

第六章 建立辅助模型

曹东刚

caodg@pku.edu.cn

北京大学信息学院研究生课程 - 面向对象的分析与设计

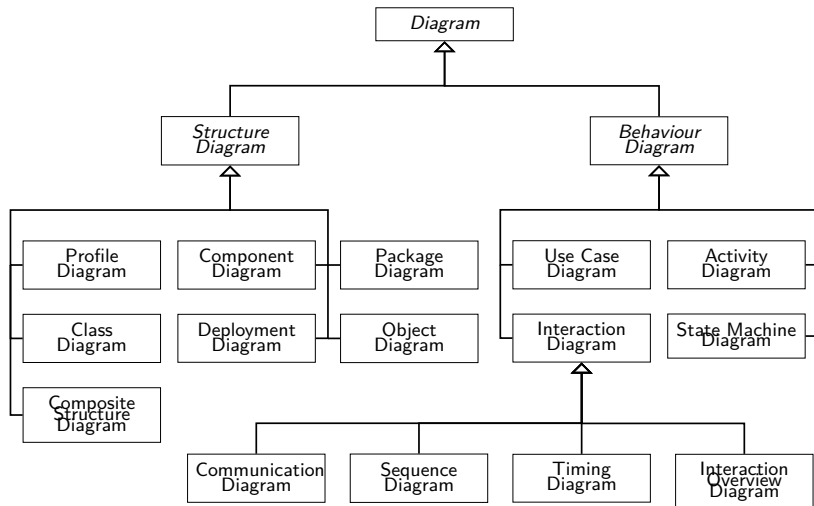
<http://sei.pku.edu.cn/~caodg/course/oo>



类图与其他模型图的关系

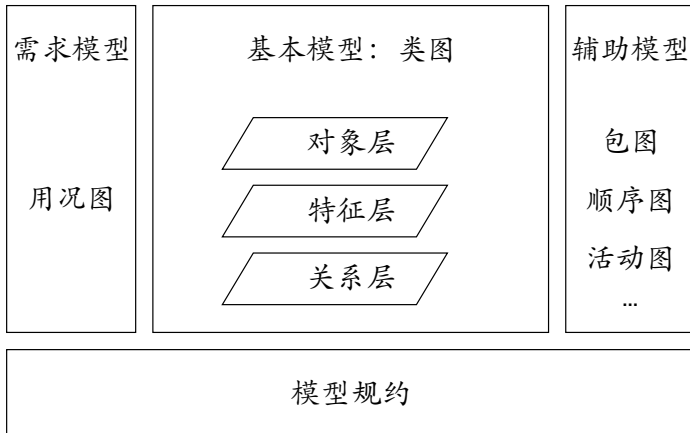
方法	模型图	特点
OMT	对象模型 + 动态模型 + 功能模型	掺杂其他方法
OOSE	需求模型 + 健壮模型 + 设计模型	解决不同阶段问题
Booch 方法	基本模型 + 补充模型	以 OO 方法为核心
Coad/Yourdon	类图 + 流程图	以 OO 方法为核心
UML	收集了大量的模型图	从不同的视角对复杂系统建模

UML2 中的各种图



本书建模方法用类图作为主要模型

类图为基本模型，用况图为需求模型，其他模型图为辅助



本书方法使用的各种 UML 图

类图	基本模型	最重要，必不可少
用况图	需求模型	建模的基础，提倡使用
包图	辅助模型	系统规模较大时使用
顺序图	辅助模型	补充类图，在交互情况较复杂时使用
活动图	辅助模型	描述对象的操作流程，也可描述高层行为
状态机图	辅助模型	准确描述状态与行为复杂的对象
构件图与 部署图	辅助模型	描述构件组织与部署方案
对象图与 通信图		不建议使用
其他		无强烈建议

内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图
- 5 构件图
- 6 部署图
- 7 模型规约

包图定义

包 (package)

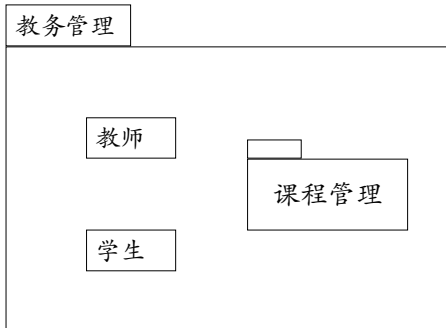
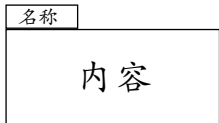
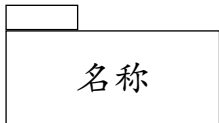
是一种将其他模型元素组织起来，形成较大粒度的系统单位的通用机制

基本思想：从不同粒度描述系统 — G. Miller 的“ 7 ± 2 原则”

注意：

- 包是一种组织机制而不是一种基本模型元素
- 包可以嵌套
- 包中的模型元素应具有某种意义的内在联系
- 包的划分有一定的灵活性或随意性

包的表示法



包之间的关系及表示法

引入 (import) 是包之间的一种依赖关系，表明源包中的模型元素能够**直接**引用目标包中的模型元素

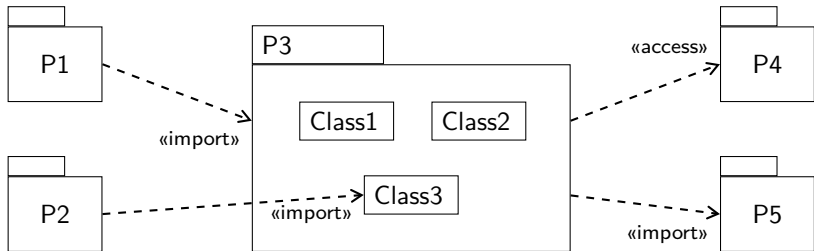
- 原由——名字空间（为了避免命名冲突）
- 以包为单位划分名字空间，引用时给出路径名
- 引入关系表明可以直接引用

访问 (access) 访问关系与引入关系类似，二者之间的差别：

UML1：目标包的元素是否可以被引入源包的其他包引入

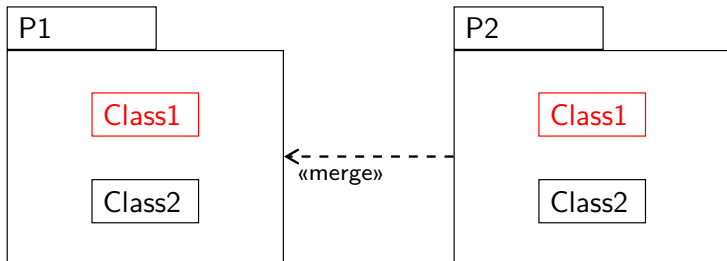
UML2：目标包（或目标元素）的可见性不同

包之间的关系及表示法



包之间的关系及表示法

合并 (merge) 是包之间的一个有向关系，表明目标包的概念定义被合并到源包中。建议对类给出完整的定义，不要分散到多个包中描述



包之间的关系及表示法

包之间的其他关系

泛化——没有正式的定义，只在包图的某些例子中出现

依赖——不加任何关键词的依赖关系，没有确切的定义

如何建立包图

1 将模型元素打包

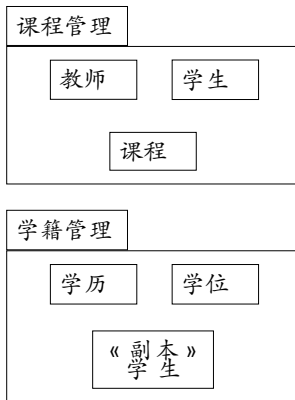
a 参照类图中的各种关系

- 一般-特殊结构
- 整体-部分结构
- 关联
- 消息
- 其他关系

b 根据问题域和用况

- 对象来源
- 功能类别
- 通信频繁程度
- 并发和分布情况

c 包的内容交叉问题



2 包的命名

- 一般-特殊结构中的根类
- 整体-部分结构中的整体对象类
- 其他
 - 根据包中元素所代表的事物
 - 根据包中元素所提供的功能
 - 根据包中元素的来源, ...

