第十五讲
软件设计及其模型化表示

OO2019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
- 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意识
  - 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
  - 测试是你的守护神

课程主题回顾

- 架构意识
  - 直接奔向代码战场的结果，常常遍体鳞伤，甚至铩羽而归
  - 架构意识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
  - 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的求解要求
- 重构
  - 轻量级：调整算法
  - 中量级：调整局部数据结构
  - 重量级：调整架构

软件设计难在何处

- 功能定义了软件的输入和输出之间的映射关系
- 难点1：输入有多种形态
  - 如何找到输入到输出映射的规律?
- 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
  - 桥的结构往往支持动态伸缩
- 难点3：多次输入之间具有逻辑联系
  - 每提供一个输入都可能会对系统的数据模型产生影响，必须进行适应
  - 必须区分出变与不变

软件设计难在何处

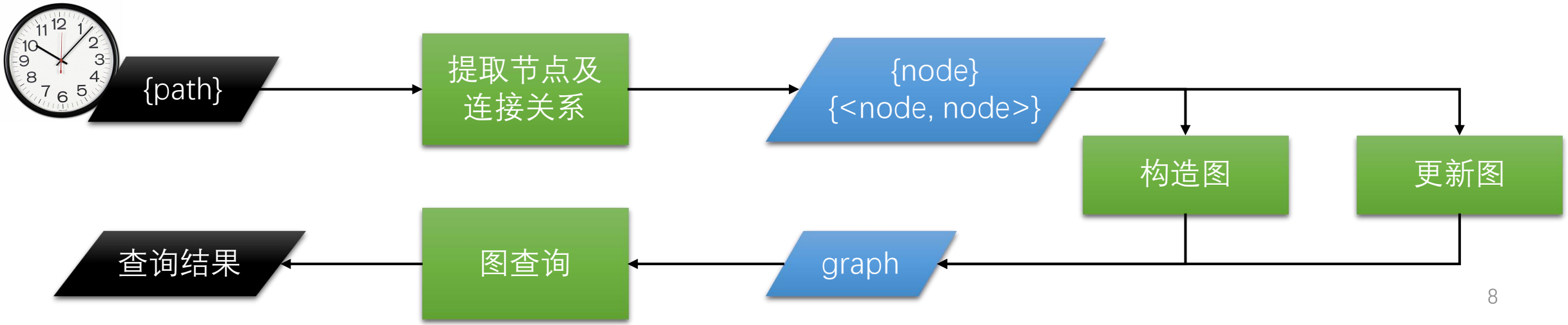
- 难点4：连续性输入，不断输出
  - 并发处理，安全保护问题
- 难点5：不只是能够产生输出，还有性能要求
  - 算法设计必须和数据模型设计配合起来
- 难点6：存在各种样式的异常输入
  - 准确区分异常输入和正常输入，识别和防范处理
- 难点7：需求容易发生变化
  - 增加输入形态
  - 调整已有的输入到输出映射关系
  - 预见输入形态的可能变化，识别并控制变化影响范围

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
- 行为视图
  - 模块间的交互行为
  - 算法流程

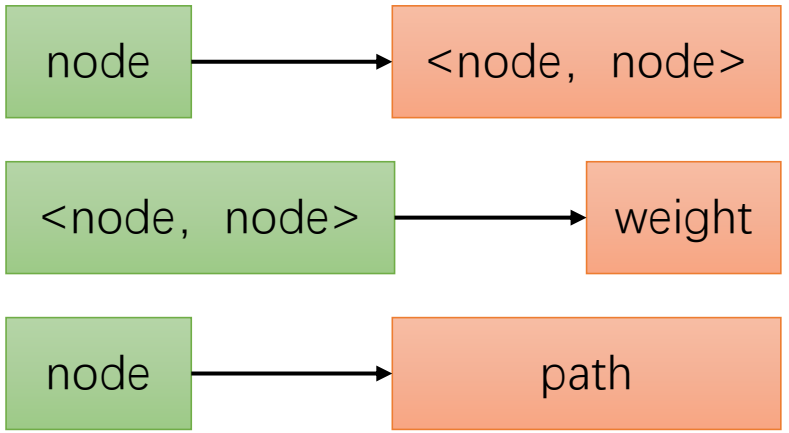
解析软件设计

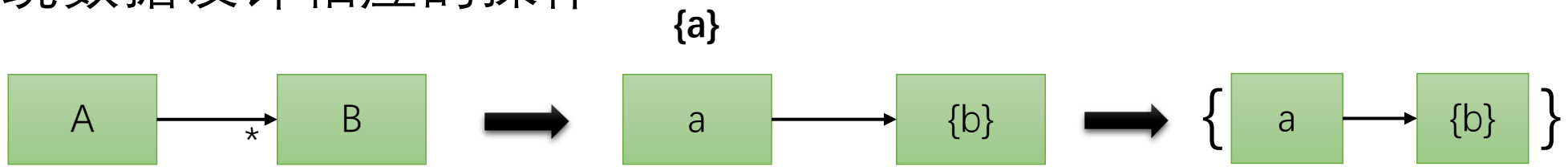
- 基于功能数据流的结构设计
  - 数据流分析是个重要的功能结构分析手段
  - 识别模块及数据依赖关系
- 数据流特征
  - 一次提供输入
  - 分批次提供输入



