特定领域建模方法在信息管理系统开发中的应用

陈晓航 （上海教育考试院信息中心 200233）
徐 锦 赵文耘 （复旦大学计算机系 200433）

摘要：特定领域软件构架(Domain－Specific Software Architecture, DSSA)方法对于软件重用，软件的可升级性和提高开发系列软件系统的效率都有重要的指导意义。本文主要介绍分层系统的 DSSA，及获得此领域的参照构架，用参考构架模型进行描述，并在一个实际系统开发过程中的应用实例。

关键字：特定领域参照构架 参考构架模型 构件 域

上海市教育考试院是一个负责上海地区各级各类教育考试及招生录取工作的专门机构，在开发信息管理系统的过程中，针对不同级别、类别的考试、招生信息处理过程中的典型需求和特殊需求，为了解决软件的重用问题，我们应用了特定领域建模的方法。

一、特定领域软件构架 DSSA 方法简介

DSSA 是通过对某个特定领域的分析，提出了该领域的典型需求，这样，此领域的一个特定需求就变成在典型需求上加上特殊需求，然后配置合适的构件，完成相应应用的开发。因而 DSSA 方法对于开发特定领域的软件生成器和指导开发系列软件都有重要的意义，由于经过特定领域的分析，设计者能够最大程度地提取此领域的可重用成分，从而提高软件的开发效率。参照构架、应用系统和其构架的关系如图 1：

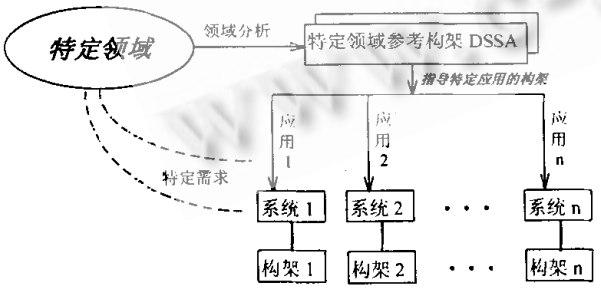

图 1

我们对分层系统软件构架的研究是基于 DSSA 的思想，寻找一种适应层次系统构架描述的模型，从而将对特定领域分析的结果(即参照构架)形式地描述出来，提高系列系统的开发效率，以求达到设计重用。

针对层次系统的特征和开发系列系统的要求，描述这种参考构架的模型应满足以下要求：

(1)能够定义原始的，可重用的构件。

(2)能够描述构件的组合，解释不同组合的不同涵义。

(3)构件应该满足可拆卸的特性，以便通过调换构件来实现不同组合。

(4)该模式能够用简洁的形式表示复杂的组合。

为了适应上述需求，我们找到了曾经用于描述 ADAGE[2]的参考构架的模型。之所以采用它，是因为它能够把“分层系统的不同系列”定义成为“可重用构件的不同组合”。在实践中，我们用参考构架模型描述了上海教育考试院信息中心的“成绩处理”领域(SCD)的参考构架，并完成了“会考成绩处理”、“普通高考成绩处理”、“成人高考成绩处理”这三种特定应用，形成一套系列系统。下面将介绍参考构架模型特征和用它描述 SCD 的具体细节。

参考构架模型主要包括构件和域的概念。构件：每个构件有其接口和实现，接口是唯一的外部可见的。每个构件是一个域的成员，而这个领域的每个构件用不同的实现方法来实现域所定义的公共接口。这样使得域中的成员可以互交换。如果构件 a1, a2, a3 是域 R 的成员，则它们也可以称为域 R 的库成员。构件可以用其他构件作为参数来实现其所属域定义的接口，而不需要了解作为参数构件本身的实现，所以一个构件可以被看成一层，而软件系统就是由多个不同的层构成的。参数构件相对于其引用构件是下层对上层的关系。域，构件，库，构件组合，领域和领域的系列软件系统的关系可以用图

