计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190100046

设计模式组合操作优化研究

纪程宇　朱雪峰
中国石油大学(北京)石油数据挖掘北京市重点实验室　北京 102249
中国石油大学(北京)信息科学与工程学院　北京 102249
(912545050@qq.com)

摘　要　作为软件设计经验的总结,恰当使用设计模式能够有效提高软件系统的可复用性,确保最终所得软件产品的质量。但在实际应用中,人们很少使用单一的设计模式,通常需要根据实际的应用场景进行多个模式的组合,这可能会导致所得结果不确定,严重影响软件产品的质量。虽然现有的模式组合形式化方法能够有效地表达模式组合后的结果,但是组合方法逻辑复杂并包含大量的冗余操作,设计人员很难熟练使用。针对上述模式组合过程中存在的问题,文中对多模式之间的组合关系进行了深入探讨,从设计模式的形式化描述出发,结合Z语言的特点对现有的模式组合形式化方法进行了深入研究,并对现有的模式组合操作符进行了初步优化;在现有操作符集合的基础上提出了基于模式的约束、叠加和扩展操作符,通过操作符定义了模式组合的精确语义,并采用代数推理过程验证了优化后的方法可以有效地替代现有的模式组合形式化方法,且能够解决现有模式组合形式化方法中操作符冗余、数量过多导致的效率低等问题。最后,通过模式组合案例的研究,验证了所提方法的有效性。

关键词:设计模式;模式组合;操作优化;形式化方法;等式推理
中图法分类号　TP311.5

Study on Optimization of Design Pattern Combination Operation

JI Cheng-yu and ZHU Xue-feng
Beijing Key Laboratory of Petroleum Data Mining,China University of Petroleum,Beijing 102249,China
School of Information Science and Engineering,China University of Petroleum,Beijing 102249,China

Abstract　As a summary of software design experience,proper use of design patterns can effectively improve the reusability of software systems and ensure the quality of the final software products. However,in practical applications,people rarely use a single design pattern,and usually software designers need to use experience to combine multiple patterns according to the actual application scenarios,which may lead to uncertainty results and seriously affect the quality of software products. Although the existing formal method of pattern combination can effectively express the result of pattern combination,the combination method has complex logic and contains a large number of redundant operations,which is difficult for designers to be familiar with and adopt. Aiming at the problems existing in the above pattern combination process,this paper deeply discussed the combination relationship between multiple patterns. Starting from the formal representation of design patterns,combined with the characteristics of Z language,this paper studied the existing formal methods of pattern combination in depth,and optimized the existing pattern combination operators. Based on the existing set of operators,the constraint,superposition and extension operators are proposed,the exact semantics of pattern composition are defined by the operators,and the algebraic reasoning process is used to verify that the optimized method can effectively replace the existing formal method of pattern combination,it can overcome the problems of redundant operators and low efficiency caused by too many operators in the existing formal methods of pattern combination. Finally,the effectiveness of the proposed method is verified by a case study of pattern combination.

Keywords　Design patterns,Combination of patterns,Operation optimization,Formal method,Equality reasoning

1　引言

模式可重用的设计方案在软件系统的开发中发挥着极其重要的作用。目前,许多设计模式被总结出来[1],得到了众多软件识别工具的支持[2],并以IDE插件的形式在代码级别[3]和模型层面[4]进行模式的识别。

到稿日期:2019-01-07　返修日期:2019-03-22　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(60496324);中国石油大学科研基金(KYJJ2012-05-17)
This work was supported by the National Natural Science Foundation (60496324) and China University of Petroleum Research Fund (KYJJ2012-05-17).
通信作者:朱雪峰(xuefeng.zhu@cup.edu.cn)

虽然设计模式都是单独定义的，每种模式都有自身独特的应用场景，但是单一的模式通常无法满足复杂的应用需求，通常需要将模式进行组合来达到设计要求。目前，模式组合还停留在依靠经验使用的非形式化阶段，这会导致很多误差。同时，模式的组合使用导致开源代码的识别遇到了困难。因此，需要形式化定义模式之间的关系，从而精确地描述多模式的组合语义。

尽管前人已经提出了许多模式的形式化方法，但是很少有作者对模式组合进行正式的研究。Dong 等[5-7]首次在文献中研究了模式组合形式化。在相关文献中，两个模式的组合被定义为一对名称映射。每个映射将模式中声明的类和对象与相组合的模式中声明的类和对象关联。Dong 等通过使用 Iterator 和 Composite 模式组合来进行说明[5]。文献[6]以图表的形式呈现了模式组合结构，并证明了模式实例及其组成结构和行为属性在正式规范中的推导过程。文献[7]将模式实例化定义为从模式中各种元素的名称到实例中类、属性、方法等的映射，模式组合被定义为从多个模式到结果模式的映射。

