第四章　系统建模技术-结构化方法

一、结构化分析方法

要回答：如何定义问题？

就如何定义问题而言，如何获得需求

如何规约需求

如何验证需求

1、关于需求获取

需求面临的挑战

- 问题空间理解
- 人与人之间的通信
- 需求的不断变化

需求目标

在任何一个设计中，精确地陈述问题总是第一步的。需求的目标是要简洁而精确地说明所要解决的问题。

为此：

- 软件人员的注意力应在做什么和为什么做，而不是如何做。
- 与用户和该领域的专家进行交流，导引出他们对软件产品的要求。
- 基于对用户要求的理解，结合计算机软件的特有能力，创造出对用户有价值的，能提高产品的质量与可用性的新的产品要求。
- 分析所定出的产品要求，判断其正确性，一致性，完整性及可行性；
- 决定解决方案，完成高层次的设计，确定出功能子系统及子系统之间的接口界面。
- 把产品要求以用户手册及工程设计技术要求的形式表达出来。（可能还包括测试的标准）。用于在开发的全过程中，验证核实所开发的产品确能满足用户的要求，支持技术文档的管理，更重要的是支持需求变化的管理。

可见，为了实现这一目标：需求(工程)包括需求的引出、创造、分析、表述、验核和管理。

需求工程的原则

（1） 抽象：抓住事物的本质（要素）。其中，一个重要方面是：捕获问题空间的“一般/特殊”关系

是认识、构造问题的一般途径。

（2） 划分：分离问题。其中，一个重要方面是：捕获问题空间的“整体/部分”关系

是降低问题复杂性的基本途径之一

（3）投影：捕获并建立问题空间的多维“视图”

是描述问题的基本手段之一

是解决“A是B，B是A”的基本方法

●●●

需求获取技术特征

需求获取技术特征：

- 方便通讯（使用易于理解的语言）
- 提供定义系统边界的方法
- 提供划分、抽象、投影等方法
- 允许采用多种可供选择的设计方法
- 适应需求的变化
- 支持使用问题空间的术语，思考问题和编制文档

…

需求获取技术- Use Case

Use Case (Ivar Jacobson,1994)

1）引言

- Use Case主要用于促进和用户的交流、沟通。为此使用了一种用户和开发人员都能理解方式描述系统功能和行为。
- Use Case可以划分系统与外部实体的界限，是系统开发的起点，而最终应该落实到类和实现代码上。
- Use Case既然是对系统行为的动态描述，因此它是类、对象、操作的来源，是系统分析和设计阶段的输入之一，是分析和设计，制定开发计划，测试计划，设计测试用例的依据之一。
- Use Case Model是系统需求分析阶段的成果之一。
- Use Case不但有助于帮助分析员理清思路；验证用户需求。而且，也是开发人员之间进行交流的重要手段。

2）语义与表示

一般地说，USE CASE是用户为了达到某一目标和系统进行的典型交互。例如：

“做一次拼写检查”

“对一个文档建立索引”

对一个用况而言，关键要素是：表示一种用户可以理解并对该用户有价值的功能。

用况提供了客户和开发人员在制订项目计划中进行交流的主要成分。

（1） USE CASE语义

一个USE CASE是系统或其它语义实体（例如子系统或一个类）所提供的一块(unit)高内聚的功能，显露该系统和一个或多个外部的交互者（称为操作者）交替出现的消息序列，以及该系统所执行的动作。

可见，一个USE CASE捕获了参与交互的各方关于其行为的一个约定。通过这一约定，描述了该语义实体在不同条件下的行为对参与者一个要求的响应，以实现某一目的。不同的行为序列，依赖于所给出的特定要求以及与这些要求相关的条件。

（2）表示与描述

USE CASE通常被表示为：

USE CASE包含一组操作和属性，这些操作和属性规约了该USE CASE的实例所执行的那个动作序列。动作包含状态的改变以及该USE CASE与其环境的通讯。

为了表明USE CASE所包含的具体内容，还应给出它的正文描述。即：

USE CASE 中包含的信息

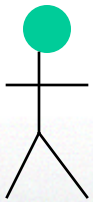
- 名称(Name)
- 标识(Identifier)
- 描述(Description)
  - 角色(Actor)
  - 状态(Status)
  - 活动及时序
  - 频度(Frequency)
  - •••

3）操作者语义与表示

一个操作者定义了一组高内聚的角色，当用户与该实体交互时，用户可以扮演这一角色。

对于每一USE CASE，一个操作者有一种角色，即每一USE CASE与具有一种角色的操作者进行通讯。

通常，一个操作者被表示为：



4）USE CASE获取

- 仿真法（Simulation）
  - 掌握用户的所有输入与输出的数据种类，通过仿真的方法，找出它们之间的对应关系，与及相应的数据处理过程。包括任何计划中将要新增加的数据类型与处理过程。
- 原型法（Prototyping）
  - 从用户处取得一组基本的USE CASE产品要求之后，

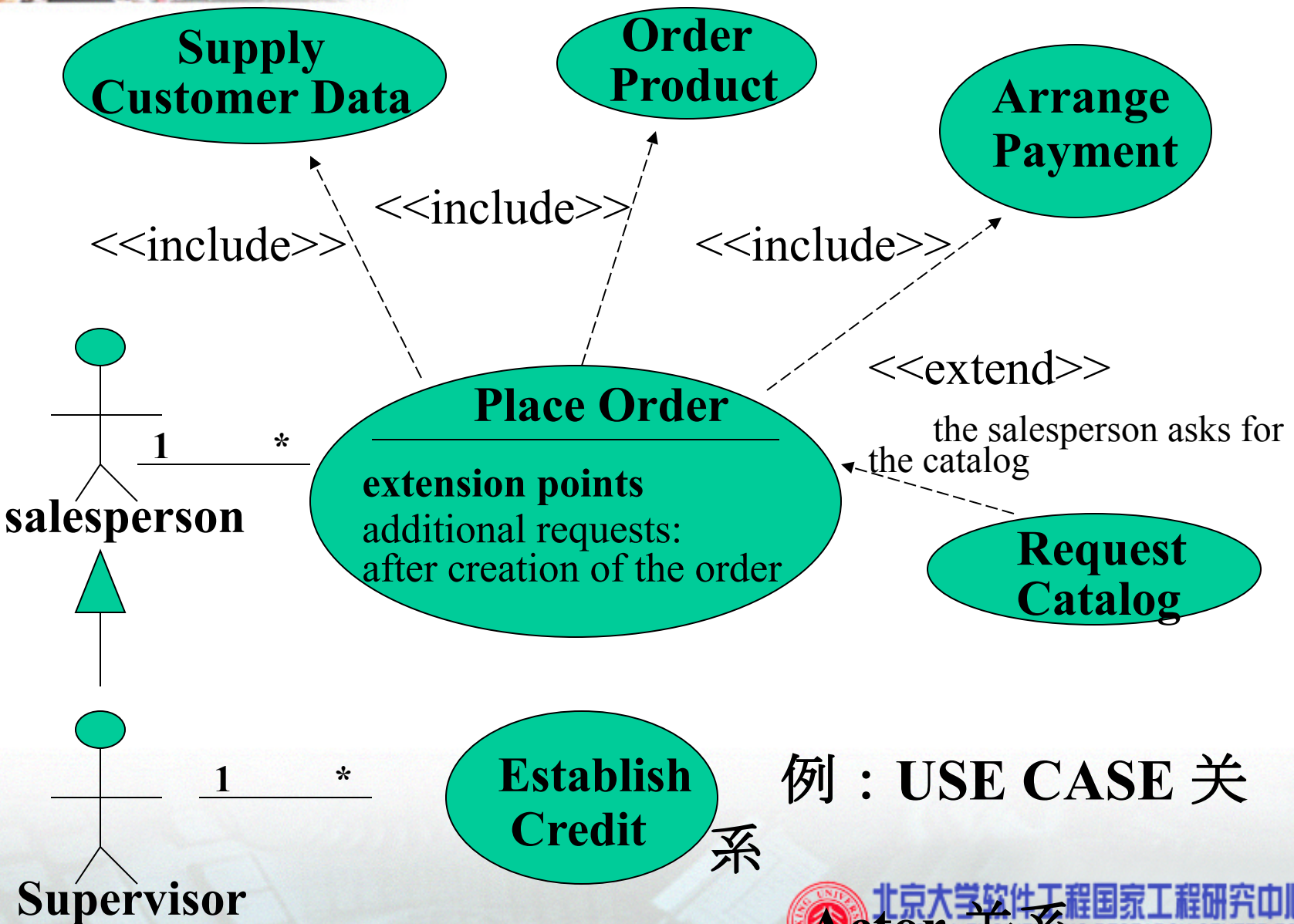
    立即建造USE CASE产品的原型（这个原型可以是实际可运行的软件（外壳），或是一个用描述来表达的产品），然后让用户去模拟使用这个原型，提出修改的意见。其中，值得注意的是，不要忽略将要新增加的数据类型与处理过程。
- 场景法（Scenario Generation）
  - 让用户穷举他们现有的所有的数据处理实践以及任何计划中将要新增加的数据类型与处理过程。

从以上三种方法中可以看出，Use Case 的功能划分均要以角色为主体，行为是角色触发的。

5）关系

在USE CASE之间，或在操作者与USE CASE之间，存在一些标准的关系：

- 关联：参与关系，即操作者参与一个USE CASE。例如，操作者的实例与USE CASE实例相互通讯。关联是操作者和USE CASE之间的唯一关系。
- 扩展：USE CASE A到USE CASE B的一个扩展关系，指出了USE CASE B的一个实例可以由A说明的行为予以扩展（根据该扩展所说明的特定条件），并依据该扩展点定义的位置，A说明的行为被插入到B中。
- 包含：USE CASE A到USE CASE B的一个包含，指出A的一个实例将包含B说明的行为，即这一行为将包含在A定义的那部分中。
- 泛化：USE CASE A到USE CASE B的泛化，指出A是B的特殊情况。



例：USE CASE 关系

Actor关系

6）USE CASE图

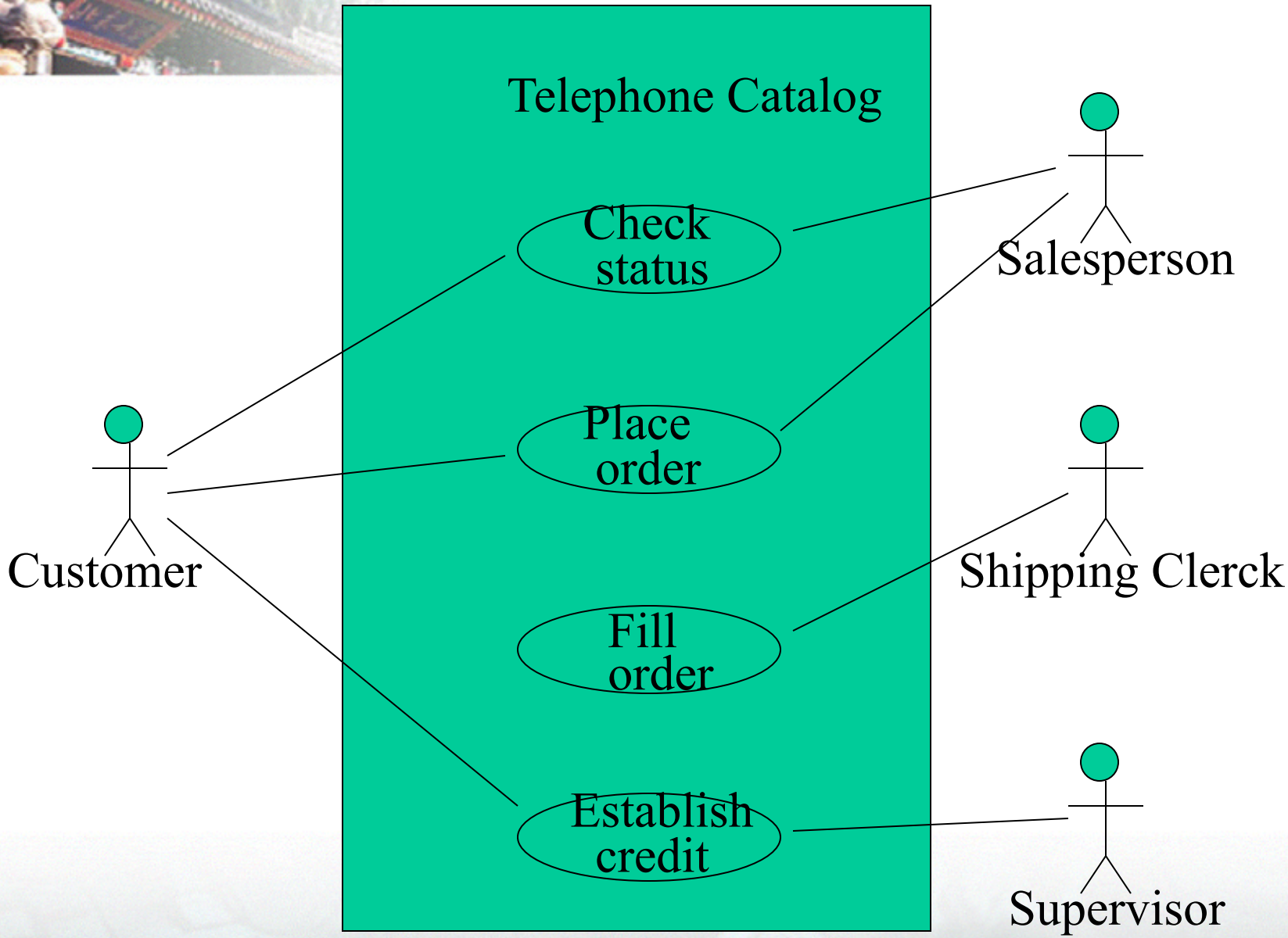
USE CASE图给出了操作者和USE CASE以及它们之间的关系。即图中给出了一些操作者、一组关系、一些接口和这些元素之间的关系。

