



第二部分：如何运用UML建模

第七章 行为模型

提纲

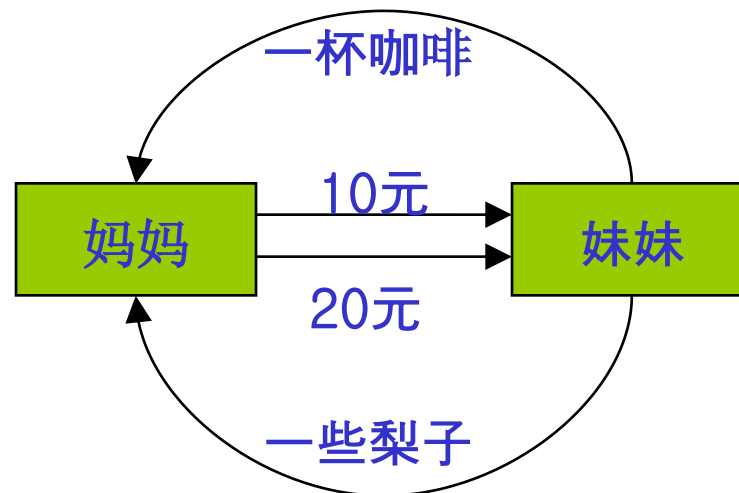
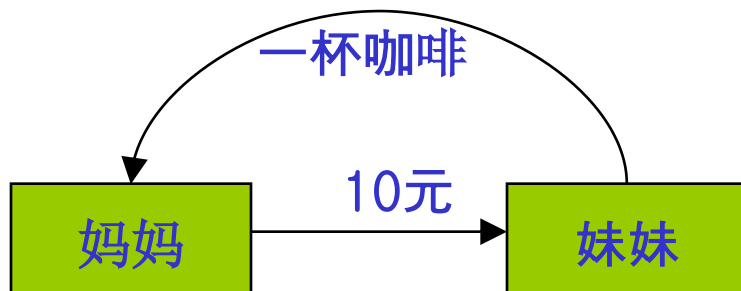


- 概述
- 交互图
- 状态图
- 活动图
- 小结

- 行为模型
 - 对系统的动态行为进行可视化、详述、构造和文档化
 - 说明在静态视图规定的事物结构下的动态行为
 - 包含
 - 交互图
 - 顺序图(时序图)
 - 通信图(协作图)
 - 活动图
 - 状态图

- 日常生活中的消息交互

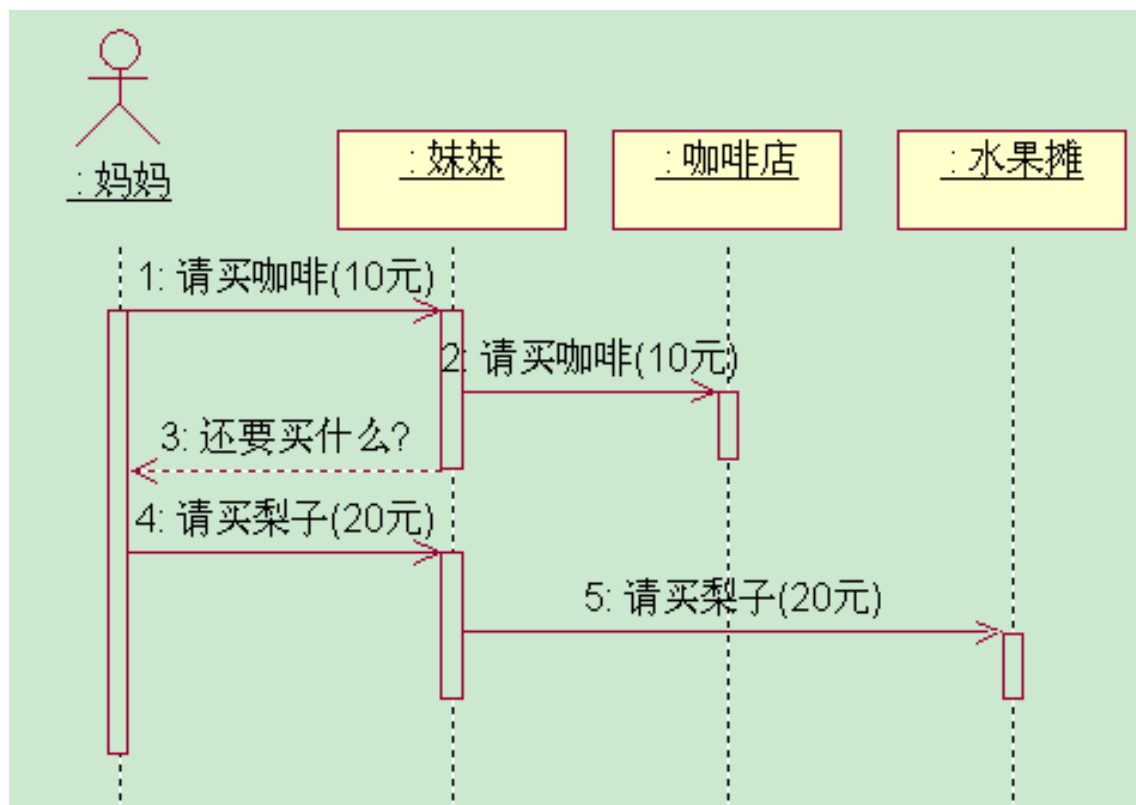
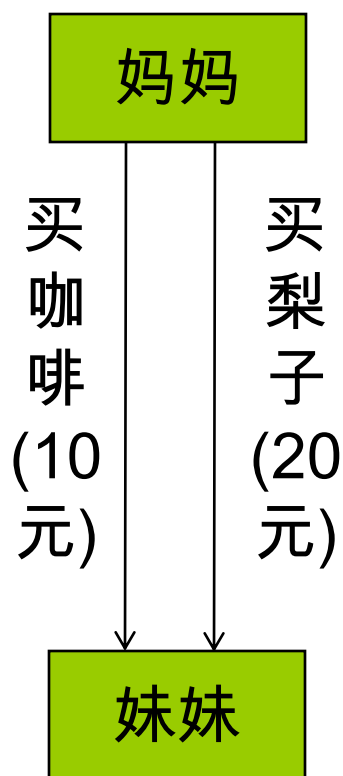
- 妈妈给妹妹10元钱，让她买杯咖啡



- 妈妈另外给妹妹20元，请妹妹买一些梨子

交互图

- 用UML的消息传递表示



- 交互

- 对象通过协作共同解决一个问题

- 每个对象负责自身的状态和行为
 - 没有一个对象能靠自身执行所有职责

- 交互

- 对协作的动态方面建模
 - 对象间的交互通过消息的传递进行
 - 定义：由一组对象为达到某一目的而交换的一组消息构成的一种行为

- 交互
 - 对象和角色
 - 命名方式：具名、匿名
 - 具体的事物
 - 原型化的事物 ←—— 角色(role)

uml :
course

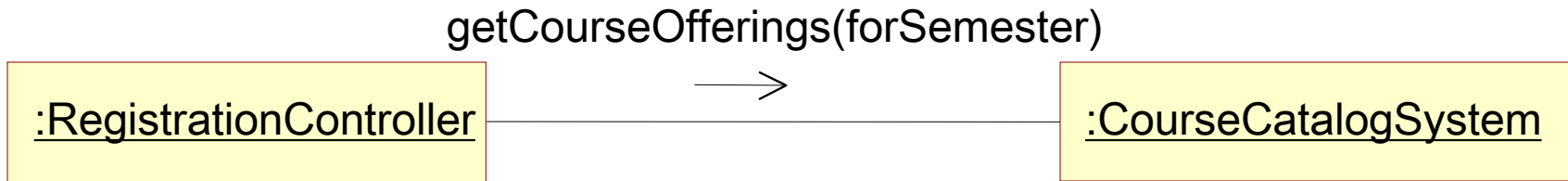
: course

UML

- 交互

- 消息： 传送信息的对象之间进行的通信的规约

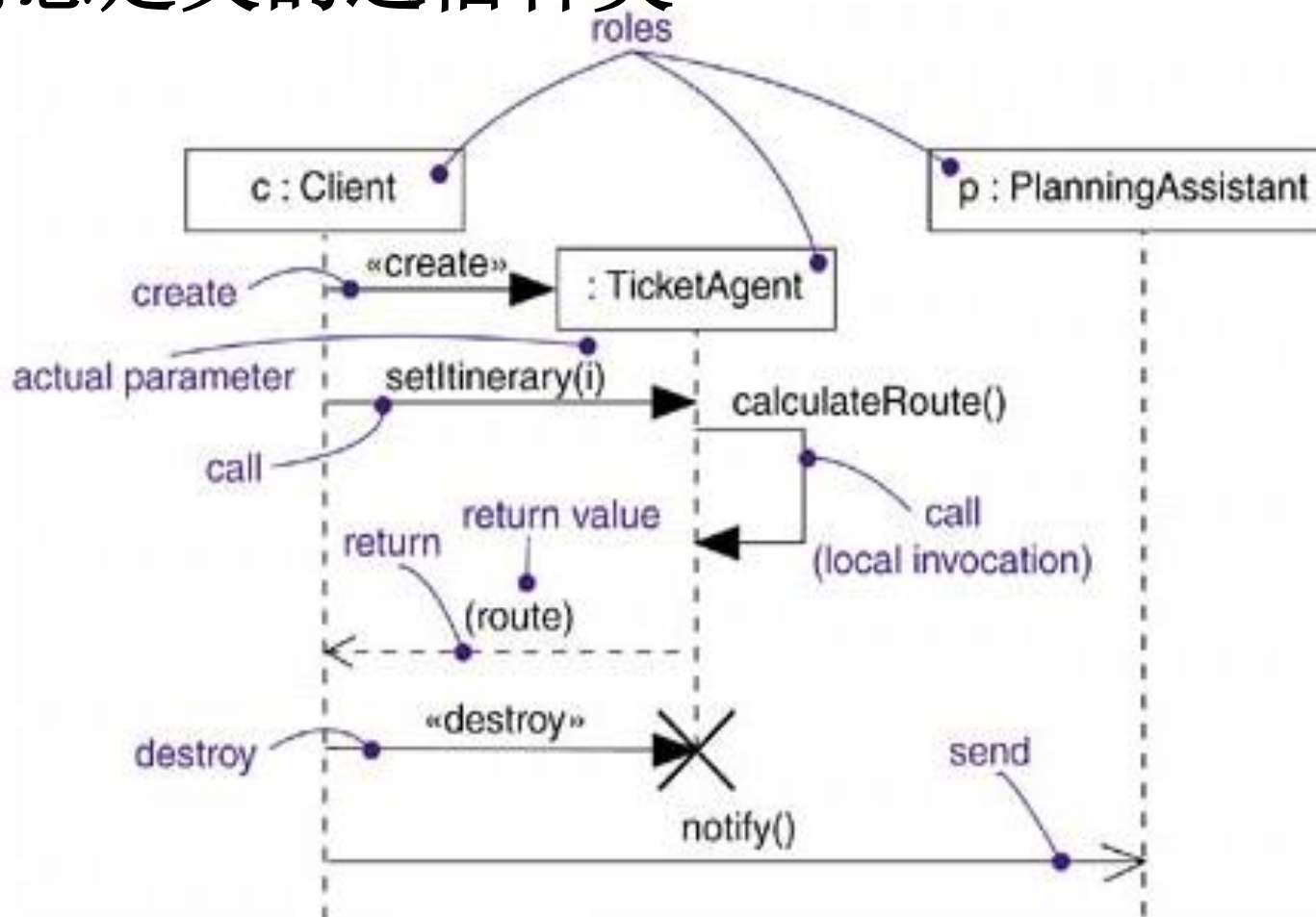
- 通常描述了一个对象如何请求另一个对象执行操作
 - 对消息的接收产生一个动作，引发目标对象的状态改变



- 消息定义的通信种类
 - 调用 (call)
 - 调用某个对象的一个操作
 - 本地调用：给自己发送消息
 - 返回 (return)：给调用者返回一个值
 - 发送 (send)：向对象发送一个信号
 - 创建 (create)：创建一个对象
 - 撤销 (destroy)：撤销一个对象

交互图

• 消息定义的通信种类



- 交互图

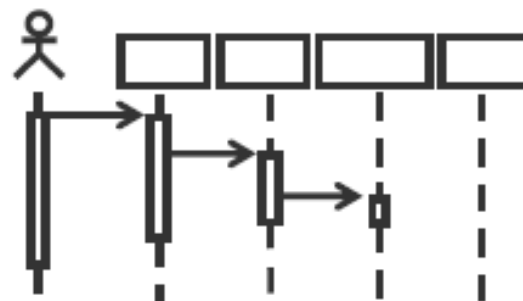
- 描述对象间的动态合作关系，以及合作过程中的行为次序
- 包含
 - 一组对象：每个对象扮演某个特定角色
 - 一组消息：每个消息代表对象间的通信活动
- 两种图：分别从不同侧面描述交互

交互图

- 交互图

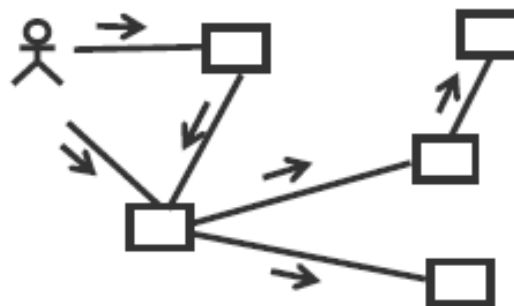
- 两种形式

- 强调消息的时间顺序



Sequence Diagrams

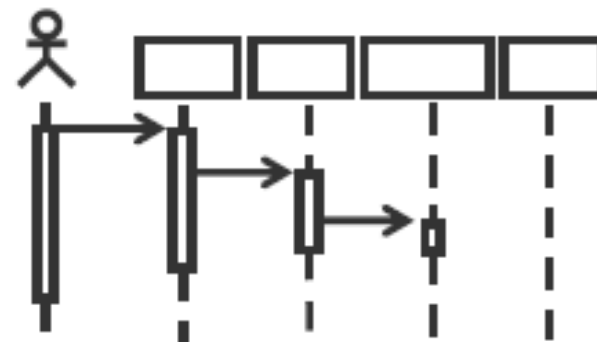
- 强调发送和接收消息的对象的结构组织



Communication Diagrams

顺序图

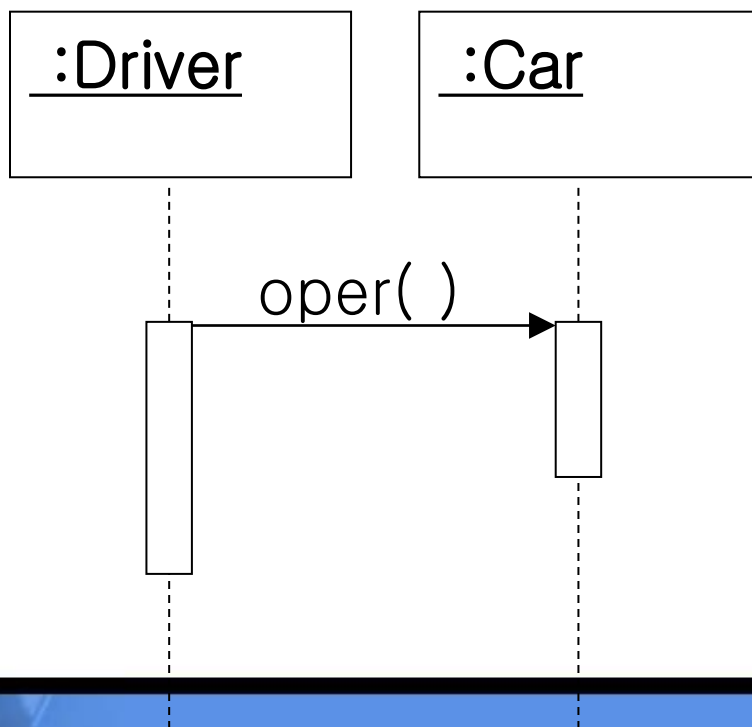
- 顺序图
 - 强调消息的时间顺序
 - 一张表
 - X轴：对象和角色
 - Y轴：按时间排序的消息
 - 特征
 - 对象生命线
 - 控制焦点



Sequence Diagrams

顺序图

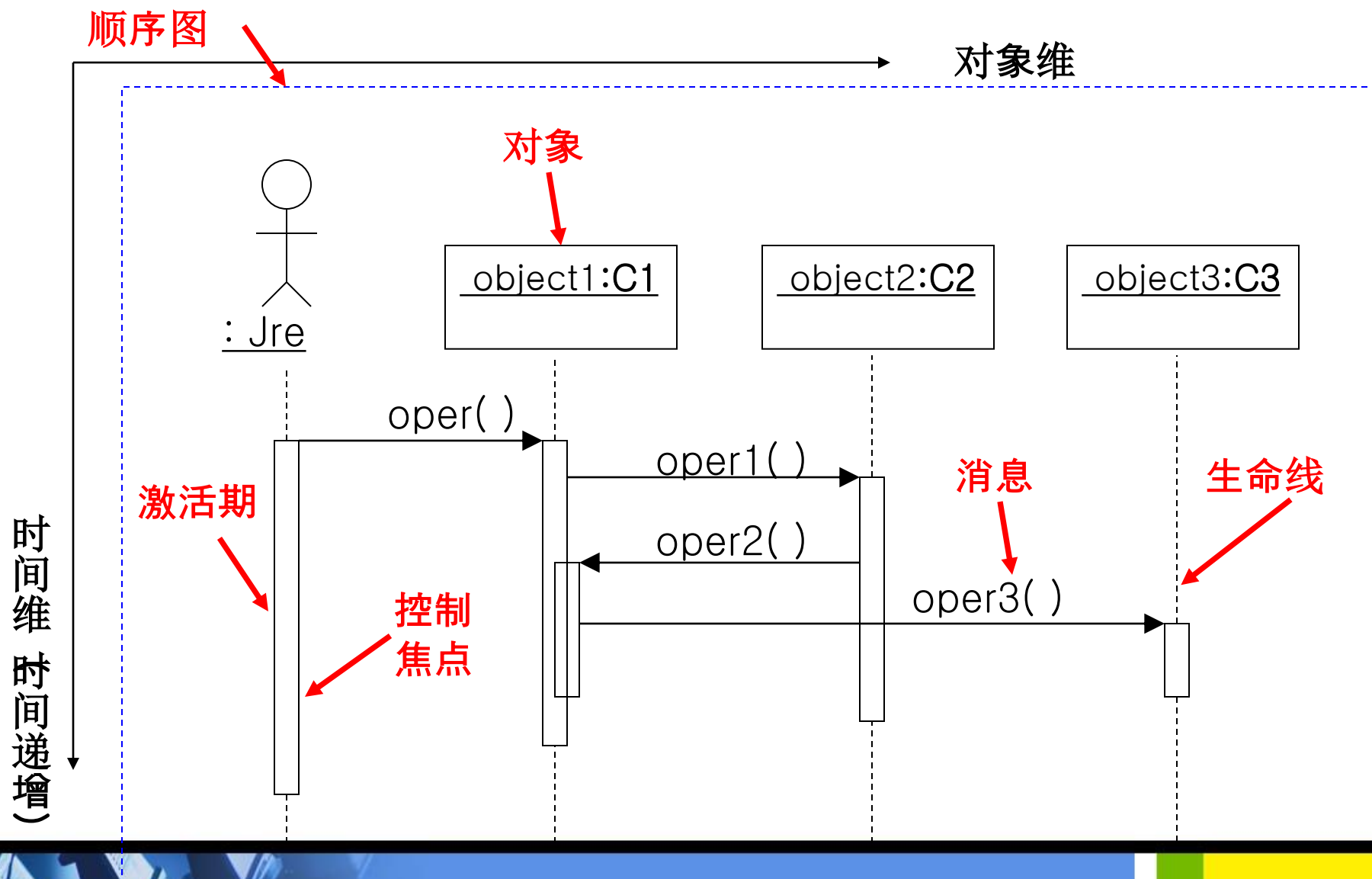
- 顺序图，也称时序图。
- Rumbaugh给出的定义：
- 顺序图是显示对象之间交互的图，这些对象是按时间顺序排列的。



顺序图

- 顺序图描述了对象之间传递消息的时间顺序。
- 它用来表示用例中的行为顺序，强调消息时间顺序的交互图。
- 当执行一个用例行为时，顺序图中的每一条消息对应了一个类操作，或状态机中引起转换的触发事件。

顺序图

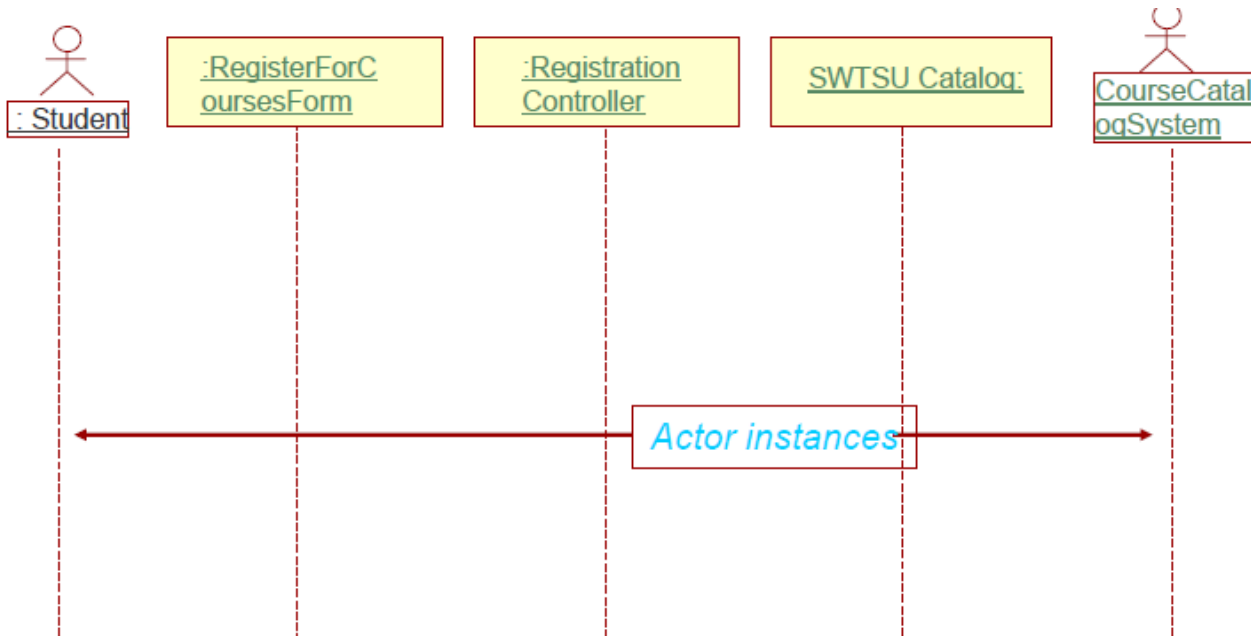


顺序图

- 顺序图

- 对象的排列原则

- 发起交互的对象或角色在左边
 - 较下级的对象依次放在右边



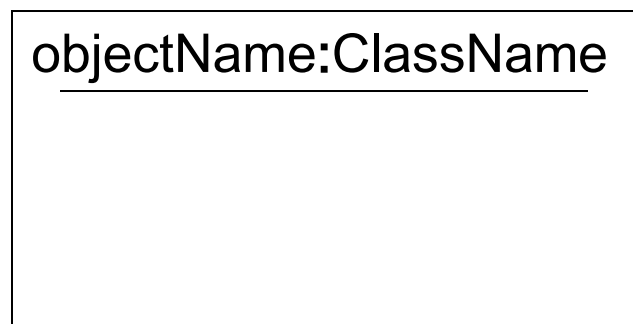
顺序图

- 顺序图中的**建模元素**:
 - 对象（参与者实例也是对象）
 - 生命线
 - 控制焦点
 - 消息

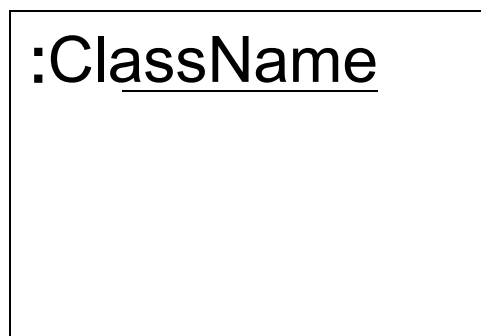
顺序图

- 对象：
 - 3种命名方式
 - 显示对象名和类名
 - 只显示类名，不显示对象名（匿名对象）。此时，用类角色代表实际的对象。
 - 只显示对象名，不显示类名（不关心对象所属的类名）

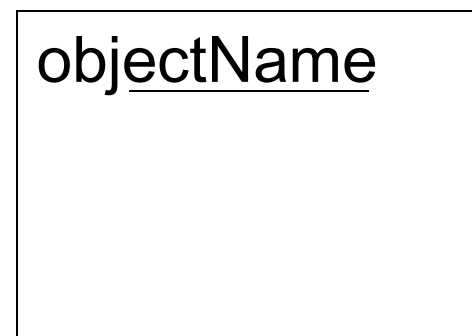
顺序图



显示对象名和类名



只显示类名

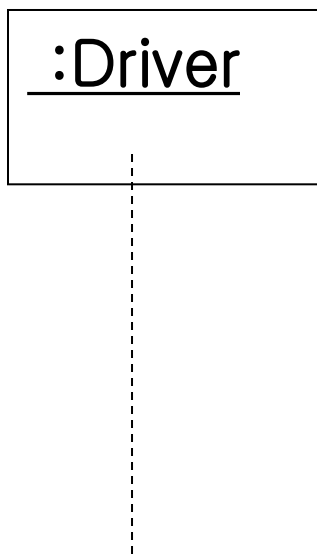


只显示对象名

顺序图

- **生命线:**

- 从对象图标向下延伸的一条虚线
- 表示对象存在的时间



顺序图

- **生命线:**

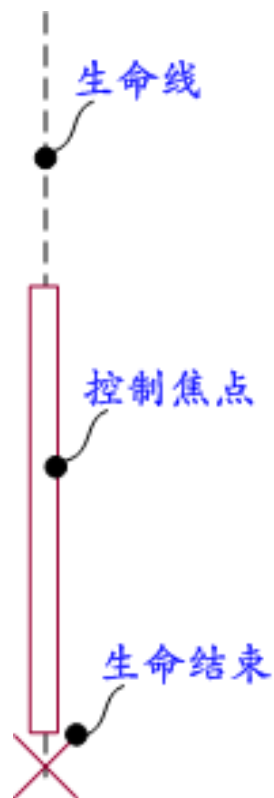
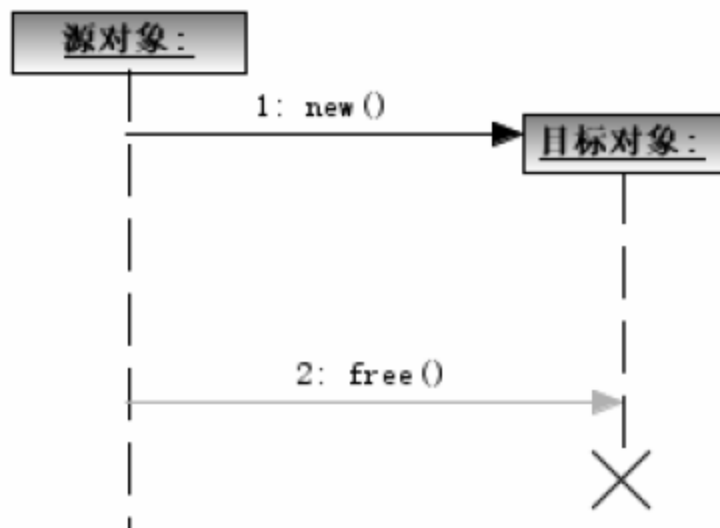
- **生命线**表示一个对象在一段时期内的存在
- 正是因为这个特性，使**顺序图**适合对象之间消息的时间顺序
- 一般情况下，对象的**生命线**从图的顶部画到底部，表示对象存在于交互的整个过程

顺序图

- 顺序图

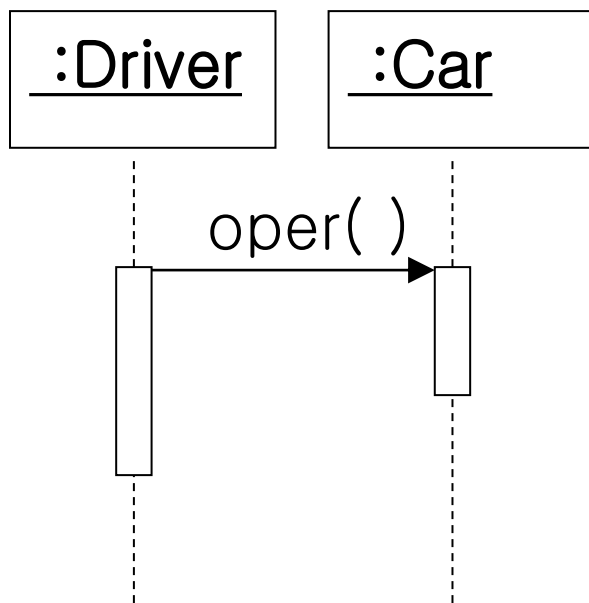
 - 对象生命线

 - 表示一个对象在一段时间内存在
 - 大多数对象存在于整个交互中
 - 交互中创建/删除的对象



顺序图

- **控制焦点**（又称**激活期**）：
 - 在生命线上的小矩形
 - 表示时间段的符号
 - 在这个时间段内，对象将执行相应的操作

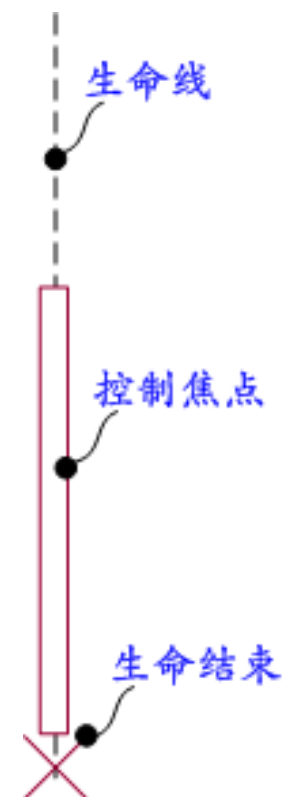
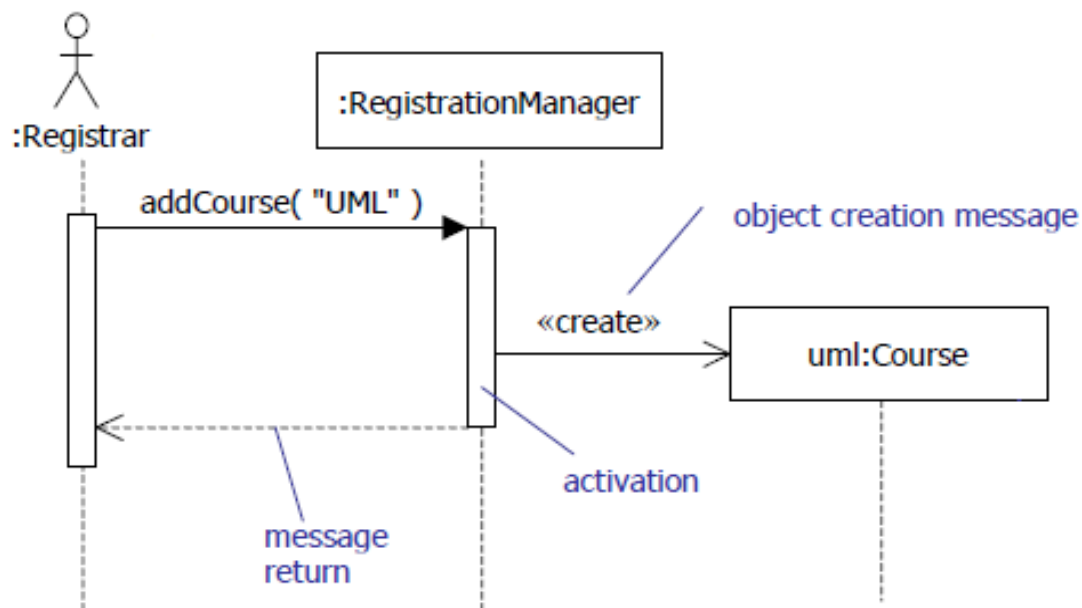