Taibi 等[8-10]采用了与 Dong 等非常相似的方法，但不是为模式组合和实例化定义映射，而是直接使用重命名替换表示模式中的变量。模式组合使用变量替换变量，而实例化使用常量替换变量，文献[8-9]运用模式组合实例进行了具体说明。此种方法实例化和模式组合运用了相同的表示方法，优于 Dong 等的方法。除此之外，它们在数学上是等价的，替换就是限制的术语映射。替换和映射必须保留变量类型在语法上的有效性，这两种方法都不能表达一对多或多对多的关系。

Zhu 等[11]正式定义了一个模式组合操作符来表示设计模式之间的关系。这个操作符虽然完整且普遍适用，但性质很复杂，在实际应用中并不是很灵活。文献[12]修改了这个方案，采取了截然不同的方法，即不再是定义一个单一的通用操作符，而是提出了一组更原始的操作符，这些操作符可以精确地表达每一种模式组件之间的内部关系；同时，区分了 3 种不同类型的重叠，即一对一、一对多和多对多[13]。文献[14-15]总结了关于操作符的代数规则以及定理，为操作符提供了更加可靠的理论依据。文献[16]提出了模式组合保持有效性的条件，即需要同时满足特征保存、语义保存和健全性保存，这为模式组合是否可行提供了验证依据。

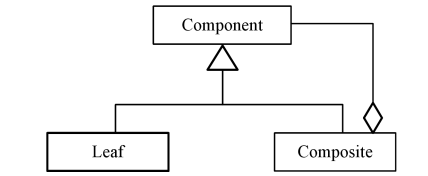
Dong 等和 Taibi 等提出的方法在数学上等同于 Zhu 等提出的限制操作符，可以表示为最简单的形式：$u=v$。因此 Zhu 等的方法更具有表达性。但是，通过分析发现，该方法的操作符之间存在若干重叠关系，使得组合操作逻辑复杂、缺乏灵活性，因此需要进一步优化操作符。针对上述问题，本文通过分析验证设计模式之间的内部关系来去除重叠，使用优化后的操作符来表达模式之间的关系，使模式组合更为简单可靠。优化后的操作符都是基于 Z 语言的，为之后利用操作符识别和验证模式组合是否有效提供了基础。

本文第 2 节引入模式组合的形式化概念，并对当前已有的研究做出分析；第 3 节详细描述优化后的模式操作符并进行验证；第 4 节通过实际案例介绍本文提出的操作符描述多模式组合的过程，并用操作符表示 GoF 所有的模式组合情况；最后总结全文，并对未来的研究工作进行展望。

2 相关工作

近年来，研究人员已经提出了多种设计模式形式化的方法。尽管这些形式化方法有所不同，但其基本思想是非常相似的。有效的模式实例通常是使用限制其结构特征的语句来规定的，有时也限制其行为特征。结构特征通常用于断言某些类型的组件是否存在以及该模式具有的静态配置，例如 UML 类图。另一方面，行为约束详细说明了组件之间交换信息的时间顺序，例如 UML 时序图。同时，禁止信息也可以包含在模式规范中，例如声明禁止条件、两个特定组件之间不允许有关联等。这种禁止条件可能会对之后验证多模式是否可以正确组合有帮助。

例如，图 1 给出了组合模式的形式化定义，同时将 GoF 书中的类图展示出来作为对比，以增强可读性。谓词和函数都是基于 UML 类图和序列图的原始谓词，由此保证了模式表示的可靠性。



Specification: Composite Pattern

Components

1) Components, Composite ∈ classes

2) Leaves ⊆ classes

Conditions

1) ∀ l ∈ Leaves • (l ⟶▷ Component ∧ ¬(l ◇⟶ Component))

2) isInterface(Component)

