第5章 总体设计

总体设计的基本目的就是回答“概括地说，系统应该如何实现?”这个问题，因此，总体设计又称为概要设计或初步设计。通过这个阶段的工作将划分出组成系统的物理元素——程序、文件、数据库、人工过程和文档等等，但是每个物理元素仍然处于黑盒子级，这些黑盒子里的具体内容将在以后仔细设计。总体设计阶段的另一项重要任务是设计软件的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系。

总体设计设计软件结构。设计出初步的软件结构后还要多方改进，从而得到更合理的结构，进行必要的数据库设计，确定测试要求并且制定测试计划。

在详细设计之前先进行总体设计的必要性：可以站在全局高度上，花较少成本，从较抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低成本开发出较高质量的软件系统。

5.1 设计过程

总体设计过程通常由两个主要阶段组成：系统设计阶段，确定系统的具体实现方案；结构设计阶段，确定软件结构。典型的总体设计过程包括下述9个步骤：

1. 设想供选择的方案
2. 选取合理的方案
3. 推荐最佳方案

4. 功能分解

通常分为两个阶段完成：首先进行结构设计，然后进行过程设计。

结构设计确定程序由哪些模块组成，以及这些模块之间的关系；

过程设计确定每个模块的处理过程。

结构设计是总体设计阶段的任务，过程设计是详细设计阶段的任务。

为确定软件结构，首先需要从实现角度把复杂的功能进一步分解。分析员结合算法描述仔细分析数据流图中的每个处理，如果一个处理的功能过于复杂，必须把它的功能适当地分解成一系列比较简单的功能。一般说来，经过分解之后应该使每个功能对大多数程序员而言都是明显易懂的。功能分解导致数据流图的进一步细化，同时还应该用IPO图或其他适当的工具简要描述细化后每个处理的算法。

5. 设计软件结构

通常程序中的一个模块完成一个适当的子功能。应该把模块组织成良好的层次系统，顶层模块调用它的下层模块以实现程序的完整功能，每个下层模块再调用更下层的模块，从而完成程序的一个子功能，最下层的模块完成最具体的功能。软件的逻辑结构可以用层次图或结构图来描绘。

6. 设计数据库

对于需要使用数据库的那些应用系统，软件工程师应该在需求分析阶段所确定的系统数据需求的基础上，进一步设计数据库。

7. 制定测试计划

在软件开发的早期阶段考虑测试问题，能促使软件设计人员在设计时注意提高软件的可测试性。

8. 书写文档

应该用正式的文档记录总体设计的结果，在这个阶段应该完成的文档通常有下述几种：

(1) 系统说明：主要内容包括用系统流程图描绘的系统构成方案，组成系统的物理元素清单，成本/效益分析；对最佳方案的概括描述，精化的数据流图，用层次图或结构图描绘的软件结构，用IPO图或其他工具简要描述的各个模块的算法，模块间的接口关系，以及需求、功能和模块三者之间的交叉参照关系等等。

(2) 用户手册：根据总体设计阶段的结果，修改更正在需求分析阶段产生的初步的用户手册。

(3) 测试计划：包括测试策略，测试方案，预期的测试结果，测试进度计划等等。

(4) 详细的实现计划

(5) 数据库设计结果

9. 审查和复审

最后应该对总体设计的结果进行严格的技术审查，在技术审查通过之后再由使用部门的负责人从管理角度进行复审。

5.2 设计原理

5.2.1 模块化

模块是由边界元素限定的相邻程序元素（例如，数据说明，可执行的语句）的序列，而且有一个总体标识符代表它。

按照模块的定义，过程、函数、子程序和宏等，都可作为模块。面向对象方法学中的对象是模块，对象内的方法也是模块。模块是构成程序的基本构件。

模块化就是把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。

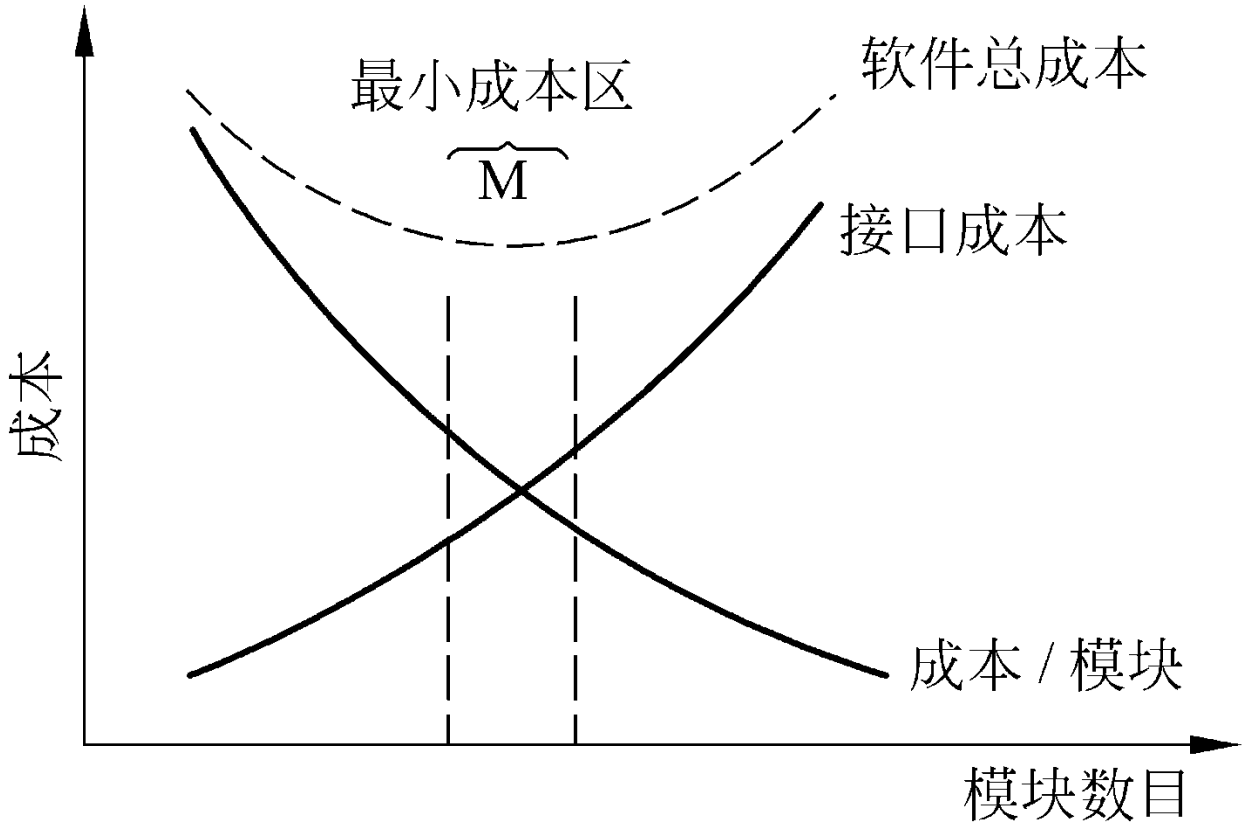


图5.1 模块化和软件成本

采用模块化原理可以使软件结构清晰，不仅容易设计也容易阅读和理解。因为程序错误通常局限在有关的模块及它们之间的接口中，所以模块化使软件容易测试和调试，因而有助于提高软件的可靠性。因为变动往往只涉及少数几个模块，所以模块化能够提高软件的可修改性。模块化也有助于软件开发工程的组织管理。

5.2.2 抽象

抽象是通过对现实世界中一类事物或概念经过高度概括总结抽取出其共性的过程。

人类在认识复杂现象的过程中使用的最强有力的思维工具是抽象。人们在实践中认识到，在现实世界中一定事物、状态或过程之间总存在着某些相似的方面(共性)。把这些相似的方面集中和概括起来，暂时忽略它们之间的差异，这就是抽象。或者说抽象就是抽出事物的本质特性而暂时不考虑它们的细节。

由于人类思维能力的限制，如果每次面临的因素太多，是不可能做出精确思维的。处理复杂系统的惟一有效的方法是用层次的方式构造和分析它。一个复杂的动态系统首先可以用一些高级的抽象概念构造和理解，这些高级概念又可以用一些较低级的概念构造和理解，如此进行下去，直至最低层次的具体元素。

这种层次的思维和解题方式必须反映在定义动态系统的程序结构之中，每级的一个概念将以某种方式对应于程序的一组成分。

考虑对任何问题的模块化解法时，可以提出许多抽象的层次。在抽象的最高层次使用问题环境的语言，以概括的方式叙述问题的解法；在较低抽象层次采用更过程化的方法，把面向问题的术语和面向实现的术语结合起来叙述问题的解法；最后在最低的抽象层次用可直接实现的方式叙述问题的解法。

软件工程过程的每一步都是对软件解法的抽象层次的一次精化。在可行性研究阶段，软件作为系统的一个完整部件；在需求分析期间，软件解法是使用在问题环境内熟悉的方式描述的；当由总体设计向详细设计过渡时，抽象的程度也就随之减少了；最后，当源程序写出来以后，也就达到了抽象的最低层。

5.2.3 逐步求精

逐步求精是人类解决复杂问题时采用的基本方法，也是许多软件工程技术的基础。可以把逐步求精定义为：“为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。”

求精方法的作用在于，它能帮助软件工程师把精力集中在与当前开发阶段最相关的那些方面上，而忽略那些对整体解决方案来说虽然是必要的，然而目前还不需要考虑的细节，这些细节将留到以后再考虑。

抽象与求精是一对互补的概念。抽象使得设计者能够说明过程和数据，同时却忽略低层细节。求精则帮助设计者在设计过程中逐步揭示出低层细节。这两个概念都有助于设计者在设计演化过程中创造出完整的设计模型。

5.2.4 信息隐藏和局部化

信息隐藏指出：应该这样设计和确定模块，使得一个模块内包含的信息对于不需要这些信息的模块来说，是不能访问的。

局部化是指把一些关系密切的软件元素物理地放得彼此靠近。局部化有助于实现信息隐藏。

局部化的概念和信息隐藏概念是密切相关的。

5.2.5 模块独立

模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。

开发具有独立功能而且和其他模块之间没有过多的相互作用的模块，就可以做到模块独立。

为什么模块的独立性很重要呢?主要有两条理由：

第一，有效的模块化软件比较容易开发出来。这是由于能够分割功能而且接口可以简化，当许多人分工合作开发同一个软件时，这个优点尤其重要。

第二，独立的模块比较容易测试和维护。这是因为相对说来，修改设计和程序需要的工作量比较小，错误传播范围小，需要扩充功能时能够“插入”代码。总之，模块独立是良好设计的关键，而设计又是决定软件质量的关键环节。

模块的独立程度可以由两个定性标准度量，这两个标准分别称为内聚和耦合。耦合衡量不同模块彼此间互相依赖(连接)的紧密程度；内聚衡量一个模块内部各个元素彼此结合的紧密程度。

1. 耦合

耦合是对一个软件结构内不同模块之间互连程度的度量。耦合强弱取决于模块间接口的复杂程度，进入或访问一个模块的点，以及通过接口的数据。

在软件设计中应该追求尽可能松散耦合的系统。易于开发、测试或维护这样的系统。此外，由于模块间联系简单，发生在一处的错误传播到整个系统的可能性就很小。因此，模块间的耦合程度强烈影响系统的可理解性、可测试性、可靠性和可维护性。

如果两个模块中的每一个都能独立地工作而不需要另一个模块的存在，那么它们彼此完全独立，这意味着模块间无任何连接，耦合程度最低。但是，在一个软件系统中不可能所有模块之间都没有任何连接。

根据模块间联系的紧密程度，把耦合分为：

数据耦合

控制耦合

特征耦合

公共环境耦合

内容耦合

如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，那么这种耦合称为数据耦合。如果传递的信息中有控制信息，则这种耦合称为控制耦合。

数据耦合是低耦合。系统中至少必须存在这种耦合，因为只有当某些模块的输出数据作为另一些模块的输入数据时，系统才能完成有价值的功能。一般说来，一个系统内可以只包含数据耦合。

控制耦合是中等程度的耦合，它增加了系统的复杂程度。控制耦合往往是多余的，在把模块适当分解之后通常可以用数据耦合代替它。

如果被调用的模块需要使用作为参数传递进来的数据结构中的所有元素，那么，把整个数据结构作为参数传递就是完全正确的。但是，当把整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素时，就出现了特征耦合。在这种情况下，被调用的模块可以使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制。

