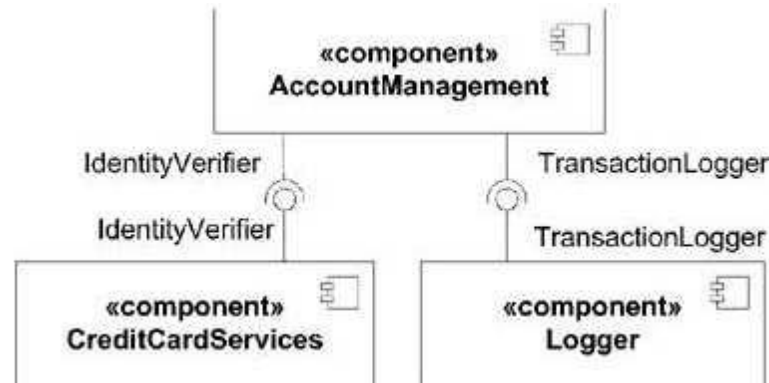


Part 1

1. UML中有多种类型的图，其中，用例图对系统的使用方式进行分类，类图显示了类及其相互关系，活动图显示人或对象的活动，其方式类似于流程图，通信图显示在某种情况下对象之间发送的消息，顺序图与通信图类似，但强调的是顺序而不是连接。
2. 下图属于UML中的组件图，其中，AccountManagement需要调用CreditCardServices实现的IdentityVerifier接口。



3. 状态图通过建立类对象的生命周期模型来描述对象随时间变化的动态行为；状态图适用于描述状态和动作的顺序，不仅可以展现一个对象拥有的状态，还可与说明事件如何随着时间的推移来影响这些状态；状态图描述了一个实体基于事件反应的动态行为，显示了该实体如何根据当前所处状态对不同的事件作出反应。
4. 在ATM自动取款机的工作模型中（用户通过输入正确的用户资料，从银行取钱的过程），用户、ATM取款机和ATM取款机管理员都是“Actor”，而“取款”则属于Use Case。
5. Composite Structure Diagram反映类、接口或构件的内部协作，用于表达运行时的体系结构、使用模式及关系。
6. 在RSA中包含很多模型模板，供开发者在系统建模时选用，如Analysis Model、Service Design Model 和XSD Model均是RSA提供的模型。

- 7.在UML的各种视图中，用例视图显示外部参与者观察到的系统功能；逻辑视图从系统的静态结构和动态行为角度显示系统内部如何实现系统的功能；实现视图显示的是源代码以及实际执行代码的组织结构。在 ROSE中，时序图和协作图（或通信图）通常建立在 逻辑视图下的**use case realization**包中。
- 8.在面向对象分析与设计中，实体类是应用领域中的核心类，一般用于保存系统中的信息以及提供针对这些信息的相关处理行为；边界类是系统内对象和系统外参与者的联系媒介；控制类主要是协调上述两种类对象之间的交互。
- 9.采用UML进行软件建模过程中，类图是系统的一种静态视图，用聚合关系可明确表示两类事物之间存在的整体/部分形式的关联关系。
- 10.Round-Trip Engineering能够帮助维持软件架构的完整性，其具有发现和评估软件架构上的改动、在每次迭代中保持模型和代码的同步、传达被接受的架构改动等好处。
- 11.Observer（观察者）设计模式定义了对象间的一种一对多的依赖关系，以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新。组合（**Composite**）模式将对象组合成树形结构以表示“部分-整体”的层次结构，并使得用户对单个对象和组合对象的使用具有一致性。
12. 继承反映了类间的一种层次关系，而组合反映了一种整体与部分的关系。
- 13.泳道技术是将一个活动图中的活动状态进行分组，每一组表示一个特定的类、人或部门，他们负责完成组内的活动。
- 14.包是用于把元素组织成组的通用机制。

1. 在用例建模中，**Scenario**（场景）强调的是可理解性，而用例强调的是完整性。
2. 在两个用例中，如果一个用例拥有另一个用例的所有结构、行为和关系，并在此基础上增加了新的特性，则此两个用例之间可以用泛化关系表示。
3. **Controlled Unit**是可以进行版本控制的模型元素，在**ROSE**中，模型文件本身被打包存储为.mdl文件从而成为受控单元，**Logical View**和**Use CaseView**被打包成.cat文件而成为受控单元。
4. 每一种UML图都能用于多个模型中，同样一个模型元素也能够出现在多个图中。
5. **RSA**所支持的查询图不是**UML2.0**中新增加的一种图，它包括浏览图（**Browse Diagram**）和主题图（**Topic Diagram**）。
6. 模型驱动的开发（**Model-Driven Development**）包含**CIM**、**PIM**和**PSM**等抽象层次。
7. 在状态图中，内部转换不会导致进入转换和离开转换的执行。
8. **UML**不仅适用于以体系结构为中心的开发过程，也适合在具有迭代特征的开发过程中使用。
9. 请求接口（**Required Interface**）用于定义一个构件所请求外部提供的服务。
10. **Use Case Realization** 和相应的**Use Case**之间是一种实现关系。
11. 分析机制（**Analysis mechanisms**）通常用于分析阶段，通过提供对系统复杂行为（如安全性、持久存储等）的简短描述来减少分析的复杂性并改善软件在各开发阶段一致性。
12. 在**RUP**中，识别设计元素（**Identify Design Elements**）是精化体系结构（**Refine the Architecture**）活动中的一个步骤。
13. 在**ROSE**中，从**Browser**窗口删除图形元素和从**Diagram**窗口中删除模型元素的效果并不相同。
14. 需求工件（**artifact**）中至少应该包括用例模型、术语表（**glossary**）和补充说明（**supplementary specification**）三份文档。
15. **UML**结构包括构造块、公共机制和架构三个部分。

1. Sequence Diagram 和 Collaboration Diagram 的异同点:

相同点: Sequence Diagram 和 Collaboration Diagram 都是用于描述模型动态特性的交互图。

Sequence Diagram 和 Collaboration Diagram 从语意上讲是相同的, 他们只是从不同的方面来描述一次交互。

不同点: Sequence Diagram 重点强调消息的时间顺序; Collaboration Diagram 强调一次交互中各个对象之间的关系。

2. 说明在 RSA 中将模型进行分块 (partition) 的原因。为什么有时候会结合 (combine) 几个没有共同祖先的模型?

To make team development of models possible when you have large models and teams. To avoid conflicts and potential merges.

You can combine models that are not related by a common ancestry, such as independently created models.

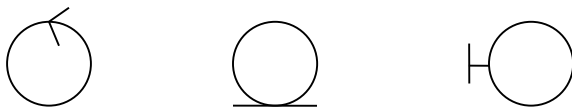
You (as a modeler) might use this feature if you want to assemble a number of models, ones that were created informally at the beginning of your project, into a formal set of models that can be managed with version control.

3. Activity diagram 和 State diagram 描述的重点不同: Activity diagram 描述的是从 activity 到 activity 的控制流, 而 State diagram 描述的是对象的状态及状态之间的转移。

Activity diagram 和 State diagram 使用的场合不同, 对于以下几种情况可以使用 Activity diagram: 分析用例、理解涉及多个用例的工作流、处理多线程应用。对于下面的情况要使用 State diagram: 显示一个对象在其生命周期内的行为。

4. Stereotypes(版型)的作用: 用来扩展 UML 元素的语意。

例子:



5. 说明UML中的关联、泛化、实现、依赖四种关系各自的含义：

关联表示两个类之间存在某种语义上的联系；

泛化关系描述了一般事物和该事物中的特殊种类之间的关系；

实现关系是用来规定接口与实现接口的类或组件之间的关系；

两个元素X、Y，如果修改元素X的定义可能会引起另一个元素Y的定义的修改，则称元素Y依赖于元素X。

组合和聚集都表示实例之间的整体/部分关系。组合是聚集的一种形式。聚集是概念性的，只是区分整体与部分。组合具有很强的归属关系，而且整体与部分的对象生存周期是一致的。

1. 阅读下列说明和图，完成问题1和问题2。

【说明】在线会议审稿系统（Online Reviewing System, ORS）主要处理会议前期的投稿和审稿事务，其中提交稿件的流程描述如下：

作者登录（login）后方可提交稿件，提交稿件必须在规定提交时间范围内，其过程为先输入标题和摘要，选择稿件所属主题类型，选择稿件所在位置（存储位置）。上述几步若未完成，则重复；若完成，则上传稿件至数据库中，系统发送通知。

系统采用面向对象的方法开发，使用UML进行建模。提交稿件涉及的活动名称参见表1-1，系统提交稿件的活动图如图1-1所示。

表1-1 活动名称列表			
名称	说明	名称	说明
select paper location	选择稿件位置	Upload paper	上传稿件
select subject group	选择主题类型	send notification	发送通知
enter title and abstract	输入标题和摘要		

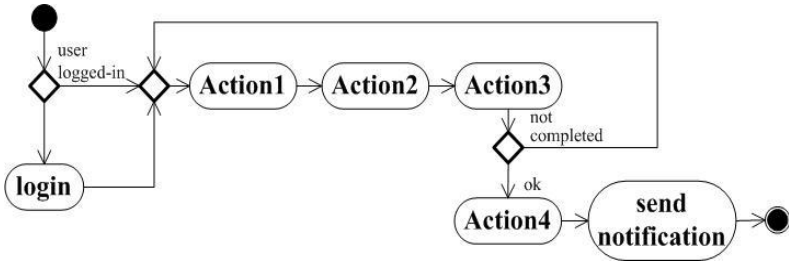


图1-1 提交稿件过程的活动图

【问题1】根据【说明】中的描述，使用表1-1中的英文名称，给出图1Action1~Action4对应的活动如下表：

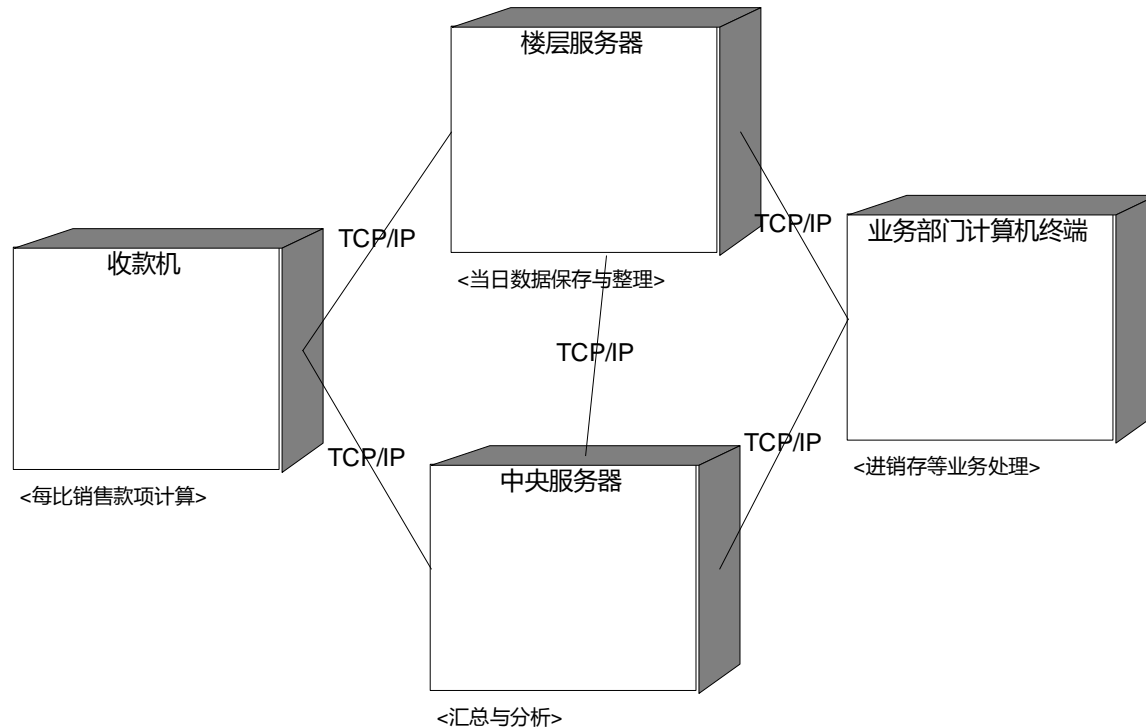
Action1	enter title and abstract
Action2	select subject group
Action3	select paper location
Action4	upload paper

【问题2】举例说明活动图中分支（branch）/合并（merge），与分叉（fork）/汇合（join）这两组概念的区别。

动作流的条件行为用分支与合并表示。

在活动图在用分叉和汇合来表达并发和同步行为。

2. 某大型商场的管理信息系统是由一个中央服务器、每个楼层的楼层服务器、各柜台的收款机和各个业务部门的计算机终端组成的局域网络，它们分别负责商场数据的汇总与分析、当日数据的保存与整理、每笔销售款项的计算和进销存等各种业务的处理。用配置图（或称部署图）描述各项任务在不同硬件设备上的配置情况。



