软件工程
第7-2章 面向对象设计

设计模型和分析模型

设计模型的元素很多都是在分析模型中使用的UML图。差别在于这些图被精化和细化为设计的一部分，并且提供了更多的与实现相关的特殊细节，突出了架构的结构和风格、架构内存在的构件以及构件和外界之间的接口。

1、面向对象设计模型

设计建模任务：

- 架构设计
- 包和子系统设计
- 类设计
- 持久化设计

UML相关模型图：

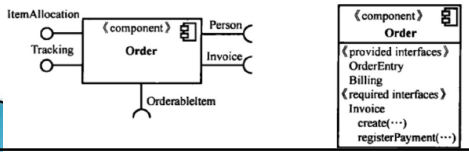
（1）构件图

构件是比类的封装粒度更大的软件重用结构，并通过接口向构件的用户提供服务。

特征：

- 内部类以紧耦合方式完成相对独立的功能；
- 有明确的供给接口和需求接口；
- 构件是可配置的；
- 构件是可组装的；

构件的表示：



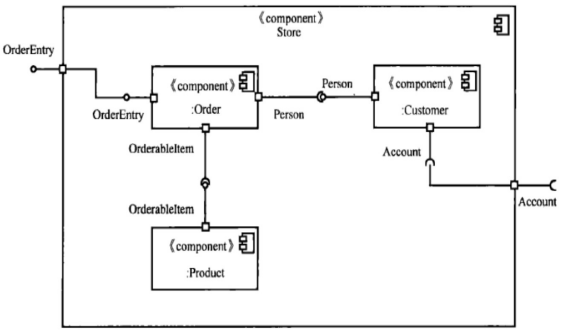


图 7-1 UML规范中给出的构件图实例

（2）部署图

软件的部署方案定义了系统的执行架构，即将软件制品分配到不同的节点上运行。

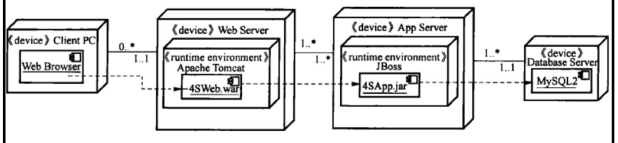


图 7-3 4S系统的部署方式之一

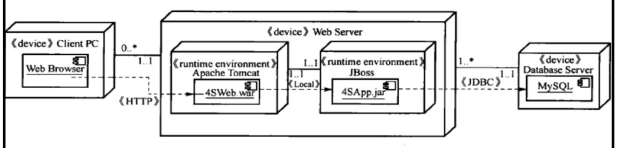


图 7-4 4S系统的部署方式之二

（3）状态机图

描述类的行为特性。

核心元素：状态、转移、事件

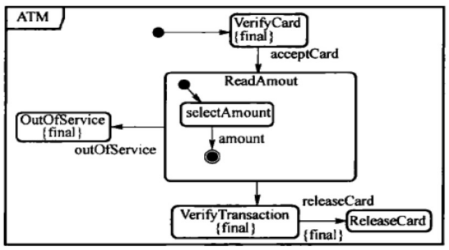


图 7－5 ATM 自动提款机状态机图实例

2、架构设计

（1）4+1架构视图

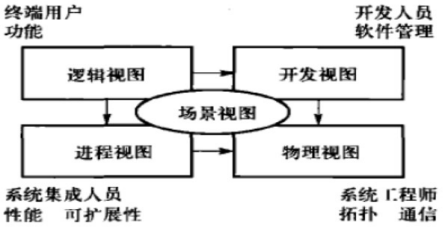


图 7－6 软件架构的 4＋1 模型

（2）逻辑视图设计

软件的逻辑结构，用于支持功能性需求。

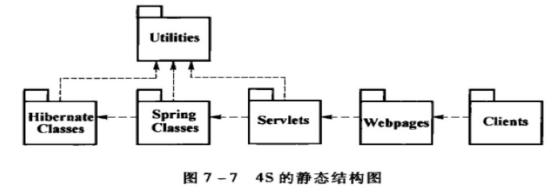


图 7－7 4S 的静态结构图

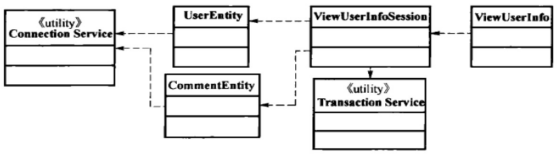


图 7－8 有关查看用户信息功能的局部静态结构图

（3）进程视图设计

软件的进程架构，针对非功能性需求。

构成进程的任务是彼此相互分隔的控制线程，这个软件被划分成这样一组彼此独立的任务。（部署图）

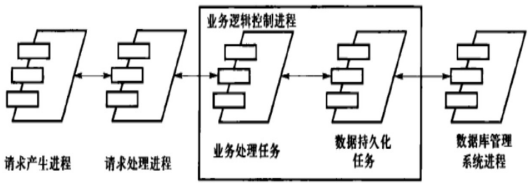
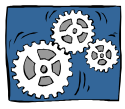


