文章编号:1007-5321(2017)05-0036-07 **DOI**:10.13190/j.jbupt.2017-076

考虑多种特征因素的设计模式自动识别

王 雷[1], 王智广[1,2]

(1. 中国矿业大学(北京) 机电与信息工程学院, 北京 100083; 2. 中国石油大学(北京) 地球物理与信息工程学院, 北京 102249)

摘要: 现有的设计模式自动识别方法大多只考虑设计模式的结构特征,识别准确率不高. 为此,提出了一种考虑多种特征因素的设计模式自动识别方法. 首先,提出了一种基于特征矩阵的待考查系统和设计模式的形式化描述方法;然后,给出了设计模式自动识别的基本流程,并详细讨论了综合考虑多种特征因素的设计模式识别算法;最后,实现了该方法的支撑工具,并使用该工具对一个开源项目进行了设计模式的识别. 实验结果表明,相对于只考虑结构特征的设计模式识别方法,该方法的识别准确率较高. 对于结构特征不够明显或者与其他模式具有相似结构特征的设计模式,识别准确率明显提升.

关 键 词: 设计模式识别; 识别准确率; 多种特征因素; 软件逆向工程

中图分类号: TP311.5 **文献标志码**: A

Automatic Design Pattern Detection on the Consideration of Multiple Characteristic Factors

WANG Lei[1], WANG Zhi-guang[1,2]

(1. School of Mechanical Electronic and Information Engineering, China University of Mining and Technology (Beijing), Beijing 100083, China; 2. College of Geophysics and Information Engineering, China University of Petroleum (Beijing), Beijing 100249, China)

Abstract: Most of the existing methods for automatic design pattern detection only consider structural characteristics of design patterns, so the detection accuracy rate is not high enough. Therefore, a method for automatic design pattern detection on the consideration of multiple characteristic factors was proposed. First, a formal description method for system under study and design patterns based on characteristic matrix was proposed. Then the basic flow of automatic design pattern detection was given, and the design pattern detection algorithm considering multiple characteristic factors was discussed in detail. Finally, a support tool for this method is implemented, and design patterns in an open source project were detected by using this tool. The experimental results show that compared with the design pattern detection methods which only consider structural characteristics, detection accuracy rate of this method is higher. For design patterns whose structural characteristics are not obvious or who have similar structure characteristics with other design patterns, accuracy rate is improved obviously.

Key words: design pattern detection; detection accuracy rate; multiple characteristic factors; software re-engineering

设计模式使人们可以更加简单方便地利用成功的设计和体系结构,在大型软件项目的开发中

收稿日期: 2017-05-12

基金项目: 国家自然科学基金项目(60873093); 国家重点基础研究发展计划(973 计划)项目(2013CB228602)

作者简介: 王 雷(1988—), 男, 博士生; 王智广(1964—), 男, 教授, 博士生导师, E-mail:wangzhiguang0602@foxmail.com.

得到了广泛的应用. 从统一建模语言(UML, unified modeling language)模型中自动识别出相应的设计模式,可以为面向设计模式的软件理解、维护和重构等活动提供自动化支持[1]. 因此,设计模式的自动识别已经成为目前软件逆向工程领域的一个研究热点.

近年来,国内外的相关文献已经提出很多设计模式自动识别的方法,例如,许等[2]从类的属性、类间关系和整个模型 3 个角度出发对设计模式进行识别;周等[3]主要考虑从源代码中识别设计模式中的聚集关系;Pradhan 等[4]提出了一种基于图同构和归一化互相关的设计模式识别方法;Bernardi 等[5-7]使用图形匹配技术对设计模式进行识别. 然而,这些方法大多只考虑了设计模式的结构特征,因此识别的准确率并不高,尤其是对于结构特征不够明显或与其他模式具有相似结构的模式更是如此. Dong 等[8]考虑了 8 个设计模式的特征并将它们集成在一个特征矩阵中,可以较为精确地实现设计模式识别. 但该方法所考虑的 8 个特征仍然全部都是结构特征.