3. 阅读下列说明和图，回答问题1至问题3。

【说明】某银行计划开发一个自动存提款机模拟系统（ATM System）。系统通过读卡器（CardReader）读取ATM卡；系统与客户（Customer）的交互由客户控制台（CustomerConsole）实现；银行操作员（Operator）可控制系统的启动（System Startup）和停止（System Shutdown）；系统通过网络和银行系统（Bank）实现通信。当读卡器判断用户已经将ATM卡插入后，创建会话（Session）。会话开始后，读卡器进行读卡，并要求客户输入个人验证码（PIN）。系统将卡号和个人验证码信息送到银行系统进行验证。验证通过后，客户可从菜单选择如下事务（Transaction）：

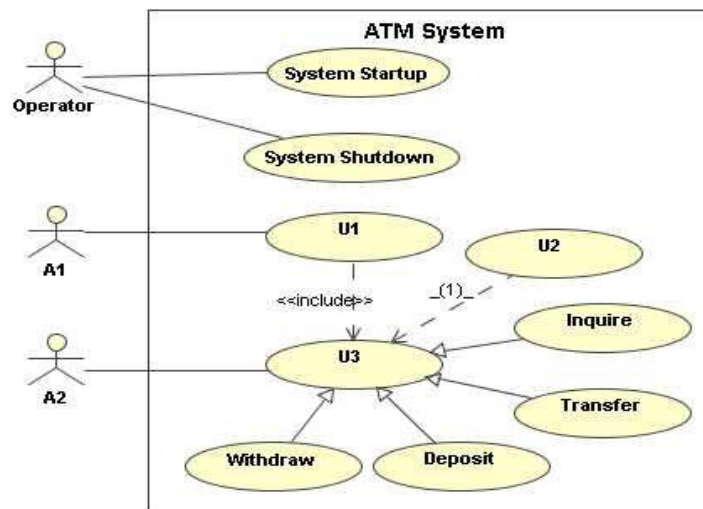
- 1.从ATM卡帐户取款（Withdraw）；
- 2.向ATM卡帐户存款（Deposit）；
- 3.进行转帐（Transfer）；
- 4.查询（Inquire）ATM卡帐户信息。

一次会话可以包含多个事务，每个事务处理也会将卡号和个人验证码信息送到银行系统进行验证。若个人验证码错误，则转个人验证码错误处理（Invalid PIN Process）。每个事务完成后，客户可选择继续上述事务或退卡。选择退卡时，系统弹出ATM卡，会话结束。

系统采用面向对象方法开发，使用UML进行建型。系统的顶层用例图如图3-1所示，一次会话的序列图（不考虑验证）如图3-2所示。消息名称见表3-1。

表3-1 可能的消息名称列表

名称	说明	名称	说明
<u>cardInserted()</u>	ATM卡已插入	<u>performTransaction()</u>	执行事务
<u>performSession()</u>	执行会话	<u>readCard()</u>	读卡
<u>readPIN()</u>	读取个人验证码	<u>PIN</u>	个人验证码信息
<u>create(atm,this,card,pin)</u>	为当前会话创建事务	<u>create(this)</u>	为当前ATM创建会话
<u>card</u>	ATM卡信息	<u>doAgain</u>	执行下一个事务
<u>ejectCard()</u>	弹出ATM卡		



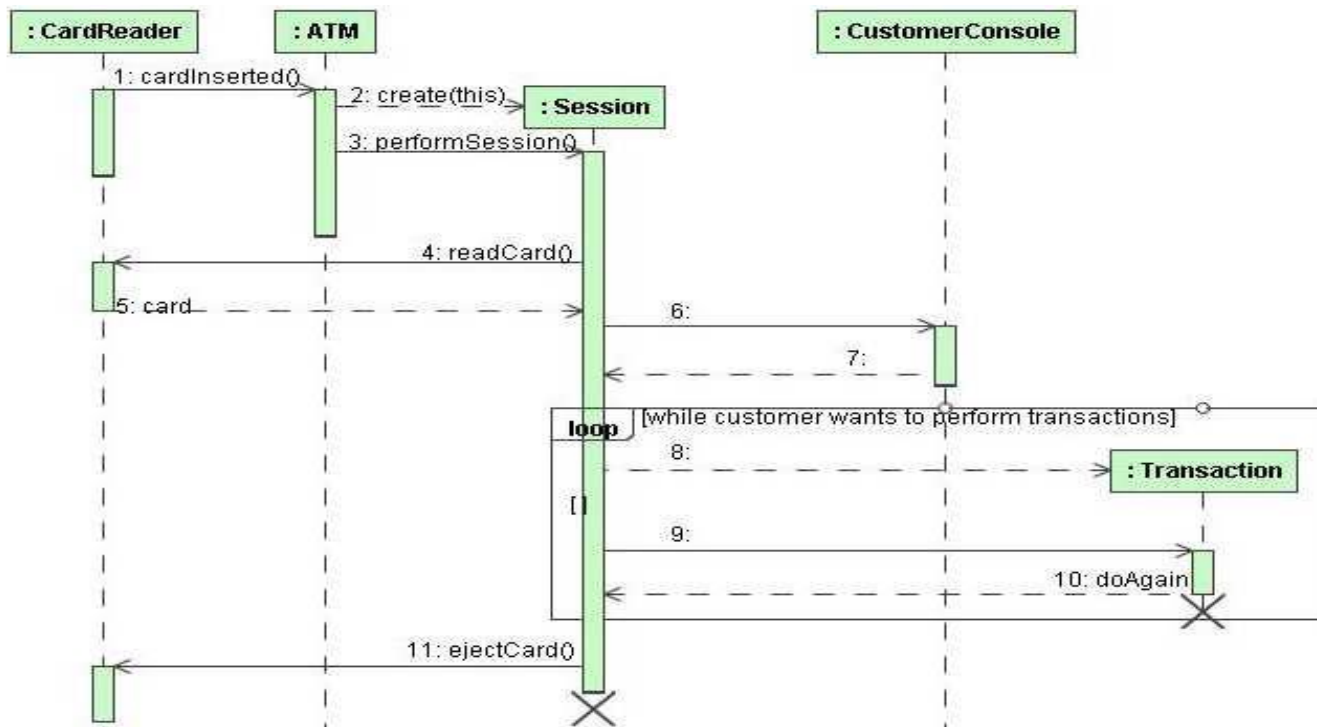


图3-2 一次会话的序列图（无验证消息）

【问题1】根据说明中的描述，给出图3-1中A1和A2所对应的参与者，U1至U3所对应的用例，以及该图中空（1）所对应的关系。（U1至U3的可选用例包括：Session、Transaction、Insert Card、Invalid PIN Process和Transfer）

A1: Customer A2: Bank U1: Session

U2: Invalid PIN Process U3: Transaction (1): 《extend》

【问题2】根据说明中的描述，使用表3-1中的英文名称，给出图3-2中6~9对应的消息。

6: readPIN() 7: PIN 8: create(atm, this, card, pin) 9: performTransaction()

【问题3】解释图3-1中用例U3和用例Withdraw、Deposit等四个用例之间的关系及其内涵。

Transaction是一个抽象泛化用例，具有其他事务类型共有的属性和行为，每个具体的事务类型继承它，并实现适合自己的特定的操作。

4. 阅读下列说明和图，回答问题1和问题2。

【说明】某汽车停车场欲建立一个信息系统，已经调查到的需求如下：

- (1) 在停车场的入口和出口分别安装一个自动栏杆、一台停车卡打印机 / 一台读卡器以及一个车辆通过传感器，示意图如图4-1所示。



图4-1 自动停车场示意图

- (2) 当汽车到达入口时，驾驶员按下停车卡打印机的按钮获取停车卡。当驾驶员拿走停车卡后，系统命令栏杆自动抬起；汽车通过入口后，入口处的传感器通知系统发出命令，栏杆自动放下。
- (3) 在停车场内分布着若干个付款机器。驾驶员将在入口处获取的停车卡插入付款机器，并缴纳停车费。付清停车费之后，将获得一张出场卡，用于离开停车场。
- (4) 当汽车到达出口时，驾驶员将出场卡插入出口处的读卡器。如果这张卡是有效的，系统命令栏杆自动抬起；汽车通过出口后，出口处的传感器通知系统发出命令，栏杆自动放下。若这张卡是无效的，系统不发出栏杆抬起命令而发出告警信号。
- (5) 系统自动记录停车场内空闲的停车位的数量。若停车场当前没有车位，系统将在入口处显示“车位已满”信息。这时，停车卡打印机将不再出卡，只允许场内汽车出场。

根据上述描述，采用面向对象方法对其进行分析与设计，得到了如表4-1所示的用例/类/状态列表、如图4-2所示的初始类图，以及如图4-3所示的描述入口自动栏杆行为的UML状态图。

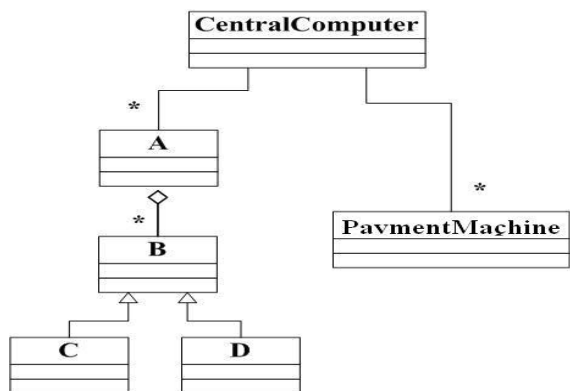


图4-2 初始类图

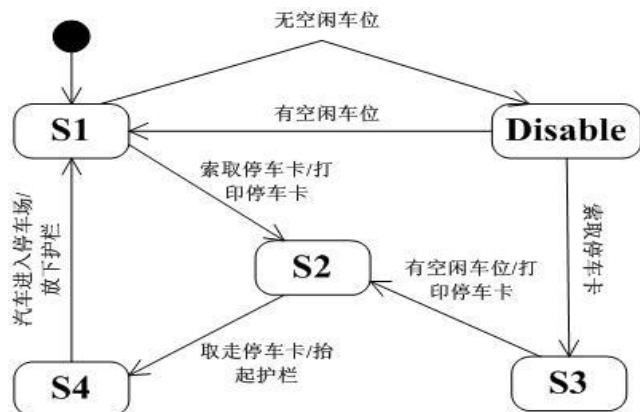


图4-3 入口自动栏杆行为的状态图

表4-1 用例/类/状态列表

用例名	说明	类名	说明	状态名	说明
Car entry	汽车进入停车场	<u>CentralComputer</u>	停车场信息系统	Idle	空闲状态，汽车可以进入
Car exit	汽车离开停车场	<u>PaymentMachine</u>	付款机器	Disable	没有车位
Report Statistics	记录停车场的相关信息	<u>CarPark</u>	停车场，保存车位信息	Await Entry	等待汽车进入
		<u>Barrier</u>	自动护栏	Await Ticket Take	等待打印停车卡
Car entry when full	没有车位时，汽车请求进入停车场	<u>EntryBarrier</u>	入口的护栏	Await Enable	等待停车场内有空闲车位
		<u>ExitBarrier</u>	出口的护栏		

【问题1】根据【说明】中的描述，使用表4-1给出的类的名称，给出图4-2中A~D所对应的类。C、D答案可以互换！！

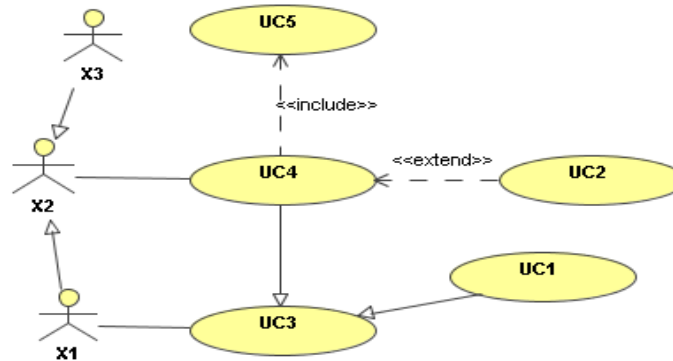
A:CarPark B:Barrier C:EntryBarrier D:ExitBarrier

【问题2】根据【说明】中的描述，使用表4-1给出的状态名称，给出图4-3中S1~S4所对应的状态。

S1 Idle
 S2 Await Ticket Take
 S3 Await Enable
 S4 Await Entry

Part 2

1. 边界对象表示了系统与参与者之间的接口。在每一个用例中，该对象从参与者处收集信息，并将之转换为一种被实体对象和控制对象使用的形式。
2. 面向对象分析侧重于理解问题并描述软件要做什么，而面向对象设计侧重于理解解决方案。
3. RUP的基本特征是“用例驱动、以架构为中心的和受控的迭代式增量开发”。RUP将一个周期的开发过程划分为4个阶段，其中精化阶段的提交结果包含了系统架构。
4. UML中关联的多重度是指一个类的实例能够与另一个类的多个实例相关联
5. 在UML的各种视图中，用例视图显示外部参与者观察到的系统功能；逻辑视图从系统的静态结构和动态行为角度显示系统内部如何实现系统的功能；实现视图显示的是源代码以及实际执行代码的组织结构。在 ROSE 中，时序图和协作图（或通信图）通常建立在逻辑视图下的use case realization包中。
6. 在下面的用例图（UseCase Diagram）中，X1、X2和X3表示参与者，已知UC3是抽象用例，那么X1可通过UC4和UC1用例与系统进行交互。并且，用例UC2是UC4的可选部分，用例UC5是UC4的必须部分。



