UML notes 05 State Machine, Activity, Component, Deployment Diagram

collected by wxb

1 State Machine Diagram

- State Chart / State Machine
- 概念

> A behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events.
>
> 描述对象生命周期中的状态序列（对事件的响应）

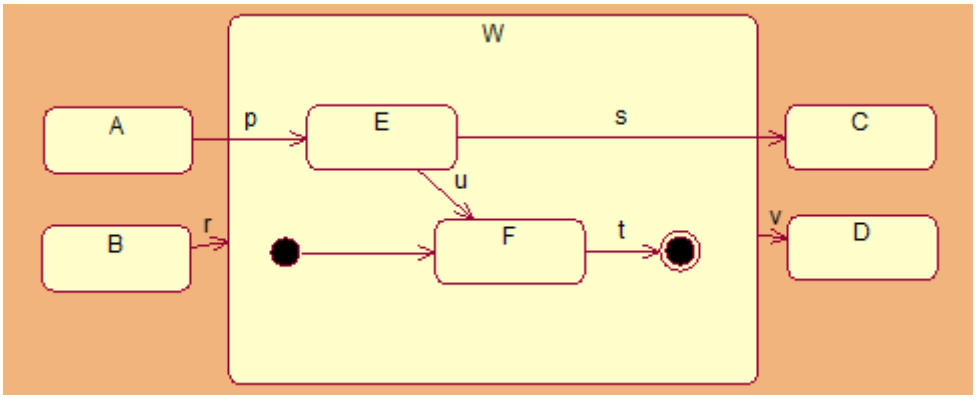
- 用途：为对象的生命周期建模，帮助测试、debug，描述动态行为
- Harel 状态图的一种变体
  - Mealy machine 米勒状态机：action 基于系统的当前状态和触发事件
  - Moore machine 摩尔状态机：只有与状态相关的进入和退出动作

1.1 State

- 一个对象生命周期中的状态 condition / situation
  - 满足某种条件
  - 执行某个活动
  - 等待某个事件
- 每个对象都有一个或多个状态
- 一个状态是一系列actions的结果
- 通常，状态会在一个事件发生后改变
- **内容**
  - name
  - entry/exit effects
  - internal transitions
  - sub-states
  - deferred events
- classification 分类
  - initial state 初始状态：每个状态图中有且只有一个
  - final state 最终状态：每个状态图中可以有0个、一个或多个

    pseudo-state 伪状态：除了名字外，两者都可能具有正常状态的常见部分

  - composite state 复合状态：nested state 有子状态的状态就叫做复合状态；可以有初始状态和结束状态
  - sub-state 子状态：被嵌套在另一个状态里面



- non-orthogonal sub-state非正交子状态：在指定时刻，复合状态中只有一个子状态可以到达
- orthogonal state 正交子状态：在指定时刻，复合状态中可以有多个子状态被到达

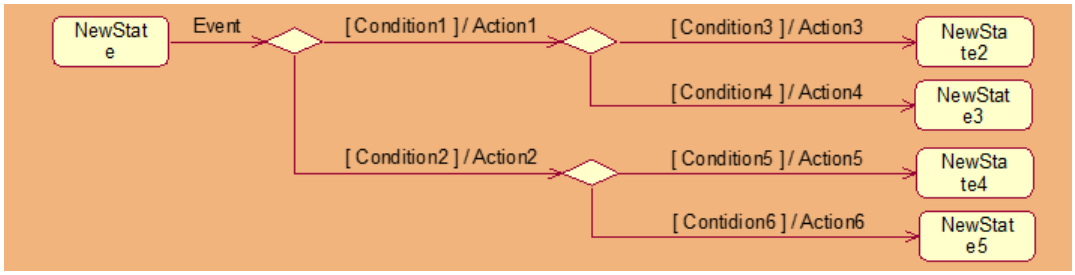
- history state 历史状态：伪状态，用于记住离开复合状态之前最后一个处于活动状态的子状态，以避免再次从初始状态开始；仅用于复合状态
  - shallow history state 浅历史状态：只记忆直接嵌套状态机的历史
  - deep history state 深历史状态：记忆任何深度的最内层嵌套状态

