基于程序源代码的设计模式自动发现技术研究*)

马越峰 李 凡 陈 平
(西安电子科技大学软件工程研究所 西安710071)

摘 要 基于程序源代码的设计模式自动发现是软件逆向工程的重要技术组成。以此为研究对象,提出了一种通用的设计模式自动发现框架,对该框架中的代码理解、模式表示、匹配算法等关键技术进行了研究。并在该框架下,分析比较了国际上三种先进解决方案在以上各技术方面的实现方法和各自特点。并对该项技术的发展前景进行了展望。

关键词 设计模式,面向对象系统,逆向工程,模式挖掘

Research on Design Patterns Automatically Discover Based on Program Source Code

MA Yue-Feng LI Fan CHEN Ping
(Institute of Software Engineering, Xidian University, Xi'an710071)

Abstract Discovering instances of design patterns automatically in the object-oriented source code is an important aspect of software reverse engineering. In this paper, a framework based on discovering design patterns is proposed and the pivotal technologies such as code analysis, pattern import and match arichmeti are studied,. Under this framework, three advanced approaches are analyzed, each approach's characteristics in these technologies are compared. The development of this technology is also prospected.

Keywords Design pattern, Object-oriented System, Reverse dngineering, Pattern discover

1 引言

对于软件系统来说,一个可重用的、灵活并且合理的结构相当重要,有利于软件的扩展和维护。然而,良好的结构并不容易设计,需要一定的开发经验和设计技巧,这使得一般软件开发人员较难设计出结构良好的软件系统。

作为一种重用技术,设计模式[1]受到越来越广泛的关注。特别是在面向对象开发领域中,已经成为衡量软件质量的一种手段。设计模式应用于两种情况,一种是正向工程,即在软件开发过程中,使用设计模式,使得系统的设计更加简洁、高效和合理;另一种是逆向工程,从面向对象系统的源代码中发现设计模式,也被称为模式挖掘,可以简化程序理解,发现设计中存在的问题,以便于改进或重新设计系统。

从面向对象系统的源代码中挖掘设计模式能为软件的设计者、开发者和维护者带来巨大的好处,但是设计模式的挖掘并不容易。首先,在实际的系统中,模式通常被修改以满足特定问题的需要,因此,在模式的挖掘过程中,很可能找不到与标准设计模式中定义的模式一模一样的信息,这就要求考虑模式识别的灵活度问题;同时,从源程序中获得的数据非常多,大部分数据对当前模式的挖掘并没有用,如果不加以精简,会非常耗时,因此,也要考虑数据的过滤问题;另外,相当一部分设计模式体现在系统的对象生成和运行特征上,需要动态的分析,这无疑增加了模式发现的难度。就目前而言,国际上在这一研究方向上基于对源代码动态分析的还相对较少。由于这些困难,在模式挖掘方面的研究进展比较缓慢,公开发表的研究成果也不多,而且,大多数研究成果仅仅限于对结构型模式的挖掘。

本文第2节描述了模式挖掘的关键技术,第3节列举了目前使用的几种方法,并对它们的性能特点进行了比较,最后对该研究方向进行了展望。

2 模式挖掘的关键技术

设计模式的挖掘本质上是一个模式匹配的过程,将从源代码中提取的信息,主要是类的方法、属性已及类间的关系,与模型中存在的信息通过特定的算法进行匹配,从而发现存在的设计模式实例。整个挖掘过程主要包含代码理解、模式表示和匹配算法三个关键部分。用一个通用的框架表示如图1所示。

2.1 代码理解

设计模式一般是由类和类间的关系组合而成。其中,包含了类的各种信息,如类名、类的范畴(抽象类或具体类)、类的属性名和类型、类的方法名和类型、方法的参数名和类型等等;类间关系,主要是关联、聚合、组合、继承以及调用委托等关系。要完成模式挖掘,就必须从源代码中获取这些信息。

代码理解就是通过面向对象的工具和方法,分析面向对象系统的源代码,从代码中提取类和类间关系的信息,并将获取的信息以独立于编程语言和开发工具的中间形式表示出来。目前,已经有一些成形的工具和方法实现了这部分功能,如 Columbus 系统[2]等。

