中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

Architectural views

Learning Objectives

- Architectural views
  - 4 + 1 views
  - Correspondence between the views
- An example of architecture design
  - Duke's Forest

视图之间的对应

Views

- Views:
  软件设计的不同高层方面可以而且应该被描述和记录
  - Different high-level facets of a software design can and should be described and documented.
  这些方面通常称为视图:“视图表示软件体系结构的一个部分，它显示了软件系统的特定属性”
  - These facets are often called views: “A view represents a partial aspect of a software architecture that shows specific properties of a software system”.
  这些不同的视图适用于与软件相关的不同问题
  - These distinct views pertain to distinct issues associated with software.

- In summary,
  软件设计是由设计过程产生的多面构件，通常由相对独立和正交的视图组成
  - a software design is a multi-faceted artifact produced by the design process and generally composed of relatively independent and orthogonal views.

An Architectural Model

- In order to eventually address large and challenging architectures, the model we propose is made up of five main views
  - The logical view,
    - which is the object model of the design (when an object-oriented design method is used),
  - The process view,
    - which captures the concurrency and synchronization aspects of the design,
  - The physical view,
    - which describes the mapping(s) of the software onto the hardware and reflects its distributed aspect,
  - The development view,
    - which describes the static organization of the software in its development environment.
- The description of an architecture—the decisions made—can be organized around these four views,
  - and then illustrated by a few selected use cases, or scenarios which become a fifth view.

为了最终处理大型且具有挑战性的架构，我们提出的模型由五个主要视图组成

逻辑视图：功能性需求

是设计的对象模型(当使用面向对象的设计方法时)

过程视图：考虑并发和同步，即模块间的通信

捕获了设计的并发和同步方面

物理视图

描述了软件到硬件的映射，并反映了它的分布式方面

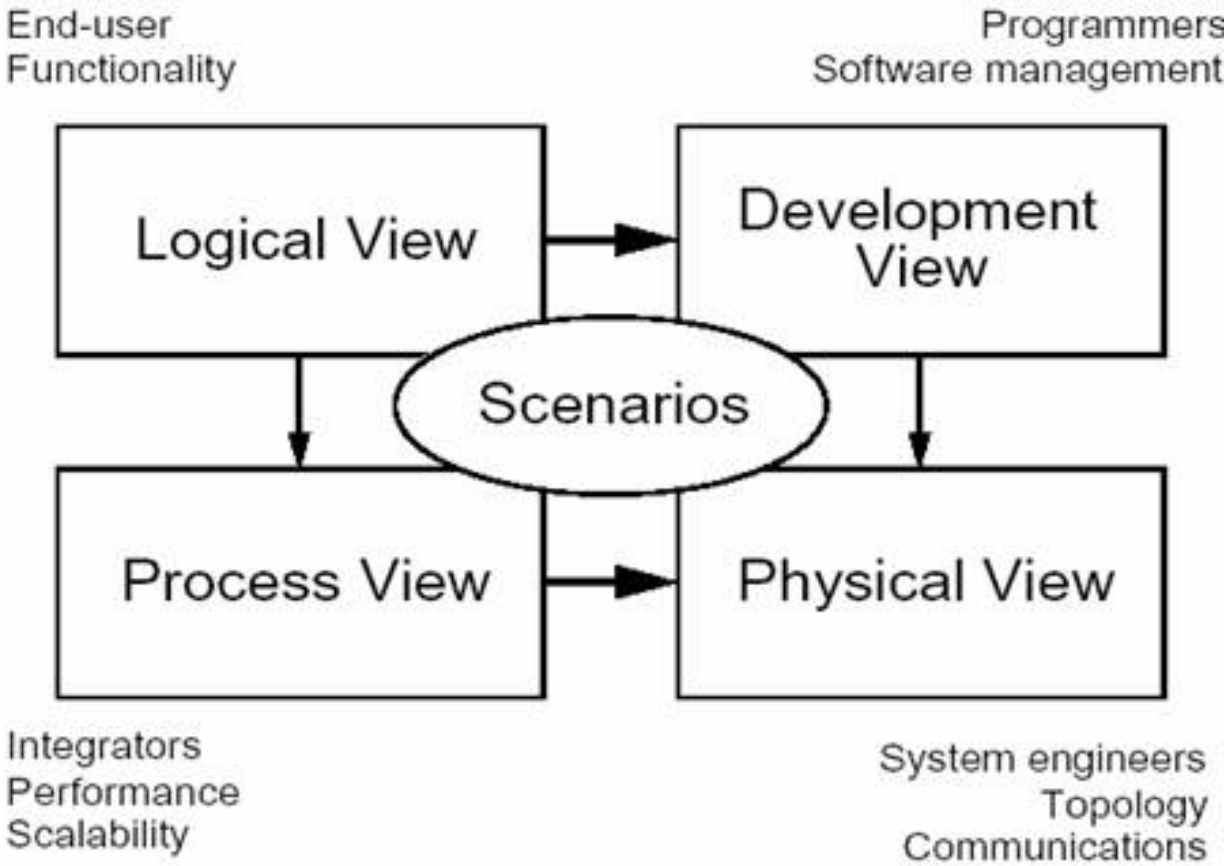
开发视图：考虑可拓展性(extensibility)和灵活性(flexibility)

描述软件在其开发环境中的静态组织

架构的描述—所做的决策—可以围绕这四个视图进行组织

然后通过一些选择的用例或者成为第五个视图的场景来说明

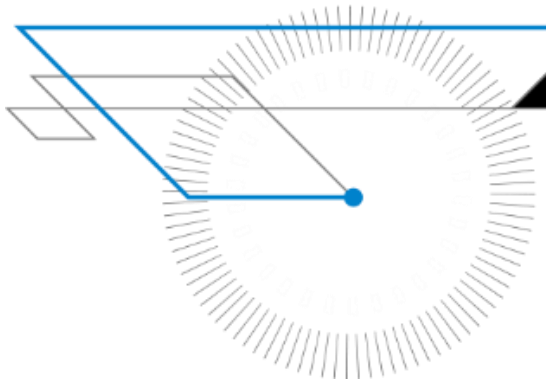
An Architectural Model



设计迭代发生在view之间，外部迭代

The Logical View

functional requirements



The Logical Architecture

逻辑体系结构主要支持功能需求

- The logical architecture primarily supports the functional requirements
  - what the system should provide in terms of services to its users.

系统应该为用户提供哪些服务

系统被分解为一组关键抽象

- The system is decomposed into a set of key abstractions,
  - taken (mostly) from the problem domain, in the form of objects or object classes.

以对象或对象类的形式，(主要)从问题领域获取

它们利用了抽象、封装和继承的原则

- They exploit the principles of abstraction, encapsulation, and inheritance.
  - This decomposition is not only for the sake of functional analysis, but also serves to identify common mechanisms and design elements across the various parts of the system.

这种分解不仅是为了进行功能分析，而且还用于识别跨系统各个部分的公共机制和设计元素

The Logical Architecture

通过类图和类模板，我们可以使用UML方法来表示逻辑架构

- We can use the UML approach for representing the logical architecture, by means of class diagrams and class templates.
  - A class diagram shows a set of classes and their logical relationships: association, usage, composition, inheritance, and so forth.
  - Class templates focus on each individual class; they emphasize the main class operations, and identify key object characteristics. If it is important to define the internal behavior of an object, this is done with state transition diagrams, or state charts..

类图显示了一组类及其逻辑关系：关联、使用、组合、继承，等等

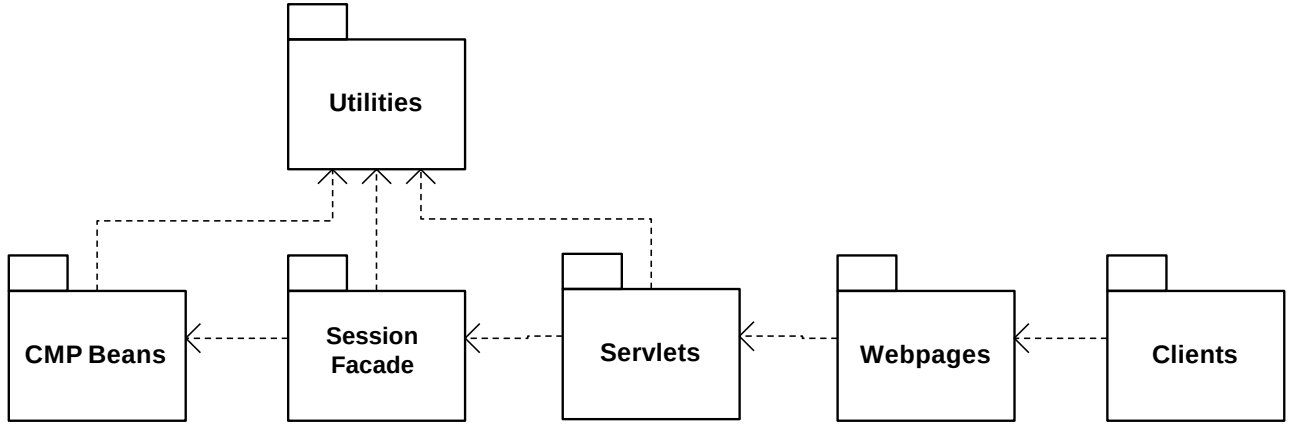
类模板关注于每个单独的类；它们强调主要的类操作，并确定关键的对象特征。如果定义对象的内部行为很重要，那么可以使用状态转换图或状态图来完成

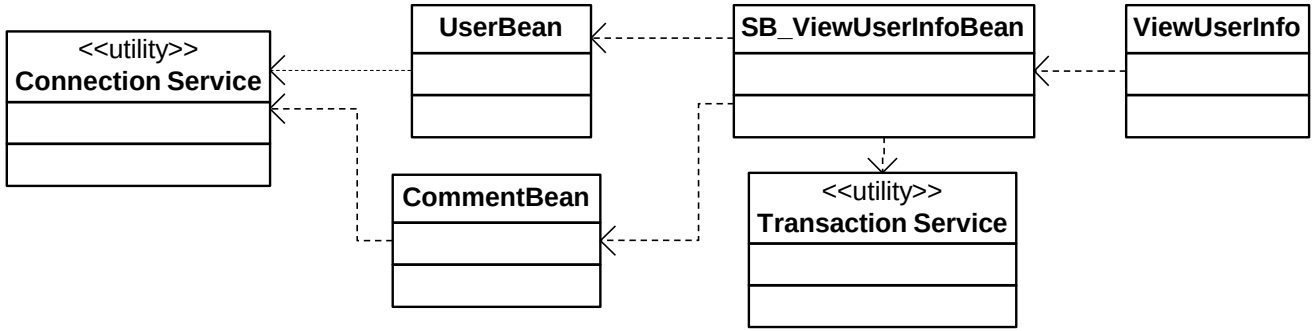
或者是OO方法

- Alternatively to an OO approach,
  - an application that is very data-driven may use some other form of logical view, such as E-R diagrams.

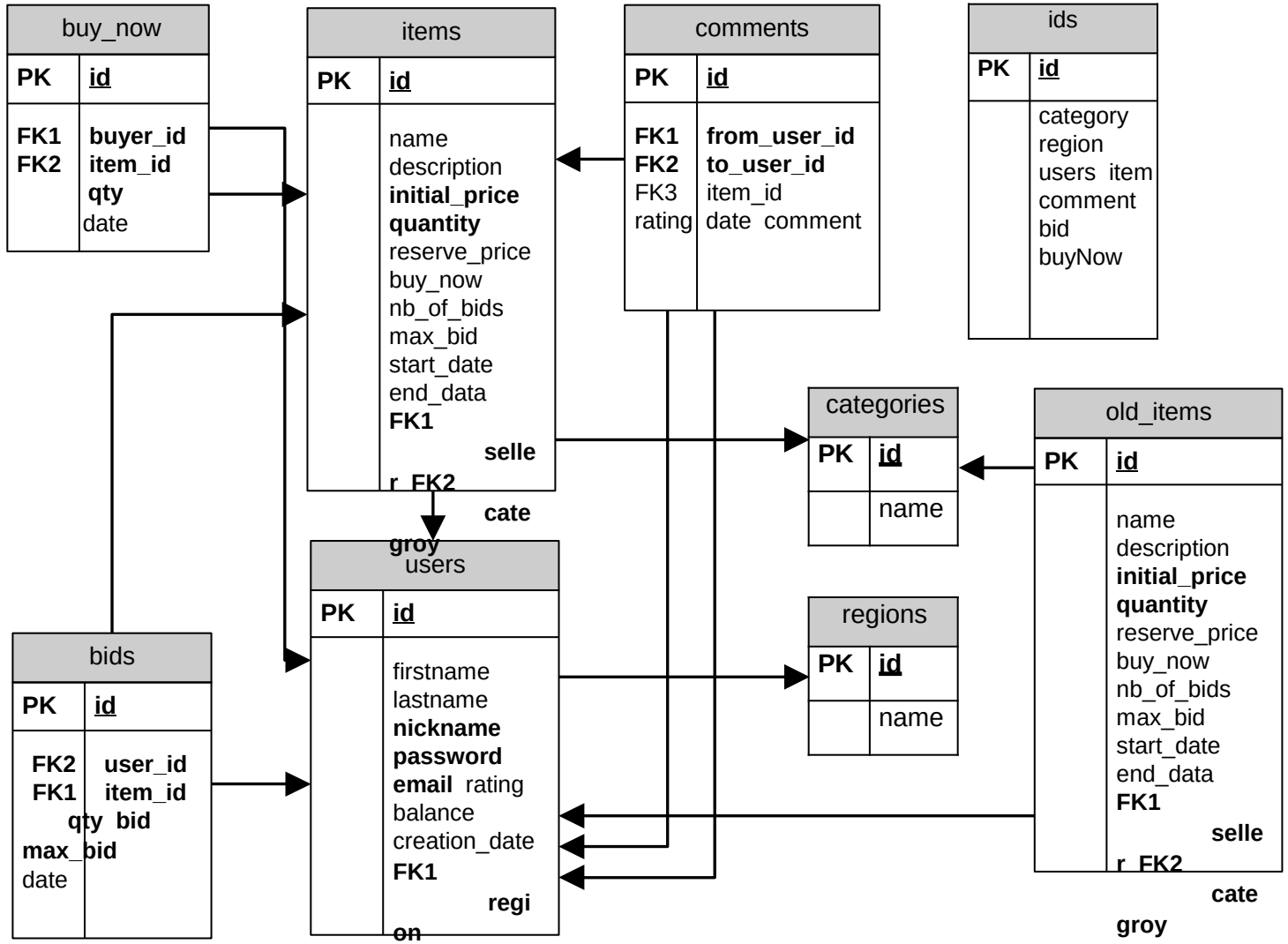
一个数据驱动的应用程序可能会使用一些其他形式的逻辑视图，比如E-R图

The Logical Architecture





The Logical Architecture



The Process View

Non-functional requirements

The Process Architecture

- The process architecture takes into account some non- functional requirements,
  过程架构考虑了一些非功能性需求
  比如性能和可用性
  - such as performance and availability.

过程体系结构可以在多个抽象级别上进行描述，每个级别处理不同的关注点

- The process architecture can be described at several levels of abstraction, each level addressing different concerns.

在最高层次上，过程体系结构可以看作是一组独立执行的通信程序逻辑网络(称为进程)，分布在由局域网或广域网连接的一组硬件资源上

  - At the highest level, the process architecture can be viewed as a set of independently executing logical networks of communicating programs (called “processes”), distributed across a set of hardware resources connected by a LAN or a WAN.

多个逻辑网络可能同时存在，共享相同的物理资源

  - Multiple logical networks may exist simultaneously, sharing the same physical resources.

进程是一组形成可执行单元的任务

- A process is a grouping of tasks that form an executable unit.

The Process Architecture

软件被划分为一组独立的任务

- The software is partitioned into a set of independent tasks.

任务是一个单独的控制线程，可以在一个处理节点上单独调度

  - A task is a separate thread of control, that can be scheduled individually on one processing node.

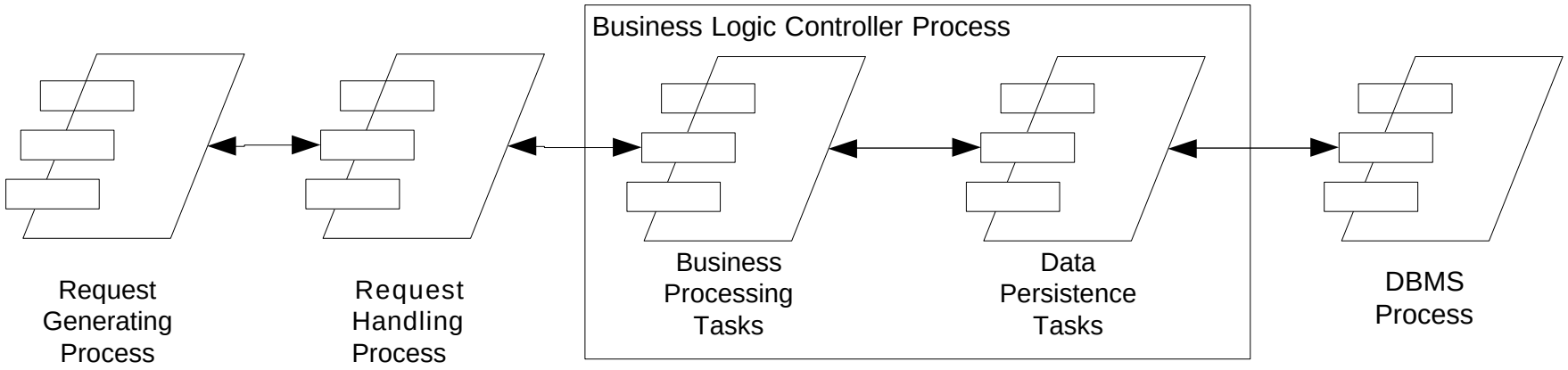
- We can distinguish then:
  - Major tasks communicate via a set of well-defined inter-task communication mechanisms: synchronous and asynchronous message-based communication services, remote procedure calls, event broadcast, etc.

Major tasks通过一组定义良好的任务间通信机制进行通信:同步和异步基于消息的通信服务、远程过程调用、事件广播等

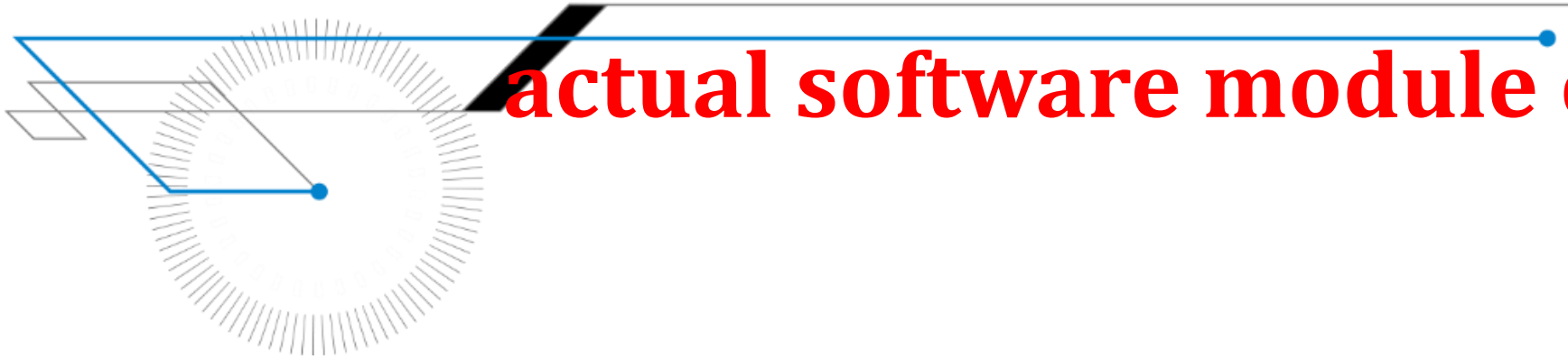
  - Minor tasks may communicate by rendezvous or shared memory. Major tasks shall not make assumptions about their collocation in the same process or processing node.

Minor tasks可以通过集合点或共享内存进行通信。Major tasks不应该假设自己在同一进程或处理节点中的配置情况

The Process Architecture



The Development View



actual software module organization

The Development Architecture

开发体系结构关注

- The development architecture focuses on the
  软件开发环境下的实际软件模块组织
  - actual software module organization on the software development environment.
    软件被打包成小块——程序库或子系统——可以由一个或少数开发人员开发
  - The software is packaged in small chunks—program libraries, or *subsystems*—that can be developed by one or a small number of developers.

系统的开发架构由显示“导出”和“导入”关系的模块和子系统图表示

- The development architecture of the system is represented by module and subsystem diagrams, showing the ‘export’ and ‘import’ relationships.

只有在确定了软件的所有元素之后才能描述完整的开发体系结构。然而，列出控制开发架构的规则是可能的:分区、分组、可见性

  - The complete development architecture can only be described when all the elements of the software have been identified. It is, however, possible to list the rules that govern the development architecture: partitioning, grouping, visibility.

The Development Architecture

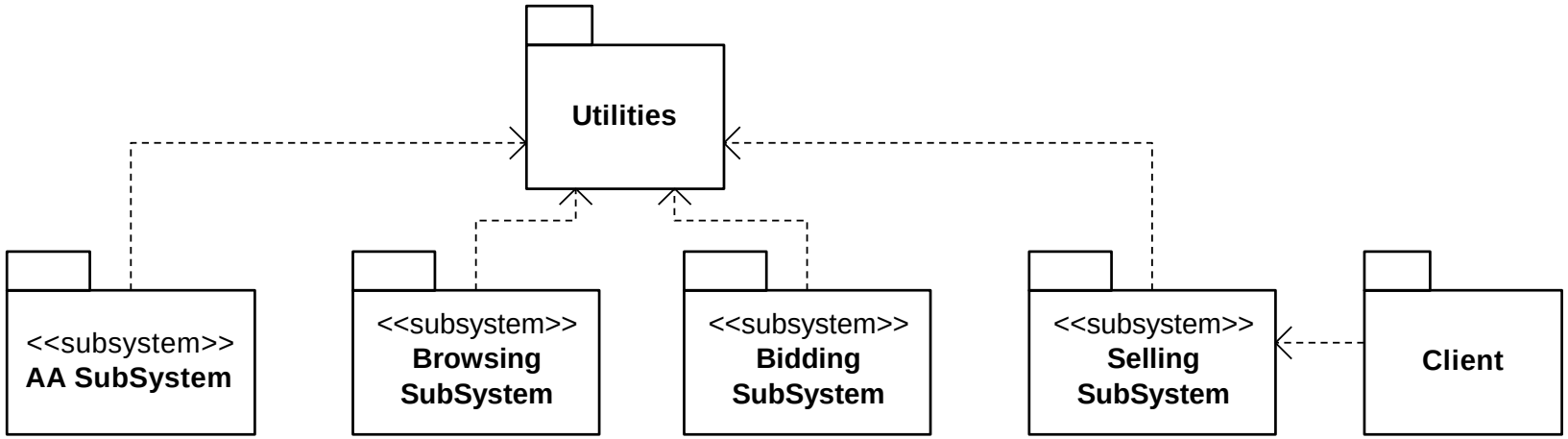
在大多数情况下，开发体系结构考虑的内在需求

- For the most part, the development architecture takes into account internal requirements related to
  - the ease of development,开发的易用性
  - software management,软件管理
  - reuse or commonality,重用或共性
  - and to the constraints imposed by the toolset, or the programming language.以及由工具集或编程语言施加的约束

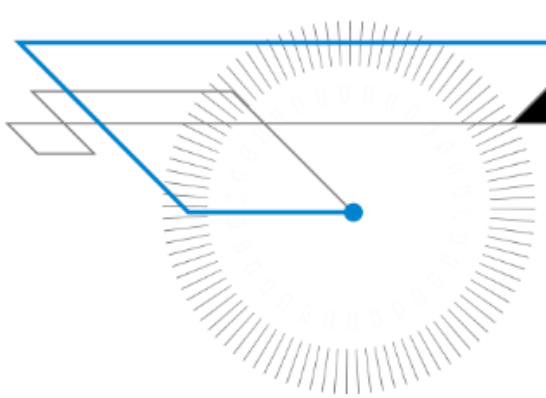
开发视图作为需求分配的基础

- The development view serves as the basis for requirement allocation,
  - 用于将工作分配给团队(甚至是团队组织)
    for allocation of work to teams (or even for team organization),
  - 用于成本评估和计划
    for cost evaluation and planning,
  - 用于监督项目的进度
    for monitoring the progress of the project,
  - 用于推理软件重用、可移植性和安全性
    for reasoning about software reuse, portability and security.

The Development Architecture



The Deployment View



actual deployment

The Deployment Architecture

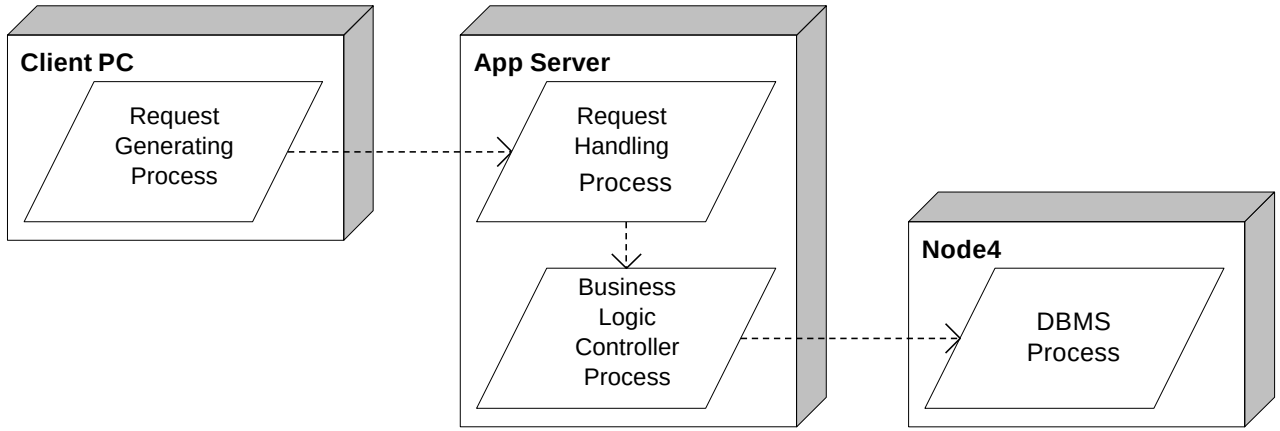
物理架构主要考虑了系统的非功能需求

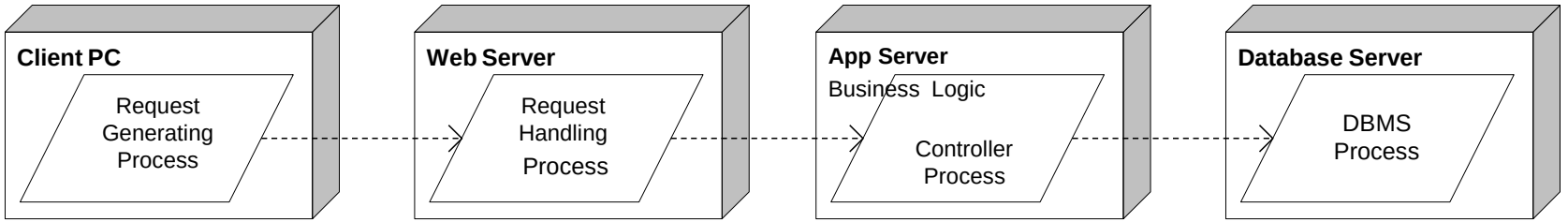
- The physical architecture takes into account primarily the non- functional requirements of the system such as
  - 可用性、可靠性(容错)、性能(吞吐量)和可伸缩性
  - availability, reliability (fault-tolerance), performance (throughput), and scalability.

我们期望使用几种不同的物理配置

- We expect that several different physical configurations will be used:
  - 有些用于开发和测试
  - some for development and testing,
  - 其他用于为不同的站点或不同的客户部署系统
  - others for the deployment of the system for various sites or for different customers.