(关系是操作者和USE CASE之间的关联

是操作者之间的泛化

是USE CASE之间的泛化、扩展和包含)

可以将一些USE CASE用一矩形括起，以表示所包括的那个系统或其它语义实体的边界。



例：USE CASE图

7）使用USE CASE图的建模类型

使用USE CASE图所建造的模型类型，可以从两个层面上进行分类，它们是“整体/部分”关系

- 系统建模（system modeling ）
  - 系统建模用于描述软件系统的结构和行为
- 业务建模（business modeling）
  - 业务建模用于企业或组织过程的优化和再工程（process re-engineering）
  - 业务建模的图形元素不仅包括普通的Actor和Use Case，还包括Worker、Artifact，Business
  - 过程描述时还应结合时序图(sequence diagram)和活动图(activity diagram)

2、关于需求规约

需求规约的主要目标：

依据需求陈述（作为输入），解决其中的歧义、不一致等问题，以系统化的形式表达用户的需求，即给出问题的形式化或半形式化的描述（建立模型），形成需求规格说明书。为了实现这一目标，

（一）结构化分析方法

1〕提出的概念有：

数据流：——加工：

数据存储：——数据源：　　数据潭：

概念是完备的。

2〕建模过程

(1)建立系统的功能模型

---使用的工具为数据流图DFD

首先：建立系统环境图，确定系统边界

继之：自顶向下，逐层分解

(2)建立数据字典

定义数据流　　定义数据存储

定义数据项

(3)给出加工小说明

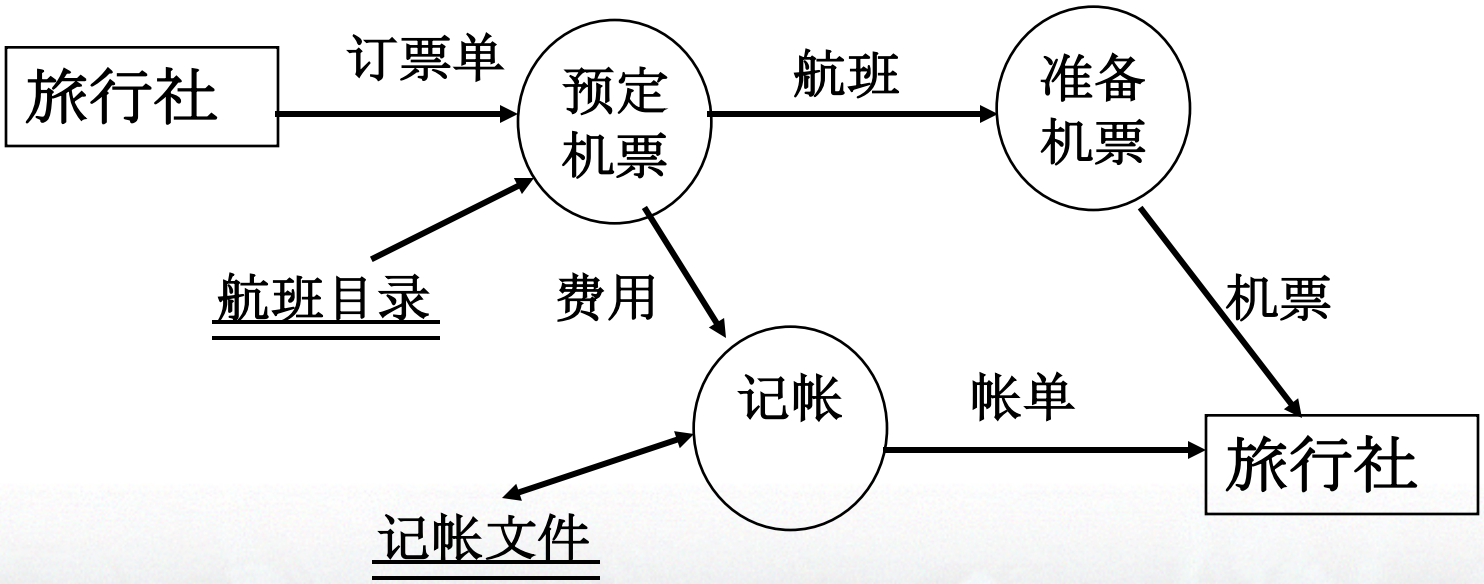
---使用的工具可以为判定表

判定树

(1)建立系统的功能模型

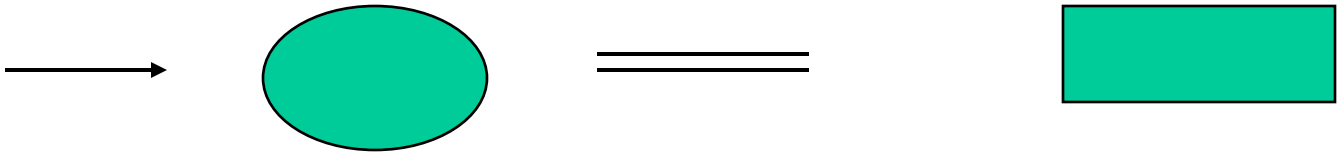
---使用的工具为数据流图DFD

数据流图：是一种描述数据变换的图形工具。例如：



数据流图由四个基本成分组成：

数据流　加工　数据存储　数据源和数据潭



其中：1 各成分的定义

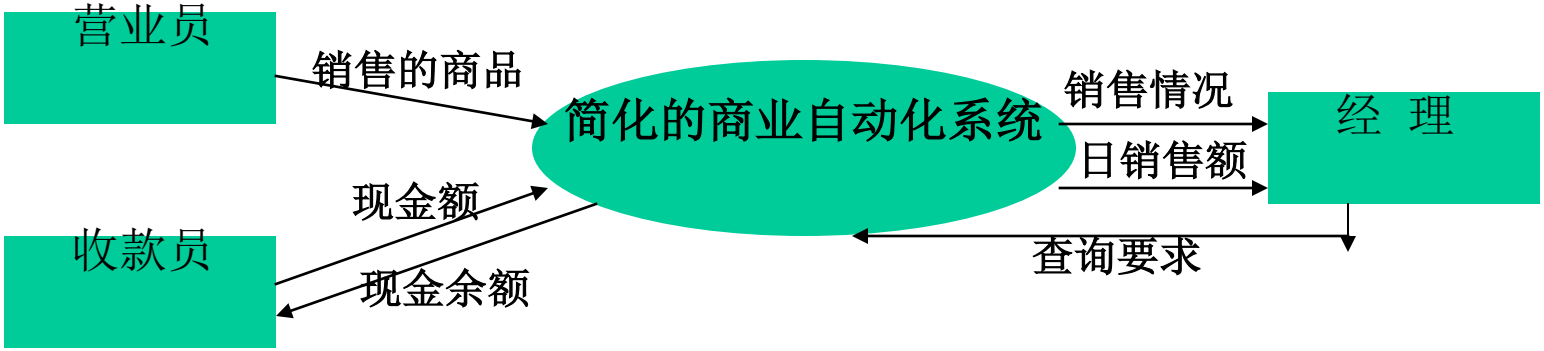
2 数据流、数据存储---支持数据抽象

加工---支持过程/功能的抽象

3 关于命名问题

首先：建立系统环境图，确定系统边界

------顶层DFD



其中：1 数据流为：销售的商品，日销售额等

3个输入流，3个输出流

