上海电力学院

Shanghai University of Electric Power

面向对象分析与设计

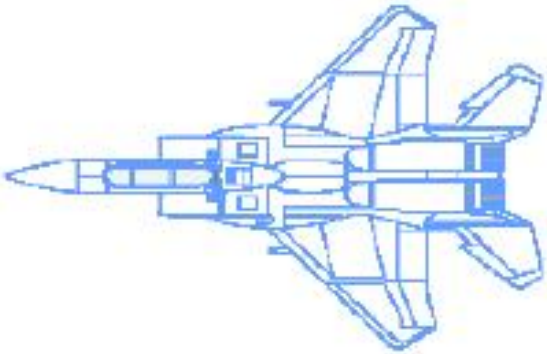
第二章 UML概述

www.shiep.edu.cn

UML简介

- 统一建模语言（UML：Unified Modeling Language）
- Unified
  - 组合了当前最好的面向对象软件建模方法
- Modeling
  - 用于表达现实的简化视图，以便于面向对象软件系统的设计与实现
- Language
  - UML主要是遵循精确语法的图形语言

为什么要建模



■ 建模的意义

- 模型是对现实的简化，建模是为了更好地理解系统
  - 模型帮助我们按照实际情况或需求对系统可视化
  - 模型能够规约系统的结构或行为
  - 模型给出了指导构造系统的模板
  - 模型对作出的决策进行文档化

■ 建模的原理

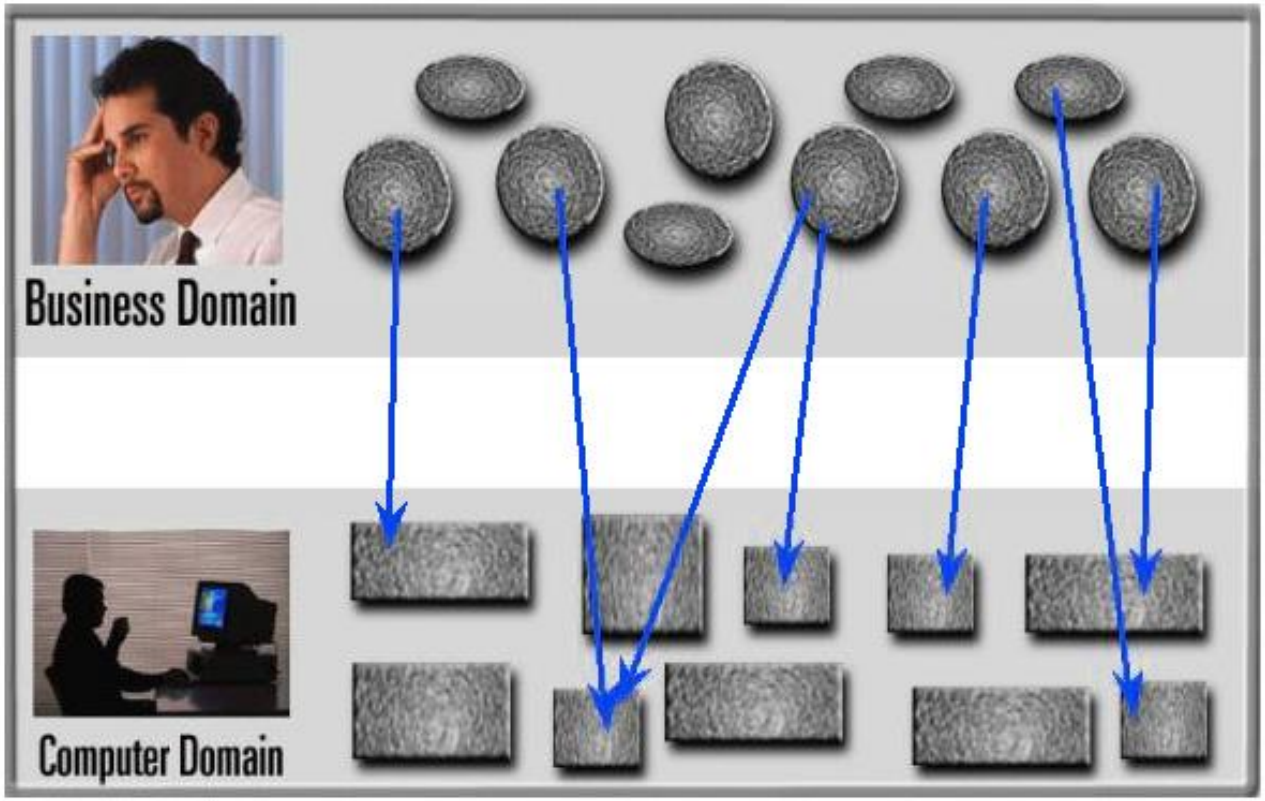
- 选择创建什么模型对如何动手解决问题和如何形成解决方案有意义深远的影响

■ 建模的原理（续）

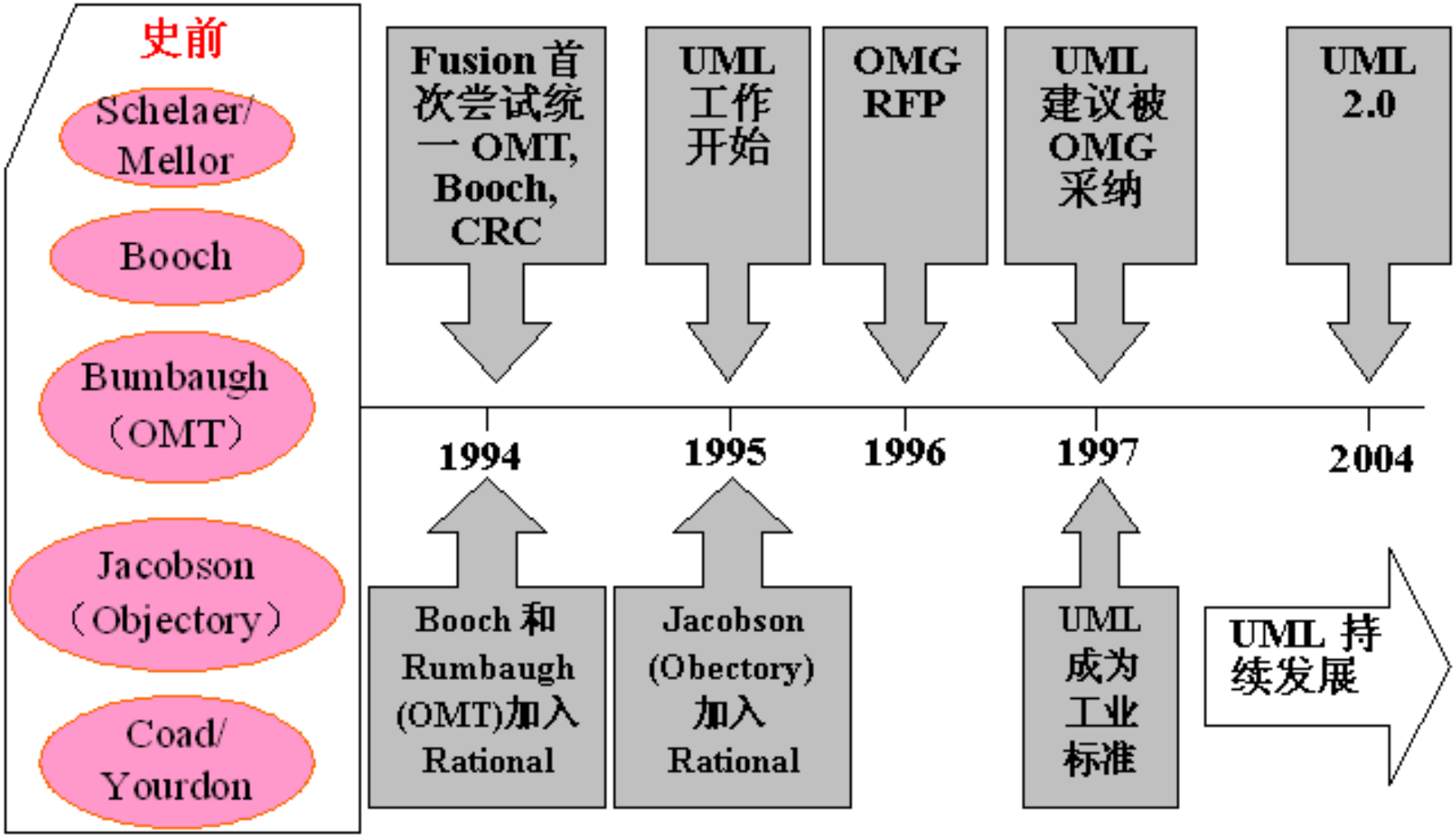
- 每一种模型可以在不同的精度级别上表示
- 最好的模型可以让你根据观察的角色及原因选择它的详细程度
- 对每个系统最好用一组几乎独立的模型去处理

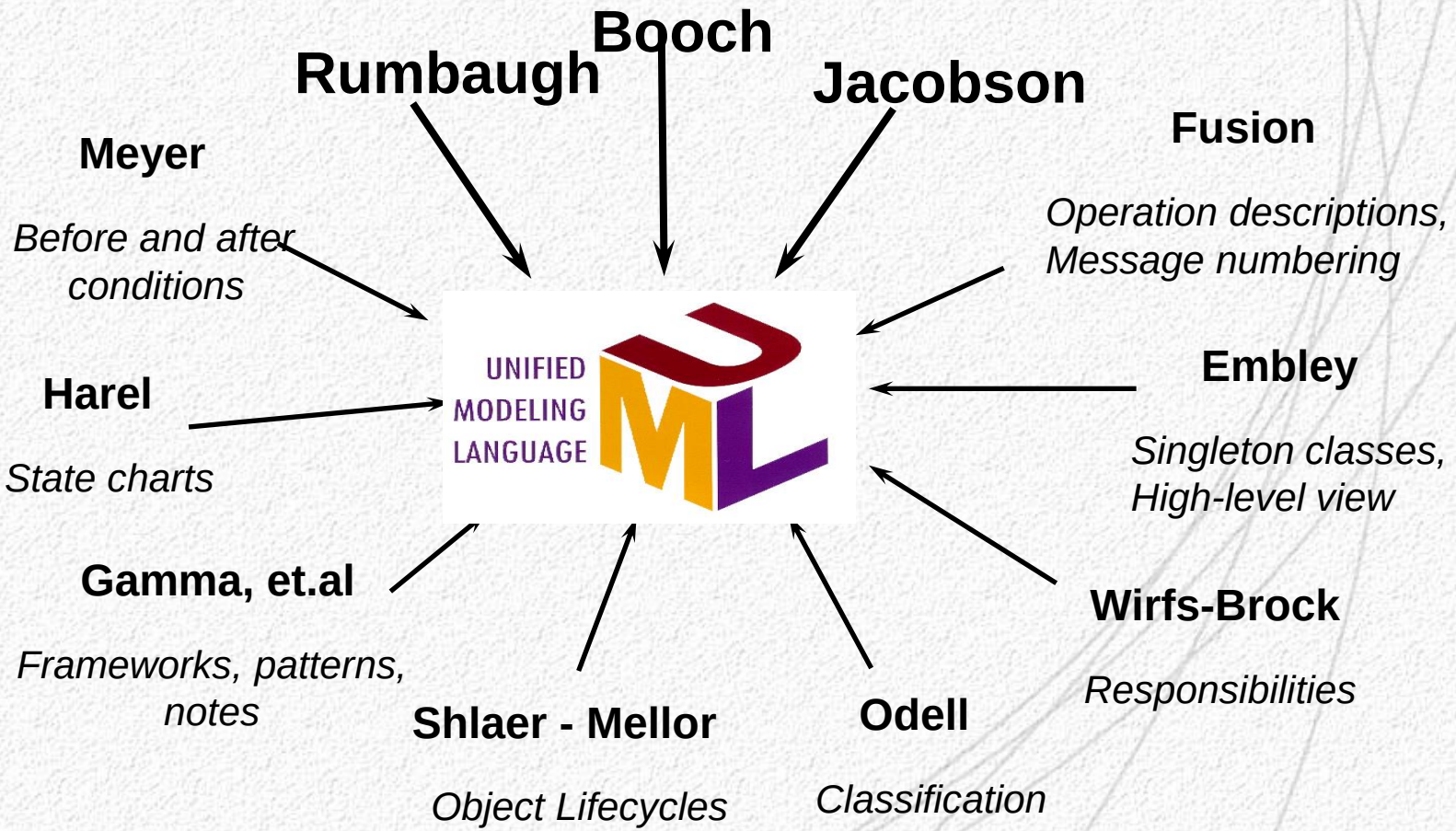
■ 选择UML

- 使用UML建立对象模型来映射现实世界



UML的发展


史前
Schelaer/ Mellor
Booch
Bumbaugh（OMT）
Jacobson（Objectory）
Coad/ Yourdon
Fusion首次尝试统一OMT, Booch, CRC
UML工作开始
OMG RFP
UML建议被OMG采纳
UML 2.0
1994
1995
1996
1997
2004
Booch和Rumbaugh (OMT)加入Rational
Jacobson (Obectory)加入Rational
UML成为工业标准
UML 持续发展


Booch
Rumbaugh
Jacobson
Meyer
Before and after conditions
Fusion
Operation descriptions, Message numbering
UNIFIED MODELING LANGUAGE
Harel
State charts
Embley
Singleton classes, High-level view
Gamma, et.al
Frameworks, patterns, notes
Wirfs-Brock
Responsibilities
Shlaer - Mellor
Object Lifecycles
Odell
Classification

■ Grady Booch

- Rational公司首席科学家，Booch方法发明人
- 提出了面向对象软件工程的概念，将Ada的工作扩展到整个面向对象领域
- Booch1993比较适合于系统的设计和构造

■ Jim Rumbaugh

- Rational公司高级研究员
- 提出了面向对象建模技术(OMT)，该方法用对象模型、动态模型、功能模型和用例模型来支持软件开发的全过程
- OMT-2特别适用于分析和描述以数据为中心的信息系统

■ Ivar Jacobson

- Rational公司副总裁，提出了OOSE方法
- OOSE的最大特点是面向用例的，用例(use-case)贯穿了整个开发过程
- OOSE比较适合于商业工程和需求分析

UML概述

- UML是一种标准语言，用以对软件密集型系统
  - 可视化（visualizing）
    - UML是图形化语言
    - 图形便于交流
  - 详述（specifying）
  - 构造（constructing）
  - 文档化（documenting）
    - 将所建造的系统记录下来
    - 便于新程序员跟进

■ UML是一种可视化建模语言

- 使用不同的语言或方式建立起来的概念模型不容易被他人理解，也不适合交流，因此采用相同的语言对系统进行描述是必要的
- 在软件系统中，某些部分很难被理解，除非建立模型
- 一个明确清晰的模型极大的有利于交流
- UML是一组图形符号，UML表示法中的每个符号都有明确语义
- 一个开发者用UML绘制一个模型，而另一个开发者（甚至工具）可以无歧义地解释这个模型

■ UML是一种可用以详细描述的语言

- 详细描述意味着所建的模型是精确的、无歧义的和完整的
- UML适于对所有重要的分析、设计和实现决策进行详细描述，这些是软件密集型系统在开发和部署时所必须的

■ UML是一种构造语言

- UML模型可以直接与大量的编程语言或数据库相关联
  - 与Java、C＋＋、Visual Basic、C#等语言的映射
  - 与关系型数据库中的表格或面向对象型数据库中的持久性存储的映射
  - 支持正向工程——从UML模型到编程语言的代码生成
  - 支持逆向工程——由编程语言代码重新构造UML模型

