

# 第12讲 详细设计之UML建模

网络空间安全学院

芦效峰

# 引言 UML交互图

**交互图**表示类（对象）如何交互来实现系统行为。交互图具有如下两种形式。

## 1) **顺序图**（时序图）

它描述对象按时间顺序的消息交换过程，它体现出系统用例的行为。

## 2) **协作图**

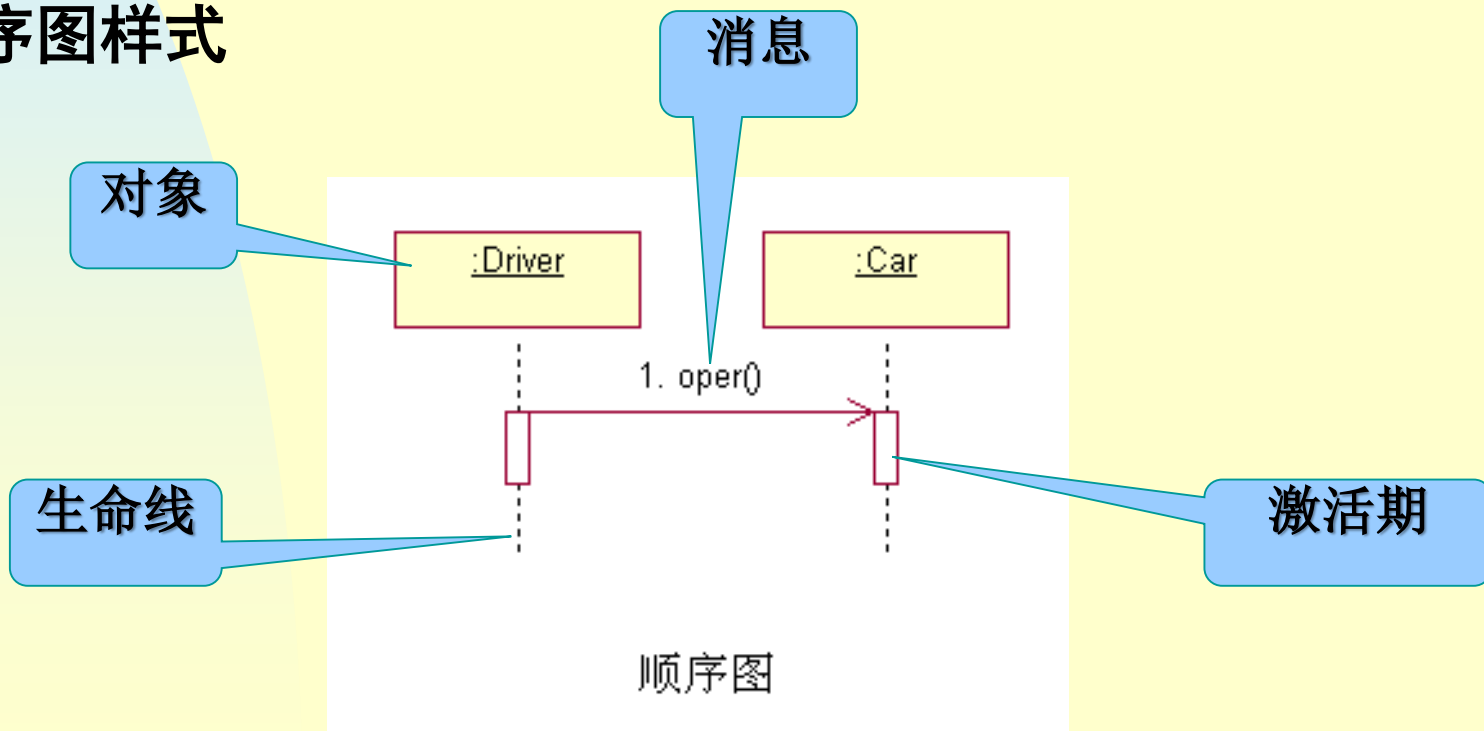
它描述对象间的组织协作关系，它也可体现出系统用例的行为。

# 顺序图建模

## 一、顺序图的概念

顺序图是两种类型的交互图之一。顺序图用来建模以时间顺序安排的对象交互，并且把用例行为分配给类(对象)。它是用来显示参与者如何采用若干顺序步骤与系统对象交互的模型。

### ■ 顺序图样式

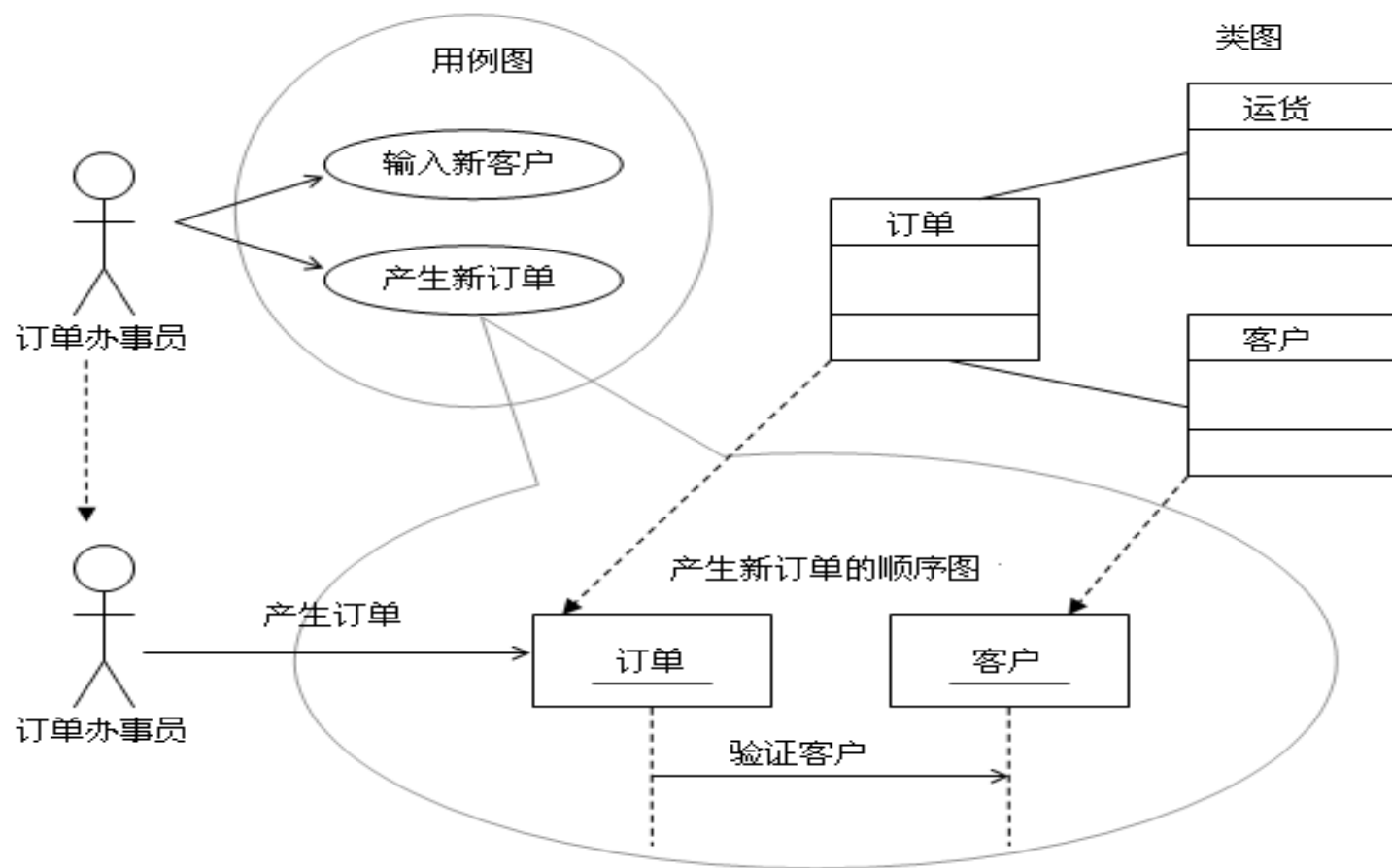


## 二、为什么要建模顺序图

建模顺序图有许多理由，顺序图与活动图具有类似的作用。其中重要的理由就是实现用例。任何用例都可以使用顺序图进一步阐明和实现。

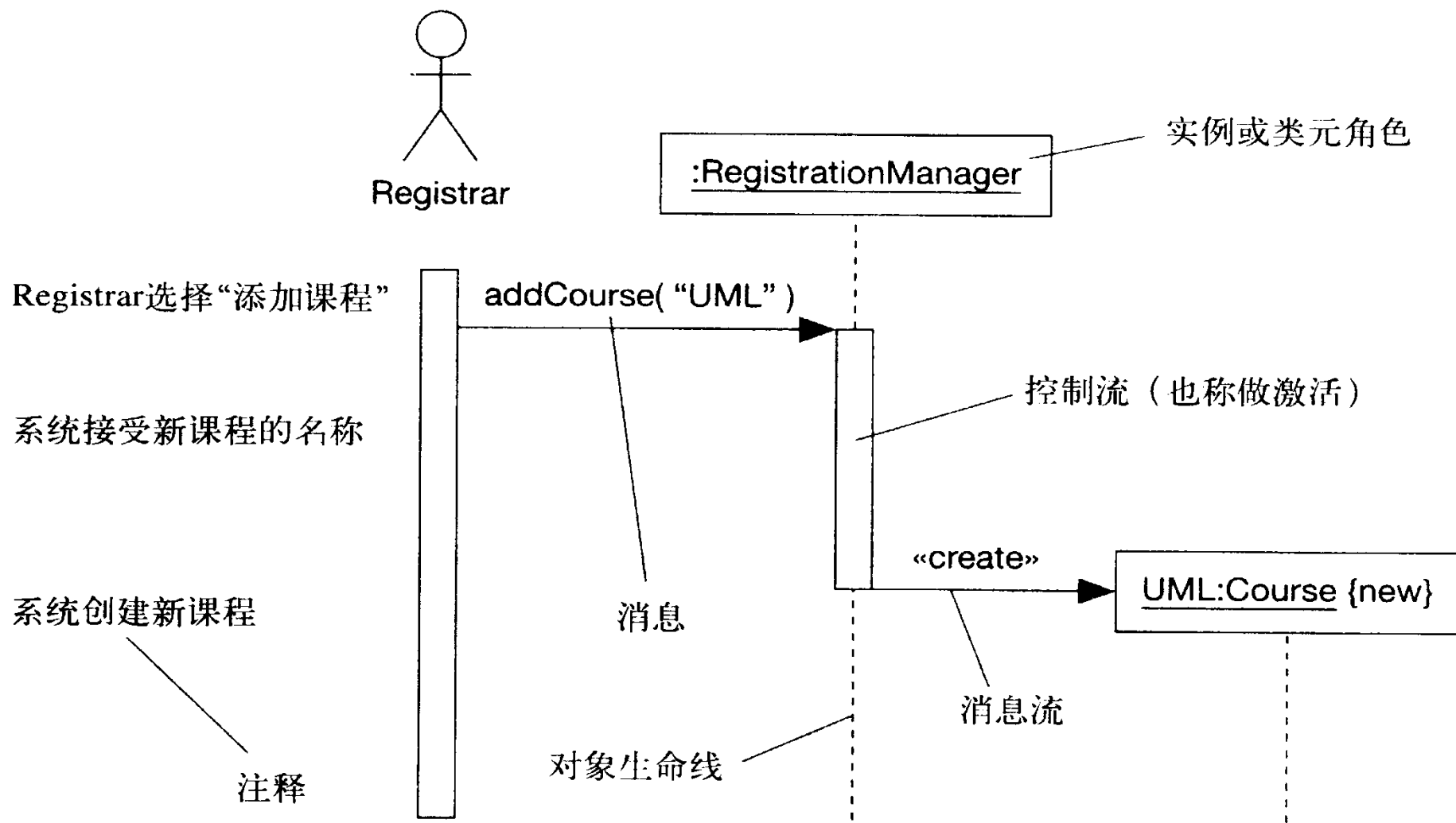
顺序图刻画了用例具体实现的流程，比活动图更能够表示细节，因此适用于详细设计。

## 顺序图与用例图和类图的关系



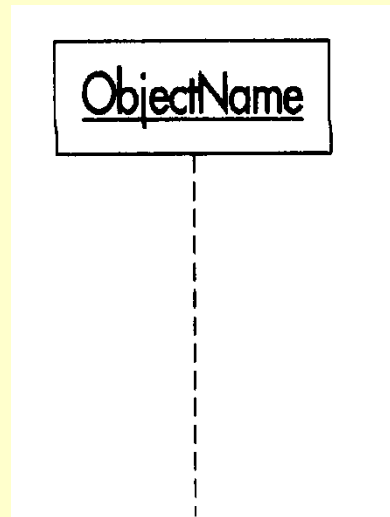
### 三、顺序图的标记符

顺序图有两个主要的标记符：**活动对象**和这些活动对象之间的**通信消息**。活动对象可以是任何在系统中扮演角色的对象，不管它是**对象实例**还是**参与者**，如下图所示。

