《面向对象设计与构造》
Lec15：模型质量问题分析

OO2023课程组

北京航空航天大学计算机学院

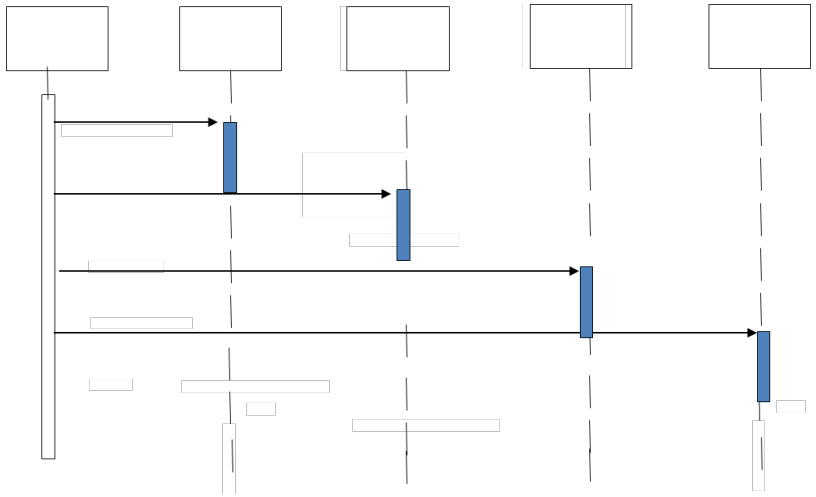
本周日8-9节，三(204)教室
最后一次课（课程总结+颁奖）

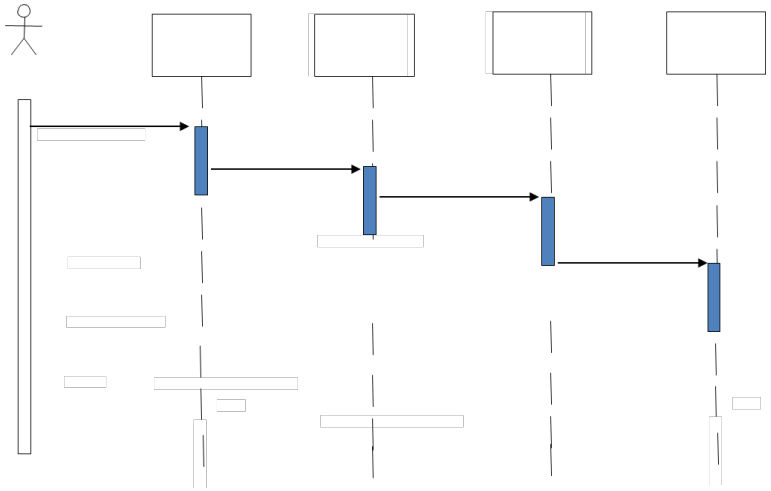
目录

- 顺序图的表示结构
- UML模型的解析式理解
- UML三种模型图之间的一致性关系
- MBSE的实践应用
- 课程主题回顾
- 软件设计难在何处
- 本周作业解析
- 课程总结博客作业

顺序图的表示结构(1)

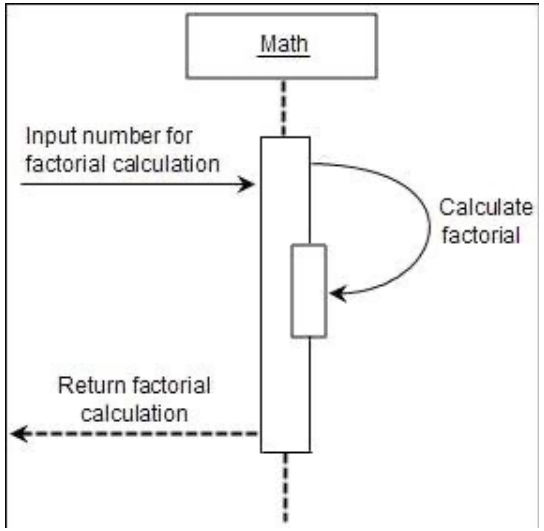
- 围绕一个**明确的行为**或**场景主题**来表达对象之间的协作关系
  - 行为主题：系统的某个功能/用例、某个接口的行为等
  - 场景主题：所关注的某个业务场景或者测试场景等，可跨越多个功能
- 有两种典型的协作模式
  - 中心控制式 vs 分级代理式

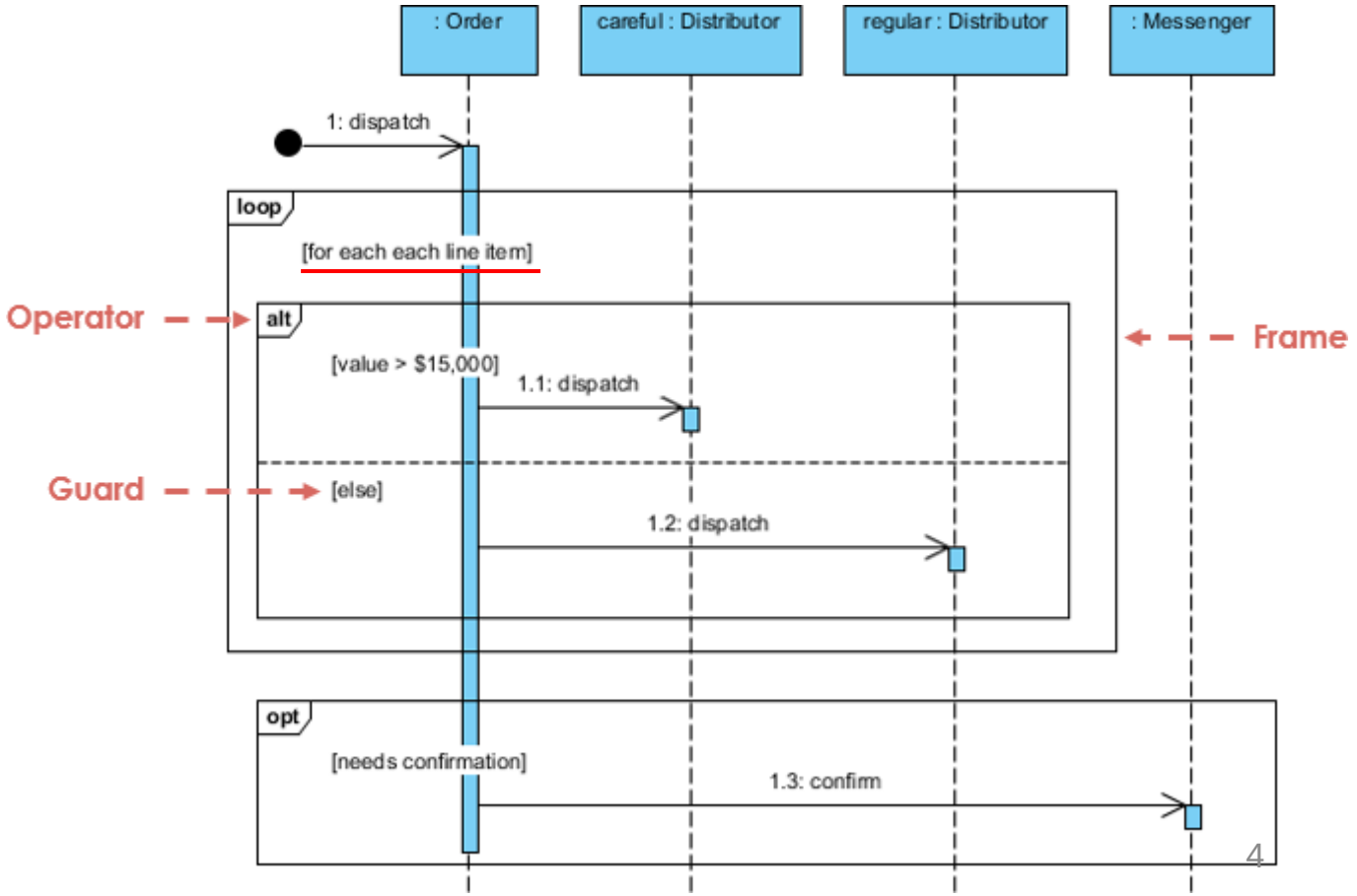




顺序图的表示结构(2)