当两个或多个模块通过一个公共数据环境相互作用时，它们之间的耦合称为公共环境耦合。公共环境可以是全程变量、共享的通信区、内存的公共覆盖区、任何存储介质上的文件、物理设备等等。

公共环境耦合的复杂程度随耦合的模块个数而变化，当耦合的模块个数增加时复杂程度显著增加。如果只有两个模块有公共环境，那么这种耦合有下面两种可能：

(1) 一个模块往公共环境送数据，另一个模块从公共环境取数据。这是数据耦合的一种形式，是比较松散的耦合。

(2) 两个模块都既往公共环境送数据又从里面取数据，这种耦合比较紧密，介于数据耦合和控制耦合之间。

如果两个模块共享的数据很多，都通过参数传递可能很不方便，这时可以利用公共环境耦合。

最高程度的耦合是内容耦合。如果出现下列情况之一，两个模块间就发生了内容耦合：

一个模块访问另一个模块的内部数据；

一个模块不通过正常入口而转到另一个模块的内部；

两个模块有一部分程序代码重叠；

一个模块有多个入口(这意味着一个模块有几种功能)。

应该坚决避免使用内容耦合。事实上许多高级程序设计语言已经设计成不允许在程序中出现任何形式的内容耦合。

总之，耦合是影响软件复杂程度的一个重要因素。应该采取下述设计原则：

尽量使用数据耦合，少用控制耦合和特征耦合，限制公共环境耦合的范围，完全不用内容耦合。

2. 内聚

内聚标志一个模块内各个元素彼此结合的紧密程度，它是信息隐藏和局部化概念的自然扩展。简单地说，理想内聚的模块只做一件事情。

设计时应该力求做到高内聚，通常中等程度的内聚也是可以采用的，而且效果和高内聚相差不多；但是，低内聚不要使用。

根据模块内联系的紧密程度，把内聚分为：

偶然内聚

逻辑内聚

时间内聚

过程内聚

通信内聚

顺序内聚

功能内聚

内聚和耦合是密切相关的，模块内的高内聚往往意味着模块间的松耦合。内聚和耦合都是进行模块化设计的有力工具，但是实践表明内聚更重要，应该把更多注意力集中到提高模块的内聚程度上。

低内聚有如下几类：

如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。有时在写完一个程序之后，发现一组语句在两处或多处出现，于是把这些语句作为一个模块以节省内存，这样就出现了偶然内聚的模块。

如果一个模块完成的任务在逻辑上属于相同或相似的一类，则称为逻辑内聚。

如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。

在偶然内聚的模块中，各种元素之间没有实质性联系，很可能在一种应用场合需要修改这个模块，在另一种应用场合又不允许这种修改，从而陷入困境。

在逻辑内聚的模块中，不同功能混在一起，合用部分程序代码，即使局部功能的修改有时也会影响全局。因此，这类模块的修改也比较困难。

时间关系在一定程度上反映了程序的某些实质，所以时间内聚比逻辑内聚好一些。

中内聚主要有两类：

如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。使用程序流程图作为工具设计软件时，常常通过研究流程图确定模块的划分，这样得到的往往是过程内聚的模块。

如果模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚。

高内聚也有两类：

如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行(通常一个处理元素的输出数据作为下一个处理元素的输入数据)，则称为顺序内聚。根据数据流图划分模块时，通常得到顺序内聚的模块，这种模块彼此间的连接往往比较简单。

如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

耦合和内聚的概念是Constantine等人提出来的。按照他们的观点，如果给上述七种内聚的优劣评分，将得到如下结果：

功能内聚 10分	时间内聚 3分
顺序内聚 9分	逻辑内聚 1分
通信内聚 7分	偶然内聚 0分
过程内聚 5分	

设计时力争做到高内聚，并且能够辨认出低内聚的模块，有能力通过修改设计提高模块的内聚程度降低模块间的耦合程度，从而获得较高的模块独立性。

5.3 启发规则

人们在开发计算机软件的长期实践中积累了丰富的经验，总结这些经验得出了一些启发式规则。

1. 改进软件结构提高模块独立性

设计出软件的初步结构以后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。例如，多个模块公有的一个子功能可以独立成一个模块，由这些模块调用；有时可以通过分解或合并模块以减少控制信息的传递及对全程数据的引用，并且降低接口的复杂程度。

2. 模块规模应该适中

经验表明，一个模块的规模不应过大，通常不超过60行语句。有人从心理学角度研究得知，当一个模块包含的语句数超过30以后，模块的可理解程度迅速下降。

过大的模块往往是由于分解不充分，但是进一步分解必须符合问题结构，一般说来，分解后不应该降低模块独立性。

过小的模块开销大于有效操作，而且模块数目过多将使系统接口复杂。因此过小的模块有时不值得单独存在，特别是只有一个模块调用它时，通常可以把它合并到上级模块中去而不必单独存在。

3. 深度、宽度、扇出和扇入都应适当

深度表示软件结构中控制的层数，它往往能粗略地标志一个系统的大小和复杂程度。深度和程序长度之间应该有粗略的对应关系。如果层数过多则应该考虑是否有许多管理模块过分简单了，能否适当合并。

宽度是软件结构内同一个层次上的模块总数的最大值。一般说来，宽度越大系统越复杂。对宽度影响最大的因素是模块的扇出。

扇出是一个模块直接调用的模块数目，扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块；扇出过小也不好。经验表明，一个设计得好的典型系统的平均扇出通常是3或4(扇出的上限通常是5～9)。

扇出太大一般是因为缺乏中间层次，应该适当增加中间层次的控制模块。扇出太小时可以把下级模块进一步分解成若干个子功能模块，或者合并到它的上级模块中去。当然分解模块或合并模块必须符合问题结构，不能违背模块独立原理。

一个模块的扇入表明有多少个上级模块直接调用它，扇入越大则共享该模块的上级模块数目越多，这是有好处的，但是，不能违背模块独立原理单纯追求高扇入。

设计得很好的软件结构通常顶层扇出比较高，中层扇出较少，底层扇入到公共的实用模块中去(底层模块有高扇入)。

4. 模块的作用域应该在控制域之内

模块的作用域定义为受该模块内一个判定影响的所有模块的集合。模块的控制域是这个模块本身以及所有直接或间接从属于它的模块的集合。例如，在图5.2中模块A的控制域是A、B、C、D、E、F等模块的集合。

