《面向对象设计与构造》Lec15：确保模型质量

OO2022课程组

北京航空航天大学计算机学院

目录

- 回顾课程主题
- 软件设计难在何处
- 解析软件设计
  - 视图与模型
- 回顾课程的设计训练
- 模型化设计的思维方法
- UML用于软件测试
- 本周作业解析
- 课程总结博客作业

课程主题回顾

- 设计
  - 基于对象词汇的架构思维
  - 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

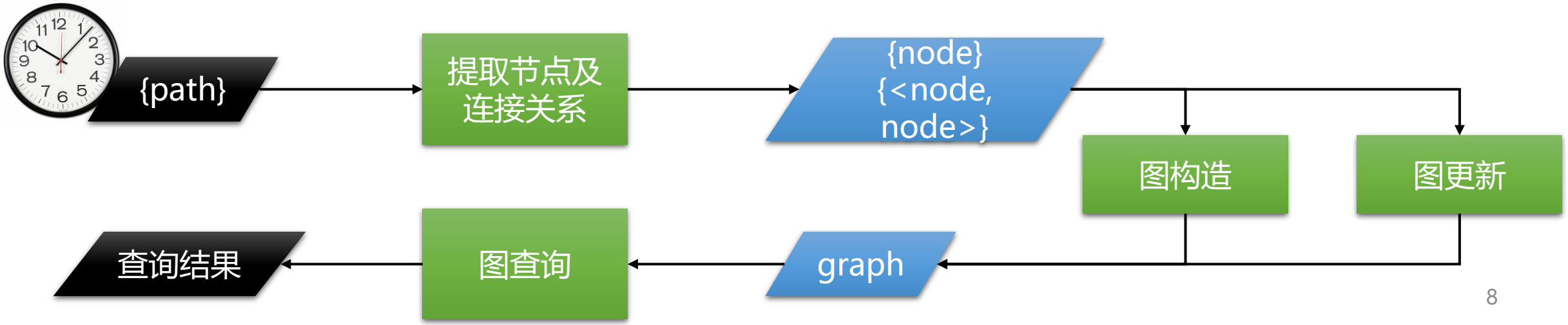
- 难点4：连续性输入，不断输出
  - 并发处理结构，既独立又协同，安全保护问题
- 难点5：不只是能够产生输出，还有性能要求
  - 算法设计必须和数据模型设计配合起来
- 难点6：存在各种样式的异常输入
  - 准确区分异常输入和正常输入，识别和防范处理
- 难点7：需求容易发生变化
  - 增加输入形态
  - 调整已有的输入到输出映射关系
  - 预见输入形态的可能变化，识别并控制变化影响范围

解析软件设计

- 结构视图
  - 模块及其接口
  - 模块间依赖关系
- 数据视图
  - 数据类
  - 抽象层次关系
  - 关联关系（数据管理层次）
- 行为视图
  - 模块间的交互行为
  - 算法流程

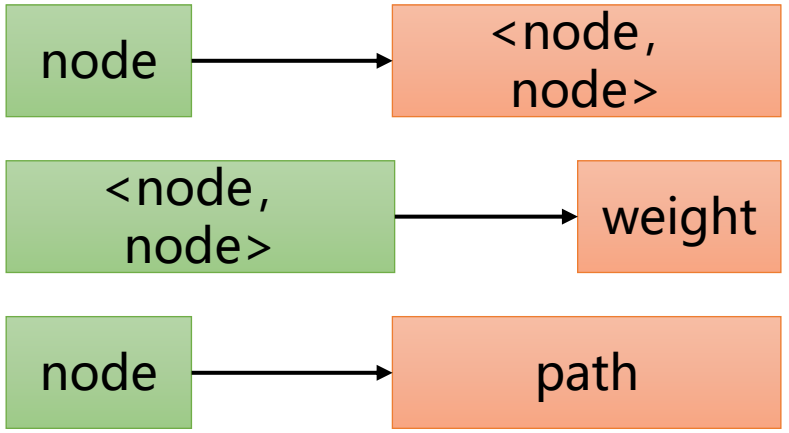
解析软件设计

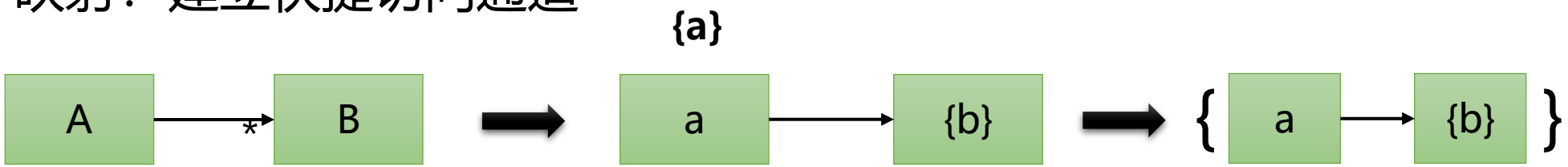
- 基于功能数据流的结构设计
  - 数据流分析是个重要的功能结构分析手段
  - 识别模块及数据依赖关系
- 数据流特征
  - 一次提供输入
  - 分批次提供输入