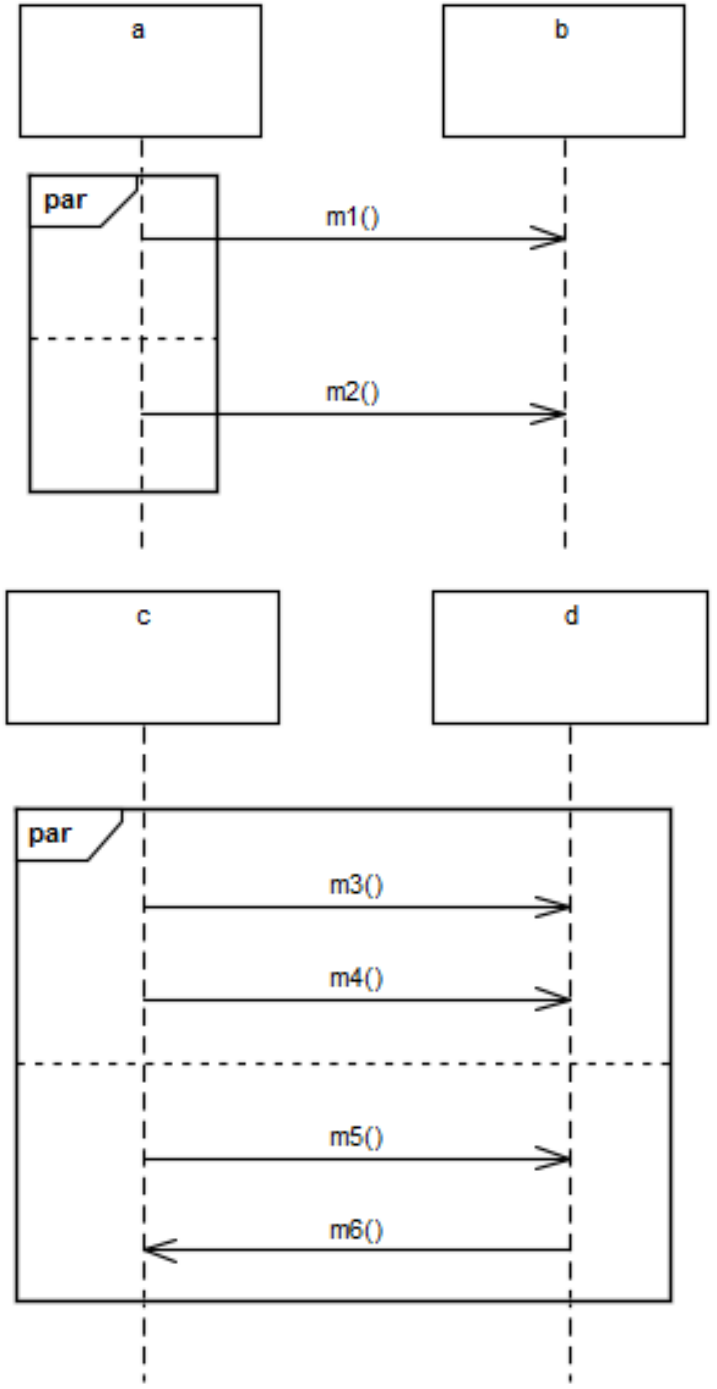
- 对象协作并不都是直线式的逻辑
  - 递归模式
  - 循环模式
  - 多分支模式
  - 备选分支模式





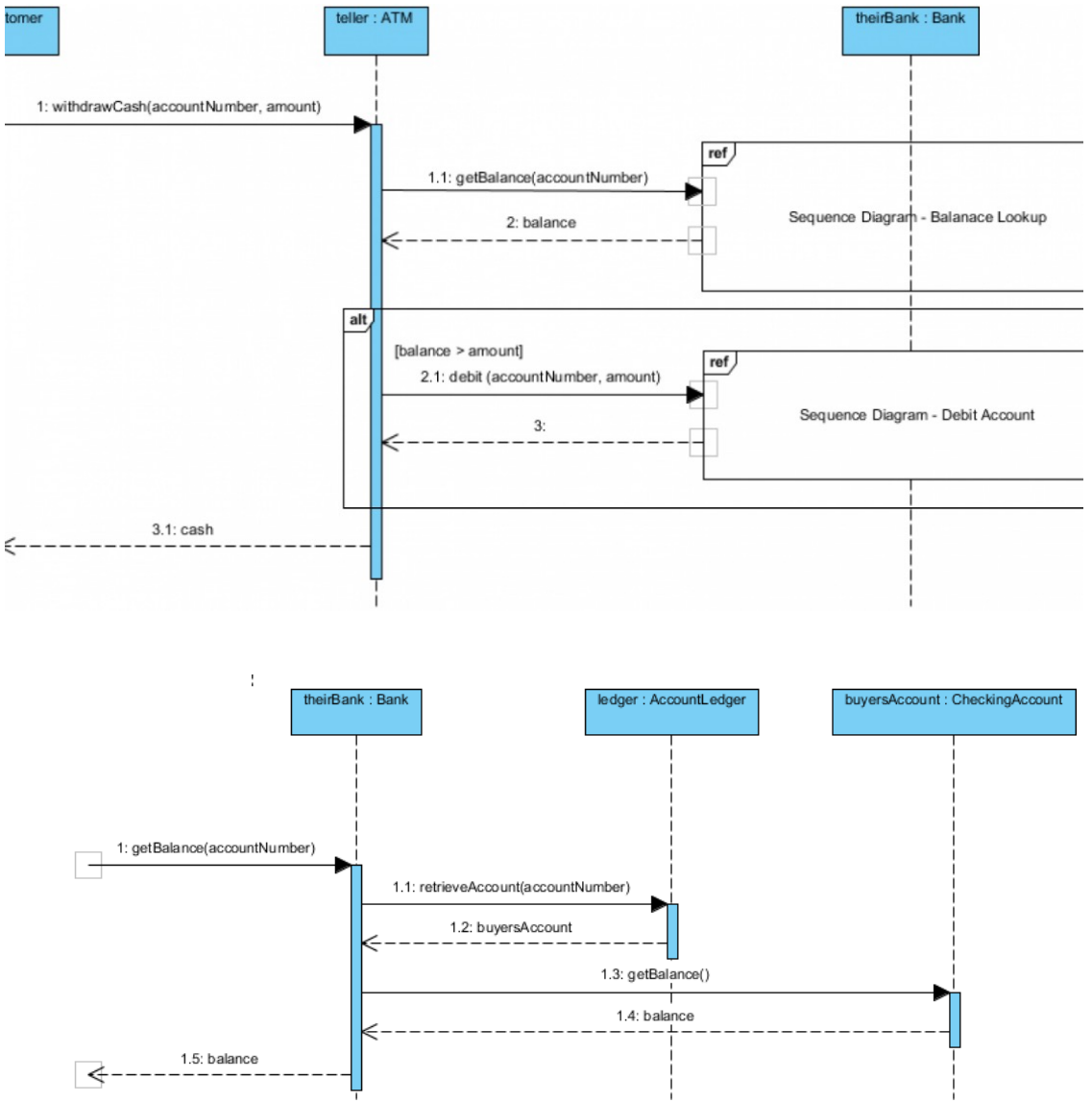
顺序图的表示结构(3)

- 并发场景下的对象协同如何表示？
  - Parallel message sent/receive events
- 单个UMLLifeline的并发活动
  - a上的m1和m2的发出顺序不确定
  - b上的m1和m2接受顺序确定：m1必须早于m2
- 多个UMLLifeline的并发活动
  - m3-sent必须早于m4-sent
  - m3-recv必须早于m4-recv
  - m5-recv必须早于m6-sent
  - c和d上分别有多少种可能的事件序？



顺序图的表示结构(4)

- 引用其他的交互场景，以层次化方式表达更丰富的交互
  - ref operator
- 封装：把顺序图中的复杂消息控制逻辑封装为一个局部的交互场景，多处ref，避免重复
- 重用：基于已有的交互场景来构造更高层次的交互场景，建立层次结构