The Deployment Architecture





Scenarios

通过使用一小组重要场景，四个视图中的元素可以无缝地协同工作

- The elements in the four views are shown to work together seamlessly by the use of a small set of important scenarios:
  - The instances of more general use cases
  - for which we describe the corresponding scripts (sequences of interactions between objects, and between processes).

更通用的用例的实例

为此，我们描述相应的脚本(对象之间和进程之间的交互序列)

这些场景在某种意义上是最重要需求的抽象

- The scenarios are in some sense an abstraction of the most important requirements.

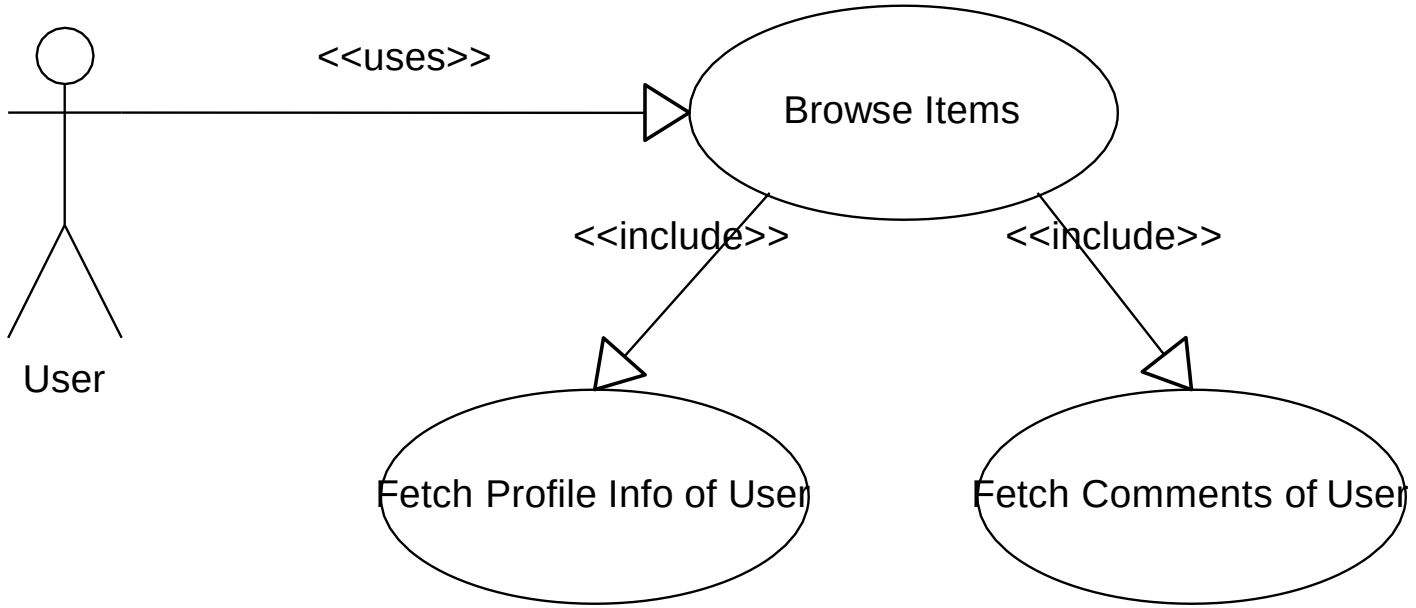
这个视图与其他视图相比是多余的(因此是“+1”)，但是它有两个主要目的

- This view is redundant with the other ones (hence the “+1”), but it serves two main purposes:
  - as a driver to discover the architectural elements during the architecture design as we will describe later
  - as a validation and illustration role after this architecture design is complete, both on paper and as the starting point for the tests of an architectural prototype.

作为在架构设计期间发现架构元素的驱动力，我们将在后面描述

作为架构设计完成后的验证和演示角色，无论是在纸上还是作为架构原型测试的起点

Scenarios


<<uses>>
Browse Items
User
<<include>>
<<include>>
Fetch Profile Info of User
Fetch Comments of User

Correspondence Between the Views

各种视图不是完全正交或独立的

- The various views are not fully orthogonal or independent.

从逻辑到过程视图

- From the logical to the process view

我们确定了逻辑体系结构类的几个重要特征

- We identify several important characteristics of the classes of the logical architecture:
  - 自主性:对象是主动的、被动的、受保护的吗
    Autonomy: are the objects active, passive, protected?
    - 主动对象主动调用其他对象的操作或自己的操作，并完全控制其他对象对自己操作的调用
      an active object takes the initiative of invoking other objects' operations or its own operations, and has full control over the invocation of its own operations by other objects
    - 被动对象从不自发调用任何操作，并且无法控制其他对象对其自己操作的调用
      a passive object never invokes spontaneously any operations and has no control over the invocation of its own operations by other objects
    - 受保护对象不会自发调用任何操作，而是对其操作的调用执行某种仲裁
      a protected object never invokes spontaneously any operations but performs some arbitration on the invocation of its operations.

Correspondence Between the Views

各种视图不是完全正交或独立的

- The various views are not fully orthogonal or independent.

- From the logical to the process view

我们确定了逻辑体系结构类的几个重要特征

- We identify several important characteristics of the classes of the logical architecture:
  - 持久性:对象是暂时的还是永久的?它们是进程或处理器的故障吗?
  - Persistence: are the objects transient , permanent? Do they the failure of a process or processor?
  - 从属:一个对象的存在或持久性是否依赖于另一个对象?
  - Subordination: are the existence or persistence of an object depending on another object?
  - 分布:对象的状态或操作是否可以从物理架构中的许多节点或过程架构中的多个进程访问?
  - Distribution: are the state or the operations of an object accessible from many nodes in the physical architecture, from several processes in the process architecture?

Correspondence Between the Views

确定“正确的”并发量，并定义所需的进程集

- To determine the ‘right’ amount of concurrency and define the set of processes that are needed:
  - 由内而外:从逻辑架构开始
  - Inside-out: Starting from the logical architecture:
  - 定义多个代理任务，其中复用控制对象的单个线程也在同一代理上执行
  - define agent tasks which multiplex a single thread of control object are also executed on that same agent;
  - 需要以互斥方式执行或只需要少量进程的几个类共享一个代理
  - several classes that need to be executed in mutual exclusion, or that require only small amount of processing share a single agent.
  - 这种集群继续进行，直到我们将进程减少到一个合理的小数目，仍然允许分布和使用物理资源
  - This clustering proceeds until we have reduced the processes to a reasonably small number that still allows distribution and use of the physical resources.

Correspondence Between the Views

- To determine the ‘right’ amount of concurrency and define the set of processes that are needed:
  - 由外向内:从物理体系结构开始
  - Outside-in: Starting with the physical architecture:
  - 识别系统的外部刺激(请求)
  - identify external stimuli (requests) to the system,
  - 定义处理刺激的客户机进程和只提供服务而不启动服务的服务器进程
  - define client processes to handle the stimuli and servers processes that only provide services and do not initiate them;
  - 使用问题的数据完整性和序列化约束来定义正确的服务器集
  - use the data integrity and serialization constraints of the problem to define the right set of servers,
  - 并将对象分配给客户端和服务器代理
  - and allocate objects to the client and servers agents;
  - 确定哪些对象必须分发
  - identify which objects must be distributed.

- 结果是类(及其对象)映射到过程体系结构的一组任务和进程
- The result is a mapping of classes (and their objects) onto a set of tasks and processes of the process architecture.

Correspondence Between the Views

- From logical to development

紧密相关的类的集合——类的类别——被分成子系统

- Collections of closely related classes—class categories—are grouped into subsystems.

子系统的定义必须考虑其他约束条件，例如

- Additional constraints must be considered for the definition of subsystems, such as
  - team organization, 团队组织
  - expected magnitude of code (typically 5K to 20K SLOC per subsystem), 预期的代码量(通常每个子系统有5K到20K SLOC)
  - degree of expected reuse and commonality, and 期望重用和通用性的程度
  - strict layering principles (visibility issues), 严格的分层原则(可见性问题)
  - release policy and configuration management.发布策略和配置管理
- Therefore we usually end up with a view that does not have a one to one correspondence with the logical view.

因此，我们通常以一个与逻辑视图没有一一对应关系的视图结束

Correspondence Between the Views

- From process to physical

过程和过程组以各种配置映射到可用的物理硬件上，以便进行测试或部署

- Processes and process groups are mapped onto the available physical hardware, in various configurations for testing or deployment.

就使用的类而言，这些场景主要与逻辑视图有关;当对象之间的交互涉及多个控制线程时，则与过程视图有关

- The scenarios relate mostly to the logical view, in terms of which classes are used, and to the process view when the interactions between objects involve more than one thread of control.

Tailoring the Model

- Not all software architecture need the full “4+1” views.

并不是所有的软件架构都需要完整的“4+1”视图

无用的视图可以从架构描述中删除，例如

- Views that are useless can be omitted from the architecture description, such as

物理视图(如果只有一个处理器)，过程视图(如果只有一个进程或程序)

  - the physical view, if there is only one processor, and the process view if there is only one process or program.

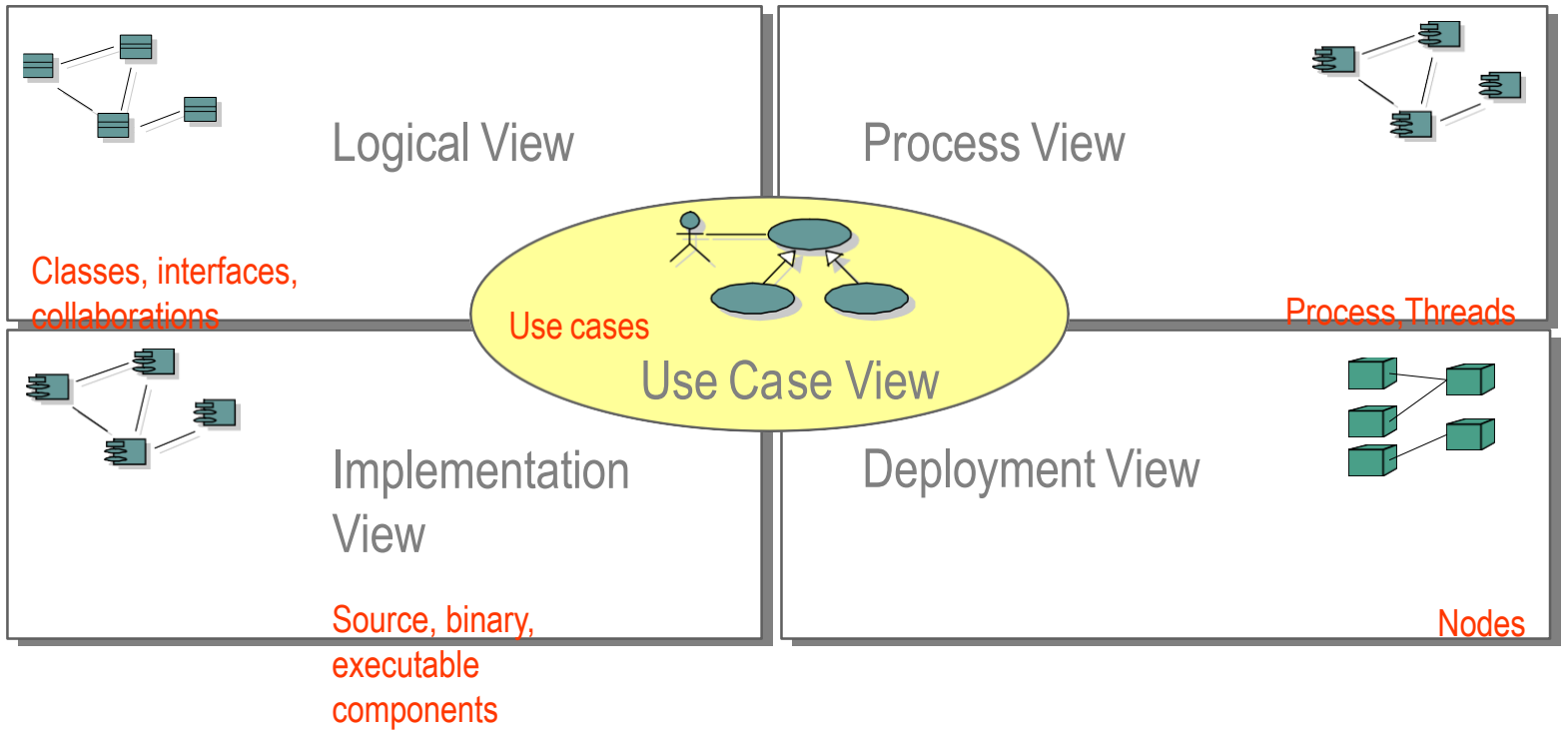
对于非常小的系统，甚至有可能逻辑视图和开发视图是如此相似以至于它们不需要单独的描述

  - For very small system, it is even possible that the logical view and the development view are so similar that they do not require separate descriptions.

这些场景在所有情况下都是有用的

  - The scenarios are useful in all circumstances.

Architectural Views



conclusion

视图	逻辑视图	过程视图	开发视图	物理视图	场景
组件	类	任务	模块、子系统	节点	步骤、脚本
连接工具	关联、继承、约束	会面、消息、广播、RPC 等	编译依赖性、“with”语句、“include”	通信媒体、LAN、WAN、总线等	
容器	类的种类	过程	子系统（库）	物理子系统	Web
涉众	最终用户	系统设计人员、集成人员	开发人员、经理	系统设计人员	最终用户、开发人员
关注点	功能	性能、可用性、S/W 容错、整体性	组织、可重用性、可移植性、产品线	可伸缩性、性能、可用性	可理解性
工具支持	Rose	UNAS/SALE DADS	Apex、SoDA	UNAS、Openview DADS	Rose

Use Case View

1. **Overview**
2. **Graphical Constructs**
3. **Textual Description**
4. **The Architectural View of the Use Case Model**

1. Overview

Use Case View

- Captures system functionality as seen by users
- Built in early stages of development
- Developed by ***analysts and domain experts***
- **System behavior**, that is what functionality it must provide, is ***documented*** in a use case model.

- 捕捉用户看到的系统功能
- 在开发的早期阶段建造
- 由分析师和领域专家开发
- 系统行为，即它必须提供的功能，被记录在用例模型中。

Use Case Model

举例说明系统的预期功能(用例)，它的环境(参与者)，以及用例和参与者之间的关系(用例图)。
·为客户或最终用户和开发人员提供讨论系统功能和行为的工具
·在先启阶段开始识别角色和系统的主要用例，然后在精化阶段通过添加更多的细节和额外的用例来成熟

- illustrates the system’s ***intended functions*** (use cases), its ***surroundings*** (actors), and ***relationships*** between the use cases and actors (use case diagrams).

÷provides a vehicle used by the customers or end users and the developers to ***discuss the system’s functionality and behavior***.

÷starts in the Inception phase with the ***identification of actors*** and ***principal use cases*** for the system, and is then matured in the elaboration phase, by adding more details and additional use cases.

2. Graphical Constructs

Actors

代表必须与系统交互的任何人或任何事物;他们不是系统的一部分

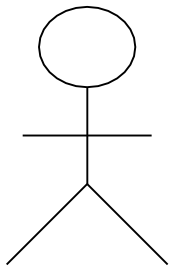
- represent anyone or anything that must interact with the system: they are not part of the system.

- may:
  - only input information to the system
  - only receive information from the system
  - input and receive information to and from the system

- 只向系统输入信息
- 只接收系统的信息
- 向系统输入和接收信息

在UML中，参与者被表示为一个有名字的火柴人

- In the UML, an actor is represented as a stickman with a name:



Administrator

Use Cases

- represent the functionality provided by the system; that is:

÷what capabilities will be provided to an actor by the system or
÷what tasks are performed by each actor?

表示系统所提供的功能;那就是:
· 系统将为参与者提供什么功能
· 每个参与者执行什么任务

由系统执行的一系列事务，为特定参与者产生可测量的值结果

- sequence of transactions performed by a system that yields a measurable result of values for a particular actor.

在UML中，用例由包含名称的椭圆表示

- In the UML, a use case is represented by an oval with a name inside:

Use Case Relationships

关联：
· 表示参与者和用例之间通信的关系；
· 以两种方式或只有一种方式导航

- *Association*:

÷a relationship that represents communication between an actor and a use case;

÷can be navigable in both ways or in only one way.

继承：
·泛化或专门化，参与者之间可能存在的关系

- Inheritance:

÷Generalization or specialization relationships that may exist between actors.

用例之间可能存在两种类型的关系：uses和extends

- Two types of relationships that may exist between use cases: *uses* and *extends*:

由多个用例共享的功能可以放在单独的用例中，该单独用例通过uses关系与这些用例相关

÷A functionality shared by multiple use cases can be placed in a separate use case which is related to these uses cases by a *uses* relationship.

extends关系用于表示

÷An *extends* relationship is used to show:

-Optional behavior

-Behavior that is only run under certain conditions, such as triggering an alarm

-Different flows which may be run based on actor selection

-可选的行为
-只在特定条件下运行的行为，如触发警报
-可以根据参与者的选择运行的不同流

Use Case Diagrams

为系统标识的参与者、用例及其交互的图形化视图

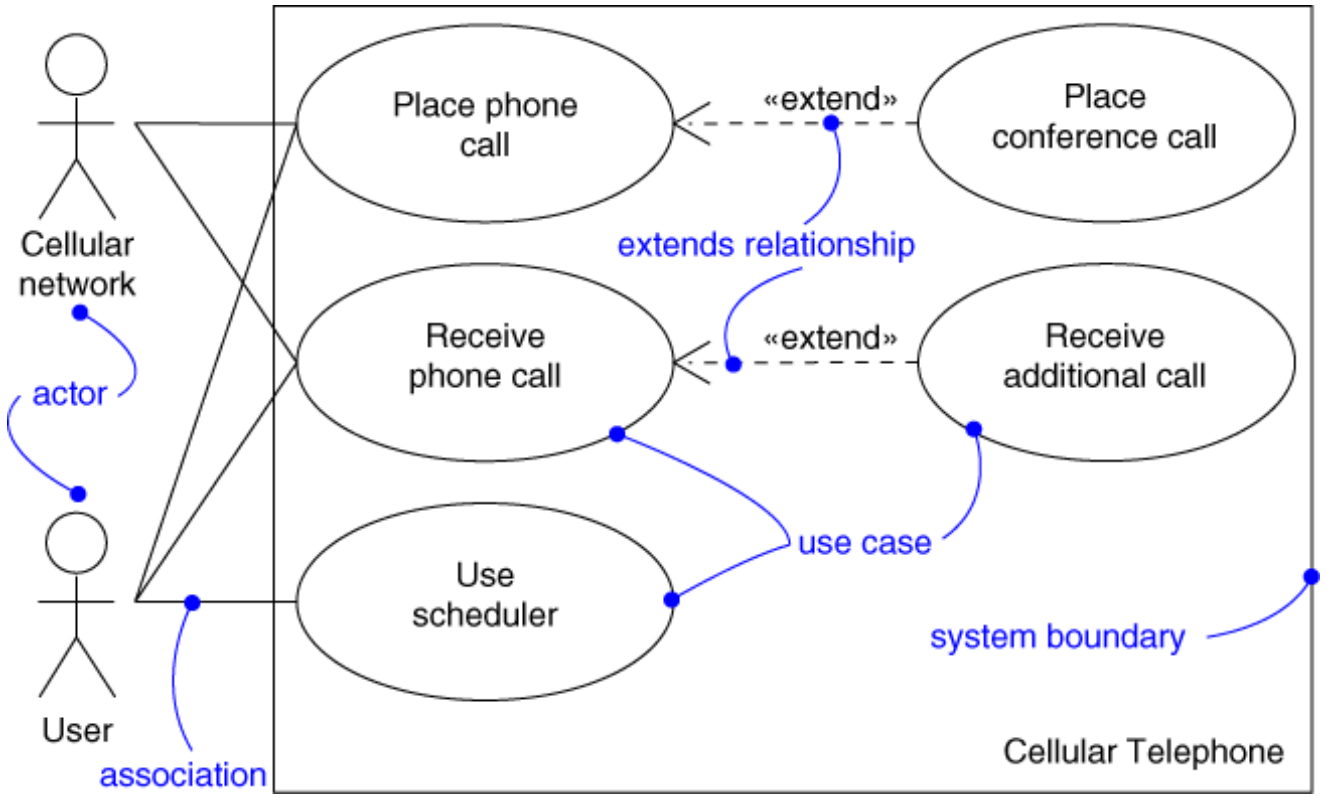
- A graphical view of the actors, use cases, and their interactions identified for a system.

由系统边界，以及参与者、用例及其关系的图形描述组成

÷Consists of the system boundary, and the graphical description of the actors, use cases, and their relationships.

例如:(嵌入式)移动电话系统

- Example: (Embedded) Cellular Telephone System



3. Textual Description of a Use Case

- Each use case is documented with a flow of events, which is a description of the events needed to accomplish the required behavior.

每个用例都用事件流进行了文档化，这是对完成所需行为的事件的描述

- The flow of events is written in the language of the domain and describe what the system should do and not how the system does it.

事件流是用领域的语言编写的，描述系统应该做什么，而不是系统如何做

- The flow of events should include:

事件流程应包括：
-用例何时以及如何开始和结束
-用例与参与者有什么交互
-用例需要什么数据
-用例的正常事件序列
-对任何备用或异常流的描述

-When and how the use case starts and ends
-What interaction the use case has with the actors
-What data is needed by the use case
-The normal sequence of events for the use case
-The description of any alternate or exceptional flows

- **Template:**

*X.Flow of Events for the <**name**> Use Case*

1. *Preconditions*
2. *Main Flow*
3. *Subflows (if applicable)*
4. *Alternative Flows*

Where X is a number from 1 to the number of use cases

4. The Architectural View of the Use Case Model

只包含架构上重要的用例(而最终的用例模型包含所有的用例)

- Contains only architecturally significant use cases (whereas the final use case model contains all the use cases).

在先启和精化阶段定义，并允许建立弹性架构

÷Is defined during inception and elaboration phases and allows the establishment of a resilient architecture.

逻辑视图是使用用例模型的体系结构视图中确定的用例派生的

÷The *logical view* is derived using the use cases identified in the architectural view of the use case model.

架构上重要的用例：

- **Architecturally significant use cases**:

÷are the ones that cover the main tasks or functions the system is to accomplish.

÷could possibly impact the architecture

÷include:

-critical use cases, those that are most important to the users of the system (from a functionality perspective)

-use cases that carry the major risks

-use cases that have the most important quality requirements, such as performance, security, usability, etc.

· 覆盖主要任务或功能系统来完成。
· 可能影响体系结构
· 包括:
-关键用例，即对系统用户最重要的用例(从功能的角度)
-带有主要风险的用例
-具有最重要质量要求的用例，如性能、安全性、可用性等

次要和可选用例对体系结构来说不是关键

- Secondary and optional use cases are not key to the architecture.

Example

提供ATM系统架构的用例视图

-Provide the use case view for the architecture of an ATM System.

需求概述

Overview of the Requirements

银行同业联盟是一个假想的金融机构，它已指示其软件开发子公司-银行同业软件-开发支持自动柜员机网络的新服务

The Interbank Consortium, a hypothetical financial institution, has directed its software development subsidiary, Interbank Software, to develop new services that support a network of automated teller machines (ATMs).

客户使用自动柜员机对他们的账户进行查询、取款、存款和资金转移。必须防止小偷或骗子干扰这些行动

Customers use ATMs to make queries, withdrawals, deposits and funds transfers involving their accounts. Thieves or crooks must be prevented from interfering with these actions.

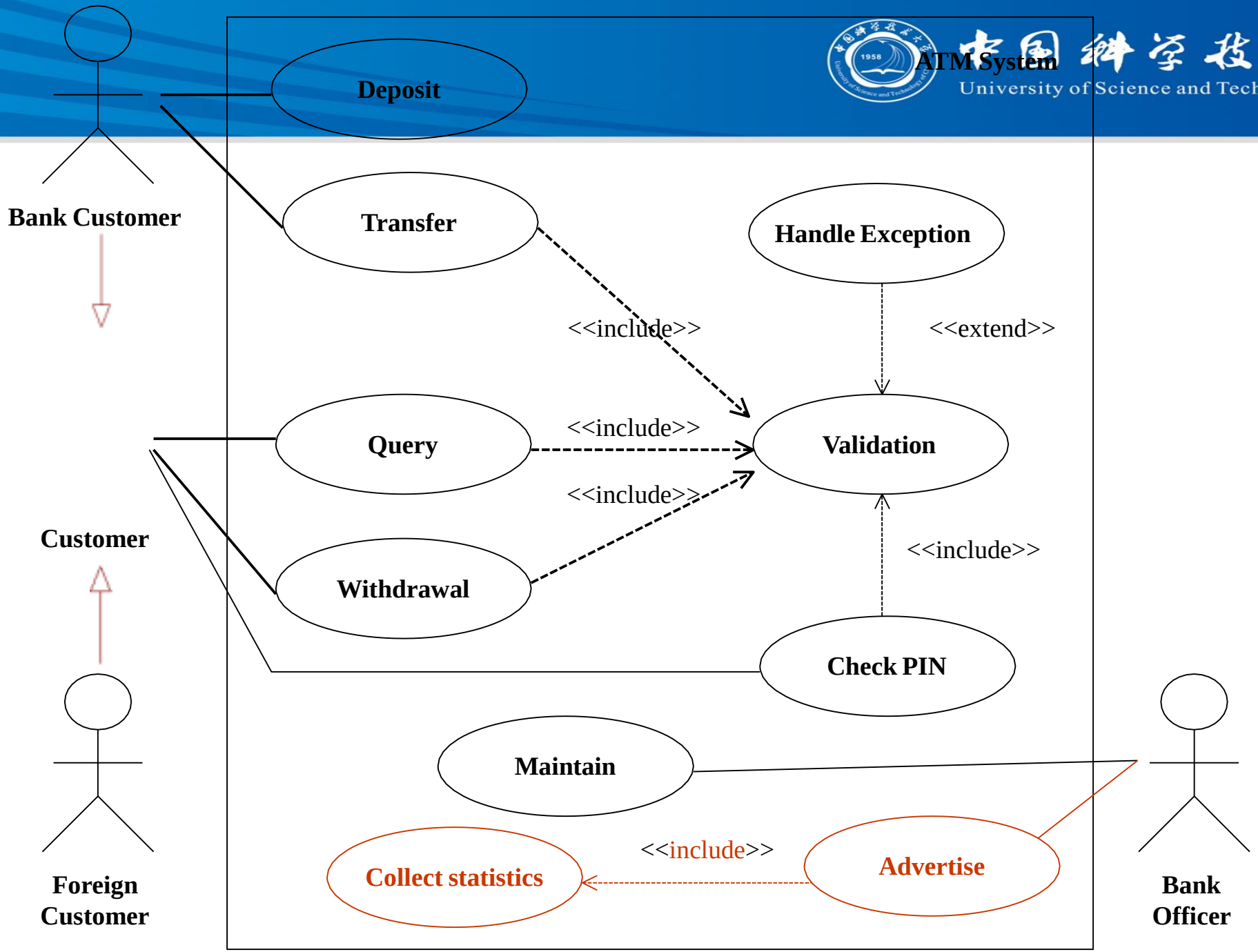
Interactions with the ATM would work like this:
1.The customer inserts his/her bank card into the ATM.
2.The ATM prompts the customer for a "password" which the user enters at the ATM.
3.The customer then selects an action to be performed; the selected action is then performed by the branch (perhaps causing dispersal of cash at the ATM).

与自动取款机的交互是这样的:
1.顾客将他/她的银行卡插入自动取款机。
2.自动取款机提示用户输入用户在自动取款机上输入的“密码”。
3.然后客户选择要执行的操作;然后由分支执行所选的操作(可能会导致ATM上的现金分散)

此外，银行希望能够使用该系统来维护有关客户行为的统计数据，以适应客户的需求，并在客户使用该系统时向他们发送一些广告

Additionally, the bank would like to be able to use the system to maintain statistics about customers’ behaviour in order to adapt its services to their needs, and also to send them some advertisements when they are using the system.

