工程科学学报，第 38 卷，第 10 期：1499－1508，2016 年 10 月
Chinese Journal of Engineering，Vol. 38，No. 10：1499－1508，October 2016
DOI：10.13374/j.issn2095－9389.2016.10.020；http://journals.ustb.edu.cn

一种形式化上下无关文法关系驱动的设计模式检测方法

肖卓宇[1)]✉，何　锫[2,3,4)]，余　波[1)]，黎　妍[5)]，杨鑫维[1)]

1）中南林业科技大学涉外学院，长沙 410200　2）广州大学计算机科学与教育软件学院，广州 510006
3）北京大学高可信软件技术教育部重点实验室，北京 100871　4）长沙理工大学计算机与通信工程学院，长沙 410114
5）湖南省高速公路管理局，长沙 410209
✉ 通信作者，E-mail：xzyxzy0770@126.com

摘　要　针对设计模式识别结果的假阴性问题与重叠问题，为提高设计模式实例恢复的精确性，提出一种形式化上下无关文法关系驱动的设计模式检测方法．依据设计模式实例中的参与者属性及其关系，以形式化可视化语言描述模式实例的识别文法．在此基础上，改进该文法检测设计模式实例参与者间的附加关系，并识别共享实例的模式．实验结果表明，新方法不仅减少了模式实例的假阴性结果，还解决了模式实例识别的重叠问题，与其他检测方法的精确度、召回率及 F-score 指标比较，新方法取得了较好的效果．

关键词　模式识别；设计模式；检测；形式化文法
分类号　TP311

An approach for design pattern detection based on the formal context-free grammar relation driver

XIAO Zhuo-yu[1)]✉，*HE Pei*[2,3,4)]，*YU Bo*[1)]，*LI Yan*[5)]，*YANG Xin-wei*[1)]

1）Swan College，Central South University of Forestry and Technology，Changsha 410200，China
2）School of Computer Science & Education Software，Guangzhou University，Guangzhou 510006，China
3）Key Laboratory of High Confidence Software Technologies of the Ministry of Education，Peking University，Beijing 100871，China
4）School of Computer and Communication Engineering，Changsha University of Science and Technology，Changsha 410114，China
5）Hunan Highway Administration Bureau，Changsha 410209，China
✉ Corresponding author，E-mail：xzyxzy0770@126.com

ABSTRACT　Aiming at the false negative problem and the overlap problem in pattern instance detection，in order to improve the accuracy of the design pattern recovery，this article introduces an approach for design pattern detection based on the formal context-free grammar relation driver. Focusing on the attribute and relationship of classes in pattern instances，the formal grammar of pattern instance identification is established using the visual language，and an improved formalism grammar is presented for identifying the additional relationships and the sharing problem of design pattern instances. Experimental results show that，compared with other well-known algorithms by precision，recall and F-score，the proposed method can reduce the false negative results and the overlap problem in pattern instance detection，indicating the effectiveness of the proposed method.

KEY WORDS　pattern recognition；design patterns；detection；formal grammars

收稿日期：2015－12－25
基金项目：国家自然科学基金资助项目（61170199）；湖南省大学生研究性学习和创新性实验计划资助项目（湘教通［2015］84 号 197）；湖南省教学改革研究立项资助项目（湘教通［2016］400 号 1068）；广东省自然科学基金资助项目（2015A030313501）；湖南省教育厅重点资助项目（11A004）；广东省普通高校创新团队建设资助项目（2015KCXTD014）；中南林业科技大学教学改革研究资助项目（ZNLJG2016－A067）

设计模式封装了解决重复设计问题的有效方案[1]；而设计模式识别有利于软件系统的理解与维护，并为重构软件项目提供了支持. 目前国内外研究人员在设计模式检测的工具、方法、技术等方面做了大量的研究工作[2−4]. 文献[5−6]中提出一种基于子模式及方法签名的设计模式实例恢复方法，其思路是将设计模式中的类参与者以图形结点的形式表示出来，并将图形结点间的关系表示为子图，最终通过子图匹配来识别设计模式实例. Fontana 等[7−8]将设计模式参与者信息及其关系分类成不同的 micro-structures 结构，从而对设计模式实例进行恢复，文献[9−11]使用机器学习对特征信息进行训练，并通过各种聚类方法提高设计模式识别的精度. 文献[12−13]引入子图同构的思想，以邻接矩阵的形式来识别类关系图之间的相似性，从而挖掘设计模式实例. 文献[14]提出对设计模式中主要角色的特征属性进行注释，一定程度上提高了设计模式实例的精确率. 文献[15]通过语义分析对设计模式中的结构型与行为型模式进行检测，依据模式命名的原则，将参与者映射到实例中的主要角色，并加以注释，从而减少假阴性结果. 文献[16]使用抽象的形式化语言来表示设计模式，其文法描述和代码过于分离，以至于模式实例很难被自动识别. 文献[17]提出一种自适应特征信息分类的设计模式恢复方法，该方法对特征信息共享而导致的设计模式实例重叠问题缺乏考虑. Guéhéneuc 和 Antoniol[18]提出依据源码抽取类关系的不同表示形式，使用不同的参数值来比较设计模式实例指标，该方法采用多阶段过滤，从而缩小搜索空间，进而减少了设计模式检测的复杂度. Pettersson 等[19]在评估真阳性(true positives)、假阳性(false positives)、假阴性(false negatives)等参数指标的基础上提出一种更精确的设计模式实例评估指标 F-score. 文献[20]对设计模式角色间存在的附加关系进行了分析，并制定了附加关系检测规则.

综上所述，现有设计模式实例恢复方法存在以下主要问题：(1)设计模式实例恢复由于算法过于抽象，导致识别的召回率、精确度等结果不够理想；(2)对设计模式实例参与者之间存在的附加关系缺乏考虑，从而导致大量假阴性结果出现；(3)在识别设计模式实例过程中，未考虑到一个参与者可能同时参与了多个设计模式实例，从而导致设计模式实例识别的重叠问题. 为此，提出一种形式化上下无关文法关系驱动的设计模式检测方法，旨在将软件项目中的主要特征抽取后，将其结果表示为文献[21]中描述的可视化字符符号及关系句型符号等，并结合文献[22]中提出的形式化上下无关文法描述了参与者符号间存在的关系，同时在文献[20]制定的基于设计模式角色的附加关系检测规则基础上，改进一种致力于发现该附加关系的形式化文法(additional relationships formal grammars，ARFGS)，从而进一步避免设计模式实例识别过程中出现的假阴性结果，再进一步扩展与优化 ARFGS 文法，解决了参与者共享实例而导致设计模式实例识别的重叠问题，并通过六个开源项目对本研究方法的识别效果进行了评估实验.