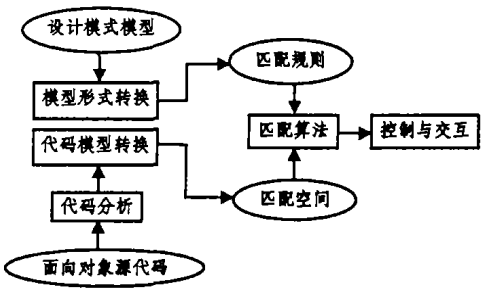


图1 模式挖掘通用框架

*)基金项目:国家"十五"军事电子预研重点项目资助课题(413060601)。马越峰 硕士研究生,研究方向为面向对象技术,逆向工程;李 凡 博士研究生,研究方向为软件逆向工程,人工智能应用;陈 平 教授,博士生导师,研究方向为面向对象技术,软件工程。

2.2 模式表示

通常情况下，设计模式是以某一种形式存在的，如 UML 类图的形式。但在实际的模式挖掘过程中，可能无法直接从类图中提取信息进行匹配，所以，必须将设计模式表示成易于比较的形式，以保证挖掘过程的顺利进行。

根据代码理解结果的表示形式，通过现有的某种面向对象工具和方法，将设计模式转化成一种中间表示，当然，这种表示必须可以和代码中信息的表示直接进行比较。如通过 XML 语言来描述这些信息。

2.3 匹配算法

根据从源代码中提取的信息和设计模式的信息，通过特定的匹配算法，如图的匹配算法对这些信息进行匹配，若匹配成功，则发现了模式的实例。

由于从源代码中获取的信息在模式匹配中并不都有用，有时，这些无用信息是相当多的，如果让这些无用信息参与匹配，则大大地增加了系统的运行时间，从而降低了系统的性能，所以，在匹配算法中还应包含对信息的过滤过程。

3 三种解决方案的分析

3.1 分析 C++ 源代码的头文件发现设计模式

由于 C++ 源代码的头文件中包含了系统的大部分的结构化信息，如类内部的各种信息和类间关系的信息，将有关联的类的信息组织起来，与设计模式的信息进行匹配，从而发现某些设计模式的实例。在 Kraemer 和 Computec GmbH 开发的 Pat 系统[3]中，就使用了这种方法，过程如图2所示。程序理解使用了面向对象系统分析工具 ooCASE[4]和 PROLOG 系统[5]的 D2polog 程序，将提取的信息转换成 Prolog facts；模型表示使用了 PROLOG 系统的 P2polog 程序，将信息表示成 Prolog rules；匹配算法使用了 PROLOG 系统的 query 程序。