图 7－9 4S 系统的一种可能的进程视图

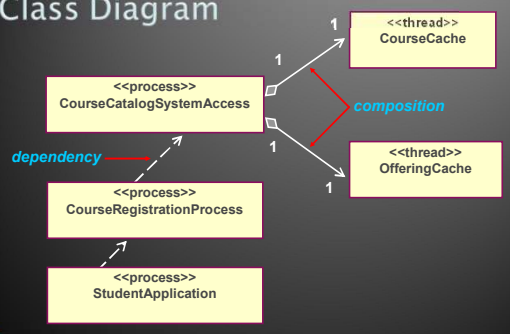
Key Concepts: Process and Thread

- Process
  - Provides heavyweight flow of control
  - Is stand-alone
  - Can be divided into individual threads



- Thread
  - Provides lightweight flow of control
  - Runs in the context of an enclosing process

Example: Modeling Processes: Class Diagram

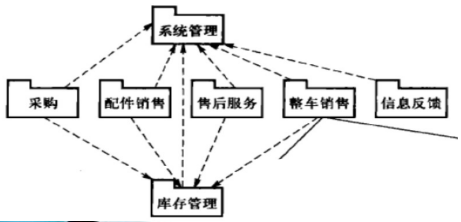


（4）开发视图设计

软件的开发架构，即如何分解成实现单元，是需求分配的基础也是开发组织结构的基础。

如图7-7水平分割方案，6个包分配给6个开发组且处于架构的不同层次，开发组可按技术层次分配人员。

下图垂直分割方案，按业务逻辑在多个开发组分配任务，每个开发组必须具备综合开发能力。



（5）物理视图设计

软件的物理架构，针对非功能性需求的可用性、可靠性、可扩展性等。（部署图）

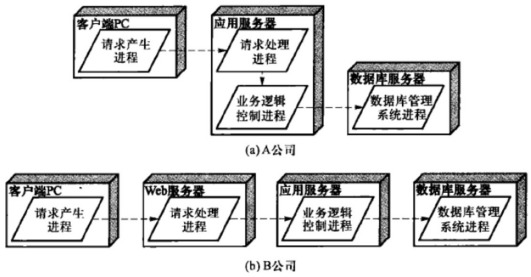


图 7－10　4S 系统在两种不同使用情况下的物理架构

（6）场景视图设计

场景是用例的实例，将4个视图有机关联起来。它是发现架构元素的动力，担负起验证和说明的角色。

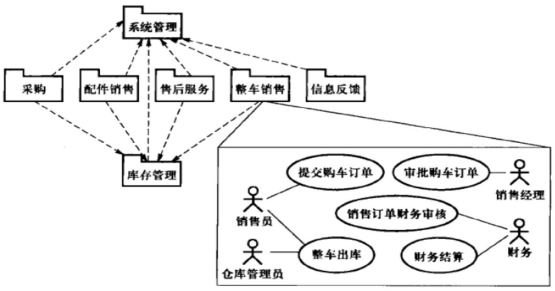


图 5－14　4S 系统的用例模型

How many views?

- Simplified models to fit the context
- Not all systems require all views:
  - Single processor: drop deployment view
  - Single process: drop process view
  - Very Small program: drop implementation view
- Adding views:
  - Data view, security view

3、包和子系统设计

Review: Class and Package

- What is a class?
  - A description of a set of objects that share the same responsibilities, relationships, operations, attributes, and semantics

- What is a package?
  - A general purpose mechanism for organizing elements into groups
  - A model element which can contain other model elements

包设计原则：

重用-发布等价：重用粒度等于发布粒度；
共同重用：包中所有类一起被重用；
共同封闭：包中的所有类对同类型的变更封闭；
无环依赖：包之间无环依赖结构；
稳定依赖：包应该依赖比他更稳定的包；
稳定抽象：最稳定的包即最抽象，不稳定包是具体包。

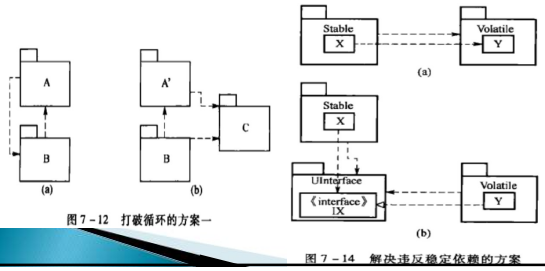
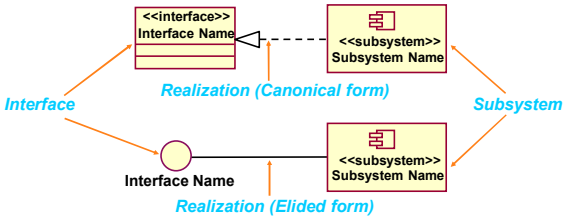