解析软件设计

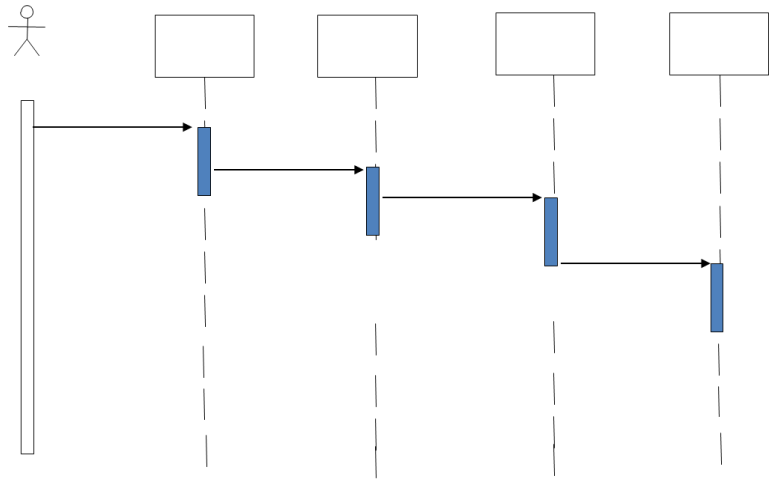
- 数据流程视角识别出了数据的结构
  - 输入、中间数据和输出
- 围绕数据设计相应的操作
- 可以按照第7讲所介绍的对象分析与设计方法来整理数据模型
  - 形成类
  - 形成数据容器
- 数据模型：数据之间的关联和映射
  - 关联：建立访问通道
  - 映射：建立快捷访问通道

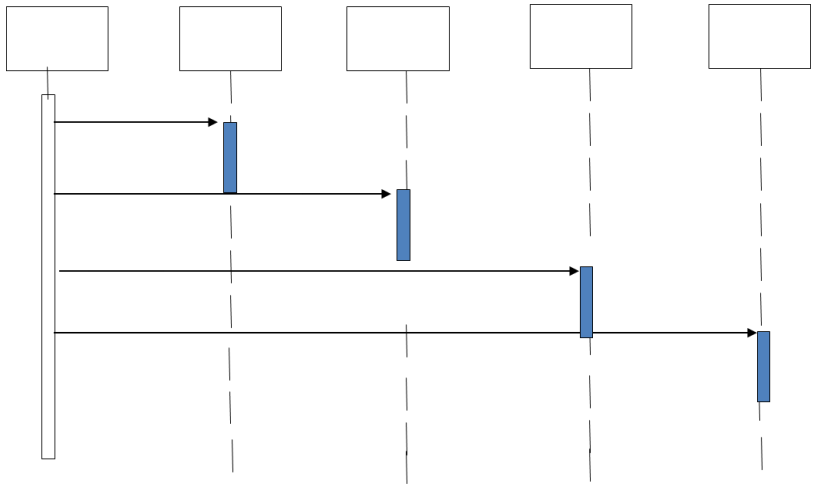




解析软件设计

- 在模块间建立交互协同行为
  - 逐层代理调用
  - 中心控制调用
- 合理分配行为职责，保持各模块的行为职责平衡
- 确定关键模块





回顾课程的设计训练

- 第一单元
  - 逐步引入不同的项、组合规则、计算规则
  - 从线性字符串输入识别出语义上的层次关系
- 核心设计目标：构造抽象层次，进行归一化处理
  - 在需求变化中保持架构的稳定
- 优化对设计提出了灵活性要求
  - 基本要求：同类项的合并(加减合并)
  - 提高要求：同型项的融合
    - sin^2+cos^2 = 1，sin^4-cos^4=sin^2-cos^2, …
  - 如何识别同类项，并归在一起?
  - 如何识别同型项，并构造融合规则?

f(x)*sin^2+f(x)*cos^2 = ?
f(x) – f(x)*cos^2 = ?

回顾课程的设计训练

- 第二单元
  - 逐步引入并发成分、不同的运载规则➔调度机制的适应性
  - 随着线程交互的频繁，其安全问题成为关注点
- 核心设计目标：识别线程与共享数据，以及共享访问模式，建立线程安全的协作结构
  - 生产者-消费者模式、订阅-发布模式、流水线模式、事件触发模式
  - ➔在线程及共享数据之间建立层次关系
- 要点1：轻线程体设计+均衡的层次化共享数据设计
- 要点2：尽可能小的同步控制范围
- 要点3：按照策略的调度算法及其应用场景

回顾课程的设计训练

- 第三单元
  - 逐步引入JML规格、增加数据模型复杂度
  - 数据复杂度（规模和连接关系）的增加带来了性能挑战
- 核心设计目标：基于规格准确捕捉功能语义，结合数据模型建立中间层数据模型和协同架构
  - 建立图数据结构和图模型层次
  - 在数据之间建立映射关系，而不只是简单的关联关系
  - 逐层构造，顶层实现规定的接口和相应的规格
- 性能和架构设计具有了更紧密的关系
  - 桥梁：图模型+中间数据缓存+缓存更新机制