7. RSA (Rational Software Architect)支持模型驱动的开发MDD (Model-Driven Development), 这种开发模式所基于的MDA技术包含CIM、PIM和PSM三个层次。
8. 在RSA中，透视图（perspective）定义了编辑器（Editor）和视图（View）集合，并针对特定的任务和职能对其提供初始的布局。Resource、Modeling和Debug 都是 RSA所提供的透视图。
9. 在UML 2.0中，Composite Structure Diagram的主要作用是：反映类、接口或构件的内部协作，用于表达运行时的体系结构、使用模式及关系，其中所描述的Assembly connector存在于一个结构类的内部端口（port）之间，而Delegate connector存在于外部结构和一个结构类的内部端口之间。
10. 活动图中可以表示分支与合并、分叉与汇合泳道、对象流等元素，但不能表达关联。
11. Controlled Unit 是可以进行版本控制的模型元素，在ROSE中， Logical View和Use Case View被打包成.cat文件而成为受控单元。

- 1、在RUP中，先启阶段、精化阶段、构建阶段和提交阶段的迭代次数不需要完全一致，尽管它们都经历了业务建模、需求、分析与设计、实现、测试、部署等6个核心过程工作流程和配置与变更管理、项目管理和环境等3个核心支持工作流程。
- 2、状态图中的内部转换和自转换的含义是不一样的。
- 3、ROSE中，在浏览器中删除一个模型元素会使得该元素从所有包含该模型元素的图中被删除。
- 4、RSA所支持的查询图包括浏览图（Browse Diagram）和主题图（Topic Diagram），其中主题图可以通过刷新功能得到修改后的程序代码的快照（snapshot）。
- 5、组件的接口包含两种：一种是为外界提供服务的接口，称为provider interface或导出接口（export interface）；另一种是向外部请求服务的接口，称为required interface或导出接口(import interface)。
- 6、Round-Trip Engineering是ROSE支持的一种开发形式，其中的逆向工程是指将较为具体的表现形式向较为抽象的表现形式转换，如从Java代码自动生成类图就是一种逆向工程过程。
- 7、组合关系(composition)是一种特殊的聚合关系（aggregation），聚合关系具有很强的归属性，而且整体与部分的对象生存周期是必须一致的。
- 8、Use Case Realization 和相应的Use Case之间的关系可以用  来表示。
- 9、顺序图和协作图（UML2.0中的通信图）在语义上是等价的，他们只是从不同的方面来描述一次交互。
- 10、在RUP中，识别设计元素（Identify Design Elements）是精化体系结构（Refine the Architecture）活动中的一个步骤。
- 11、部署图（或配置图）中包含两种节点（Node），一种是设备节点(Device)，另一种是执行环境接点（Execution Environment）。其中设备节点一般是可以嵌套其它节点并具备处理能力的物理计算资源。
- 12、UML结构中的构造块包括物件、关系和图三个部分。

1. RUP中将软件生命周期划分为哪些阶段，每个阶段所完成的工作有哪些。

Rational Unified Process 有四个阶段：

- 先启阶段(Inception)：定义项目范围
- 细化阶段(Elaboration)：规划项目、说明项目新特性，定义系统架构
- 构造阶段(Construction)：具体构建和实现软件产品
- 移交阶段(Transition)：向用户提交软件产品，并完成安装和部署工作

2. UML中包含哪几种图，它们的作用是什么。

- **Class diagrams** show the classes of the system, their interrelationships, and the operations and attributes of the classes. Class diagrams are used for a wide variety of purposes, including both conceptual (domain) modeling and detailed design modeling.
- An **object diagram** shows a complete or partial view of the structure of a modeled system at a specific point in time. This snapshot focuses on a particular set of object instances and attributes, and the links between the instances.
- A **composite structure diagram** reflects the internal collaboration of classes, interfaces, or components to describe a functionality. It is used to express run-time architectures, usage patterns, and relationships that might not be reflected by static diagrams.
- A component is a modular part of a system that hides its implementation behind a set of external interfaces. A **component diagram** shows components of the system and their interrelationships.
- **Deployment diagram** shows the assigning, or mapping of software artifacts to physical nodes during execution.

- **Use case diagrams** model the behavior of a system, and help capture the requirements of the system and describe the high-level functions and scope of a system. They identify the interactions between the system and its actors.
- **Activity diagrams** specify the behavior of the system and is expressed as a flow of execution through the sequencing of subordinate units.
- A **sequence diagram** is an interaction diagram that emphasizes the time ordering of messages.
- A **communication diagram** emphasizes the organization of the objects that participate in an interaction.
- A state machine diagram models dynamic behavior. It specifies the sequence of states in which an object can exist

1. 阅读下列说明和图，回答问题1至问题4，将解答填入答题纸的对应栏内。

【说明】 已知某唱片播放器不仅可以播放唱片，而且可以连接电脑并把电脑中的歌曲刻录到唱片上（同步歌曲）。连接电脑的过程中还可自动完成充电。

关于唱片，还有以下描述信息：

1. 每首歌曲的描述信息包括：歌曲的名字、谱写这首歌曲的艺术家以及演奏这首歌曲的艺术家。只有两首歌曲的这三部分信息完全相同时，才认为它们是同一首歌曲。艺术家可能是一名歌手或一支由2名或2名以上的歌手所组成的乐队。一名歌手可以不属于任何乐队，也可以属于一个或多个乐队。
2. 每张唱片由多条音轨构成；一条音轨中只包含一首歌曲或为空，一首歌曲可分布在多条音轨上；同一首歌曲在一张唱片中最多只能出现一次。
3. 每条音轨都有一个开始位置和持续时间。一张唱片上音轨的次序是非常重要的，因此对于任意一条音轨，播放器需要准确地知道，它的下一条音轨和上一条音轨是什么（如果存在的话）。

根据上述描述，采用面向对象方法对其进行分析与设计，得到了如表1-1所示的类列表、如图1-1所示的初始类图以及如图1-2所示的描述播放器行为的UML状态图。

- **【问题1】**

- 根据说明中的描述，使用表1-1给出的类的名称，给出图1-1中的A~F所对应的类。

- **【问题2】**

根据说明中的描述，给出图1-1中（1）~（6）处的多重度。

- **【问题3】**

图1-1中缺少了一条关联，请指出这条关联两端所对应的类以及每一端的多重度

Track(0..1)-----→Track(0..1)

- **【问题4】**

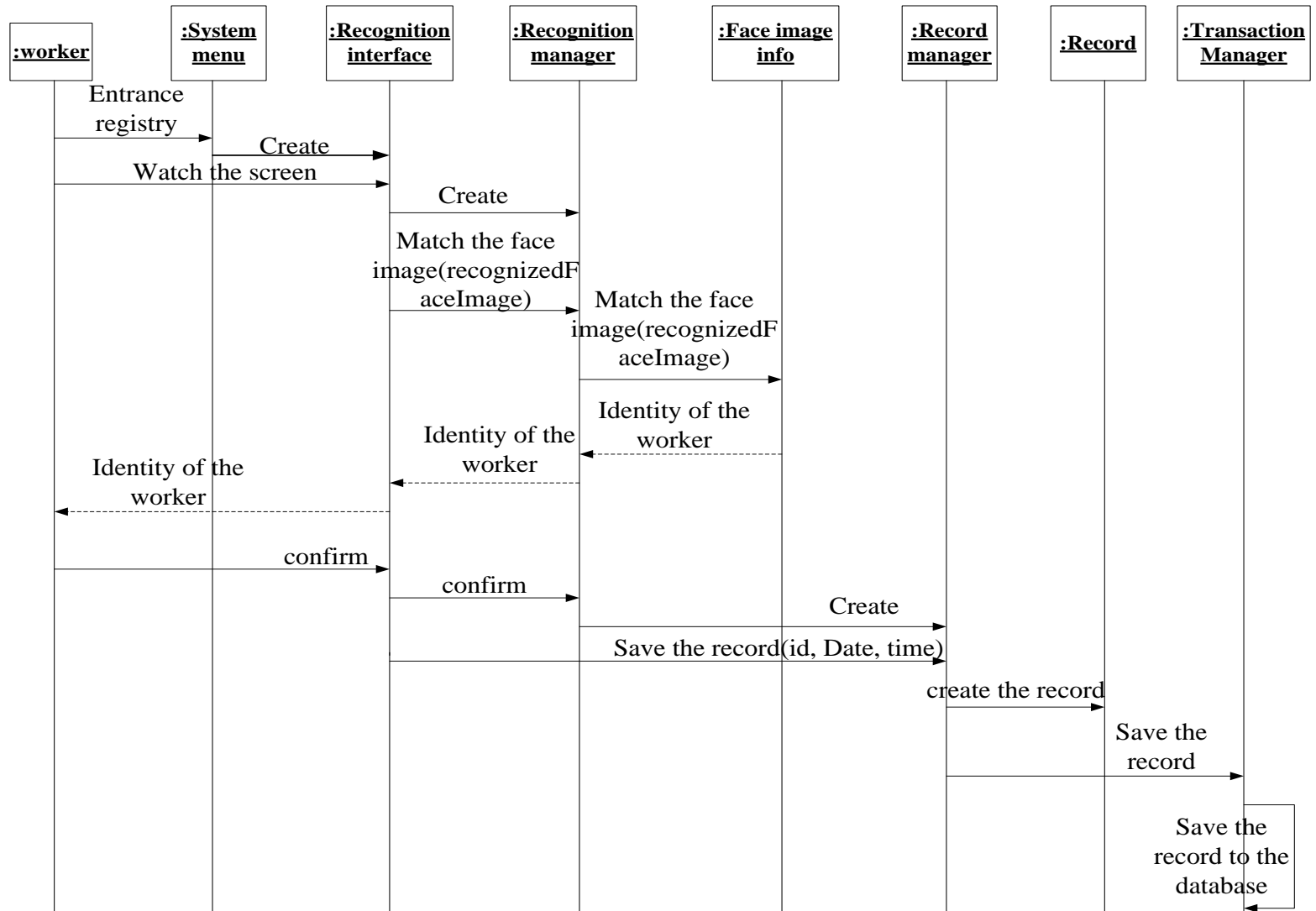
根据图1-2所示的播放器行为UML状态图，给出从“关闭”状态到“播放”状态所经过的最短事件序列（假设电池一开始就是有电的）。

