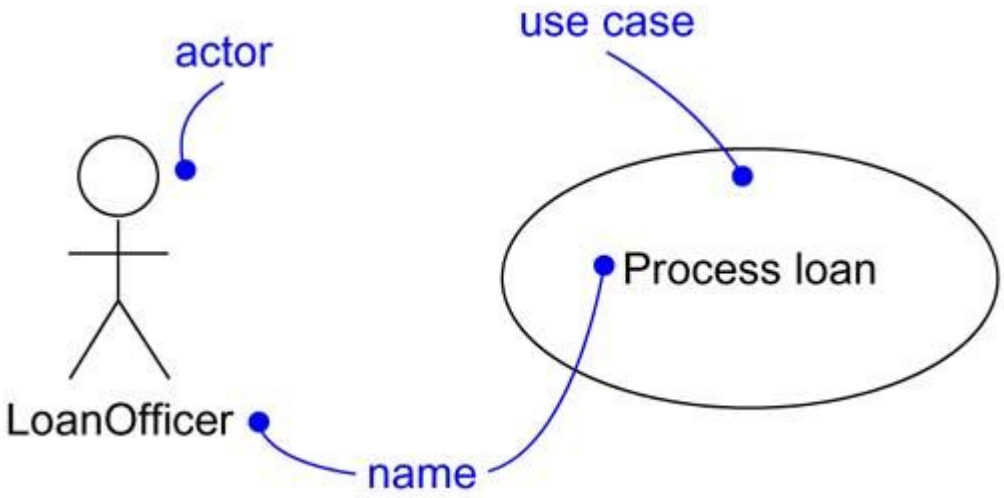
- 软件系统的使用价值是通过和它所存在的环境发生交互实现的
- 分析软件产品与外界环境的联系
  - 分析共有哪些用户将要使用软件系统
  - 分析软件系统的运行结果将要对哪些对象产生影响
- 为这些对象与软件系统进行的交互指定具体内容
  - 对交互中包含的动态行为进行详细描述
  - 对这些动态行为进行合理的分类和组织

2、系统作用者（actor）

- 对软件产品的需求分析就是定义软件系统的边界，包括两个方面的内容
  - 第一、分析软件产品与外界的联系
  - 第二、确定软件产品与外界的联系时包含动态行为及其相互关系。
- 在UML中，下列建模元素为上述两个方面的内容提供支持
  - 系统作用者(actor)
  - 用例(use case)

2、系统作用者（actor）（cont’）

- 系统作用者代表位于系统之外并和系统进行交互的一类对象（图）。用它可以对软件系统与外界发生的交互进行分析和描述。软件系统在和外界发生交互时涉及的具体的对象，在UML里，用系统作用者来建模。



2、系统作用者（actor）（cont’）

- 可以用系统作用者建模的对象
  - 软件系统的使用者：软件系统的使用者要通过使用软件和系统交互。使用者处于系统之外
  - 还可以代表直接和软件系统交互的软件系统赖以运行的软/硬件平台
  - 以及与软件系统有信息交换的其他计算机系统
- 系统作用者的数目和应用情况确定

3、用例（use case）

- 指定系统作用者以后
  - 详细描述系统作用者和软件系统交互的具体内容
  - 对交互所代表的软件系统的功能进行分类
  - 对这些功能详细指定其代表的软件系统的动态行为
- 在UML里，软件系统的功能和其代表的动态行为是用用例来建模的