ATM System


Bank Customer
Deposit
Transfer
Handle Exception
<<include>>
<<extend>>
Query
<<include>>
Validation
<<include>>
Customer
Withdrawal
<<include>>
Check PIN
Maintain
Collect statistics
<<include>>
Advertise
Foreign Customer
Bank Officer

• Example: Textual Description for Check PIN Use Case

Flow of Events for *Check PIN* Use Case

Main flow of events: The use case starts when the system prompts the *User* for a PIN number. The *User* can now enter a PIN number via the keypad **(E1)**. The *User* commits the entry by pressing the Enter button **(E2)**. The system then checks this PIN number to see if it is valid **(S1, E3)**. If the PIN number is valid, the system acknowledges the entry, thus ending the use case.

当系统提示用户输入PIN号时，用例开始。用户现在可以通过键盘(E1)输入PIN号码。用户通过按Enter按钮(E2)提交条目。然后系统检查这个PIN号，看它是否有效(S1，E3)。如果PIN号是有效的，系统确认条目，从而结束用例

Subflows:

S1: The system invokes ***Validate*** use case.

系统调用验证用例

Alternative flow of events:

E1: The *User* can clear a PIN number any time before committing it and reenter a new PIN number.

用户可以在提交密码之前清除密码，并重新输入新的密码

E2: The *User* can cancel a transaction at any time by pressing the Cancel button, thus restarting the use case. No changes are made to the *User*'*s* account.

用户可以在任何时候按下cancel按钮取消事务，从而重新启动用例。不更改用户的帐户

E3: If the *User* enters an invalid PIN number, the use case restarts. If this happens three times in a row, the system cancels the entire transaction, preventing the *User* from interacting with the ATM for 30 minutes.

如果用户输入了无效的PIN号，则用例将重新启动。如果这种情况连续发生三次，系统将取消整个事务，使用户在30分钟内无法与ATM交互

Logical View

1. Overview
2. Static Structures
3. Interactions
4. Dynamic Behavior
5. Example: Logical View for the ATM

1.The Logical Architecture

逻辑视图的目的是指定系统的功能需求。逻辑视图的主要工件是设计模型

-The purpose of the logical view is to ***specify the functional requirements of the system***. The main artifact of the logical view is the design model:

设计模型对系统的功能行为进行了具体的描述。它是由分析模型导出的

÷The ***design model*** gives a **concrete** description of the functional behavior of the system. It is derived from the analysis model.

分析模型给出了基于用例模型的系统行为的抽象描述

÷The ***analysis model*** gives an **abstract** description of the system behavior based on the use case model.

一般情况下，只有设计模型在逻辑视图中维护，因为分析模型提供了一个粗略的草图，然后将其细化为设计工件

÷In general only the design model is maintained in the logical view, since the analysis model provides a rough sketch, which is later refined into design artifacts.

Design Model

设计模型由协作类组成，它们被组织成子系统或包

-The design model consists of collaborating classes, organized into subsystems or packages.

-Artifacts involved in the design model may include:

设计模型所涉及的工件可能包括:
-类、交互和状态图
-使用包图描述的子系统和它们的接口

÷***class***, ***interaction***, and ***state*** diagrams

÷the ***subsystems and their interfaces*** described using ***package*** diagrams

2.Static Structures

Notion of Class

类的概念

- a ***description of a group of objects*** with:

÷common properties (***attributes***)
÷common behavior (***operations***)
÷common ***relationships*** to other objects, and common semantics.

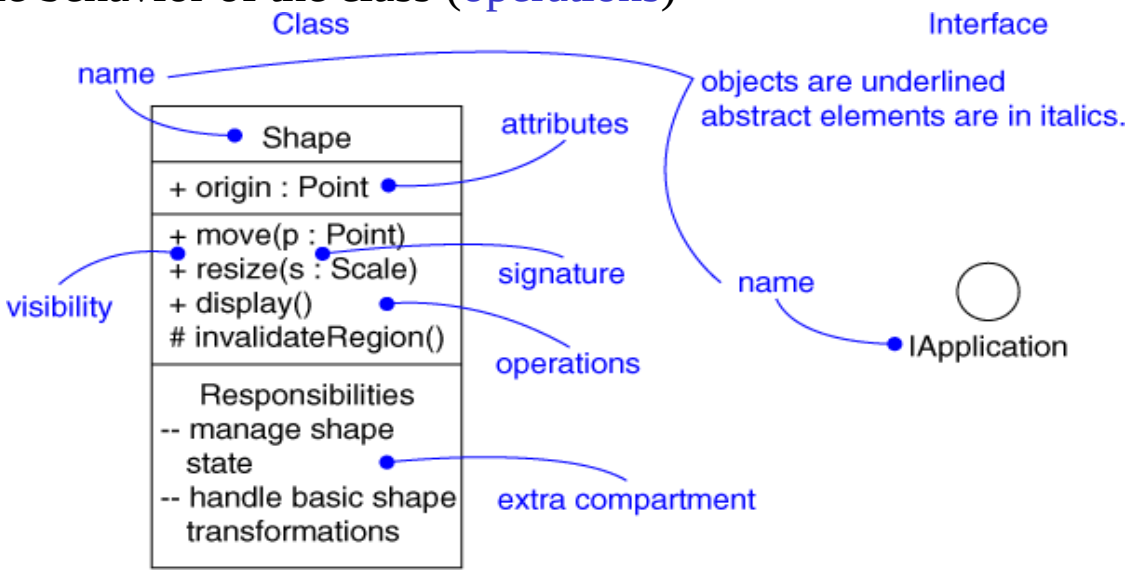
一组对象的描述:
-共同特性(属性)
-共同行为(操作)
-与其他对象之间的共同关系和共同的语义

- in the UML classes are represented as compartmentalized rectangles:
  - top compartment contains the name of the class
  - middle compartment contains the structure of the class (attributes)
  - bottom compartment contains the behavior of the class (operations)

在UML中，类被表示为划分的矩形:
•顶层包含类的名称
•中间区域包含类的结构(属性)
•底层包含类的行为(操作)

Visibility:

+	*public*
#	*protected*
-	*private*

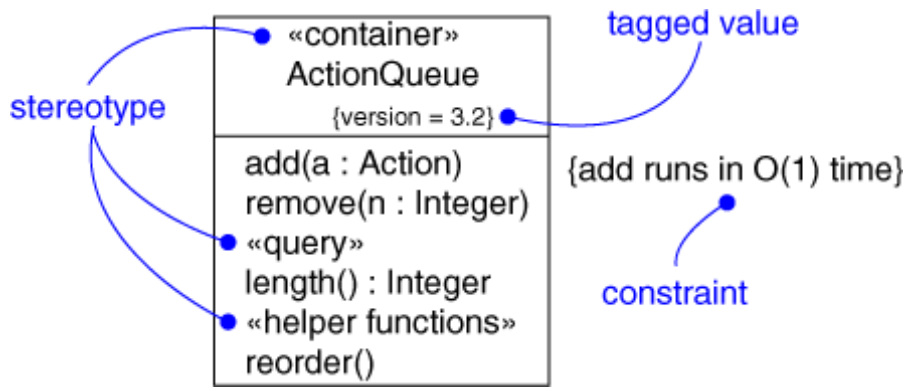


扩展机制

Extensibility Mechanisms

- Stereotype
- Tagged value
- Constraint

原型
标记的值
约束



Notion of Stereotype

- provides the capability to ***create a new kind of modeling element***.
- we can create new kinds of classes by defining stereotypes for classes.
- the stereotype for a class is shown below the class name enclosed in guillemets (<< >>).
- examples of class stereotypes: ***exception, utility etc.***

· 提供创建一种新的建模元素的能力
· 我们可以通过定义类的原型来创建新的类类型。
· 类的原型显示在书名号中包含的类名下面(< < > >)。
· 类原型的例子:exception，utility等

Boundary, Entity, and Control Classes

Rational统一过程提倡通过查找边界、控制和实体类来找到系统的类

- The ***Rational Unified Process*** advocates for finding the classes for a system by looking for *boundary*, *control*, and *entity* classes.

Entity classes:

· 对通常存在较长时间的信息和相关行为进行建模
· 可能反映真实世界的实体，或者可能需要执行系统内部的任务
· 与应用程序无关:可以在多个应用程序中使用

- model information and associated behavior that is ***generally long lived***

•may ***reflect a real-world entity***, or may be needed to perform tasks internal to the system

•are ***application independent***: may be used in more than one application.

Boundary classes:

•handle the ***communication between the system surroundings and the inside*** of the system

•can provide the interface to a user or another system

· 处理系统环境与系统内部的通信
· 能够为用户或其他系统提供接口

Control classes:

- model ***sequencing behavior*** specific to one or more use cases.

•typically are ***application-dependent*** classes.

· 为特定于一个或多个用例的行为排序建模
· 通常是依赖于应用程序的类

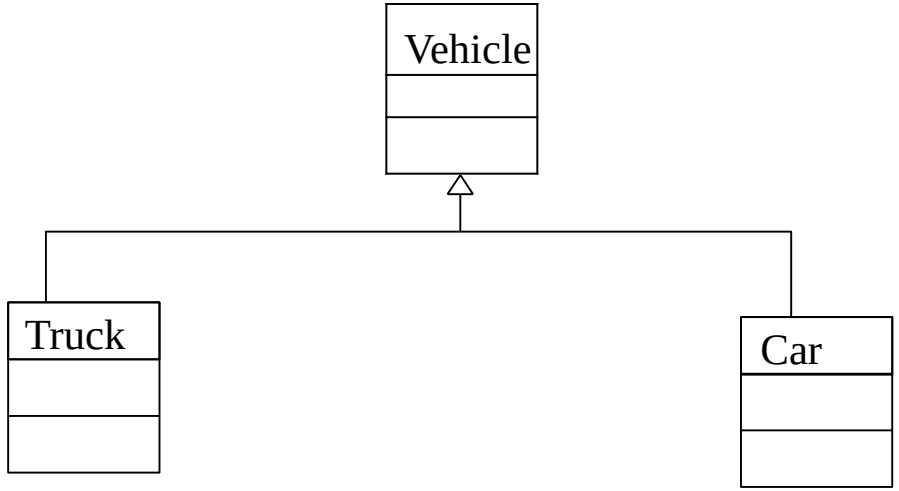
Relationships

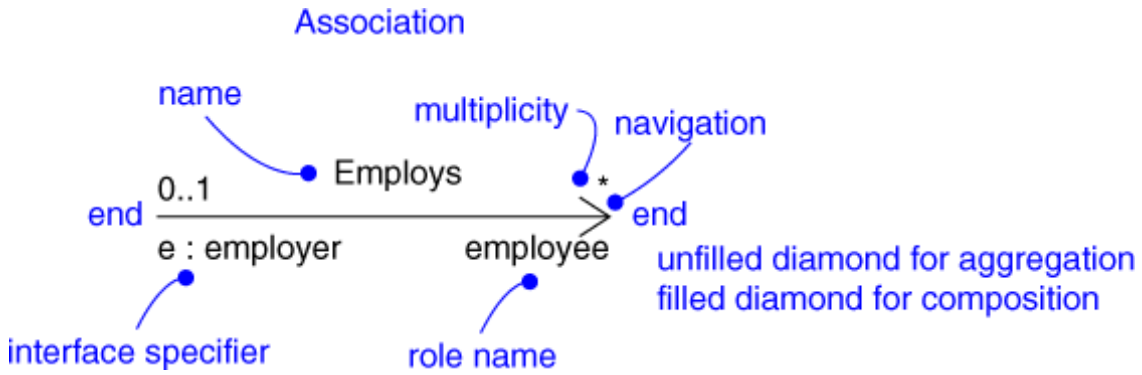
为对象交互提供管道

- Provide the conduit for object interaction
- Several kinds of relationships:

关联 依赖 实现 聚合 继承

- ***Association***
- ***Dependency***
- ***Realization***
- ***Aggregation***
- ***Inheritance***



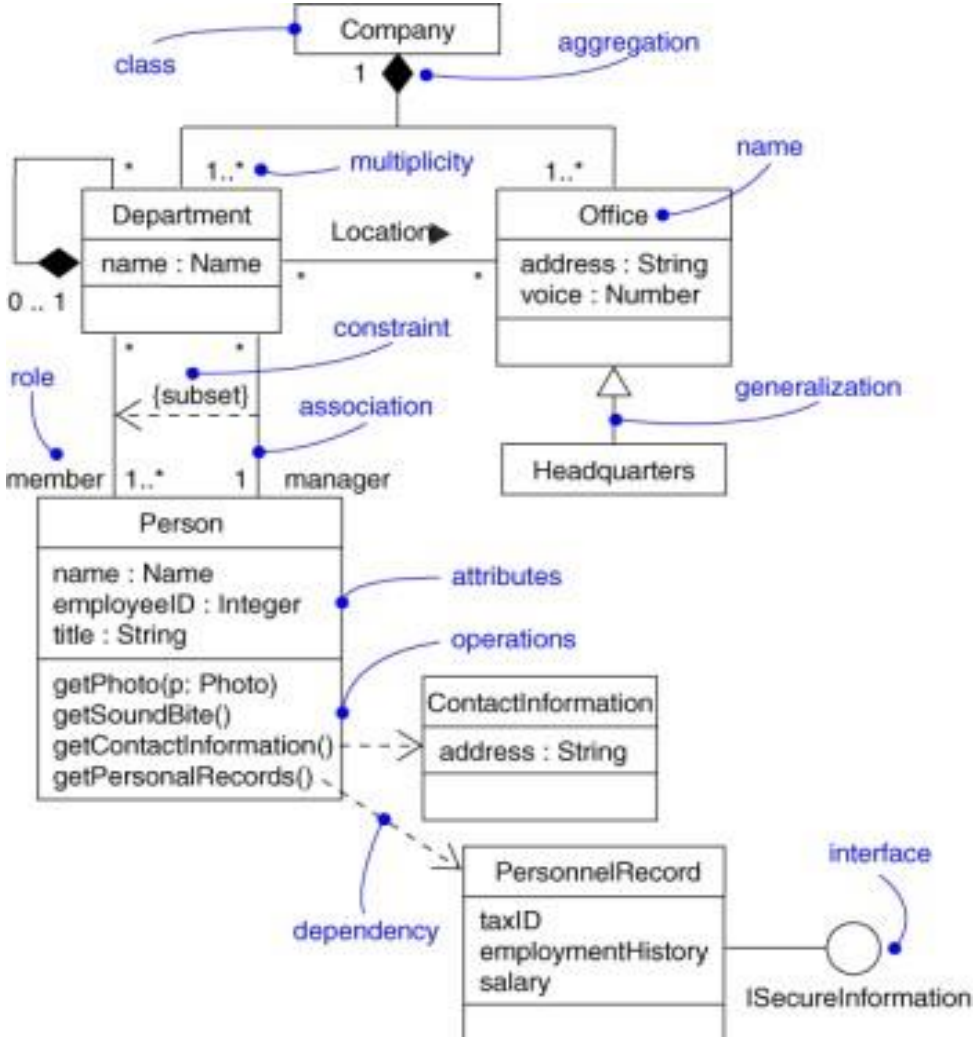


Class Diagram

- Purpose

目的
•提供模型中部分或所有类/接口的图片或视图
•系统的静态设计视图

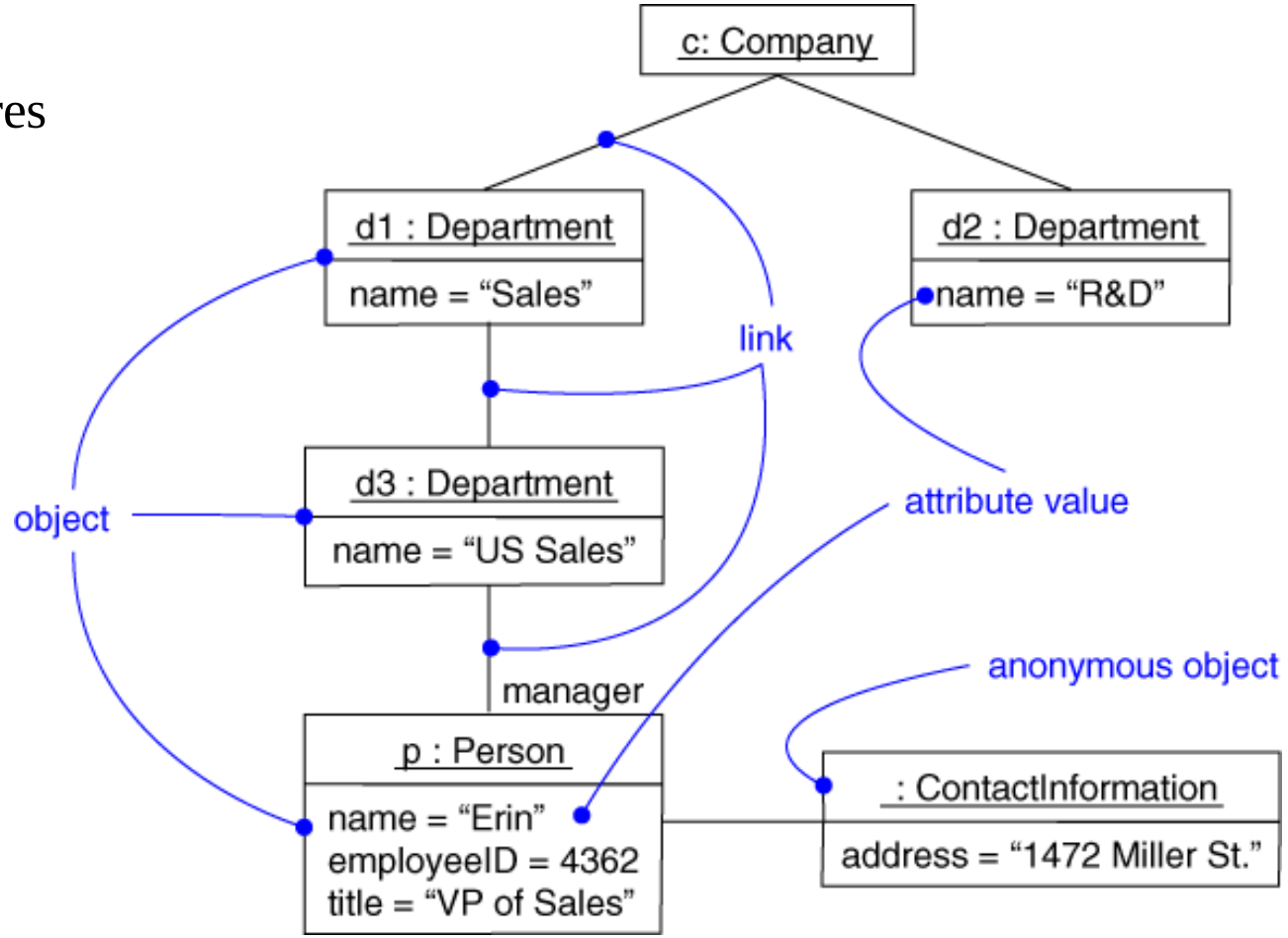
  - •Provide a picture or view of some or all the ***classes/interfaces in the model***
  - •Static design view of the system



Object Diagram

- Shows a ***set of objects*** and ***their relationships*** at a point in time
- Shows ***instances*** and ***links***
- Built during analysis and design (address the static design view)
- Purpose
  - Illustrate data/object structures
  - Specify snapshots

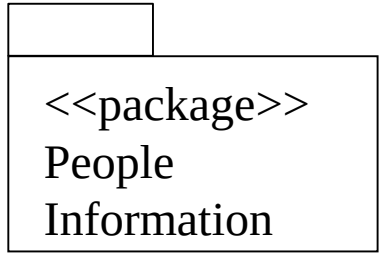
· 在某一时刻显示一组对象及其关系
· 显示实例和链接
· 在分析和设计期间构建(解决静态设计视图)
· 目的
  · 说明数据/对象结构
  · 指定快照



Package Diagrams

- ***Package:*** Independent unit of functionality that consists of a collection of related classes and/or other subsystems.
- Offer interfaces and uses interfaces provided by other subsystems.
- In the UML, packages or subsystems are represented as folders:

包:由一系列相关的类和/或其他子系统组成的独立的功能单元。
•提供接口并使用其他子系统提供的接口。
•在UML中，包或子系统用文件夹表示

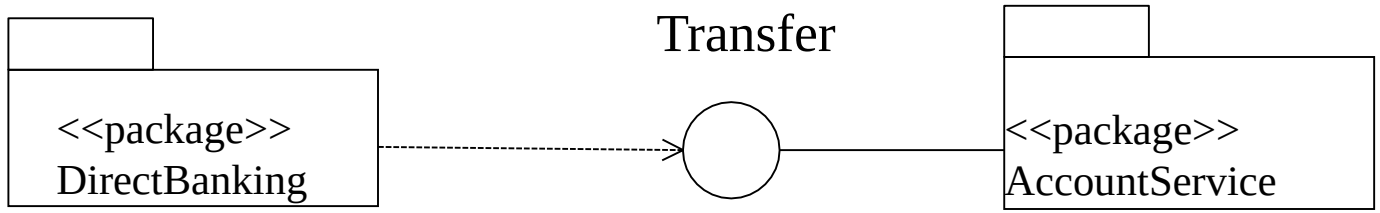


- **Dependency Relationships**: provides and uses relationships
- **Uses** relationship, shown as a dashed arrow to the used interface.
- **Provides** relationship, shown as a straight line to the provided interface.

· 依赖关系:提供和使用关系
· 使用关系，显示为使用接口的虚线箭头。
· 提供关系，显示为与所提供接口的直线

包A依赖于包B，意味着A中的一个或多个类与B中的一个或多个公共类发起通信:A被称为客户端，B称为供应商

- Package A is dependent on package B implies that one or more classes in A initiates communication with one or more public classes in B: A is called the **client** and B the **supplier**.



3.Interactions

Use Case Realization

用例的功能通过描述所涉及的场景来定义

•the functionality of a use case is defined by describing the scenarios involved.

场景是用例的实例:它是用例事件流的一个路径

÷a scenario is an instance of a use case: it is one path through the flow of events for the use case.

每个用例都是场景的web:主要场景(用例的正常流)和次要场景(用例的假设逻辑)

÷***each use case is a web of scenarios***: primary scenarios (the normal flow for the use case) and secondary scenarios (the what-if logic of the use case).

场景帮助识别对象、类和对象交互（执行用例指定的功能部分所需的）

÷scenarios help identify the objects, the classes, and the object interactions needed to carry out a piece of the functionality specified by the use case.

用例的事件流是在文本中捕获的，而场景是在交互图中捕获的

•the flow of events for a use case is captured in text, whereas scenarios are captured in interaction diagrams.

•Main types of interaction diagrams:

交互图的主要类型:
·序列图
·通信图

÷***sequence diagrams***

÷***communication diagrams***

Sequence Diagram

显示按时间顺序排列的对象相互作用

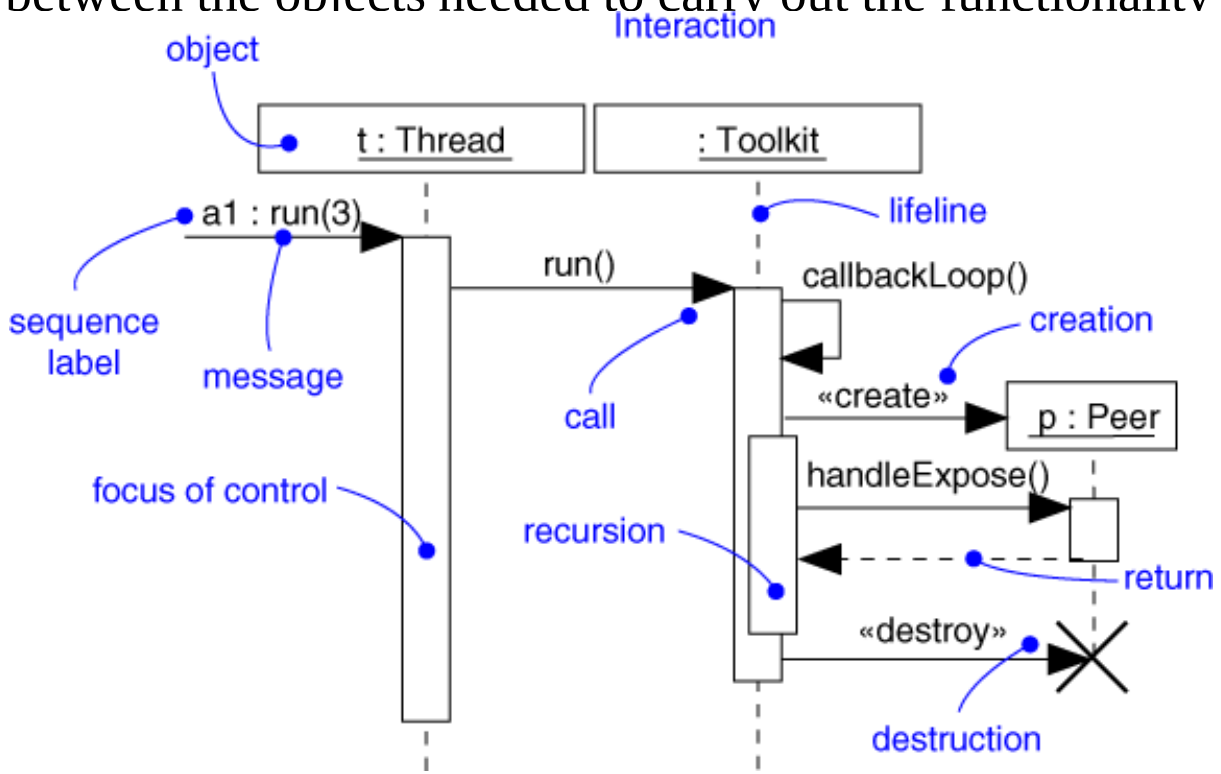
- Shows object interactions ***arranged in time sequence***

目的
-控制流程模型
-说明典型场景

- Purpose
  - Model flow of control
  - Illustrate typical scenarios

描述场景中涉及的对象和类，以及执行场景功能所需的对象之间交换的消息序列

- Depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.



Communication Diagram

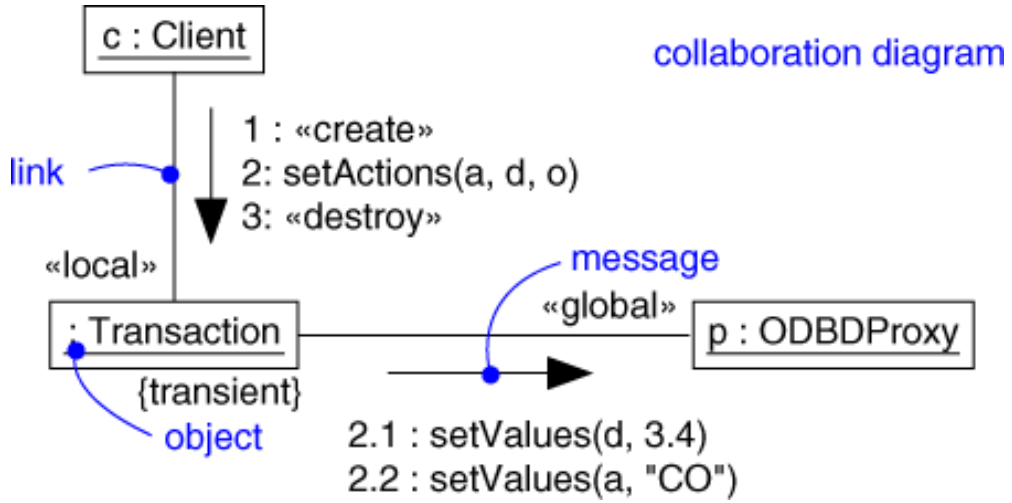
显示组织在对象周围的对象交互以及它们彼此之间的链接(安排强调结构化组织)

- Shows object interactions organized around the objects and their links to each other (Arranged to ***emphasize structural organization***)

目的
-控制流程模型
-说明对象结构和控制的协调

- Purpose
  - Model flow of control
  - Illustrate coordination of object structure and control
- Represent an alternate way to describe a scenario

表示描述场景的另一种方法



通信图包括:
-绘制成矩形的对象
-对象之间的链接，以线的形式显示
-显示为文本和从客户指向供应商的箭头的消息

- A communication diagram contains:
  - -objects drawn as rectangles
  - -links between objects shown as lines connecting the linked objects
  - -messages shown as text and an arrow that points from the client to the supplier.