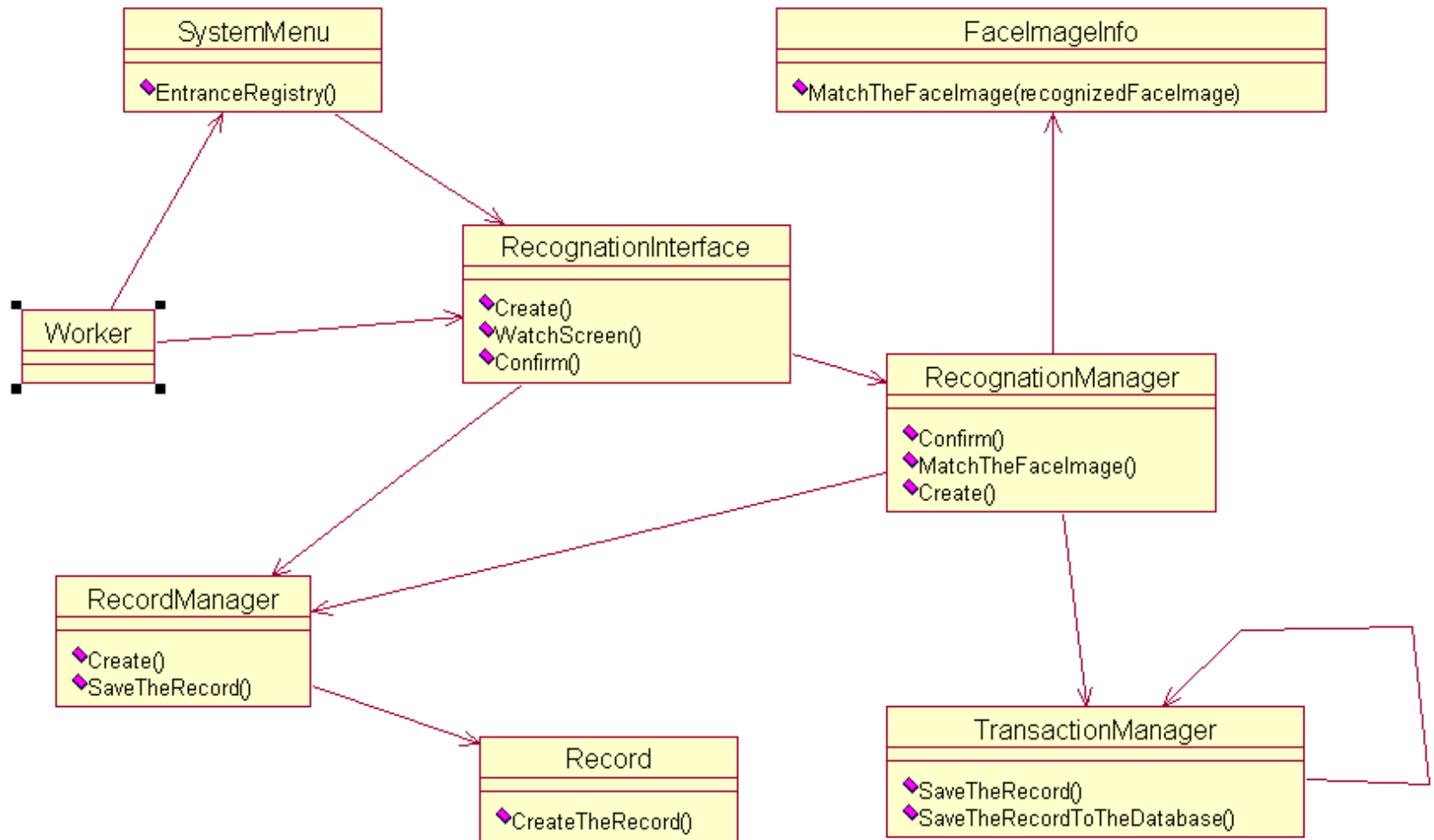
按任意键，选择歌曲

- 员工脸部识别考勤系统是为考察员工出勤情况而开发的，用于对员工的出勤情况进行监控。该系统要对员工的脸部图像信息进行采集，即系统识别员工的脸部图像，并根据员工输入的个人信息（包括姓名和工作号）生成员工图像记录；员工在每天上班和下班时使用该系统进行身份识别，即系统识别员工的脸部图像，然后将识别的图像信息与系统中保存的员工脸部图像信息进行匹配，如果匹配成功则将识别出的员工身份和当前日期、时间等信息保存到员工出勤数据库中；如果匹配不成功，则返回错误信息，员工需再次进行脸部识别；考勤员可以在需要的时候使用该系统生成员工的出勤情况统计分析表。
- 假设在该员工脸部识别考勤系统中，有一个用例名为“上班登记”。此用例允许员工在上班时使用系统识别自己的脸部图像进而识别自己的身份，同时系统可以将登录信息存储在数据库中。该用例的一个基本事件流如下：
 - 员工从系统菜单中选择“上班登记” (worker selects “entrance registry” from the system menu);
 - 系统显示脸部识别窗口; (the system displays the face recognition interface)
 - 员工将脸部对准识别窗口; (worker watches the interface)
 - 系统捕获并识别员工的脸部图像，向员工返回识别的身份信息; (the system captures and recognizes the worker’s face and returns the recognized identity information)
 - 员工选择“确认”按钮; (worker presses the “confirm” button)
 - 系统生成一个关于该员工及当前日期和时间的新记录，并将该记录保存到数据库中。(the system creates a new record about the worker, date and time, and saves this record to the database)
- 【问题1】（5分）
- 请根据上述的事件流识别和标识类，
- 【问题2】（8分）
- 画出UML顺序图和类图。


- 1、识别出来的类有7个：SystemMenu、RecognitionInterface、RecognitionManager、FaceImageInfo、RecordManager、Record、TransactionManager

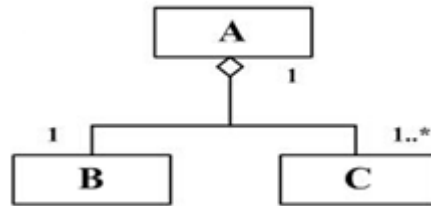
2、





Part 3

1. UML文档描述了面向对象分析与设计的结果。
2. 在面向对象分析与设计中，实体类是应用领域中的核心类，一般用于保存系统中的信息以及提供针对这些信息的相关处理行为；这种类的构造型通常用  来表示。
3. 已知3个类A、B和C，其中类A由类B的一个实例和类C的一个或多个实例构成。用UML类图表示类A、B和C之间的关系如下图。

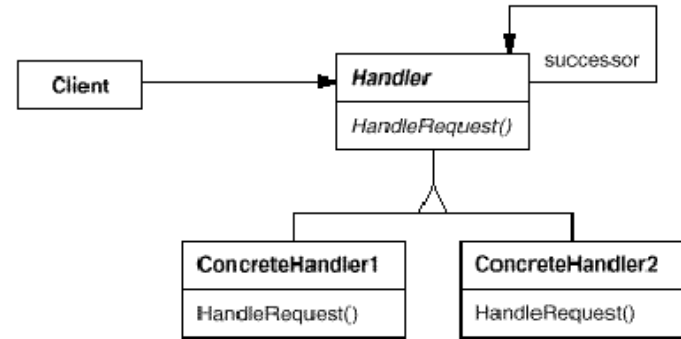


4. UML的设计视图包含了类、接口和协作，其中，设计视图的静态方面由类图 and 对象图表现；其中的类图可以明确表示两类事物之间存在的整体/部分形式的关联关系。动态方面由交互图、状态图和活动图表现。
5. RUP将一个周期的开发过程划分为4个阶段，其中构建阶段的提交结果包含了完整的用例和设计模型，以及在适当平台上集成的软件产品。
6. 在进行面向对象设计时，采用设计模式能够复用相似问题的相同解决方案。工厂方法（**Factory Method**）设计模式定义一个用于创建对象的接口，让子类决定实例化哪一个子类，它使一个类的实例化延迟到其子类。组合（**Composite**）模式将对象组合成树形结构以表示“部分-整体”的层次结构，并使得用户对单个对象和组合对象的使用具有一致性。
7. 包是用于把元素组织成组的通用机制。该机制在开发过程中有很多作用，包括有利于并行开发和提高软件生产率、有利于有效的软件配置管理、有利于提高软件的模块独立性和软件可理解性等。
8. 在RSA中，透视图定义了编辑器（**Editor**）和视图（**View**）集合，并针对特定的任务和职能对其提供初始的布局。但是，**Refactoring**不是RSA所提供的一种透视图。

9. 在UML 2.0中， composite structure diagram反映类、接口或构件的内部协作，用于表达运行时的体系结构、使用模式及关系，其中所描述的 **Assembly connector**存在于一个结构类的内部端口（port）之间，而**Delegate connector** 存在于外部结构和一个结构类的内部端口之间。
10. Rational Software Architect 通过Compare and Merge功能来发现模型版本的变动、支持并行开发和探索多种可能的解决方案。

11. 右图所表示的设计方案可以：

- 1)动态决定由一组对象中某个对象处理该请求；
- 2)使多个对象都有机会处理请求，避免请求的发送者和接收者间的耦合关系；
- 3)将对象连成一条链，并沿着该链传递请求；

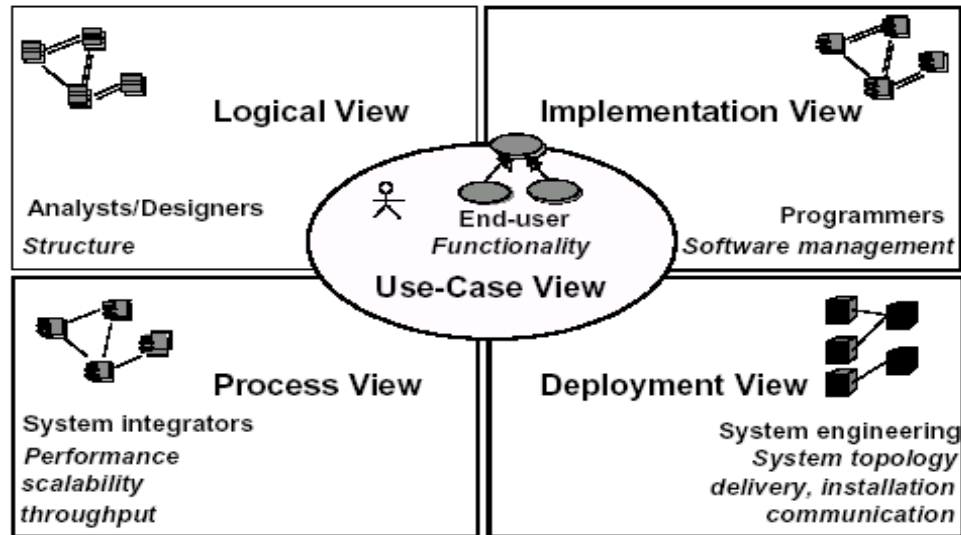


12. 在UML中，约束信息用{ } 符号表示。
13. 状态图可以表现一个对象在生存期的行为，所经历的状态序列，引起状态转移的时间以及因状态转移而引起的动作。
14. Round-Trip Engineering能够帮助维持软件架构的完整性，Round-trip Engineering的好处包括发现和评估软件架构上的改动、传达被接受的架构改动、在每次迭代中保持模型和代码的同步等。
15. 建模的基本原则包括：要仔细的选择模型，因为模型关系着系统解决方案的构造；每一种模型可以在不同的精度级别上表示所要开发的系统；模型要与现实相联系等。
16. UML中有的四种基本关系是：依赖，泛化，关联和实现。

1. 在活动图和状态图中，有且只有一个初始节点，但可以没有终止节点，也可以有多个终止节点。
2. 顺序图和协作图（UML2.0中的通信图）在语义上是等价的，通信图用链接（link）刻画对象间的拓扑关系，顺序图则可以描述执行的发生（execution occurrence）或控制焦点（focus of control）。
3. RSA所支持的查询图包括浏览图（Browse Diagram）和主题图（Topic Diagram），这两种图并非是UML2.0中新增的两种图，而是RSA用以可视化程序代码的不同方式。
4. 在RUP中，先启阶段、精化阶段、构建阶段和提交阶段往往需要多次迭代，但是如果不需要探索系统可行性或项目范围很容易确定，则先启阶段中可以不进行真正意义的迭代。
5. 模型和图在概念上不同，一种UML图可以用于多个模型中，一个模型也可以用多种图来描述。
6. UML中的构件遵从一组接口并提供一组接口的实现，与类不同，构件往往是系统的物理模块。
7. 在状态图中，历史状态是基于组合状态构造的，也就是说如果没有组合状态的图形语法和语义，就无法进一步定义历史状态。
8. Controlled Unit是可以进行版本控制的模型元素，在ROSE中，模型文件本身被打包存储为.mdl文件从而成为受控单元，Component View则被打包成.sub文件而成为受控单元。
9. UML不仅适用于用例驱动的开发过程，也适合在以体系结构为中心的开发过程中使用。
10. Use Case Realization明确了分析与设计（analysis and design）活动和需求(requirement)活动之间的可追踪性（traceability）。每一个use case realization中都可以包括若干实现某个用例的类图、顺序图和通信图。
11. 部署图（或配置图）中包含两种节点（Node），一种是设备节点(Device)，另一种是执行环境接点（Execution Environment）。其中设备节点一般是可以嵌套其它节点并具备处理能力的物理计算资源。
12. 在RUP中，识别设计元素（Identify Design Element）是精化体系结构（Refine the Architecture）活动中的一个步骤。

13. 泛化关系可以通过” is a kind of” 短语来判定，而聚合关系（aggregation）可以通过” is a part of” 短语来判定。
14. 在用例建模中，用例强调的是完整性，而**Scenario**（场景）强调的是可理解性。
15. UML结构中的构造块包括物件、关系和图三个部分。

1. Philippe Kruchten在《IEEE Software》上发表了题为《The 4+1 View Model of Architecture》的论文，引起了业界的极大关注，并最终被 RUP 采纳，现在已经成为架构设计的标准。列出5种视图的名称并对它们的内容给出解释。



2. UML中包含哪几种图？它们的作用是什么？

Class diagram \ Object diagram \ Component diagram \ Composite Structure diagram \ deployment diagram

Use Case diagram \ Sequence diagram \ Communication diagram \ State machine diagram \ Activity diagram

3. 简述RUP的6个核心过程工作流和3个支撑工作流。

Business Modeling \ Requirement \ Analysis and Design \ Implementation \ Test \ Deployment
Configuration Management \ Project Management \ Environment

4. 用例规约（Use case specification）是面向对象分析与设计的基础，举例说明用例规约大体应包含哪些内容。

Use case name \ ID \ Brief description \ Primary actors \ Secondly actors \ Pre-condition \ Basic Flow (Main Flow) \ Post Condition \ Alternative flow