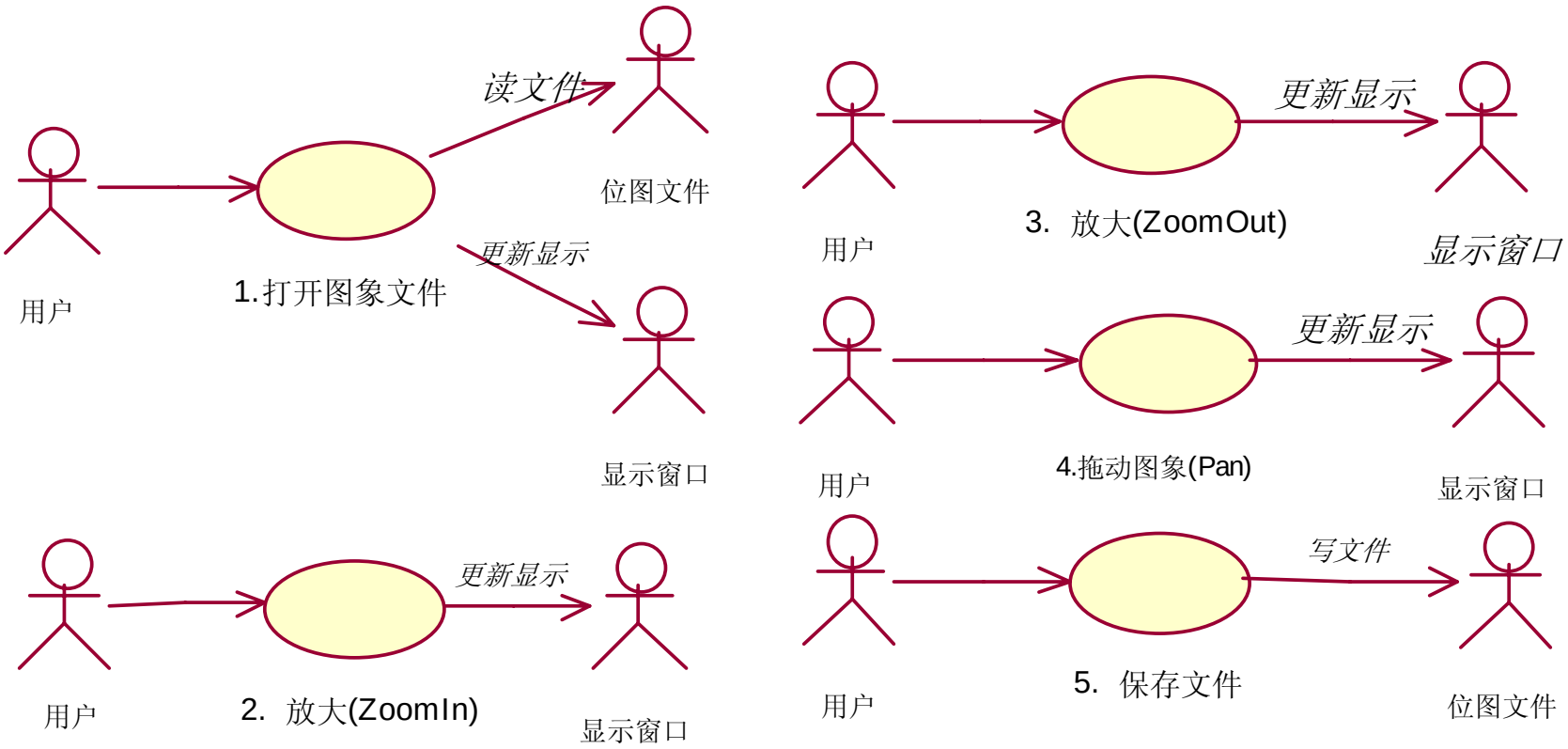
3、用例（use case）（cont’）

- 用例代表系统为响应系统作用者引发的一个事件而执行的一系列的处理，而且这处理应该为系统作用者产生一种可见的价值
- 用例描述了当系统作用者和软件系统进行交互时，软件系统所执行的一系列的动作序列。
- 这些动作序列
  - 不但应包含正常使用的各种动作序列，
  - 而且应包含对非正常使用时软件系统的动作序列的描述。

4、关联关系

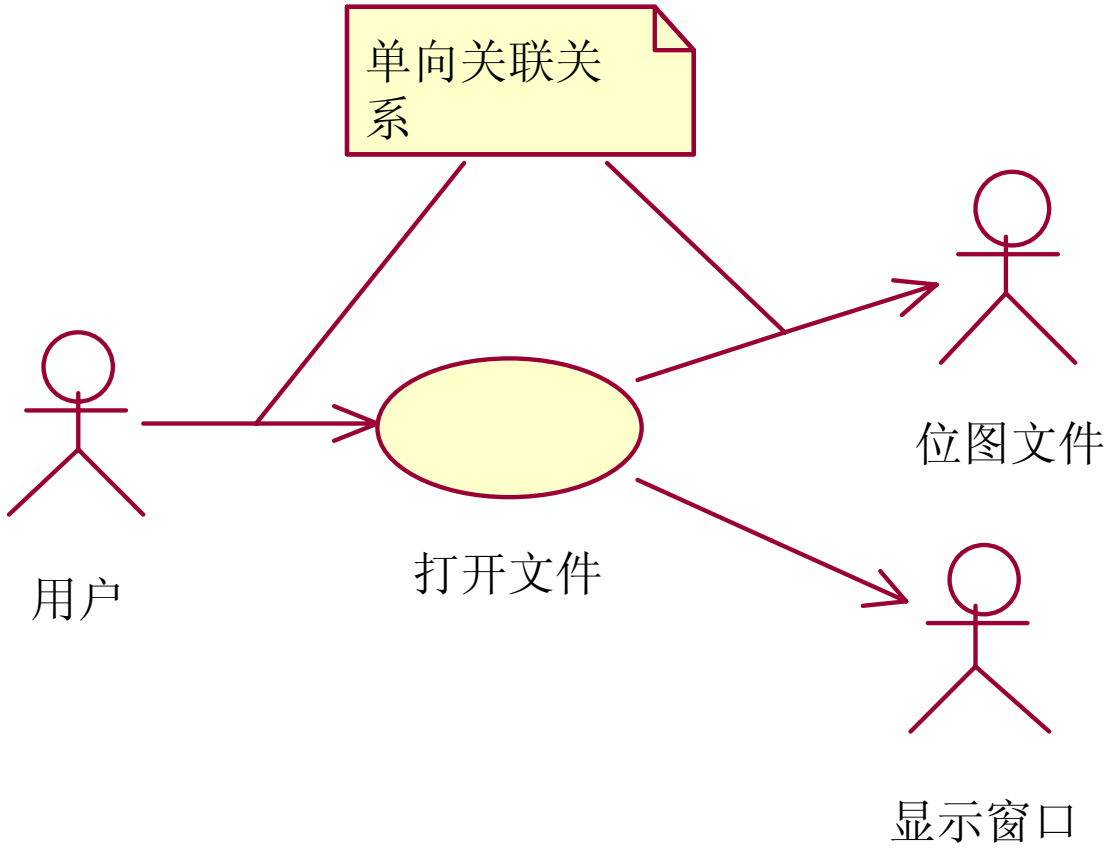
- 系统作用者和用例之间的联系：关联关系
  - 例子：通过用例描述软件系统的功能。设计开发一个名为“位图观察器(bmpviewer)”的软件产品，它为用户提供图像显示和浏览的功能。
    - 打开一个bitmap文件
    - 可对文件进行放大显示(Zoom-in)
    - 可对文件进行缩小显示(Zoom-out)
    - 可对文件进行浏览(Pan)
    - 可将文件继续保存到磁盘

4、关联关系（cont’）


读文件
位图文件
更新显示
用户
1.打开图象文件
显示窗口
更新显示
3. 放大(ZoomOut)
用户
显示窗口
更新显示
4.拖动图象(Pan)
用户
显示窗口
更新显示
用户
2. 放大(ZoomIn)
显示窗口
写文件
用户
5. 保存文件
位图文件

