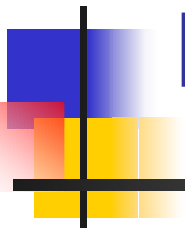


Rational Rose基础

课程目标

- 能解释可视化建模的好处
- 通过实际使用能够具有使用工具的能力
- 了解如果用Rational Rose进行round-trip Engineering

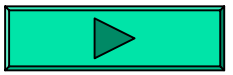
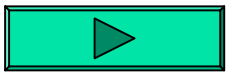
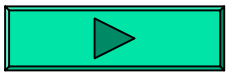
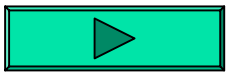
术语

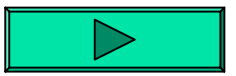
- 工件(artifact)
  - 工件是流程产生、修改或使用的信息
  - 它定义了一个责任域，可以进行版本控制
  - 可以是模型、模型元素或文档

Contents

- Module1:可视化建模和UML
- Module2: Rose建模基础
- Module3:在团队中使用Rose
- Module4:用例模型
- Module5:用例实现的结构
- Module6:交互图
- Module7:类图
- Module8:round-trip engineering的介绍

Module1:可视化建模和UML

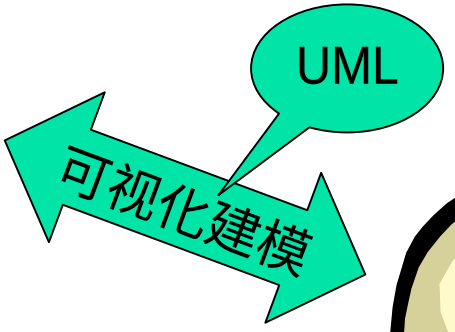
- 什么是可视化建模 
- 什么是UML 
- UML diagrams 
- 对UML表示法的扩展 



什么是可视化建模

“建模获取系统的关键部分”

业务流程

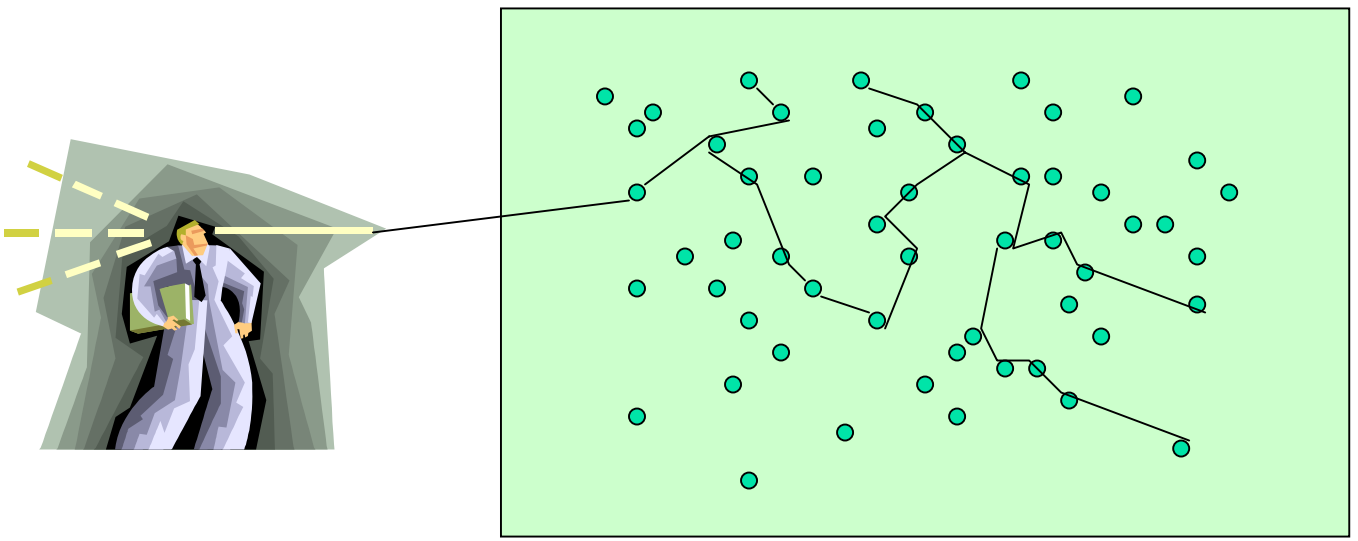


计算机系统

可视化建模就是用标准的图形表示法来建模

可视化建模的作用（1）

- 可视化建模获取业务流程
  - 用例（use case）分析是一种从用户的角度获取业务流程的技术
  - 使用相同的语言，不至于产生歧义
  - 用例分析能让分析师在构建系统之前理解要构建什么