图 7－12　打破循环的方案一

图 7－14　解决违反稳定依赖的方案

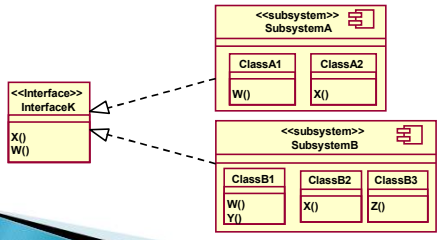
Subsystems and Interfaces

- Realizes one or more interfaces that define its behavior



Subsystems and Interfaces (continued)

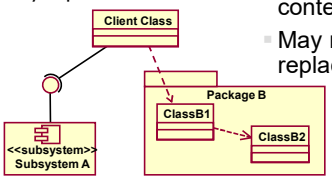
- Subsystems :
  - Completely encapsulate behavior
  - Represent an independent capability with clear interfaces (potential for reuse)
  - Model multiple implementation variants



Packages versus Subsystems

Subsystems
- Provide behavior
- Completely encapsulate their contents
- Are easily replaced

Packages
- Don't provide behavior
- Don't completely encapsulate their contents
- May not be easily replaced



Encapsulation is the key!

Subsystem Usage

- Subsystems can be used to partition the system into parts that can be independently:
  - ordered, configured, or delivered
  - developed, as long as the interfaces remain unchanged
  - deployed across a set of distributed computational nodes
  - changed without breaking other parts of the systems
- Subsystems can also be used to:
  - partition the system into units which can provide restricted security over key resources
  - represent existing products or external systems in the design (e.g. components)

Subsystems raise the level of abstraction.