1　基于可视化符号的模式实例结构分析

文献[21]提出了可视化语言的概念，可视化语言由一组可视化字符符号与句型符号组成，是一种具有关系类型名及句型属性的图形化对象，每个图形化的对象都具有其相关的属性，并且符号类型名可以通过基于字符串的可视化符号来定义，而句型属性表示可视化符号间的关系. 每个可视化符号 S_i 的属性可进行编号，其值表示为 S_i. attrtibute [n]，S_i 表示第 i 个可视化符号，n 表示符号的属性数 $n \in \{1,2,\cdots,N\}$. 句型属性表示为可视化符号彼此联系的连接区域或连接点.

图 1 中，Composite 模式中 Component、Leaf、Composite 类及 Inheritance、Aggregation 等关系皆可表示为不同的图形符号，而连接点作为句型属性对应两个不同的端点符号. 图 2 在图 1 基础上强调了句型属性. 其中圆角矩形表示 CLASS 类型符号，ClassA、ClassB 等表示符号名，其相应的连接区域在圆角矩形中用数字 1 标注，空心圆点表示句型属性的连接点，用 1、2 等数字表示，连接区域通过连接点联系起来. 例如：图 2 中 ClassB 与 Inheritance 符号之间存在一个连接区域 1 及空心连接点 1，符号属性 a 表示描述了二者间的连接关系点，而符号属性 b 描述了 Inheritance 符号中连接点 2 与 ClassA 中连接区域 1 的连接关系点. 表 1 描述了图 2 中各个符号的属性列表. 其中 Symbol_i 表示第 i 个符号名，Attrtibute [j] 表示符号的第 j 个属性.

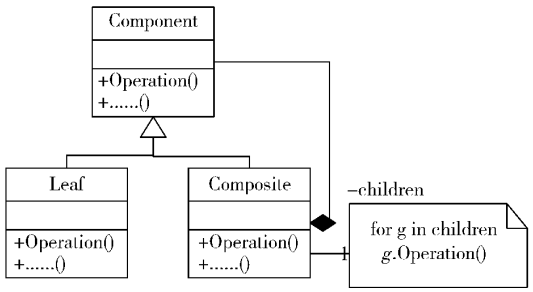

图 1　Composite 模式

Fig. 1　Composite pattern

文献[22]将图形表示为一个文本句子，通过交替符号及属性来定义符号间的关系. 在此基础上，笔者对其符号及其关系以形式化文法的形式进行了扩展.

定义 1　SymbolA，$\text{Connect}_{i,j}$，SymbolB.

$\text{Connect}_{i,j}$ 表示符号 SymbolA 的连接区域或连接点

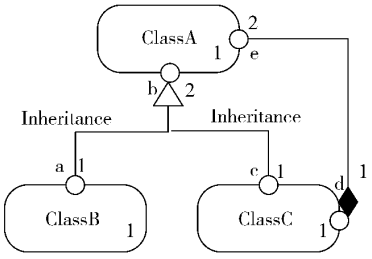

图 2 Composite 模式句型属性

Fig. 2 Syntactic attributes of Composite pattern

表 1 符号属性表

Table 1 Symbol attribute table

$Symbol_i$	Attrtibute [1]	Attrtibute [2]
ClassA	[b]	[e]
ClassB	[a]	
ClassC	[c]	[d]
Aggregation	[d]	[e]
Inheritance	[a]	[b]
Inheritance	[c]	[b]

i 与 SymbolB 的连接区域或连接点 j 存在联系. 例如: 图 2 中 ClassA 与 ClassB 符号的关系可表示为

$$\text{ClassA Connect}_{1,2}\ \text{Inheritance Connect}_{1,1}\ \text{ClassB}. \quad (1)$$

式中 $\text{Connect}_{1,2}$ 表示符号 ClassA 的连接区域 1 与符号 Inheritance 的连接点 2 存在联系,而 Inheritance 的连接点 1 与 ClassB 的连接区域 1 通过 $\text{Connect}_{1,1}$ 连接. 值得注意的是符号与符号的连接存在多样性,如 ClassC 的连接区域 1 不仅与符号 Inheritance 的连接点 1 有联系,同时也与 Aggregation 符号的连接点 1 存在联系,为解决这个问题,给出定义 2.

定义 2 $\text{SymbolA Connect}^n_{i,j}\ \text{SymbolB}$.

定义 2 中 $\text{Connect}^n_{i,j}$ 的指数上标 n 用来区分符号间不同的关系. 例如: ClassC 符号与 Inheritance 符号可表示为

$$\text{ClassC Connect}_{1,1}\ \text{Inheritance}. \quad (2)$$

ClassC 符号不仅与 Inheritance 符号存在关系,也与 Aggregation 符号有关联,见下式:

$$\text{ClassC Connect}^2_{1,1}\ \text{Aggregation}. \quad (3)$$

式中 $\text{Connect}^2_{1,1}$ 表示 ClassC 符号的连接区域 1 与 Aggregation 符号的连接点 1 存在联系. 这样能够处理与式(2)的冲突. 为了表示符号之间的关系集合,给出定义 3.

定义 3 $\text{SymbolA}\ \{\text{Connect}^n_{i,j}\}\ \text{SymbolB}$.

定义 3 表示符号 SymbolA 的连接区域或连接点 i 与 SymbolB 连接区域或连接点 j 之间存在多种关系的集合,n 表示符号间关系数 $n \in \{1,2,\cdots,N\}$.

定义 4 $\text{SymbolA} \neg \text{Connect}^n_{i,j}\ \text{SymbolB}$.

$\text{Connect}^n_{i,j}$ 表示符号 SymbolA 的连接区域或连接点 i 和 SymbolB 的连接区域或连接点 j 之间不存在联系,n 表示不同符号间的关系,$n \in \{1,2,\cdots,N\}$. 依据上述定义,图 2 的 Composite 模式可描述为:

$$\text{ClassA Connect}_{1,2}\ \text{Inheritance Connect}_{1,1}\ \text{ClassB Connect}^2_{1,2}\ \text{Inheritance Connect}_{1,1}\ \text{ClassC Connect}^2_{1,1}\ \text{Aggregation}. \quad (4)$$

图 2 中 ClassA 符号表示为圆角矩形,而圆角矩形内的 1 表示该符号的连接区域,连接区域附近的空心圆点 2 表示该符号的连接点,依据定义 1,式(4)中 ClassA 符号的连接区域 1 与表 1 中符号属性点 [b] 处的连接点 2 表示为 $\text{Connect}_{1,2}$,可与 Inheritance 符号进行连接,同理,Inheritance 符号通过 $\text{Connect}_{1,1}$ 连接符号 ClassB. 考虑到不只 ClassB 符号与 Inheritance 存在联系,依据定义 2,ClassC 与 Inheritance 的联系表示为 $\text{Connect}^2_{1,2}$,而 ClassC 与 ClassA 之间除继承关系外还存在聚合关系. 符号 ClassC 与符号 Aggregation 之间通过 $\text{Connect}^2_{1,1}$ 表示,并在表 1 中的 ClassC 符号属性点 [d] 处进行连接.