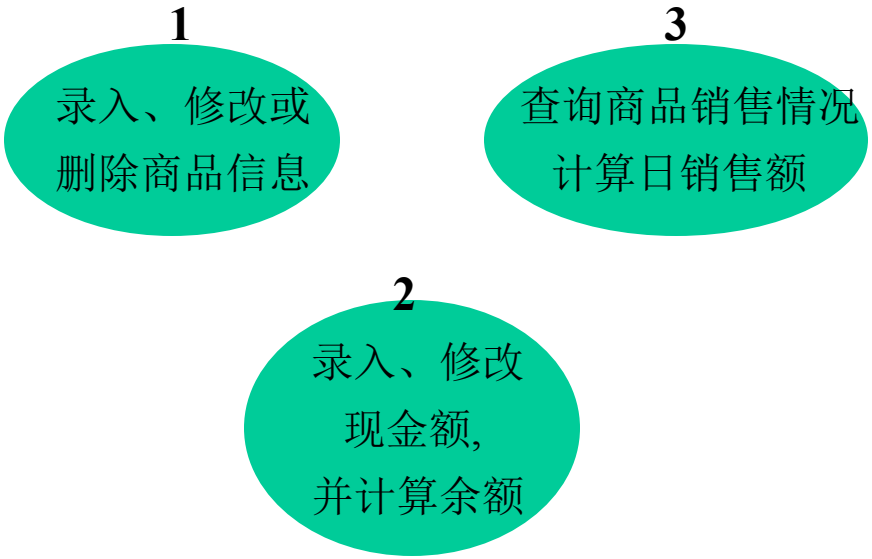
数据源为：营业员，经理，收款员

数据潭为：经理，收款员

2 加工名为：要建立的系统名字

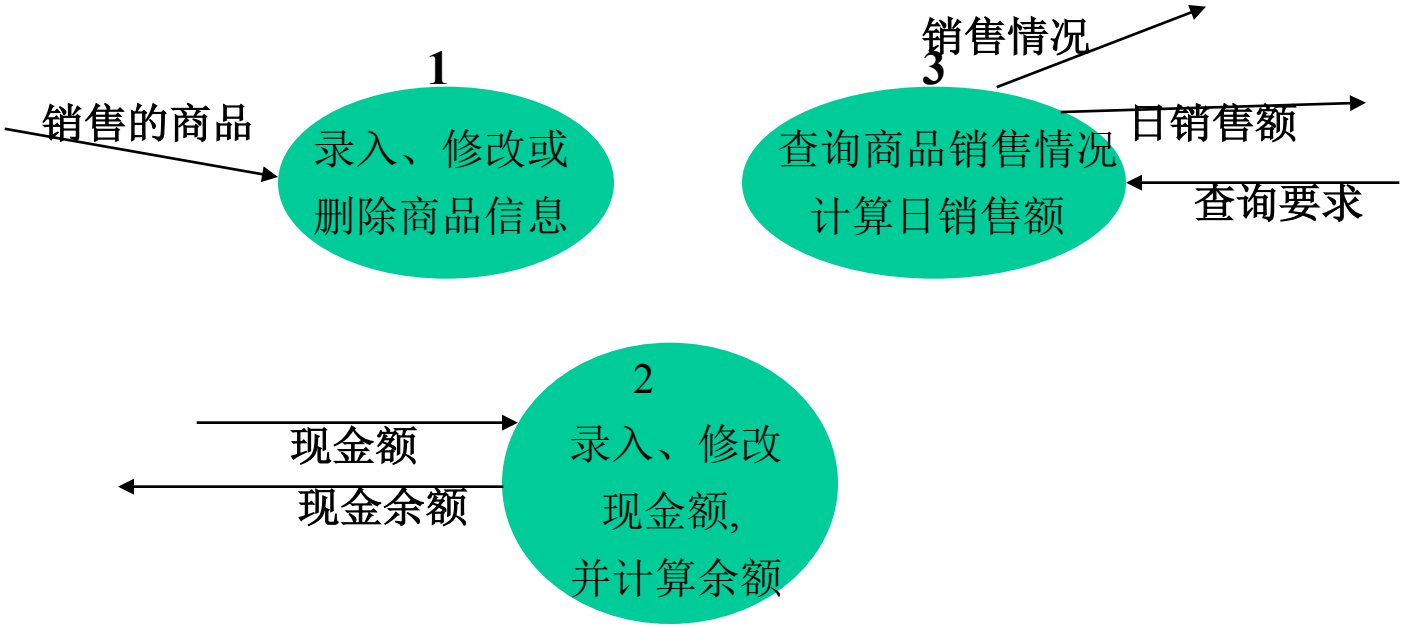
继之：自顶向下，逐层分解

A、按人或部门的功能要求，将加工“打碎”，形成：



注：需给每一加工编号；

B、“分派”数据流，形成：

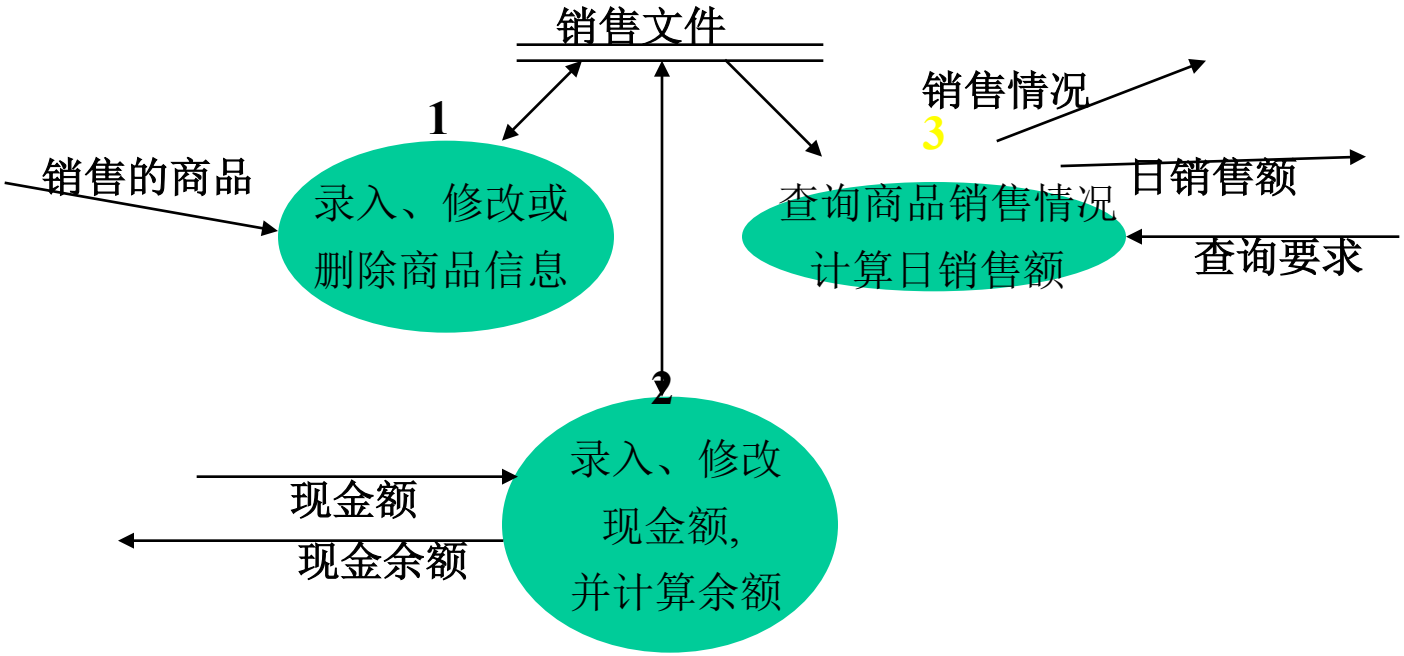


其中：要根据特定的加工要求进行分派；

保持与顶层数据流的一致；

可以不引入数据源和数据潭。

C、引入文件，使之形成一个有机整体—系统：

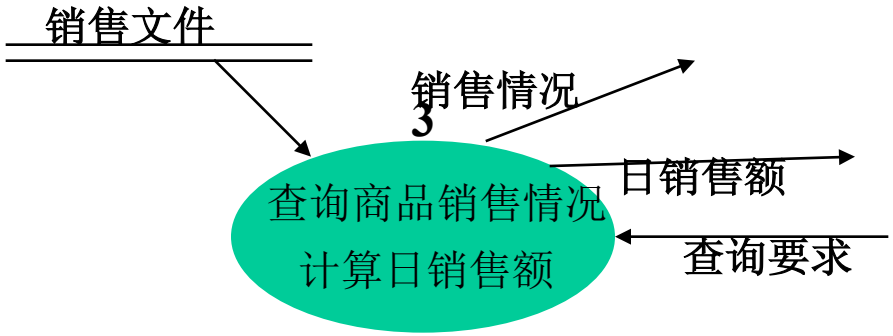


注：到一个文件，既有输入流，又有输出流，则可简化为

←，并可不给出标识。

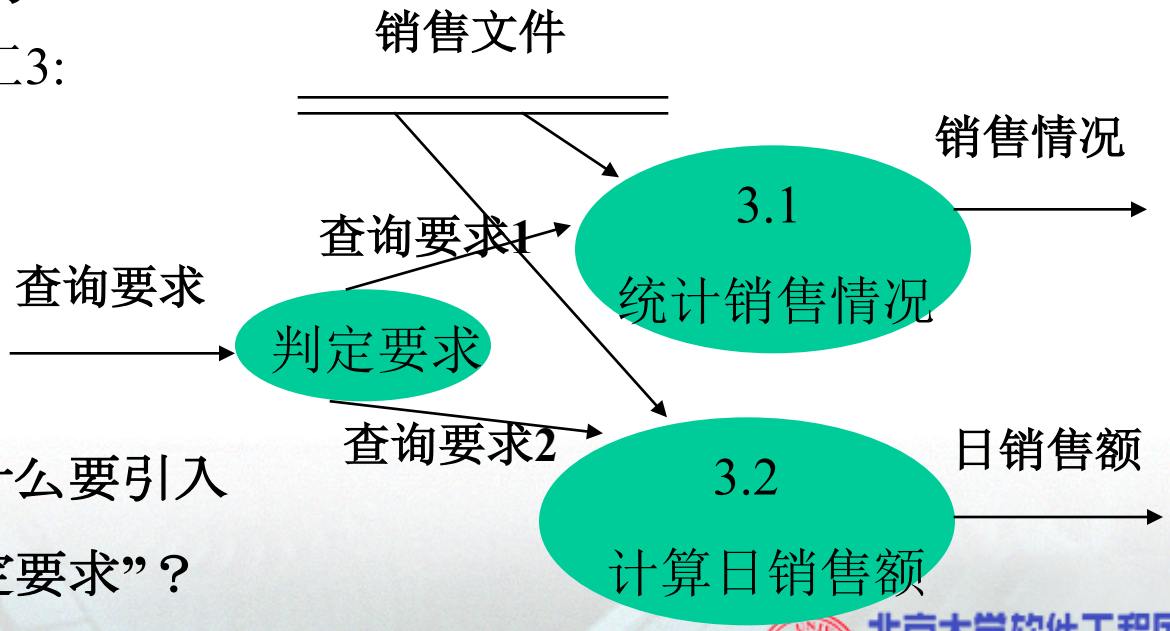
至此，体现精化，形成0层数据流图。

继续A、B、C：自顶向下，逐层分解。例如：加工3



可分解为：

加工3:



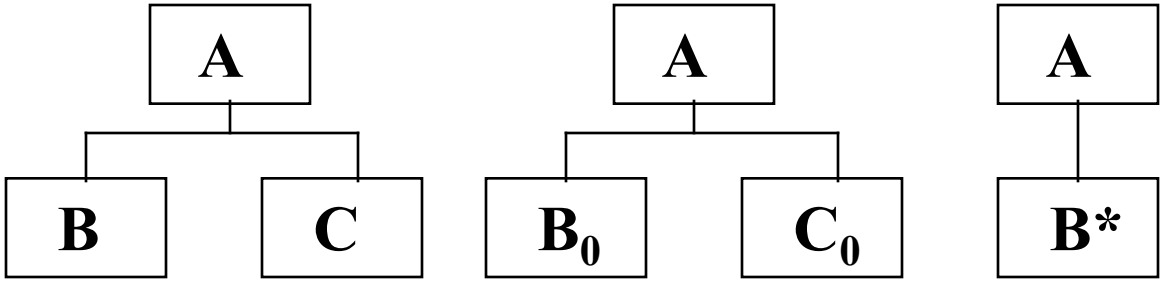
*其中为什么要引入加工“判定要求”？

(2)建立数据字典

定义数据流　定义数据存储　定义数据项

引入：结构符　＋　　|　　{ }

—用于定义数据结构



数据字典:

1、数据流:

销售的商品=商品名+商品编号+单价+数量+日期

现金额 = 余额 =日销售额=非负实数

查询要求=[商品编号|日期]

查询要求1=商品编号

查询要求2=日期

销售情况=商品名+商品编号+金额

2、数据存贮:

销售文件={销售的商品}

3、数据项

(3)给出加工小说明

---使用的工具可以为判定表

判定树

判断表　Ⅰ 条件类别　Ⅱ 条件组合

Ⅲ 操作　Ⅳ 操作执行

例如：

考试总分	>=620	>=620	<620
单科成绩	有满分	有不及格	有满分
发升级通知书	y	y	n
发留级通知书	n	n	y
发重修通知书	n	y	n

3〕建模中注意的问题

（1）模型平衡规则

· 父图和子图必须平衡

· 每个数据流和数据存储必须在数据字典中予以定义

· “叶”加工（最低层）必须给出加工小说明

· 小说明和数据流图的图形表示必须一致，例如：

在小说明中，必须说明“输入数据流”如何使用，必须说明如何产生“输出数据流”，必须说明如何选取、使用、修改“数据存储”

（2）控制复杂性规则

· 上层数据可以“打包”