3) Composite ⟶▷ Component

4) Composite ◇⟶ Component

图 1 组合模式的规范

Fig. 1 Specification of composite pattern

一般来说，设计模式 P 可以被抽象[17]地定义为一个有序对 $\langle V, Pr\rangle$，其中 Pr 是软件系统表示域上的谓词，V 是谓词 Pr 中的一组可用变量的声明。简而言之，Pr 指定了模式的结构特征和行为特征，V 指定了模式的组成部分。假设 $V=\{v_1:T_1,\cdots,v_n:T_n\}$，其每个 v_i 表示模式中组件的变量，T_i 是该变量的对应类型。根据以上说明，规范的模式语义可以表示为以下形式：

$$\exists v_1:T_1,\cdots,\exists v_n:T_n \bullet (Pr) \tag{1}$$

规定用 $Spec(P)$ 来表示上面的谓词 (1)，其中 $Vars(P)$ 表示在 V 中声明的变量集合，$Pred(P)$ 表示谓词 Pr。

通过分析目前流行的模式形式化方法发现，目前流行的多模式组合操作符数量过多，具有重叠关系。因此，本文从模式之间的本质关系出发，结合模式的抽象定义，精确地定义模式之间的内部关系，并且通过理论推导验证了本文提出的操作符的有效性，从而解决了操作符之间存在重叠关系、个数过多等问题。

3 基于模式的操作符方法

本文通过总结已有工作[5-17]存在的问题，基于模式规范的形式化逻辑思想，深入研究模式之间的关系，同时限制模式的抽象定义，结合 Z 语言的关系操作符，正式定义了基于多模式的操作符，从而有效地表示模式之间的关系。此组操作符更简洁易懂，且具有很强的表达性。

3.1 限制操作符

设 P 是给定的任意模式，c 是 P 的组件上的谓词。用约束 c 来限制 P，可以表示为 $P \triangleright c$。模式 $P \triangleright c$ 是通过将谓词 c 作为模式 P 的附加条件而获得的新模式。从形式上来看：

1) $Vars(P \triangleright c) = Vars(P)$

2) $Pred(P \triangleright c) = Pred(P) \wedge c$

例如，存在一种 Composite 模式，其中只有一个 Leaf 组件，将其表示为 $Composite_1$，那么其就可以通过限制操作符正式定义为：

$$Composite_1 = Composite \triangleright (Leaves = 1)$$

在案例研究中经常会使用限制关系，特别是对于模式 P 及相同类型的变量 u 和 v，可以表示为 $P \triangleright (u = v)$。这个表达式就是将模式 P 通过统一变量 u 和 v，使 u 和 v 成为相同的元素，从而获得新的模式。

限制运算符不会在模式结构中引入任何新的额外组件或者模式，接下来介绍的运算符则可以引入新的组件。

3.2 叠加操作符

设 P 和 Q 是两种模式，且它们的组成变量是不相交的，即 $Vars(P) \cap Vars(Q) = \Phi$。$P$ 和 Q 的叠加可以表示为 $P;Q$，其含义为模式 P 和模式 Q 的组合使用，形式定义如下：

1) $Vars(P;Q) = Vars(P) \cup Vars(Q)$

2) $Pred(P;Q) = Pred(P) \wedge Pred(Q)$

例如，Composite 模式和 Adapter 模式的叠加可以表示为 $Composite;Adapter$，并且要求每个实例一部分满足 Composite 模式，另一部分满足 Adapter 模式。这些部分可能重叠，也可能不重叠，但下面的表达式将强制重叠，要求 Leaves 中的类是 Adapter 模式的 Target 组件。

$$(Composite;Adapter) \triangleright (Target \in Leaves)$$

当然，$Vars(P)$ 和 $Vars(Q)$ 不相交的要求很容易通过重命名来实现。限制操作符和叠加操作符经常结合使用，以限制模式之间的变量。

3.3 扩展操作符

假设 P 是一个模式，$Vars(P) = \{x_1 : T_1, \cdots, x_n : T_n\}$，$n > 0$，并且 $Pred(P) = p(x_1, \cdots, x_n)$。令 $X = \{x_1, \cdots, x_k\}$ $(1 \leqslant k < n)$ 是模式中变量的一个子集。以 X 为键的模式 P 的扩展记为 $P \uparrow X$，定义如下：

1) $Vars(P \uparrow X) = \{xs_1 : PT_1, \cdots, xs_n : PT_n\}$

2) $Pred(P \uparrow X) = \forall x_1 \in xs_1 \cdots \forall x_k \in xs_k \cdot \exists x_{k+1} \in xs_{k+1} \cdots \exists x_n \in xs_n \cdot p(x_1 \cdots x_n)$

在键是单例的情况下，为了简单起见，省略了设置的括号，表示为 $P \uparrow X$，而不是 $P \uparrow \{X\}$。非正式地，扩展模式 P 将会导致模式 P' 包含多个 P 模式的实例。例如，$Adapter \uparrow Target$ 是包含许多个 Target 类的模式，其中每个 Target 都有一个独立的 Adapter 和 Adaptee 类。换句话说，模式 $Adapter \uparrow Target$ 将由多个 Adapter 模式组成，该模式中的组件 Target 起着与关系数据库中的主键类似的作用。