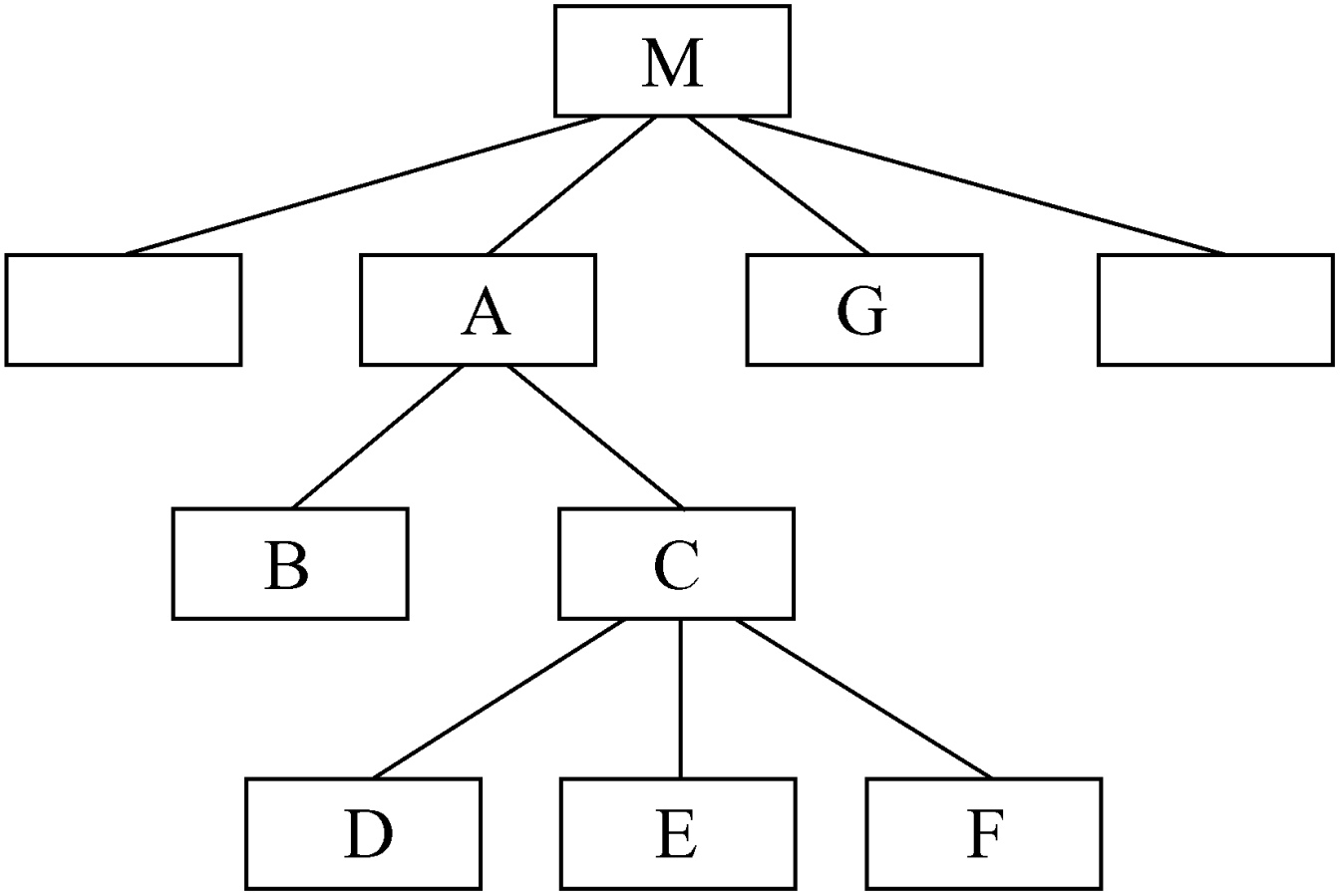


图5.2 模块的作用域和控制域

5. 力争降低模块接口的复杂程度

模块接口复杂是软件发生错误的一个主要原因。应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。

接口复杂或不一致，是紧耦合或低内聚的征兆，应该重新分析这个模块的独立性。

6. 设计单入口单出口的模块

这条启发式规则警告软件工程师不要使模块间出现内容耦合。当从顶部进入模块并且从底部退出来时，软件是比较容易理解的，也是比较容易维护的。

7. 模块功能应该可以预测

模块的功能应该能够预测，但也要防止模块功能过分局限。

如果一个模块可以当做一个黑盒子，也就是说，只要输入的数据相同就产生同样的输出，这个模块的功能就是可以预测的。

以上列出的启发式规则多数是经验规律，对改进设计，提高软件质量，往往有重要的参考价值；但是，它们既不是设计的目标也不是设计时应该普遍遵循的原理。

5.4 描绘软件结构的图形工具

5.4.1 层次图和HIPO图

层次图用来描绘软件的层次结构。层次图中的一个矩形框代表一个模块，方框间的连线表示调用关系。图5.3是层次图的一个例子。

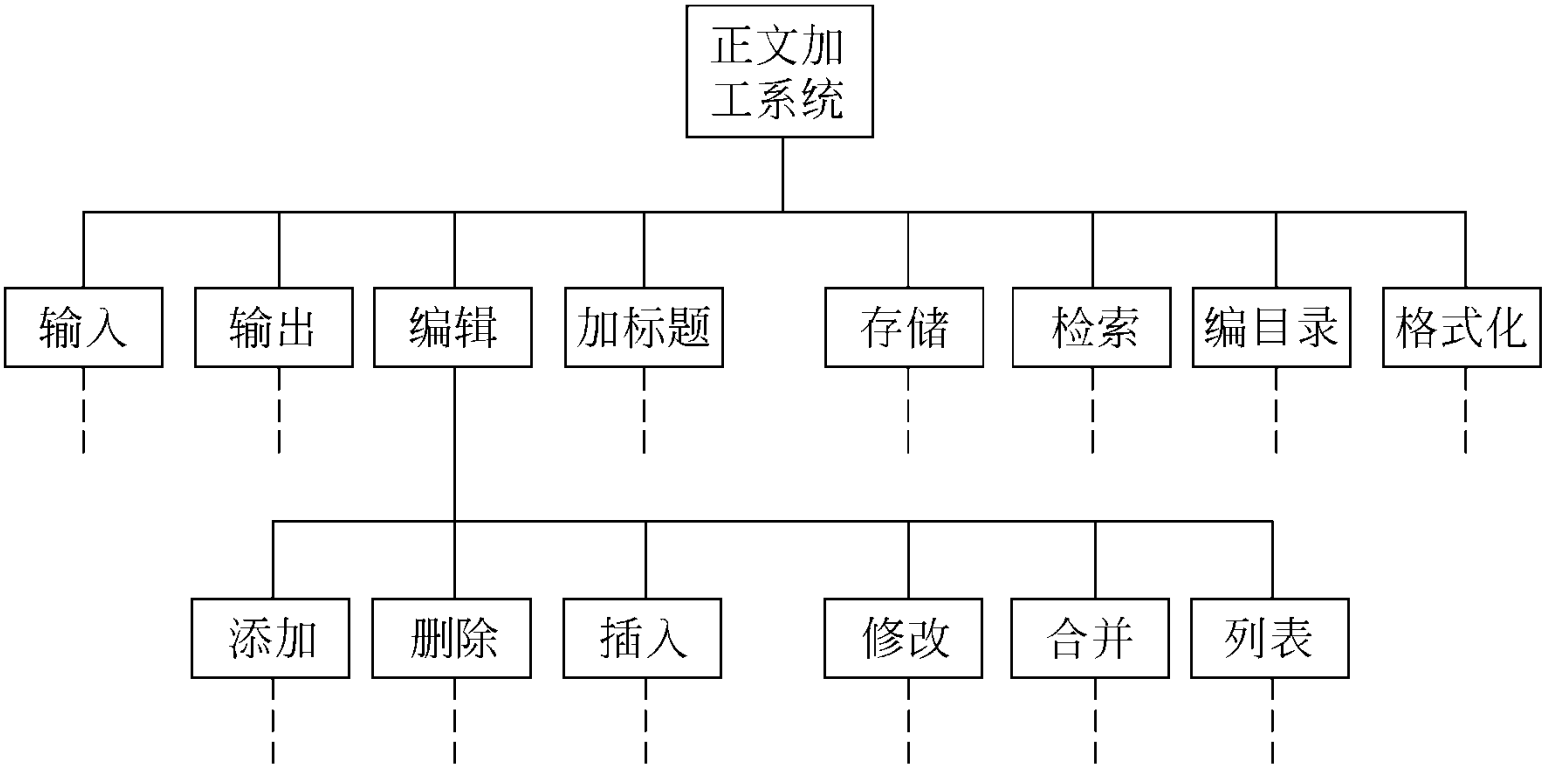


图5.3 正文加工系统的层次图

层次图很适于在自顶向下设计软件的过程中使用。

HIPO图是IBM公司发明的“层次图加输入/处理/输出图”的缩写。为了能使HIPO图具有可追踪性，在层次图里除了最顶层的方框之外，每个方框都加了编号。

图5.3加了编号后得到图5.4。

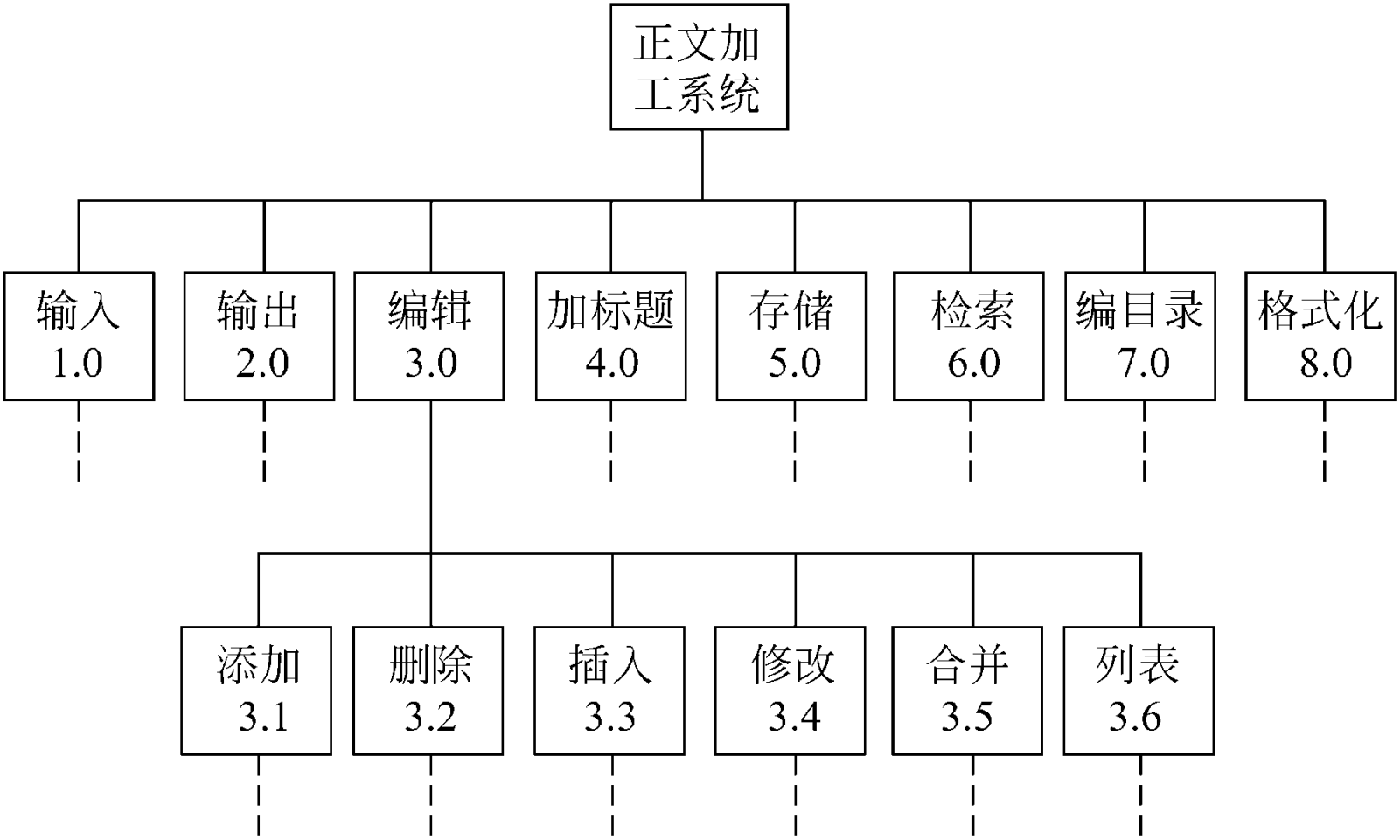


图5.4 带编号的层次图(H图)

5.4.2 结构图

Yourdon提出的结构图是进行软件结构设计的另一个工具。结构图和层次图类似，也是描绘软件结构的图形工具，图中一个方框代表一个模块，框内注明模块的名字或主要功能；方框之间的箭头(或直线)表示模块的调用关系。因为按照惯例总是图中位于上方的方框代表的模块调用下方的模块，即使不用箭头也不会产生二义性，为了简单起见，可以只用直线而不用箭头表示模块间的调用关系。

在结构图中通常还用带注释的箭头表示模块调用过程中来回传递的信息。如果希望进一步标明传递的信息是数据还是控制信息，则可以利用注释箭头尾部的形状来区分：尾部是空心圆表示传递的是数据，实心圆表示传递的是控制信息。图5.5是结构图的一个例子。

以上介绍的是结构图的基本符号，也就是最经常使用的符号。此外还有一些附加的符号，可以表示模块的选择调用或循环调用。图5.6表示当模块M中某个判定为真时调用模块A，为假时调用模块B。图5.7表示模块M循环调用模块A、B和C。