基于上述定义,引入一种类似于上下文无关文法的形式化语法. 该语法通过产生式来交替终结符符号与非终结符符号,并结合文献 [22] 中提出的符号属性来描述类图. 图 2 中的 Composite 设计模式可描述如下:

$$\text{Composite\_pattern} \Rightarrow \text{Component Connect}_{1,2}\ \text{Inheritance Connect}_{1,1}\ \text{Leaf Connect}^2_{1,2}\ \text{Inheritance Connect}_{1,1}\ \text{Composite Connect}^2_{1,1}\ \text{Aggregation}. \quad (5)$$

$$\text{Component} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Component}_i = \text{CLASS}_i\}. \quad (6)$$

$$\text{Leaf} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Leaf}_i = \text{CLASS}_i\}. \quad (7)$$

$$\text{Composite} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Composite}_i = \text{CLASS}_i\}. \quad (8)$$

式(5)给出了 Composite 模式的形式化语法实现,式中非终结符 Component、Leaf 和 Composite 符号通过 Inheritance、Aggregation 等终结符进行了可视化字符串的连接. 式(6)~式(8)中,Δ 规则用来关联产生式左侧的非终结符与右侧终结符的初始化属性值,例如: 式(6)中符号 Component 的属性 i 关联了 CLASS 属性 i 的值. 使用基于上下无关文法的产生式允许扩展经典的 LR 分析技术及一些经典的自动文法生成工具,克服了文献 [22] 中提出的可视化文法低效性问题. 事实上,本研究提出的基于形式化上下无关文法关系驱动的方法集合了表 1 中不同符号存储的属性值. 图 3 描述了 Composite 模式实例的识别过程. 通过式(5)~(8)的定义,使用 Δ 规则来将产生式左侧的非终结符与右侧终结符进行映射,并给出符合规范的输入符号及句型属性.

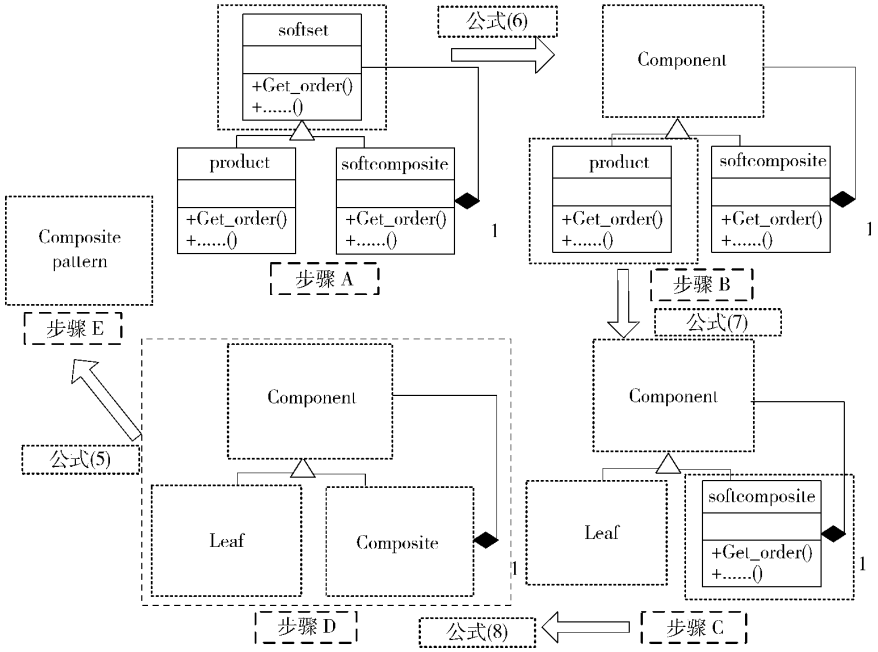

图 3　Composite 模式识别步骤

Fig. 3　Recognition step for Composite pattern

图 3 的步骤 A 中，虚线框表示要通过产生式处理的符号，语法分析器开始搜寻充当 Component 角色的 CLASS 符号. 因此，依据 Δ 规则，并应用式(6)，CLASS 符号被替换为非终结符 Component，步骤 B 描述了替换后的结果. 然后，语法分析器通过 $\text{Connect}_{1,2}$ 搜寻连接 Component 符号的 Inheritance 符号. 同理，使用式(7)，步骤 B 中虚线框对应的 CLASS 符号替换为非终结符 Leaf，再使用式(8)，步骤 C 中虚线框对应的 CLASS 符号替换为非终结符 Composite，步骤 D 阶段之后，应用式(5)搜寻各符号属性间的关系，并发现了一个完整的 Composite 模式实例，见步骤 E.

2　设计模式参与者符号间的附加关系

第 1 节中，形式化文法一定程度上描述了符号间的关系，但对其附加关系缺乏考虑[20]. 为解决这个问题，提出一种检测类符号间附加关系的形式化文法(additional relationships formal grammars，ARFGS)，该文法涉及符号间关系处理的机制，并能在文献[22]基础上构建更为严谨的文法. ARFGS 制定了符号间附加关系的形式化文法，从而避免设计模式检测时假阴性实例被识别后导致的结果误差. 图 4 中虚线为 Composite 模式符号间的附加关系，为识别附加关系制定下述 2 条规则.

规则 1　Leaf 与 Composite 类之间不存在继承(Inheritance)关系.

规则 2　Leaf 与 Composite 类之间不存在关联(Associate)关系.