3.4 操作符的验证

文献[12-16]已详细介绍了前人提出的一组操作符语义以及模式组合操作符的代数法则，本文不再详细介绍。本节通过推理验证本文提出的操作符与前人提出的 6 个操作符的等价性，可以完整地描述前人提出的 6 个操作符语义。在逻辑系统中，谓词个数决定了逻辑系统的复杂度，同时减少操作符也可以提高定理证明器的效率。

第 3.1 节在介绍限制操作符时已给出了 Composite 模式和 $Composite_1$ 模式的定义，接下来将通过这个例子进行验证。

使用 Zhu 等提出的操作符[11]可以得出如下表示方法：

$$Composite_1 = Composite[Leaves = 1] \tag{2}$$

$$Composite_1 = Composite \Downarrow Leaves \backslash Leaf \tag{3}$$

$$Composite \approx Composite_1 \Uparrow Leaf \backslash Leaves \tag{4}$$

使用本文优化后的 3 个操作符的表示如下：

$$Composite_1 = Composite \triangleright (Leaves = 1) \tag{5}$$

$$Composite \approx Composite_1 \triangleright (Leaves = n) \tag{6}$$

首先证明表示 $Composite_1$ 模式的式(2)和式(3)这两个定义是等价的，即证明下面的等式成立：

$$Composite[Leaves = 1] \approx Composite \Downarrow Leaves \backslash Leaf$$

通过操作符的代数法则证明如下：

$$\begin{aligned} &Composite \Downarrow Leaves \backslash Leaf \\ &\quad \approx Composite \sharp \{(Leaf : class)\} \cdot (Leaves = \{Leaf\}) \\ &\quad \approx Composite[\exists Leaf : class \cdot (Leaves = \{Leaf\})] \\ &\quad \approx Composite[Leaves = 1] \end{aligned}$$

如果将方程(3)代入方程(4)，可以得到以下方程：

$$\begin{aligned} Composite &\approx (Composite \Downarrow Leaves \backslash Leaf) \\ &\quad \Uparrow Leaf \backslash Leaves \\ &= Composite \end{aligned}$$

类似地，将 Composite 的定义(4)代入方程(3)，可以得到以下方程：

$$\begin{aligned} Composite_1 &\approx (Composite_1 \Uparrow Leaf \backslash Leaves) \\ &\quad \Downarrow Leaves \backslash Leaf \\ &= Composite_1 \end{aligned}$$

通过式(2)－式(4)的定理证明可以发现，Zhu 等提出的展平操作符 $\Downarrow$ 和泛化操作符 $\Uparrow$ 实际就是对模式中组件个数的限制，而且它们还是互逆操作，因此可以直接用限制操作符 $\triangleright$ 来替代。

$$\begin{aligned} Composite_1 &= Composite[Leaves = 1] \\ &= Composite \Downarrow Leaves \backslash Leaf \\ &= Composite \triangleright Leaves = 1 \end{aligned}$$

$$\begin{aligned} Composite &\approx Composite_1 \Uparrow Leaf \backslash Leaves \\ &= (Composite \triangleright Leaves = 1) \\ &\quad \Uparrow Leaf \backslash Leaves \\ &= (Composite \triangleright Leaves = n) \end{aligned}$$

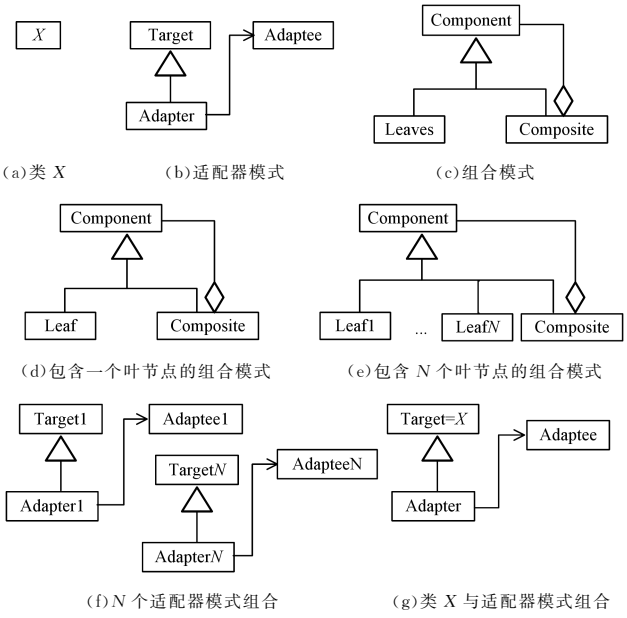
Zhu 等提出的叠加操作符 $*$、升降操作符 $\uparrow$ 和本文提出的叠加操作符；、扩展操作符 $\uparrow$ 都用于描述模式之间的组合关系，通常结合限制操作符一起使用。虽然上述两对操作符的语法和语义类似，但是本文提出的操作符都是基于 Z 语言的关系操作符，因此其支持基于 Z 语言的定理证明器，为之后模式组合的自动识别和有效性验证做了铺垫。例如，Composite 模式和 Adapter 模式的组合使用可以表示为：