顺序图

- 顺序图

- 控制焦点（激活期）

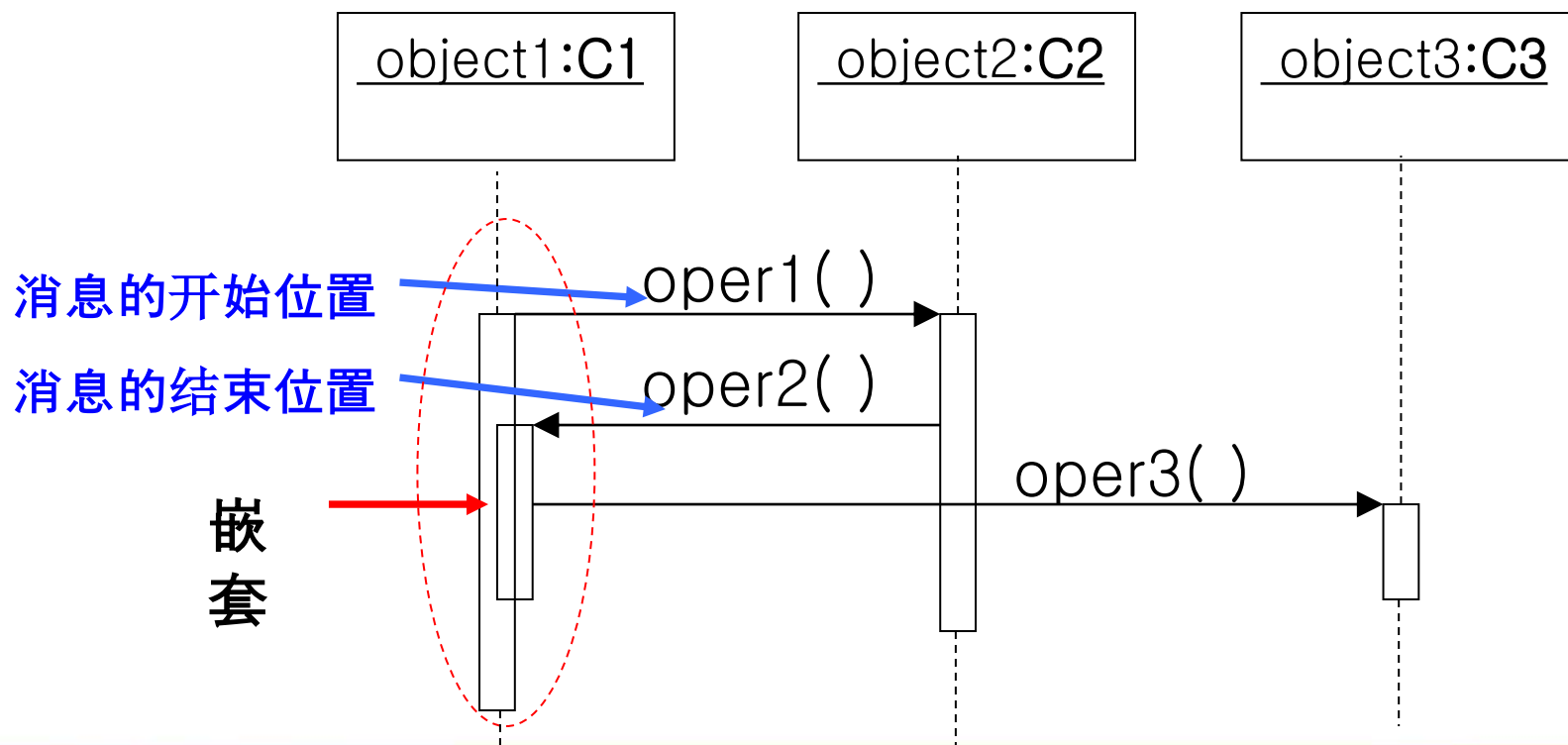
- 对象执行一个动作所经历的时间段



顺序图

- 控制焦点:

- 控制焦点可以嵌套, 更精确地说明消息的开始和结束位置



顺序图

- **控制焦点：**

- 激活期表示一个对象执行一个动作的期间，即对象激活的时间段
- 激活矩形的高度代表激活持续时间
- 这个特性可视化地描述了对象执行一项操作的时间

顺序图中的消息——交互图

- **消息：**

- 从一个对象的生命线到另一个对象的生命线的箭头
- 定义交互和协作中交换的信息
- 一个对象或类可以通过**消息**请求另一个对象或类来完成特定功能

顺序图中的消息——交互图

- 消息种类:
 - 调用消息
 - 异步消息
 - 返回消息
 - 反身消息
 - 阻止消息
 - 超时消息

顺序图中的消息——交互图

- **调用消息:**

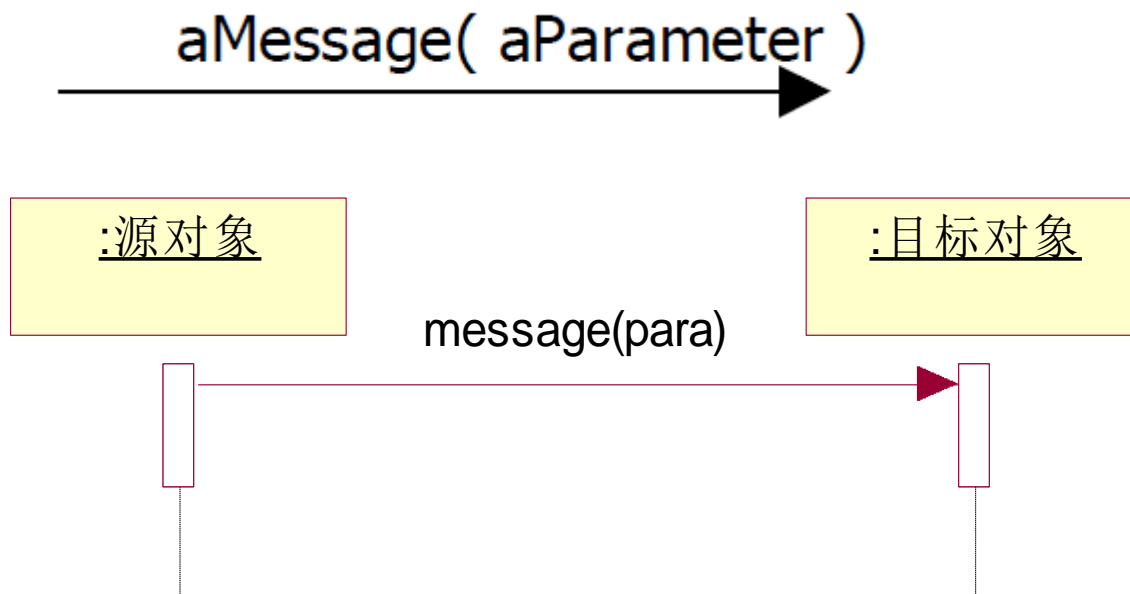
- 调用消息的发送者把控制传递给消息的接收者，然后停止活动，等待消息接收者放弃或返回控制
- 早期版本中，也称为(**同步消息**)
- 为了图的简洁，与调用消息配对的返回消息可以不用画出来
- 接收者是一个需要通过消息驱动才能执行动作的对象 (**被动对象**)

顺序图中的消息——交互图

- 消息的类型

- 调用消息 (同步消息) (Procedure call)

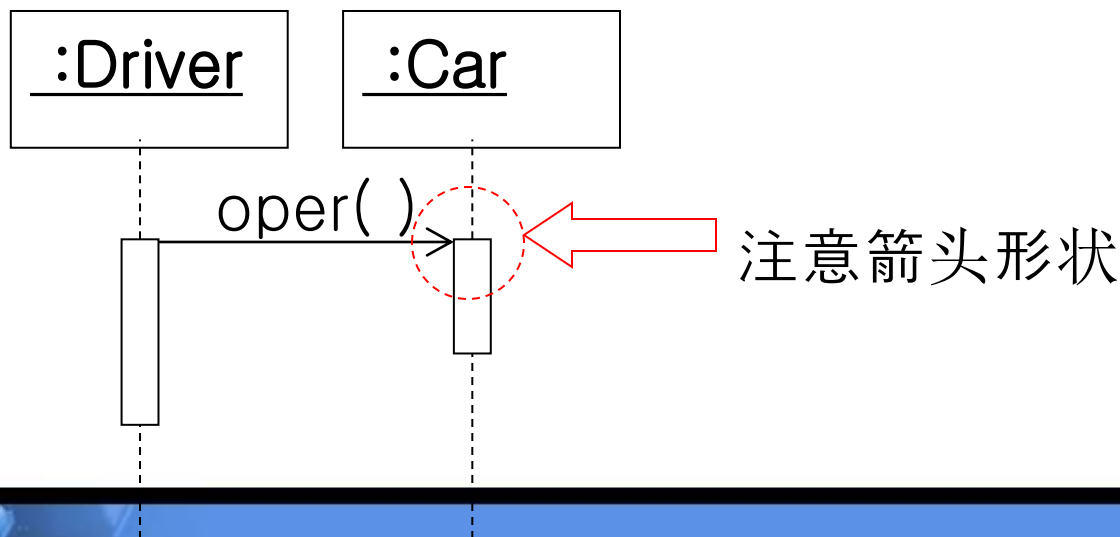
- 发送者发送消息后，等待直到接收者返回控制



顺序图中的消息——交互图

- 异步消息:

- 异步消息的发送者通过消息把信号传递给消息的接收者，然后继续自己的活动，不等待消息接收者返回消息或控制
- 接收者和发送者是并发工作的

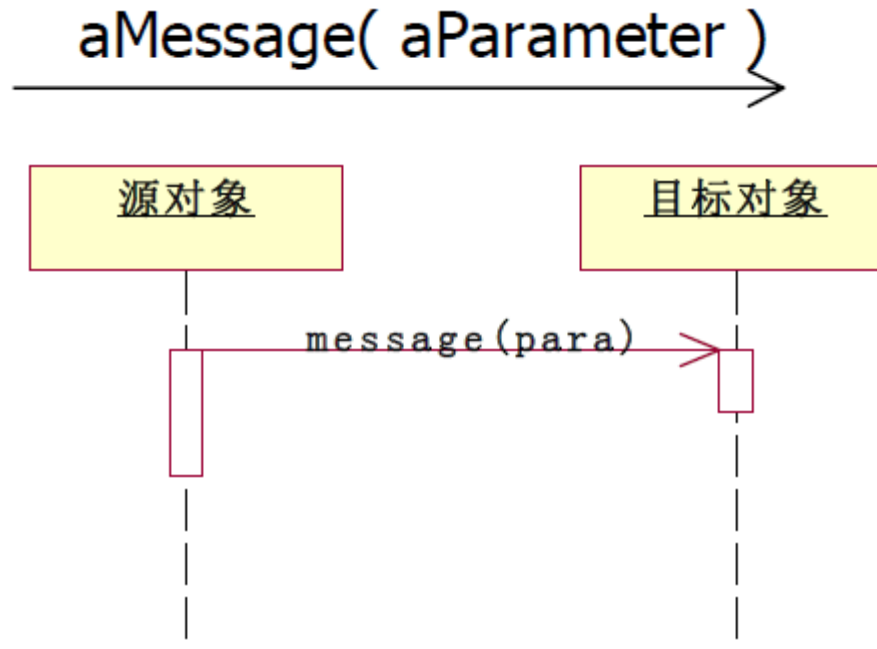


顺序图中的消息——交互图

- 消息的类型

- 异步消息 (Asynchronous)

- 发送者发送消息后，继续操作不等待



顺序图中的消息——交互图

- 返回消息:

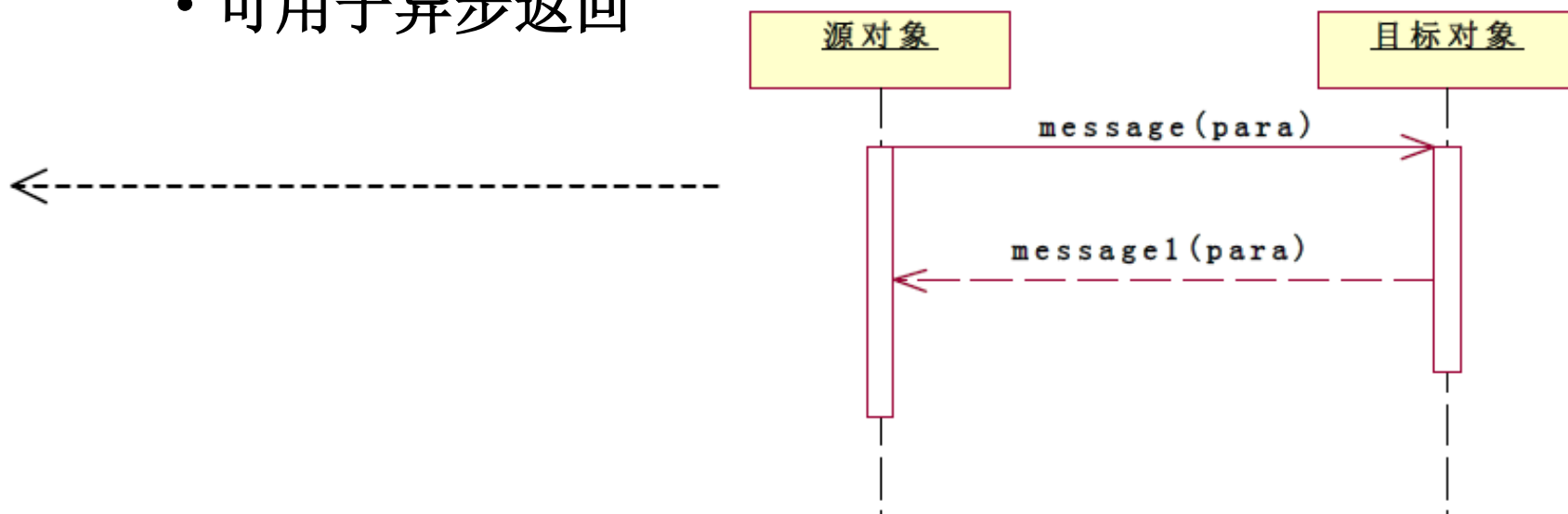
- 返回消息表示从过程调用返回
- 如果是从过程调用返回，则返回消息是隐含的，所以返回消息可以不用画出来
- 对于非过程调用，如果有返回消息，必须明确表示出来
- 返回消息用虚箭头表示

顺序图中的消息——交互图

- 消息的类型

- 返回消息 (Return)

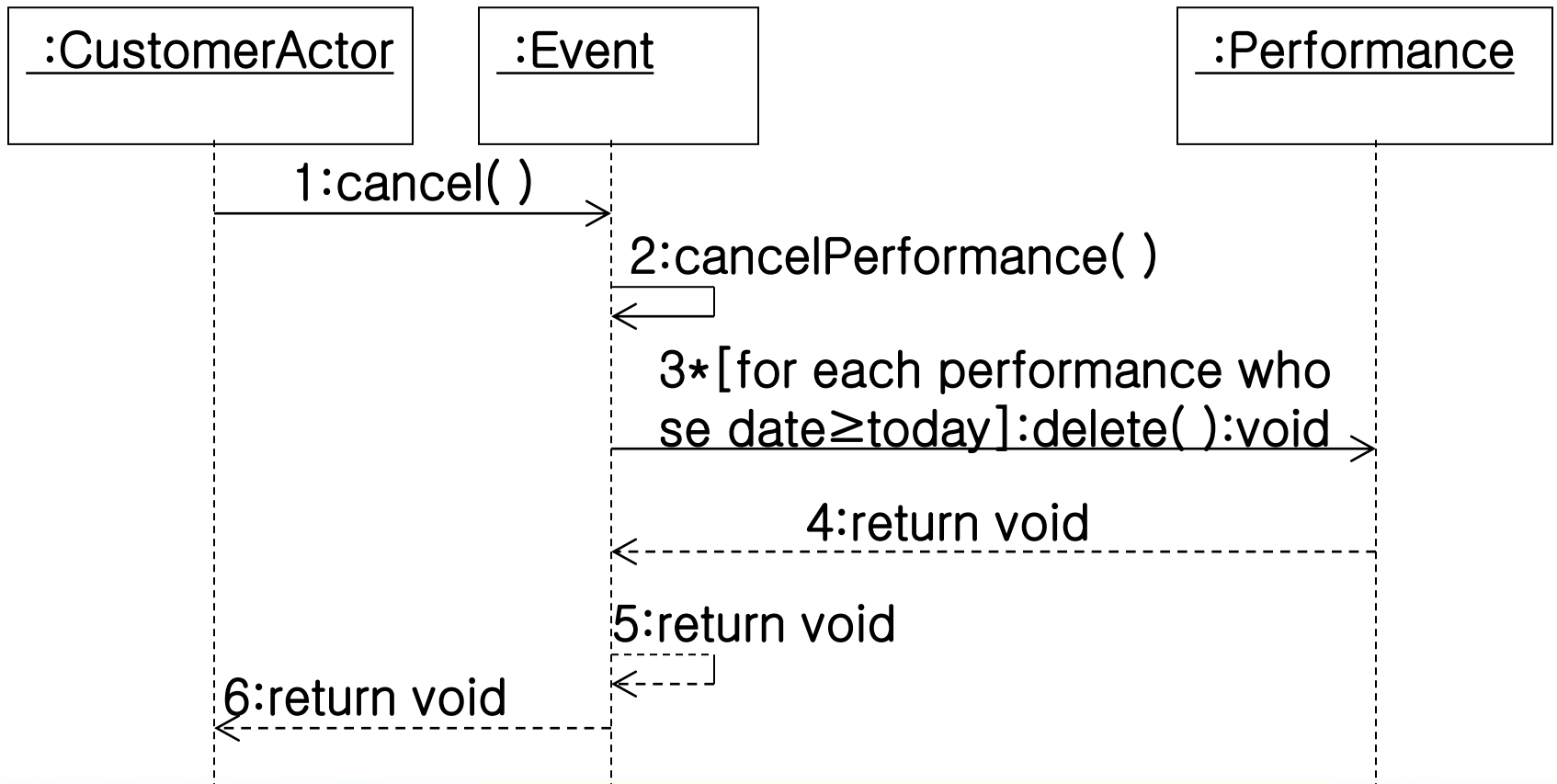
- 更早消息的接收者返回控制焦点给发送者
 - 同步的返回一般无需画出，直接隐含
 - 可用于异步返回



顺序图中的消息——交互图

- **反身消息:**

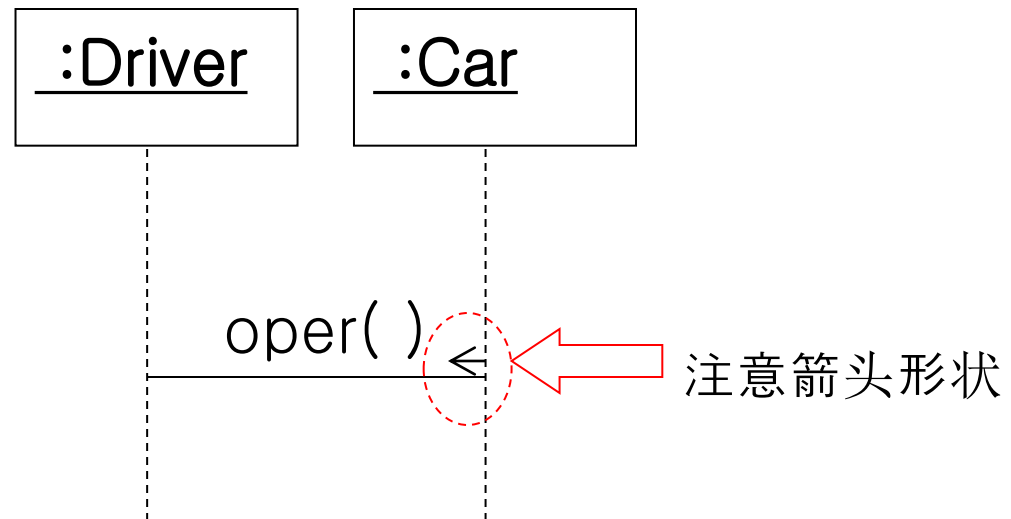
- 在反身消息中，消息的发送者和接收者是同一个对象。



顺序图中的消息——交互图

- **阻止消息:**

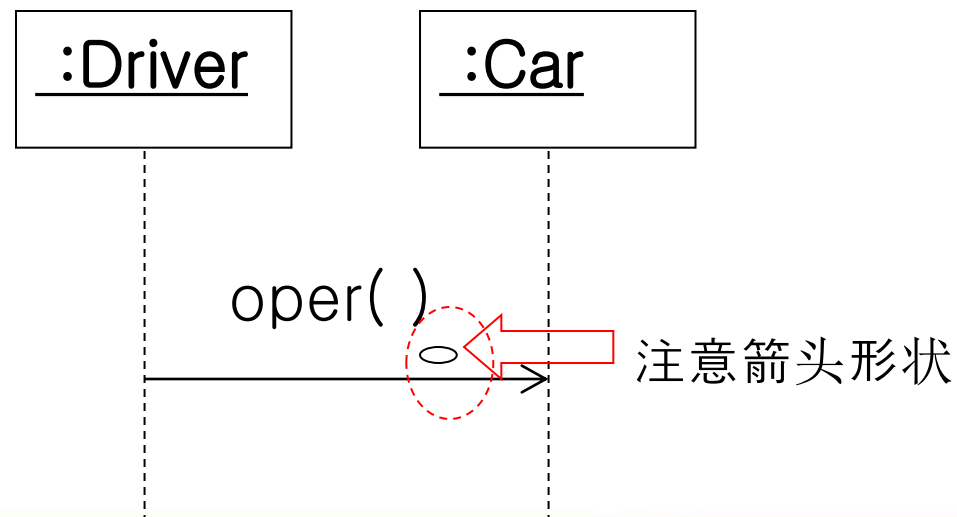
- 消息发送者发出消息给接收者，如果接收者无法立即接收消息，则发送者放弃这个消息
- 用折回的箭头表示



顺序图中的消息——交互图

- 超时消息:

- 消息发送者发出消息给接收者，并按指定时间等待。如果接收者无法在指定时间内接收消息，则发送者放弃这个消息
- 用附带小圆圈表示

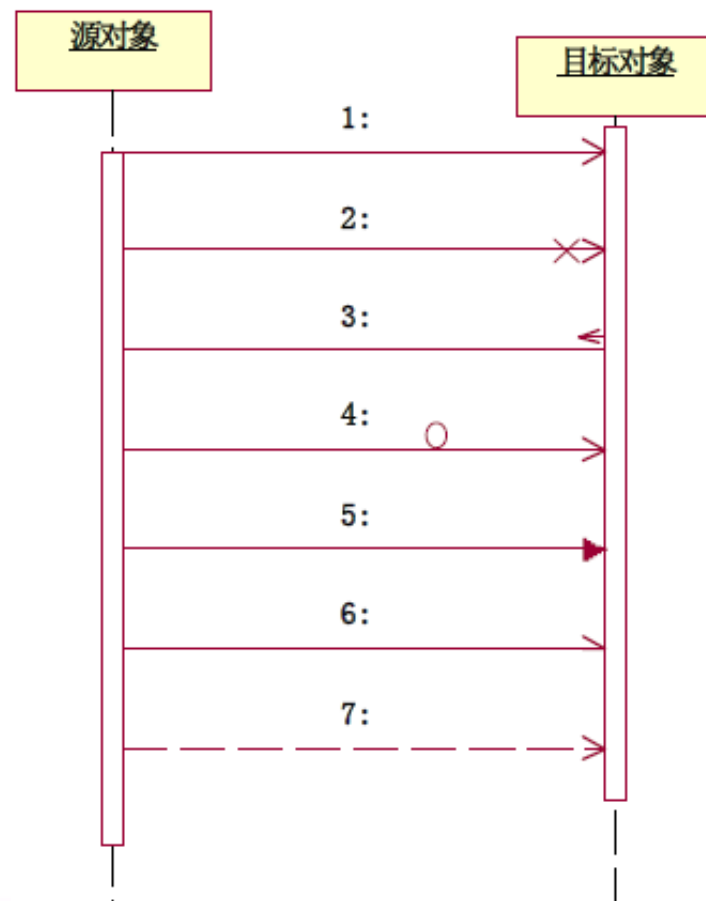
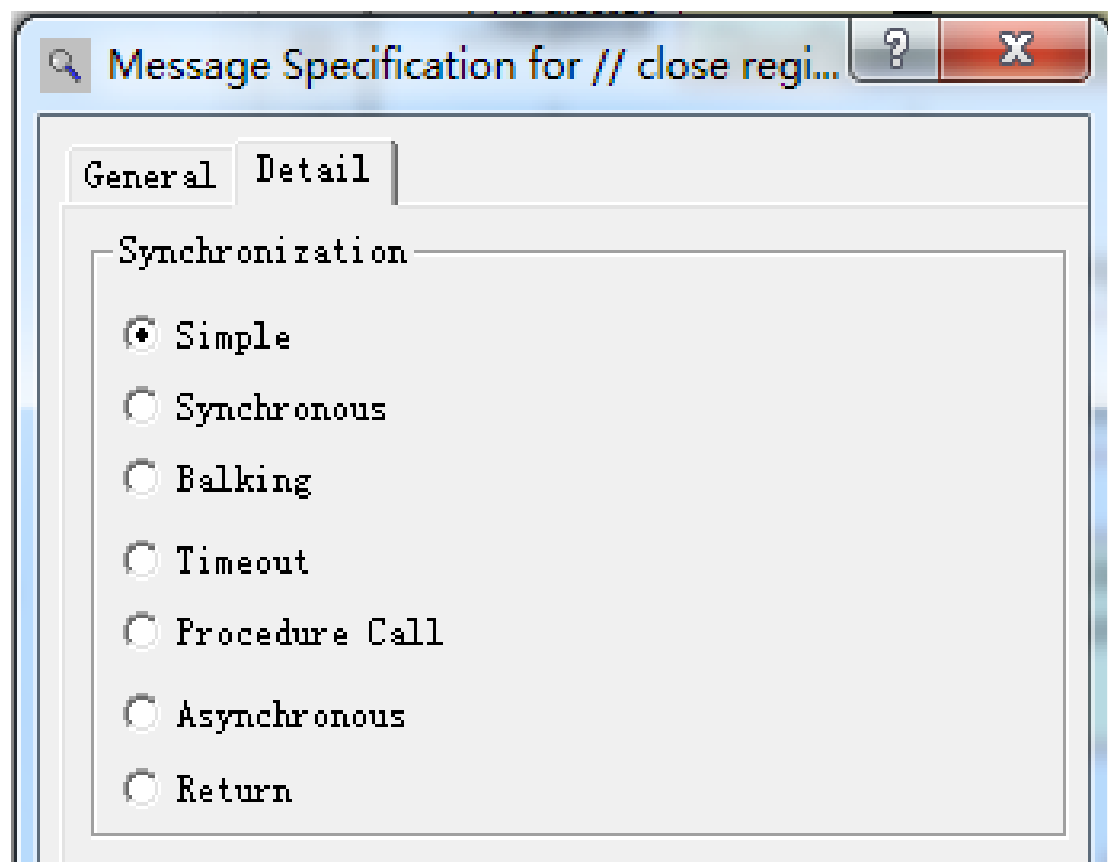


顺序图中的消息——交互图

- **消息的语法格式:**
- **[predecessor] [guard-condition]
[sequence-expression] [return-value :=]
message-name ([argument-list])**
 - **predecessor:** 必须先发生的消息的列表
 - **guard-condition:** 警戒条件
 - **sequence-expression:** 消息顺序表达式
 - **return-value:** 消息的返回值的名字列表
 - **message-name:** 消息名
 - **argument-list:** 消息的参数列表