4、关联关系（cont’）

这条直线在UML里代表关联关系。
例如，上页图中“用户”和“打开文件”用例之间的的关联关系代表用户对软件系统的打开文件功能的使用。



4、关联关系（cont’）

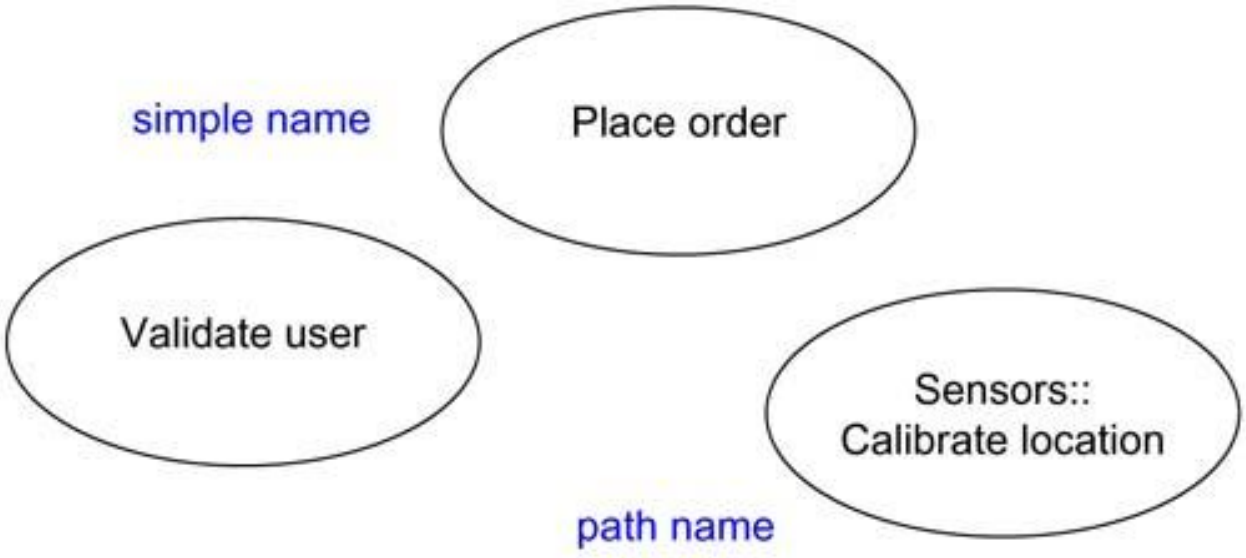
- 关联关系是UML的关系的一种，它代表两个类的对象之间的语义联接。未经过修饰的关联关系是双向的。
  - 在用例图中使用的关联关系当被修饰为单向的关联关系，它在模型图被表示为一条带箭头的直线
  - 单向关联关系是关联关系的一个特例：它由关联关系经修饰而得，它意味着访问是有向的。

5、Usecase和Actor的绘制

- 在用例图上，系统作用者、用例和关联关系都有一个名字来标识(图),名字被设置在相应的模型元素的图符附近。
- 用例的名字分为两种
  - 简单名字:是一个不包含冒号的字串
  - 路径名字:路径名则是简单名字前面加上一个包含此用例的所在的模型包的名字。它们之间用两冒号隔开。

5、Usecase和Actor的绘制

- Use case的命名



6、Use Case重要概念

- 事件流（Flow of Events）
  - 主事件流 main flow of events
  - 次要事件流 alternative flow of events
- 场景 Scenario
- 协作 Collaboration
- 关系 relationships
  - 泛化关系 generalization
  - 包含关系 include
  - 扩展关系 extend

6. 1 事件流 Flow of Events

- 用例代表系统和系统作用者之间发生的一系列的事件，这些事件流构成了用户对系统功能的一次使用。
- 在用例图中必须对事件流进行描述，以构成一个完备的用例模型。
  - 通常包括：简要说明、前提条件、主事件流、其他事件流和事后条件。

6. 1 事件流 Flow of Events (cont')

- 对事件流的描述包括四种形式，即：
  - 形式文本
  - 非形式文本
  - 交互图
  - 状态图
- 有相应的UML成员作为这些描述的载体
  - 文本和正式文本可用注标（note）表示，
  - 交互图和活动图本身即是一个标准的UML成员。

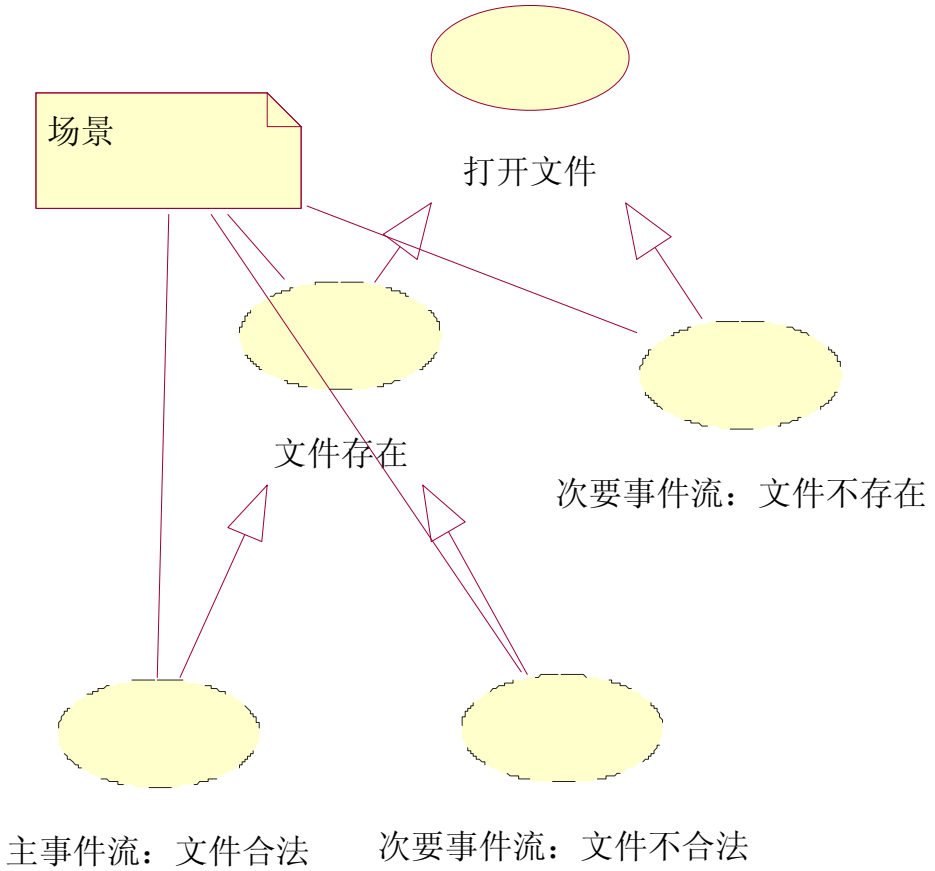
6. 1 事件流 Flow of Events (cont')

