第 36 卷 第 3 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.3 Computer Engineering February 2010
·软件技术与数据库· 文章编号：1000—3428(2010)03—0066—03 文献标识码：A 中图分类号：N945

结合扇入分析与克隆探测的方面挖掘方法

古 辉，刘思聪，阳继旭
(浙江工业大学计算机科学与技术学院，杭州 310023)

摘 要：针对目前遗产系统中普遍存在的关注点分离而导致系统可理解性降低、维护成本提高的问题，提出一种将扇入分析与克隆探测方法结合的方面挖掘方法，对系统中存在的横切关注点进行识别并重构出新的系统结构，将传统系统向面向方面系统迁移。实例分析结果证明，该方法能够提高挖掘的精确度和速度，并降低克隆探测所需的时间。

关键词：方面挖掘；扇入分析；克隆探测；程序理解

Approach of Aspect Mining Combining Fan-in Analysis and Clone Detection

GU Hui, LIU Si-cong, YANG Ji-xu
(College of Computer Science & Technology, Zhejiang University of Technology, Hangzhou 310023)

【Abstract】In order to solve the problem of the separation of the concerns in legacy systems, which makes the system hard to understand and raises the cost of maintenance, this paper mines the aspect using an approach combining the fan-in analysis and clone detection. It identifies the hiding crosscutting concerns and refactors a new system structure, which migrates the legacy system to an aspect-oriented system. It verifies that this approach improves the preciseness and the speed of aspect mining, and needs less time in clone detection.

【Key words】aspect mining; fan-in analysis; clone detection; program comprehension

1 概述

遗产系统中普遍存在着的代码散布(code scattering)和代码交织(code tangling)问题是导致系统关注点分离的主要原因。而现有的编程技术(如面向对象技术)，由于其本身的特点，在对系统的核心关注点进行封装的同时，仍有许多相关的关注点游离于封装体之外，形成了散布的横切关注点，使系统难以形成模块化和清晰的结构，从而降低系统的可读性、可重用性和可维护性。于是，有学者提出面向方面的软件开发技术(Aspect-Oriented Programming, AOP)[1]，希望以此解决这一难题。

AOP 概念提出的 10 多年间，不少相关的技术以及工具已经被开发出来，逐步形成了一个新的研究领域。为了将面向方面技术应用到现有的遗产系统中，或者将现有系统迁移到面向方面的架构，就需要有工具或方法来帮助定义和挖掘系统中散布着的横切关注点(cross-cutting concerns)作为候选，然后将它们重构成一种特殊的结构——方面。方面的概念与面向对象编程中的类有相似之处，是在支配性分解的基础上提供的一种辅助的模块化机制。通过将系统的各关注点封装成方面，使得系统的模块化程度提高。方面挖掘及重构的过程如图 1 所示。本文主要研究方面挖掘技术。

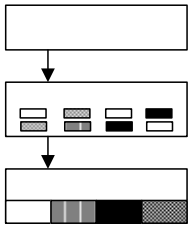

图 1 方面挖掘及重构过程

AOP 技术使得解决系统各个关注点的代码从业务逻辑中独立出来，业务逻辑的代码中不再含有针对各关注点代码的调用，业务逻辑同系统关注点的关系通过方面来封装、维护，这样原本分散在整个系统中的变动就可以很好地管理起来，各个关注点也不再混乱地相互交织，系统的结构更加清晰，有助于程序开发人员更好地理解整个软件系统。

2 方面挖掘

遗产系统向面向方面系统的迁移需要有工具来辅助程序开发人员发现系统中存在的横切关注点或者由工具自动发现。常见的横切关注点有日志记录、错误检验、安全验证等。人工或工具辅助发现系统中隐藏的横切关注点的逆向工程方法即称为方面挖掘，也称为方面识别[2]。

方面挖掘一般有 2 种方法：自顶向下和自底向上。自顶向下方法从整个系统出发，对典型的横切关注点进行分类，然后搜索与这些分类相关的代码，最后试图重构这些代码和分类；自底向上方法分析系统中散布(同一个关注点的实现代码散布在系统各处)和交织(多个关注点中有同一个关注点的代码)的代码，而这些代码可能是隐藏的横切关注点，相关的技术有克隆探测和分片，然而现在的技术一般将这 2 种方法结合使用。