为执行检测规则 1 与规则 2，ARFGS 形式化文法

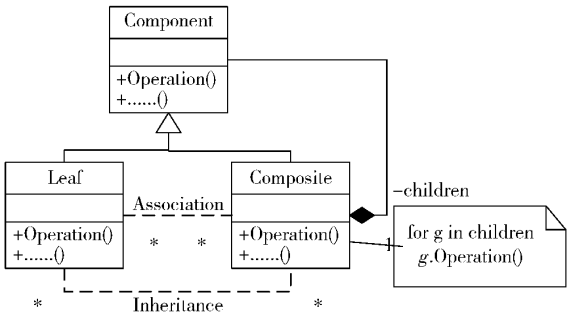

图 4　Composite 模式附加关系

Fig. 4　Additional relationships of Composite pattern

通过增加产生式规则来提升识别附加关系的能力. 依据式(11)与式(12)，如符号间关系满足附加关系标准，则 Composite 模式实例识别时，其附加关系标志 access 将被标记为 1，表示已访问，从而在后续识别过程中不再考虑该符号关系的检索. 为此，将式(9)中关联其他属性的 CLASS 符号标识为 Tagging，并跟踪已被文法分析的符号，Tagging 初值设置为 0.

$$\begin{aligned}&\text{Composite\_pattern}\Rightarrow\\&\text{Component Connect}_{1,2}\ \text{Inheritance Connect}_{1,1}\\&\text{Leaf Connect}^{2}_{1,2}\ \text{Inheritance}\{\text{Connect}_{1,1},\neg\ \text{Tagging}\}\\&\text{Composite}\{\text{Connect}_{1,1},\text{Connect}^{3}_{1,2}\}\ \text{Aggregation}.\end{aligned}\tag{9}$$

$$\text{Component}\Rightarrow\text{CLASS}\ \Delta:\{\text{Component}_i=\text{CLASS}_i\}.\tag{10}$$

$$\begin{aligned}&\text{Leaf}\Rightarrow\text{Leaf}'\ \text{Connect}_{1,2}\ \text{Inheritance Connect}_{1,1}\\&\text{CLASS}\ \Delta:\{\text{Leaf}_1=\text{Leaf}'_1\}\end{aligned}$$

$$\Gamma:\{(\mathrm{CLASS}':\ \mathrm{CLASS}'_1=\mathrm{CLASS}_1;\ \mathrm{CLASS}'_{\mathrm{access}}=1)\}. \tag{11}$$

$$\mathrm{Leaf}\Rightarrow\mathrm{Leaf}\ \mathrm{Connect}_{1,2}\ \mathrm{Association}\ \mathrm{Connect}_{1,1}\ \mathrm{CLASS}\ \Delta:\{\mathrm{Leaf}_1=\mathrm{Leaf}'_1\}$$

$$\Gamma:\{(\mathrm{CLASS}':\ \mathrm{CLASS}'_1=\mathrm{CLASS}_1;\ \mathrm{CLASS}'_{\mathrm{access}}=1)\}. \tag{12}$$

$$\mathrm{Leaf}\Rightarrow\mathrm{CLASS}\ \Delta:\{\mathrm{Leaf}_i=\mathrm{CLASS}_i\}. \tag{13}$$

$$\mathrm{Composite}\Rightarrow\mathrm{CLASS}\ \Delta:\{\mathrm{Composite}_i=\mathrm{CLASS}_i\}. \tag{14}$$

本研究在文献[21]提出的Γ规则上,引入新的终结符符号描述上述产生式. 式(11)描述Leaf符号的属性1与Leaf′符号的属性1在识别过程中存在映射关系,并具有相同的值. 另一个新的符号CLASS′被引入,它继承了CLASS符号属性1的值,其附加关系标志access置为1,所有的CLASS联系Leaf需通过一个Inheritance关系. 相似的产生式(12)中CLASS′符号的附加关系标志access也设置为1,所有的CLASS联系Leaf需通过一个Association关系,避免下次识别过程再次检测该附加关系,从而很大程度上减低了假阴性结果. 如access属性值为0,则产生式(9)中的Tagging无需标记,这意味着违反规则1和规则2定义的附加关系检测规则.

以图5为例,该图是Jhotdraw开源系统中的类关系图,依据产生式(9)~式(14),该实例不能被识别为Composite模式,因为Composite Figure类中的方法draw(*g*)被Figure类的方法draw(*g*)代理,这是一个典型的附加关系,违反了规则2,依据式(9),Figure类不能被Composite模式中的Leaf类替换.

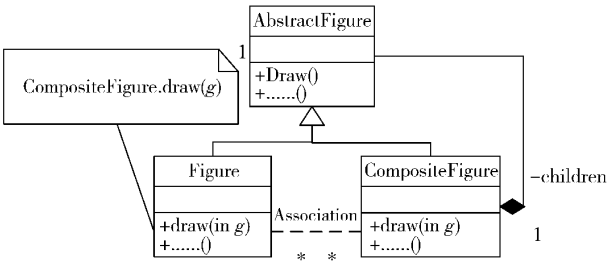

图5 类图违反规则2

Fig. 5 Class diagram violating negative Rule 2

3 识别重叠的设计模式实例

在识别设计模式实例过程中,一个类可能参与多个设计模式实例,可理解为类参与者由于共享模式实例而导致的实例重叠问题. 这类问题产生原因是因为通过随机搜寻类图中的CLASS符号作为文法的开始符号,当匹配到某种设计模式实例时,即认为识别成功,而忽略了CLASS符号的共享问题,这样的方法不能保证所有模式实例识别的成功.

图6中两个Composite模式实例共享了类符号CLASSA与CLASSB,但如文法开始符号选定为CLASSC,通过第2节中定义的产生式搜寻CLASSA与CLASSB符号及其三者之间的关系,左侧虚线框中的Composite模式实例1将被识别出来,这样的方法并没有考虑到CLASSA与CLASSB符号是可以共享的. 事实上,如果开始符号选定的不是CLASSC而是CLASSC',再次通过第2节中定义的产生式搜寻CLASSA与CLASSB符号及其三者之间的关系,则最终被识别出来的是右侧虚线框中的Composite模式实例2.

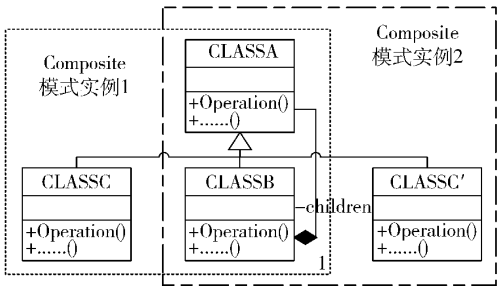

图6 Composite模式重叠

Fig. 6 Overlapped composite pattern

CLASS符号及其关系的共享不限于相同设计模式实例的重叠,同样也适用于不同设计模式实例的重叠. 图7是JhotDraw系统中的类关系图. 其Composite模式与Decorator模式存在重叠,Composite模式实例与Decorator模式实例共享了类符号Abstract Figure、Composite Figure和Decorator Figure. 但如果开始符号选定为Border Decorator,通过第2节中定义的产生式搜寻Scroll Decorator、Abstract Figure、Composite Figure和Decorator Figure符号及其关系,外侧虚线框中的Decorator模式实例将被识别出来,但这样的做法并没有考虑到Abstract Figure、Composite Figure和Decorator Figure是可以共享的,其直接结果将导致内侧虚线框中的Composite模式实例不能被识别.

