北京大學
PEKING UNIVERSITY

Thinking in UML
Chapter 16-22

孙鹏晖

Apr, 2017

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

用况是系统思维

- 线性思维----考虑问题时只从一个角度线性地顺藤摸瓜式地去思考和理解一个问题领域，线性思维也称为静态思维.

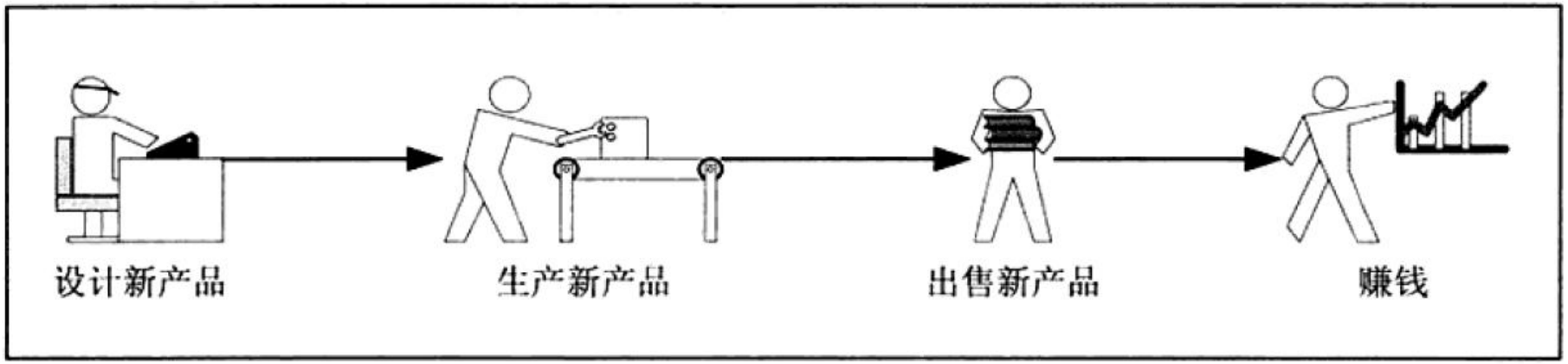

图 16.1　线性思维

用况是系统思维

- 系统思维----考虑系统内事物的相互影响而不是观察单个事物的变化，归纳，抽象系统内的运行规律而不是研究单个点的存在意义.

