# 基于过程蓝图的重构操作*)

刘建宾　杨林邦

（北京信息科技大学计算机科学与工程系　北京 100101）（汕头大学计算机系　汕头 515063）

**摘　要**　提出基于过程蓝图的模型重构操作形式，将传统的基于源代码的程序过程重构变为基于过程蓝图的可视化重构，使基于过程源代码的重构能够在更高抽象层次的过程模型中得到应用，避免程序源代码的语法分析，简化重构过程及其实现，从而提高了重构处理的效率。

**关键词**　模型重构，过程蓝图，前置条件，后置条件，行为保持

## Refactoring Operations Based on Procedure Blueprint

LIU Jian-Bin　YANG Lin-Bang

(Dept. of Computer Science and Engineering, Beijing Information Science and Technology University, Beijing 100101)

(Department of Computer Science, ShanTou University, ShanTou 515063)

**Abstract**　In this paper, a refactoring's formalization based on procedure blueprint is proposed, which changes refactoring form from based on source program code to based on visual software model. By using this formalization, refactoring transformation operations based-on the source code at procedure level can be applied on the more abstractive procedure model, and the process of analyzing the syntax of the program is avoided, so that the process and implementation of the refactoring is simplified, and the higher efficiency than the traditional method can be gained.

**Keywords**　Model refactoring, Procedure blueprint, Pre-condition, Post-condition, Behaviour preservation

## 1　引言

重构是行为保持的程序变换和对软件进行改进的过程。重构不改变程序的外部行为，仅仅改变程序的内部结构[1]。通过重构，不仅可以提高程序的可维护性和重用性，而且还能帮助程序员更快地找到程序错误，提高软件的质量。因此，软件重构技术成为软件工程学的重要分支与内容。

目前，在软件重构技术领域取得的理论研究与工具开发成果主要集中在基于程序源码的半自动代码重构方面。现有研究表明，文本形式的程序源码表示并不能很好地支持重构。由于源码对程序元素之间的很多关系缺乏直观和明确的表示，从而需要构造其他支持重构的表示，例如控制流图，程序依赖图等，这样就导致了程序分析和计算费用高，重构过程也变得缓慢。为此，不少学者提出需要在更高抽象层次的软件模型上应用重构并解决相关理论和技术问题，使模型重构成为一个新的研究方向。此外，OMG（Object Management Group）在颁布统一建模语言 UML（Unified Modeling Language）规范的基础上，于 2001 年进一步提出模型驱动体系结构 MDA（Model Driven Architecture）规范[2]，为软件技术的研究与开发指明方向，受到产业界和学术界的高度关注。模型驱动软件开发方法以模型为中心，目标是要实现从软件模型到最终代码的自动转换与生成，是 OMG 推行的新一代软件开发范型。在此背景下，基于模型的软件重构理论与方法的研究正成为软件重构技术研究的热点。

模型重构可以分为框架级和过程级两个层面。框架级重构是对软件的体系结构进行行为保持的高层模型变换，主要在类图等表示软件架构的 UML 模型图上完成。过程级重构是对类方法的过程模型进行行为保持的底层模型变换，主要在表示算法过程的模型图上完成。对完整的模型重构而言，框架级与过程级的模型重构二者都不可缺少，他们相辅相成。近年来，在基于类图和设计模式的框架级高层模型重构方面已开展较多研究工作并取得成果[3~6]，而较底层的过程级模型重构的研究较为罕见。虽然 William G. Griswold[7] 和 Bowdidge[8] 分别用程序依赖图和星型图对过程级的重构进行研究，但他们的方法属于代码重构，需要构造其他额外的图形，存在程序分析和计算费用高等问题。本文提出一种新的过程级模型重构形式和方法—基于过程蓝图[9]的重构。

## 2　重构操作定义的基础

### 2.1　程序表示模型及实体

模型重构基于表示程序的可视化模型。在本文研究中，采用类图和过程蓝图作为程序表示的模型。类图表示组成程序的类及其关系，过程蓝图表示类方法的过程模型。类图和过程蓝图从框架和过程两个层面完整地表示了程序类结构与类方法的静态可视化模型，且与程序代码存在直接的映射关系。

Roberts 指出带有类型和引用信息的抽象语法树是表示重构程序的一种较为理想的形式；抽象语法树所包含的丰富信息足够用于重构，并且能够快速的创建[10]。过程蓝图是一种基于抽象语法树的可视化过程建模语言，实现了抽象语法树内外部表示的统一，显示包含产生程序过程代码的所有语义和表示信息[9]。因此，过程蓝图具有抽象语法树对重构支持的适用性、能力和优点，为过程级模型重构提供了程序模型的表示方法。