回顾课程的设计训练

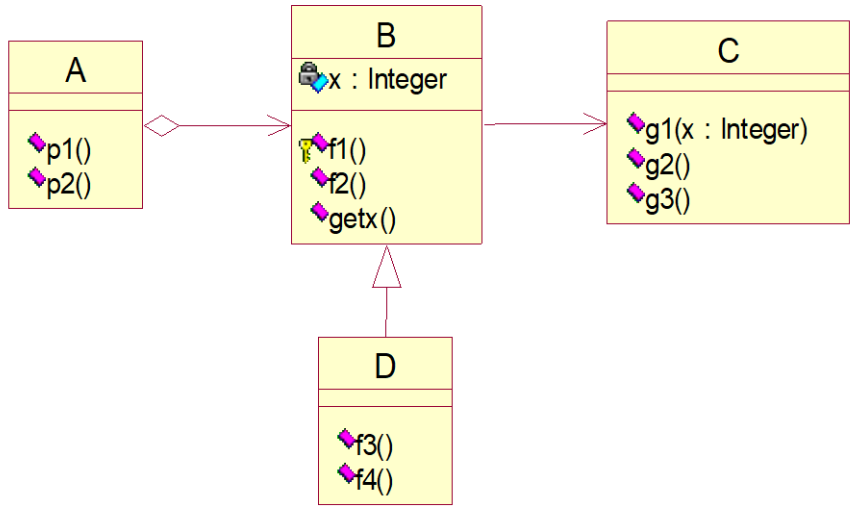
- 第四单元
  - 逐步引入UML模型的理解和解析规则、UML模型的语义规则
  - 复杂数据对象的**解析**、管理与访问
- 核心设计目标：从树形结构的对象管理关系中识别出对象的语义关系，从而构造合适的图数据模型
  - 通过图数据模型识别UML模型的语义，按照相应语义实现需求
  - 组合模式、访问者模式、MVC等
- 要点1：理解UML的关键在于其对象化的语义表达
- 要点2：格式解析与语义提取相分离，常用的架构设计方式
- 要点3：图模型的结构化搜索，提取相关对象，并进行规则检查

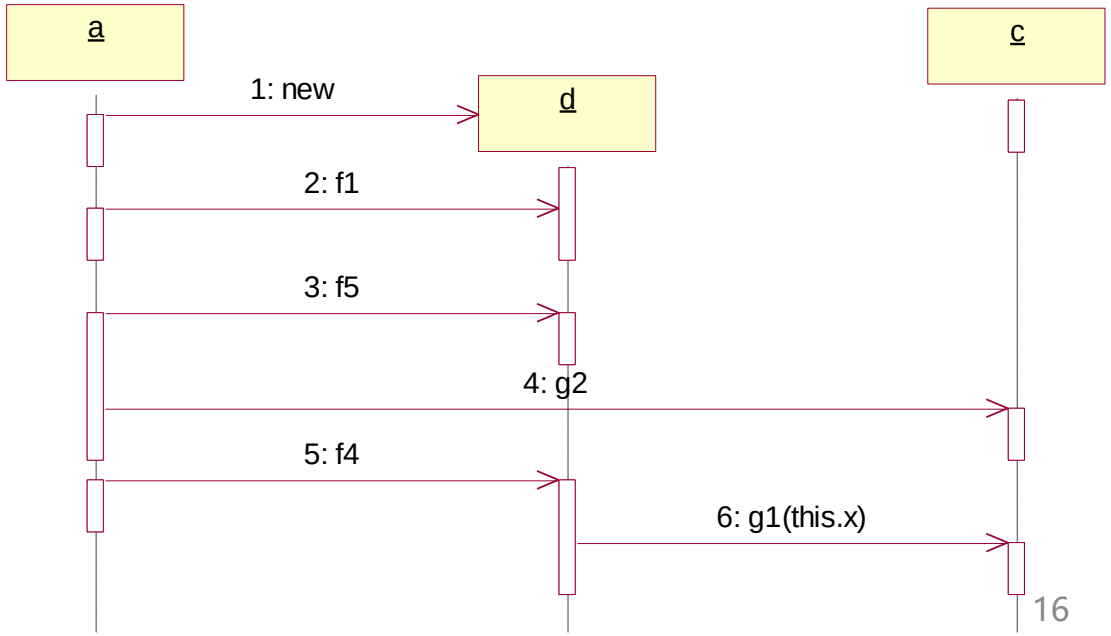
类图内容与顺序图内容的关系

- 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?