5. 举例说明UML的三种扩展机制。

Constraint \ Tag Value \ Stereotype

6. 说明UML中的关联、泛化、实现、依赖四种关系各自的含义，并区分聚合（Aggregation）和组合（Composition）两种关系的不同。

关联表示两个类之间存在某种语义上的联系；

泛化关系描述了一般事物和该事物中的特殊种类之间的关系；

实现关系是用来规定接口与实现接口的类或组件之间的关系；

两个元素X、Y，如果修改元素X的定义可能会引起另一个元素Y的定义的修改，则称元素Y依赖于元素X；

组合和聚集都表示实例之间的整体/部分关系，组合是聚集的一种形式，聚集是概念性的，只是区分整体与部分。组合具有很强的归属关系，而且整体与部分的对象生存周期是一致的。

7. 在识别类、类的职责以及类的关系的过程中，实践中存在两种观点：一种方法是通过筛选名词动词等方式来进行，另一种是通过观察由用例事件流获得的顺序图来进行。你同意哪一种做法，为什么？

各有优缺点：第一种方法适合开发有一定开发经验的领域；第二种方法适合陌生的领域，缺乏相关系统的开发经验

1. 阅读下列说明和图，完成问题1至问题4。

【说明】在线会议审稿系统（Online Reviewing System, ORS）主要处理会议前期的投稿和审稿事务，其功能描述如下：

（1）用户在初始使用系统时，必须在系统中注册（register）成为作者或审稿人。

（2）作者登录（login）后提交稿件和浏览稿件审阅结果。提交稿件必须在规定提交时间范围内，其过程为先输入标题和摘要，选择稿件所属主题类型，选择稿件所在位置（存储位置）。上述几步若未完成，则重复；若完成，则上传稿件至数据库中，系统发送通知。

（3）审稿人登录后可设置兴趣领域，审阅稿件给出意见，以及罗列录用和（或）拒绝的稿件。

（4）会议委员会主席是一个特殊的审稿人，可以浏览提交的稿件、给审稿人分配稿件、罗列录用和（或）拒绝的稿件，以及关闭审稿过程。其中关闭审稿过程须包括罗列录用和（或）拒绝的稿件。

系统采用面向对象的方法开发，使用UML进行建模。在建模用例图时，常用的方式是先识别参与者，然后确定参与者如何使用系统来确定用例，每个用例可以构造一个活动图。参与者名称、用例和活动名称分别参见表1、表2和表3。系统的部分用例图和提交稿件的活动图分别如图1和图2所示。

表 1 参与者列表

名称	说明	名称	说明
User	用户	Author	作者
Reviewer	审稿人	PCCChair	委员会主席

表 2 用例名称列表

名称	说明	名称	说明
login	登录系统	Register	注册
submit paper	提交稿件	Browse review results	浏览稿件审阅结果
close reviewing process	关闭审稿过程	assign paper to reviewer	分配稿件给审稿人
set preferences	设定兴趣领域	enter review	审阅稿件给出意见
list accepted/rejected papers	罗列录用和/或拒绝的稿件	browse submitted papers	浏览提交的稿件

表 3 活动名称列表

名称	说明	名称	说明
select paper location	选择稿件位置	upload paper	上传稿件
select subject group	选择主题类型	send notification	发送通知
enter title and abstract	输入标题和摘要		

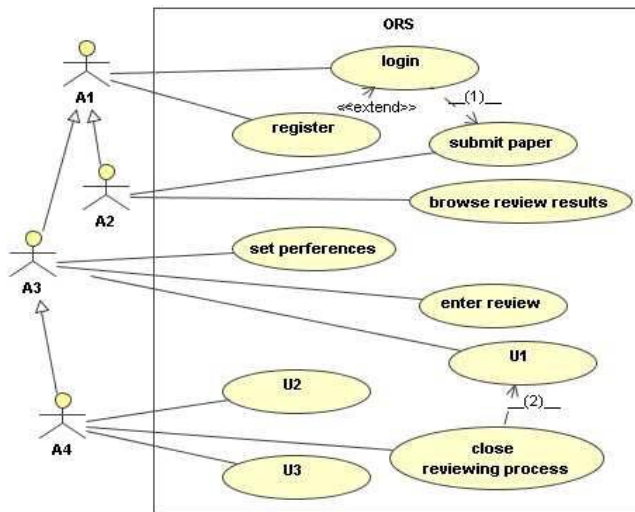


图1 ORS的用例图

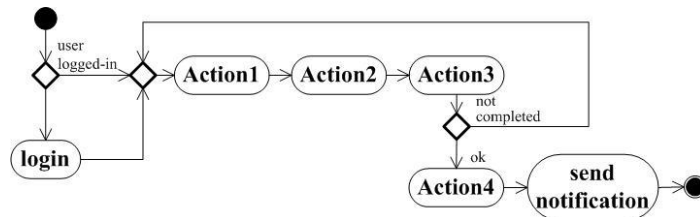


图2 提交稿件过程的活动图

【问题1】根据【说明】中的描述，使用表1中的英文名称，给出图1中A1~A4所对应的参与者。

A1 User , A2 Author, A3 Reviewer, A4 PCChair

【问题2】根据【说明】中的描述，使用表2中的英文名称，给出图1中U1~U3所对应的用例。！！注意：U2和U3的答案可以互换。！！

U1 list accepted / rejected papers

U2 browse submitted papers

U3 assign paper to reviewer

【问题3】根据【说明】中的描述，给出图1中（1）和（2）所对应的关系及其含义。

（1）<<extend>>：将常规动作放在一个基本Use Case中，将非常规动作放在其扩展Use Case中。

（2）<<include>>：两个Use Case，如果其中一个在其事件流中包含了另一个，那么它们间就有包含关系。

【问题4】根据【说明】中的描述，使用表2和表3中的英文名称，给出图2中Action1~Action4对应的活动。

Action1 enter title and abstract

Action2 select subject group

Action3 select paper location

Action4 upload paper

2. 阅读下列说明和图，回答问题1至问题3。

【说明】某图书管理系统的主要功能如下：

- 1. 图书管理系统的资源目录中记录着所有可供读者借阅的资源，每项资源都有一个唯一的索引号。系统需登记每项资源的名称、出版时间和资源状态（可借阅或已借出）。
- 2. 资源可以分为两类：图书和唱片。对于图书，系统还需要登记作者和页数；对于唱片，还需登记演唱者和介质类型（CD或者磁带）。
- 3. 读者信息保存在图书管理系统的读者信息数据库中，记录的信息包括：读者的识别码和读者姓名。系统为每个读者创建了一个借书记录文件，用来保存读者所借资源的相关信息。

现采用面向对象方法开发该图书管理系统。识别类是面向对象分析的第一步。比较常用的识别类的方法是寻找问题描述中的名词，再根据相关规则从这些名词中删除不可能成为类的名词，最终得到构成该系统的类。表1给出了说明中出现的所有名词。

表 1

图书管理系统	资源目录	读者	资源
索引号	系统	名称	出版时间
资源状态	图书	唱片	作者
页数	演唱者	介质类型	CD
磁带	读者信息	读者信息数据库	识别码
姓名	借书记录文件	信息	

通过对表1中的名词进行分析，最终得到了图1所示的UML类图（类的说明如表2所示）。

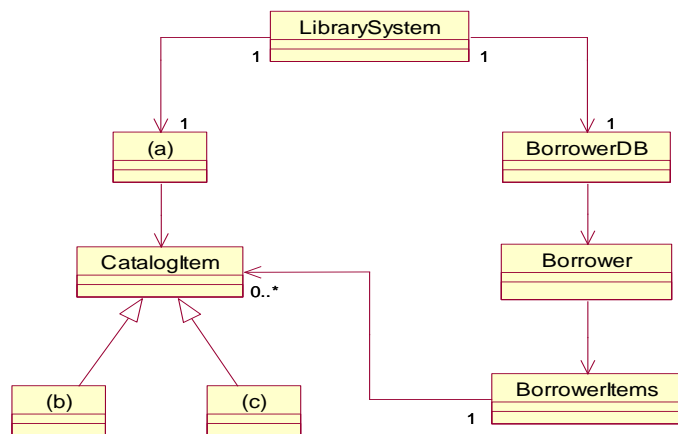


图1

表2

类名	说明
<u>LibrarySystem</u>	图书管理系统
<u>BorrowerDB</u>	保存读者信息的数据库
<u>CatalogItem</u>	资源目录中保存的每项资源
<u>Borrower</u>	读者
<u>BorrowerItems</u>	为每个读者创建的借书记录文件

【问题 1】表2所给出的类并不完整，根据说明和表1，将图1中的(a)-(c)处补充完整。

(a) 资源目录 (b) 图书 (c) 唱片

注：(b) 和 (c) 的答案可以互换

【问题 2】根据【说明】中的描述，给出图1中的类catalogItem以及(b)、(c)处所对应的类的关键属性（使用表1中给出的词汇），其中，CatalogItem有4个关键属性；(b)、(c)处对应的类各有两个关键属性。

CatalogItem的属性：索引号、名称、出版时间、资源状态

图书的属性：作者、页数

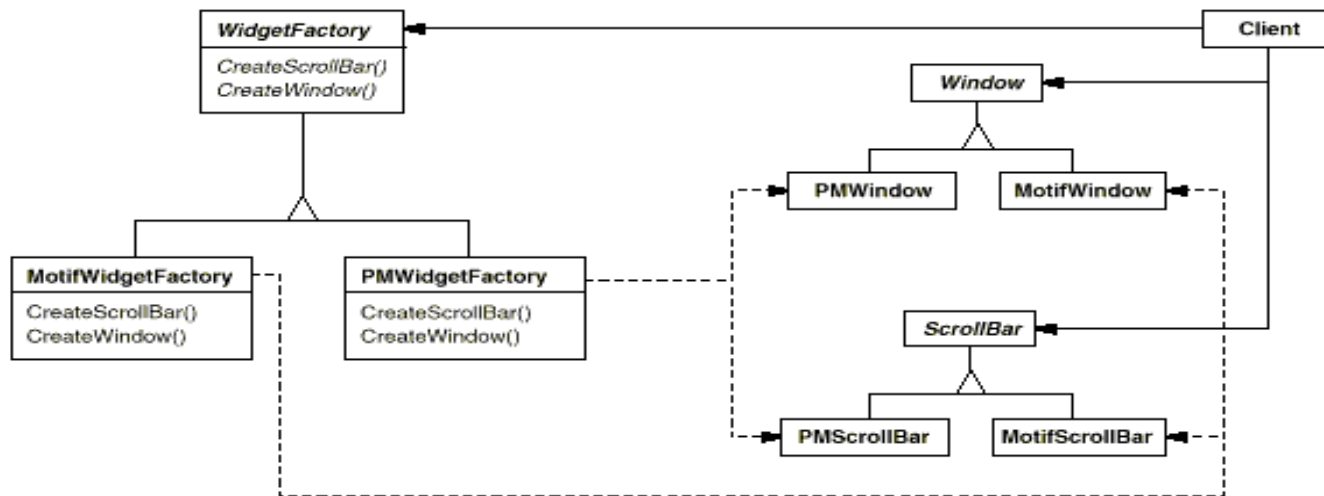
唱片的属性：演唱者、介质类型

【问题 3】识别关联的多重度是面向对象建模过程中的一个重要步骤。根据说明中给出的描述，完成图1中的(1)-(6)。

(1) 1 (2) 0..* (3) 1 (4) 0..* (5) 1 (6) 1或者0..1

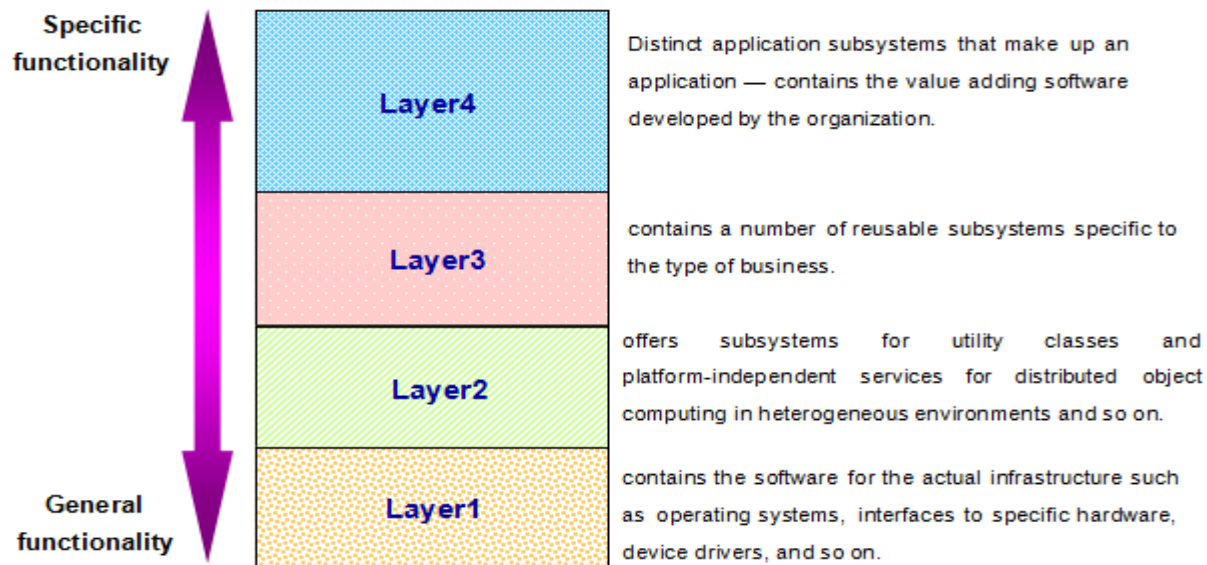
Part 4


1. UML类图中类与类之间的关系有5种：依赖、关联、聚合、组合与继承。若类A仅在其方法Method1中定义并使用了类B的一个对象，类A其他部分的代码都不涉及类B，那么类A和类B的关系应为依赖关系；若类A中包含了其他类的实例，且当类A的实例消失时，其包含的其它类的实例也消失，则类A和它所包含的类之间存在组合关系；若类A的实例消失时，其它类的实例仍然存在并继续工作，那么类A和它所包含的类之间存在聚合关系。
2. 如下UML类图所示，该设计可以提供创建一系列相关或相互依赖的对象的接口，而无需指定这些对象所属的具体类；另外，该设计可应用于一个系统要由多个产品系列中的一个来配置或者强调一系列产品对象的设计以便进行联合使用的时候。