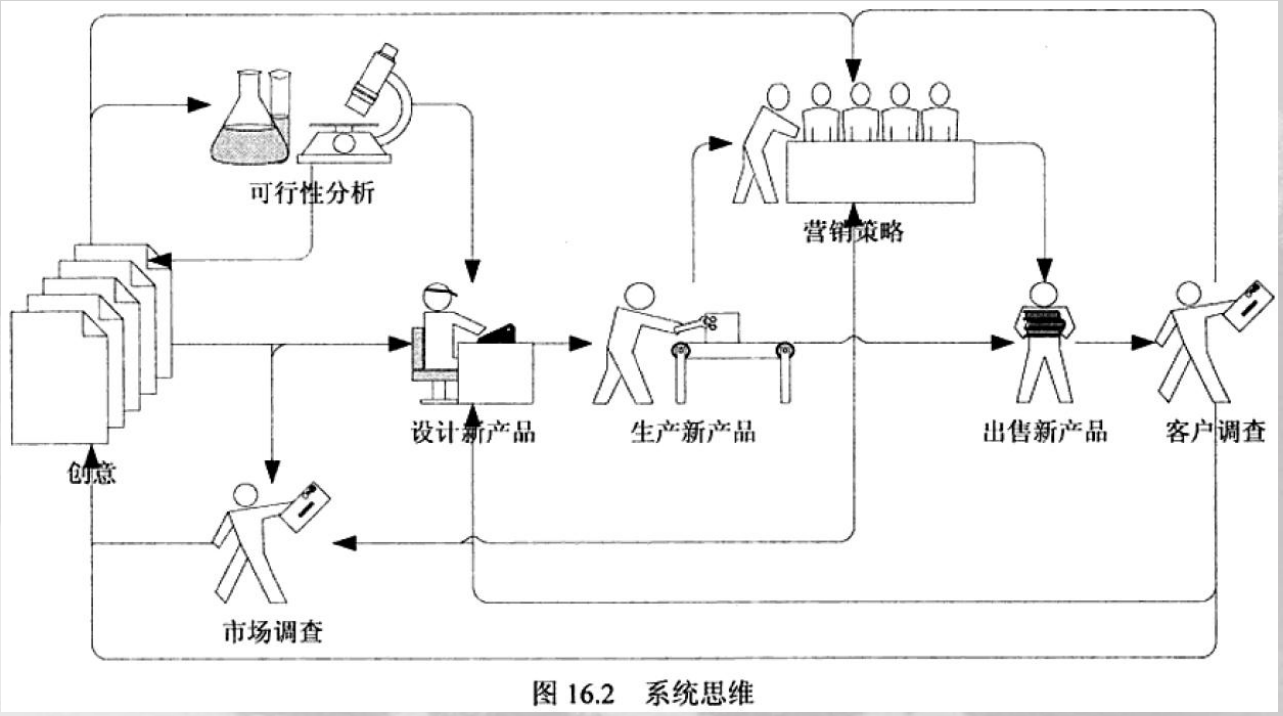

图 16.2　系统思维

用况是系统思维

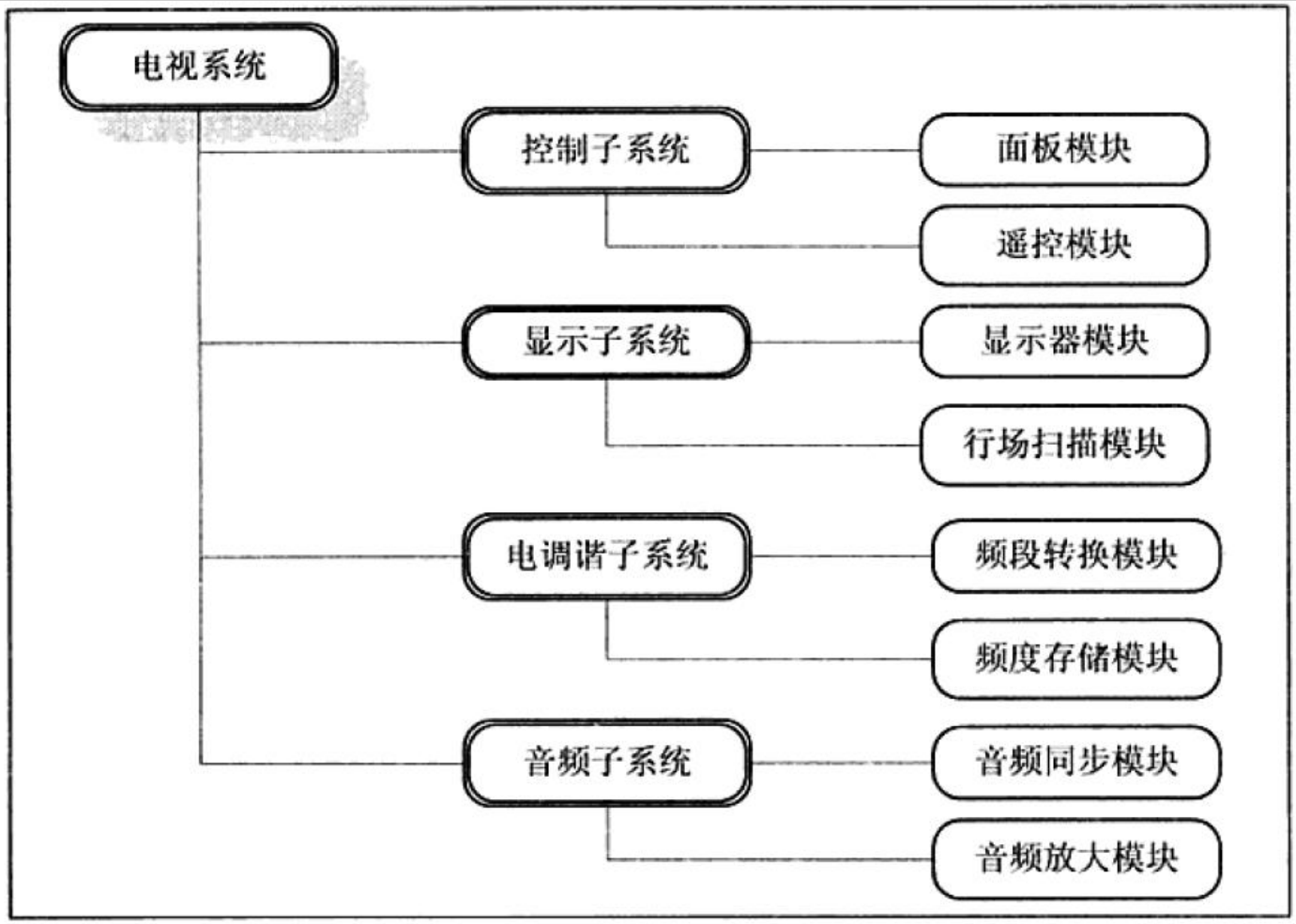

图 16.3　线性思维下的系统分析设计结果

用况是系统思维

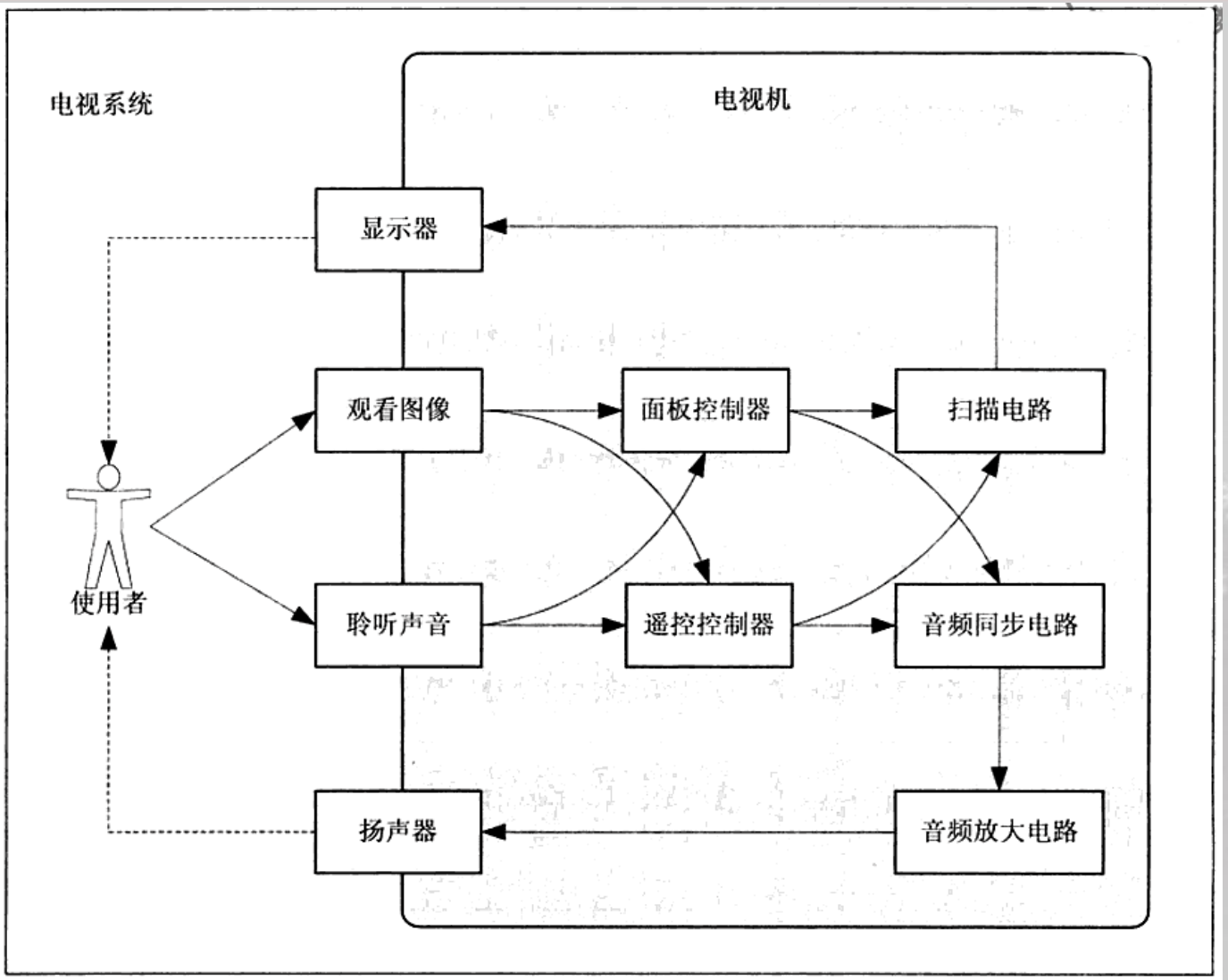

图 16.4　系统思维下的分析设计结果

用况是面向服务的

- SOA(面向服务的架构)的第一条准则：业务驱动服务，服务驱动技术.