顺序图中的消息——交互图

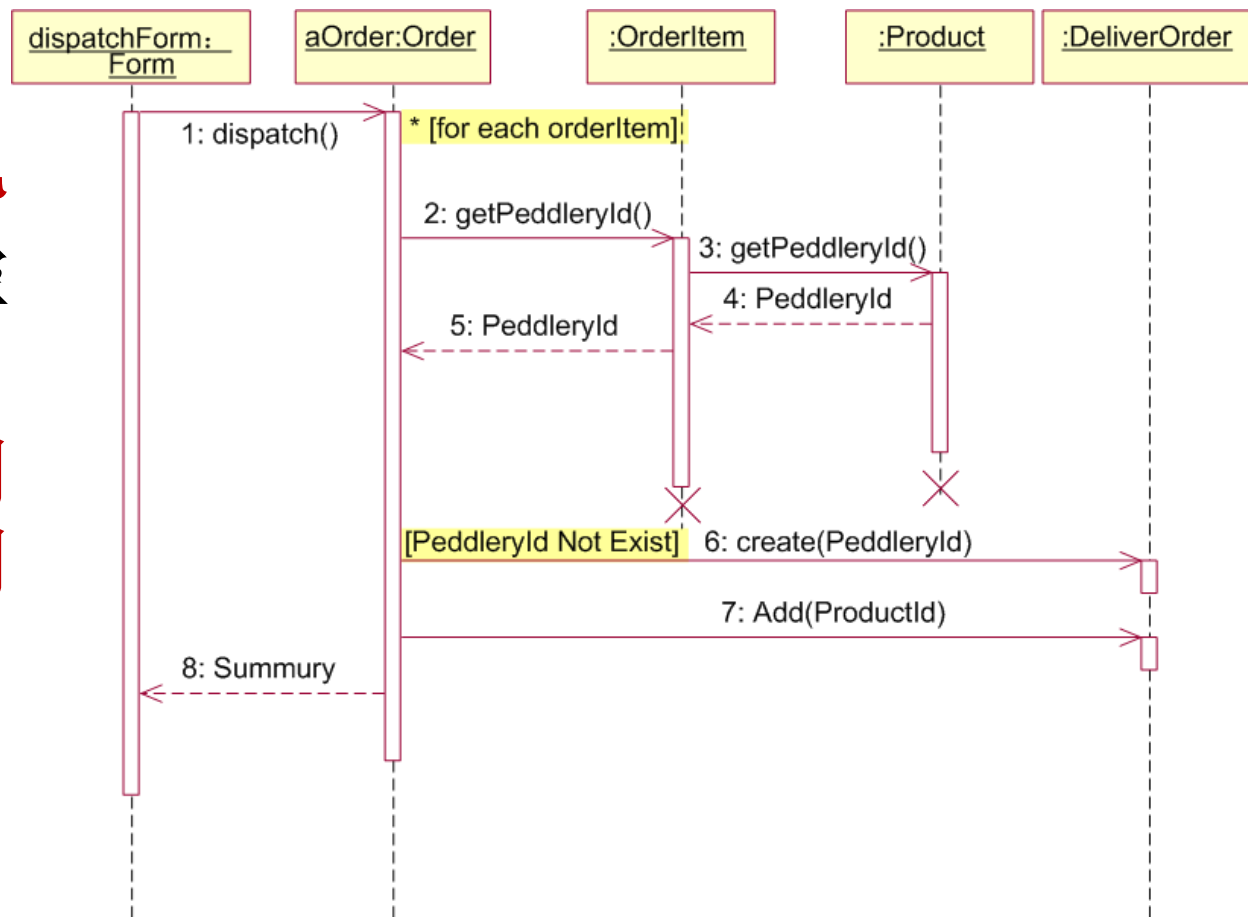
- ROSE中的消息类型



顺序图



- 阅读顺序图
 - 对象与角色
 - 生命线与控制焦点
 - 按时间排列的消息序列
 - 顺序编号（可选）



• 阅读顺序图

- 在dispatchForm（分发窗体）中，对于某个已支付的Order进行分发时，就会调用该订单（一个Order类的实例对象aOrder）的dispatch()方法
- dispatch()方法将逐个调用该Order对应的所有OrderItem对象的getPeddleryId()方法还获取供应商ID（PeddleryId），而OrderItem对象则是通过其所对应的Product对象来的getPeddleryId()方法来获取供应商ID
- 当Order的实例对象aOrder得到返回的PeddleryId后，根据该值判断是否已经有相对应的DeliverOrder对象，如果没有就创建它（调用create(PeddleryId)），然后再将对应的Product添加到这个DeliverOrder对象中。否则就直接添加到相应的DeliverOrder对象中

顺序图

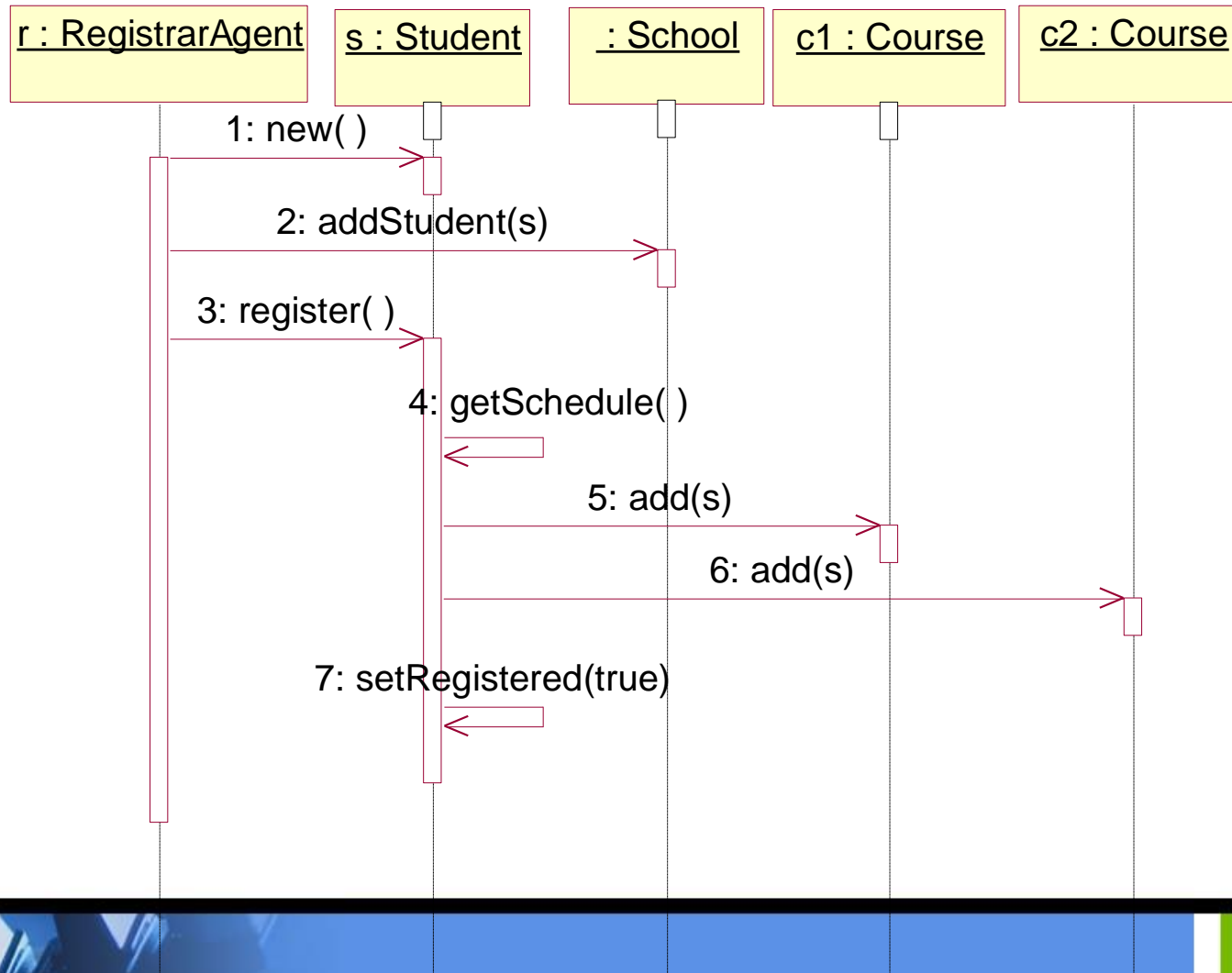


- 绘制顺序图
 - 确定交互的范围
 - 识别参与交互的对象（角色、参与者）
 - 设置对象生命线的开始和结束
 - 设置消息
 - 细化消息
- 不同抽象级别的顺序图
 - 分析级：用于概要描述交互的场景
 - 设计级：涉及实现细节

顺序图



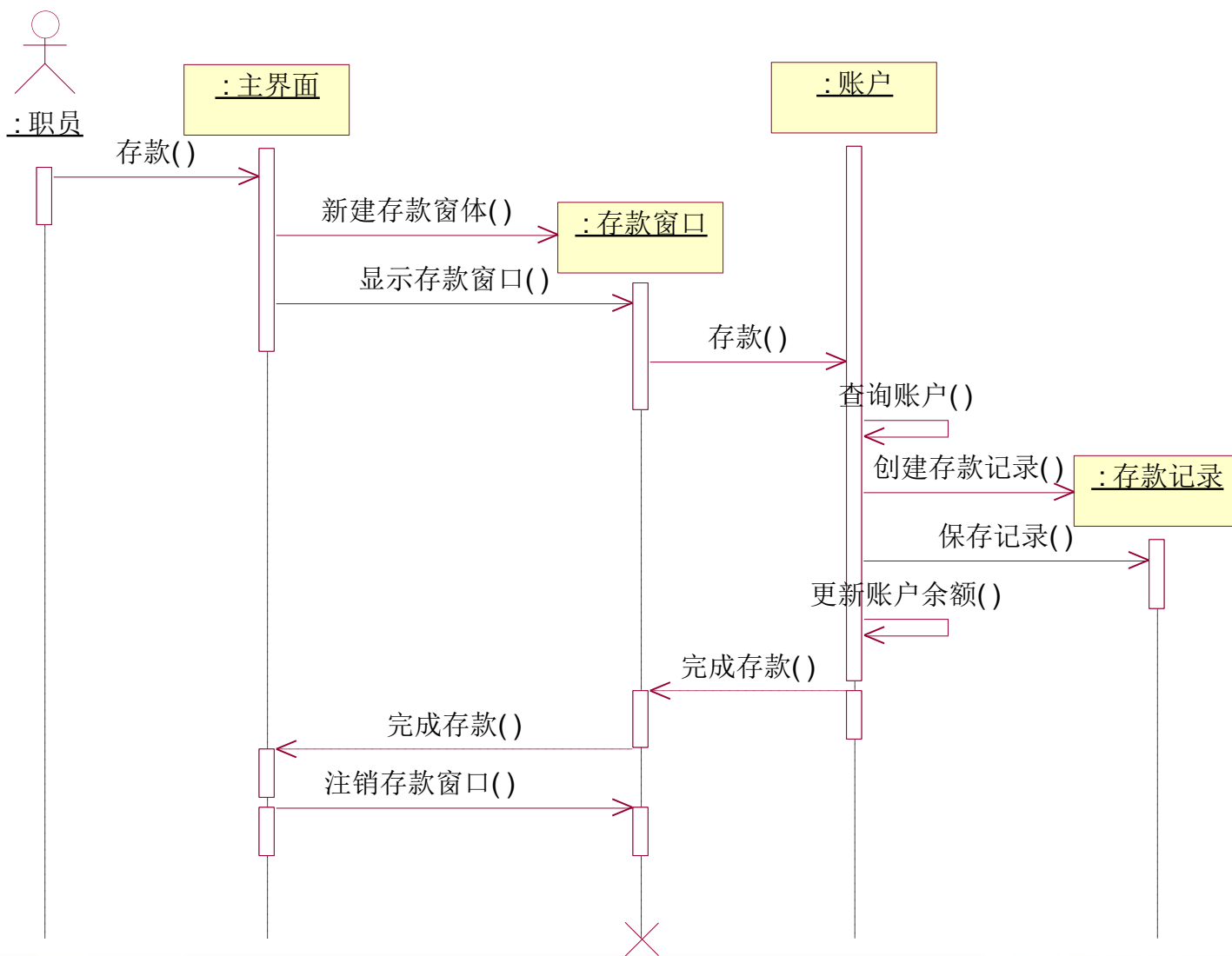
- 学生注册课程的顺序图



顺序图

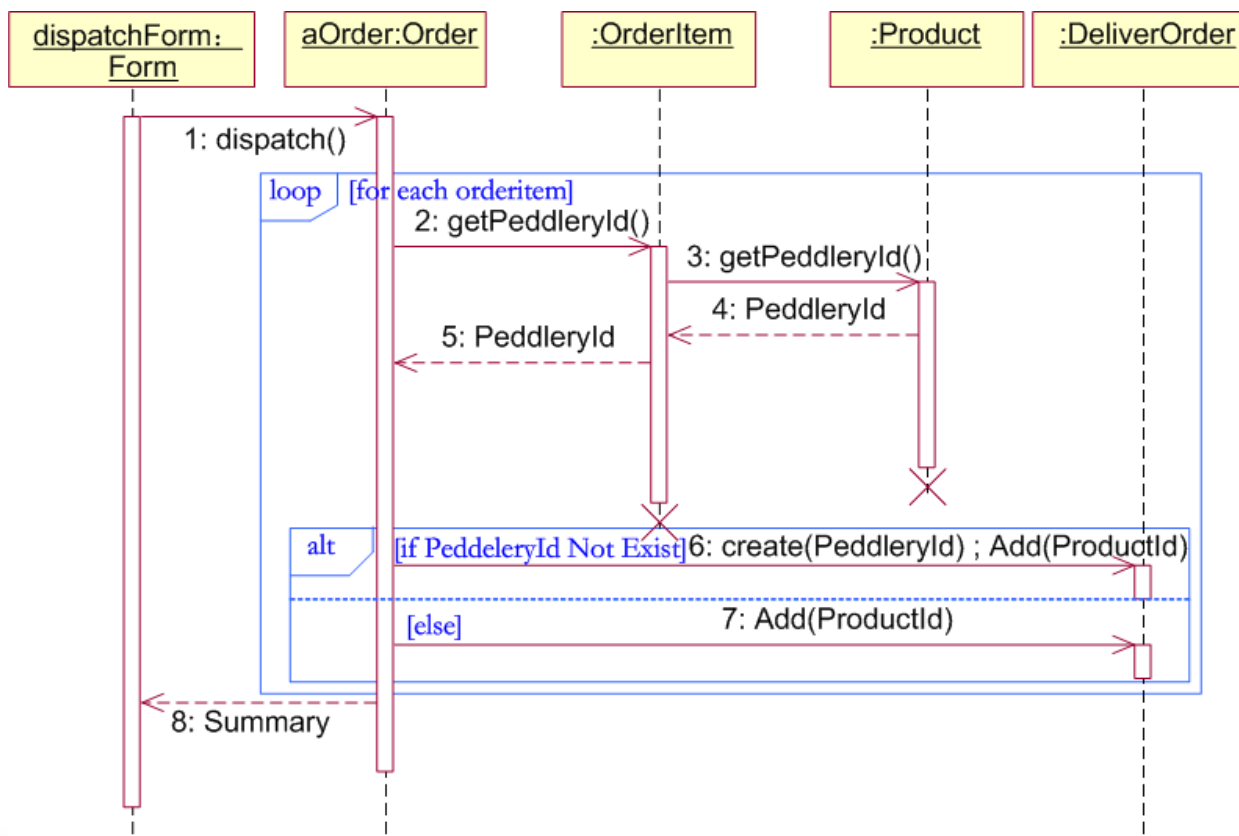


一个分析级的
顺序图：
描述银行职员
存款用例中成功存款
的场景



顺序图

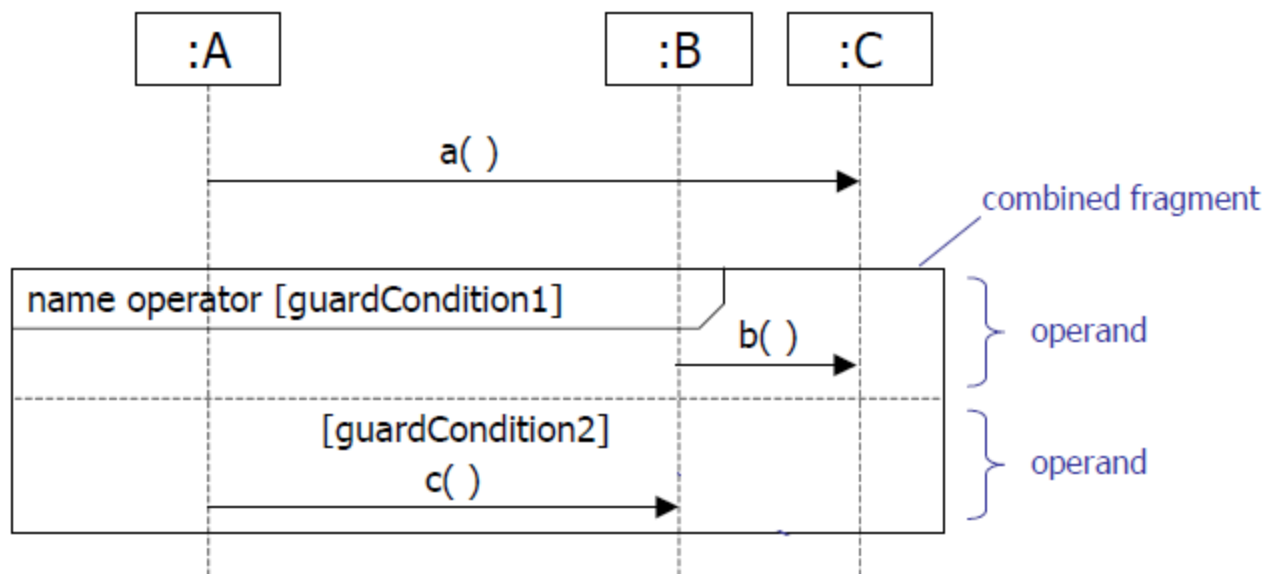
- 顺序图中的结构化控制
 - 展示条件、循环或并发等高层控制



顺序图



- 顺序图中的结构化控制
 - 组合片段：顺序图中的区域
 - 操作符：确定运算单元如何执行
 - 监护条件：确定运算单元是否被执行



- 顺序图中的结构化控制
 - 操作符标签标明不同的控制类型
 - 组合片段常见的控制类型
 - 可选执行 `opt` (option)
 - 条件执行 `alt` (alternatives)
 - 循环执行 `loop`
 - 并行执行 `par` (parallel)

顺序图

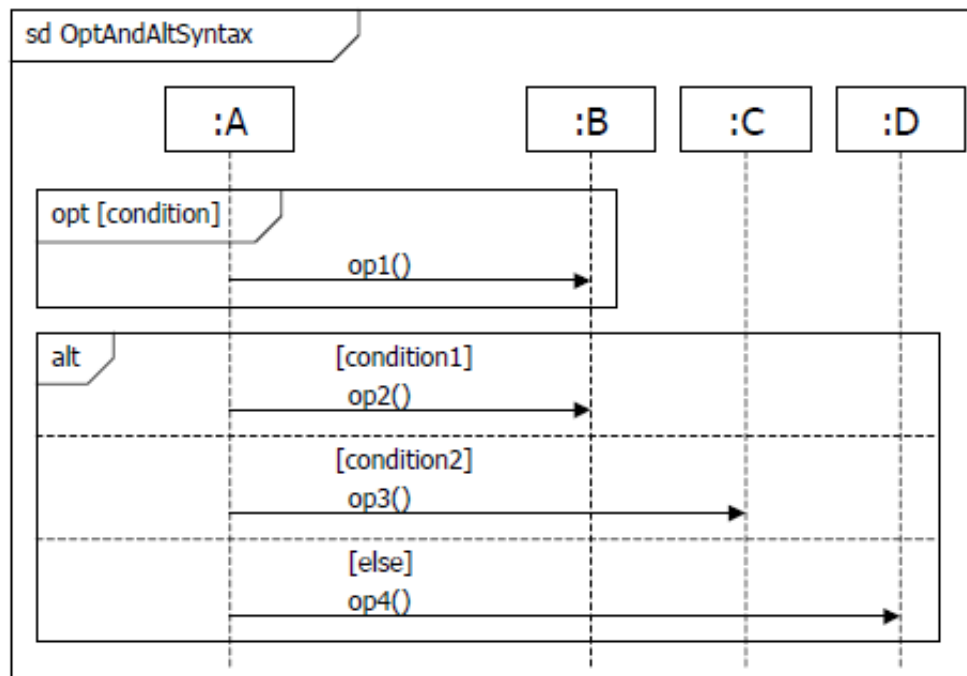
• 组合片段

– 可选执行Opt

- 存在单一运算单元，若条件为真就执行
- 相当于if语句

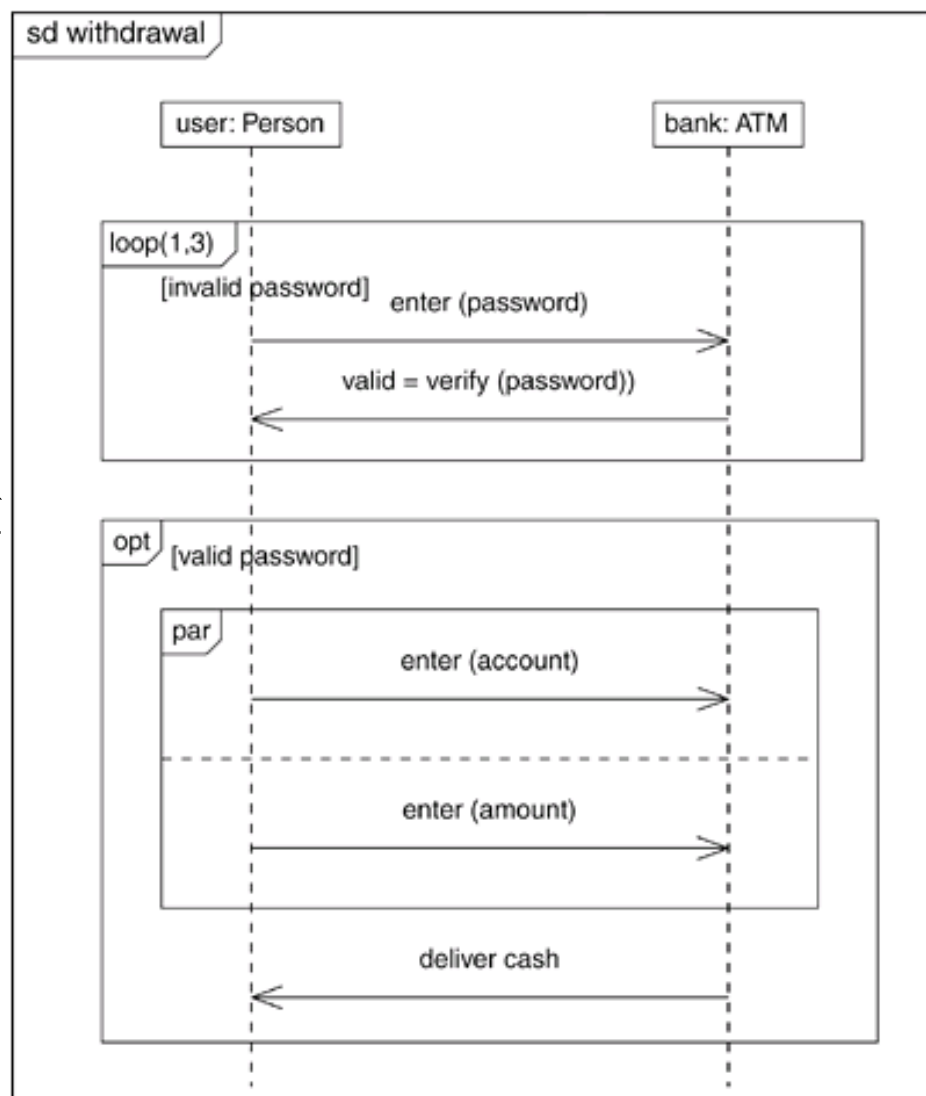
– 条件执行Alt

- 多个运算单元，条件为真的单元运行
- 相当于select...case语句



顺序图

- 组合片段
 - 循环执行Loop
 - 每次迭代前判断监护条件
 - 可以指明最小/最大循环次数
 - 并行执行Par
 - 每个区域内的计算单元并发执行
 - 分区内的消息顺序执行



顺序图



• UML定义的组合片段

Operator	Long name	Semantics
opt	option	There is a single operand that executes if the condition is true (like if ... then)
alt	alternatives	The operand whose condition is true is executed. The keyword else may be used in place of a Boolean expression (like select... case)
loop	loop	This has a special syntax: loop min, max [condition] loop min times then while condition is true, loop (max - min) times
break	break	If the guard condition is true the operand is executed <i>not</i> the rest of the enclosing interaction
ref	reference	The combined fragment refers to another interaction
par	parallel	All operands execute in parallel
critical	critical	The operand executes atomically without interruption
assert	assertion	The operand is the only valid behavior at that point in the interaction - any other behavior would be an error Use this as a way of indicating that some behavior <i>must</i> occur at a certain point in the interaction

顺序图



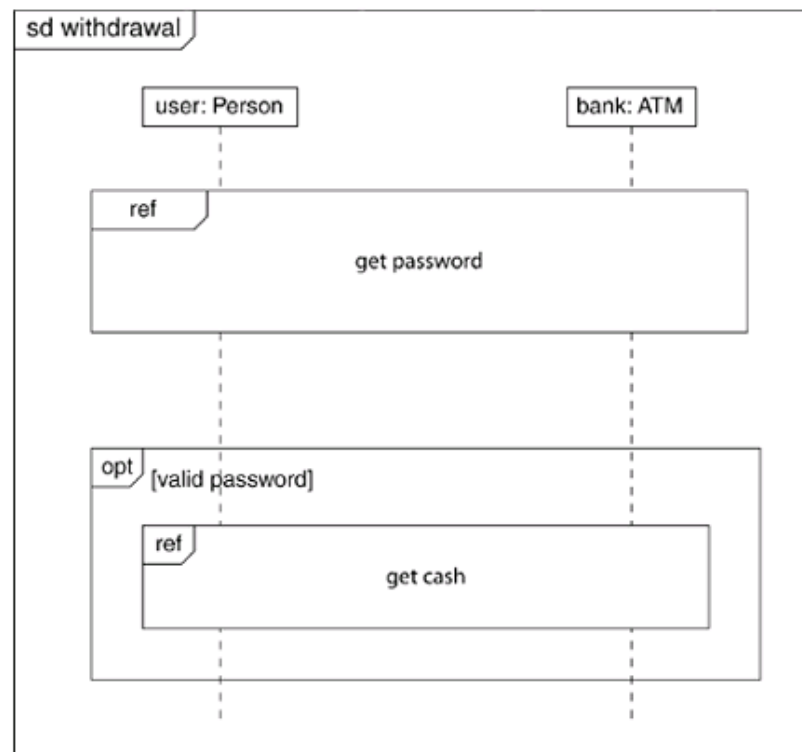
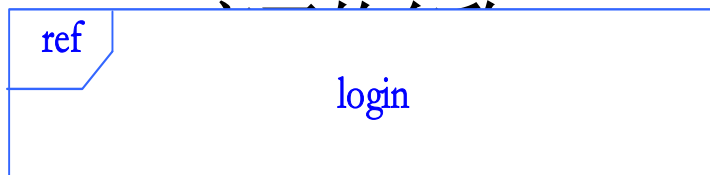
• UML定义的组合片段（续）

Operator	Long name	Semantics
seq	weak sequencing	All operands execute in parallel subject to the following constraint: events arriving on the <i>same</i> lifeline from <i>different</i> operands occur in the same sequence as the operands occur This gives rise to a weak form of sequencing - hence the name
strict	strict sequencing	The operands execute in strict sequence
neg	negative	The operand shows invalid interactions Use this when you want to show interactions that <i>must not</i> happen
ignore	ignore	Lists messages that are intentionally omitted from the interaction - the names of the ignored messages are placed in braces in a comma delimited list after the operator name e.g. {m1, m2, m3} For example, an interaction might represent a test case in which you choose to ignore some of the messages
consider	consider	Lists messages that are intentionally included in the interaction - the names of the messages are placed in braces in a comma delimited list after the operator name For example, an interaction might represent a test case in which you choose to include a subset of the set of possible messages

顺序图



- 组合片段的嵌套
 - 不同的组合片段间可以嵌套
 - 用ref表示对其他交互的引用
 - 分解过大的顺序图
 - 在ref片段中指明子



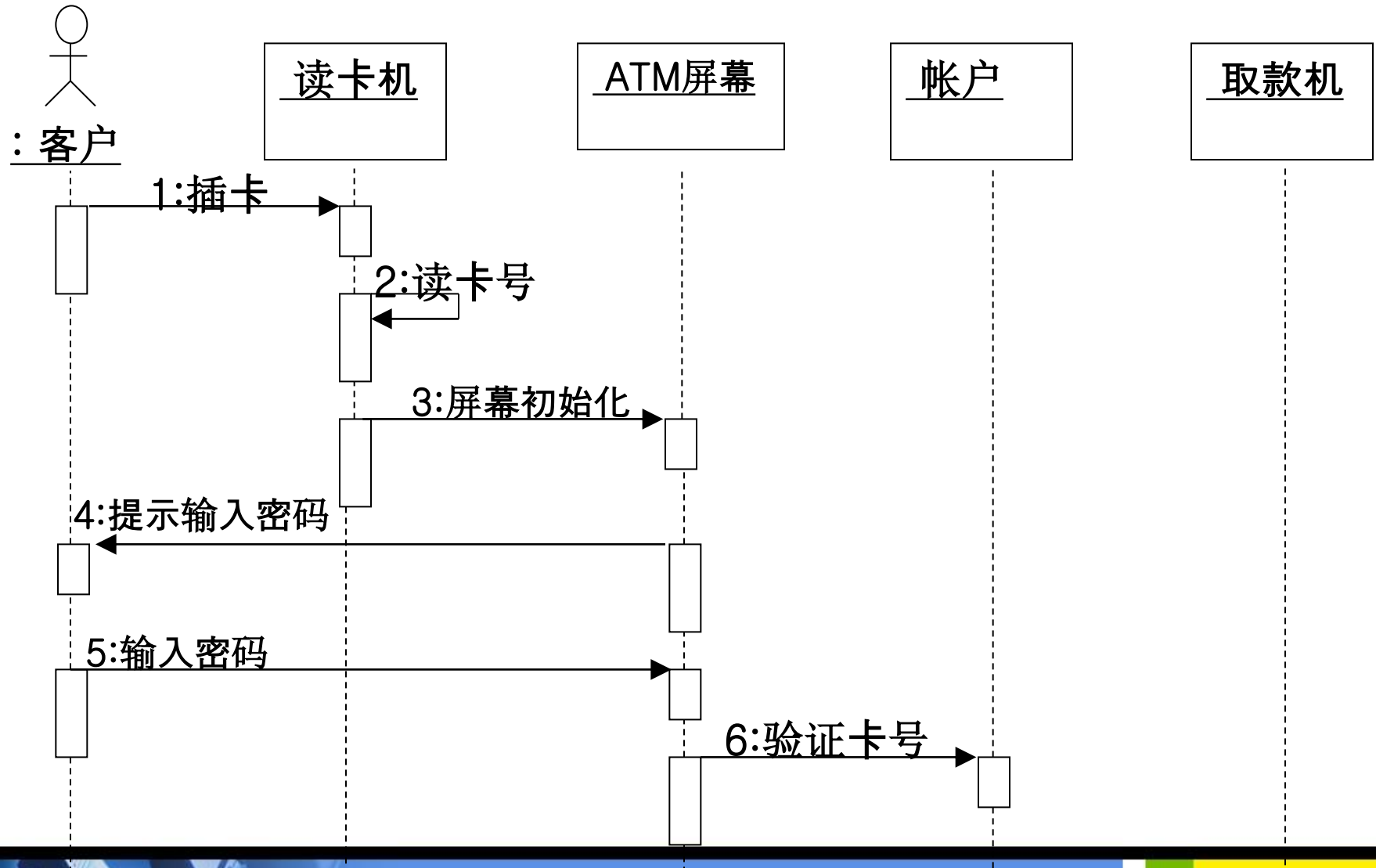
建立顺序图的步骤

- ①确定交互过程的上下文
- ②识别参与交互过程的对象
- ③为每个对象设置生命线，即确定哪些对象存在于整个交互过程中，哪些对象在交互过程中被创建和撤消
- ④从引发这个交互过程的初始消息开始，在生命线之间自顶向下依次画出随后的各个消息