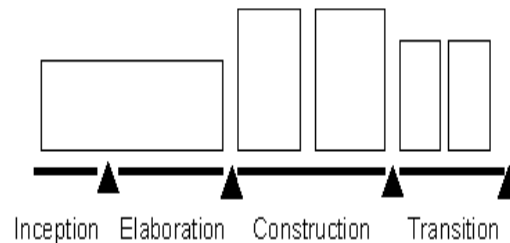
3. 采用UML进行面向对象开发时，部署图通常在实施阶段使用。
4. 用例模型（The Use-Case Model）、术语表（Glossary）和补充规约（Supplementary Specifications）均属于需求工件，而用例实现（Use-case Realization）不是需求工件（Requirement artifacts）之一。

5. 下图显示了一个典型的软件分层架构，Layer1至Layer4依次为：System software, Middleware, Business-specific, Application-specific。



6. 在面向对象分析与设计中，边界对象表示了系统与参与者之间的接口。在每一个用例中，该对象从参与者处收集信息，并将之转换为一种被实体对象和控制对象使用的形式。该对象的构造型通常用  来表示。
7. 用例分析（Use-Case Analysis）的目的包括识别参与到用例事件流执行的类，使用用例实现（Use-Case Realization）将用例行为分配到类中，识别类的责任、属性和类见的关联，并标注体系结构机制的使用等。它是由软件设计人员负责完成的工作，用例分析产生的工件（Artifact）有用例实现、分析模型和分析类。
8. 包(package)是UML的分组事物；UML的结构事物包括类、接口、协作、用例、构件和节点等，其中接口用于说明类或构件的某种服务的操作集合。

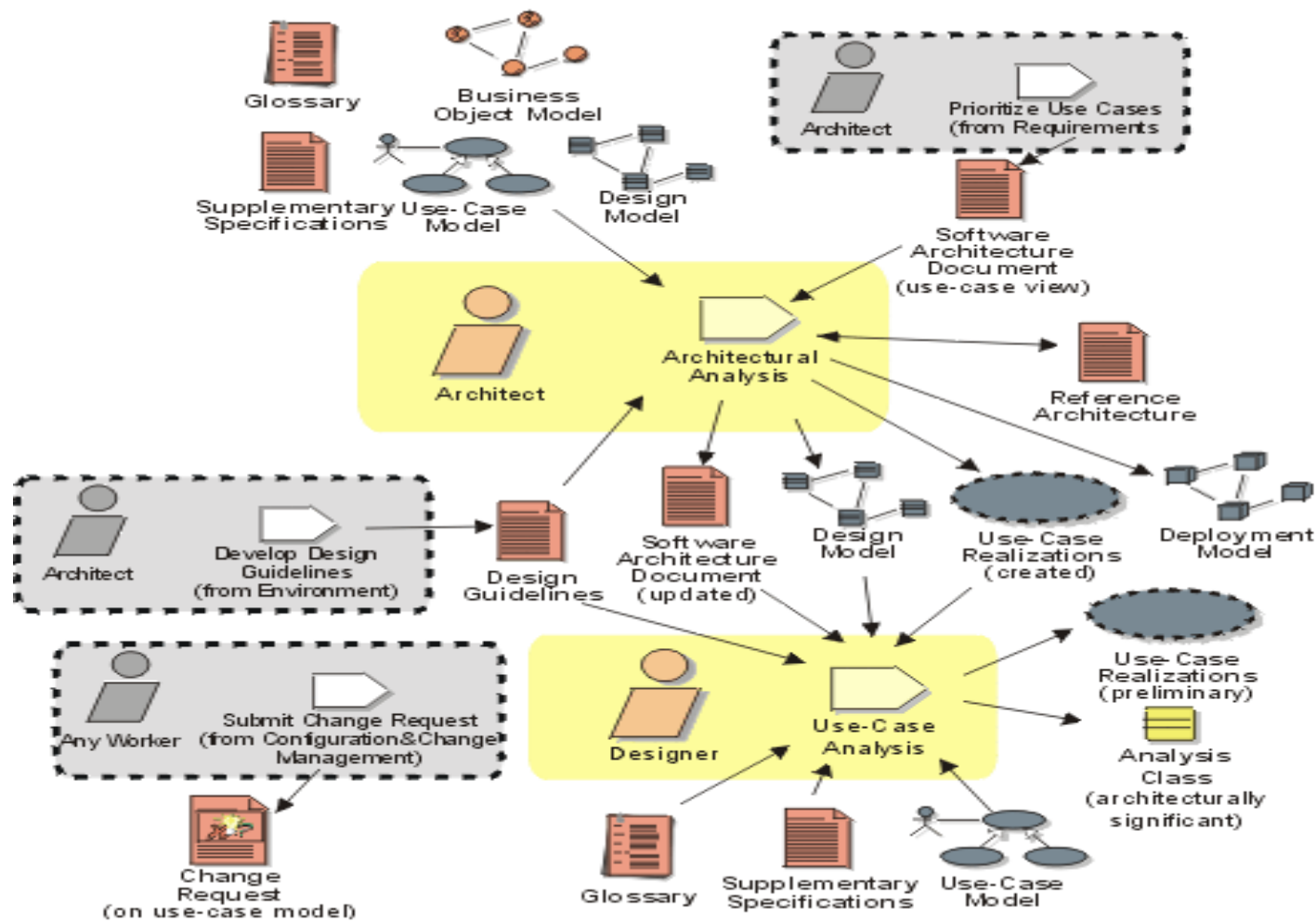
9. 在面向对象软件开发过程中，采用设计模式以复用成功的设计。
10. 在较高的抽象层次上，传统的程序流程图与UML中的活动图最根本的区别在于程序流程图明确指定了每个活动的先后顺序，而活动图仅描述活动和必要的工作顺序。
11. UML的设计视图包含类、接口和协作，其中，设计视图的动态方面由交互图 and 状态图表现。
12. 如果要表示待开发软件系统中软件组件和硬件组件之间的物理关系，通常采用UML中的部署图。
13. RSA(Rational Software Architect)通过将模型分块（**Model Partitioning**）来支持面向团队的并行开发，它可以将模型元素和某些图转换成片段（**Fragment**），从而形成独立的后缀名为`.efx`的文件。然而，不是所有的图都可以定义成片段，以下选项中，不可以被封装成`.efx`文件的是类图。
14. 如果项目的可行性无需研究、系统的架构已经存在、系统需求稳定并易于理解，那么可用如下图准确表示RUP迭代开发生命周期。



15. 主题图和浏览图都属于查询图，浏览图仅用于研究和探索，并不作为模型的一部分或项目资源而存在，主题图捕获模型或代码状态的快照（**snapshot**），它是项目资源的一个组成部分。
16. 体系结构分析（**Architecture Analysis**）的目的包括：1)基于从相似问题域和相似系统获取的经验，为系统定义候选的架构；2)为系统定义体系结构模式和关键机制；3)定义复用策略等。它是由软件架构师负责完成的工作，体系结构分析产生的工件（**Artifact**）包括软件架构文档、设计模型、部署模型，体系结构分析通常每个项目只在细化阶段（**Elaboration Phase**）做一次。

1. 在部署图（**Deployment Diagram**）中有两类节点（**Node**），**Device**节点和**Execution Environment**节点，因为**Device**节点具有物理计算资源和处理能力，所以它可以包含**Execution Environment**节点。
2. 包括**RSA(Rational Software Architect)**工具在内**Rational Software Delivery Platform**是一套基于**Eclipse**的软件开发解决方案。
3. **VOB**是指版本化对象库（**Versioned Object Base**），建模时所产生的模型元素和图都可以通过**VOB**进行配置管理。
4. 状态图中的自转换不同于内部转换，会导致出口动作和入口动作的执行。
5. **UML**适合在用例驱动的、以体系结构为中心的、迭代和渐增的过程模型中应用。
6. 在**RSA**中，透视图（**Perspective**）定义了编辑器（**Editor**）和视图（**View**）集合，并针对特定的任务和职能对其提供初始的布局。**Resource**、**Modeling**、**Java**和**Debug**都是**RSA**提供的透视图。
7. 在组合结构图（**composite structure diagram**）中，每一个接口都是由多个端口（**port**）构成的。端口的连接分为两类：委托连接（**delegate connector**）和组装端口（**assembly connector**）。
8. 对象图展示了所建模的系统在某个特定时间点的一个完整的或部分的结构视图，这个快照关注类的实例、实例的属性以及实例间的链接。
9. 用例图中的包含关系(**include**)和扩展关系(**extend**)都是通过构造型（**stereotype**）对**UML**中的依赖关系的一种扩展。
10. 组合关系（**composition**）是一种特殊的聚合关系（**aggregation**），它在聚合关系的基本语义上增加了新的约束。
11. 体系结构机制(**Architectural mechanism**)是系统在通用的标准、原则、和实践方面的决策，分为三类：分析机制、设计机制和实现机制。
12. 一个模型可以包括多种图，如系统的逻辑模型中可以包括类图、顺序图和通信图等；一种图也可以被多个模型所包含，如类图既可以出现在逻辑模型中，也可以出现在进程视图中。
13. 用**UML**的顺序图表达对象的交互时，不仅可以表示同步消息、异步消息，还可以表示消息发送的先后次序和消息的循环。
14. 泛化关系（**generalization**）可以用在类元之间，无论是类、用例还是组件。
15. 在**ROSE**工具中，任何情况下，从图形窗口中删除一个模型元素时，如果该元素没有名称并仅在此图形窗口中出现，那么该元素也会从浏览器窗口中消失。

1. 说明软件设计师和架构师在工作内容和工作产品方面的区别。



2. 什么是MDA，说明它所包含的三个层次并给出解释。

An approach to using models in software development: It separates the specification of the operation of a system from the details of the way that system uses the capabilities of its platform.

(1)Computational Independent Model (CIM).

Focus is on environment of the system and requirements for the system

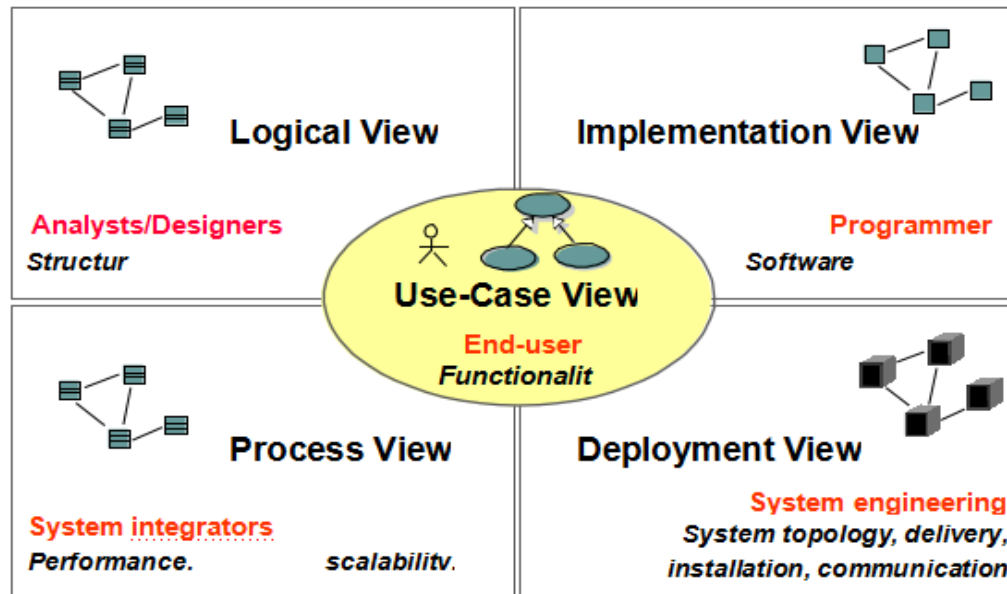
(2)Platform Independent Model (PIM).