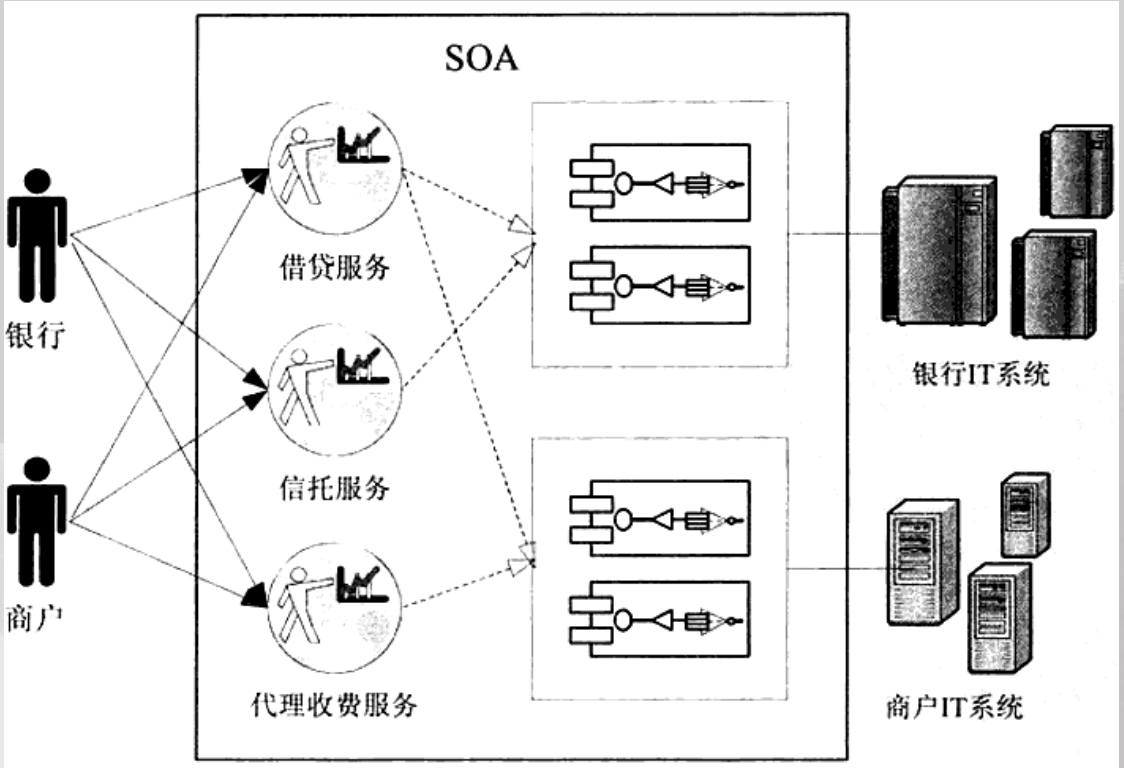

图 16.5　SOA 应用示例

用况是面向服务的

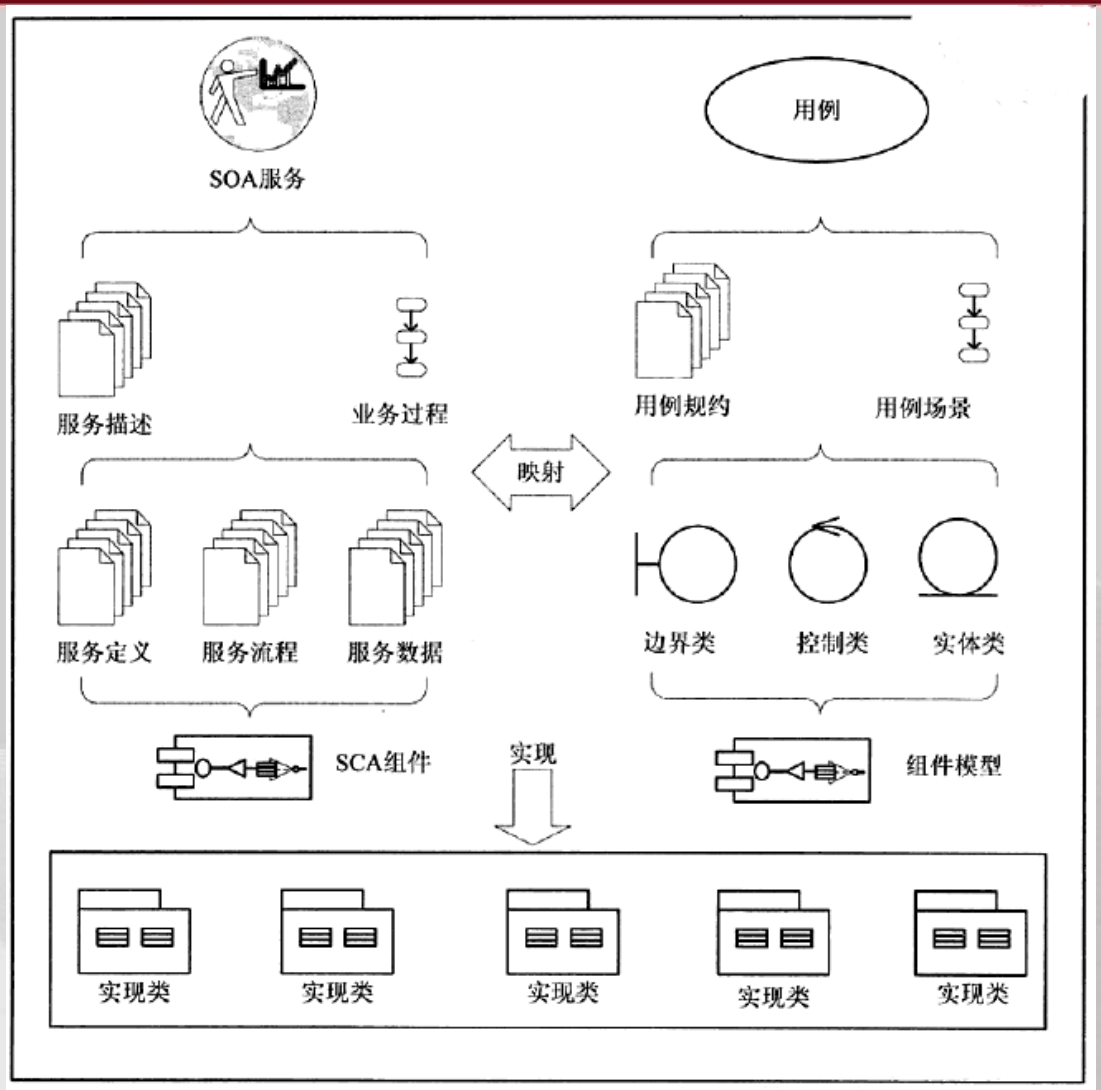

图 16.6　面向服务与用例驱动

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

用况与项目管理

- 软件开发的核心问题是工作量估算+时间安排+人力资源分配.
- 工作量估算源于开发范围，以工作包为基本计算单位，通常用人天，人周，人月等单位来衡量.
- 时间安排和人力资源分配是一对需要平衡的变量，通常时间和人力资源是呈反比状态的.

用况与项目管理

- 具有依赖关系的工作包首尾连接起来的开发时间决定了项目的整体开发时间，由这些工作包连接起来的链路也称为关键路径.

- 对于绘制项目计划来说，经验告诉我们一个工作包的工作量控制在1至2人周是比较容易控制的.