针对以上问题,提出一种考虑多种特征因素的设计模式自动识别方法,并实现了该方法的支撑工具. 该方法基于相似度评分算法,在设计模式自动识别过程中考虑了设计模式的 3 种因素:结构特征、命名特征和图形特征,具有较高的识别准确率. 对于结构特征不够明显或者与其他模式具有相似结构特征的设计模式,效果尤其明显.

1　相似度评分算法

提出的设计模式自动识别方法基于相似度评分算法. 下面对该算法进行简单介绍.

该算法由 Blondel 等[9]提出,用来计算两个矩阵的相似度. 设 $\boldsymbol{A}$、$\boldsymbol{B}$ 分别为有向图 G_A 和 G_B 的邻接矩阵,n_A 和 n_B 分别为 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 的阶数,$\boldsymbol{Z}_0$ 为一个 $n_B \times n_A$ 的矩阵,其元素全为 1. 相似度矩阵 $\boldsymbol{S}$ 定义为一个 $n_B \times n_A$ 的矩阵,其元素 s_{ij} 称为顶点 j(在 G_A 中)和顶点 i(在 G_B 中)之间的相似度得分,表示这两个顶点的相似程度.

计算 $\boldsymbol{S}$ 的详细算法见文献[9].

2　系统和设计模式的形式化描述方法

要实现设计模式的自动识别,首先要解决的问题是待考查系统和设计模式的形式化描述[10]. 通过计算待考查系统和设计模式之间的相似度矩阵来实现设计模式的识别,因此需要将系统和设计模式表示为矩阵形式. 为提高识别的准确率,考虑设计模式的 3 种特征因素:结构特征、命名特征和图形特征. 下面给出结构特征矩阵、命名特征矩阵和图形特征矩阵的定义.

2.1　结构特征矩阵

结构特征矩阵共包含关联关系矩阵等 4 个矩阵.

定义 1　类图 G 的关联关系矩阵定义为

$$\boldsymbol{M}_G^{\mathrm{Ass}} = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & \cdots & c_j & \cdots & c_n \end{array} \\ \begin{bmatrix} & & & & & \\ & & & & & \\ & & & r_{ij} & & \\ & & & & & \\ & & & & & \end{bmatrix} \end{array} \tag{1}$$

其中

$$r_{ij} = \begin{cases} 1 & \text{类 } c_i \text{ 到 } c_j \text{ 之间存在关联关系} \\ 0 & \text{类 } c_i \text{ 和 } c_j \text{ 之间不存在关联关系} \end{cases}$$

类似地可以定义类图 G 的泛化关系矩阵 $\boldsymbol{M}_G^{\mathrm{Gen}}$、依赖关系矩阵 $\boldsymbol{M}_G^{\mathrm{Dep}}$ 和聚合关系矩阵 $\boldsymbol{M}_G^{Agg}$.

2.2　命名特征矩阵

为了增强模型的可读性和可维护性,不同设计模式的 UML 表示具有不同的命名特征,如表 1 所示.

定义 2　类图 G 的关于设计模式 pattern 的命名矩阵定义为

$$\boldsymbol{M}_{G,\mathrm{pattern}}^{\mathrm{Name}} = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{array} \begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & \cdots & c_j & \cdots & c_n \end{array} \\ \begin{bmatrix} & & & & & \\ & & & & & \\ & & & r_{ij} & & \\ & & & & & \\ & & & & & \end{bmatrix} \end{array} \tag{2}$$

其中

$$r_{ij} = \begin{cases} 1 & i = j \text{ 且类 } c_i \text{ 包含模式 pattern 命名特征字符串} \\ 0 & \text{其他} \end{cases}$$