  如果嵌套状态机达到最终状态，则会丢失所有存储的历史记录

  > 如果没有历史状态，则可以使用flat state machine，但是会很混乱。
  >
  > - 对于每个连续的子状态，都需要在其执行 exit action 时，抛出一个值给复合状态中的 某个局部变量
  > - The initial state to this composite state would need a transition to every sub-state with a guard condition, querying the variable
  >
  >   复合状态机中的初始状态，则需要该查询变量（作为保护条件），要有其他子状态转换对应的 transition

1.2 Transition

- 两种状态之间的关系，表示当指定事件发生并满足指定条件时，处于第一种状态的对象将执行某些操作并进入第二种状态
- 对于给定的state，只能产生一个transition；并且transition中的每个 condition 都要互斥
- 内容
  - event trigger
  - guard condition：一个布尔表达式
  - effect：一个可执行的行为，可以作用于拥有状态机的对象，间接作用于对对象可见的其他对象
  - source state
  - target state
- 分类
  - external transition：最常见，会改变object的状态
  - internal transition：不会改变对象的state
  - completion transition: self-transition，箭头指向自己
  - composite transition：由简单的transition组成，通过 branch, fork, join 组合



- 表达方式：`event-signature [guard-condition] / action`

  例：`targetAt(p) [isThreat] / t.addTarget(p)`

1.3 Event

- external：系统和actor之间传递
- internal：系统内部的对象之间传递，如overflow exception
- type
  - signal event 信号事件：对象接收到触发transition的信号
  - call event 调用事件：调用操作
  - change event 改变事件：condition 会因为布尔表达式改变而改变
  - time event 时间事件：时间表达式被满足

signal event: asynchronous

- signal 是一种 message

  message是一个对象，会被一个对象异步发送，并被另一个对象接受
- 表示：class <<signal>>

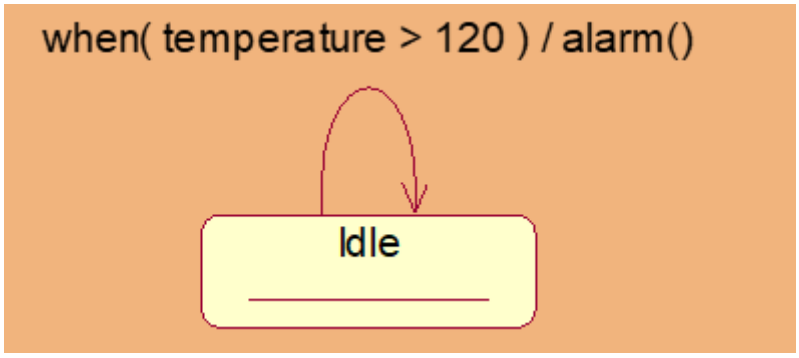
  可以有属性和方法
- signal event 表示对象接收到信号后，通常会触发状态转变
- 通常交由状态机处理