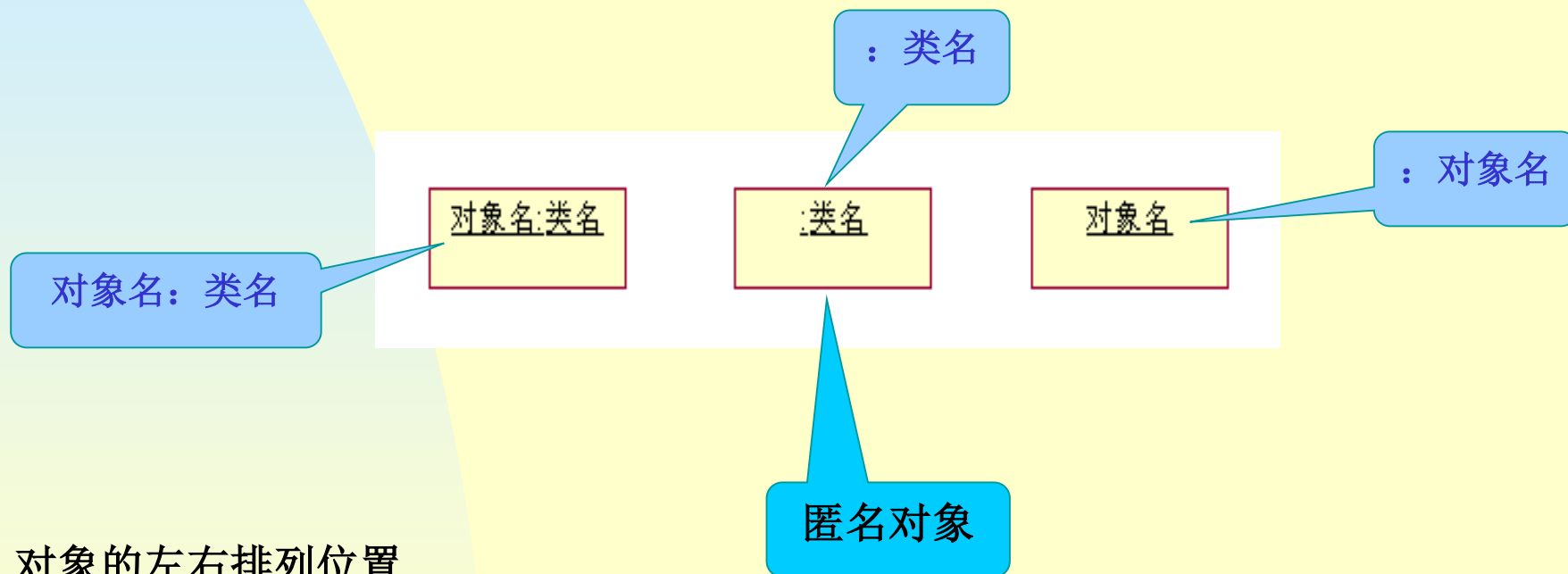


## 1. 活动对象

活动对象可以是系统的参与者或者任何有效的系统对象。对象是类的实例，它使用包围名称的矩形框来标记。名称带下划线，顺序图中对象的标记符如下图所示。



## (1) 对象的命名



对象的左右排列位置

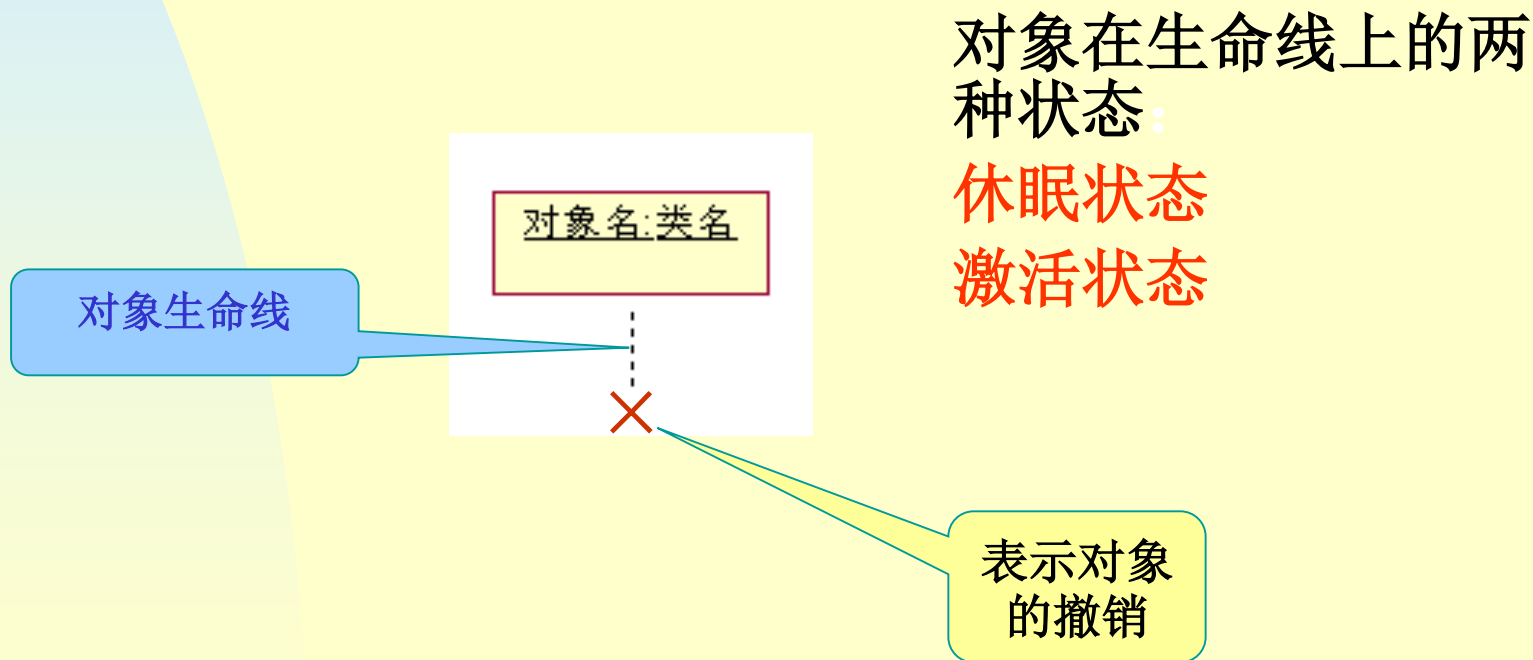
对象的左右顺序并不重要，但是为了图面的清晰整洁起见，通常应遵循以下两个原则：

- (1) 把交互频繁的对象尽可能地靠拢
- (2) 把初始化整个交互活动的对象（有时是一个参与者）放置在最左边



## ② 生命线

- 表示对象存在的时间，对象下面一条虚线表示。生命线从对象创建开始到对象销毁时终止。



### ③ 控制焦点 / 激活期

- 小矩形，表示这个时间对象将执行操作。

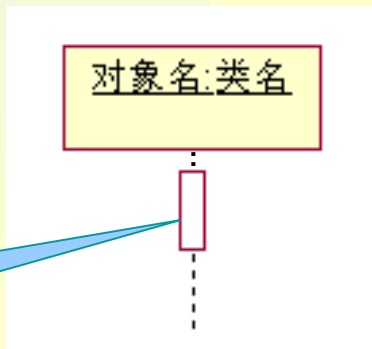
当一个对象没有被激活期时，该对象处于休眠状态，什么事都不做，但它仍然存在，等待新的消息来激活它。当一条消息被传递给对象的时候，它会触发该对象的某个行为，这就是说该对象被激活了。

当一个对象处于激活期时，表明该对象正在执行某个动作。

矩形框的高度表示对象执行一个操作所经历的时间段，矩形的顶部表示动作的开始，底部表示动作的结束。

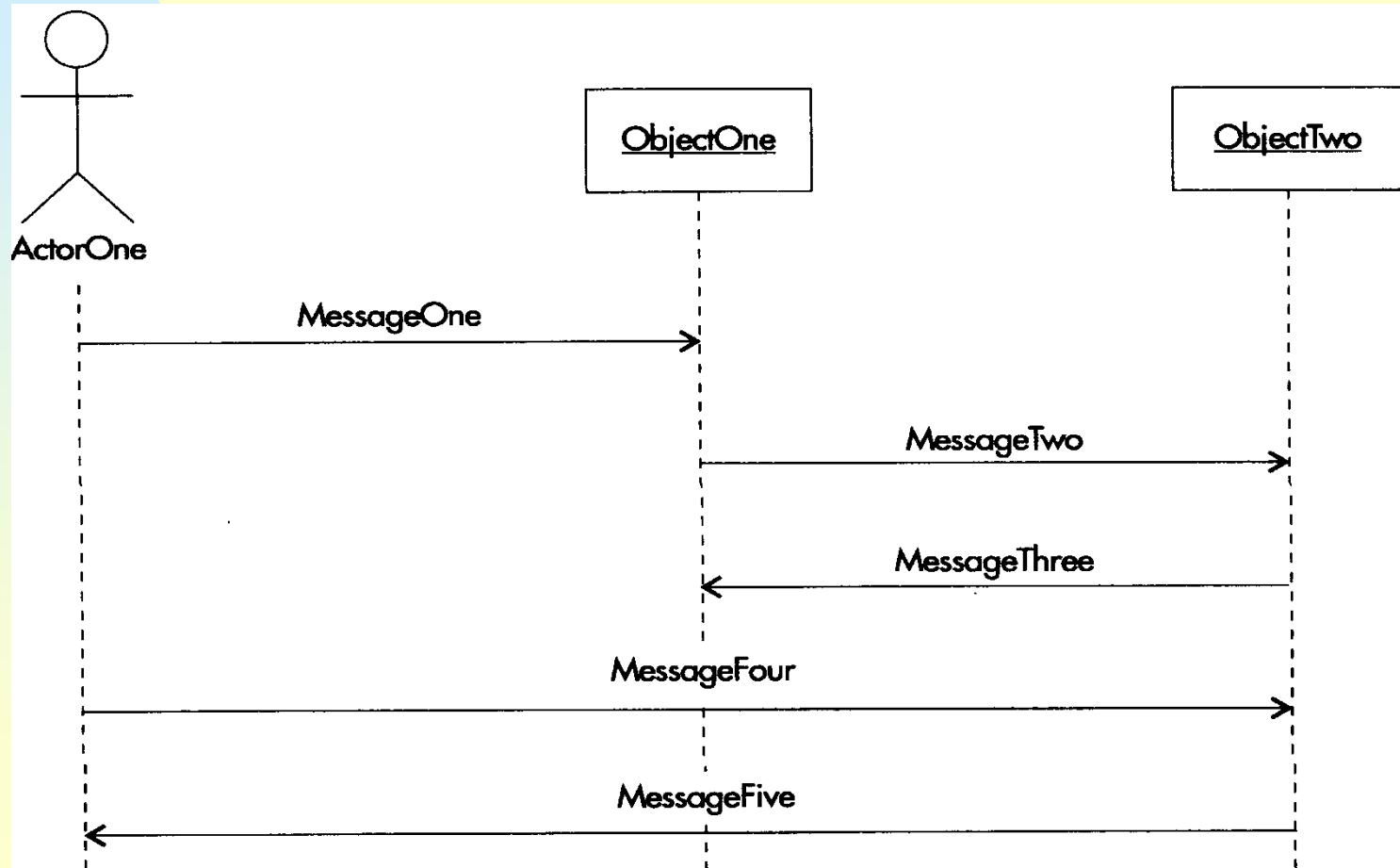
对象接收消息后可以由自己的某个操作来完成，也可以通过其他对象的操作来完成。

激活期



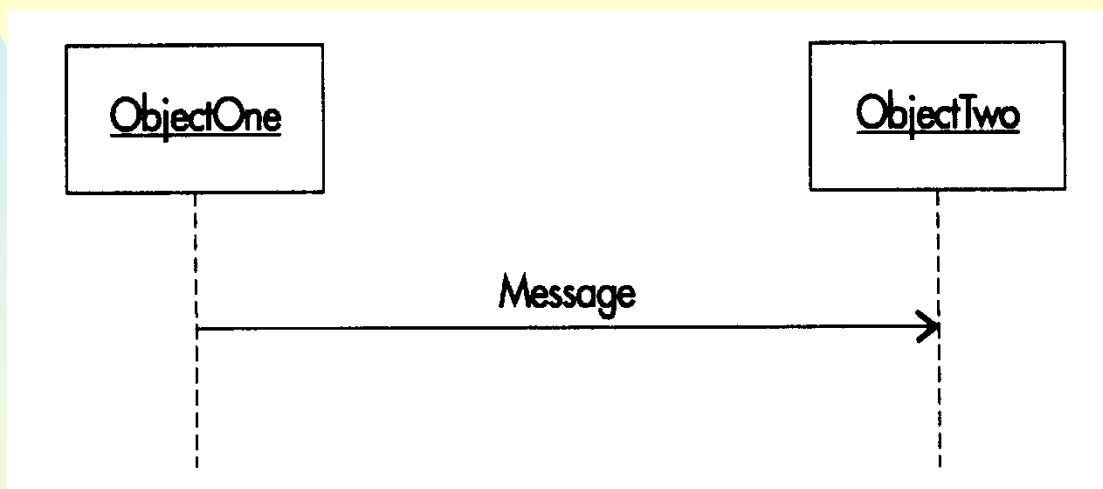
把**参与者表示为活动对象**的建模可以说明参与者如何与系统交互，以及系统如何与用户交互。参与者可以调用对象，对象也可以通知参与者，如下图所示。

可以把消息发送给不是其直接相邻的参与者或者对象。



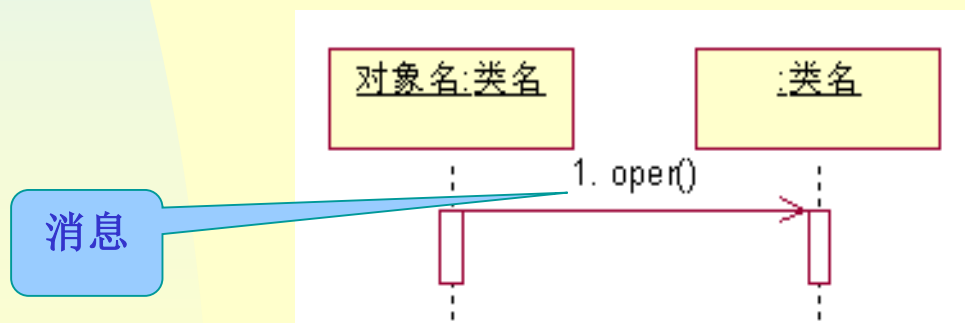
## 2. 消息

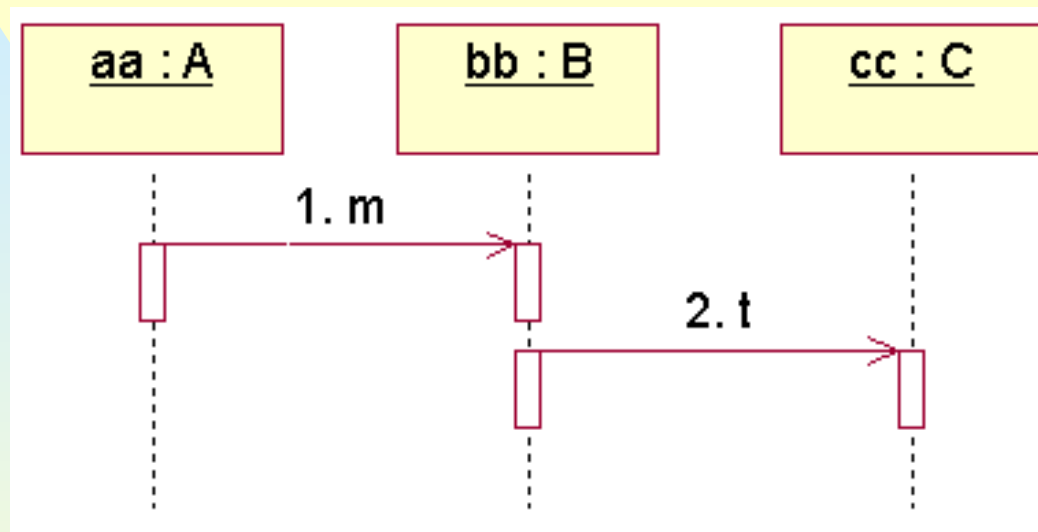
消息用来说明顺序图中不同活动对象之间的通信。消息从活动对象生命线到接收对象生命线的箭头表示。箭头上标记要发送的消息，如下图所示。



活动对象之间发送的消息是顺序图的关键。消息说明了对象之间的控制流，对象是如何交互的。

- 带箭头的连线，表示对象之间传输的信息。
  - 对象之间的交互是通过互发消息来实现的。一个对象可以请求（要求）另一个对象做某件事件。
  - 消息从源对象指向目标对象。消息一旦发送便将控制从源对象转移到目标对象。