用况与可扩展架构

- 耦合的方
- 在 耦合，用

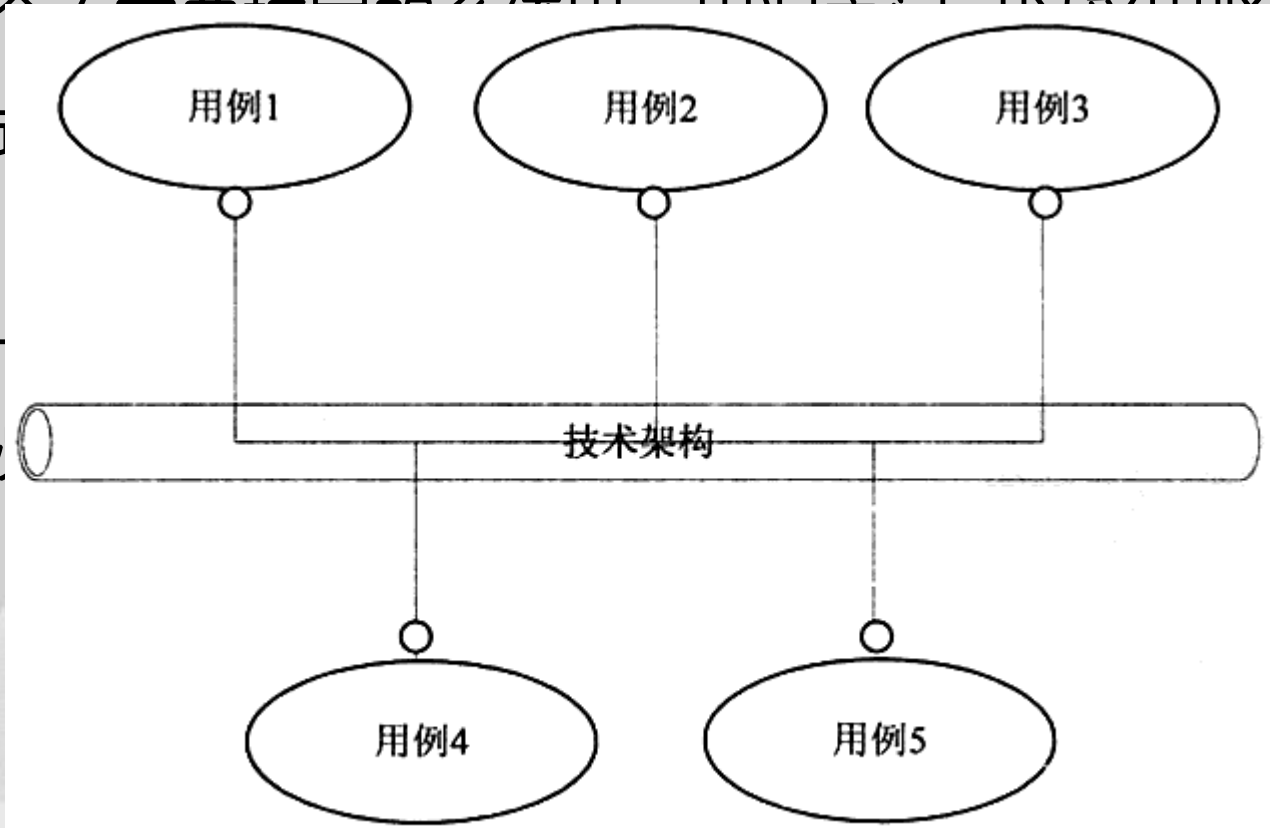

图 17.2　用例与可扩展架构

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

用况驱动与领域驱动的差异

- 用况驱动(UDD)和领域驱动(DDD)都是通过 一种方法定义(或建模)需求，再根据需求定义(或模型)来驱动后续的软件生产工作.
- 用况驱动的结果是软件以实现场景为目的，当系统行为满足所有涉众的期望之后，该系统即成功.
- 领域驱动要求团队有资深业务领域专家，对领域极其了解高度抽象业务本质，从根本上解决问题.

领域驱动的理想与现实

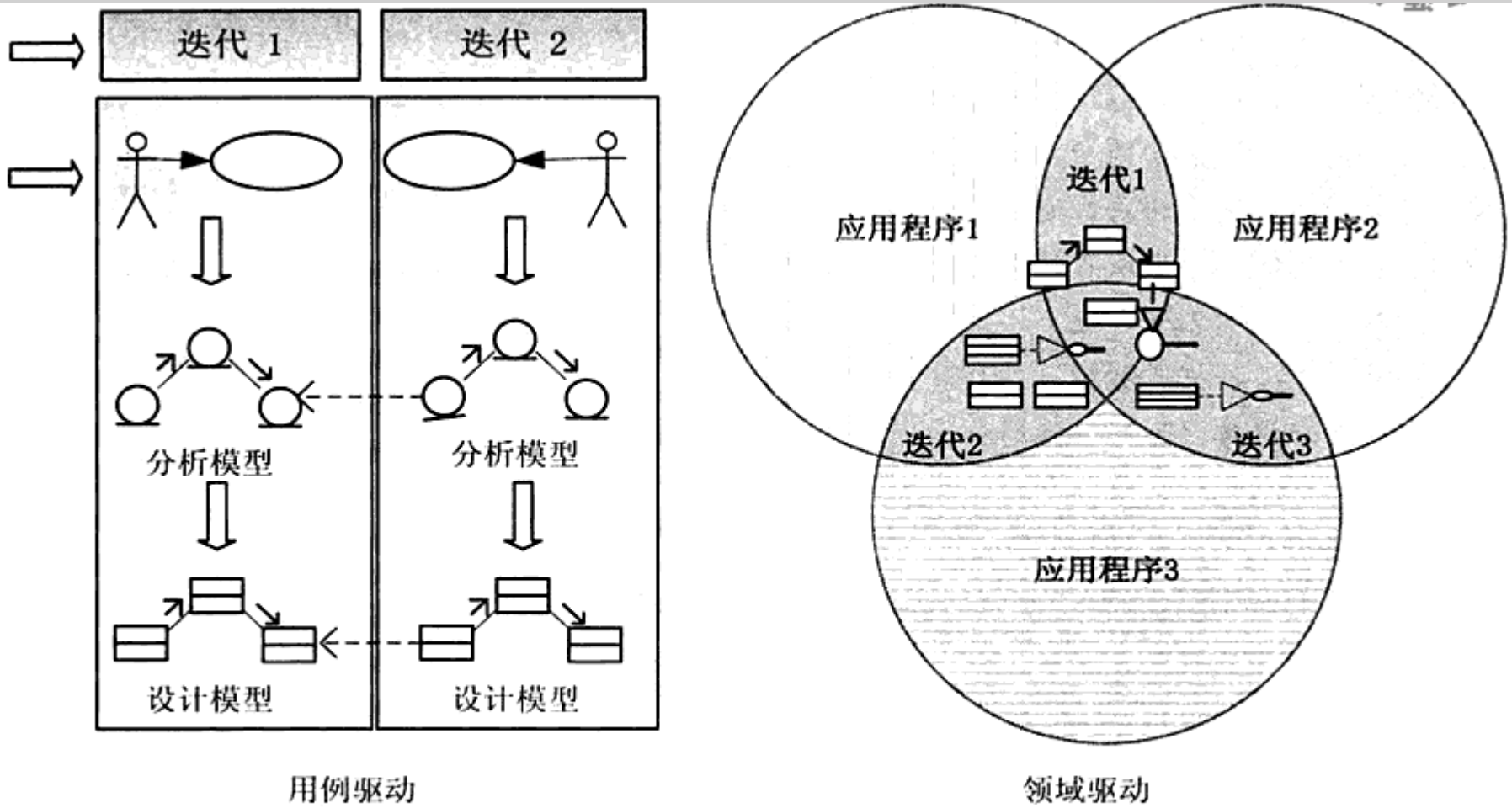

图 18.1　用例驱动与领域驱动

如何决定是否采用领域驱动方法

- 作者建议：如果没有资深的领域专家带头，就不要采用领域驱动方法来实施项目.