2.3　图形特征矩阵

另外,不同设计模式的 UML 表示具有不同的图形特征. 例如,在观察者、抽象工厂、工厂方法、外观、享元、解释器、迭代器、中介者模式的 UML 表示

表1 23种设计模式命名特征

类型	设计模式	命名特征字符串
创建型	抽象工厂	factory, product
	生成器	builder, product
	工厂方法	product, creator, builder
	原型	client, prototype
	单件	singleton
结构型	适配器	client, target, adaptee, adapter
	桥接	refined, implementor
	组合	client, component, composite
	装饰	component, decorator
	外观	façade
	享元	flyweight, factory
	代理	client, subject
行为型	职责链	client, handler
	命令	product, invoker, receiver, command Invoker
	解释器	context,client, expression
	迭代器	client, aggregate, iterator Aggregate
	中介者	mediator, colleague Mediator
	备忘录	originator, memento, caretaker
	观察者	subject, observer
	状态	context, state
	策略	context, strategy
	模板方法	template
	访问者	client, visitor, element

中,类的关系会形成一个环状.

定义3 类图 G 的图形特征矩阵定义为

$$\boldsymbol{M}_G^{\text{Graph}}=\begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_i \\ \vdots \\ c_n \end{array}\begin{array}{c} \begin{array}{cccccc} c_1 & c_2 & \cdots & c_j & \cdots & c_n \end{array} \\ \begin{bmatrix} & & & & & \\ & & & & & \\ & & & & & \\ & & & r_{ij} & & \\ & & & & & \\ & & & & & \end{bmatrix} \end{array} \tag{3}$$

其中

$$r_{ij}=\begin{cases}1 & \text{存在从类 } c_i \text{ 到 } c_j \text{ 的关联、泛化或依赖关系} \\ 0 & \text{类 } c_i \text{ 和 } c_j \text{ 之间不存在关系}\end{cases}$$

3 设计模式的自动识别

3.1 基本流程

UML模型的矩阵描述包含了原模型的结构特征、命名特征和图形特征信息. 在本文中,将待考查系统划分为若干个子系统,通过计算子系统的特征矩阵和设计模式特征矩阵之间的相似度矩阵来实现设计模式的识别. 基于相似度评分的设计模式自动识别流程如图1所示.

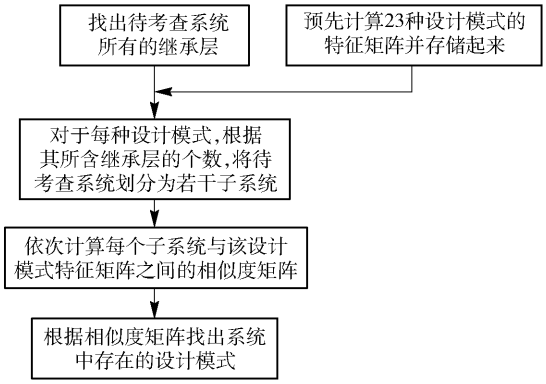

图1 设计模式自动识别基本流程

3.2 23种设计模式的形式化描述

根据上述流程可知,需要提前获取23种设计模式的特征矩阵,并存储起来.

装饰模式的UML类图描述如图2所示.

记 Component、ConcreteComponent、Decorator 和 ConcreteDecorator 分别为 c_1、c_2、c_3 和 c_4. 根据结构特征矩阵的定义,可得

$$\boldsymbol{M}_{\text{decorator}}^{\text{Ass}}=\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{4a}$$

$$\boldsymbol{M}_{\text{decorator}}^{\text{Gen}}=\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{4b}$$

$$\boldsymbol{M}_{\text{decorator}}^{\text{Dep}}=\boldsymbol{M}_{\text{decorator}}^{\text{Agg}}=\boldsymbol{0} \tag{4c}$$

根据命名特征矩阵的定义,可得

$$\boldsymbol{M}_{\text{decorator,decorator}}^{\text{Name}}=\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \tag{5}$$