解析软件设计

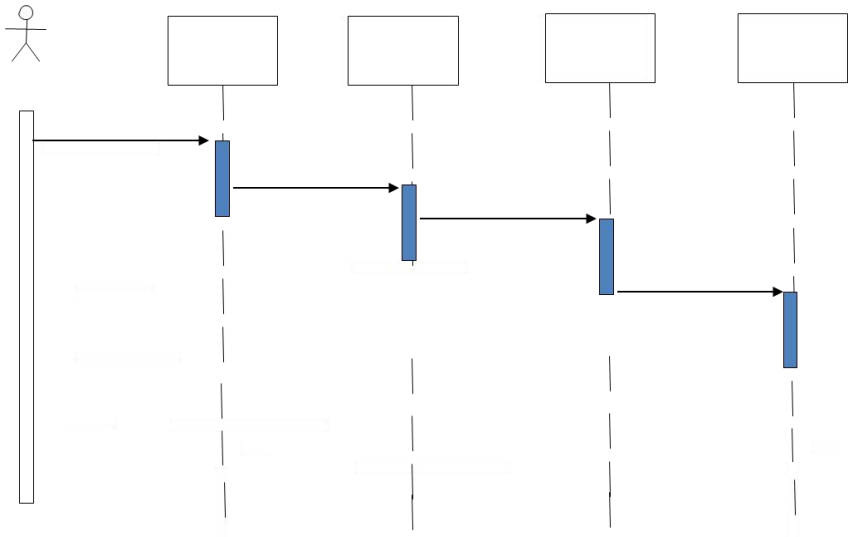
- 数据流程视角识别出了数据的结构
  - 输入、中间数据和输出
- 可以按照第7讲所介绍的面向对象分析与设计方法来整理数据模型
  - 形成类
  - 形成数据容器
- 数据模型：数据之间的关联和映射
  - 关联：建立访问通道
  - 映射：建立快捷访问通道
- 围绕数据设计相应的操作

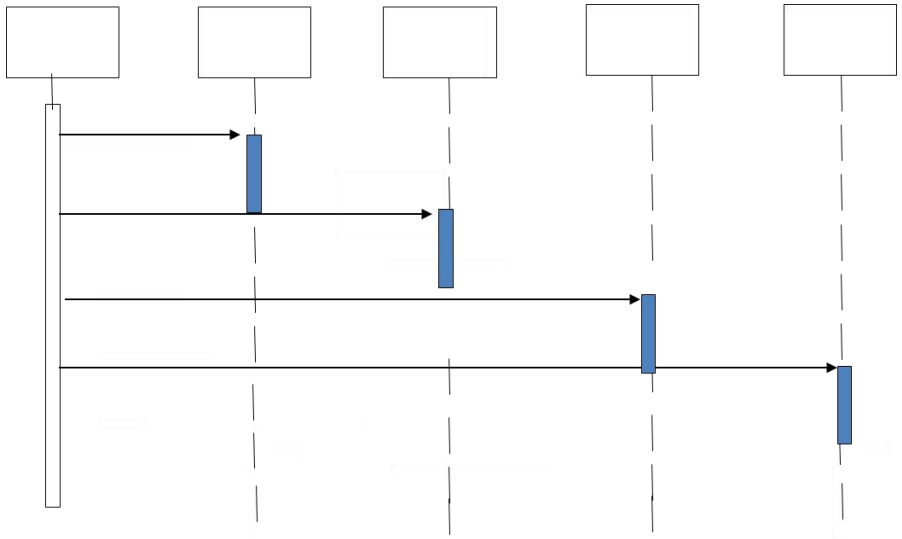




解析软件设计

- 在模块间建立交互协同行为
  - 逐层代理调用
  - 中心控制调用
- 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
  - 逐步引入组合规则
- 核心设计目标：构造抽象层次，进行归一化处理
  - 组合模式
- 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
  - 逐步引入调度机制
- 核心设计目标：识别线程及其共享数据，控制共享安全
  - 生产者-消费者模式
  - 订阅-发布模式
  - 链状多层生产-消费模式
- 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：局部化算法作用范围

回顾课程的设计训练

- 第三单元
  - 逐步引入JML规格
  - 逐步引入复杂的中间数据模型
- 核心设计目标：根据功能需要适应性能要求的中间数据模型和协同架构
  - 在图数据结构和图模型之间建立层次
  - 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

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
  - 逐步引入UML模型的语义规则
- 核心设计目标：针对诸多不同类型的对象构造层次和关系，在构造UML模型的过程中动态维护相关的查询数据
  - 通过graph层次的对象解析和关系提取构造UML模型的语义
  - 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：模型图的结构化搜索，提取相关对象，并进行规则检查