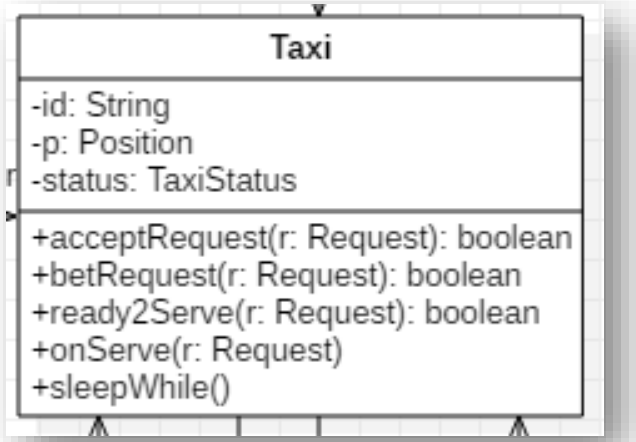


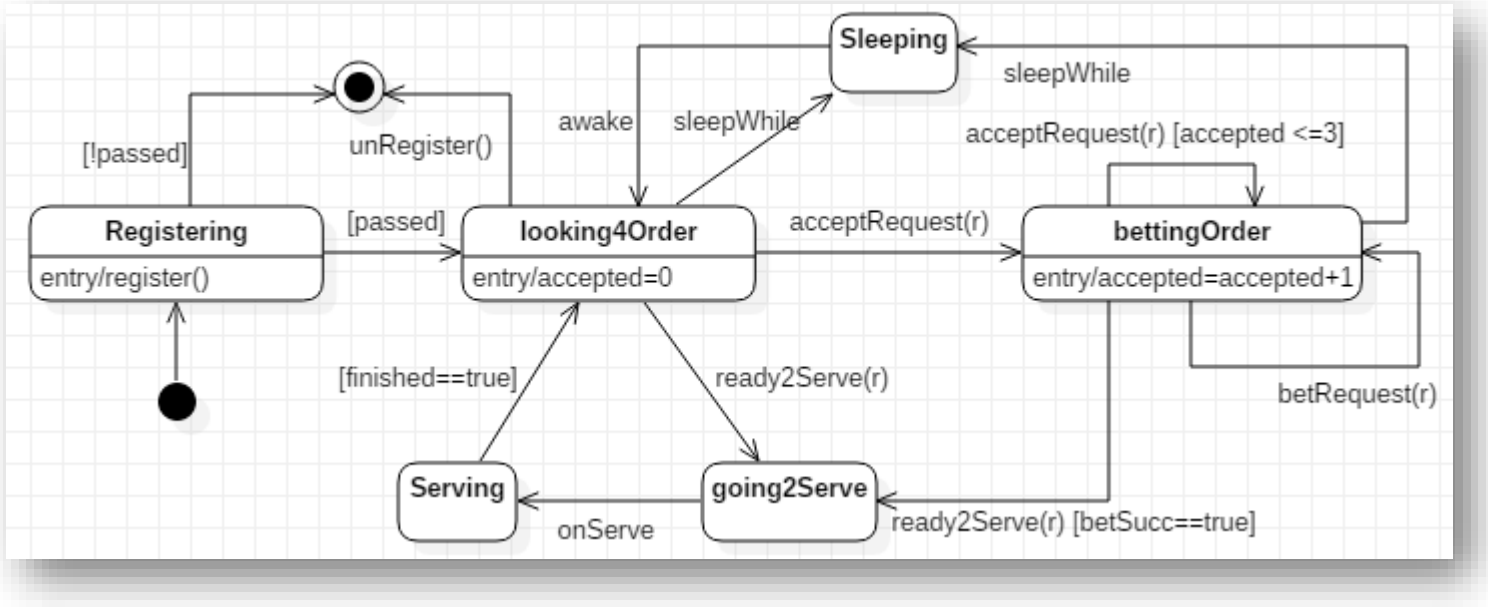


类图内容与状态图内容的关系

- 基础：状态图表示一个类的行为
  - UMLStateMachine的parent必须引用到某个UMLClass对象
- 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

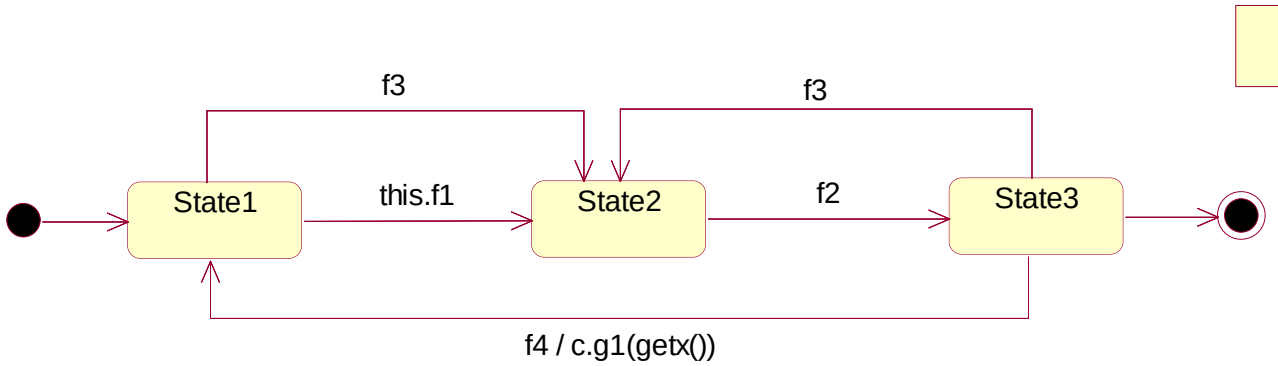

Taxi
-id: String
-p: Position
-status: TaxiStatus
+acceptRequest(r: Request): boolean
+betRequest(r: Request): boolean
+ready2Serve(r: Request): boolean
+onServe(r: Request)
+sleepWhile()