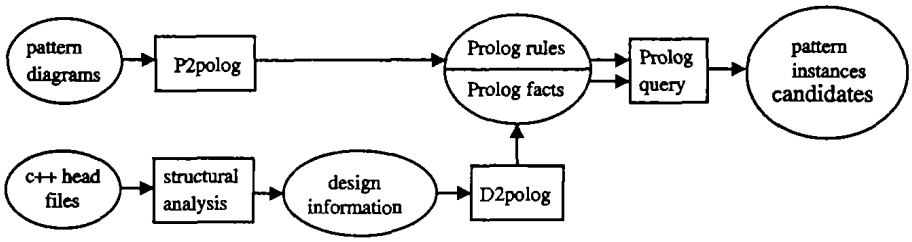


图2 Pat 系统框架图

Pat 系统的流程描述如下：

1. 每一种模式表示成一个静态的 OMT[6]图，这些图构成了模式的知识库 P。

2. 通过程序 P2prolog 将知识库 P 转化成 PROLOG rule 表示，一个模式的 OMT 图对应一个 PROLOG rule。

3. 通过工具 ooCASE 分析 C++ 头文件，提取设计信息并以 OMT 的形式表示，产生系统结构的知识库 D。

4. 通过程序 D2prolog 将知识库 D 转化成 PROLOG facts 表示。

5. PROLOG 的查询 Q 将这两种 PROLOG 表示进行比较，从而发现设计模式的实例。然后自动移除重复发现的模式，手工移除发现的不正确的模式，最终将结果输出。

C++ 源代码的头文件中只涉及类的结构信息，而对类的方法的行为信息则都包含于 cpp 文件中，因而无法获取这些行为信息，所以不能发现创建型设计模式和行为型设计模式。另一方面，从头文件中抽取的信息并不都有用，由于方法中没有对这些无用信息进行过滤，这些信息都参与了匹配，虽然匹配的算法变得简单了，但是却大大增加了系统的时间开销。同时，一些与精确发现模式实例相关的信息也不能抽取出来，这些信息包括类的范畴(抽象或具体，所有的类都被认为是具体的类)，方法的语义类型(构造函数、析构函数、selector、iterator 或 modifier)，完全的聚合和关联的区分，参数列表的调用兼容性，和方法调用的委托(这种信息在头文件中不存在)。由于调用委托信息无法获得，致使最后发现模式的结果的精确率在14%～50%之间，如果可以获得调用委托信息，则精确率可以达到53%～100%。

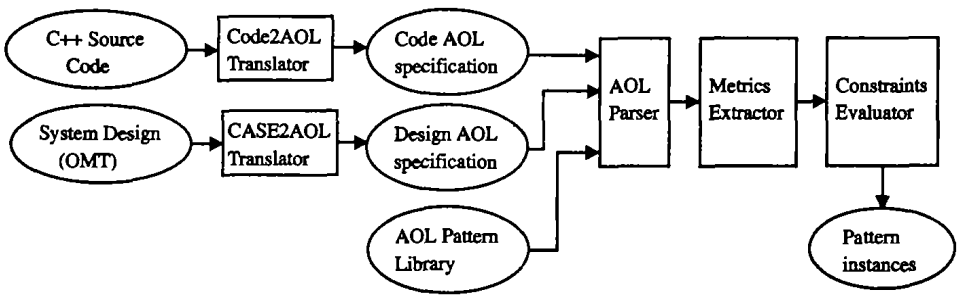


图3 使用 metircs 系统框架图

3.2 使用 metrics 方法发现设计模式

根据面向对象软件系统的 metrics[7]和结构化的特性，对获得的信息经过多阶段过滤，从中剔除无用信息，减少了发现模式实例的时间，提高了系统的性能。过程如图3所示。程序理解使用了 CASE2AOL 工具，将提取的信息表示成 Code AOL specification；模型表示直接将模式库表示成 AOL(抽象对象语言)[8]形式；匹配算法使用 AOL 解析器，将获得的候选模式通过 metrics 和其它限制过滤，最终发现设计模式实例。

方法流程描述如下：

1. 获取系统的 AOL(抽象对象语言)表示。通过 CASE-2AOL 从系统的设计信息中获取系统的 AOL 表示；通过 Code2AOL 从源代码中获取系统的 AOL 表示。

2. 解析系统的 AOL 和模式库的 AOL。通过 AOL 解析器解析系统的 AOL 表示和模式库的 AOL 表示，产生 AST(抽

象语法树),其中模式库中每一个模式都对应一个 AOL 表示,因此每一个模式都产生一个 AST。

3. 类的 metrics 抽取。metrics 抽取器从 AOL AST 中包含的类声明中计算 metrics,并将获得的信息添加到 AST 中。

4. 模式识别。多阶段的约束计算过程处理过滤后的设计模式实例,候选模式的数量,匹配模式信息,发现设计模式实例。

软件 metrics 是从解析 AOL 表示后产生的抽象语法树(AST)中抽取出来的。通过 metrics,可以获得类的属性和操作的数量,以及与其它类之间关系的数量,还有继承的深度和导出类的数量等信息。通过这些信息,根据最短路径规则,可以直接过滤掉那些与这些信息不相符的候选模式。根据软件 metrics 筛选的结果,利用类的方法的委托(delegation)进一步减少候选模式的数量。(方法委托就是一个类执行一个操作,通过简单的调用与它关联的另一个类的一个操作。)正是这种多阶段的信息过滤,防止了检查每一个类的组合爆炸情况的发生。