4、类设计

单一职责、李氏替换、依赖倒置、接口隔离、开发-关闭原则。

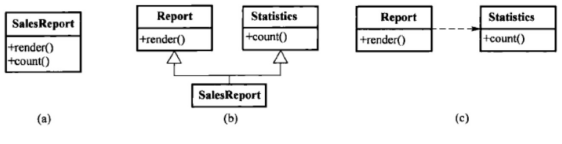


图 7－15 单一职责原则实例

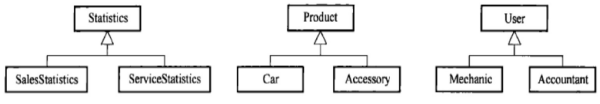


图 7－16 里氏替换原则实例

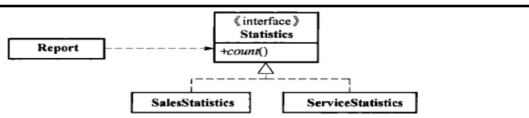


图 7－17 依赖倒置原则实例

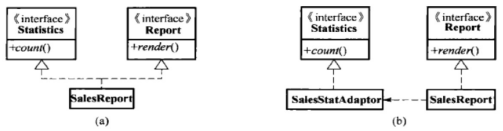


图 7－18 接口隔离原则实例

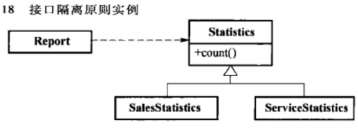


图 7－19 开放－关闭原则实例

5、持久化设计

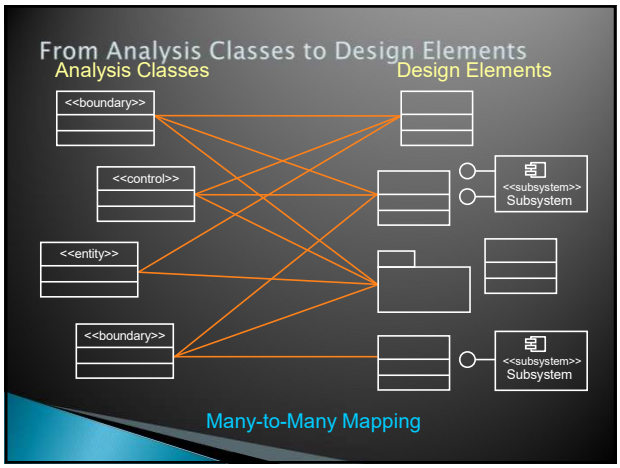
（1）实体对象建模；

（2）数据库设计；

（3）持久化框架。

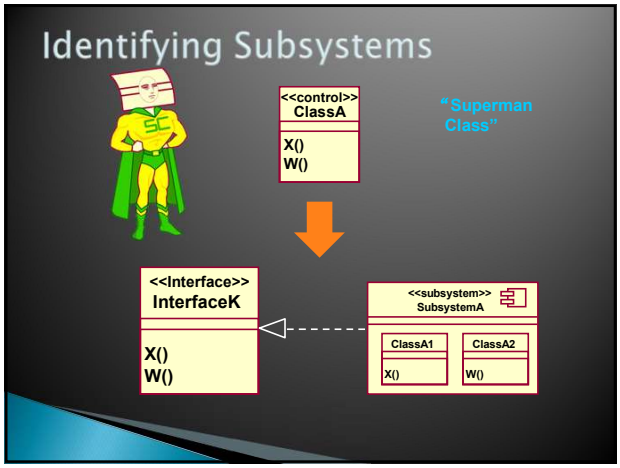
6、面向对象设计过程

- 识别设计元素
- 确定架构风格，设计整体结构
- 构件级设计



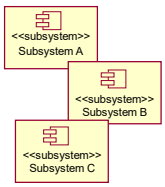
Identifying Design Classes

- An analysis class maps directly to a design class if:
  - It is a simple class
  - It represents a single logical abstraction
- More complex analysis classes may
  - Split into multiple classes
  - Become a package
  - Become a subsystem (discussed later)
  - Any combination …



Candidate Subsystems

- Analysis classes which may evolve into subsystems:
  - Classes providing complex services and/or utilities
  - Boundary classes (user interfaces and external system interfaces)
- Existing products or external systems in the design (e.g., components):
  - Communication software
  - Database access support
  - Types and data structures
  - Common utilities
  - Application-specific products



子系统设计过程：

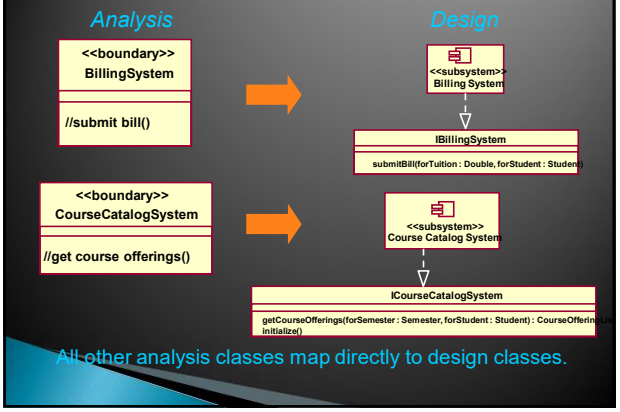
（1）对子系统职责进行定义，即接口的定义；

（2）通过职责分配确定子系统中的元素，由构件等元素来实现职责；

（3）对子系统中各元素进行设计，即类设计（静态结构和动态结构）；

（4）确定子系统间的依赖关系。

Example: Design Subsystems and Interfaces

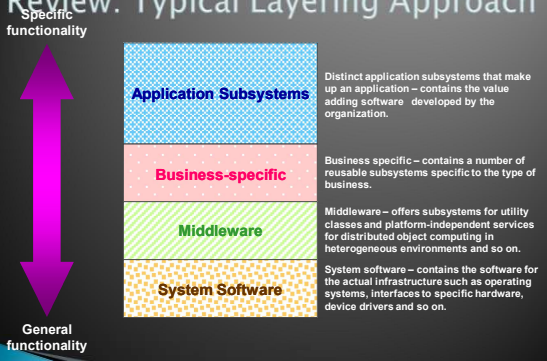


All other analysis classes map directly to design classes.

识别架构风格

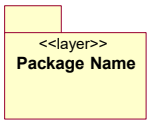
- Layers
- Model-view-controller (M-V-C)
- Pipes and filters
- Blackboard
- ……

Review: Typical Layering Approach

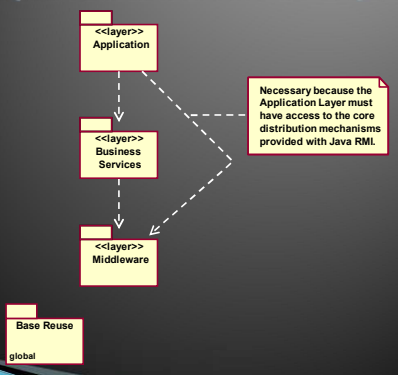


Modeling Architectural Layers

- Architectural layers can be modeled using stereotyped packages.
- <<layer>> stereotype