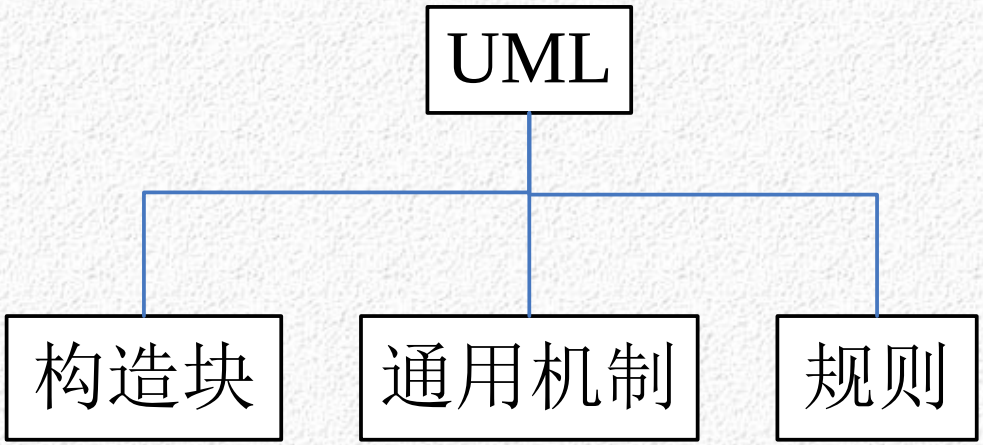
- **UML是一种文档化语言**
  - 使用UML可以对软件的架构、需求、测试、项目计划以及发行的版本等进行说明，并进行文档化

- UML是下面这些最好的建模方法中最好部分的集成
  - 数据建模的概念Data Modeling concepts（Entity Relationship Diagrams）
  - 业务建模Business Modeling（work flow）
  - 对象建模Object Modeling
  - 构件建模Component Modeling
- 它能支持用不同实现技术进行的软件开发全过程

- **UML提供了以下方面的应用建模语言**
  - 基于用例的业务建模
  - 类和对象建模
  - 组件建模
  - 分布和部署建模

UML的构成

- 从三种相关但不同的角度建模系统
  - 类模型
    - 表示系统静态的、结构化的“数据”层面
  - 状态模型
    - 表示系统时序的、行为的“控制”层面
  - 交互模型
    - 表示独立对象的协作，系统的“交互”层面



- 构造块：基本UML建模元素（事物）、关系 和图
- 通用机制：达到特定目标的公共UML方法
- 规则：将构造块放在一起的规则

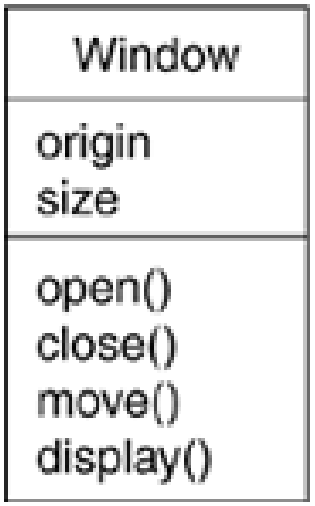
UML构造块

- UML的词汇表包括3种构造块（Building Blocks）
  - 事物（Things）：构成模型图的一些基本图示符号，它们表示一些面向对象的基本概念
  - 关系（Relationships）：表示基本图示符号之间的关系
  - 图（Diagrams）：特定的视角对系统所作的抽象描述
- 事物是对模型中最具有代表性成分的抽象；关系把事物结合在一起；图聚集了相关事物

UML构造块（事物）

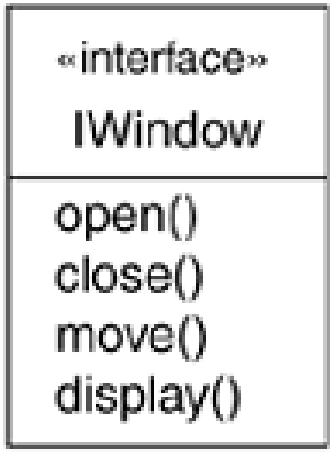
- 事物是UML中面向对象基本的模块，属于静态部分，代表物理或概念上的元素
  - 结构事物：类、接口、协作、用例、活动类、组件和节点
  - 动作事物：交互、状态机和活动
  - 分组事物：包
  - 注释事物：注解

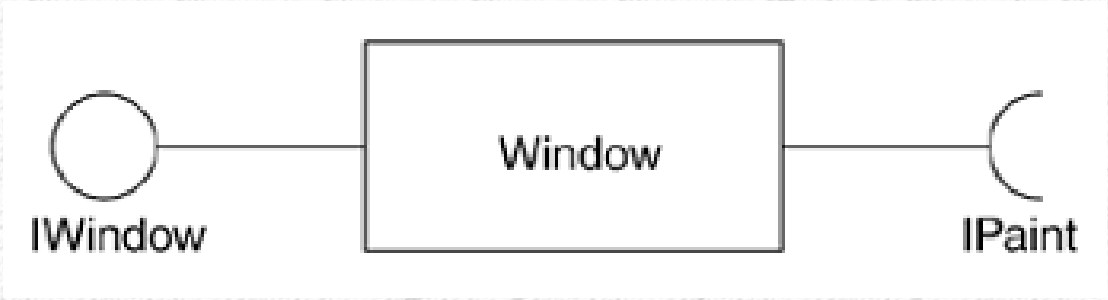
- **结构事物：是静态实体，不显示动 态行为，它们构成UML模型的名词，表示一个物理和概念元素，有7种结 构建模元素：**
  - **类(Class)：** 对一组具有相同属性、方法、关系和语义的对象的描述。一个类实现一个或多个接口，在图形上，把类画成一个矩形，矩形中通常包括类的名称、属性和操作。



- **接口(Interface)：**一个类或组件所提供的服务的操作集。接口仅仅是定义了一组操作的规范，它并没有给出这组操作的具体实现。接口看上去是在名称的上方标注着关键字<<interface>>的类

接口很少单独出现，把由类提供的对外接口表示成用线连接到类框的一个小圆圈，把类向其他类请求的接口表示成用线连接到类框的半个小圆圈

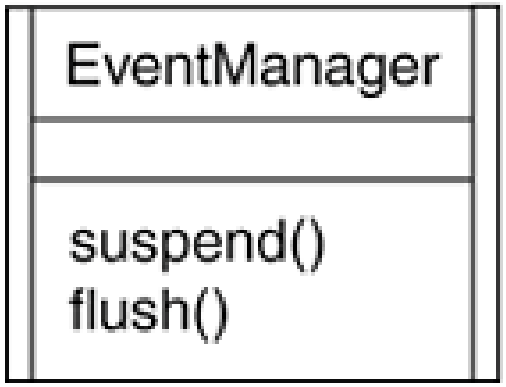




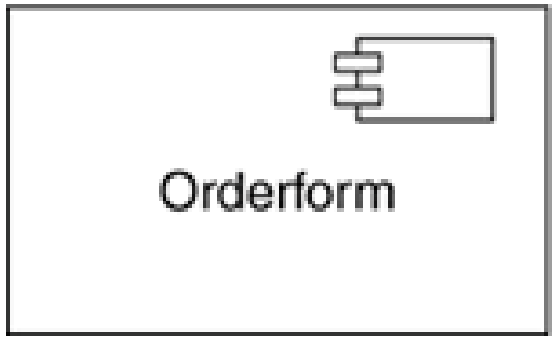
- 协作(Collaboration)：定义了一个交互，它是由一组共同工作以提供某合作行为的角色和其它元素构成的群体，这些合作行为大于所有元素的各自行为的总和。因此，协作有结构、行为和维度。一个给定的类可以参与几个协作。在图形上，协作画成虚线椭圆，有时仅包含它的名称

- 用例(Use Case)：对一组动作序列的描述，系统执行这些动作将产生一个对特定的参与者有价值且可执行的结果。用例用于构造模型中的行为事物，用例是通过协作实现的。在图形上，把用例画成实线椭圆，通常仅包含它的名称。

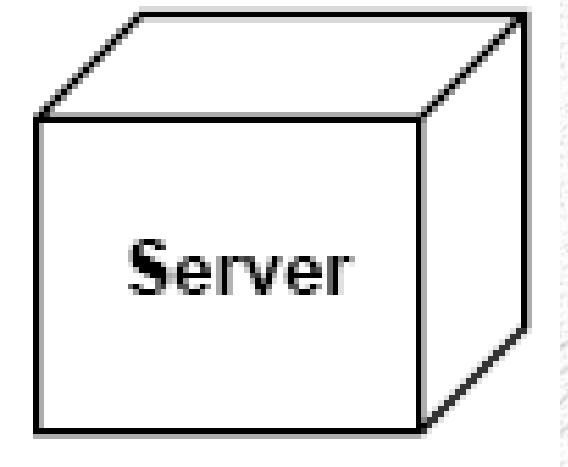
- 活动类(active class)：是这样的类，其对象至少拥有一个进程或线程，因此它能启动控制活动。活动类的对象所表现的元素的行为与其他元素的行为并发。在图形上，把活动类绘制成类图符，只是它的左右外框是双线，通常它包括名称、属性和操作。



- 组件(Component)：是系统设计的模块化部件，将实现隐藏在一组外部接口之后。在一个系统中，共享相同接口的组件可以相互替换，只要保持相同的逻辑行为即可。在图形上，组件表示很象类，只是在其右上角有一个特殊的图标。



- **节点(Node)：**运行时存在的物理元素，它表示一个计算机资源，它通常至少有一些记忆能力处理能力。一组组件可以驻留在一个节点内，也可以从一个节点迁移到另一个节点。在图形上，把节点画成一个立方体，通常在立方体中只写它的名称

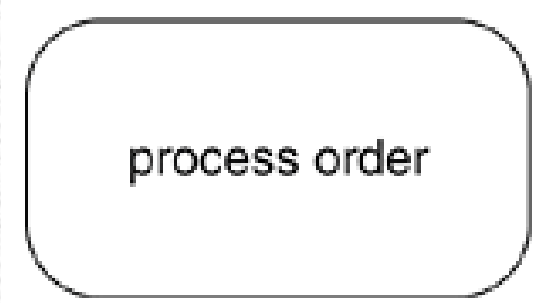


- 动作事物：UML模型的动态部分。它们是模型中的动词，描述了跨越时间和空间的行为。共有三类主要的行为事物：

- 交互(Interaction)：是一种行为，它由在特定语境中共同完成一定特定任务的一组对象之间交换的消息组成。一个对象群体的行为或单个操作的行为可用一个交互来描述。Interaction涉及一些其它元素，包括消息、动作和连接件（对象间的连接）。在图形上，把消息画成一条有方向的直线，通常在其上总是带有操作名。

- **状态机(State Machine)：**它描述了一个对象或一个交互在生命期内相应时间所经历的状态序列。单个类或一组类之间协作的行为可以用状态机来描述。一个状态机涉及到一些其它元素，包括状态、转移（从一个状态到另一个状态的流）、事件（触发转换的事物）和活动（对一个转换的响应）。图形上表示成圆角矩形

- **活动(Activity)：**它描述了计算过程执行的步骤序列。活动的一个步骤称为一个动作。在图形上，把动作画成一个圆角矩形，在其中含有指明其用途的名字。状态和动作靠不同的语境加以区别。



■ 交互、状态机、活动的关系

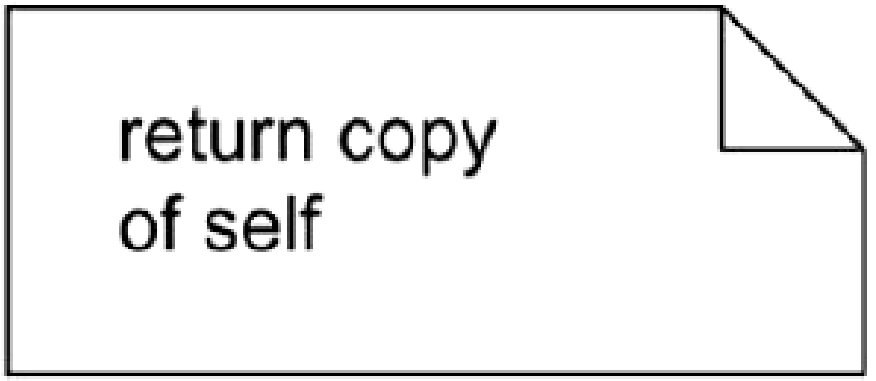
- 相同
  - 交互、状态机和活动都是可以包含在UML模型中的基本行为事物
  - 在语义上，这些元素通常与各种结构元素相关
- 不同
  - 交互注重的是一系列相互作用的对象
  - 状态机注重的是一定时间内一个对象的生命周期
  - 活动注重的是步骤之间的流而不关心哪个对象执行那个步骤

- **分组事物：UML模型的组织部分，UML唯一的分组事物是包（Package）。**

包是把元素组织成组的机制。在包的基础上，有一些构造型，如框架(frameworks)、子系统(subsystems)，它们也是一种组织形式。

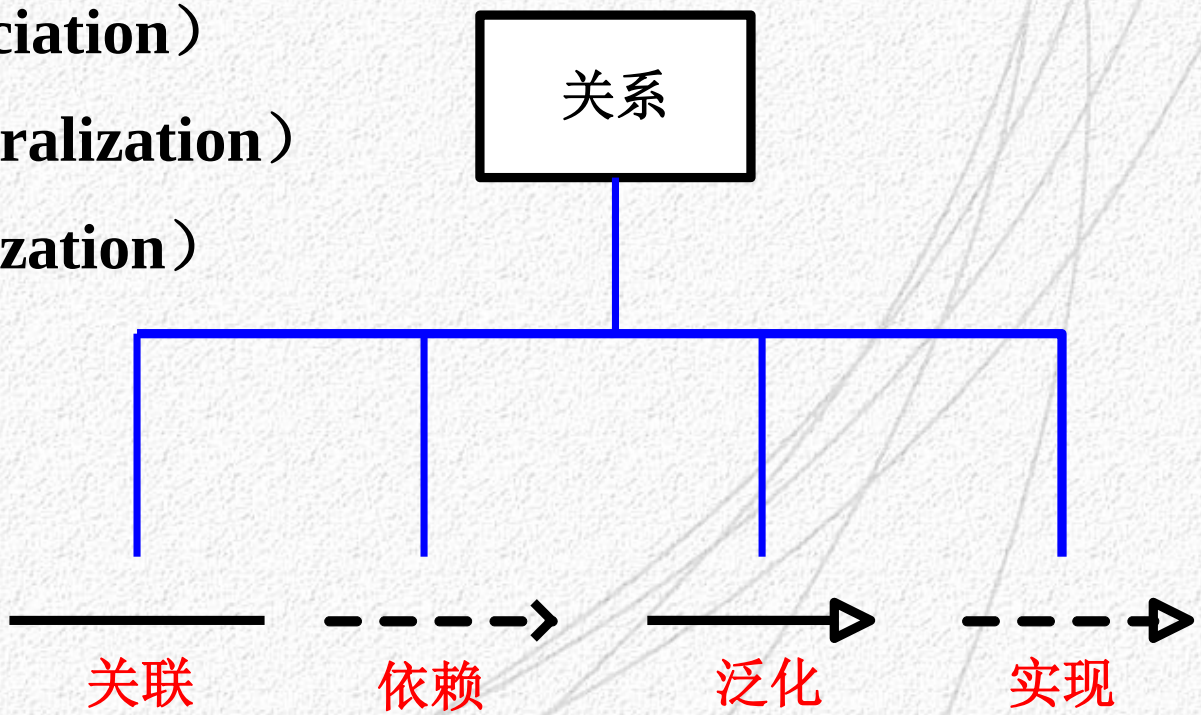
- 包是UML中唯一的分组事物。
- 包可以拥有其它元素，这些元素可以是类、接口、组件、节点、协作、用例和图，甚至可以是其他包
- 一个包形成了一个命名空间，在一个包中同一种元素的名称必须是唯一的。不同种类的元素可以有相同的名称。
- 包与组件的一点区别：
  - 包是一种概念上的组织方式，存在于系统开发过程中
  - 组件是物理上的，存在于运行过程中。