根据图形特征矩阵的定义,可得

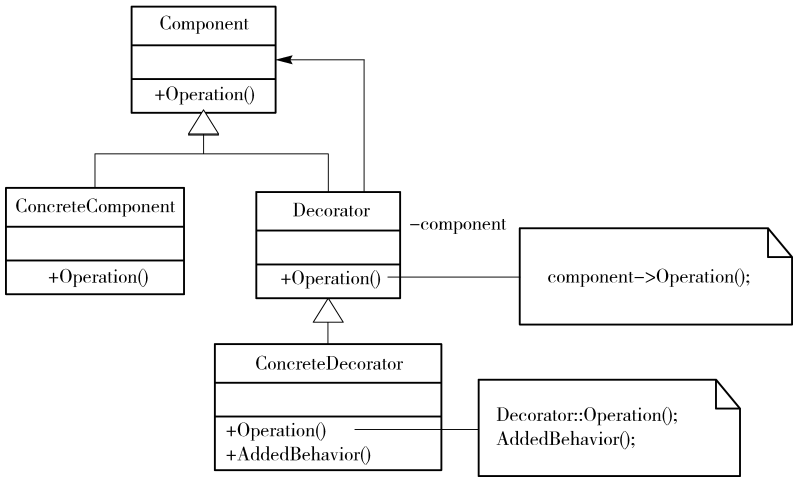

图 2　装饰模式的 UML 类图描述

$$\boldsymbol{M}^{\text{Graph}}_{\text{decorator}}=\begin{array}{c}\\ c_1\\ c_2\\ c_3\\ c_4\end{array}\begin{array}{c}\begin{array}{cccc}c_1 & c_2 & c_3 & c_4\end{array}\\ \begin{bmatrix}0 & 1 & 1 & 0\\ 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\end{bmatrix}\end{array} \tag{6}$$

类似地,可以得到其他 22 种设计模式的特征矩阵.

3.3　子系统的划分

为提高识别的准确率,提升算法执行的性能,需要将待考查系统划分为若干子系统[11].

根据待识别的设计模式所含继承层的个数,划分子系统有以下两种方法:

1) 如果设计模式不包含继承层或只包含一个继承层,则将待考查系统的每个继承层划分为一个独立的子系统. 此时,子系统的个数和系统的继承层个数相等. 该类设计模式包括 15 种设计模式:生成器、原型、单例、组合、装饰、享元、代理、职责链、命令、解释器、备忘录、状态、策略、模板方法和访问者.

2) 如果设计模式包含两个继承层,则每次从所有的继承层中选择两个划分为一个子系统. 此时,子系统的个数等于$\frac{m(m-1)}{2}$,其中 m 为系统中继承层的个数. 该类设计模式包括 8 种设计模式:抽象工厂、工厂方法、适配器、桥接、外观、迭代器、中介者和观察者.

3.4　子系统与设计模式相似度矩阵的计算

对于每种设计模式,首先根据其所含继承层的个数,将待考查系统划分为 m 或$\frac{m(m-1)}{2}$个子系统(m 为系统中继承层的个数). 然后依次计算各子系统的特征矩阵和设计模式特征矩阵之间的相似度矩阵.

计算某个子系统 subSystem 和某种设计模式 pattern 之间的相似度矩阵 $S_{\text{subSystem,pattern}}$ 的算法如下(函数 Similarity( ) 对应第 2 节中的相似度评分算法):

步骤 1　计算 subSystem 和 pattern 之间的结构特征相似度矩阵

$$\begin{aligned}\boldsymbol{S}^{\text{Structure}}_{\text{subSystem,pattern}}=&\text{Similarity}(\boldsymbol{M}^{\text{Ass}}_{\text{pattern}},\boldsymbol{M}^{\text{Ass}}_{\text{subSystem}})+\\&\text{Similarity}(\boldsymbol{M}^{\text{Gen}}_{\text{pattern}},\boldsymbol{M}^{\text{Gen}}_{\text{subSystem}})+\\&\text{Similarity}(\boldsymbol{M}^{\text{Dep}}_{\text{pattern}},\boldsymbol{M}^{\text{Dep}}_{\text{subSystem}})+\\&\text{Similarity}(\boldsymbol{M}^{\text{Agg}}_{\text{pattern}},\boldsymbol{M}^{\text{Agg}}_{\text{subSystem}})\end{aligned} \tag{7}$$