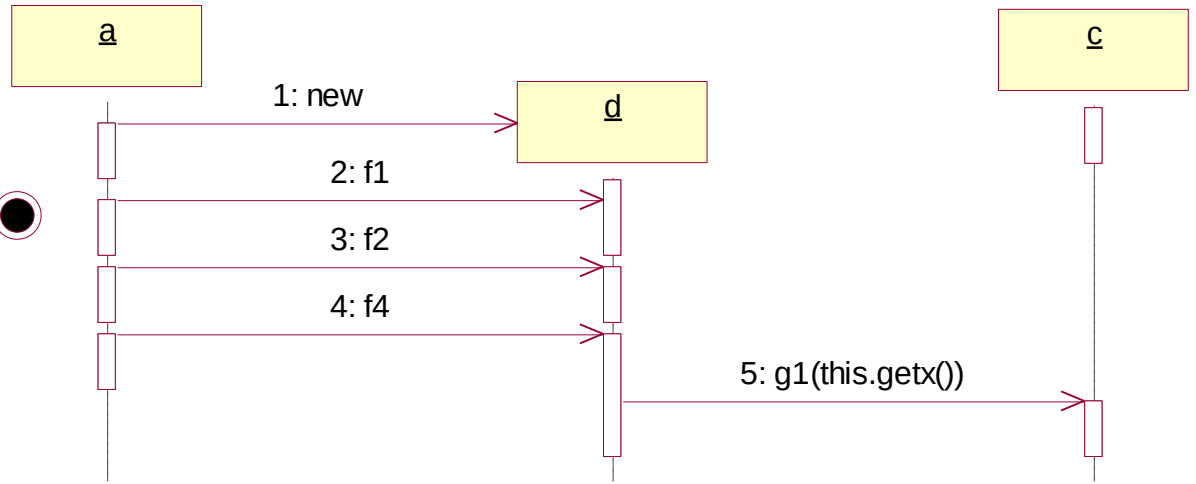

Sleeping
sleepWhile
awake
sleepWhile
acceptRequest(r) [accepted <=3]
[!passed]
unRegister()
Registering
entry/register()
[passed]
looking4Order
entry/accepted=0
acceptRequest(r)
bettingOrder
entry/accepted=accepted+1
[finished==true]
ready2Serve(r)
betRequest(r)
Serving
going2Serve
onServe
ready2Serve(r) [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
- 模型有效性和一致性是个复杂问题，同时也是一个学术研究问题
  - 课程目标：理解这个问题，并能就一些简单的规则进行推理分析

模型的有效性与一致性问题

- 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，并在一个顺序图中给A对象发送消息，对应的操作为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

关于类的检查规则

- starUML定义的几个规则:
  - (UML002) Name is already defined.
    - If element has a name, then it should be unique in the namespace.
    - Applies to: `UMLModelElement`.
    - Exceptions: `UMLOperation`.
  - (UML003) Conflict with inherited attributes.
    - Applies to: UMLAttribute
  - (UML004) Signature conflict.
    - Same signature is not allowed in a classifier.
    - →不允许在一个classifier中新定义两个signature一致的operation
    - Applies to: UMLOperation

https://docs.staruml.io/user-guide/validation-rules

关于类的检查规则

- (UML007) Duplicated generalizations.
  - Do not make duplicated generalizations from the same element.
  - Applies to: UMLClassifier.
- (UML008) Circular generalizations.
  - Do not generalize from one of the children.
  - Applies to: UMLClassifier.
- (UML009) Duplicated realizations.
  - Applies to: UMLClassifier.
- (UML010) Duplicated role names of associated classifiers.
  - Applies to: UMLClassifier

关于状态图的检查规则

- (UML021) An initial vertex can have at most one outgoing transition.
  - Applies To: `UMLPseudostate (kind = 'initial')`
- (UML022) The outgoing transition from an initial vertex must not have a trigger or guard.
  - Applies To: UMLPseudostate (kind = 'initial')
- (UML033) A final state cannot have any outgoing transitions.
  - Applies To: UMLFinalState
- (UML044) The classifier context of a state machine cannot be an interface.
  - Applies To: UMLStateMachine

关于顺序图的检查规则

- A lifeline must represent an attribute defined in the same collaboration model
  - Applies to: UMLLifeline, UMLAttribute

- A UMLMessage without explicit messageSort property specified, it must be a synchCall message
  - Applies to: UMLMessage

关于顺序图/状态图和类图的检查规则

- 应该为顺序图中的对象属性定义相应的类型，且该类型应该在类图中有定义
  - Applies to: UMLCollaboration, UMLAttribute, UMLClassifier
- 针对所有发送到一个UMLLifeline的UMLMessage (messageSort为synchCall)，则该lifeline所关联的attribute中一定存在一个UMLOperation，使得message.name=operation.name
- 状态图中的每个UMLTransition中的trigger，都应关联到context的某个UMLOperation

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能力
  - 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**是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
  - 模型定义了测试需要覆盖的流程
  - 模型定义了测试需要覆盖的状态迁移路径
  - 模型定义了测试需要覆盖的对象协同
  - 模型定义了测试数据及其关联关系
- 基于模型的测试MBT(Model-Based Testing)