3 组织嵌套的包: 7 ± 2 原则

将若干低层包合并为高层包

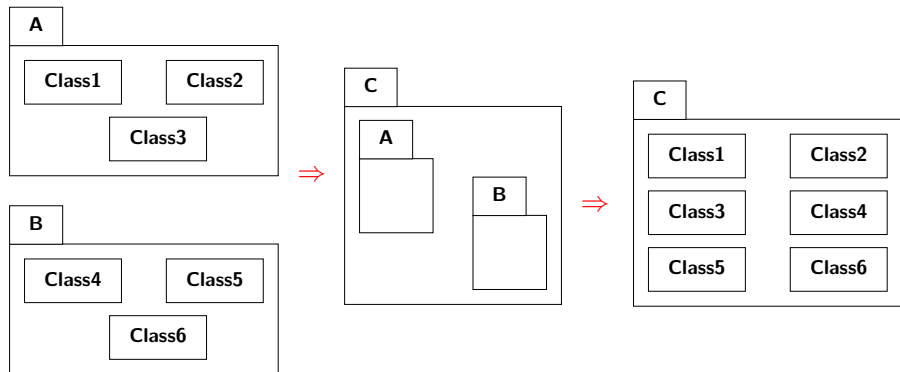
将低层的包与零散模型元素组织到高层的包中

合并的依据

- 参照将模型元素打包的考虑因素
- 包之间内容是否有交叉
- 包之间的关系是否紧密

如何建立包图

4 减少包的嵌套层次



5 建立包之间的关系

- 建立引入关系或访问关系
 - 源包和目标包的模型元素不能有命名冲突
- 建立一般的依赖关系
- 关于合并关系和泛化关系

内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图
- 5 构件图
- 6 部署图
- 7 模型规约

顺序图

顺序图

是一种详细地表示对象之间行为关系的图。

它按**时间顺序**展现了一组相互协作的对象在完成一项功能时所执行的操作，以及它们之间所传送的消息，从而清晰地表示对象之间的行为关系以及操作和消息的**时序关系**

适应范围：通常只适合表示一组相互协作的对象执行一项功能时的交互情况，包括外部可见的功能和内部功能。难以表示整个系统的交互情况

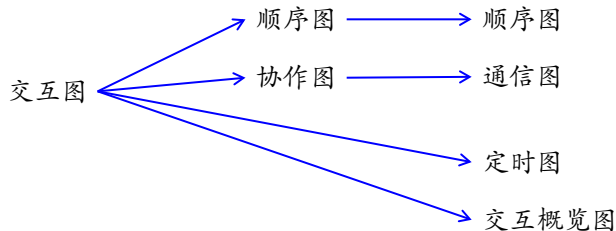
顺序图

名称的演变:

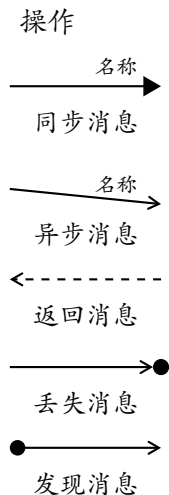
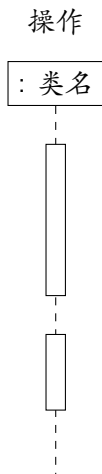
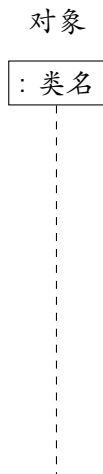
UML 之前

UML1

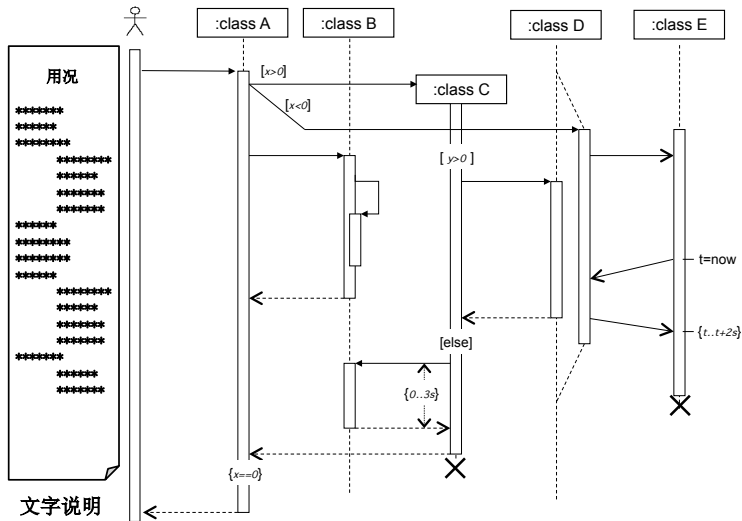
UML2



主要概念及表示法

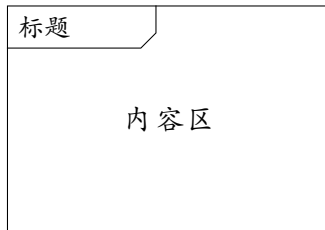


主要概念及表示法



帧 (frame)

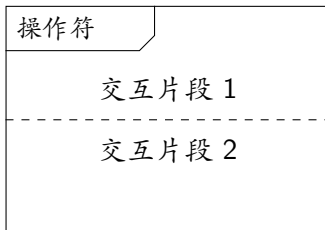
多种图中共同使用的图形表示机制，不仅用于顺序图，也可以在其他多种图中使用，特别是各种交互图



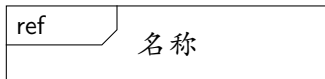
组织机制与复用

交互片段 (interaction fragment): 交互中的一个片段

组合片段 (combined fragment): 若干交互片段的组合

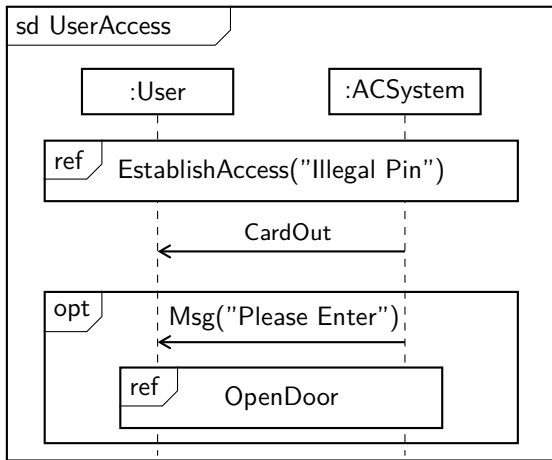


引用 (reference) 与交互使用 (interaction use)