2 来描述：

我们使用参考构架的设计原则主要为以下几点：

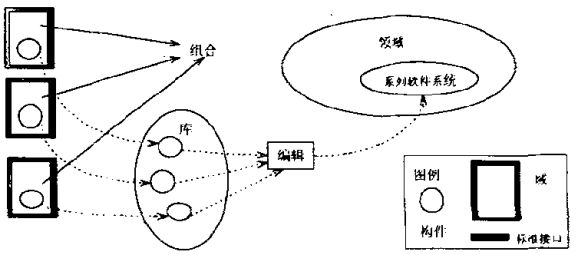

图 2

①可拆卸的构件如同积木一样，而形成系统的过程就是搭积木的过程。

②可交换的构件应该有标准接口。

③构件可以作为参数来传递。参数不同，则组合不同。

针对这种原则，参考构架的基本特征有：构件和构件域，参数，类型等式。

1. **构件**(component)**和域**(realm)

如果将层次系统中的一层看作一个虚拟机，则一个构件就是这个虚拟机的一种实现，而构成这个虚拟机所有实现的构件集合就是域。一个域有一个标准接口，这个域中的每个构件必须满足这个接口，但不同构件可以有不同实现。例如域 R={r1, r2}有两个构件 r1, r2, 它们是 R 的接口的不同实现。

2. **参数**

构件可以作为另一个构件的参数，表示两个构件的组合。构件的不同组合，即不同的参数可以表示不同性能和语义的系统。例如：S={s1(R), s2(R), ...}，则 s1 表示它通过组合 R 中的构件来实现 S 的接口。s1 完成了从 R 到 S 间的映射。另外有些构件组合了自身域的构件，则这种构件(即对称构件)可以描述递归的组合，例如 D={d1(D), d2(D), d3, d4}，则序列 d1(d2(d3))、d1(d2(d4)).....就描述了这种构件的任意组合的特性，由此系统的灵活性和多样性就可以通过这种构件的不同复合来表达。

3. **类型等式**

参考构架将系统简单地表达成为构件的组合，例如下面的两个系统：

system1 = s1(r1); system2 = s2(r2);

这两个系统是 S 的两个可互相交换的实现。当多个表达式使用同一个构件(组合不同，所以系统不同)，则该构件的重用度就被提高了。

4. **分层系统**

一个复杂系统总可以被表述为构件(节点)和构件的联系(边)的形式，如果构件之间的引用基本是单向和无环的，构件和被引用的构件之间的关系可以被看成分层的，其中，被引用的构件相对处于低层，如图 3(其中 A、B、C、D 分别为各个层次)：

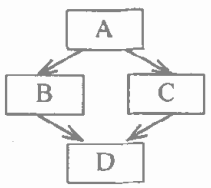

图 3

当然，为了能表达更为复杂的关系，参考构架还支持自身的递归引用的描述。

二、SCD 领域的分析和参考构架

下面介绍我们对“成绩处理”领域的分析和描述结果。特别地，为进一步开发系列“成绩处理”系统，我们的参考构架将“面对多种考试”作为目标。

领域分析：上海教育考试院专门负责上海市的普通高考、中学会考和成人高考的成绩处理，由于涉及面广，关系重大，所以，系统处理的数据量大，安全性，正确性要求高，所以长期以来，在考试院已经形成了一种处理模式：每个考生的每门成绩都是经过两遍录入，并且通过校对两遍录入，报告录入中的错误，并更正错误数据，才形成最终的考生成绩，成绩出来以后，通过计算得到总分，综合分，专业总分等处理分。在正确的成绩基础上，系统必须提供分析数据——对原始数据的统计结果，以供领导做决策。在决策之后，成绩可能还要进行调整。最后，各种“原始分数”，“再处理分数”和统计结果都要打印出来。所以整个处理系统将包括：数据子系统(DL)，成绩录入子系统(SCOREIN)，成绩校对子系统(SCORECHK)，成绩再处理子系统(SCORERED)，统计

