文章编号:1674-2974(2019)12-0050-08 DOI:10.16339/j.cnki.hdxbzkb.2019.12.007

一种基于相似度评分的设计模式识别方法

王雷[1,2†],宋慧娜[1],王文发[1]
(1.延安大学 数学与计算机科学学院,陕西 延安 716000;
2.中国矿业大学(北京) 机电与信息工程学院,北京 100083)

摘 要:使用软件工具自动识别UML(Unified Modeling Language)模型中包含的设计模式,可以帮助软件开发人员理解、维护和重构大型软件项目.现有设计模式识别方法大多是将所考虑的若干个特征分别进行匹配,准确率和时间性能不高.为此,提出了一种基于相似度评分的设计模式识别方法.首先,给出该方法的基本流程;然后,提出一种基于有向图/矩阵的设计模式和系统的表示;接着,详细讨论了基于相似度评分的模式实例搜索算法;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.该方法不是将所考虑的若干个特征分别进行匹配,而是使用总特征矩阵进行匹配.实验结果表明,相对于将所考虑的若干个特征分别进行匹配的方法,该方法的识别准确率和时间性能更高.

关键词:设计模式识别;准确率;时间性能;有向图;软件逆向工程
中图分类号:TP311.5 **文献标志码**:A

A Design Pattern Detection Method Based on Similarity Scoring

WANG Lei[1,2†],SONG Huina[1],WANG Wenfa[1]
(1.College of Mathematics and Computer Science,Yan'an University,Yan'an 716000,China;
2.School of Mechanical Electronic and Information Engineering,China University of Mining and Technology (Beijing),Beijing 100083,China)

Abstract:Detecting design pattern instances in UML models by using software tools can help software developers to understand,maintain and reconstruct the large-scale software projects. Most of the existing methods for automatic design pattern detection let the several considered features match separately,so that the accuracy rate and time performance are not high enough. Therefore,a design pattern detection method based on similarity scoring was proposed. First,the basic process of this method was given; then,a representation of system and patterns based on directed graph/matrix was proposed; the pattern instance search algorithm based on similarity scoring was discussed in detail; finally,a supporting tool for this method was implemented,and design patterns in an open source project were detected by using this tool. This method does not let the several considered features match separately,but uses the integral feature matrixes to match. The experimental results show that,compared with the design pattern detec-

* **收稿日期**:2019-01-24
基金项目:国家自然科学基金资助项目(60873093),National Natural Science Foundation of China (60873093);国家科技重大专项资助项目(2017ZX05018-005),National Science and Technology Major Project of China(2017ZX05018-005)
作者简介:王雷(1988—),男,陕西延安人,延安大学讲师,博士
† 通讯联系人,E-mail:wanglei0823@foxmail.com

tion methods which let the several considered features match separately, the detection accuracy rate and time performance of this method are higher.