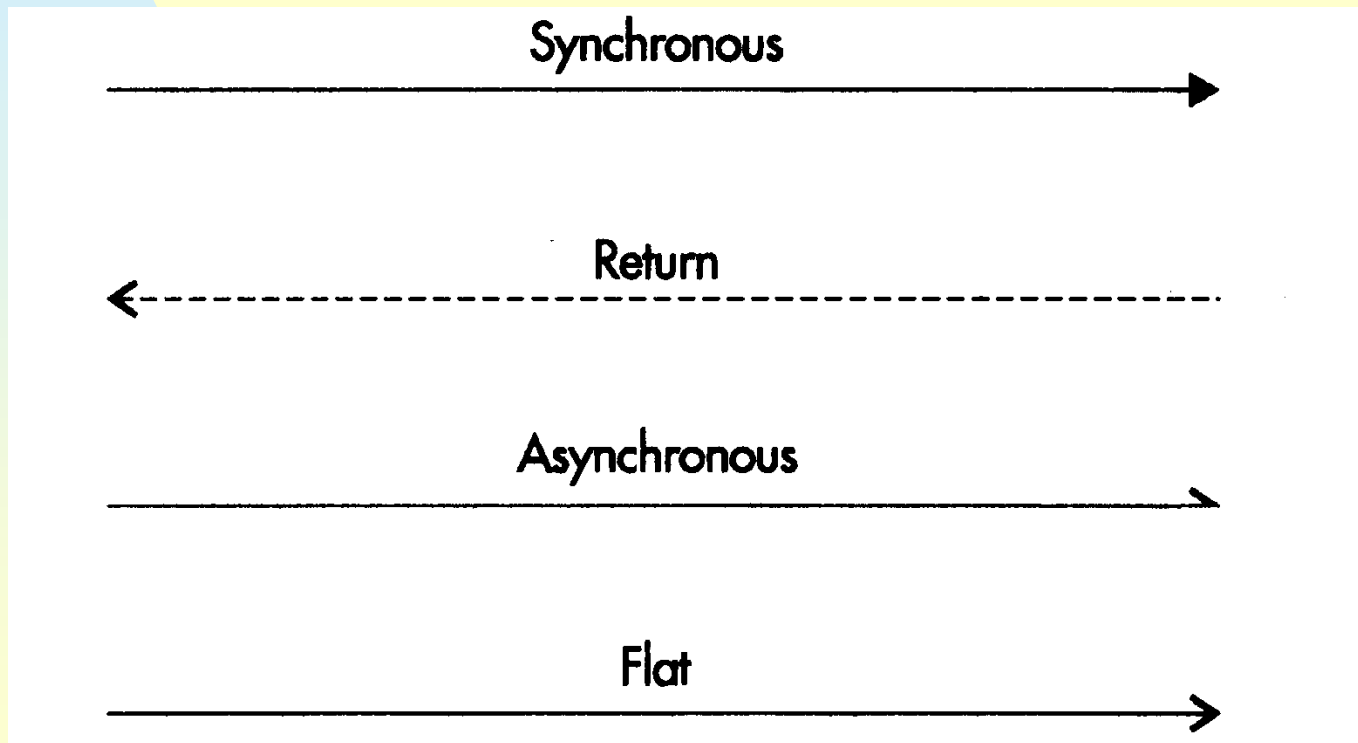


时序图中，消息的阅读顺序是严格自上而下的

# 消息的类型：

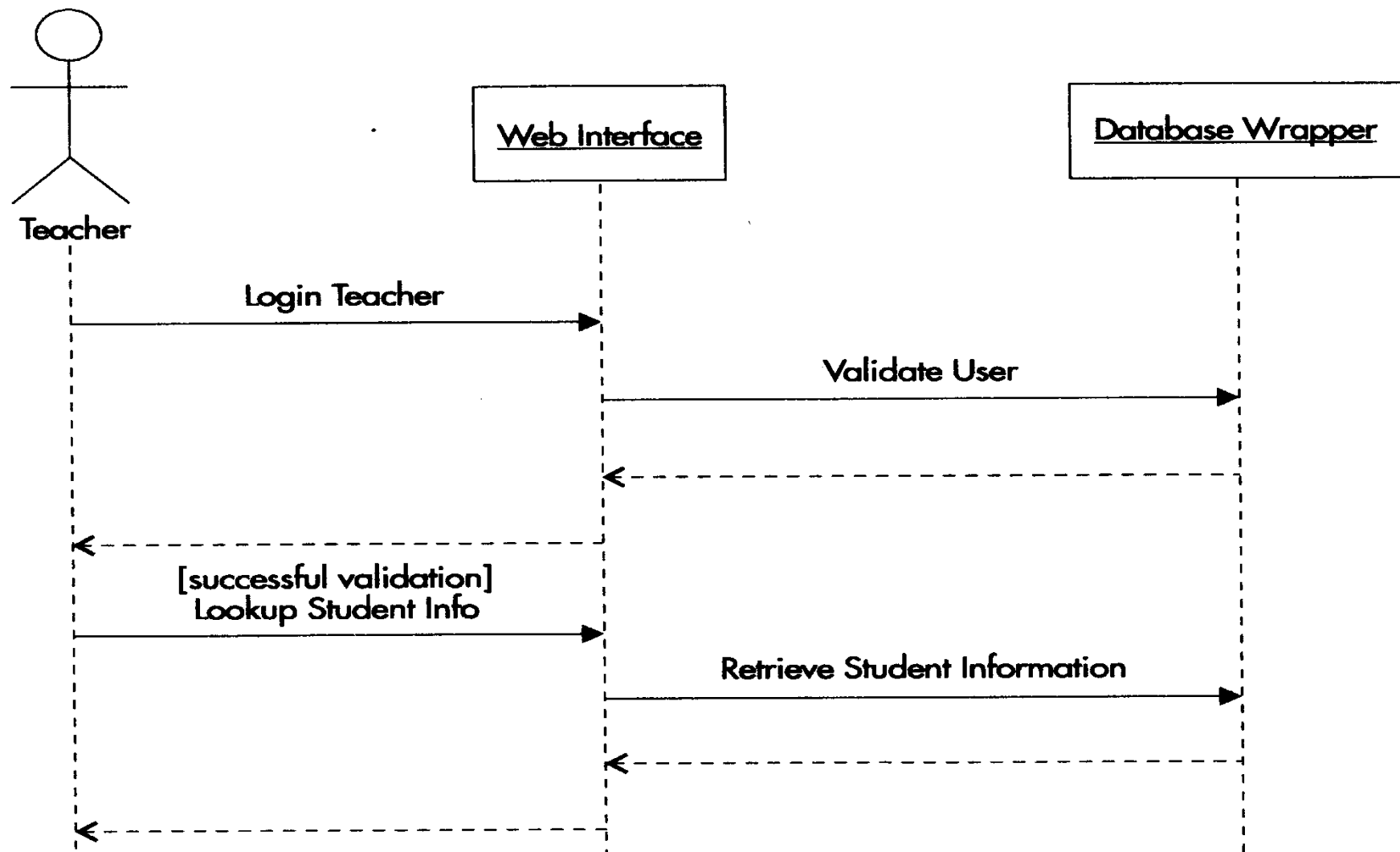
在UML中，总共有4种类型的消息，如下图所示。

到目前为止只看到了一种消息，即简单消息（flat message）。



## (1). 同步消息

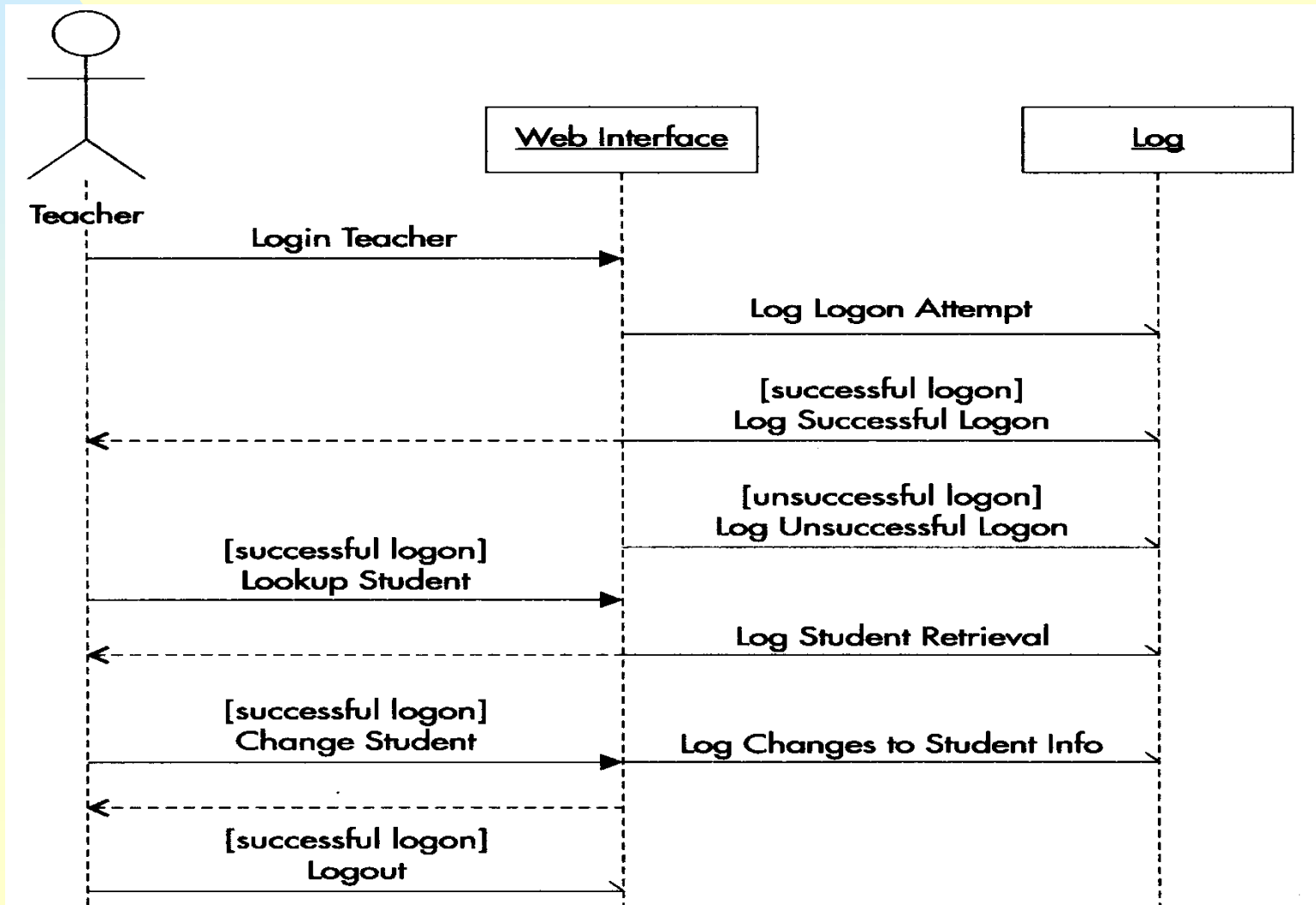
**同步消息** (synchronous message) 代表一个操作调用的控制流。同步消息的发送者把控制传递给消息的接收者，然后暂停活动，等待消息接收者的应答，收到应答后才继续自己的操作。





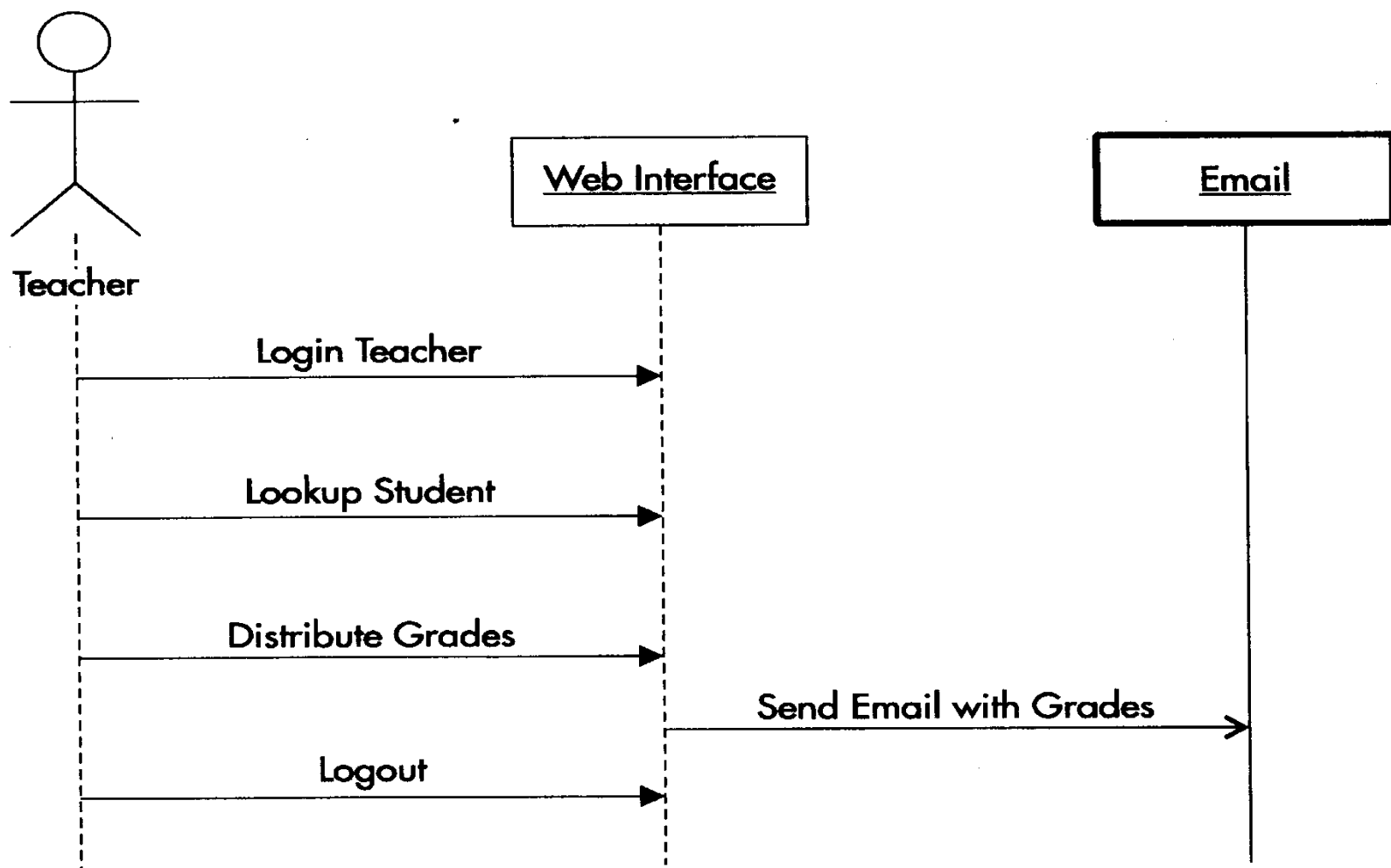
## (2) . 异步消息

**异步消息**（Asynchronous message）用于控制流在完成前不需要中断的情况。同步消息的发送者把控制传递给消息的接收者,然后继续自己的活动，不需等待接收者返回信息或控制。下面示例演示了如何在登录文件的情况下使用异步消息。



### (3) . 简单消息

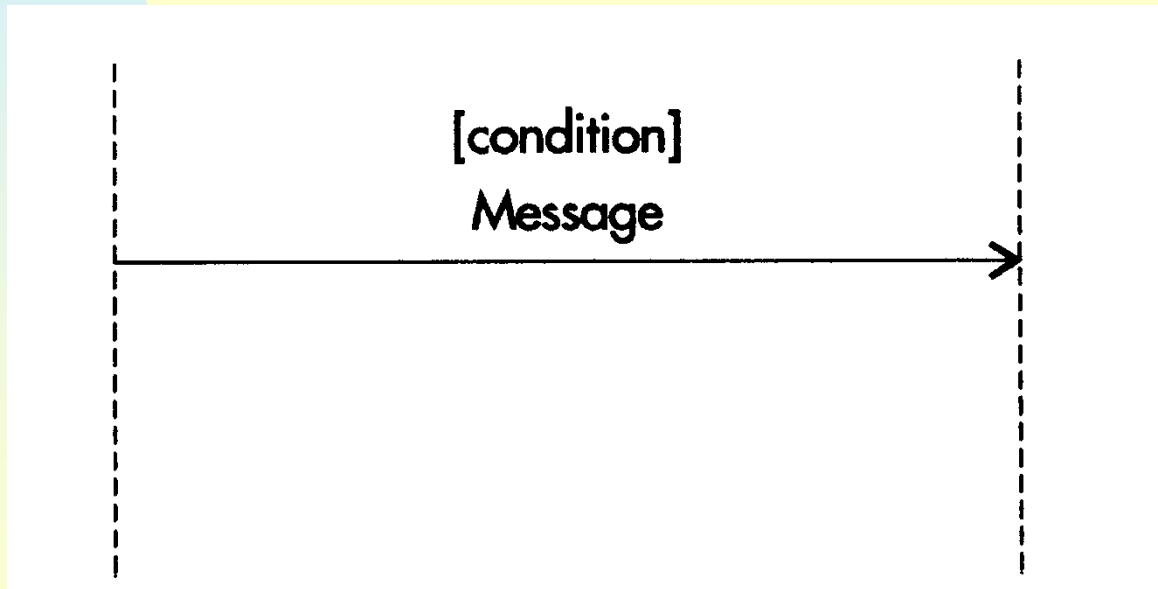
如果所有的消息都是同步或者异步消息，那么为什么还要简单消息呢？因为有时候我们不关心消息是同步还是异步，此外在高层分析中，有时候没有必要指定一个消息是同步的还是异步的。如下面的示例所示。