4.Dynamic Behavior

状态转换图

State Transition Diagram

•Use cases and scenarios provide a way to describe system behavior, that is the interaction between objects in the system.

用例和场景提供了一种描述系统行为的方法，即系统中对象之间的交互

状态转换图允许对单个对象内部的行为进行建模

•A state transition diagram allows the modeling of the behavior inside a single object.

÷It ***shows the events or messages*** that cause a ***transition*** from ***one state to another***, and the actions that result from a state change.

它显示导致从一种状态转换到另一种状态的事件或消息，以及状态更改导致的操作

÷It is ***created only for classes with significant dynamic behavior***, like control classes.

它仅为具有重要动态行为的类(如控制类)创建

State:

•a condition during the life of an object when it ***satisfies some condition, performs some action***, or ***waits for an event***

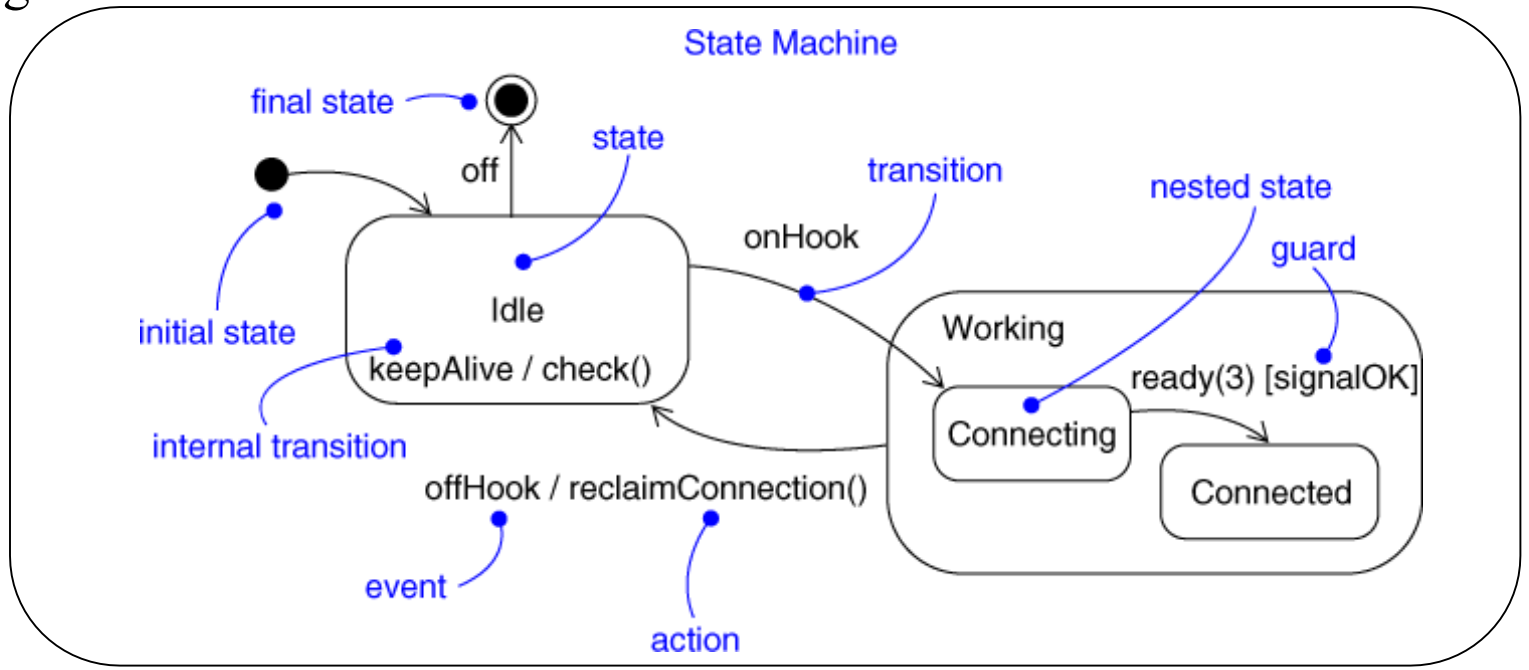
·在生命周期中满足某些条件、执行某些操作或等待事件时的一种条件

•found by examining the attributes and links defined for the object

·通过检查对象定义的属性和链接来找到

•represented as a rectangle with rounded corners

·表示为带有圆角的矩形



- **Transitions**:

•represents a change from an originating state to a successor state (that may be the same as the originating state).

·表示从原始状态到后继状态(可能与原始状态相同)的更改。

- may have an action and/or a guard condition associated with it, and may also trigger an event.

·可能有一个动作和/或与之相关的保护条件，也可能触发一个事件

Activity Diagram

•Captures dynamic behavior (activity-oriented)

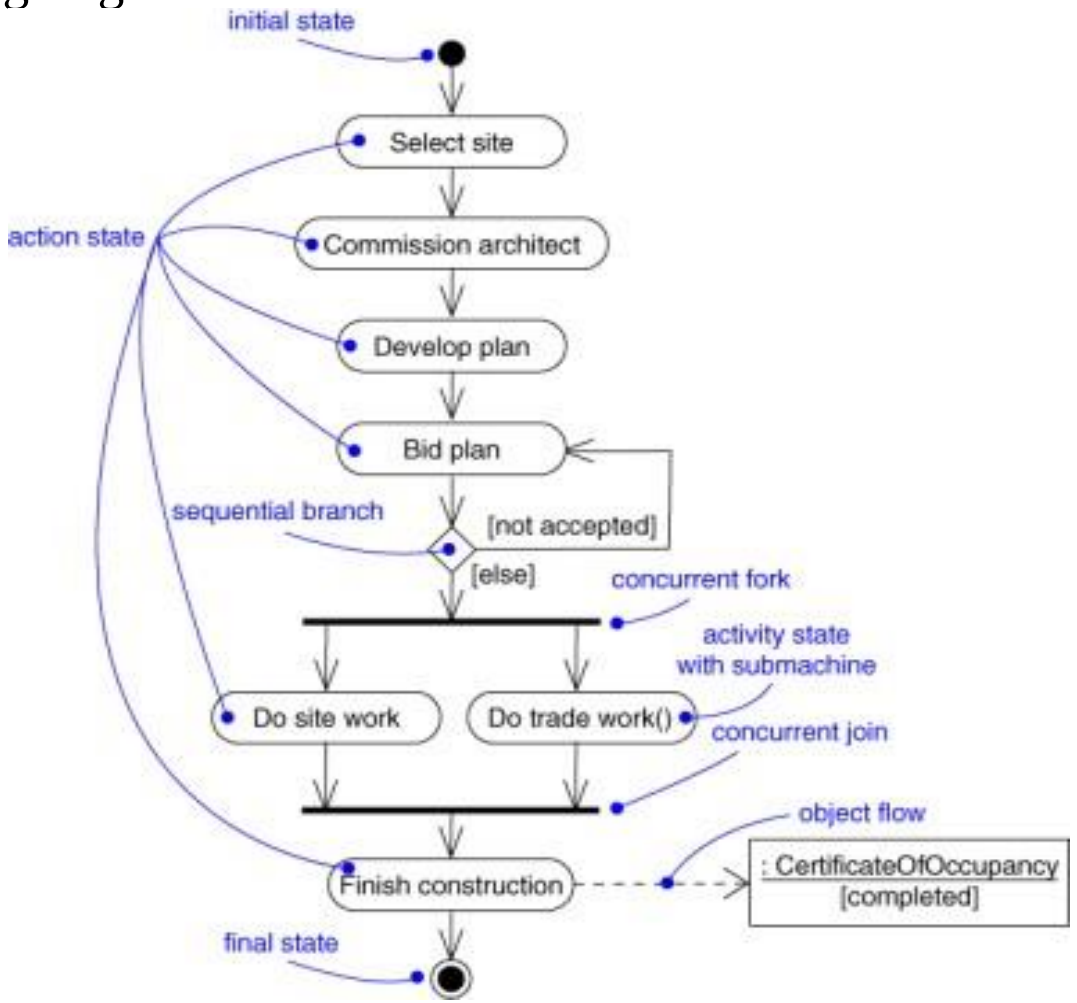
•Behavior that occurs within the state is called an **activity**: starts when the state is entered and either completes or is interrupted by an outgoing transition.

•Purpose

-Model business workflow

-Model operations

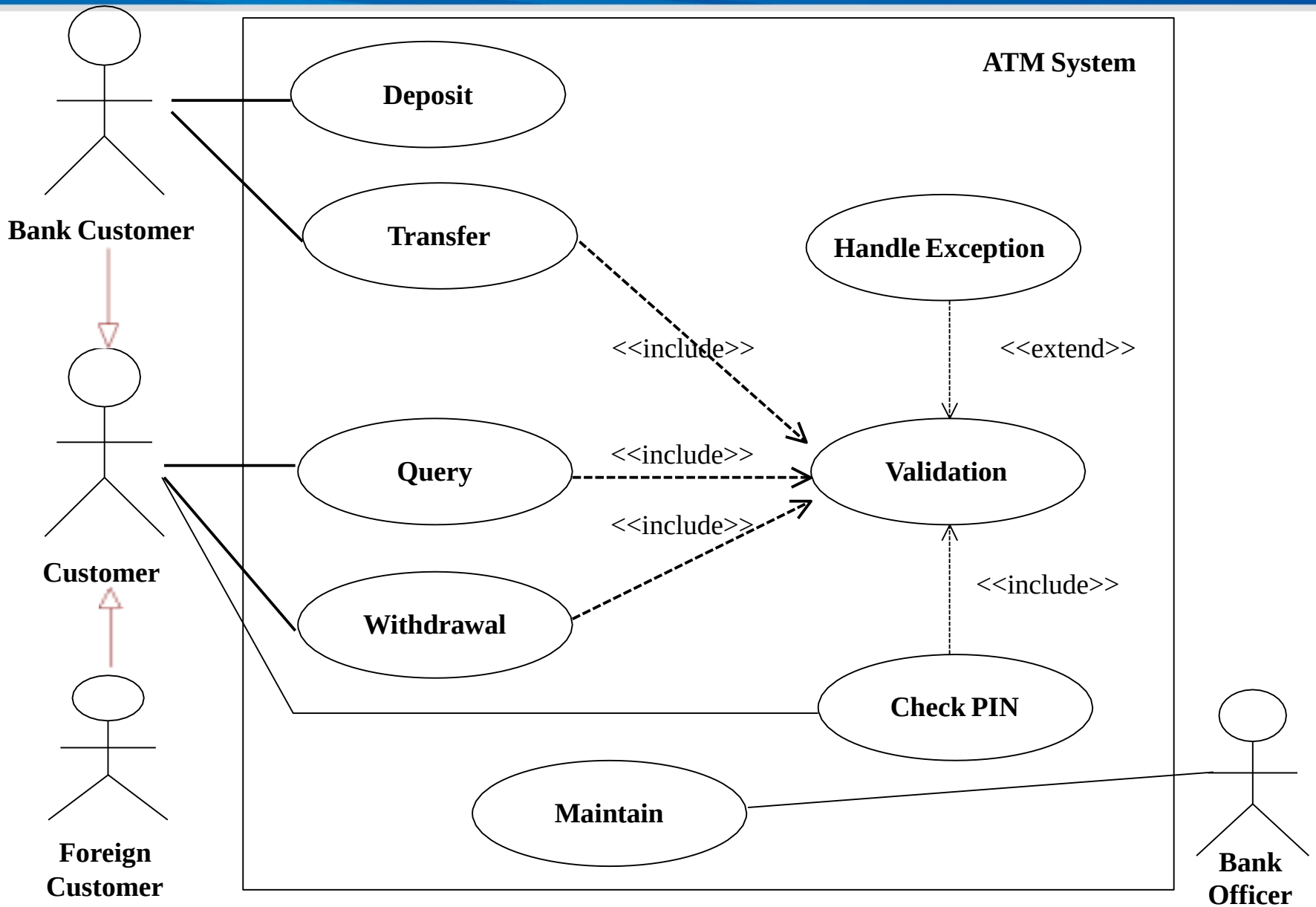
· 捕获动态行为(面向活动)
· 在状态中发生的行为称为活动:当状态进入时开始，完成或被传出的转换中断。
· 目的
-建模业务流程
-模型操作



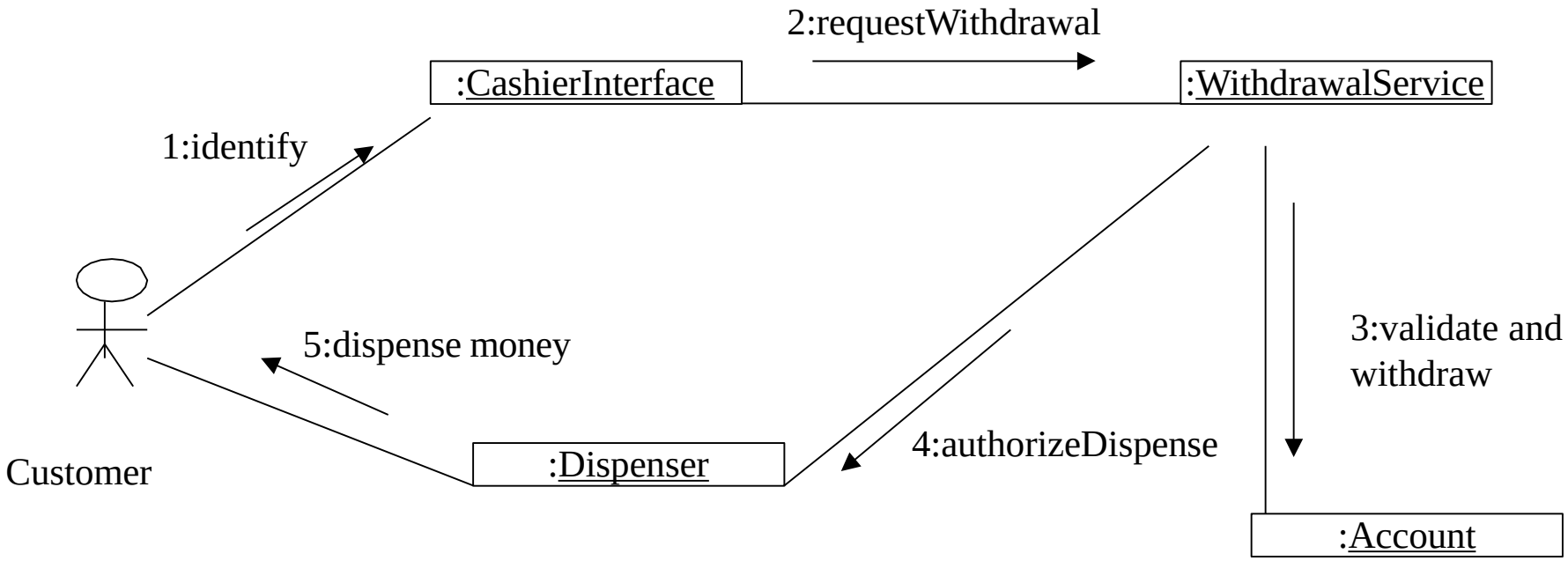
5.Example: Logical View for the ATM

从用例视图中定义的架构上重要的用例派生出来的

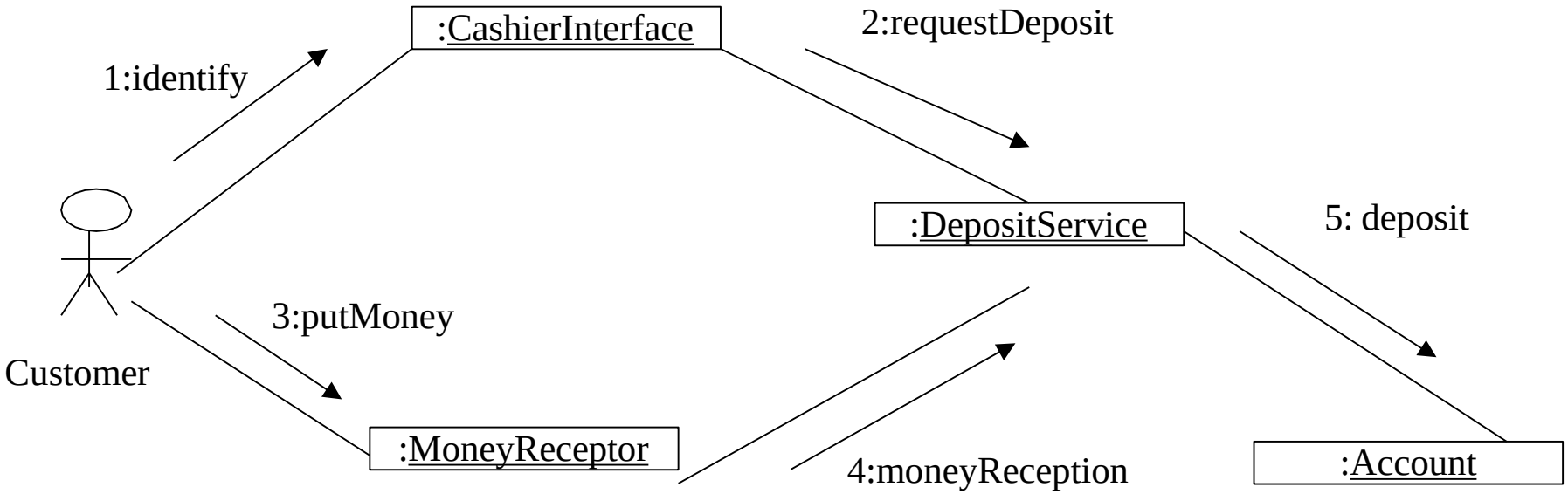
- Is derived from architecturally significant use cases defined in the use case view



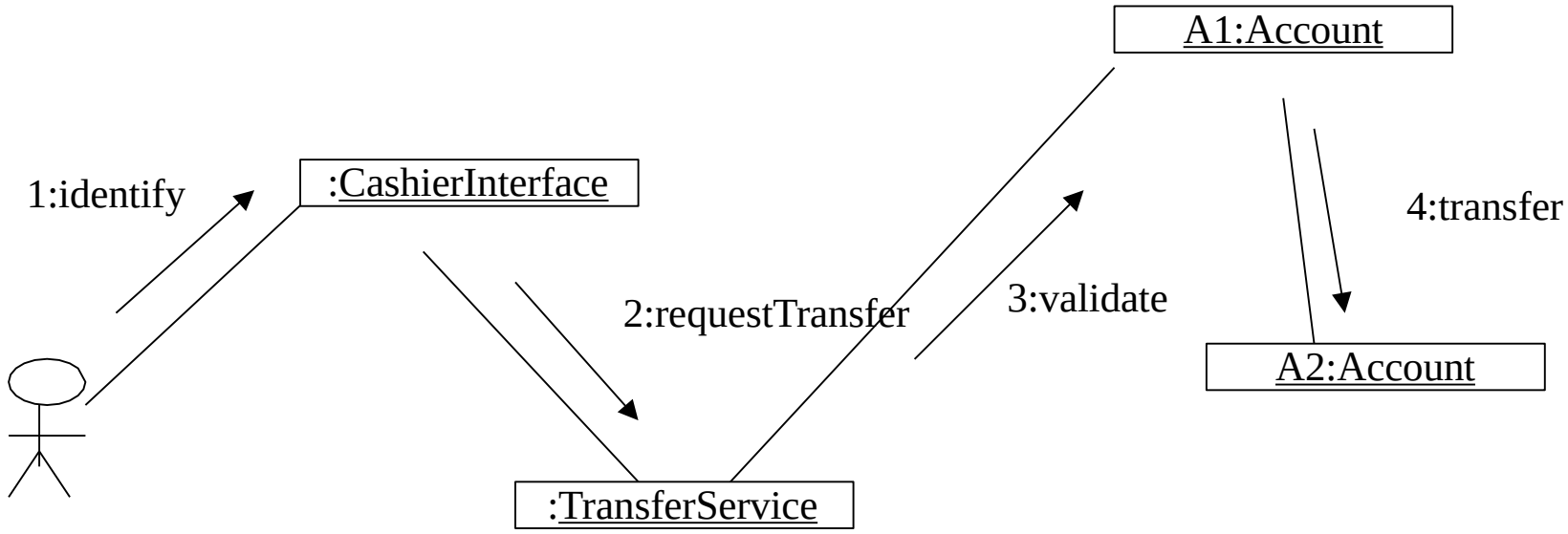
Communication Diagram: Withdraw Money Use case



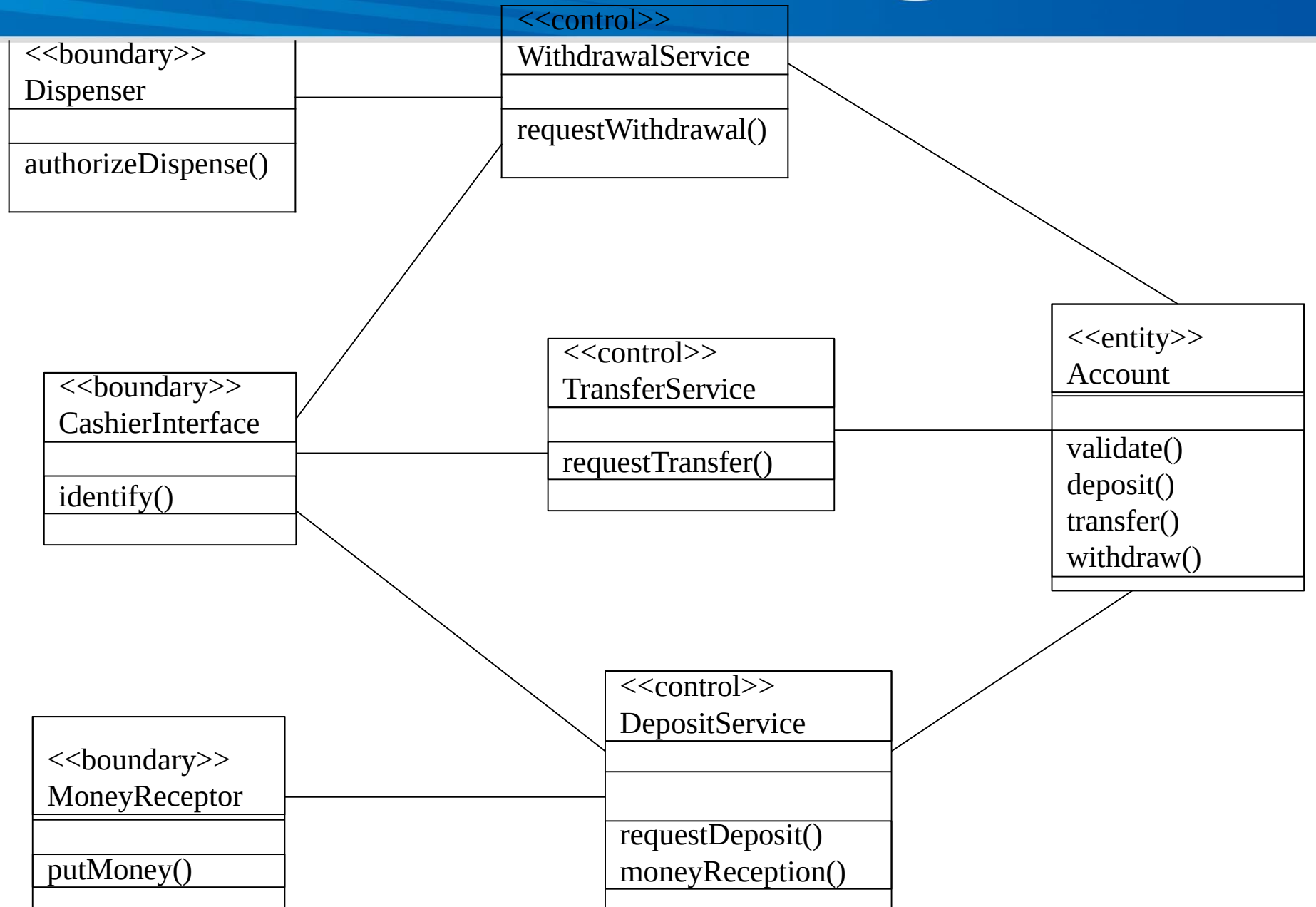
Communication Diagram: Deposit Use Case



Communication Diagram: Transfer Use Case

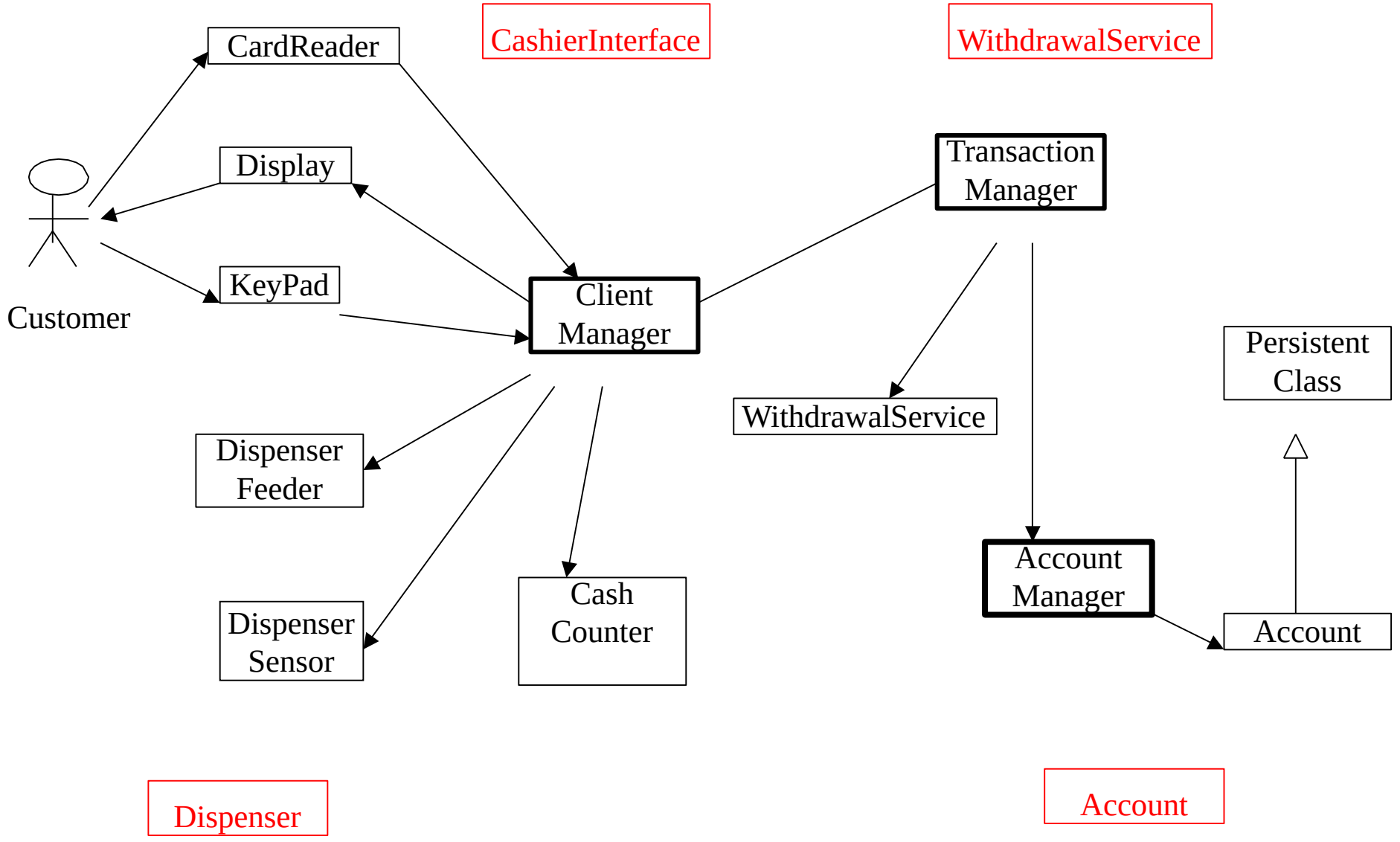


Class Diagram



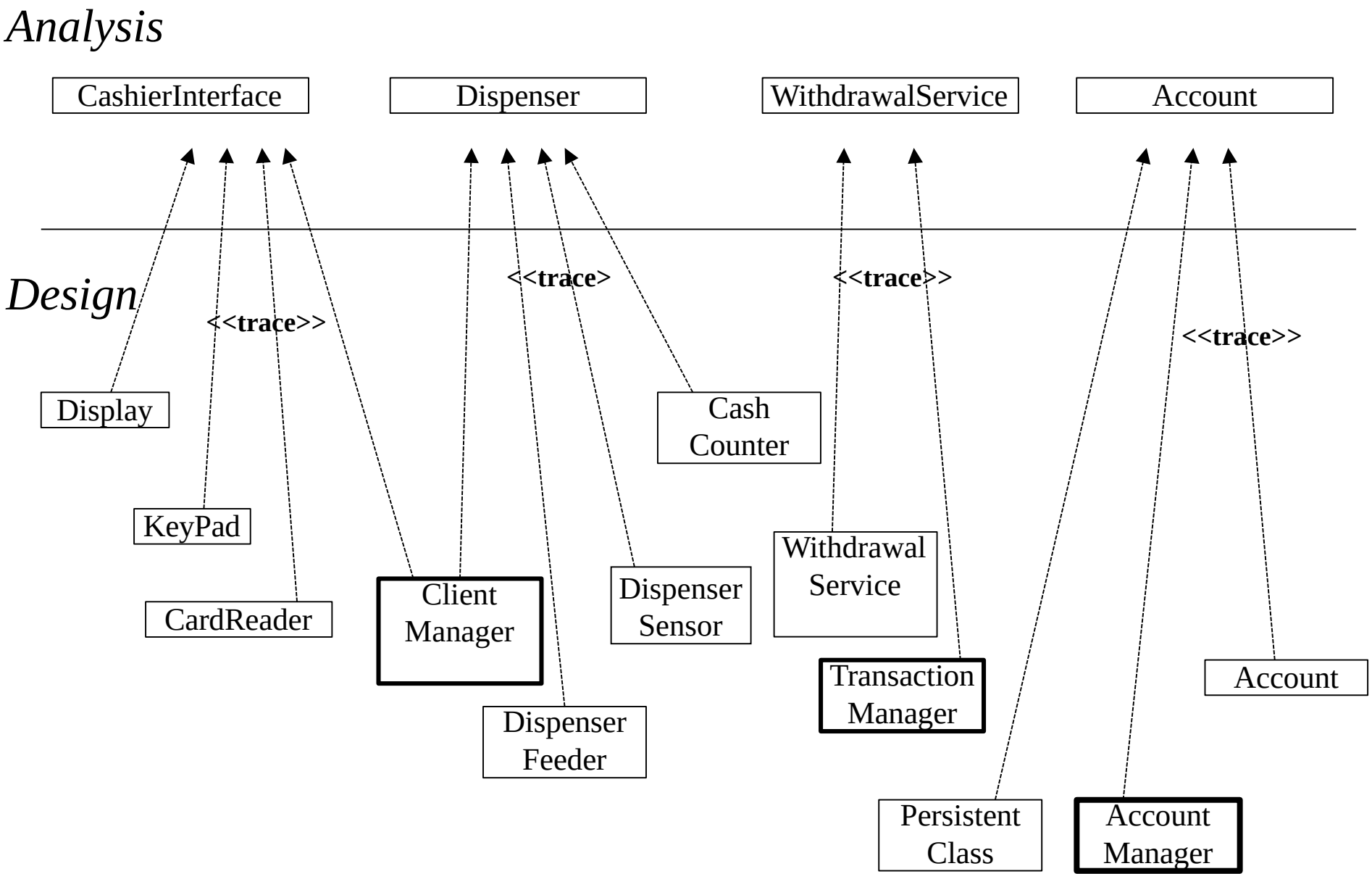
(Refined) Class diagram providing a view of the classes involved in withdraw Money use case (design model)