Key words: design pattern detection; accuracy; time performance; directed graph; 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,其在大型软件项目的开发中得到了广泛的应用. 从源代码或统一建模语言(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成为目前逆向软件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.许涵斌等[2]、Yu 等[3]、Bernardi 等[4-6]将系统和设计模式以有向图的形式呈现,通过图同构判定算法在系统图中寻找模式子图;Dong 等[7-8]将泛化、关联、抽象、不同的调用方式等 8 个设计特征编码到有向图/矩阵中,并将 8 个矩阵组合成一个矩阵,使用模板匹配算法计算系统和设计模式之间的互相关值来寻找系统中存在的模式实例;Tsantalis 等[9]将源代码和设计模式的关联、泛化、抽象类、对象创建、抽象方法调用等信息均表示为一个单独的有向图/矩阵,使用相似度评分算法[10]计算各子系统与设计模式之间的相似度矩阵来寻找子系统中的模式实例;Costagliola 等[11-12]和 Lucia 等[13-14]将可缩放矢量图形(SVG, scalable vector graphics)格式用于源代码的中间表示,而设计模式用视觉语言表示,通过将每种模式的视觉语言语法与系统的 SVG 表示进行映射来恢复模式;Balanyi 等[15]使用一种基于 XML 的语言进行设计模式描述,用 Columbus 框架分析 C++源代码并从中构建抽象语义图(ASG, abstract semantic graph), 通过将模式与 ASG 匹配来寻找模式实例;Bernardi 等[16-18]将设计模式的行为属性表示为选择性 μ 演算公式,将文献[4-6]得到的候选实例(Java 文件或字节码)转换为时序规范语言(LOTOS, Language of Temporal Ordering Specification)模型,使用模型检测工具 CADP 验证候选实例是否满足 μ 演算公式来判断候选实例是否为模式实例;与 Bernardi 等[16-18]方法类似,Lucia 等[19]使用工具 SPIN 验证文献[11-14]得到的候选实例是否满足设计模式的行为属性转换得到的线性时态逻辑(LTL, Linear Temporal Logic)公式,来判断候选实例是否为模式实例;Wendehals 等[20]将行为型模式的行为特征转换为有限自动机, 并将候选实例的方法调用与有限自动机进行匹配, 以此来判断候选实例是否为模式实例;Zhu 等[21]使用 LAMBDES 系统将待识别系统的 UML 图转换为一组一阶逻辑 (FOL, first order logic)语句,借助定理证明器 SPASS 验证设计是否符合用 FOL 描述设计模式的结构特征和行为特征[22-24];苗康等[25]借用关系演算语言 UTP 描述系统和设计模式, 通过关系演算算法检查系统是否满足模式的属性;Kim 等[26]计算系统每个类的面向对象度量、结构型度量和过程型度量 3 种产品度量, 并将这些度量与支撑工具中嵌入的模式签名进行比较, 根据匹配程度判断是否模式实例;Hayashi 等[27]将从待识别系统中抽取到的信息表示为 Prolog 中的事实, 执行定义为 Prolog 中的规则的检测条件, 来推断满足设计模式条件的类结构的存在;类似工作,Luitel 等[28]将待识别系统的类图和序列图表示为 ASP 中的事实, 使用 ASP 求解器输出分别遵循描述结构模式和行为模式的规则的结构和行为元素.

综上所述,国内外的相关文献已经将图论算法[2-9]、视觉语言解析技术[11-14]、XML 匹配[15]、形式化验证技术[16-21,25]、软件度量的匹配[26]、逻辑推理[27-28]等多种匹配技术引入到设计模式的搜索中来. 这些方法大多是将所考虑的若干个特征分别进行匹配. 单个特征的匹配并不能保证整体匹配, 反之亦然. 此外,多次匹配需要耗费大量的时间,因此识别准确率和时间性能并不高.

本文提出一种基于相似度评分的设计模式识别方法,并实现了该方法的支撑工具. 该方法将所考虑的 7 个特征对应的矩阵组合成一个总特征矩阵,并使用总特征矩阵进行匹配, 具有更高的识别准确率和时间性能.

1　基本流程

本文将关联、泛化、依赖、聚合、抽象类、对象创建、抽象方法调用等特征对应的矩阵组合成一个总

特征矩阵,通过计算子系统和设计模式的总特征矩阵之间的相似度矩阵来寻找子系统中的模式实例. 基本流程如图 1 所示.

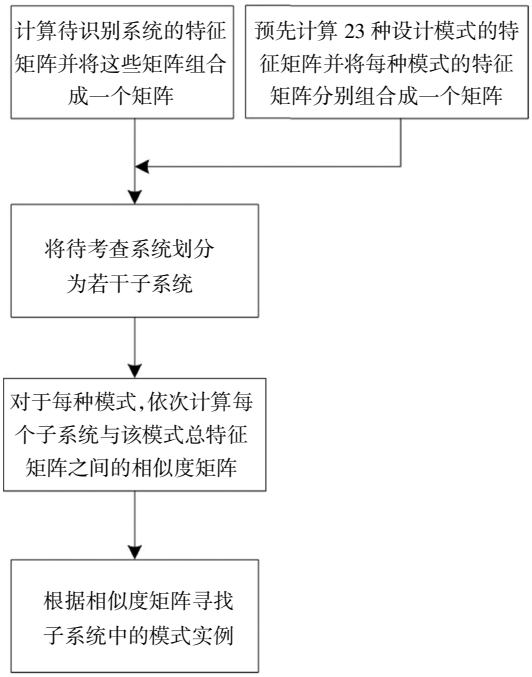

图 1 基本流程

Fig.1 Basic process

2 系统和设计模式的表示

本文使用相似度评分算法计算系统和设计模式之间的相似度矩阵来寻找系统中存在的模式实例,因此需要将系统和设计模式表示为有向图/矩阵形式. 本文考虑关联、泛化、依赖、聚合、抽象类、对象创建、抽象方法调用 7 个特征. 设系统和设计模式的类图为 G,类图 G 中的类为 $c_1,c_2,\cdots,c_n$,下面给出系统和设计模式的矩阵表示的一种形式化定义.