虽然有一些方面挖掘技术已经比较成熟了，但这些方法仍旧存在着一些不足，执行的效率和准确性也存在差异[3]。

基金项目：浙江省自然科学基金资助项目“基于最小信息集的程序信息抽取模型研究”(Y1080189)
作者简介：古 辉(1956－)，男，教授，主研方向：程序理解技术，智能信息处理；刘思聪、阳继旭，硕士研究生
收稿日期：2009-08-26 **E-mail：**gh@zjut.edu.cn

本文所提出的方法融合了扇入分析和克隆探测 2 种方法，优化了整个挖掘过程。另外，本文在相关研究的基础上定义一些性能分析指标，用以对各种方法进行挖掘速度、挖掘准确度及挖掘数量等方面的比较，从而验证本文提出的方法的优越性。

3 挖掘算法

大型系统往往包含大量的方法，而且其中还包括许多显然不适合作为候选横切关注点的方法(如 set 和 get 方法)。结果是使得必须对系统中大量的非横切点方法进行大量的计算，探测的效率大大下降。所以，本算法融合了克隆探测与扇入分析，从而优化了整个方面挖掘过程。挖掘的过程与步骤如下：

过程 1：对系统进行扇入分析。

常见的横切关注点往往是由一个或几个方法构成的，而这些方法往往有很高的扇入度。扇入度被定义为一个方法 m 被其他方法调用的次数[2]。面向对象程序中多态的存在，使得扇入度的计算复杂了许多。在如下所示的 Java 代码中，类 D 包含了 3 个对多态方法 m 的调用。对这段代码进行扇入分析后的结果如表 1 所示。

```
interface A {public void m();}
class B implements A {public void m() {};}
class C1 extends B {public void m() {};}
class C2 extends B {public void m() { super.m();};}
class D {
void f1(A a) { a.m(); }
void f2(B b) { b.m(); }
void f3(C1 c) { c.m(); }
}
```

表 1　多态调用实例的扇入度

方法	调用集	扇入度
A.m	{D.f1, D.f2, D.f3}	3
B.m	{D.f1, D.f2, D.f3, C2.m}	4
C1.m	{D.f1, D.f2, D.f3}	3
C2.m	{D.f1, D.f2}	2

在一般的扇入分析中，往往要取一个特定扇入度的阈值，超过这个阈值的方法会被认为是存在着代码重复，从而被视为候选的横切关注点。小于这个阈值的及其他符合约束条件的方法则被过滤掉，得到分析结果。根据多次实验结果分析，此阈值取 10 较为合适，能过滤掉整个系统中约 90%的方法，而且余下的方法的代码重复度都较高。

对系统进行扇入分析的目的是在进行克隆探测前进行过滤，以减轻克隆探测的工作量，同时预先去除了明显不是横切关注点的方法，提高了克隆探测的准确度。分析过程主要分为以下 3 步：

步骤 1 计算所有方法的扇入度，并过滤掉扇入度小于阈值的所有方法，在这里，阈值取 10。

步骤 2 过滤掉所有的 set 和 get 方法。

步骤 3 过滤掉实用方法(如 toString()方法)以及集合操作方法(collections manipulation methods)。

过程 2：对余下的方法生成细颗粒的 PDG 并进行最大相似子图的计算。

横切关注点很重要的一个特征是它的代码在整个软件系统的各个地方都以克隆代码的形式出现，即代码反复重复出现和调用。克隆探测方法的目标便是找出程序代码或系统中重复的代码，尤其是复制的代码，从而将这些重复代码所处的上下文的方法作为横切关注点予以挖掘出来。现有的克隆探测方法主要有基于文本、基于符号、基于抽象语法树(Abstract Syntax Tree, AST)及基于 PDG 等。