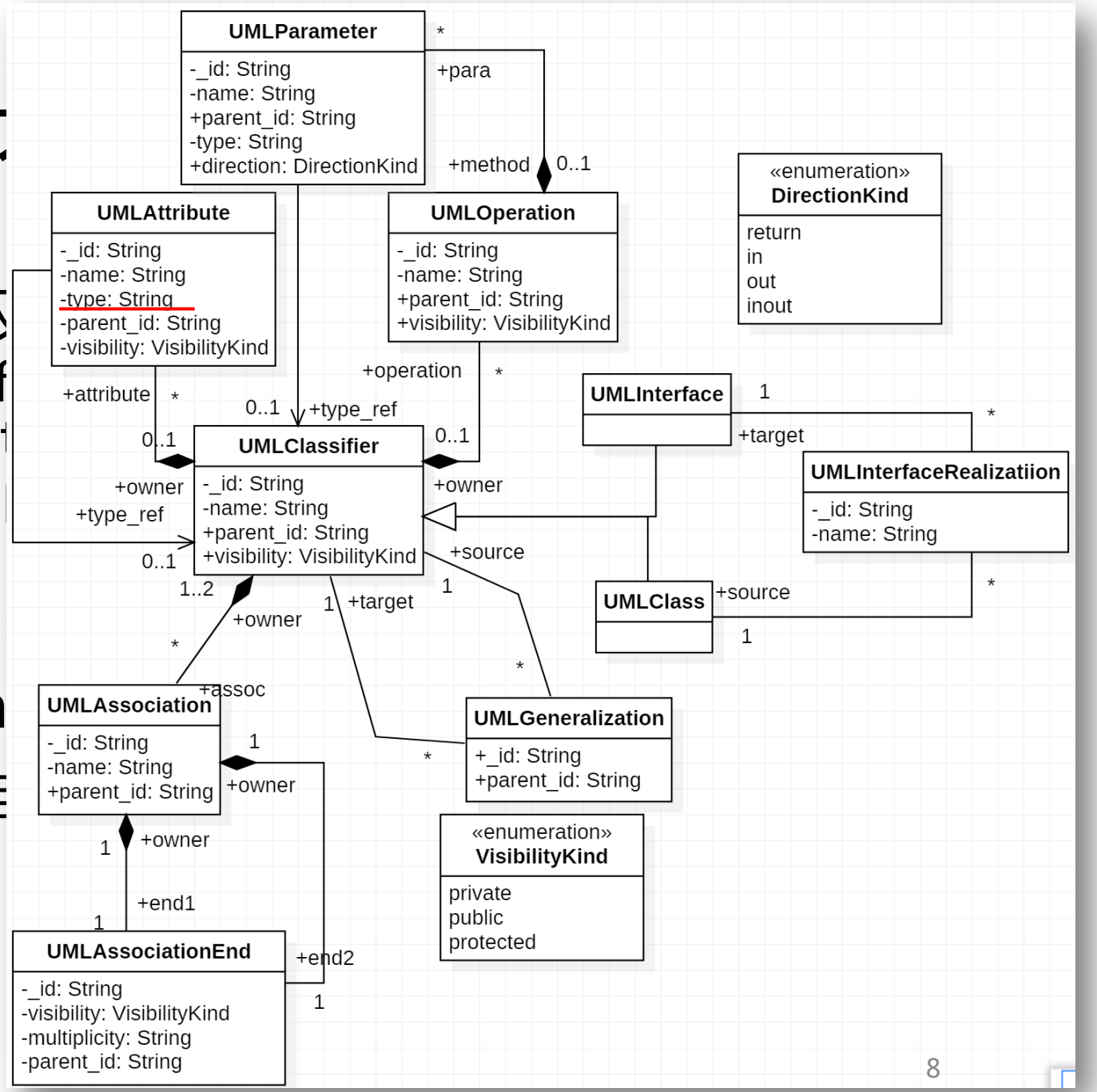


模型内容很丰富

- 初步的mdj文件结构认识
  - 多种UML***标签
  - 多种自动生成和维护的_field：_id, _type
  - 多种关系引用：src, target
  - 多个层次：parent, member, ownedElements
- 还应在解析计算层次理解
  - UML类图里有什么对象和关系
  - UML状态图有什么对象和关系
  - UML顺序图有什么对象和关系

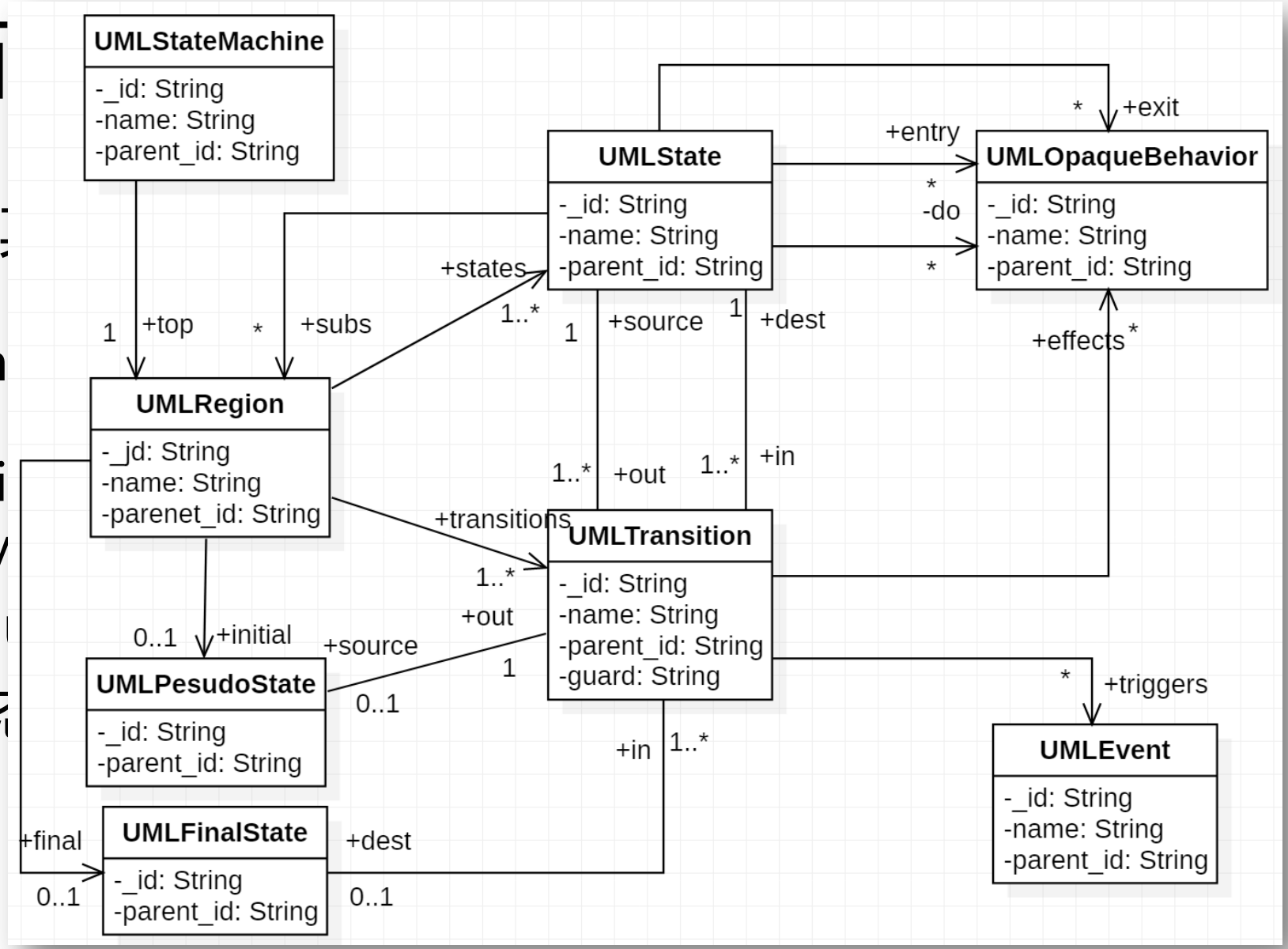
UML类图及其描述

- UML类图提供了一个描述类及
  - 顶层：UMLClass，UMLInterf
    UMLGeneralization，UMLIn
  - 下一层：UMLAttribute，UM
    UMLAssociationEnd
  - 再下一层：{<property, value
- 可以从输入的{<property, va
- 在graph上可以进行查询和推