定义 1 类图 G 的关联关系矩阵定义为:

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

其中

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

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 $\boldsymbol{M}_G^{\mathrm{Dep}}$、聚合关系矩阵 $\boldsymbol{M}_G^{\mathrm{Agg}}$ 和抽象方法调用矩阵 $\boldsymbol{M}_G^{\mathrm{Inv}}$.

定义 2 类图 G 的抽象类矩阵定义为:

$$\boldsymbol{M}_G^{\mathrm{Abs}}=\begin{array}{c} \\ c_1\\ c_2\\ \vdots\\ c_j\\ \vdots\\ c_n\end{array}\begin{array}{c}\begin{array}{cccccc} c_1 & c_2 & \cdots & c_j & \cdots & c_n\end{array}\\ \begin{bmatrix} & & & & & \\ & & & & & \\ & & & r_{ij} & & \\ & & & & & \\ & & & & & \end{bmatrix}\end{array} \tag{2}$$

其中

$$r_{ij}=\begin{cases}1,i=j \text{ 且类 } c_i \text{ 为抽象类}\\ 0,\text{其他}\end{cases}$$

定义 3 类图 G 的对象创建矩阵定义为:

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

其中

$$r_{ij}=\begin{cases}1,i=j \text{ 且类 } c_i \text{ 为抽象类}\\ 0,\text{其他}\end{cases}$$

为提高识别的准确率和时间性能,本文借鉴 Dong 等人[7-8]的方法,将所考虑的 7 个特征对应的 7 个有向图/矩阵组合成一个有向图/矩阵. 我们给每个矩阵一个不同素数的根值,然后将每个矩阵的单元值(x)更改为新值,新值为其根(root)的旧单元值次幂 (root^x). 总特征矩阵的每个单元的值为这 7 个新矩阵中相应单元值的乘积. 这里,对于每种模式,我们根据模式中涉及到每种特征的类的个数将 7 个特征进行排序,并按照顺序将 7 个特征的根值分别设为素数 2、3、5、7、9、11 和 13. 例如,对于装饰模式,有 2 个类涉及到关联关系,有 4 个类涉及到泛化关系,有 0 个类涉及到依赖关系和聚合关系,有 3 个类涉及到抽象方法调用,有 2 个类涉及到抽象类,2 个类涉及到对象创建,则将关联、泛化、依赖、聚合、抽象方法调用、抽象类、对象创建的根值分别设为素数 9、13、3、2、11、7 和 5. 由此可得类图 G 关于模式 p 的总特征矩阵定义如下.

定义 4 设 分别为关联、泛化、依赖、聚合、抽象方法调用、抽象类、对象创建的关于 p 的根值,则类图 G 的关于模式 p 的总特征矩阵定义为:

$$
\boldsymbol{M}_{G,p}^{\text{Integ}} = \begin{array}{c} \\ c_1 \\ c_2 \\ \vdots \\ c_j \\ \vdots \\ c_n \end{array} \begin{array}{c} c_1 \quad c_2 \quad \cdots \quad c_j \quad \cdots \quad c_n \\ \left[\begin{array}{ccccc} & & & & \\ & & & & \\ & & r_{ij} & & \\ & & & & \\ & & & & \\ & & & & \end{array} \right] \end{array} \tag{4}
$$

其中

$$
r_{ij} = \text{root}_1^{\boldsymbol{M}_G^{\text{Ass}}[i,j]} \cdot \text{root}_2^{\boldsymbol{M}_G^{\text{Gen}}[i,j]} \cdot \text{root}_3^{\boldsymbol{M}_G^{\text{Dep}}[i,j]} \cdot \text{root}_4^{\boldsymbol{M}_G^{\text{Agg}}[i,j]} \cdot \text{root}_5^{\boldsymbol{M}_G^{\text{Inv}}[i,j]} \cdot \text{root}_6^{\boldsymbol{M}_G^{\text{Abs}}[i,j]} \cdot \text{root}_7^{\boldsymbol{M}_G^{\text{Cre}}[i,j]}
$$

以装饰模式和开源项目 JHotDraw 5.2 为例说明设计模式和系统的表示. 为了便于说明问题,仅考虑 JHotDraw 5.2 的一个子系统(见 3.1 节),记为 s. 装饰模式和子系统 s 的 UML 类图描述分别如图 2 和图 3 所示.