本文采用的方法是一种区别于传统方法的 PDG 生成方法，称为细颗粒的 PDG，用以展现系统的结构与数据流[4]。传统的 PDG 用顶点来表示赋值声明和控制判断式，边来表示依赖关系。而细颗粒的 PDG 定义为由 4 个元组组成的有向图 $G=(V,E,\mu,\nu)$。其中，V 表示顶点的集合；$E\subseteq V\times V$ 为边的集合；$\mu:V\to A_V$ 表示顶点的属性；$\nu:E\to A_E$ 表示边的属性。$\Delta:E\to A_V\times A_E\times A_V$ 表示映射关系：

$$\Delta(V_1,V_2)=(\mu(V_1),\nu(V_1,V_2),\mu(V_2))$$

对给定的 2 个图 $G_1=(V_1,E_1,\mu_1,\nu_1)$ 和 $G_2=(V_2,E_2,\mu_2,\nu_2)$，如果存在如下的相互映射关系 $\Phi:V_1\to V_2$，其中 $(V_i,V_j)\in E_1\Leftrightarrow(\Phi(V_i),\Phi(V_j))\in E_2$，$\Delta_1(V_i,V_j)=\Delta_2(\Phi(V_i),\Phi(V_j))$，则认为 G_1 和 G_2 是同构的。

要判断 2 个图是否同构是一个 NP 完全问题，所以，本文考虑的是构建 2 个图的最大相似子图。只要 2 个图中存在相同的路径$\{v_0,e_1,v_1,e_2,v_2,\cdots,e_n,\ v_n\}$和$\{v'_0,e'_1,v'_1,e'_2,v_2',\cdots,e'_n,v'_n\}$，而且每条边和每个顶点的属性都相同，则认为这 2 个图是相似的。其中的相似部分(即相同路径)便认为是系统中的克隆代码，这些代码所处的上下文的方法则相应地被认为是横切关注点。但要是从每个顶点开始把所有路径计算一遍是不现实的，因此，在计算时只取一个顶点作为初始的起点，这样，所有的点只可能被遍历一遍。

在进行克隆探测时，需要将所有的 PDG 两两配对，然后对任一对(G_1, G_2)，生成最大相似子图对(G_{v1}, G_{v2})，算法如下：

```
generate (v1, v2)
{    初始化 Gv1, Gv2 为空;
     Call propagate ({v1}, {v2});
     Return Gv1, Gv2;}
propagate (V1, V2)
{    For (所有以 V1, V2 为起点的边 E1, E2)
     {    根据 Δ 将 E1 和 E2 分类为 E1i 和 E2i;
          将 E1i 和 E2i 添加到 Gv1, Gv2;
          For (E1i 和 E2i 所达的顶点 V1i 和 V2i)
          {         If (V1i 和 V2i 非空)
                    {    将 V1i 和 V2i 添加到 Gv1, Gv2;
                         Call propagate (V1i,V2i);
}}}}
```

尽管对图的遍历方法进行了优化，对每个点只需要遍历一次，但因为要对所有方法进行配对，计算量还是相当大的。而且系统中往往还有很大一部分的方法根本不可能存在横切关注点，但根据算法还是对其进行了配对并计算了最大相似子图。采用了扇入分析后，就可以有效地将大量典型的非横切关注点方法预先过滤掉，这样就使得对方法进行配对和相似子图的计算的工作量都大大降低，节省了计算时间，也提高了提取横切关注点的质量。

4 实例及挖掘结果分析

本文采用的实例系统是 JHotDraw v5.4。其中包括大约 2 800 个方法及近 18 000 行代码(不包含注释行)。而运行的环境则是在 Eclipse 中以插件形式实现的 Ophir 工具[5]，支持多种自动方面挖掘的分析，并可以以手动或自动方式重构方面。

对 JHotDraw 系统进行方面挖掘后，Ophir 会返回所有候选的横切关注点。这些关注点会被分类为 3 种类型：(1)强候

选者：能在以后被重构成方面的方法；(2)重复方法：只是代码重复的方法；(3)过滤掉的方法：Ophir 能自动对一些方法进行过滤，比如空的克隆代码和只有一个节点的方法等。