$$Composite * Adapter \approx Composite ; Adapter$$

$$Adapter \uparrow Target \approx Adapter \uparrow Target$$

3.5 图例说明

图 2 通过一些简单的例子来说明模式运算符的运用方式，同时也将模式组合后的抽象定义表示出来。第 4 节将用上述操作符来表示模式组合实例中的关系。



(a)类 X (b)适配器模式 (c)组合模式

(d)包含一个叶节点的组合模式 (e)包含 N 个叶节点的组合模式

(f)N 个适配器模式组合 (g)类 X 与适配器模式组合

图 2 例子图例说明

Fig. 2 Illustration of patterns in example

图 2 中，图 2(b)是适配器模式，图 2(c)是组合模式。其中，$X \longrightarrow Y$ 表示类 X 是 Y 的一个子类；$\diamond\!\!\rightarrow$ 表示类 X 包含类 Y 并且类 X 是类 Y 的一部分，即从 X 到 Y 存在复合和聚集关系。

$$a = \langle \{X : Class\} \rangle$$

$$b = \langle \{Target, Adapter, Adaptee : Class\}, Adapter \longrightarrow Target, Adapter \rightarrow Adaptee \rangle$$

$$c = \langle \{Component, Leaves, Composite : Class\}, \forall l \in Leaves \cdot (l \longrightarrow Component \wedge \neg (l \diamond\!\!\rightarrow Componet)), Composite \longrightarrow Component, Composite \diamond\!\!\rightarrow Component \rangle$$

$$Spec(a) = \exists X : Class \cdot (X : Class)$$

$$Spec(b) = \exists Target, Adapter, Adaptee : Class \cdot (Adapter \longrightarrow Target, Adapter \rightarrow Adaptee)$$

$$Spec(c) = \exists Component, Leaves, Composite : Class \cdot (\forall l \in Leaves \cdot (l \longrightarrow Component \wedge \neg (l \diamond\!\!\rightarrow Component)), Composite \longrightarrow Component, Composite \diamond\!\!\rightarrow Component)$$

可以用模式组合操作符定义图 2(d)－图 2(g)：

$$d = c \triangleright (Leaves = 1)$$

$$e = c \triangleleft (Leaves = n)$$

$$f = b \uparrow Target$$

$$g = a ; b \triangleright (X = Target)$$

通过以上的操作符定义，可以得到：

$$Spec(d) = \forall Leaves : Leaf, Component, Composite : Class \cdot (\forall l \in Leaves \cdot (l \longrightarrow Component \wedge \neg (l \diamond\!\!\rightarrow Component)), Composite \longrightarrow Component, Composite \diamond\!\!\rightarrow Component)$$

$$Spec(e) = \exists Leaves : P(Class), Component, Composite : Class \cdot (\forall l \in Leaves \cdot (l \longrightarrow Component \wedge \neg (l \diamond\!\!\rightarrow Component)), Composite \longrightarrow Component, Composite \diamond\!\!\rightarrow Component)$$

$$Spec(f) = \exists Target, Adapter, Adaptee : P(Class) \cdot (\forall Target \in Targets \cdot \exists Adapter \in Adapter) \exists Adaptee \in Adaptee \cdot (Adapter \longrightarrow Target, Adapter \rightarrow Adaptee)$$

$$Spec(g) = \exists X, Target, Adapter, Adaptee : Class \cdot (X \wedge (Adapter \longrightarrow X, Adapter \longrightarrow Target, Adapter \rightarrow Adaptee))$$

上文通过简单的模式来介绍操作符的用法，接下来将通过具体的案例进行验证。

4 案例研究

设计模式的组成通常用“图形：角色注释”来表示。图 3 给出了一个面向模式的通用的请求处理框架实例。它由 3 个设计模式组成：*Command*，*Chain of Resposibility*，*Composite*。下文将通过本文提出的操作符来形式化该组成。