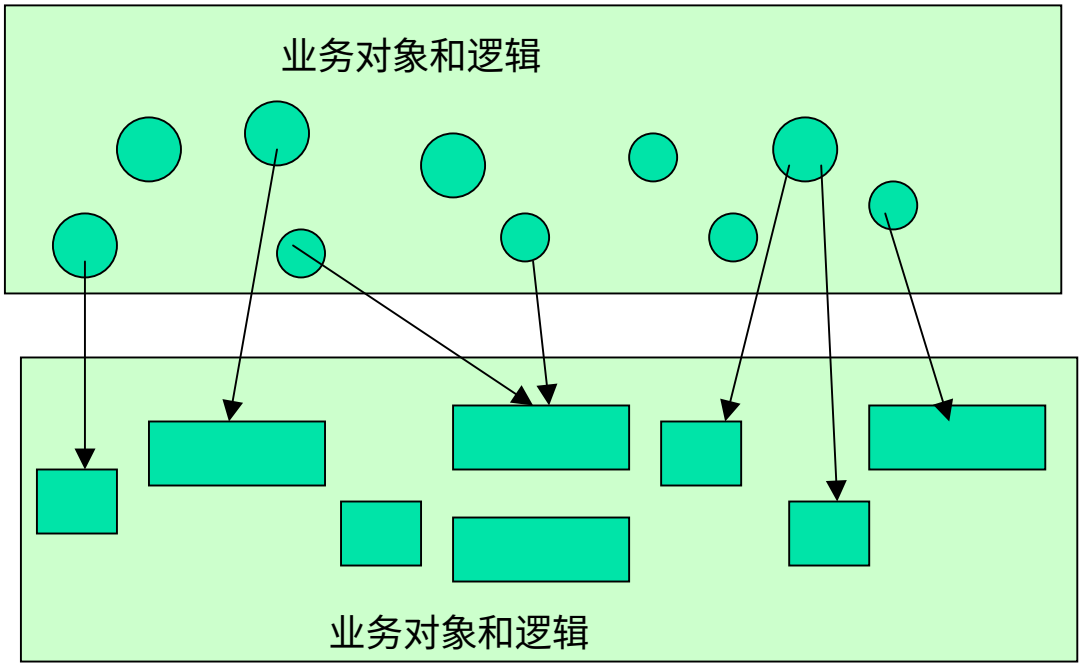


可视化建模的作用（2）

- 可视化建模是一个交流工具

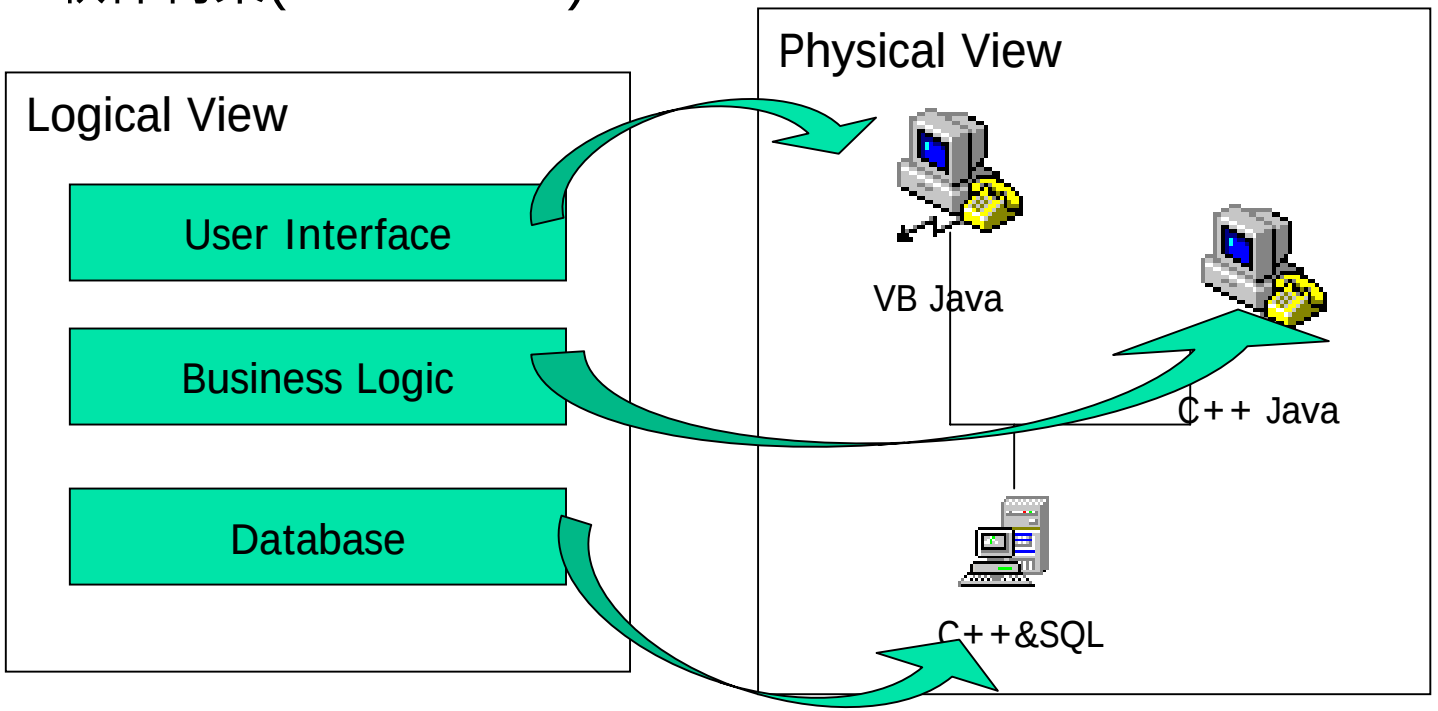
业务领域

计算机领域



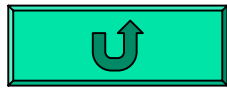
可视化建模的作用（3）

- 管理复杂性
  - 把3000多个类放在一张图中不好
  - 可视化建模的“包”（package）
    - 把元素模型化成有意义的组合
    - 为不同的人提供不同级别的抽象
  - 软件构架(architecture)



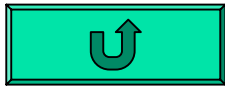
可视化建模的作用（4）

- 促进复用(reuse)
  - 复用是软件的“圣杯”
  - 可以有一个类复用、多个类（或一个组件）的复用、应用模式等复用方式
  - 不止是复用代码，而是复用建立原始工件时需要的所有分析、设计、实现、测试、文档化
  - 可视化建模让你从复用的角度看，如果想复用工件，什么是可用的



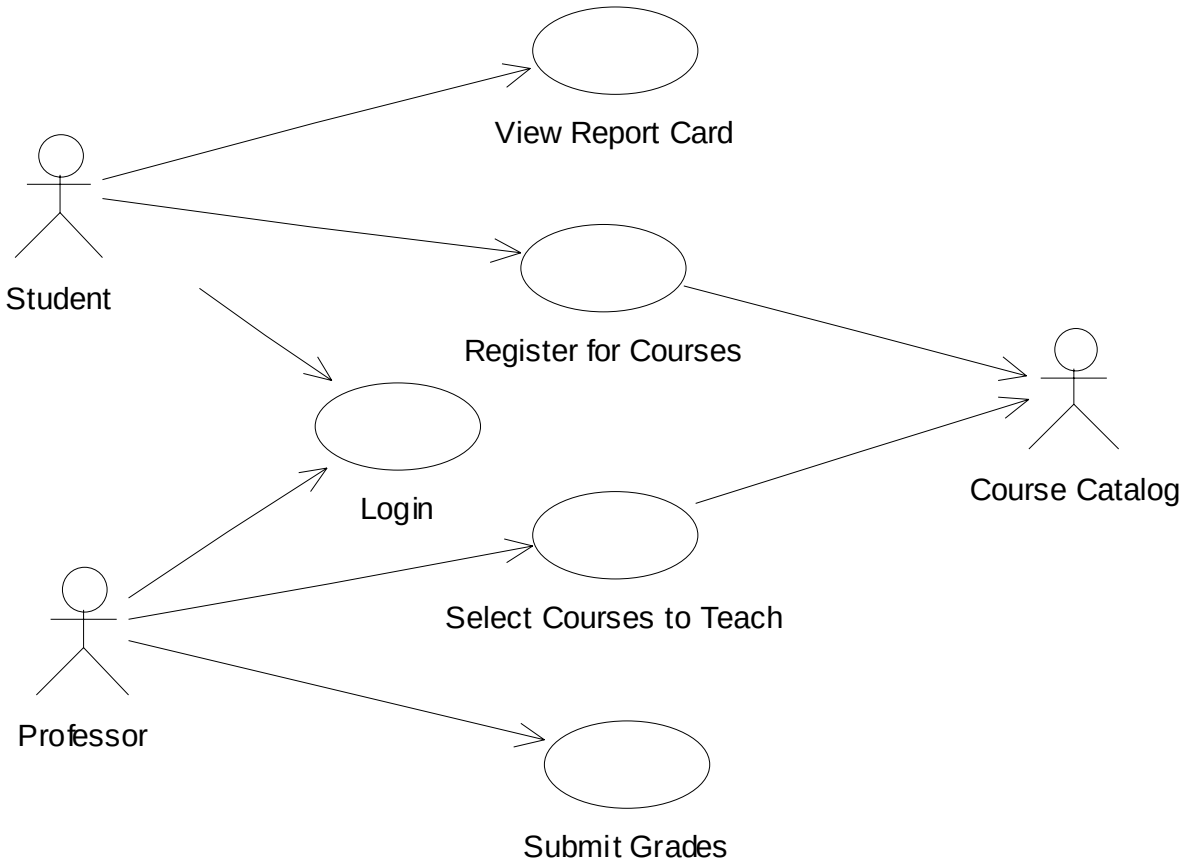
什么是UML

- UML(Unified Modeling Language)是可视化、说明、构建和文档化软件系统工件的标准语言
- UML可以做下面的建模
  - 数据建模
  - 业务建模
  - 对象建模
  - 组件建模
- UML可以用于可视化建模
  - 系统与外界的交互
  - 系统的行为
  - 系统的结构
  - 系统的构架
  - 系统的组件