上、下数据流对应关系在数据字典中给出，但包内数据流的性质（输入、输出）必须一致。

· 一幅图中的图元个数应控制在7+/-2以内

· 与每一加工相关的数据流的数目应适中

（与层次有关）

· 分析数据内容，确定是否所有的输入信息都用于产生输出信息；

分析加工，确定一个加工所产生的输出，是否都能由该加工的输入信息导出

实例讲解：图书管理系统—问题陈述见P35。

根据问题陈述，在一定的层次上，可以把该系统分为两“大块”，即：借还书等事务的处理，以及咨询事务处理。

---进行功能抽象。

（注：不同的功能抽象将导致不同的结果！但应该是等价的。）

于是，可以根据这一抽象，可以识别：

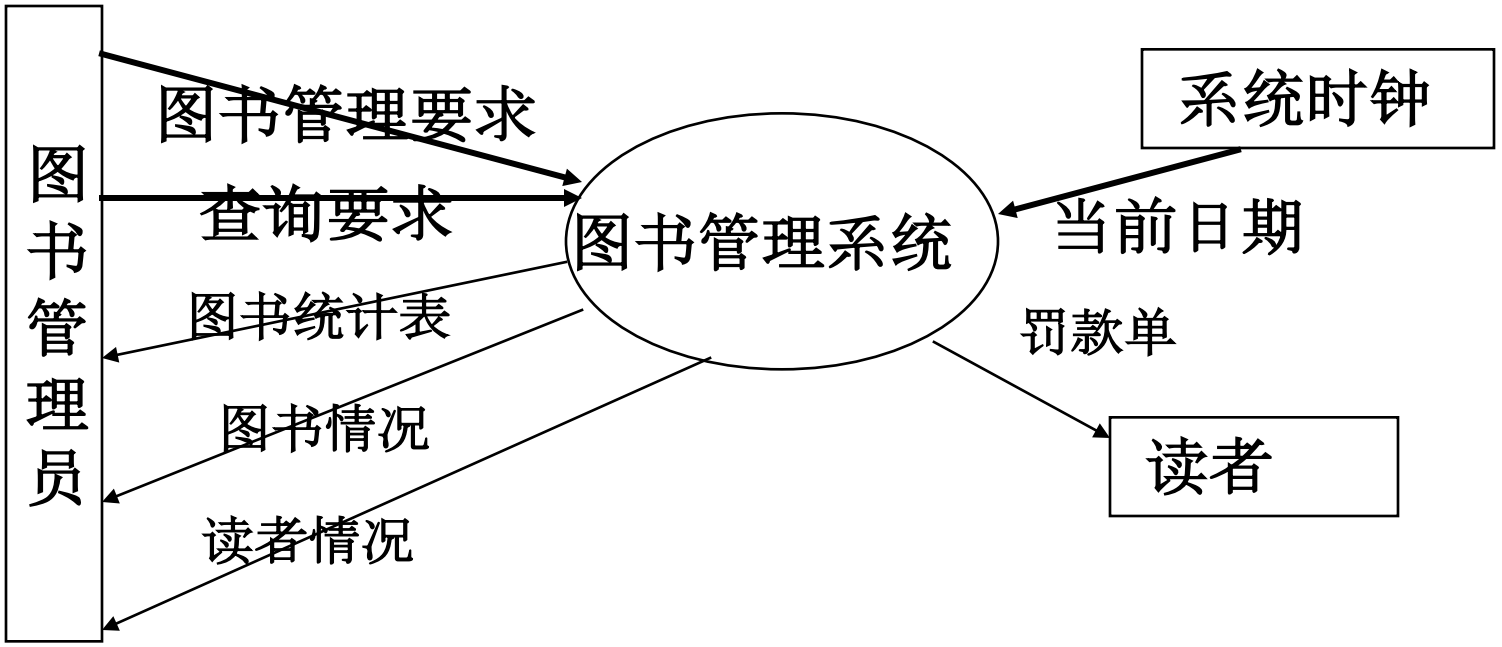
1）顶层数据流：借还书等事务处理要求

咨询事务要求

以及相关的数据流

2）数据源和数据潭为：图书管理人员，读者以及时钟。

基于以上分析，可形成该系统的环境图：



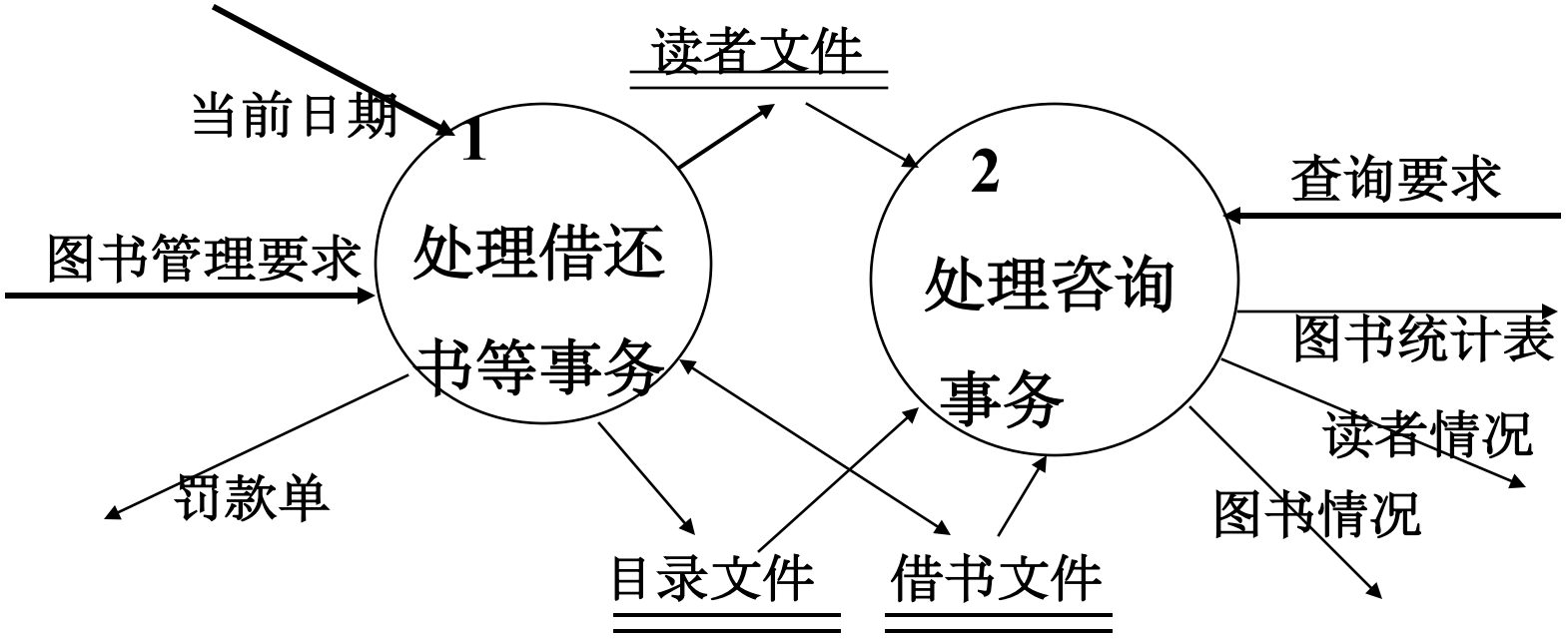
其中：<u>3个输入流</u>：图书管理要求，查询要求，系统时钟

图书管理要求=入库单 借书单 还书单 注销单

查询要求=读者情况 图书情况 图书统计表

<u>4个输出流</u>：图书统计表，图书情况，读者情况、罚款单

通过“打碎”、“分派”，可形成如下0层DFD：

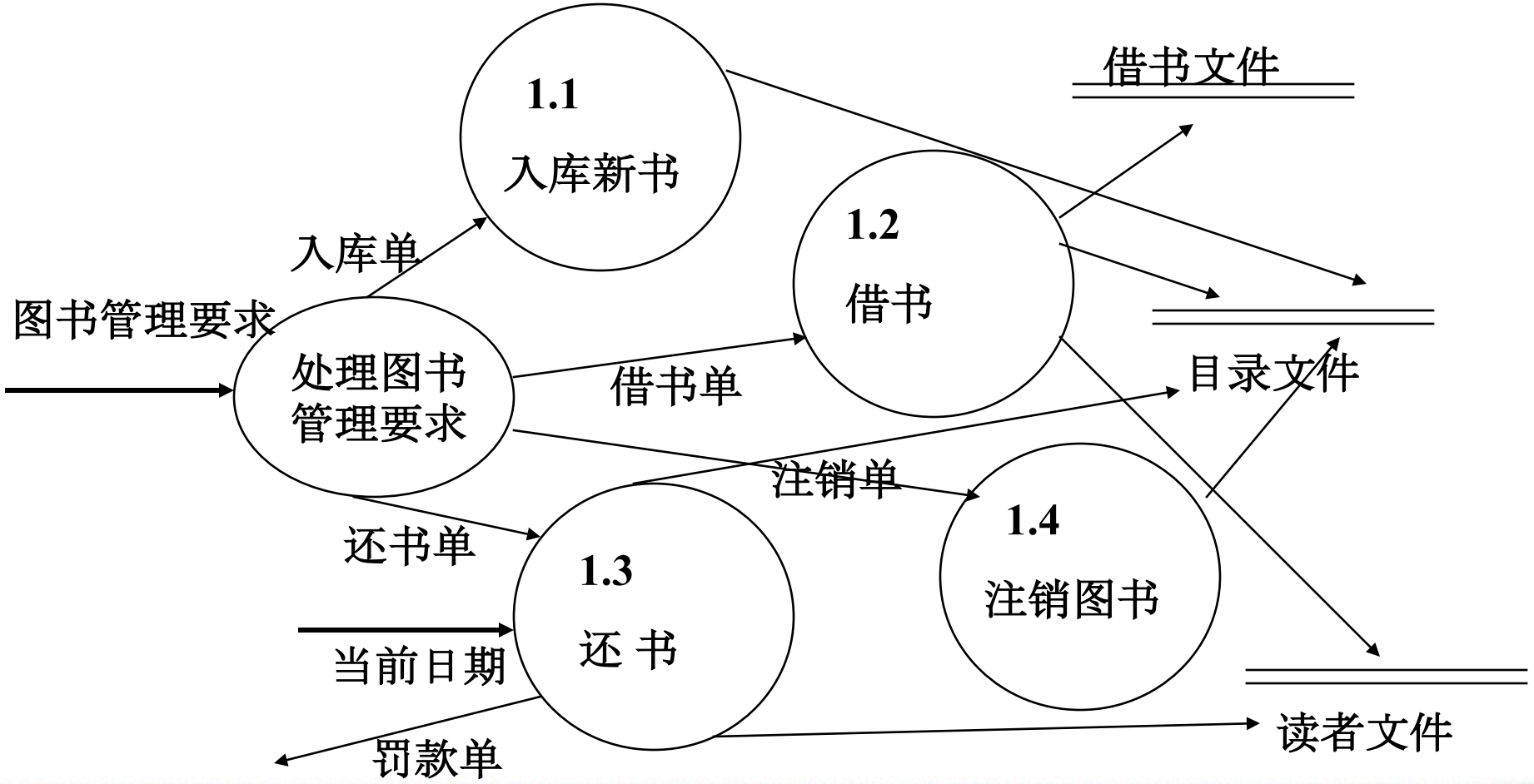


其中：保持输入与输出的一致；

引入三个文件，对顶层DFD进行细化。

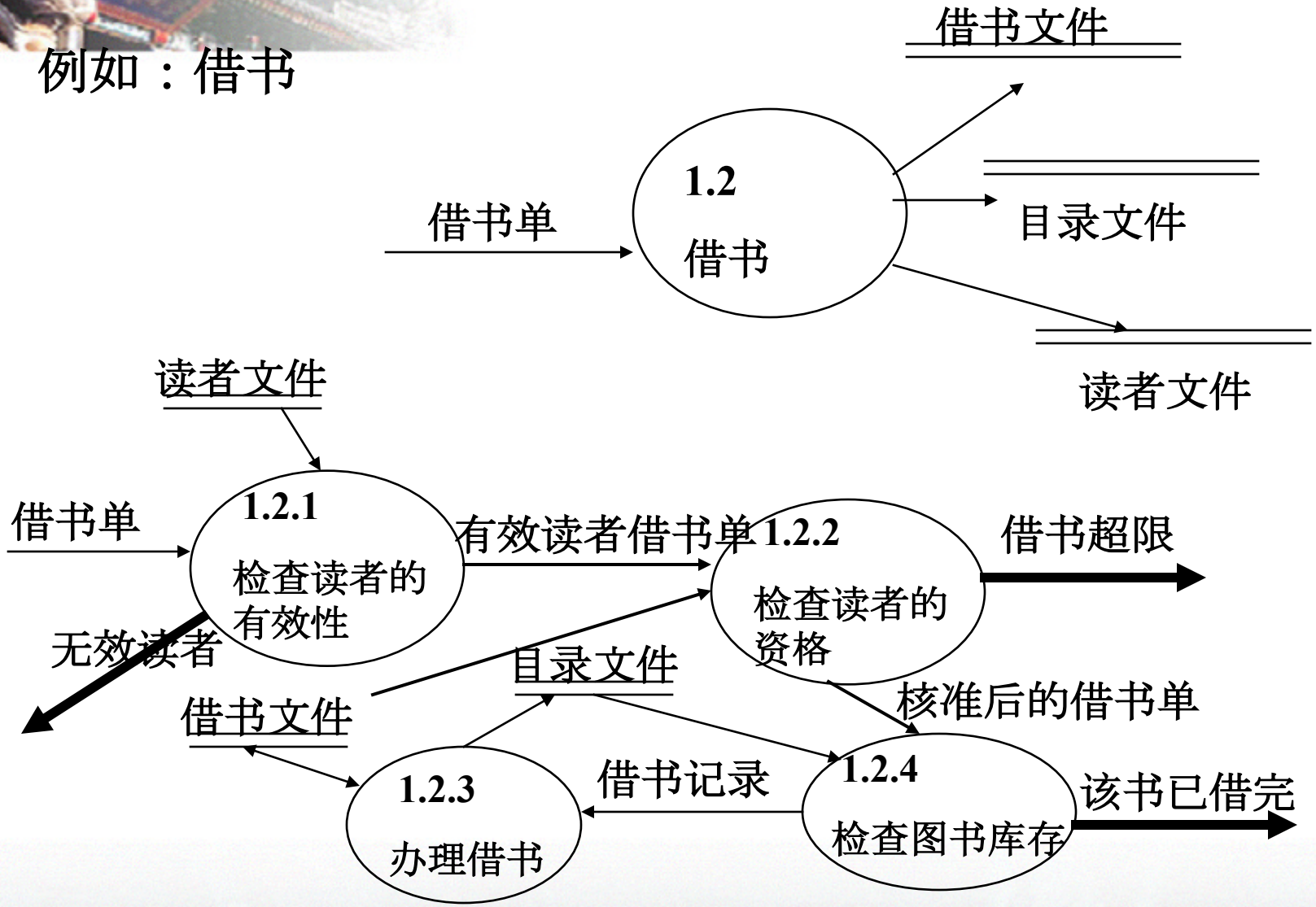
（注：存在数据库设计问题）

以同样方式，对加工1进行分解，形成：



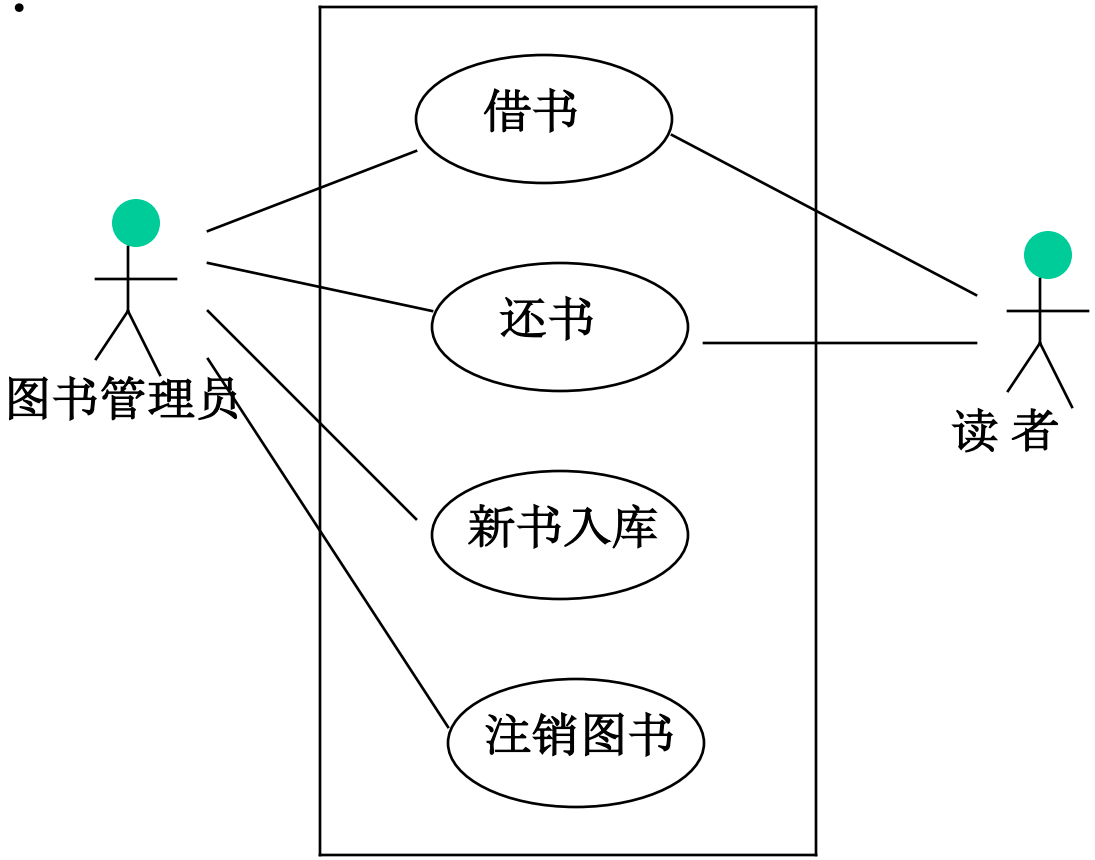
其中：注意平衡问题；平面化问题。当然，还可以继续细化

例如：借书



注意：其中粗线数据流！