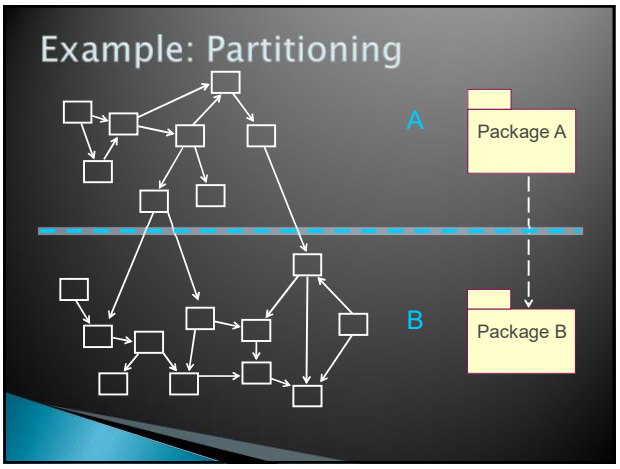


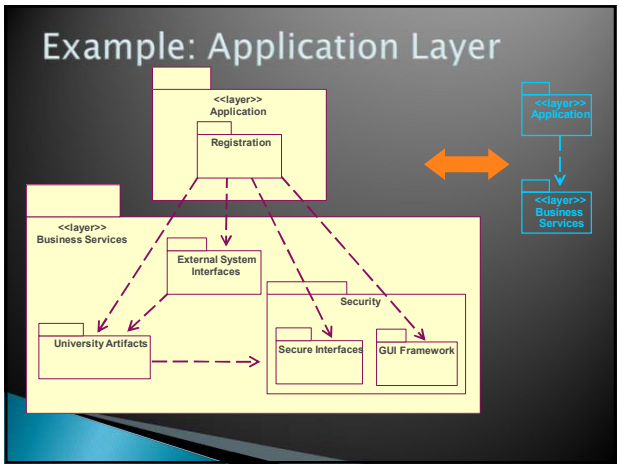
Example: Architectural Layers

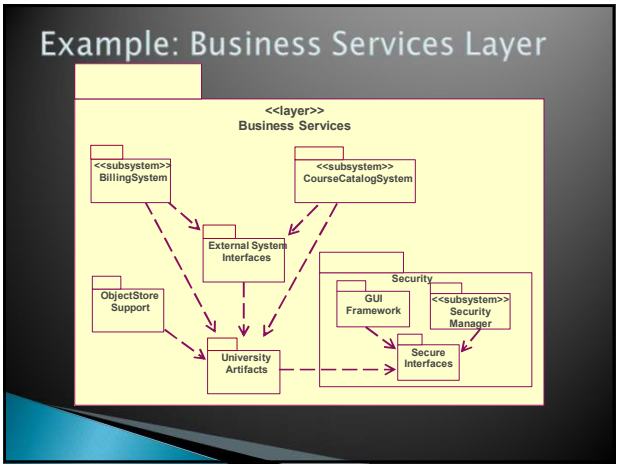


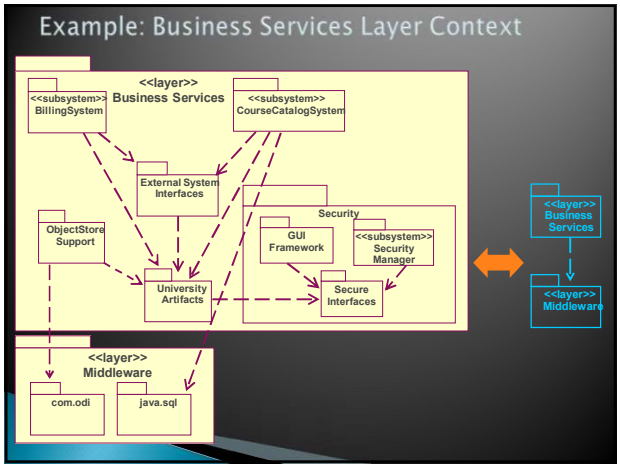
对设计元素进行分层和分包

- Partitioning Considerations
  - Coupling and cohesion
  - User organization
  - Competency and/or skill areas
  - System distribution
  - Secrecy
  - Variability



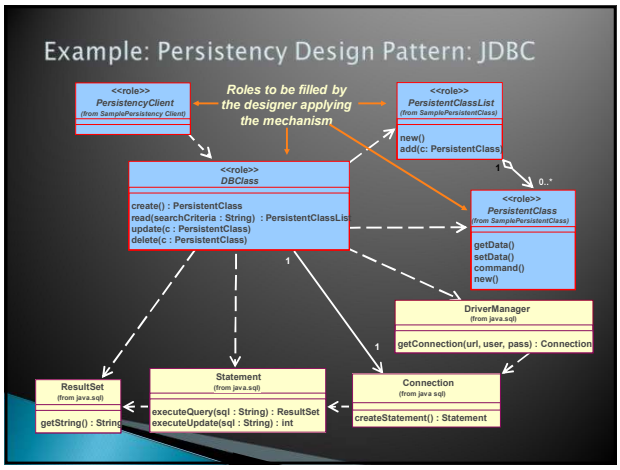






Example: Persistence Design Pattern

- Describe Persistence-Related Behavior
  - Modeling Transactions
  - Writing Persistent Objects
  - Reading Persistent Objects
  - Deleting Persistent Objects



构件级设计的步骤

1) 更新 *Use-case Realization*
2) 子系统的设计
3) 类的设计

Example: Incorporating Subsystem Interfaces

Analysis Classes

<<boundary>>
BillingSystem
//submit bill()

<<boundary>>
CourseCatalogSystem
//get course offerings()