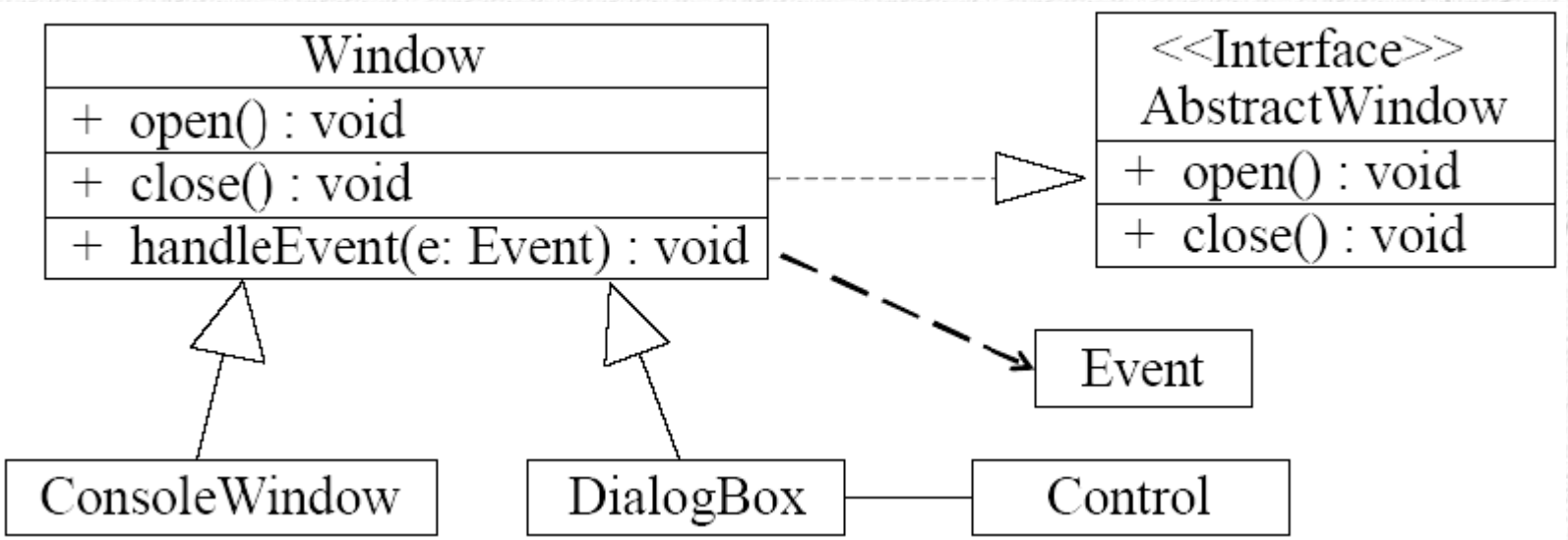
- **注释事物：UML模型的解释部分。这些注释事物用来描述、说明和标注模型的人和元素。有一种主要的注释事物，称为注解（note）。**
  - 注解（note）是一个依附于一个元素或一组元素之上，对它进行约束或解释的简单符号。



UML构造块（关系）

- 在UML中有4种关系
  - 依赖（Dependency）
  - 关联（Association）
  - 泛化（Generalization）
  - 实现（Realization）




Window
+ open() : void
+ close() : void
+ handleEvent(e: Event) : void
<<Interface>>
AbstractWindow
+ open() : void
+ close() : void
Event
ConsoleWindow
DialogBox
Control

- 依赖：两个事物间的语义关系，其中一个事物（独立事物）发生变化会影响另一个事物（依赖事物）的语义。在图形上，把依赖画成一条可能有方向的虚线，偶尔在其上还带有一个标记

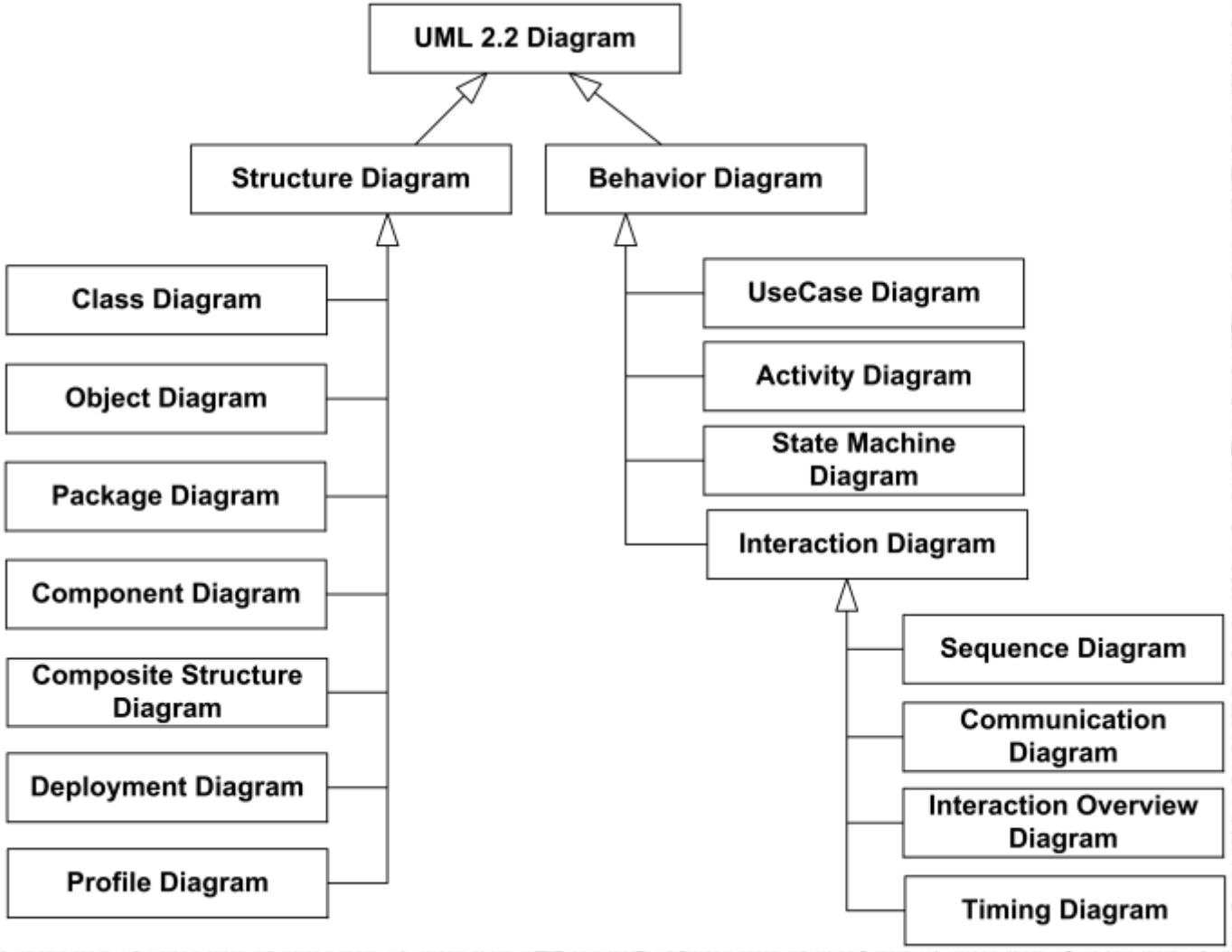
- 关联：一种结构关系，它描述了一组链，链是对象之间的连接。聚合是一种特殊类型的关联，它描述了整体和部分间的结构关系。在图形上，把关联画成一条实线，可能有方向，偶尔还带有一个标记，还经常含有诸如多重性和端名等修饰。

- 泛化：一种特殊/一般关系，特殊元素（子元素）的对象可替代一般元素（父元素）的对象。用这种方法，子元素共享了父元素的结构和行为。在图形上，泛化关系画成一条带空心箭头的实现，箭头指向父元素

- 实现：类元之间的语义关系，其中的一个类元指定了由另一个类元保证执行的契约。
- 在两种地方要遇到实现关系：
  - 在接口和实现它们的类或构件之间
  - 在用例和实现它们的协作之间

在图形上，实现关系画成一条带有空心箭头的虚线

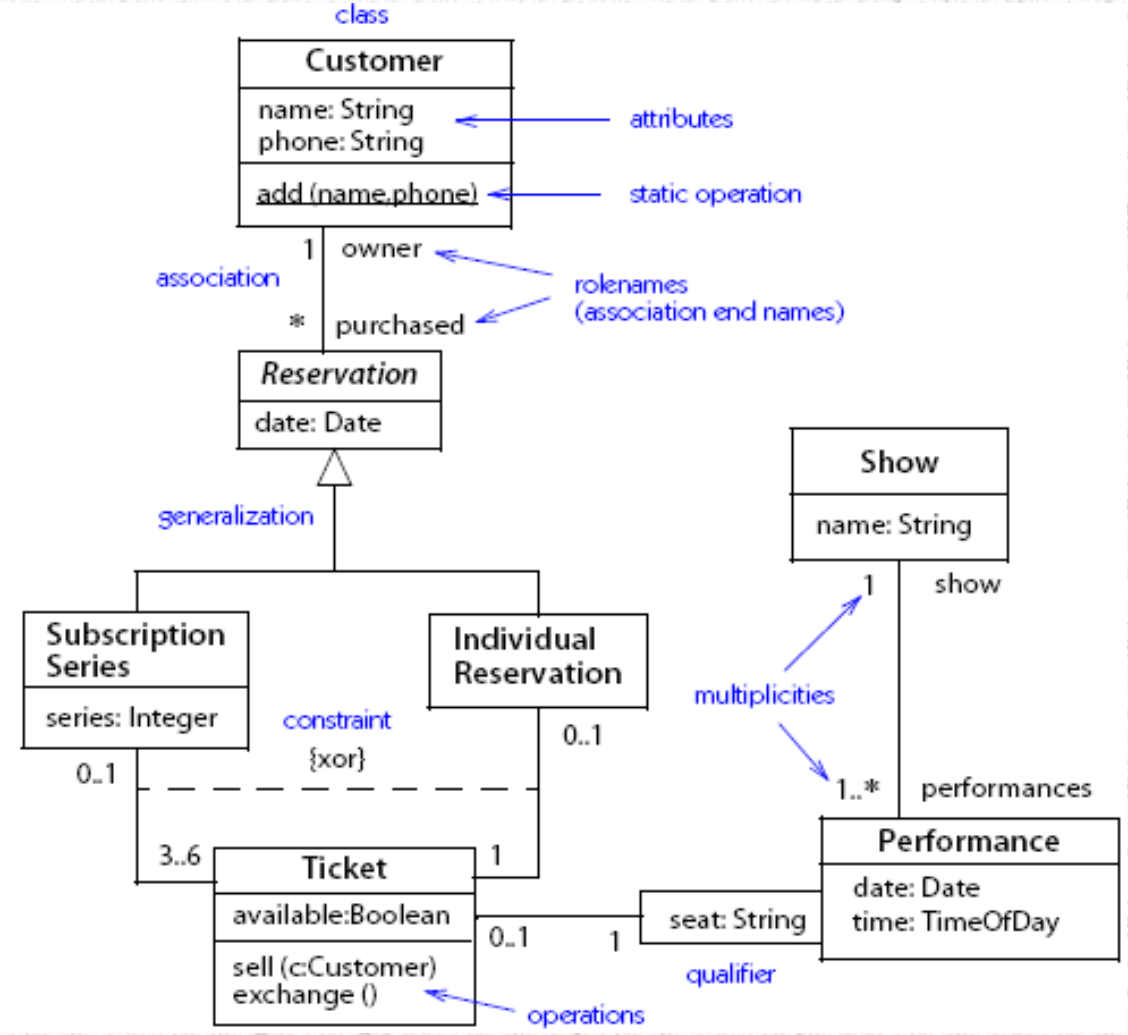
UML构造块（图）


UML 2.2 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Component Diagram
Composite Structure Diagram
Deployment Diagram
Profile Diagram
UseCase Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

■ 类图（Class Diagrams）

- 展现了一组对象、接口、协作和它们之间的关系
- 类图给出系统的静态设计视图。包含活动类的类图给出系统的静态进程视图。
- 组件图是类图的变体

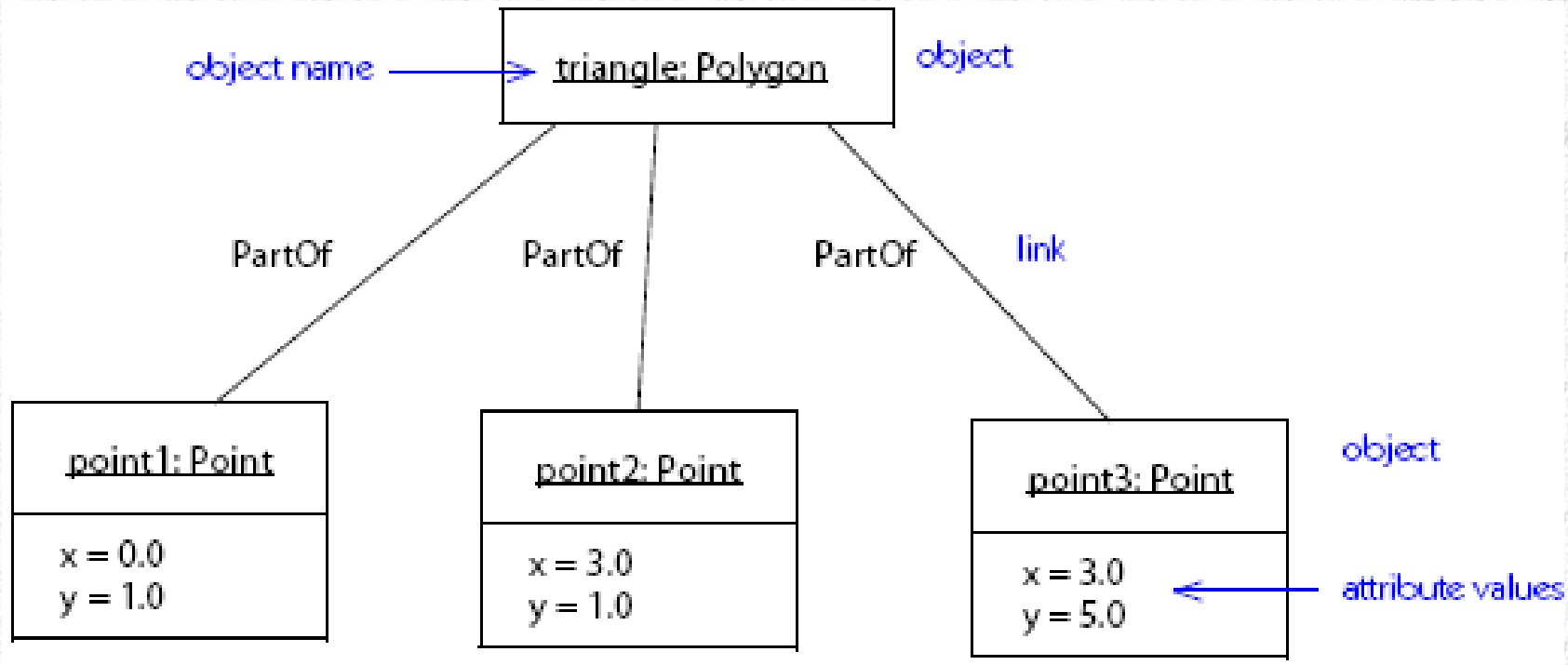
■ 类图（示例）


class
Customer
name: String
phone: String
add (name,phone)
attributes
static operation
1
owner
association
*
purchased
rolenames
(association end names)
Reservation
date: Date
generalization
Show
name: String
1
show
Subscription Series
series: Integer
0..1
constraint
{xor}
Individual Reservation
0..1
multiplicities
1..*
performances
3..6
Ticket
available:Boolean
sell (c:Customer)
exchange ()
1
0..1
1
seat: String
qualifier
Performance
date: Date
time: TimeOfDay
operations