图书管理系统的USE CASE模型（基本思路）：



请：1、在这一思路的基础上，建立该系统的USE CASE模型；并给出每一USE CASE的描述。

2、对该系统的两种模型进行比较。（在哪些方法进行？）

有关SRS内容方面：

（1）正确性：指的是SRS中陈述的每个需求是否都表达了系统的某个要求。

（2）无二义性：指的是SRS中陈述的每个需求是否都只有一种解释。

（3）完整性：

· 未来系统所做的任何事情都包含在SRS的陈述中；

· 未来系统响应所有可能的输入（包括有效和无效）；

· SRS中没有被标识为“待定”的内容。

（4）可验证性：SRS中陈述的每个需求都是可验证的-即当且仅当存在一个有限代价的过程（人工或机器）可以检查构造的软件产品是否符合用户的需求。

（5）一致性：

· SRS中陈述的需求没有与以前的文档发生冲突；

· SRS中陈述的各个需求之间没有发生冲突。

（6）可理解性：

有关SRS格式与风格方面

（7）可修改性：指的是SRS的结构和风格使任何对需求的必要修改都易于完整、一致的进行。

（8）可被跟踪性：指的是SRS中的每个需求的出处都是清楚的，这意味着SRS中包含对前期支持文档的引用表。

（9）可跟踪性：指的是SRS的书写方式有助于对其中陈述的每个需求进行引用。

（10）设计无关性：指的是SRS不暗示特定的软件结构和算法。

二、结构化设计

要回答如何解决问题

—即给出软件解决方案

1〕总体设计的任务：如何将DFD转化为MSD

分二步实现：

第一步：如何将DFD转化为初始的MSD

分类：变换型数据流图

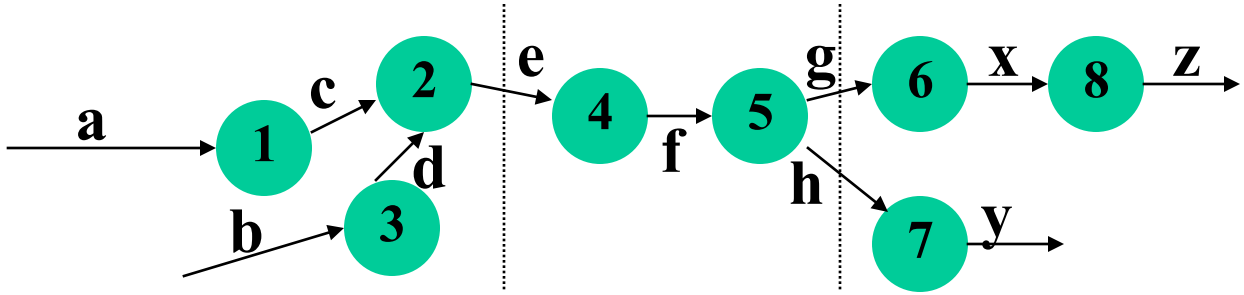
事务型数据流图

变换设计

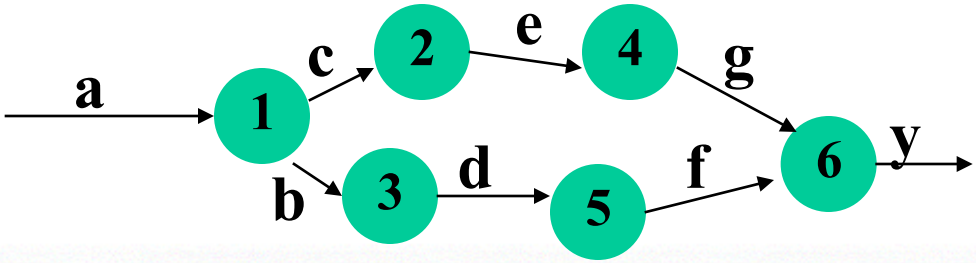
事务设计

●数据流图分类

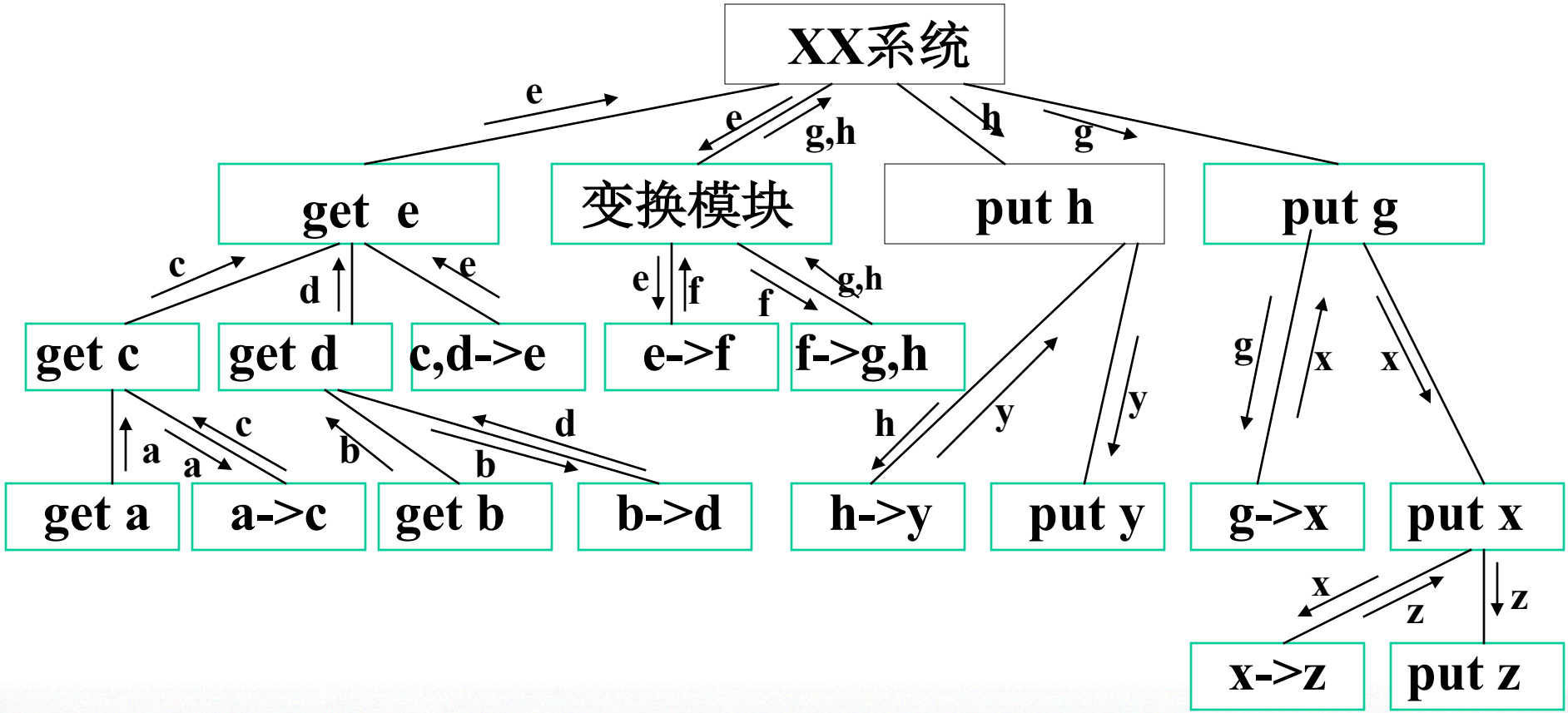
变换型:



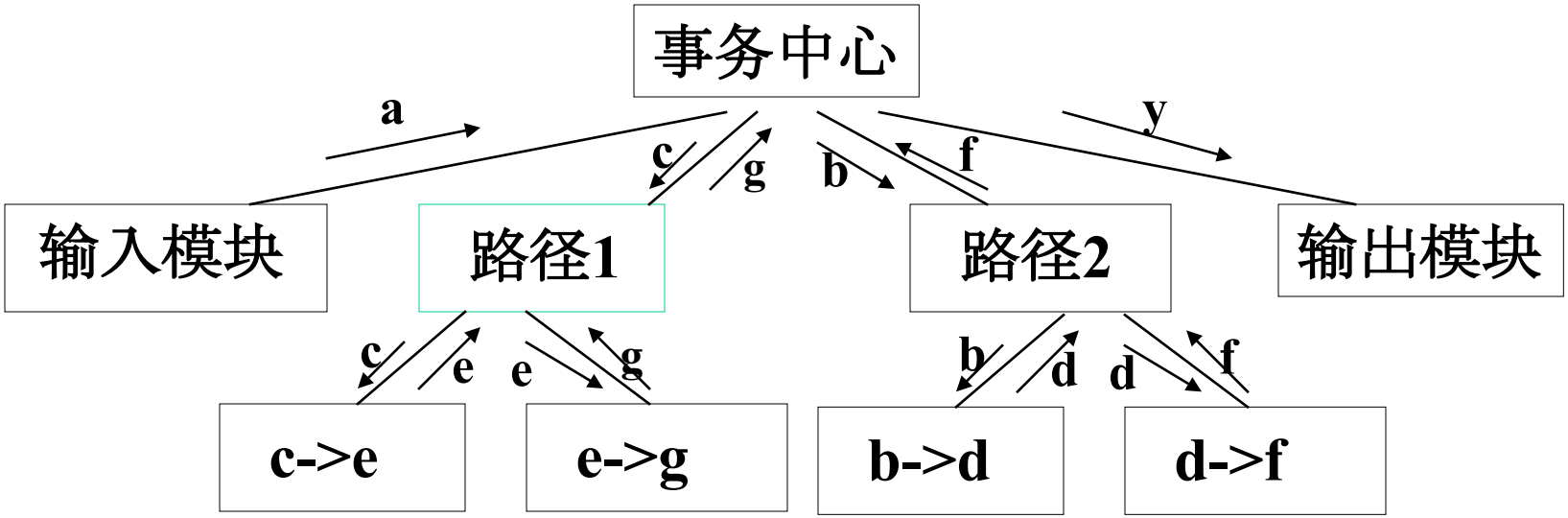
事务型



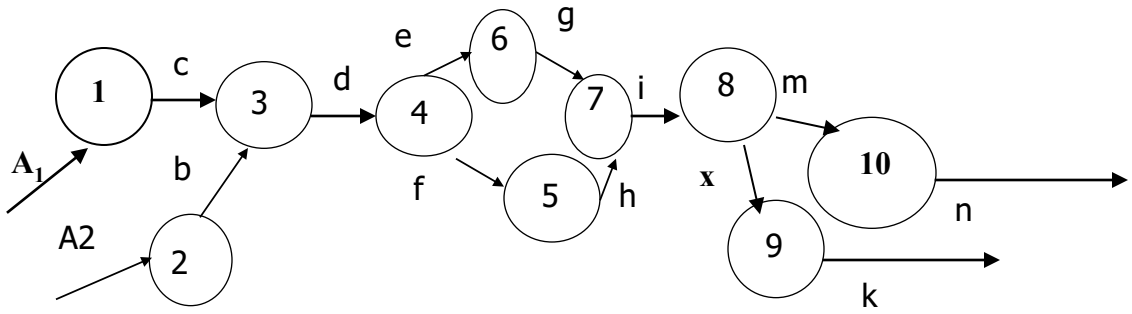
●变换设计



事务设计



一个系统的DFD，通常是变换型数据流图和事务型数据流图的组合。如下所示：



第二步：如何将初始的MSD转化为最终可供详细设计使用的MSD

- 概念：模块
- 模块化

  模块化度量：内聚 耦合
- 设计规则－经验规则
- 精化初始的MSD

  －体现设计人员的创造

1）耦合：不同模块之间相互依赖程度的度量。

耦合类型：

（1）内容耦合：

（2）公共耦合：两个以上的模块共同引用一个全局数据项。

（3）控制耦合：一个模块向另一模块传递一个控制信号，接受信号的模块将依据该信号值进行必要的活动。

（4）标记耦合：两个模块至少有一个通过界面传递的公共有结构的参数。

（5）数据耦合：模块间通过参数传递基本类型的数据。

2）内聚：一个模块之内各成分之间相互依赖程度的度量。

内聚类型：

（1）偶然内聚：一个模块之内各成分之间没有任何关系。

（2）逻辑内聚：几个逻辑上相关的功能放在同一模块中。

（3）时间内聚：一个模块完成的功能必须在同一时间内完成，而这些功能只是因为时间因素关联在一起。

（4）过程内聚：处理成分必须以特定的次序执行。

（5）通信内聚：各成分都操作在同一数据集或生成同一数据集。

（6）顺序内聚：各成分与一个功能相关，且一个成分的输出作为另一成分的输入。

（7）功能内聚：模块的所有成分对完成单一功能是最基本的，且该模块对完成这一功能而言是充分必要的。

启发性规则-经验的总结

（1）改进软件结构，提高模块独立性；

（2）模块规模适中-每页60行语句；

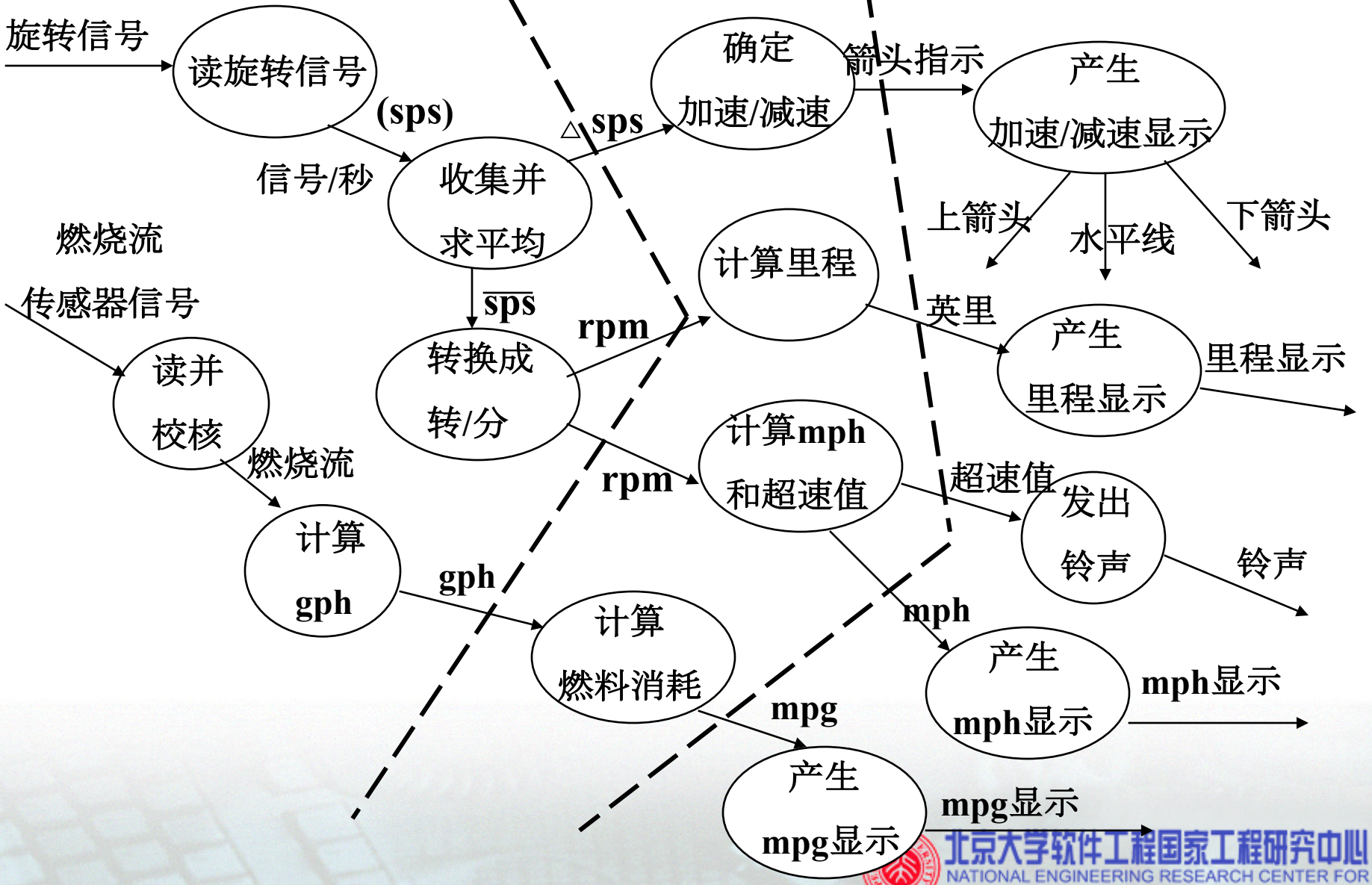
（3）深度、宽度、扇入和扇出适中；

（4）模块的作用域力争在控制域之内；

（5）降低模块接口的复杂性；

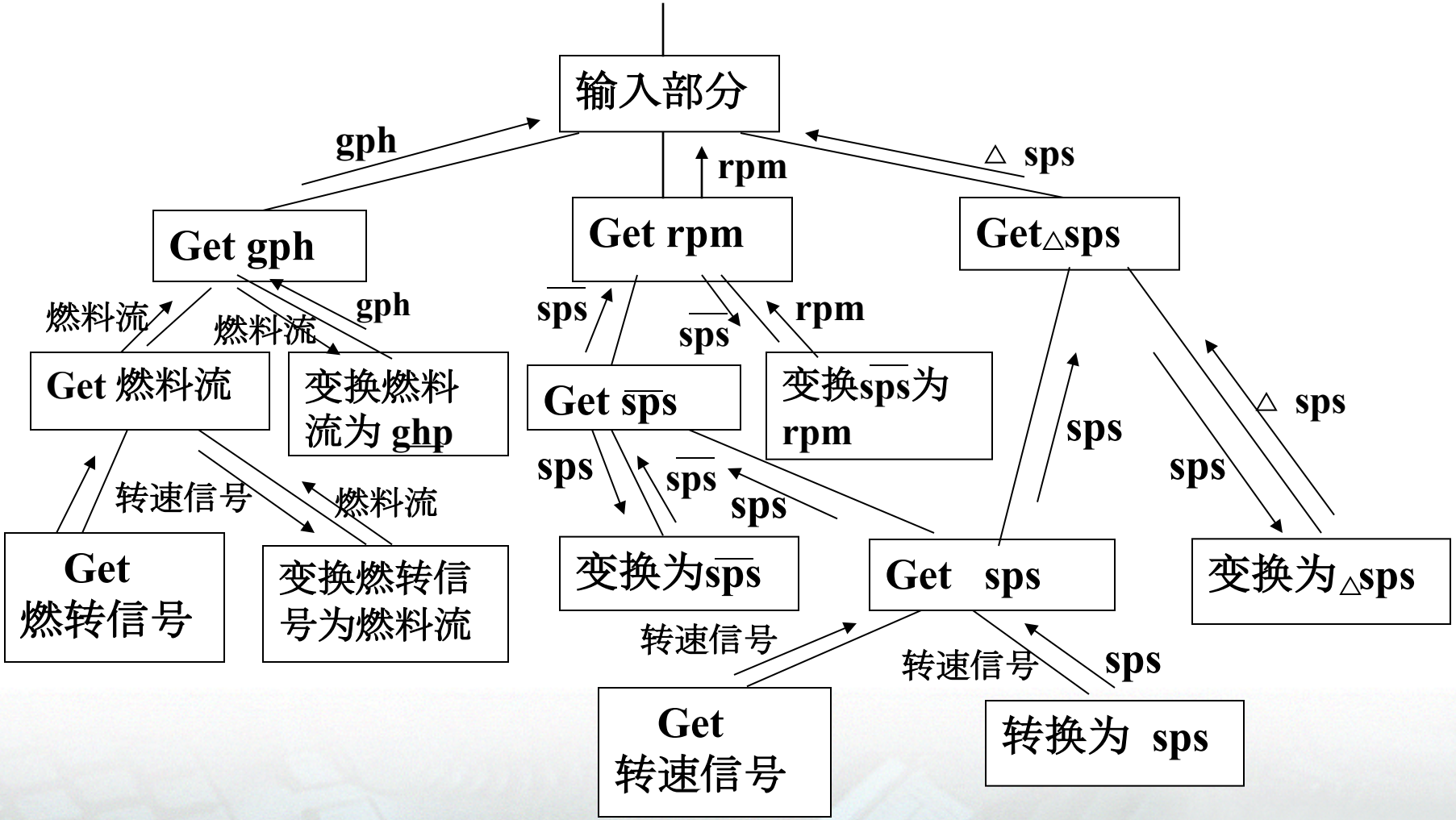
（6）模块功能应该可以预测。

示例：数字仪表板系统的精化

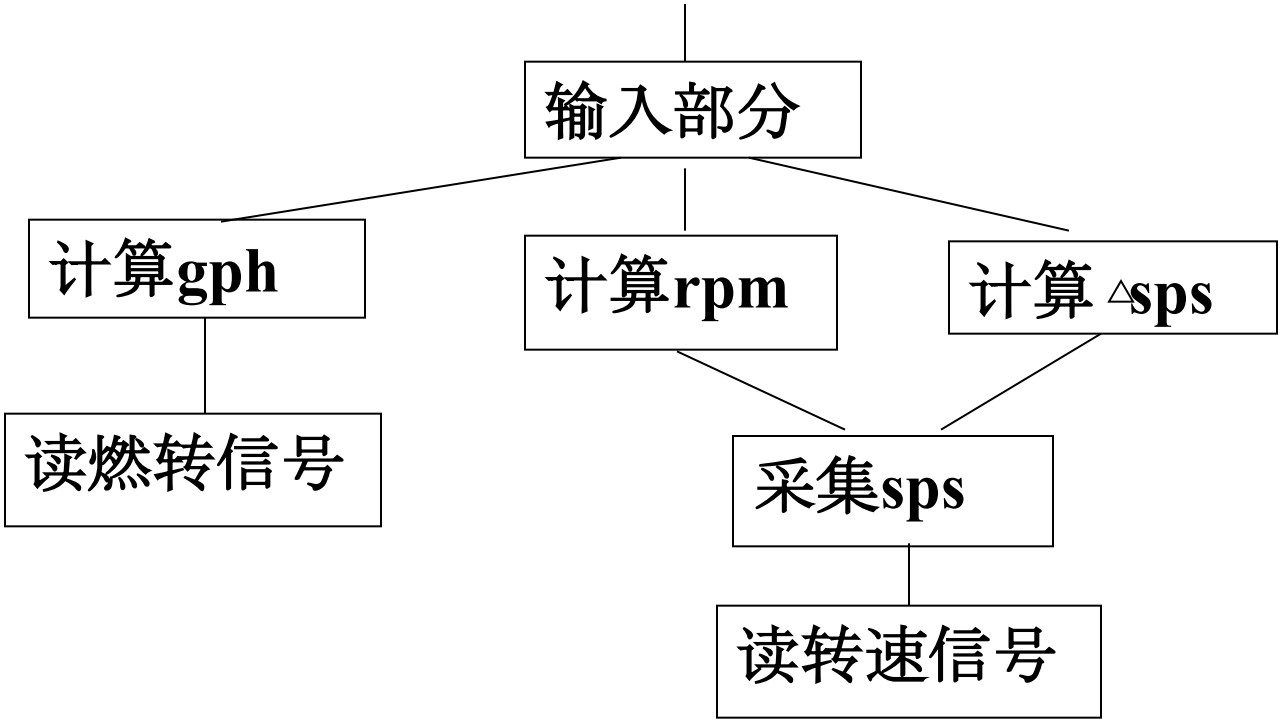


数字仪表板系统输入部分的精化

输入部分的初始模块结构图



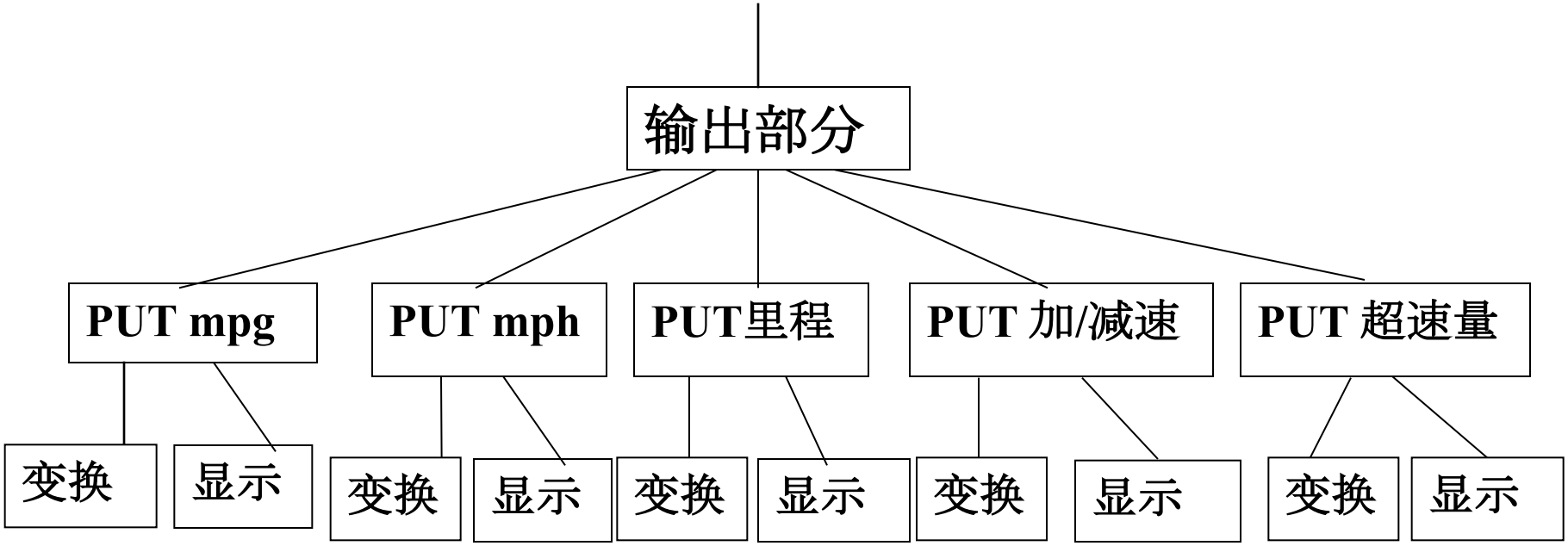
使用启发式规则1，并考虑其它规则，