- 主事件流（main flow of events）和次要事件流（alternative flow of events）用来区分对系统功能的合法使用和非法使用
- 主事件流（main flow of events）
  - 合法使用
  - 只有一个
- 次要事件流（alternative flow of events）
  - 可包含若干个

6.2 场景（Scenario）

- 一个用例可以有多个事件流过程，每一个事件流过程是一个完整的用户对系统的功能的使用。
  - 它们出于同一个目的，但出于事件流过程不同，使得系统产生的响应不同
- 这样的用例/事件流过程的对应关系， 它是对系统功能使用的特例（合法或非法的），因此事件流过程是用例的实例（instance）。
- 用例的实例在UML中被称为场景（Scenario）。

6.2 场景（Scenario）（cont’）



例子说明

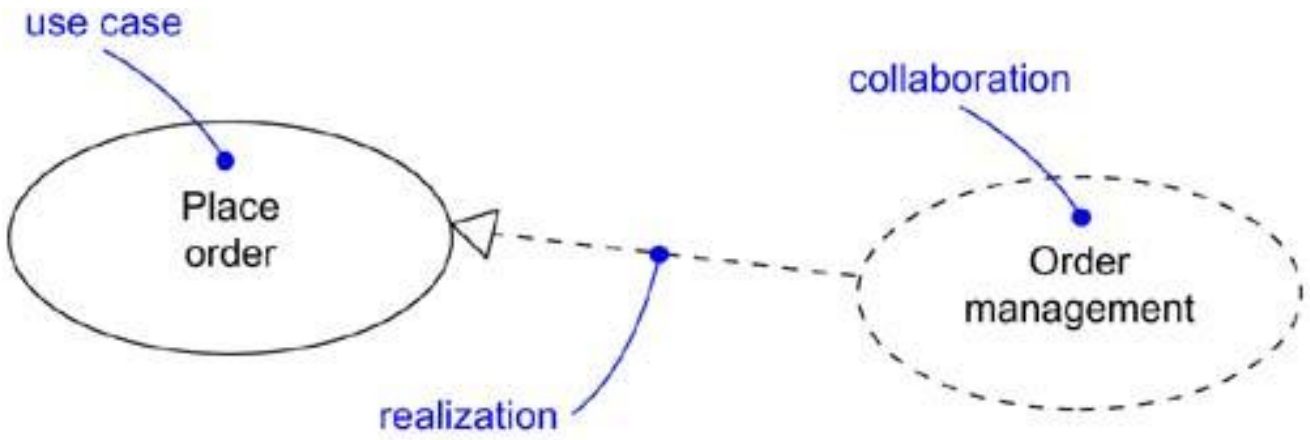
- 关于事件流和场景的例子
  - 事件流说明和文档格式

6.3 协同（Collaboration）

- 用例对应着系统内部的一系列的建模元素，这些建模元素实现了用例规定的动态行为
- 这些建模元素中既包括结构建模元素也包括行为建模元素,它们的共同工作构成了用例的动态行为的实现
- 这些建模元素,在UML中，被称为协同（Collaboration）

6.3 协同 Collaboration (cont')

- 接口和实现的划分
  - 用例是接口，协同是其实现
  - 用例/协同之间的规定(接口)与实现的关系称为实现关系(realization), 见下图



6.3 协同 Collaboration (cont')

- 协同包括的内容
  - 动态内容在UML里可以用通讯图和顺序图表示
  - 静态内容在UML中可以用类图表示。
- 在工具中，通常将表达协同的动态结构和静态结构的模型图（顺序图、通讯图、类图、状态机图）附加到协同实现的用例上
- 这样，通过工具提供的链接浏览功能，可以方便地进入相应的表达协同的动态或静态特性的模型图