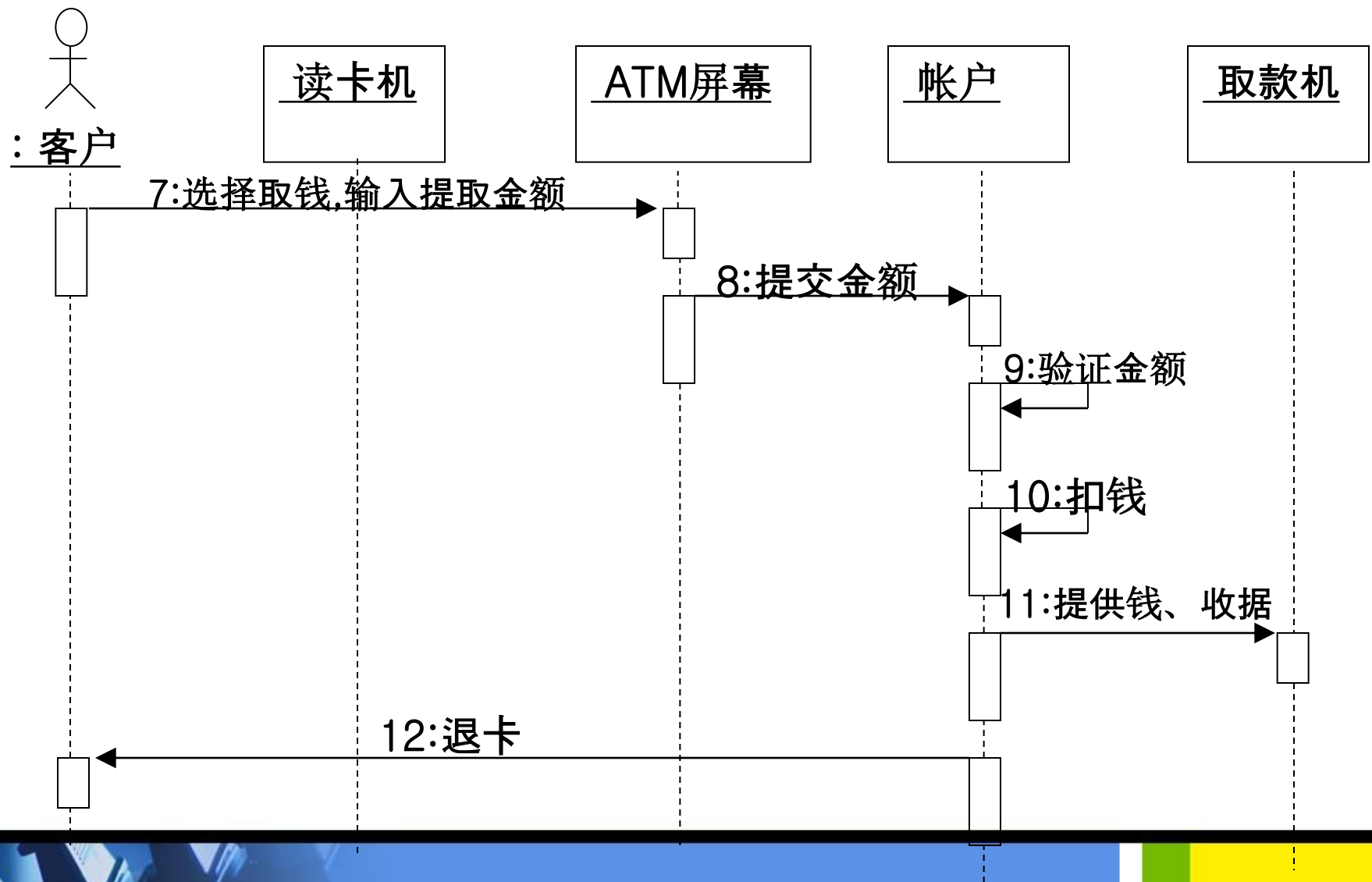
建立顺序图的步骤

- ⑤如果需要表示消息的嵌套，或/和表示消息发生时的时间点，则采用控制焦点
- ⑥如果需要说明时间约束，则在消息旁边加上约束说明
- ⑦如果需要，可以为每个消息附上前置条件和后置条件

建立顺序图的步骤



建立顺序图的步骤

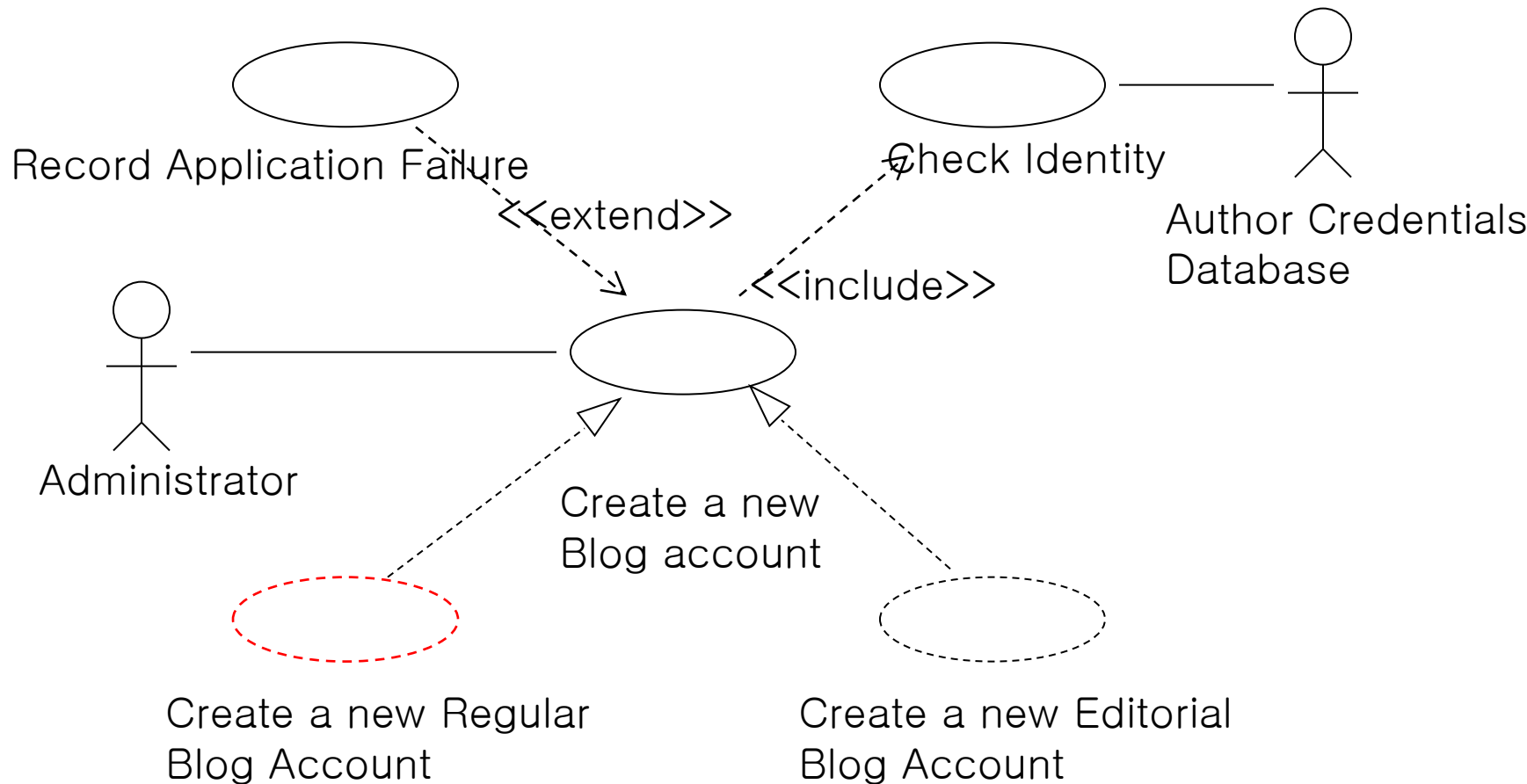


顺序图——交互图

- **顺序图**用来表示**用例**中的行为顺序。
- 当执行一个用例行为时，顺序图中的每条消息对应了一个类操作或状态机中引起转换的触发事件。
- 一个用例需要多个顺序图或协作图
- 顺序图通常用来描述一个用例的行为，显示该用例中所涉及的对象和这些对象之间的消息传递情况。

顺序图——交互图

• 用例图



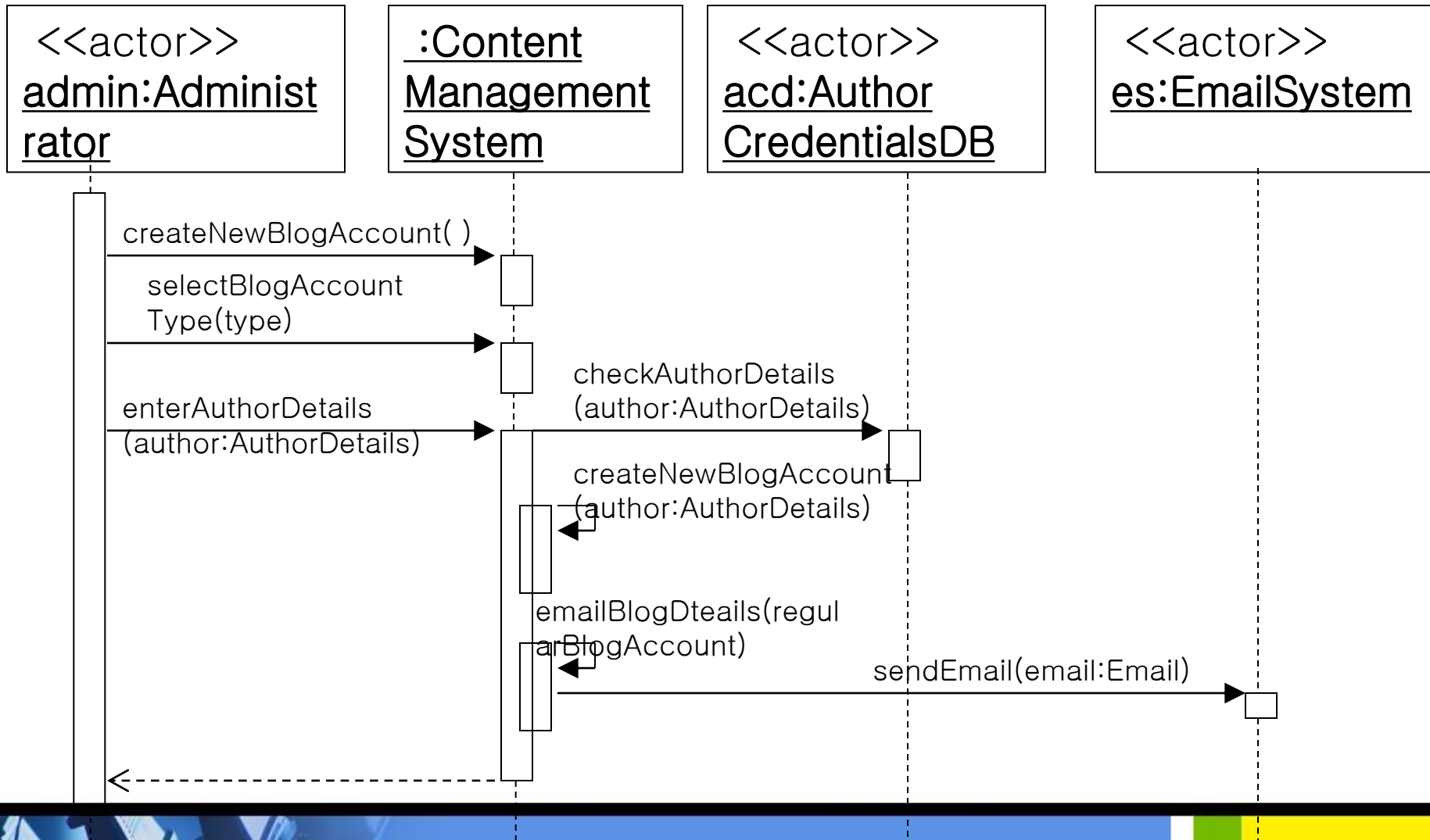
顺序图——交互图

- 用例“Create a new Regular Blog Account”的用例描述

主要流程	步骤	动作
	1	管理者要求系统创建一个新博客帐户
	2	管理者选择一般（ regular ）的帐户类型
	3	管理者输入作者详细数据
	4	使用作者凭证数据库，验证作者详细数据
	5	创建新的一般型的博客帐户
	6	通过电子邮件将新博客帐户详细数据的摘要发给作者

顺序图——交互图

用例“Create a new Regular Blog Account”的顺序图



建立顺序图的步骤

- 总结：

- 激活期代表一个对象直接或间接的执行一个动作（操作）的时间。
- 激活矩形的高度代表激活持续时间。
- 激活期可以被理解成C语言中一对花括号“{ ... }”中的内容。
- 一个单独的顺序图只能显示一个控制流。即用例的一个脚本。
- 一般来说，一个完整的控制流肯定是复杂的，因此需要多个顺序图来描述。
- 一些顺序图是主要的，另一些顺序图用来描述可选择的路径和一些例外。
- 再用一个包，对它们进行统一的管理。这样就可以用一些交互图来描述一个冗大复杂的控制流。

通信图(协作图)

- **协作图**是用于描述系统的行为是如何由系统的成分协作实现的图
- **协作图**强调参加交互的各对象的组织
- **协作图**只对相互间有交互作用的对象和这些对象间的关系建模，忽略其它对象和关联
- **协作图**可以被视为对象图的扩展，但它除了展现出对象间的关联外，还显示出对象间的消息传递

通信图(协作图)

- 协作图描述的是和对象结构相关的信息
- 协作图的一个用途是表示类操作的实现
- 协作图可以说明类操作中用到的参数、局部变量、操作中的永久链。
- 当实现一个行为时，消息编号对应了程序中嵌套的调用结构和信号传递过程。

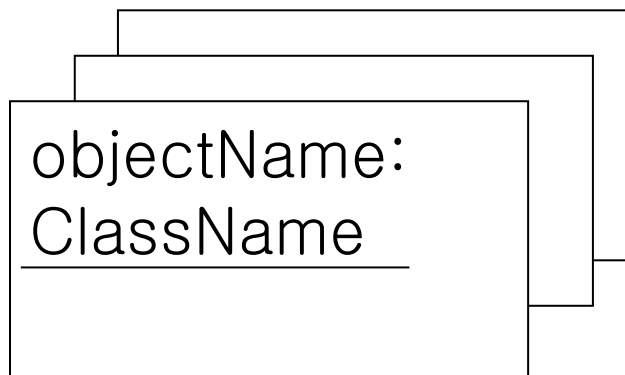
通信图(协作图)

- 协作图中的建模元素：
 - 对象（参与者实例也是对象，多对象，主动对象）
 - 链
 - 消息

通信图(协作图)

- **多对象:**

- 多对象指的是由多个对象组成的对象集合
- 一般情况下，这些对象属于同一个类
- 当需要把消息同时发送给多个对象，而不是单个对象时，使用多对象概念



通信图(协作图)

- 主动对象:

- 主动对象是一组属性和一组方法的封装体，其中至少有一个方法不需要接收消息，就能主动执行（称为主动方法）。
- 主动对象可以在不接收外部消息的情况下，自己开始一个控制流。
- 除含有主动方法外，主动对象的其他方面与被动对象没有区别。

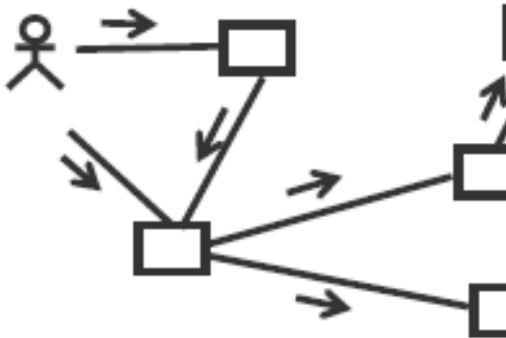
objectName:
ClassName

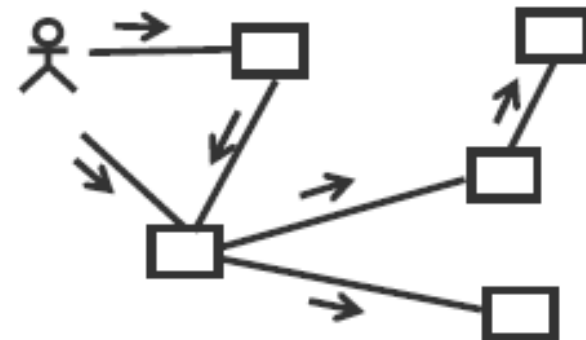
:Robot

active

Rose中的
主动对象

• 通信图(协作图)

- 强调发送和接受消息的对象的结构组织
 - 顶点和弧组成
 - 顶点：对象和角色
 - 弧：链上的消息
 - 特征
 - 路径：对象之间的链
 - 消息的序号
- 
- Communication Diagrams



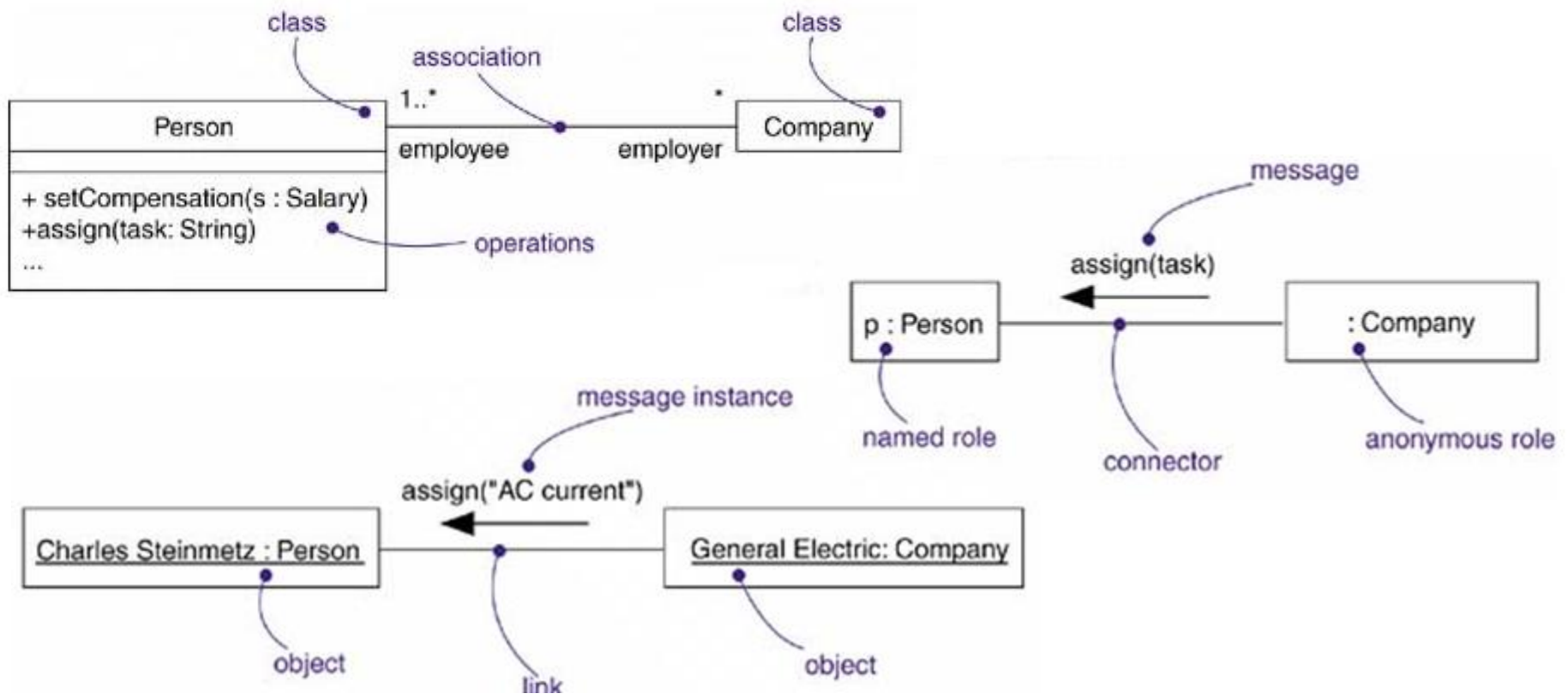
Communication Diagrams

通信图(协作图)

- 链 (link)和连接件
 - 对象间的语义连接
 - 指明一个对象向另一对象发送消息的路径
 - 原型化的链 ← 连接件 (connector)
 - 在链上可以加一些修饰，如角色名，导航，链两端是否有聚集关系
 - 修饰链有关端点的约束
 - 关联 (association)
 - 自身 (self)
 - 全局 (global)
 - 局部 (local)
 - 参数 (parameter)

通信图(协作图)

- 链 (link)和连接件



通信图(协作图)

- 消息：
 - 消息代表协作图中对象间通过链接发送的消息
 - 消息的箭头指向接受消息的对象
 - 消息流上标有消息的序列号和对象间发送的消息
 - 一条消息会触发接收对象中的一项操作

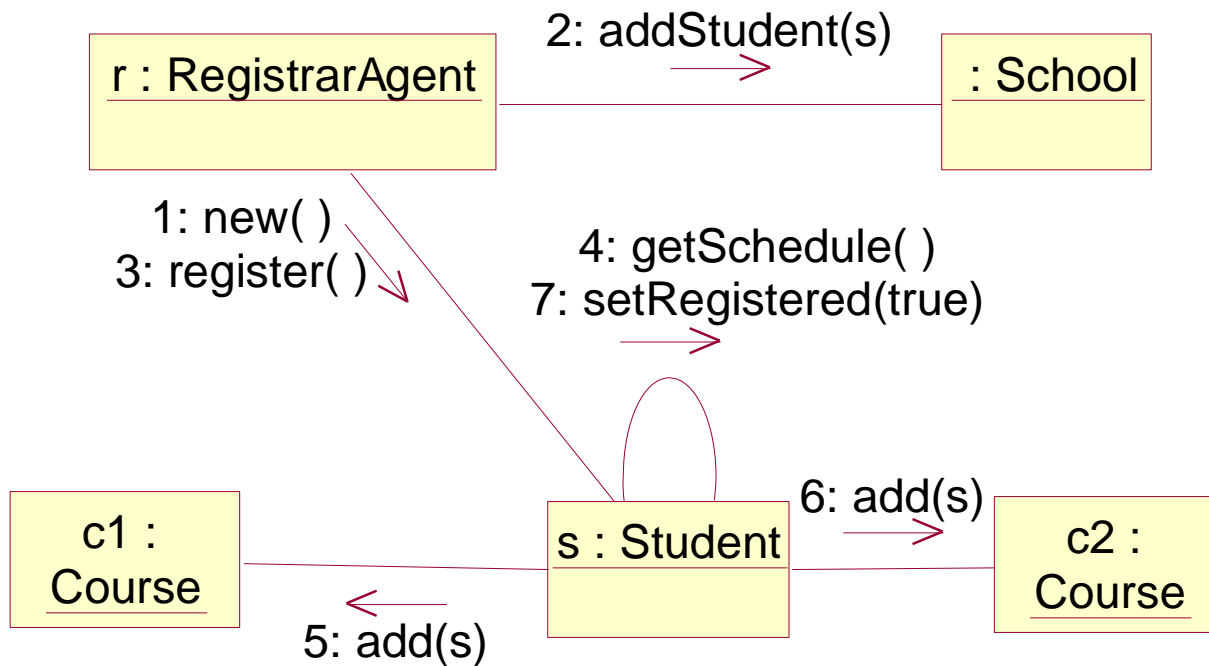
通信图(协作图)

- 消息:

- 消息的序列号（顺序号）是消息的一个数字前缀，是一个整数，由1开始递增。
- 通过整数之间的“点”表示法，可以描述控制的嵌套关系。
- 例如，有：消息1，消息1.1，消息1.2。
消息1.1和1.2是嵌套在消息1中的，且消息1.1排在消息1.2的前面

通信图(协作图)

- 消息序号
 - 表示消息流动的时间顺序
 - 每个消息有唯一序号
 - 同一个链上可以有多个消息

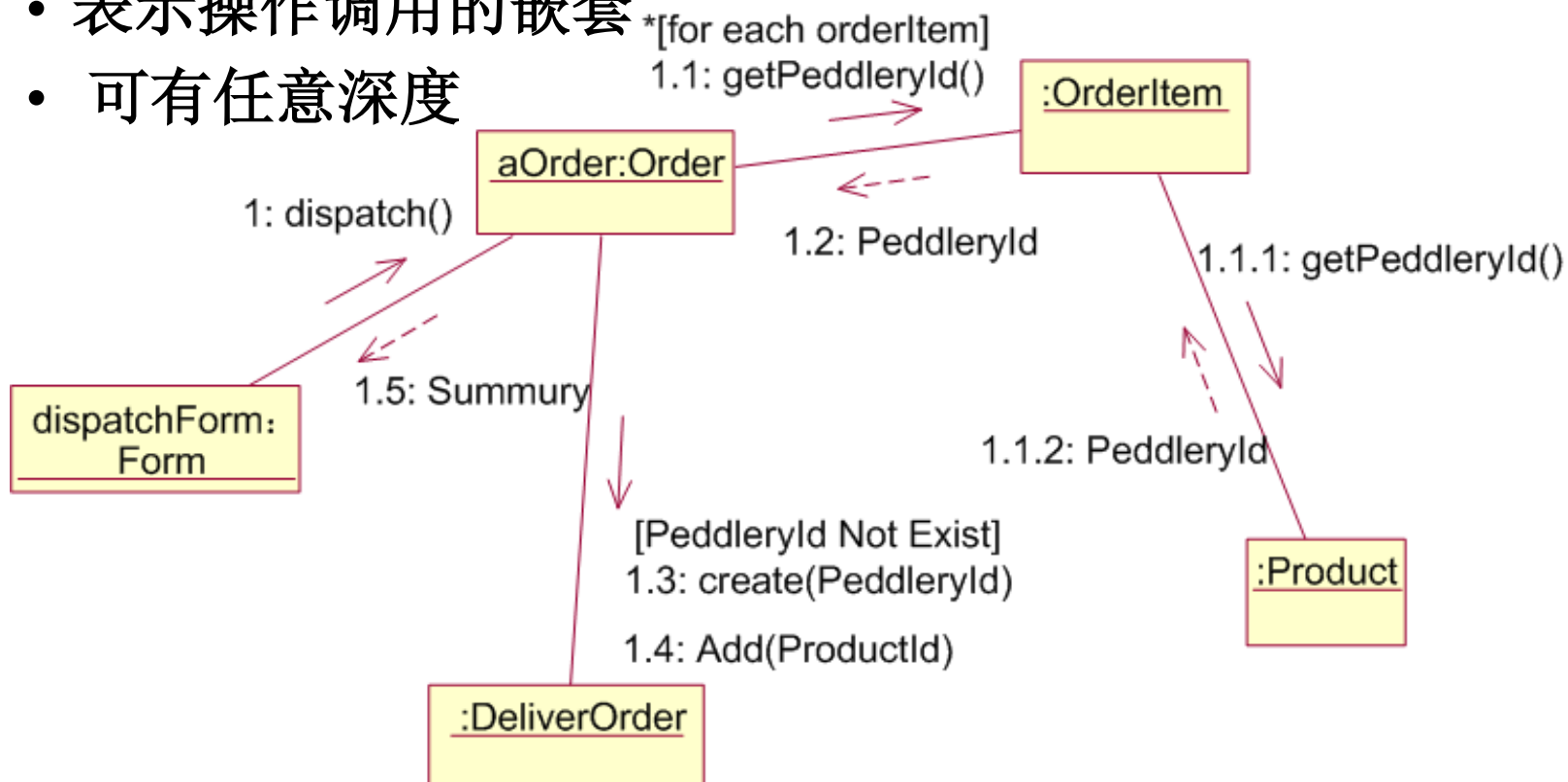


通信图(协作图)

- 消息序号

- 嵌套的序号

- 表示操作调用的嵌套
 - 可有任意深度



通信图(协作图)