■ 对象图（Object Diagrams）

- 展现了一组对象以及它们之间的关系
- 描述了在类图中所建立的事物的实例的静态快照
- 给出系统的静态设计视图或静态进程视图，但它们是从真实案例或原型案例的角度建立的

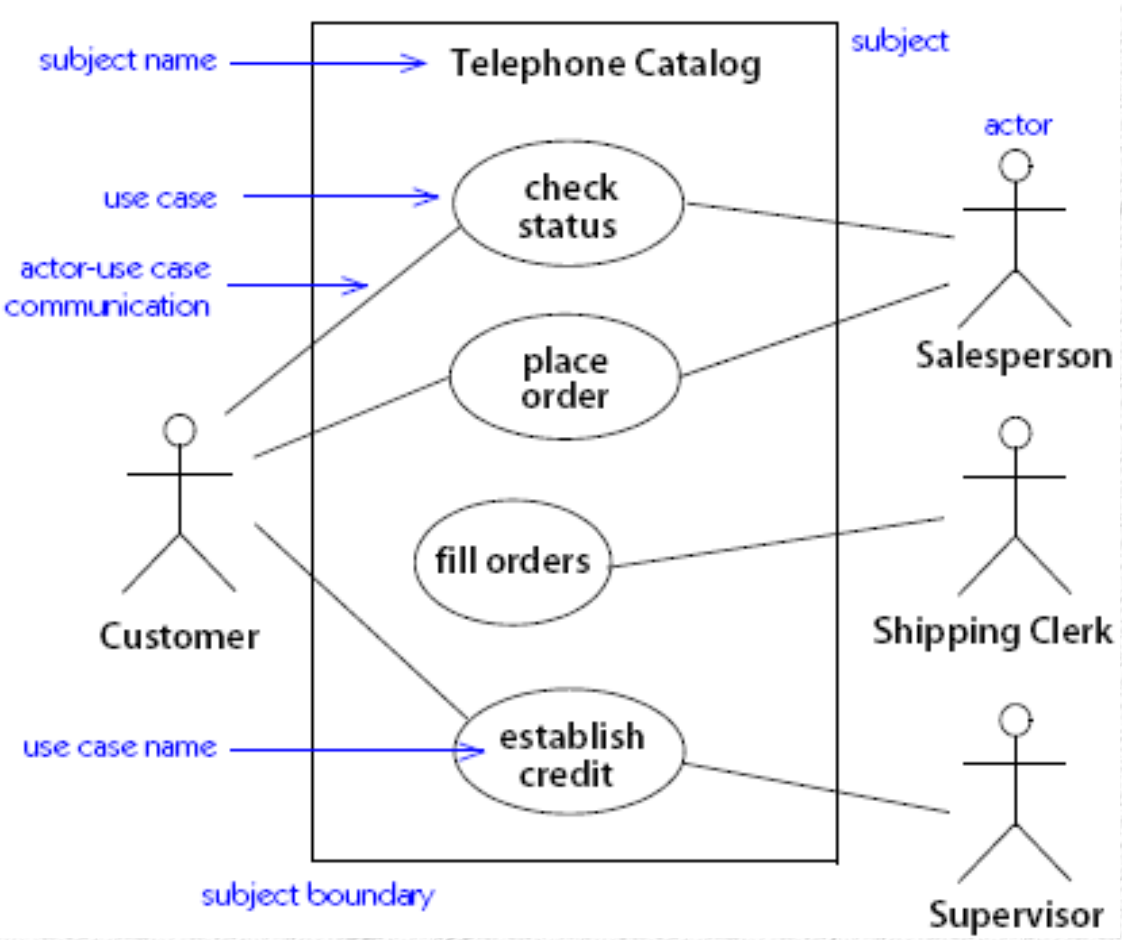
■ 对象图（示例）


object name
triangle: Polygon
object
PartOf
PartOf
PartOf
link
point1: Point
x = 0.0
y = 1.0
point2: Point
x = 3.0
y = 1.0
point3: Point
x = 3.0
y = 5.0
object
attribute values

■ 用例图（Use-Case Diagrams）

- 展现了一组用例、参与者（一种特殊的类）及它们之间的关系
- 给出系统的静态用况视图
- 对系统的行为进行组织和建模

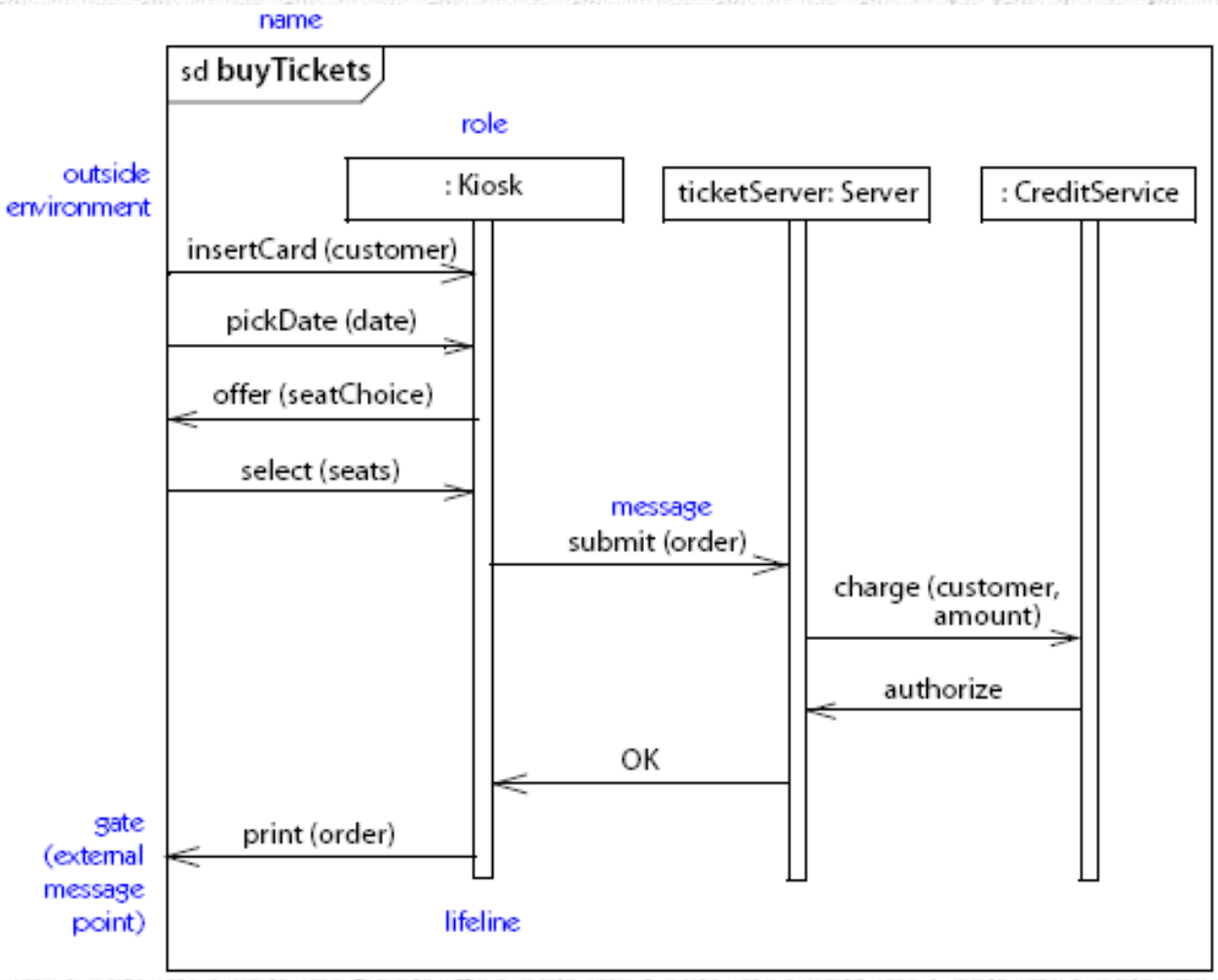
■ 用例图（示例）


subject name
Telephone Catalog
subject
actor
use case
check status
actor-use case communication
place order
Salesperson
Customer
fill orders
Shipping Clerk
use case name
establish credit
subject boundary
Supervisor

■ 顺序图（Sequence Diagrams）

- 强调消息的时间顺序
- 显示了一个场景中每个参与对象的控制焦点

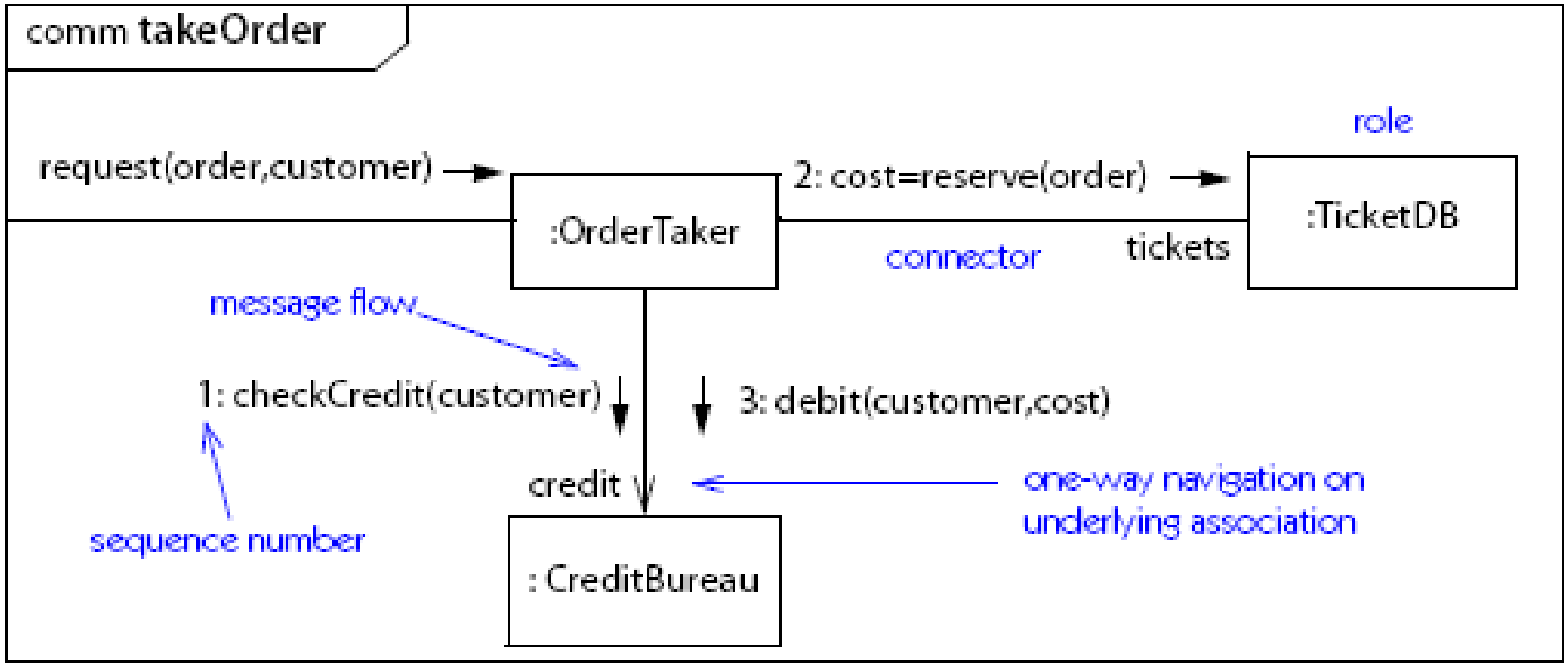
■ 顺序图（示例）


name
sd buyTickets
role
outside environment
: Kiosk
ticketServer: Server
: CreditService
insertCard (customer)
pickDate (date)
offer (seatChoice)
select (seats)
message
submit (order)
charge (customer, amount)
authorize
OK
print (order)
gate (external message point)
lifeline

■ 通信图（Communication Diagrams）

- 显示了相互交互的对象的结构组织
- 顺序图和通信图是同构的，它们可以相互转换
- 顺序图和通信图都是交互图
  - 交互图展现了一种交互，它由一组对象或角色以及它们之间可能发送的消息构成
  - 顺序图强调时序
  - 通信图强调消息流经的数据结构
  - 定时图展现了消息交换的实际时间