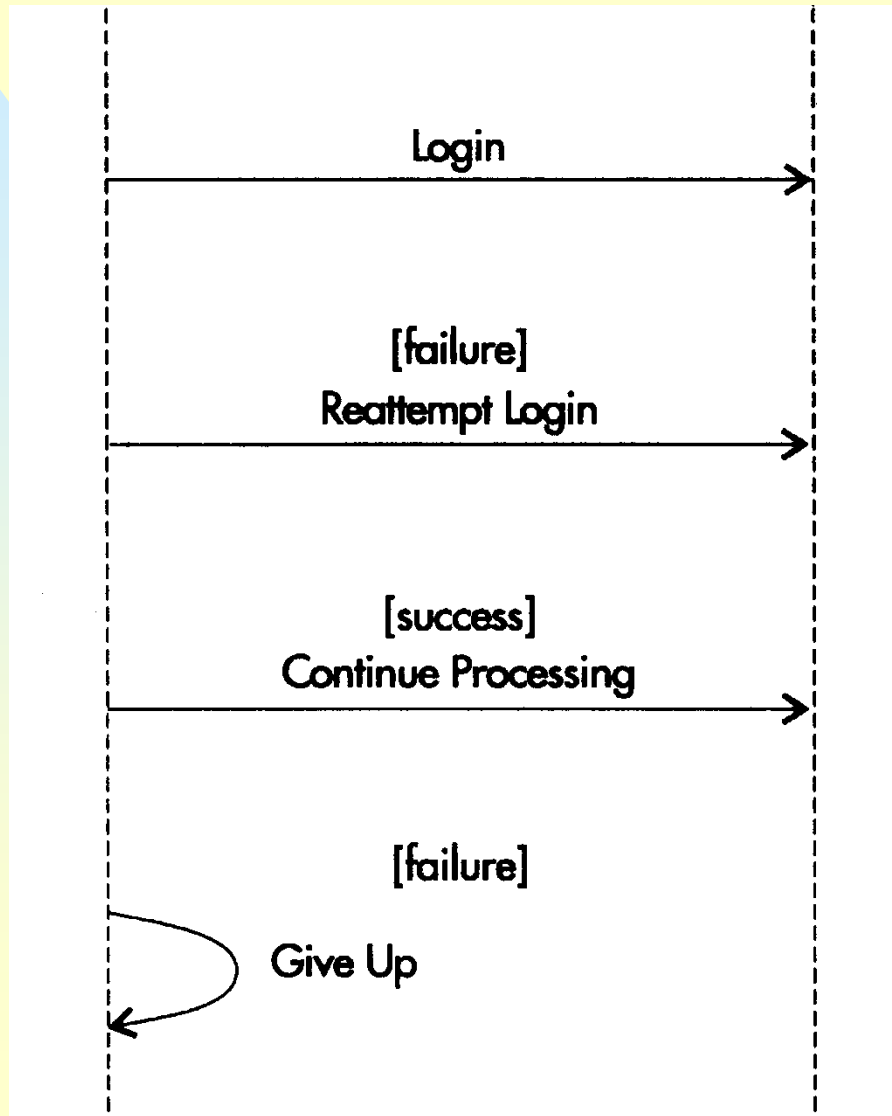
## 四、如何使用消息进行通信

消息是顺序图活动对象之间通信的惟一方式。UML中的消息使用了一些简洁的标记符。

消息可以包含**条件**以便限制它们只在满足条件时才能发送。条件显示在消息名称上面的方括号中，如下图所示。



下面示例演示了如何建模一个顺序图来显示登录尝试。如果登录失败，会在放弃登录之前重试一次，如下图所示。



下面看一个意义更加丰富的示例。对于Compile Application用例，我们可以创建一个成功编译工作流的顺序图。

这个顺序图中有4个活动对象：Developer、Compiler、Linker和FileSystem。Developer是系统的参与者。Compiler是Developer交互的应用程序。Linker是一个用来链接对象文件的独立进程。FileSystem是系统层功能的包装器，用来执行文件的输入和输出例程。

### Compile Application用例的顺序图操作：

Developer请求Compiler执行编译

Compiler请求FileSystem 加载文件

Compiler通知自己执行编译

Compiler请求FileSystem 保存对象代码

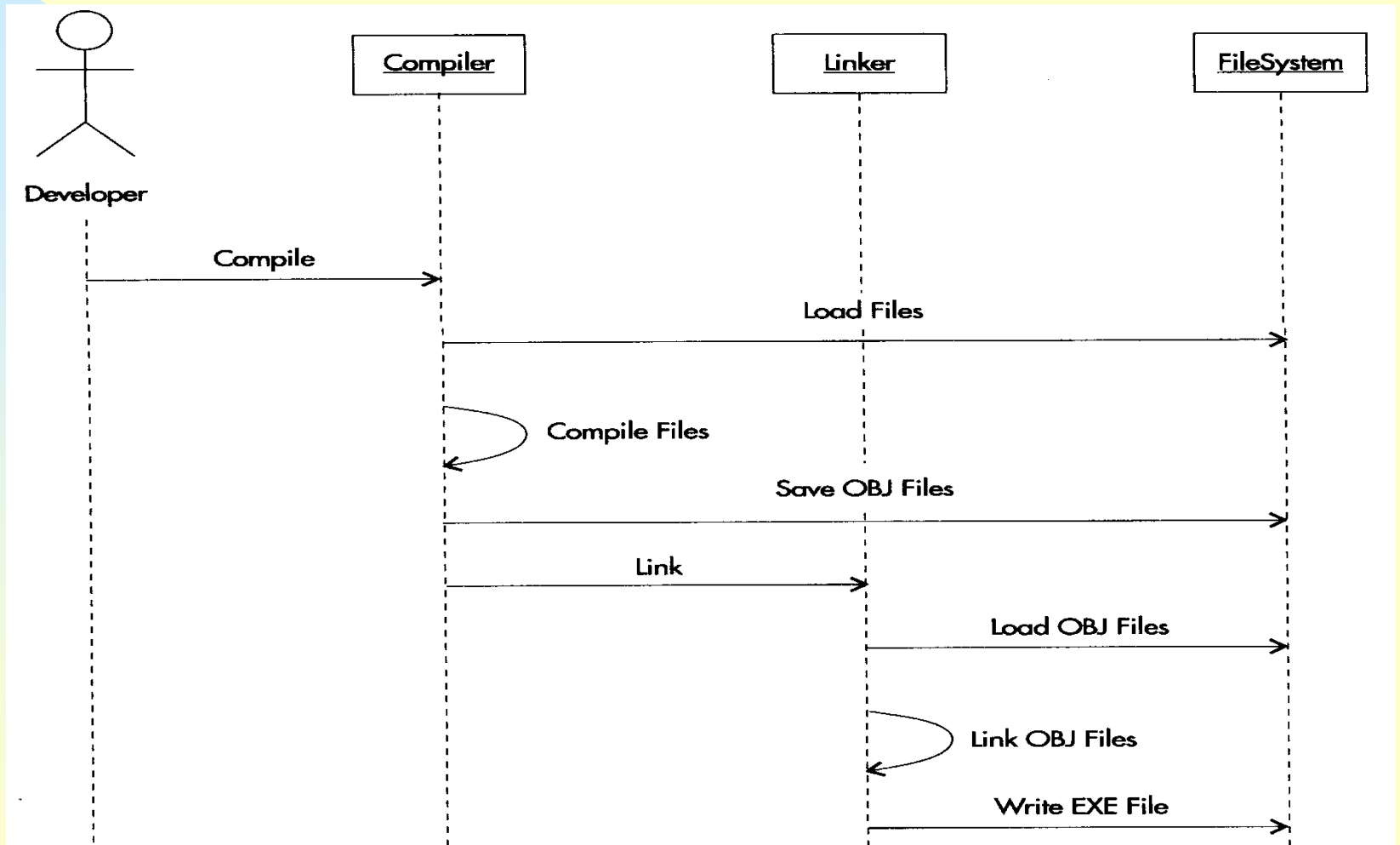
Compiler请求Linker链接对象代码

Linker请求 FileSystem加载对象代码

Liker通知自己执行链接

Linker请求FileSystem保存编译的结果

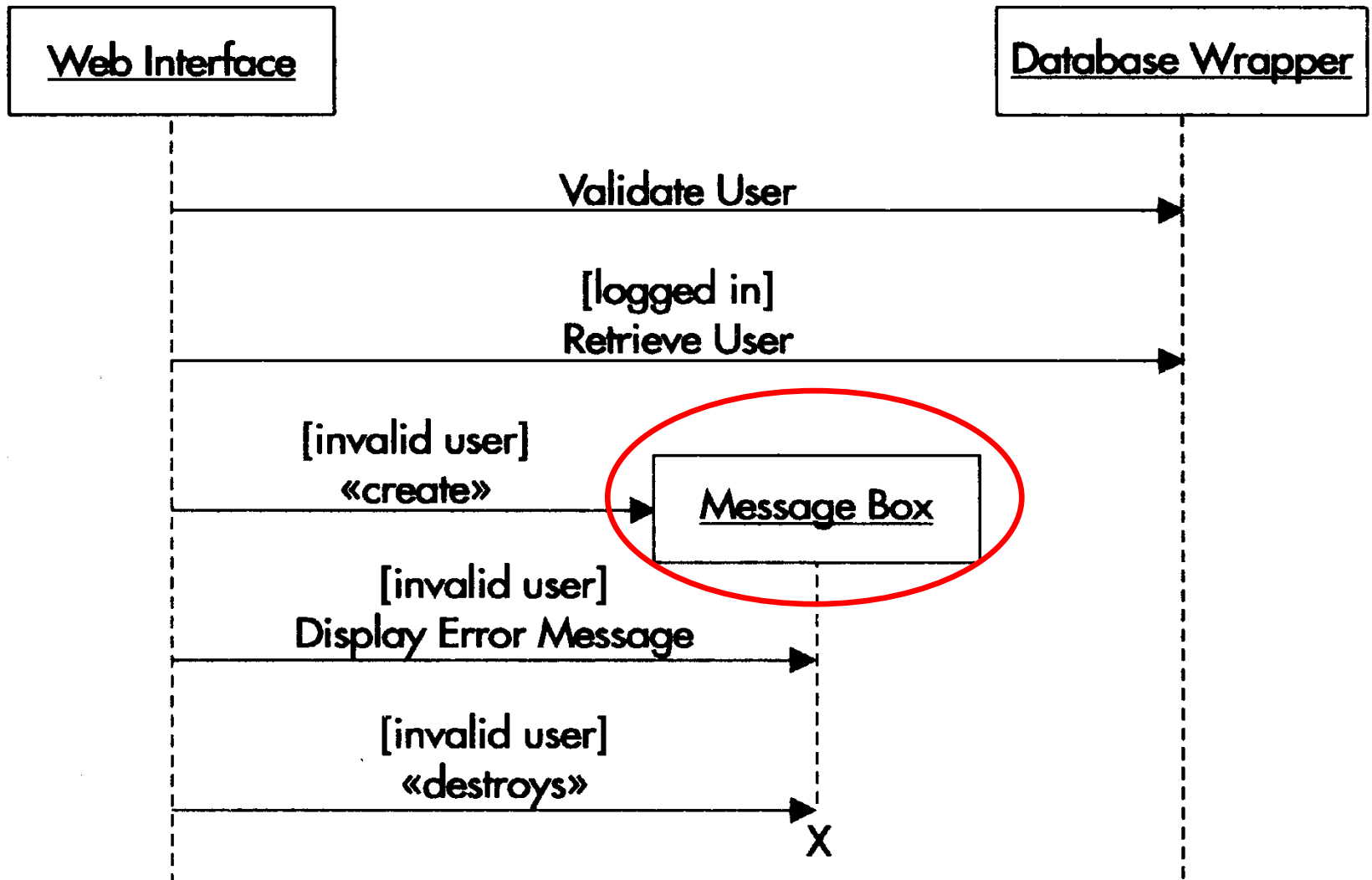
## 一个成功编译工作流的顺序图



## 五、顺序图的其他技术

### 1. 创建对象

创建对象的标记符如下图中的示例所示。有一个主要步骤用来把“create”消息发送给对象实例。对象创建之后就会具有生命线，就像顺序图中的任何其他对象一样。现在可以像顺序图中的其他对象那样来使用该对象发送和接收消息。在处理新创建的对象，或者处理顺序图中的任何其他对象时，都可以发送“destroys”消息来删除对象。若要想说明某个对象被销毁，需要在被销毁对象的生命线上放一个X字符。