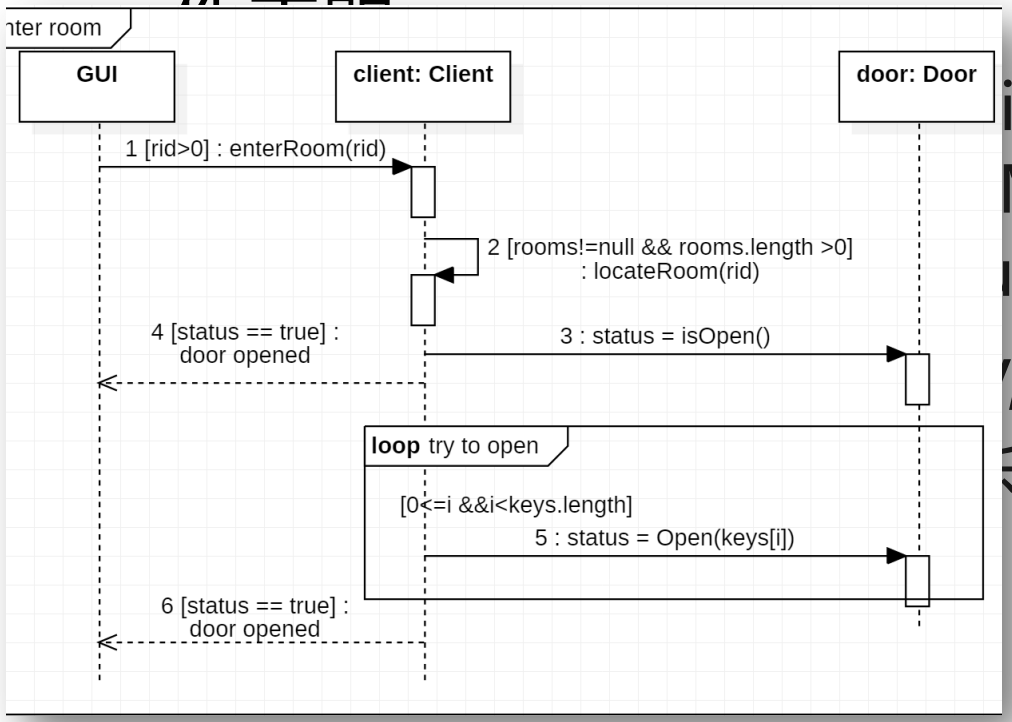
UML状态图描

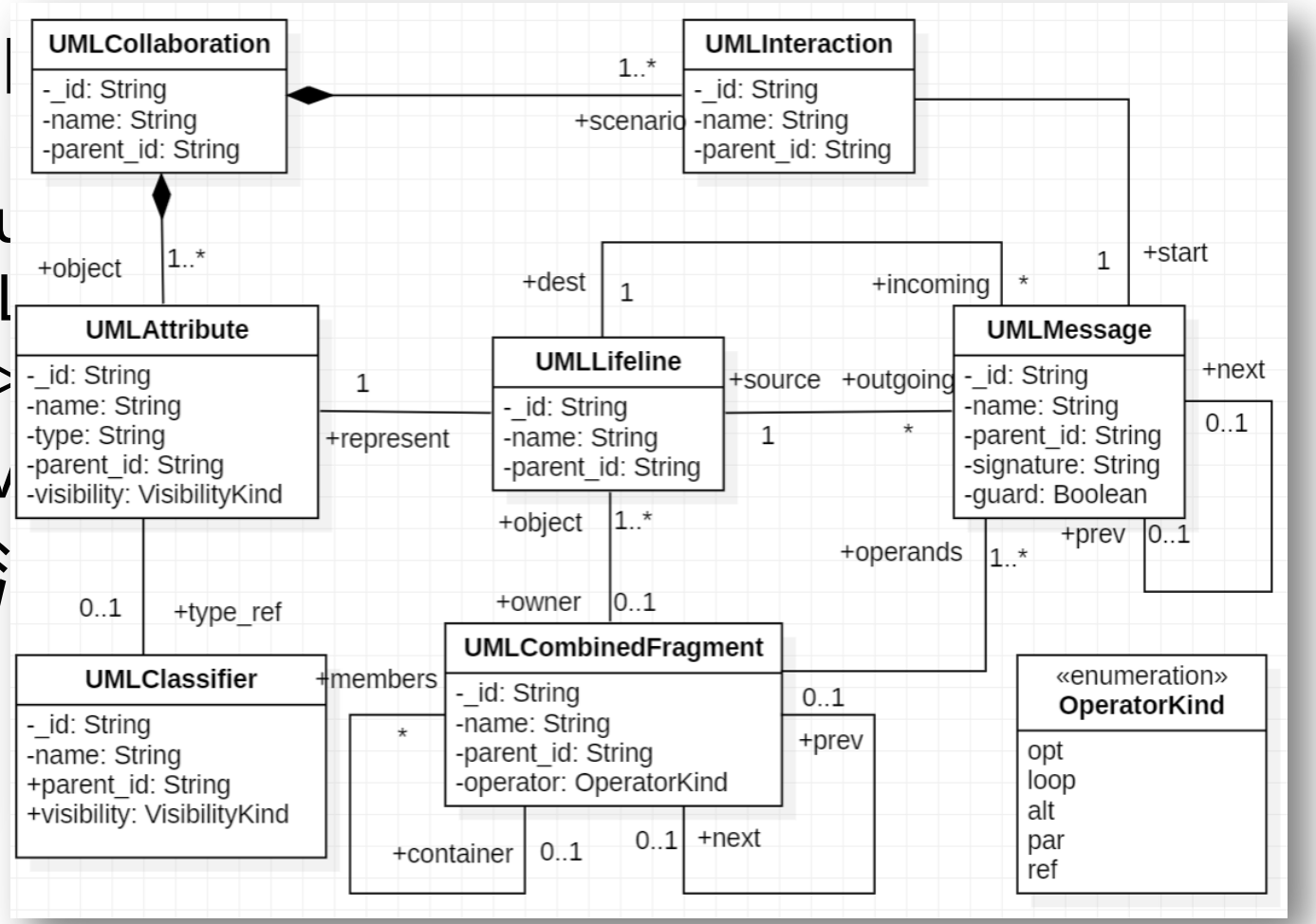
- 状态图描述了状态及
  行为
  - 顶层：UMLStateMa
  - 下一层：UMLState,
    UMLOpaqueBehavi
  - 最下层：{<property
- 依据{<property, val
- 在graph可以进行状