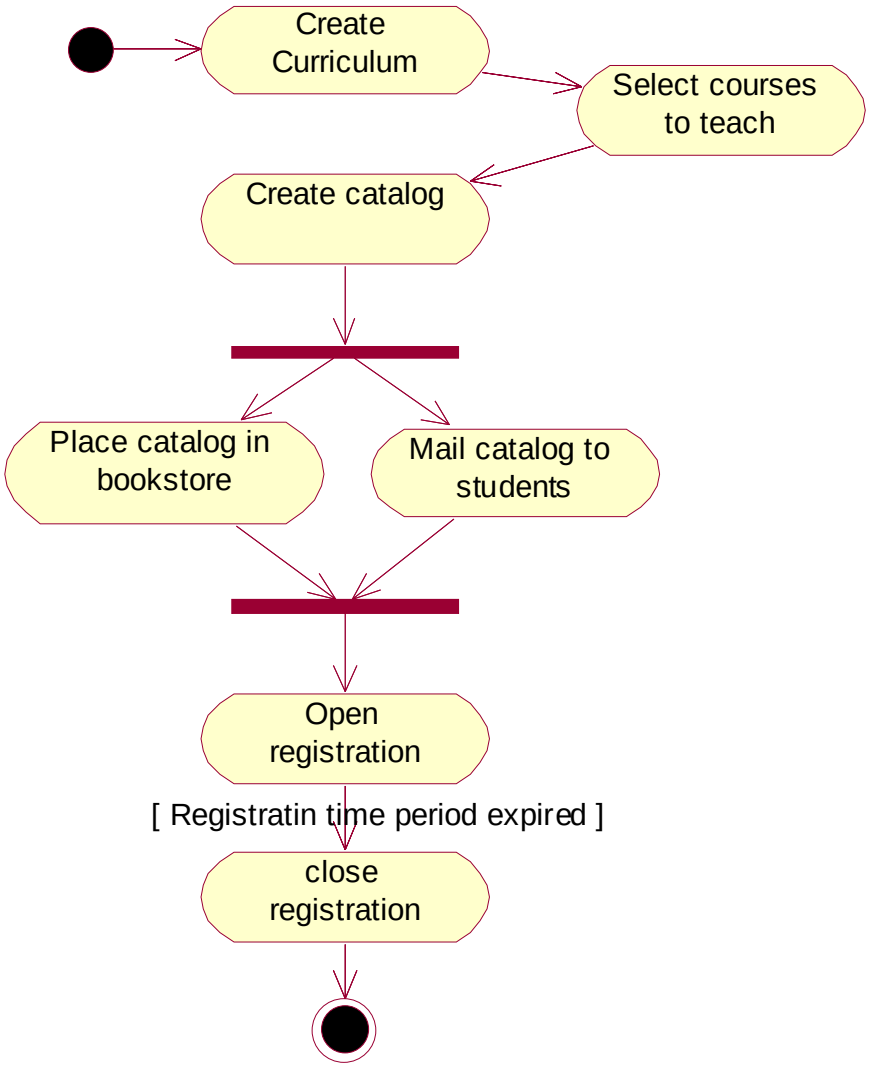
用例图(use case diagram)

- 创建用例图可以可视化应用与外界的交互



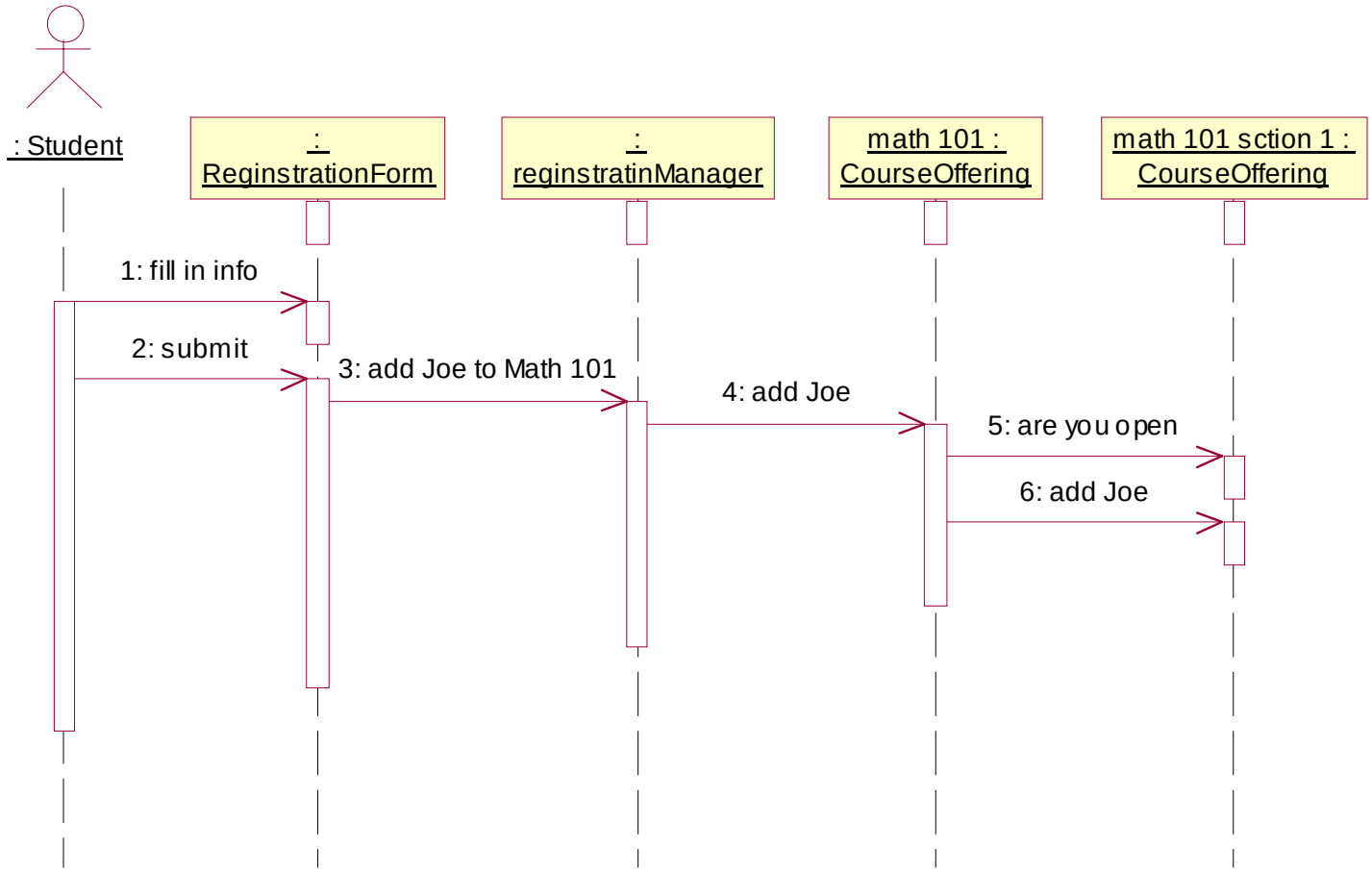
活动图(activity diagram)