- 如果实力不足，在实际项目中应采用用况驱动方法；
- 如果暂时没有顶级行业深度但又想追逐行业领导者，应当建立研发中心，将实施项目和研发产品分离积累经验；
- 如果已经是行业领头羊，有深厚的业务知识背景，或者能够找到资深业务专家，可以采用领域建模方法；

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

再讨论抽象层次

- 抽象是分析事物从中获得并描述共性的过程，也有自顶向下和自底向上两种方法.

- 对软件分析来说，从需求开始，分析设计到实现的过程是采用自顶向下方法.
- 采用自底向上的方法优化和更深一步了解这些结果，我们一般说的接口设计，分包，重构等都是自底向上抽象的例子.

层次高低问题

- 软件也有一些约定俗成的层次划分，例如项目，系统，子系统，模块，子模块，包，类等.
- 令人迷惑的用况粒度问题也是抽象层次的问题.

图 19.1　抽象层次

层次不交叉问题

- 层次不交叉是指在描述事物的过程中，同一层次的描述内容是等值的.

- 例如在生物学中，描述哺乳纲时用恒温, 胎生, 哺乳的脊椎动物来描述, 与之相对, 应当是等值的其他纲, 例如鸟纲, 描述为恒温, 卵生, 喂食的脊椎动物.

如何决定抽象层次

- 在开始一个分析设计之前，应当先确定有多少个抽象层次，例如在结构化设计过程中，系统,子系统,模块,子模块就是预定的抽象层次.
- 面向对象的方法中并非一成不变，简单地问题抽象层次少，(如MIS系统对数据库增删改查三个层次:操作层+程序层+数据层)复杂问题抽象层次多(如ERP需要增加抽象层+工作流层等).
- 代码->类->组件->框架->架构……

抽象层次与UML建模的关系

- UML建模是用况为基础的，是用况驱动.
- 传统MIS系统要解决谁操作哪些数据，一个层次的用况就可以解决问题.
- ERP加入业务流程需要两个层次用况，一业务用况解决谁如何参与业务流程的问题，是业务建模；二是系统用况，解决谁在业务流程中做什么的问题，是系统建模.

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

面向对象的子系统问题

- 结构化设计分析软件是从系统子系统划分开始的，以数据为中心的软件系统按业务划分导致数据依赖关系严重，同一数据被多个子系统访问修改，导致了程序复杂化.

- UC矩阵被提出解决这个问题.

UC矩阵还适用么

- UC矩阵不适合做面向对象的分析和设计.
- 原因:面向对象的系统中构成应用程序的基础不是数据而是对象,UC矩阵可以解决数据依赖问题,但无法解决对象依赖问题.

如何划分子系统

- RUP对子系统有如下定义：
  - 可以独立预定配置或交付
  - 可以独立开发(只要保持接口不变)
  - 可以在一组分布式计算节点上独立部署
  - 可以在不破坏系统其他部分情况下独立更改
- 划分子系统最重要的依据就是依赖关系.

如何划分子系统

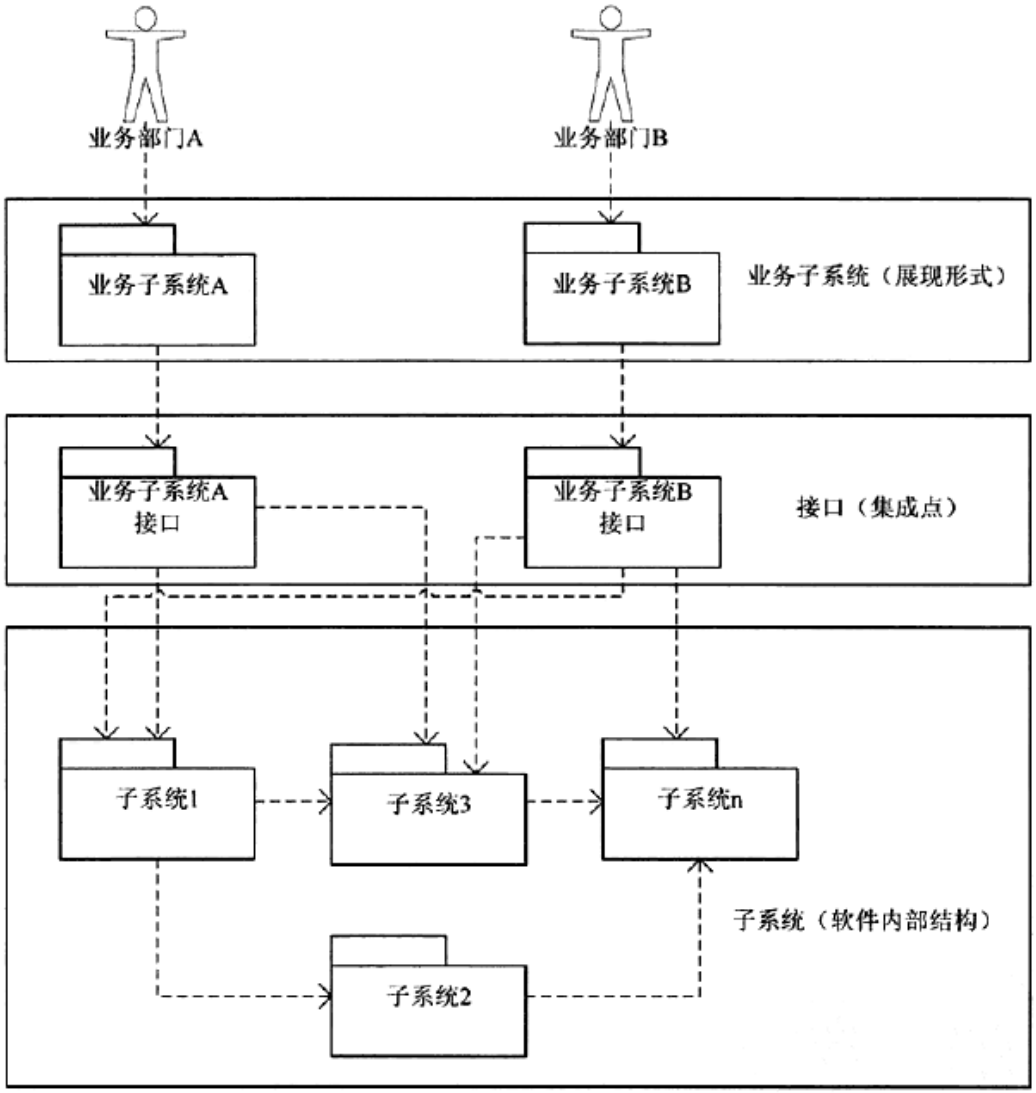

图 20.1　业务子系统与计算机子系统

如何划分子系统

- 作者发明了一种DO矩阵，Dependence&Own.