(改进的)类图，提供取款用例(设计模型)中涉及的类的视图



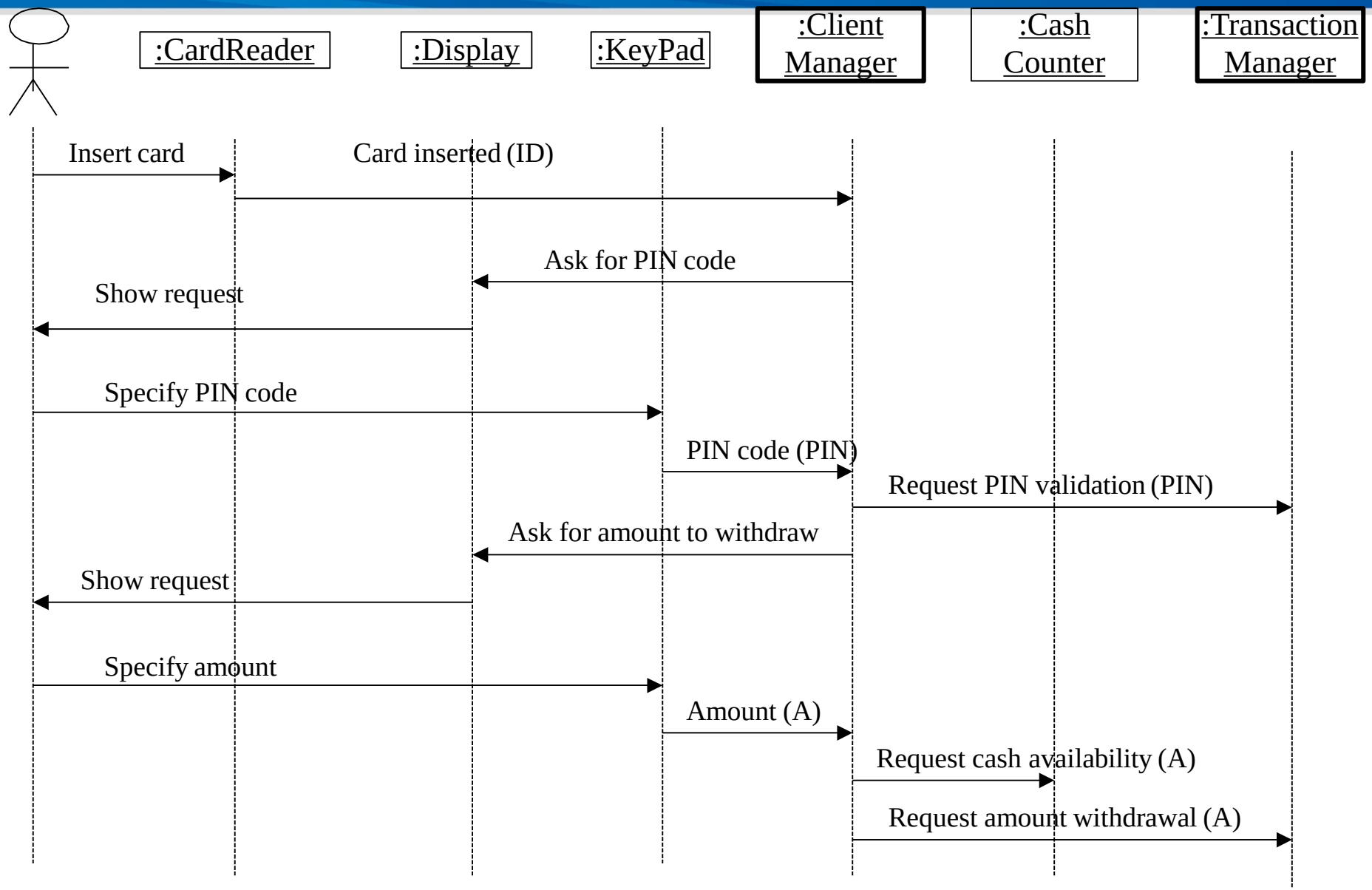
Traceability (Withdraw use case)

追溯性(提款用例)

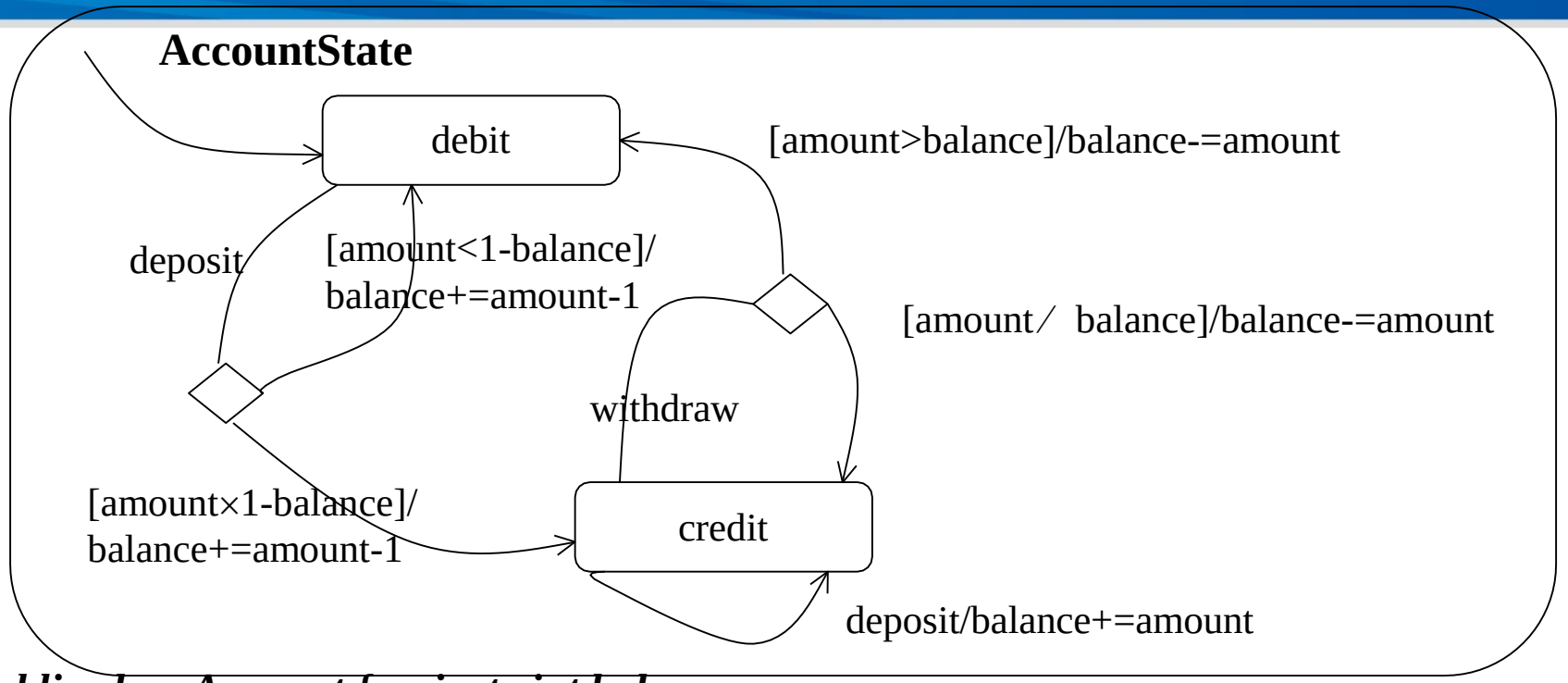


A Scenario of the Withdraw Money Use Case (Design Model)

取款用例的场景(设计模型)

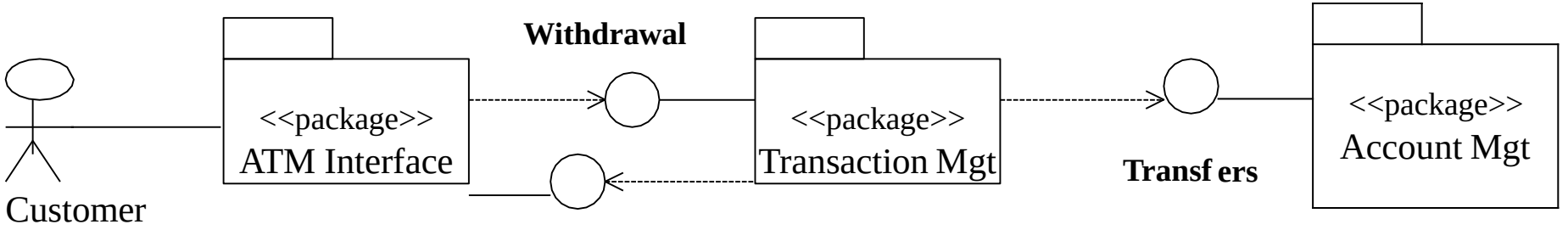


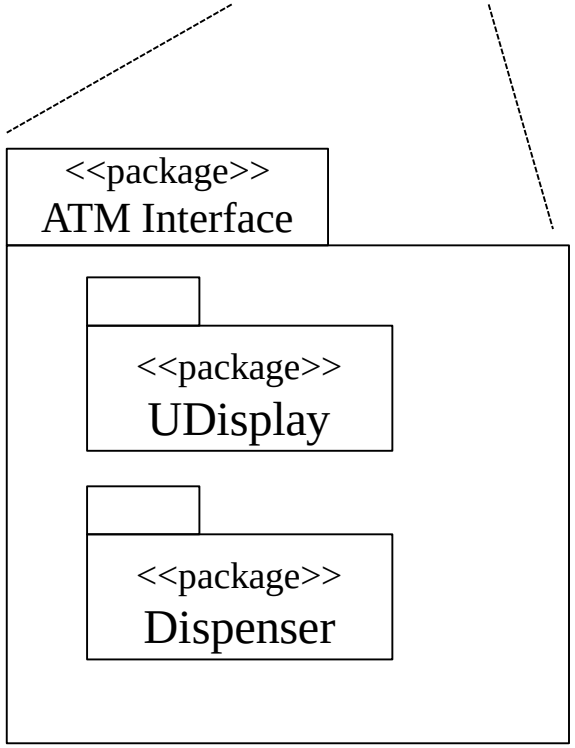
State Transition Diagram for Class Account




```
public class Account {  private int balance;
public void deposit (int amount) {
if (balance × 0) balance = balance + amount;
else balance = balance + amount – 1; // transaction fee
}
public void withdraw (amount) {
if (balance × 0) balance = balance – amount;
}}
```

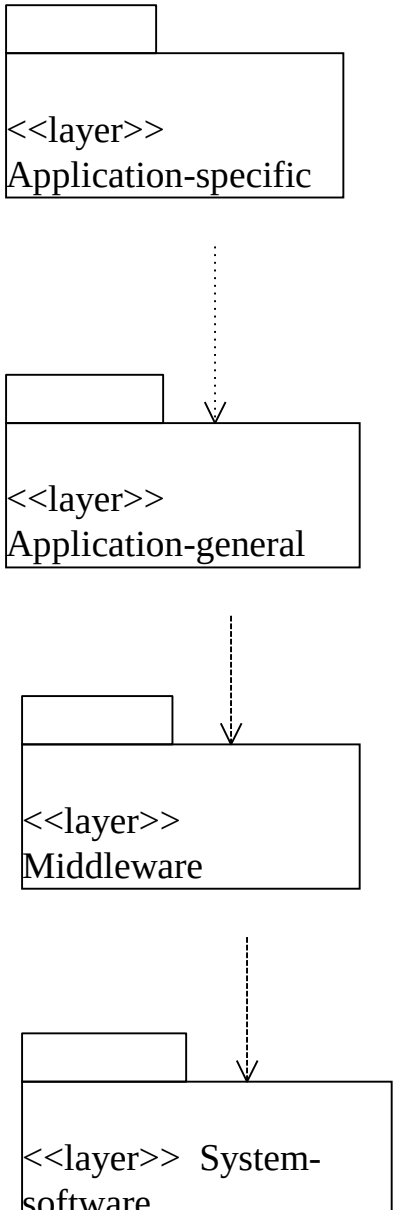
Package Diagram



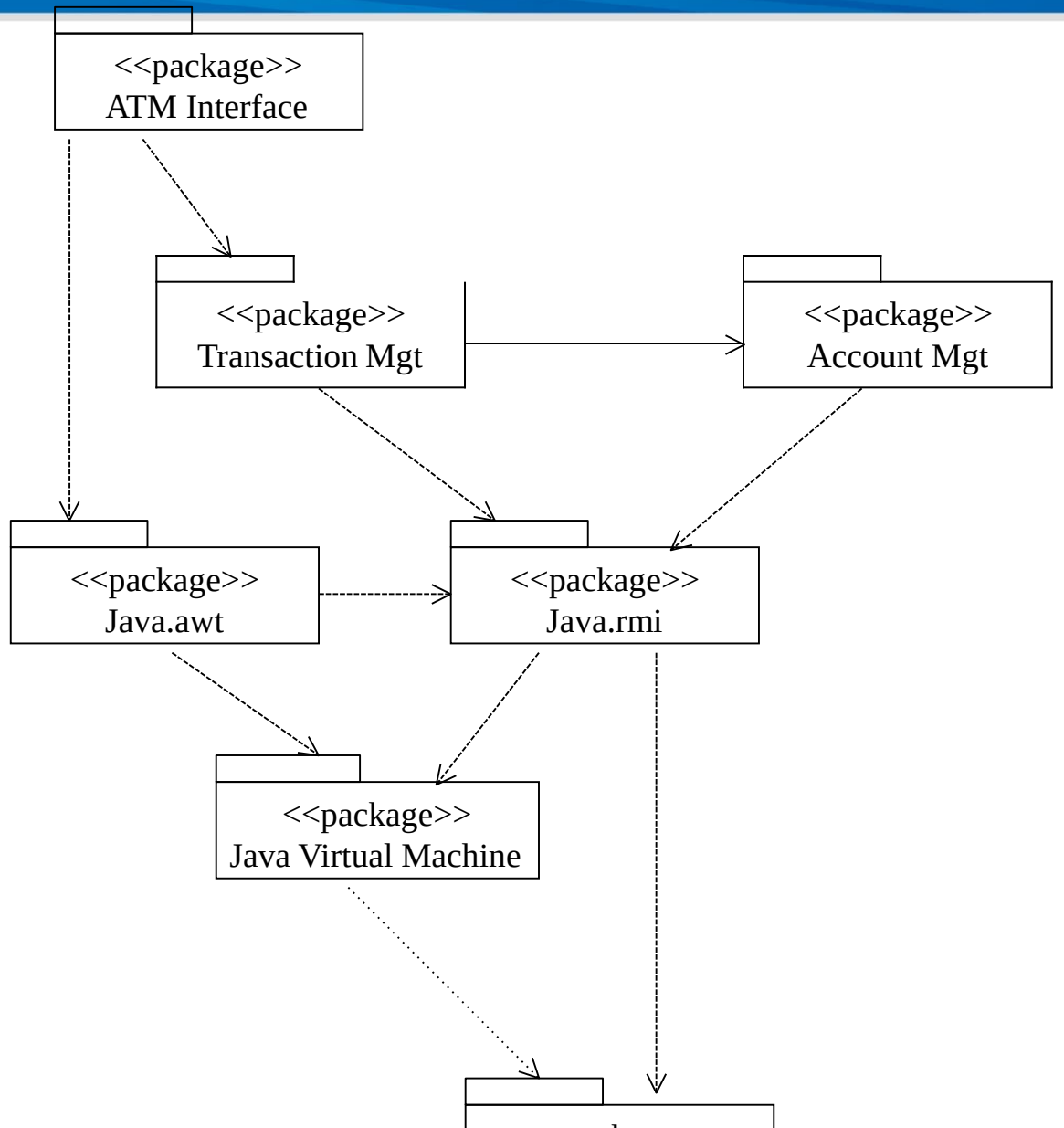


Classes	Packages
CardReader, Display, KeyPad, ClientMgr	UDisplay/ATM Interface
DispenserFeeder, DispenserSensor, CashCounter	Dispenser/ATM Interface
WithdrawalService, TransactionMgr	TransactionMgt
Account, PersistentClass, AccountMgr	AccountMgt

Structuring Using Layer Architectural Pattern



Packages	**Layers**
ATM Interface	Application-specific
Transaction Mgt, Account Mgt	Application-general
	Middleware
	System-software

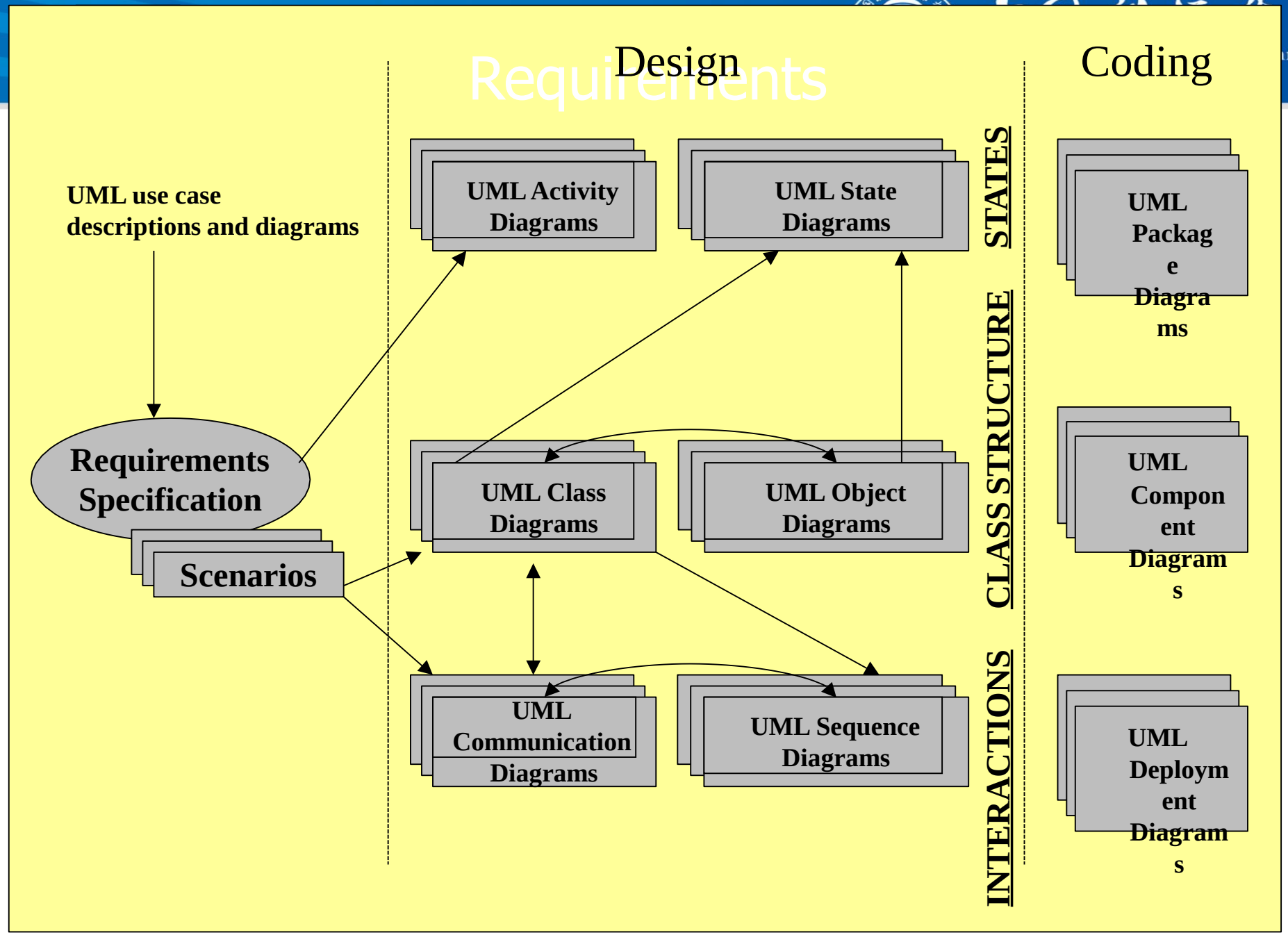


Application-specific layer

Application-general layer

Middleware layer

System-software layer


Requirements
Design
Coding
UML use case descriptions and diagrams
Requirements Specification
Scenarios
UML Activity Diagrams
UML State Diagrams
UML Class Diagrams
UML Object Diagrams
UML Communication Diagrams
UML Sequence Diagrams
STATES
CLASS STRUCTURE
INTERACTIONS
UML Package Diagrams
UML Component Diagrams
UML Deployment Diagrams

Implementation, Process, and Deployment Views

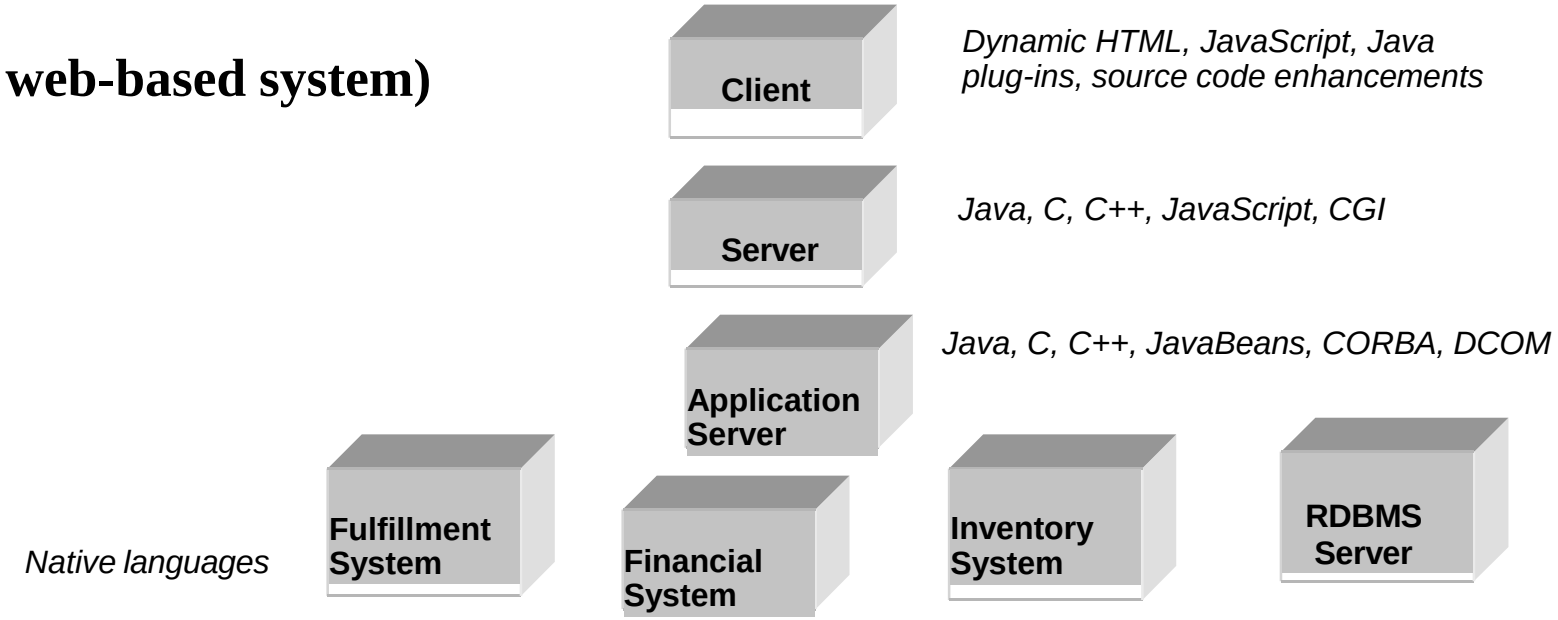
1. Motivation
2. Process View
3. Implementation View
4. Deployment View
5. ATM Example

1.Motivation

复杂的软件系统涉及广泛的功能，部署在独立的处理节点上，涉及多种语言、平台和技术

÷Complex software systems involve a wide range of functionality, deployed on independent processing nodes, involving a wide variety of languages, platforms, and technologies.