Focus is on system operation, independent of platform

(3)Platform Specific Model (PSM).

Focus is on detailed usage of system on specific platform

3. 4+1View Model已成为架构设计的标准，说明这五种视图的名称及其对应的涉众（Stakeholder）。



4. 什么是用例实现（Use-case Realization），用例实现结构中应该包含哪些图。

The use-case realization structure helps to organize the model elements needed to realize the use cases in the design model. The use-case realization structure includes the following elements: Use-case realization packages; Traceabilities diagrams; Interaction diagrams; Class diagrams.

5. 说明识别三种分析类（boundary class, control class, entity class）的基本策略。

(1)Boundary class: One boundary class per actor/use case pair; User Interface Classes--Concentrate on what information is presented to the user, Do NOT concentrate on the UI details; System and Device Interface Classes--Concentrate on what protocols must be defined and Do NOT concentrate on how the protocols will be implemented

(2)Control class: In general, identify one control class per use case, as analysis continues, a complex use case's control class may evolve into more than one class.

(3)Entity class: Use use-case flow of events as input; Key abstractions of the use case; Traditional, filtering nouns approach--Underline noun clauses in the use-case flow of events\Remove redundant candidates\Remove vague candidates\Remove actors (out of scope)\Remove implementation constructs\Remove attributes (save for later)\Remove operations.

6. 什么是分析机制，列举任意两种分析机制，并说明它们的特性。

Analysis mechanisms capture the key aspects of a solution in a way that is implementation-independent. They either provide specific behaviors to a domain-related class or component, or correspond to the implementation of cooperation between classes and/or components. They may be implemented as a framework. Examples include mechanisms to handle persistence, inter-process communication, error or fault handling, notification, and messaging, to name a few.

(1)Persistency mechanism:Granularity,Volume,Duration,Access mechanism,Access frequency (creation/deletion, update, read),Reliability.

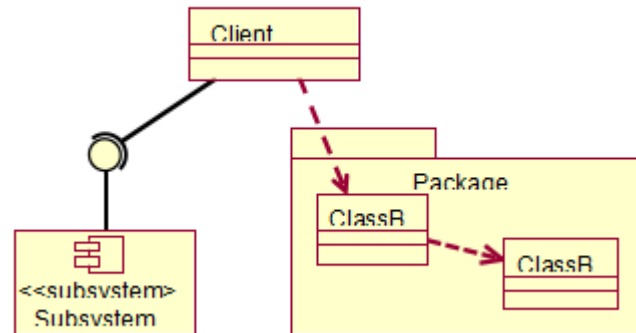
(2)Inter-process Communication mechanism:Latency,Synchronicity,Message size,Protocol.

7. 说明包、组件和子系统的区别和相似之处。

组件是系统中遵从一组接口且提供实现的一个物理部件，通常指开发和运行时类的物理实现。它具有很广泛的定义，以下的一些内容都可以被认为是组件：程序源代码、子系统、动态链接库等。

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.



8. 说明体系结构分析(Architecture Analysis)和用例分析(Use case Analysis)的步骤

Architecture Analysis:

- Identify Key Concepts
- Define the High-Level Organization of the model
- Identify Analysis mechanisms
- Identify Key Abstractions
- Create Use-Case Realizations
- Checkpoints

Use case Analysis:

- Supplement the Use-Case Description
- For each Use-Case Realization
- Find Classes from Use-Case Behavior
- Distribute Use-Case Behavior to Classes
- For each resulting analysis class
- Describe Responsibilities
- Describe Attributes and Associations
- Qualify Analysis Mechanisms
- Unify Analysis Classes
- Checkpoints

1. 阅读下列说明和图，回答问题1至问题3。

【说明】某城市的各国家公园周边建造了许多供游客租用的小木屋和营地，为此，该城市设置了一个中心售票处和若干个区域售票处。游客若想租用小木屋或营地，必须前往中心售票处进行预定并用现金支付全额费用。所有的预定操作全部由售票处的工作人员手工完成。现欲开发一信息系统，实现小木屋和营地的预定及管理功能，以取代手工操作。该系统的主要功能描述如下：

1. 管理预定申请。游客可以前往任何一个售票处提出预定申请。系统对来自各个售票处的预定申请进行统一管理。
 2. 预定。预定操作包含登记游客预定信息、计算租用费用、付费等步骤。
 3. 支付费用。游客付费时可以选择现金和信用卡付款两种方式。使用信用卡支付可以享受3%的折扣，现金支付没有折扣。
 4. 游客取消预定。预定成功之后，游客可以在任何时间取消预定，但需支付赔偿金，剩余部分则退还给游客。赔偿金的计算规则是，在预定入住时间之前的48小时内取消，支付租赁费用10%的赔偿金；在预定入住时间之后取消，则支付租赁费用50%的赔偿金。
 5. 自动取消预定。如果遇到恶劣天气（如暴雨、山洪等），系统会自动取消所有的预定，发布取消预定消息，全额退款。
 6. 信息查询。售票处工作人员查询小木屋和营地的预定情况和使用情况，以判断是否能够批准游客的预定申请。
- 现采用面向对象方法开发上述系统，得到如表1所示的用例列表和表2所示的类列表。对应的用例图和类图分别如图1和图2所示。

【问题1】根据说明中的描述与表1，给出图1中UC1-UC6所处对应的用例名称。

UC1: CheckAvailability UC2: MakeReservation UC3: GetDiscount
UC4: ManageCasePayment UC5: ManageCrCardPayment UC6: CalculateRefund (5、6可互换)

【问题2】根据说明中的描述与表2，给出图2中C1-C7处所对应的类名。

C1: NationalPark C2: Rate C3: TicketOfficer C4: Payment C5: Discount
C6: CashPayment C7: CreditCardPayment (6、7可互换)

【问题3】对于某些需求量非常大的小木屋或营地，说明中功能4的赔偿金计算规则，不足以弥补取消预定所带来的损失。如果要根据预定的时段以及所预定场地的需求量，设计不同层次的赔偿金计算规则，需要对图2进行怎样的修改？
(请用文字说明)

增加一个新类，并使该类与类ReservationItem之间有关联。

表 1 用例列表

用例名	说明	用例名	说明
<u>ManageInquiries</u>	管理预定申请	<u>ManageCashPayment</u>	现金支付
<u>MakeReservation</u>	预定	<u>ManageCrCardPayment</u>	信用卡支付
<u>ManagePayment</u>	支付管理	<u>GetDiscount</u>	计算付款折扣
<u>CancelReservation</u>	游客取消预定	<u>AutoCancelReservation</u>	系统自动取消预定
<u>CheckAvailability</u>	信息查询	<u>CalculateRefund</u>	计算取消预定的赔偿金
<u>PublishMessage</u>	发布取消预定消息		

表 2 类列表

类名	说明	类名	说明
<u>NationalPark</u>	国家公园	Customer	游客
Reservation	预定申请	<u>ReservationItem</u>	预订申请内容
<u>TicketingOfficer</u>	售票处	<u>CampSite</u>	营地
Bungalow	小木屋	Payment	付款
Discount	付款折扣	<u>CashPayment</u>	现金支付
<u>CreditCardPayment</u>	信用卡支付	Rate	租赁费用