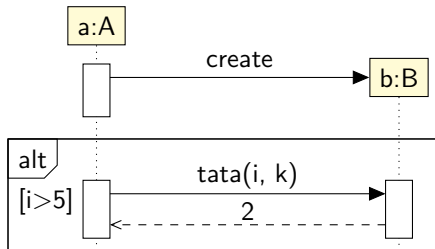


组织机制与复用

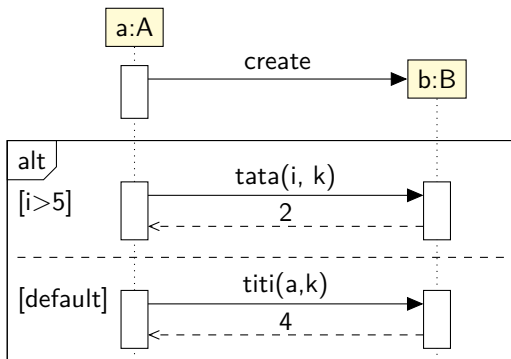
交互使用的示例



组织机制与复用



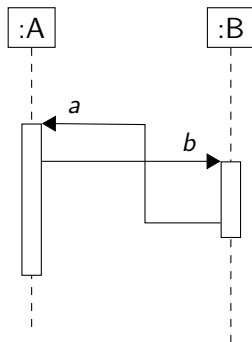
组织机制与复用



顺序图的几个问题

- 抽象层次问题
- 局部与全局问题
- 时间的表示问题
- 文字描述问题

避免操作细节
简述交互过程
体现外部行为
围绕系统功能



例：递归问题

如何建立顺序图

- 决定建立哪些顺序图。基本以用况为单位，但是不绝对
 - 简单的用况不必用顺序图描述
 - 系统内部的功能也可以用顺序图描述
- 确定参加交互的对象和参与者
 - 确定参加交互的参与者
 - 找出与参与者直接交互的对象
 - 以消息为线索，找出与交互有关的全部对象
- 绘制顺序图

内容提要

- 1 包图
- 2 顺序图
- 3 活动图**
- 4 状态机图
- 5 构件图
- 6 部署图
- 7 模型规约

活动图 (activity diagram)

是一种描述系统行为的图，它把一项行为表示成一个可以由计算机、人或者其他执行者执行的活动，通过给出活动中的各个动作以及动作之间的转移关系来描述系统的行为

活动图起源于流程图 (flow chart)，同时借鉴了工作流、Petri 网等领域的若干概念，使其表达能力比流程图更强，应用范围也更宽

主要概念

活动 (activity)

是由一系列动作构成的，是对一项系统行为的描述，它不是活动图的模型元素，而是一个整体概念，对应着整个活动图

动作 (action)

是活动的基本构成单位，被看作一种原子的构造成分
如果要展开一个动作内部的细节，则：

UML1: 定义为“子活动”

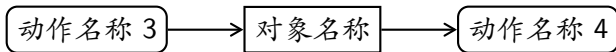
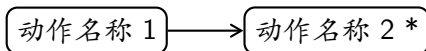
UML2: 定义为“调用行为”动作

表示法

活动图由**结点 (node)** 和**边 (edge)** 两种基本元素构成

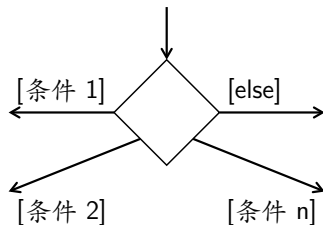
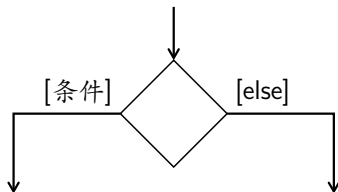
结点: 动作、判断、合并、分岔、汇合、起点、结束

活动边: 控制流和对象流



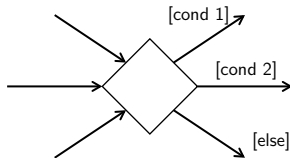
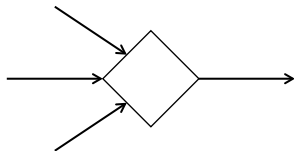
判断与合并

判断 (decision)：控制结点。表示执行到这一点时将判断是否满足某些条件，以决定从不同的分支选择下一个动作



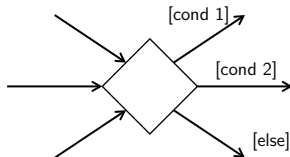
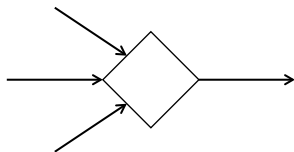
判断与合并

合并 (merge)：控制结点。表示把多个分支合并到一起



判断与合并

合并 (merge)：控制结点。表示把多个分支合并到一起

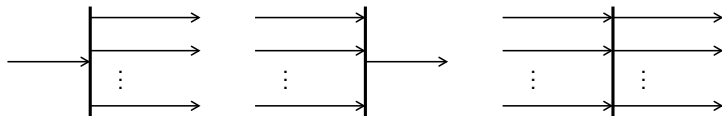


讨论：判断与合并结点是成对出现的吗？

分叉与汇合：用来表示并发行为的控制结点

分岔 (fork)：表示一旦前面的动作结束而流入这个结点，它的每个流出边所指的動作都可以执行

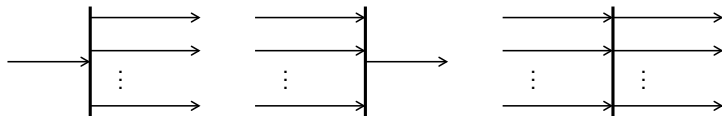
汇合 (join)：表示汇合点之前有多个控制流在汇合点上需要取得同步，并汇合为一个控制流



分叉与汇合：用来表示并发行为的控制结点

分岔 (fork)：表示一旦前面的动作结束而流入这个结点，它的每个流出边所指的动作都可以执行

汇合 (join)：表示汇合点之前有多个控制流在汇合点上需要取得同步，并汇合为一个控制流



讨论：分岔与汇合是否必须匹配？

起点、活动结束和流结束

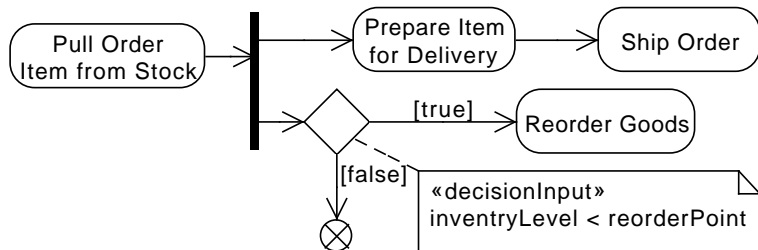
起点 (initial node) : 一个活动图所描述的整个活动的开始

活动结束 (activity final) : 活动图所描述的整个活动到此终结

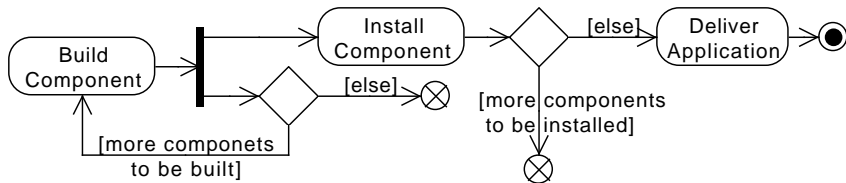
流结束 (flow final) : 表示活动图中的一个控制流的终结, 但并不是整个活动终结