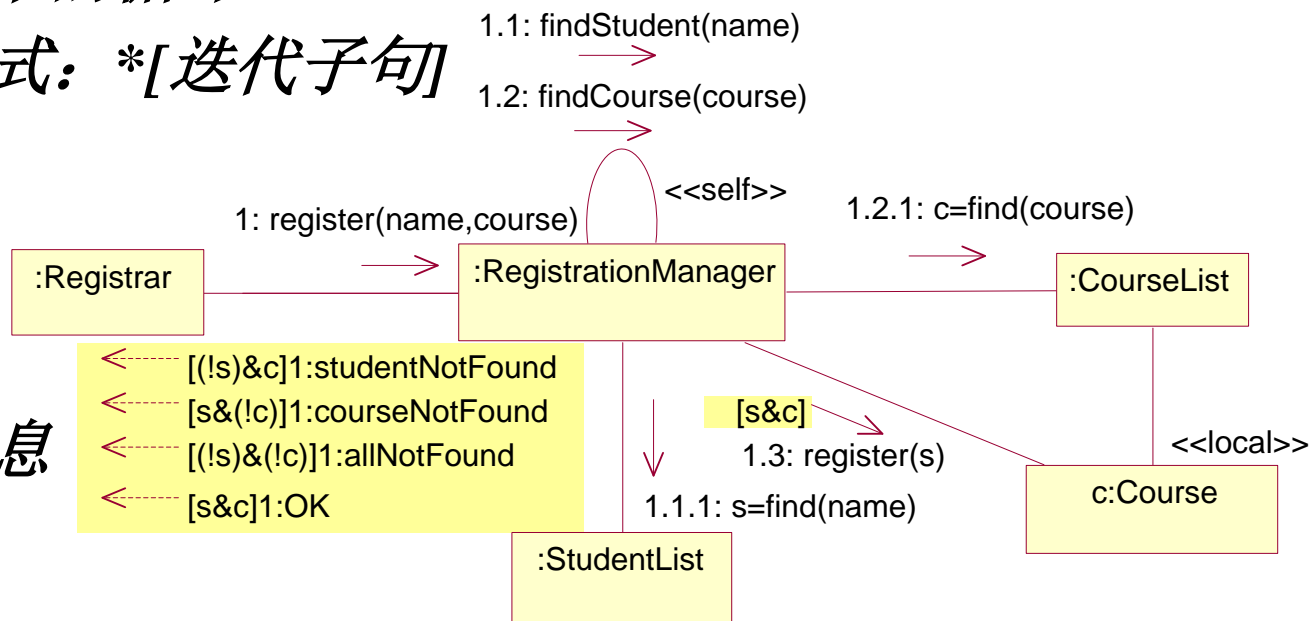
• 消息

– 迭代

- 表示通信中的循环
- 迭代表达式: $*[$ 迭代子句

– 分支

- 监护条件
- 条件为真才发送消息



建立协作图的步骤

- ①确定交互过程的上下文
- ②识别参与交互过程的对象
- ③如果需要，为每个对象设置初始特性
- ④确定对象之间的链，以及沿着链的消息
- ⑤从引发这个交互过程的初始消息开始，将随后的每个消息附到相应的链上

建立协作图的步骤

- ⑥如果需要表示消息的嵌套，则用Dewey十进制数表示法
- ⑦如果需要说明时间约束，则在消息旁边加上约束说明
- ⑧如果需要，可以为每个消息附上前置条件或后置条件

顺序图和协作图的比较

- 顺序图：强调的是消息的时间顺序。
- 协作图：强调的是参与交互的对象的组织。
- 顺序图：建模元素有生命线和控制焦点。
- 协作图：建模元素有路径，消息必须有消息顺序号。
- 顺序图：在表示算法、对象的生命期、具有多线程特征的对象等方面，相对来说更容易一些。
- 协作图：如果按组织对控制流建模，应该选择使用协作图

顺序图和协作图的比较

- 顺序图：不能表示对象与对象之间的链。对于多对象和主动对象，也不能直接显示出来。
- 协作图：不能表示生命线的分叉。
- 但是，两者之间可以相互转换，但不能完全相互代替。

通信图(协作图)

- 顺序图VS通信图(协作图)

- 侧重不同

- 顺序图显示外在的消息序列，适用于表达较复杂的场景和实时的规格说明
 - 通信图(协作图)表达对象间的组织关系，适用于理解给定对象集合上的交互

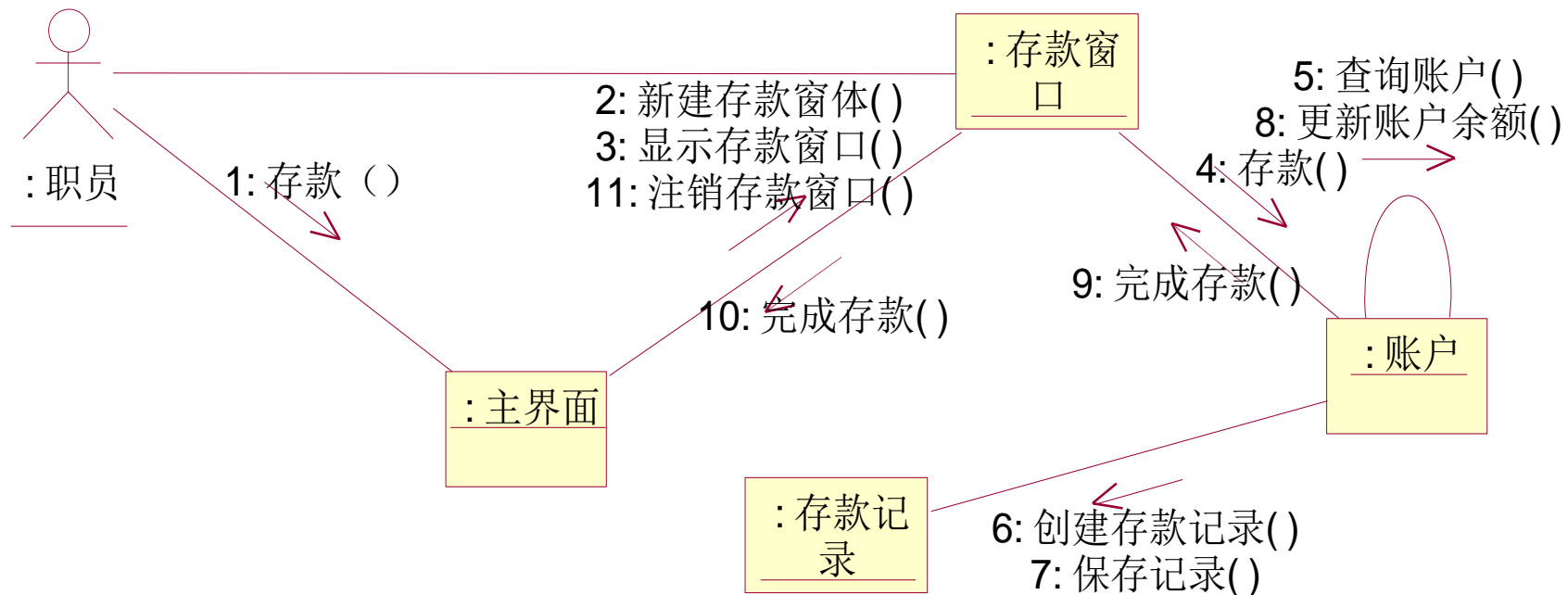
- 语义等价

- 表达了相同的信息
 - 可以互相转换而不丢失信息

通信图(协作图)



• 银行职员成功存款的通信图(协作图)



通信图(协作图)



- 交互图

- 用于描述用例的行为

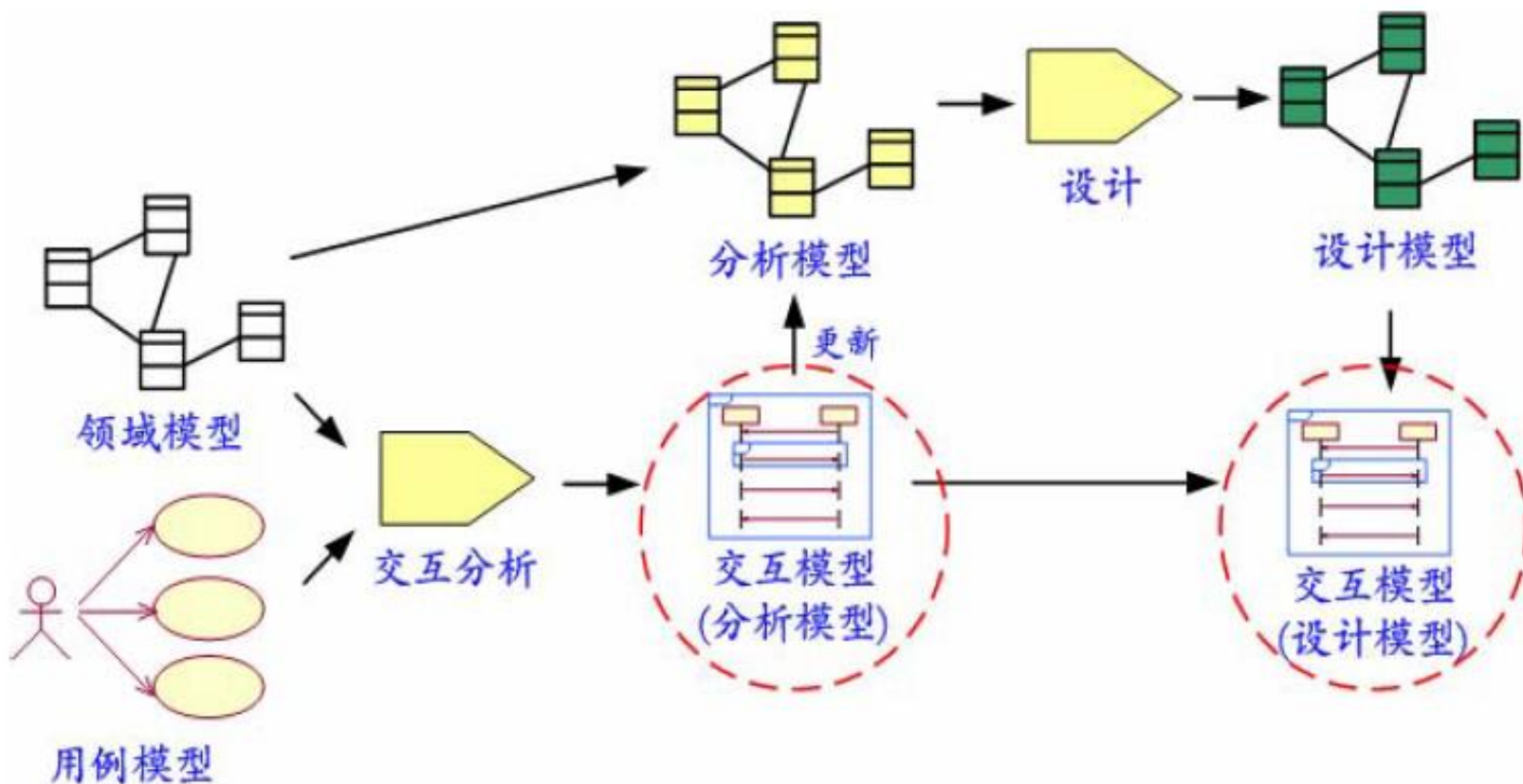
- 显示用例涉及的对象及对象间的消息传递关系
 - 包含在用例实现中

- 适应于

- 条件判断和循环不太多的交互
 - 条件过多时使用多个交互图描述

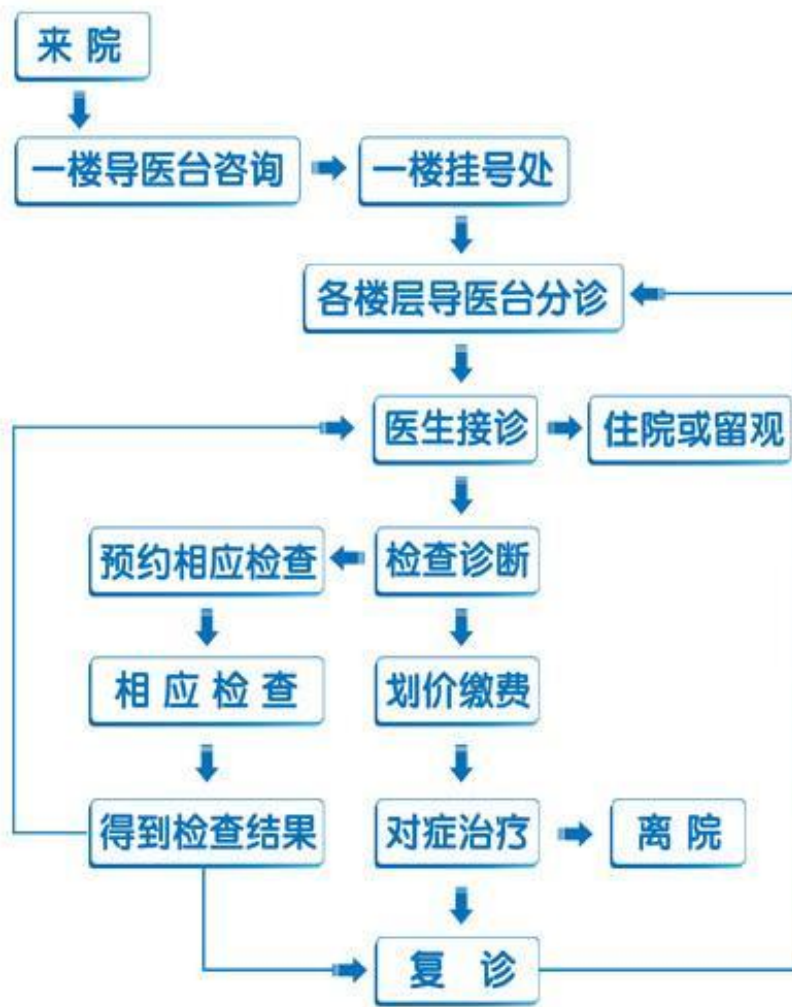
通信图(协作图)

- 交互模型的类型与演变



活动图

- 活动图
 - 一个流程图的例子：就诊流程



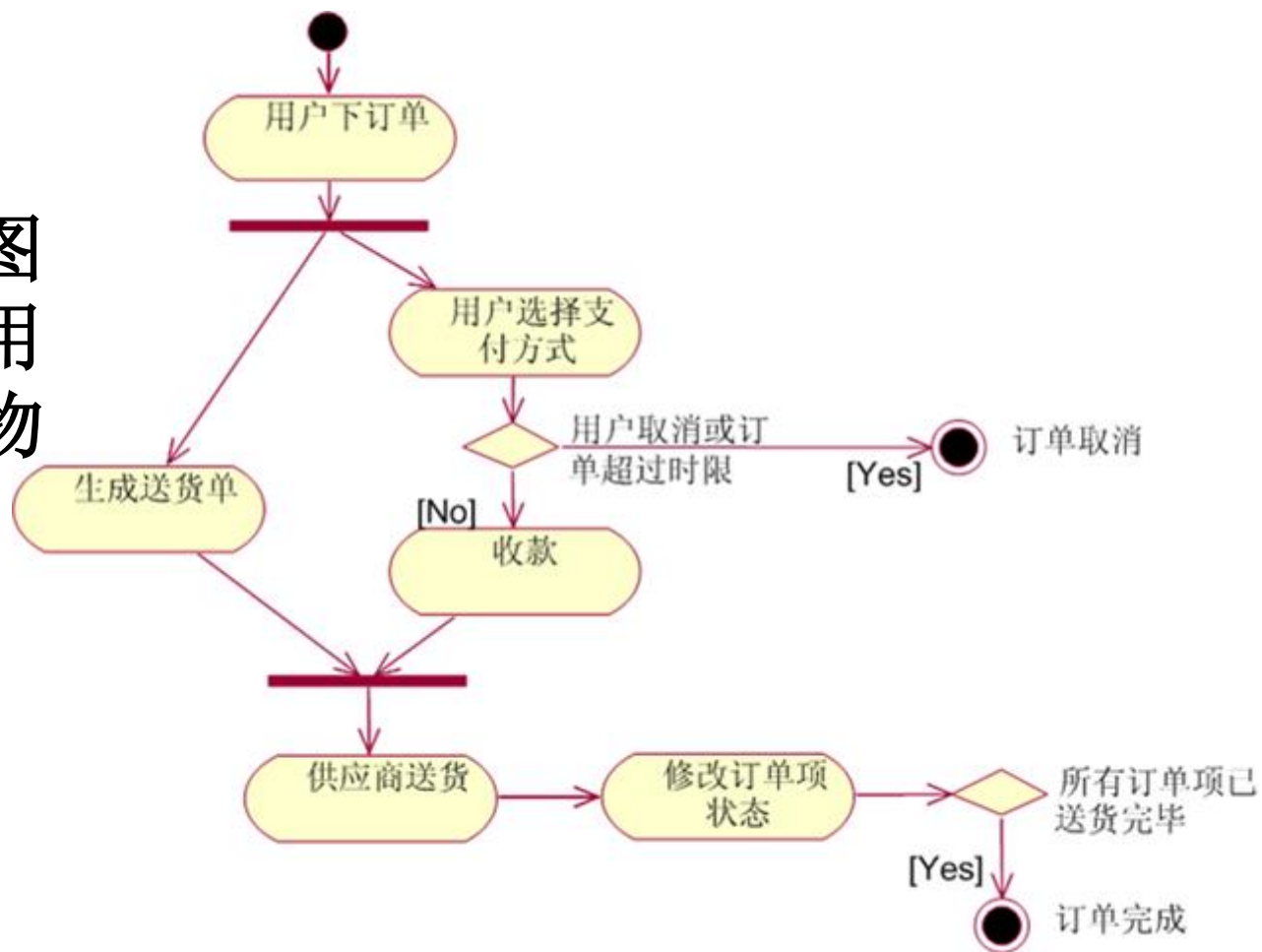
- 活动图

- 描述需要做的活动以及执行这些活动的顺序
 - 描述从活动到活动的控制流
 - 能够描述并发和控制分支
- 本质：OO流程图
- 典型应用
 - 描述一个 workflows 或业务流程
 - 描述一个操作

活动图

- 活动图

- 一个活动图的例子：用户在线购物的流程



- 活动图组成元素
 - 由边连接的节点网络
 - 三类节点
 - 动作节点：原子的、具体的工作单元
 - 控制节点：控制工作流的节点
 - 对象节点：活动中使用的对象
 - 两类边
 - 控制流：活动中的控制流
 - 对象流：活动中的对象流

活动图

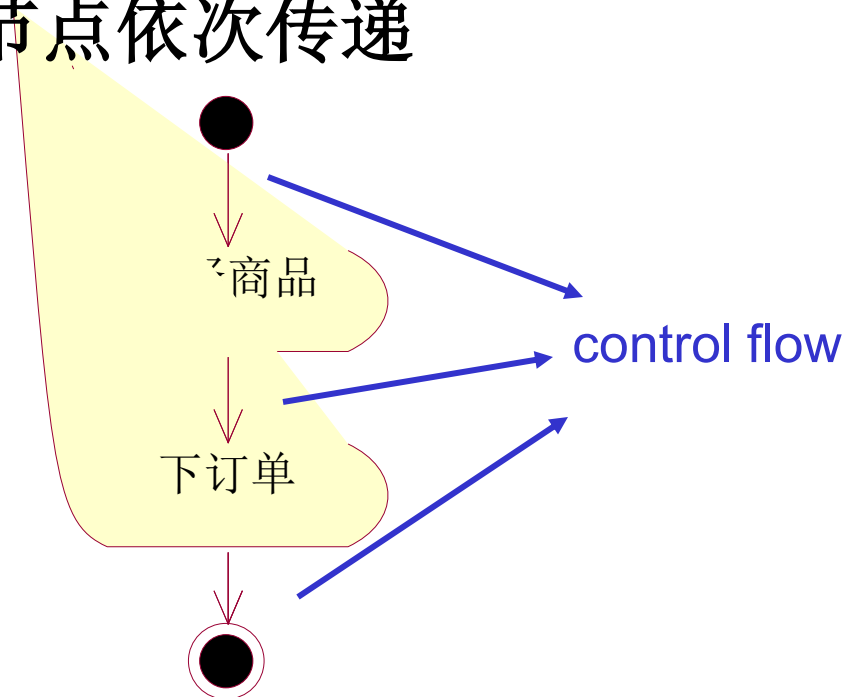


- 动作 (Action)
 - 可执行的原子计算
 - 动作描述
 - 动词 (短语)
 - 表达式
 - 原子性: 不可分解

收款

`index:=lookup(e)+7`

- 控制流 (Control flow)
 - 动作间的控制路径
 - 控制流通过每个动作节点依次传递
 - UML2活动图语义
 - Petri网
 - 控制流令牌的流动



活动图



- 控制节点 (Control node)

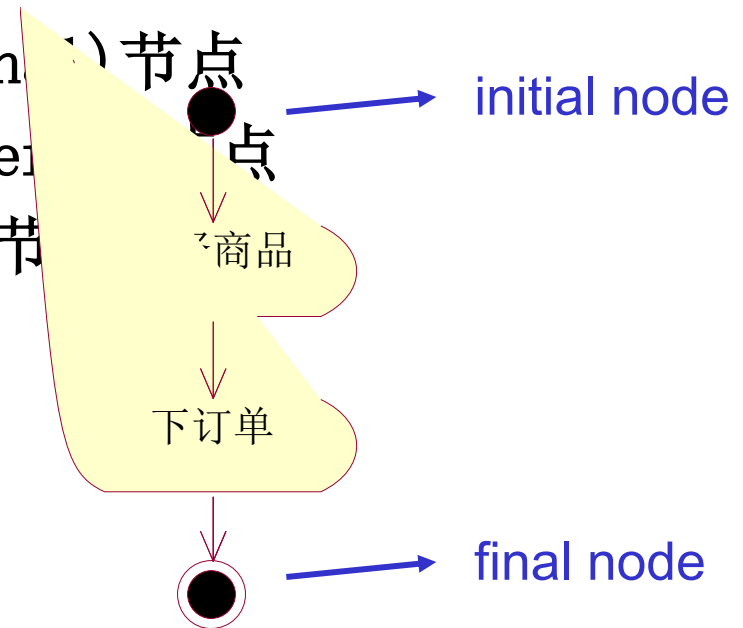
- 管理活动内的控制流

- 分为三组

- 开始 (initial)/结束 (final) 节点

- 分支 (decision)/合并 (merge) 节点

- 并发 (fork)/会聚 (join) 节点



活动图

- 分支 (decision) / 合并 (merge) 节点

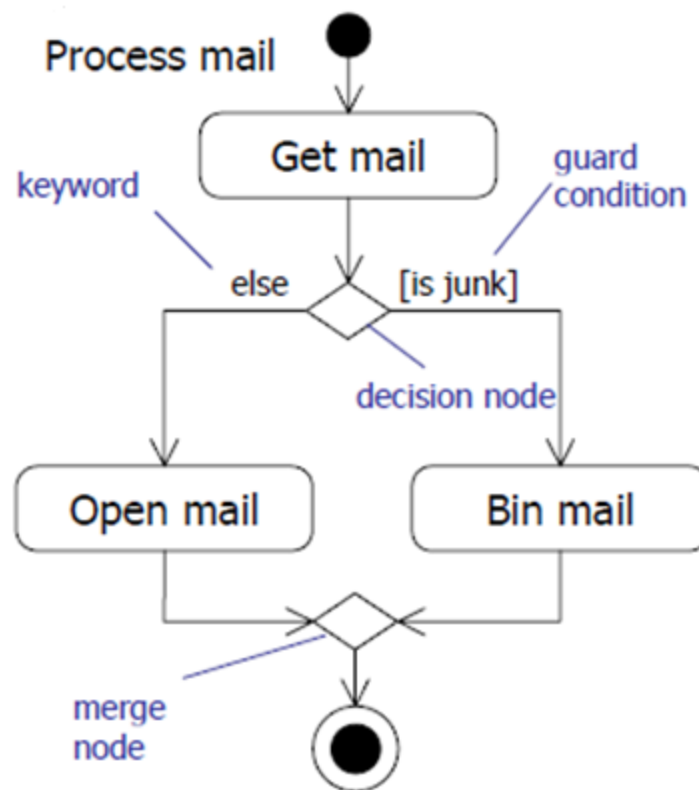
- 描述可选择路径的分支与汇合

- 分支节点

- 一个进入流；多个离去流
 - 监护条件：布尔表达式
 - 监护条件的互斥与完整性

- 合并节点

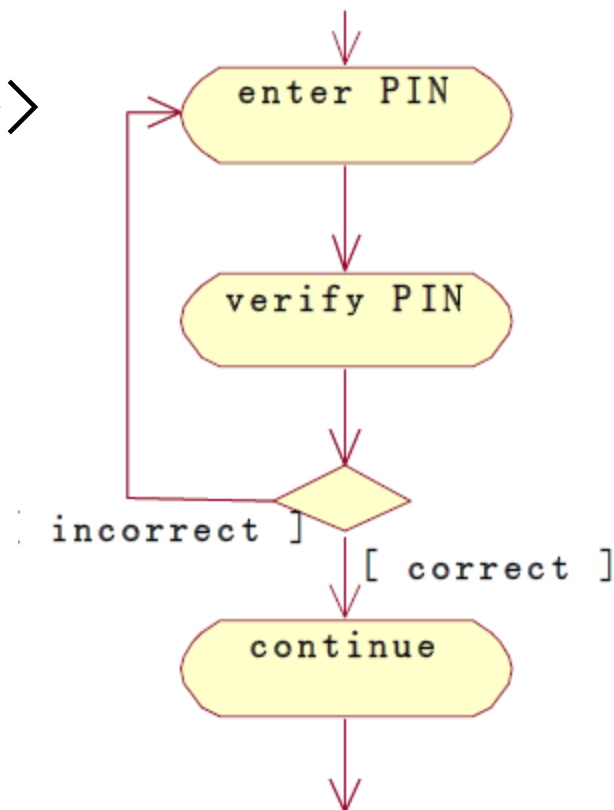
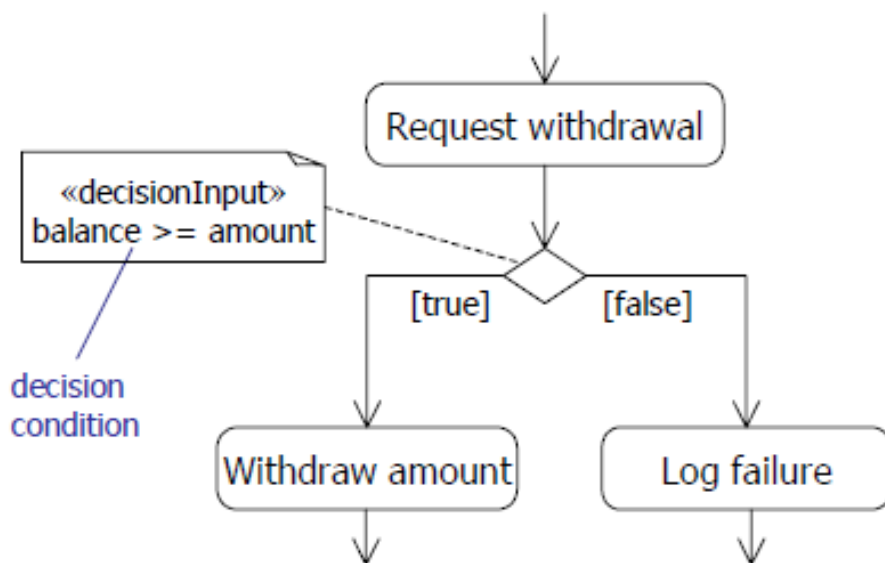
- 多个进入流；一个离去流



活动图

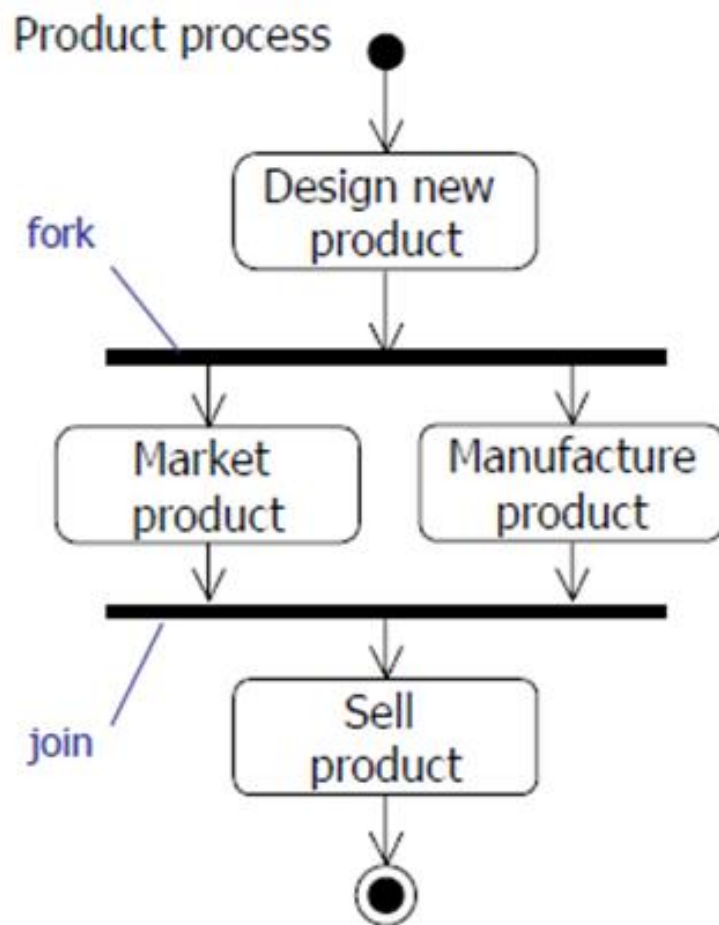
- 分支 (decision) / 合并 (merge) 节点

- 判决条件 <<decisionInput>>
- 分支/合并节点可描述循环



活动图

- 并发(fork)/会聚(join)节点
 - 描述并行控制流的分岔和汇合
 - 并发节点
 - 一个进入流；多个离去流
 - 把一个单独的控制流分成多个并发控制流
 - 会聚节点
 - 多个进入流；一个离去流
 - 并发的流取得同步