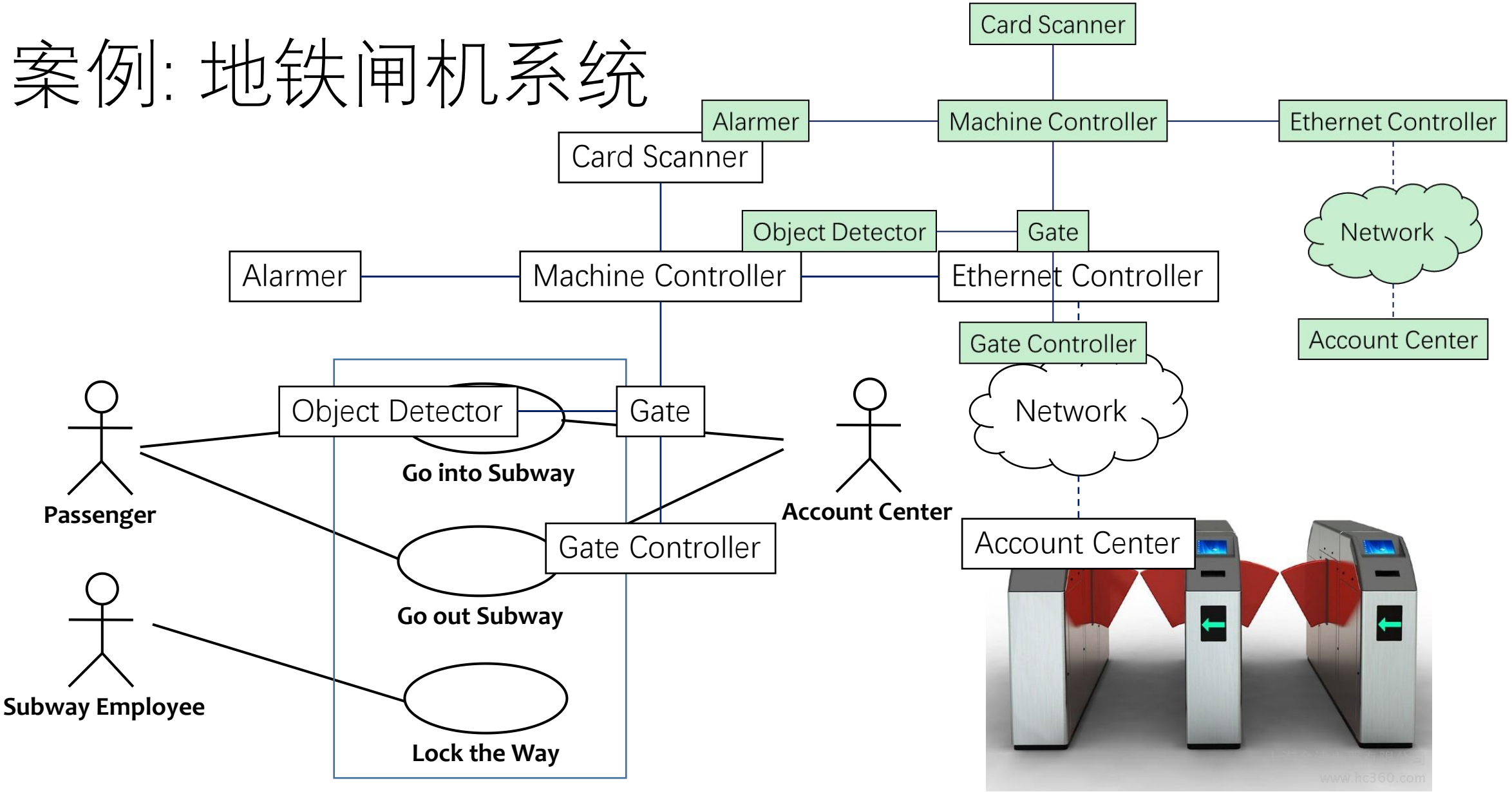
模型化设计

- 使用系统化的模型语言来表示设计结果，进而开展设计思考
  - UML
- UML采用了视图与模型相分离的设计
  - 在提供的diagram中表达相应的元素和关系
  - 建模工具维护UML模型
- UML提供了四种模型视图
  - 功能视图
  - 结构视图
  - 行为视图
  - 部署视图

模型化设计

- UML功能视图(use case diagram)支持的元素及关系表达
  - 用例(use case)：系统提供给用户的功能及其交互场景
    - Precondition, postcondition, flow of events
  - 执行者(actor)：为系统执行提供输入激励或者记录系统执行结果的相关对象
    - 自然人、设备、其他系统等
  - actor与use case之间的关系
    - 哪些用户为这个用例提供输入?
    - 哪些用户关心这个用例的执行结果?
  - use case与use case之间的关系
    - 依赖关系
    - 抽象层次关系

案例: 地铁闸机系统


Card Scanner
Alarmer
Machine Controller
Ethernet Controller
Card Scanner
Network
Object Detector
Gate
Alarmer
Machine Controller
Ethernet Controller
Account Center
Gate Controller
Network
Object Detector
Gate
Go into Subway
Passenger
Account Center
Gate Controller
Account Center
Go out Subway
Subway Employee
Lock the Way