可以将输入部分的模块结构图精化为：



其中：sps为转速的每秒信号量；sps 为sps的平均值； sps为sps的瞬时变化值；rpm为每分钟转速；mph为每小时英里数；gph为每小时燃烧的燃料加仑数；rpm为行进里程。

2、 数字仪表板系统输出部分的精化

输出部分的初始模块结构图



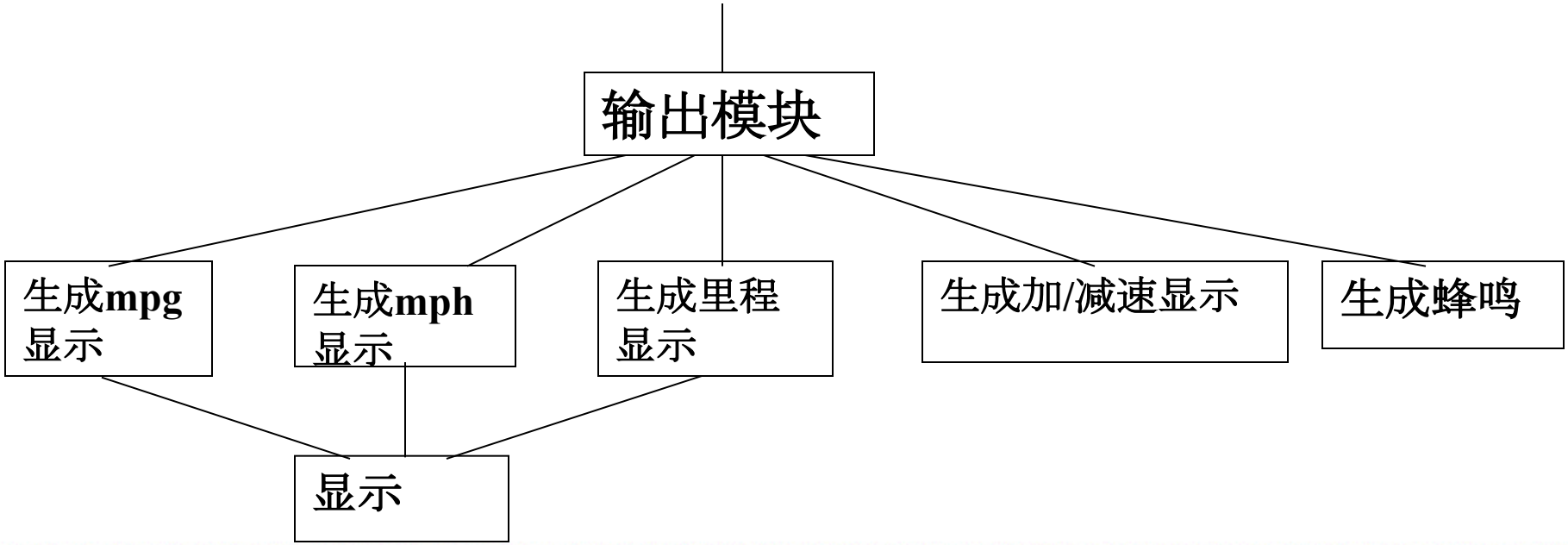
对于这一初始的模块结构图，一般情况下应：

- 把相同或类似的物理输出合并为一个模块，以减少模块之间的关联。就本例而言：

左边前三个“显示”，基本上属于相似的物理输出，因此可以把它们合并为一个显示模块。而将“PUT mpg”模块和相关的“生成显示‘的模块合并为一个模块；同样地，应把“PUT mph”模块、“PUT里程”各自与相关的生成显示的模块合并为一个模块，参见下图。

- 其它求精的规则，与输入部分类同。例如，可以将“PUT 加/减速”模块与其下属的两个模块合并为一个模块，将“PUT 超速量”模块与其下属的两个模块合并为一个模块。

通过以上求精之后,可得如下的模块结构图

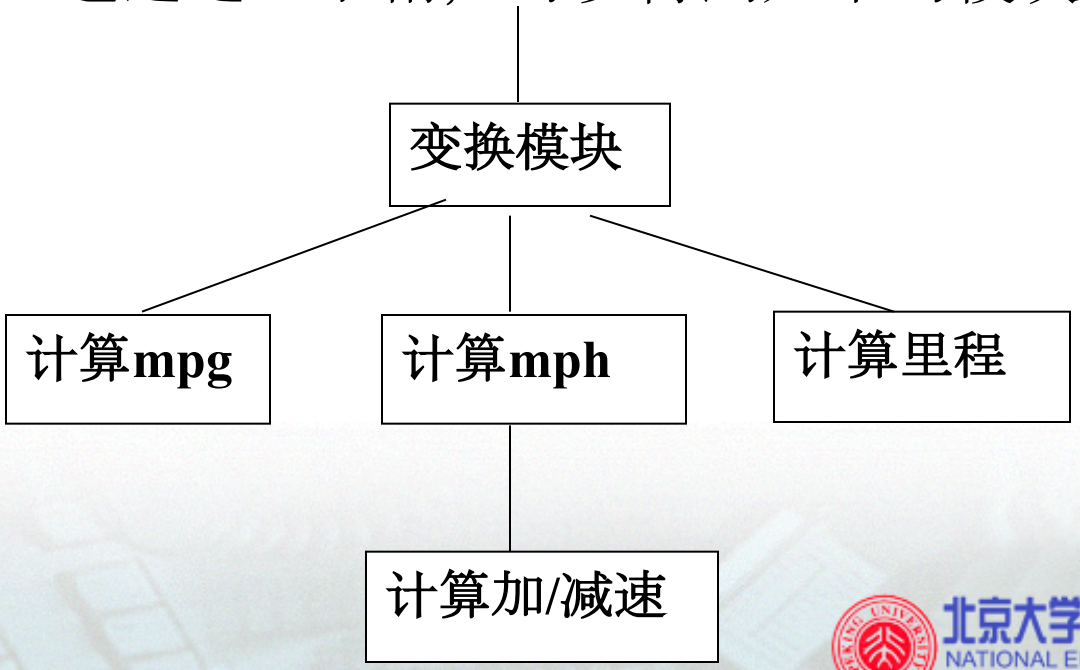


3、变换部分的精化

1）首先，应该了解：对于变换部分的求精，是一项具有挑战性的工作。其中主要是根据设计准则，并要通过实践，不断地总结经验，才能设计出合理的模块结构。

2）就给定的数字仪表板系统而言，如果把“确定加/减速”的模块放在“计算速度mph”模块下面，则可以减少模块之间的关联，提高模块的独立性。

通过这一求精，可以得到如下的模块结构图：



通过以上讨论，可以看出：在总体设计中

（1）将一个给定的DFD转换为初始的模块结构图基本上是一个“机械”的过程，一般体现不了设计人员的创造力；

（2）优化设计-将一个初始的模块结构图转换为最终的模块结构图，对设计人员将是一种挑战，其结果将直接影响软件系统开发的质量。

总体设计小结：

1、总体设计的目标和任务；

2、总体设计的表示：层次图，HIPO图，模块结构图；

3、基本概念：模块，以及由此产生的“鸿沟”；

4、总体设计的基本思想与步骤：

通过：变换设计和事务设计

DFD--→ 初始的MSD （几乎可“机械”地进行）

使用：启发式规则

初始的MSD-→ MSD （体现设计人员的创造）

2〕详细设计的任务：定义每一模块

- 结构化程序设计

三种控制结构:顺序

begin $s_1;s_2;\ldots s_n$ end;