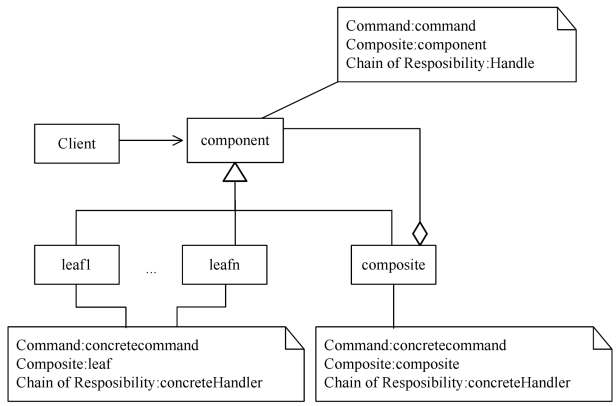


图 3 请求处理框架模型

Fig. 3 Request handling framework mode

该请求处理框架的实际应用场景是应用抽象类组成的 *Command* 模式来声明一组要执行的抽象方法的命令。当客户端发出请求命令时，系统会使用 Chain of Resposibility 模式依次匹配可以处理该请求的程序，匹配的程序如果不能处理该请求，则会推卸责任，请求会发给其他程序处理；同时，该系统使用 Composite 来支持复合命令。

首先，引入一种表达方式来表示一个变量属于一种模式，

而另一个变量属于另一个模式，但是它们却充当相同的角色。比如，假设 $x \in Vars(P)$ 是模式 P 的一个分量并且 $x' \notin Vars(P)$，x 到 x' 的重命名可以表示为 $(x = x')$。显然，重命名不影响模型的可满足性。然后，可以用限制操作符对每个注释进行限制。表 1 通过不同的模式组合形式化方法对比表示了图 3 的请求处理框架。

表 1 不同模式组合形式化方法的对比

Table 1 Comparison of different formal methods of pattern combination

形式化方法	模式组合形式化表示
Dong 等的方法	不能表达一对多或多对多的关系，因此不能描述该组合
Taibi 等的方法	不能表达一对多或多对多的关系，因此不能描述该组合
Zhu 等的方法	$RHF^1 = (Command * Chain\ of\ Resposibility * ((Composite_1 \Uparrow Leaf \setminus Leaves))[command = component = Handle] \wedge [concretecommand = leaves = concreteHandler] \wedge [concretecommand = composite = concreteHandler]$
本文方法	$RHF^2 = (Command; Chain\ of\ Resposibility; Composite) \triangleright (command = component = Handle) \wedge (concretecommand = leaves = concreteHandler = n) \wedge (concretecommand = composite = concreteHandler)$

通过表 1 可以看出，在目前存在的模式组合形式化方法中，Dong 等和 Taibi 等的方法无法表示复杂的模式组合模型，因为这两种方法仅仅相当于本文的限制操作符语义，无法表达一对多或多对多的关系；Zhu 等和本文提出的方法可以完整地表示图 3 所示的模型，其中 Zhu 等的方法采用了限制操作符[ ]、叠加操作符 * 、泛化操作符 $\Uparrow$，而本文方法只采用了限制操作符 $\triangleright$ 和叠加操作符。因此，本文运用了更少的操作符来表达该模型的语义，使得表达式逻辑更加简单，在系统很复杂的情况下更能体现操作符个数少的优点。

在 GoF[1] 书中，每个模式的文档都总结了一个简短的模块，叫做“相关模式”。正如书中的标题所表达的意思，它主要比较和对比模式之间的关系；更重要的是，它提出了其他模式与正在讨论的模式的组合建议，并将这些组合建议总结了出来。文献[18-19]提出将现有的设计模式整理成表格，以帮助软件新手开发人员从列表中选择更合适的设计模式，从而解决软件开发生命周期内设计阶段的问题[20-21]。

将每种模式排列组合，并通过实例验证发现这些表格中的关系并不完善，还有很多关系没有被总结出来。对此，本文将这些关系总结完善并且把它们利用操作符全部形式化地表示出来，供开发人员使用。例如，GoF 一书中指出“A Composite is what the builder often builds”。这可以通过本文提出的操作符正式表示为：

$$(Builder; Composite) \triangleright (Product = Component)$$

在书中，Decorator 模式和 Strategy 模式之间的关系是两者的比较，而不是一个组合建议，Strategy 模式和 Template Method 模式之间的关系也是如此。另外，Iterator 模式和 Visitor 模式以及 Composite 模式和 Iterator 模式之间的关系也没有被正式化，只是在图中有提及，没有在正文中被扩展。案例研究已经证明，本文定义的操作符具有表达性，可以形式化表示设计模式之间的组合关系。