模型化设计

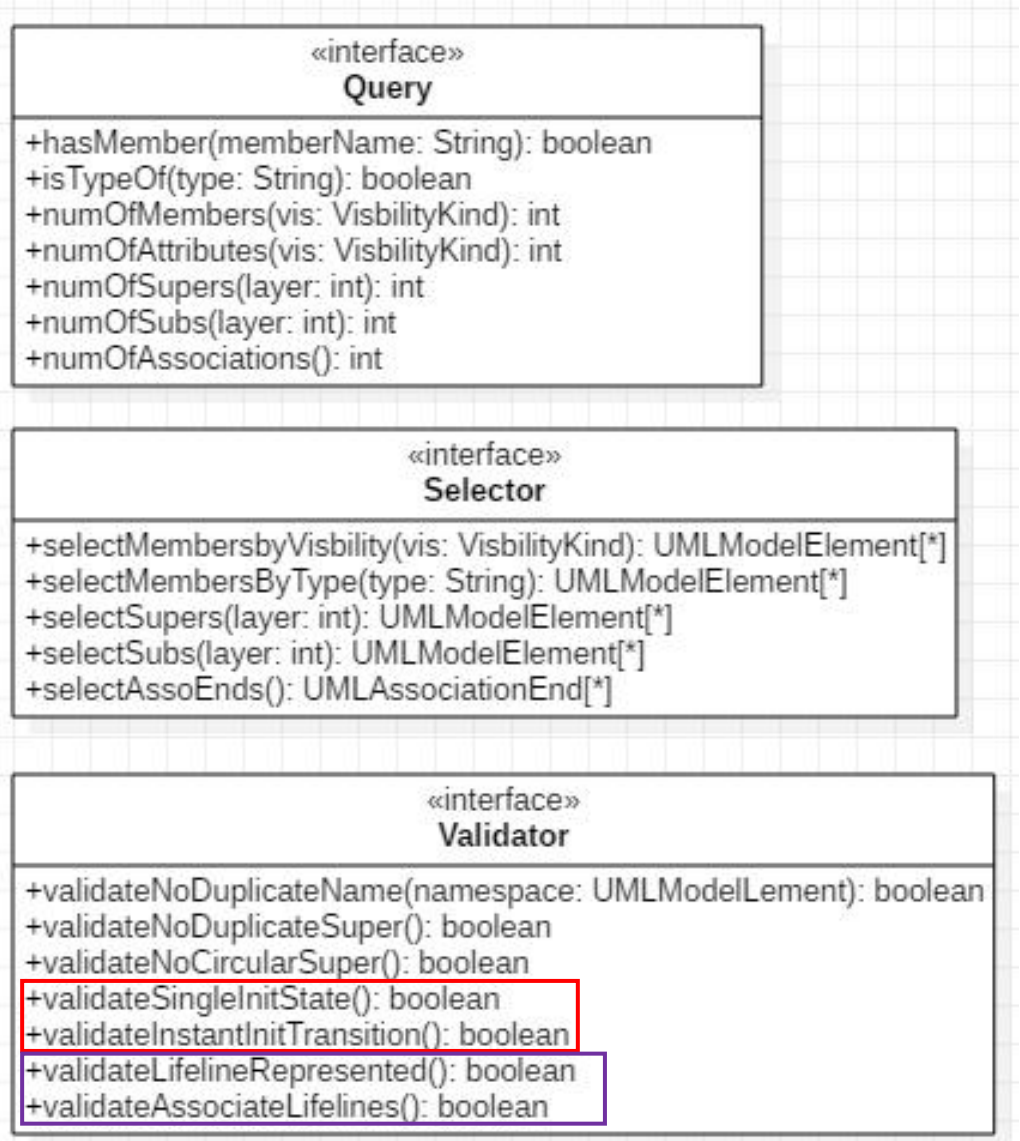
- 结构视图
  - 组件图
  - 类图
- 类图支持的元素及关系表达
  - 类、接口
    - 属性、操作
  - 关联关系
  - 继承关系
  - 接口实现关系

模型化设计

- 我的系统为什么需要这个类?
- 从与用户交互功能场景角度
  - 边界类
  - 实体类
  - 控制类
- 从数据封装与处理角度
  - 映射到功能需求中的数据项
  - 类中所封装数据项之间的聚合特性
- 从层次关系角度
  - 容器类
  - 控制策略类
  - 归一化类/接口