步骤 2　计算 subSystem 和 pattern 之间的命名特征相似度矩阵

$$S^{\text{Name}}_{\text{subSystem,pattern}}=\text{Similarity}(\boldsymbol{M}^{\text{Name}}_{\text{pattern,pattern}},\boldsymbol{M}^{\text{Name}}_{\text{subSystem,pattern}}) \tag{8}$$

步骤 3　计算 subSystem 和 pattern 之间的图形特征相似度矩阵

$$\boldsymbol{S}^{\text{Graph}}_{\text{subSystem,pattern}}=\text{Similarity}(\boldsymbol{M}^{\text{Graph}}_{\text{pattern}},\boldsymbol{M}^{\text{Graph}}_{\text{subSystem}}) \tag{9}$$

步骤 4　计算总相似度矩阵

$$\begin{aligned}\boldsymbol{S}_{\text{subSystem,pattern}}=&\boldsymbol{S}^{\text{Structure}}_{\text{subSystem,pattern}}+\boldsymbol{S}^{\text{Name}}_{\text{subSystem,pattern}}+\\&\boldsymbol{S}^{\text{Graph}}_{\text{subSystem,pattern}}\end{aligned} \tag{10}$$

3.5　基于相似度矩阵的设计模式识别

获取子系统和某种设计模式之间的相似度矩阵 $\boldsymbol{S}_{\text{subSystem,pattern}}$ 后,就可以根据相似度矩阵找出系统中包含的该种设计模式的实例.

通常情况下,对于每种设计模式,每个子系统只包含该设计模式的一个实例,此时每个模式角色关联子系统中的一个类. 提出的方法目前只考虑子系

统包含待识别模式的一个实例的情况,多个实例的情况将在后续的研究中进行讨论.

相似度矩阵 $\boldsymbol{S}_{\text{subSystem,pattern}}$ 的元素表示两个类之间的相似度得分. 所以,需要选择一个值,当相似度得分大于该值时,就认为这两个类匹配. 将该值称为匹配临界值,记为 v.

根据子系统和某种设计模式之间的相似度矩阵识别该种设计模式的算法如下:

步骤 1 依次判断 $\boldsymbol{S}_{\text{subSystem,pattern}}$ 的每列,若所有的列均有至少一个元素的值大于等于匹配临界值 v,则说明该子系统包含模式 pattern;否则不包含.

步骤 2 若包含模式 pattern,则需要找到该模式的每个角色在子系统 subSystem 中关联的类. 依次考查 $S_{\text{subSystem,pattern}}$ 的每列,从中找出值最大的元素,则该列对应的设计模式角色关联该元素所在行对应的子系统类.

若临界值 v 选取过大,则可能会遗漏掉某些包含的设计模式实例;而若 v 选取过小则可能会出现误判的情况. 根据经验,这里取 $v=0.6$.

4 实验及结果分析

目前提出方法的支撑工具已经实现. 该工具采用 MFC 开发,输入 UML 类图模型,输出识别结果. 为说明该工具的有效性,对开源项目 C++图像处理库 ImageProcessor 进行了设计模式识别. 下面给出其中的一个识别实例,然后对该项目的识别结果进行分析和对比.

4.1 识别实例

在对该项目进行设计模式识别之前,使用 Microsoft Visio 逆向工程将源码转化为 UML 类图模型[12]. 这里以装饰模式的识别为例,首先将待考查系统划分为 m 个子系统(m 为系统中继承层的个数). 图 3 为其中的一个子系统,记为 subsystem.