程序实体是程序模型中各种各样的元素（如类、属性、方法、接口等），是程序分析和重构操作要处理的对象。过程蓝

*)北京市自然科学基金（编号：4073033）；广东省自然科学基金项目（编号：032027）；北京信息科技大学科学研究基金。**刘建宾**　博士、教授、硕士生导师，主要从事软件方法工具、软件工程、管理信息系统的研究工作；**杨林邦**　硕士研究生，主要研究领域为软件方法、CASE 工具、面向对象技术等。

图重构是一种过程级的程序模型重构，需要用到的程序实体包括：类(Class)、方法(Method)、方法调用(MethodInvocation)、过程蓝图(T)、抽象概念结构图[9](Tc)、抽象逻辑结构图[9](Tl)、抽象实现结构图[9](Ti)、结点(Node)、结点类型(NType)、语句块(Statement)、表达式(Expression)、变量(Variable)、常量(Constant)和参数(Parameter)。

### 2.2 分析函数

分析函数是用来分析程序信息的函数。重构操作需要在程序分析的基础上完成。因此，分析函数成为定义和执行重构操作的基础。

分析函数可分为原始分析函数和导出分析函数两种。原始分析函数是分析程序用到的粒度最小的和功能最基本的分析函数。导出分析函数是在原始分析函数基础上通过叠加调用而导出的更复杂和功能更强的分析函数。

在 Opdyke 和 Roberts 给出的分析函数[10,11]基础上，针对过程蓝图的特点并结合过程蓝图重构的实际需要，重新定义了过程蓝图重构操作需要的原始分析函数和导出分析函数(见表 1)。在表 1 中，编号为 9—27 的原始分析函数，以及编号为 6～10 的导出分析函数是针对过程蓝图的特点和重构需要而新增加的分析函数。

表 1 分析函数列表

| 原始分析函数 | |
|---|---|
| 函数表达式 | 函数说明 |
| 1、Boolean Is (*Class* (*Class c*) | 如果存在类 *c*，则为 *True*；否则为 *False*。 |
| 2、Body Method (*Class c*, *Method m*) | 返回类 *c* 中方法 *m* 的方法体。 |
| 3、Set of Method Senders (*Class c*, *Method m*) | 返回类 *c* 中的方法 *m* 的所有调用节点的集合。 |
| 4、Set of Class And Method References ToInstance Variable (*Class c*, *Variable varName*) | 返回引用类 *c* 的实例变量 *varName* 的所有类和所有方法对的集合。 |
| 5、Set of Class And Method Class Referencese (*Class c*) | 返回直接引用类 *c* 的所有类和所有方法对的集合。 |
| 6、Boolean Semantically EquivalentP (*Statement s*, *MethodInvocation mi*) | 如果语句块 *s* 和函数 *m* 的调用 *mi* 语义等价，则返回值为 *True*；否则返回 *False*。 |
| 7、Class Superclass (*Class c*) | 返回在继承层次中类 *c* 的直接超类。 |
| 8、Set of Variable Instance Variables DefinesBy (*Class c*) | 返回类 *c* 定义的实例变量集合。 |
| 9、Boolean Is Global Variable (*Class c*, *Variable varName*) | 如果类 *c* 中的变量 *varName* 是全局变量，则返回值为 *True*；否则返回 *False*。 |
| 10、Set of Variable Temp Variables DefinesBy (*T t*) | 返回过程蓝图 *t* 定义的临时变量集合。 |
| 11、Set of Expression Expression Assign To (*Variable varName*) | 返回变量 *varName* 的赋值表达式集合。 |
| 12、Set of Variable Unbound Variables (*Class c*, *Method m*, *T t*) | 返回类 *c* 中的过程蓝图 *t* 使用的非临时变量集合。 |
| 13、Boolean Semantically MassP (*Statement s*) | 如果语句块 *s* 中有一组为完成某种特定功能而逻辑上联系紧密的一组语句，则返回值为 *True*；否则返回 *False*。 |
| 14、Boolean Has Goto (*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 中的过程蓝图 *t* 的结点 *n* 存在有 *goto* 语句，则返回值为 *True*；否则返回 *False*。 |
| 15、Boolean Has Lable (*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 中的过程蓝图 *t* 的结点 *n* 存在有 *lable* 语句，则返回值为 *True*；否则返回 *False*。 |
| 16、Boolean Before Goto (*Class c*, *Method m*, *T t*, *Node lable_n*, *Node goto_n*) | 如果类 *c* 中的过程蓝图 *t* 的标记结点 *lable_n* 在 *goto* 结点 *goto_n* 之前，则返回值为 *True*；否则返回 *False*。 |
| 17、Set of Method References To Temp Variable (*Class c*, *Variable varName*) | 返回类 c 中引用临时变量 varName 的方法集合。 |
| 18、Boolean Is Empty (*Class c*, *Method m*, *T t*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 为空，则返回值为 *True*；否则返回 *False*。 |
| 19、Node Root (*Class c*, *Method m*, *T t*) | 返回类 *c* 的方法 *m* 的过程蓝图 *t* 的根结点。 |
| 20、Expression Value (*Class c*, *Method m*, *T t*, *Node n*) | 返回类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的内容。 |
| 21、NType Type(*Class c*, *Method m*, *T t*, *Node n*) | 返回类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的类型。 |
| 22、Node Parent(*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 是非根结点，则返回它的双亲结点，否则返回 *null*。 |
| 23、Node First Child (*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 是非叶子结点，则返回它的第一个孩子结点，否则返回 *null*。 |
| 24、Node Last Child (*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 是非叶子结点，则返回它的最后一个孩子结点，否则返回 *null*。 |
| 25、Node First Elder Brother (*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 存在兄长结点，则返回它的最近兄长结点，否则返回 *null*。 |
| 26、Node First Brother(*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 存在兄弟结点，则返回它的最近兄弟结点，否则返回 *null*。 |
| 27、T Tree of(*Class c*, *Method m*, *T t*, *Expression exp*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 存在表达式 *exp*，则返回过程蓝图 *t*，否则返回 *null*。 |
| 导出分析函数 | |
| 1、Body Look Up Method (*Class c*, *Method m*) | 如果方法 *m* 被类 *c* 引用，则返回 *m* 的方法体，否则返回 *null*。 |
| 2、Boolean Hierarchy ReferencesInst Var (*Class c*, *Instance Variable varName*) | 如果类 *c* 的任一子类或者超类引用了实例变量 *varName*，则返回值为 *True*；否则返回 *False*。 |
| 3、Set of Class Superclass * (*Class c*) | 返回在继承层次中类 *c* 的所有超类的集合。 |
| 4、Boolean DefinesInstance Variable (*Class c*, *Variable varName*) | 如果类 *c* 定义了实例变量 *varName*，则返回值为 *True*；否则返回 *False*。 |