图 4 除了包含 GoF 书中已有的组合关系外，还有新发现的组合关系。其中，编号 26－编号 39 这 14 个关系是本文新增加的关系，这些关系是可形式化的。另外，还有 5 个模式之间的比较关系也在图中被提出，并且用星号代替了数字，这些并不是模式之间的关系，因此不能被形式化表示出来。另有编号 1－编号 25 共 25 个关系是原始的 GoF 书中[1]已经提出的模式之间的组合关系，在此图中不再赘述。同时，将新增的组合关系应用操作符表示出来汇总在表 2 中，以供开发人员参考使用。

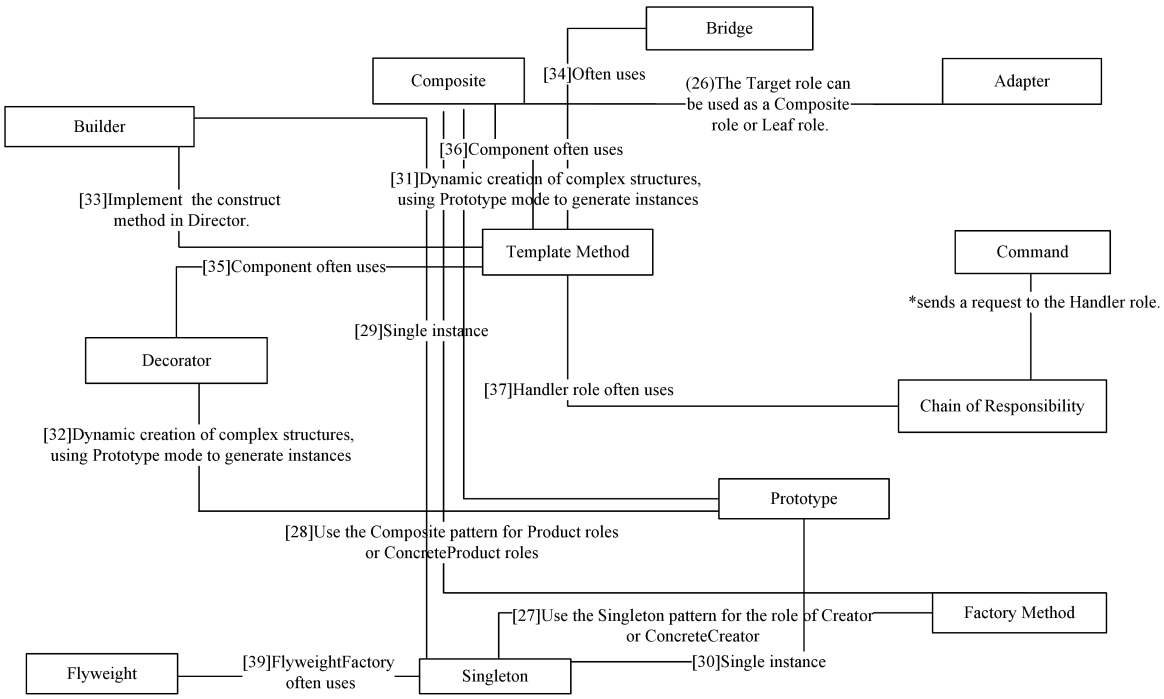


图 4 新增的 GoF 模式之间的关系

Fig. 4 Newly relationship between GoF pattern

表 2 形式化定义模式之间的组合关系

Table 2 Formal definitions of composition relationships between patterns

序号	组合关系定义
26	$(Composite;Adapter) \triangleright (Targets=Leaves)$
27	$(Singleton;FactoryMethod) \triangleright (Singleton=Factory)$
28	$(Factory\ Method;Composite) \triangleright (Product=Component \wedge ConcreteProduct=\{Leaf,composite\})$
29	$(Builder;Singleton) \triangleright (Director=Singleton)$
30	$(Prototype;Singleton) \triangleright (Prototype=Singleton)$
31	$(Prototype;Composite) \triangleright (Prototype=Component)$
32	$(Prototype;Decorator) \triangleright (Prototype=Component)$
33	$(Builder;Template\ Method) \triangleright (Director=AbstractClass)$
34	$(Bridge;Template\ Method) \triangleright (Abstraction=AbstractClass)$
35	$(Decorator;Template\ Method) \triangleright (Prototype=Component)$
36	$(Composite;Template\ Method) \triangleright (Component=AbstractClass)$
37	$(Chain\ of\ resposibility;Template\ Method) \triangleright (Handler=AbstractClass)$
38	$(State;Singleton) \triangleright (ConcreteStates=Singleton)$
39	$(Flyweight;Singleton) \triangleright (FlyweightFactory=Singleton)$