对于结构型设计模式来说,它们的绝大多数信息清晰地表现在系统的语法表示中,因而很容易获得;而对于另外两种设计模式,必须获取它们的行为信息,而这些行为信息包含在类间消息的交互和类的方法的具体代码中,这些信息是很难获得的,只有类图中的一些静态信息可以获得,而这些信息对于发现创建型和行为型模式是远远不够的,必须到源代码中发现,有些信息必须在执行中才能获得。因此,通过这种方法,只能识别结构型设计模式,识别的精确率可达90%。

这种方法还支持从系统的设计中发现模式实例,但是与从同一系统的源代码中发现的模式实例往往不一致,主要有三方面原因:首先,由于设计中无法获得方法的委托信息,因此会发现比实际存在的模式多的实例;其次,代码中包含的类库的集合在设计中并不存在,因此这些信息只能在源代码中发现;最后,有时设计文档与源代码并不完全一致。

3.3 通过 Columbus 系统分析 C++ 源代码发现设计模式

Columbus 是一个支持工程控制、数据抽取、数据表示、数据存储和过滤的系统。它能够分析大规模的 C++ 工程,根据 Columbus Schema 从源代码中抽取数据信息,并将抽取的信息用抽象语义图(ASG)表示出来。设计模式的信息则通过一种基于 XML 的语言--设计模式描述语言(DPML)[9]来描述。然后将源码中发现的类与模式中的类绑定,判断它们是否存在某种联系,将不存在联系的类过滤掉,再将剩余的类组合后与模式信息匹配,最终发现设计模式的实例。过程如图4所示。

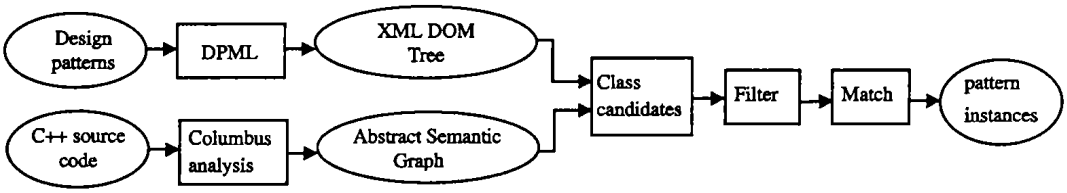


图4 使用 Columbus 发现设计模式系统框架图

方法流程描述如下:

1. Columbus 分析 C++ 源代码,建立一个抽象语义图(ASG)。

2. 使用 DPML 描述设计模式,并将模式描述文件中的信息加载到一个标准的 XML DOM 树中。

3. 解析抽象语义图和 DOM 树,获取模式类的候选类。

4. 过滤候选类,剔除无用信息。

5. 将候选类的组合体与设计模式的信息进行匹配,获得模式的实例。

在设计模式的表示中,使用了设计模式描述语言。由于它是基于 XML 的语言,因此可以非常方便地修改模式的信息,甚至可以完全脱离标准的设计模式,创建用户自定义的模式,这对于用户特定的应用是非常有用的。

Columbus 系统分析完整的 C++ 源代码,除了头文件中的静态信息之外,还可以获取调用委托,对象创建和重载的信息。因此,通过这个系统,除了可以发现结构型设计模式外,还可以发现一些行为型设计模式和创建型设计模式。发现的模式实例的精确率达到了90%以上。由于从 C++ 源代码中抽取的信息比较多,对这些信息的处理花费了较多的时间;可以识别的大多数的设计模式,因而算法比较复杂,系统的复杂度比较高。

结论和展望 至此,我们讨论了三种识别设计模的方法,这些方法都是遵循一个共同的框架,即解析系统源代码和设计模式库,产生两个中间表示,再对这两种中间表示进行比较,发现设计模式的实例。只是它们使用的解析工具和发现算法不同,从而获取的系统信息不同,发现的设计模式实例的多少和精确率不同。