6.4 关系 Relationships

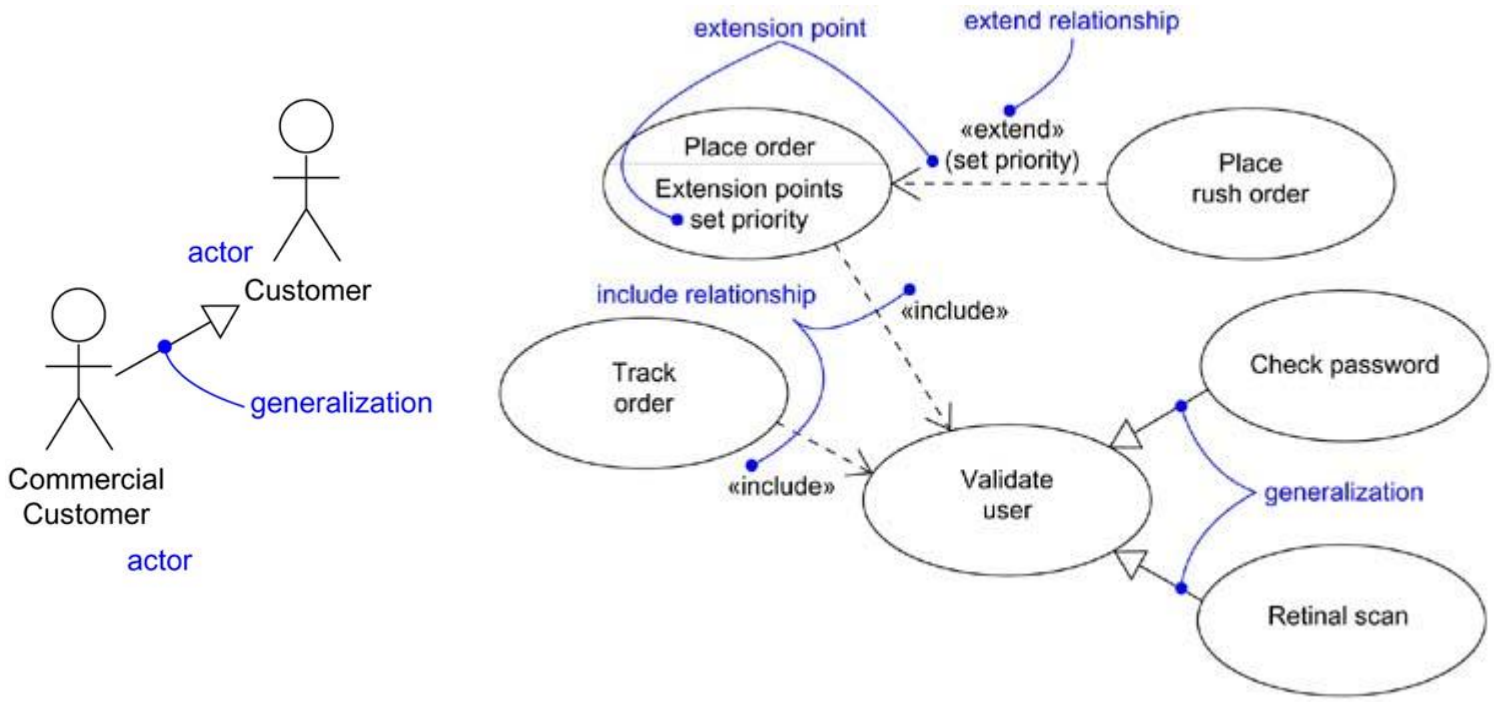
- 关系 Relationships
  - 泛化关系 generalization
  - 包含关系 include
  - 扩展关系 extend

6.4.1 泛化关系 generalization

- 在产生的用例或系统作用者的语义和行为之间的联系比较复杂时如果各个不同用例或系统作用者之间具有语义或行为方面的继承性
- 可以用泛化关系(generalization)对它们组织，就如同用泛化关系描述基类和导出类之间的关系一样。
- 可以用泛化关系类描述系统作用者或用例之间的继承。

6.4.1 泛化关系 generalization

- 包括actor和use case的泛化关系



6.4.2 包含关系(include)

