维普资讯 http://www.cqvip.com

基于元设计模式的构造型说明语义*)

蔡家楣
(浙江工业大学信息工程学院　杭州310014)

摘　要　设计模式对于构建可复用的软件框架有重要意义。本文用构造型说明法讨论了满足结构化要求的各种元设计模式应具备的语义特性。提出了用构造型说明来描述基于元模式的模式规约的思想。

关键词　设计模式，元模式，语义，构造型说明法

The Semantics of Structured Specifications Based on Meta- Design Pattern

CAI JIA-MEI
(Institute of Information, Zhejiang University of Technology, Hangzhou 310014)

Abstract　The study of design pattern contains important significance for setting up the reusable software frame. This paper with the constructive specification method discusses the semantics to meet the structured requirements for various meta- design pattern. And provides a approach to describe the design patterns based on meta- design pattern with the constructive specification method.

Keywords　Design pattern, Meta-design pattern, Semantics, Approach of structured specification

1　前言

设计模式是一种面向对象软件设计的经验记录，使设计者可以以原有经验为基础，重复使用成功的设计。每个设计模式都系统地命名、解释和评价了面向对象系统中某一个重要的设计。这些模式解决专门的设计问题，使面向对象的设计更灵活、精巧，最终可以重复使用。因此，设计模式已经成为软件界最受关注的问题之一，许多当代的先进软件中已大量采用了软件设计模式的概念。

设计模式被大量应用在框架设计中。框架是指在一个特定的领域中的一组相互协作的类，集中了常用于该应用领域的设计决策，当要为一个特定要求的软件构建一个可复用的设计时，可以通过重载框架类的方法建立框架抽象类的特定应用子类，来定制一个特定应用的框架。框架强调设计复用，其抽象程度高于代码复用。大部分设计模式都可描述框架：在框架中应用设计模式的关键是与应用领域相关但与具体应用无关。

目前，描述模式主要有以下方法：

·格式描述(Alexander)针对上下文、问题、实例、含义，说明应用的原理、设计模式、规则、构成的解、导出的新上下文和模式。这是一种接近自然语言的古老方法。

·图示方法：用类似 Jemes Rumbaugh 的 OMT 方法的类和对象关系表示设计的结果。记录决策和权衡过程、设计模式运行的实例。这是目前重要和实用的方法。但该方法对于框架(尤其是应用程序框架)来说描述能力受限。

·对象语言描述，一般是给出一段典型的例程来描述一个设计模式。这种描述方法优势在于可以自动进行设计模式的实现。目前已经实现了一些自动代码生成系统或工具。但主要缺点是：在某些情况下过于繁琐；引入新的符号，会使符号系统复杂化；要求过分精密，可能造成冗余和主次不易分清。总之这种方法的抽象程度过低。无法简洁地表现客观对象。

·形式化方法。其核心思想是用基于规范的符号来描述面向对象系统。其中著名的如 Helm(1990)形式合约(formal contracts)方法、Shah (1989)对象/关系建模语言等。其特点是抽象，有局限性，仅在小规模程序中用于程序规范的规格说明和验证。基于上述原因，许多学者认为，形式化使模式与具体问题相匹配变得困难，因为问题通常不是形式化的。让解决方案的形式化难以抓住模式的关键并创建有效的变例。形式化的解决方案可能使得模式太概括而缩小了模式的可应用性[1]。Erich Gamma 甚至建议目前更重要的问题是探索新的模式而不是将模式形式化[2]。

然而，当前大多数设计模式，包括著名的 Erich Gamma 所总结的设计模式目录(design Patter catalog)存在的主要缺点是：所涉及的框架与应用领域关系不大；缺乏把握框架设计的方法，以能在新框架的设计过程中复用。

最初提出设计模式的目的是希望用结构化方式使设计活动成为一种确定的和可重复的过程。最终形成一种模式语言。因此，尽管模式的形式化遇到上述阻碍，但并不影响我们在设计模式的结构化语义方面作更多的探索。本文即希望通过讨论模式构造型说明来探讨复合设计模式的构建问题。

2　模式和元模式

2.1　模式基本方法和类

2.1.1　基本方法　一个应用程序总是存在两种成分：可变部分和不变部分。可变部分称热点(hot spot)，不变部分称为凝固点(frozen spot)。确定热点是面向对象设计的难点。好的框架将提供足够多的热点使之适用于不同的应用。设计模式就是要指出如设计实现与热点匹配的框架，即适应需求的