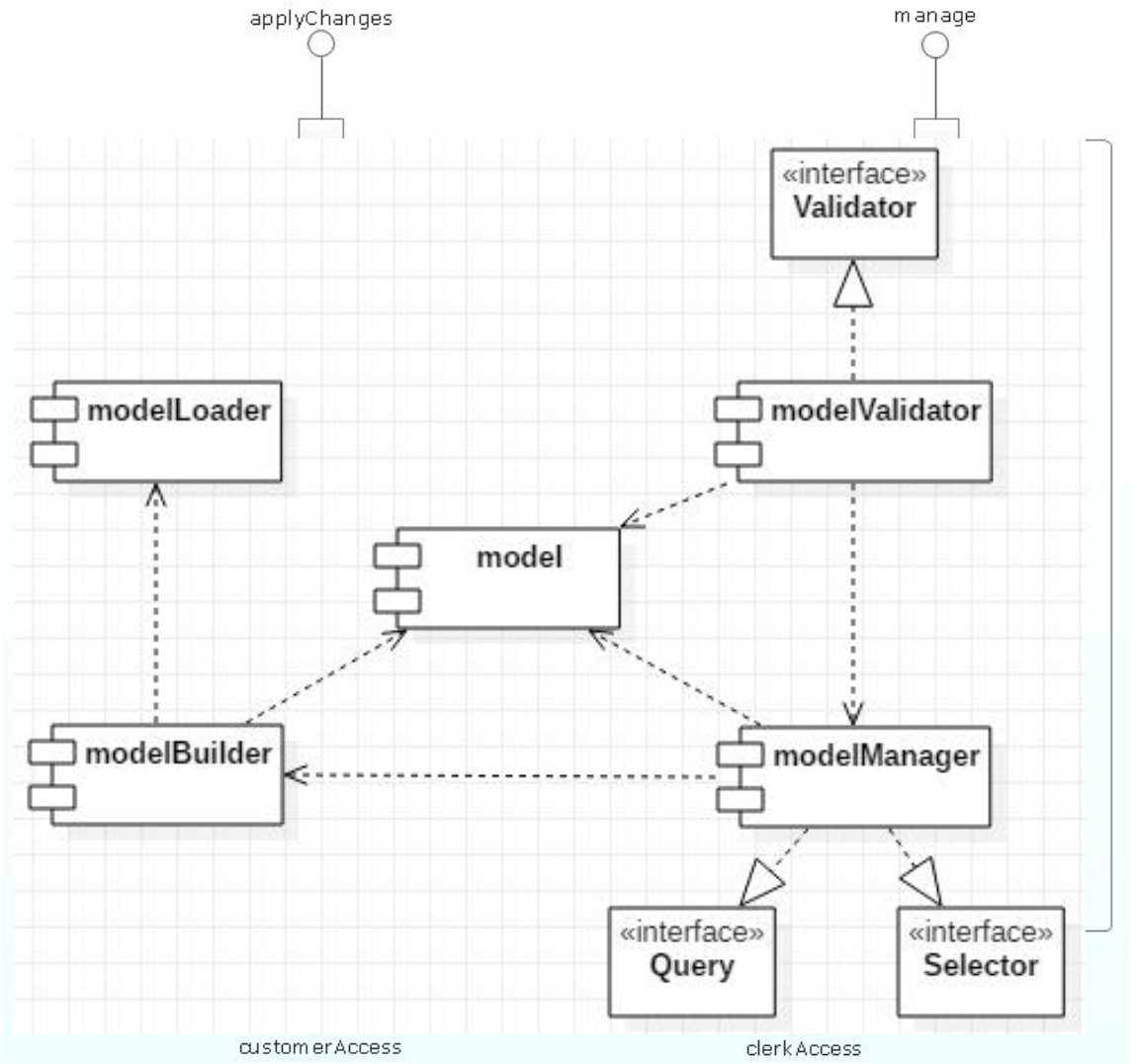
Case分析

- 第十四次作业的用户交互功能
  - 查询(query)
    - 存在型查询：某个类中是否有某个方法/属性
    - 数量型查询：某个类中有多少个方法/属性
  - 检索(selector)
    - this，{o|this→o}
    - this, {o|o→this}
  - 确认(validator)
    - validate***:boolean
    - validate(objSet, constraint):boolean
- 涉及容器类对象和对象连接关系
  - 设计相应的接口
  - 让相关的类实现这个接口



模型化设计

- 组件图支持的元素及其关系表达
  - 组件：提供相对完整功能的设计单位
  - 端口：组件对外的交互界面
  - 接口：组件对外的交互能力
  - 组件与组件
    - 依赖关系
  - 组件与接口
    - 提供的接口(provided interface)
    - 要求提供的接口(required interface)
  - 端口与接口
    - 通过端口提供接口操作



模型化设计

- 组件图在系统架构设计中具有重要地位
  - 组件有多种类型
  - 软件组件、硬件组件、资源组件、网络组件等
- AADL(Architecture Analysis & Design Language)是专门用于嵌入式分布式系统的架构设计模型语言
  - 高层次架构设计
  - www.aadl.info
- 本课程涉及的架构设计更接近于实现层次的设计

模型化设计

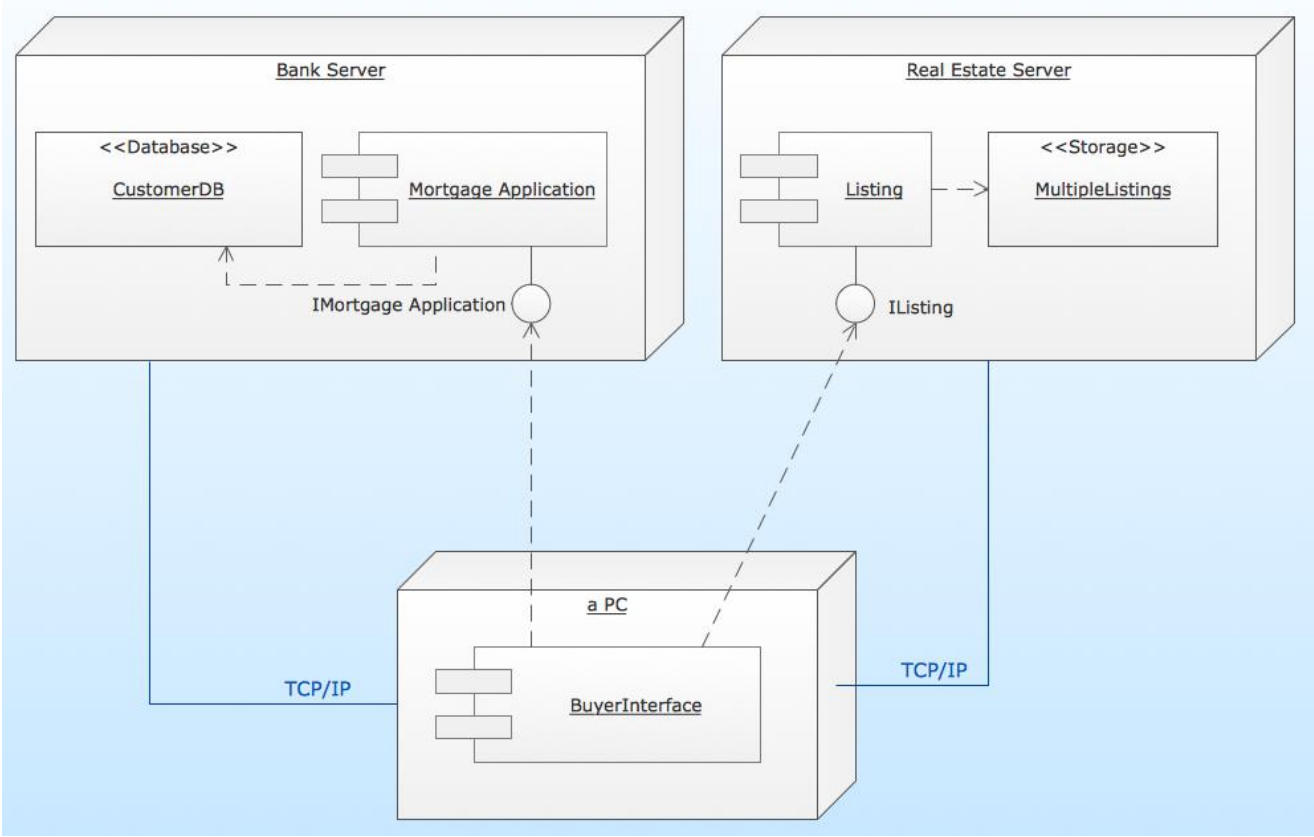
- 行为视图
  - 顺序图
  - 状态图
- 顺序图围绕一个功能场景(如use case)，从对象交互角度给出相应use case的设计方案
  - 识别类应提供的操作
  - 识别类之间的关联关系
- 顺序图采用消息交互机制
  - 消息与对象操作相关联