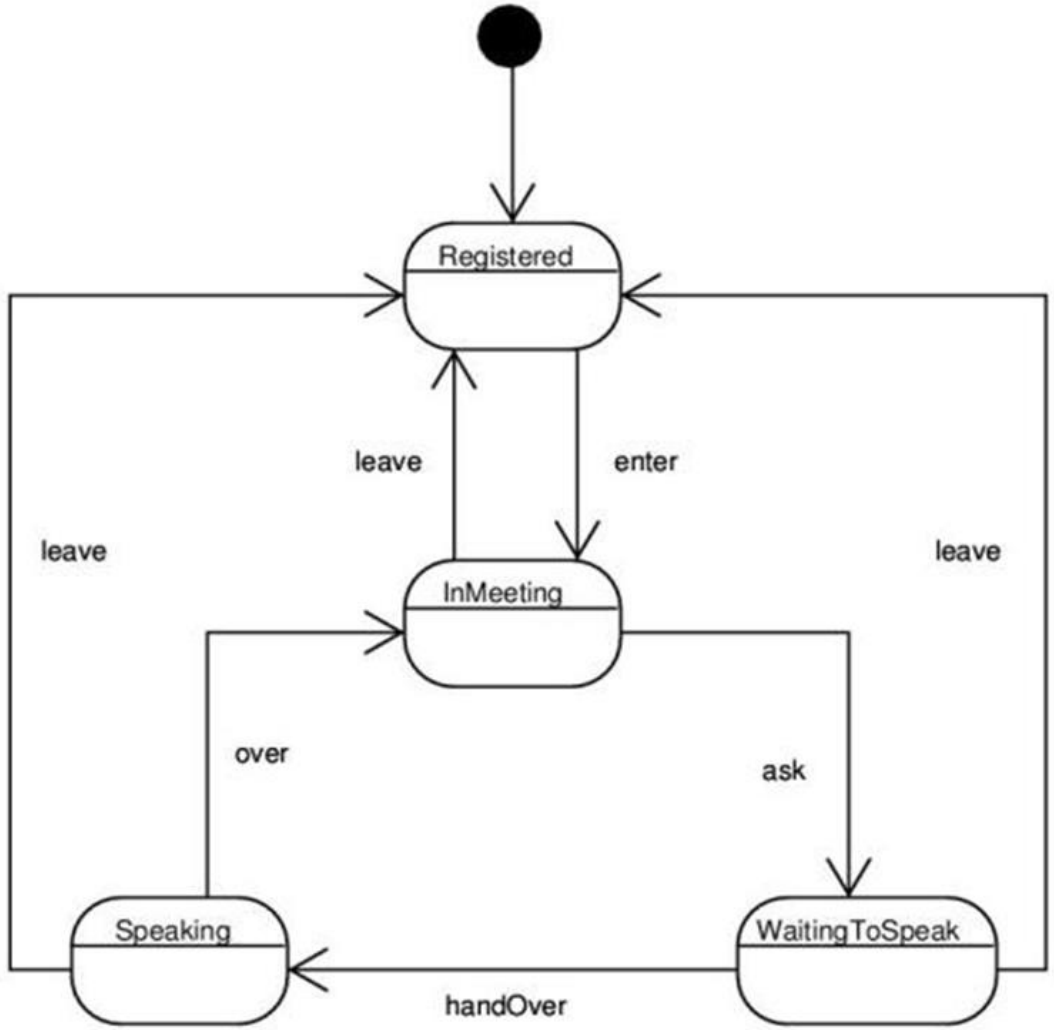
MBT—基于状态图的测试

- 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
- 给定测试用例，测试数据是满足相应trigger和guard的数据求解结果