| | |
|---|---|
| 5、SetOfClass ClassContaining (*Method m*) | 返回定义方法 *m* 的类集合。 |
| 6、Boolean Defines Temp Variable (*Class c*, *Method m*, *T t*, *Variable varName*) | 如果类 *c* 中的过程蓝图 *t* 定义了临时变量 *varName*，则返回值为 *True*；否则返回 *False*。 |
| 7、Node Brother(*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 存在兄弟结点，则返回它的兄弟结点，否则返回 *null*。 |
| 8、Set of Node Child(*Class c*, *Method m*, *T t*, *Node n*) | 如果类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 是非叶子结点，则返回它的孩子结点集合，否则返回 *null*。 |
| 9、Int Child Num (*Class c*, *Method m*, *T t*, *Node n*) | 返回类 *c* 中的过程蓝图 *t* 的结点 *n* 的孩子结点个数。 |
| 10、Int Brother Num (*Class c*, *Method m*, *T t*, *Node n1*, *Node n2*) | 返回类 *c* 中的过程蓝图 *t* 的结点 *n1* 到结点 *n2* 之间的兄弟结点个数。 |

分析函数有两个作用：第一，用作一个函数，在重构的前后置条件定义中使用，计算进行重构所必须的前置条件。第二，在重构操作中对程序实体做一些处理，从中提取出某些信息。这两个作用是相辅相成的，实际上都是对程序实体的分析操作。

分析函数的重要特性是只对程序实体进行分析处理，返回一定的值，不改变程序实体本身的内容，因而没有副作用，具有良性性质。

### 2.3 基本树变换操作

重构是行为保持的程序变换，需要对程序实体做出改变。过程蓝图是表示程序过程的树形结构模型，仅通过分析函数在过程蓝图模型中提取信息是不够的，还需要定义一些基本的支持重构的树变换操作，这些操作就是基本树变换操作，通过这些基本树变换操作来实现重构操作。因此，定义基本树变换操作是实现重构操作的基础。表 2 给出定义过程蓝图重构操作需要用到的常用基本树变换操作列表。