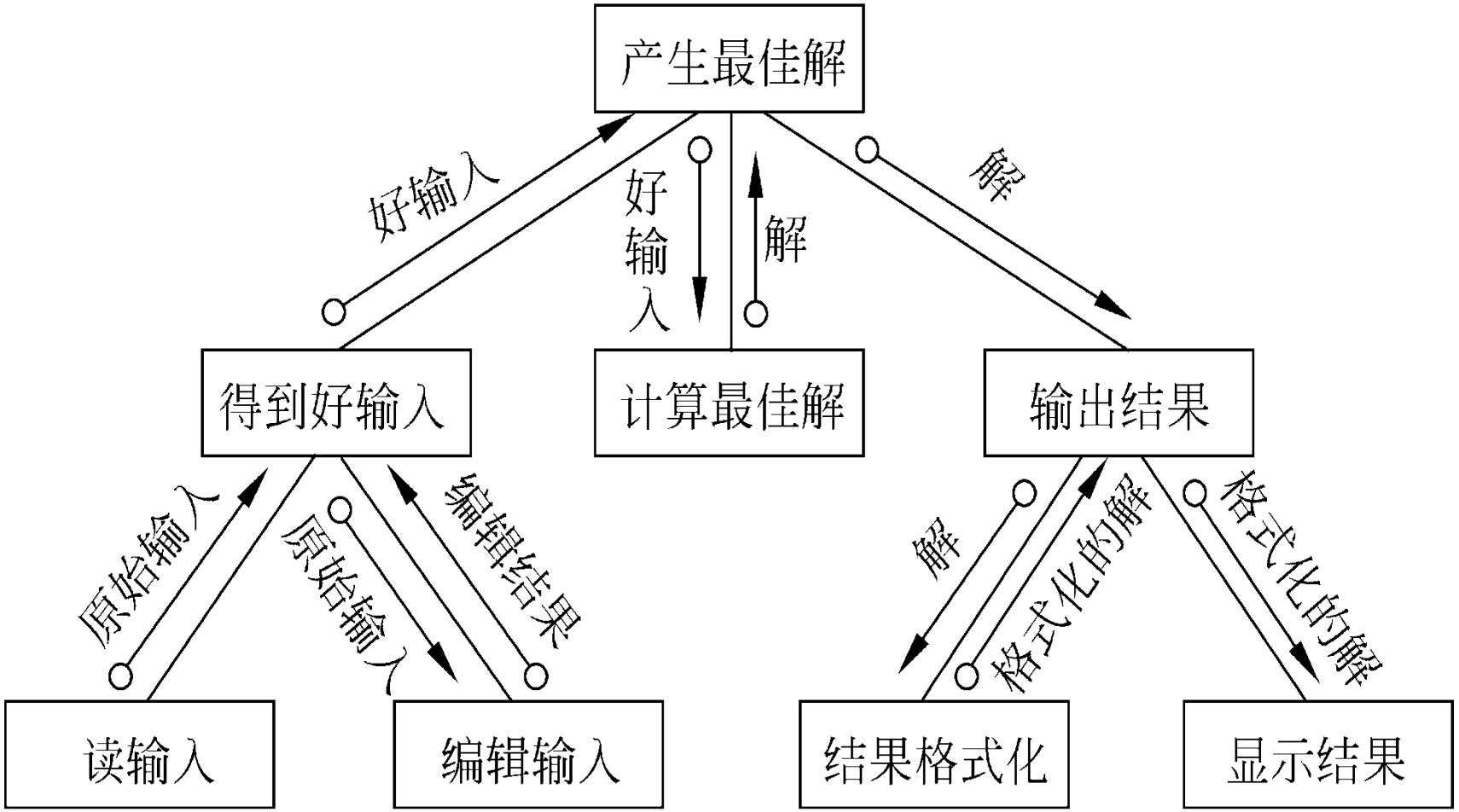


图5.5 结构图的例子——产生最佳解的一般结构

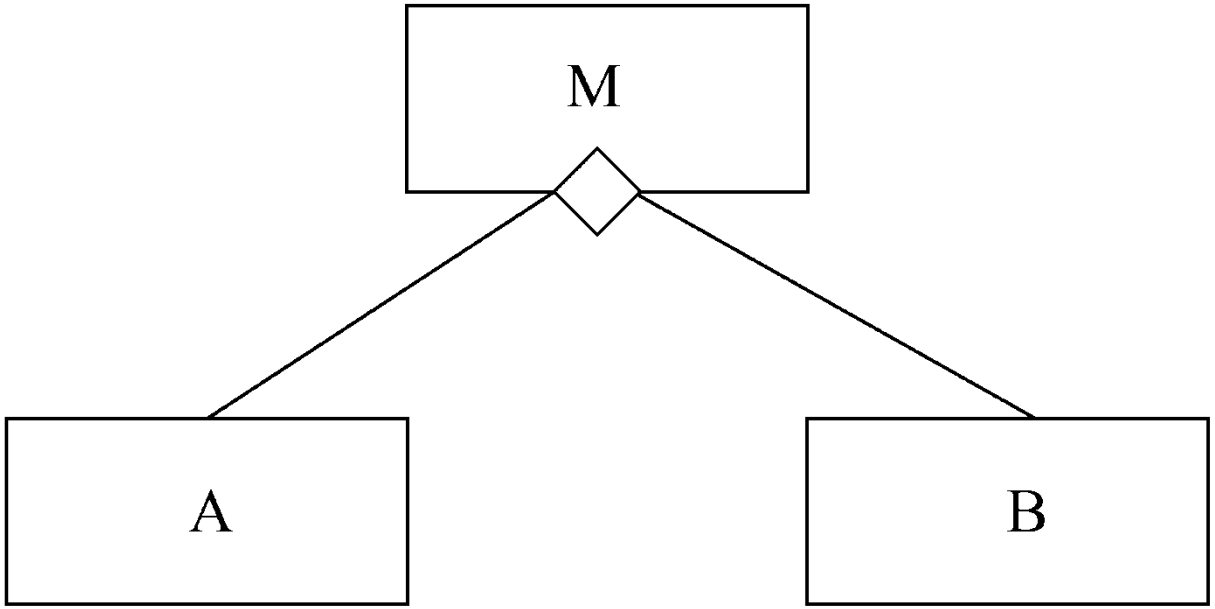


图5.6 判定为真时调用A，为假时调用B

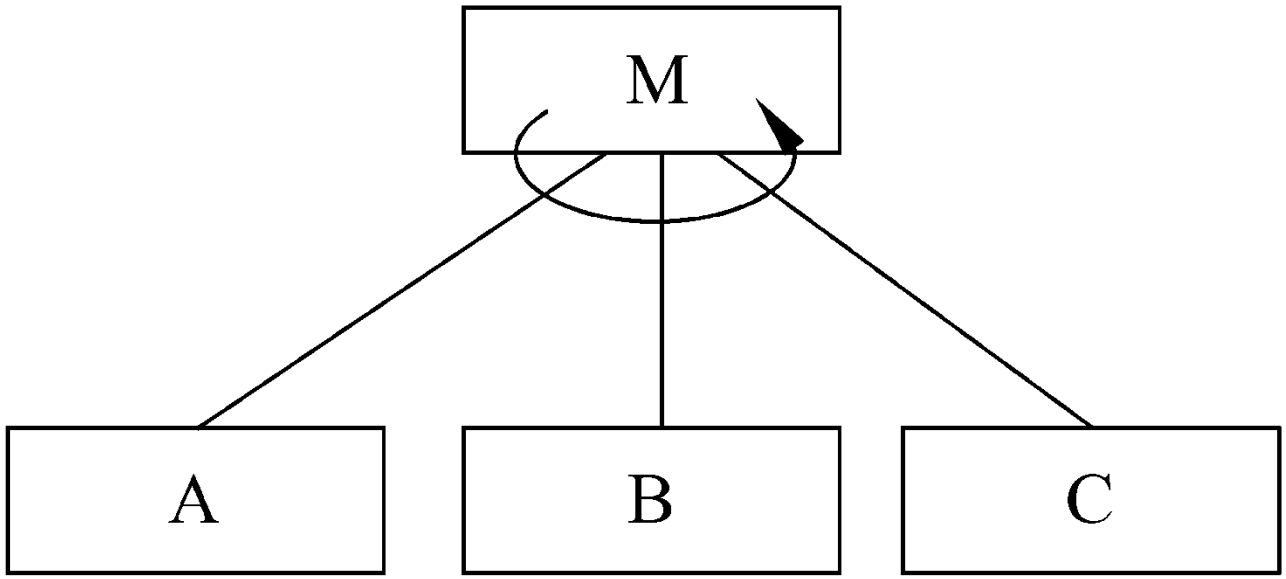


图5.7 模块M循环调用模块A、B、C

5.5 面向数据流的设计方法

面向数据流的设计方法的目标是给出设计软件结构的一个系统化的途径。

在软件工程的需求分析阶段，信息流是一个关键考虑，通常用数据流图描绘信息在系统中加工和流动的情况。面向数据流的设计方法定义了一些不同的“映射”，利用这些映射可以把数据流图变换成软件结构。因为任何软件系统都可以用数据流图表示，所以面向数据流的设计方法理论上可以设计任何软件的结构。通常所说的结构化设计方法(简称SD方法)，也就是基于数据流的设计方法。

5.5.1 概念

面向数据流的设计方法把信息流映射成软件结构，信息流的类型决定了映射的方法。信息流有下述两种类型。

1. 变换流

进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式离开软件系统。当数据流图具有这些特征时，这种信息流就叫作变换流。

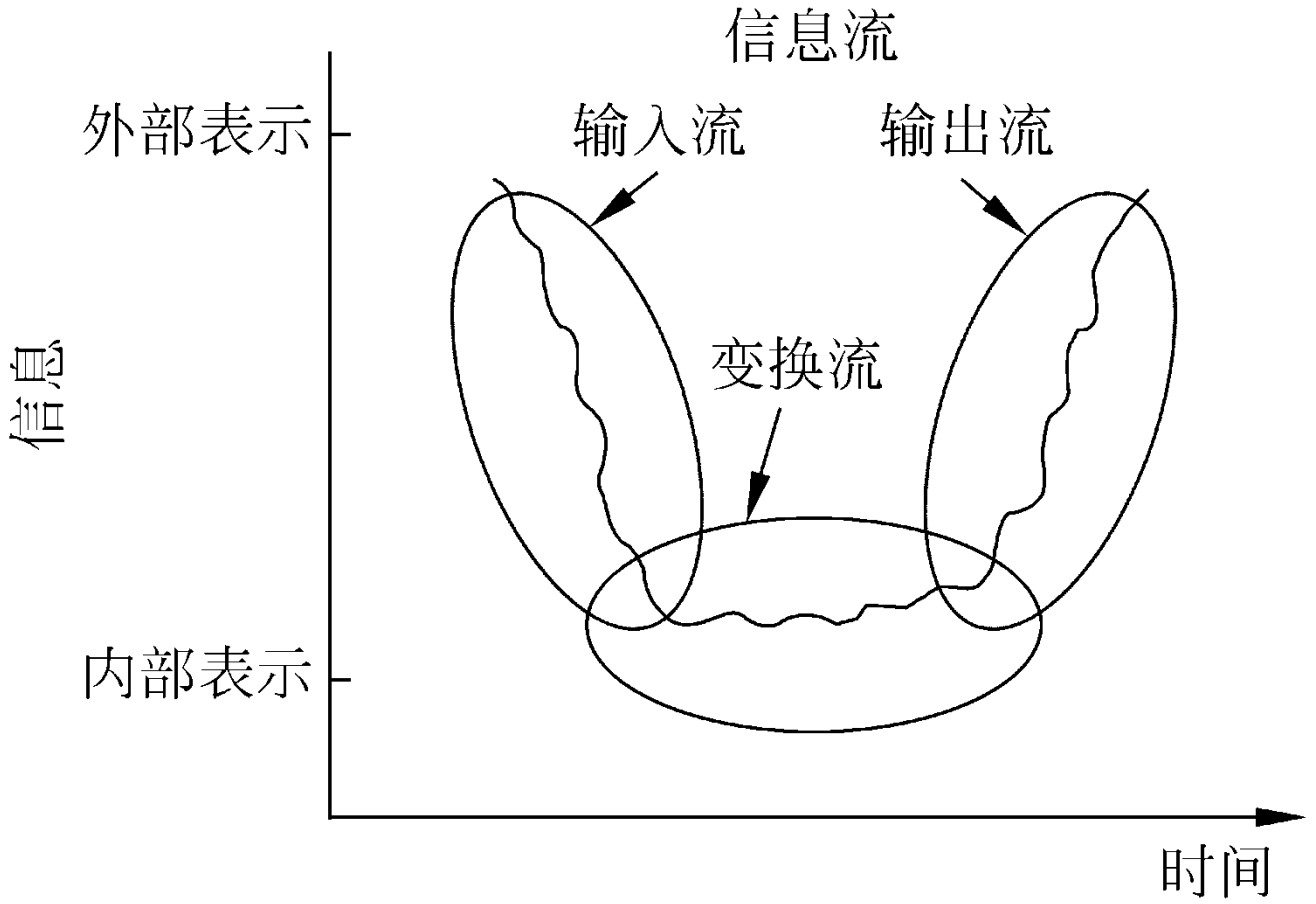


图5.8 变换流

2. 事务流

数据沿输入通路到达一个处理T，这个处理根据输入数据的类型在若干个动作序列中选出一个来执行。当数据流图具有类似的形状时，这种数据流是“以事务为中心的”，称为事务流。图5.9中的处理T称为事务中心，它完成下述任务：