结束语 本文对设计模式组合关系进行了分析，并在前人工作的基础上结合 Z 语言定义了一组全新的模式组合操作符，减少了操作符之间的重叠关系，使模式组合逻辑更为简洁和灵活。通过排列组合的方式逐一分析 GoF 模式之间的组合关系并进行了完善，同时运用本文提出的操作符来表示，以供开发人员使用。

操作符的提出为开发人员的相互交流提供了便利，接下来将运用支持 Z 语言的定理证明器分析模式组合的语义，同时研究运算符满足的性质，通过这些代数性质对多模式的可组合性进行验证。

参 考 文 献

[1] GAMMA E, HELM R, JOHNSON R, et al. Design Patterns: Elements of Reusable Object Oriented Software [M]. Addison-Wesley, 1995.

[2] HOU D, HOOVER H J. Using SCL to specify and check design intent in source code [J]. IEEE Transactions on Software Engineering, 2006, 32(6): 404-423.

[3] SHI N, OLSSON R A. Reverse Engineering of Design Patterns from Java Source Code[C]// IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2007: 123-134.

[4] KIM D K, LU L. Inference of Design Pattern Instances in UML models via Logic Programming[C]// IEEE International Conference on Engineering of Complex Computer Systems. IEEE, 2006: 47-56.

[5] DONG J, ALENCAR P S C, COWAN D D. A behavioral analysis and verification approach to pattern-based design composition [J]. Software and Systems Modeling, 2004, 3(4): 262-272.

[6] DONG J, PENG T, ZHAO Y. Automated verification of security pattern compositions [J]. Information and Software Technology, 2010, 52(3): 274-295.

[7] DONG J, PENG T, ZHAO Y. On Instantiation and Integration Commutability of Design Pattern [J]. The Computer Journal, 2011, 54(1): 164-184.

[8] TAIBI T, NGO D C L. Formal Specification of Design Patterns-A Balanced Approach [C]// Acs/ieee International Conference on Computer Systems & Applications. IEEE, 2003: 14-18.

[9] TAIBI T. Formalising design patterns composition [J]. IEE Proceedings-Software, 2006, 153(3): 127-136.

[10] TAIBI T, NGO D C L. Formal specification of design pattern combination using BPSL [J]. Information and Software Technology, 2003, 45(3): 157-170.

[11] BAYLEY I, ZHU H. Specifying behavioural features of design patterns[C]// IEEE International Computer Software & Applications Conference. Oxford Brookes University, 2008: 203-210.

[12] BAYLEY I, ZHU H. A formal language of pattern composition [C]// Proceedings of The 2nd International Conferences on Pervasive Patterns and Applications. Portugal, 2010: 1-6.

[13] ZHU H. An institution theory of formal meta-modelling in graphically extended BNF [J]. Frontiers of Computer Science, 2012, 6(1): 40-56.

[14] ZHU H, BAYLEY I. An algebra of design patterns [J]. ACM Transactions on Software Engineering and Methodology, 2013, 22(3): 1-35.

[15] ZHU H, BAYLEY I. Laws of pattern composition[C]// International Conference on Formal Engineering Methods & Software Engineering. Springer-Verlag, 2010: 630-645.

[16] ZHU H, BAYLEY I. On the Composability of Design Patterns [J]. IEEE Transactions on Software Engineering, 2015, 41(11): 1138-1152.

[17] BENNETT M, BACLAWSKI K, BENNETT M. Ontology design patterns and semantic abstractions in ontology integration [J]. Applied Ontology, 2017, 12(3/4): 341-349.

[18] LUCIA A D, DEUFEMIA V, GRAVINO C, et al. Detecting the Behavior of Design Patterns through Model Checking and Dynamic Analysis [J]. Acm Transactions on Software Engineering & Methodology, 2018, 26(4): 1-41.

[19] CACHO N, CLAUDIO S, FIGUEIREDO E, et al. Blending design patterns with aspects: A quantitative study [J]. Journal of Systems and Software, 2014, 98(23): 117-139.

[20] HUSSAIN S, KEUNG J, KHAN A A. Software design patterns classification and selection using text categorization approach [J]. Applied Soft Computing, 2017, 58(6): 225-244.

[21] DWIVEDI A K, TIRKEY A, RATH S K. Software design pattern mining using classification-based techniques [J]. Frontiers of Computer Science, 2018, 12(1): 1-15.

JI Cheng-yu, master, is not member of China Computer Federation (CCF). Her main research interests include pattern combination and verification and so on.

ZHU Xue-feng, Ph. D, master supervisor, is not member of China Computer Federation (CCF). His main research interests include software engineering and so on.