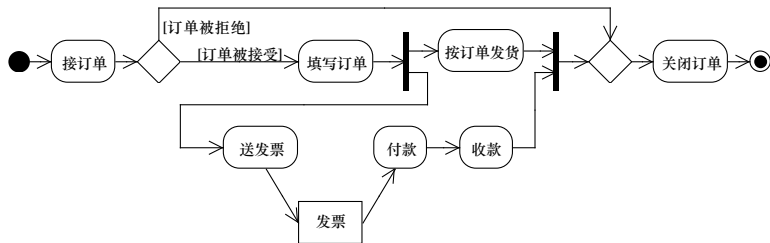
起点、活动结束和流结束



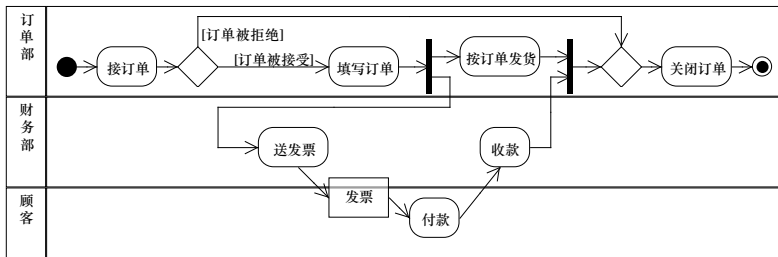
起点、活动结束和流结束



一个活动图的例子



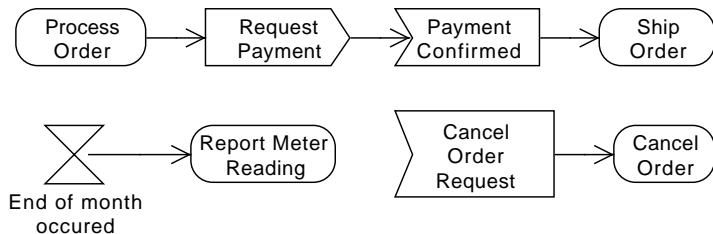
泳道 (swim lane)：一种辅助机制，其作用是把活动图中的各个动作划分到与它们的执行者相关的若干区域中，从而清晰地表现出不同的执行者分别执行了哪些动作



发送信号与接受事件动作

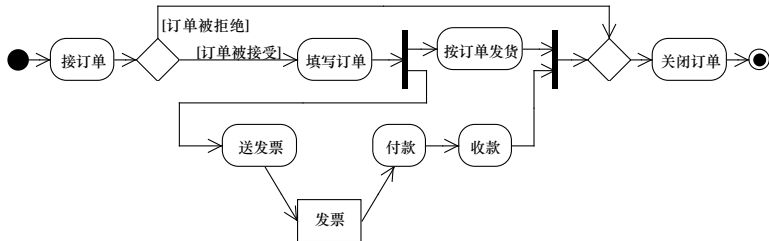
接受事件动作 (AcceptEventAction) : 如果接受事件动作没有输入边, 则当活动开始时, 它立即开始执行, 并且在接受了一个事件后也不会终止

发送信号动作 (SendSignalAction) : 发送异步信号



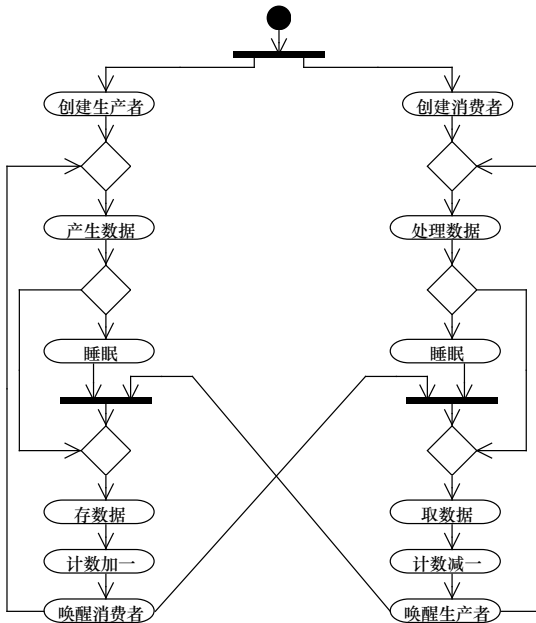
问题讨论

- 业务流程和执行过程的差异
- 并发描述上的误差
- 对象流问题
- 复杂性问题



如何使用活动图

- 描述对象的操作流程
 - 未必每个操作，未必十分详细
- 描述系统某些局部的行为
 - 判断是否真正必要
- 描述系统外部可见的行为
 - 实际上是描述用况，如果用文字更清楚就用文字
- 描述系统的业务流程
 - 注意业务流程和执行过程的差别和并发描述的误差



内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图**
- 5 构件图
- 6 部署图
- 7 模型规约

状态机图 (state machine diagram)

是一种描绘系统中的对象（或者其他实体）在其生命期内所经历的各种状态，状态之间的转移，发生转移的动因、条件及活动的模型图

别称：状态图 (state chart)、状态转移图 (state transition diagram, STD)

行为状态机与协议状态机

状态建模

通过分析系统（或其局部）所经历的状态和状态之间的转移，用状态、转移等概念来建立系统模型

在某些领域可以作为一种独立的建模方法，在面向对象建模中可以起到一种辅助作用

长处：对状态复杂多变，并且在不同状态下呈现不同行为的对象，通过状态建模将有助于准确地认识和描述对象的行为

局限性：一个状态机图通常只适合描述系统中一个或少数几个对象的状态及其转移情况，很难用于描述整个系统

状态 (state)

UML 的定义:

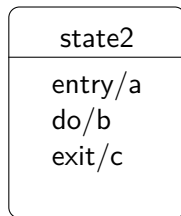
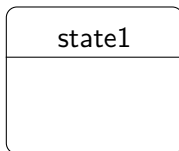
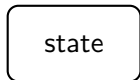
- “对象生命期中的一种条件或者情形，在此期间它满足某些条件，执行某些活动，或者等待某些事件。”
- “状态是对一种状况的模型表示，在此期间保持了某些（通常是固有的）条件。”

《对象技术词典》的定义:

- 对象或者类的所有属性的当前值
- 对象或者类的整体行为（例如响应消息）的某些规则所能适应的（对象或类的）状况、情况、条件、形式或生存周期阶段

状态 (state)

状态机图中的结点，有名字，还可有 **do/action**, **entry/action**, **exit/action** 等行为



伪状态 (pseudo state)

伪状态实际上并不是一种状态，只是为了加强状态机图的可视化效果而引入的一些图形符号，是结点（顶点）型的图形成分



initial



final



junction



decision



enter



exit



end



history

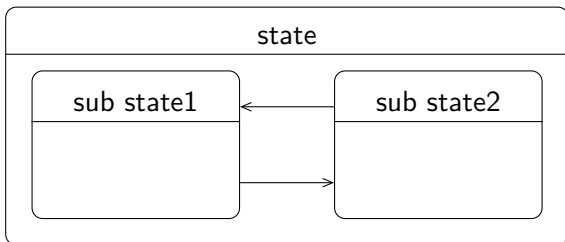


deep history

组合状态 (composite state)

由若干状态组织在一起所形成的状态称为**组合状态**

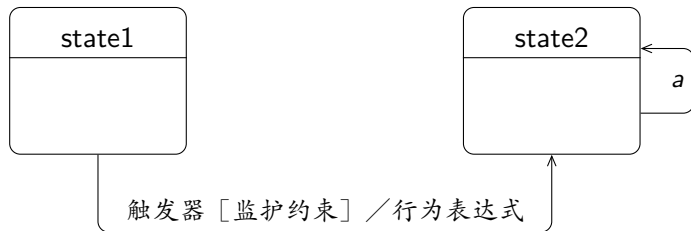
包含在组合状态内部的状态称为**子状态**，内部不包含其他状态的状态称为**简单状态**



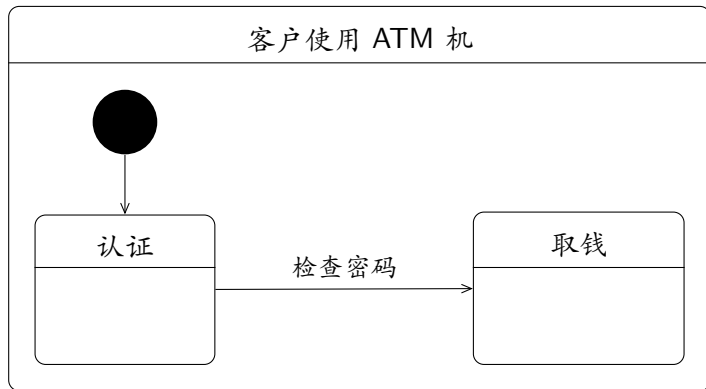
组合状态的作用：使模型更清晰、便于复用、简化图的绘制

转移 (transition)