call event: synchronous

- 调用事件表示对象收到对对象操作的调用请求：
  - 可能会出发状态机中的状态转变
  - 可能会调用目标对象的方法
- 通常由方法处理

change event

- 改变事件表示状态或某个条件满足情况的改变
- 关键词：`when()`
- 只会在某个condition的值由 false 变为 true 的时候触发



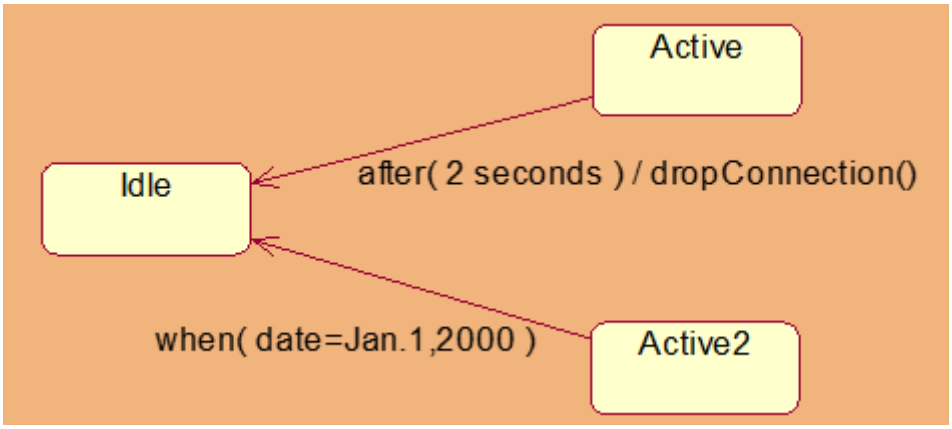
- 与 guard condition 的区别

  guard condition 是transition的一部分，会在相关事件发生时被计算，如果false则transition不发生

  change event 表示一个需要被反复 test 的事件

time event

- 时间事件表示时间的流逝
- 关键词：`after()`
- 关键词：`at()`，`when()`



else

- 如果没有任何对象，则事件也不会发生
- 时间事件和改变事件都涉及一个对象
- 信号事件和调用事件至少涉及两个对象
  - 发送信号/调用操作的对象
  - 事件指向的对象
- The call event that an object may receive is modeled as operations on the class of the object; the named signals that an object may receive is modeled by naming them in an extra compartment of the class

  对象接收到的调用事件被建模为对对象类的操作； 对象接收的命名信号是通过在类的额外隔间中命名它们来建模的

1.4 action

- 一种可执行的计算，导致模型 state 或返回值 变化
- 不能被中断，操作时间可以被忽略
- 两种特殊的action
  - entry action 进入事件：每次进入状态，entry action就会被执行

    `entry / action-expression`
  - exit action 退出事件：每次退出状态，exit action就会被执行

    `exit / action-expression`

    例：`entry / setMode(onTrack)  exit / setMode(offTrack)`

2 Activity Diagram

- **dynamic** 对系统动态方面进行建模
- 本质上是一个流程图 flow chart

  展示了从一个活动到另一个活动的控制流，显示了顺序、并发以及分支控制
- 与交互图的区别

  交互图侧重于从对象到对象的控制流

  活动图强调一步一步的控制流
- 可以通过 forward 正向和 reverse 逆向工程构建可执行系统
- 用途
  - 在整个系统、子系统、操作或类的上下文中使用
  - 和用例、协作绑定
  - 通常用来工作流建模、操作
- 基础概念
  - action / activity node
  - initial node / final node
  - flow / edge
  - condition
  - branch
  - fork & join
  - decision / merge
  - swim-lane

2.1 action / activity node

action

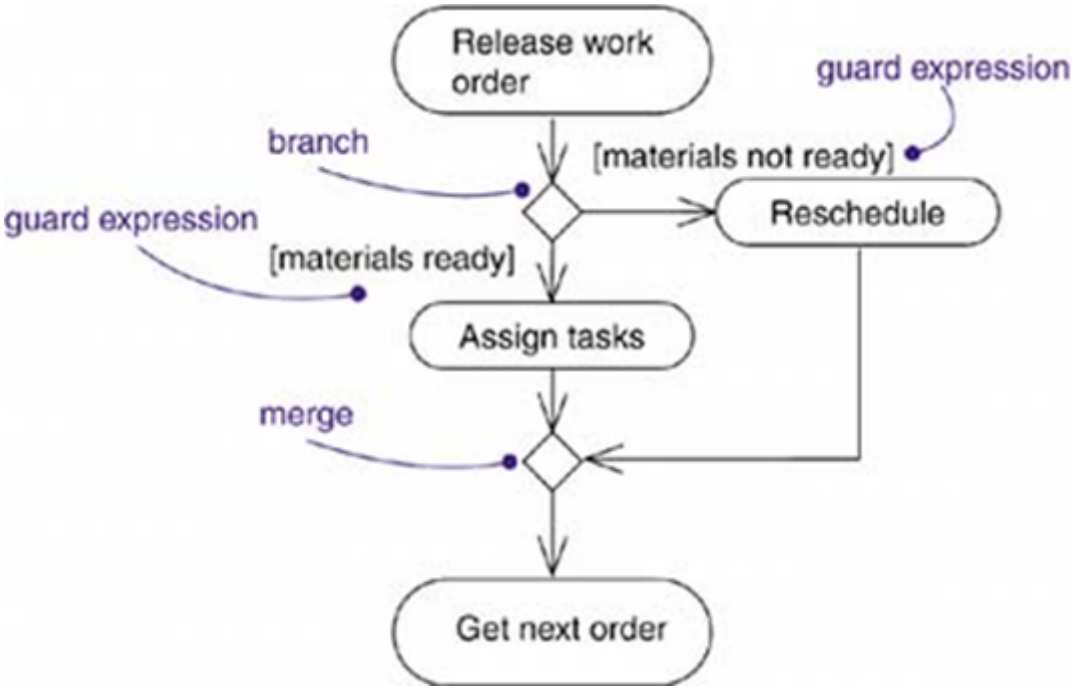
- 原子的 => 不能再被分解
- 不能被分步执行
- 花费时间微不足道
- 目标：执行 entry action，然后改变到另一个 state