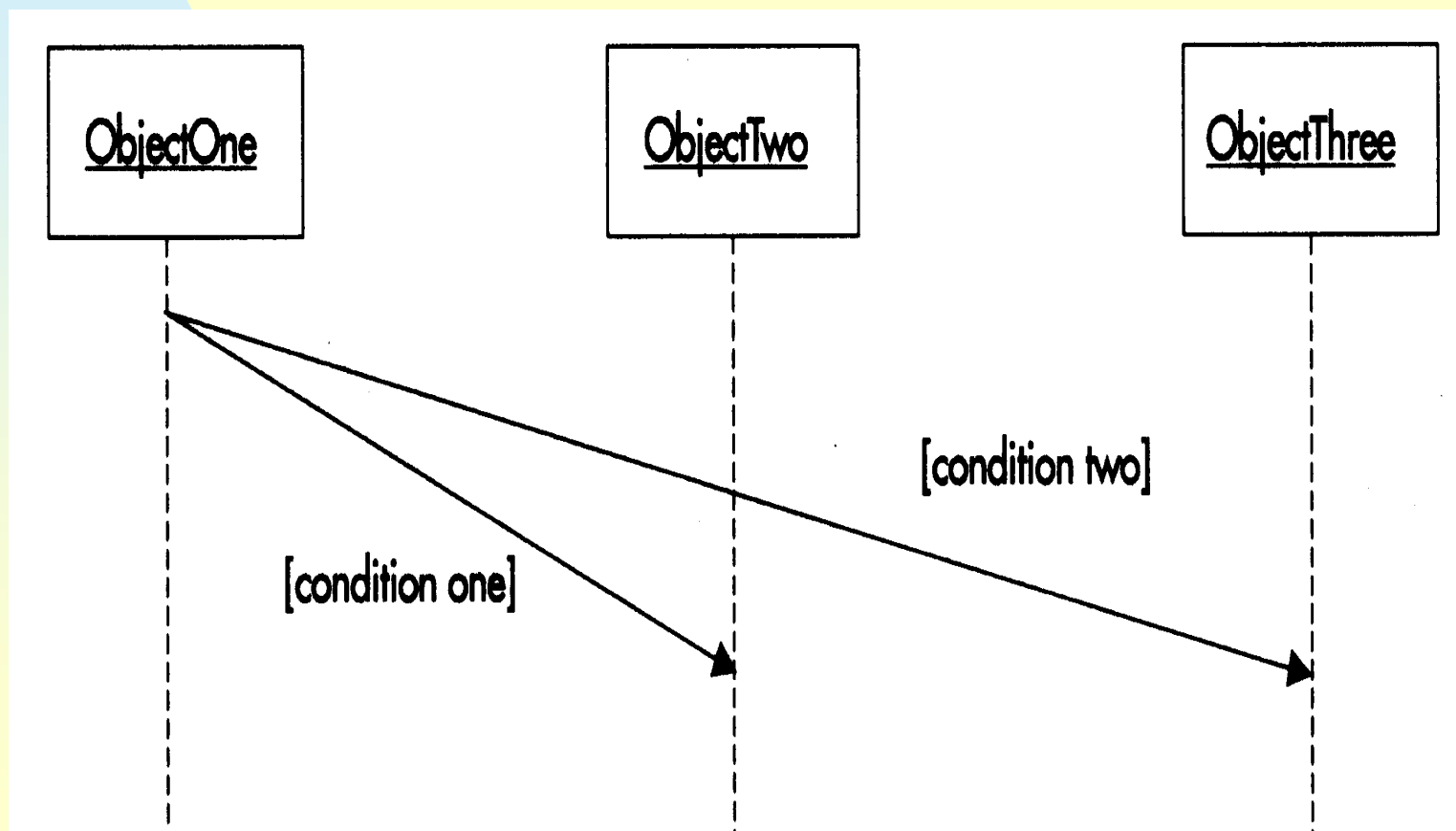




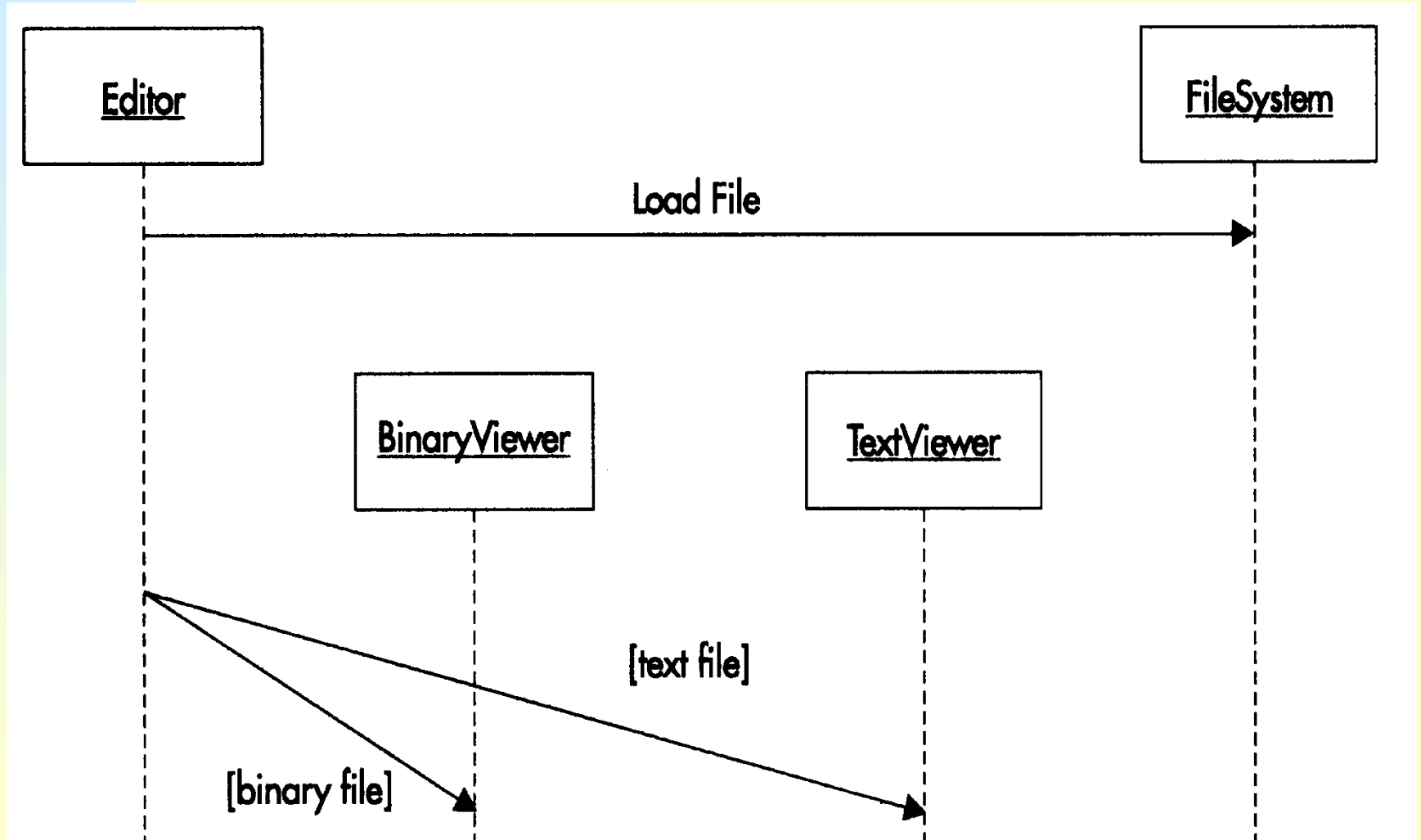
### 3. 分支和从属流

有两种方式来修改顺序图的控制流：使用分支和使用从属流。控制流的改变是由于不同的条件导致控制流走向不同的道路。

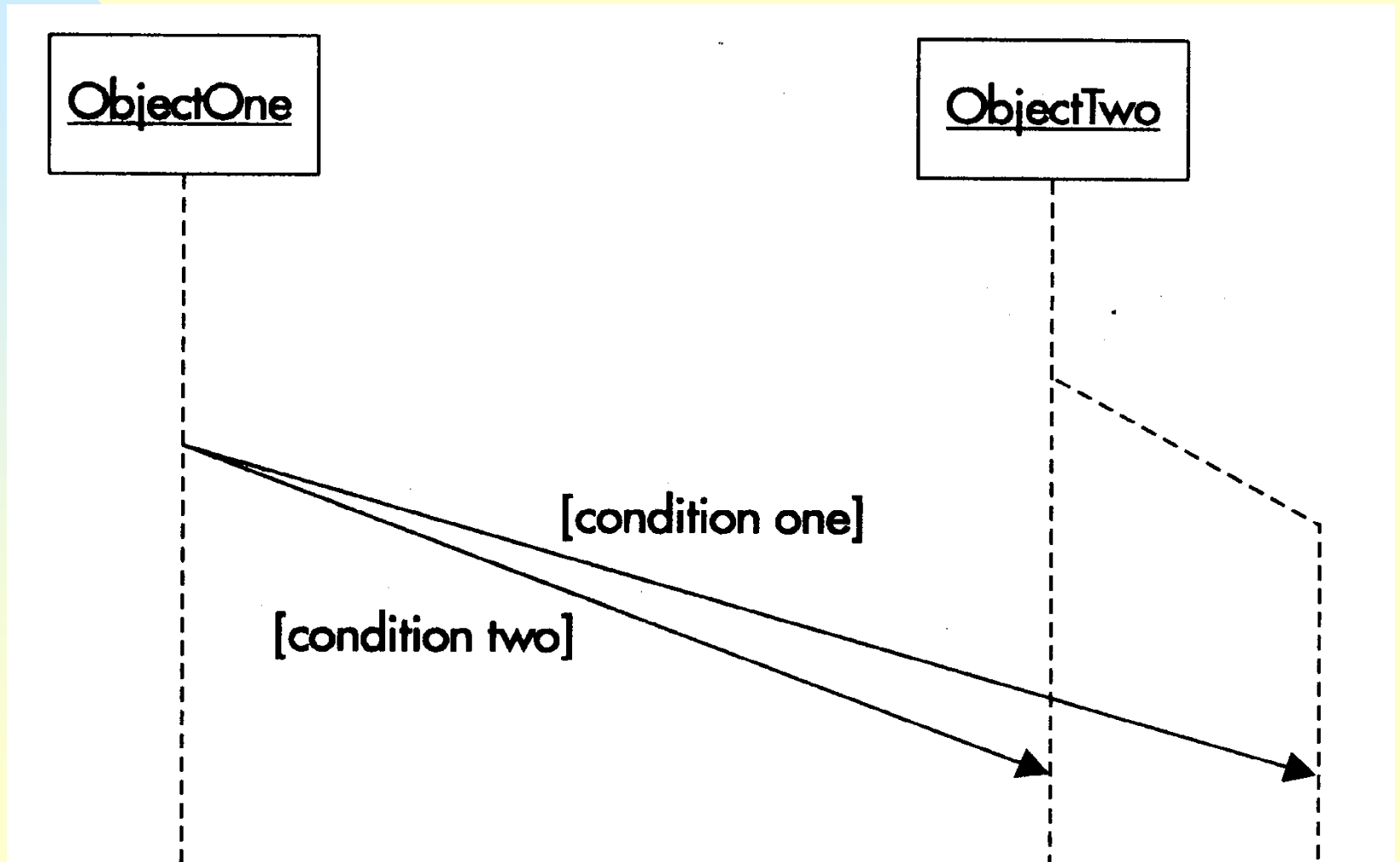
**分支**允许控制流走向不同的对象，如下图所示。



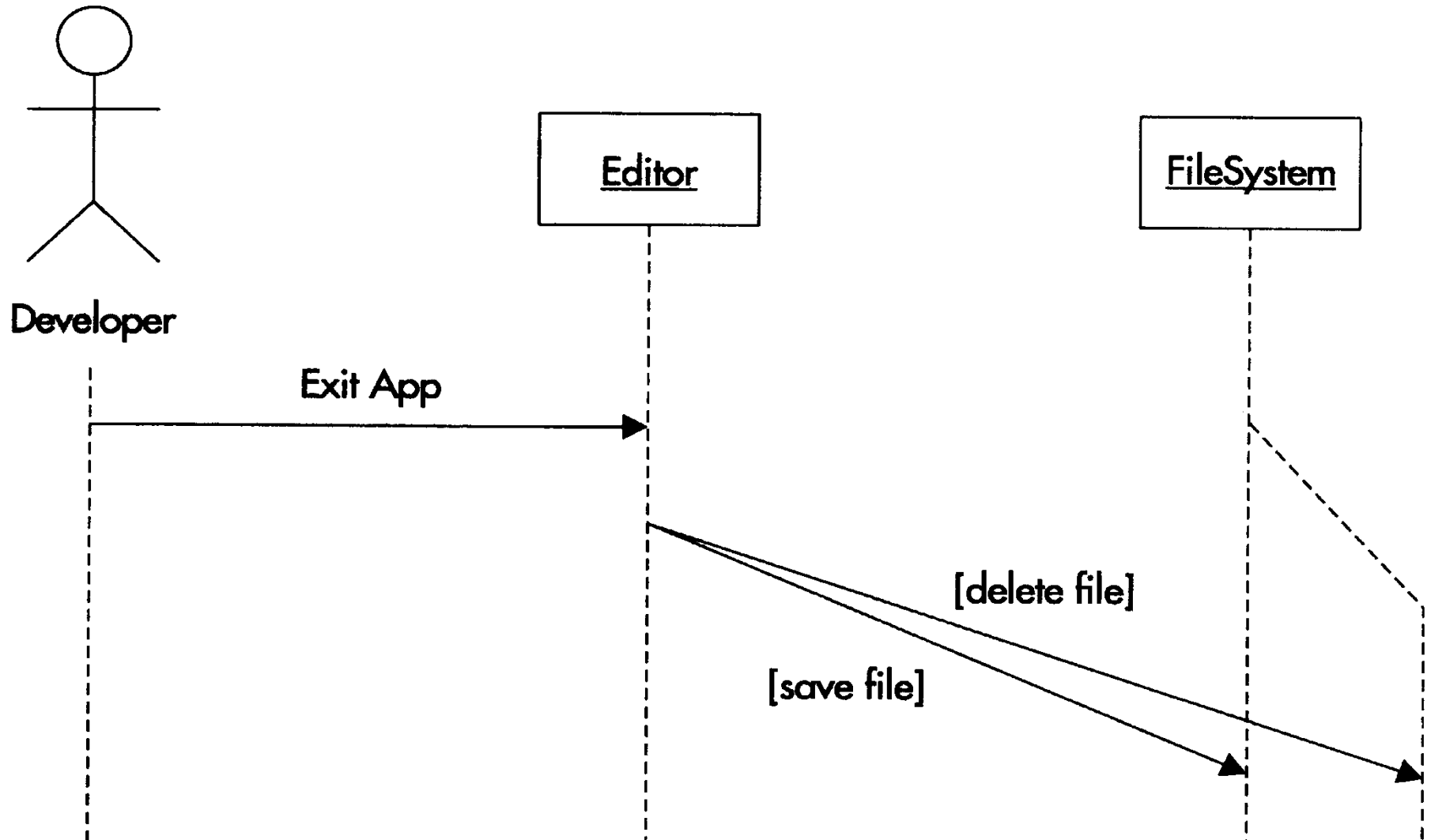
注意消息的开始位置是相同的，分支消息的结束“高度”也是相等的。这说明在下一步中，其中之一将会执行，如下图所示。



从属流还允许控制流根据条件改变，但是只允许控制流改变为相同对象的另一条生命线分支，如下图所示。



在下面的示例中，Editor在用户删除文件或者保存文件时向Filesystem发送一条消息。显然，Filesystem将会执行两种完全不同的活动，并且每一个工作流都需要独立的生命线，如下图所示。