表 20-2　利用 DO 矩阵划分子系统示例

包 / 用例	包 1	包 2	包 3	包 4	包 5	包 6	包 7	包 8
业务用例 1	O							
业务用例 2		O	O					
业务用例 3		D		O			D	
业务用例 4	D		D	O		D		
业务用例 5		D			O			D
业务用例 6			D		O		O	
业务用例 7	D			D		O		
业务用例 8					D	O		O

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

边界是面向对象的保障

- 封装和接口都来源于边界，接口是边界的体现；
- 对象封装提供的接口在应用程序中为API，在系统内部为SPI，实质上就是对象的边界.
- 任何对象都有边界，任何有交互的两个对象都只能看到对方的边界.

边界分析示例一

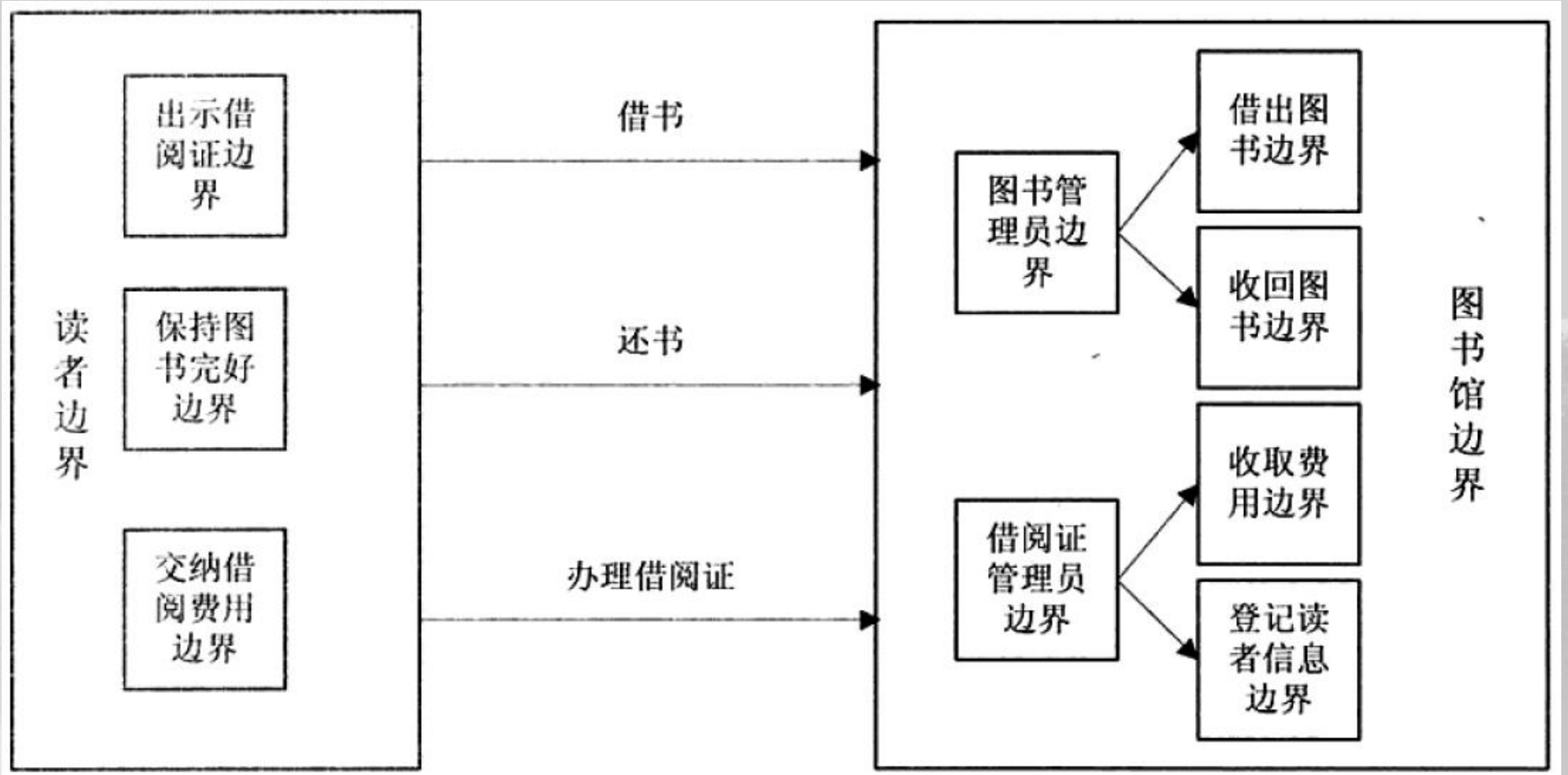

图 21.2　边界分析示例二

边界分析示例一

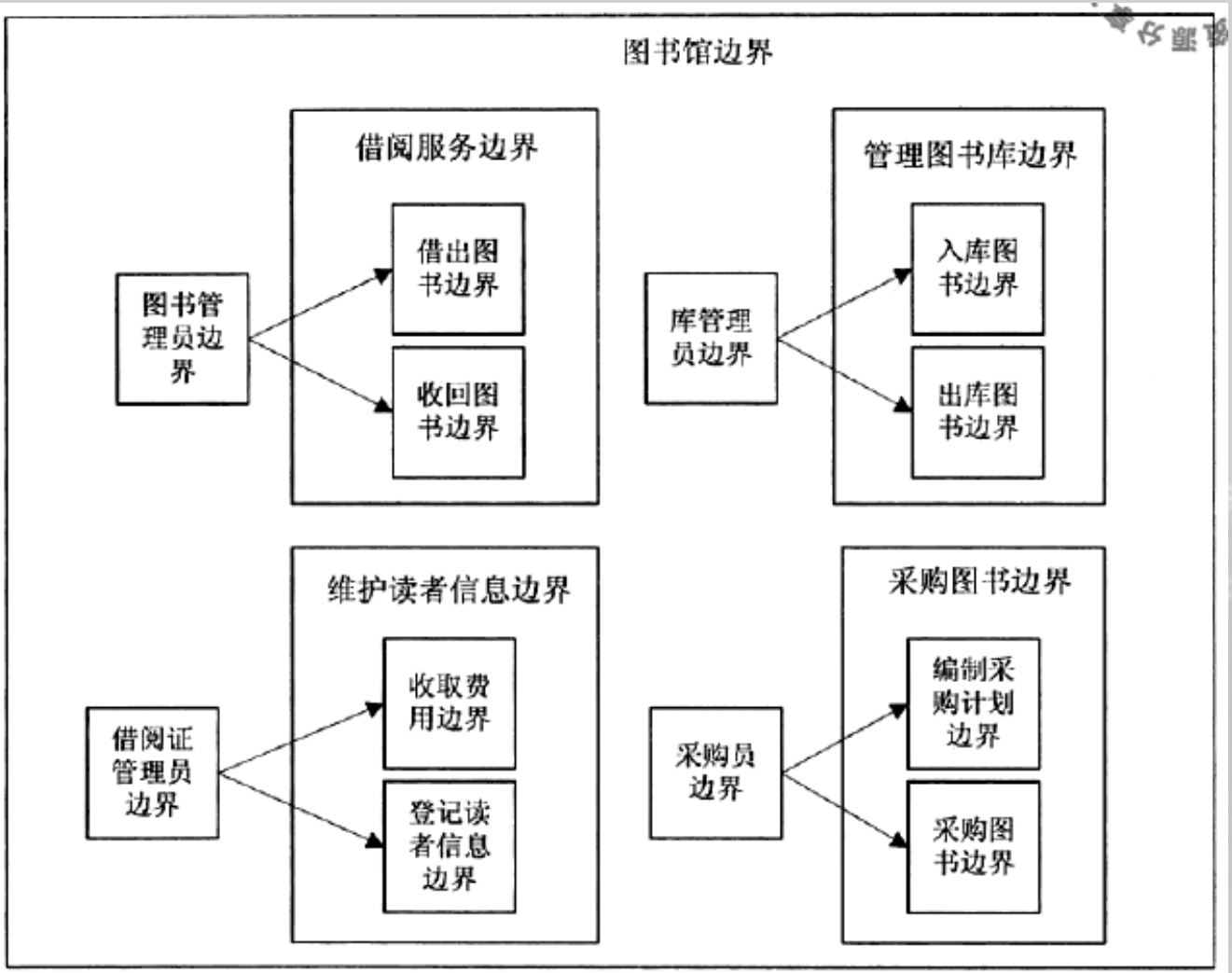

图 21.3　边界分析示例三

边界分析示例一

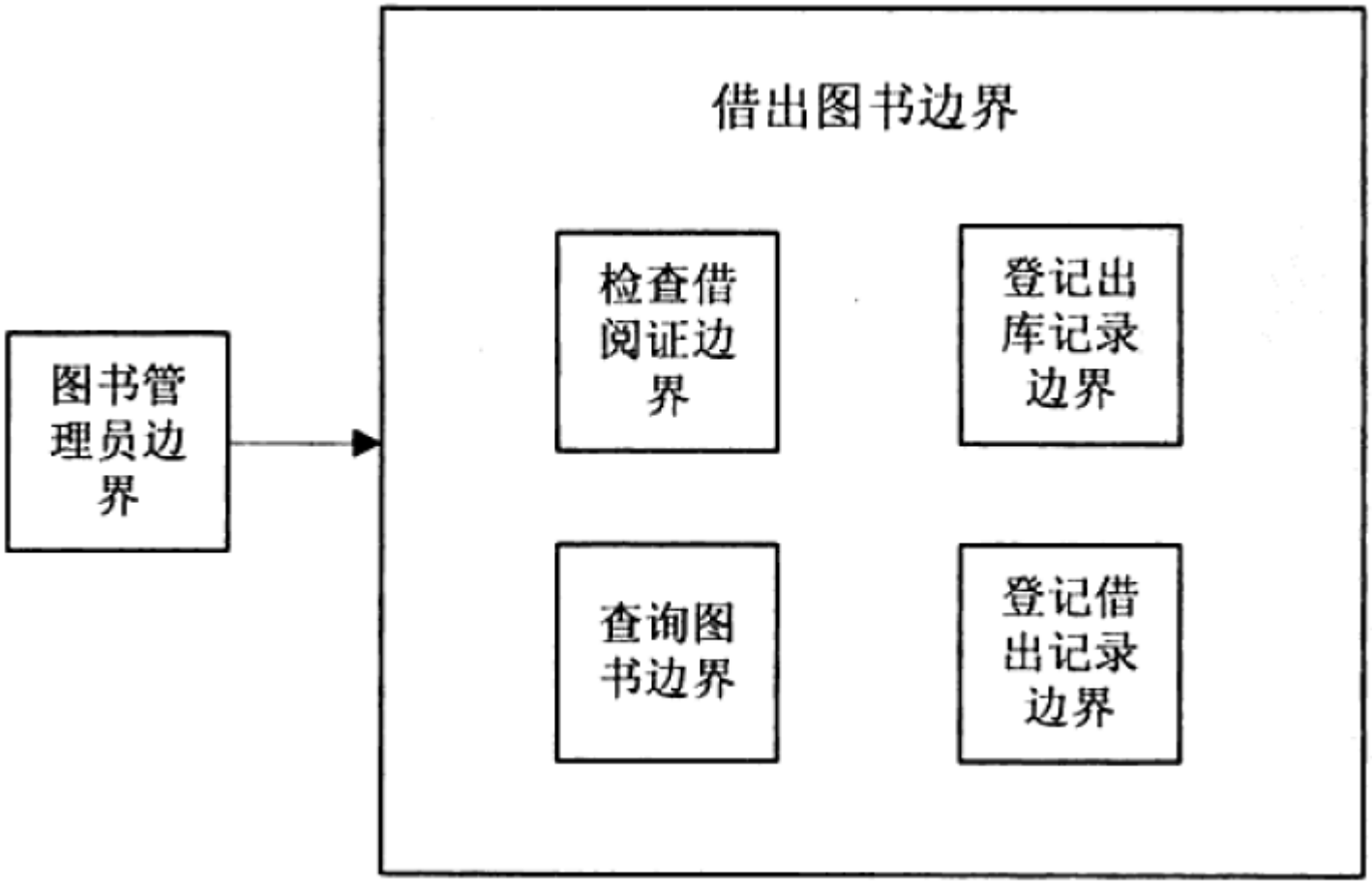

图 21.4　边界分析示例四

边界分析示例一

- 每个小边界包含的业务可以用几十个字说明白；
- 每个小边界包含业务的开发量大约在1-2人周；
- 每个小边界包含业务的步骤不超过10步等；

边界分析示例二

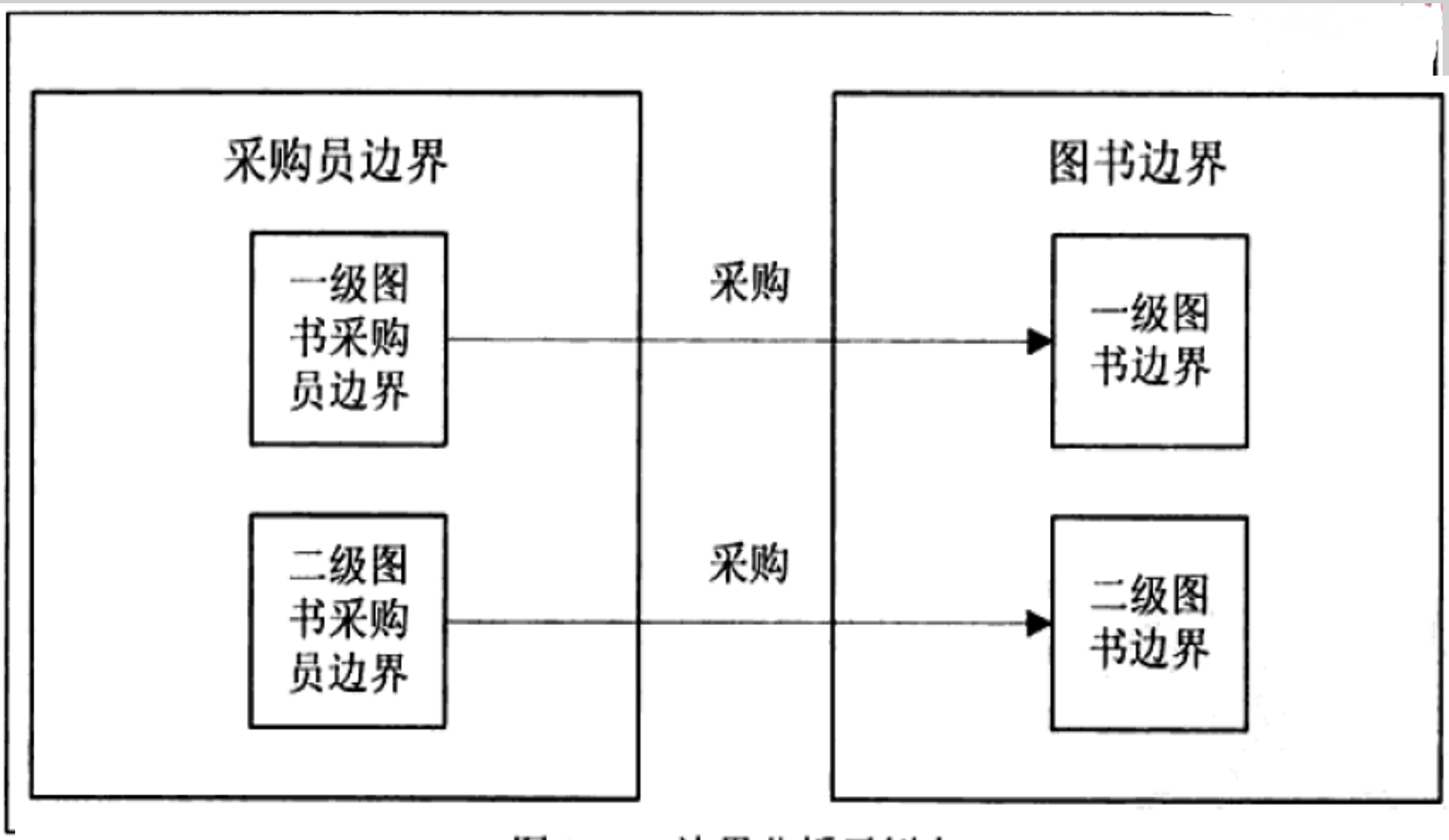

图 21.7　边界分析示例七

Outline

- 理解用况的本质
- 理解用况驱动
- 用况驱动与领域驱动
- 理解建模的抽象层次