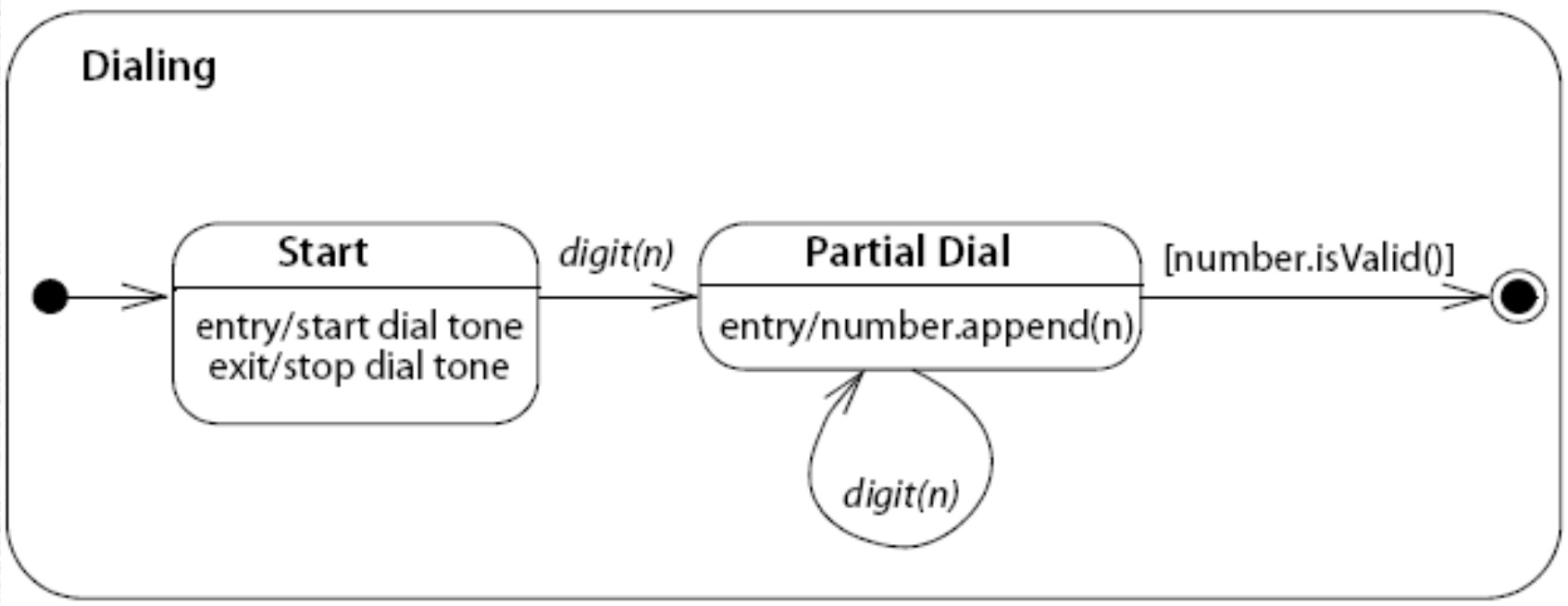
■ 通信图（示例）


comm takeOrder
role
request(order,customer)
2: cost=reserve(order)
:OrderTaker
:TicketDB
connector
tickets
message flow
1: checkCredit(customer)
3: debit(customer,cost)
credit
one-way navigation on
underlying association
sequence number
: CreditBureau

■ 状态图（State Diagrams）

- 展现了一个状态机，它由状态、转换、事件和活动组成
- 状态图专注于系统的动态视图
- 对于接口、类或协作的行为建模尤为重要，而且它强调对象行为的事件顺序，这非常有助于对反应式系统建模

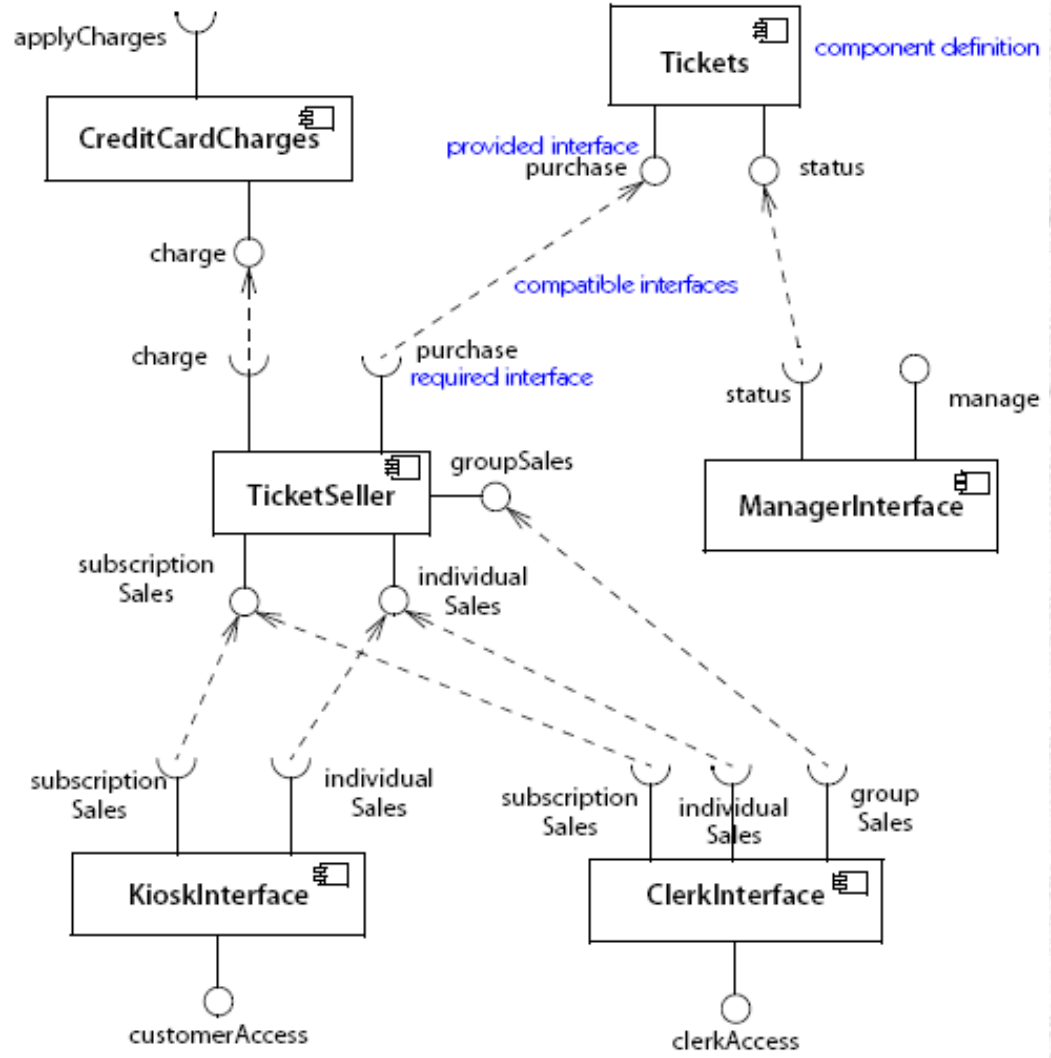
■ 状态图（示例）


Dialing
Start
entry/start dial tone
exit/stop dial tone
digit(n)
Partial Dial
entry/number.append(n)
[number.isValid()]
digit(n)

■ 组件图（Component Diagrams）

- 展现了一个封装的类和它的接口、端口以及由内嵌的组件和连接件构成的内部结构。
- 专注于系统的静态设计实现视图
- 对于由小的部件组建大的系统来说，组件图非常重要
- UML将组件图和适用于任意类的组合结构图区分开来

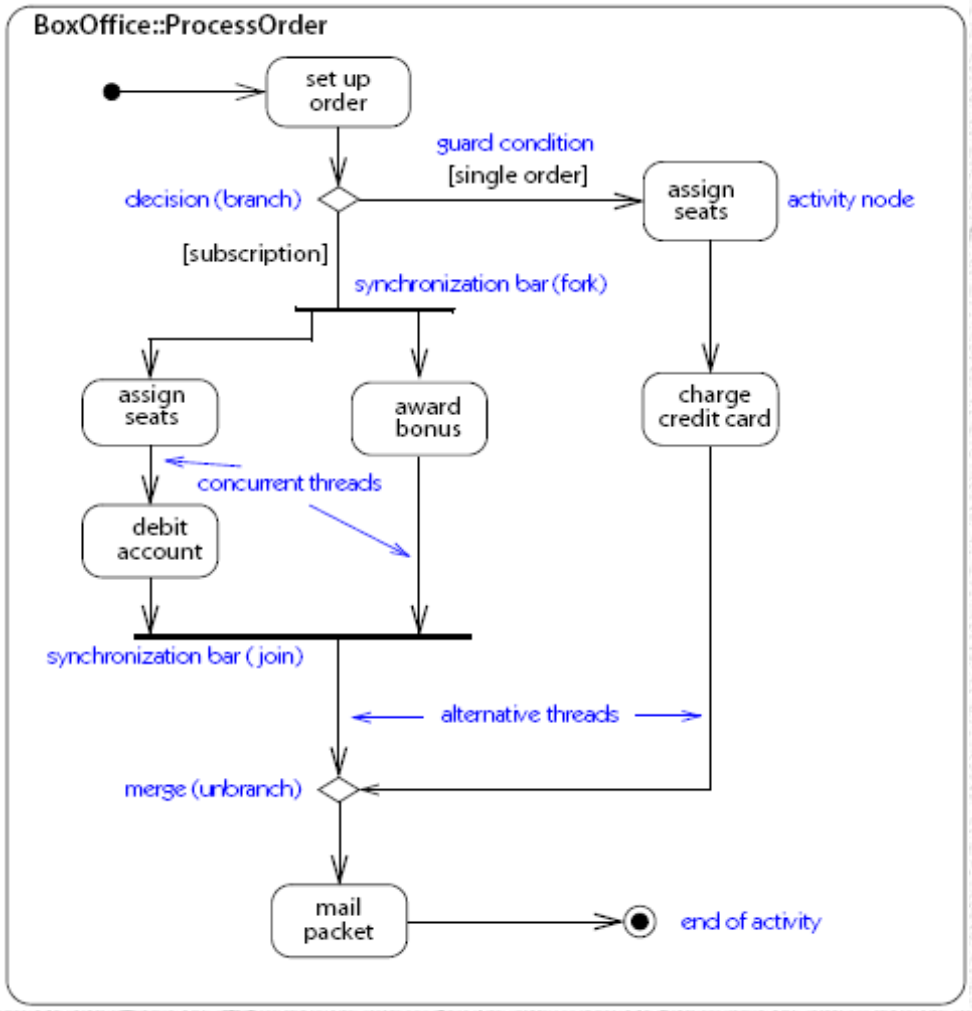
■ 组件图（示例）


applyCharges
CreditCardCharges
charge
charge
Tickets
component definition
provided interface
purchase
status
compatible interfaces
purchase
required interface
status
manage
groupSales
TicketSeller
ManagerInterface
subscription Sales
individual Sales
subscription Sales
individual Sales
subscription Sales
individual Sales
group Sales
KioskInterface
ClerkInterface
customerAccess
clerkAccess

■ 活动图（Activity Diagrams）

- 将进程或其他计算的结构展示为计算内部一步步的控制流和数据流
- 活动图专注于系统的动态视图
- 对于系统的功能建模特别重要，并强调对象间的控制流程

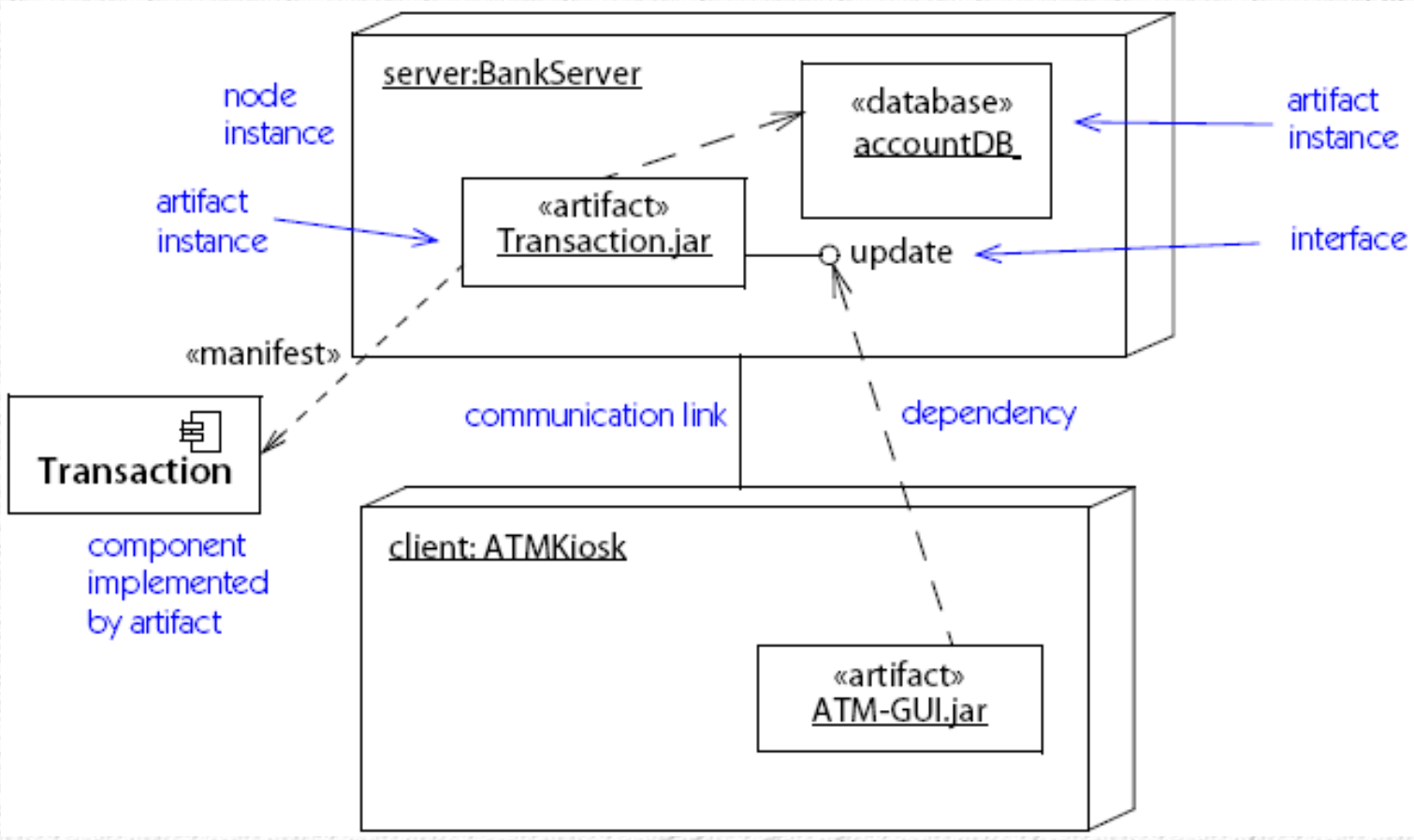
■ 活动图（示例）


BoxOffice::ProcessOrder
set up order
guard condition
[single order]
decision (branch)
assign seats
activity node
[subscription]
synchronization bar (fork)
assign seats
award bonus
charge credit card
concurrent threads
debit account
synchronization bar (join)
alternative threads
merge (unbranch)
mail packet
end of activity