记 Component、View、Director、TextView、Decorator、ImageBuilder、BorderDecorator 和 ImageProduct 分别为 C_1、C_2、C_3、C_4、C_5、C_6、C_7 和 C_8. 经计算可得子系统 subsystem 和装饰模式之间的相似度矩阵为

$$\boldsymbol{S}_{\text{subSystem,decorator}} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \\ C_8 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{bmatrix} 1.001\,6 & 0.402\,6 & 0.501\,6 & 0.402\,6 \\ 0.848\,1 & 0.056\,5 & 0.238\,5 & 0.056\,5 \\ 0.085\,8 & 0 & 0.585\,8 & 0 \\ 0.646\,8 & 0.803\,8 & 0.646\,8 & 0.108\,9 \\ 0.539\,6 & 0.499\,3 & 1.539\,6 & 0.413\,4 \\ 0.063\,6 & 0.049\,3 & 0.063\,6 & 0.220\,5 \\ 0.042\,6 & 0.402\,6 & 0.402\,6 & 0.673\,8 \\ 0.021\,4 & 0.021\,4 & 0.021\,4 & 0.021\,4 \end{bmatrix} \end{array} \tag{11}$$

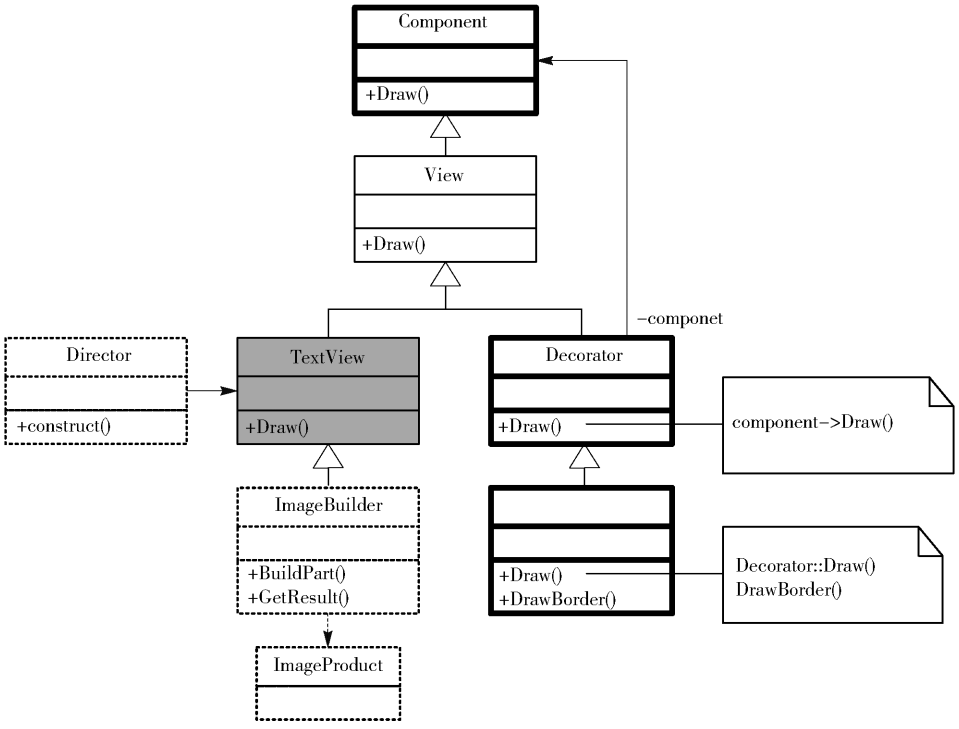

图 3 C++图像处理库 ImageStone 的一个子系统

易见 $\boldsymbol{S}_{\text{subSystem,decorator}}$ 的每列均有至少一个元素的值大于等于匹配临界值 $v=0.6$，则说明该子系统包含装饰模式实例.

在 c_1 对应的列中，值最大的元素对应 C_1 行，则说明子系统中的类 C_1 关联装饰模式的角色类 c_1. 类似地，可得类 C_4、C_5、C_7 分别关联装饰模式的角色类 c_2、c_3、c_4. 同理可得该子系统中包含生成器模式实例. 在图 4 中，加粗的类关联装饰模式的角色，虚线的类关联生成器模式的角色，灰色填充的类为两个设计模式实例共用的类.