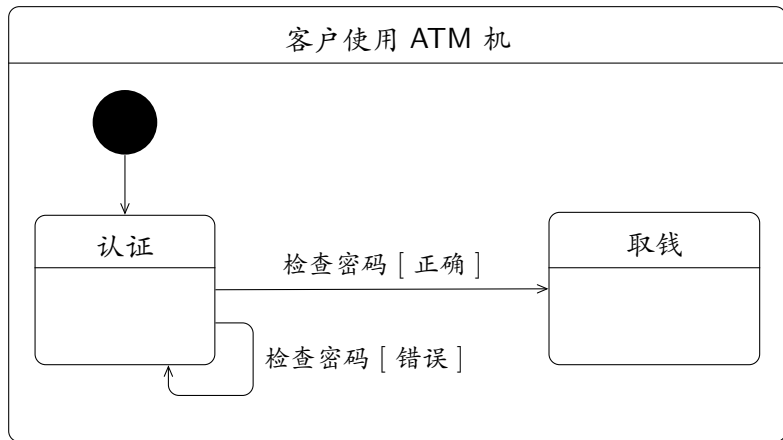
转移：状态机图中的边，描述了从一个状态到另一个状态的转变可以有触发器 (trigger, UML1 中称为事件 event)，监护约束 (guard) 或行为表达式 (behaviour-expression)



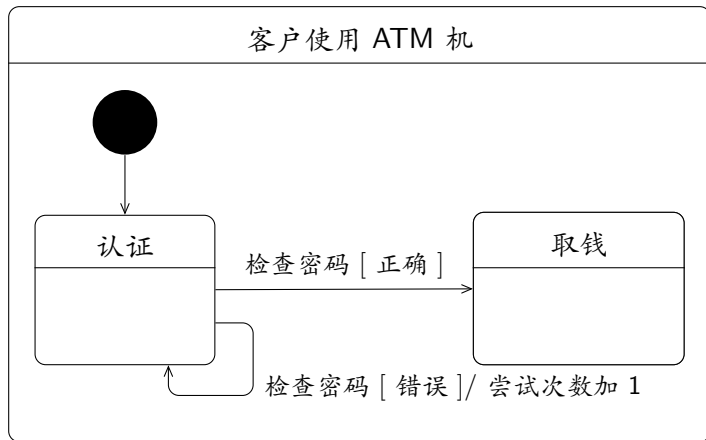
ATM 示例



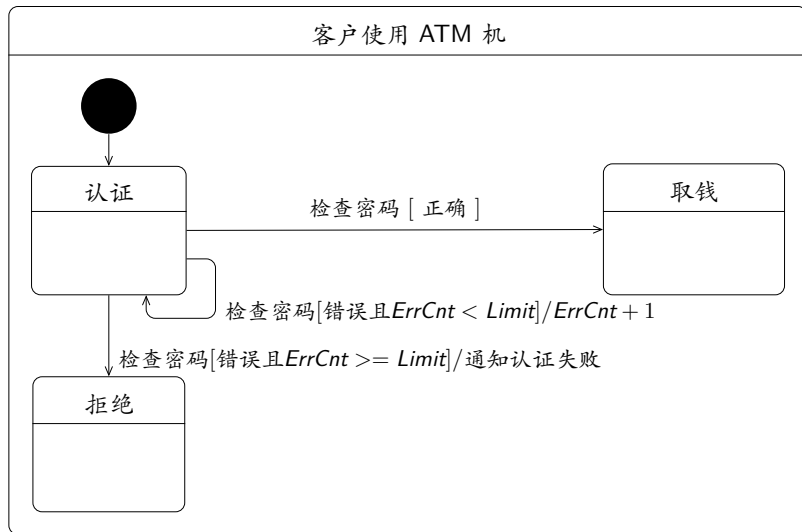
ATM 示例



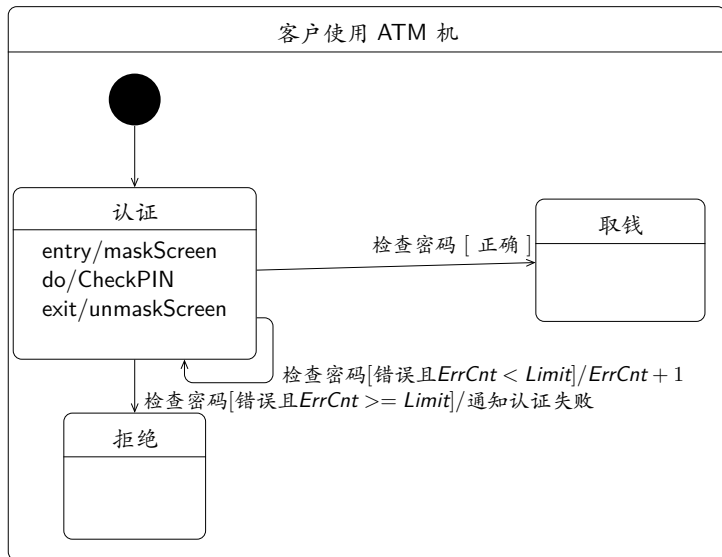
ATM 示例



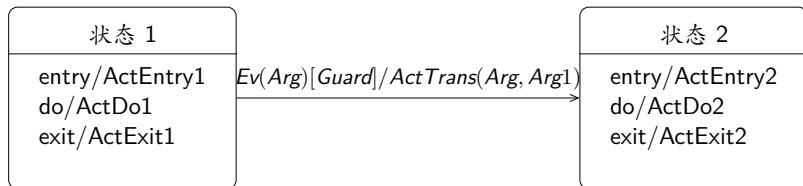
ATM 示例



ATM 示例

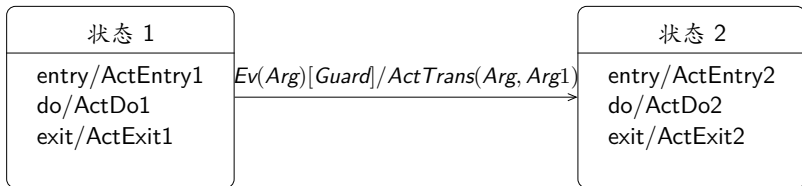


深入理解事件



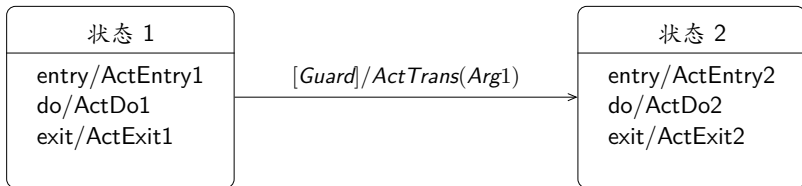
- 1 某处的某个动作产生了一个事件 Ev
- 2 Ev 传播到了当前对象
- 3 Ev 进入当前对象的事件队列
- 4 Ev 被从队列中取出，变成当前事件
- 5 Ev 被处理