表 2 基本树变换操作列表

| 基本树变换操作 | | |
|---|---|---|
| 树变换操作表达式 | 树变换操作说明 | 类型 |
| 1、T Init Tree (*Class c*, *Method m*, *T t*) | 在类 *c* 为方法 *m* 构造空过程蓝图 *t*。 | S |
| 2、Boolean Destroy Tree(*Class c*, *Method m*, *T t*) | 在类 *c* 为方法 *m* 销毁过程蓝图 *t*。 | S |
| 3、T Create Tree(*Class c*, *Method m*, *T t*, *Node n*) | 在类 *c* 为方法 *m* 构造一颗根结点为 *n* 的过程蓝图 *t*。 | S |
| 4、Node Create Node(*Class c*, *Method m*, *T t*, *Node n*, *NType type*, *Expression exp*) | 在类 *c* 的方法 *m* 的过程蓝图 *t* 中构造一个结点 *n*，结点类型为 *type*，内容为 *exp*。 | N |
| 5、Boolean Clear Tree(*Class c*, *Method m*, *T t*) | 在类 *c* 的方法 *m* 中将过程蓝图 *t* 清为空过程蓝图 | S |
| 6、void Change Type(*Class c*, *Method m*, *T t*, *Node n*, *NType newType*) | 改变类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的类型为 *newType*。 | N |
| 7、void Assign (*Class c*, *Method m*, *T t*, *Node n*, *Expression exp*) | 把类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 赋值为 *value* | N |
| 8、Void Insert Child(*Class c*, *Method m*, *T t*, *Node n*, *Int i* , *T t1*) | 插入非空过程蓝图 *t1* 为类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的第 *i* 个孩子。 | S |
| 9、Void Insert Elder Brother(*Class c*, *Method m*, *T t*, *Node n*, *T t1*) | 插入非空过程蓝图 *t1* 为类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的最近兄长。 | S |
| 10、Void Insert Brother (*Class c*, *Method m*, *T t*, *Node n*, *T t1*) | 插入非空过程蓝图 *t1* 为类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的最近兄弟。 | S |
| 11、Void Delete Child(*Class c*, *Method m*, *T t*, *Node n*, *Int i*) | 删除类 *c* 的方法 *m* 的过程蓝图 *t* 的结点 *n* 的第 *i* 个孩子 | S |
| 12、Boolean Traverse Tree (*Class c*, *Method m*, *T t*, *Method m1*) | 对类 *c* 的方法 *m* 的过程蓝图 *t* 的每个结点调用函数 *m1*，如果失败返回 *flase*。 | N |
| 13、Void Remove References(*Class c*, *Method m*, *T t*, *Variable varName*) | 删除类 *c* 的方法 *m* 的过程蓝图 *t* 的 *varName* 的引用。 | S |
| 14、Void Delete Node (*Class c*, *Method m*, *T t*, *Node n*) | 删除类 *c* 的方法 *m* 的过程蓝图 *t* 结点 *n*。 | S |
| 15、Void Rename References(*Class c*, *Method m*, *T t*, *Variable varName*, *Variable newName*) | 把类 *c* 的方法 *m* 的过程蓝图 *t* 中的 *varName* 的引用改名为 *newName* 的引用。 | N |

过程蓝图的基本树变换操可作分为结点操作和结构操作两种基本类型。在表 2 中，分别用 N 和 S 标识结点和结构操作类型。

### 2.4 组合操作方式

一般而言，操作的组合方式有顺序、循环和选择三种，过程蓝图的组合操作方式也可分为这三种基本方式。组合操作是过程蓝图基本树变换操作按照顺序、循环、选择三种方式任意组合而成的操作。过程蓝图的组合操作方式及其表达式定义如下：

**定义 1** 顺序组合方式是一个操作序列被依次执行的方式。给定树操作序列 Tr1，Tr2，…，Trn 和一组程序实体 X，顺序组合方式 STr::= +(X)〈Tr1，Tr2，…，Trn〉，其中＋表示顺序，STr 表示顺序组合操作名。

**定义 2** 循环组合方式是一个操作被迭代执行零次或多次的方式。给定树操作 TR1、一组程序实体 X 和循环条件 C，循环组合方式 ITr::＝＊(X，C)〈Tr1〉，其中＊表示循环，ITr 表示循环组合操作名。

**定义 3** 选择组合方式是在两个操作中根据条件选择其中一个操作被执行的方式。给定树操作 Tr1、Tr2、一组程序实体 X 和选择条件 C，选择组合方式 CTr::＝？(X，C)〈Tr1，Tr2〉，其中？表示选择，CTr 表示选择组合操作名。