Design Elements

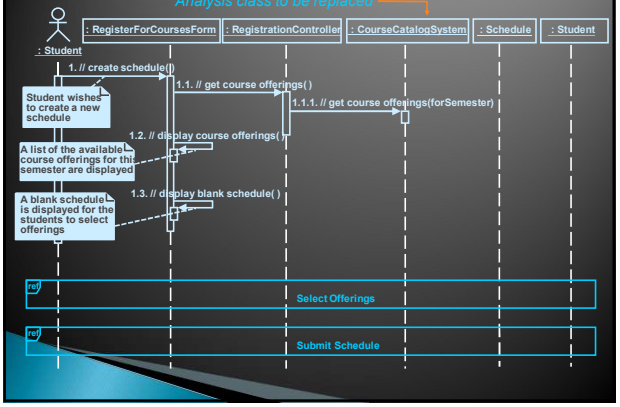
<<subsystem>>
Billing System
IBillingSystem
submitBill(forTuition : Double, forStudent : Student)

<<subsystem>>
Course Catalog System
ICourseCatalogSystem
getCourseOfferings(forSemester : Semester, forStudent : Student) : CourseOfferingList
initialize()

All other analysis classes are mapped directly to design classes.

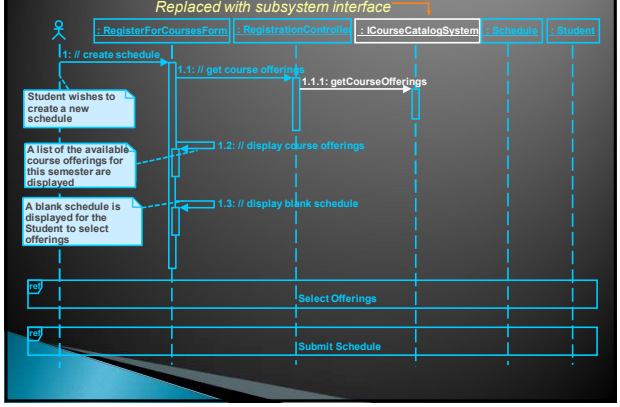
Example: Incorporating Subsystems (Before)

Analysis class to be replaced



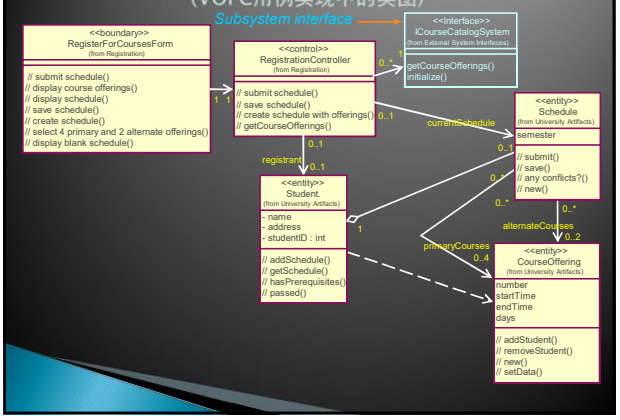
Example: Incorporating Subsystems (After)

Replaced with subsystem interface



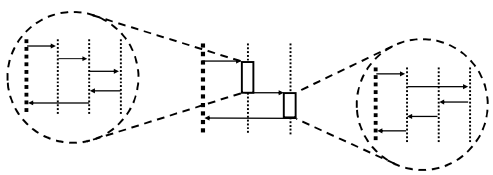
Example: Incorporating Subsystem Interfaces (VOPC用例实现中的类图)

Subsystem interface



Encapsulating Subsystem Interactions

- Interactions can be described at several levels
- Subsystem interactions can be described in their own interaction diagrams

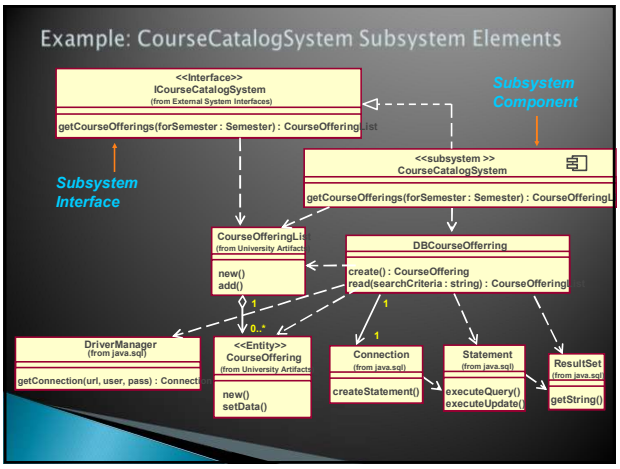


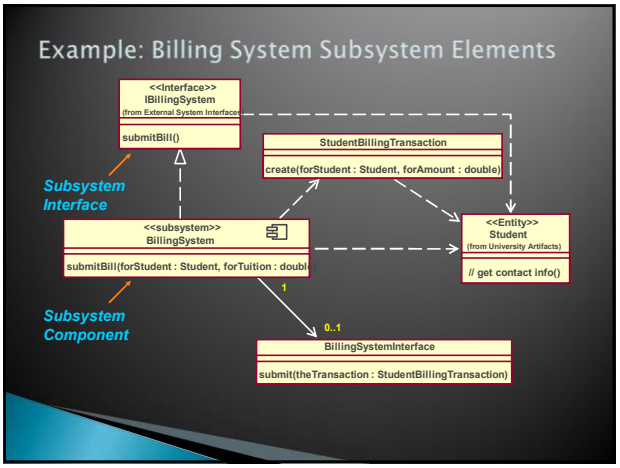
Raises the level of abstraction.