第一种方法从 C++ 头文件中获取信息,这种信息是静态的,只反映了系统的结构,所以只能发现结构型设计模式实例,由于不能获得调用委托等重要信息,致使发现结果的精确率只有15%~50%,而且对信息没有过滤,虽然算法较简单,但是系统相当耗时;第二种方法从 C++ 源代码中获取静态信息,同时可以获取调用委托等信息,虽然也只能发现结构型设计模式,但精确率可达50%~90%,由于对信息进行了过滤,使得算法复杂了一些,但节省了系统的执行时间;第三种方法从 C++ 源代码中获取信息,包括动态信息,可以发现三种设计模式的大多数模式实例,精确率也达到了50%~90%,由于获取动态信息的难度较大,导致算法的复杂度较高,虽然对信息进行了过滤,但由于信息太多,也比较耗时。

虽然每一种方法的时间复杂度不同,但是它们都是相当高的,因此,在进一步的研究中还需考虑更加高效的过滤机制,最大限度地删除与发现模式实例不相干的信息。同样,对于发现算法也需要最大限度的优化,提高系统的执行效率。

由于面向对象系统的行为信息隐藏在执行的过程中,很多设计模式也是针对系统的动态行为的,而 UML 的序列图、交互图和状态图等可以从不同的侧面反映这些行为信息。通过逆向工程工具,逆向产生面向对象系统的这些 UML 图,并解析这些 UML 图,就可以获取系统的行为信息;同时通过解析逆向产生的系统的类图,获取系统的结构信息。我们在以后的工作中,将尝试把这两种信息结合在一起,更精确地发现设

计模式的实例。

参 考 文 献

1 Gamma E, Helm R, Johnson R, Vlissides J. Design Patterns: Elements of Reusable Object-Orien-ted Software. Addison-Wesley Pub Co, 1995

2 Ferenc R, Beszedes A, Tarkiainen M, Gy-imothy T. Colubus - Reverse Engineering Tool and S-chema for C++. In: Proc. of the 6th Intl. Conf. on Software Maintenance, IEEE Comp-uter Society, Oct. 2002. 172～181

3 Kramer C, Prechelt L. Design Recovery by Automated Search for Structural Design Pattern in Object-Oriented Software. In: Proc. of the Third Working Conf. on Reverse Engineering, Nov. 1996. 208～215

4 Houston T X. Paradigm Plus 2. 01 Reference Manual, ProtoSoft Inc., 1994

5 Brondby D. Visual Prolog 4. 0, Prolog Development Center, 1996

6 Rumbaugh J, Blaha M, et al. Object-Oriented Modeling and Design. Prentice-Hall, Englewood Cliffs, NJ, 1991

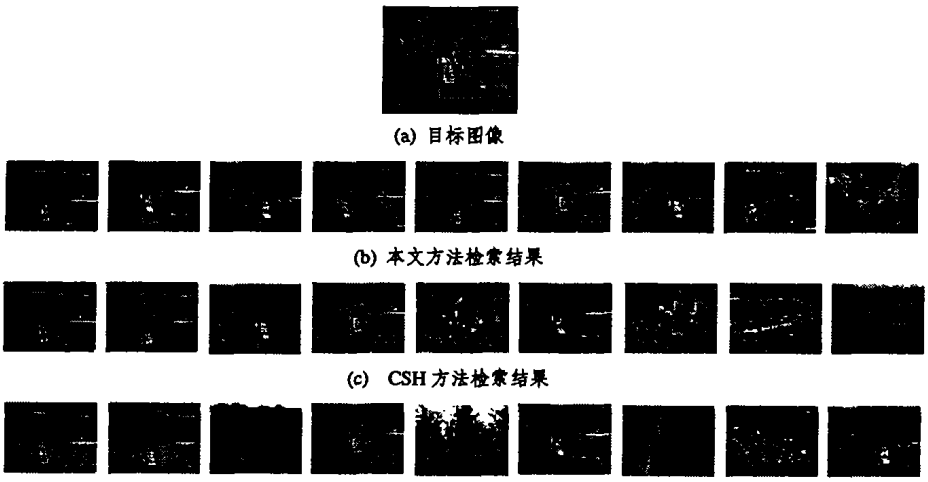
7 Antoniol R F G, Cristoforetti L. Using Metrics to Identify Design Patterns in Object-Oriented Software. In: Proc. of the Fifth Intl. Symposium on Software Metrics, Nov. 1998. 23～34