- 位于两个用例之间的包含关系意味着基用例显式地在其指定位置将另一个用例包含进来, 使其成为自己的行为的一部分
- 包含关系可用于提取共用的用例

- 在具有包含关系的两个用例中，被包含的那个用例不能单独存在，它只能以实例的形式存在于包含它的用例之中

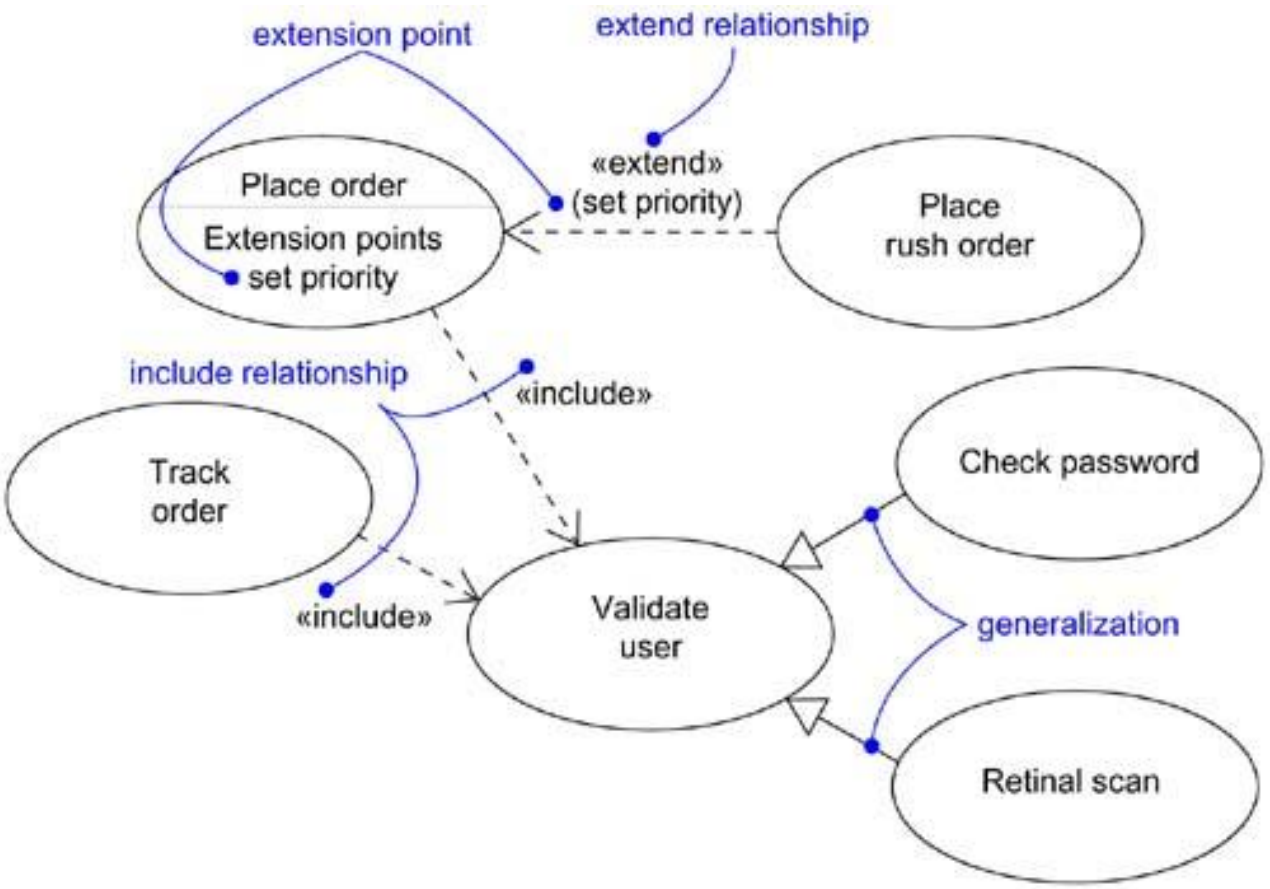
6.4.3 扩展关系 （extend）

- 两个用例之间的扩展关系，代表基用例可以隐式地包含另一个用例作为其行为的一部分，包含的位置间接地由另一个用例（扩展用例）确定。基用例可以独立于扩展用例单独存在。
- 当一个用例有多个子流程时，可以用扩展关系对其进行扩展，使得此基用例的不同子流程能在不同的情形下以扩展用例的形式被激活

示例 include & extend

- 在任何一种图形表示中，箭头所指的模型元素分别代表被包含的用例或被扩展用例(基用例), 而包含关系和扩展关系的构造型标记分别是<<include>>和<<extend>>

示例 include & extend （cont’）



7、用例的组织和用例图

- 在UML里, 用例图是表达用例和系统作用者及其之间关系的载体
- 用例图是模型图,用例图可包含
  - 用例
  - 系统作用者以及
  - 它们之间的关系这些关系可以是关联关系,泛化关系,扩展关系,包含关系,实现关系

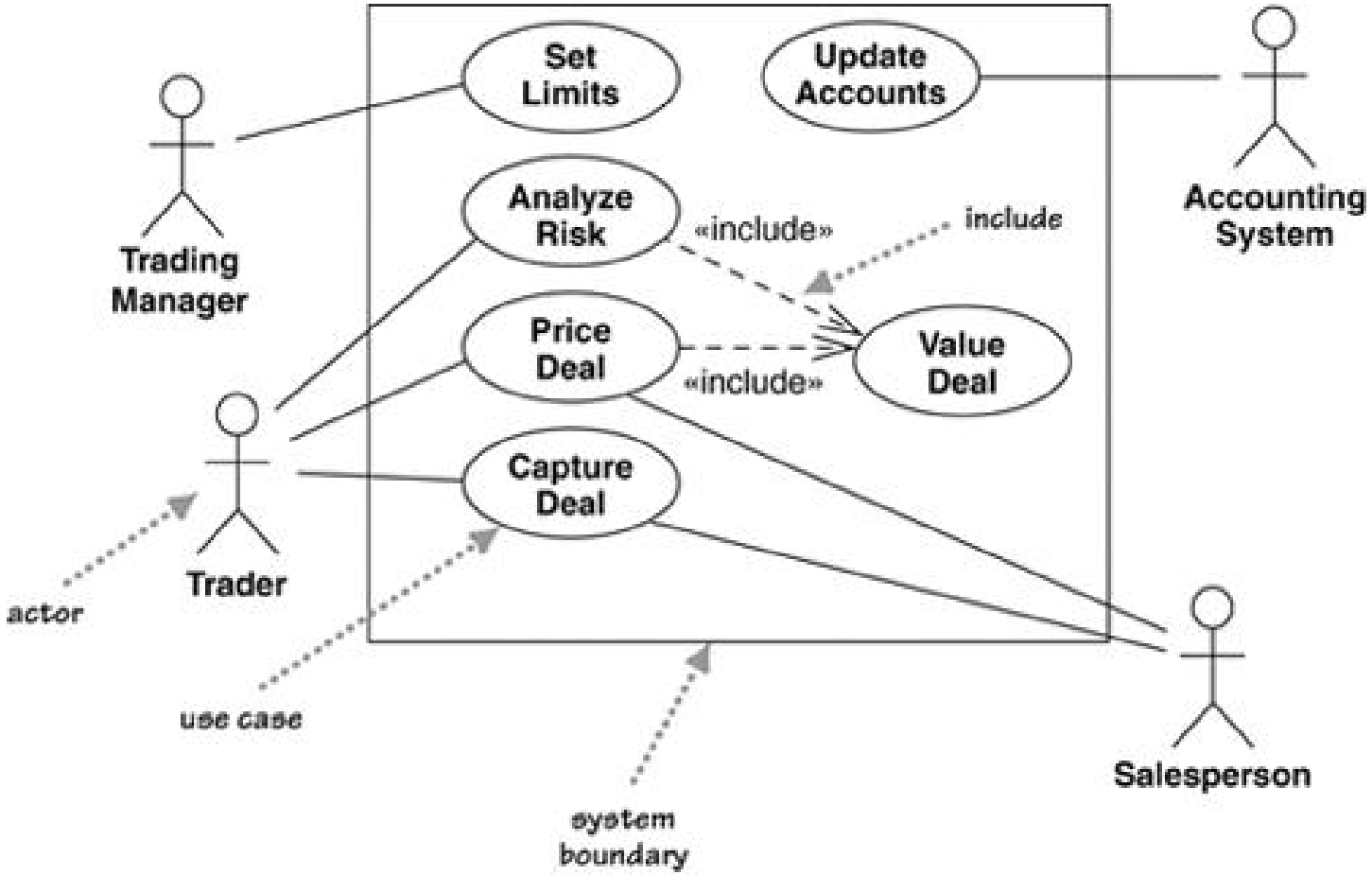
7、用例的组织和用例图（cont’）

- 用例图的用途是为,软件系统,软件子系统,类的动态行为建模,它从两个方面对其建模对象的内容进行描述，即：
  - 第一、描述它们的边界；
  - 第二、对它们进行需求分析。