创建顺序图包含4项任务：

- 1) 确定需要建模的工作流。
- 2) 从左到右布置对象。
- 3) 添加消息和条件以便创建每一个工作流。
- 4) 绘制总图以便连接各个分图。

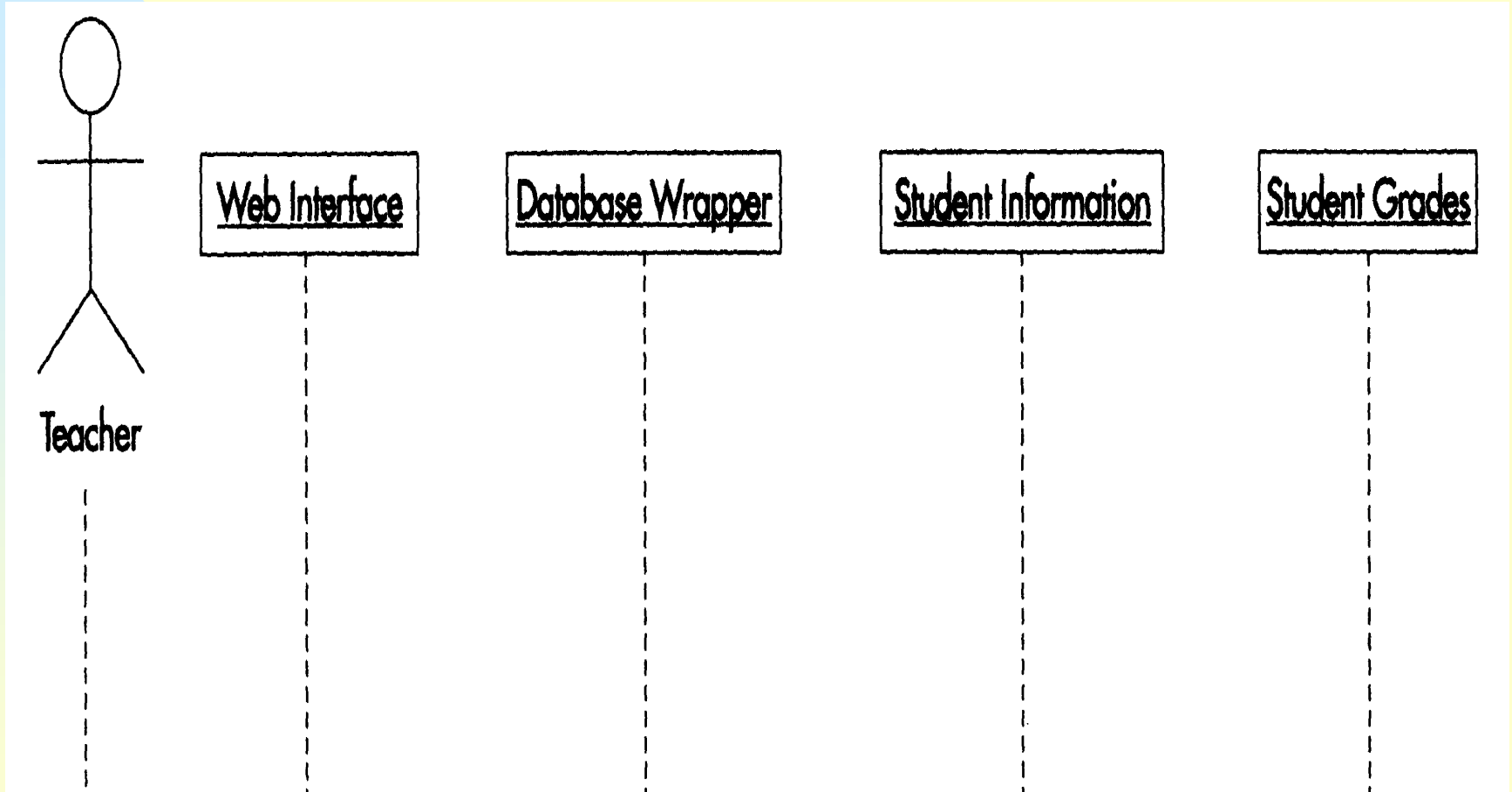
## 例子：建模“教师查询学生分数”

### 1. 确定工作流

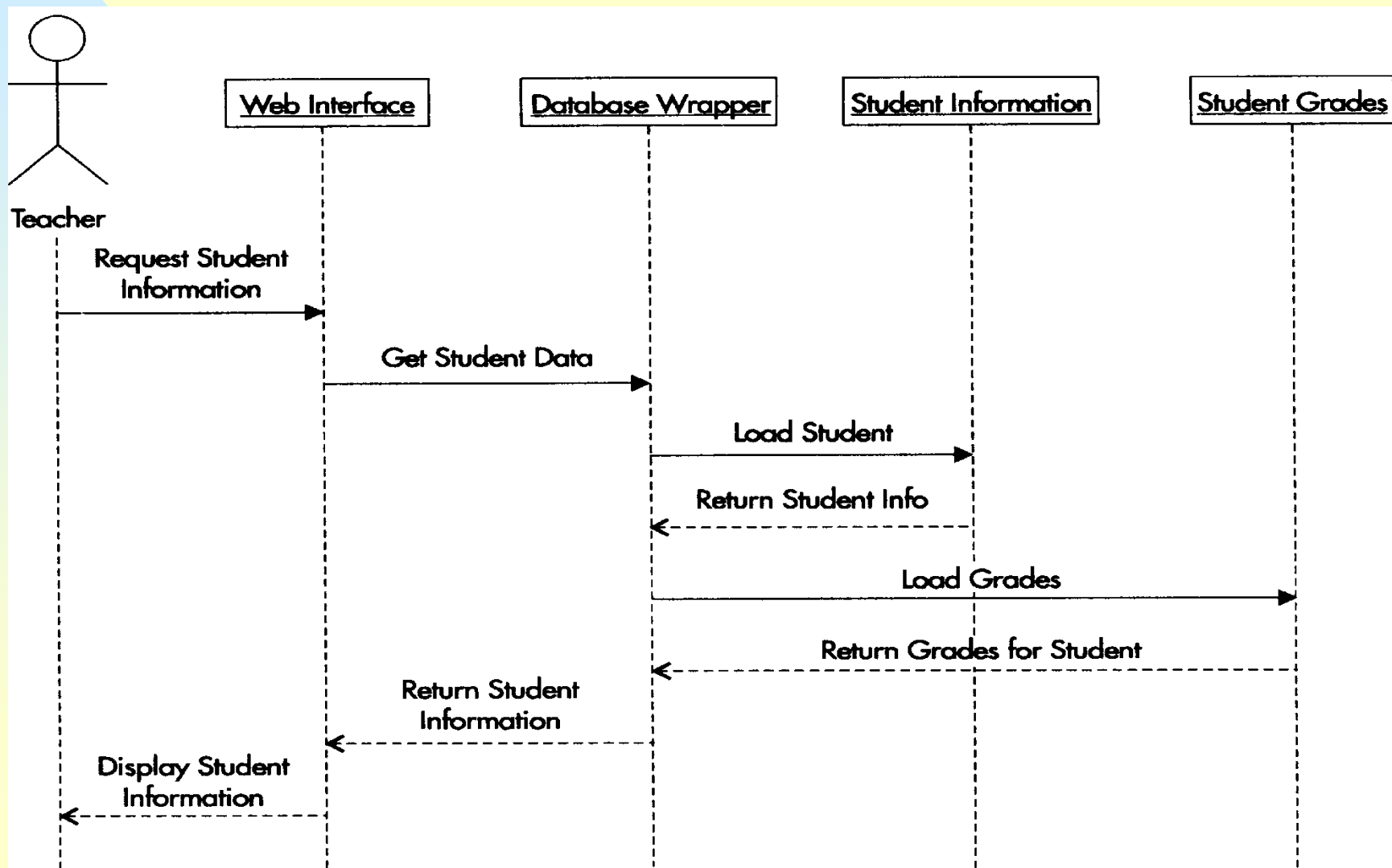
建模顺序图的第一步是确定将要建模的工作流。为此，需要至少标识出3个要建模的工作流：

- 教师成功地检查学生分数
- 教师试图检查某个学生分数，但是该学生在系统中不存在。
- 教师试图检查某个学生分数，但是该学生分数在系统中不存在。

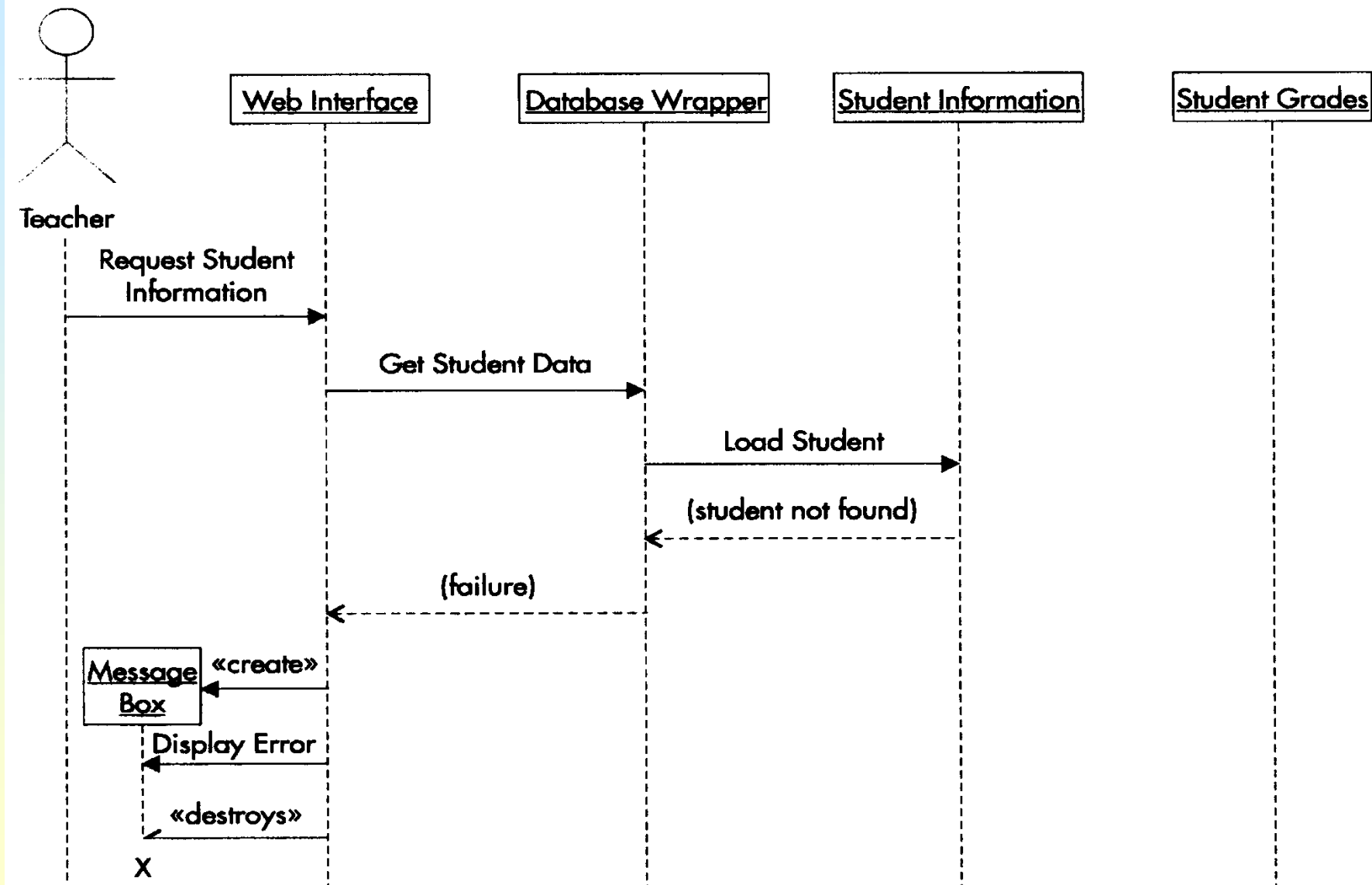
建模顺序图的下一步是从左到右布置所有的参与者和对象，包含要添加消息的对象生命线，如下图所示。



接下来，对每一个工作流作为独立的顺序图建模。从基本的工作流开始，它是没有出错条件，并且需要最少决策的工作流。在本例中，基本工作流是教师成功地检查某个学生的分数，如下图所示。