*)本项目得到国家重点基础研究发展规划项目(国家973项目 G1999032701)和浙江省自然科学基金的资助。**蔡家楣**　教授，主要研究方向：软件开发环境，软件工程技术和方法。

维普资讯 http://www.cqvip.com

框架。

在包含单个类或一组交互类的框架模式的设计中，可以将对象的操作方法定义为模板方法和挂钩方法：

·模板方法（复杂方法）是定义类的抽象行为、控制流或者类之间关系的一种方法。通常用于实现应用框架的凝固点。模板方法可以定义在同一个类中，类的父类中，或者其他的类中。复杂方法（模板方法）是基于基本方法（挂钩方法）实现的。

·挂钩方法（基本方法），通常用于实现热点。挂钩方法可以是：抽象方法：只定义接口，不定义实现的方法；常规方法：能完成一定意义行为，但不引用别的方法的方法；模板方法。

2.1.2 *基本类* 元模式的类/对象组成的通常做法是由模板方法来表示凝固点，而用挂钩方法表示凝固点中的热点。模板方法和挂钩方法可以结合在一个类中，但更好的方式是将凝固点与热点分布到不同的类中。这样，根据模板方法和挂钩方法的区别可以定义模板类和挂钩类（template and hook classes）。挂钩类是相应模板类的参数。

2.2 元模式

元模式（metapattern）是一类特定的设计模式，适合于构造与具体领域无关的框架。用元模式能在元层次上构造框架，方便而有力地对任何框架设计模式进行分类和描述。有利于形成以框架为中心的软件开发方法、热点驱动的方法等等。由元模式表达面向对象的设计思想，以及将元模式与框架的组成部分相联系的做法不仅仅对那些成熟的框架有用，对开发框架设计也有意义。

2.2.1 *元模式的组成属性* 元模式可以由构建模板类和挂钩类以及相应的对象的组成属性来决定。这种组成属性有引用关系和继承关系。引用关系的语义可以是一个类对象使用另一个类对象，或者是一个类对象包含另一个类对象。采用何种语义则取决于特定的应用领域。

模板类和挂钩类的继承关系有两类：模板类和挂钩类之间没有继承关系（可以是同一个类）；模板类是挂钩类的子类或反之。

但模板类只有从其挂钩类派生才是有意义的，也即挂钩类是基本类。

2.2.2 *不同基本组合的元模式* 根据其组成属性有如下不同基本组合的组成元模式。

·同一性元模式：模板类和挂钩类是同一个类，它们的对象也是同一个。

·连接元模式：模板类对象包含一个或多个挂钩类对象的引用，两个类之间没有继承关系。

·递归连接元模式：模板类对象引用一个或多个挂钩类对象，模板类是挂钩类的子类，根据模板类和挂钩类不是同一个类。

·递归同一性元模式：模板类对象引用挂钩类任意数的对象，模板类是挂钩类的下层子类。根据模板类和挂钩类是同一个类[3]。

3 元模式的构造型说明

3.1 模式的构造说明法

因为模式是一组相互协作的类，而模式中的类（class）很象抽象数据类型。在这一思想下，模式就可以被看成一组程序规约。而通常，一个程序规约可以看成为一个多类子代数（multisorted algebra）。对每个类子（sort）而言，有一个该类子的载体集（carrier set）。而该代数则由这些载体集和其上的方法集组成。

模式的构造说明法（constructive specification method）是用构造型方法来定义载体和方法。在规约中，载体集被定义为一个形式语言的等价类，而方法则被定义为递归程序。这种说明法在结构化特性和抽象特性两方面都有较好的表现[4]。

3.2 元模式基本元素的构造型说明

由上述讨论可见，要说明元模式的构造型语义，就是要找到描述模式的基本方法、基本类和满足不同组成属性的类的交互关系的说明。

3.2.1 *基本类和基本方法的说明* 如果我们用 Σ 代数作为构造说明法的形式机制，那么有：

定义1 一个基本类的规约的基调（signature）可以简单地表示为 $\Sigma=(S,O)$。其中 $S=\{S_i|i\in I\}$ 为一个有限集，其中 I 是一个有限下标集，每个 S_i 是一个类子，$O=\{O_j\ j\in J\}$ 为另一个有限集，J 也是一个有限下标集，每个 O_j 是一个方法。一个 k 目方法 $O_j(k\geqslant 0)$ 可表示为