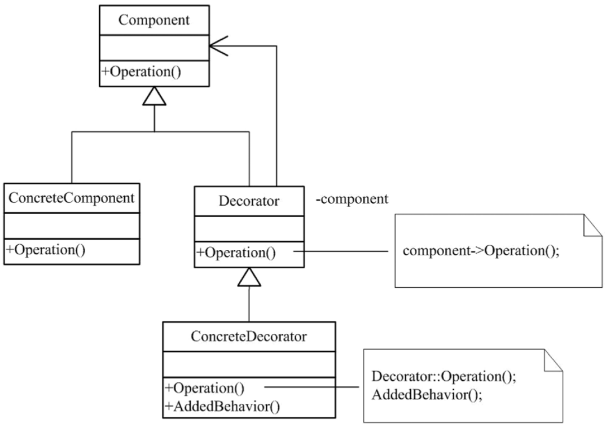

图 2　装饰模式的 UML 类图描述

Fig.2　UML class diagram description of decorator patter

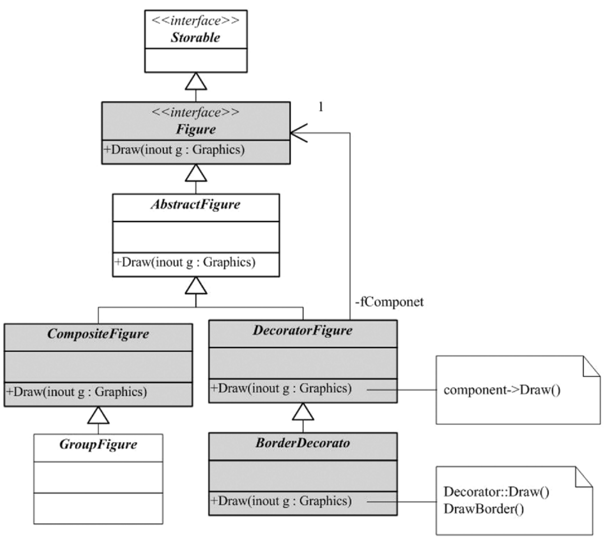

图 3　开源项目 JHotDraw 5.2 的子系统 s

Fig.3　Subsystem s of the open source project JHotDraw 5.2

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:

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Ass}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \end{array} \tag{5}
$$

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Gen}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array} \right] \end{array} \tag{6}
$$

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Dep}} = \boldsymbol{Agg}_{\text{decorator}} = 0 \tag{7}
$$

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Inv}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{array} \right] \end{array} \tag{8}
$$

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Abs}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \end{array} \tag{9}
$$

$$
\boldsymbol{M}_{\text{decorator}}^{\text{Cre}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right] \end{array} \tag{10}
$$

根据总特征矩阵的定义,可得:

$$
\boldsymbol{M}_{\text{decorator},\text{decorator}}^{\text{Integ}} = \begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \left[\begin{array}{cccc} 9^0 13^0 3^0 2^0 11^0 7^1 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 \\ 9^0 13^1 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 \\ 9^1 13^1 3^0 2^0 11^1 7^0 5^1 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^1 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 \\ 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^0 3^0 2^0 11^0 7^0 5^0 & 9^0 13^1 3^0 2^0 11^1 7^0 5^0 & 9^0 13^0 3^0 2^0 11^1 7^0 5^0 \end{array} \right] \end{array} =
$$

$$
\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{array}{c} c_1 \quad c_2 \quad c_3 \quad c_4 \\ \left[\begin{array}{cccc} 7 & 1 & 1 & 1 \\ 13 & 1 & 1 & 1 \\ 6435 & 1 & 7 & 1 \\ 1 & 1 & 143 & 11 \end{array} \right] \end{array} \tag{11}
$$