### 2.5 不变量

不变量是整个重构变换过程中程序模型始终应该保持的不变的属性。表 3 给出过程蓝图模型重构需要保持的 10 个不变量。

表 3 不变量列表

| 不变量 | |
|---|---|
| 不变量表达式 | 不变量说明 |
| 1、Boolean Unique Superclass(*Class c*) | 唯一的超类。 |
| 2、Boolean Unique ClassName(*Class c*) | 唯一的类名称。 |
| 3、Boolean Unique Member Name(*Class c*, *Variable var Name*) | 唯一的成员名称。 |
| 4、Boolean RedefineInherited Variable (*Class c*, *Variable var Name*) | 被继承的成员变量不能被重新定义。 |
| 5、Boolean Compatible Signatures(*Class c*, *Method m1*, *Method m2*) | 成员函数重定义时保持函数签名一致性。 |
| 6、Boolean Type Safe(*Class c*, *Variable varName*, *Expression ass*) | 赋值时的类型安全。 |
| 7、Boolean Semantically Equivalent (*Statement s1*, *Statement s2*) | 引用和运算的语义等价。 |
| 8、Boolean Unique Root(*Class c*, *Method m*, *T t*, *Node root*) | 唯一的根结点。 |
| 9、Boolean Unique Parent (*Class c*, *Method m*, *T t*, *Node n*) | 唯一的父结点。 |
| 10、Boolean Unancestor(*Class c*, *Method m*, *T t*, *Node n*) | 任何结点都不是自己的祖先。 |

表 3 中前 7 个不变量由 Opdyke 提出，最后 3 个是针对过程蓝图的特点新定义的不变量。这 3 个新增的不变量保证重构过程中过程蓝图具有树结构的不变性质。表 3 的不变量在重构操作的始终必须同时为真，这样才能保持程序模型的基本属性。

## 3 重构操作的形式、类别与层次

### 3.1 重构形式

重构操作是行为保持的程序变换操作。行为保持指一个程序经过重构后不改变它的外部可观察行为[1]。根据 Opdyke 和 Roberts 提出的重构分类形式[10,11]，将基于过程蓝图的重构分为原子重构、组合重构和复杂重构三种类型。

**定义 4** 原子重构(Primitive Refactoring)PR 是由一序列的基本树变换操作按照顺序、循环、选择三种方式任意组合而成的不能再细分的行为保持操作。

**定义 5** 组合重构(Composite Refactoring)CpoR 是由多个原子重构按照顺序、循环、选择三种方式任意组合而成的行为保持操作。

**定义 6** 复杂重构(Complex Refactoring)CplR 是由原子重构、组合重构、基本树变换操作按照顺序、循环、选择三种方式任意组合而成的具有规模性的行为保持操作。

### 3.2 重构类别

依据过程蓝图的基本操作类型，可将过程蓝图的重构操作分为结点重构和结构重构两种。

**定义 7** 结点重构 (Node Refactoring)NR 是只改变过程蓝图结点类型或者结点内容，而不改变树结构的重构操作。

**定义 8** 结构重构(Structure Refactoring)SR 是改变过程蓝图树结构的重构操作。

### 3.3 重构层次

过程蓝图是具有概念、逻辑和实现三层外部视图和统一内部结构的程序可视化过程建模语言，是对过程源代码进行描述的模型[9]。由于过程蓝图使用抽象概念结构图[9]、抽象实现结构图[9]和抽象实现结构图[9]表示概念、逻辑和实现三个不同抽象层次的模型，所以可将过程蓝图模型重构划分成对应的三个层次，分别用 AIR、ALR 和 AIR 标识。

概念层重构 AIR 是作用于抽象概念结构图上的行为保持模型变换。由于抽象概念结构图是一种与实现语言无关的抽象模型表示，因此，概念层重构是与实现无关的高层重构。逻辑层重构 ALR 和实现层重构 AIR 是分别在抽象逻辑结构图和抽象实现结构图上实施的重构。由于这两种模型图使用与实现语言相关的控制构造和数据流表示，因此，逻辑层和实现层重构是与实现相关的中低层重构。由于过程蓝图具有统一的内部表示，所以每一层上的重构都会反映在内部的统一结构上并引起其他层表示的相应变化，由此引发的三层模型表示一致性问题由过程蓝图模型的一致性维护方法[12]得到解决。

由于过程蓝图使用三层外部视图表示并通过概念层到逻辑层的控制流映射和逻辑层到实现层的数据流映射提供一定程度的程序独立性(控制流独立性和数据流独立性)[9]，因此，重构的层次划分能够使重构操作的影响尽可能被局部化，从而提供了一定的重构独立性，即发生在实现层的结点重构操作不会引起概念层和逻辑层模型的变化；发生在逻辑层的部分结构操作不会引起概念层模型的改变。

## 4 重构操作的定义

### 4.1 重构形式定义框架

断言[10,11](前后置条件和不变量)和图变换[8]是重构形式化的二种主要方法。断言用于重构的行为保持方面，而图变换可以增强重构的直观性。由于过程蓝图本身是一种图形化的规格说明方法，因此对其执行的重构操作是一种行为保持的图变换。图变换由被变换的过程蓝图和基于树操作的变换规则来定义。为了可靠地执行图变换并确保图变换的行为保持性质，需要定义变换的前、后置条件并提供行为保持的论证过程。每个重构操作的名称，加上模型图、变换规则、变换的前后置条件和行为保持证明构成描述和定义过程蓝图重构操作的六要素。