解决上述问题首先需确定选择哪个CLASS符号作为开始符号,并依据文法对收索结果依次进行记录. 该方法要求每个CLASS符号都至少有一次被选定为产生式的开始符号,当一个类参与者担任多个设计模式实例的起始符至少一次时,可解决符号共享导致的实例重叠问题. 图6中CLASSA类共享了两个Composite模式实例,可以考虑作为起始符两次. 为了不影响文中1、2节得出的结果,上述问题将扩展ARFGS形式化文法来解决. 为此,引入递归思想来解决共享问题,并通过指定下述文法的产生式来解决.

$$\mathrm{CompositePattern}\Rightarrow\mathrm{Components}. \tag{15}$$

$$\mathrm{Components}\Rightarrow\mathrm{Components}\ \mathrm{Connect}_{1,2}\ \mathrm{Inheritance}$$

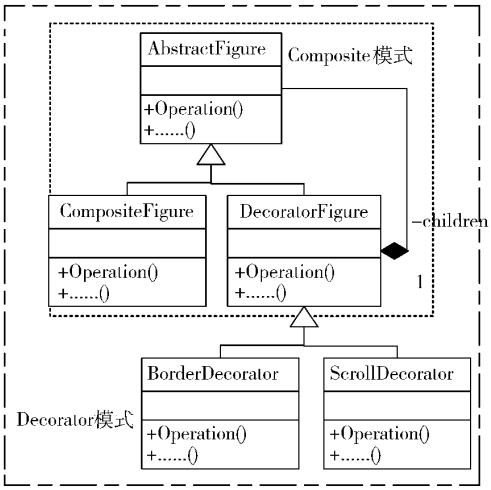

图 7　Composite 模式与 Decorator 模式重叠

Fig. 7　Overlapping between Composite and Decorator pattern

$$\text{Connect}_{1,1}\ \text{Composites}\ \Delta\text{:}\{\text{Components}_1 = \text{Components}_1\}. \tag{16}$$

$$\text{Components} \Rightarrow \text{Component}\ \text{Connect}_{1,2}\ \text{Inheritance}\ \text{Connect}_{1,1}\ \text{Composites}\ \Delta\text{:}\{\text{Components}_1 = \text{Component}_1\}. \tag{17}$$

$$\text{Component} \Rightarrow \text{Component}'\ \text{Connect}_{1,2}\ \text{Aggregation}\ \text{Connect}_{1,1}\ \text{CLASS}\ \Delta\text{:}\{\text{Component}_1 = \text{Component}_1'\}\ \Gamma\text{:}\{(\text{CLASS}'\text{:}\ \text{CLASS}_1' = \text{CLASS}_1)\}. \tag{18}$$

$$\text{Component} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Component}_1 = \text{CLASS}_1\}. \tag{19}$$

$$\text{Component} \Rightarrow \text{Components}\ \text{Connect}_{1,2}\ \text{Inheritance}\ \text{Connect}_{1,1}\ \text{Leaf}\ \Delta\text{:}\{\text{Components}_1 = \text{Components}_1\}. \tag{20}$$

$$\text{Composites} \Rightarrow \text{Composites}'\ \text{Connect}_{1,1}\ \text{Inheritance}\ \{\text{Connect}_{1,1}^2, \neg\ \text{Tagging}\}\ \text{Leaf}\ \Delta\text{:}\{\text{Composites}_1 = \text{Composites}_1'\}\ \Gamma\text{:}\{(\text{CLASS:}\ \text{CLASS}_1 = \text{Leaf}_1; \text{CLASS}_{\text{access}} = 1)\}. \tag{21}$$

$$\text{Composites} \Rightarrow \text{Composites}\ \text{Connect}_{1,1}\ \text{Association}\ \{\text{Connect}_{1,1}^2, \neg\ \text{Tagging}\}\ \text{Leaf}\ \Delta\text{:}\{\text{Composites}_1 = \text{Composites}_1\}\ \Gamma\text{:}\{(\text{CLASS}:\ \text{CLASS}_1 = \text{Leaf}_1; \text{CLASS}_{\text{access}} = 1)\}. \tag{22}$$

$$\text{Composite} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Composite}_1 = \text{CLASS}_1\}. \tag{23}$$

$$\text{Leaf} \Rightarrow \text{CLASS}\ \Delta\text{:}\{\text{Leaf}_1 = \text{CLASS}_1\}. \tag{24}$$

式(15)考虑了一个类符号可以参与多个模式实例识别的问题，式(16) ~式(20)通过对开始符号引入递归的思想来担任多个 Composite 模式实例的 Component 及 Composite 角色. 式(16)与式(17)允许 Component 作为一个开始符号遍历所有可能存在继承关系的 Composite 符号. 式(18)与式(19)描述了能与 Composite 符号之间存在 Aggregation 关系的 CLASS 符号. 式(20)描述了 Component 作为一个开始符遍历所有可能存在继承关系的 Leaf 符号，结合图 6，并依据式(17)，Component 与 Leaf 的继承关系不存在重叠. 式(21)与式(22)目的是避免 CLASS 符号间引入 Inheritance 与 Association 附加关系.

图 8 描述了存在重叠的 Composite 模式实例识别过程，图中有三个重叠的 Composite 模式实例，其中 CLASSA 可替换为 Component，CLASSB 可替换为 Composite，而 CLASSC、CLASSC′和 CLASSC″可替换为 Leaf，依据式(15) ~式(24)，识别过程的起始符定义为 CLASSA，而被识别出的 Composite 模式实例用虚线框表示. 步骤 A、步骤 B 和步骤 C 依次识别出 Composite 模式实例 1、Composite 模式实例 2 和 Composite 模式实例 3，而在步骤 D 全部 3 个 Composite 模式实例皆被成功检测出来.

4　设计模式实例识别步骤

设计模式实例识别包括五个主要步骤，见图 9.

步骤 1　通过逆向工程工具 F. T. [17] 抽取特征信息.

步骤 2　在步骤 1 基础上对可视化符号进行分析.

步骤 3　在步骤 2 基础上，形式化描述设计模式实例.

步骤 4　对步骤 3 结果进行优化，制定设计模式参与者间的附加关系检测规则.

步骤 5　进一步恢复重叠的设计模式实例.

5　实验设计与评估

为评估本研究的有效性，选用 JhotDraw 等六个开源系统进行了设计模式实例识别实验. 表 2 展示了这些开源系统的特征参数，执行实验的操作系统采用 WINDOW7，CPU 为 INTEL Core2 G630 2.7 GHz，内存 8 G.