类似地，记 Storable、Figure、AbstractFigure、CompositeFigure、DecoratorFigure、GroupFigure 和 BorderDecorator 分别为 C_1、C_2、C_3、C_4、C_5、C_6 和 C_7，可以得到子系统 s 的总特征矩阵为

$$\boldsymbol{M}_{s,\text{decorator}}^{\text{Integ}}=\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{array}\begin{array}{c} \begin{array}{ccccccc} C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 \end{array} \\ \begin{bmatrix} 7 & 1 & 1 & 1 & 1 & 1 & 1 \\ 13 & 7 & 1 & 1 & 1 & 1 & 1 \\ 1 & 13 & 7 & 1 & 1 & 1 & 1 \\ 1 & 1 & 13 & 7 & 1 & 1 & 1 \\ 1 & 45\,045 & 13 & 1 & 7 & 1 & 1 \\ 1 & 1 & 1 & 13 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 143 & 1 & 11 \end{bmatrix} \end{array} \tag{12}$$

3 设计模式的自动识别

3.1 子系统的划分

为从系统中搜索模式实例，需要将待考查系统划分为若干子系统. 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，子系统的个数为 $\frac{m(m-1)}{2}$，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.

3.2 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关于该模式的总特征矩阵和设计模式总特征矩阵之间的相似度矩阵.

以装饰模式和子系统 s 为例来说明相似度评分的计算. 经计算可得(函数 Similarity()对应文献[7]第 3.1 节中的相似度评分算法)，子系统 s 和装饰模式之间的相似度矩阵为：

$$\boldsymbol{M}_{s,\text{decorator}}^{\text{Similarity}}=\text{Similarity}(\boldsymbol{M}_{\text{decorator},\text{decorator}}^{\text{Integ}},\boldsymbol{M}_{s,\text{decorator}}^{\text{Integ}})=$$

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{array}\begin{array}{c} \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \end{array} \\ \begin{bmatrix} 0.000\,0 & 0.000\,0 & 0.000\,0 & 0.000\,0 \\ 0.996\,6 & 0.000\,2 & 0.001\,2 & 0.000\,2 \\ 0.000\,3 & 0.000\,0 & 0.000\,3 & 0.000\,0 \\ 0.000\,0 & 0.002\,0 & 0.000\,0 & 0.000\,0 \\ 0.001\,2 & 0.000\,0 & 0.998\,4 & 0.000\,2 \\ 0.000\,0 & 0.000\,0 & 0.000\,0 & 0.000\,0 \\ 0.000\,0 & 0.000\,0 & 0.000\,0 & 0.000\,1 \end{bmatrix} \end{array} \tag{13}$$

3.3 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 $\boldsymbol{M}_{s,p}^{\text{Similarity}}$ 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[29]. 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 $\boldsymbol{M}_{s,p}^{\text{Similarity}}$ 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 $\boldsymbol{M}_{s,p}^{\text{Similarity}}$ 的每列，若至少存在一列中的某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 p；否则不包含.

步骤 2 若包含模式 p，则需要找到该模式的每个角色在子系统 s 中关联的类. 依次考查 $\boldsymbol{M}_{s,p}^{\text{Similarity}}$ 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 $v=0.9$.

这里继续以装饰模式和子系统 s 为例来说明基于相似度矩阵的设计模式识别. 易见 $\boldsymbol{M}_{s,p}^{\text{Similarity}}$ 的第 1

列第 2 行元素的值大于等于匹配临界值 $v = 0.9$,则说明该子系统包含装饰模式实例.

在 c_1 对应的列中,数值最大的元素对应 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_2. 类似地,可以得到子系统中的类 C_4、C_5、C_7 分别关联装饰模式的角色类 c_2、c_3、c_4. 在图 3 中,灰色填充的类关联装饰模式的角色.

4 实验及结果分析

目前本文方法的支撑工具 EasyDetector 1.0 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. 为说明本文方法的有效性,使用文献[26]方法、文献[9]方法和本文方法对开源项目 JHotDraw 5.2、JRefactory 2.6.24 和 JUnit 3.7 进行了设计模式的识别.

表 1~3 和表 4~6 分别列出了文献[26]方法、文献[9]方法和本文方法的支撑工具对 JHotDraw 5.2 进行设计模式识别的识别准确率和 CPU 时间花费.