子系统设计

- 对每个子系统进行设计
  - 对每个接口的每个操作设计交互图和VOPC类图

类似于系统的设计
即Use case realization



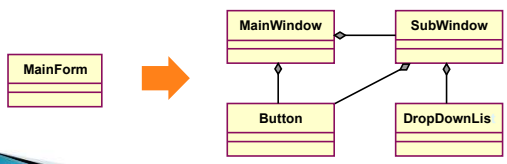


类设计过程：

（1）创建设计类：将分析类映射成设计类；
（2）定义操作：实现单一的职责；
（3）定义方法：对操作的内部实现进行描述；
（4）定义状态：描述对象的状态对行为的影响，将对象的属性和操作关联起来；
（5）定义属性：包括方法中的参数、对象的状态等；
（6）定义依赖：类与类之间的存在关系，非结构关系；
（7）定义关联：对关联关系的细化，包括聚合与组合、导向性、多重性、关联类；
（8）形成设计类的规格说名书。

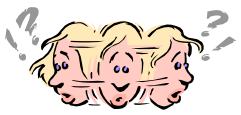
优化UI类

- User interface (UI) boundary classes
  - What user interface development tools will be used?
  - How much of the interface can be created by the development tool?



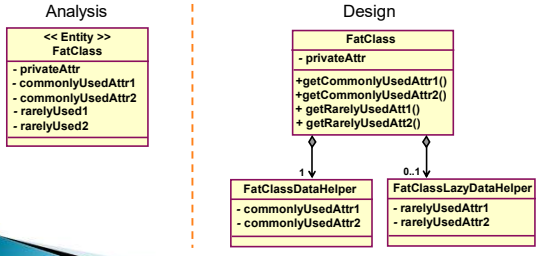
优化Control类

- What happens to Control Classes?
  - Are they really needed?
  - Should they be split?
- How do you decide?
  - Complexity
  - Change probability
  - Distribution and performance
  - Transaction management



优化Entity类

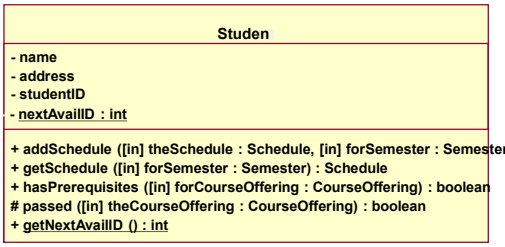
- Entity objects are often passive and persistent
- Performance requirements may force some re-factoring
- See the Identify Persistent Classes step



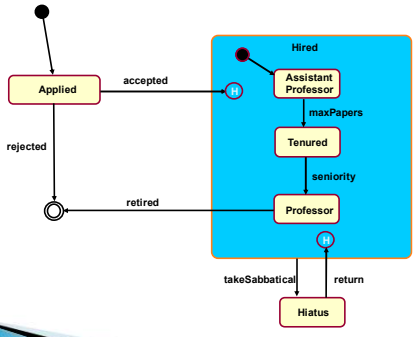
Define operation signatures

- Operation Signatures
  - operationName([direction]parameter : class,..) : returnType
- Operation Visibility
  - + Public access
  - # Protected access
  - - Private access
- Operation Scope
  - Instance: one instance for each class instance
  - Classifier: one instance for all class instances

Example: Operation

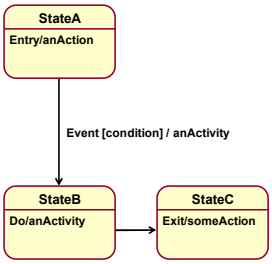


Example: State Machine



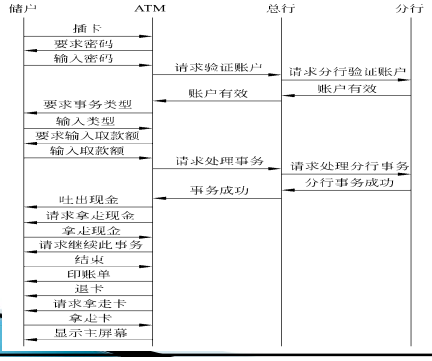
Activities and Actions

- Entry
  - Executed when the state is entered
- Do
  - Ongoing execution
- Exit
  - Executed when the state is exited
- Event [condition] / anActivity
  - Executed during transition