选择

if 条件表达式 **then** s_1

else s_2;

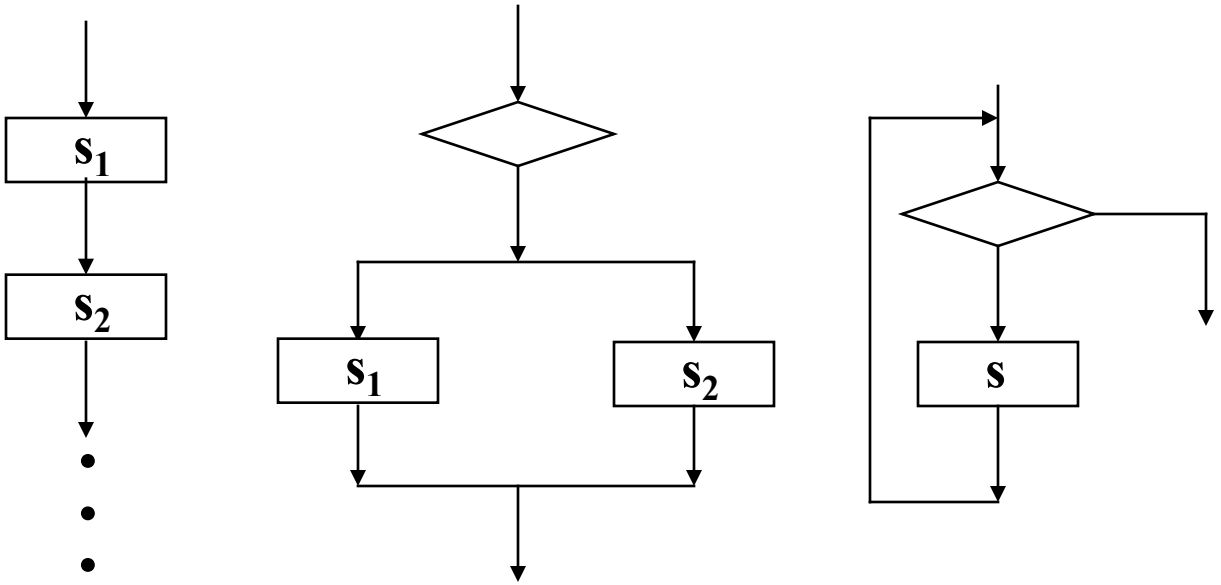
循环

while 条件表达式 **do** s ;

几种表示工具

流程图、PAD、N－S图、伪码等

1)框图



2)伪码

伪码是一种混合语言。外部采用形式语言的控制结构，内部使用自然语言。

Begin

输入一元二次方程的系数a,b,c;

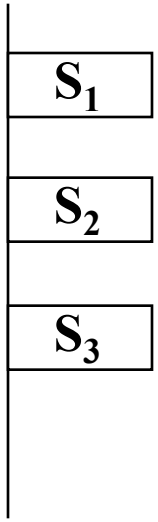
if b^2-4ac≥o then 计算两实根

else 输出无实根；

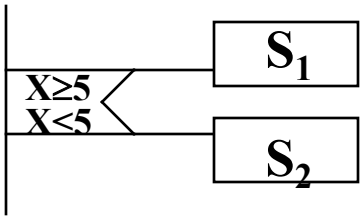
end.

3)PAD图

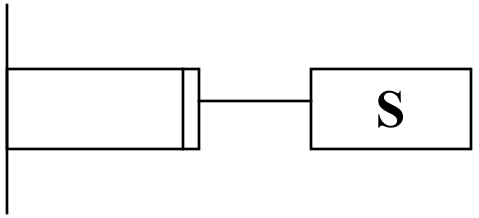
顺序:



选择：



循环:

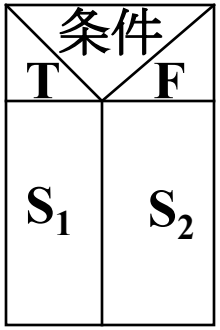


注:支持逐步求精设计

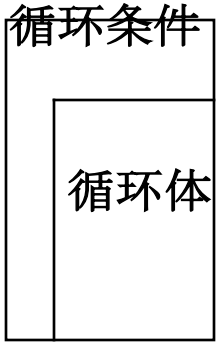
3)N-S图

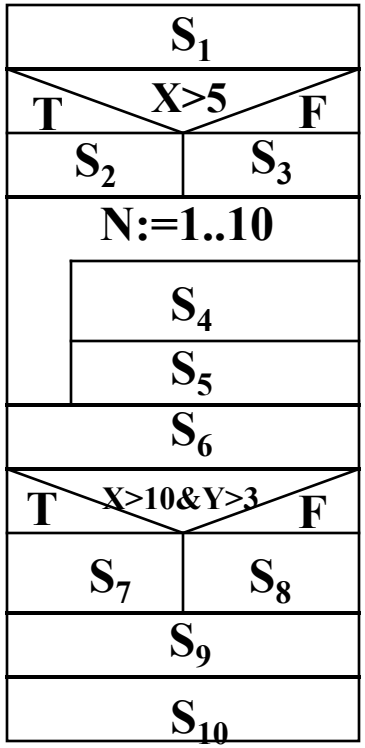
顺序：

选择：



循环：





支持逐步求精设计举例

结构化方法小结

1、结构化方法是一种比较系统的软件开发方法学。

包括：结构化分析和结构化设计

2、紧紧围绕“过程抽象”和“数据抽象”，

给出了　完备的符号体系

---概念与表示

可操作的过程

---步骤与准则

易理解的表示工具

提供了　控制信息组织复杂性的机制，例如

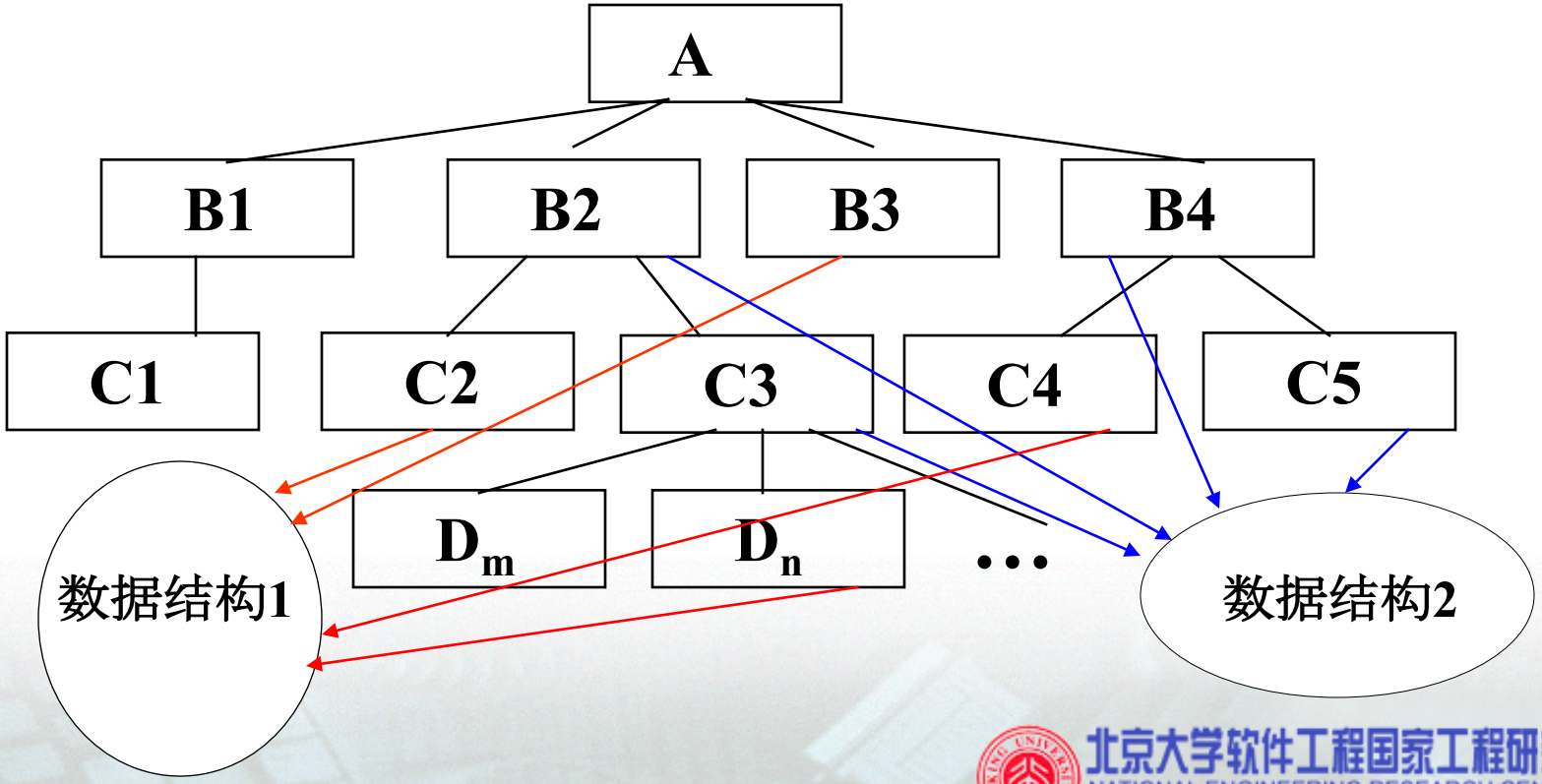
逐层分解，数据打包等

3、问题：捕获的“过程”和“数据”

恰恰是客观事物的易变性质，

解的结构也不保持原系统的结构，

从而：造成　维护，验证上的困难。



- 概念

软件方法学－以软件方法为研究对象的学科。主要涉及指导软件设计的原理和原则，以及基于这些原理、原则的方法和技术。狭义的也指某种特定的软件设计指导原则和方法体系。

- 从构造的角度，软件开发方法学主要由三部分组成

  - **NOTATION**
  - **PROCESS**
  - **TOOLS**

- 从能力的角度，软件开发方法学应能表达：
  - 系统的说明性信息
  - 系统的行为信息
  - 系统的功能信息，并要给出以下机制：
    - 控制信息组织复杂性
    - 控制文档组织复杂性

5、学习、掌握、运用系统建模技术的基本“技巧”：

1）知识

知识=概念+关系+条件/过程

2）建模

建模=（实际事物⇒概念）+表示(形式化或半形式化符号)

抽象：关注一个事物的重要的或主要方面，而忽略或去掉不重要的或没意义的细节。

其中，就软件开发而言，

- 根据当前情况和需要，应以细节的不同层次来观察问题；
- 控制复杂性，并考虑正确性、可维护性、可复用性和可理解性等。

模型：**any abstraction that includes all essential capabilities, properties,or aspects of what is being modeled without any extraneous details.[Firesmith,Henderson-Sellers]**

3） 实践

第一次作业

- 软件工程 第三版 P119-120,第5、6、7题