$O_j:S_1\times S_2\times\cdots\times S_k\rightarrow S_{k+1}$

$S_1\times S_2\times\cdots\times S_k\rightarrow S_{k+1}$ 叫作该方法的算术，或叫全秩。诸 $S_i(1\leqslant i\leqslant k+1)$ 可以相同或部分相同，也可以不相同。方法集是从载体集中去获取自变元和值的。所谓载体集是指用这个数据类型中涉及的类子的某几个方法，即由所谓构造者（constructor）来生成的一种项语言。

定义2 如果有 $\Sigma=(S,O)$，让 x 为 S 上的变量集，则关于某个类子 σ 的项 $\Sigma(x)$ 为

(i)变量 x_σ $\sigma\in S$；

(ii)方法 $OP:\rightarrow\sigma$ $OP\in O$；

(iii)方法 $F:\sigma_1\times\sigma_2\times\cdots\times\sigma_n\rightarrow\sigma(n\geqslant 1)$

记作 $F(t_1,t_2,\cdots,t_n)$，而 $t_1,t_2,\cdots,t_n$ 分别是类子 $\sigma_1,\sigma_2,\cdots,\sigma_n$ 的 $\Sigma(x)$ 项。

这样，具有类子集 S 的基调为 Σ 的一个代数系统 A 就包含了两个部分：

(1)一个关于每个 $\sigma\in S$ 的类子的载体集 σ^A

(2)一个来自 Σ 中方法名为

$F:\sigma_1\times\sigma_2\times\cdots\times\sigma_n\rightarrow\sigma(n\geqslant 0)$ 的方法的完全函数

$FA:\sigma_1^A\times\sigma_2^A\times\cdots\times\sigma_n^A\rightarrow\sigma^A(n\geqslant 0)$

的集[5]。

于是，如果一个挂钩方法集记作 $G=\{g_i|i\in I\}$，模板方法集就记作 $M=\{M_j|j\in J\}$。那么一个模板方法 m 为：

$m(g_1,g_2,\cdots,g_n)(n\geqslant 0)g_i\in G$

而挂钩方法 g 为：(1)方法 OP；(2)方法 F；(3)模板方法 m。

这样构成的规约叫做一个基本类的规约，由于这一规约中所有的载体和方法都来自本规约所定义的基调 $\Sigma=(S,O)$，或者系统内嵌的基本规约定义的类子的载体和方法集，于是具有较好的模块化结构化特性。

3.2.2 *元模式的构造型说明语义*

定义3 如果全部 Σ 代数的类记作 AlgΣ，输入规约的基调为 Σ_i，输出规约的基调为 Σ_e，一个模式规约的基调应被看作序偶 (Σ_i,Σ_e)。一个规约操作 ρ 有：

$\rho:\mathrm{Alg}\Sigma_i\rightarrow\mathrm{Alg}\Sigma_e$

这一映射也可以是多元的，这样上式就扩展为：

$\rho:(\mathrm{Alg}\Sigma_{i1},\mathrm{Alg}\Sigma_{i2},\cdots,\mathrm{Alg}\Sigma_{in})\rightarrow\mathrm{Alg}\Sigma_e$

一个模式的规约的基调应被看作序偶 (Σ_i,Σ_e)，但是不同的模式可以有不同的语义。如果 $m1$，$m2$ 是两个基本类代数的

维普资讯 http://www.cqvip.com

规约，$S(m)$是组成元规约，A 是有基调 $S(m)$的代数系统类中的一个，有 $A\in \mathrm{Alg}\ S(m)$，M 为 A 的语义，用 $M(m)(A)$表示。

（一）同一性元模式：由于模板类和挂钩类是同一个类，它们的对象也是同一个，因此它的元模式规约就是一个基本类的规约，这一规约中所有的载体和方法都来自本规约所定义的基调 $\Sigma=(S,O)$，或者系统内嵌的基本规约定义的类子的载体和方法集。

（二）连接元模式：模板类对象包含一个或多个挂钩类对象的引用，两个类之间没有继承关系。

该规约 $m1\bigcirc m2$的输入基调是 $m1$的。而输出基调则考虑下述情况。

如果 $lso1$和 $lso2$是 $m1$和 $m2$的类子和操作表，那么存在

$S(m1\bigcirc m2)=(Si(m1),Se(m1)\cup\rho(Si(m2)))$

ρ 是一个在 $S_i(m2)$基调上的操作序偶$(lso1,lso2)$，调用 ρ 就可以用$(lso1,lso2)$来改变 $Si(m1)\cup Se(m2)$的基调。其条件为：