activity node

- 活动节点是嵌套的action 分组，或者其他嵌套的activity nodes
- 有可视的子结构
- 不是原子的，可以被解构
- 需要花费一些时间才能完成
- 一个 action可以被看作是 activity node的特殊情况

2.2 branch & merge

- 用来表示可选的路径（基于布尔表达式）
    - 要覆盖所有可能性
    - 输出流上的guards不应重叠
    - merge时不需要guards（即合并时不需要判断条件真假）



2.3 fork & join

- 用来表示并行的控制流
- The number of flows that leave a fork should match the number of flows that enter its corresponding join

fork

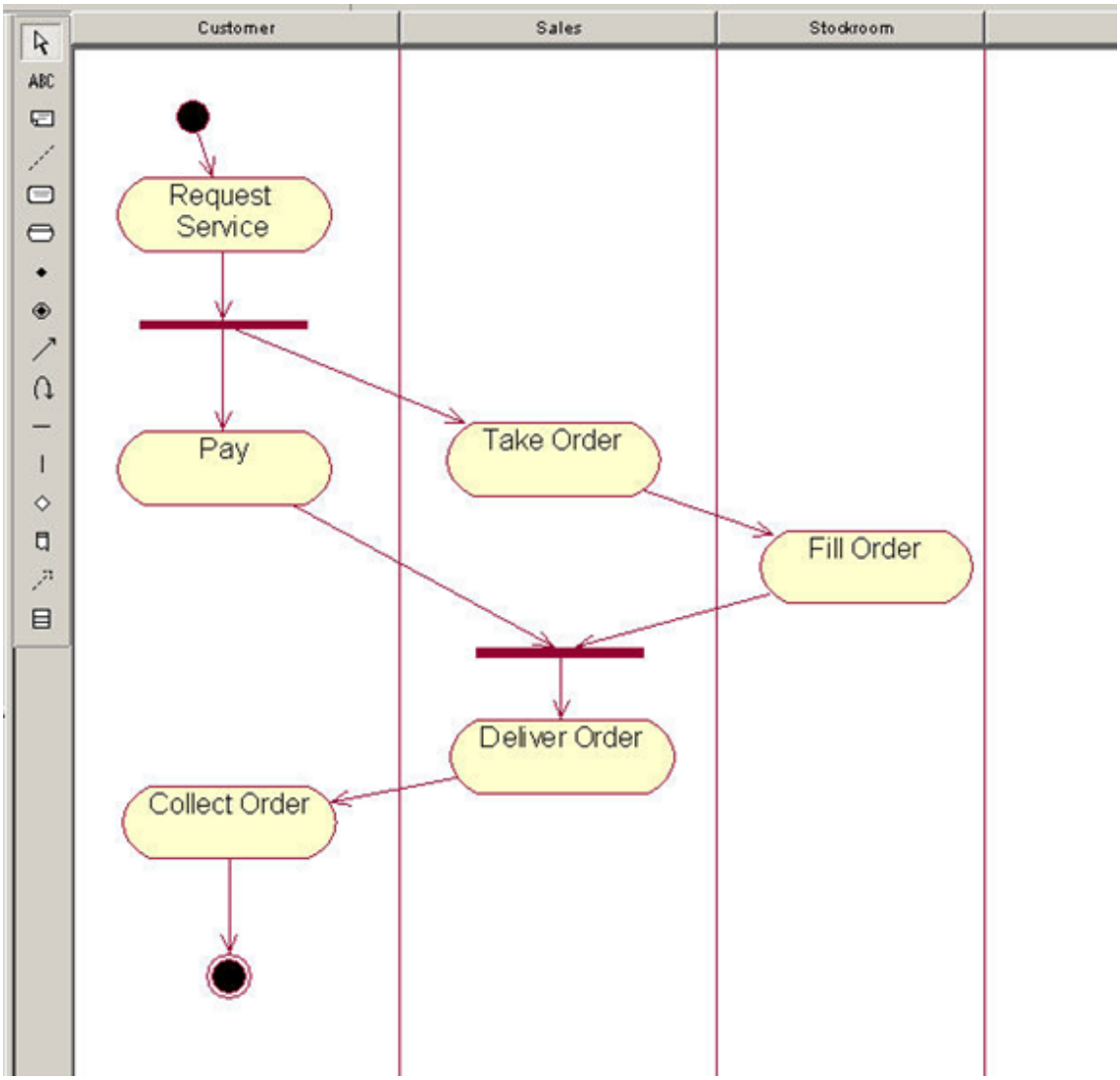
- 一个incoming transition 和两个或多个 outgoing transition，每一个都表示独立的控制流
- 在fork下，这些路径上的活动并行进行