(1) 接收输入数据(输入数据又称为事务)；

(2) 分析每个事务以确定它的类型；

(3) 根据事务类型选取一条活动通路。

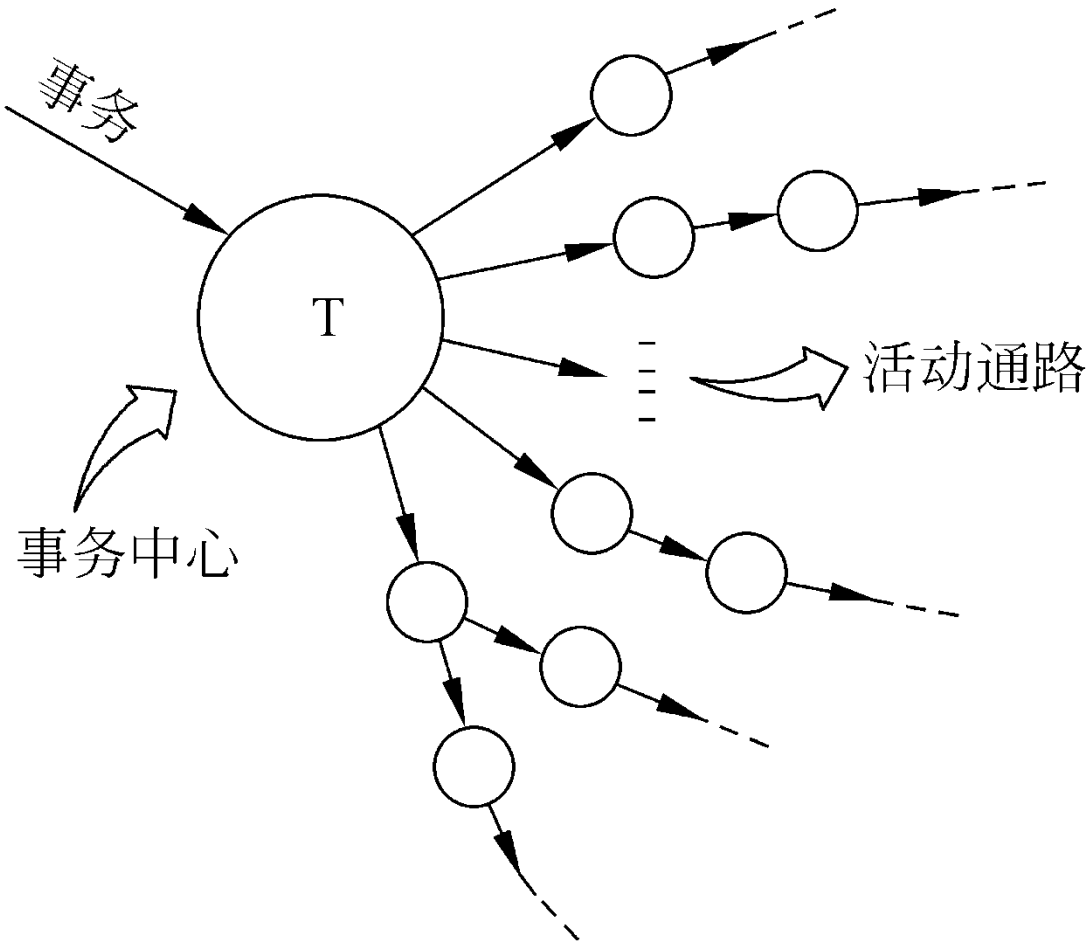
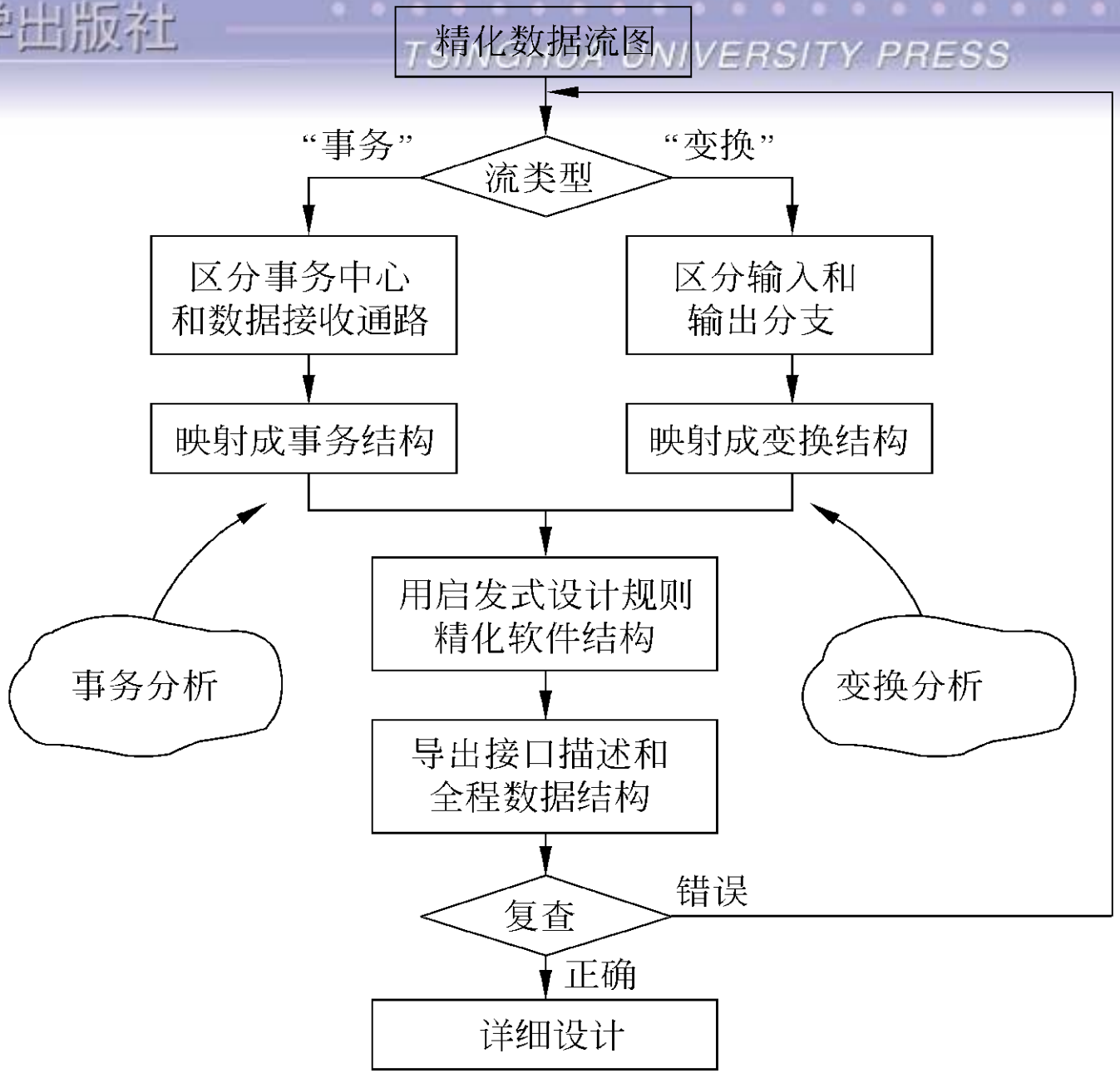


图5.9 事务流

3. 设计过程

图5.10说明了使用面向数据流方法逐步设计的过程。

注意，任何设计过程都不是机械地一成不变的。



5.5.2 变换分析

变换分析是一系列设计步骤的总称，经过这些步骤把具有变换流特点的数据流图按预先确定的模式映射成软件结构。

1. 例子

考虑汽车数字仪表板的设计。

假设的仪表板将完成下述功能：

(1) 通过模数转换实现传感器和微处理机接口；

(2) 在发光二极管面板上显示数据；

(3) 指示每小时英里数(mph)，行驶的里程，每加仑油行驶的英里数(mpg)等等；

(4) 指示加速或减速；

(5) 超速警告：如果车速超过55英里/小时，则发出超速警告铃声。

在软件需求分析阶段应该对上述每条要求以及系统的其他特点进行全面的分析评价，建立起必要的文档资料，特别是数据流图。

2. 设计步骤

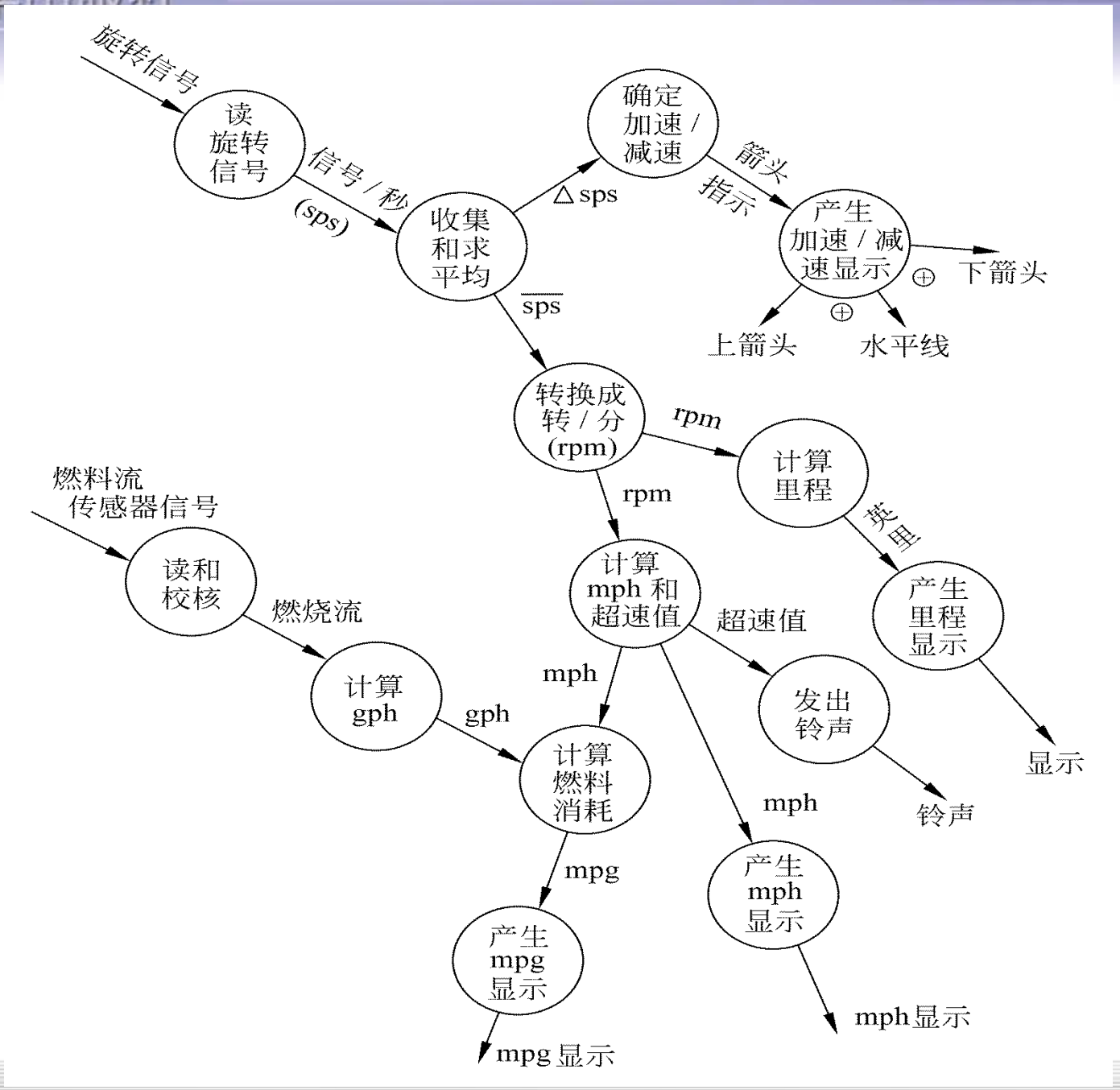
第1步 复查基本系统模型。

复查的目的是确保系统的输入数据和输出数据符合实际。

第2步 复查并精化数据流图。

应该对需求分析阶段得出的数据流图认真复查，并且在必要时进行精化。不仅要确保数据流图给出了目标系统的正确的逻辑模型，而且应该使数据流图中每个处理都代表一个规模适中相对独立的子功能。

假设在需求分析阶段产生的数字仪表板系统的数据流图如图5.11所示。


旋转信号
读旋转信号
信号/秒
(sps)
收集和求平均
△sps
确定加速/减速
箭头指示
产生加速/减速显示
⊕ 下箭头
⊕
上箭头
水平线
sps
转换成转/分(rpm)
rpm
计算里程
英里
产生里程显示
显示
rpm
计算mph和超速值
超速值
发出铃声
铃声
燃料流传感器信号
读和校核
燃烧流
计算gph
gph
mph
计算燃料消耗
mpg
产生mpg显示
mpg显示
mph
产生mph显示
mph显示

这个数据流图对于软件结构设计的“第一次分割”而言已经足够详细了，因此不需要精化就可以进行下一个设计步骤。

第3步 确定数据流图具有变换特性还是事务特性。

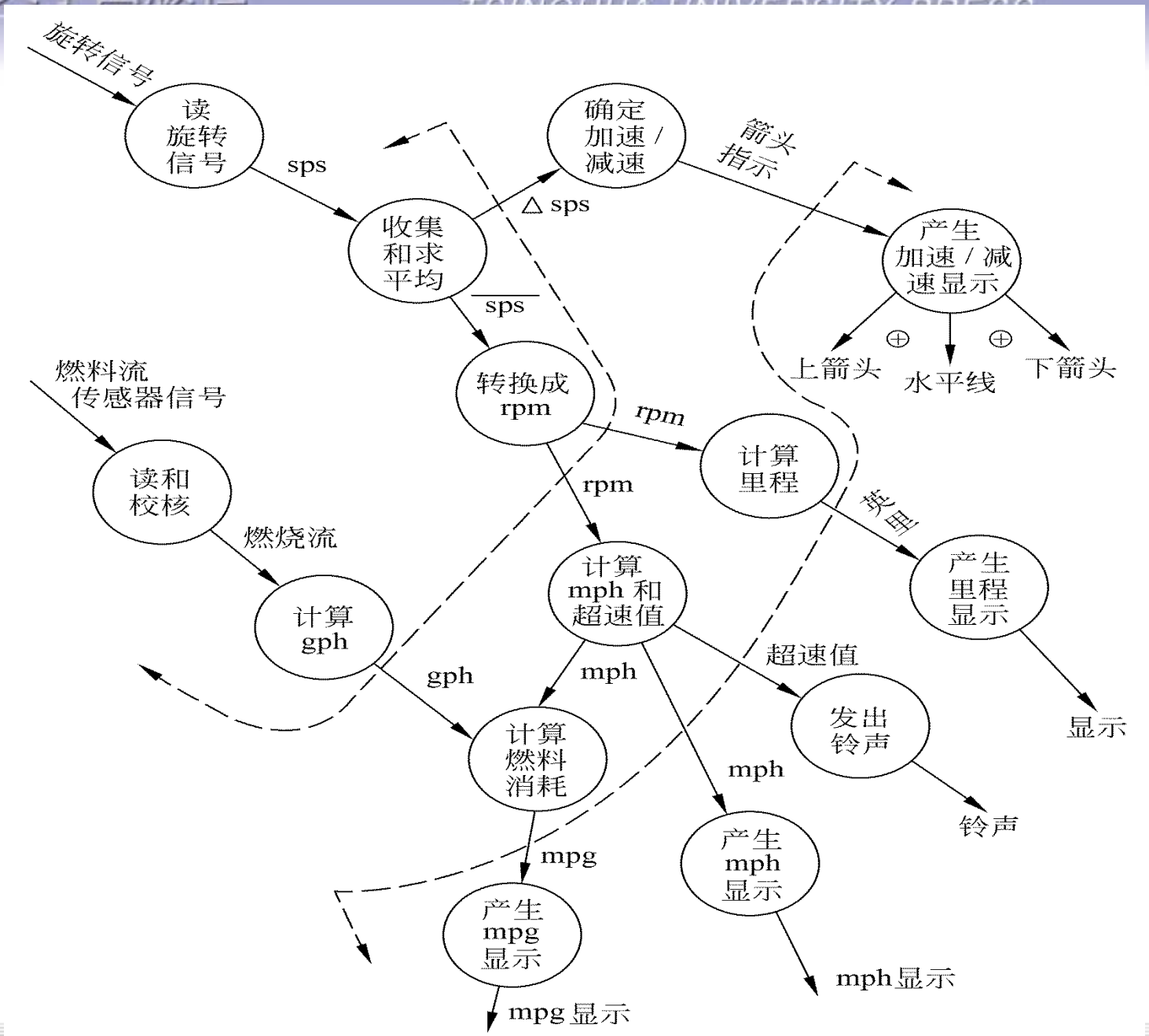
一般地说，一个系统中的所有信息流都可以认为是变换流，但是，当遇到有明显事务特性的信息流时，建议采用事务分析方法进行设计。在这一步，设计人员应该根据数据流图中占优势的属性，确定数据流的全局特性。此外还应该把具有和全局特性不同的特点的局部区域孤立出来，以后可以按照这些子数据流的特点精化根据全局特性得出的软件结构。