**定义 9** 一个过程蓝图重构操作 pbr 是如下定义的五元组：

pbr::=(pre,pb,post, rules,bp)

其中 pbr 为重构操作名称，pre 为重构前所必须满足的前置条件，pb 为过程蓝图表示的模型，post 为重构后所必须满足的后置条件，rules 是用树变换操作表达的图变换规则，bp 为行为保持的半形式化证明。

该定义集成了断言和图变换这两种主要的形式化方法，是一种基于一阶谓词逻辑的形式化定义框架。

#### 4.1.1 前后置条件

对于重构来说，行为保持是一个重要的方面。Opdyke 提出用前置条件来保证重构的行为保持[11]，Roberts 在 Opdyke 的基础上增加了后置条件，提出使用前后置条件相结合的方法来达到重构的行为保持的目的[10]。在 Opdyke 和 Roberts 的研究工作基础上，使用基于过程蓝图的分析函数和一阶谓词逻辑来定义过程蓝图重构操作的前后置条件。下面给出前置条件和后置条件的定义。

**定义 10** 一个重构 pbr 的前置条件 pre(pbr)是由一阶谓词逻辑和分析函数组合而成的合法条件表达式。

**定义 11** 一个重构 pbr 的后置条件 post(pbr)是一组描述重构后应该满足的合法条件表达式，用 $f'=f[x/y]$ 表示。其中 $f[x]$ 是分析函数，$f'$ 表示重构后的分析函数，$f[x/y]$ 表

示重构把分析函数 $f[x]$的结果改变为 $y$，用来描述重构对程序的影响。

此外，在后置条件中用“⊥”表示未定义的值。例如，如果一个变换把类 $c$ 的方法 $m$ 的方法体移去了，那么在它的后置条件中，为了表示 $m$ 的方法体不属于任何方法，分析函数 $Method(c,m)$就需要更新，即 $Method'=Method[(c,m)/\perp]$。

4.1.2 变换图及变换规则

定义 9 中的 pb 是欲施加重构变换的过程蓝图。它使重构操作发生在图形接口上，并直观形象地表示出重构操作发生前后的模型变化情况，并最终通过过程蓝图规格说明到源代码的变换，将图形上做的重构改变自动反映到最终程序代码上。

变换规则 rules 是由过程蓝图基本树变换操作按照顺序、循环、选择三种方式任意组合而成的合法操作表达式。计算机可以通过解释 rules 的表达式来实现重构操作执行的自动化。

4.1.3 行为保持证明

重构的行为保持至今未被严格定义，完全形式化的行为保持证明因而难以实现。因此，使用 Opdyke 提出的半形式化行为保持论证方法[11]来构造过程蓝图重构操作的行为保持证明较为可行。这种基于一阶谓词逻辑的半形式化证明方法是通过前后置条件、不变量和自然语言来论证重构变换对程序可观察行为的影响，从而达到证明重构的行为保持的目的，但难以实现证明的自动化并存在一定的局限性[13]。

原子重构和复杂重构的行为保持从以下三个步骤来保证：前置条件计算，程序修改和后置条件计算。它们顺序进行，任何一个出现错误都中止，并重新回到初试状态，从而通过前置条件和后置条件的约束保证了重构的行为保持。

Opdyke 和 Robert 已经证明：组合重构的行为保持可通过原子重构的行为保持来保证[10,11]。因此，组合重构不需要给出重构定义的 bp 部分，而只需要给出原子重构和复杂重构的行为保持证明。

### 4.2 重构操作规格说明

在 Martin Fowler 整理的 70 多种程序源码的重构[1]基础上，在过程蓝图语境下提取和构造了过程级模型重构操作（见表 4），并使用定义 9 中的形式框架对每个重构操作进行了严格定义。表 4 给出过程蓝图模型重构操作的规格说明和分类。其中编号为 24 和 25 的操作为语义聚集和语义消解操作，他们是在过程蓝图上新提出的二个重要且实用的高层抽象复杂重构操作，与实现语言无关，对程序理解和维护具有积极意义。