join

- 同步所有并发的flows，等待每一个输入流到达这个join
- 一个控制流会在join下继续

2.4 swim-lane

- 每个泳道代表活动图整体活动的一部分的high-level responsibility
- 一个泳道最后可能被一个或多个类实现
- 泳道关注 responsibility，不是和类一对一的



2.5 object flow

- 对象流表示对象和活动之间的association
  - 一个activity创建一个object（作为activity的输出）
  - 一个activity使用object（作为activity的输入）
- 对象流属于控制流

2.6 Comparison of activity and state machine diagram

- state machine
  - 表示对象的状态（用属性描述）
  - 表示对象的生命周期、
  - 节点是state，用名词或名词短语描述
  - 描述了从一个state到另一个state的转变
- activity diagram
  - 表示对象的行为（用action描述）
  - 表示系统的flow，类似流程图，可以囊括多个用例

    - 节点是action，用动词描述
    - 描述了从一个action到另一个action 的控制流
- 都不适合描述几个对象之间的交互，交互还是要顺序图或协作图
- 都用于行为建模 behavior modeling

3 Component Diagram

3.1 component 组件

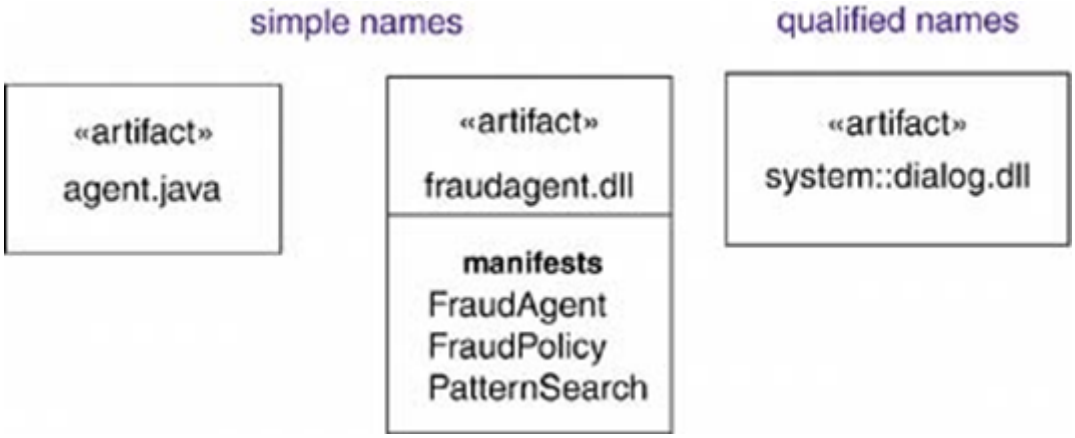
代表系统的模块化部分

- 封装了其内容
- 表现形式在环境中是可以替换的（当且仅当他们被提供的和所需的接口完全一样时）
- 根据提供的和所需的接口，定义自身行为
- 对组件的粒度没有限制，例：小至数字转字符的转换器，大至整个文档管理系统
- 关键字：<<component>>