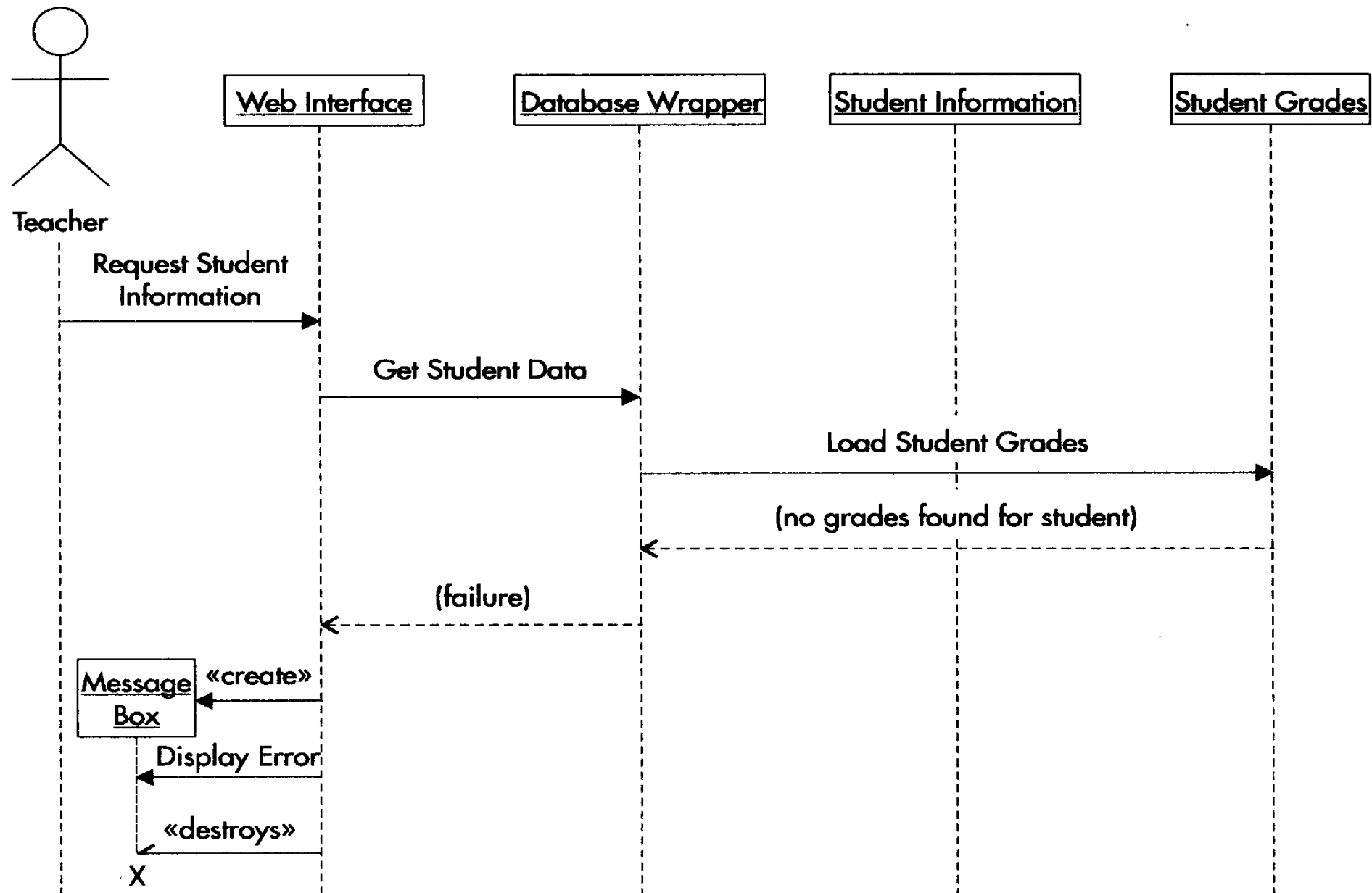


注意选择适当的消息类型（异步、同步、简单和返回）。接下来以独立的顺序图建模从属工作流。此处只建模否定的条件，如下图所示。





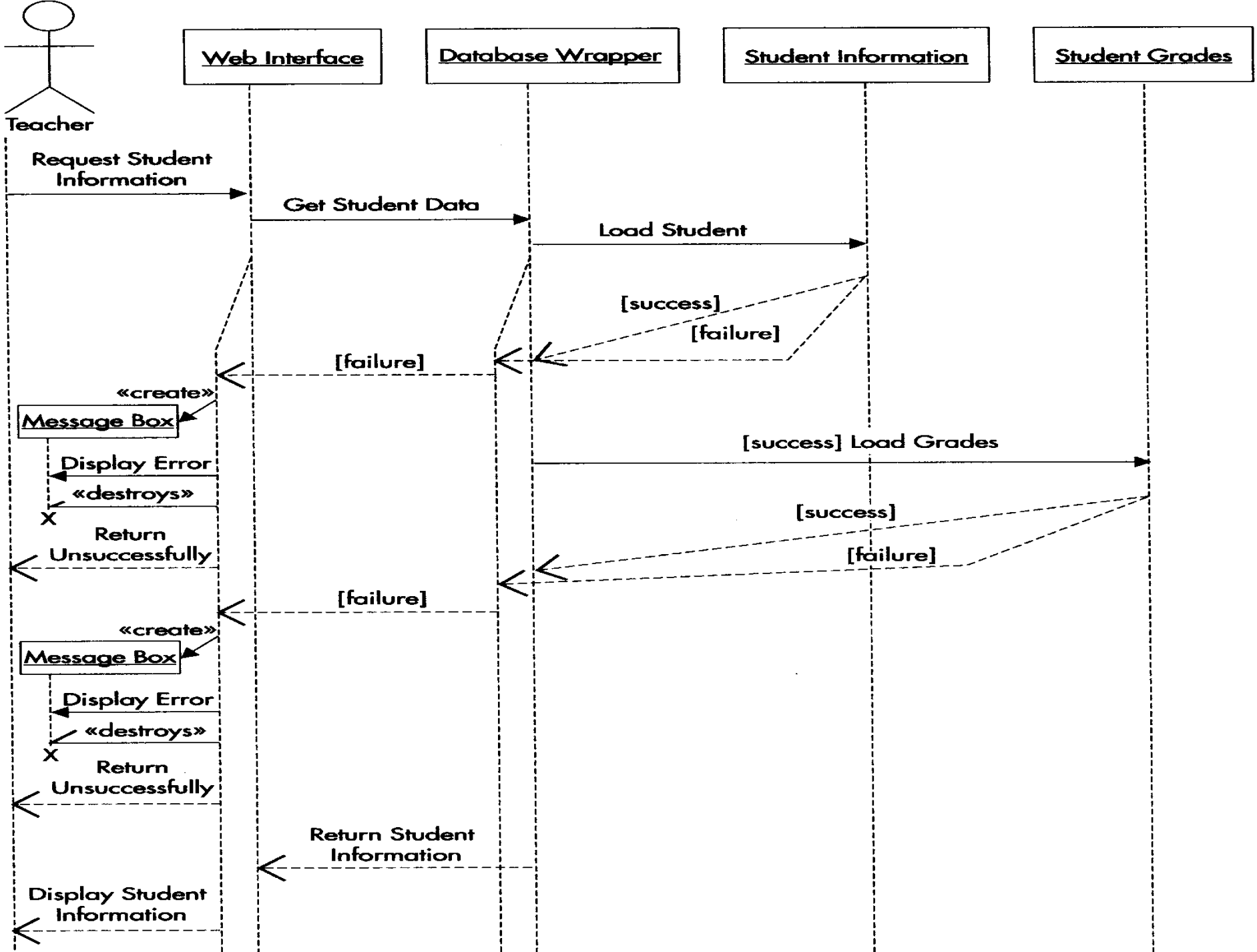
学生没分数，注意使用条件来指示在什么时候发送什么消息，如下图所示。

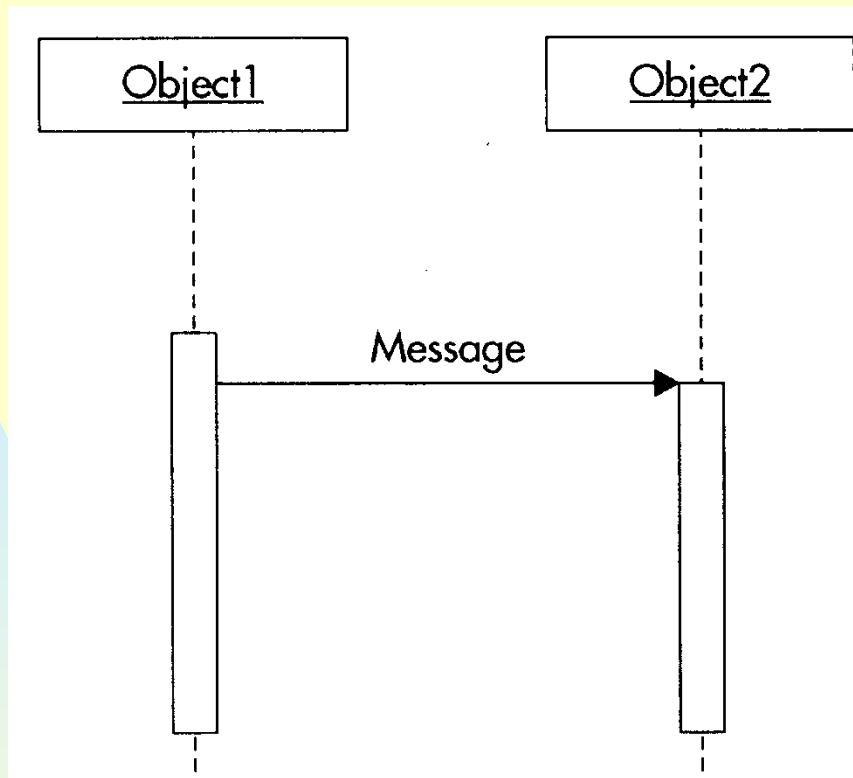


## 4. 绘制总图

建模顺序图的最后一步是**把所有独立的工作流连接为一个总图**，如下图所示。

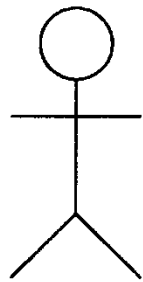
在此阶段，如果觉得前面的消息和交互对于当前的顺序图过于详细，可以让它们更加泛化一些，但是在软件建模的下一个阶段，就会觉得初始的各个顺序图越详细越好。





## 1. 使用控制矩形

控制矩形是一种用来帮助读者理解消息序列中对象涉及**时间**的标记符。在大多数情况下，某一时刻有一个对象活动焦点，但是，在带有异步功能的事件驱动的应用程序中，情况并非总是如此。例如下面这个对前节顺序图增强了的顺序图示例，如下图所示。



Teacher

Web Interface

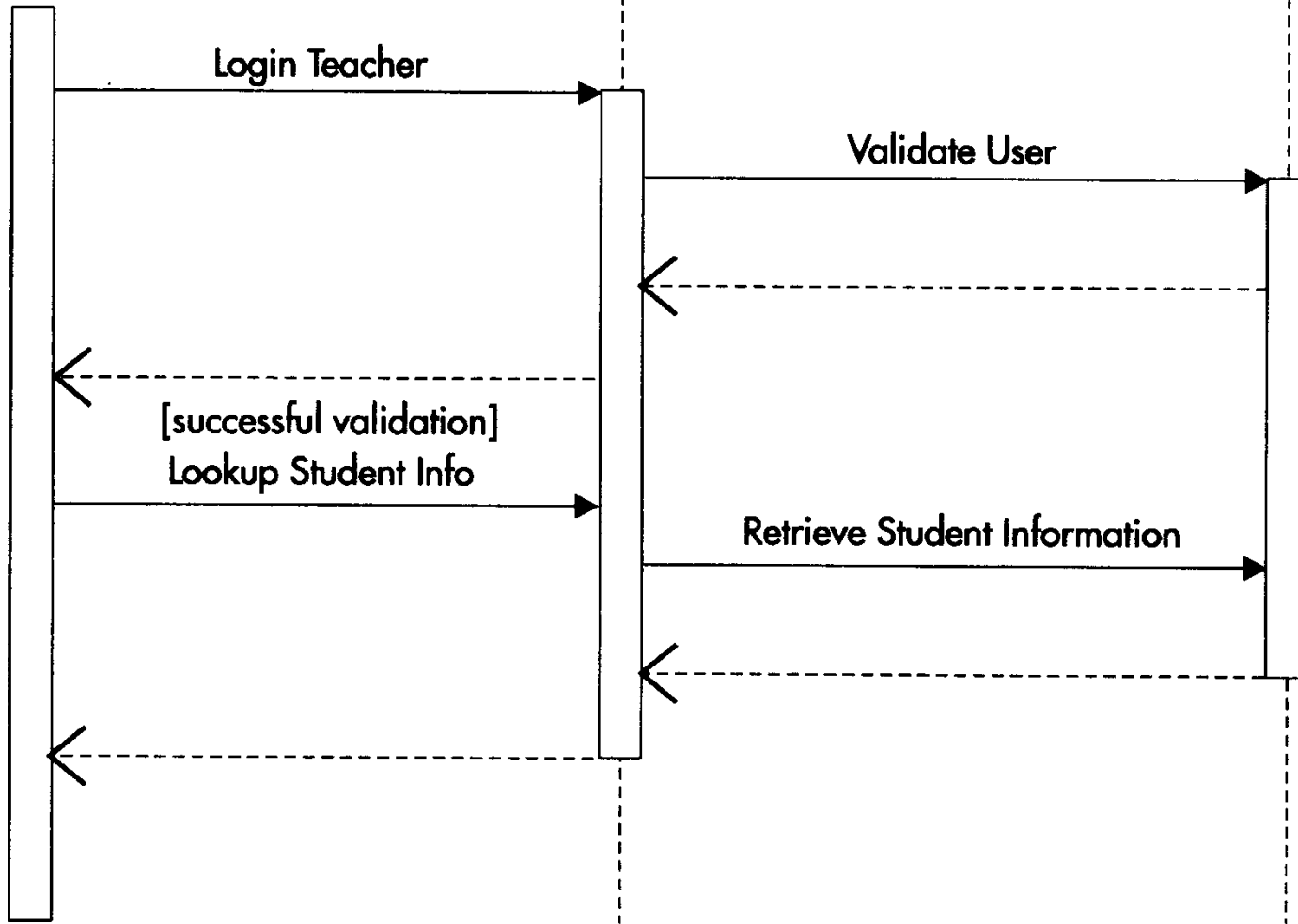
Database Wrapper

Login Teacher

Validate User

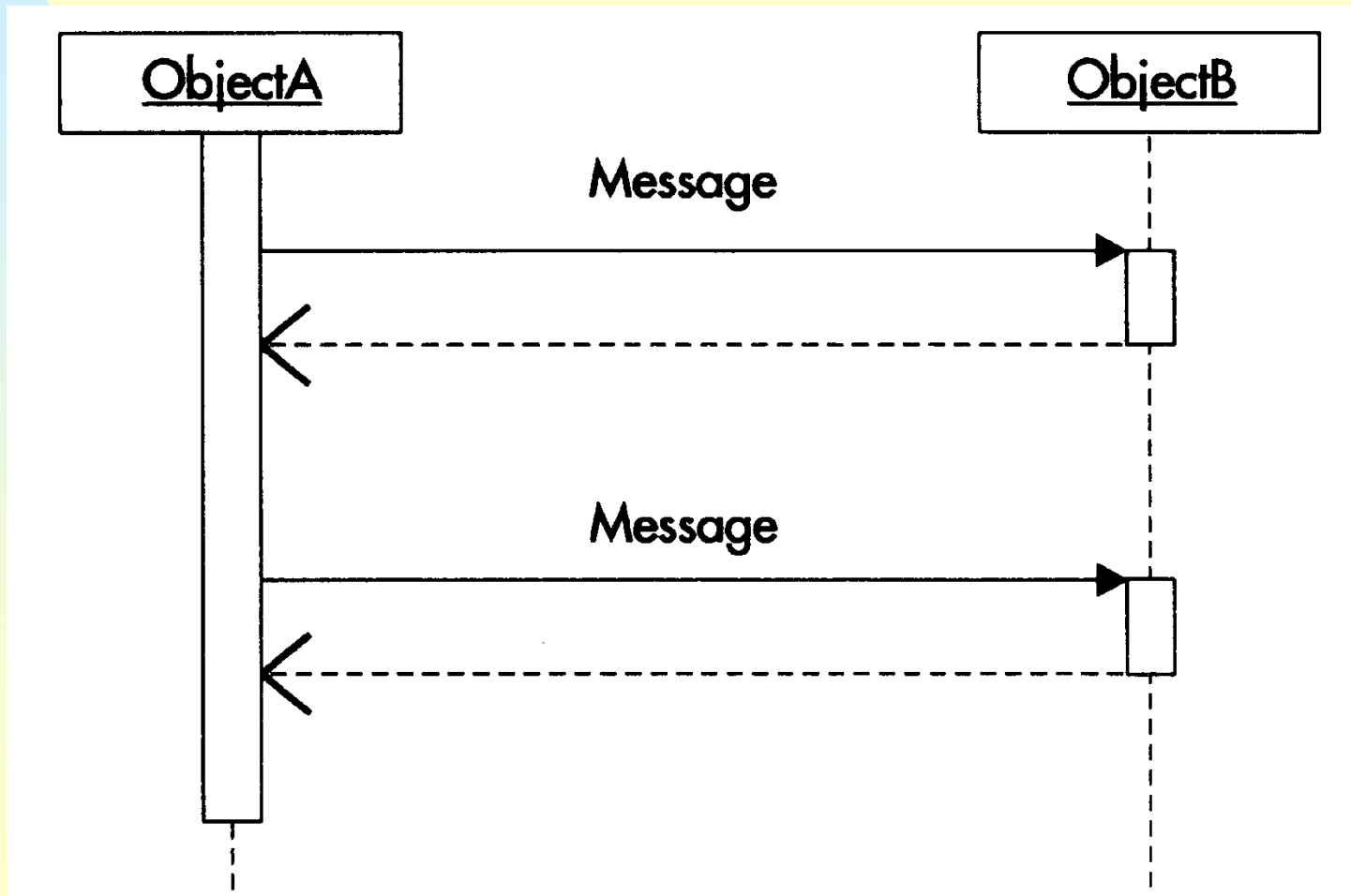
[successful validation]  
Lookup Student Info