■ 部署图（Deployment Diagram）

- 展现了对运行时处理节点以及其中的组件的配置
- 部署图给出了体系结构的静态部署视图
- 通常一个结点包含一个或多个制品

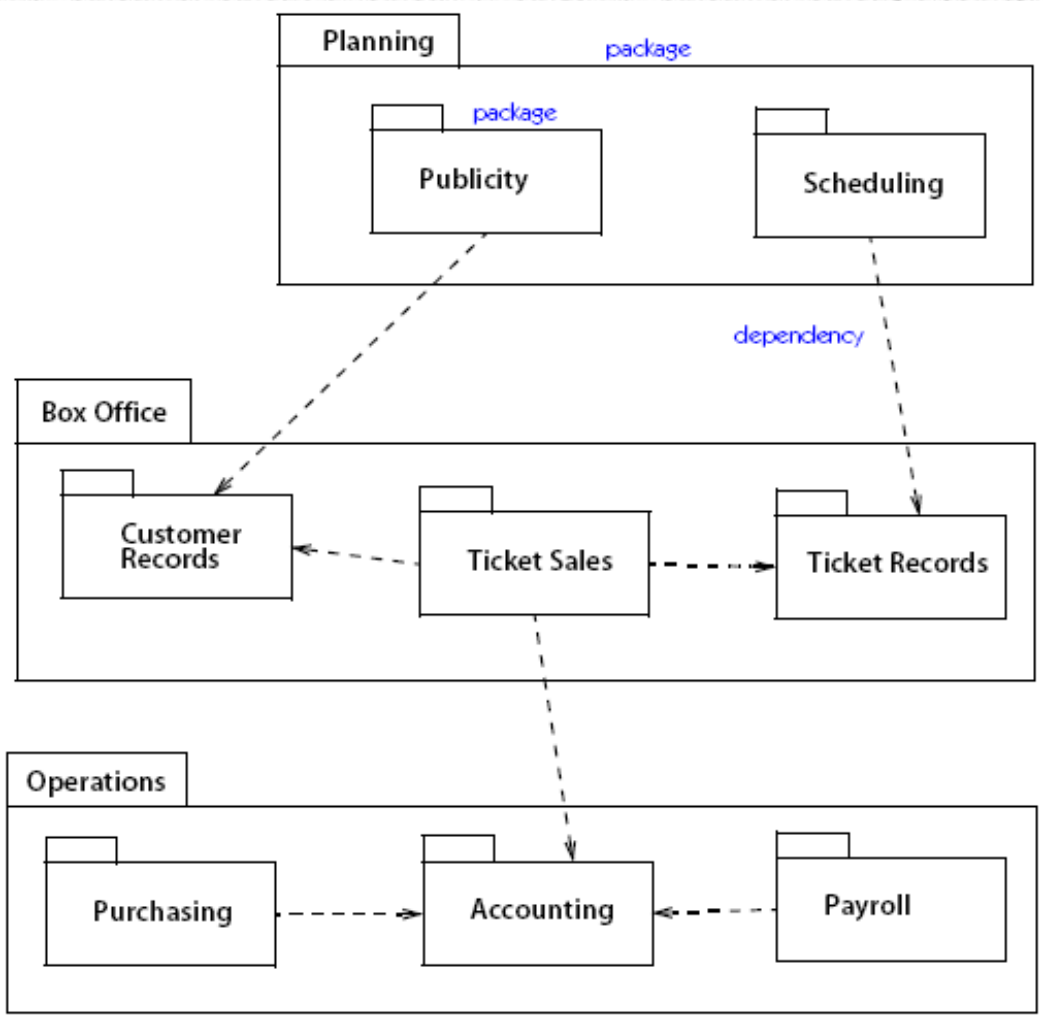
■ 部署图（示例）


server:BankServer
node instance
«database» accountDB
artifact instance
artifact instance
«artifact» Transaction.jar
update
interface
«manifest»
communication link
dependency
Transaction
component implemented by artifact
client: ATMKiosk
«artifact» ATM-GUI.jar

■ 包图（Package diagram）

- 展现了由模型本身分解而成的组织单元以及它们的依赖关系

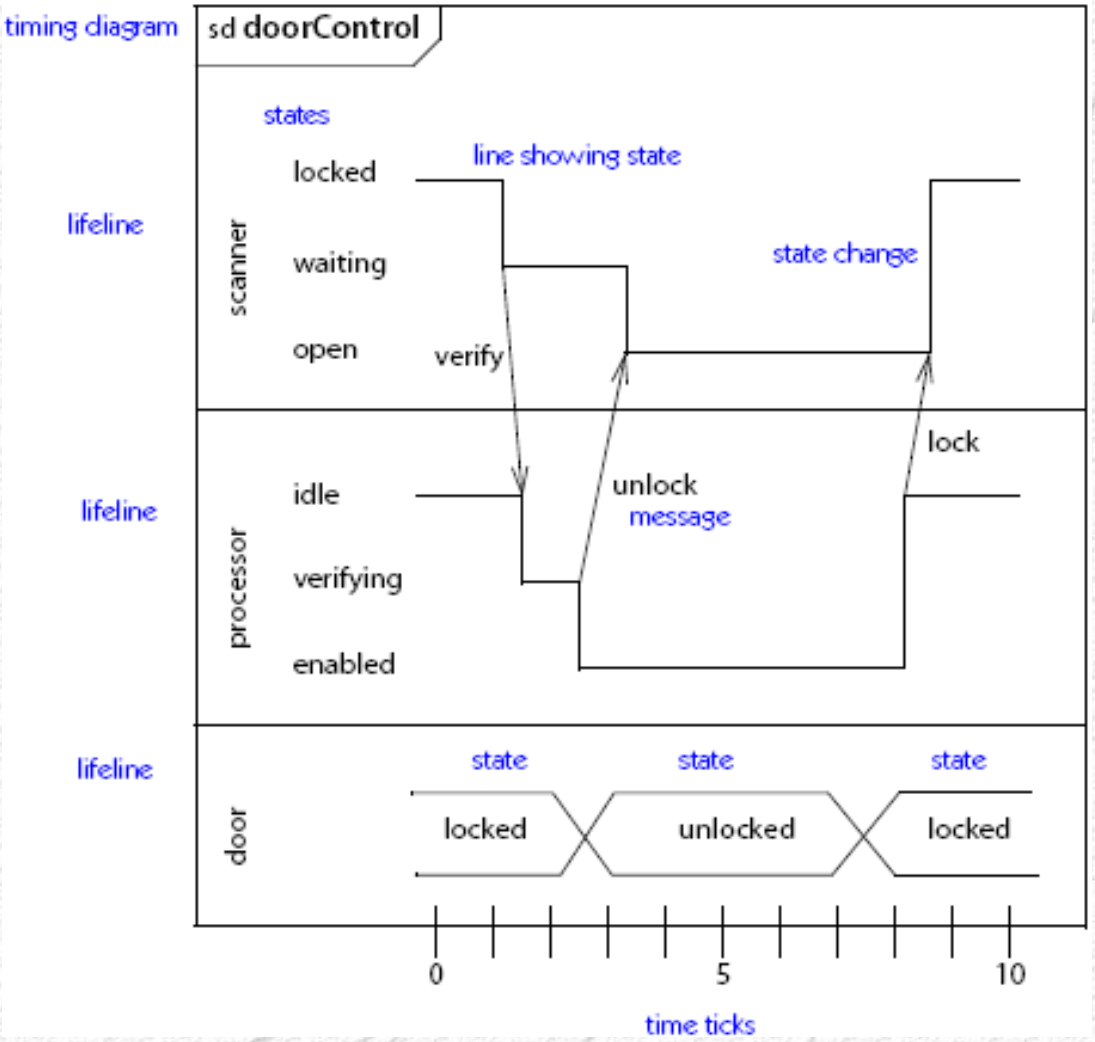
■ 包图（示例）


Planning
package
package
Publicity
Scheduling
dependency
Box Office
Customer Records
Ticket Sales
Ticket Records
Operations
Purchasing
Accounting
Payroll

■ 定时图（timing diagram）

- 展现了消息跨越不同对象或角色的实际时间，而不仅仅是关心消息的相对顺序

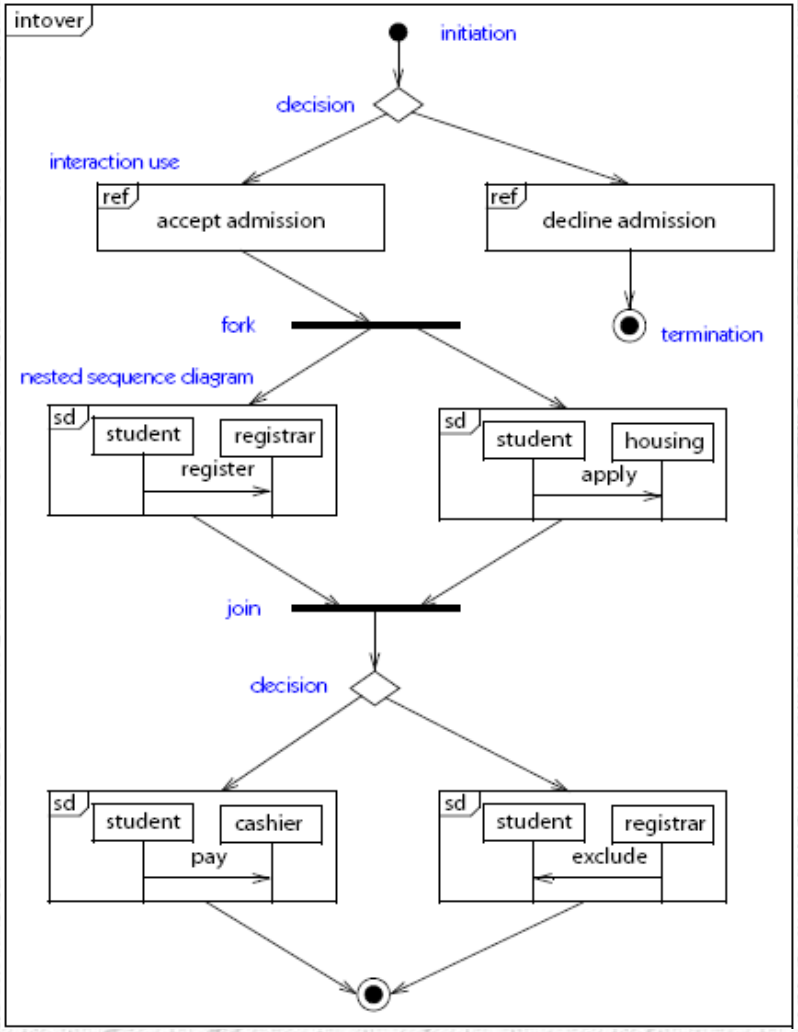
■ 定时图（示例）


timing diagram
sd doorControl
states
line showing state
lifeline
locked
waiting
open
scanner
verify
state change
lock
lifeline
idle
verifying
enabled
processor
unlock
message
lifeline
state
state
state
door
locked
unlocked
locked
0
5
10
time ticks

■ 交互概览图（Interaction overview diagram）

- 是活动图和顺序图的混合物

■ 交互概览图（示例）


intover
initiation
decision
interaction use
ref
accept admission
ref
decline admission
fork
termination
nested sequence diagram
sd
student
registrar
register
sd
student
housing
apply
join
decision
sd
student
cashier
pay
sd
student
registrar
exclude

- 组合结构图：是一种高级视图，其显示了如下内容：
  - 内部结构： 显示包含在类里的各个成员，以及各个成员之间的关系。
  - 如何使用类： 显示类如何通过端口作用于系统。
  - 合作： 显示系统中一组对象共同协作完成某件事。

■ 组合结构图（示例）

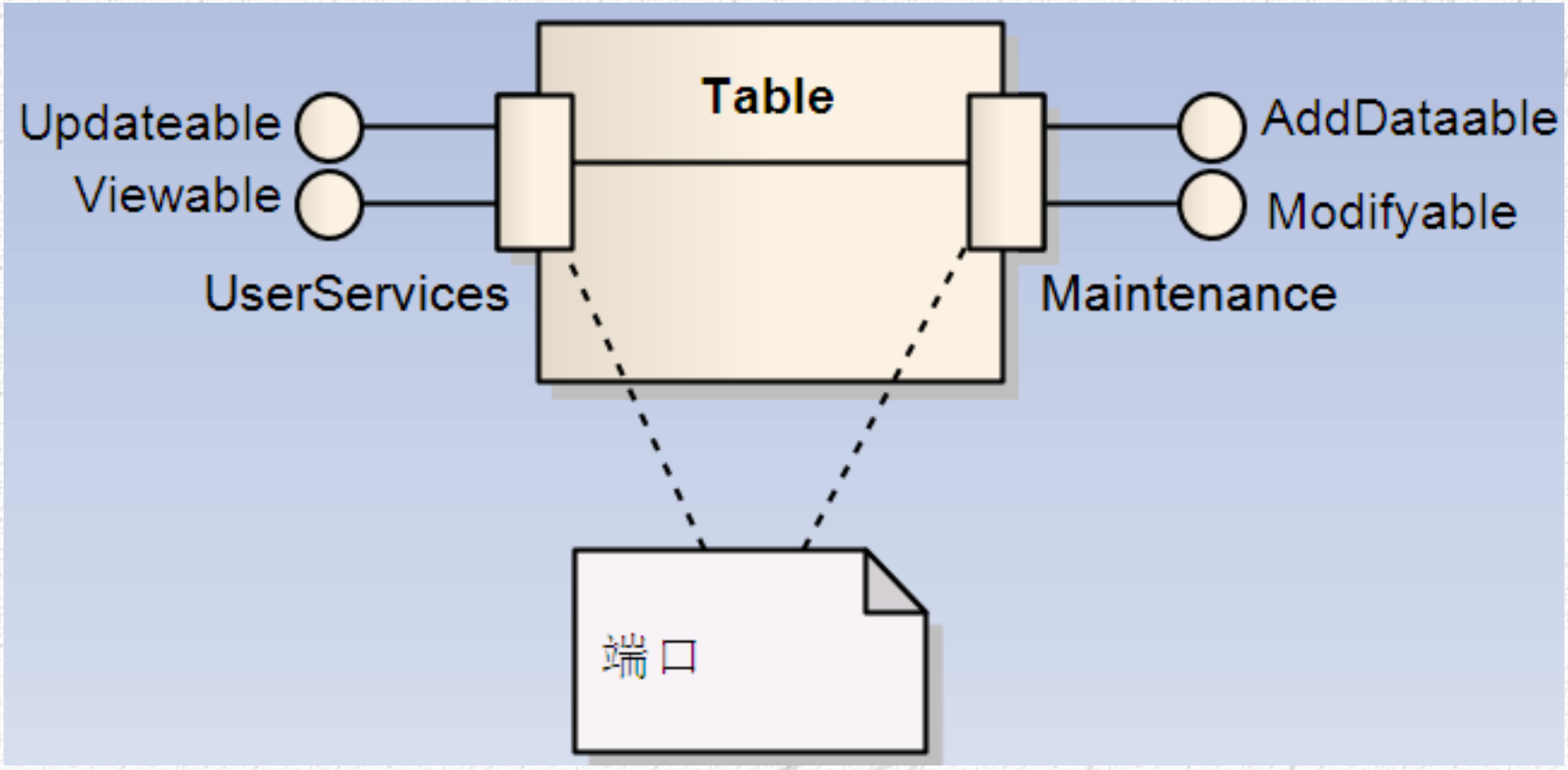


Table
Updateable
Viewable
UserServices
AddDataable
Modifyable
Maintenance
端口

通用机制

- UML中有四种贯穿整个语言且一致应用的通用机制
  - 详述（Specification）
  - 修饰（Adornment）
  - 通用划分（Common Division）
  - 扩展机制（Extensibility Mechanism）

■ 详述（Specification）

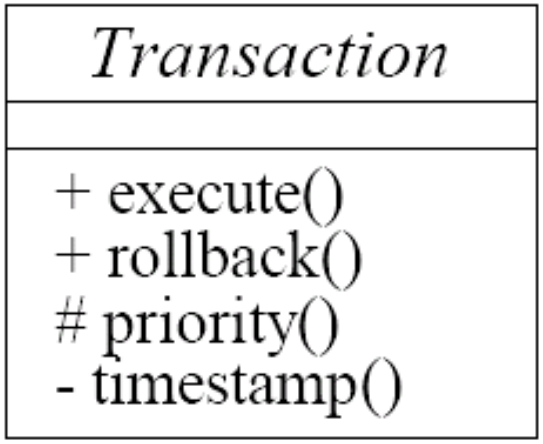
- UML不只是一种图形语言。实际上，在它的图形表示法的每部分背后都有一个详述，这个详述提供了对构造块的语法和语言的文字叙述
- UML的图形表示法用来对系统进行可视化；UML的详述用来描述系统的细节
- UML的详述提供了一个语义底版，它包含了一个系统的各模型的所有部分，并且各部分相互联系，并保持一致。UML图只不过是对底版的简单视觉 投影，每一个图展现了系统的一个特定方面