子系统(STAT),打印子系统(PRINT)。

其中,DL 完成系统所需数据的收集和访问。这些数据包括"政策"数据,考生信息库,学校代码库,成绩库等等。SCOREIN 完成成绩的输入,SCORECHK 完成成绩的校对。SCORERED 完成单科成绩合并、成绩的调整、计算总分、综合分等等。STAT 提供统计功能。PRINT 完成数据集合的输出和用户自定义的表格的打印。这样一个"成绩处理"系统的各域可以表示成:

DL = {d1, d2, d3, ... | SCOREIN = {scorein}

SCORECHK = {scorechk({SCOREIN}) }

SCORERED = { scorered (DL, SCOREIN, SCORECHK)}

STAT = {stat(DL, SCORERED)}

PRINT = {print(DL, STAT)}

其分层构架如图 4 所示,在上述的域中,我们将列举某些域的构件,介绍这些构件将完成的功能。

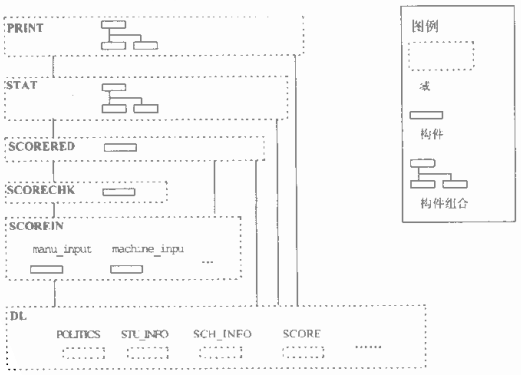

图 4

1. DL 域

它包括多个子域,如:POLITICS(政策制定和访问域),STU-INFO(学生基本信息访问域),SCH-INFO(学校代码信息访问域),SCORE(成绩库访问域)。所以 DL 域可以表示成:

DL = POLITICS ∪ STU-INFO ∪ SCH-INFO ∪ SCORE ∪ ...

2. SCOREIN 域

SCOREIN = {

manu-input(SCORE), /* 手工输入 */

machine-input(SCORE), /* 机读卡输入 */

...

add({SCOREIN}, {SCORECHK}),

select({SCOREIN})

},

其中,add({SCOREIN})构件表示单科成绩可能分为几种方式录入,例如高考外语考试成绩,是读答题卡(machine-input)成绩和手工阅卷(manu-input)成绩的和,所以,add(manu-input, machine-input)就可以表示上述需求(其另一个 SCORECHK 参数的作用将在 SCORECHK 域中讲)。另外,select({SCOREIN})表示可以选择参数 SCOREIN 域中的一种输入方法。例如:select(manu-input, machine-input)构件表示成绩录入将提供两种录入的方式供选择。add 和 select 构件是两个对称构件,用在这里,可以表达系统的可能的多种组合。

3. SCORECHK 域

以 SCOREIN 域的构件构件为参数,这里不详细说明。一个有趣的现象是,它和 SCOREIN 域的 add 构件发生关联——作为其参数。这表示从成绩录入到成绩校对再到成绩加和,可以有个反馈过程,如果将手工录入的两遍数据通过校对后再和机阅数据合并形成某单科的最终成绩。则此需求就可以表达为:

add (machine-input, scorechk (manu-input, manu-input))

而 add(machine-input, manu-input)就没有校对这个步骤,而直接加和得到最终成绩。

4. STAT 域

这个域较为复杂,但从抽象层次上讲,统计总可以表示成如图 5 所示:

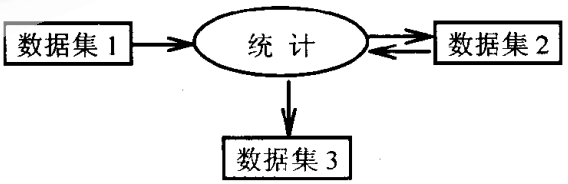

图 5

这种表示,如果将原始数据作为数据集 1 输入,则通过统计,可以得到数据集 2,而数据集又可以作为一次中间结果,所以经过再次统计,得到最终所需结果—数据集 3。因而根据这个特点,参考构架的对称构件的描述为我们提供了表达这种递归的过程。统计域可定义如下:

```
STAT = {
average(SCORERED, DL, intepretor1 / * 字段 1 * /,
intepretor2 / * 字段 2 * /),
    / * 对最终成绩或中间成绩按字段 1 分类求字段 2 的平均 * /
total(SCORERED, DL, intepretor1, intepretor2),
    / * 对最终成绩或中间成绩按字段 1 分类求字段 2 的和 * /
standard-mk ( SCORERED, DL, intepretor1, intepretor2),
    / * 对最终成绩或中间成绩按字段 1 分类求字段 2 的标准分 * /
count(SCORERED, DL, intepretor1),
    / * 对最终成绩或中间成绩求按字段 1 分类的人数 * /
cal-zgx(SCORERED, DL, POLITICS),
    / * 对最终成绩或中间成绩按指定政策计算资格线 * /
......
serial({STAT}),
vertical(STAT),
combine(STAT, STAT) }
```

其中，serial 构件表示按参数次序来依次统计，vertical 构件将一些分类统计结果(竖表)按分类的第一字段汇合成横表，combine 构件将两个统计的结果进行自然连接。

5. PRINT 域

一般，打印可以分为以下几类：基于数据库的简单表打印和自由表格打印。我们将打印域定义为：

```
printer = {
simpleheadtab(DL, STAT, Varlist), / * 简单表头基于数据库打印 * /
complexheadtab(DL, STAT, Script, Varlist), / * 复杂表头基于数据库打印 * /
freereport(Varlist, Script) / * 自由报表构件 * /
}
```

其中，Varlist 是变量表构件，Script 是报表格式描述构件。

6. **简单实例**

下面，我们将就上述的几个域，例举一些构件，并介绍一个应用系统的模型是如何使用这些构件组合出来的，同时我们还可以看到这种描述使相同领域的构件很容易被替换。在 DL 域，我们为 POLITICS 子域和 SCORE 子域制定了如下的构件：

```
POLITICS = {
cr-total, / * 成人高考计算总分方法 * /
cr-compose, / * 成人高考计算综合分方法 * /
cr-totalsub, / * 成人高考计算专业总分方法 * /
pt-cal-total, / * 普通高考计算总分方法 * /
pt-adj-subject, / * 普通高考单科调整分计算法 * /
cr-stat-zgx, / * 成人高考计算资格线方法 * /
pt-stat-zgx, / * 普通高考计算资格线方法 * /
......
};
SCORE = {
tabin1, / * 一遍录入库读写 * /
tabin2, / * 二遍录入库读写 * /
scorepart / * 部分分读写 * /
......
}
```

其他域的构件如上几节所例举。现在有一个需求是：成人高考考试成绩需要两遍录入，并校对，有些课程的分数分为机器阅卷分 + 手工阅卷分之和，手工阅卷分两遍录入后，经比较合成一致成绩，再和机器阅卷分合并，当成绩完全正确后，对成绩计算总分和综合分，并且计算资格线。(由于此系统表达式很长，我们用宏来替换一些子表达式)

```
X = add(machine-input(scorepart), scorechk(manu-input(tabin1), manu-input(tabin2))
Y = scorered(X, {cr-cal-total, cr-cal-compose})
system = cal-zgx(Y, cr-stat-zgx)
```

这样，上述需求的应用系统就可以表达出来了。这里的例子较为简单，一般复杂的系统需要分成若干子系统(如 X 和 Y)，这些子系统可以被多个构件引用形成其他子系统，最综的系统可以是一个表达式，也可以是多个表达式的并(∪)。

参考文献

[1] The Design and Implementation of Hierarchical Software Systems with Reusable Components. ACM Transactions Software Engineering and Methodology. Vol1. No4, Oct. 1992

[2] Design reuse and framework in th smalltalk - 80 programming system. ACM Press, 1989.

[3] The GenVoca Model of Software - System Generators. IEEE Software 1994.

(来稿时间：1998 年 7 月)