UML顺序图描述的内容

- 顺序图描述了基于消息机制



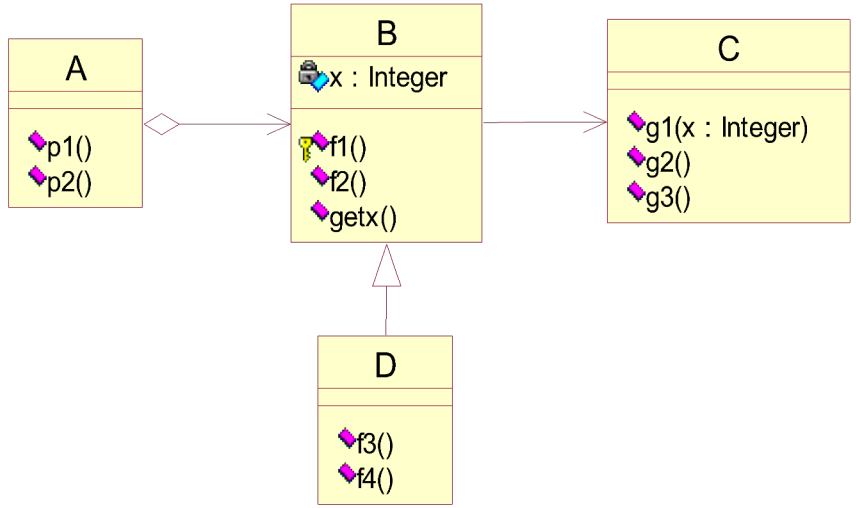


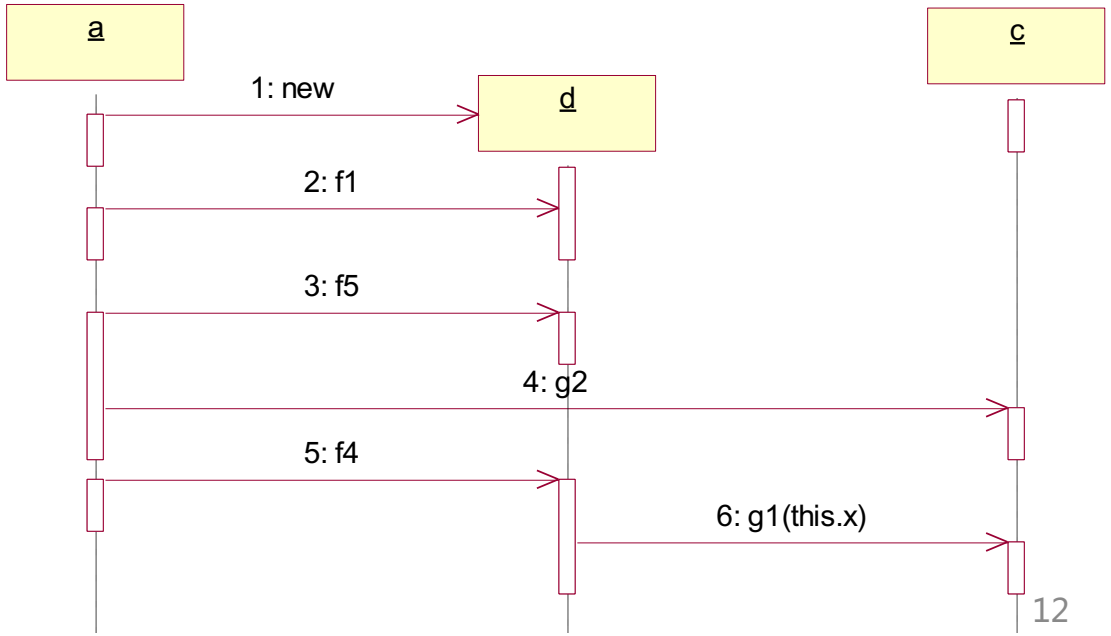
类图内容与顺序图内容的关系

- 基础：顺序图中的UMLAttribute引用到类图中定义的类
- 推导：每个发送给UMLLifeline的UMLMessage都带来一个问题
  - 该UMLLifeline关联的UMLAttribute是否能够处理？
- 从OO角度来看
  - 消息是一种交互机制，映射到消息receiver的operation
  - 同步operation→messageSort == synchCall
  - 异步operation→messageSort == asynchCall
- starUML提供了一个signature属性，用来建立这种连接关系

讨论1: 指出不一致

- 检查顺序图中的消息与类图中的相关内容的一致性
- 检查规则
  - Sender对象与receiver对象之间是否有关联？
  - 消息的signature与receiver提供的operation是否匹配？
  - receiver对象的相应operation能否被外部访问？

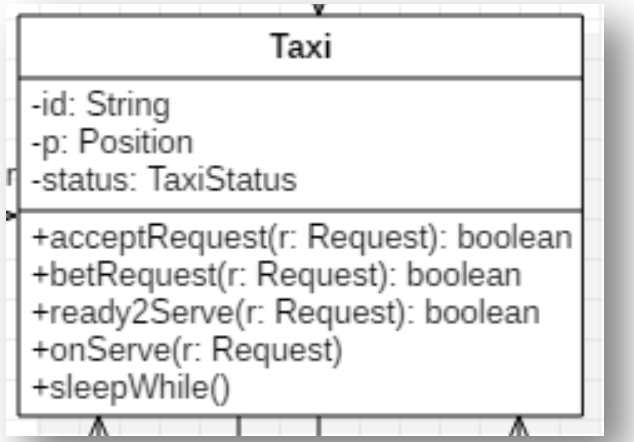


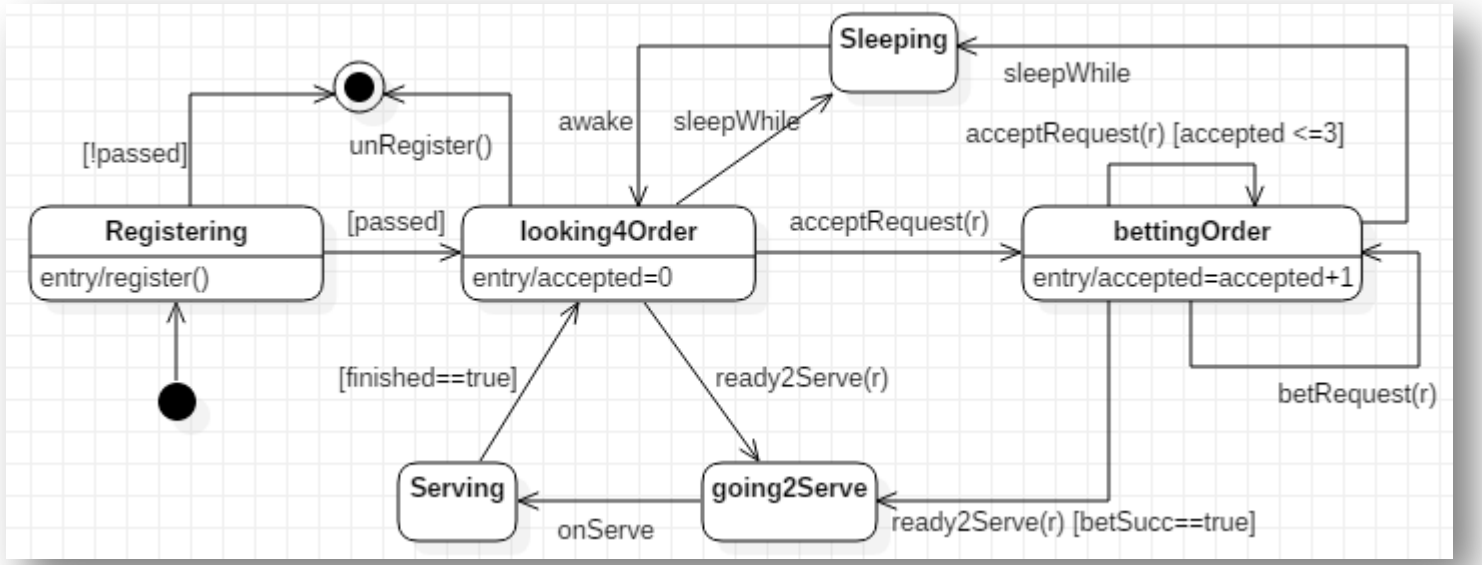


类图内容与状态图内容的关系

- 基础：状态图表示一个类的行为
- 推导：状态图中的内容必然都和相应类中的内容对应起来
  - 状态行为：所在类的行为
  - 状态迁移触发：所在类的行为
  - 状态迁移守护：对所在类属性数据取值的检查
  - 状态迁移效果行为：对其他类行为的触发

讨论2：指出不一致

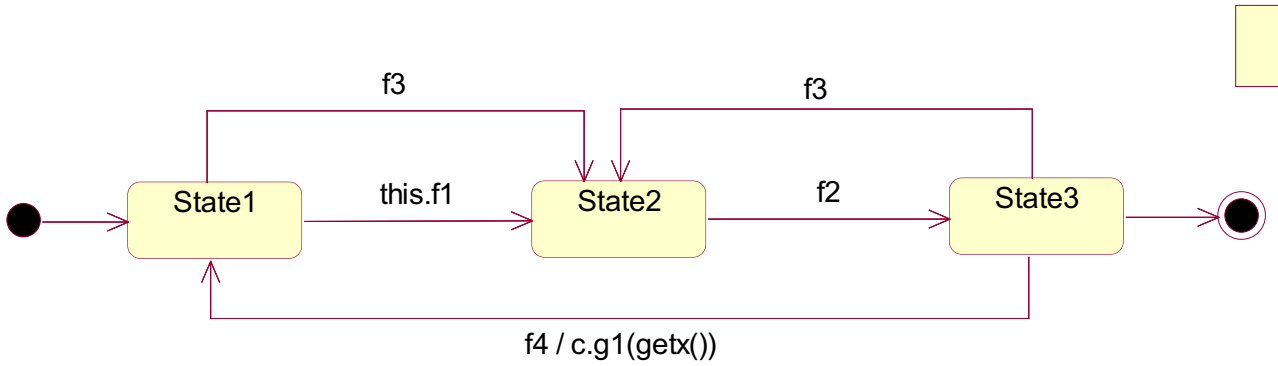

Taxi
-id: String
-p: Position
-status: TaxiStatus
+acceptRequest(r: Request): boolean
+betRequest(r: Request): boolean
+ready2Serve(r: Request): boolean
+onServe(r: Request)
+sleepWhile()