模型化设计

- 状态图针对具有一定状态复杂度的类来专门设计其行为
  - 多种状态和转移关系
  - 在运行时会随输入动态改变其状态
- 一般多用于描述控制行为：针对被控对象的状态来实施相应的控制行为，并维护其状态更新
  - 线程调度(线程对象的状态空间)
    - 根据线程对象的状态来调度请求来更改其状态
  - 电梯调度(电梯对象的状态空间)
    - 根据电梯对象的状态来分配乘客请求并改变其状态
- 不同状态之间在属性取值上必须严格分离

模型化设计

- 部署视图支持的元素及其关系
  - 部署节点：提供运行所需的资源
  - 组件：一个部署单位，提供相对完整的业务功能和相应数据管理功能
  - 部署节点与部署节点：依赖和通信交互关系
  - 部署节点与组件：节点为组件提供运行时资源
- 部署图展示系统的部署安排和拓扑结构



模型化设计的思维方法

- 抽象
  - 按数据或行为
- 分类
  - 概念分类
  - 对象分类
- 分段
  - 按业务处理时间线
- 层次
  - 按数据管理层次
- 映射
  - 按数据间的因果或者关联关系

模型化设计的思维方法

- 抽象是建模中的最重要方法
- 忽略细节，抓住本质
- 几乎每个UML模型图都需要使用这种思考方法
  - 识别类、属性、操作、关联和继承等
  - 识别接口和接口实现关系
  - 识别状态、迁移
  - 识别消息连接关系

模型化设计的思维方法

- 分类是最常用的一种抽象方法
  - UML把类和接口抽象为UMLClassifier
- 类图
  - 识别类和接口，建立它们的继承关系
  - 建立多重关联，按照角色和特征进行分类化对象管理
- 状态图
  - 识别类的多个状态
  - 按照状态来设计类的行为

模型化设计的思维方法

- 按照输入到输出处理过程，区分活动段，按段来识别相应数据抽象和行为抽象
- 在段之间建立数据流关系，形成协同结构
- 系统设计中必然涉及诸多数据，如何管理这些数据是一个不能忽视的问题
  - 建立数据管理层次
  - 结合数据分组构建多叉管理层次
  - 管理层次往往和协同结构一致
- 过深的数据管理层次显著加大数据检索的代价
  - 跨层次间建立映射结构，快速检索和更新

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)