事件处理



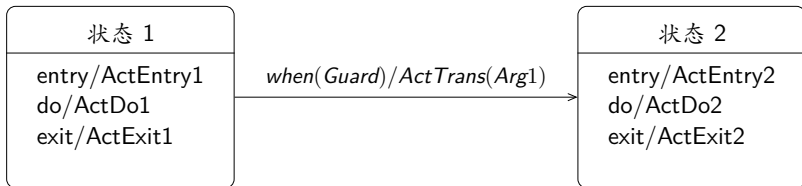
- 1 检查 *Guard*, 如果 *false* 放弃
- 2 中止 ActDo1
- 3 执行 ActExit1
- 4 执行 ActTrans(Arg, Arg1). 注: 同步调用
- 5 执行 ActEntry2
- 6 执行 ActDo2

事件处理：Completion 事件



- 1 等待直到 ActDo1 结束 (触发 completion 事件)
- 2 检查 *Guard*, 如果 *false* 放弃
- 3 执行 ActExit1
- 4 执行 ActTrans(Arg1)
- 5 执行 ActEntry2
- 6 执行 ActDo2

事件处理：Change 事件



- 1 等待直到 Guard 由 *false* 变为 *true* (触发 change 事件)
- 2 中止 ActDo1
- 3 执行 ActExit1
- 4 执行 ActTrans(Arg1)
- 5 执行 ActEntry2
- 6 执行 ActDo2

Completion 和 Change 事件的比较

Activity

Completion 事件: ActDo1 执行完成

Change 事件: 中止 ActDo1

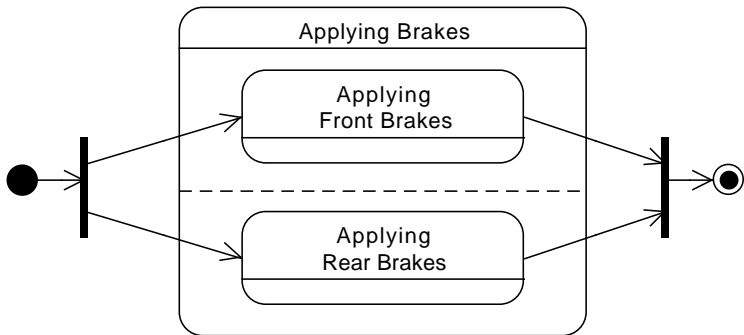
Guard

Completion 事件: 监护条件只检查一次

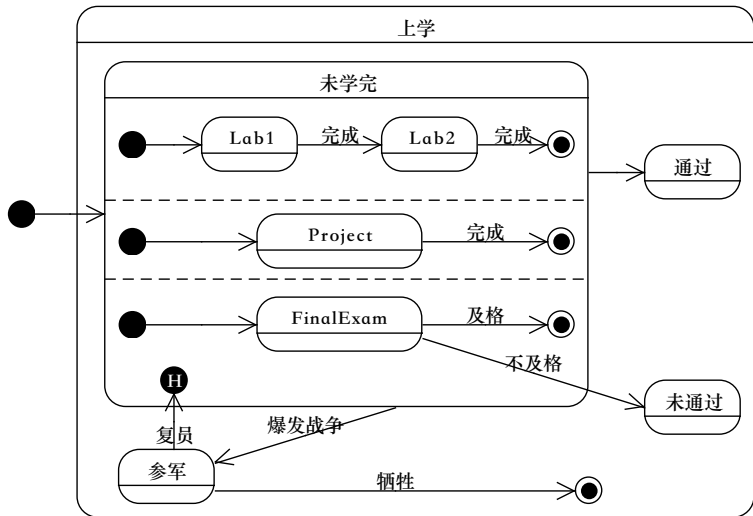
Change 事件: 持续检查监护条件

区域 (region)

区域之间的状态是正交的，区域用于表示并发转移



历史状态



如何使用状态机图：作为辅助模型

- 仅对状态对行为影响复杂的对象建立状态机图
- 使用主要概念：状态、转移、组合状态、伪状态
- 识别对行为有不同影响的对象状态，达到如下效果：
 - 在不同的状态下对象将呈现不同的行为规则
 - 同一种状态下对象的行为规则始终一致
- 认识和描述转移：触发器、监护约束、行为表达式
- 为类图提供有用的信息：状态——属性；转移——操作

内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图
- 5 构件图**
- 6 部署图
- 7 模型规约

构件图 (component diagram)

构件图

是一种表示构件的组织结构和相互关系的图，用于表达在实现时如何将系统元素组织成构件，从而支持以构件为单位进行软件制品的实现和发布

UML1 的构件图没有太多地反映当时构件技术的发展，它只是着眼于把软件的逻辑蓝图转化为计算机世界中的事物

UML2 为构件图增添了许多内容，能够表示构件技术领域的大部分常用的概念

构件 (component)

OMG: “一个构件表示系统中一个模块部件，它封装了它的内容，而其表现形式在其环境中是**可替换**的。”

通常：可复用构件的简称，特指按某种构件标准设计，并可在多个系统中重复使用的软件构造块。构件被认为是自治和独立的。

接口 (interface)

UML2: “接口是一种类目，它表示对一组紧凑的公共特征和职责的声明。一个接口说明了一个合约；实现接口的任何类目的实例必须履行这个合约。”

供接口 (provided interface) : 实现的接口

需接口 (required interface) : 使用的接口

端口 (port)

UML2: “类目的一个特征，指出类目与外部环境之间或者与内部的部件之间的一个明显的交互点。”

向内，可以连接到与外部交互的内部成分，称为供端口 (provided port)

向外，连接到供接口或者需接口，称为需端口 (required port)

实现（realization）和使用（use）是构件和接口之间的两种关系

- 构件与供接口之间为**实现关系**，称为 **接口实现**（InterfaceRealization）
- 构件与需接口之间为**使用关系**

构件实现（ComponentRealization）：构件中的类实现了该构件依据其接口所提供的合约

委派连接件 (delegation connector)


是从构件的端口连接到构件内部成分的连接件，它“把构件外部的一个合约（由它的端口说明）链接到由构件中的部件对这个行为的内部实现。”

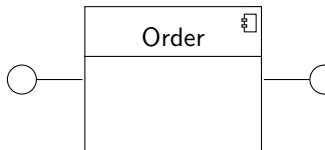
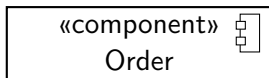
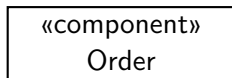
组装连接件 (assembly connector)