(i)$lso1$的类子和操作全部来自 $Si(m2)$；(即只允许对 $m2$ 输入操作)

(ii)ρ 被加进 $Se(m1)\backslash Si(m1)$的非继承操作中去，并要改变它们的算术；

(iii)对于 $Si(m1)$的每个类子和操作 so

$\rho(so)\notin\rho(Se(m)\backslash Si(m))$

这样可以避免新的输入名和非继承输出名之间的混淆，即对某个 so 的规约操作，不会和非继承类子和操作名混同。

于是 $M(m1\bigcirc m2)(A)=M(m1)(A)\cup M(m1)(M(m2)(A))$。其逻辑含义如图1所示。

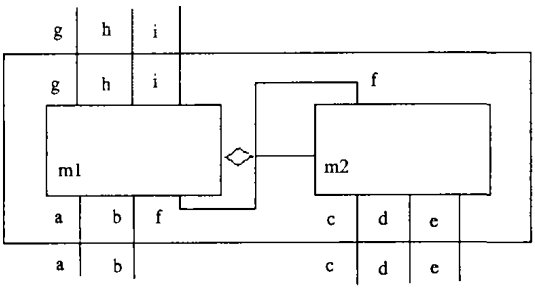

图1　连接元模式的逻辑含义

（三）递归连接元模式：模板类对象引用一个或多个挂钩类对象，模板类是挂钩类的子类，但模板类和挂钩类不是同一个类。

规约 $m1◎m2$的输出基调和输入基调都是 $m1$的。为 $S(m1◎m2)=(Si(m1),\rho Se(m1))$。$\rho$ 是一个在 $Si(m)$基调上的递归操作序偶$(lso1,lso2)$。其上下文条件是

(i)在 $Se(m1)$中和$(Se(me)$中有同名操作 ρ，且 $\rho(Se(m1))=\rho(\rho Se(m2))$。

(ii)$Se(m2)=Se(m1\backslash lso)$即 $m2$的输出是 $m1$的部分 lso 输出。

(iii)$Se(m2)\cap Se(m1)\subseteq Si(m1)$某些由 $m1$输出并由 $m2$ 输出的类子或操作则是由 $m1$向 $m2$继承的。

其代数系统 A 的语义为

$M(m1◎m2)(A)=M(m1)(A)\backslash lso\cup M(m1)(A|\rho(Se(m2)))(so)$

$m1◎m2$的逻辑含义如图2。

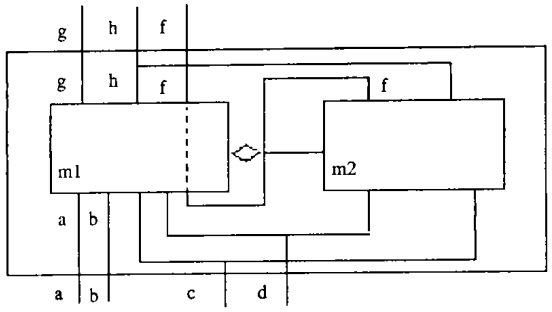

图2　递归连接元模式的逻辑含义

（四）递归同一性元模式：模板类对象引用挂钩类任意数的对象，模板类是挂钩类的下层子类。模板类和挂钩类是同一个类。因此，该规约的输入基调和输出基调是本规约所定义的基调，即 $S(m)=(Si(m),\rho Se(m))$，即调用 ρ 就可以用$(lso1,lso2)$来改变 $Se(m)$的基调。