需要注意的是，该子系统包含的装饰模式实例是一个变形的装饰模式（装饰模式角色所关联的类 component 和 decorator 中间增加了一个类 view），如果只考虑结构特征，可能会错误地得到子系统中的类 View 关联装饰模式的角色 Component 的结论. 但是通过综合考虑结构特征、命名特征和图形特征 3 种因素，避免了这种情况的出现.

4.2　结果分析

为将该方法与其他方法进行定量对比，给出成功识别个数、错误识别个数、遗漏识别个数和识别准确率的一种严格定义.

成功识别个数是指系统中确实包含的并且被正确识别出的设计模式实例个数，记为 N_S；错误识别个数是指系统中不包含，却被误判为包含的设计模式实例个数，记为 N_W；遗漏识别个数是指系统中包含，却未正确识别出的设计模式实例个数，记为 N_O. 显然，系统中实际包含的设计模式实例总数为 N_S+N_O.

定义 4　识别准确率定义为

$$R_A=\frac{N_S-N_W}{N_S+N_O}\times 100\% \tag{12}$$

笔者进一步完善了文献[4-6]方法的支撑工具. 表 2 列出了采用这些工具和提出方法的支撑工具对开源项目 C++图像处理库 ImageProcessor 进行设计模式识别的结果.

表 2　不同方法下的识别结果

设计模式	文献[4]方法				文献[5]方法				文献[6]方法				提出方法			
	N_S	N_W	N_O	R_A/%	N_S	N_W	N_O	R_A/%	N_S	N_W	N_O	R_A/%	N_S	N_W	N_O	R_A/%
抽象工厂	2	0	1	66.67	0	1	3	-33.33	1	0	2	33.33	2	0	1	66.67
单件	0	0	3	0	1	0	2	33.33	0	0	3	0	2	0	1	66.67
适配器	3	2	1	25.00	4	2	0	50.00	4	1	0	75.00	3	2	1	25.00
桥接	1	0	3	25.00	3	2	1	25.00	2	0	2	50.00	3	1	1	50.00
组合	2	1	3	20.00	3	1	2	40.00	2	0	3	40.00	2	1	3	20.00
工厂方法	1	0	1	50.00	1	0	1	50.00	1	1	1	0	1	1	1	0
外观	3	1	2	40.00	3	2	2	20.00	4	0	1	80.00	4	0	1	80.00
享元	1	0	1	50.00	1	0	1	50.00	2	1	0	50.00	2	1	0	50.00
装饰	1	2	2	-33.33	1	1	2	0	1	1	2	0	2	0	1	66.67
生成器	0	0	1	0	1	1	0	0	1	1	0	0	1	0	0	100.00
状态	0	0	5	0	3	6	2	-60.00	3	5	2	-40.00	4	1	1	60.00
策略	3	4	4	-14.29	0	0	7	0	0	0	7	0	5	1	2	57.14
模板方法	0	0	3	0	0	0	3	0	1	0	2	33.33	3	0	0	100.00
平均				15.27				11.67				21.44				49.48

由表 1 可知，文献[4-6]的方法的平均识别准确率分别为 15.27%、11.67% 和 21.44%. 而提出方法的平均准确率达到了 49.48%. 模板方法模式的结构特征不够明显，仅包含一个继承关系. 对于模板方法模式，文献[4]方法和文献[5]方法的准确率为 0，文献[6]方法的准确率也仅为 33.33%，而提出的方法通过模板方法模式的命名特征使得识别的准确率达到了 100%. 状态模式和策略模式具有相似的结构特征，仅依靠结构特征来区分这两种模式非常困难. 对于状态模式与策略模式，文献[4-6]提出的方法均无法准确地区分这两种设计模式，使得识别准确率很低，而提出的方法这两种模式的准确率分别达到 60.00% 和 57.14%.