表 4 重构操作规格说明和分类

| 重构操作 | | | | |
|---|---|---|---|---|
| 重构操作表达式 | 重构操作说明 | 形式 | 类型 | 层次 |
| 1、Add Temp Variable (*Class c, Method m, Tit, Node n, Variable var Name*) | 在类 c 的方法 m 的过程蓝图 t 中，增加一个没有被引用的临时变量 varName，在过程蓝图中对应结点 n。 | PR/TR | NR/SR | AIR |
| 2、Remove Temp Variable (*Class c, Method m, Tit, Node n, Variable var Name*) | 在类 c 的方法 m 的过程蓝图 t 中，删除一个没有被引用的临时变量 varName，在过程蓝图中对应结点 n。 | PR/TR | NR/SR | AIR |
| 3、Rename Temp Variable (*Class c, Method m, Tit, Node n, Variable var Name, Variable new Name*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个临时变量 varName 改名为 newName。 | PR/TR | NR | AIR |
| 4、Replace Expression With Variable (*Class c, Method m, Tit, Expression exp, Variable var Name*) | 在类 c 的方法 m 的过程蓝图 t 中，一个临时变量 varName 来替换表达式 exp。 | PR/TR | NR | AIR |
| 5、Inline Variable (*Class c, Method m, Tit, Variable varName, Expression ass*) | 在类 c 的方法 m 的过程蓝图 t 中，把临时变量 varName 的赋值表达式 ass 来替换临时变量 varName 的引用。 | PR/TR | NR | AIR |
| 6、Introduce Explaining Variable (*Class c, Method m, Tit, Expression exp, Variable var Name*) | 在类 c 的方法 m 的过程蓝图 t 中，引入临时变量 varName 并赋值为表达式 exp 的结果，然后替换 exp 的引用。 | CpoR | NR/SR | AIR |
| 7、Replace Magic Number With Constant (*Class c, Method m, Tit, Expression exp, Constant con Name*) | 在类 c 的方法 m 的过程蓝图 t 中，一个常量 conName 来替换表达式 exp。 | CpoR | NR/SR | AIR |
| 8、Replace Temp With Query (*Class c, Method m, Tit, Variable var Name, Method m2*) | 在类 c 的方法 m 的过程蓝图 t 中，将变量 varName 的赋值表达式提炼到一独立函数 m2，将这个变量的被引用点替换为 m2 调用。 | CpoR | NR/SR | AIR |
| 9、Replace Statement With Invocation (*Class c, Method m, Tlt, Node n, Statement s, Method Invocation mi*) | 在类 c 的方法 m 的过程蓝图 t 中，将语句块 s 替换为一个方法调用 mi。 | PR/TR | NR/SR | ALR |
| 10、Inline Method (*Class c, Method m, Tlt, Method Invocation mi, Tlti*) | 在类 c 的方法 m 的过程蓝图 t 中，将方法调用 mi 替换为这个方法的方法体 ti。 | PR/TR | NR/SR | ALR |
| 11、Extract Method (*Class c, Method m, Tlt, Statement s, Method m1*) | 在类 c 的方法 m 的过程蓝图 t 中，将语句块 s 抽取为一个方法 m1。 | CpoR | NR/SR | ALR |
| 12、Decompose Conditional Expression (*Class c, Method m, Tlt, Node n, Method m1, Method m2, Method m3*) | 在类 c 的方法 m 的过程蓝图 t 中，if 条件式的结点为 n，将 if、then、else 段落分别抽取为方法 m1，m2，m3 并将他们替换为 m1，m2，m3 的调用。 | CplR | NR/SR | ALR |