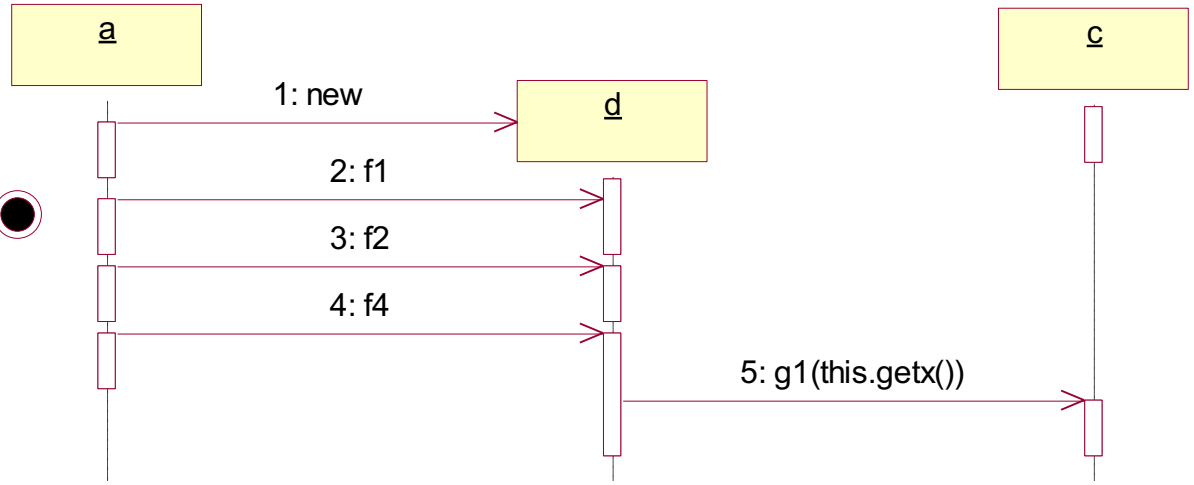

Sleeping
sleepWhile
awake
sleepWhile
acceptRequest(r) [accepted <=3]
[!passed]
unRegister()
Registering
entry/register()
[passed]
looking4Order
entry/accepted=0
acceptRequest(r)
bettingOrder
entry/accepted=accepted+1
[finished==true]
ready2Serve(r)
betRequest(r)
Serving
going2Serve
onServe
ready2Serve(r) [betSucc==true]

顺序图内容与状态图内容的关系

- 基础：顺序图描述多个对象之间的交互行为
  - 消息与对象操作关联起来
- 事实：在消息交互过程中，对象状态可能会发生变化
  - 消息接收时所处状态
  - 消息处理后所处状态
- 推导：在给定状态下是否能够响应的消息？
  - 对照状态图进行检查

讨论3: 指出不一致





- 检查对象d是否具备处理这些消息的能力
  - 顺着消息连接检查接收消息的对象当前处于什么状态
  - 对照状态图检查在相应状态下是否可以响应发送来的消息

模型的有效性问题

- 模型有效性是建模中的一个核心问题
  - 每个图中的元素有效
  - 不同图中的元素之间如果**关联**，相关属性或内容应该一致
- 不一致的模型会导致最终实现的系统无法集成，或者运行时出现莫名其妙的错误
- 模型的有效性是个复杂问题
  - 课程目标：在建模实践中遵循规则避免出现此类问题

模型的有效性问题

- 是个可判定问题
  - 需要定义清楚判定规则
- 举例：类操作定义与使用的不一致
  - 类A只提供了操作func
  - 类B关联到类A，在一个顺序图中给A对象发消息，对应操作为func1
- 举例：循环继承带来的无效继承范围
  - 类A定义了属性x
  - 类B继承了类A，定义了属性y
  - 类A也继承了类B

关于顺序图的检查规则

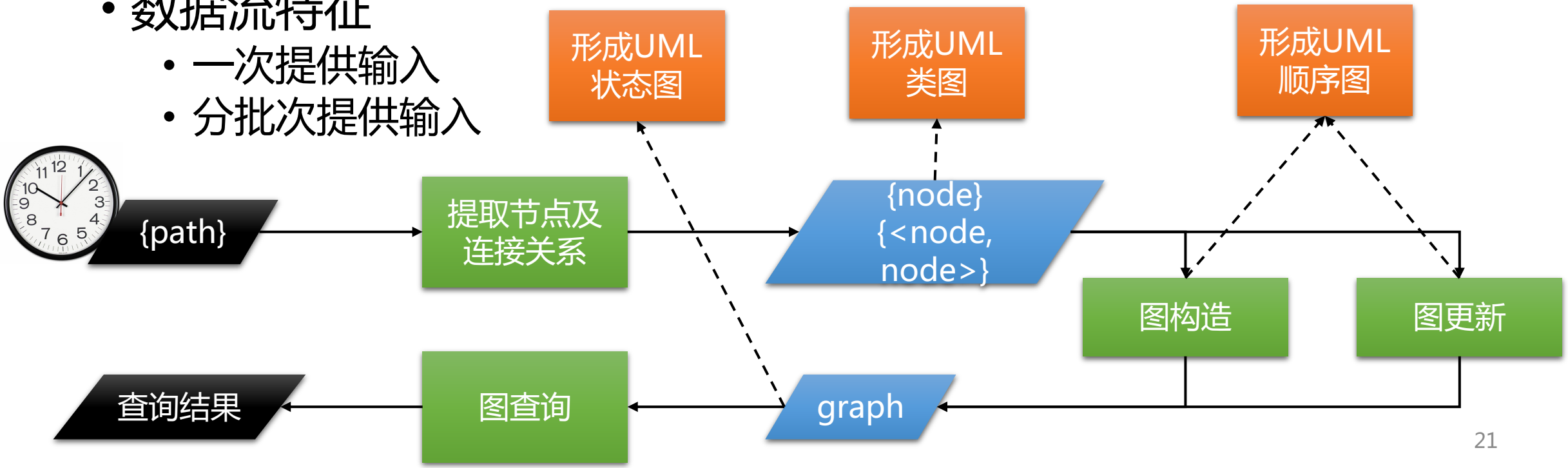
- R1: 对于需求中给定的场景，顺序图中应存在相应的消息路径来实现该场景
  - 给定场景：<起始消息a，结束消息b>
  - 消息路径：消息序列，每条消息的 source 为前一条消息的 target
  - 消息序列中的第一条消息为a，结束消息为b
- R2: Lifeline 应与代码中某个类的名称相同
  - 命名规则：“obj:class”|“obj”|“:class”
- R3: 任何一个message 名称应与target所对应类的某个方法相同
  - R3.1: 如果source与target所对应的类不同，相应方法不能为private
- R4: 任何一个message的soure和target所对应两个类之间有从source到target的关联关系

本单元的训练设计目标

- 总目标
  - 引入基于UML的正向建模，MBSE的核心能力
  - 通过正向建模来梳理项目需求中“繁琐”的逻辑结构→形成架构设计
- 有效的行动步骤
  - 识别和梳理**数据抽象**，建立关系，形成初步的类图
  - 识别和梳理**流程场景**，建立顺序图，完善类的职责和关联关系
  - 识别和梳理**关键类的状态和迁移**，建立状态图，完善类的数据和行为
  - 基于三种图来动态“执行”系统业务场景，发现不一致问题
  - 基于三种图来构造测试用例
  - 基于三种图来实现程序和开展测试

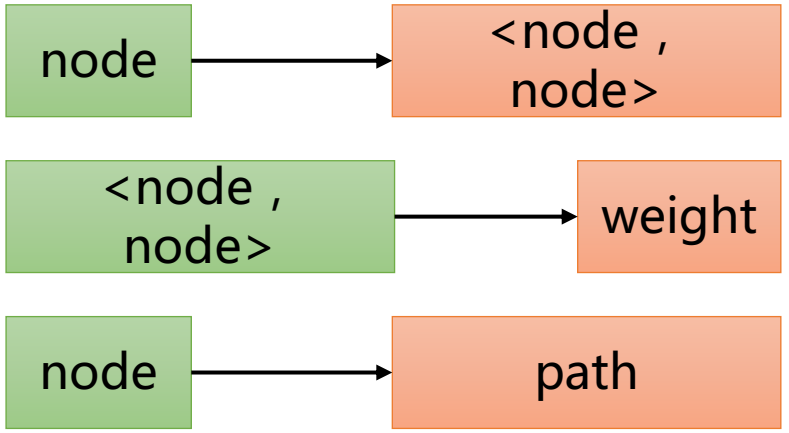
数据流视角的模型化设计

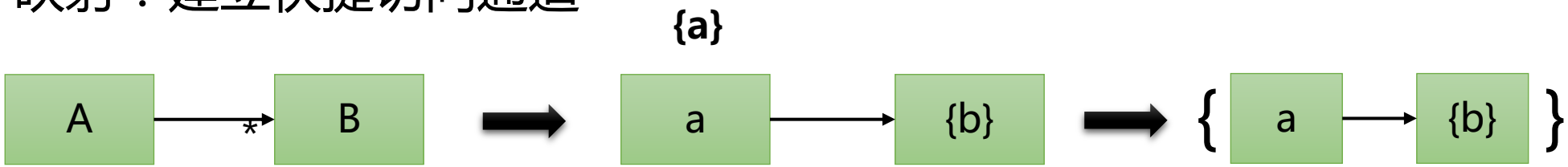
- 基于功能数据流的结构设计
  - 数据流分析是个重要的功能结构分析手段
  - 识别模块及数据依赖关系
- 数据流特征
  - 一次提供输入
  - 分批次提供输入