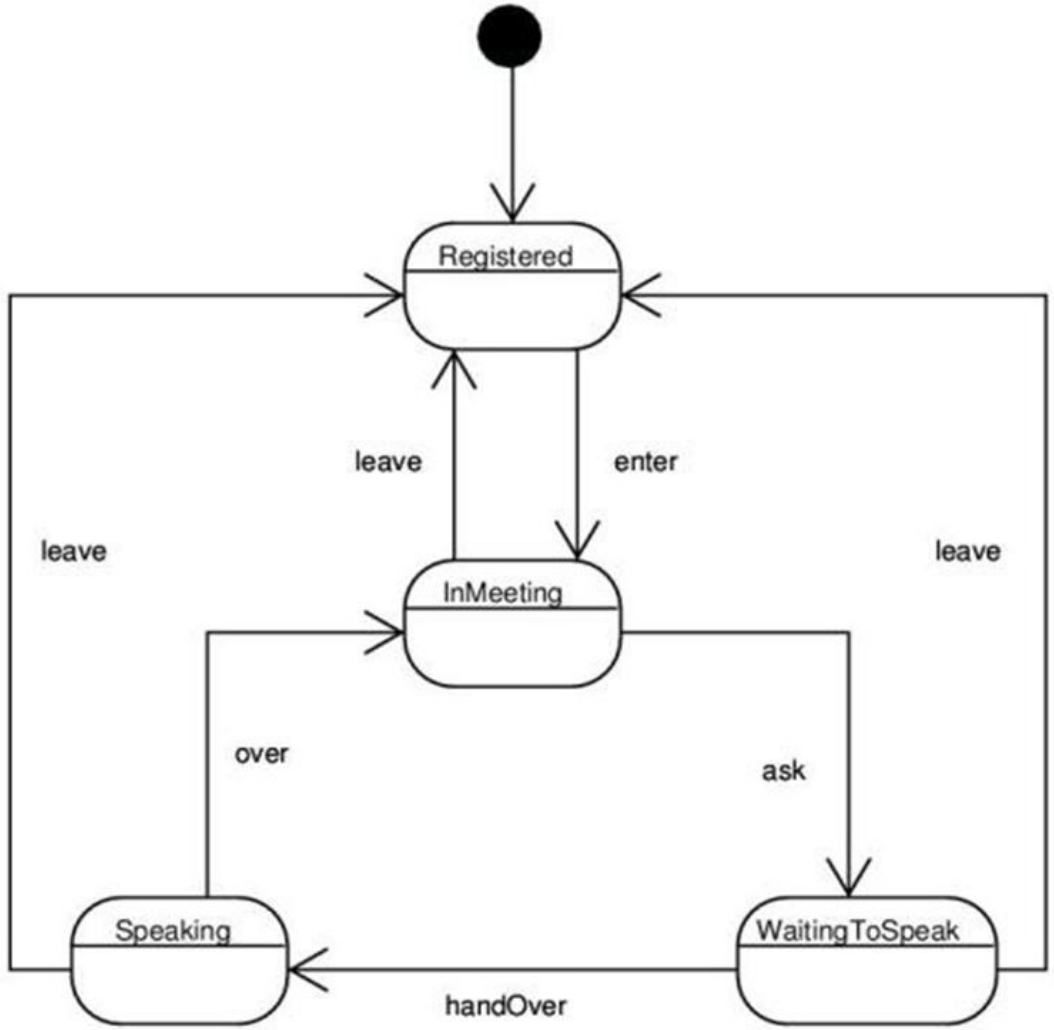
MBT—基于状态图的测试

- 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

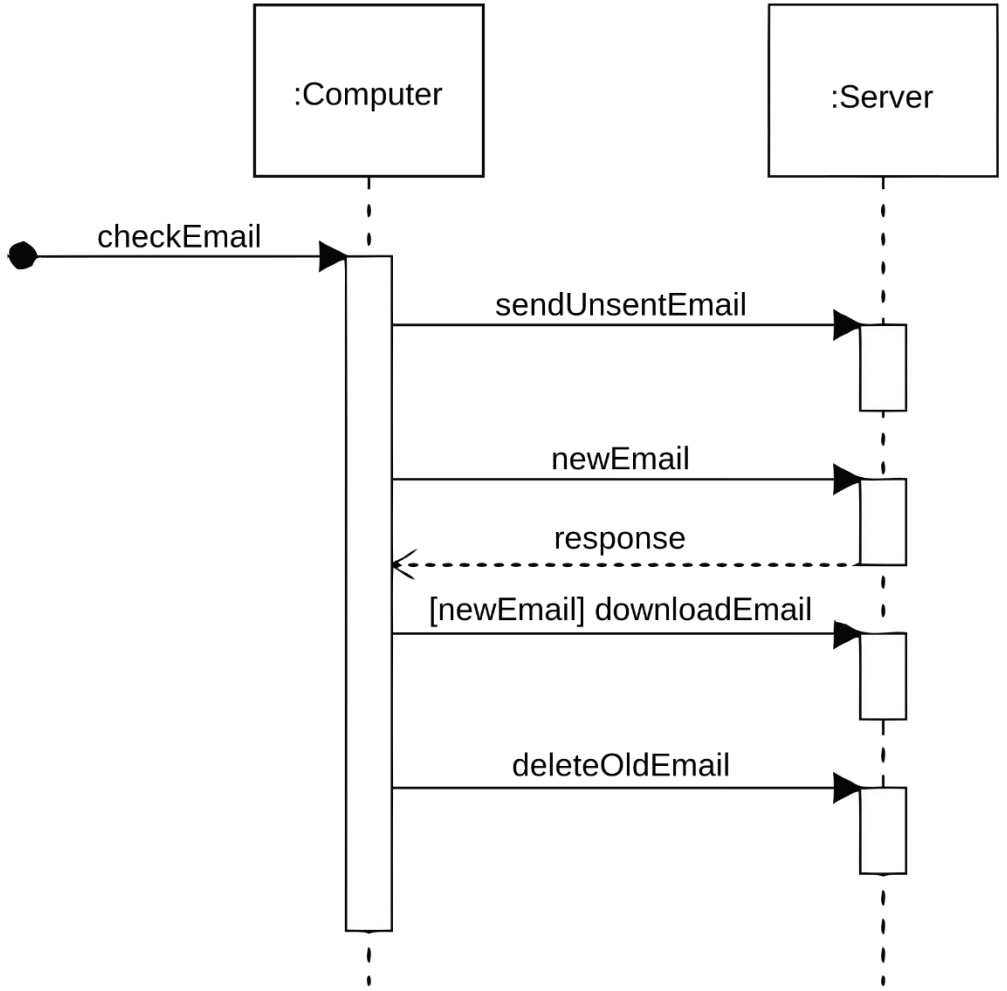
- 基于顺序图的测试
  - 输入: 顺序图模型
  - 输出: 消息序列(测试用例)