3.2 artifact 工件

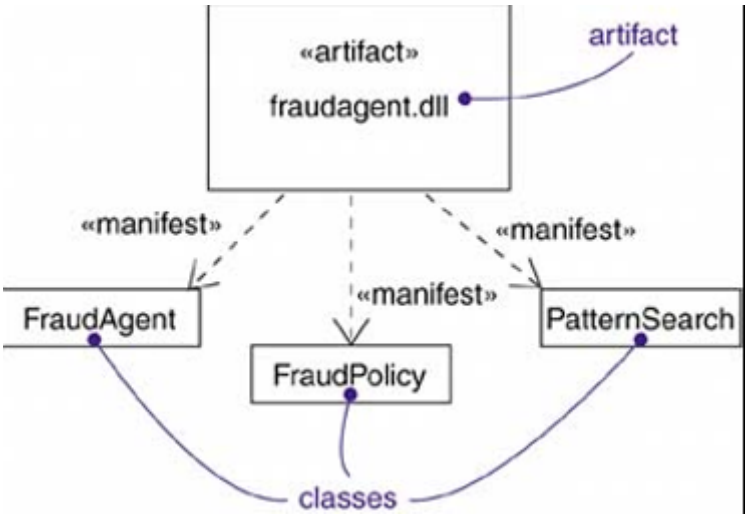
存在于系统 implementation platforms level 的物理部分

- 是部署在节点、设备、执行环境上的物理实体
- 关键字：<<artifact>>
- 示例：模型文件、源文件、脚本、二进制可执行文件、数据库系统中的表、开发可交付成果、文字处理文档、邮件消息等



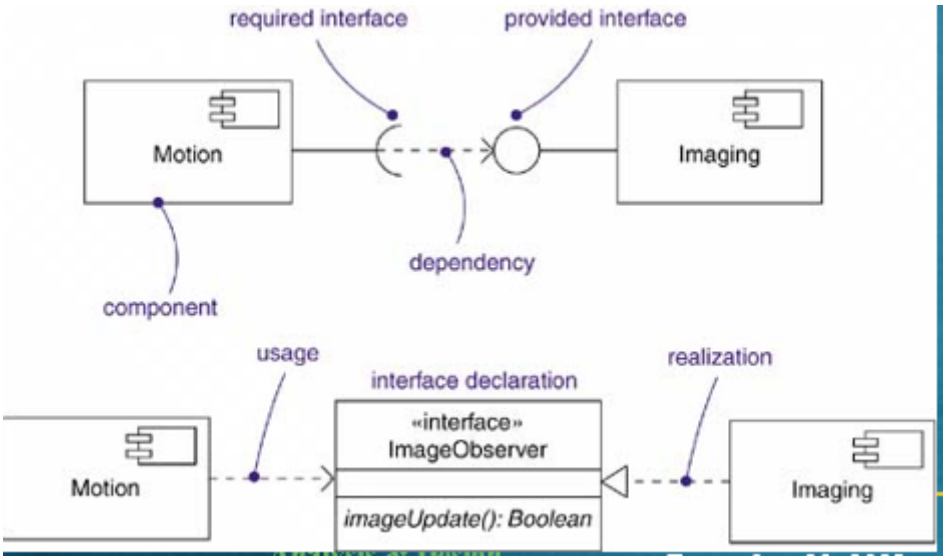
- 组件和类的区别
    - 类
        - 是逻辑抽象
        - 可以有属性和操作
    - 组件
        - 是物理抽象，可以在节点上放置；是逻辑元素的物理实现
        - 操作通常只能被接口使用
- 组件和工件的区别
    - 都是分类器 classifiers
    - 类
        - 表示逻辑抽象，不能放在节点上
        - 有属性和操作
    - 工件

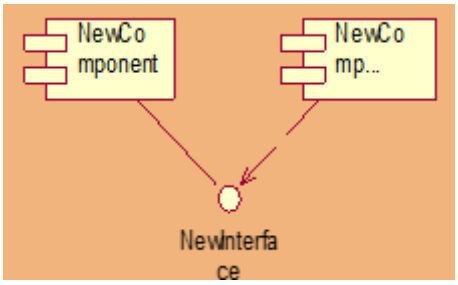
- 表示物理实体，在现实世界以比特形式存在，可以放置在节点上
- 表示implementation platform上物理的比特包
- 可以实现类和方法，但自身没有属性和操作