表 1　JHotDraw 5.2 识别准确率

Tab.1　JHotDraw 5.2 detection accuracy

设计模式	文献[26]方法					文献[9]方法					本文方法				
	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%
适配器命令	16	3	2	84.2	88.9	18	3	0	85.7	100.0	18	3	0	85.7	100.0
组合	1	1	0	50.0	100.0	1	1	0	50.0	100.0	1	1	0	50.0	100.0
装饰者	3	2	0	60.0	100.0	3	1	0	75.0	100.0	3	1	0	75.0	100.0
工厂方法	1	0	2	100.0	33.3	2	0	1	100.0	66.7	2	0	1	100.0	66.7
观察者	5	1	0	83.3	100.0	5	2	0	71.4	100.0	5	1	0	83.3	100.0
原型	1	0	0	100.0	100.0	1	0	0	100.0	100.0	1	0	0	100.0	100.0
单例	0	0	2	0.0	0.0	2	1	0	66.7	100.0	2	0	0	100.0	100.0
状态策略	7	2	16	77.8	30.4	22	3	1	88.0	95.7	23	2	0	92.0	100.0
模板方法	3	0	2	100.0	60.0	5	3	0	62.5	100.0	5	0	0	100.0	100.0
访问者	1	2	0	33.3	100.0	1	1	0	50.0	100.0	1	1	0	50.0	100.0
平均	—	—	—	68.9	71.3	—	—	—	74.9	96.2	—	—	—	83.6	96.7

表 2　JRefactory 2.6.24 识别准确率

Tab.2　JRefactory 2.6.24 detection accuracy

设计模式	文献[26]方法					文献[9]方法					本文方法				
	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%
适配器命令	6	3	1	66.7	85.7	7	2	0	77.8	100.0	7	1	1	87.5	87.5
装饰者	1	0	0	0.0	100.0	1	0	0	100.0	100.0	1	0	0	100.0	100.0
工厂方法	2	2	2	50.0	50.0	1	0	3	100.0	25.0	3	1	1	75.0	75.0
单例	9	1	3	90.0	75.0	12	2	0	85.7	100.0	11	2	1	84.6	91.7
状态策略	10	3	2	76.9	83.3	11	5	1	68.8	91.7	12	1	0	92.3	100.0
模板方法	15	6	2	71.4	88.2	17	13	0	56.7	100.0	16	2	1	88.9	94.1
访问者	1	3	1	25.0	50.0	2	1	0	66.7	100.0	2	1	0	66.7	100.0
平均	—	—	—	54.3	76.0	—	—	—	79.4	88.1	—	—	—	85.0	92.6

表 3 JUnit 3.7 识别准确率

Tab.3 JUnit 3.7 detection accuracy

设计模式	文献[26]方法					文献[9]方法					本文方法				
	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%	TP	FP	FN	Precision/%	Recall/%
适配器命令	1	1	1	50.0	50.0	1	1	0	50.0	100.0	1	1	0	50.0	100.0
组合	1	1	0	50.0	100.0	1	0	0	100.0	100.0	1	0	0	100.0	100.0
装饰者	1	1	0	50.0	100.0	1	1	0	50.0	100.0	1	1	0	50.0	100.0
观察者	3	1	1	75.0	75.0	4	2	0	66.7	100.0	4	1	0	80.0	100.0
状态策略	2	1	1	66.7	66.7	3	2	0	60.0	100.0	3	1	0	75.0	100.0
模板方法	1	3	0	25.0	100.0	1	2	0	33.3	100.0	1	0	0	100.0	100.0
平均	—	—	—	52.8	81.9	—	—	—	60.0	100.0	—	—	—	75.8	100.0

表 4 JHotDraw 5.2 时间花费

Tab.4 JHotDraw 5.2 CPU time cost **s**

方法	预处理	识别	总计
文献[26]	172	8 712	8 884
文献[9]	206	7 452	7 658
本文	211	5 742	5 953

表 5 JRefactory 2.6.24 时间花费

Tab.5 JRefactory 2.6.24 CPU time cost **s**

方法	预处理	识别	总计
文献[26]	686	34 848	35 534
文献[9]	845	28 932	29 777
本文	865	23 542	24 407

表 6 JUnit 3.7 时间花费

Tab.6 JUnit 3.7 CPU time cost **s**

方法	预处理	识别	总计
文献[26]	112	5 663	5774
文献[9]	113	4 686	4 799
本文	127	3 445	3 572

表 1 ~ 表 3 中准确率的评估是基于以下术语进行的:

1) 真阳性 (TP,true positive); 2) 假阳性(FP,false positive);3)假阴性(FN,false negative);4)精确率(precision);5)召回率(recall).