表 2　开源系统特征参数

Table 2　Characteristic parameters of the open source system

开源项目名称	类的数目	千行代码的数目
JhotDraw 6.0b1	300	19
Dom4j 1.6.1	179	18
Junit v3.8	56	4
Apache ant v1.6.2	79566	895
Quick UML2001	8792	203
JavaAWT 5.0	345	56

5.1　指标评估实验

实验评估采用精确度 Precision (P)、召回率 Recall (R)、正确性 Accuracy (A)、F-score 等指标. 值得注意的是当待识别系统规模较大时，假阳性 FP (false posi-

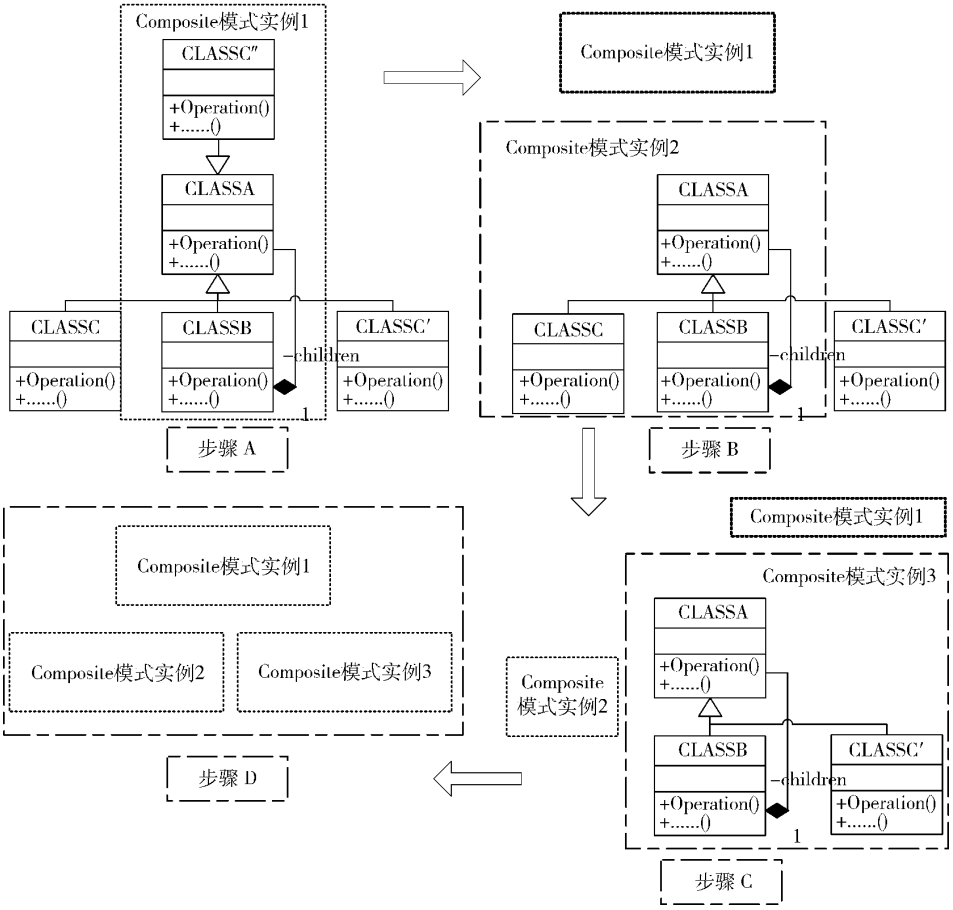

图 8　重叠的 Composite 模式识别过程

Fig. 8　Recognition process of overlapped Composite pattern

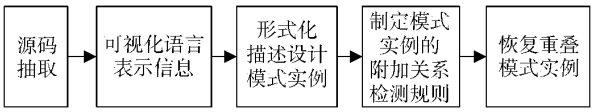

图 9　设计模式实例识别流程

Fig. 9　Design pattern instance identification flow chart

tive)、假阴性 FN (false negative) 的识别结果容易出现偏差. 此外,精确度和召回率指标之间的关系确定了方法的实用与否,而精确度和召回率在进行设计模式检测时依赖于四个重要指标,即真阳性(true positive, TP)、假阳性(false positive, FP)、真阴性(true negative, TN)和假阴性(false negative, FN).

(1) 真阳性 TP: 表示正确实例识别为正确.

(2) 假阳性 FP: 表示错误实例识别为正确.

(3) 真阴性 TN: 表示错误实例识别为错误.

(4) 假阴性 FN: 表示正确实例识别为错误.

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}. \tag{25}$$

$$R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}. \tag{26}$$

$$A = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{FP} + \mathrm{TN} + \mathrm{FN}} \tag{27}$$

式(25)中 P 表示精确度,式(26)中 R 表示召回率,式(27)中 A 表示正确性.

依据表 3,JhotDraw 6.0b1 中 State、Template、Visitor、Singleton 和 Adapter 五个模式存在假阴性 FN,从而导致召回率 R 与正确性 A 受到影响. 表 3 Apache ant v1.6.2 系统中 State、Singleton、Factory method、Bridge、Visitor 和 Adapter 六个模式存在假阴性 FN. 表 4 中 Junit 3.8 仅有 Adapter 一个模式存在假阴性 FN. Apache ant v1.6.2 中 Factory method、Bridge 和 Visitor 模式也存在假阴性 FN(false negative). 表 5 中 Quick UML2001 系统的 State 与 Adapter 模式存在假阴性 FN, Java AWT 5.0 中只有 Decorator 模式不存在假阴性 FN.

综合表 3 ~ 表 5 数据分析发现以下两个显著的问题.

(1) 六个测试系统中,State 模式的假阴性结果 FN 在其中四个开源系统都存在,由于另外两个测试系统 Junit 3.8 和 Apache ant v1.6.2 未识别到 State 模式实例. 统计发现,在已识别出 State 模式实例存在假阴性结果 FN 的概率为 100%. State 模式在表 3 JhotDraw 6.0b1 与 Dom4j 1.6.1 系统的召回率 R 分别为 80% 与

表 3　Jhot Draw 6.0b1 与 Dom4j 1.6.1 设计模式实例识别结果

Table 3　Identification results of design patterns in Jhot Draw 6.0b1 and Dom4j 1.6.1

设计模式名	JhotDraw 6.0b1							Dom4j 1.6.1						
	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%
Composite	28	0	0	0	100	100	100	57	0	0	0	100	100	100
State	4	0	0	1	100	80	80	30	0	0	2	100	93.8	93.8
Template	125	0	0	2	100	98.4	98.4	41	0	0	0	100	100	100
Decorator	28	0	0	0	100	100	100	11	0	0	0	100	100	100
Singleton	6	0	0	1	100	85.7	85.7	32	0	0	4	100	88.9	88.9
Factory method	45	0	0	0	100	100	100	53	0	0	1	100	98.1	98.1
Bridge	—	—	—	—	—	—	—	41	0	0	2	100	95.3	95.3
Visitor	4	0	0	1	100	80	80	12	0	0	1	100	92.3	92.3
Adapter	35	0	0	3	100	92.1	92.1	65	0	0	2	100	97.0	97.0
Prototype	3	0	0	0	100	100	100	16	0	0	0	100	100	100
平均统计					100	92.4	92.4					100	96.5	96.5

注：“—”表示该模式在系统中不存在.