从图5.11看出，数据沿着两条输入通路进入系统，然后沿着5条通路离开，没有明显的事务中心。因此可以认为这个信息流具有变换流的总特征。

第4步 确定输入流和输出流的边界，从而孤立出变换中心。

输入流和输出流的边界和对它们的解释有关，也就是说，不同设计人员可能会在流内选取稍微不同的点作为边界的位置。当然在确定边界时应该仔细认真，但是把边界沿着数据流通路移动一个处理框的距离，通常对最后的软件结构只有很小的影响。

对于汽车数字仪表板的例子，设计人员确定的流的边界如图5.12所示。


旋转信号
读旋转信号
sps
收集和求平均
△ sps
sps
确定加速/减速
箭头指示
产生加速/减速显示
上箭头
水平线
下箭头
燃料流传感器信号
读和校核
燃烧流
计算gph
转换成rpm
rpm
rpm
计算里程
英里
产生里程显示
显示
计算mph和超速值
超速值
mph
gph
计算燃料消耗
发出铃声
铃声
mph
mpg
产生mpg显示
mpg显示
产生mph显示
mph显示

第5步 完成“第一级分解”。

软件结构代表对控制的自顶向下的分配，所谓分解就是分配控制的过程。

对于变换流的情况，数据流图被映射成一个特殊的软件结构，这个结构控制输入、变换和输出等信息处理过程。图5.13说明了第一级分解的方法。位于软件结构最顶层的控制模块Cm协调下述从属的控制功能：

输入信息处理控制模块Ca,协调对所有输入数据的接收；

变换中心控制模块Ct,管理对内部形式的数据的所有操作；

输出信息处理控制模块Ce，协调输出信息的产生过程。

虽然图5.13意味着一个三叉的控制结构，但是，对一个大型系统中的复杂数据流可以用两个或多个模块完成上述一个模块的控制功能。应该在能够完成控制功能并且保持好的耦合和内聚特性的前提下，尽量使第一级控制中的模块数目取最小值。

对于数字仪表板的例子，第一级分解得出的结构如图5.14所示。每个控制模块的名字表明了为它所控制的那些模块的功能。

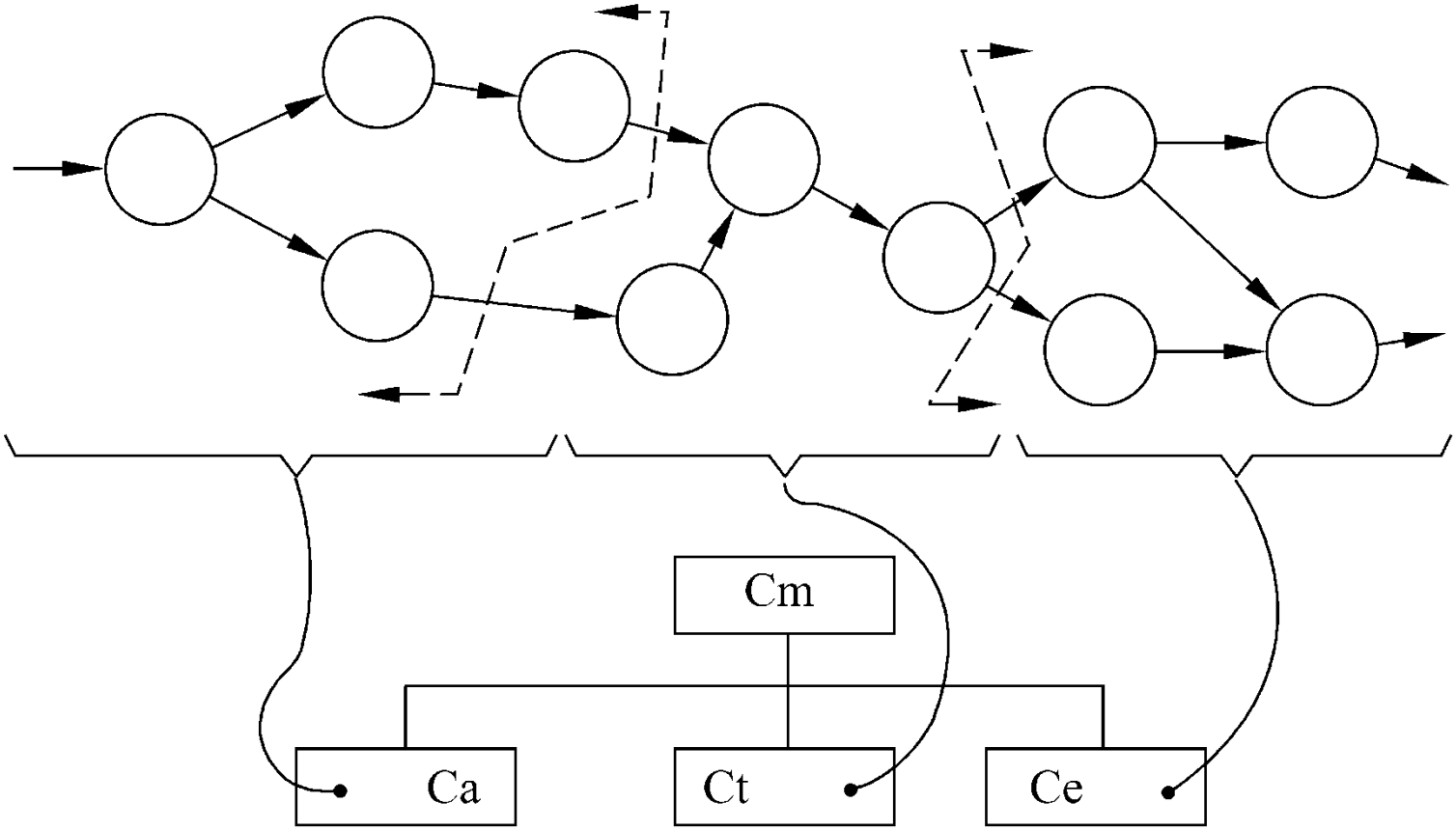


图5.13 第一级分解的方法

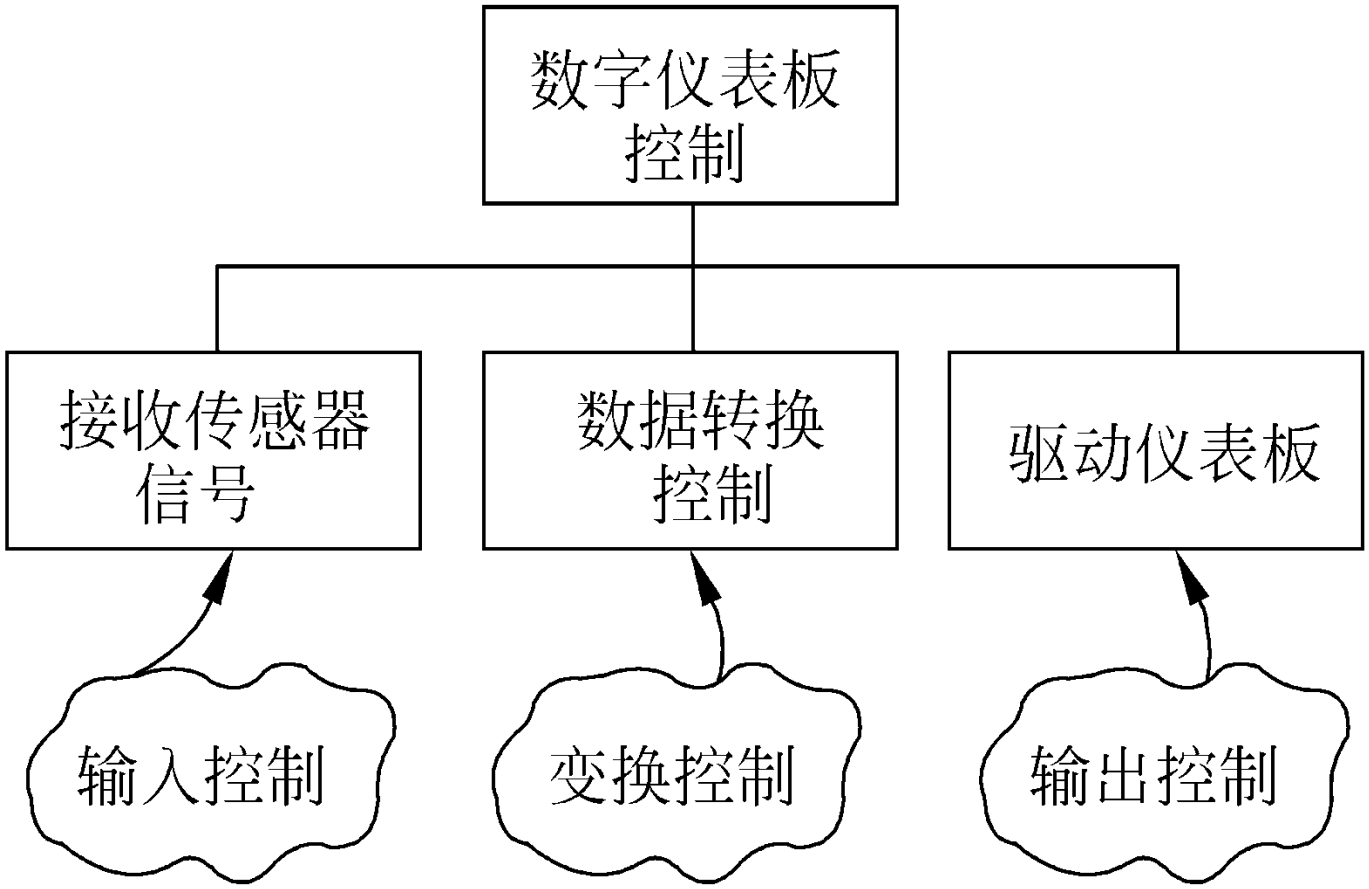


图5.14 数字仪表板系统的第一级分解

第6步 完成“第二级分解”。

所谓第二级分解就是把数据流图中的每个处理映射成软件结构中一个适当的模块。完成第二级分解的方法是，从变换中心的边界开始沿着输入通路向外移动，把输入通路中每个处理映射成软件结构中Ca控制下的一个低层模块；然后沿输出通路向外移动，把输出通路中每个处理映射成直接或间接受模块Ce控制的一个低层模块；最后把变换中心内的每个处理映射成受Ct控制的一个模块。图5.15表示进行第二级分解的普遍途径。