■ 修饰（Adornment）

- UML表示法中的每一个元素都有一个基本符号，可以把各种修饰细节加到这个符号上
- 修饰是帮助我们理解系统中事物的简单辅助标记

■ 修饰（举例）

对类的详述可以包含其他细节，例如，它是否是抽象类，或它的属性和操作是否可见。可以把很多这样的细节表示为图形或文字修饰。



■ 通用划分（Common Division）

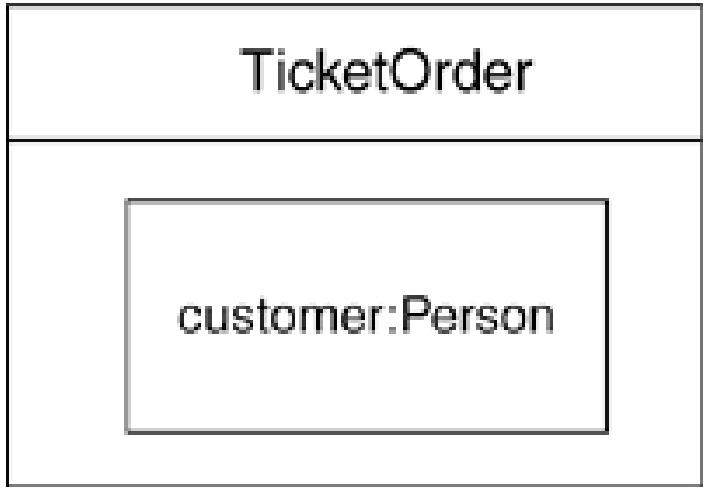
构建面向对象系统导致将现实世界划分为不同的阶段，一种划分涉及类和对象：

- 类和类的实例
- 用况和用况执行
- 构件和构件的实例

■ 另一种划分涉及接口和实现(Implementation)分离，实现实现了接口指定的契约。

- 用况和实现它们的协作
- 操作和实现它们的方法

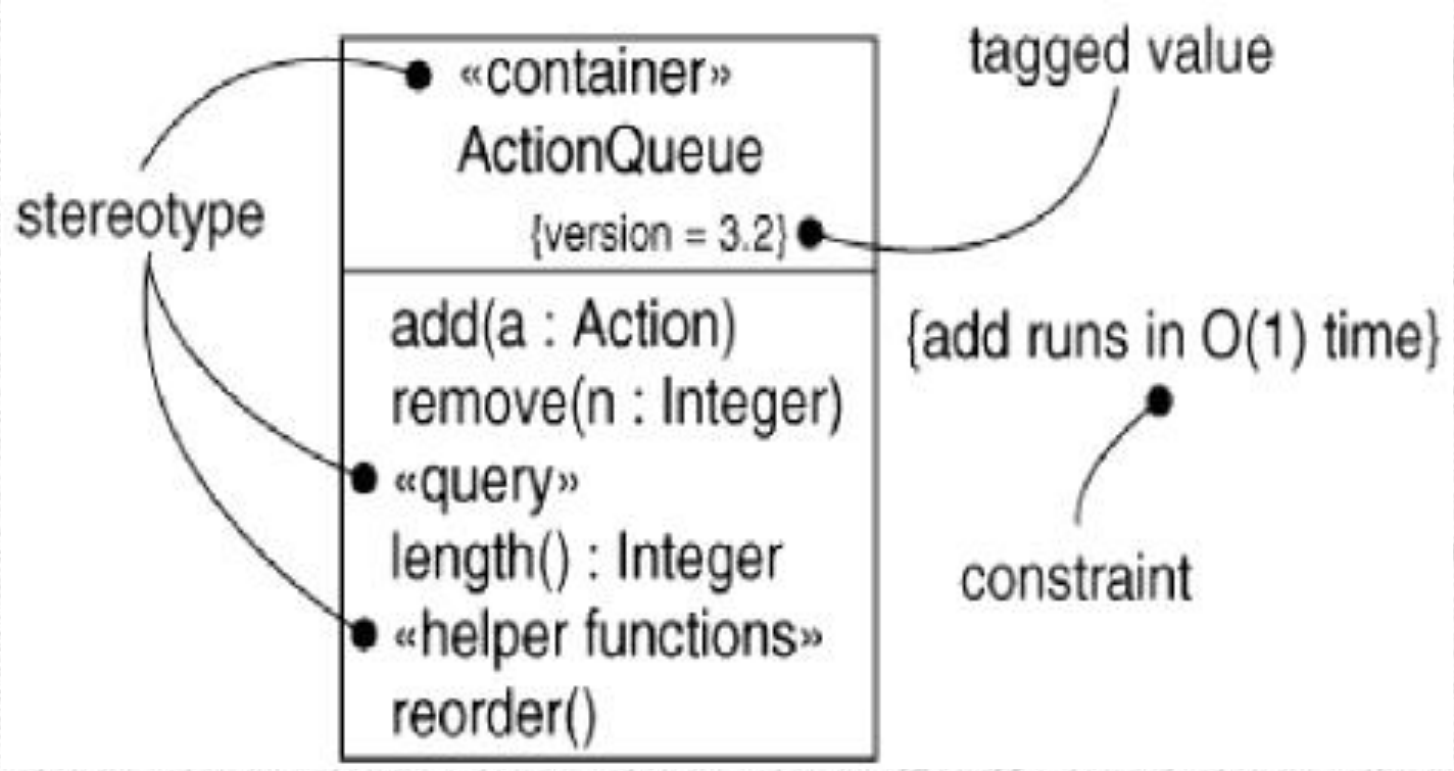
- 还有一种划分位于类型和角色的分离
  - 类型声明了尸体的种类（如对象、属性或参数）
  - 角色描述了实体在语境中的含义（如类、构件或协作等）。
  - 任何作为其他实体结构中的一部分的实体（如属性）都具有两个特征：
    - 从它固有的类型派生出一些含义
    - 从它在语境中的角色派生出一些含义



■ 扩展机制（Extensibility Mechanism）

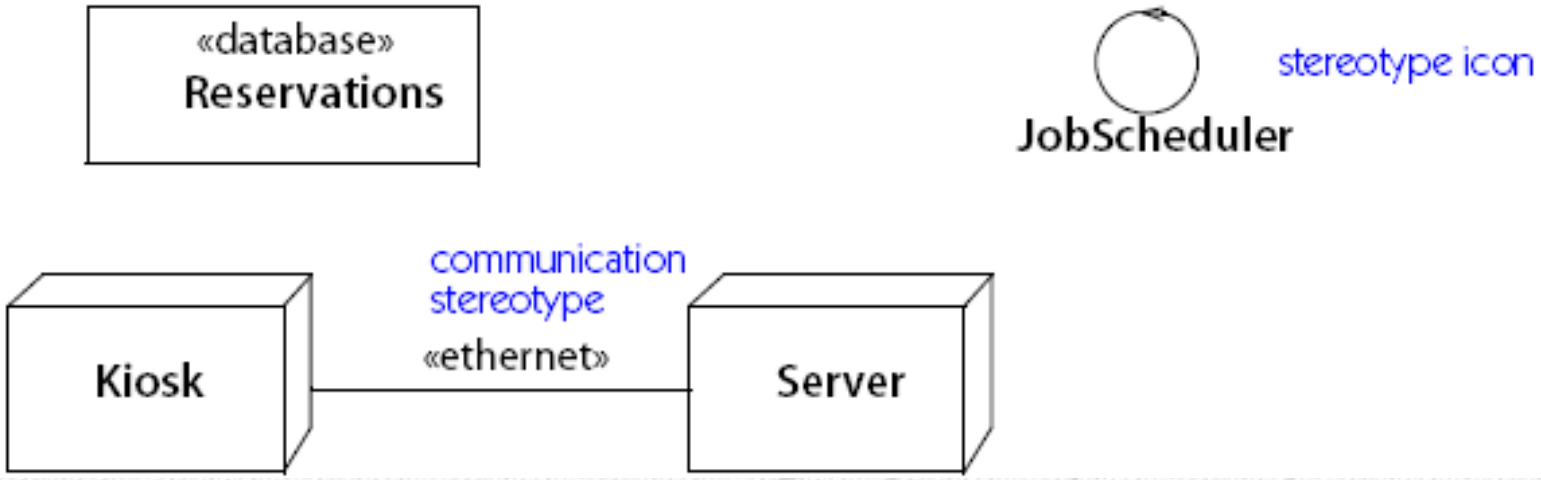
- UML提供了一种绘制软件蓝图的标准语言，但是一种闭合的语言即使表达能力再丰富，也难以表示出各种领域中各种模型在不同时刻所有可能的细微差别。
- UML是可扩展的，可以以受控的方式扩展该语言
- UML的扩展机制包括：
  - 衍型（Stereotype）
  - 标记值（Tagged value）
  - 约束（Constraint）

■ 扩展机制（示例）


«container»
ActionQueue
{version = 3.2}
tagged value
stereotype
add(a : Action)
remove(n : Integer)
«query»
length() : Integer
«helper functions»
reorder()
{add runs in O(1) time}
constraint

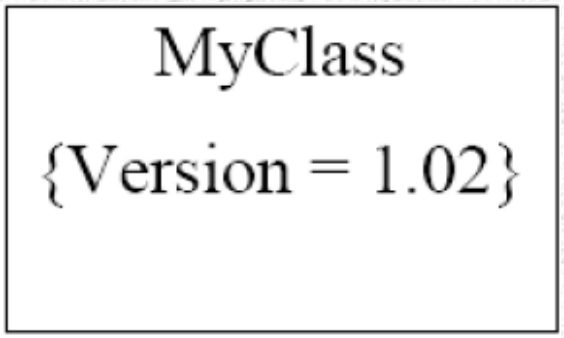
➢ 衍型（Stereotype）

- ◆衍型扩展了UML的词汇，它允许你创造新的构造块，这个新构造块既可从现有的构造块派生，又专门针对你要解决的问题
- ◆UML预定义了部分构造型，用户也可以自己定义



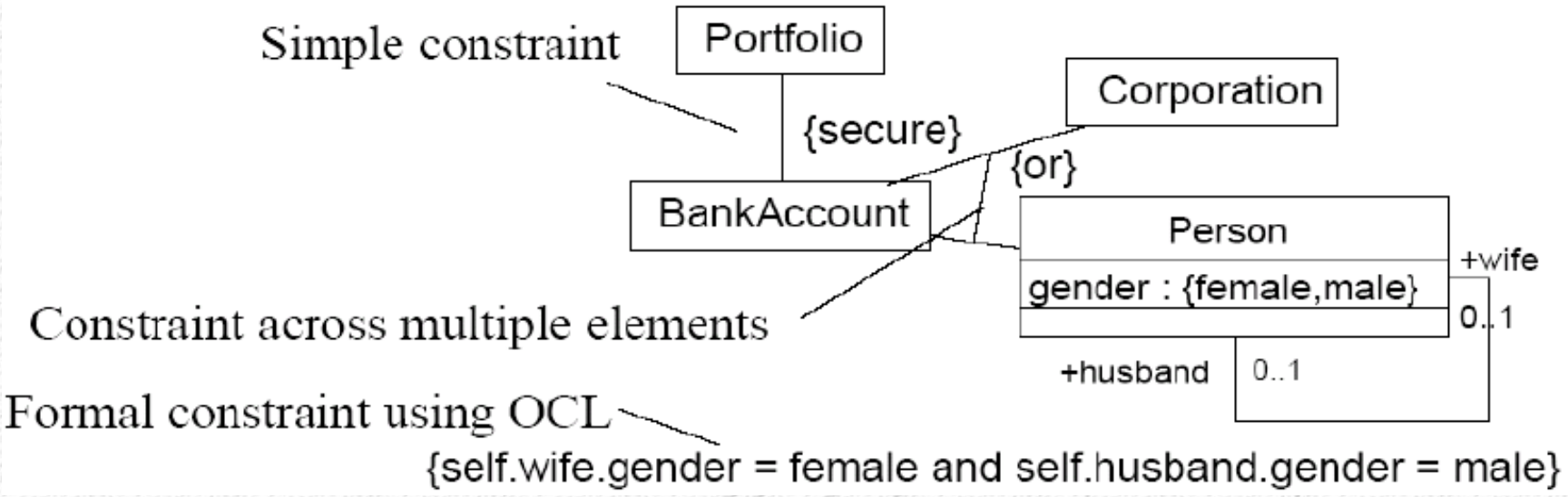
标记值

- 标记值扩展了UML衍型的特性，允许创建详述元素的新信息
- 每个标记值由关键字和值的配对组成（keyword-value pair）。标记值可以附加到模型中的任何元素上。标记值中的关键字称为Tag。
- 常见的标记值示例有：
  - {Author=Dave,ROn}
  - {version Number=3}
  - {Location=d:\java\uml\examples}
  - {Location=Node:Middle Tier}



约束

- 约束扩展了UML扩展块的语义，它允许增加新的规则或修改现有的规则
- 约束可以写成自由格式的文本，但是，如果希望把语义写得更精确，你可以采用UML的对象约束语言（OCL：Object Constraint Language）