是两个构件之间的连接件，它表明一个构件提供了另一个构件所需要的服务。

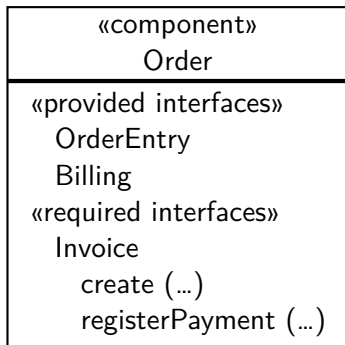
一个供接口和一个需接口之间的衔接就表示一个组装连接件

表示法

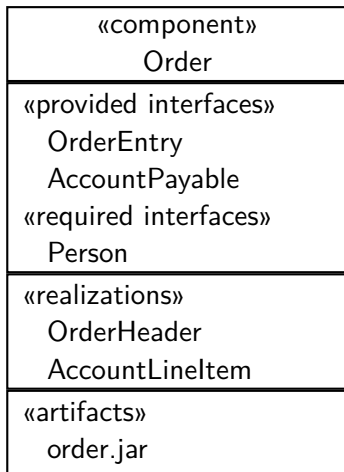
用 «component» 或者  表示构件，二者等价



外部视图或黑盒视图

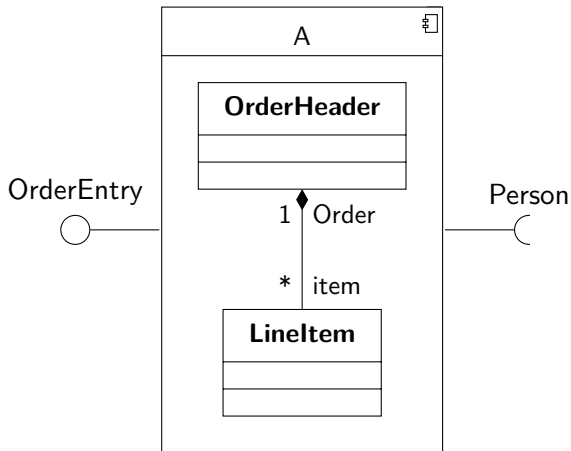


内部视图或白盒视图

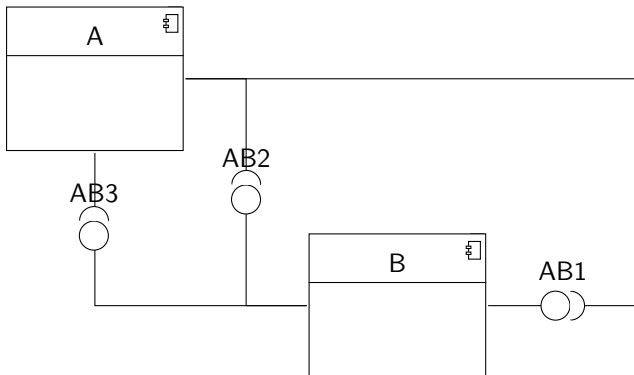


表示法

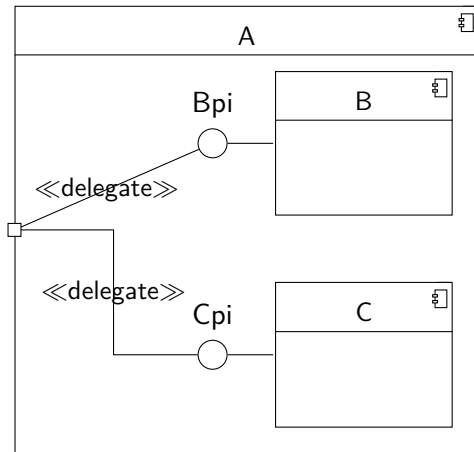
显示构件内部成分



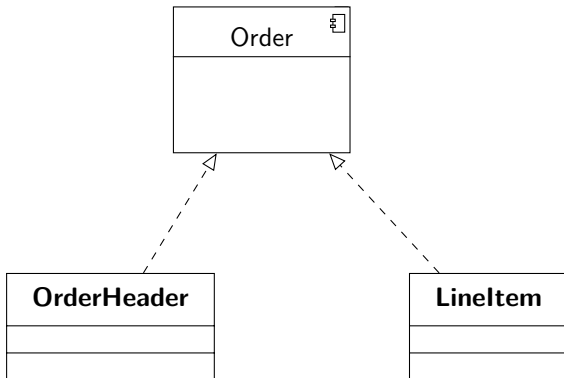
组装连接件



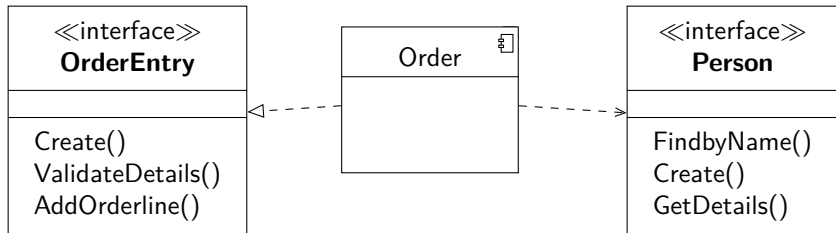
委派连接件



构件实现

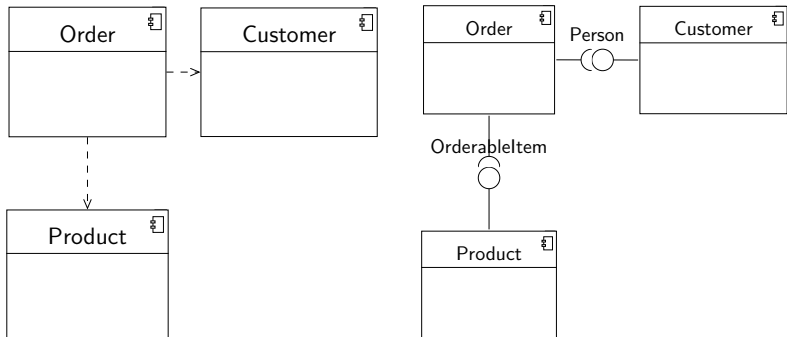


接口实现

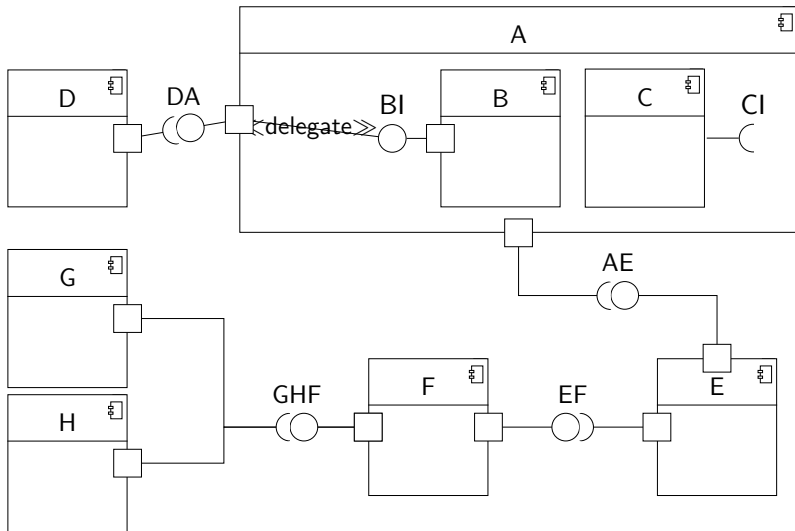


表示法

构件依赖



端口与连接件



如何使用构件图

- 用构件的图形符号表示每个构件
 - UML 提供了多种构件表示方式，可以有选择地采用
 - 建模工具应支持多种表示方式之间的切换
- 定义构件的接口
 - 考察构件中各个类的操作，发现构件的供接口和需接口
- 定义连接件
 - 考察构件之间的关系，定义组装连接件
 - 根据需要为构件接口和内部成分定义委派连接件

内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图
- 5 构件图
- 6 部署图**
- 7 模型规约

UML1 的定义：“一种显示运行时的处理结点以及在其上生存的构件、进程及对象配置的图。”

UML2 的定义：“部署图展示了制品在结点上根据它们之间部署的定义所做的定位。”

变化：部署在结点上的软件成分不再是构件，而是制品

主要概念

结点 (node)

表示一个硬件设备或执行环境，即被开发的软件制品将要部署于其上的宿主设备或环境

制品 (artifact)

“制品是对软件开发过程中或系统的部署与操作中所使用或产生的信息的物理块的说明。制品的例子包括模型文件、源文件、草稿、二进制可执行文件、数据库表、开发交付项、或者字处理的文档、邮件消息等。”

主要概念

执行环境 (ExecutionEnvironment)

“执行环境是一个结点，它为以可执行制品的形式部署于其上的各类构件提供了一个执行环境。”