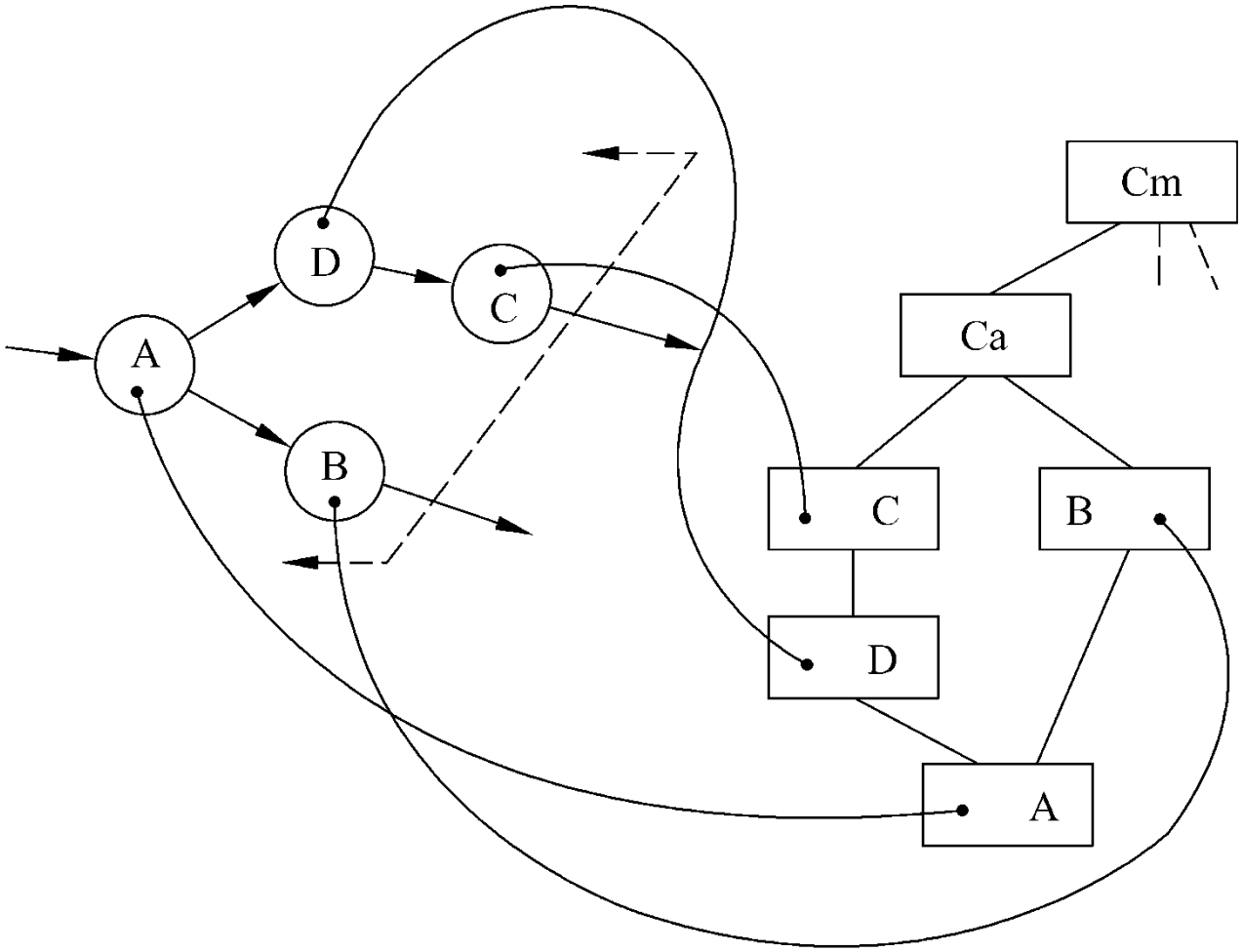


图5.15 第二级分解的方法

虽然图5.15描绘了在数据流图中的处理和软件结构中的模块之间的一对一的映射关系，但是，不同的映射经常出现。应该根据实际情况以及“好”设计的标准，进行实际的第二级分解。

对于数字仪表板系统的例子，第二级分解的结果分别用图5.16，5.17和5.18描绘。这3张图表示对软件结构的初步设计结果。虽然图中每个模块的名字表明了它的基本功能，但是仍然应该为每个模块写一个简要说明，描述：