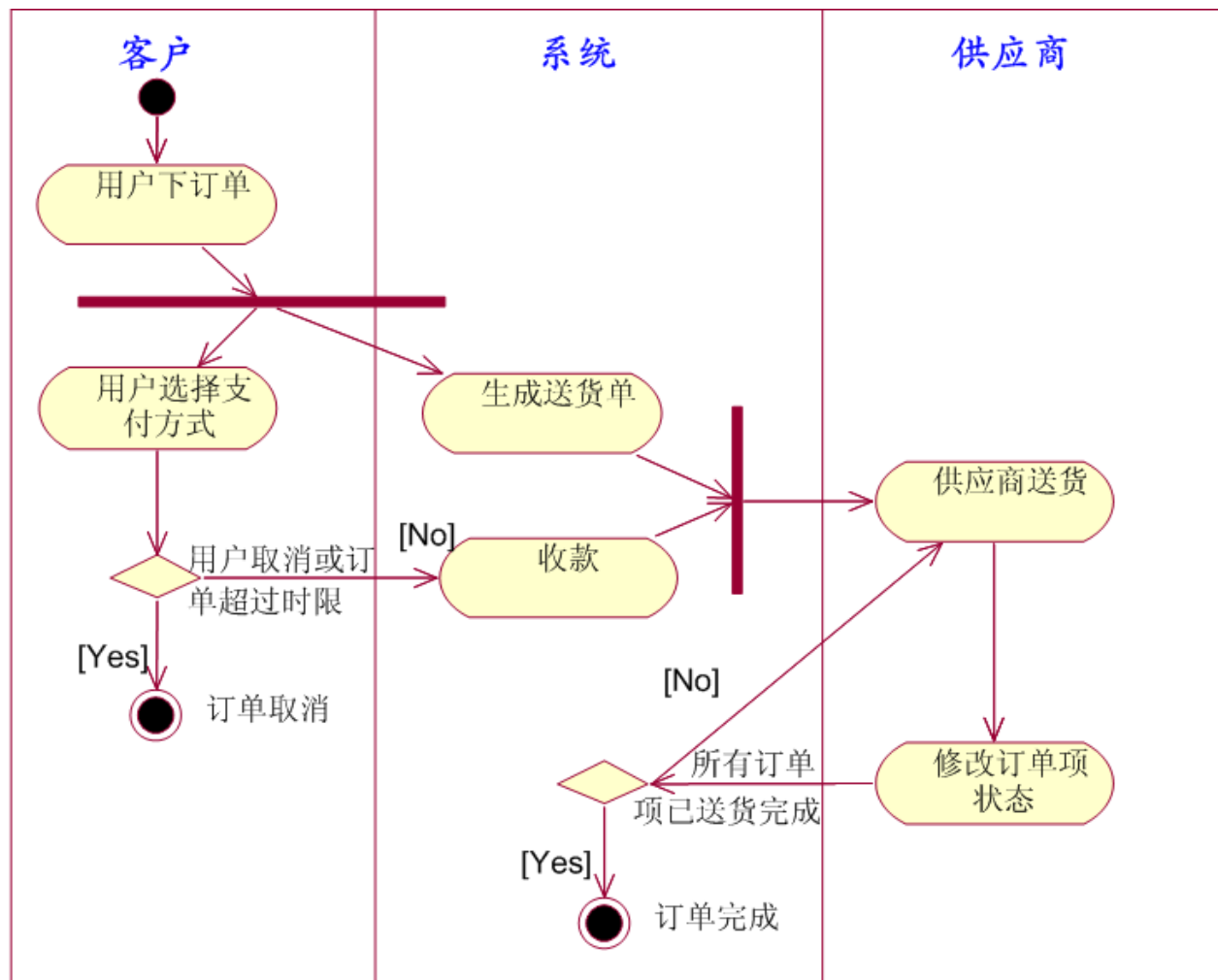


- 并发(fork)/会聚(join)节点
 - 每条路径并行执行
 - 真实并发
 - 顺序交替
 - 并行控制流中活动的通信

活动图

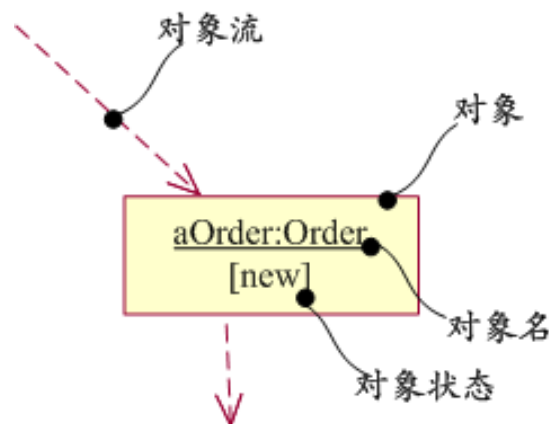
- 泳道

- 活动图中引入面向对象机制
- 显示一组活动的负责者



- 泳道 (swimlanes)
 - 活动分区 (Activity partitions)
 - 对一个活动图中相关动作的高级分组
 - 泳道的语义
 - 表示一组相关活动的高层职责
 - 分区依据：类、组件、对象、角色、组织单元、部门责任区、……
 - 每个活动属于一个泳道；转移可以跨越泳道
 - 作用：便于提取类和分析对象间的交互

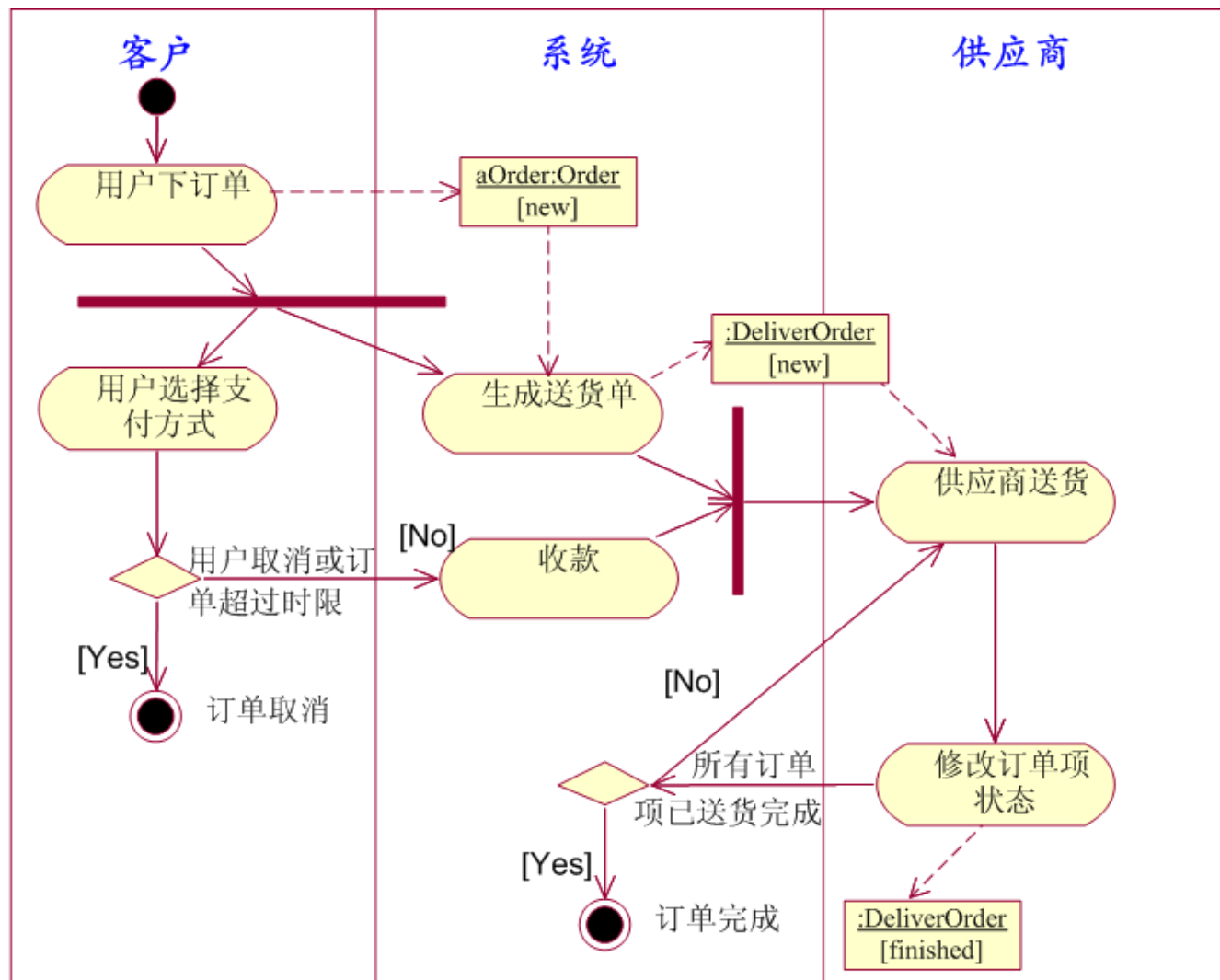
- 对象节点和对象流
 - 描述活动图中涉及的特定类的实例
 - 将对象连接到产生/使用对象的活动
 - 一个对象值从一个动作流向另一个动作
 - 表示方法
 - 对象节点：对象+[状态]
 - 对象流：连接到活动



Rose表示法

活动图

- 带对象流的活动图



- 高级活动图概念

- 动作 (Action) 类型

- 调用动作节点：调用活动、行为、操作
 - 发送信号：发送一个异步信号
 - 接收信号：等待接收一个信号
 - 时间信号：随着时间推移自动发送信号



时间信号



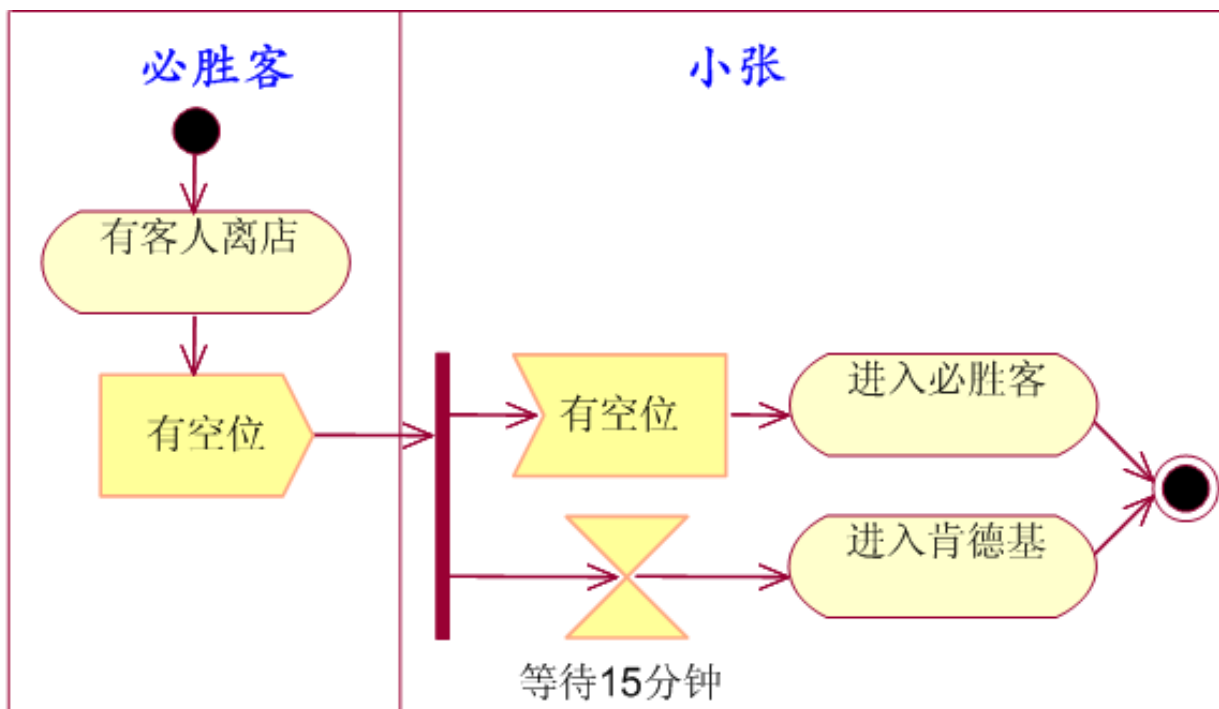
发送信号



接收信号

活动图

- 高级活动图概念
 - 发送信号，接收信号，时间信号



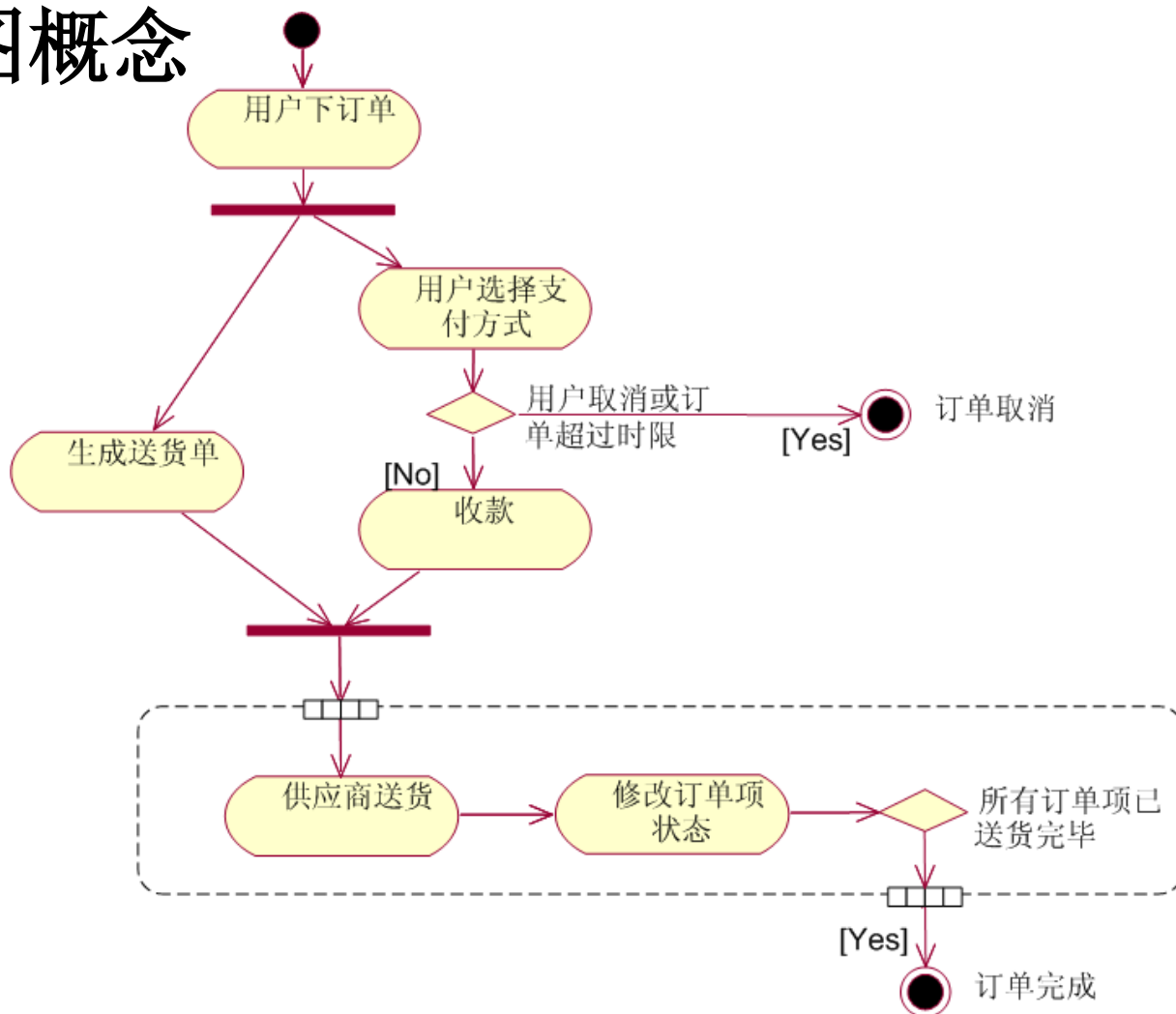
- 高级活动图概念

- 扩展区：表示在元素列集上执行的活动片段

- 输入集：流入扩展区的对象列集；一个或多个
 - 执行区域为每个元素执行一次
 - 输出集：流入扩展区的对象列集；零个或多个
 - 相当于“for all”操作

活动图

- 高级活动图概念
- 扩展区



- 绘制活动图
 - 首先决定是否采用泳道
 - 然后尽量使用分支/合并、并发/会聚等基本建模元素来描述活动控制流程
 - 如果需要，加入对象流以及对象的状态变化
 - 利用一些高级的建模元素表示更多的信息

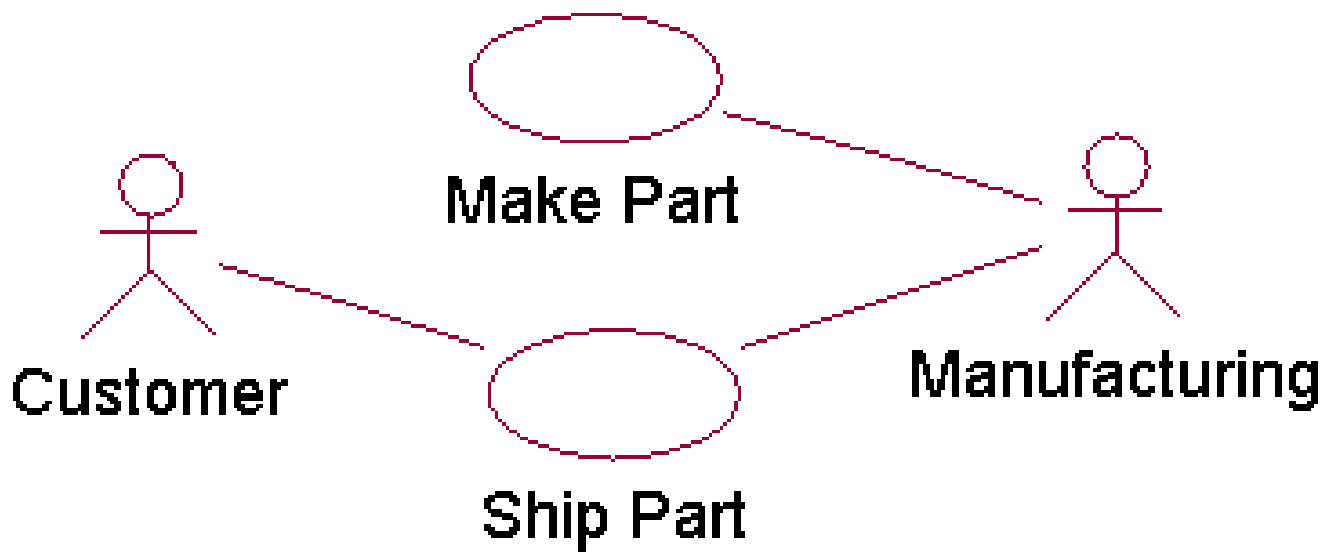
原则：活动图的建模关键是表示出控制流，其它的建模元素都是围绕这一宗旨所进行的补充

活动图的用途

- 活动图对表示并发行为很有用。
- 活动图的应用非常广泛，包括：
 - 1. 对系统的工作流(workflow)建模，即对系统的业务过程建模。(Use Case分析)
 - 2. 对具体的操作建模，描述计算过程的细节。

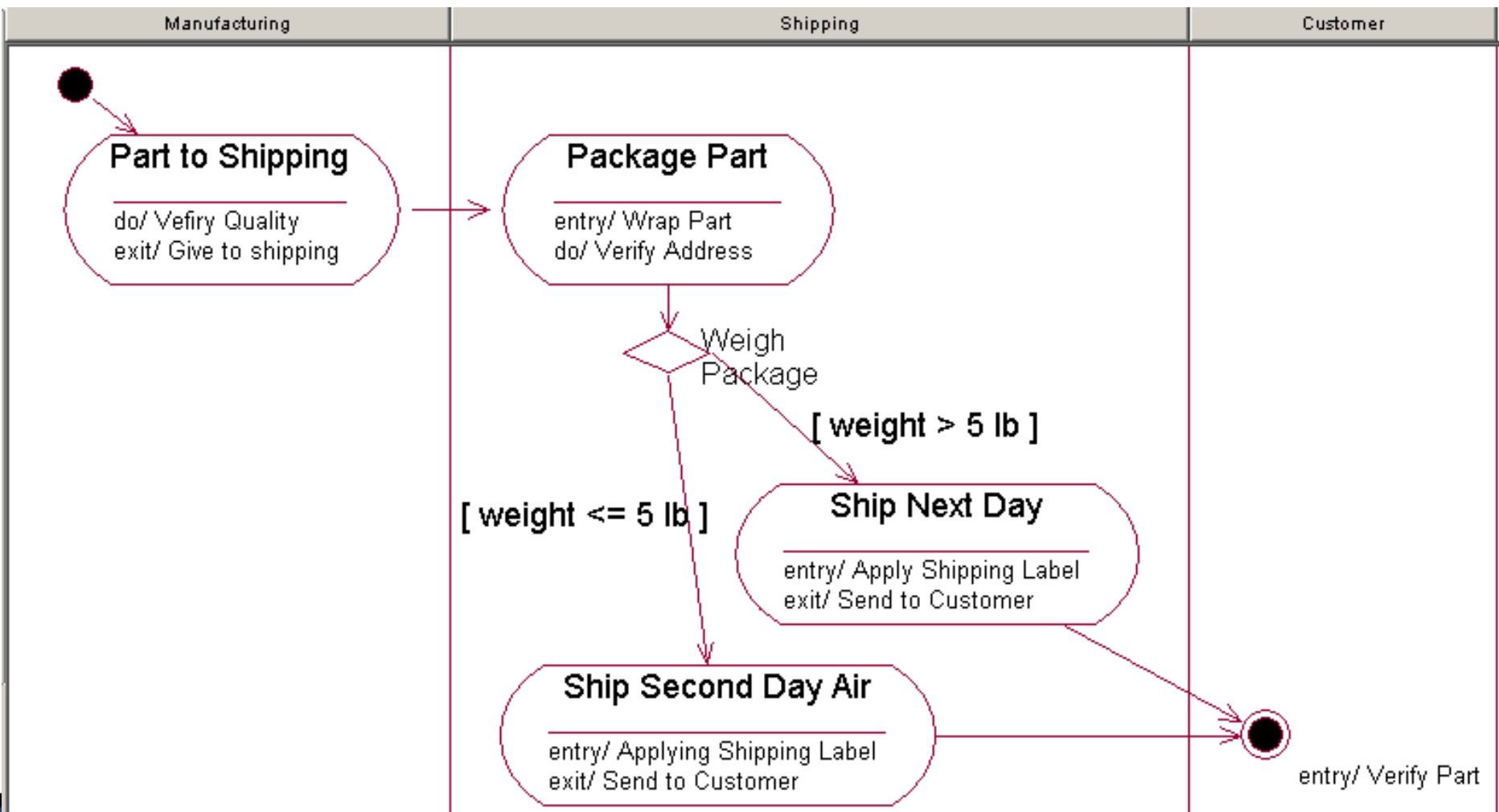
活动图的用途

- 例1：用活动图对工作流程建模的例子。
- (1)：产品制造和发货的use case图。



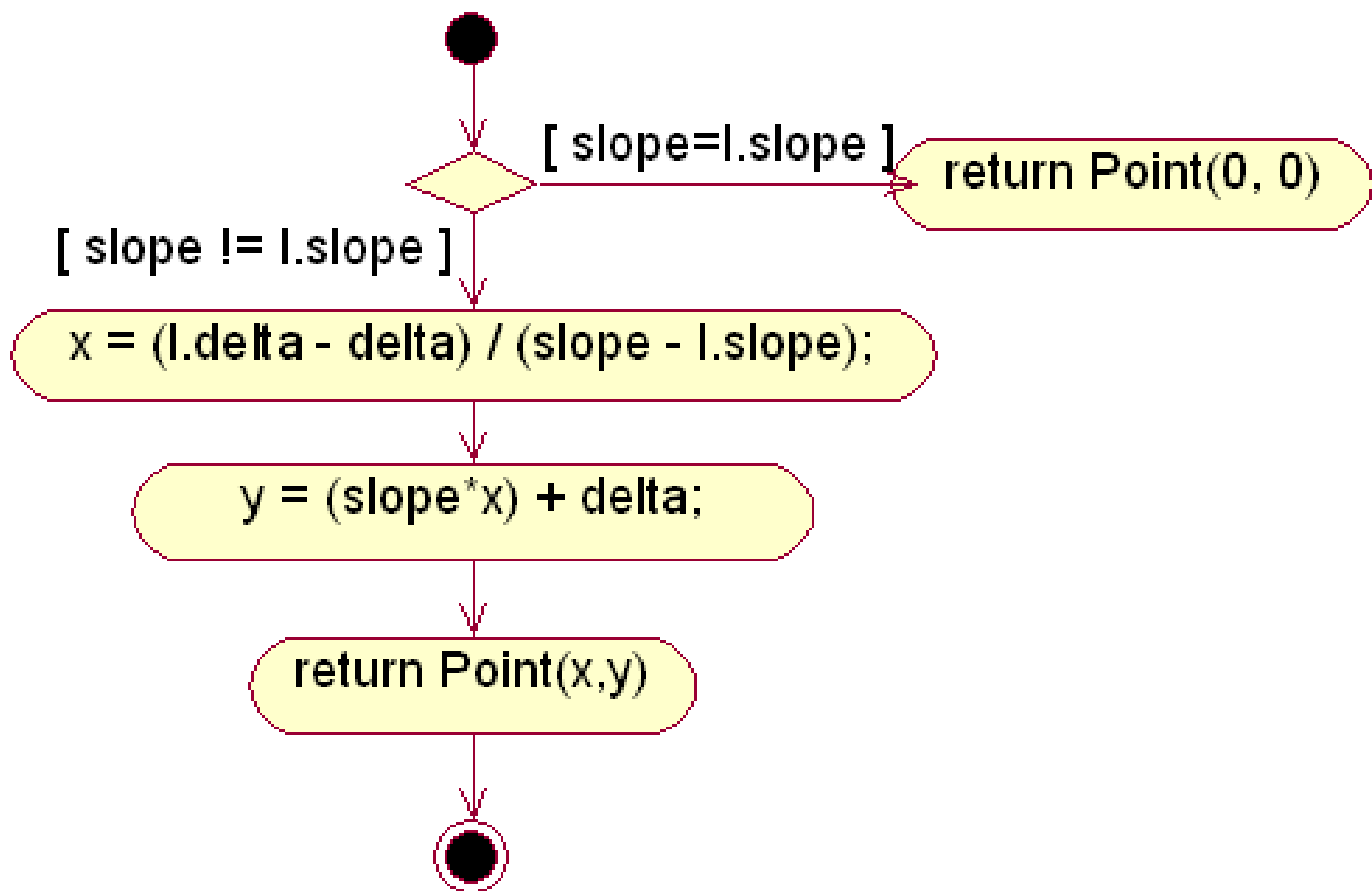
活动图的用途

- (2). 用活动图来说明具体的工作流程。



活动图的用途

- 例2：用活动图对操作建模的例子：用活动图描述类Line的操作intersection的算法。



活动图的工具支持

- 正向工程：
 - 利用活动图产生代码，特别是对具体操作建模的活动图。
 - 例：根据intersection活动图生成的代码：

```
Point Line::intersection (l : Line) {  
    if (slope == l.slope) return Point(0,0);  
    int x = (l.delta - delta) / (slope - l.slope);  
    int y = (slope * x) + delta;  
    return Point(x, y);  
}
```

- 活动图的作用
 - 在业务建模阶段
 - 建模业务流程
 - 在分析阶段
 - 描述用例场景
 - 建模用例间的控制流（交互概述图）
 - 在设计阶段
 - 建模操作/算法的细节
 - 描述多线程

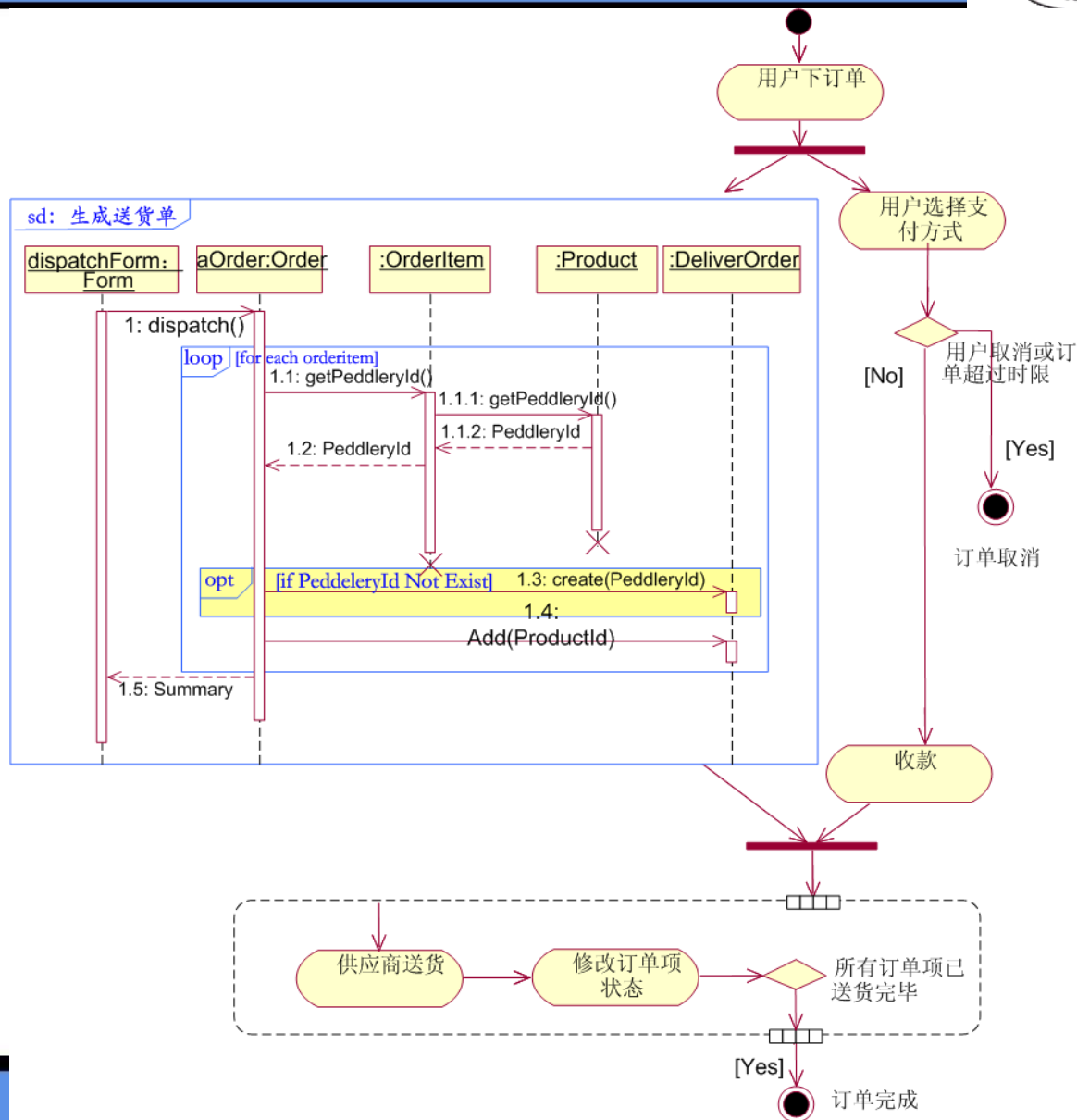
活动图



- 交互概述图 (Interaction overview diagram)
 - 活动图+顺序图
 - 活动图的变体：用顺序图细化活动节点
 - 顺序图的变体：用活动图补充顺序图
 - 作用
 - 建模交互之间的高级控制流
 - 以可视化的方式表达分支、并发和迭代
 - 在草图中更加适用