以上术语的定义详见文献[1].

现有方法大多是将所考虑的若干个特征分别进行匹配,识别准确率和时间性能不高.

由表 1、表 2 和表 3 可知,文献[26]方法和文献[9]方法的 JHotDraw 5.2 平均精确率和召回率分别为 68.9%/71.3%、74.9%/96.2%,JRefactory 2.6.24 平均精确率和召回率分别为 54.3%/76.0%、79.4%/88.1%,JUnit 3.7 平均精确率和召回率分别为 52.8%/81.9%、60.0%/100.0%. 而本文方法使用总特征矩阵进行匹配, 使得平均精确率和召回率较之文献[26]方法和文献[9]方法更高,JHotDraw 5.2 平均精确率和召回率达到 83.1%97.9%,JRefactory 2.6.24 达到 85.0%/92.6%,JUnit 3.7 达到 75.8%/100.0%.

由表 4、表 5 和表 6 可知,对于 JHotDraw 5.2,文献[26]方法和文献[9]方法分别花费 8 884ms 和 7 658 ms, 而本文方法仅用 5 953 ms; 对于 JRefactory 2.6.24,文献[26]方法和文献[9]方法分别花费 35 534 ms 和 29 777 ms,而本文方法仅用 24 407 ms;对于 JUnit 3.7,文献[26]方法和文献[9]方法分别 5 774 ms 和 4 799 ms, 而本文方法仅用 3 572 ms. 可以看出,计算总特征矩阵花费了额外的时间, 使得本文方法的预处理阶段花费了更多时间. 然而通过计算总特征矩阵,本文只需要进行一次匹配,使得与文献[26]方法和文献[9]方法相比本文的设计模式识别阶段节省大量时间.

根据以上分析可知, 本文方法可以对设计模式进行识别,且准确率和时间性能更高.

5 结 论

现有设计模式识别方法大多是将所考虑的若干个特征分别进行匹配,限制了其准确率和时间性能. 本文方法将所考虑的 7 个特征对应的矩阵组合成一个总特征矩阵,并使用总特征矩阵进行匹配,改善了识别准确率和时间性能.

目前该方法仍存在一些缺陷和不足. 今后的主要工作如下:

1)目前所考虑的 7 个特征均是静态特征,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)提出的方法目前仅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)本文目前的识别规则均是从设计模式的理论描述中获取的,后期将使用机器学习从实际软件系统中获取识别规则.

参考文献

[1] RASOOL G,STREITFDERT D. A survey on design pattern recovery techniques [J]. International Journal of Computer Science Issues,2011,8(6):251—260.

[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 H B,ZHANG X L,ZHENG X M,*et al*. UML design pattern recognition method based on structured query [J].Computer Science,2014,41(11):50—55. (In Chinese)

[3] YU D,ZHANG Y,CHEN Z. A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures [J]. Journal of Systems & Software,2015,103:1—16.

[4] BERNARDI M L,LUCCA G A D. Model-driven detection of design patterns [C]// IEEE International Conference on Software Maintenance. Timisoara:IEEE Computer Society,2010:1—5.

[5] BeRNARDI M L,CIMITILE M,LUCCA G A D. A model-driven graph-matching approach for design pattern detection [C]// Working Conference on Reverse Engineering. Beverly:IEEE,2013:172—181.

[6] 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.

[7] DONG J,SUN Y,ZHAO Y. Design pattern detection by template matching [C]// The 23rd Annual ACM Symposium on Applied Computing. Fortaleza:DBLP,2008:765—769.

[8] DONG J,ZHAO Y,SUN Y. A matrix-based approach to recovering design patterns [J]. IEEE Transactions on Systems,Man,and Cybernetics - Part A:Systems and Humans,2009,39(6):1271—1282.

[9] TSANTALIS N,CHATZIGEORGIOU A,STEPHANIDES G,*et al*. Design pattern detection using similarity scoring [J]. IEEE Transactions on Software Engineering,2006,32(11):896—909.