- 活动图可以表示系统内的事件流



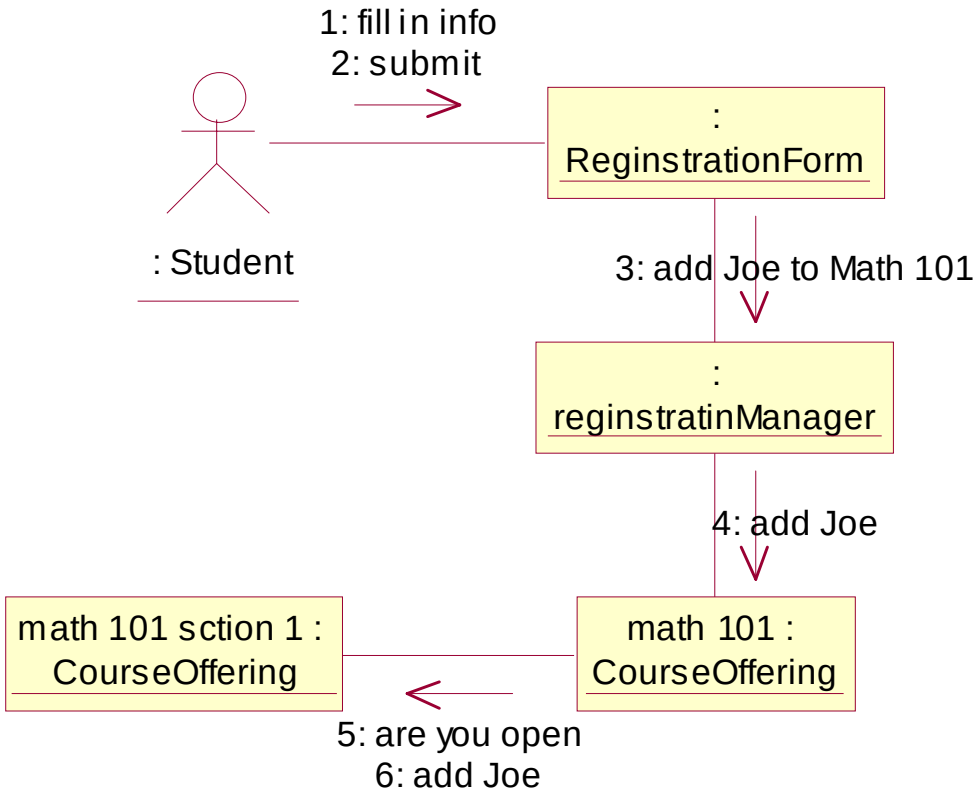
时序图(sequence diagram)

- 时序图表示如何一步步的完成系统的一个功能
  - 时序图表示的是一个场景(scenario)



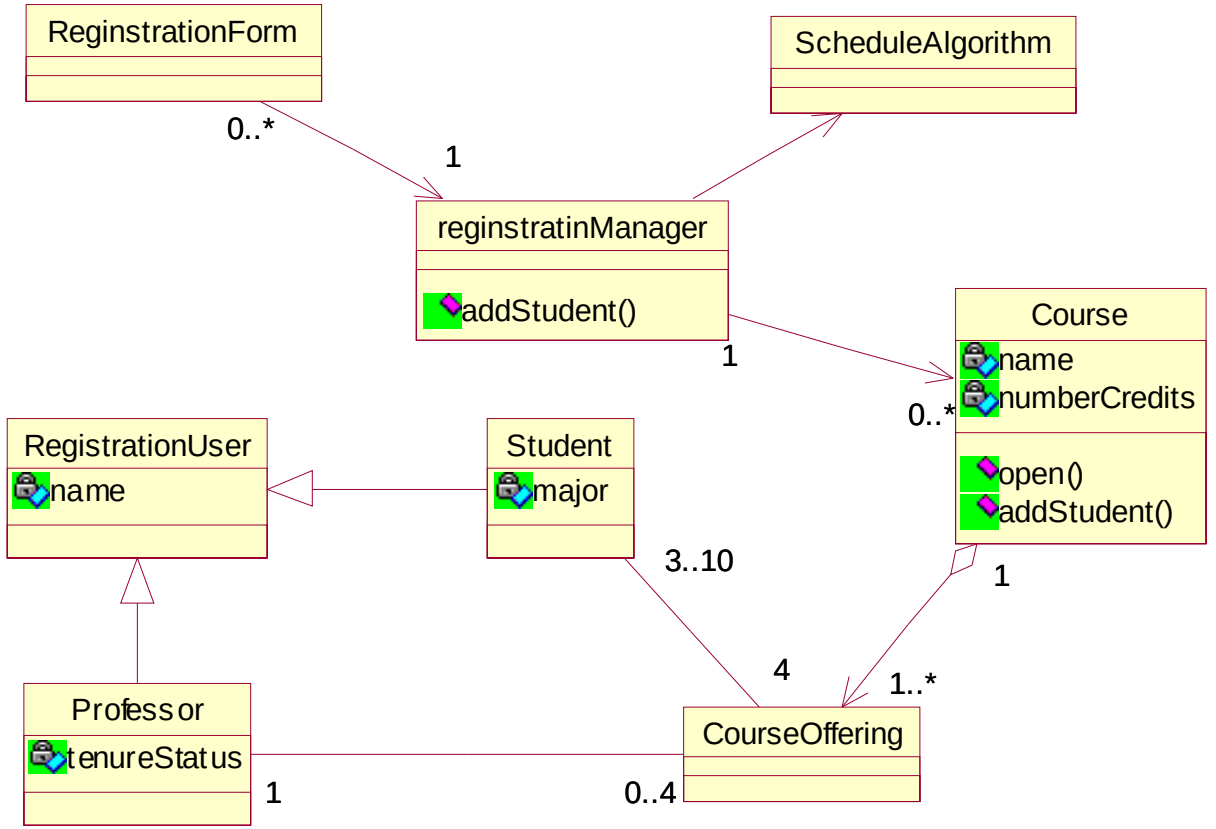
协作图(collaboration diagram)