÷**Example: (a complex web-based system)**



过程、实现和部署视图通过以下方式捕获这种复杂性：
· 描述运行时实体：形成系统并发和同步的线程和进程。
· 描述源和可执行组件，它们的组织，和它们的依赖关系。
· 描述硬件拓扑，并将软件组件映射到处理节点
· 从构建过程

The process, implementation, and deployment views capture this complexity by:

-Describing runtime entities: the threads and processes that form the system’s concurrency and synchronization.
-Describing source and executable components, their organization, and their dependencies.
-Describing hardware topology and mapping software components to processing nodes
-Describing build procedures

2.Process View

从逻辑视图派生出软件产品底层的并发和同步机制

Derives from the Logical view the concurrency and synchronization mechanisms underlying the software product.

Overview

由形成系统并发和同步机制的进程和线程以及它们之间的交互组成

•Consists of the ***processes*** and ***threads*** that form the system’s ***concurrency*** and ***synchronization*** mechanisms, as well as their ***interactions***

解决的问题包括:
-并发性和并行性(如同步、死锁等)
-系统启动和关闭
-系统的性能、可伸缩性和吞吐量

•**Addresses issues** such as:

-Concurrency and parallelism (e.g., synchronization, deadlocks etc.)

-System startup and shutdown

-Performance, scalability, and throughput of the system.

使用关注主动类和对象的类、交互和状态转换图捕获

•Is captured using ***class, interaction and state transition diagrams*** with a ***focus on active classes and objects***.

Processes and Threads

- ***Process***: a heavyweight flow of control that can execute independently and concurrently with other processes.

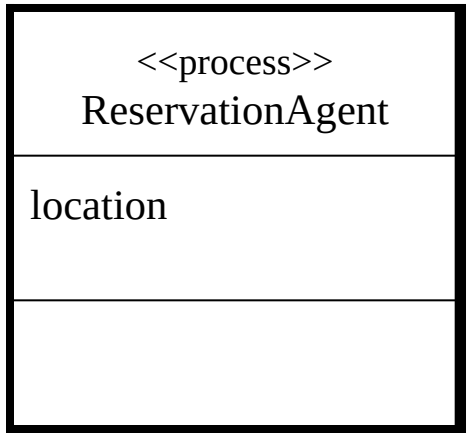
进程:重量级的控制流，可以独立地和其他进程并发地执行

- ***Thread***: a lightweight flow that can execute independently and concurrently with other threads within the same process.

线程:一个轻量级的流，可以独立地和同一进程中的其他线程并发地执行

独立的控制流(如线程和进程)被建模为活动对象。活动对象是活动类的实例。您可以使用原型进程指定一个进程，也可以使用原型线程指定一个线程

- Independent flows of control such as ***threads*** and ***processes*** are modeled as *active objects*. An active object is an instance of an *active class*. You may specify a process using the stereotype *process* and a thread using the stereotype *thread*.



活动对象是拥有进程或线程并可以发起控制活动的对象。
在图形上，活动类用粗线表示。

An ***Active Object*** is an object that owns a process or thread and can initiate control activity.

-Graphically an Active Class is represented as a class with thick lines.

普通类之所以称为被动类，是因为它们不能独立地初始化控制

Plain classes are called ***passive*** because they cannot independently initiate control.

Communication

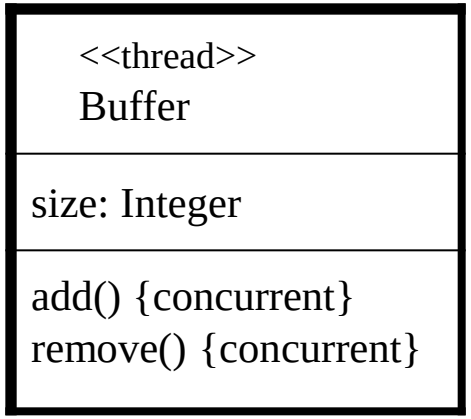
- You model interprocess communication using interaction diagrams:
  - ***Synchronous communication***
  - ***Asynchronous communication***
- Two approaches: *RPC* (synchronous) and *message passing* (asynchronous)

· 您使用交互图为进程间通信建模:
-同步通信
-异步通信
· 两种方法:RPC(同步)和消息传递(异步)

Synchronization

- Modeled by adding constraints to the operations; there are three kinds of synchronization:
  - Sequential
  - Guarded
  - Concurrent

同步
· 通过向操作添加约束来建模;有三种同步:
-顺序
-保护
-并发



sequential

调用方必须在对象外部进行协调，以便在同一时间对象中只有一个流。如果同时发生调用，则无法保证系统的语义和完整性

÷Callers must coordinate outside the object so that only one flow is in the object at a time. If simultaneous calls occur, then the semantics and integrity of the system cannot be guaranteed.

guarded

在存在多个控制流的情况下，通过对所有对象保护操作的所有调用进行顺序化，可以保证对象的语义和完整性。实际上，一次只能对对象调用一个操作，从而将其简化为顺序语义

÷The semantics and integrity of the object is guaranteed in the presence of multiple flows of control by sequentializing all calls to all objects' guarded operations. In effect, exactly one operation at a time can be invoked on the object, reducing this to sequential semantics.

concurrent

÷Multiple calls from concurrent threads may occur simultaneously to one ***Instance*** (on any concurrent ***Operations***). All of them may proceed concurrently with correct semantics.

The semantic and integrity of the object is guaranteed in the presence of multiple flows of control by treating the operation as atomic.

来自并发线程的多个调用可能同时发生在一个实例上(在任何并发操作上)。它们都可以使用正确的语义并发进行。在存在多个控制流时，通过将操作视为原子操作，可以保证对象的语义和完整性

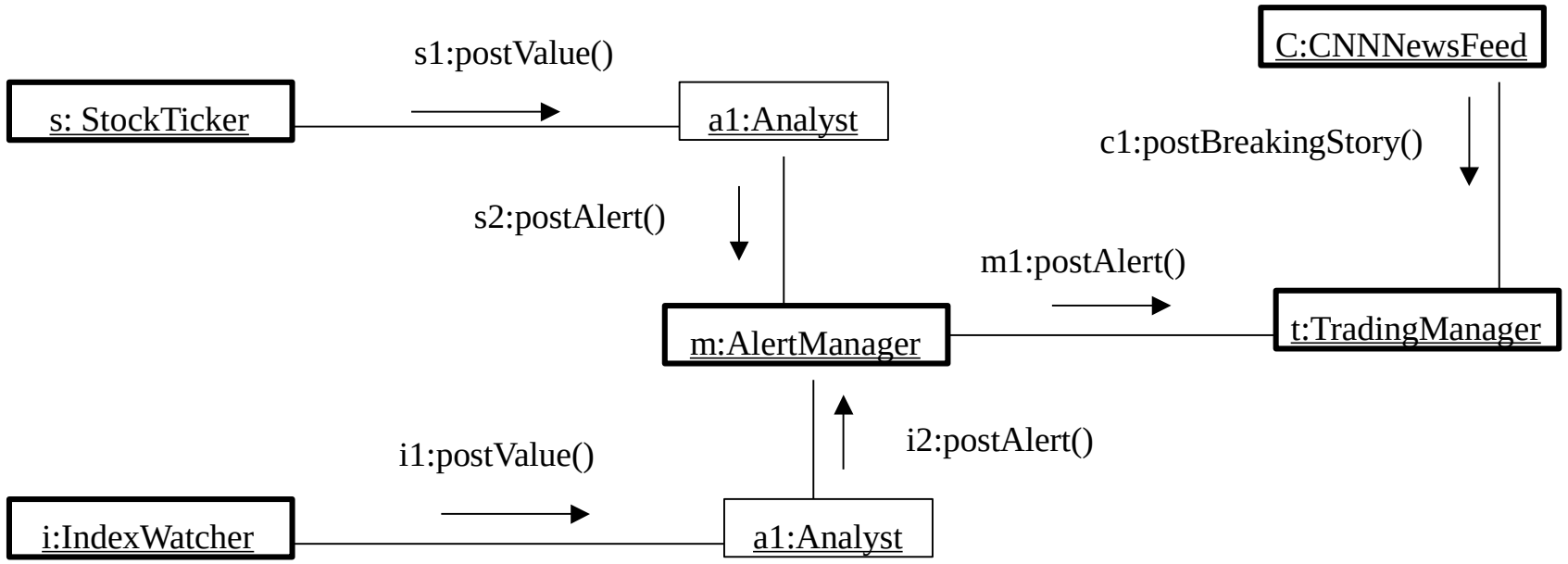
Note: Java use the Synchronized modifier, which maps to UML Concurrent property.

Java使用Synchronized修饰符，它映射到UML并发属性

Example

Consider a trading system, where trading decisions are based on information collected from three different sources: a stock ticker, an index watcher, and a CNNNewsFeed. Information from the stock ticker and the index watcher are first analyzed and then forwarded to the trading manager via an alert manager. The CNNNewsFeed communicates directly with the Trading manager.

以一个交易系统为例，该系统的交易决策基于从三个不同来源收集的信息:股票行情、指数观察者和CNNNewsFeed。来自股票行情和指数观察者的信息首先被分析，然后通过一个警报管理器转发给交易经理。CNNNewsFeed直接与交易经理沟通



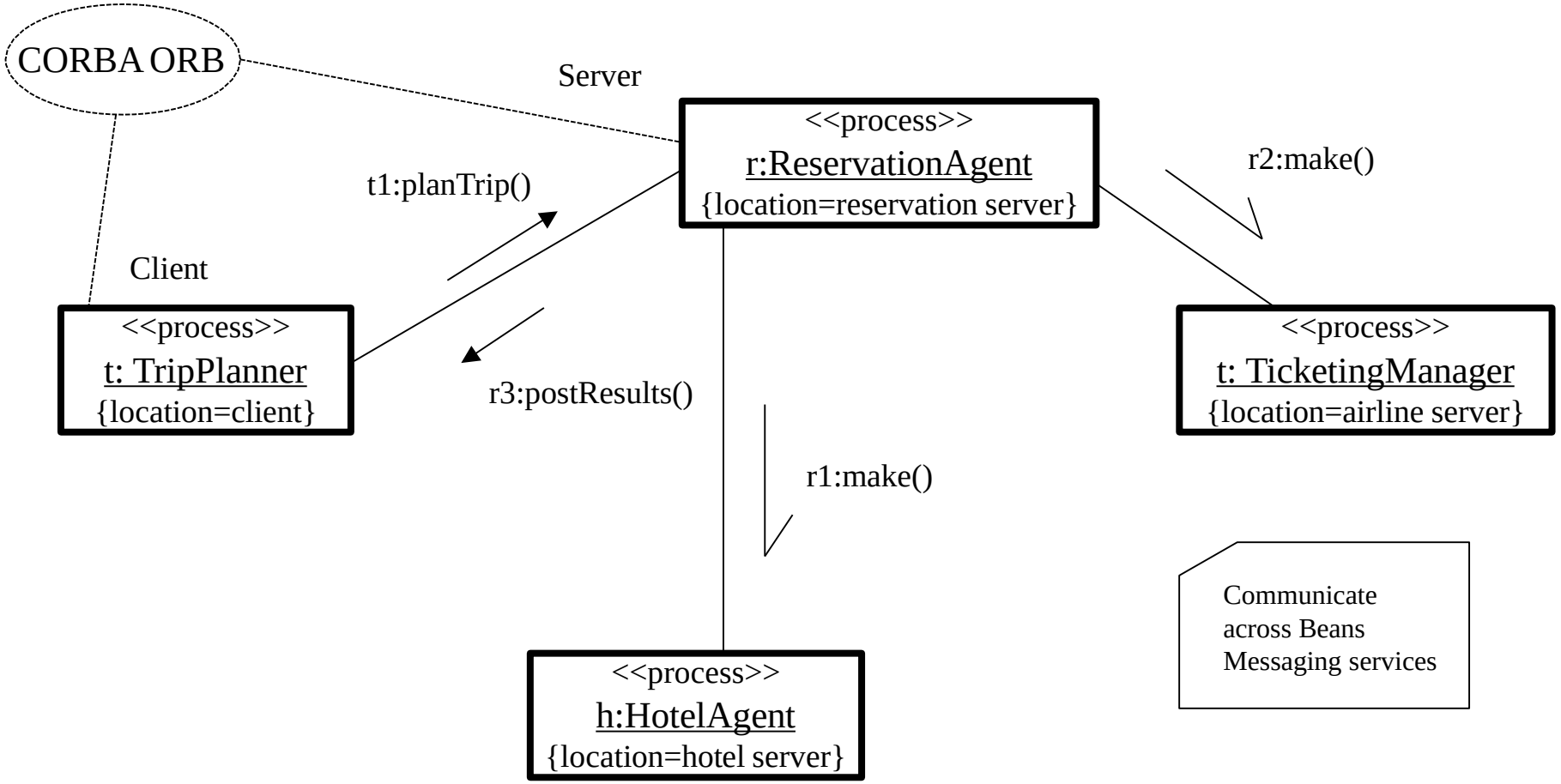
诸如此类的交互图在帮助您可视化两个控制流可能在何处交叉路径，以及因此在何处必须特别注意通信和同步问题方面非常有用

- Interaction diagrams such as these are useful in helping you to visualize where two flows of control might cross paths, and therefore where special attention must be paid to communication and synchronization problems.

Example

Consider a trip planning service (e.g., expedia) that is used by travelers to identify and book all at once the best deal in terms of flight, hotel, car rental etc. Model a basic scenario where a customer uses the system to book flight and hotel room by highlighting the concurrency and synchronization involved.

考虑一个旅行计划服务(例如expedia)，旅行者可以利用它在航班、酒店、汽车租赁等方面一次性确定并预订最优惠的价格。建模一个基本场景，其中客户使用系统通过突出所涉及的并发性和同步性来预订机票和酒店房间



3.Implementation View

Concentrates on taking the Logical view and dividing the logical entities into actual software components. 专注于获取逻辑视图并将逻辑实体划分为实际的软件组件

Overview

-Describes the ***organization of static software modules*** (source code, data files, executables, documentation etc.) in the development environment in terms of:

描述在开发环境中的静态软件模块的组织方式(源代码、数据文件、可执行文件、文档等)，具体如下:
•包装和分层
•配置管理(所有权、发布策略等)

•***Packaging and layering***

•***Configuration management*** (ownership, release strategy etc.)

-Are modeled using *UML Component Diagrams*.

使用UML组件图建模
•UML组件是系统的物理和可替换的部分，符合并提供一组接口的实现

•UML components are physical and replaceable parts of a system that conform to and provide the realization of a set of interfaces

Three kinds of components:

三种组件：
-部署组件:构成可执行系统所需且足够的组件，如dll、可执行文件等。
-工作产品组件:开发过程遗留的源代码文件、数据文件等
-执行组件:创建作为执行系统的结果，如COM+，它是从DLL实例化的

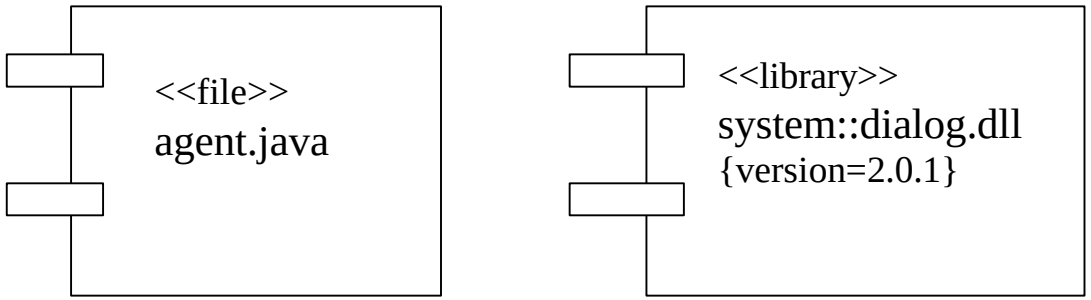
-***Deployment components***: components necessary and sufficient to form an executable system, such as DLLs, executables etc.

-***Work product components***: residue of development process such as source code files, data files etc.

-***Execution components***: created as a consequence of executing system such as COM+ which is instantiated from a DLL.

UML Components

Notation



Standard Component Stereotypes

- ***-executable***: a component that may be executed on a node
- ***-library***: a static or dynamic object library
- ***-table***: a component that represents a database table
- ***-file***: a component that represents a document source code or data
- ***-document***: a component that represents a document

标准组件的原型
-可执行的：可以在节点上执行的组件
-库：静态或动态对象库
-表：表示数据库表的组件
-文件：表示文档源代码或数据的组件
-文档：表示文档的组件

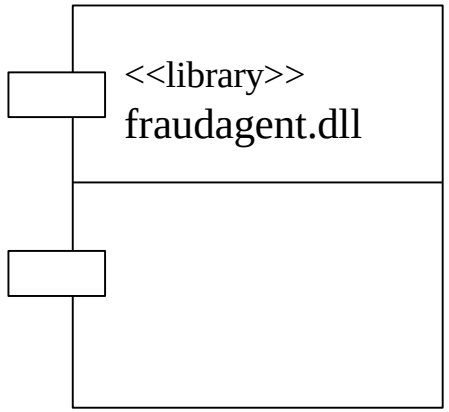
Components and Classes

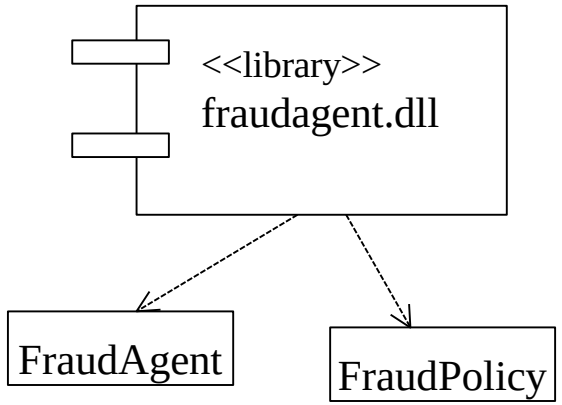
- There are significant differences between components and classes:
  - ***classes represent logical abstractions***
  - ***components represent physical entities that live on nodes***

组件和类之间有显著的区别：
· 类代表逻辑抽象
· 组件表示节点上的物理实体

- A component is a physical element that provides the implementation of logical elements such as classes (that is shown using a dependency relationship)

组件是一个物理元素，它提供了诸如类(使用依赖关系显示)等逻辑元素的实现





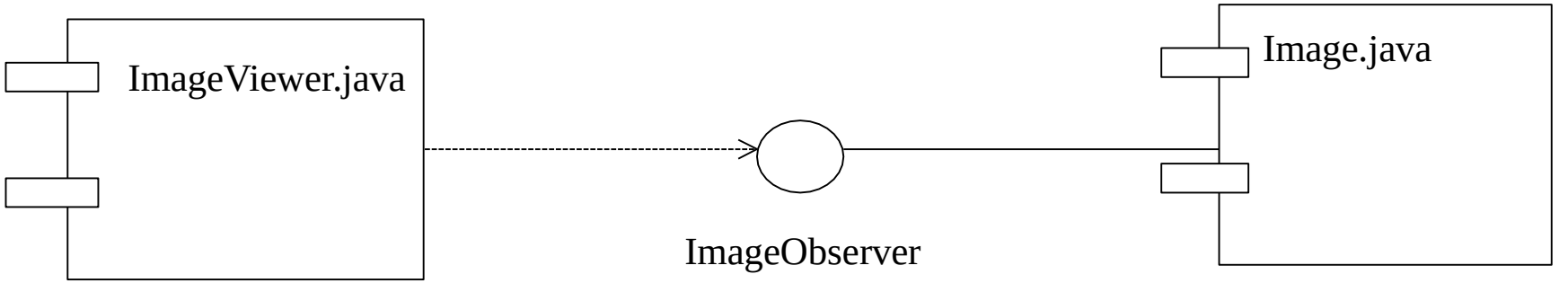
Component Interfaces

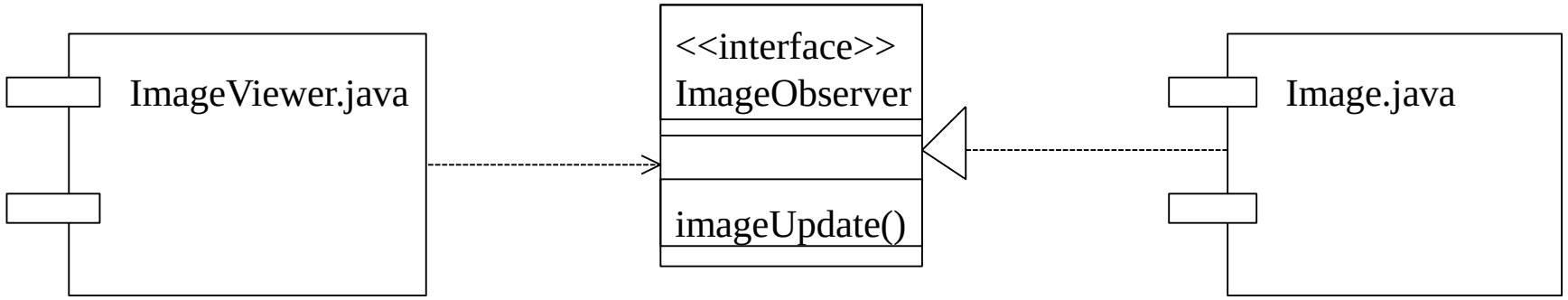
接口是用于指定类或组件的服务的操作集合

An interface is a collection of operations that are used to specify a *service of a class or a component*.

· 接口提供了将组件绑定在一起的粘合剂
· 组件可以提供接口(实现)的实现，也可以访问其服务(依赖关系)

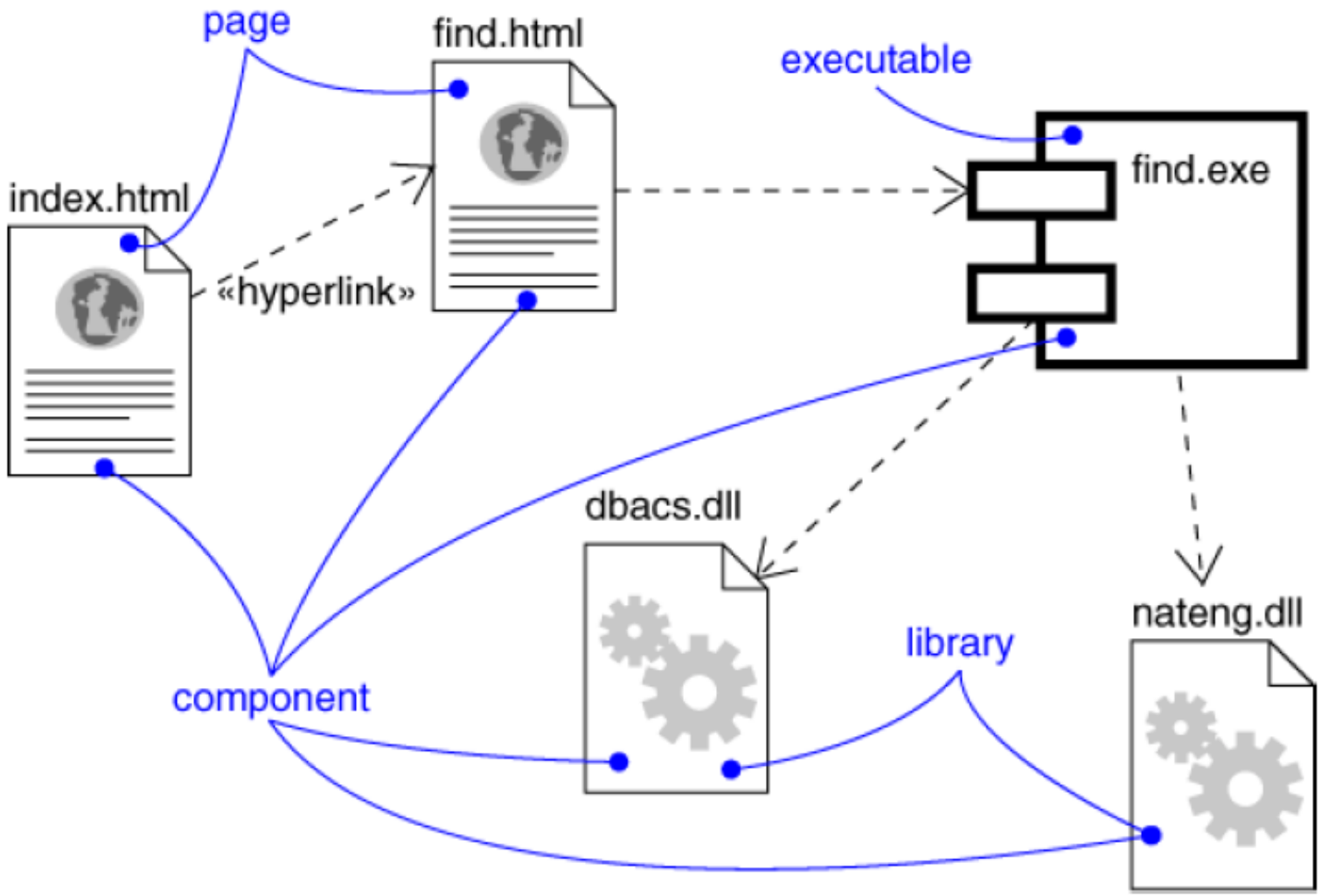
- *Interfaces* provide the glue that binds components together
- A component may provide the implementation of an interface (realization) or may access its services (dependency).



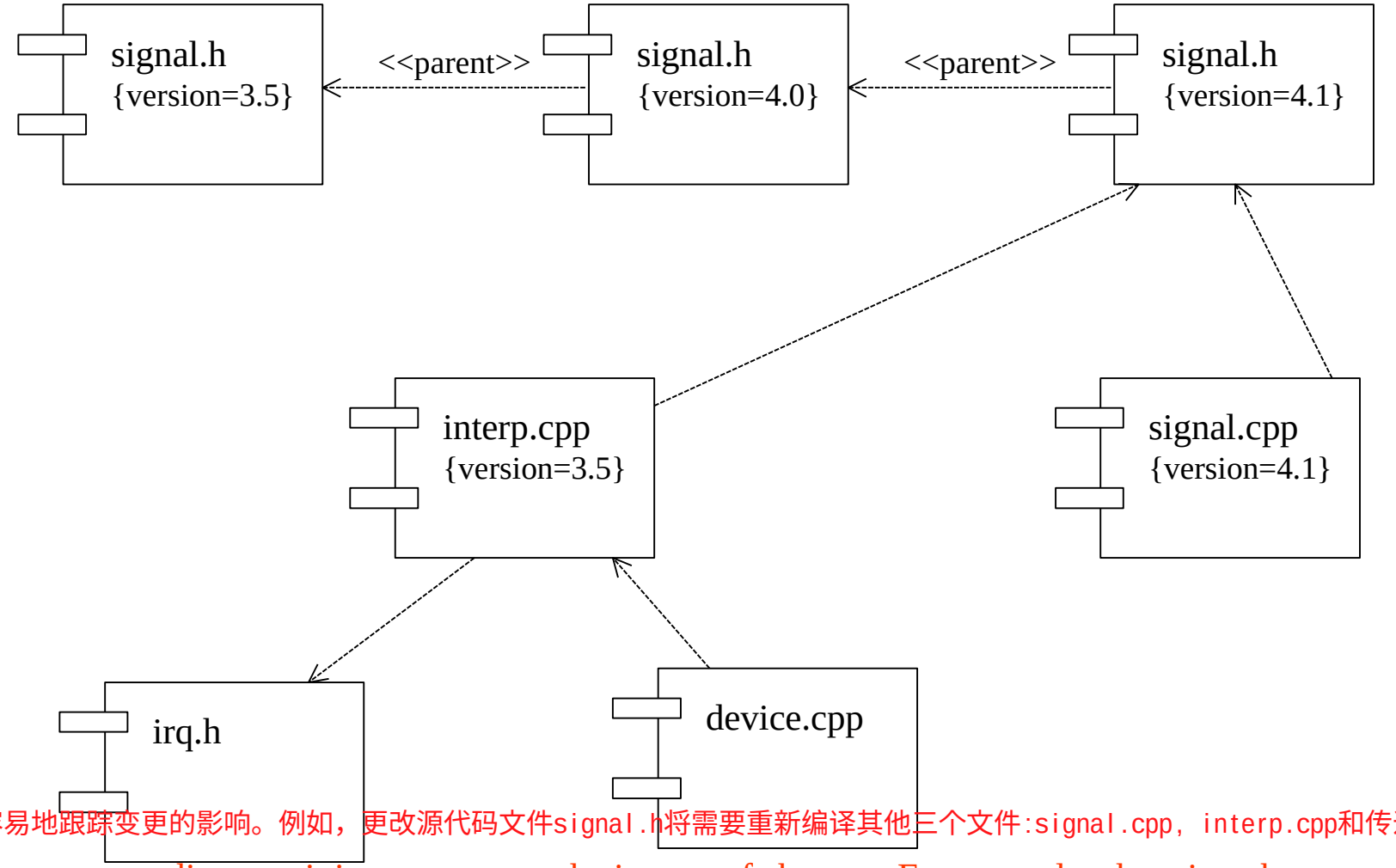


Examples

- ***Executable Release*** (for a web-based application)



- Source Code (showing different versions of the same program)



根据这个组件图，可以很容易地跟踪变更的影响。例如，更改源代码文件signal.h将需要重新编译其他三个文件:signal.cpp, interp.cpp和传递的device.cpp。但是，文件irq.h不受影响

- Based on this component diagram, it is easy to trace the impact of changes. For example, changing the source code file signal.h will require recompilation of three other files: signal.cpp, interp.cpp, and transitively device.cpp. However, file irq.h is not affected.