表 4　Junit v3.8 与 Apache ant v1.6.2 设计模式实例识别结果

Table 4　Identification results of design patterns in Junit v3.8 and Apache ant v1.6.2

设计模式名	Junit 3.8							Apache ant v1.6.2						
	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%
Composite	4	0	0	0	100	100	100	11	0	0	0	100	100	100
State	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Template	2	0	0	0	100	100	100	6	0	0	0	100	100	100
Decorator	6	0	0	0	100	100	100	14	0	0	0	100	100	100
Singleton	—	—	—	—	—	—	—	7	0	0	2	100	77.8	77.8
Factory method	4	0	0	0	100	100	100	38	0	0	0	100	100	100
Bridge	—	—	—	—	—	—	—	157	0	0	3	100	98.1	98.1
Visitor	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Adapter	28	0	0	2	100	93.3	93.3	145	0	0	4	100	97.3	97.3
Prototype	—	—	—	—	—	—	—	—	—	—	—	—	—	—
平均统计					100	98.3	98.3					100	96.2	96.2

注：“—”表示该模式在系统中不存在.

表 5　Quick UML2001 与 Java AWT 5.0 设计模式实例识别结果

Table 5　Identification results of design patterns in Quick UML2001 and Java AWT 5.0

设计模式名	Quick UML2001							Java AWT 5.0						
	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%	TP	FP	TN	FN	*P*/%	*R*/%	*A*/%
Composite	—	—	—	—	—	—	—	108	0	0	2	100	98.1	98.1
State	13	0	0	2	100	86.7	86.7	134	0	0	12	100	91.8	91.8
Template	5	0	0	0	100	100	100	290	0	0	6	100	98.0	98.0
Decorator	—	—	—	—	—	—	—	69	0	0	0	100	100	100
Singleton	1	0	0	0	100	100	100	135	0	0	11	100	92.5	92.5
Factory method	—	—	—	—	—	—	—	55	0	0	2	100	96.5	96.5
Bridge	—	—	—	—	—	—	—	162	0	0	10	100	94.2	94.2
Visitor	—	—	—	—	—	—	—	13	0	0	2	100	86.7	86.7
Adapter	82	0	0	3	100	72.7	72.7	394	0	0	15	100	96.3	96.3
Prototype	6	0	0	1	100	85.7	85.7	—	—	—	—	—	—	—
平均统计					100	90.0	90.0					100	94.7	94.7

注：“—”表示该模式在系统中不存在.

93.8%,准确率 A 分别为 80% 和 93.8%. State 模式在表 5 中 Quick UML2001 与 Java AWT 5.0 系统中的召回率 R 分别为 86.7% 和 91.8%,准确率 A 分别为 6.7% 和 91.8%. Adapter 模式在表 3 JhotDraw 6.0b1 与 Dom4j 1.6.1 系统的召回率 R 分别为 92.1% 和 97%,准确率 A 分别为 92.1% 和 97%,Adapter 模式在表 4 中 Junit 3.8 和 Apache ant v1.6.2 系统中的召回率 R 分别为 93.3% 和 97.3%,准确率 A 分别为 93.3% 和 97.3%. Adapter 模式在表 5 中 Quick UML2001 与 Java AWT 5.0 系统中的召回率 R 分别为 72.7% 和 93.3%,准确率 A 分别为 72.7% 和 93.3%. 而导致出现该问题的原因是 State 和 Strategy 模式以及 Adapter 模式和 Command 模式等众多模式存在实例重叠,并且相同设计模式实例之间也存在实例重叠问题. 虽然,第 3 节提出的方法一定程度上解决了设计模式实例的重叠问题,见表 6,但对涉及代理等复杂关系的模式实例重叠问题仍不能 100% 避免假阴性结果,在后续工作中将继续完善 ARFGS 文法,并通过演化计算等对识别效果进行优化.

表 6 六个测试系统设计模式重叠实例统计表

Table 6 Design pattern instance overlap statistics in six systems

设计模式名	设计模式实例重叠数						
	JH	DO	JU	AP	QU	JA	统计
Composite	1	7	0	2	0	9	19
State	0	0	0	0	0	1	1
Template	0	1	0	0	0	0	1
Decorator	0	1	3	2	0	4	10
Singleton	2	1	0	0	0	1	4
Factory method	0	0	0	0	0	0	0
Bridge	0	3	0	7	1	11	21
Visitor	0	0	0	0	0	0	0
Adapter	2	2	1	9	0	14	28
Prototype	0	0	0	0	0	0	0

(2) Singleton 模式的假阴性结果 FN 在四个开源系统都存在,由于 Junit 3.8 软件系统中未识别到 Singleton 模式实例,而 Quick UML2001 中 Singleton 模式的 FN 值为 0. 统计发现,Singleton 模式存在假阴性结果 FN 的概率为 80%. 表 3 中 Singleton 模式在 JhotDraw 6.0b1 系统的召回率 R 与准确率 A 皆为 85.7%,在 Dom4j 1.6.1 系统的召回率 R 与准确率 A 皆为 88.9%,表 4 中 Singleton 模式在 Apache ant v1.6.2 系统的召回率 R 和准确率 A 皆为 77.8%,表 5 中 Singleton 模式在 Java AWT 5.0 系统的召回率 R 与准确率 A 皆为 92.5%. 分析原因后发现 Singleton 模式自身结构比较简单,且存在自循环而导致很难被正确识别出来.

表 6 描述了上述六个测试系统中模式实例重叠的统计结果,其中 JH 表示 JhotDraw 6.0b1,DO 表示 Dom4j 1.6.1,JU 表示 Junit v3.8,AP 表示 Apache ant v1.6.2,QU 表示 Quick UML2001,JA 表示 Java AWT 5.0,最后一列统计了六个测试系统中十个不同设计模式实例发生重叠的总数. 由此可见,六个测试系统总共识别的 Adapter 实例数为 28,Bridge 实例数为 21,Composite 实例数为 19,Decorator 模式实例数为 10,这四种设计模式共享实例的情况较多,而深层次思考后,发现上述 4 种模式具备一个共同的特点,即都为结构型模式.