进出该模块的信息(接口描述)；

模块内部的信息；

过程陈述，包括主要判定点及任务等；

对约束和特殊特点的简短讨论。

这些描述是第一代的设计规格说明，在这个设计时期进一步的精化和补充是经常发生的。

第7步 使用设计度量和启发式规则对第一次分割得到的软件结构进一步精化。

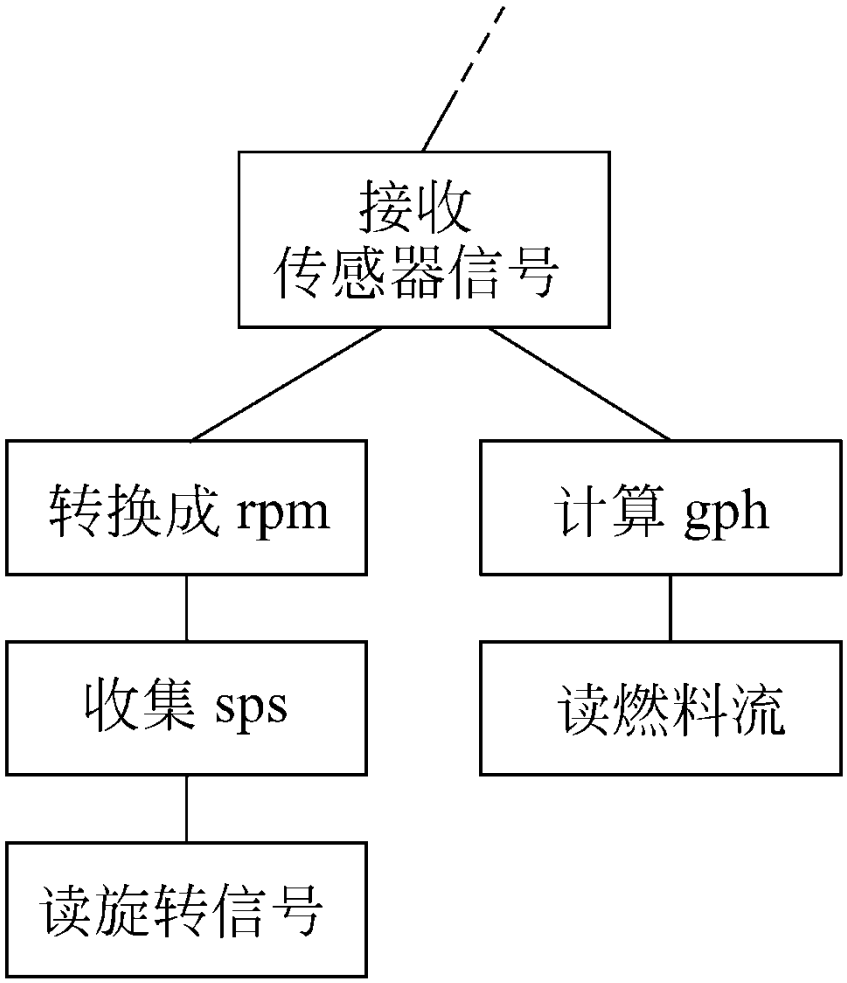


图5.16 未经精化的输入结构

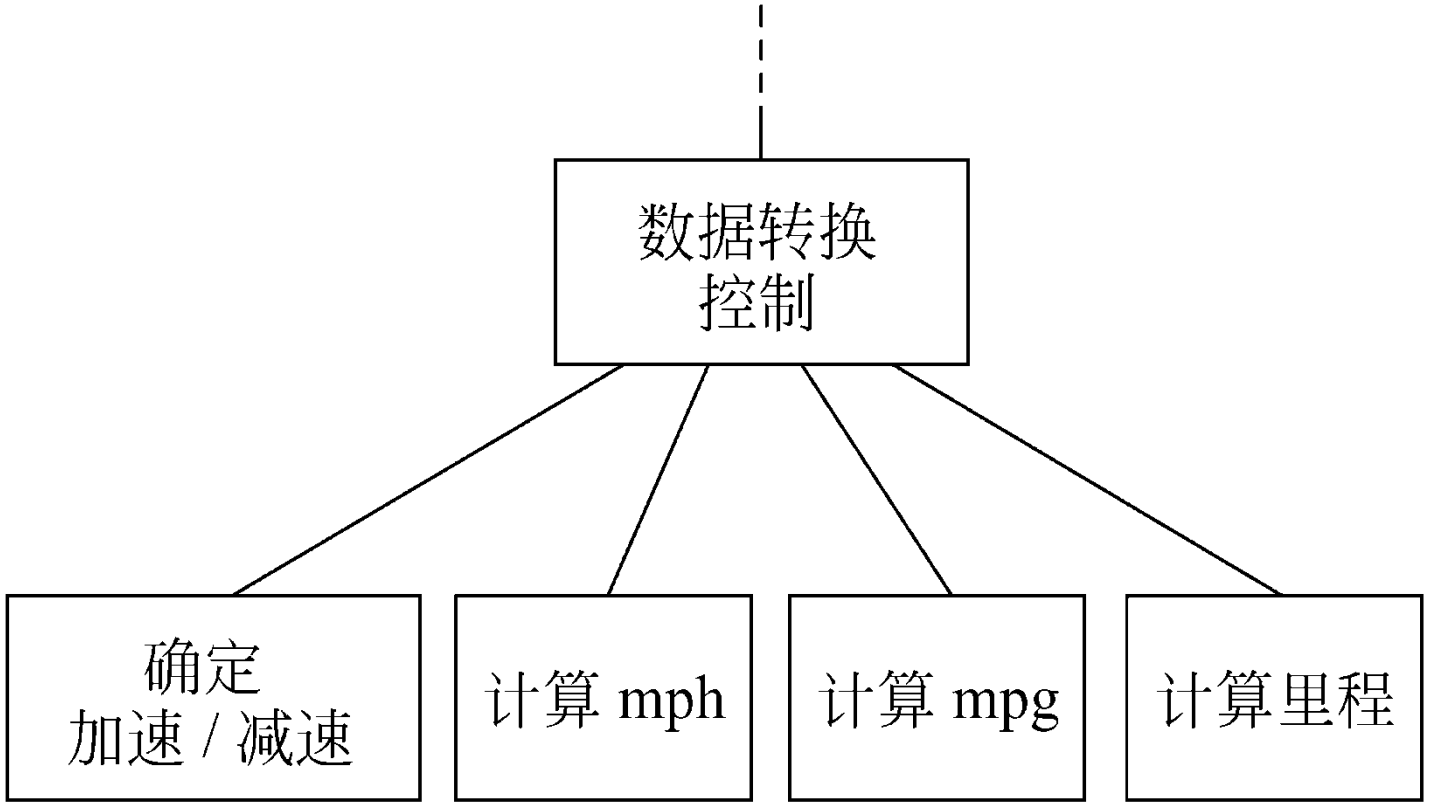


图5.17 未经精化的变换结构

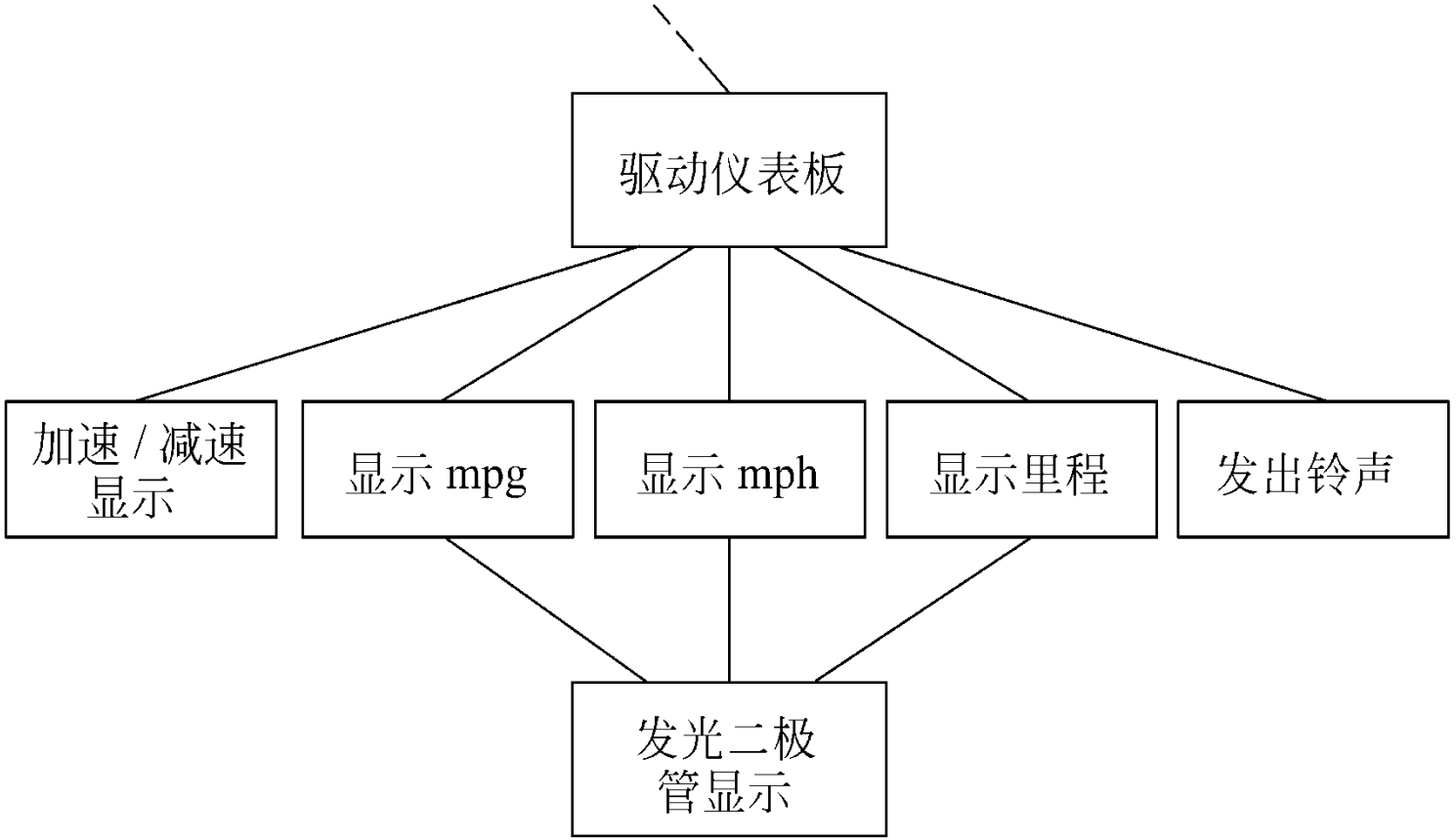


图5.18 未经精化的输出结构

对第一次分割得到的软件结构，总可以根据模块独立原理进行精化。为了产生合理的分解，得到尽可能高的内聚、尽可能松散的耦合，最重要的是，为了得到一个易于实现、易于测试和易于维护的软件结构，应该对初步分割得到的模块进行再分解或合并。

具体到数字仪表板的例子，对于从前面的设计步骤得到的软件结构，还可以做许多修改。下面是某些可能的修改：

输入结构中的模块“转换成rpm”和“收集sps”可以合并；

模块“确定加速/减速”可以放在模块“计算mph”下面，以减少耦合；

模块“加速/减速显示”可以相应地放在模块“显示mph”的下面。

经过上述修改后的软件结构画在图5.19中。

上述7个设计步骤的目的是，开发出软件的整体表示。也就是说，一旦确定了软件结构就可以把它作为一个整体来复查，从而能够评价和精化软件结构。在这个时期进行修改只需要很少的附加工作，但是却能够对软件的质量特别是软件的可维护性产生深远的影响。

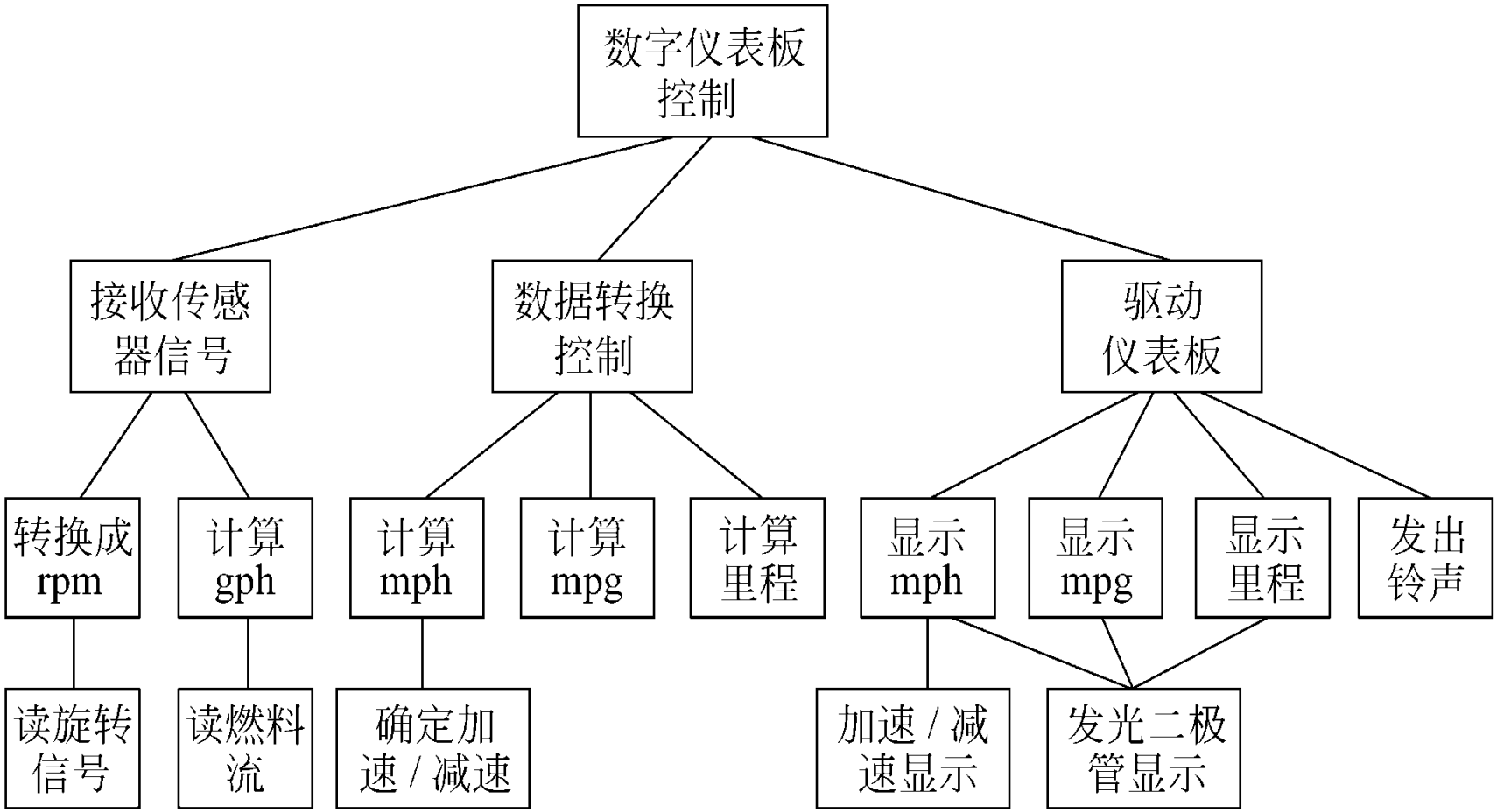


图5.19 精化后的数字仪表板系统的软件结构

5.5.4 设计优化

软件设计人员应该致力于开发能够满足所有功能和性能要求，而且按照设计原理和启发式设计规则衡量是值得接收的软件。

应该在设计的早期阶段尽量对软件结构进行精化。可以导出不同的软件结构，然后对它们进行评价和比较，力求得到“最好”的结果。

结构简单通常既表示设计风格优雅，又表明效率高。设计优化应该力求做到在有效的模块化的前提下使用最少量的模块，以及在能够满足信息要求的前提下使用最简单的数据结构。

(1) 在不考虑时间因素的前提下开发并精化软件结构；

(2) 在详细设计阶段选出最耗费时间的那些模块，仔细地设计它们的处理过程(算法)，以求提高效率；

(3) 使用高级程序设计语言编写程序；

(4) 在软件中孤立出那些大量占用处理机资源的模块；

(5) 必要时重新设计或用依赖于机器的语言重写上述大量占用资源的模块的代码，以求提高效率。

上述优化方法遵守了一句格言：“先使它能工作，然后再使它快起来。”

5.6 小结

总体设计阶段的基本目的是用比较抽象概括的方式确定系统如何完成预定的任务，也就是说，应该确定系统的物理配置方案，并且进而确定组成系统的每个程序的结构。因此，总体设计阶段主要由两个小阶段组成。首先需要进行系统设计，然后进行软件结构设计，确定软件由哪些模块组成以及这些模块之间的动态调用关系。层次图和结构图是描绘软件结构的常用工具。

在进行软件结构设计时应该遵循的最主要的原理是模块独立原理，也就是说，软件应该由一组完成相对独立的子功能的模块组成，这些模块彼此之间的接口关系应该尽量简单。

抽象和求精是一对互补的概念，在进行软件结构设计时一种有效的方法就是，由抽象到具体地构造出软件的层次结构。

软件工程师总结出一些很有参考价值的启发式规则，对如何改进软件设计给出宝贵的提示。在软件开发过程中既要充分重视和利用这些启发式规则，又要从实际情况出发避免生搬硬套。

自顶向下逐步求精是进行软件结构设计的常用途径；但是，如果已经有了详细的数据流图，也可以使用面向数据流的设计方法，用形式化的方法由数据流图映射出软件结构。这样映射出来的只是软件的初步结构，还必须根据设计原理并且参考启发式规则，认真分析和改进软件的初步结构，以得到质量更高的模块和更合理的软件结构。

在进行详细的过程设计和编写程序之前，首先进行结构设计，其好处正在于可以在软件开发的早期站在全局高度对软件结构进行优化。在这个时期进行优化付出的代价不高，却可以使软件质量得到重大改进。

习题

5-1 为每种类型的模块耦合举一个具体例子。

5-2 为每种类型的模块内聚举一个具体例子。

5-3 用面向数据流的方法设计下列系统的软件结构：

(1) 储蓄系统(参见习题2第2题)；

(2) 机票预订系统(参见习题2第3题)；

(3) 患者监护系统(参见习题2第4题)。

5-4 美国某大学共有200名教师，校方与教师工会刚刚签订一项协议。按照协议，所有年工资超过＄26 000(含＄26 000)的教师工资将保持不变，年工资少于＄26 000的教师将增加工资，所增加的工资数按下述方法计算：给每个由此教师所赡养的人(包括教师本人)每年补助＄100，此外，教师有一年工龄每年再多补助＄50，但是，增加后的年工资总额不能多于＄26 000。

教师的工资档案储存在行政办公室的磁带上，档案中有目前的年工资、赡养的人数、雇用日期等信息。需要写一个程序计算并印出每名教师的原有工资和调整后的新工资。要求：

(1) 画出此系统的数据流图；

(2) 写出需求说明；

(3) 设计上述的工资调整程序(要求用HIPO图描绘设计结果)，设计时请分别采用下述两种算法，并比较这两种算法的优缺点：

(a) 搜索工资档案数据，找出年工资少于＄26 000的人，计算新工资，校核是否超过＄26 000，储存新工资，印出新旧工资对照表；

(b) 把工资档案数据按工资从最低到最高的次序排序，当工资数额超过＄26 000时即停止排序，计算新工资，校核是否超过限额，储存新工资，印出结果。

(4) 你所画出的数据流图适用于哪种算法?

5-5 下面将给出两个人玩的扑克牌游戏的一种玩法，请你设计一个模拟程序，它的基本功能是：(1)发两手牌(利用随机数产生器)；(2)确定赢者和赢牌的类型；(3)模拟N次游戏，计算每种类型牌赢或平局的概率。要求用HIPO图描绘设计结果并且画出高层控制流程图。

扑克牌游戏规则如下：

(1) 有两个人玩，分别称为A和B；

(2) 一副扑克牌有52张牌，4 种花色(方块、梅花、红桃和黑桃)，每种花色的牌的点数按升序排列有2，3，4，…，10，J，Q，K，A等13种；

(3) 给每个人发3张牌，牌面向上(即，亮牌)，赢者立即可以确定；

(4) 最高等级的一手牌称为同花，即3张牌均为同一种花色，最大的同花牌是同一种花色的Q、K、A；

(5) 第二等级的牌称为顺子，即点数连续的3张牌，最大的顺子是花色不同的Q、K、A；

(6) 第三等级的牌是同点，即点数相同的3张牌，最大的同点是A、A、A；

(7) 第四等级的牌是对子，即3张牌中有两张点数相同，最大的对子是A、A、K；

(8) 第五等级的牌是杂牌，即除去上列四等之外的任何一手牌，最大的杂牌是不同花色的A、K、J；

(9) 若两人的牌类型不同，则等级高者胜；若等级相同，则点数高者胜；若点数也相同，则为平局。