- 协作图显示对象之间的交互



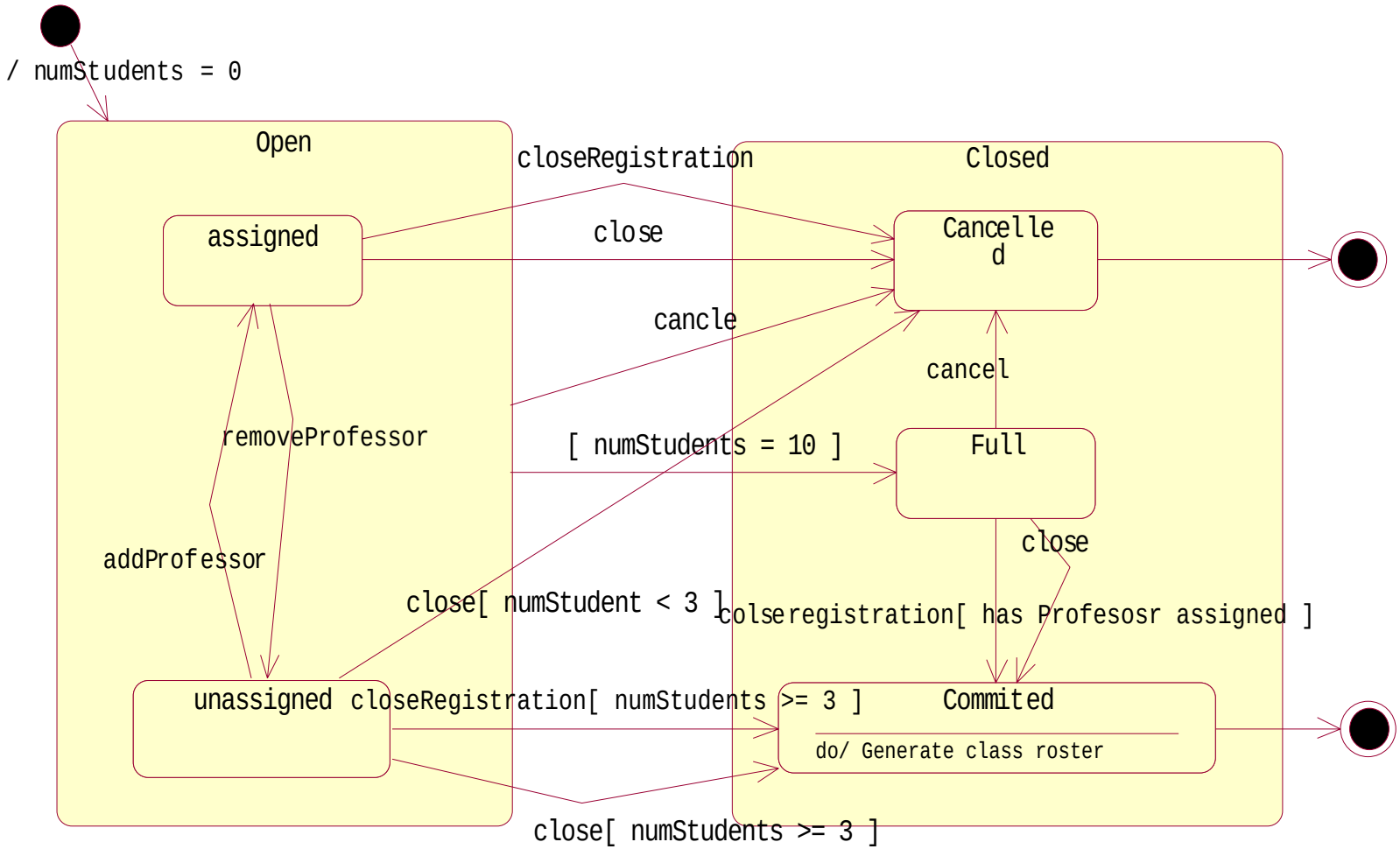
类图(class diagram)

- 类图表示软件的结构
  - 包括类、关系和multiplicity



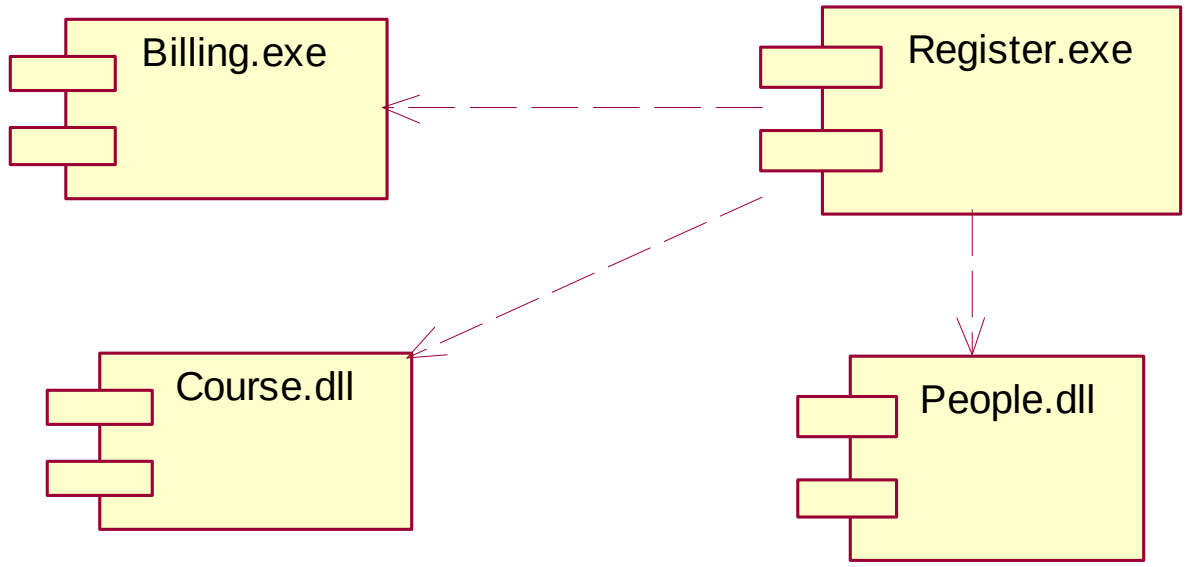
状态图(statechart diagram)

- 状态图表示一个类的生命周期



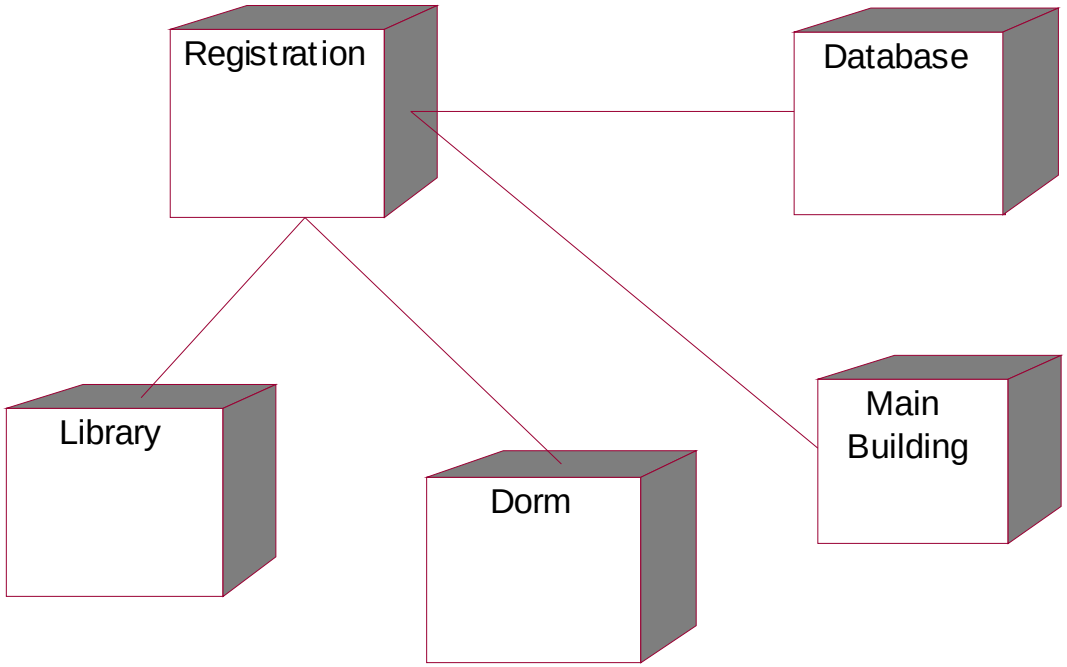
组件图(Component Diagram)