数据流视角的模型化设计

- 数据流程视角识别出了数据的结构
  - 输入、中间数据和输出
- 围绕数据设计相应的操作
- 可以按照第7讲所介绍的对象分析与设计方法来整理数据模型
  - 形成类、形成数据容器
- 数据模型：数据之间的关联和映射
  - 关联：建立访问通道
  - 映射：建立快捷访问通道





实践中的模型化设计应用

- MBSE(Model Based Software/System Engineering)
- 应用好的必要条件
  - 使用合适的语言/图，仅使用需要的
  - 有易于使用的工具链
    - 模型管理
    - 模型检查
    - 模型验证
  - 有丰富的模型库
    - 领域模型
    - 通用处理模型

实践中的模型化设计应用

- 三大障碍
  - 过高的预期
    - 银弹传说
  - 技术人员缺乏必要的建模思维训练
    - 建模不是过家家游戏
  - 技术转型不够平滑
- 关于技术转型
  - 能否兼容已有的表示法
    - DSL(Domain Specific Language)的设计
  - 能否重用已有设计
    - 组件库+潜在设计策略

实践中的模型化设计应用

- 随着时间的推移，模型与系统实际行为偏差会越来越大
  - 模型在抽象层次描述系统的预期行为，与系统实际运行产生的具体行为之间存在偏差具有必然性
    - 关键是这种偏差是否影响模型对系统行为的表示和推理分析效果
  - 开发时的偏差放大
    - 开发过程中代码实现会增加很多细节并不断变化
  - 运行时的偏差放大
    - 系统运行时处理的数据在不断变化
    - 在某个时候偏差大到模型不具备对系统的表示和分析能力
- 如何确保模型与系统实际行为的**一致性**近年来的热点研究问题
  - 对模型进行演化

UML模型服务于测试

- UML模型整合了解决方案结构、行为、功能和部署，也整合了设计规约
- UML模型同样为测试提供了依据，定义了
  - 测试需要覆盖的流程
  - 测试需要覆盖的状态迁移路径
  - 测试需要覆盖的对象协同
  - 测试数据及其关联关系
  - 并发场景和行为
- 基于模型的测试MBT(Model-Based Testing)
  - 这可能是MBSE在实践中应用效果最好的一个技术

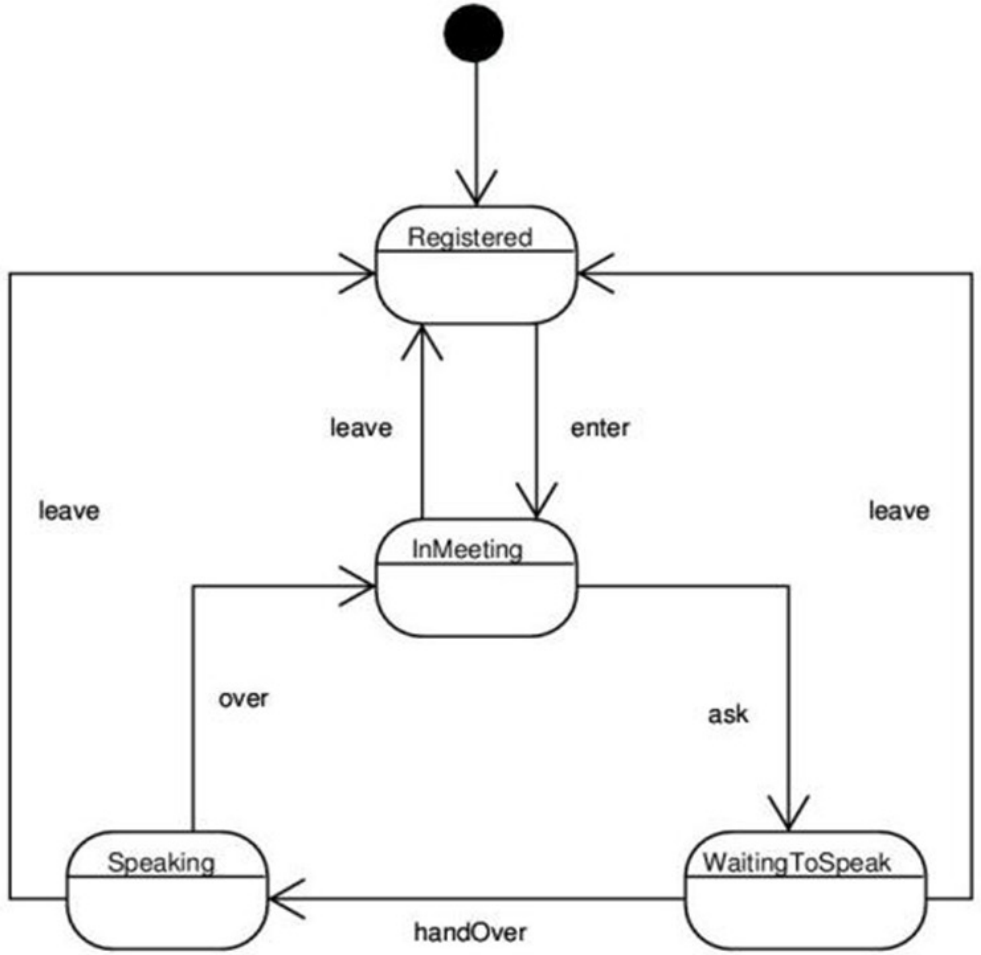
MBT—基于状态图的测试

- 基于状态图的测试
  - 输入: UML状态机模型
  - 输出: 迁移序列/状态序列

enter-leave
enter-ask-leave
enter-ask-handover-leave
enter-ask-handover-over-leave

What is the test strategy?

Where is the test data?



MBT—基于状态图的测试

- 状态图定义了对象状态及其迁移路径
- 在各个层次的测试中都发挥着重要作用
  - 单元测试：关注一个类的控制行为测试
  - 模块测试：关注一个组件的行为测试
  - 系统测试：关注整个系统的行为测试
- 测试用例是对状态图进行遍历的结果
  - 覆盖策略：状态覆盖、迁移覆盖、简单迁移路径覆盖
- 给定测试用例，测试数据是满足相应trigger和guard的求解结果