其条件为：

(i)ρ 被加进 $Se(m)$中去

(ii)$\rho(Se(m))=\rho(\rho Se(m))$其代数系统的语义是：

$M(m)(A)=M(m)(A)|\rho^{-1}$

即(A 关于 ρ 的商集)，其逻辑含义见图3。

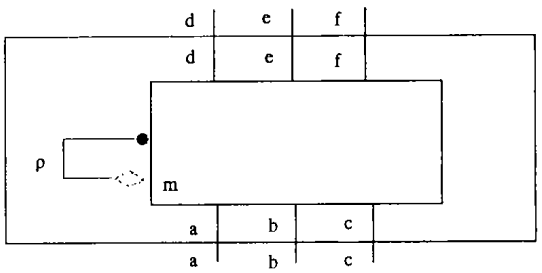

图3　递归同一性元模式的逻辑含义

4　基于元模式的构造型模式规约说明

对于需要进行分步求精开发的大型程序来说，传统意义上的仅对程序的代数语义加以研究已显不足，可以参照表示抽象数据类型的说明方法应用于模式构造过程，用具有松散语义的抽象数据类型规约及其集成来表示它们。模式构造型说明的基本思想是：如何用统一的方式来描述一个设计模式的语义，如何用一些基本的模式(元模式)来复合成新的设计模式，直至把任意复杂的应用问题用一些循环递归的元模式来表示。在这种复合中，一个模式既可以是只包含一些基本类(元类)和基本类的交互操作的元模式，也可以是一个包含多个元模式的复合模式，甚至是复合模式的集合。

因此，仅使用元模式规约来描述高度复杂的客观世界显然是远远不够的，于是我们希望从一些已有的规约出发，来构成任意复杂的设计模式规约。为了达到这一点，仅对一个代数的数据和操作进行说明是不够的，更重要的工作是扩充一个代数。也就是说，建立一种模式规约操作，把它作为映射一个输入代数到输出代数的算符来看待。如果把输入代数定义为一个标准代数，它只包含了某些类子和操作，当把某个规约操作施用于这个输入代数时就产生一个扩充的包含了附加输出类子或操作的代数，它可以成为一个新的标准代数。通过不同的结构来对规约进行诸如组合、合并、更名、“忘记”部分类子或操作和产生子代数系统或商代数系统的规约等。这样产生

（下转第163页）

维普资讯 http://www.cqvip.com

令序列中的停顿。微调度使得指令调度模块在执行时不用考虑与机器相关的一些约束条件，如功能部件的分配，指令模板的选择和机器的宽度等等。

从图7可以看出，微调度是如何与指令调度协同工作的：当指令调度选定一个待调度的指令后，它通过调用微调度来判断该指令是否能够合法地调度到指定的周期。微调度在判断这一调度是否合法需要考虑三个方面的因素：功能部件的分配、指令模板的选择以及依赖关系的检查。微调度首先为这个指令分配相应的功能部件，分配成功就形成一个合法的预留表，我们根据预留表查询自动机的状态（自动机的状态中保存了对应于这个预留表的所有合法指令格式）来选择合适的模板，在查询的同时也要兼顾到满足依赖关系。最后，微调度给出是否能在本周期中调度给定的指令。当指令调度最终决定要调度这一条指令时，指令调度调用微调度来实现指令的重排和拼装。

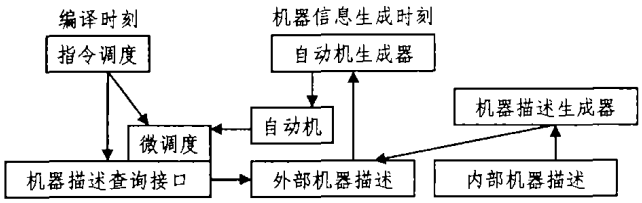

图7　参数化机器模型与微调度

关于微调度的具体实现详见文[14]。

6　试验数据

在这一节我们评估 ORC 代码生成的性能。用于试验的 ORC 版本是2.0[1]。测试例是 SPEC CPU2000[15]的整型基准测试程序。运行测试例的机器是 Itanium-1工作站（Itanium-1是 IA-64体系结构的一个具体实现），频率为733Mhz，具有2MB L3 cache 和1GB 的内存。运行的操作系统是 RedHat 7.2 for IA-64。

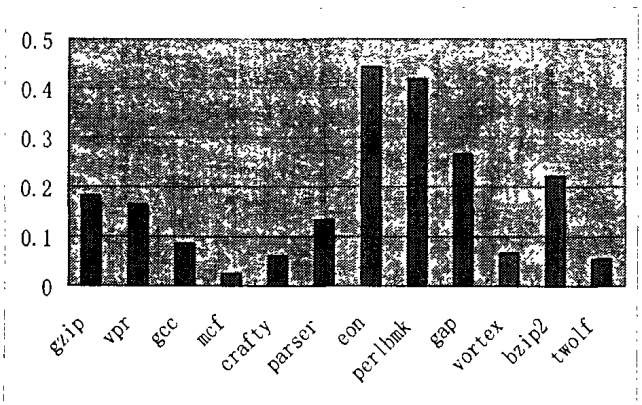

图8　ORC 相对于 Pro64的加速比

图8给出了 ORC 相对于 Pro64的加速比。两者的编译选项均为 O3。我们不打开 IPA 选项，这样使得两者的性能差别只和代码生成有关。从图8可以看出，ORC 相对于 Pro64对12个测试例子均有不同程度的提高，其中 eon 和 perlbmk 的加速比最为明显，分别达到了44%和42%。12个例子的平均加速比为23%。