- 组件图可视化的表示组件的组织和依赖
  - 组件：软件组件(如C++中的头文件)、运行时组件（如，DLL）、可执行的组件
  - 接口



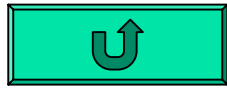
部署图(Deployment Diagram)

- 部署图可视化的表示企业内组件的分布

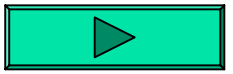
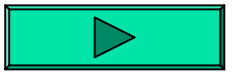
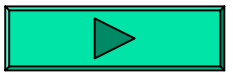


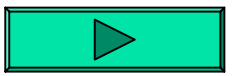
对UML表示法的扩展

- Stereotypes用于扩展UML表示法的元素
- Stereotypes用于对association、继承关系、类和组件进行分类和扩展
- 例如
  - 类的Stereotypes：interface,exception,server page
  - Association Stereotypes:identifying,non-identifying
  - Dependency Stereotypes:include,extend
  - Component Stereotypes:subsystem



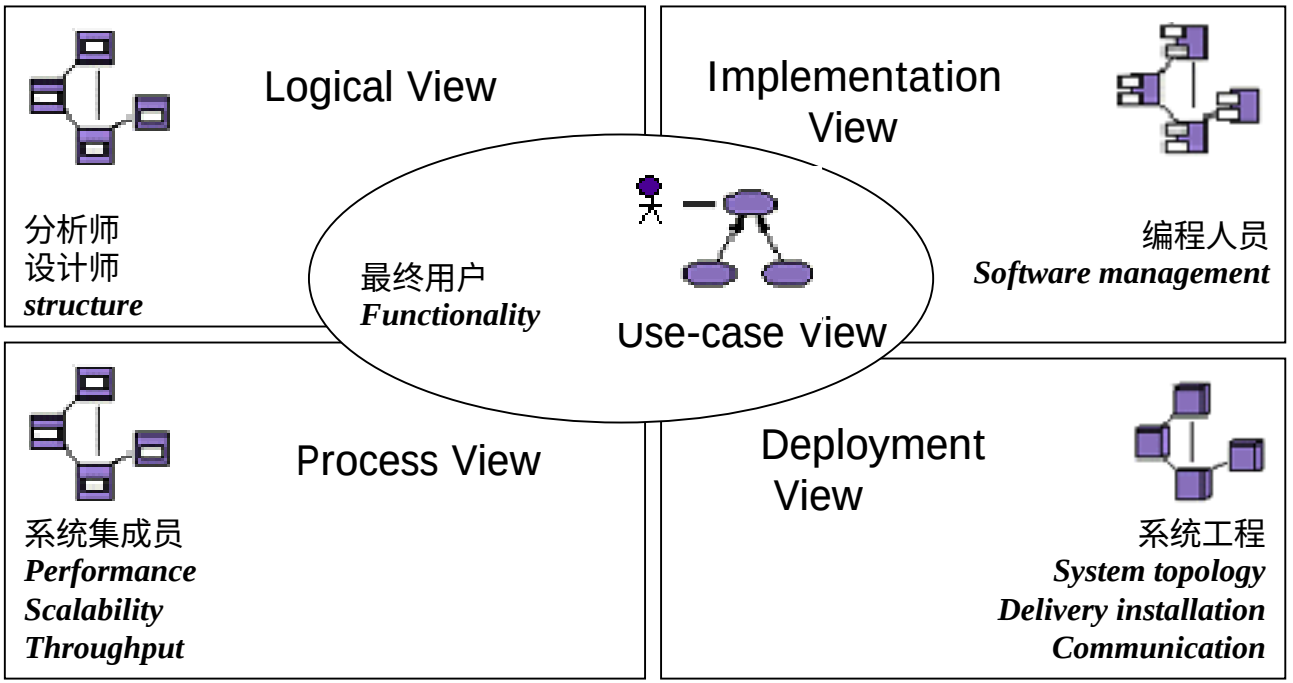
Module2: Rose建模基础

- Rational Rose中的view和diagram 
- Rational Rose界面 
- 基本的工具技能 



Views

- 模型由不同的view和diagram构建而成，描述了不同的视点和系统的构建块
  - 模型是现实的简化或系统的蓝图
- View是一个对特定涉众有意义的模型的视点
  - View是模型的“碎片”
  - Rose中的“4+1view”

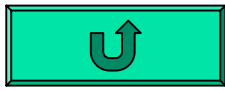


Views

- Use-case View是其他view的“心脏”，因为它说明了系统做什么
  - 包括用例图、用例事件流和补充规约，也可以包括活动图
  - 是其他view的“心脏”，因为它说明了塑造系统构架需要的精力
- Logical View支持系统的功能性需求
  - 包括用例实现、类图和交互图，也包括状态图和活动图
- Process View阐述系统的性能、伸缩性和吞吐量
  - 包括形成系统并发和同步机制的线程和进程
  - 对于单处理环境是不必要的
- Component View（implementation View）说明开发的容易，软件的管理、复用、子合同和off-the-shelf组件
  - 以分包、分层和配置管理的形式描述了静态软件模型的组织（源码、数据文件、组件、可执行体等）
- Deployment View说明部署、安装和性能的主题
  - 只用于分布式系统
  - 表示如何把各种执行体和其他运行时组件映射要下层平台或计算节点
  - 显示一个部署

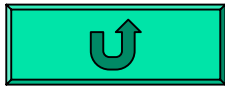
Diagrams

- Diagrams是视图化系统构建块的一种方式
- 使用Rose，可以创建各种diagrams
  - 用例图
  - 活动图
  - 交互图（协作图和时序图）
  - 类图
  - 状态图
  - 组件图
  - 部署图



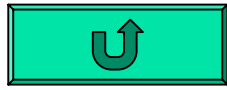
Rational Rose的界面

- Brower让你可以文本化查看和导航Views和Diagrams
  - 不在browser中的元素就不是你模型化系统的一部分
- Diagram Window让你可以创建、修改和模型化当前模型的图形化视图
- Diagram Toolbar
  - 包括构建diagram的元素
  - 每个diagram都有自己独特的toolbar
  - 只有显示diagram时才是活动的
- Documentation Window用于创建、查看或修改解释diagram中被选项目的文本
- Log Window报告进度、结果和错误
- Option Window
  - 用于设定建模的缺省设置
  - 注意修改设置时，已经存在的元素不被修改
- Stereotype display
  - 这个选项让你选择选择如何在diagram中显示Stereotype