| | | | | |
|---|---|---|---|---|
| 13、Consolidate Conditional Expression (*Class c*, *Method m*, *Tl t*, *Node n1*, *Node n2*, *Node n*) | 在类 c 的方法 m 的过程蓝图 t 中，把两个 if 条件式 n1 和 n2 合并为一个 if 条件式 n。 | CplR/TR | NR/SR | ALR |
| 14、Consolidate Duplication Fragments (*Class c*, *Method m*, *Tl t*, *Node n*, *Statement clone*, *Node n1*) | 在类 c 的方法 m 的过程蓝图 t 中，把 if 条件式 n 中的重复片段 clone 提炼出 if 前面或后面，形成结点 n1。 | CplR/TR | NR/SR | ALR |
| 15、Replace Nested Conditional With Guard Claues (*Class c*, *Method m*, *Tl t*, *Node n*, *Node n1*, *Node n2*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个嵌套 if 条件式 n 用卫式语句 n1 和 n2 替代。 | CplR/TR | NR/SR | ALR |
| 16、Remove PreGoto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，移除一个前置 goto 语句 goto_n。 | PR/TR | NR/SR | ALR |
| 17、Remove Post Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，移除一个后置 goto 语句 goto_n。 | PR/TR | NR/SR | ALR |
| 18、 | | | | |
| 19、Move Out Switch Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个 goto 语句 goto_n 移出 Switch 语句。 | CplR/TR | NR/SR | ALR |
| 20、Move Out If Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个 goto 语句 goto_n 移出 If 语句。 | CplR/TR | NR/SR | ALR |
| 21、Move In While Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个 goto 语句 goto_n 移入 While 语句。 | CplR/TR | NR/SR | ALR |
| 22、Move In If Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个 goto 语句 goto_n 移入 If 语句。 | CplR/TR | NR/SR | ALR |
| 23、Move In Swith Goto (*Class c*, *Method m*, *Tl t*, *Node goto_n*, *Node lable_n*) | 在类 c 的方法 m 的过程蓝图 t 中，把一个 goto 语句 goto_n 移入 Switch 语句。 | CplR/TR | NR/SR | ALR |
| 24、Consolidate Sematics (*Class c*, *Method m*, *Tc t*, *Node n1*, *Node n2*, *Node n*) | 在类 c 的方法 m 的过程蓝图 t 中，将一组为完成某种特定功能而逻辑上联系紧密的一组兄弟结点 n1 和 n2 提取成一个新的顺序结点 n，使得这组兄弟结点成为新结点的子结点，并用新结点取代这组兄弟结点在树中的原有的位置。 | CplR/TR | NR/SR | ACR |
| 25、Decompose Sematics (*Class c*, *Method m*, *Tc t*, *Node n*, *Node n1*, *Node n2*) | 在类 c 的方法 m 的过程蓝图 t 中，将结构树中没有必要存在的顺序结点 n 删除，把删除结点的所有子结点 n1，n2 上提一级并取代位置。 | CplR/TR | NR/SR | ACR |
| 26、Add Body (*Class c*, *Method m*, *Tc t*) | 在类 c 的方法 m 中，增加过程蓝图 t。 | PR/TR | NR/SR | ACR |
| 27、Remove Body (*Class c*, *Method m*, *Tc t*) | 在类 c 的方法 m 中，移除过程蓝图 t。 | PR/TR | NR/SR | ACR |
| 28、Substitute Algorithm (*Class c*, *Method m*, *Tt1*, *Tt2*) | 在类 c 的方法 m 的过程蓝图 t1 中，替换方法体为 t2。 | CpoR/TR | NR/SR | ACR |

**结束语** 基于文本形式的源码程序重构存在处理过程复杂、效率低的缺点。本文提出基于过程蓝图的模型重构，使过程级的重构操作发生在抽象语法树外部表示的图形接口上，避免了程序源代码的语法分析，简化了重构过程，并提高重构处理的效率，是在源码重构研究基础上提出的一种比源码重构更为理想的程序重构形式。

本文在提出分析函数、基本树变换操作、组合操作方式和重构不变量的基础上，对过程蓝图重构操作的形式、类别与层次进行了划分，给出过程蓝图模型重构操作定义的形式框架及其操作规格说明。

基于过程蓝图的重构是一种适用于程序过程级的模型重构。这种新的重构形式与方法支持重构程序的可视化表示及其行为保持的图变换，将为研制带有重构功能的可视化程序模型编辑器奠定基础。

## 参 考 文 献


1 Fowler M. Refactoring: Improving the Design of Existing Programs. Addison-Wesley, 1999

2 Miller J, Mukerji J. Model Driven Architecture (MDA). Object Management Group, Draft Specification ormsc/2001-07-01 [EB/OL]. http://www.omg.org/MDA/, July 9 2001

3 Astels D. Refactoring with UML. In: Proc. 3rd Int'l Conf eXtreme Programming and Flexible Processes in Software Engineering, 2002. 67～70

4 Boger M, Sturm T, Fragemann P. Refactoring browser for UML. In: Proc 3rd Int'l Conf on eXtreme Programming and Flexible Processes in Software Engineering, 2002. 77～81

5 Bottoni P, Parisi-Presicce F, Taentzer G. Coordinated distributed diagram transformation for software evolution. Electronic Notes in Theoretical Computer Science, 2002, 72(4)

6 Suny'e G, Pollet D, LeTraon Y, et al. Refactoring UML models. In: Proc UML 2001, vol 2185 of Lecture Notes in Computer Science. Springer-Verlag, 2001. 134～138

7 Griswold W G. Program Restructuring as an Aid to Software Maintenance: [PhD thesis]. University of Washington, August 1991

8 Bowdidge R W. Supporting the Restructuring of Data Abstractions through Manipulation of a Program Visualization: [PhD thesis]. University of California, San Diego, Department of Computer Science and Engineering, 1995

9 刘建宾. 过程蓝图设计方法学. 北京：科学出版社，2005

10 Roberts D. Practical Analysis for Refactoring: [PhD thesis]. Univ. of Illinois at Urbana-Champaign, 1999

11 Opdyke W F. Refactoring Object-Oriented Frameworks: [Ph D thesis]. University of Illinois, 1992

12 刘建宾，郝克刚. 抽象概念结构图到 JAVA 过程蓝图的平滑过渡及一致性. 计算机科学，2001, 28(8)：89～93

13 Mens T. A Survey of Software Refactoring. IEEE Transactions on Software Engineering, 2004, 30(2)