[10] 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.

[11] COSTAGLIOLA G,LUCIA A D,DEUFEMIA V,*et al*. Design pattern recovery by visual language parsing [C]// European Conference on Software Maintenance and Reengineering. Manchester:IEEE,2005:102—111.

[12] COSTAGLIOLA G,LUCIA A D,DEUFEMIA V,*et al*. Case studies of visual language based design patterns recovery [C]// European Conference on Software Maintenance & Reengineering. Los Alamitos:IEEE,2006:1—10.

[13] LUCIA A D,DEUFEMIA V,GRAVINO C,*et al*. Behavioral pattern identification through visual language parsing and code instrumentation [C]// European Conference on Software Maintenance & Reengineering. Kaiserslautern:IEEE,2009:99—108.

[14] LUCIA A D,DEUFEMIA V,GRAVINO C,*et al*. Design pattern recovery through visual language parsing and source code analysis [J]. Journal of Systems & Software,2009,82(7):1177—1193.

[15] BALANYI Z,FERENC R. Mining design patterns from C++ source code [C]// Proc International Conference on Software Maintenance. Amsterdam:IEEE,2003:305—314.

[16] BERNARDI M L,CIMITILE M,RUVO G D,*et al*. Improving design patterns finder precision using a model checking approach [C]// The 27th International Conference on Advanced Information Systems Engineering. Stockholm:Springer-Verlag,2015:1—8.

[17] BERNARDI M L,CIMITILE M,RUVO G D,*et al*. Integrating model driven and model checking to mine design patterns [M]. Berlin:Springer International Publishing,2015:1—8.

[18] BERNARDI M L,CIMITILE M,RUVO G D,*et al*. Model checking to improve precision of design pattern instances identification in OO systems [C]// International Joint Conference on Software Technologies. Lisbon:IEEE,2016:53—63.

[19] LUCIA A D,DEUFEMIA V,GRAVINO C,*et al*. Improving behavioral design pattern detection through model checking [C]// European Conference on Software Maintenance and Reengineering. Oldenburg:IEEE,2011:176—185.

[20] WENDEHALS L,ORSO A. Recognizing behavioral patterns at runtime using finite automata [C]// Proceedings of the 2006 International Workshop on Dynamic Analysis. Shanghai:ACM,2006:33—40.

[21] ZHU H,BAYLEY I,SHAN L,*et al*. Tool support for design pattern recognition at model level [C]// The 33rd Annual IEEE International Computer Software and Applications Conference. Seatle:IEEE,2009:228—233.

[22] BAYLEY I,ZHU H. Formalising design patterns in predicate logic [C]// IEEE International Conference on Software Engineering and Formal Methods. IEEE,2007:25—36.

[23] BAYLEY I,ZHU H. Specifying behavioural features of design patterns in first order logic [C]// The 32nd Annual IEEE International Computer Software and Applications Conference. Turku:IEEE Computer Society,2008:203—210.

[24] BAYLEY I,ZHU H . Formal specification of the variants and behavioural features of design patterns [J]. Journal of Systems & Software,2010,83(2):209—221.

[25] 苗康,余啸,赵吉,等. 基于关系演算的 Java 模式识别[J]. 计算机应用研究,2010,27(9):3425—3430.
MIAO K ,YU X ,ZHAO J ,*et al*. Java design pattern recognition based on relational calculus [J]. Application Research of Computers,2010,27(9):3425—3430. (In Chinese)

[26] KIM H,BOLDYREFF C. A method to recover design patterns using software product metrics [C]// International Conference on Software Reuse:Advances in Software Reusability. Vienna:Springer-Verlag,2000:318—335.

[27] HAYASHI S,KATADA J,SAKAMOTO R,*et al*. Design pattern detection by using meta patterns [J]. IEICE Transactions on Information & Systems,2008,E91-D(4):933—944.

[28] LUITEL G,STEPHAN M,INCLEZAN D. Model level design pattern instance detection using answer set programming [C]// International Workshop on Modeling in Software Engineering. Austin:ACM,2016:13—19.

[29] SCANNIELLO G,GRAVINO C,RISI M,*et al*. Documenting design-pattern instances:a family of experiments on source-code comprehensibility [J].ACM Transactions on Software Engineering and Methodology,2015,24(3):1—35.