7.1 定义软件系统的边界

- 当用用例图来描述系统的边界时，
  - 首先要考察位于系统外部并且和系统打交道的那些对象。这些对象包括
    - 系统的使用者: 如各种不同的用户、系统管理员等；
    - 和系统发生关系的位于系统外部的软件或硬件子系统如：文件系统、显示系统、打印机、传感器等
  - 把这些外部对象标识为系统作用者。
  - 最后，在用例图上用关联关系标出系统作用者和用例之间的语义连接。

7.1 定义软件系统的边界（cont’）



7.2 用例级别 Levels of Use Cases

- 风筝级别 kite-level
- 海洋级别 sea-level
- 海鱼级别 fish-level

7.3 需求建模

- 需求是一个系统的设计规格、属性以及动态行为
  - 规格和属性决定系统的性能
  - 动态行为决定系统的功能
- 需求可以
  - 用非形式的文本来表达，或者
  - 用形式化的文本表达，还可以
  - 用形式化程度位于此次两者之间的任意的方式来表达
    - 用例图就是这样的方法之一
    - 还有状态图

7. 3 需求建模（cont’）

- 在考察系统的动态行为时
  - 既要考虑功能被正确使用时的主事件流，
  - 又要考虑功能被非法使用时的次要事件流的正确处理
- 要把这些动态行为用系统用例、系统作用者以及他们之间的关系在用例图上表示出来。
  - 主事件流和次要事件流可以用同一用例的不同场景来代表。
  - 在不同的场景中，可以附加不同的交互图或状态图以精确描述对应的事件流。
  - 场景和用例之间的关系用泛化关系描绘，泛化关系的“基类”一端是用例，“导出类”一端是场景

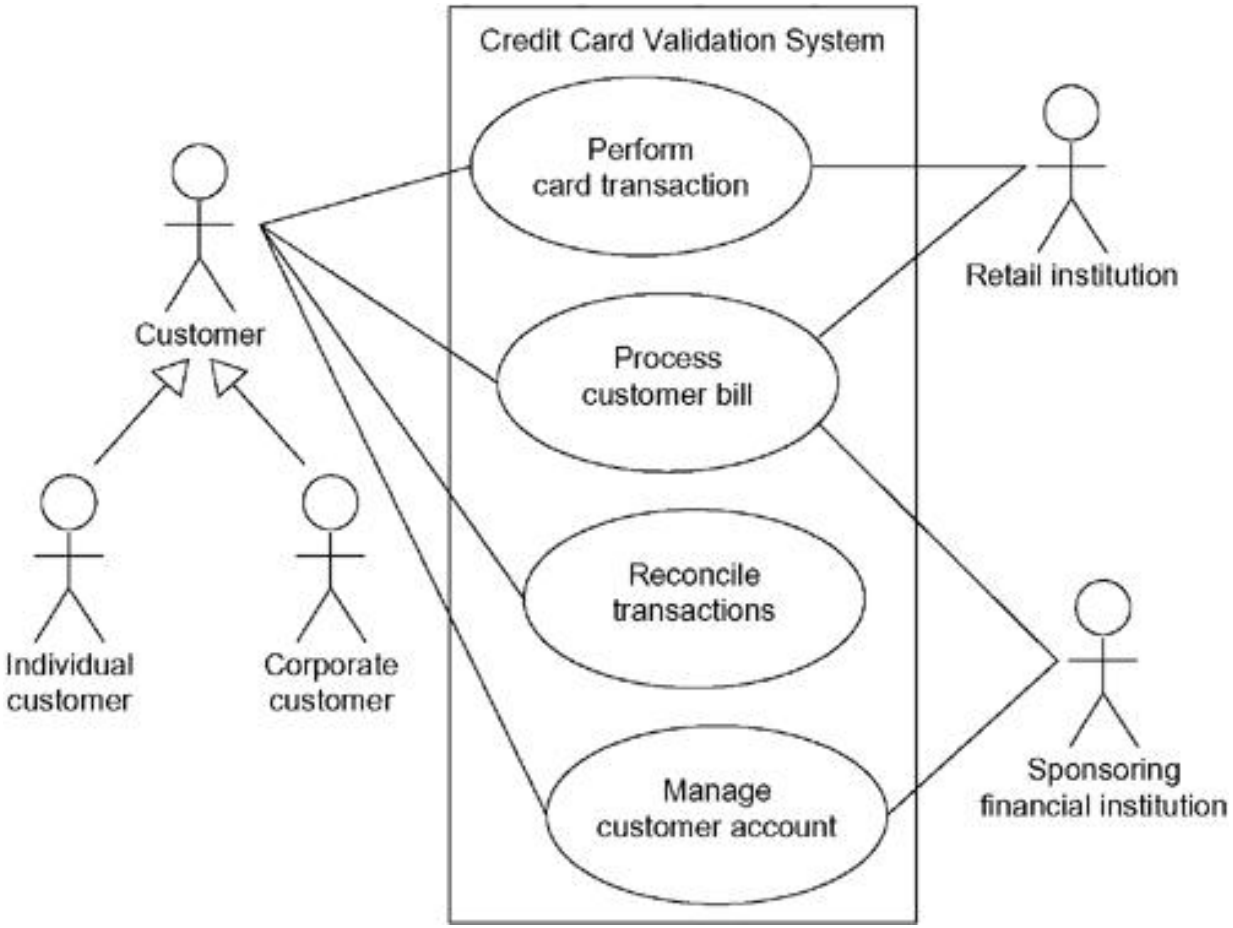
8 普通建模技术

- 对系统的语境建模
- 对系统的需求建模
- 提示与技巧

8.1 对系统的语境建模

- 对系统的语境进行建模，包括围绕整个系统画一条线，并声明有哪些作用者位于系统之外并与系统进行交互。在这里，用例图说明了参与者以及他们所扮演的角色的含义
  - 需要从系统中得到帮助以完成其任务的组
  - 执行系统的功能时所必须的组
  - 与外部硬件或其他软件系统进行交互的组
  - 已经为了管理和维护而执行某些辅助功能的组