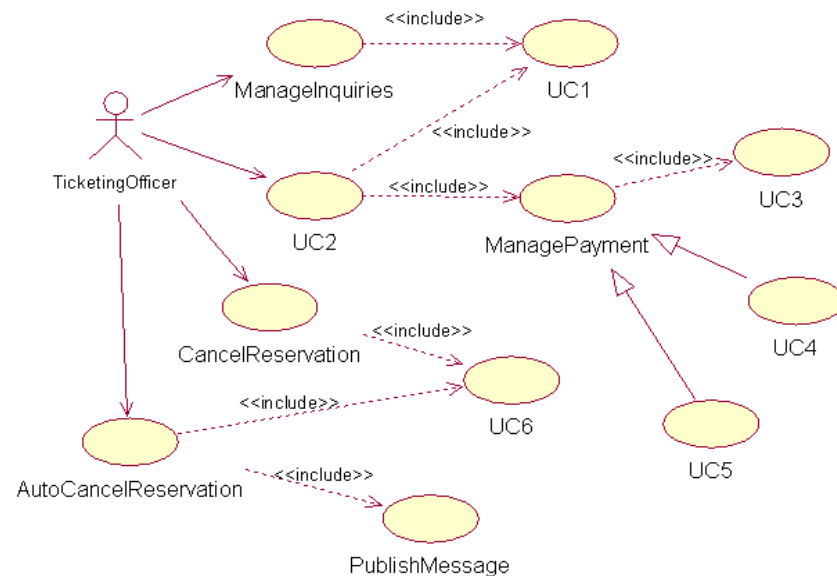


图1 用例图

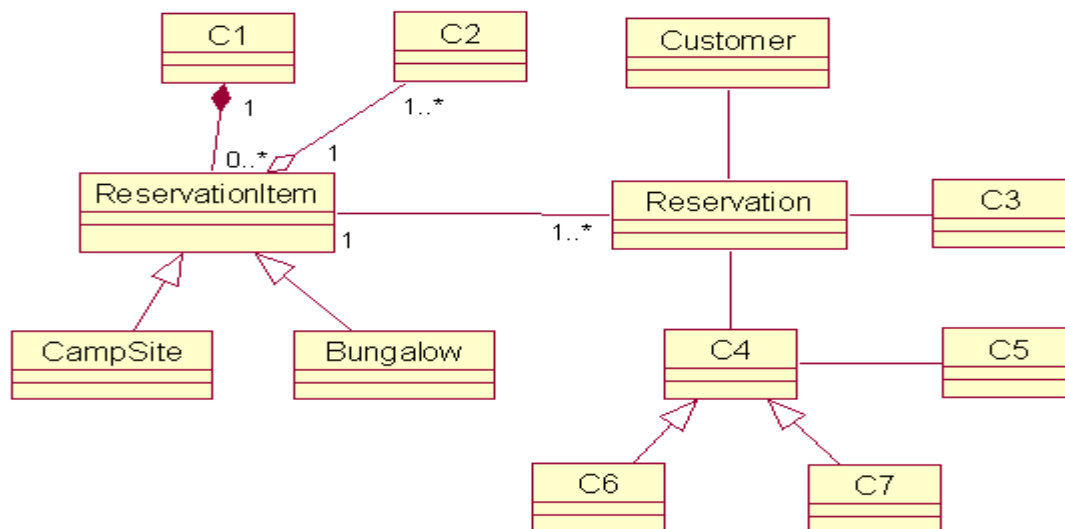


图2 类图

2. 根据图3中的顺序图（Sequence diagram）回答问题。

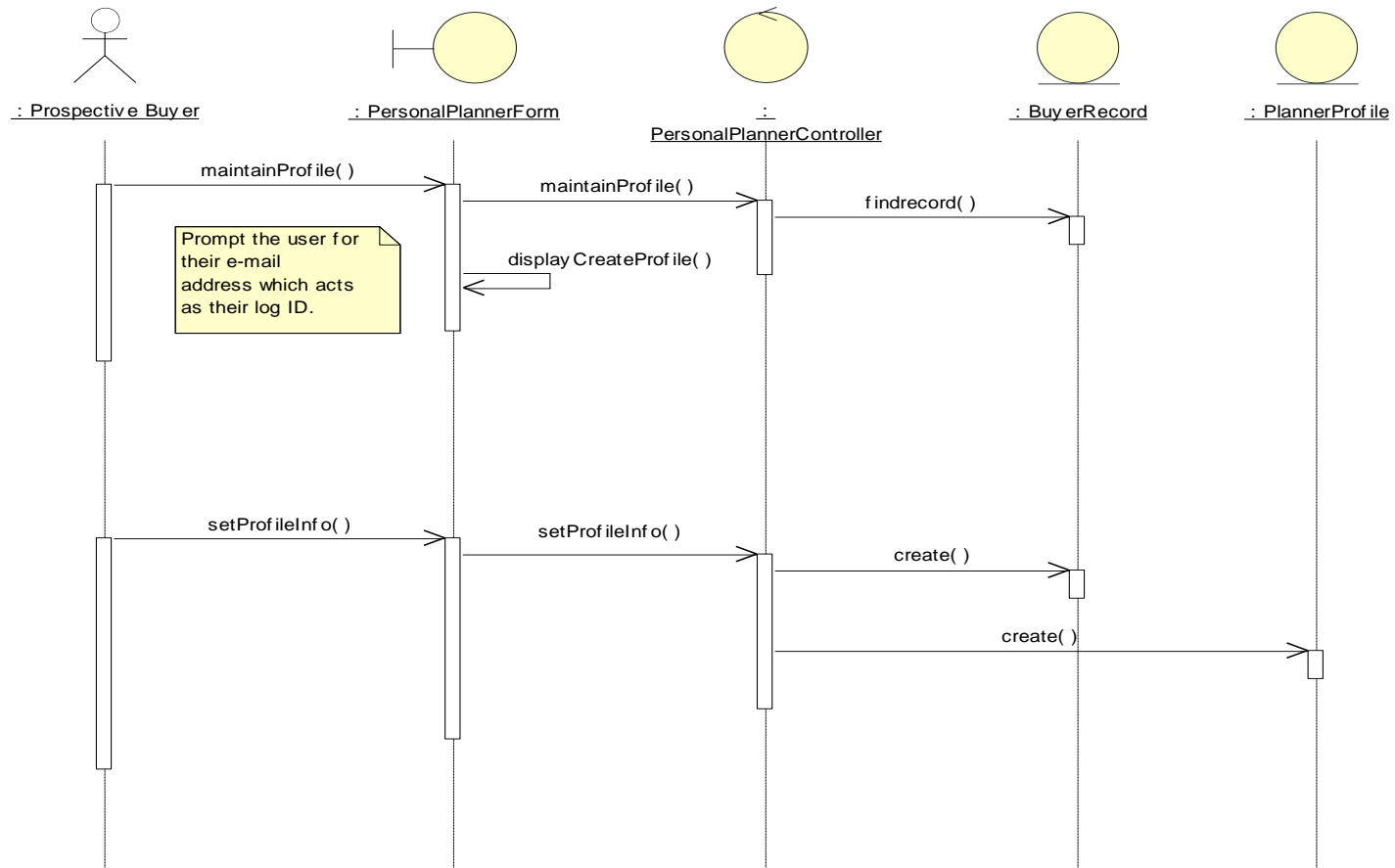
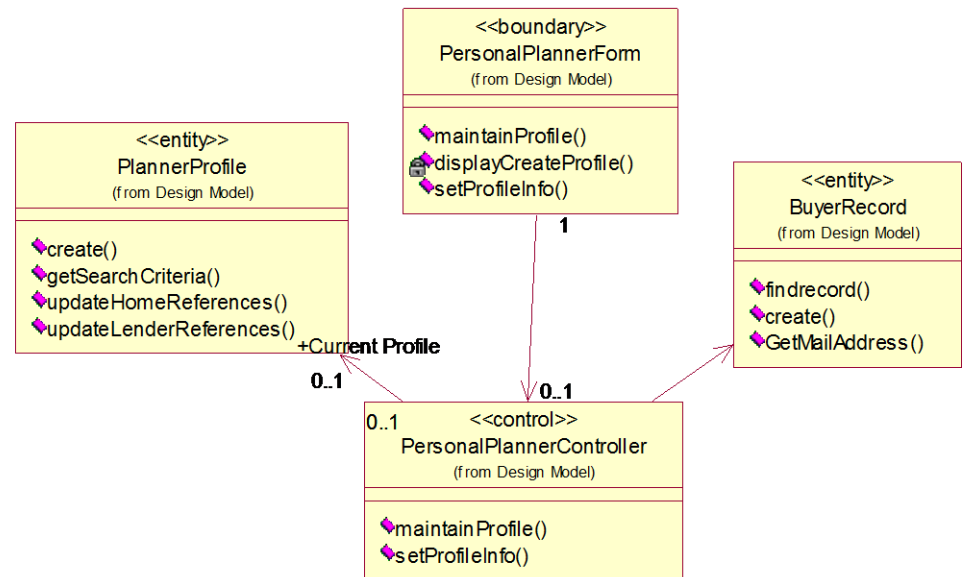
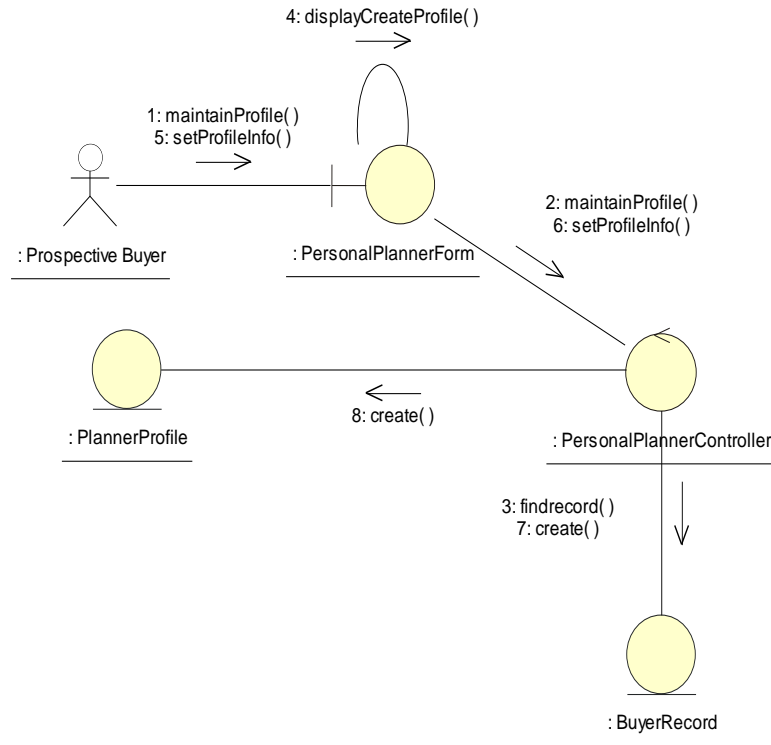


图3 顺序图

【问题1】画出相应的通信图（Communication diagram）。如下图左

【问题2】画出VOPC，要求在其中列出所有的类、类的操作以及类之间的关系。如下图右



3. 阅读下列说明和程序代码，将应填入（n）处的字句填写完整。（问题1和问题2任选一题完成）

策略模式是一种对象行为型模式，它定义一系列的算法，把它们一个个封装起来，并且使它们可相互替换。该模式使得算法可独立于使用它的客户而变化。其结构如图4所示：

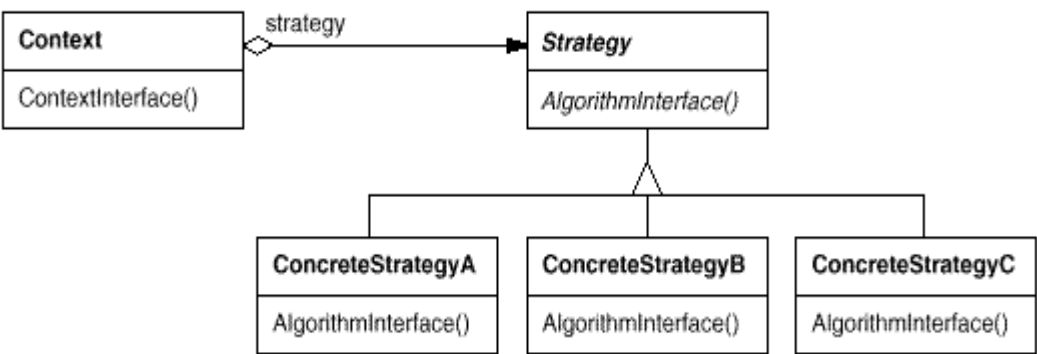


图4 策略模式类图

某软件公司现欲开发一款飞机飞行模拟系统，该系统主要模拟不同种类飞机的飞行特征与起飞特征。需要模拟的飞机种类及其特征见表3。

表 3 需模拟的飞机种类及其特征

飞机种类	起飞特征	飞行特征
直升机(Helicopter)	垂直起飞(VerticalTakeOff)	亚音速飞行(SubSonicFly)
客机(AirPlane)	长距离起飞(LongDistanceTakeOff)	亚音速飞行(SubSonicFly)
歼击机(Fighter)	长距离起飞(LongDistanceTakeOff)	超音速飞行(SuperSonicFly)
鹞式战斗机(Harrier)	垂直起飞(VerticalTakeOff)	超音速飞行(SuperSonicFly)

为支持将来模拟更多种类的飞机，采用策略设计模式设计的类图如图5所示。

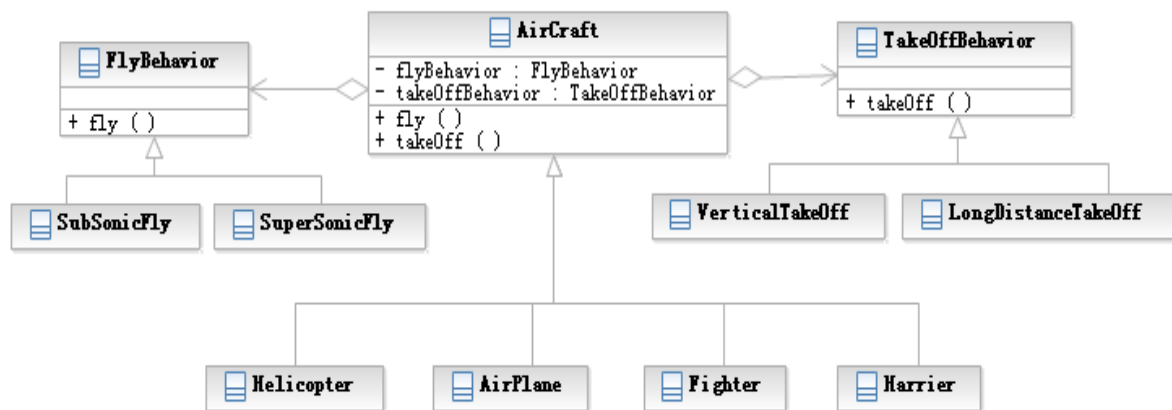


图5 飞机飞行模拟系统类图

在图5中，**AirCraft**为抽象类，描述了抽象的飞机，而类**Helicopter**、**AirPlane**、**Fighter**和**Harrier**分别描述具体的飞机种类，`fly()`和`takeoff()`分别表示不同飞机都具有飞行特征和起飞特征；类**FlyBehavior**与**TakeOffBehavior**为抽象类，分别用于表示抽象的飞行行为与起飞行为；类**SubSonicfly**与**SuperSonicFly**分别描述亚音速飞行和超音速飞行的行为；类**VerticalTakeOff**和**LongDistanceTakeOff**分别描述垂直起飞与长距离起飞的行为。

【问题1】Java代码

```
interface FlyBehavior {
    public void fly();
};
class SubSonicFly implements FlyBehavior{
    public void fly() { System.out.println("亚音速飞行！"); }
};
class SuperSonicFly implements FlyBehavior{
    public void fly() { System.out.println("超音速飞行"); }
};

interface TakeOffBehavior {
    public void takeOff();
};
class VerticalTakeOff implements TakeOffBehavior{
    public void takeOff() { System.out.println("垂直起飞！"); }
};
class LongDistanceTakeOff implements TakeOffBehavior{
    public void takeOff() { System.out.println("长距离起飞！"); }
};
```

```
abstract class AirCraft{
    protected ____ (1) ____;
    protected ____ (2) ____; //(1)、(2)可互换
    public void fly() { ____ (3) ____; }
    public void takeOff() { ____ (4) ____; }
};
class Helicopter ____ (5) ____ Aircraft{
    public Helicopter() {
        flyBehavior= new ____ (6) ____;
        takeOffBehavior=new ____ (7) ____;
    }
}
//其他代码省略
```

- (1)FlyBehavior flybehavior
- (2)TakeOffBehavior takeOffBehavior
- (3)flyBehavior.fly()
- (4)takeOffBehavior.takeOff()
- (5)extends
- (6)SubsonicFly()
- (7)VerticalTakeOff() （1、2可互换）

【问题2】C++代码

```
#include <iostream>
using namespace std;
class FlyBehavior {
    public: virtual void fly()=0;
};
class SubsonicFly: public FlyBehavior {
    public: void fly() { cout << "亚音速飞行！" << endl; }
};
class SuperFly: public FlyBehavior {
    public: void fly() { cout << "超音速飞行！" << endl; }
};

class TakeOffBehavior {
    public: virtual void TakeOff()=0;
};
class VerticalTakeOff: public TakeOffBehavior {
    public: void TakeOff() { cout << "垂直起飞！" << endl; }
};
class LongDistanceTakeOff: public TakeOffBehavior {
    public: void TakeOff() { cout << "长距离起飞！" << endl; }
};
```

```
class AirCraft {
    protected:
        _____(1)_____;
        _____(2)_____; //(1)、(2)可互换
    public:
        void fly() { _____(3)_____;}
        void TakeOff() { _____(4)_____;}
};
class Helicopter: public Aircraft {
    public:
        Helicopter() {
            flyBehavior = new _____(5)_____;
            takeOffBehavior = new _____(6)_____;
        }
        _____(7)_____{
            if (!flyBehavior) delete flyBehavior;
            if (!takeOffBehavior) delete takeOffBehavior;
        }
};
//其他代码省略
```

- (1) FlyBehavior *flyBehavior
- (2) TakeOffBehavior *takeOffBehavior
- (3) flyBehavior->fly()
- (4) takeOffBehavior->takeOff()
- (5) SubSonicFly()
- (6) VerticalTakeOff()
- (7) ~Helicopter() (1、2可互换)