MBT—基于顺序图的测试

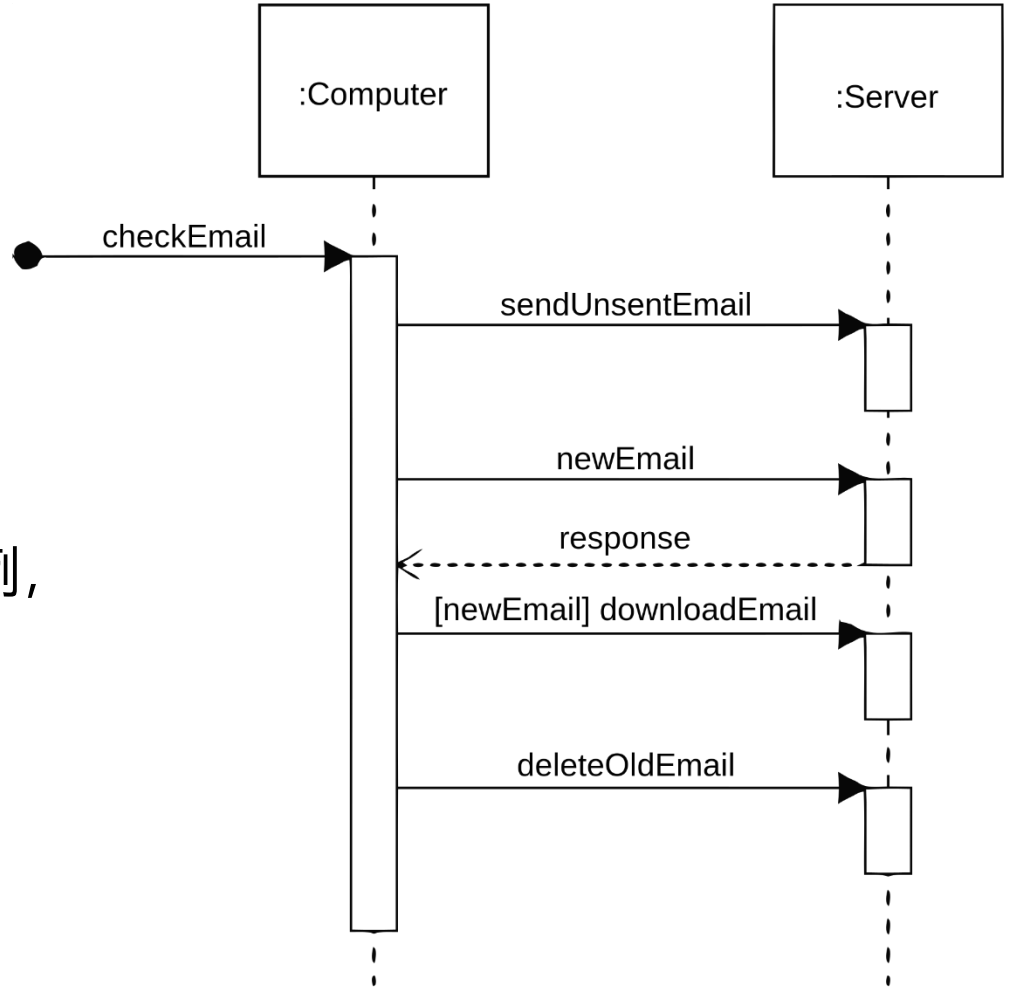
- 基于顺序图的测试
  - 输入: 顺序图模型
  - 输出: 消息序列(测试用例)

1: 通过UMLAttribute确定参与协同的对象：准备相应的对象

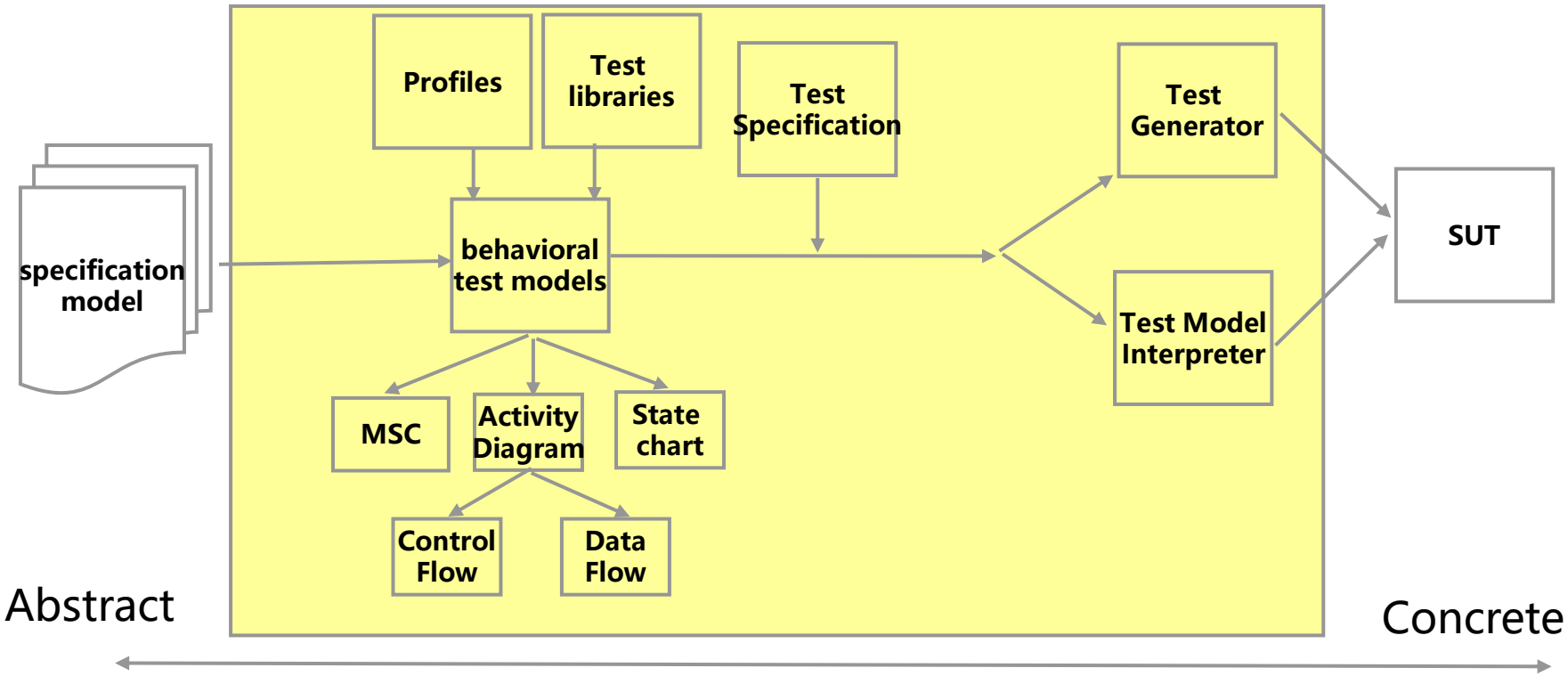
2: 选择目标对象，以incoming消息来构造测试用例，使用对象的outgoing消息来检查执行效果

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?