MBT—基于顺序图的测试

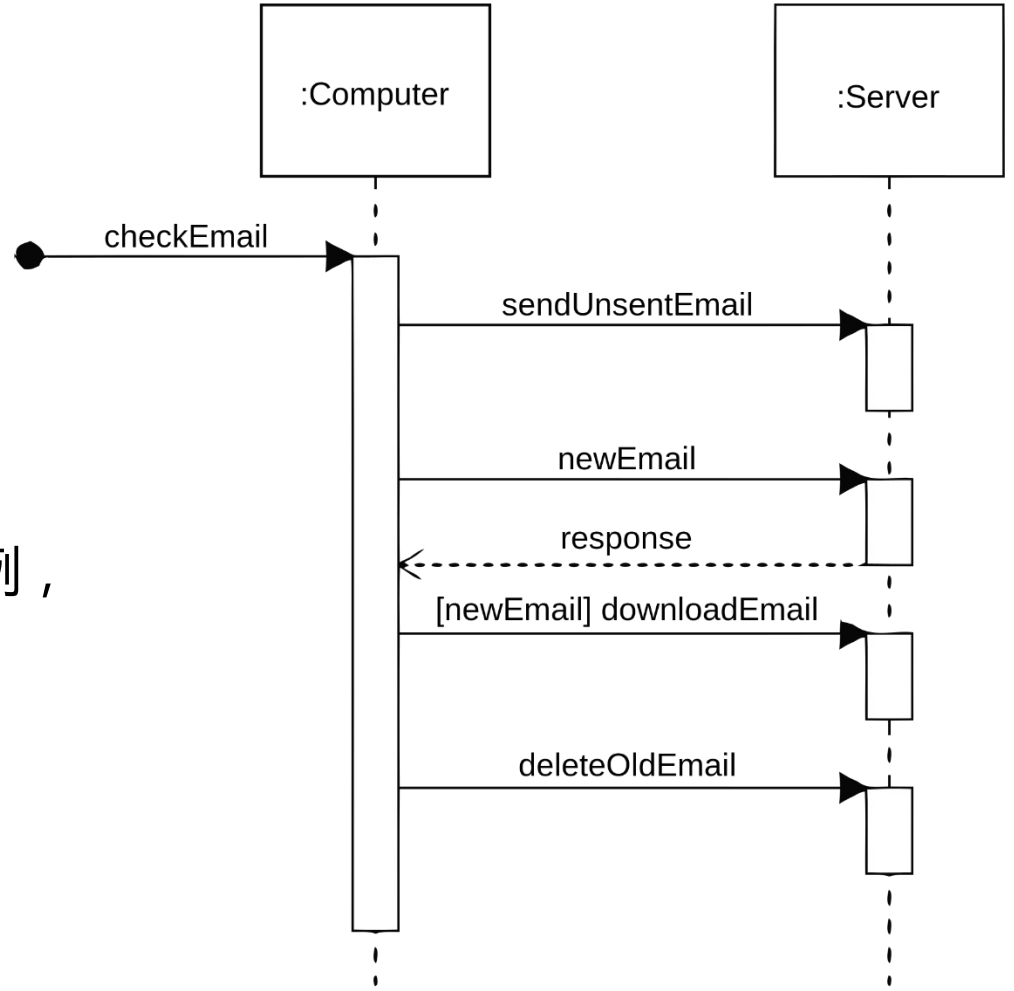
- 基于顺序图的测试
  - 输入: 顺序图模型
  - 输出: 消息序列(测试用例)

1: 通过UMLAttribute确定参与协同的对象：准备相应的对象

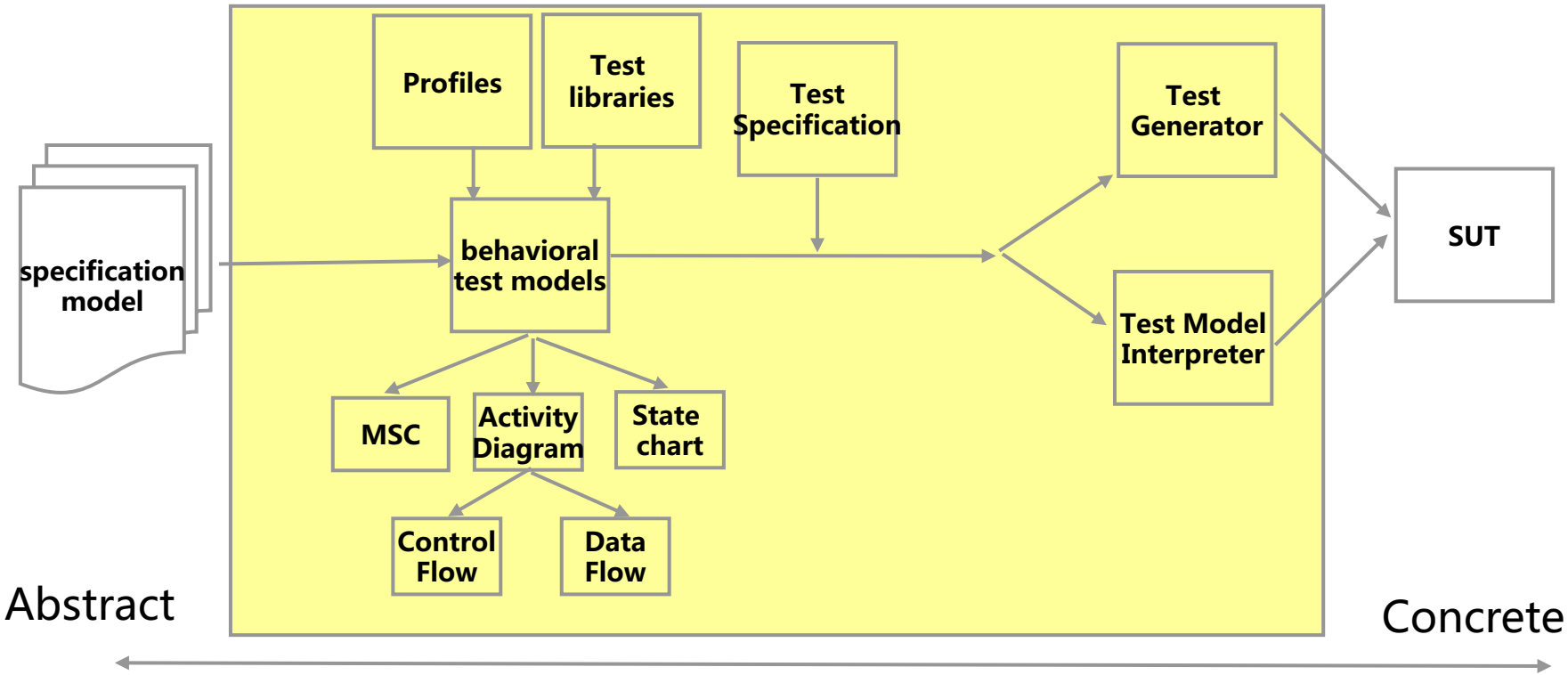
2: 选择目标对象，以incoming消息来构造测试用例，使用对象的outgoing消息来检查执行效果

3: 为相应的消息和对象准备好数据，消息数据，对象状态

Q: 如何测试多个对象之间的协同？



MBT的工作原理与架构



课程主题回顾

- 设计
  - 基于对象词汇的架构思维，**建立层次**
  - 针对问题特征的解决方案**规划**
  - 问题演化下的解决方案**重构**
- 构造
  - 如果不能自己做出来，就不会有真正的技术掌控力
  - 构造过程的自觉化
  - 测试是质量的守护

课程主题回顾

- 架构思维
  - 直接奔向代码战场的结果，常常遍体鳞伤，甚至铩羽而归
  - 架构思维的形成往往始于发现自己的代码**不能适应需求的变化**
- 解决方案
  - 问题特征➔架构+核心数据结构+算法考虑
  - 架构把数据结构组织起来
  - 算法针对数据特征和功能特征给出计算流程
- 重构
  - 轻量级：调整算法
  - 中量级：调整局部结构
  - 重量级：调整全局结构

软件设计的七大难点

- 功能定义了软件的输入和输出及其映射关系
- 难点1：输入有多种形态
  - 如何处理不同形态所对应的结构，识别其中的内在关系和约束
- 难点2：输入到输出的距离有些远且忽远忽近
  - 必须在中间搭桥
  - 桥的结构往往决定了程序的动态伸缩能力
- 难点3：多次输入之间具有逻辑联系
  - 每一个输入都可能会对系统的数据模型产生影响，必须进行动态调整
  - 应区分出**变**与**不变**

软件设计的七大难点

- 难点4：连续性输入，不断输出
  - 并发处理结构，既独立又协同，安全保护问题
- 难点5：不只是能够产生输出，还有性能要求
  - 算法设计必须和数据模型设计配合起来
- 难点6：存在各种样式的异常输入
  - 准确区分异常输入和正常输入，识别和防范处理
- 难点7：需求容易发生变化
  - 增加输入形态
  - 调整已有的输入到输出映射关系
  - **预见**输入形态的可能变化，识别并控制变化影响范围

作业解析

- 轻量级功能扩展
  - 引入图书的借阅期限属性，以及逾期归还的违规行为
- 要求围绕指导书的业务流程，从对象协作角度设计协作模型，形成UML顺序图
  - 注意是针对本次作业中的所有内容，包括前两次已经实现的流程
- 引入顺序图评测
  - 类图评测、状态图评测仍然继续
  - 三种图的一致性是重点
- HW15初步定于周四发布

课程总结博客

- 总结本单元所实践的正向建模与开发
- 总结本单元作业的架构设计，并对比分析最终的代码设计和UML模型设计之间的追踪关系
- 总结自己在四个单元中架构设计思维的演进
- 总结自己在四个单元中测试思维的演进
- 总结自己的课程收获
- 给课程提1~3个具体改进建议