8 Petroni A. Rappresentazione di design orientatiag-li oggetti ed analisi basata su metriche. Tesidi Diplom-a, University of Trento, Trento, Italy, March 1997

9 Balanyi Z, Ferenc R. Mining Design Patterns from C++ Source Code. In: Proc. Intl. Conf. on Software Maintenance, Sept. 2003. 305～314

(上接第160页)

同一类图像，区域子块的尺寸越小，相似性测量的效果越好，而且对图像中的小尺寸物体也有更好的识别能力，但是减小子块的尺寸以后，检索速度却受到了较大影响。因此，在实际应用中可以折中考虑两种因素，选取合适的分块尺寸。同时，对不同类别的图像，通过调整式(12)中参数 ω_1，ω_2 的取值，可获取更好的检索效果。



(a) 目标图像

(b) 本文方法检索结果

(c) CSH方法检索结果

(d) 直方图方法检索结果

图3 实验结果

结论 本文提出了一种新方法，将彩色图像的颜色特征和空间特征用于彩色图像的相似性检索。色调分量反映了图像颜色的不变量，图像的状态矩阵不仅反映了图像的空间位置关系，也反映了图像的形状信息，采用两种特征相结合的方法进一步提高了图像检索的准确性。本文方法推导简单，且对图像没有特殊的限制和预处理要求。试验结果表明，本文方法不但有效，而且具有较好的检索性能，其检索结果能较好地接近人的视觉感知效果。但如何在检索过程中引入学习机制(如相关反馈技术)，是本文方法下一步的工作。

参 考 文 献

1 王涛，胡事民，孙家广. 基于颜色-空间的图像检索[J]. 软件学报，2002，13(10)：2031～2036

2 孙君顶，武学东，周利华. 基于颜色和形状的图像检索[J]. 计算机科学，2004，31(5)：180～183

3 Gevers T, Smeuder A W M. Evaluating Color and Shape Invariant Image Indexing of Consumer Photograph[A]. In: Proc. of the 1st Intl. Conf. on Visual Information Systems[C], Melbourne, Australia, 1996. 254～261

4 Gevers T, Smeuder A W M. Content-Based Image Retrieval by Viewpoint-Invariant Image Indexing[J]. Image and Vision Computing, 1999, 17(7): 475～488

5 Swain M J, Ballard D H. Color Indexing. Intern Journal of Computer Vision, 1991, 7(1): 11～32

6 Stricker M, Orengo M. Similarity of Color Images[J]. In: Proc. of SPIE Storage and Retrieval for Image and Video Databases, 1995, 2420: 381～392

7 刘忠伟. 利用局部累加直方图进行彩色图像检索[J]. 中国图象图形学报，1998，3(7)：533～537

8 Huang J. Image Indexing Using Color Correlograms[A]. In: Proc. on the IEEE Computer Society Conf. on Computer Vision and Pattern Recognition[C], San Juan, Puerto Rico, USA, 1997. 762～768

9 Nastar C, Mitschke M, Meihac C. Efficient Query Refinement for Image Retrieval[A]. In: Proc. of IEEE Computer Society Conf. on Computer Vision and Pattern Recognition[C], Santa Barbara, California, USA, 1998. 547～552

10 Smeulders A W M, Santini S, Worring M, et al. Content Based Image Retrieval at the End of the Early Years[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000, 22(12): 1349～1380

11 Stehling R O, Nascimento M A, Falcao A X. On 'shapes' of colors for content-based image retrieval[A]. In: Proc. of the 2000 ACM Workshops on Multimedia, California, United States 2000. 171～174