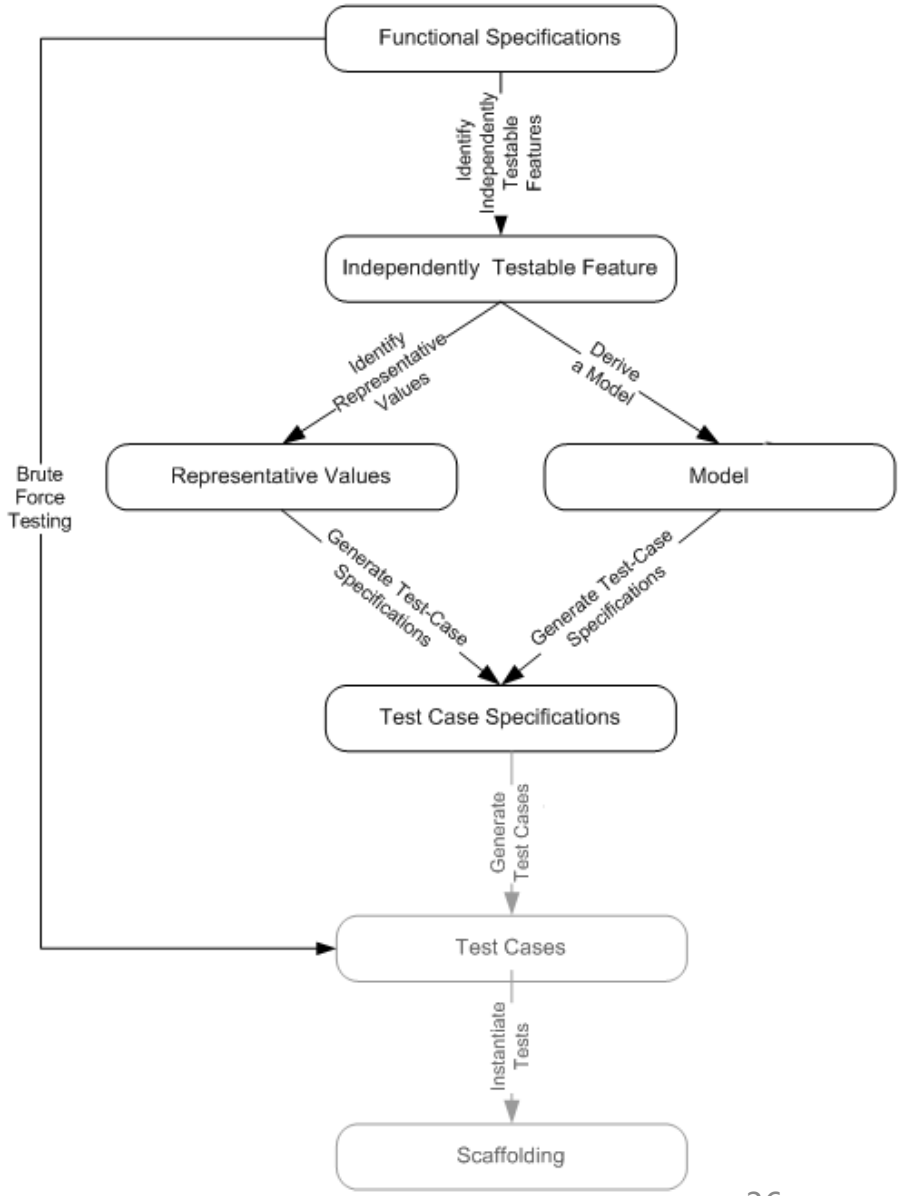


MBT的工作原理与架构



MBT的工作原理与架构

- 相比较于随机测试或暴力测试，MBT是一种效率得到了优化的测试
- 对被测对象的功能结构进行独立性分析
  - 简化覆盖空间
- 功能的代表性输入是测试划分(partition)的结果
- 功能行为模型在输入与被测对象的响应之间建立逻辑关联



Mauro Pezze and Michal Young, *Software Testing And Analysis*

作业解析

- 根据给定的检查规则，对输入的UML模型进行检查，报告检查结果
  - 准确理解规格的语义
  - 梳理每个规则需要检查的对象和有效性约束条件
    - 如何从模型中快速查询得到相关的对象？
  - 多个规则之间具有相关性
    - 涉及相同的对象➔避免多次查询
    - 设计约束条件的强化➔递进的约束检查
- 作业虽然简单，但要认识到真正的模型验证故事才刚开始
  - 是否允许对模型的高层特性进行深度检查？➔model checking
  - 是否允许用户自定义规则?
  - 是否能够学习人是如何发现模型缺陷的?
  - …

课程总结博客

- 总结本单元作业的架构设计
- 总结自己在四个单元中架构设计思维及OO方法理解的演进
- 总结自己在四个单元中测试理解与实践的演进
- 总结自己的课程收获
- 立足于自己的体会给课程提三个具体改进建议