根据以上分析可知，该方法可以实现设计模式

的自动识别,且准确率较高. 尤其是当设计模式的结构特征不够明显或者与其他模式具有相似的结构特征时,识别准确率明显提升.

5 结束语

现有的设计模式自动识别方法大多只考虑设计模式的结构特征,因此识别的准确率并不高. 提出的方法在对设计模式进行自动识别的过程中借助了相似度评分算法,考虑了设计模式的结构特征、命名特征和图形特征3个因素,增大了识别的准确率. 该方法尤其适用于结构特征不够明显或者与其他模式具有相似结构特征的设计模式.

目前该方法在设计模式的识别中考虑了其结构特征、命名特征和图形特征. 今后的主要工作如下:

1) 目前所考虑的3个特征因素均是静态特征,后期将研究如何结合静态结构与设计模式的动态特征综合进行识别;

2) 提出的方法目前只考虑子系统包含待识别模式的一个实例的情况,多个实例的情况将在后续的研究中进行讨论;

3) 如何对子系统的划分和相似度矩阵的计算算法进行优化以提高算法的时间性能也是下一步研究的重点.

参考文献:

[1] Issaoui I, Bouassida N, Ben-Abdallah H. Using metric-based filtering to improve design pattern detection approaches [J]. Innovations in Systems and Software Engineering, 2015, 11(1): 39-53.

[2] 许涵斌, 张学林, 郑晓梅, 等. 一种基于结构查询的UML设计模式识别方法 [J]. 计算机科学, 2014, 41(11): 50-55.
Xu Hanbin, Zhang Xuelin, Zheng Xiaomei, et al. UML design pattern recognition method based on structured query [J]. Computer Science, 2014, 41(11): 50-55.

[3] 周晓宇, 钱巨, 陈林, 等. 基于形态分析识别设计模式中的集中管理式聚集 [J]. 软件学报, 2010, 21(11): 2725-2737.
Zhou Xiaoyu, Qian Ju, Chen Lin, et al. Identification of centrally managed aggregations in design patterns using shape analysis [J]. Journal of Software, 2010, 21(11): 2725-2737.

[4] Pradhan P, Dwivedi A K, Rath S K. Detection of design pattern using graph isomorphism and normalized cross correlation [C] // Eighth International Conference on Contemporary Computing. Noida: IEEE, 2015: 208-213.

[5] Bernardi M L, Cimitile M, Lucca G A D. A model-driven graph-matching approach for design pattern detection [C] // 20th Working Conference on Reverse Engineer-ing. Beverly: IEEE, 2013: 172-181.

[6] Rao R S, Gupta M. Design pattern detection by greedy algorithm using inexact graph matching [J]. International Journal of Engineering Research and Technology, 2013, 10(2): 3658-3664.

[7] Bernardi M L, Cimitile M, Lucca G D. Design pattern detection using a DSL-driven graph matching approach [J]. Journal of Software Evolution & Process, 2014, 26(12): 1233-1266.

[8] Dong J, Sun Y T, Zhao Y J. Design pattern detection by template matching[C] // The 23rd Annual ACM Symposium on Applied Computing. Fortaleza: ACM, 2008: 765-769.

[9] Blondel V D, Gajardo A, Heymans M, et al. A measure of similarity between graph vertices: applications to synonym extraction and web searching [J]. SIAM Review, 2004, 46(4): 647-666.

[10] Bayley I, Zhu H. Formal specification of the variants and behavioural features of design patterns [J]. Journal of Systems and Software, 2010, 83(2): 209-221.

[11] Tsantalis N, Chatzigeorgion A, Stephanides G, et al. Design pattern detection using similanity scoring [J]. IEEE Transactions on Software Engineering, 2006, 32(11): 896-909.

[12] Nanthaamornphong A, Morris K, Filippone S. Extracting UML class diagrams from object-oriented Fortran: for UML [C] // International Workshop on Software Engineering for High Performance Computing in Compu-ta-tional Science and Engineering. San Francisco: IEEE, 2013: 9-16.