画出时序图

从用例事件流中提取出各类事件并确定事件交互行为的发送对象和接受对象，用时序图把事件序列以及事件与对象的关系表示出来。（如下为简化的UML顺序图）



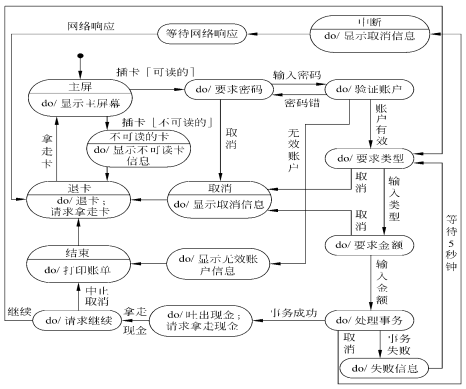
画状态机图(Drawing States Diagram)

状态图描绘事件与对象状态的关系。当对象接受了一个事件以后，引起的状态改变称为“转换”。

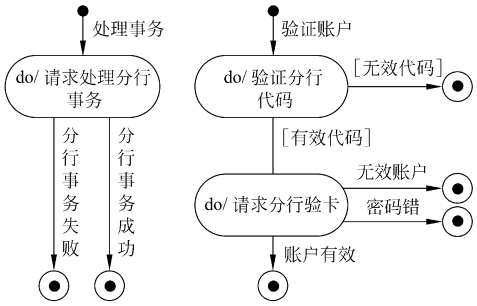
用一张状态图描绘一类对象的行为，它确定了由事件序列引出的状态序列。仅考虑具有重要交互行为的那些类。

事件跟踪图中入事件作为状态图中的有向边(即箭头线)，边上标以事件名。两个事件之间的间隔就是一个状态。

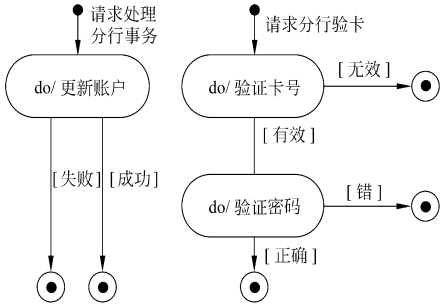
事件跟踪图中的射出的箭头线，是这条竖线代表的对象达到某个状态时所做的行为(往往是引起另一类对象状态转换的事件)。



ATM机类的状态机图



总行类的状态机图



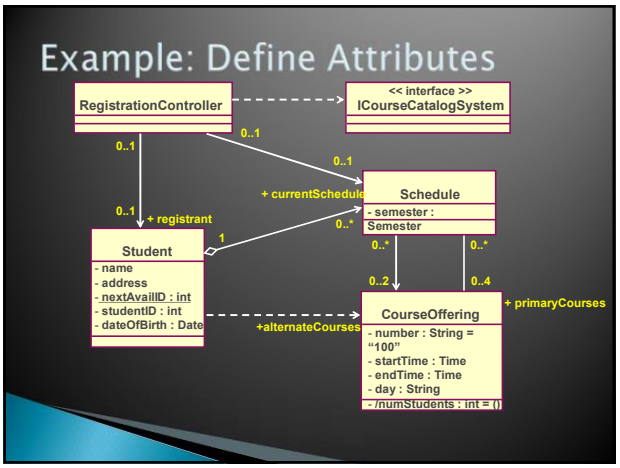
分行类的状态机图

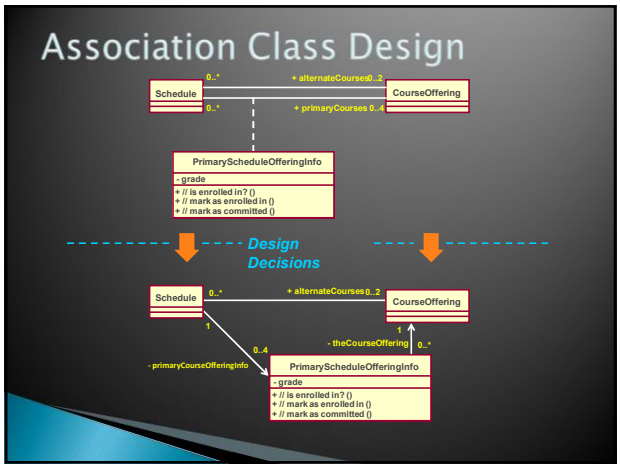
Which Objects Have Significant State?

- Objects whose role is clarified by state transitions
- Complex use cases that are state-controlled
- It is not necessary to model objects such as:
  - Objects with straightforward mapping to implementation
  - Objects that are not state-controlled
  - Objects with only one computational state

Attribute Representations

- Specify name, type, and optional default value
  - attributeName : Type = Default
- Follow naming conventions of implementation language and project
- Type should be an elementary data type in implementation language
  - Built-in data type, user-defined data type, or user-defined class
- Specify visibility
  - Public: +
  - Private: -
  - Protected: #





Round-Trip Engineering (双向工程)

- Round-trip engineering is the ability to move back and forth between model and code.
  - Forward engineering
  - Reverse engineering