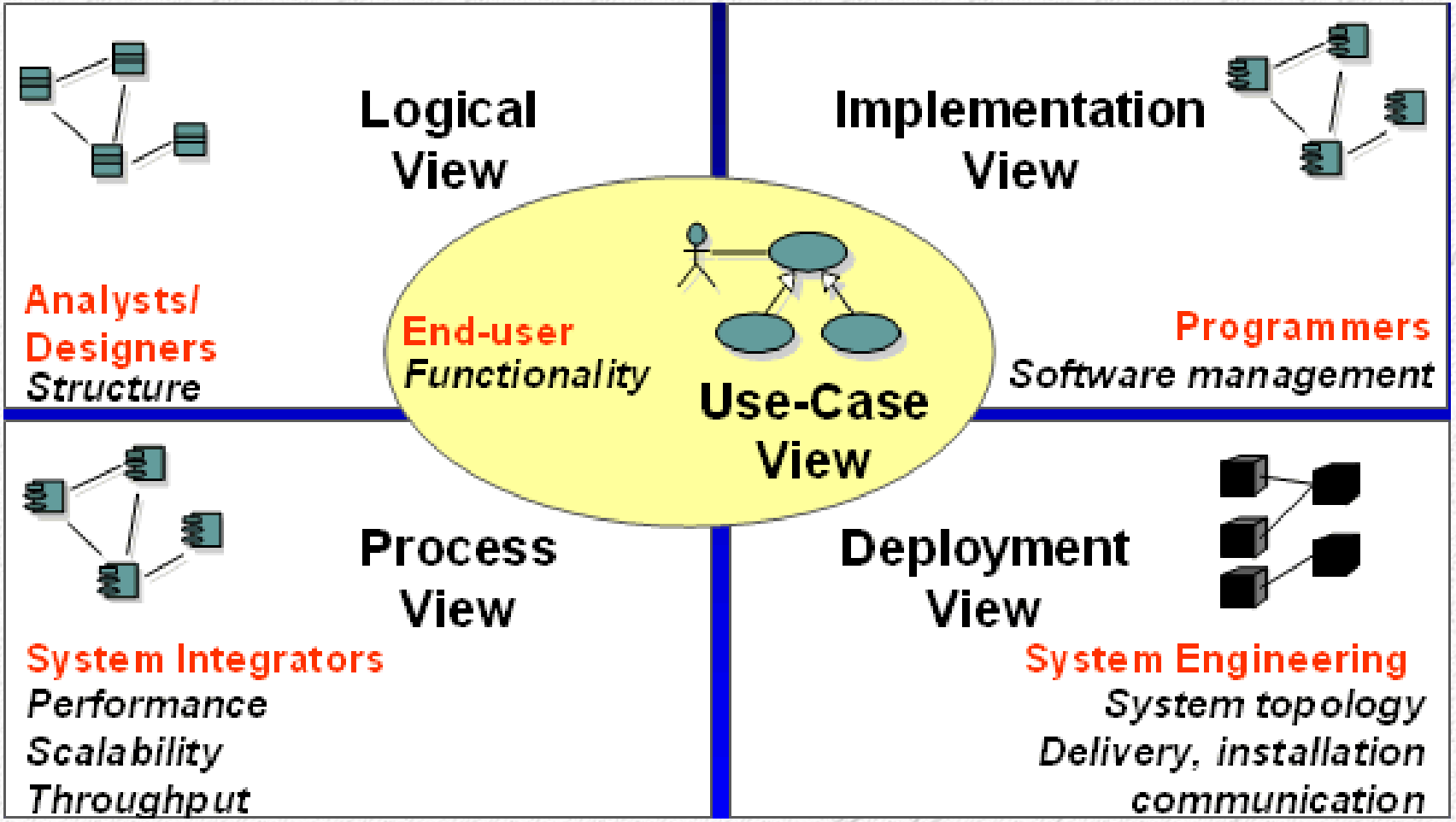


UML规则

- 帮助UML的构造块指定一个结构良好的模型
- 一个模型如果语义上自相一致而且与其它相关模型协调一致，那么它是结构良好的。
- UML包括了如下的语义规则
  - 命名（Name）：为事物、关系和图起名
  - 范围（Scope）：给一个名称以特定含义的语境
  - 可见性（Visibility）：如何让其他成分看见和使用
  - 完整性（Integrity）：事物如何正确、一致地相互关系
  - 执行（Execution）：运行或模拟动态模型的含义是什么

- 在软件密集型系统的开发期间所建造的模型往往需要发展变化，并可以由许多人员以不同的方式、在不同的时间进行观察。因而开发组不仅要建造一些结构良好的模型，也要建造一些这样的模型
  - 省略：隐藏某些元素以简化视图
  - 不完全性：可以遗漏某些的元素
  - 不一致性：不保证模型的完整性
- UML的规则鼓励你专注于最重要的分析、设计和实现问题，而这些问题将促使模型随着时间的推移而逐渐具备良好的结构。

UML视图


Logical View
Analysts/ Designers
Structure
Implementation View
Programmers
Software management
End-user
Functionality
Use-Case View
Process View
System Integrators
Performance
Scalability
Throughput
Deployment View
System Engineering
System topology
Delivery, installation
communication

■ 用例视图（use case view）

- 由描述可被最终用户、分析人员和测试人员看到的系统行为的用例组成
- 在UML中，该视图的静态方面由用例图表现；动态方面由交互图、状态图和活动图表现

■ 设计视图（design view)

- 包含了类、接口和协作，形成了问题及其解决方案的词汇
- 主要支持系统的功能需求，即系统应该提供给最终用户的服务
- 在UML中，设计视图的静态部分由类图和对象图表现；动态方面由交互图、状态图和活动图表现。类的内部结构图特别有用

■ 交互视图（interaction view）

- 展示了系统的不同部分之间的控制流，包括可能的并发和同步机制
- 主要针对性能、可伸缩性和系统的吞吐量
- 在UML中，静态方面和动态方面的表示和设计视图相同，但着重于控制系统的主动类和在它们之间流动的消息

■ 实现视图（implementation view）

- 包含了用于装配与发布物理系统的制品
- 主要针对系统发布的配置管理，由一些独立的文件组成
- 也关注逻辑的类和构件到物理制品的映射
- 在UML中，该视图的静态方面由组件图表现，动态方面由交互图、状态图和活动图表现

■ 部署视图（deployment view）

- 包含了形成系统硬件拓扑结构的结点
- 主要描述组成物理系统的部件的分布、交付和安装
- 在UML中，该视图的静态部分由部署图表现，动态部分由交互图、状态图和活动图表现

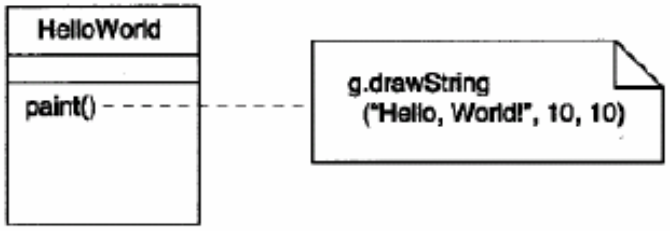
综合示例

■ 世界，你好！

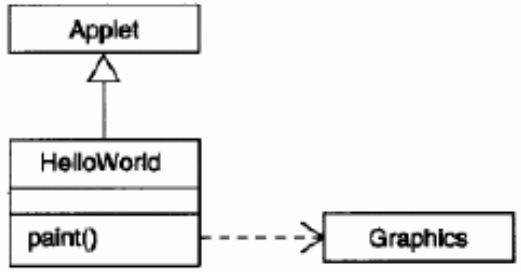
在Web浏览器中，打印“世界，你好！”的Java程序：

```
import java.awt.Graphics;
class HelloWorld extends java.applet.Applet {
  public void paint (Graphics g) {
    g.drawString("世界,你好!", 10, 10);
  }
}
```

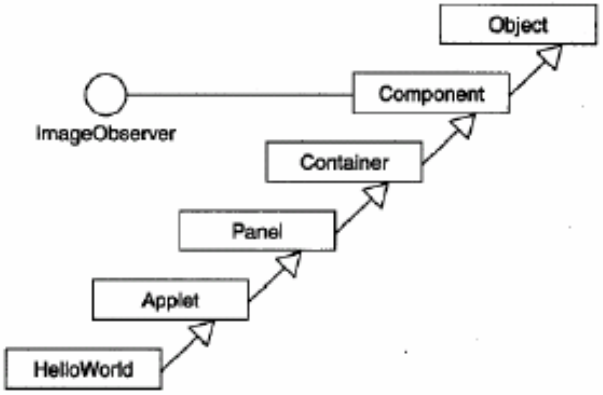
■ 类图



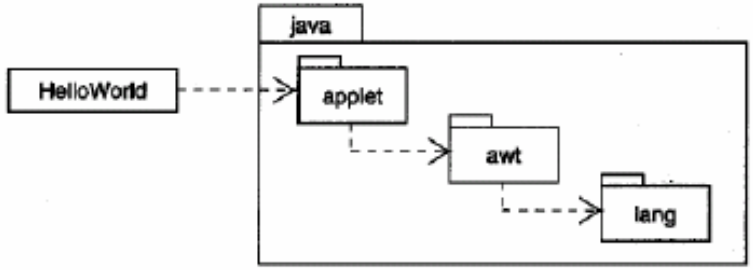
对HelloWorld的关键抽象



与HelloWorld直接相关的类

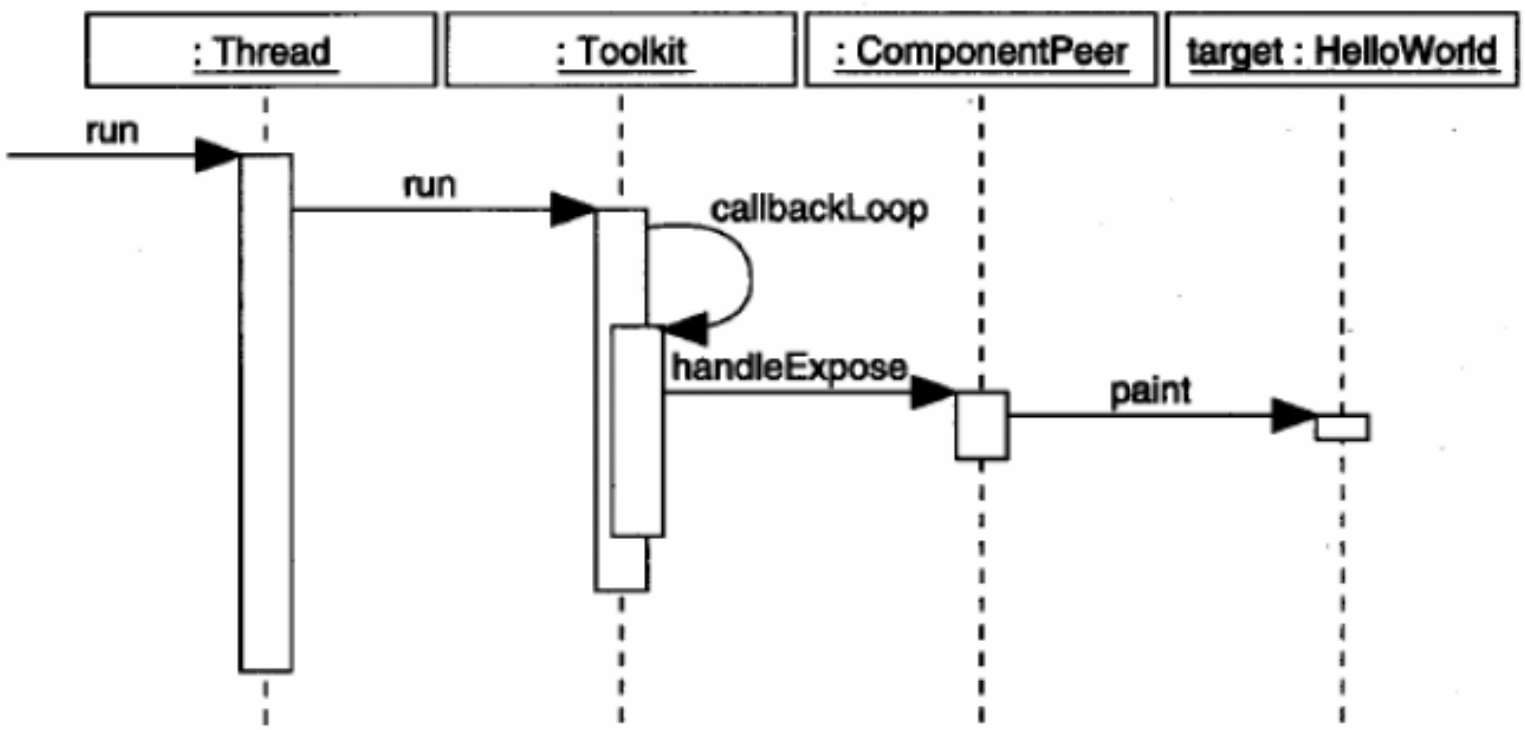


HelloWorld的继承层次



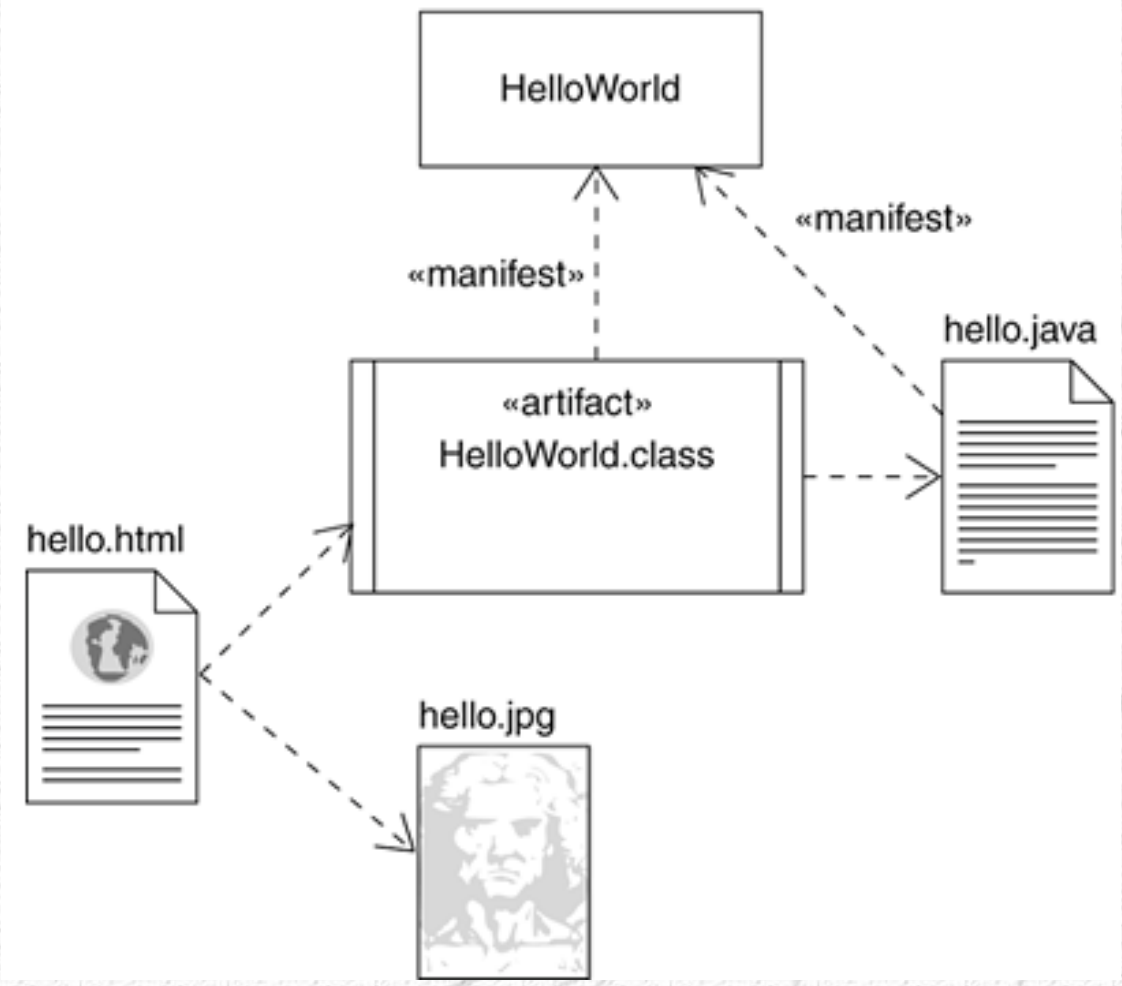
HelloWorld的包结构

■ Painting调用机制



顺序图

■ 制品


HelloWorld
«manifest»
«manifest»
hello.java
«artifact»
HelloWorld.class
hello.html
hello.jpg

图的作用

- 科技文档写作人员利用用例文档开始写用户手册和培训计划
- 分析人员和开发者将利用顺序图和交互图了解系统的逻辑关系、对象以及对象之间传送的消息
- 质量验证人员将利用用例图、顺序和通信图了解他们需要测试的信息
- 开发者将利用类图和状态图获得系统各个模块的详细信息以及它们之间的联系

- 部署人员利用组件图和部署图了解将产生哪些可执行文件、dll文件和其他组件，并且了解这些组件将如何安装
- 整个团队将利用模型确保需求和代码的一致性