3.3 interface 接口

- 一组操作的集合，用来表示基于组件建模的系统提供的一种服务
- 接口用来将基于组件建模的系统中的组件bind在一起
- 打破组件之间的直接依赖
- 类型
  - required interface
  - provided interface
- 接口和组件的关系：依赖dependency 和 实现 realization





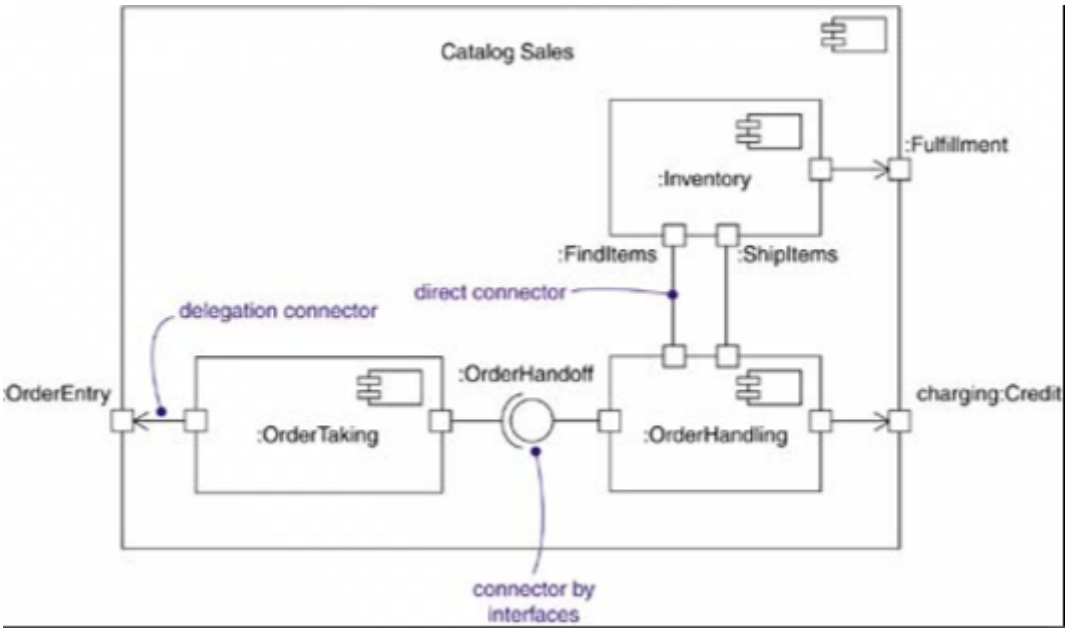
3.4 port 端口

- 端口是进入封装组件的特定窗口
- 接受来自指定接口的组件的消息
- 组件外部可视的行为：端口数量
- 允许组件的接口们被分成离散的包或独立使用

  端口提供的封装和独立性允许更高级别的封装和可替代性
- 端口是组件的一部分，端口的实例随着它们从属的组件的实例创建和毁灭
- 端口可以有多个

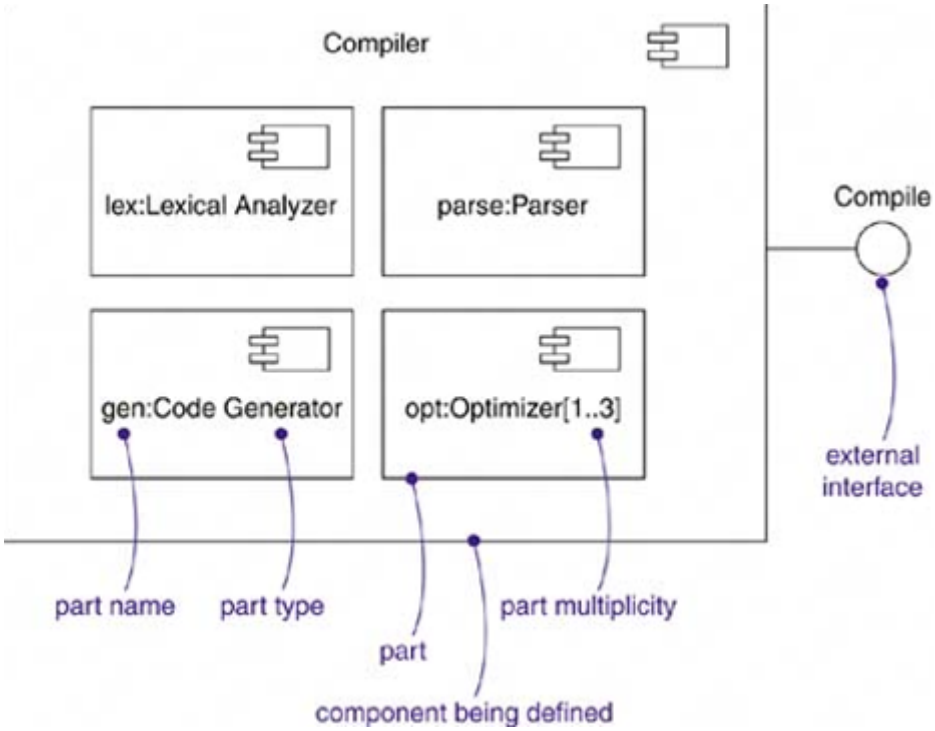
3.5 connector 连接器

- 连接器是两个端口之间的线，用来连接两个组件
  - 在组件中表示一种（短暂的）连接link
  - 一个link是普通的association的一个实例，一个短暂的link表示两个组件之间的使用关系
  - 表示方式：直接在两个组件或组件的端口之间画一条线，或者用一个圆（提供接口）和半圆弧（接受接口）



3.6 Internal Structure & Part 内部结构和部件

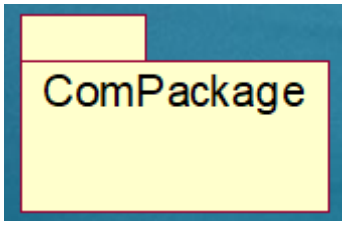
- part 部件：是一个组件的实现单元
  - 有名字name 和类别 type