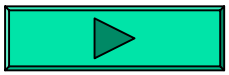
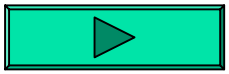
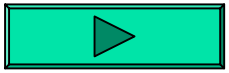
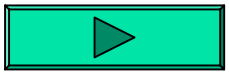
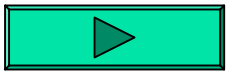


基本工具技术

- 从browser中删除diagram元素
- 从diagram中删除diagram元素
- 加入diagram元素

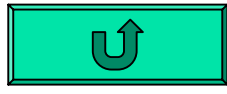


Module3：在团队中使用Rose

- 基于团队的建模 
- 受控单元 
- 虚路径 
- 复用 
- 版本控制add-ins
- 模型集成器 

基于团队的建模

- 受控的演化
  - Rose通过使用UML包和子系统支持基于构架的建模
  - Rose帮助用户在不影响其他人工作的情况下进行详细设计
  - Rose帮助用户避免创建构架单元之间不恰当的联系
- 把模型划分成在构架上有重大意义的单元
  - 可以分成叫做受控单元的单独的文件
- 构架上有重大意义的模型元素的复用



受控单元

- 受控单元
  - 是Rational Rose保存模型全部或一部分的文件
  - 是能进行版本控制的模型元素
  - 定义了单个开发人员可以操作模型的一部分
  - 在团队中共享
  - 让团队可以平行开发
  - 一定是包
- 下面元素可以作为受控单元
  - 模型文件本身(.mdl)
  - 逻辑视图和用例视图包(.cat)
  - 组件视图包(.sub)
  - 部署视图的图(.prc)
  - 模型属性(.prp)

受控单元

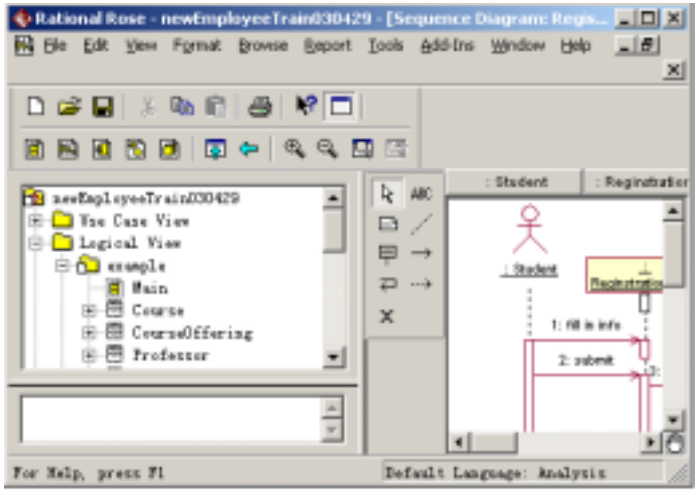
- 受控单元可以被loaded或unloaded
  - 可以load模型的一部分，这样就减少了启动延迟、资源消耗和维护对unloaded的单元的索引
- 可以被write enable或write protect（手动的或自动的）
  - 子受控单元具有独立于父受控单元的写保护
  - 即使有版本控制系统，也可以对受控单元进行写保护控制
  - Reload模型后，写保护失效
  - 手动写保护不影响文件系统的访问权限
- Model workspace是当前装载的受控单元和打开的diagram的快照
  - Model包括组成完成模型的图、元素和受控单元
  - Model workspace包括某个model在某一点上的打开的图和受控单元的实际状态

虚路径

- 虚路径使模型可以在不同的目录结构中移动，可以从不同的workspace上修改。

虚路径使model脱离物理位置

物理路径



虚路径

复用

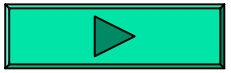
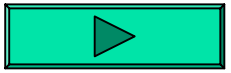
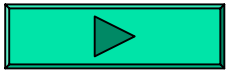
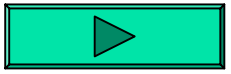
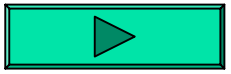
- 复用是指为将来的项目完成和维护工件
- 组织可以从复用大规模的设计元素中获得非常大的利益，这些可以设计元素如
  - 框架（Framework）
    - Rational Rose中的框架是一套用于模型化某种系统并提供一套可复用的组件的预定义的模型元素
  - 构架上具有重大意义的包
  - 子系统
  - 机制
- 为了复用，要
  - 完成重要的工件，如模式（pattern）、框架和组件
  - 给团队所有成员访问可复用工件的权限
  - 要能在一个模型中方便的分类、找到和应用这些工件

模型集成器

- 模型集成器是一个单独的工具用于
  - 比较Rose模型之间的不同
  - 把不同的模型合并成一个

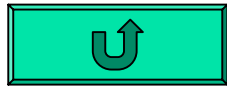
Module4：用例模型

- 为什么要创建以及什么是用例模型
- 用例图的元素
- 事件流和补充规约
- 活动图的元素
- 报告

用例模型

- 为什么要创建用例模型
  - 用例模型允许顾客和系统开发者之间用一种用户可以理解的语言交流系统要做什么
  - 你可以认为用例模型是顾客和开发者之间的可视化契约
- 什么是用例模型
  - 在Use-case View中创建
  - 用例模型代表了从最终用户角度看的系统的功能和环境
  - 是顾客和开发者之间的契约
  - 对于分析、设计和测试活动都是至关重要的
  - 包括用例图、用例事件流和补充规约，也可以包括活动图

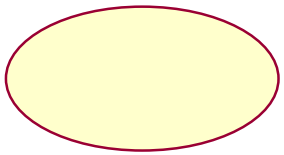


什么是用例图

- 用例图表示了用例和主角以及用例和用例之间的关系
  - 可视化的表示出了客户希望系统做什么
  - 代表一个大的完整的功能
  - 表示系统完成的有明确结果的对主角有价值的一系列动作
- 来源
  - 词汇表
  - 涉众需求
  - 前景文档
  - 用例建模指南
  - 业务用例模型
  - 业务对象模型
  - 补充规约
- 可以模型化
  - 所有的主角和用例（global view）
  - 某个选定主角的所有用例
  - 一个用例以及它所有的关系
  - 一个迭代的所有的用例

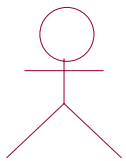
用例图的元素

- 一个用例是系统为一个主角完成的有度量的结果的一系列事务



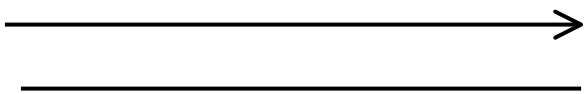
Registration

- 主角是与系统由交互的人或事物
  - 通常可以从问题域中或在与客户或领域专家的交谈中发现



Student

- 关系表示主角和用例或用例与用例之间的联系
  - 通常可以从问题域中或在与客户或领域专家的交谈中发现



事件流和补充规约

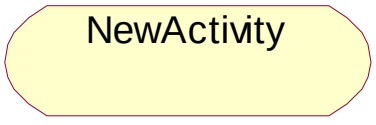
- 事件流是系统对于某个用例做了什么的文本描述，是用例规约(use case specification)的一部分
  - 要被客户理解
  - 描述了用例何时以及如何开始和结束，用例何时与主角交互以及主角与用例之间的信息交换
  - 不描述用户界面的细节
  - 通常是包括在Use-case 包下面的基于文本的文件
  - 一旦与Rose关联，就能直接从Rose中访问
- 补充规约用于细化和描述项目
  - 也可以关联到rose中