Gamma 等[1] 将设计模式分为结构型、行为型和创建型三类. 分析三者特点发现行为型模式存在难以跟踪的复杂控制流,而创建型模式存在委托等难以复制的关系,以至于很难被共享,可理解为很难发生设计模式实例识别中的重叠问题,而结构型模式涉及如何组合类及对象,以获得更大的结构,并采用继承机制来组合接口或实现,这样的特点使其结构具有更大的灵活性,也更容易共享模式实例. 依据表 6 的统计结果分析,图 10 中结构型设计模式实例重叠占据了主导地位,达到了 93%.

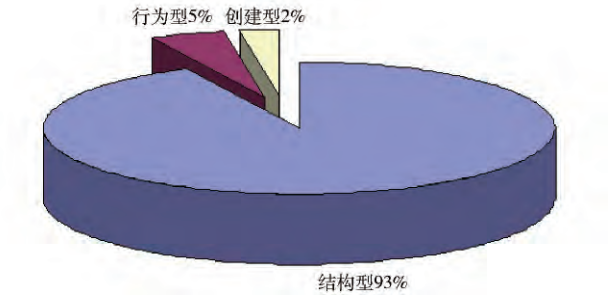

图 10 三类模式重叠实例识别所占比例

Fig. 10 Proportion of three types of pattern overlapping instances

5.2 F-score 指标比较实验

文献[19] 提出了融合精确率 P 与召回率 R 的 F-score 指标.

$$\text{F-score} = \frac{(1+W^2)\times P\times R}{W^2\times P+R}. \tag{28}$$

式中,W 为权值,其值给定为 2.8,F-score 值与设计模式实例恢复的效果成正比. 为检验本研究提出的方法,结合表 3 中 JhotDraw 系统的平均精确度 P 与平均召回率 R 进行了实验.

除所提新方法外,还与参考文献[4,12−13,15,18] 中提出的检测方法进行比较. 由表 7 的 F-score 值可知,本文方法取得了较好的效果.

6 结论

文中提出一种形式化上下无关文法关系驱动的设

表 7　F-score 值比较

Table 7　F-score value comparison

识别技术	平均精确度/%	平均召回率/%	F-score/%
笔者工作	100	91.57	92.45
文献[4]	94	62	64.48
文献[12]	91	67	69.06
文献[13]	68	82	80.13
文献[15]	94	72	73.96
文献[18]	39	100	84.97

计模式检测方法，通过逆向工程工具将源代码中的特征信息进行抽取，之后以文献[21]中提出的可视化形式对抽取信息进行表示与分析，并结合文献[22]中描述的文法，以形式化文法描述了符号间存在的关系，接着在文献[20]制定的设计模式参与者间附加关系检测规则的基础上，改进该文法以发现附加关系，并优化 ARFGS 文法进一步避免了模式实例参与者共享实例的问题. 由实验结果可见，由于综合考虑了设计模式角色间的附加关系与实例共享问题，本研究在避免设计模式实例识别的假阴性结果与重叠问题方面取得了较好的效果，使得模式实例识别的精确度显著提高.

参 考 文 献

[1] Gamma E, Helm R, Johnson R, et al. *Design Pattern: Elements of Reusable Object-Oriented Software*. Pearson Education India, 1995

[2] Ampatzoglou A, Frantzeskou G, Stamelos I. A methodology to assess the impact of design patterns on software quality. *Inf Software Technol*, 2012, 54(4): 331

[3] Ampatzoglou A, Michou O, Stamelos I. Building and mining a repository of design pattern instances: practical and research benefits. *Entertain Comput*, 2013, 4(2): 131

[4] Ampatzoglou A, Charalampidou S, Stamelos I. Research state of the art on GoF design patterns: a mapping study. *J Syst Software*, 2013, 86(7): 1945

[5] Yu D J, Zhang Y Y, Chen Z L. A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures. *J Syst Software*, 2015, 103: 1

[6] Yu D J, Zhang Y Y, Ge J L, et al. From sub-patterns to patterns: an approach to the detection of structural design pattern instances by subgraph mining and merging // 2013 *IEEE 37th Annual Computer Software and Applications Conference (COMPSAC)*. Kyoto, 2013: 579

[7] Fontana F A, Maggioni S, Raibulet C. Design patterns: a survey on their micro-structures. *J Software Evol Process*, 2013, 25(1): 27

[8] Fontana F A, Maggioni S, Raibulet C. Understanding the relevance of micro-structures for design patterns detection. *J Syst Software*, 2011, 84(12): 2334

[9] Zanoni M, Fontana F A, Stella F. On applying machine learning techniques for design pattern detection. *J Syst Software*, 2015, 103: 102

[10] Alhusain S, Coupland S, John R, et al. Towards machine learning based design pattern recognition // 2013 *13th UK Workshop on Computational Intelligence (UKCI)*. Guildford, 2013: 244

[11] Chihada A, Jalili S, Hasheminejad S M H, et al. Source code and design conformance, design pattern detection from source code by classification approach. *Appl Soft Comput*, 2015, 26: 357

[12] Dong J, Zhao Y J, Sun Y T. A matrix based approach to recovering design patterns. *IEEE Trans Syst Man Cybern Part A*, 2009, 39(6): 1271

[13] Bernardi M L, Cimitile M, Lucca G D. Design pattern detection using a DSL-driven graph matching approach. *J Software Evol Process*, 2014, 26(12): 1233

[14] Sabo M, Porubän J. Preserving design patterns using source code annotations. *J Comput Sci Control Syst*, 2009(1): 53

[15] Rasool G, Philippow I, Mäder P. Design pattern recovery based on annotations. *Adv Eng Software*, 2010, 41(4): 519

[16] Garlan D, Monroe R, Wile D. Acme: an architecture description interchange language // *CASCON First Decade High Impact Papers*. Riverton, 2010: 159

[17] Rasool G, Mäder P. A customizable approach to design patterns recognition based on feature types. *Arab J Sci Eng*, 2014, 39(12): 8851

[18] Guéhéneuc Y G, Antoniol G. DeMIMA: a multilayered approach for design pattern identification. *IEEE Trans Software Eng*, 2008, 34(5): 667

[19] Pettersson N, Löwe W, Nivre J. Evaluation of accuracy in design pattern occurrence detection. *IEEE Trans Software Eng*, 2010, 36(4): 575

[20] Xiao Z Y, He P, Li Y. Study on additional relationships based on design patterns' roles. *Appl Res Comput*, 2015, 32(7): 2042
(肖卓宇，何铭，黎妍. 基于设计模式角色的附加关系检测研究. 计算机应用研究，2015，32(7)：2042)

[21] Costagliola G, Delucia A, Orefice S, et al. A classification framework to support the design of visual languages. *J Visual Lang Comput*, 2002, 13(6): 573

[22] Costagliola G, De Lucia A, Orefice S, et al. A parsing methodology for the implementation of visual systems. *IEEE Trans Software Eng*, 1997, 23(12): 777