8.1 对系统的语境建模（cont’）



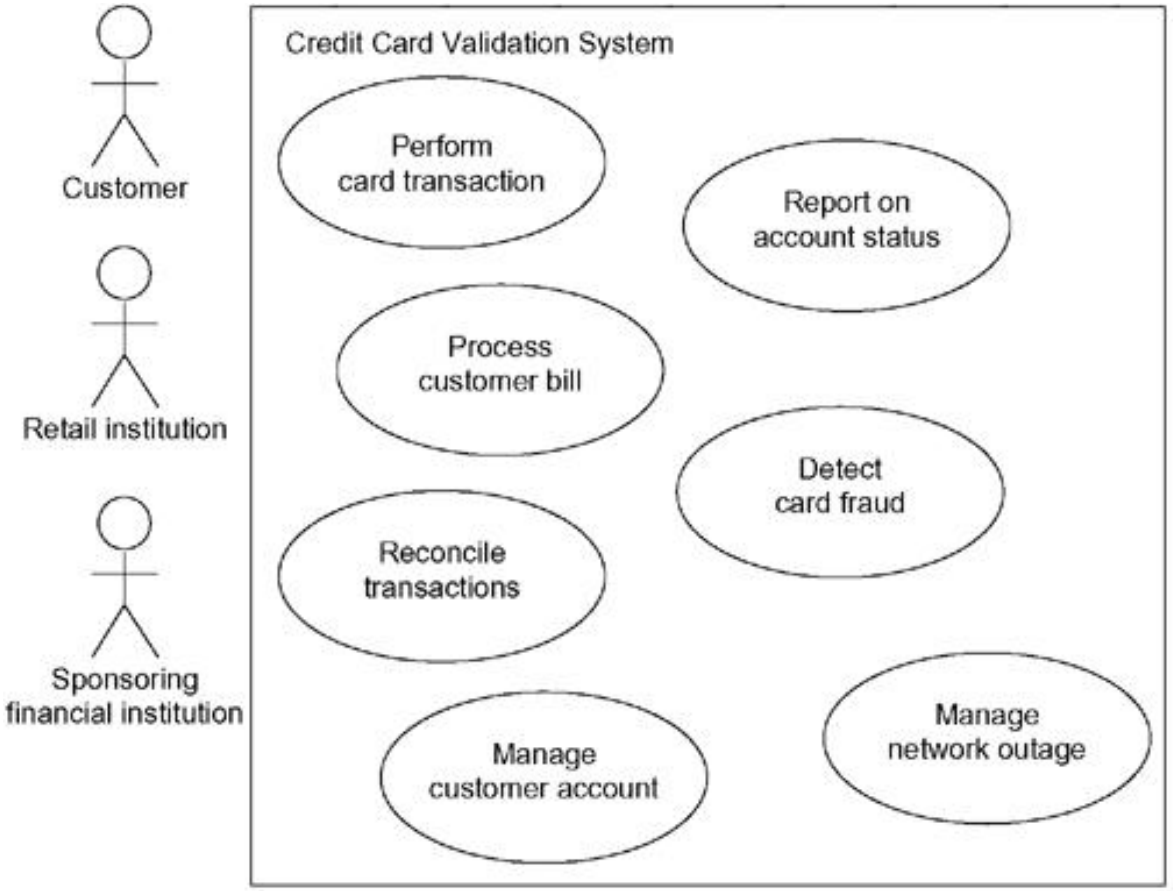
8.2 对系统的需求建模

- 对系统的需求进行建模，包括说明这个系统应该做什么（从系统外部的一个视点出发），而不考虑系统应该怎么做。在这里，用例图说明了系统想要的行为。通过这种方式，用例图使我们能把整个系统看作一个黑盒子；你可以观察到系统外部有什么，系统怎样与那些外部事物相互作用。

8.2 对系统的需求建模（cont’）

- 需要遵循如下策略
  - 通过识别系统周围的参与者来建立系统的语境
  - 对于每个参与者，考虑它期望的行为或需要系统提供的行为
  - 把这些公共的行为命名用例
  - 分解公共行为，放入到新的用例中以供其他的用例使用；分解异常行为，放入新的用例中以延伸较为主要的控制流
  - 在用例图中对这些用例、参与者已经他们的关系进行建模
  - 用陈述非功能需求的注解修饰这些用例；可能还要把其中的一些附加到整个系统

8.2 对系统的需求建模（cont’）



8.3 提示与技巧

- 一个结构良好的用例图，应满足如下的要求：
  - 关注于与一个系统的静态用例视图的一个方面的通信
  - 只包含那些对于理解这方面必不可少的用例和作用者
  - 提供与它的抽象层次相一致的详细表示；只能加入那些对于理解问题必不可少的修饰
  - 不应该过分简化和抽象信息，以至于读者误解重要的语义

8.3 提示与技巧（cont’）

- 当绘制一张用例图时，要遵循如下策略：
  - 给出一个表达其目的的名称
  - 摆放元素时，尽量减少线的交叉
  - 从空间上组织元素，使得在语义上接近的行为和角色在物理位置上也接近
  - 使用注解和颜色作为可视化提示，以突出图的重要特征
  - 尝试不显示太多的关系种类。

8.3 提示与技巧（cont’）

- Fowler的工程意见
  - 用例表示系统外观。据此，不要指望在用例与系统内部的各个类之间有任何联系
  - 用例越多，似乎用例图的价值就越小
  - 对用例，要全神贯注的是正文而不是图
  - 用案的一大危险是，人们要它做的太复杂
  - 坚决建议略去extend关系，只使用简单的include关系

9 参考读物

- 《Writing Effective Use Cases》 Cockburn
- http://usecases.org
- 相关的需求分析系统书籍和文章