4.Deployment View

专注于如何将软件部署到我们称之为“硬件”的那个比较重要的层面上

Concentrates on how the software is deployed into that somewhat important layer we call 'hardware'.

Overview

展示如何将各种可执行程序和其他运行时实体映射到底层平台或计算节点

-Shows how the ***various executables and other runtime entities*** are ***mapped to the underlying platforms or computing nodes.***

-Addresses issues such as:

- ***Deployment***
- ***Installation***
- ***Maintenance***

解决的问题包括：
- 部署
- 安装
- 维护

Exposes:
- System performance
- Object/data distribution
- Quality of Service (QoS)
- Maintenance frequency and effects on uptime
- Computing nodes within the system

公开：
- 系统性能
- 对象/数据分布
- 服务质量(QoS)
- 维护频率和对正常运行时间的影响
- 计算系统内的节点

Deployment Diagram

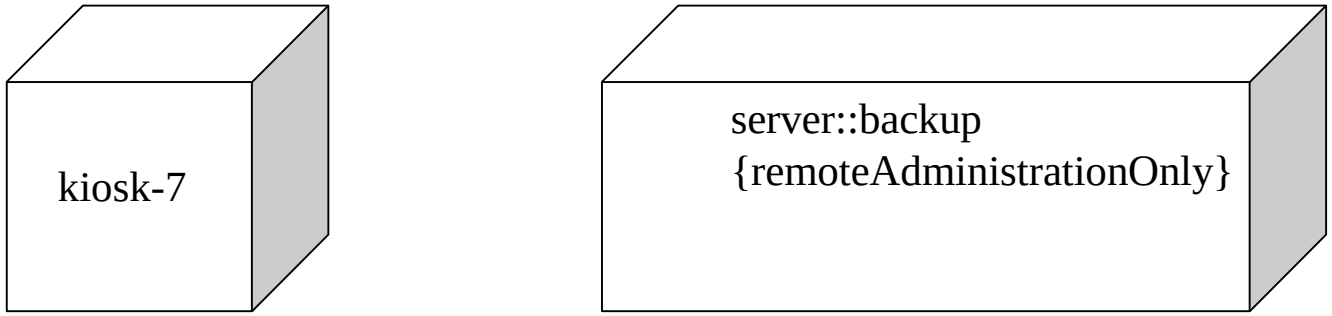
Notation

节点是表示计算资源的物理元素，通常具有一定的内存和处理能力

- A ***node*** is a ***physical element representing a computational resource***, generally having some memory and processing capability.

节点用于对执行系统的硬件的拓扑进行建模:可以在其上部署组件的处理器或设备

- Nodes are used to model the topology of the hardware on which the system executes: processor or device on which components may be deployed.



您可以通过指定节点之间的关系来组织节点

- You may ***organize nodes by specifying relationships*** among them.

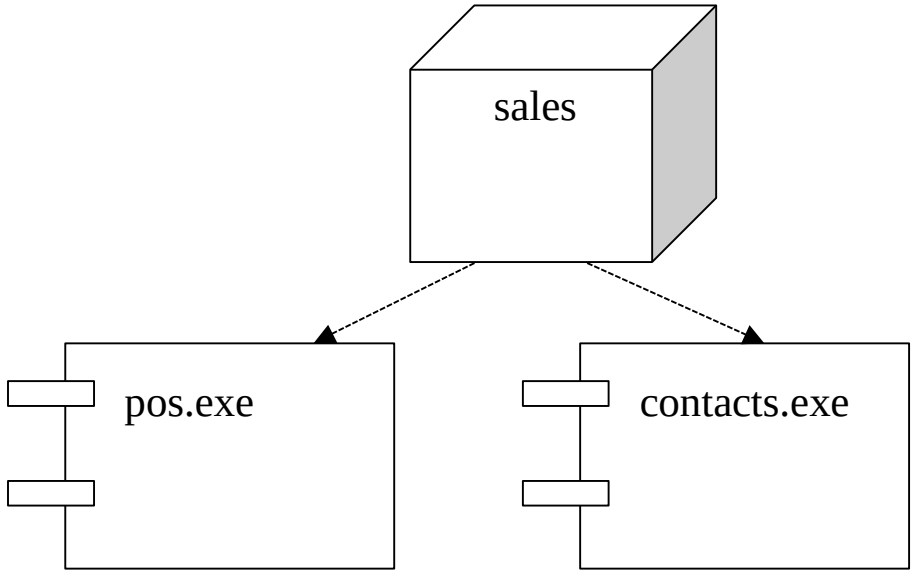
Nodes and Components

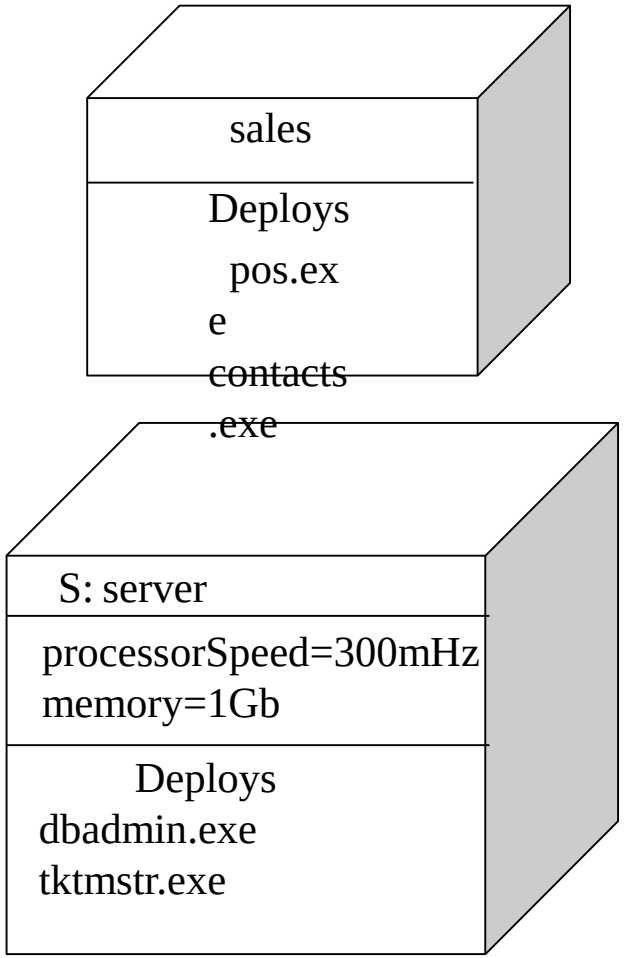
节点是部署组件的位置

- Nodes are locations upon which components are deployed.

作为一个组分配给节点的一组对象或组件称为分布单元

- A set of objects or components that are allocated to a node as a group is called a ***distribution unit***.





- You may also ***specify attributes and operations*** for them: *speed*, *memory*

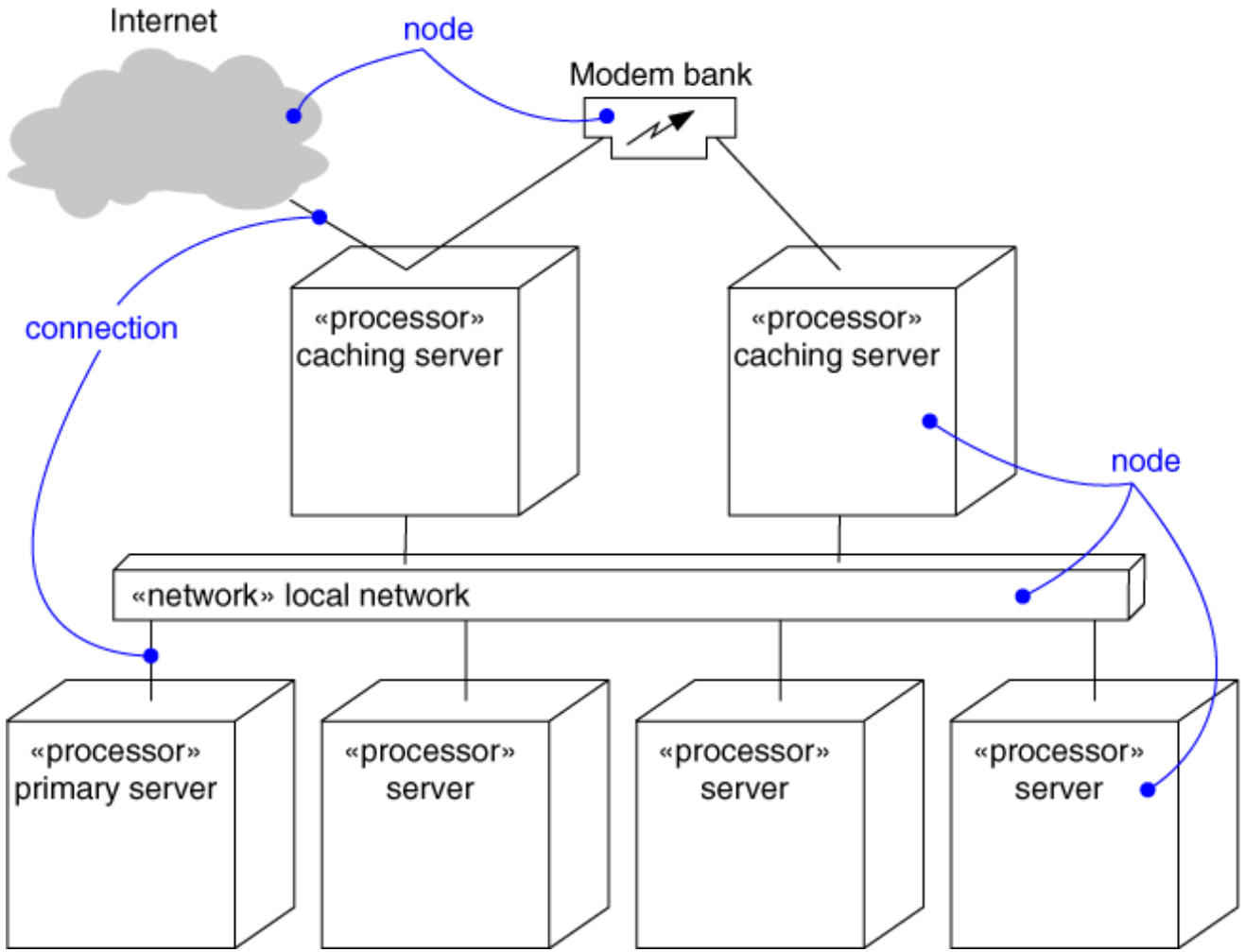
您还可以为它们指定属性和操作:速度、内存

Deployment Diagram

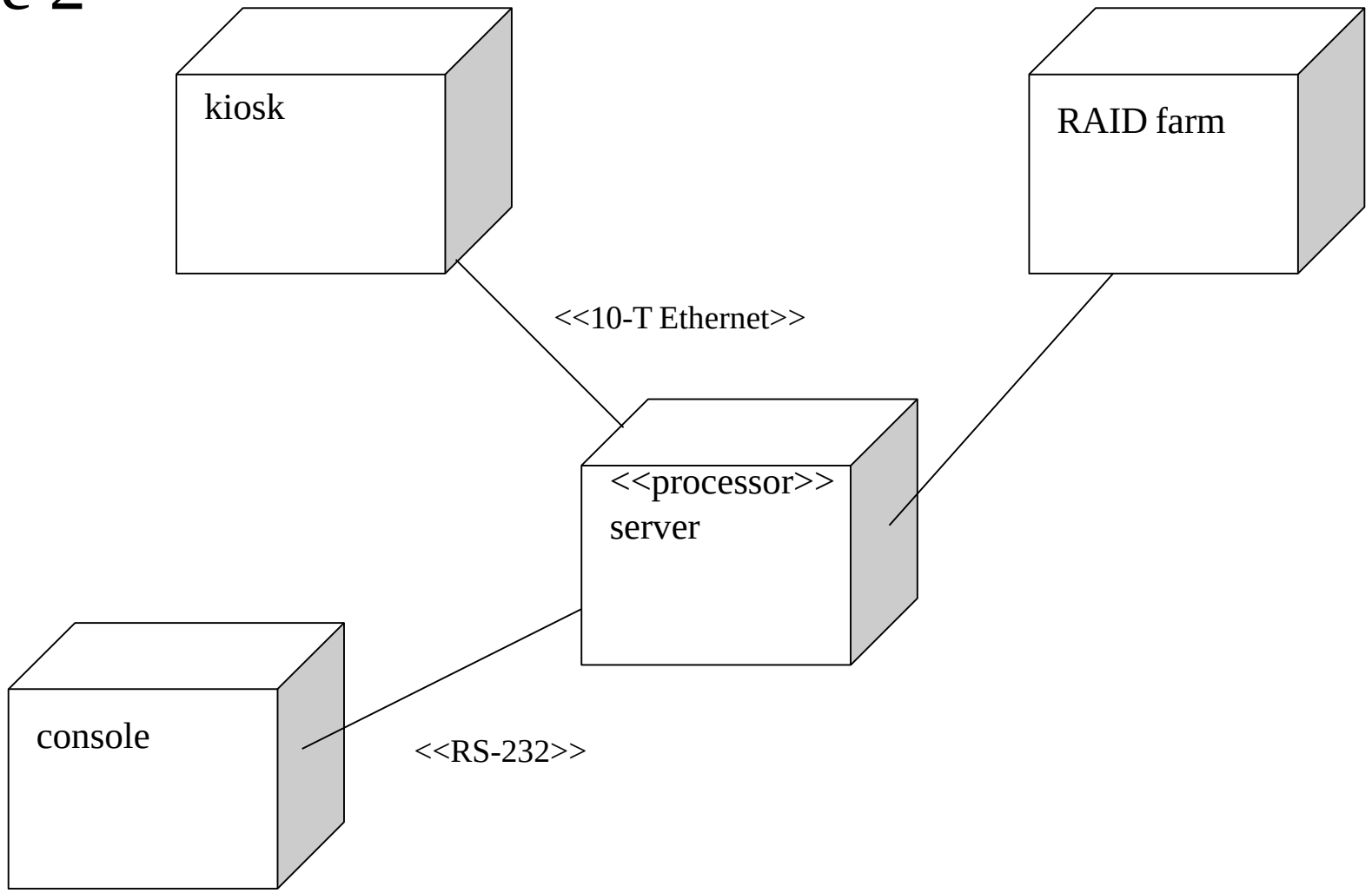
您可以使用部署图来为系统的静态部署视图建模

- You use a deployment diagram to model the static deployment view of a system.

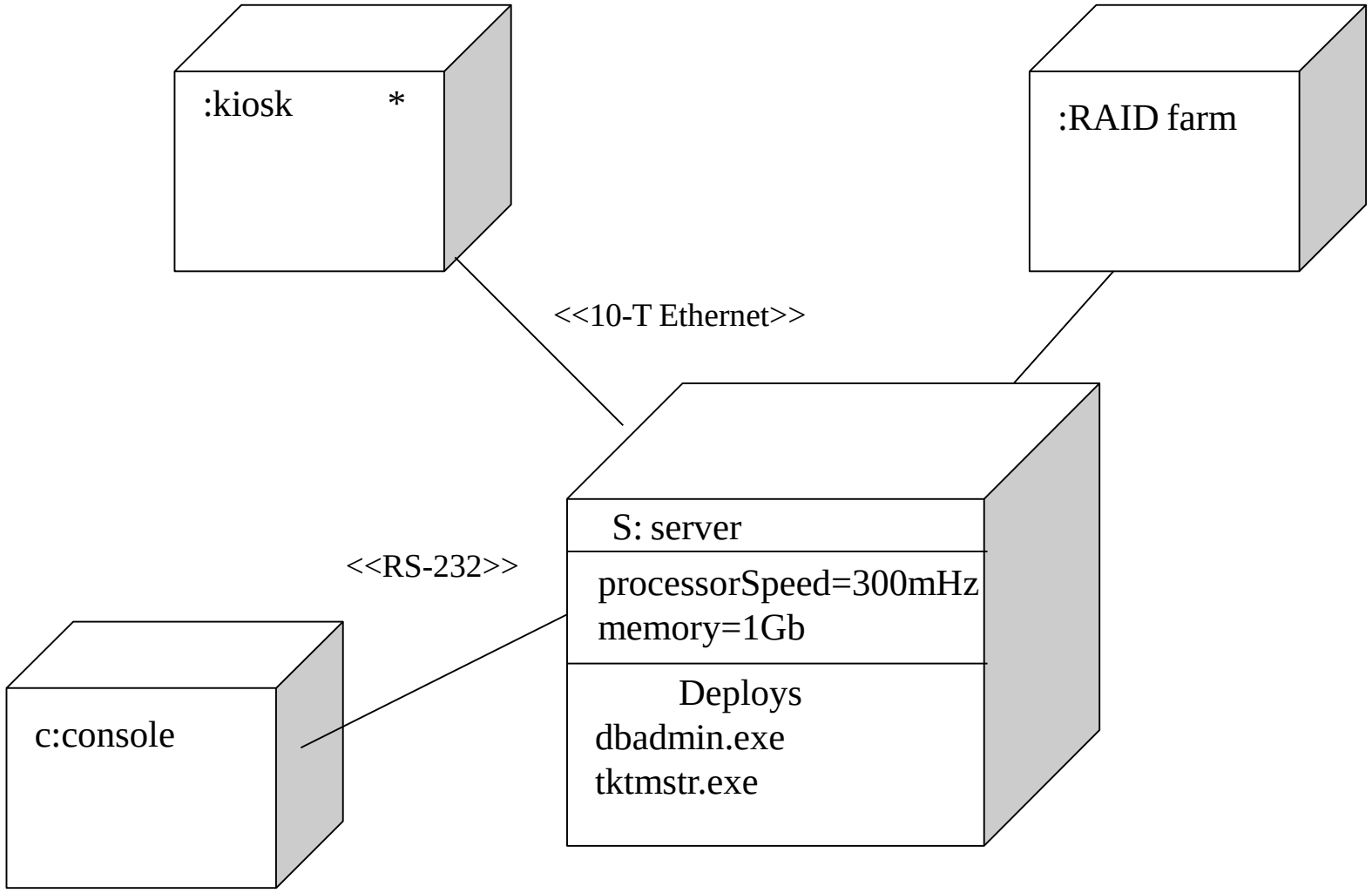
- *Example 1*



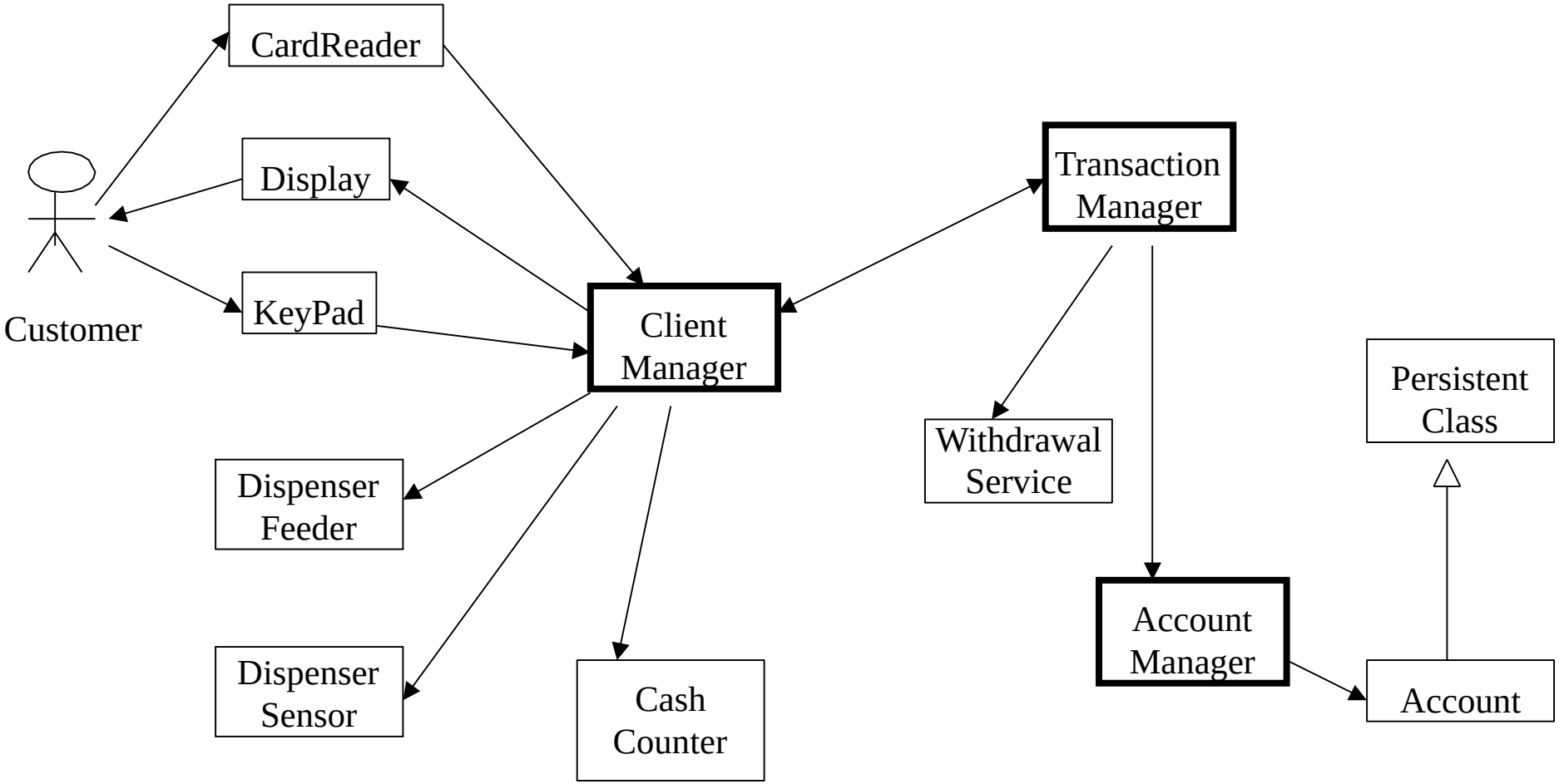
- Example 2


kiosk
RAID farm
<<10-T Ethernet>>
<<processor>>
server
console
<<RS-232>>

- *Distribution of Components*

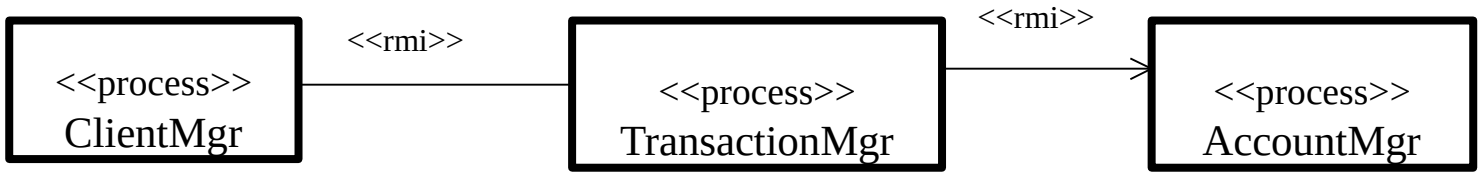


5.ATM Example



Process View

Class diagram

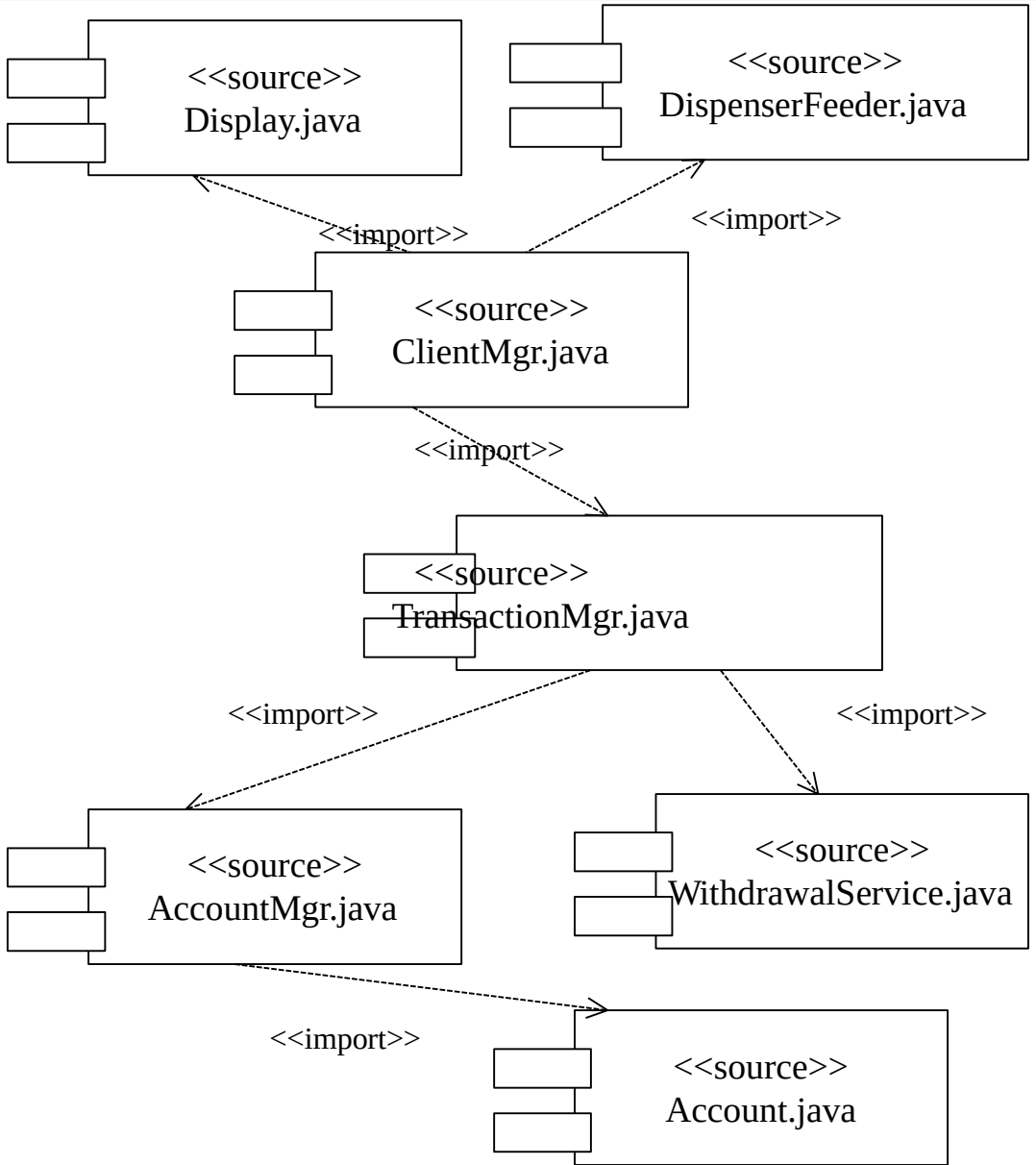


Classes	**Processes**
Display, KeyPad, CardReader, ClientMgr, DispenserFeeder, DispenserSensor, CashCounter	ClientMgr
Transaction Mgr, WithdrawalService	TransactionMgr
AccountMgr, Account, Persistent class	AccountMgr