总结　本文主要介绍了 ORC 的代码生成部分的关键技术。ORC 充分利用了 IA-64体系结构的特点，如谓词、控制投机和数据投机等硬件支持，来挖掘指令级并行的潜力。实验结果表明，ORC 的代码生成充分发挥了 IA-64体系结构的特点，相对于 Pro64有23% 的性能提高。IA-64作为一种新的体系结构，有很多的特性可以加以利用。ORC 良好的性能为研究 IA-64体系结构和基于区域的编译优化技术提供了良好的编译平台。

参 考 文 献

1　ORC team, ORC 2.1 suite. http://sourceforge.net/projects/ipf-orc, 2001-2003
2　Intel Inc., http://www.intel.com/design/itanium/family/, 2003
3　Allen J R, et al. Conversion of control dependence to data dependence. In: Proc. 10th ACM Symp. Principles of Programming Languages, Jan. 1983. 177～189
4　Huff R. Lifetime-Sensitive Modulo Scheduling. In: Proc. of the Int. Conf. on Programming Languages, Design and Implementation, 1993. 318～328
5　Rau B, et al. Register allocation for software pipelined loops. In: Proc. of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation, June 1992. 283～299
6　Chow F, Hennessy J. Register allocation by priority-based coloring. In: Proc. of the ACM SIGPLAN Symposium on Compiler Constructions, June 1984
7　Gillies D M, Ju D R, Johnson R, Schlansker M. Global predicate analysis and its application to register allocation. In: Proc. 29th Int. Symposium. Microarchitecture, Dec. 1996. 114～125
8　刘旸，张兆庆，乔如良. 基于区域的编译框架. 计算机学报，2003，26(2)：188～194
9　Bernstein D, et al. Code duplication: An assist for global instruction scheduling. In: Proc. of the 24th Annual Intl. Symposium on Microarchitecture, Dec. 1991
10　Bharadwaj J, et al. Wavefront Scheduling: Path-Based Data Representation and Scheduling of Subgraphs. In: Proc. of the 32nd Annual Intl. Symposium on Microarchitecture (MICRO32), Dec. 1999
11　Moon S-M, Ebcioglu K. Parallelizing Non-numerical Code with Selective Scheduling and Software Pipelining. ACM Transactions on Programming Languages and Systems, 1997. 853～898
12　Roy, et al. A unified compiler framework for control and data speculation. In: Intl. Conf. on parallel architectures and compilation techniques, Oct. 2000
13　Intel Inc., KAPI Programmer's Guide. http://developer.intel.com, June 2000
14　Chen D-Y, et al. Efficient Resource Management during Instruction Scheduling for the EPIC Architecture. In: Intl. Conf. on parallel architectures and compilation techniques, Sept. 2003
15　Stan Performance Evaluation Corporation. http://www.spec.org/cpu2000, 2003

（上接第158页）

的规约叫复合规约。

这样一个任意复杂的代数就可以由循环递归的嵌套的标准代数的规约来表示，形成一种自然的层次结构，它使我们联想到面向对象设计中的类层次。因此这种模式规约的构成对于解决面向对象的框架应用是很有利的。

参 考 文 献

1　Buschmann F, et al. Pattern-Oriented Software Architecture Volume: A system of Patterns. Publisher Wiley, John & Sons, Incorporated, Nov. 1995
2　Gamma E, et al. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995
3　周之英. 现代软件工程（下）新技术篇. 北京：科学出版社，1999
4　Loeckx J. Algorithmic specification. A constructive specification method for abstract Data Types. New York, ACM Transactions on Programming Languages and System, 1987, 9(4): 646～685
5　Caijia-mei. The Sequence Modeling Method Based on ECC in Developing Program Specifications. Journal of Computer Science and Technology, 1999, 14(7): 337～348