Retrieve Student Information



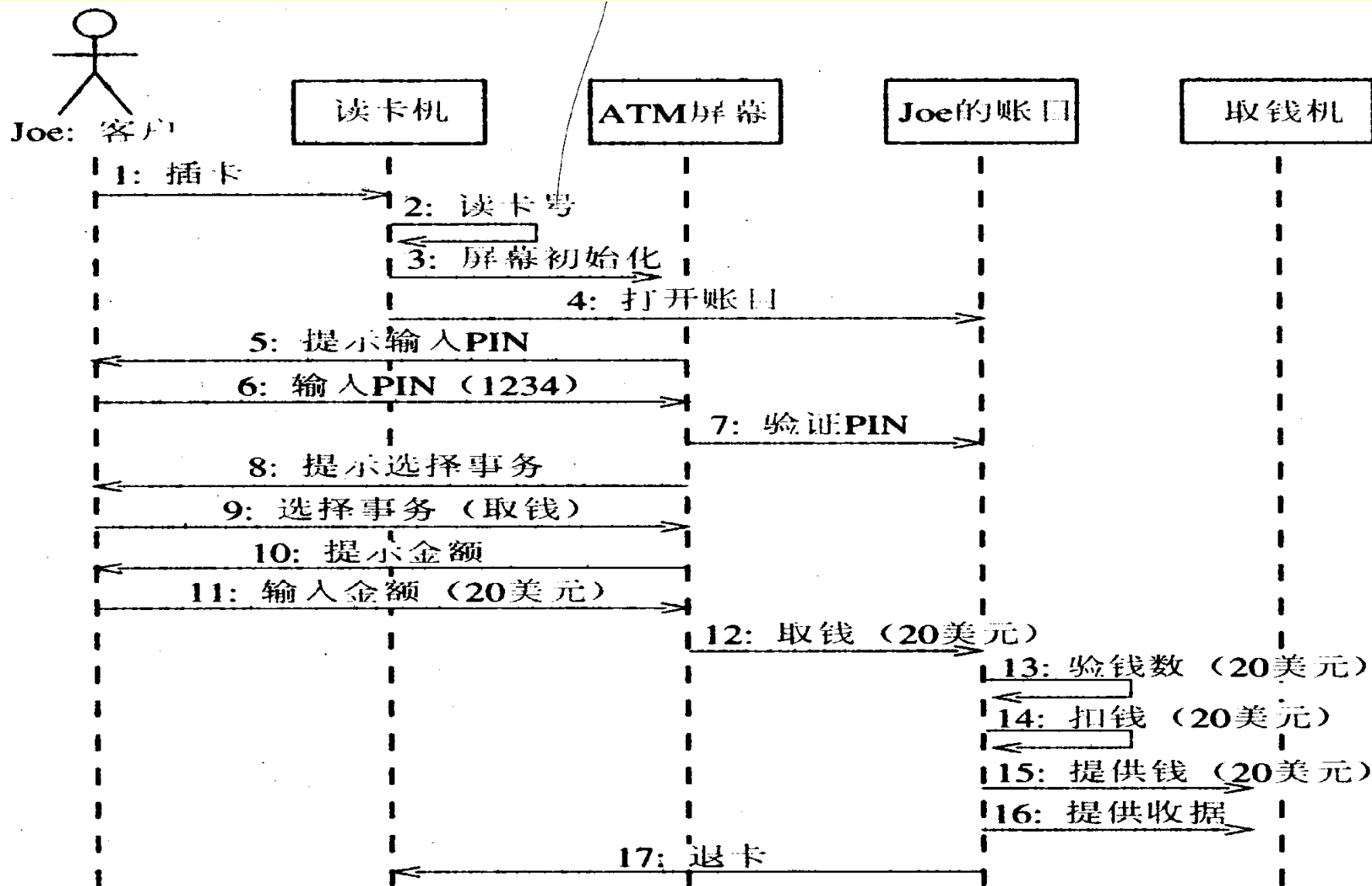
## 2. 指定活动期间

控制矩形不必总是扩展到对象生命线的末端，也不必连续不断，如下图中的示例所示。这个标记符演示了当消息处理完之后对象完全丧失控制权的情况。这表示对象没有在等待进一步的指令。



练习：建模一个ATM取款机取款的过程顺序图

对象：读卡机、ATM屏幕、账户、出钱机

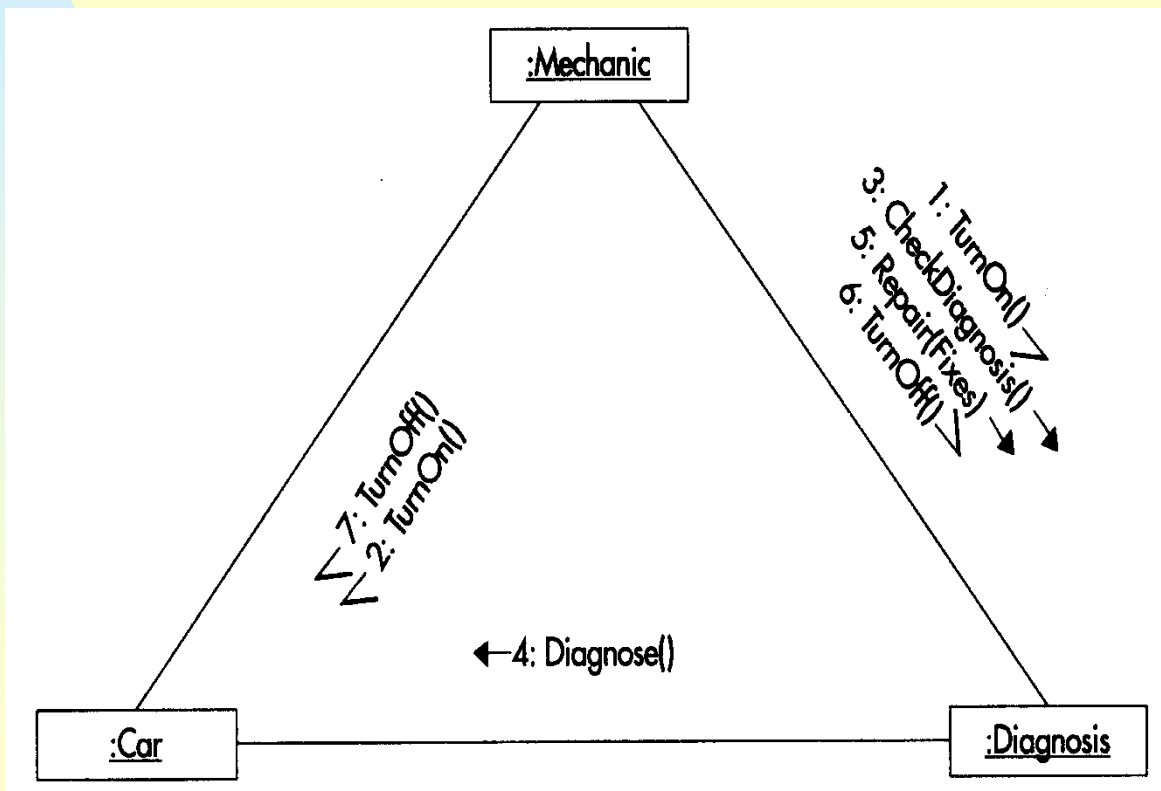





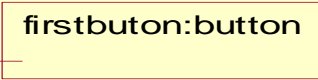

# 协作图

## 一、协作图

协作图描述对象之间的关联及其它们彼此之间的消息通信。要想使由类构成的系统具有功能，这些类的实例（对象）需要彼此通信和交互。换句话说，它们需要协作。



## 协作图中的事物及解释

事物名称	解释	图
参与者	发出主动操作的对象，负责发送初始消息，启动一个操作。	
对象	对象是类的实例，负责发送和接收消息，冒号前为对象名，冒号后为类名。	
消息流 (由箭头和标签组成)	箭头指示消息的流向，从消息的发出者指向接收者。标签对消息作说明，其中， <b>顺序号</b> 指出消息的发生顺序，并且指明了消息的嵌套关系；冒号后面是消息的名字。	

## 协作图中的关系及解释

关系名称	解释	关系实例
链接	用线条来表示链接，链接表示两个对象共享一个消息，位于对象之间或参与者与对象之间	

由于协作图要建模系统的交互，因此它必须处理类的实例。由于类在运行时不做任何工作，而是由它们的实例形式（对象）完成所有工作，因此，我们现在主要关心对象之间的交互。在协作图中可以使用3种标记类型的对象，如下图所示。

ObjectA

Object A (classifier not specified)

ObjectB : ClassB

Object B of Class B

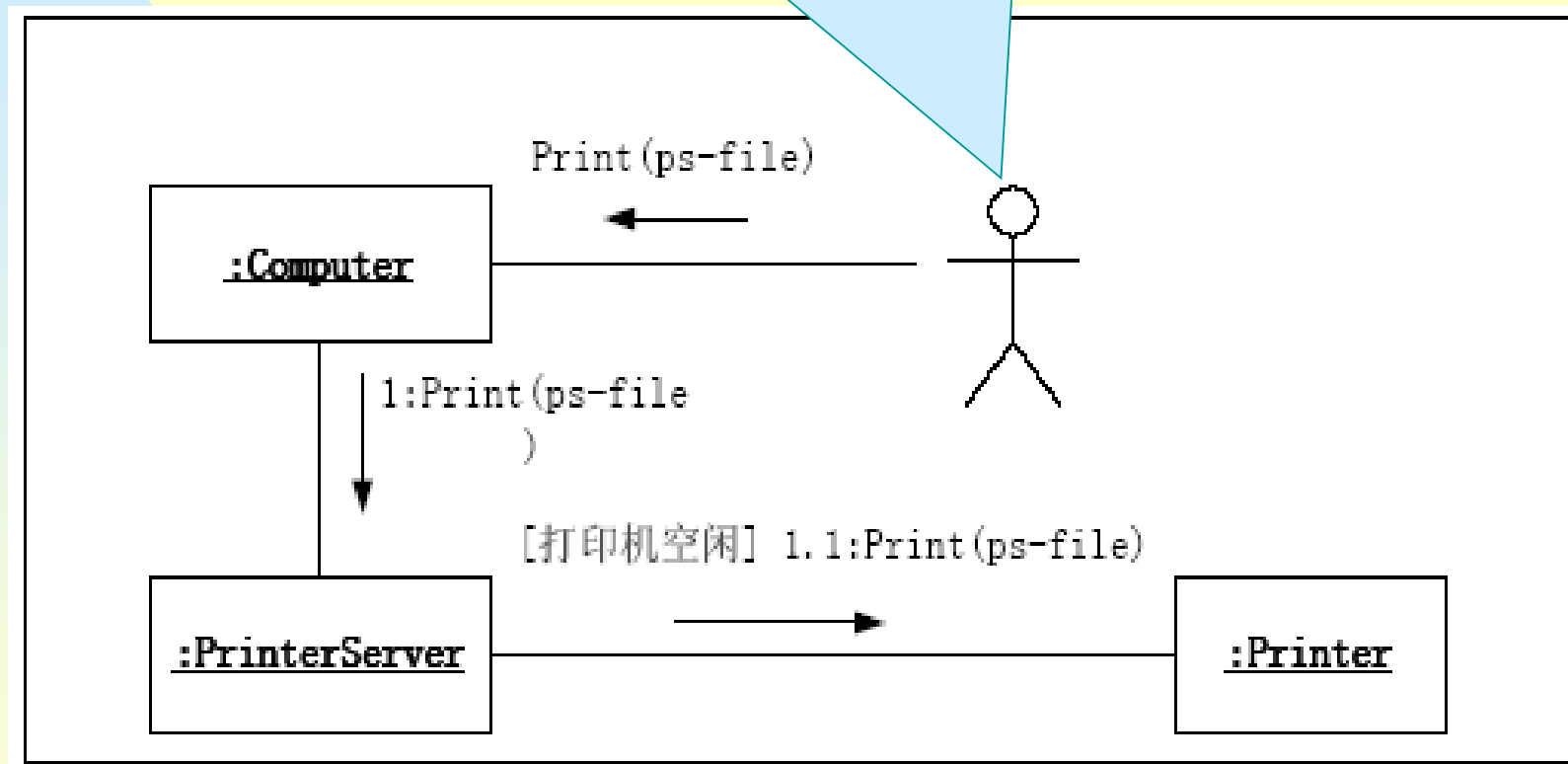
: ClassC

Unnamed object of Class C

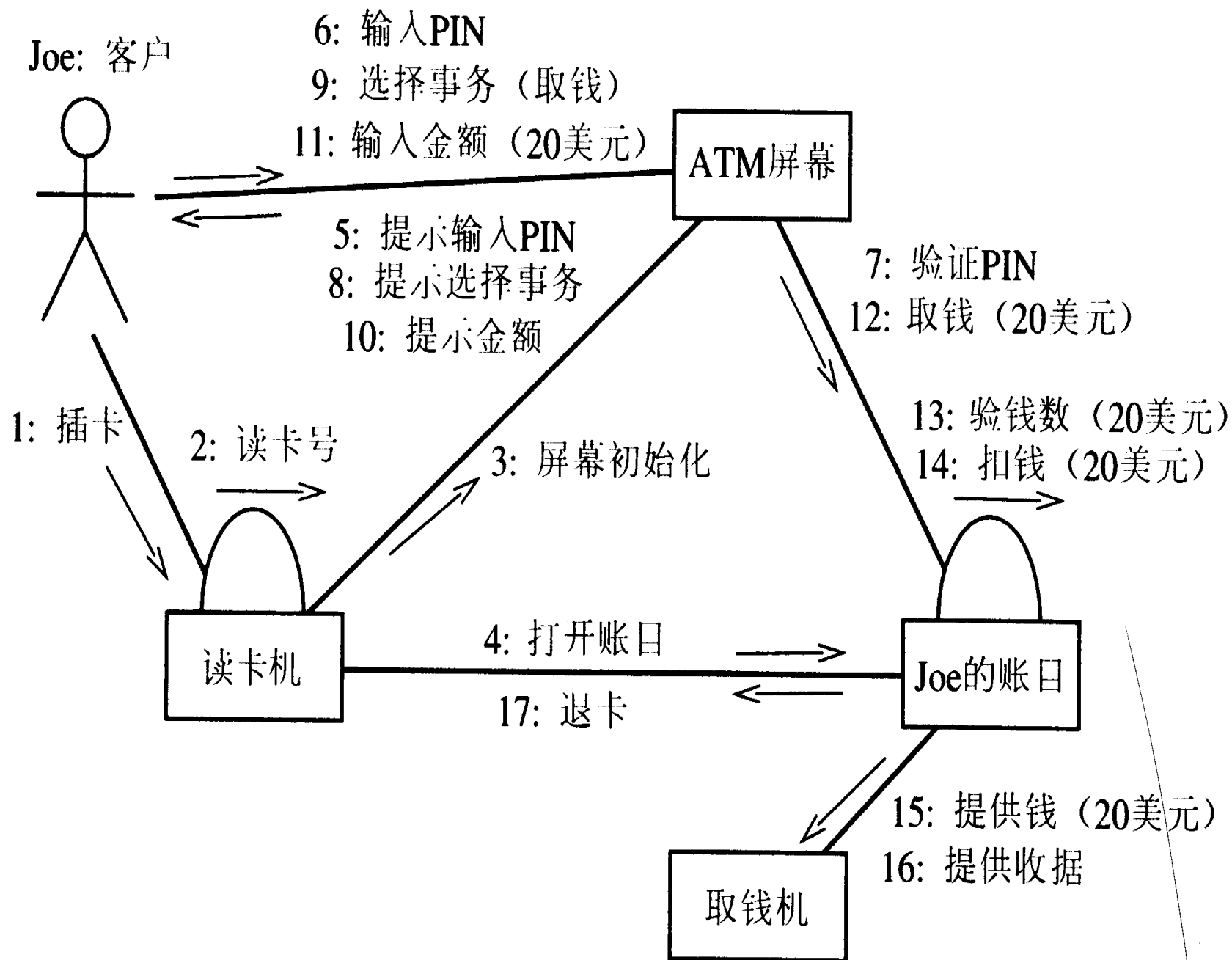
# 协作图例子

## 1. 打印操作的协作图

actor发送Print消息给Computer，Computer发送Print消息给PrintServer，如果打印机空闲，PrintServer发送Print消息给printer



# 取款机取钱的协作图



# 序列图和协作图区别

- 序列图和协作图都可以表示对象间的交互关系，但它们的侧重点不同。
- 序列图用消息的几何排列关系来表达对象间交互消息的先后时间顺序。
- 而协作图则建模对象（或角色）间的通信关系。