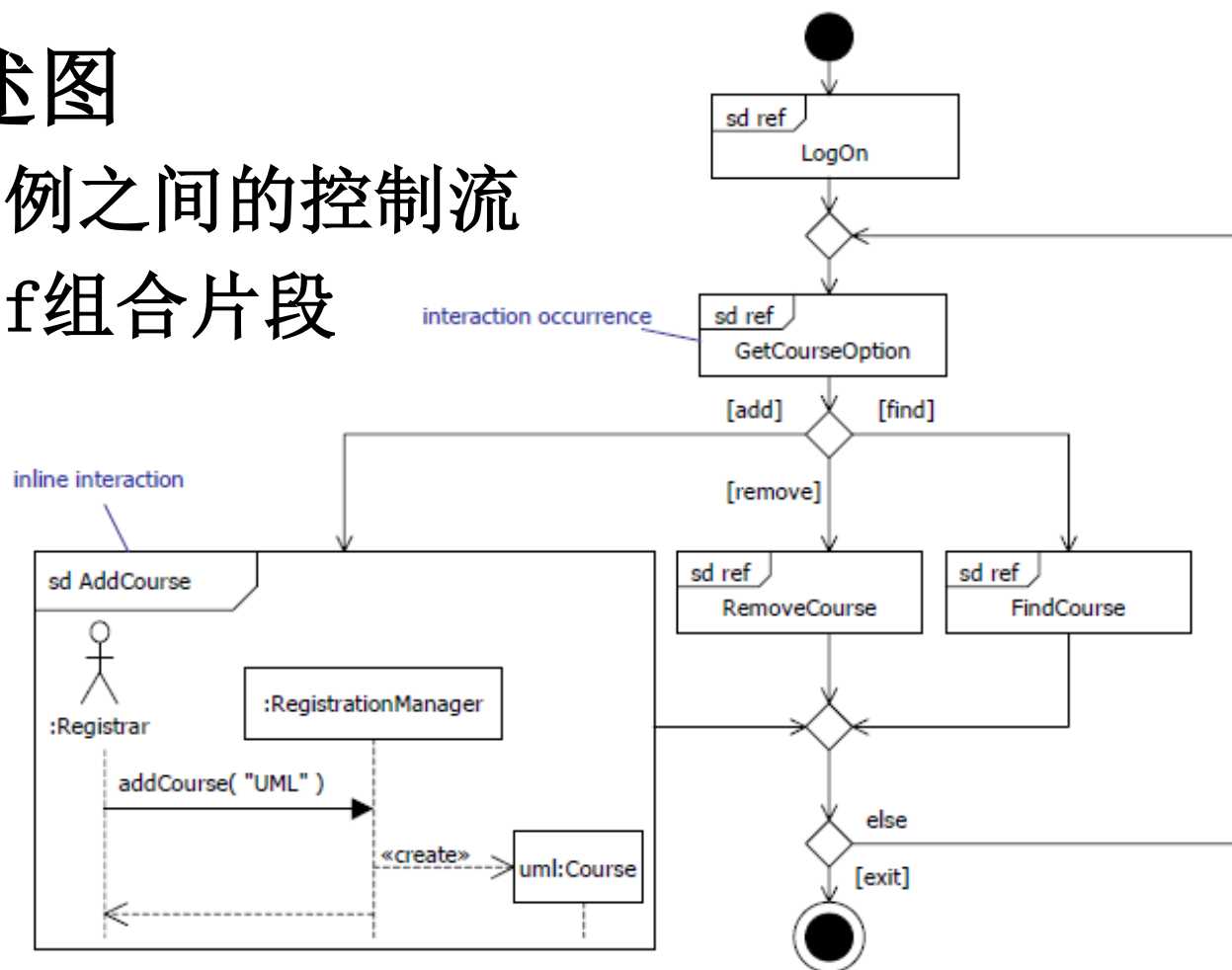
活动图

- 阅读交互概述图
 - 理解活动控制流
 - 理解活动节点



活动图

- 交互概述图
 - 表达用例之间的控制流
 - 使用ref组合片段



状态图

- 对象的状态
 - 表达对象可能存在的一种状况
 - 属性+属性值
 - 值随时间变化



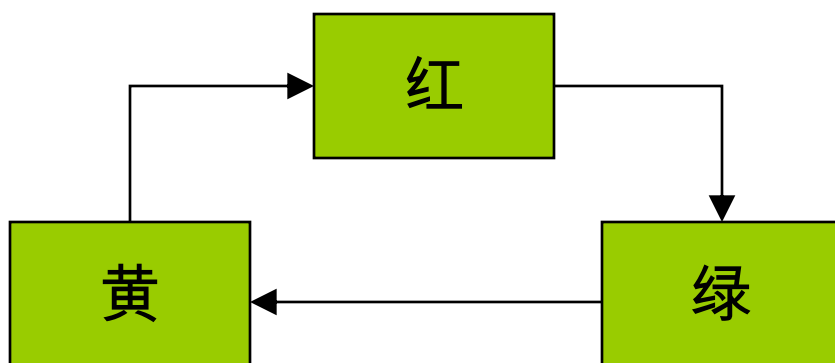
Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes



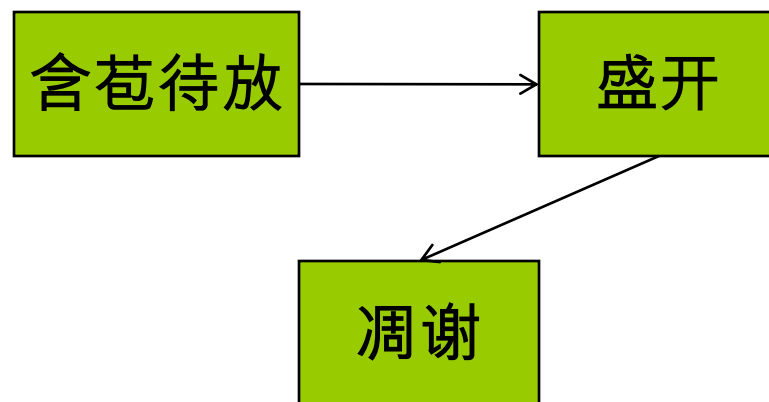
Professor Clark

状态图

- 状态的迁移
 - 状态随时间或事件的刺激发生变化



交通信号灯的状态迁移



花朵的状态迁移

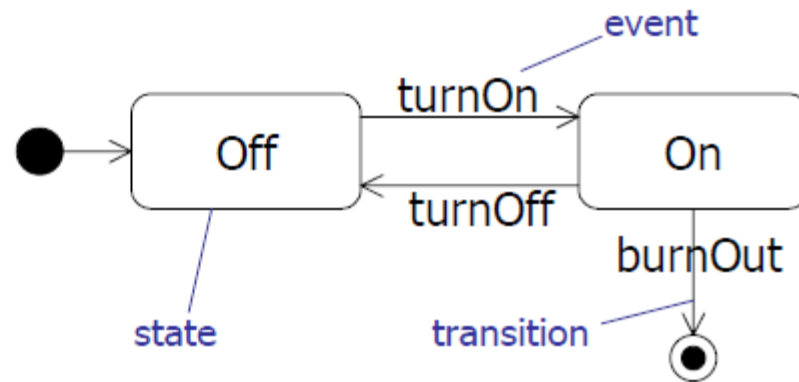
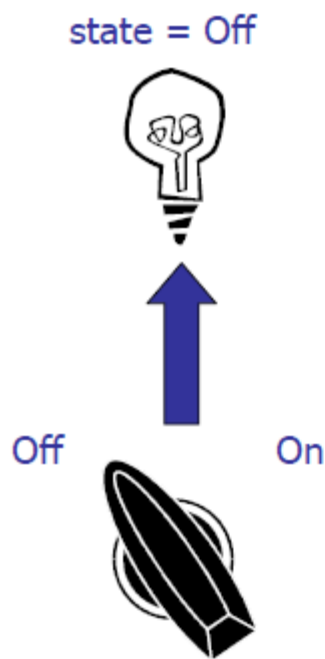
状态图

- 状态机

- 说明对象在生命周期中响应事件所经历的状态序列及对事件的响应

- 组成

- 状态
 - 迁移
 - 事件



- 状态

- 对象在生命期中所处的一种状况

- 状态的组成部分

- 名称：名词（短语）

state name

- 入口/出口动作

entry and exit
actions

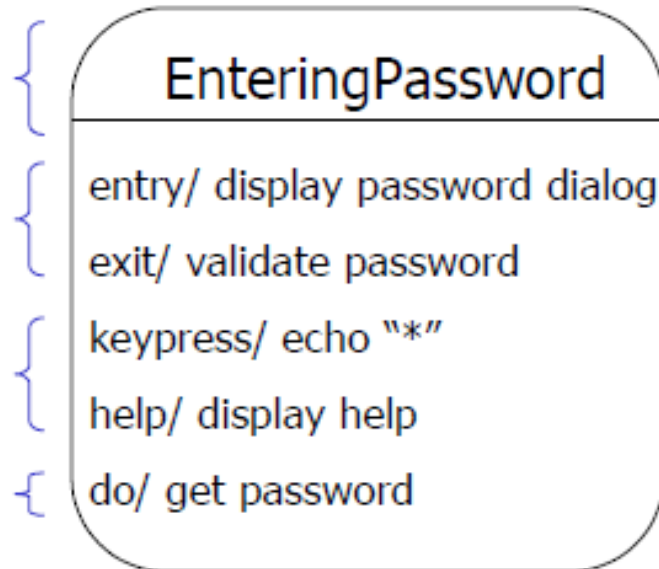
- 内部迁移

internal
transitions

- 内部活动

- 延迟事件

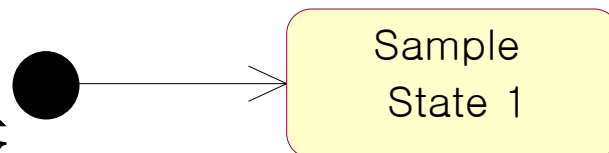
internal
activity



- 状态

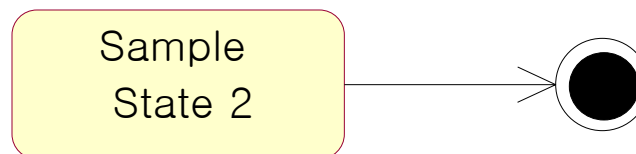
- 初态

- 状态机执行的开始
 - 一个状态机只有一个初态
 - 嵌套状态中可有自己的初态



- 终态

- 最后的终止状态
 - 数目不确定
 - 无终态：反应式/嵌入式系统



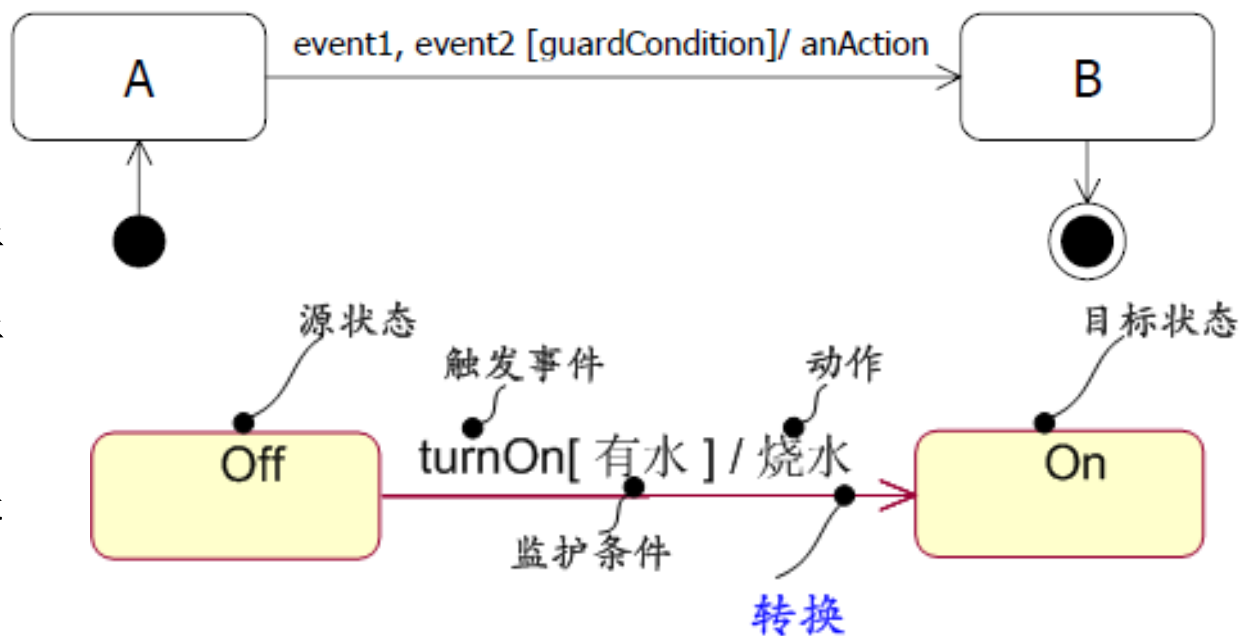
状态图

- 迁移

- 对象在某个特定事件发生且特定条件满足时从第一个状态进入第二个状态

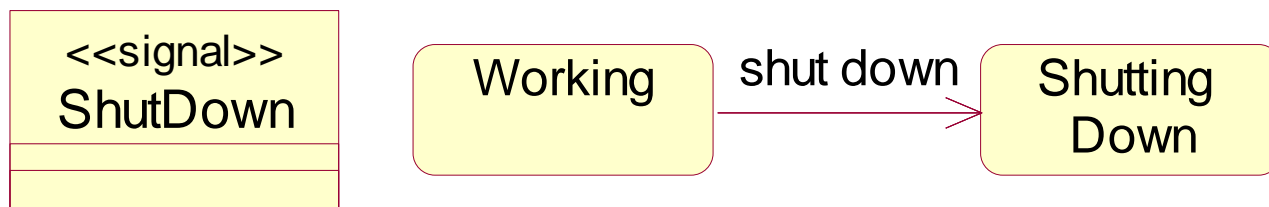
- 组成

- 源状态
 - 触发事件
 - 监护条件
 - 动作
 - 目标状态



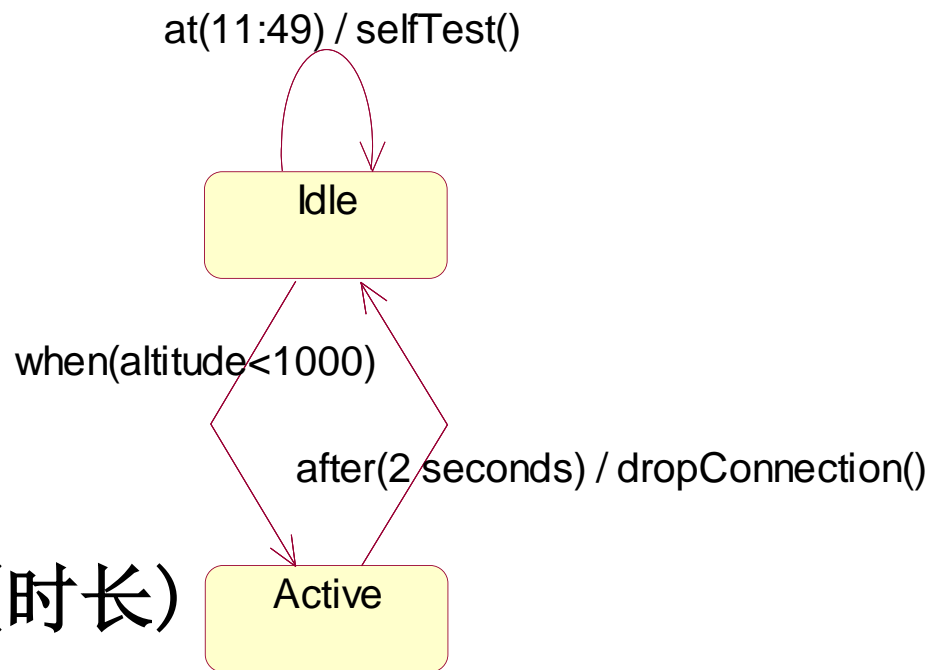
- 事件
 - 具有时间和空间位置的、有意义的事情的规格说明
 - 在状态机中触发状态迁移
 - 种类
 - 信号事件
 - 调用事件
 - 变化事件
 - 时间事件

- 信号 (signal) 事件
 - 表示对象接收到某个信号
 - 信号：
 - 对象间异步传递的信息包
 - 构造型<<signal>>化的类，属性中包含通信信息
 - 由状态机中的迁移发送；作为交互中的消息发送



- 调用事件
 - 表示对象接收到一个操作调用请求
 - 通常是同步调用
 - 引起
 - 触发状态机的一个转移
 - 调用目标对象的一个方法

- 变化事件
 - 满足某些条件时发生
 - when (布尔表达式)
- 时间事件
 - 时间相关的事件
 - at (时间点); after (时长)

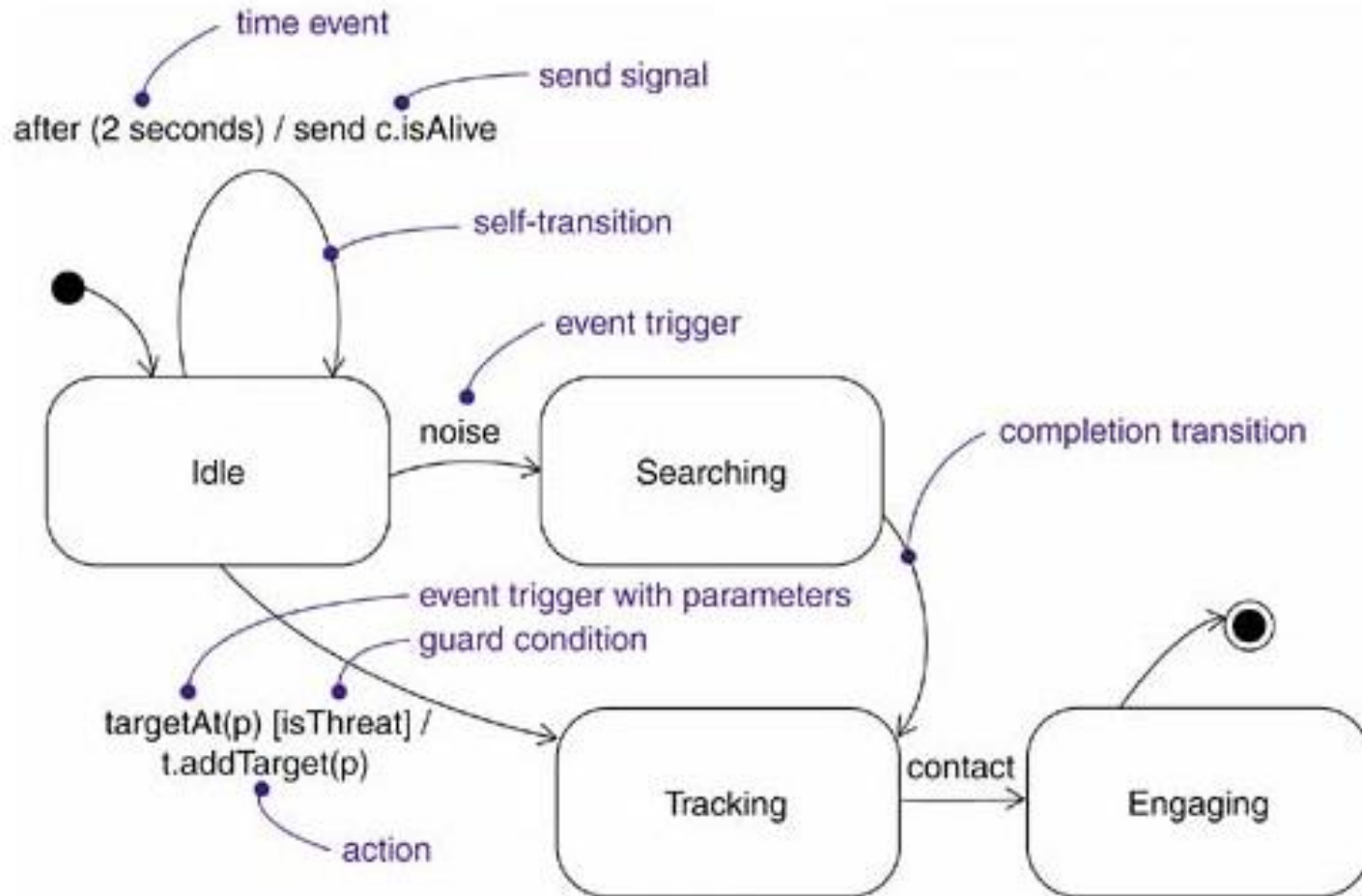


状态图

- 监护条件
 - [布尔表达式]
 - 触发事件发生后计算一次
 - 表达式为真则激活迁移
- 动作
 - 迁移激活时执行的行为
 - 不能被别的迁移中断



状态图



一个导弹制导系统的状态机

状态图



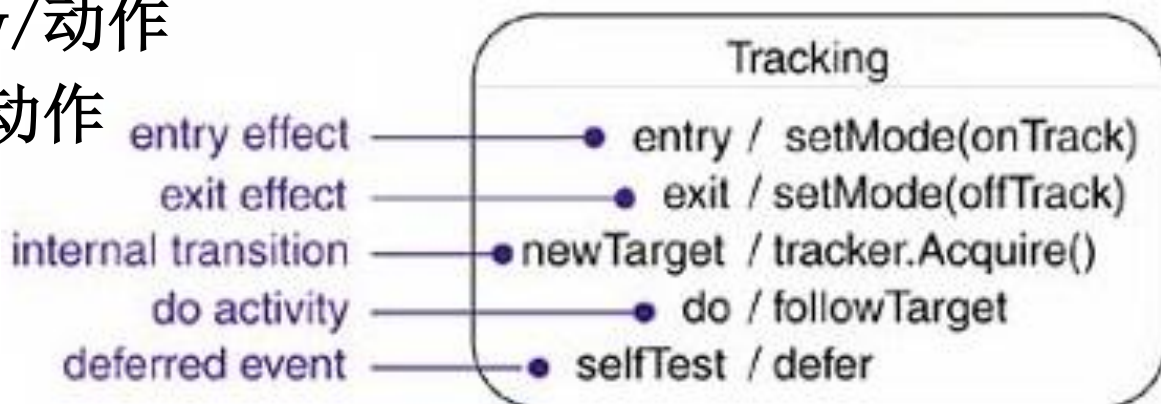
- 高级状态

- 入口/出口动作

- 入口: entry/动作
 - 出口: exit/动作
 - 原子的
 - 无监护条件

- 内部迁移

- 执行一个动作来响应事件，但状态不变
 - 转移串: 事件[监护条件]/动作



状态图

- **动作：**
- 一个动作是一个可执行的原子计算
- 动作是原子的，不可被中断的，其执行时间可忽略不计的。
- UML并没有规定描述action的语言格式，一般建模时采用实际的程序设计语言来描述。

状态图

- **动作:**
- UML规定了两种特殊的动作: entry action(进入动作)和exit action(退出动作)。
 - Entry动作: 进入状态时执行的活动, 格式如下:
`'entry' '/' action-expression`
 - Exit动作: 退出状态时执行的活动, 格式如下:
`'exit' '/' action-expression`
(其中 action-expression 可以用到对象本身的属性和输入事件的参数)

- Rose中状态的Action Specification 对话框。
 - On Entry
 - On Exit
 - Do
 - On Event

Action Specification for Run

Detail

When: On Entry

On Event: On Entry

Event: On Exit

Arguments: Do

Conditio: On Event

Type: Action

Name: Run

Send arguments:

Send target:

OK Cancel Apply Browse Help

状态图



- 迁移的类型

转换类型	描述		语法
外部迁移	对事件做出响应，引起状态迁移，同时引发一个特定动作，并引发入口/出口动作	不同状态间的迁移	事件[监护条件]/动作
		自身迁移	
内部迁移	对事件做出响应，并执行一个特定的动作，但并不引起状态变化		事件[监护条件]/动作

- 高级状态

- 内部活动

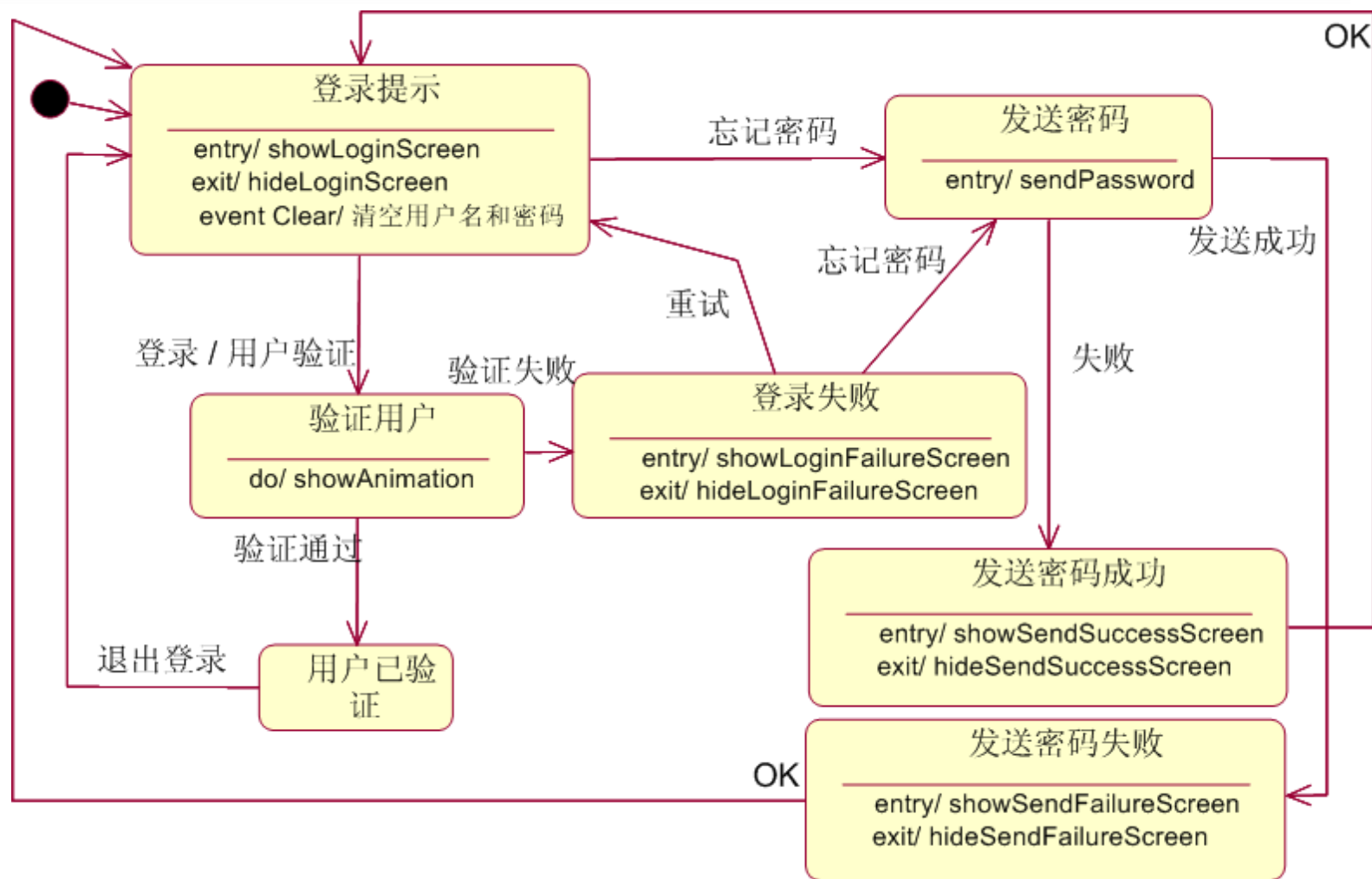
- 处于一个状态时持续进行的活动
 - 格式：do/活动名
 - 会被外部迁移中断