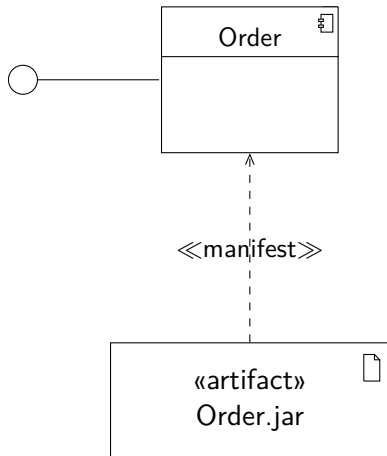
各种关系

部署 (deploy) 依赖

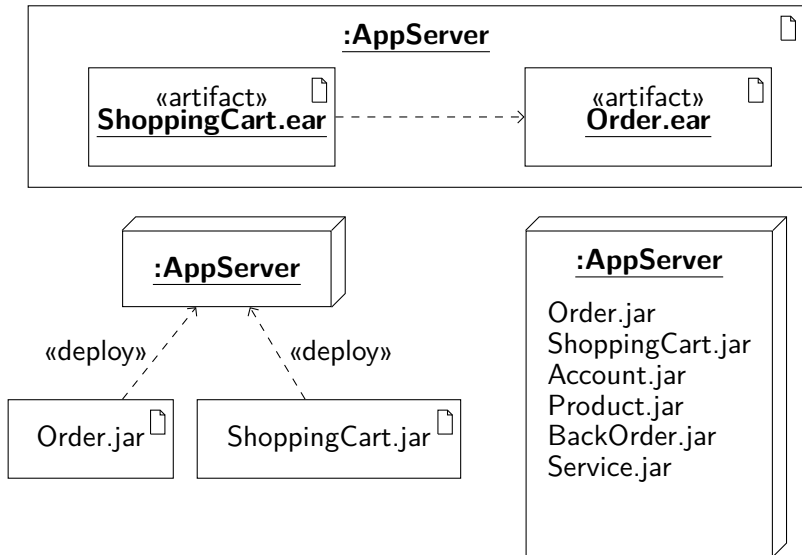
制品之间的依赖

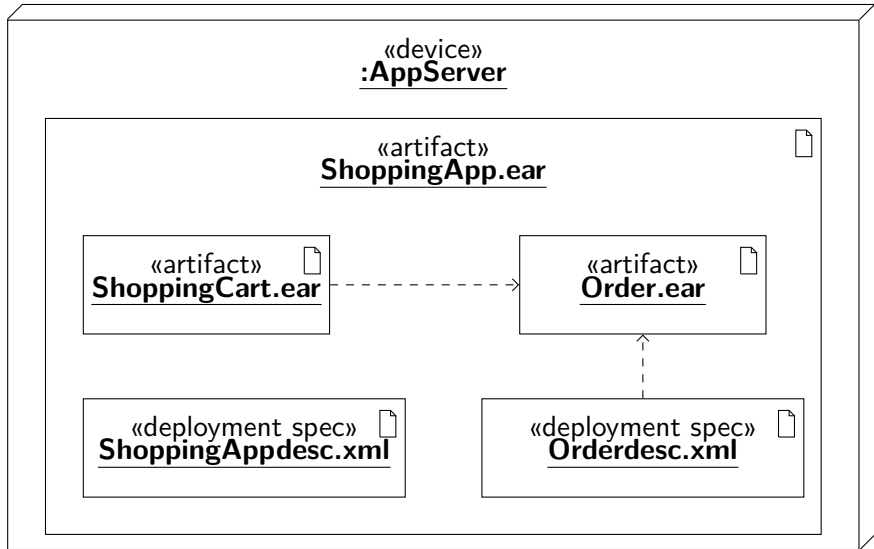
结点之间的关联

表示法

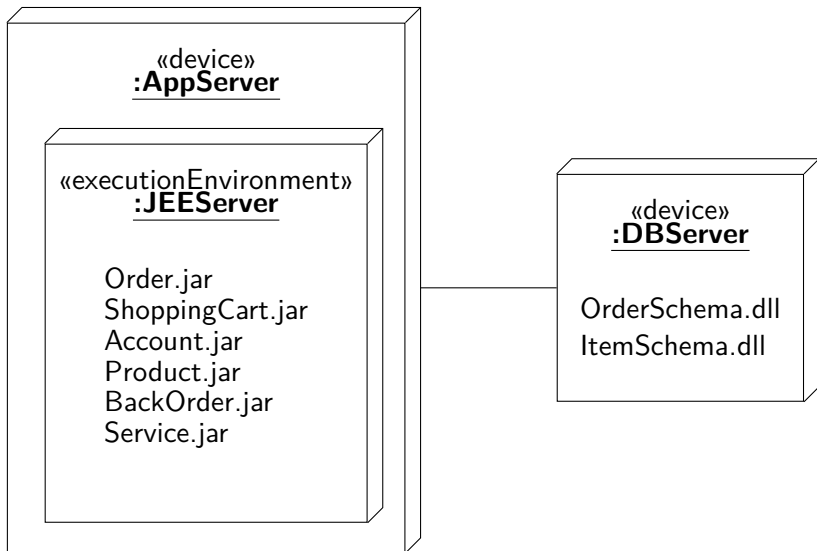


表示法

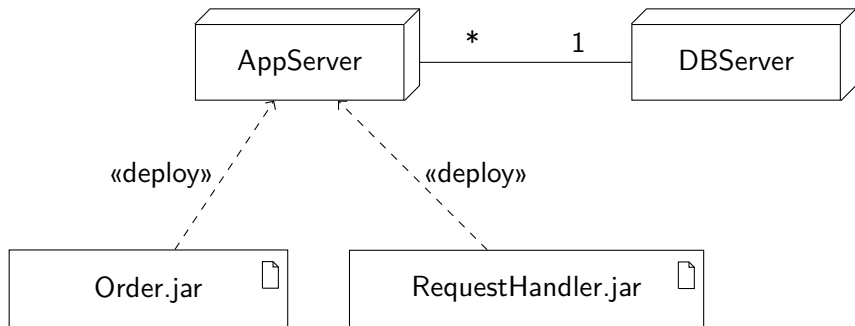




表示法



表示法



内容提要

- 1 包图
- 2 顺序图
- 3 活动图
- 4 状态机图
- 5 构件图
- 6 部署图
- 7 模型规约**

模型规约 (model specification)

模型规约是对系统模型的详细解释与说明

仅靠图形文档还不足以详细、精确地表达建模阶段应该给出的全部系统信息。在软件工程中，各种分析与设计方法通常需要在以图形方式表达系统模型的同时，又以文字的方式对模型进行解释和说明

问题讨论

- 规约是给**人**看的还是给**机器**看的?
- 采用**形式语言**还是**自然语言**?
- 规约如何组织模型图及其描述? 采取**分离方式**还是**混合方式**?

目标:

- 完整性——提供完整准确的模型信息
- 易读性——容易被人的阅读和理解
- 支持自动化处理——使尽可能多信息可自动转换为源程序
- 支持软件复用——规约中的信息容易在其他系统中复用

措施:

- 采用分离方式
- 采用格式化的描述方式，自然语言与形式化语言结合使用
- 在类的规约中定义类之间的关系，有向关系只在源端类中描述

几种模型图的规约

- 类图：类图规约是最重要的规约
 - 类的总体说明、属性说明、操作说明、对象实例说明
- 用况图：对每个用况的文字描述就是用况图的规约
- 包图：对每个包，必要时可以说明它的意义
- 活动图、状态机图、顺序图：一般不必做更多说明

模型规约的建立过程

与模型图同步进行，或者分别进行

OOA 与 OOD 有不同的分工，但是没有严格的界限

原则上，描述问题域中固有的事物的对象及其特征和相互关系，应该在 OOA 阶段定义；与实现条件有关的对象及其特征和相互关系，应该在 OOD 阶段定义

在这个原则下，有相当一部分工作既可以在 OOA 阶段进行，又可以在 OOD 阶段进行