  - 一个部件可以有多个
  - 可以和工件对应



- internal structure 内部结构：由组件实现的部分以及它们之间的联系组成。

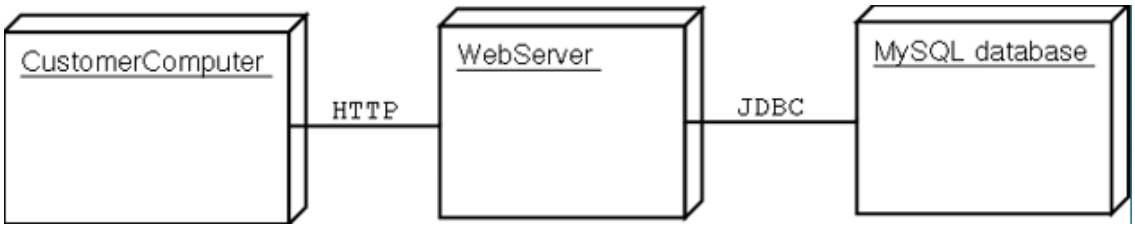
3.7 component package 组件包

- 包含了一组逻辑相关的组件，或者系统的主要组件，与类图的逻辑包类似
- 命名：文件系统的路径 `System.Web.UI.java.util`



4 Deployment Diagram

- 对部署在节点上的物理工件 artifacts 进行建模
- 系统只有一个部署图，帮助设计人员理解分布式系统
- 两个主要元素：node节点、connector连接器



4.1 node

- 运行时存在的一种物理元素
- 表示可计算的资源，通常至少含有内存和处理能力
- 表示方式：正方体
- 命名：简单命名和qualified 命名

- 和artifact的对比
    - 都有名字
    - 都可以嵌套，有实例，可以是交互的参与者
    - 都可以参与依赖、泛化、连接关系
    - 但是artifact是在系统执行时参与的事物，nodes是执行artifact的事物
    - artifact 表示逻辑元素的物理打包，节点表示artifacts物理层面的部署
- processor 处理器：一个有处理能力的节点，可以执行artifact
- device 设备：一个没有处理能力的节点，通常表示与现实世界接口的东西，如modem调制解调器

4.2 connector 连接器

- 是节点之间的association联系，就是connection