- 延迟事件

- 该事件不会触发状态的转换，当对象处于该状态时事件不会丢失，但会被延迟执行

状态图



包含高级状态的状态机

状态图



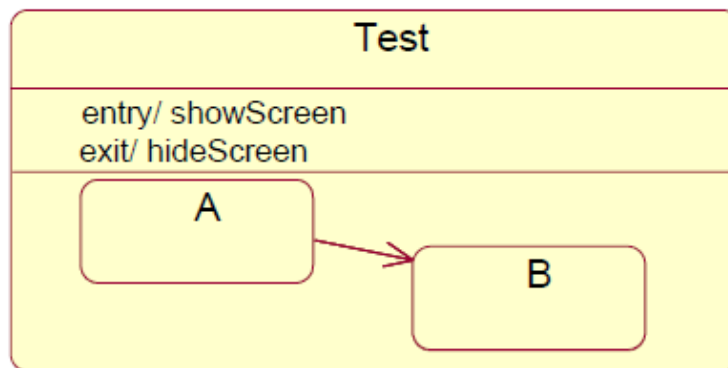
- 组合状态

- 子状态：嵌套在另一个状态中的状态
- 组合状态：含有子状态的状态

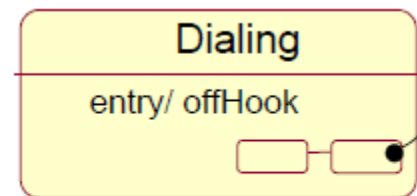
- 顺序
- 并发
- 历史

- 表示法

- 嵌套区域
- 分解指示符



嵌套区域表示法



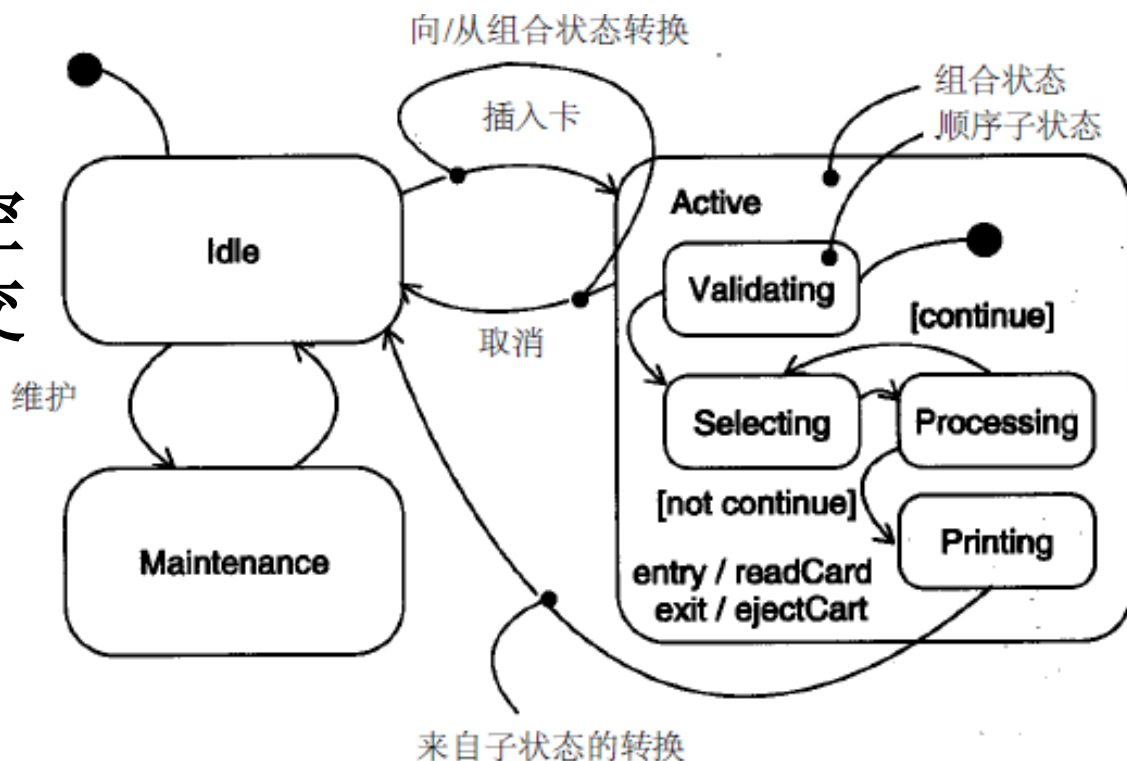
分解指示符

分解指示符法

状态图

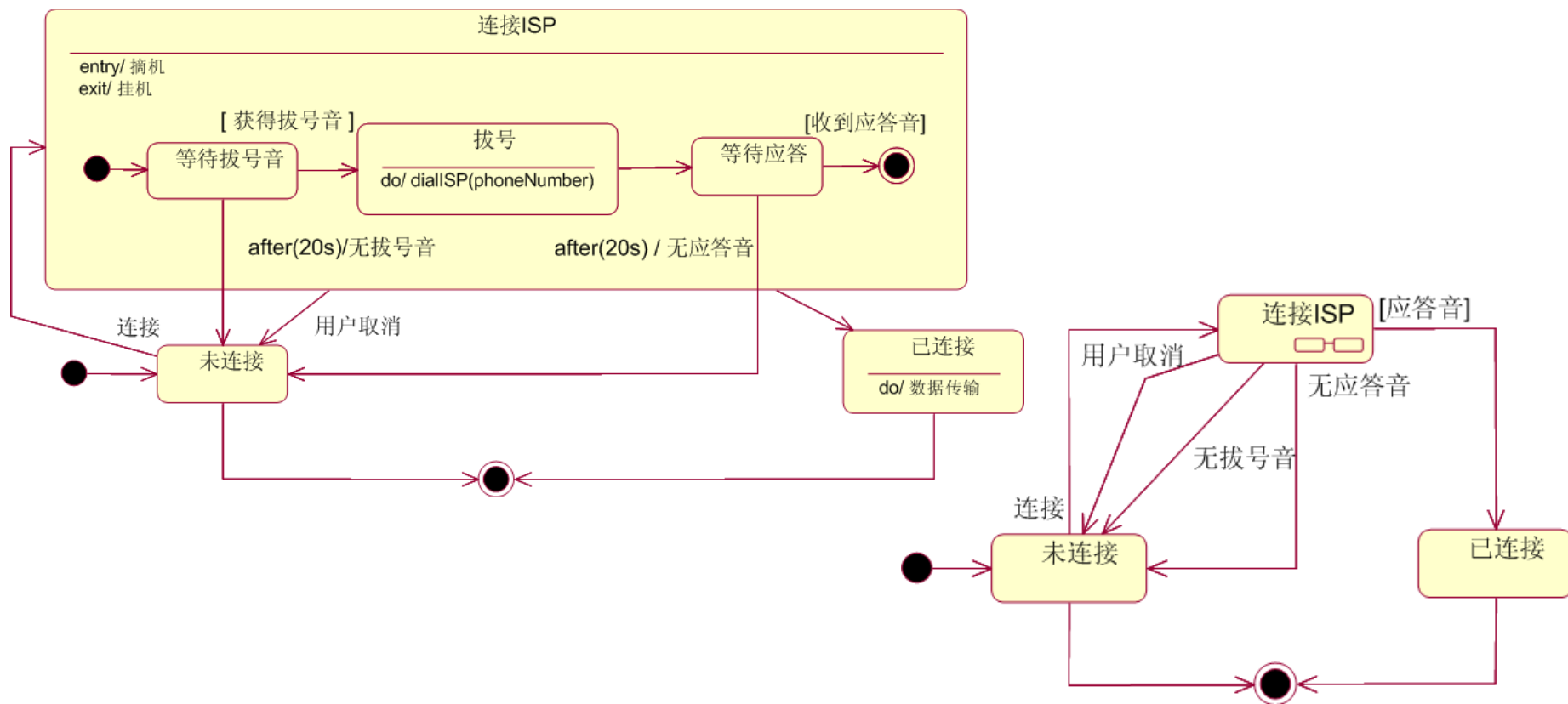
- 顺序组合状态

- 非正交子状态
- 将组合状态的空间分解为不相交的状态
- 作为
 - 源状态
 - 目标状态



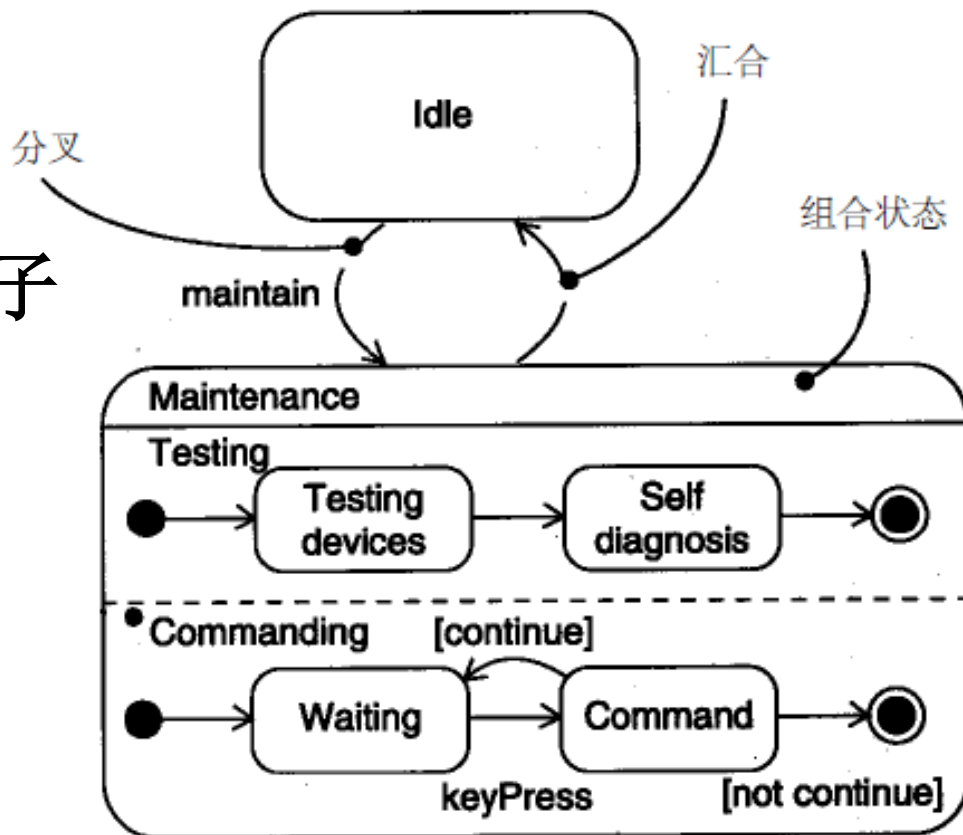
状态图

• 顺序组合状态



状态图

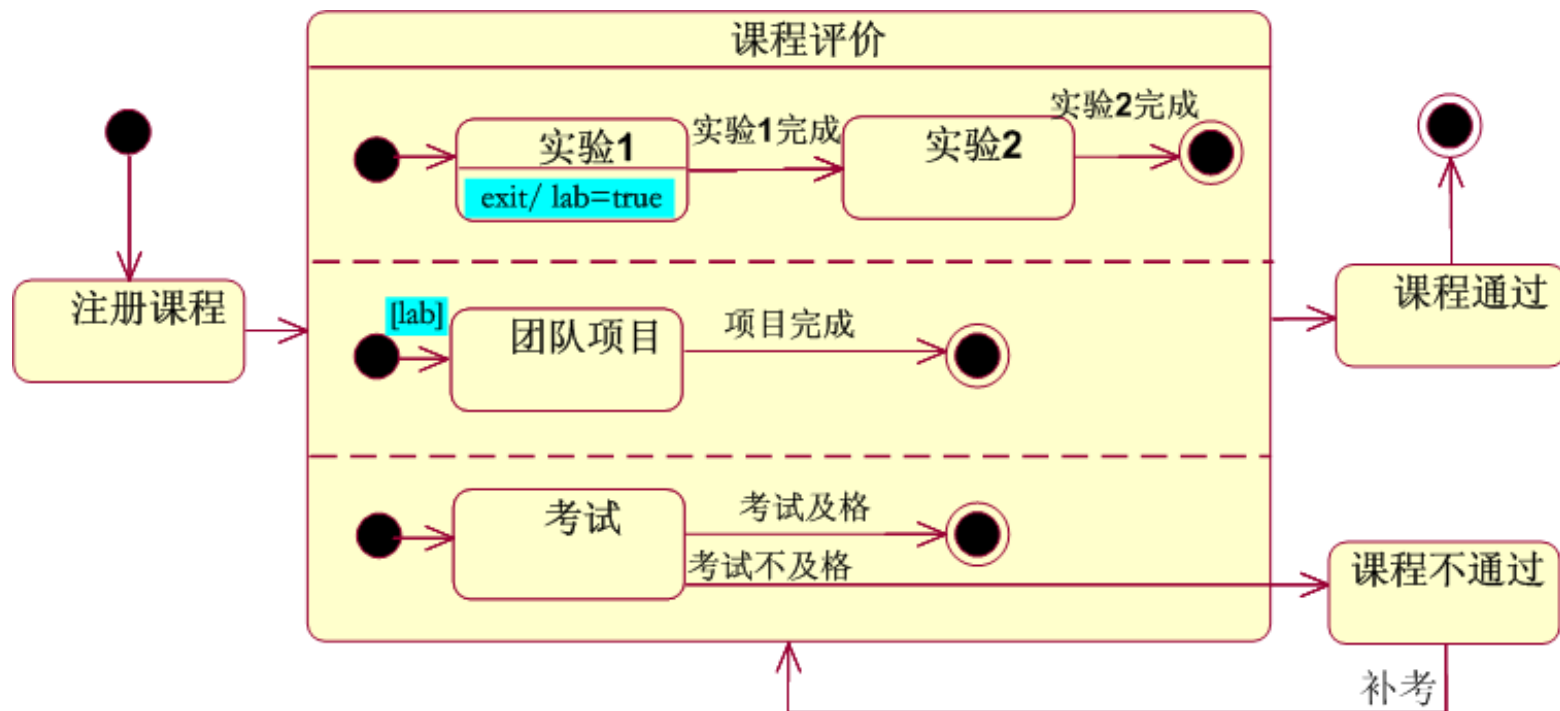
- 并发组合状态
 - 正交子状态
 - 并行执行的多个子状态机
 - 作为
 - 源状态
 - 目标状态



状态图



- 并发组合状态



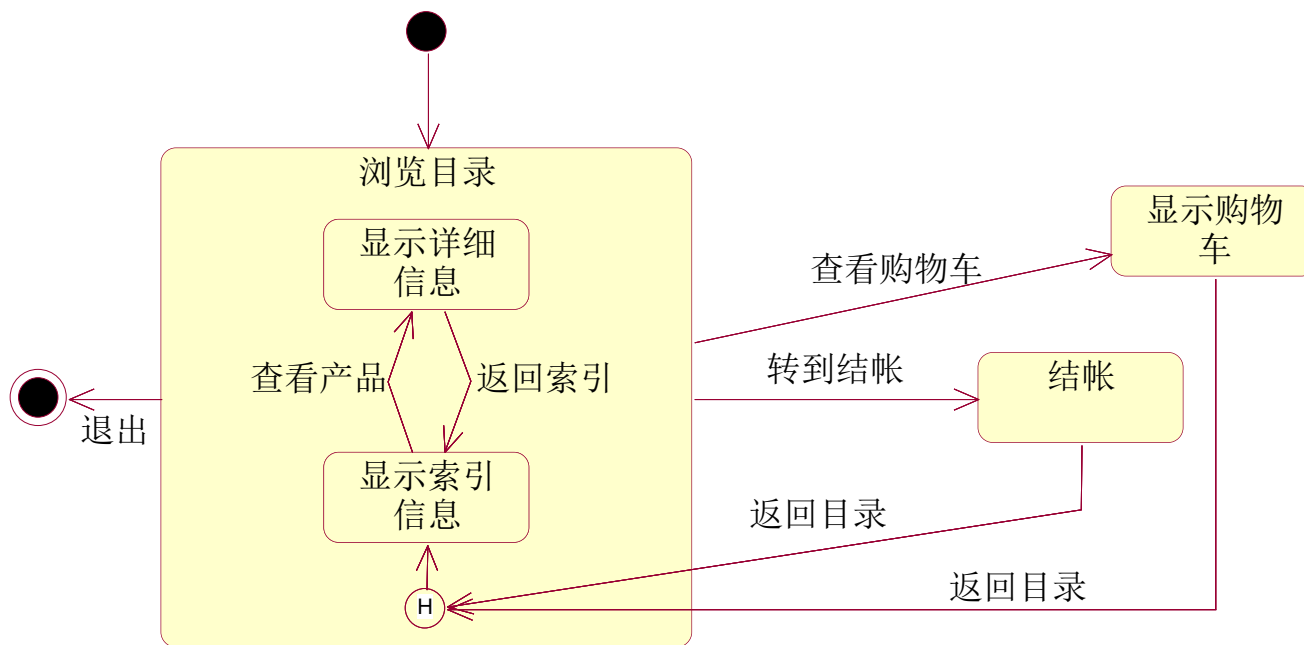
状态图

- **历史状态：**
- 历史状态是一个伪状态，其目的是记住从组合状态中退出时所处的子状态。当再次进入组合状态时，可直接进入到这个子状态，而不是再次从组合状态的初态开始。
- **H和H*的区别：**
 - H只记住最外层的组合状态的历史。
 - H*可记住任何深度的组合状态的历史。

状态图

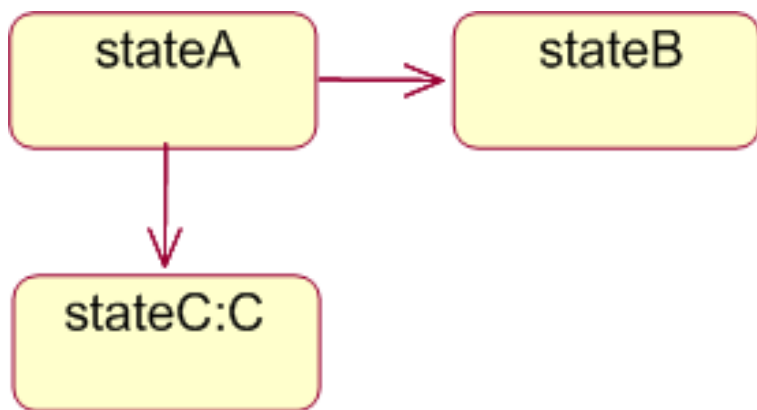
- 历史组合状态

- 用于存储退出组合状态时所处的子状态
- 返回组合状态时可回到相应子状态

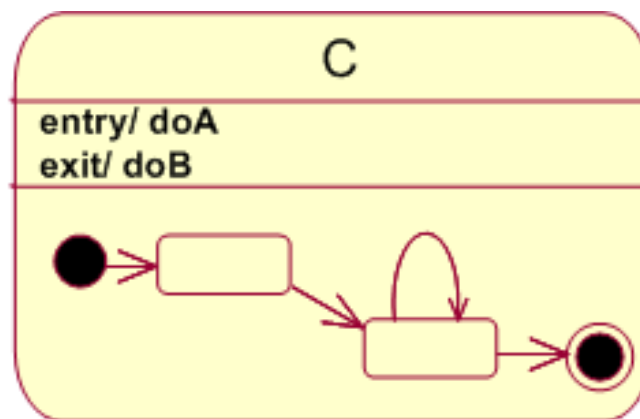


状态图

- 子状态机
 - 将子状态机单独定义，并对其进行命名
 - 在需要使用的方式来引用它



状态机图

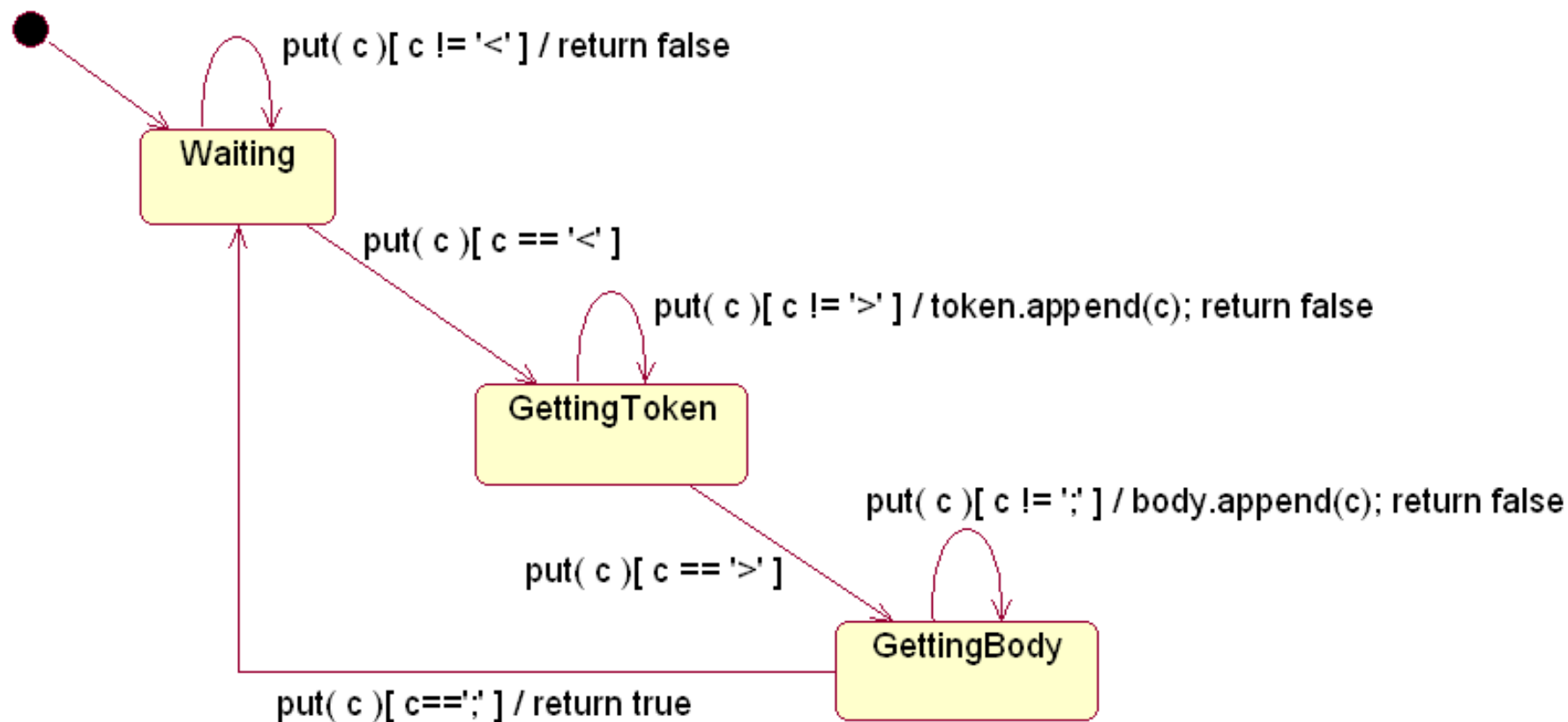


子状态机

- 状态图
 - 强调从状态到状态的控制流的状态机的简单表示
 - 基于David Harel发明的状态图表示法
 - 状态机中元素的投影
 - 用于
 - 表示一个业务领域的知识
 - 描述设计阶段对象的状态变迁

状态图的工具支持

- 正向工程：根据状态图生成代码。例：词法分析



```
class MessageParser {
public boolean put(char c) {
    switch (state) {
        case Waiting:
            if (c == '<') {
                state = GettingToken;
                token = new StringBuffer();
                body = new StringBuffer();
            }
            break;
        case GettingToken :
            if (c == '>')
                state = GettingBody;
            else
                token.append(c);
            break;
        case GettingBody :
            if (c == ';') {
                state = Waiting;
                return true;
            }
    }
}
```

```
        else
            body.append(c);
    }
    return false;
}

public StringBuffer getToken() {
    return token;
}

public StringBuffer getBody() {
    return body;
}

private final static int Waiting = 0;
private final static int GettingToken = 1;
private final static int GettingBody = 2;
private int state = Waiting;
private StringBuffer token, body;
}
```

- 状态图的应用

- 对对象生命周期建模

- 主要描述对象能够响应的事件、对这些事件的反应，以及过去对当前行为的影响

- 对反应型系统建模

- 这个对象可能处于的稳定状态、从一个状态到另一个状态之间的转换所需的触发事件，以及每个状态改变时发生的动作

- 状态图

- VS. 交互图

- 交互图显示特定交互中多个对象的行为
 - 状态图显示一个对象所有的动态行为

- VS. 活动图

- 活动图以活动到活动的控制流为焦点
 - 状态图表达单个对象的状态变化控制流

状态图



- 绘制状态图的步骤
 - 寻找主要的状态
 - 确定状态之间的迁移
 - 细化状态内的活动与迁移
 - 用复合状态来展开细节

状态图



- 绘制状态图举例

- 航班订票系统状态图

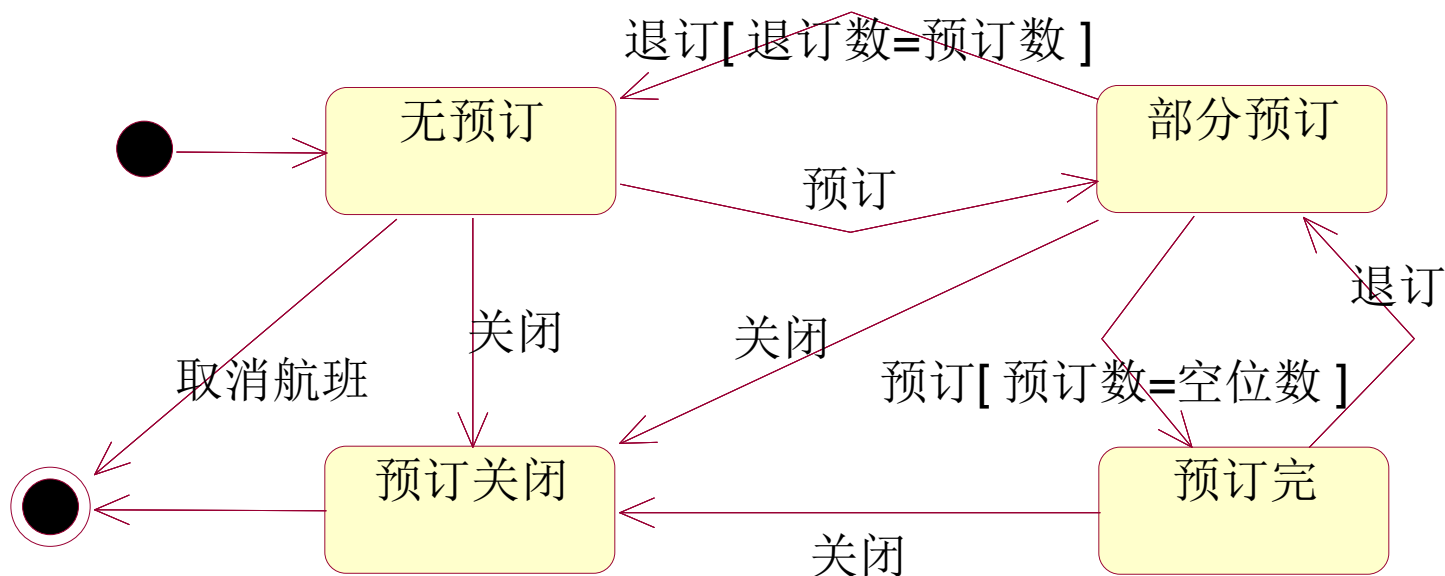
- 状态：无预订、部分预订、预订完、预订关闭
 - 外部迁移

源\目标	无预订	部分预订	预订完	预订关闭	结束
开始状态	无条件				
无预订		预订		关闭	取消航班
部分预订	退订[退订数= 已预订数]		预订[预订数= 空位数]	关闭	
预订完		退订		关闭	
预订关闭					无条件

状态图



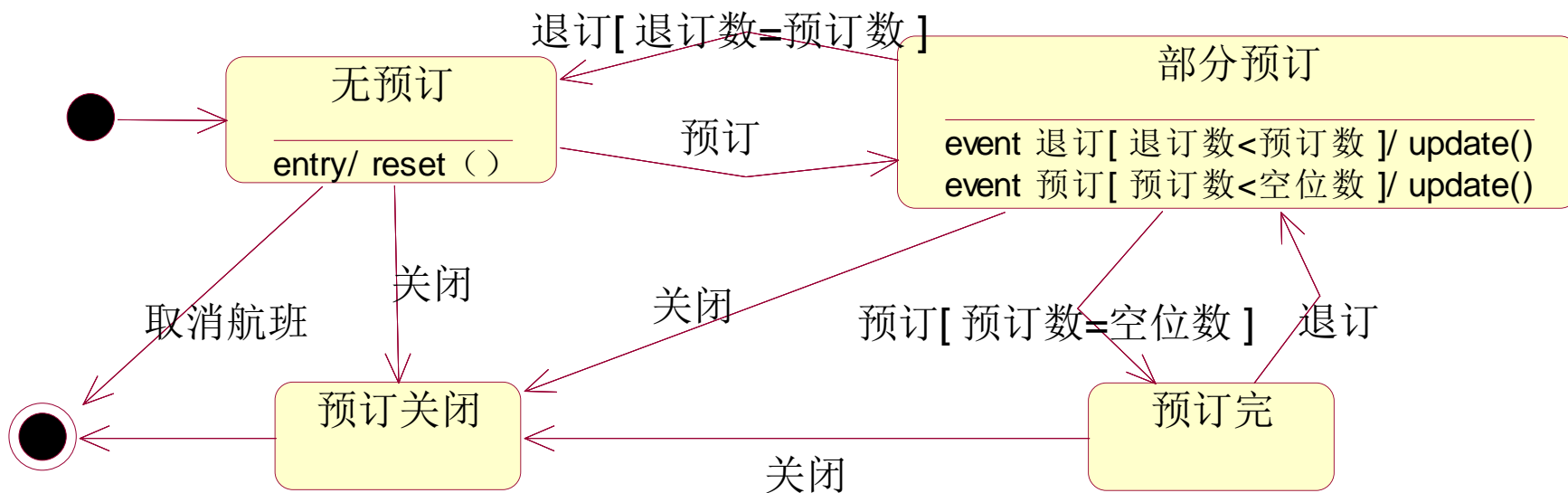
- 绘制状态图举例
— 航班订票系统的状态图



状态图



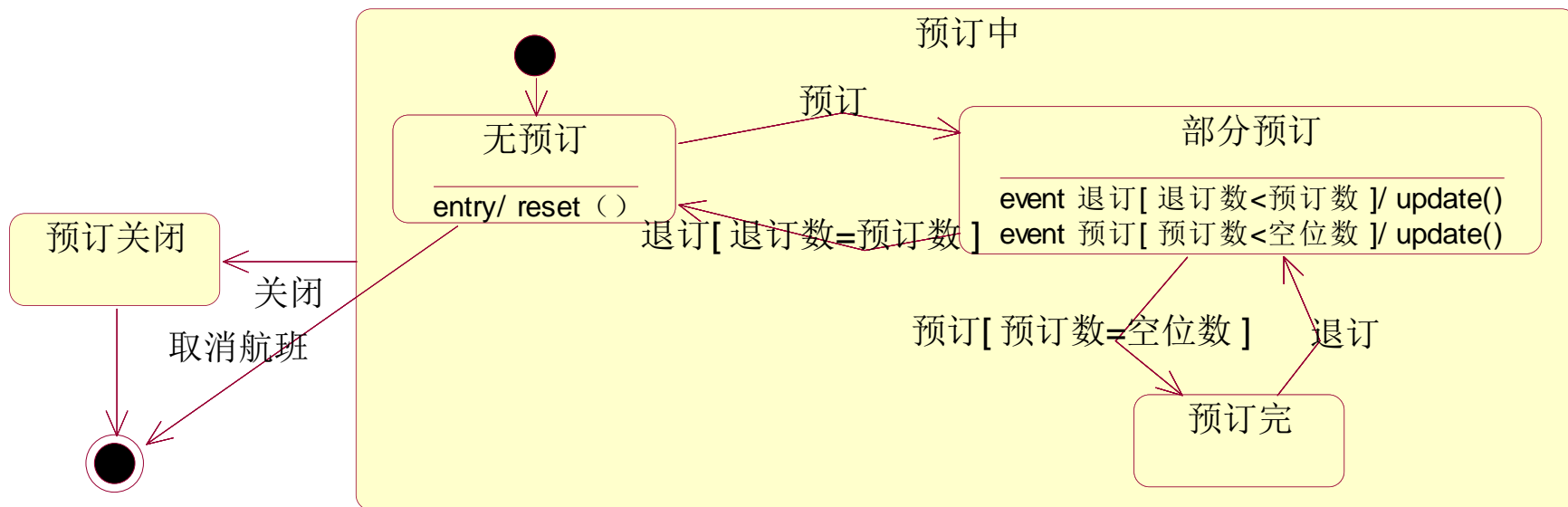
- 绘制状态图举例
— 细化状态内活动与迁移



状态图



- 绘制状态图举例
— 使用组合状态



小结



- 重点
 - 阅读交互图
 - 绘制基本的顺序图
 - 阅读活动图和状态图
 - 绘制基本的活动图和状态图
 - 理解几种图的不同视角和特点