- 划分子系统的问题
- 学会使用系统边界
- 学会从接口认知事物

怎样描述一件事物

- 如何描述电视机?
  - 一种观看动态影像的机器;
  - 通过电信号控制电子流打到荧光屏将电信号转换成光信号以显示图像的机器.
- 人们认识世界与认知软件的过程是一致的，从接口(表象)而不是从对象内部(本质)来认识对象的，不论在分析设计还是描述软件的过程中，接口及其交互比对象实现更重要，因为是它们决定了系统行为.

接口是系统的灵魂

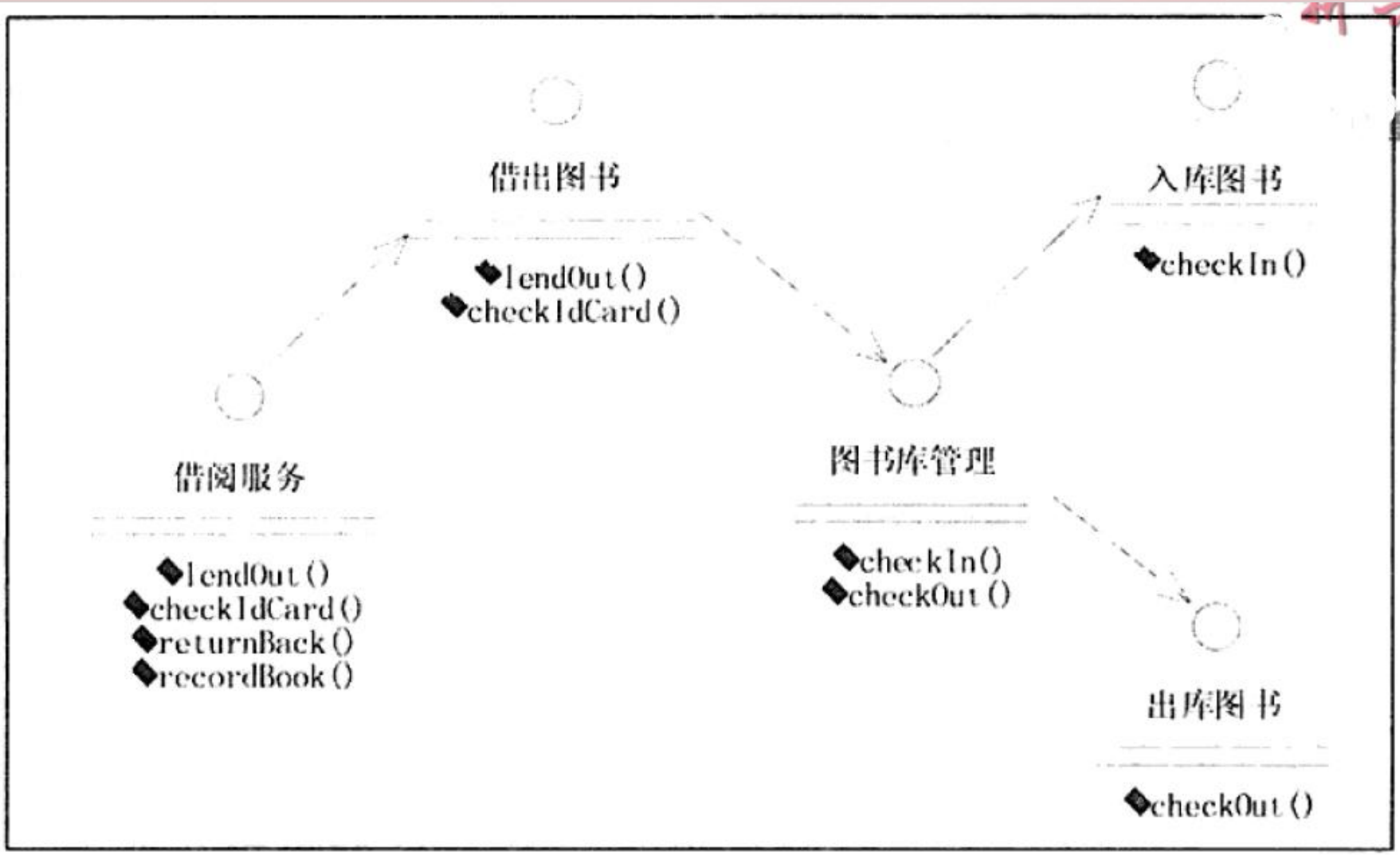

图 22.1　边界分析的接口形式

接口是系统的灵魂

- 考虑具体实现时确保能实现接口，再考虑内部实现；
- 可以简单的认为边界就是接口，接口有着属性和行为.
- 接口是系统的灵魂，学会使用接口来描述和定义系统；

Q & A

Thanks !