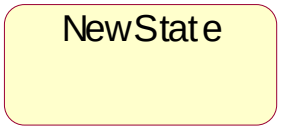
活动图

- 用例模型中的活动图用于获取用例中的活动
- 活动图是一个流程图，表示从活动到活动的控制流
- 用例的流程包括一系列活动，这些活动一起为主角产生有价值的结果
  - 通常由一个基本流和一个或多个备选流组成
  - 可以用活动图表示用例的流程
- 活动图还可以用于其它部分
- 活动图对于用例模型来说不是必需的

活动图中的一些概念

- 活动(activity)代表流程中任务的执行
  - 活动与状态类似，但是活动不等待event

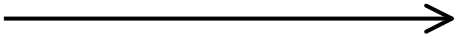




- 初始状态(start state)表示活动图中流程的起点
  - 只能有一个初始状态

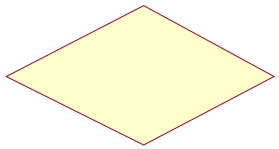
- 最终状态(end state)表示活动图中流程的最终或终结的状态
  - 可以有一个以上的最终状态

- 状态迁移(state transition)表示一个活动跟着另一个活动
  - 可以有一个以上的最终状态



活动图中的一些概念

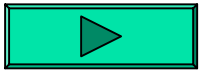
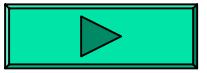
- Decision是活动图中的一点，在这点上用守卫条件表示不同的迁移
  - 也可以用decision表示线程的合并点

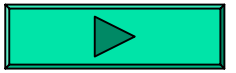


- 同步条（synchronization bar）表示用例的流程中并发的线程

- 泳道(swimlanes)用于划分活动图以帮助我们更好的理解谁或什么初始化活动
  - 一个泳道可能完全由一个组织单元或业务对象模型中的一系列类的活动组成

Module5:用例实现结构

- 什么是用例实现结构 
- 用例实现结构的元素 



什么是用例实现结构

- 通过在Logical View中构建用例实现结构实现系统做什么向如何做的迁移
- 用例实现结构帮助在设计模型中组织实现用例需要的模型元素
- 用例实现结构是设计师需要履行的契约

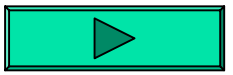
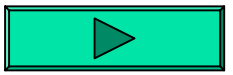
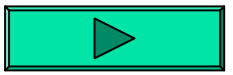
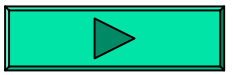
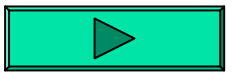
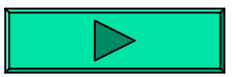
用例实现结构的元素

- 用例实现结构包括下列元素
  - 用例实现包
    - 是组织用例的类和交互图的方式
    - 每个用例都对应一个用例实现包
  - 可跟踪性图

  - 交互图
    - 协作图和时序图
  - 类图

Module6:交互图

- 类
- 交互图
- 时序图的元素
- 协作图的元素
- 报告

类

- 类是拥有相同的属性、操作、关系和语义的一系列对象
  - 在开发周期中，从分析类向设计类演变，分析类属于问题域，设计类属于解决域
- 在开发初期，类的属性和操作不一定被定义，但是我们知道了类的责任
  - 类的责任就是类可以提供的事物的状态或契约
  - 类的责任用//…表示
- 类的识别
  - 用例规约
  - 词汇表
  - 涉众需求
  - 前景文档

交互图

- 交互图通过显示对象以及它们之间分发的消息来模型化系统的动态特征
  - 时序图
  - 协作图
- 输入
  - 用例模型，包括用例事件流
  - 词汇表
  - 软件构架文档
  - 设计模型
  - 设计指南

时序图

- 时序图是强调消息的时间顺序的交互图
  - 象协作图一样，时序图是用于决定类责任和接口的主要信息来源
  - 时序图描述了对象间的交互模式
  - 通过对象的“生命线”和他们相互发送的消息来显示对象
  - 时序图与协作图在语义上是相同的，只是时序图中的消息是按时间顺序分布的
  - Rose可以加上和去掉sequence number
- 组成
  - 主角
  - 对象
  - 消息
  - 生命线
  - Focus of control

时序图的元素

- 对象
  - 是一个具体的有标识和状态的实体
  - 是类的一个实例
  - 对象的表示
    - 只用对象名
    - 对象名和类
    - 只用类名
- 消息
  - 消息是对象之间交流的规约
- Focus of Control
  - Focus of Control表示对象直接或通过子过程执行动作的一段时间

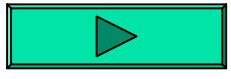
协作图

- 协作图强调参与交互的对象的组织
  - 适合分析活动，适合表示少数对象的简单交互
- 组成
  - 主角
  - 对象
  - Links
    - Link是对象通信的途径
  - 消息

报告

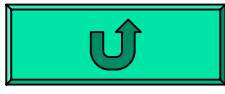
- Show unsolved objects

Module7:类图

- 什么是类图 
- 类图的元素 
- 报告 

类图

- 类图
  - 表示一系列类、接口、协作和它们的关系
  - 表示系统的静态视图，在Logical View中的某个用例实现中
- 输入工件
  - 补充规约
  - 设计指南
  - 设计类
  - 设计模型
  - 用例
  - 用例实现
- 类图可以用于模型化
  - 系统的词汇表
  - 简单的交互
  - 逻辑数据库模式
- View of Participating Classes(VOPC)
  - VOPC类图表示用例实现的参与类以及这些类之间的关系
  - 确保跨越子系统的用例实现的一致性

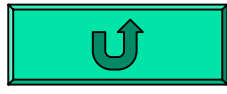


类图的元素

- 类
- 关系
  - Association
    - 最通用的关系，只表示通信
    - 可以存在于主角、用例、类和接口之间
    - 可以是单向的或双向的
    - 元素
      - Association name
      - Role name
      - Multiplicity
  - Aggregation（聚合）
    - 是一种部分与整体的association
    - 也叫“part of”关系
    - Composite（组合）关系：强聚合
  - Generalization
    - 父/子关系
    - 也叫“is a kind of ”关系
    - 也就是继承

报告

- Show Access Violations
  - Export control value
    - Public:对包的外部是可见的，可以被import进模型的其他部分，操作对所有client都可访问的
    - Protected：只对nested类，friend类和类自己是可访问的
    - Private：只对nested类，friend类和类自己是可访问的
    - Implementation:只在定义它的包内可见，操作是类实现的一部分
- Show instance
  - 显示有选定类的实例出现的图



Module8:介绍Round-Trip Engineering

- Round-Trip Engineering是在模型和代码之间相互移动的能力
  - 正向工程
  - 逆向工程
- Round-Trip Engineering帮助维护构架集成
  - 察觉并评估构架上的变化
  - 传达构架上的变化
  - 在迭代中同步你的模型和代码

正向工程和逆向工程

- 正向工程通过映射到一种特定的实现语言把模型转化成代码的能力
- 逆向工程就是从代码向模型转化的能力