1: 通过UMLAttribute确定参与协同的对象：准备相应的对象

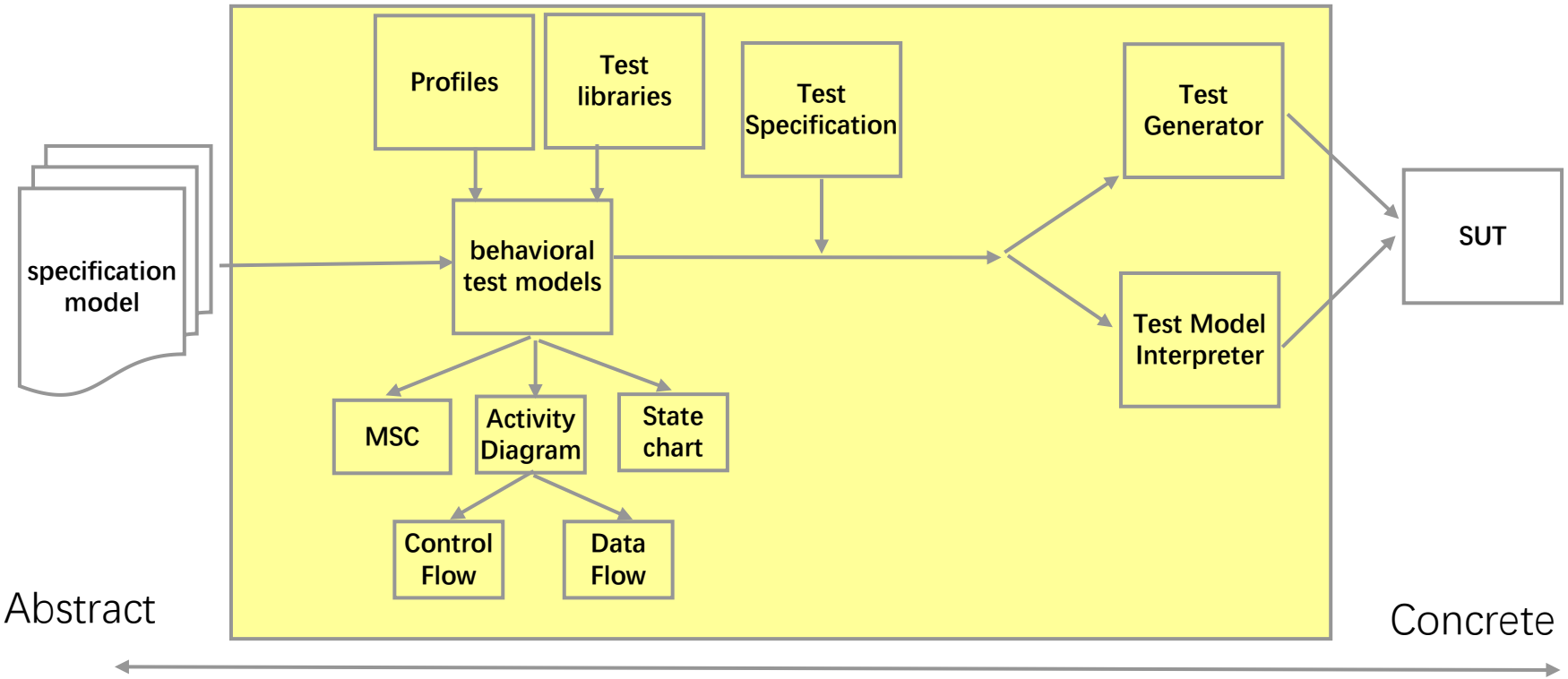
2: 选择目标对象，以incoming消息来构造测试用例，使用对象的outgoing消息来检查执行效果

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?

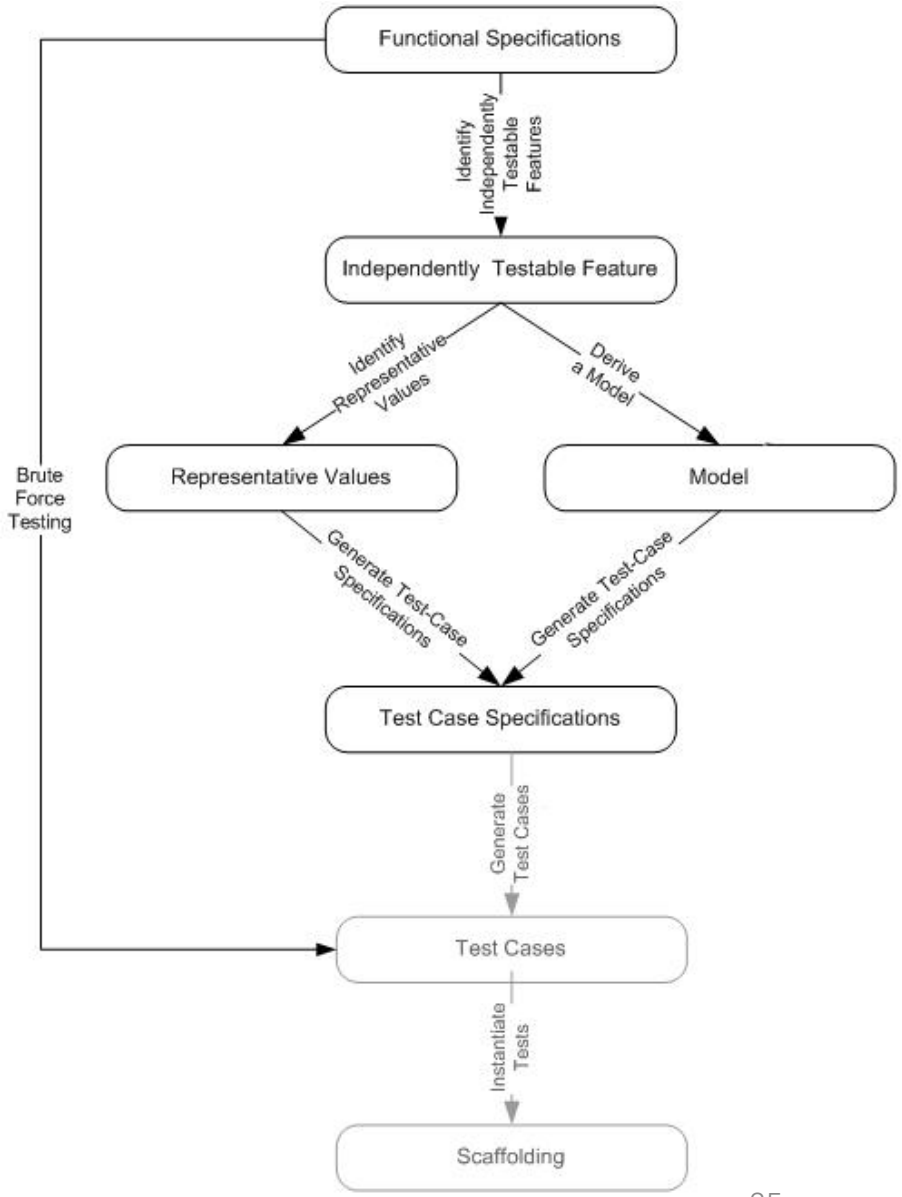


MBT的工作原理与架构



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*

课程总结博客

- 总结本单元两次作业的架构设计
- 总结自己在四个单元中架构设计及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