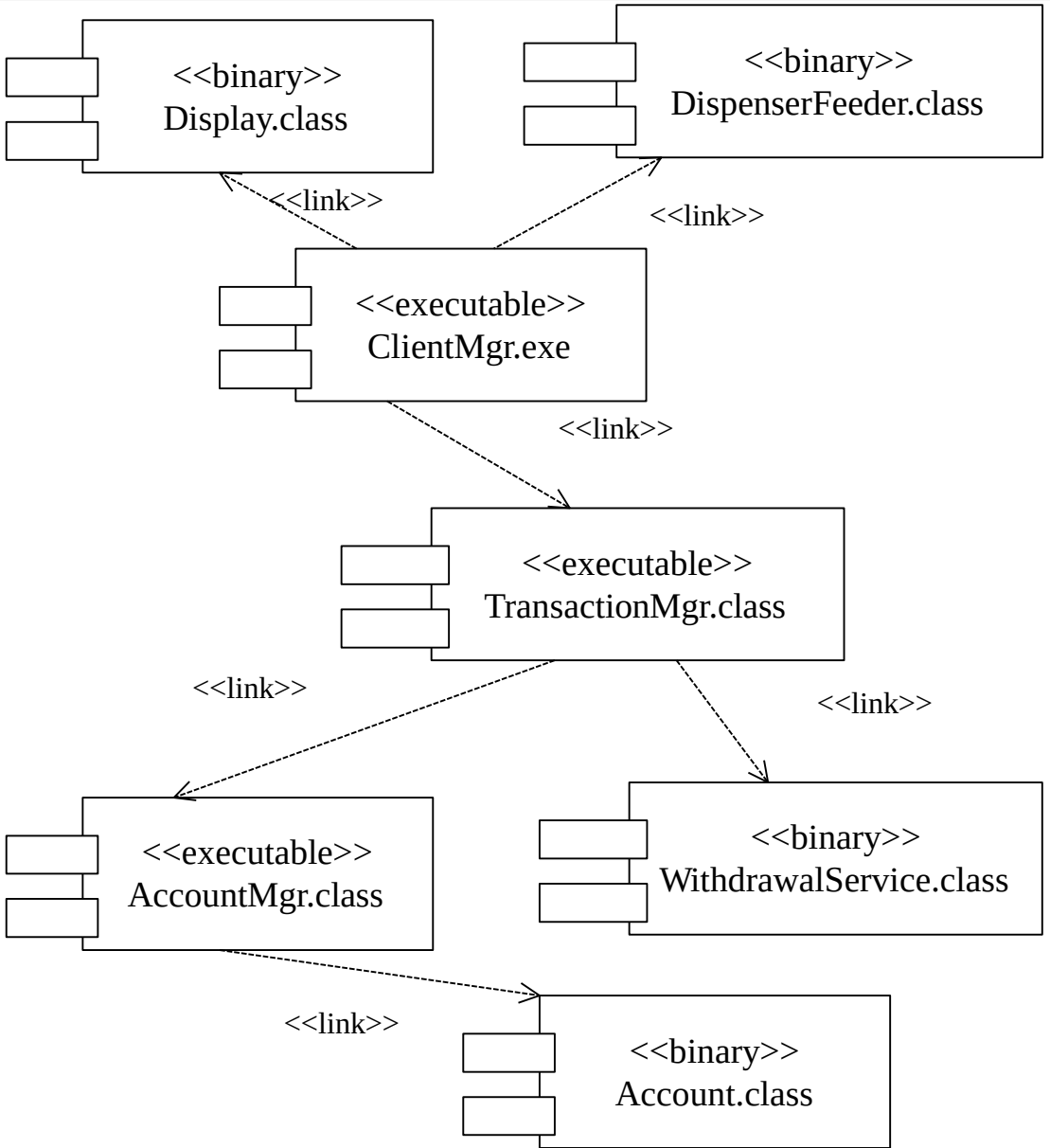
Implementation View

-Source Components

Classes	Source Components
CardReader, Display, KeyPad	Display.java
DispenserFeeder, DispenserSensor, CashCounter	DispenserFeeder.java
ClientMgr	ClientMgr.java
TransactionMgr	TransactionMgr.java
AccountMgr	AccountMgr.java
WithdrawalServic e	WithdrawalService.java
Account, Persistent class	Account.java

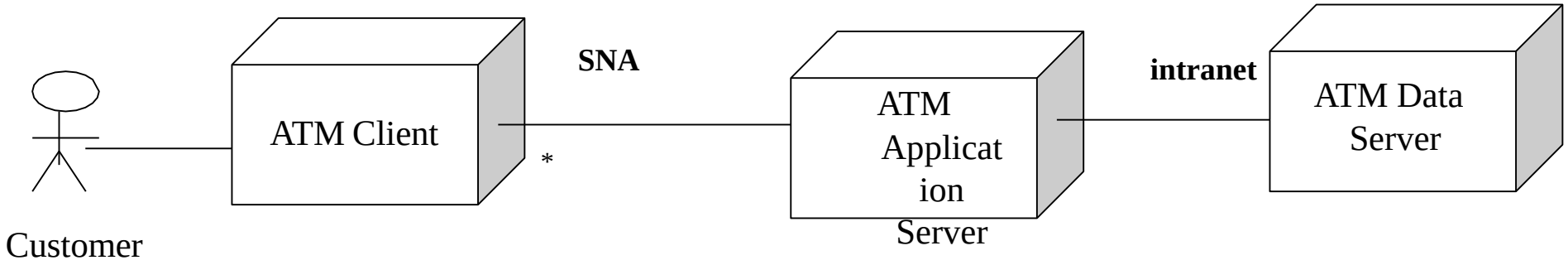


-Executable Release

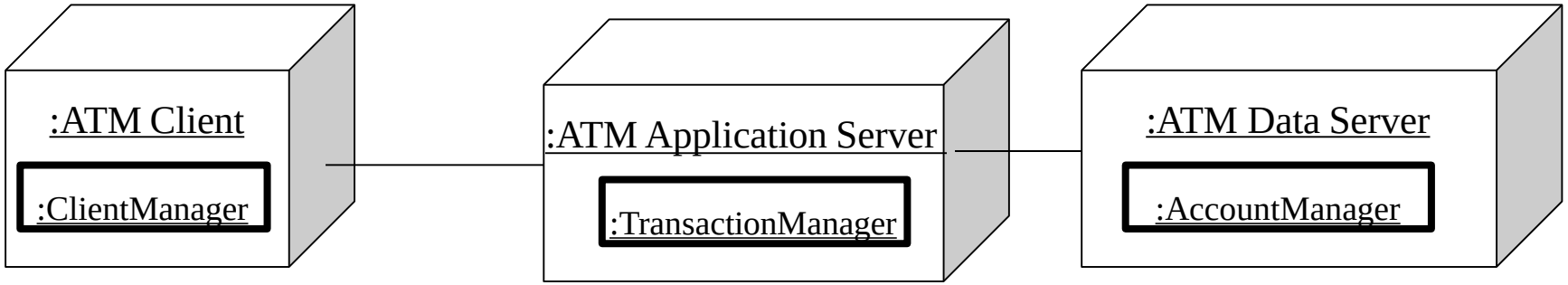


Deployment View

- *Deployment Diagram*



- *Deployment of Active Objects*



Another Architectural views

我们可以从不同的角度来审视一个系统

- We can examine a system from different points of view

  Different kinds of views

概念:组件是一组职责，连接器是信息流

- **Conceptual**: components are set of responsibilities and connectors are flow of information

执行:组件是执行单元(进程)，连接器是进程之间的消息

- **Execution**: components are execution units (processes) and connectors are messages between processes

实现:组件是库、源代码、文件等，连接器是协议、api调用等

- **Implementation**: components are libraries, source code, files, etc and connectors are protocols, api calls, etc.

Another Architectural views

概念架构根据域级功能来考虑系统的结构

- The **conceptual architecture** considers the structure of the system in terms of its **domain-level functionality**

执行架构根据运行时结构来考虑系统

- The **execution architecture** considers the system in terms of its **runtime structure**

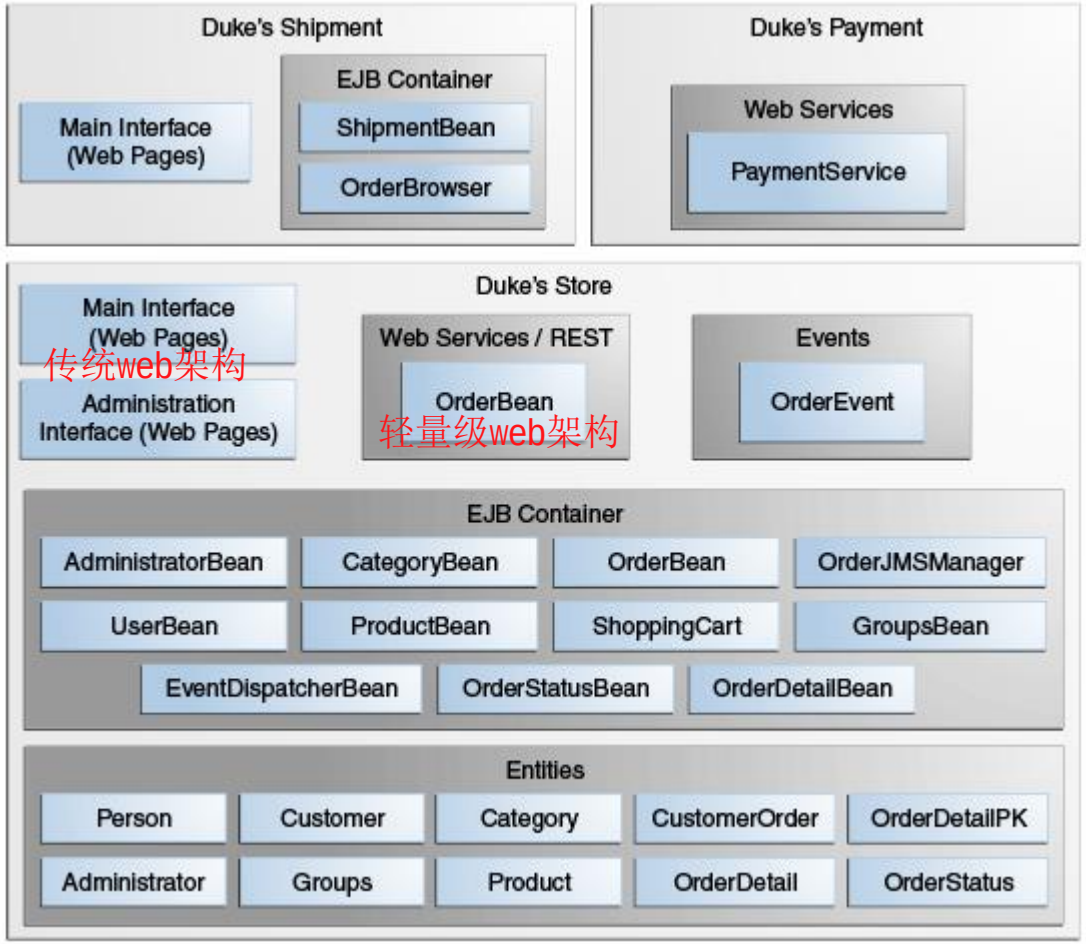
实现架构根据系统的构建时结构来考虑系统

- The **implementation architecture** considers the system in terms of its **build-time structure**

An example: Duke's Forest

Duke's Forest是一个电子商务应用程序，由三个主要项目和三个子项目组成

- Duke's Forest is an e-commerce application consisting of three main projects and three subprojects



An example: Duke's Forest

Duke's Forest使用了以下Java EE 7平台特性:

- Duke's Forest uses the following Java EE 7 platform features:
- Java Persistence API entities:

Java持久化API实体:
-实体上的Bean验证注释，用于验证数据
-用于XML绑定(JAXB)序列化的Java API的XML注释

  - Bean Validation annotations on the entities for verifying data
  - XML annotations for Java API for XML binding (JAXB) serialization
- Web services:

Web服务:
-带有安全约束的JAX-RS支付web服务
-基于EJB的JAX-RS web服务

  - A JAX-RS web service for payment, with security constraints
  - A JAX-RS web service that is EJB based
- Enterprise beans:

企业bean:
-本地会话bean
-所有企业beans打包进WAR

  - Local session beans
  - All enterprise beans packaged within the WAR
- Contexts and Dependency Injection (CDI):

上下文和依赖注入(CDI):
-JavaServer Faces组件的CDI注释
-用作购物车的CDI托管bean，具有会话范围
-限定符
-事件和事件处理器

  - CDI annotations for JavaServer Faces components
  - A CDI managed bean used as a shopping cart, with conversation scoping
  - Qualifiers
  - Events and event handlers

An example: Duke's Forest

- Duke's Forest uses the following Java EE 7 platform features:
- Servlets:
  - A servlet for dynamic image presentation
- JavaServer Faces 2.2 technology, using Facelets for the web front end
  - Templating
  - Composite components
  - File upload
  - Resources packaged in a JAR file so they can be found in the classpath
- Security:
  - Java EE security constraints on the administrative interface business methods (enterprise beans)
  - Security constraints for customers and administrators (web components)
  - Single Sign-On (SSO) to propagate an authenticated user identity from Duke's Store to Duke's Shipment

servlet:
-一个动态图像显示的servlet

JavaServer Faces2.2技术，使用Facelets作为web前端
-模板
-复合组件
-文件上传
-打包在JAR文件中的资源，以便可以在类路径中找到它们

安全:
-管理接口业务方法(企业bean)的Java EE安全约束
-针对客户和管理员的安全约束(web组件)
-单点登录(SSO)将经过身份验证的用户身份从Duke的Store传播到Duke的Shipment

An example: Duke's Forest

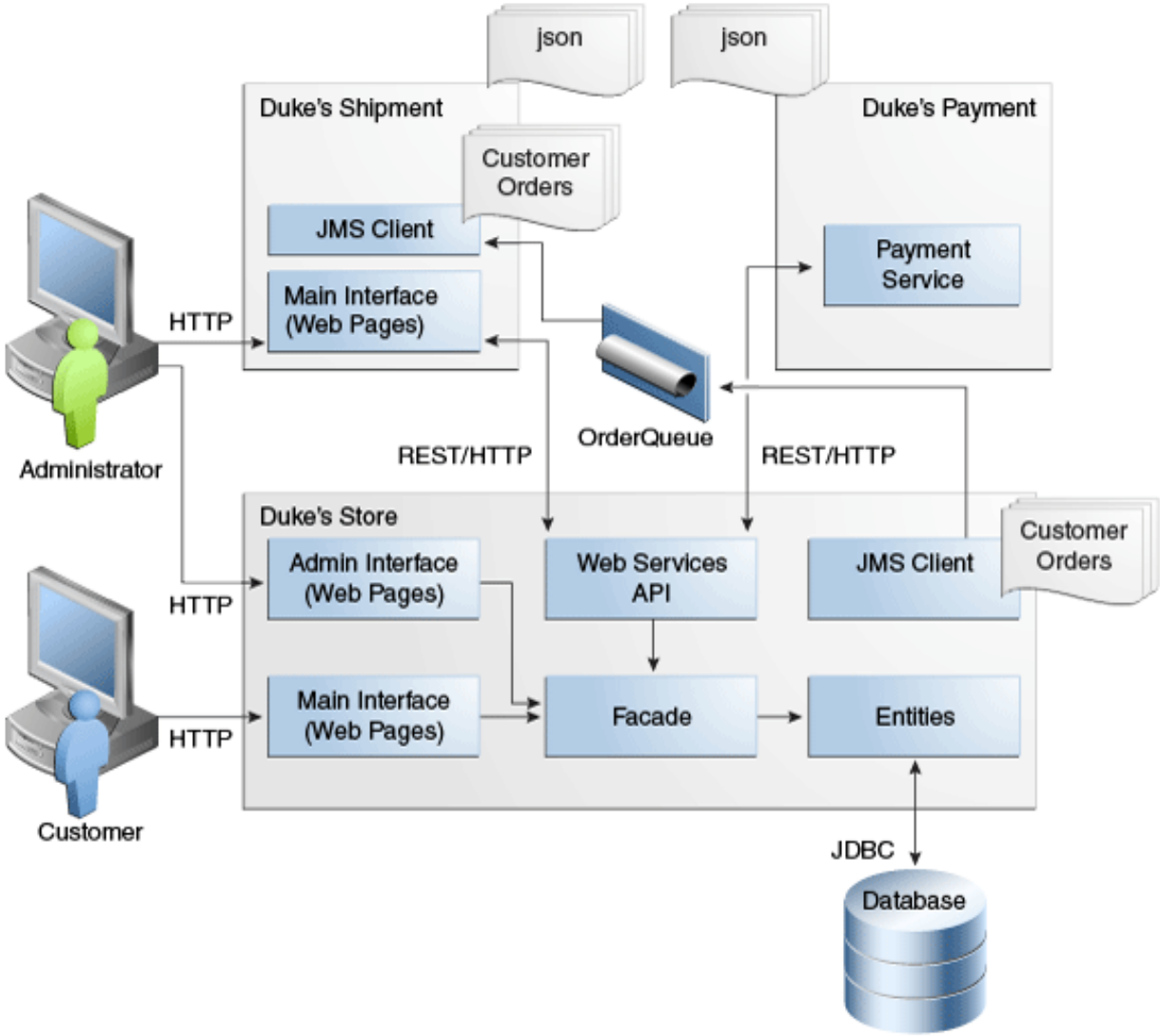
- The Duke's Forest application has two main user interfaces, both packaged within the Duke's Store WAR file:
  - The main interface, for customers and guests
  - The administrative interface used to perform back office operations, such as adding new items to the catalog

Duke的Forest应用程序有两个主要的用户界面，都打包在Duke的Store WAR文件中:
-主界面，供客户和客人使用
-用于执行后台办公室操作的管理界面，如向目录中添加新项目

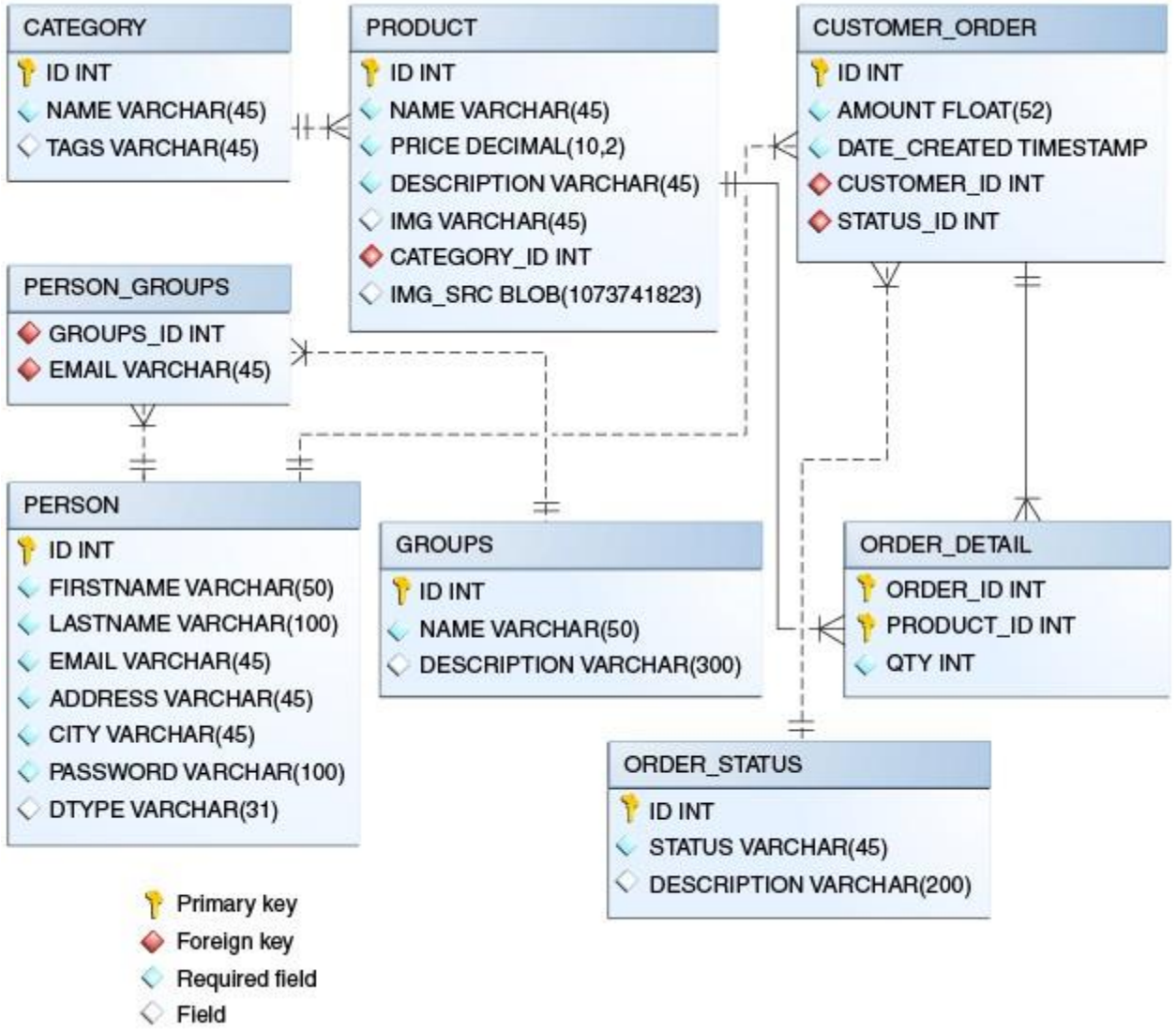
Duke的Shipment应用程序也有一个用户界面，供管理员访问

- The Duke's Shipment application also has a user interface, accessible to administrators.

An example: Duke's Forest



An example: Duke's Forest



Requirements of Duke's Forest Project

- To build the architectural views of SNS web site on the base of your comprehension:
  - Logic view
  - Process view
  - Deployment view
  - Development view
  - Conceptual view
  - Execution view
  - Implementation view
  - Necessary description of the views

在理解的基础上构建SNS网站的架构视图:
-逻辑视图
-过程视图
-部署视图
-开发视图
-概念视图
-执行视图
-实现视图
-视图的必要说明

Experiment SMART monitor project

- 简介
  - 分布式监控管理系统，方便用户随时了解分布式集群的情况（机器的状态）以及及时进行各种机器的操作。
  - Google云计算、Hardoop等平台的重要组成部分。

Experiment SMART monitor project

- 功能
  - **PC端**
    - 实现用户登录，实时监控各设备节点状态、配置设备、接收节点状态变更推送等功能。
  - **分布式机器端**
    - 接收服务器端命令，发送心跳包，发送异常信号，发送设备状态变化命令等。

LoadRun负载测试工具

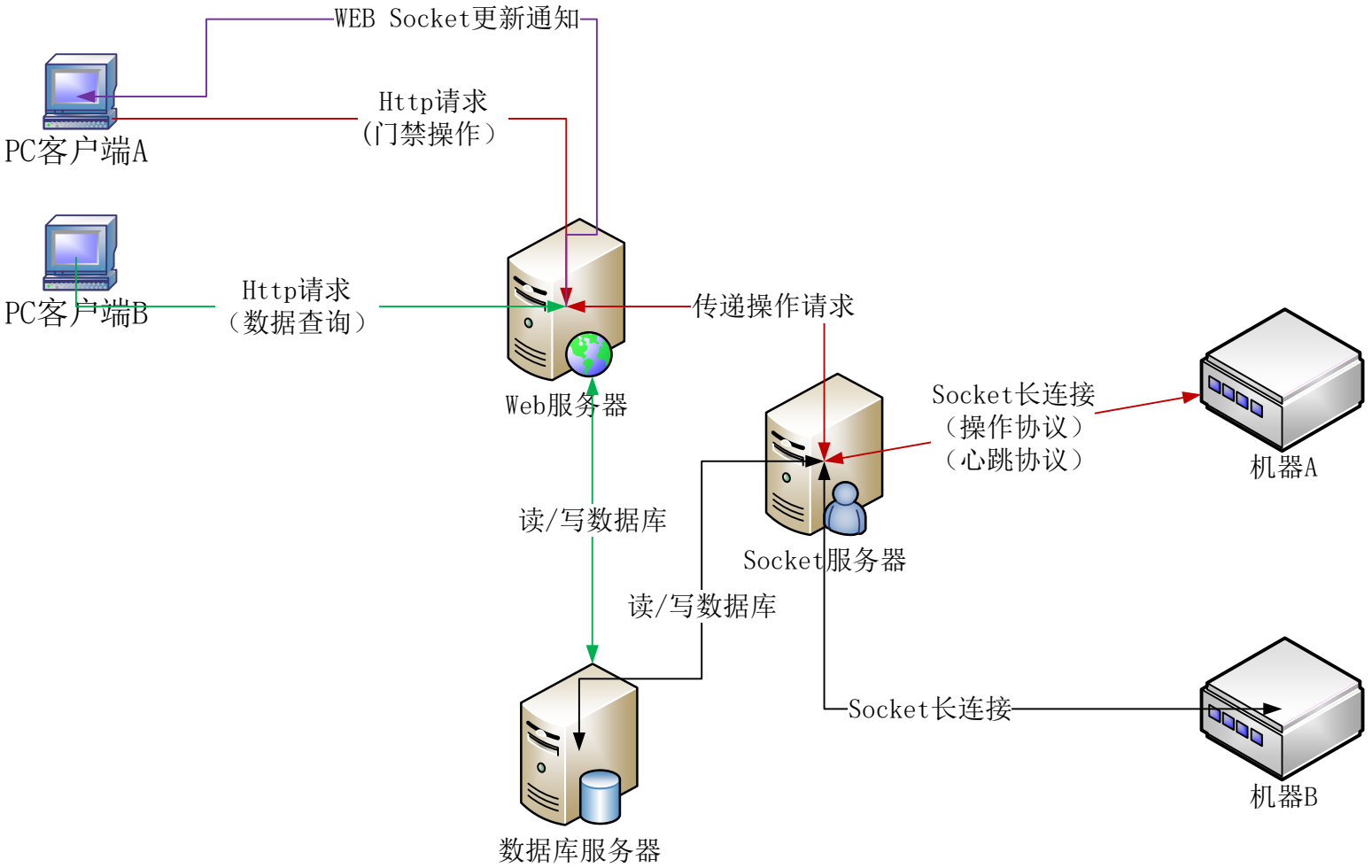
Experiment SMART monitor project

- 非功能需求
  - 性能需求
    - 响应时间迅速
    - 数据传输快速
  - 可扩性需求 能监控的机器数量
    - 链接的站点能够扩展到几千个
    - 能满足多用户并发访问
  - 灵活性需求
    - 提供外部访问接口，能够实现后期如手机应用、外部应用的调用。
    - 程序升级简单

Experiment SMART monitor project

- 非功能需求
  - 稳定性需求
  - 可管理性需求
  - 安全性需求

Experiment SMART monitor project



Requirements of IOT SMART Project

- To build the architectural views of SNS web site on the base of your comprehension:
  - Logic view
  - Process view
  - Deployment view
  - Development view
  - Conceptual view
  - Execution view
  - Implementation view
  - Necessary description of the views

References

1. SWEBOK,
   - http://www.swebok.org
2. Philippe Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture
3. Java EE Tutorial, Duke’s Forest
   - https://docs.oracle.com/javaee/7/tutorial/dukes-forest001.htm#GLNRJ