挖掘的结果如表 2 所示，相比文献[5]的挖掘结果，在强候选者和总挖掘数上都有了一定程度的提高，系统中隐藏的更多横切关注点得以挖掘并可以重构成方面。对如此规模的系统来说，挖掘数的提高无论是对于横切关注点的理解还是对于后续的方面重构质量的提高都有很大的助益。

表 2　候选横切关注点挖掘结果

候选类型	挖掘数	百分比/(%)
强候选者	58	56.9
重复方法	21	20.6
过滤掉的方法	23	22.5
合计	102	-

而表 3 是挖掘各个阶段所耗费的时间统计。在以往的克隆检测挖掘方法中，时间大部分花在识别上，这是因为生成 PDG 和相似子图的计算量往往非常巨大。而采取本文的方法后，虽然有部分时间消耗在了扇入分析上，但通过扇入分析的过滤后，大量的不适合作为候选的方法被排除在外，使得系统在构建 PDG 和最大相似子图上的工作量大大减少，从而使得整个挖掘的时间也得到了很好的改善。

表 3　挖掘时间比较

挖掘阶段	消耗时间/s	
	本文方法	传统方法
构建	18	17
过滤	865	0
识别	1 025	2 158
合并	256	248
合计	2 164	2 451

5　结束语

本文提出的方面挖掘方法很好地结合了扇入分析和克隆探测 2 种技术，在方面挖掘的速度和准确度上都有一定程度的进步。在过滤掉大量非候选的横切关注点后，克隆探测的工作显得更加轻量级，且不失准确性。但是，在运用这种方法时还要进行一部分的人工操作，因此，接下来的工作是考虑如何全部实现自动化并实现方面重构的结合，形成完整的面向方面程序开发环境。

参考文献

[1] Kiczales G, Lamping J, Anurag M, et al. Aspect-oriented Programming[C]//Proc. of European Conference on Object-oriented Programming. [S. l.]: Springer-Verlag, 1997: 220-222.

[2] Marin M, Deursen A, Moonen L. Identifying Aspects Using Fan in Analysis[C]//Proc. of the 11th Working Conference on Reverse Engineering. Washington D. C., USA: IEEE Computer Society, 2004.

[3] Nora B, Said G, Fadila A. A Comparative Classification of Aspect Mining Approaches[J]. Journal of Computer Science, 2006, 2(4): 322-325.

[4] Krinke J. Identifying Similar Code with Program Dependence Graphs[C]//Proc. of the 8th Working Conference on Reverse Engineering. [S. l.]: IEEE Computer Society Press, 2001: 301-309.

[5] Shepherd D, Gibson E, Pollock L. Design and Evaluation of an Automated Aspect Mining Tool[C]//Proc. of International Conference on Software Engineering Research and Practice. Las Vegas, USA: [s. n.], 2004.

编辑　任吉慧

（上接第 65 页）

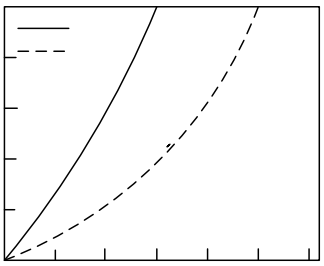

图 2　模型 M_{movies} 的生成时间

图 3 比较了本文算法和 MGS_{sd} 算法生成的 M_{auto} 模型。

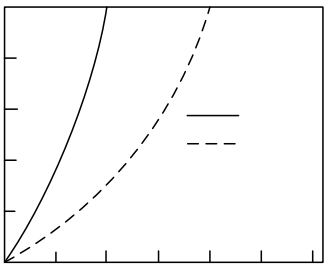

图 3　模型 M_{auto} 的生成时间

由于汽车领域属性集合的规模比电影小，因此与图 2 相比，本文算法和 MGS_{sd} 算法生成的 M_{auto} 模型的执行时间相应减少，本文算法的用时小于 MGS_{sd} 算法。

通过实验证明，本文算法与 MGS_{sd} 算法相比，其执行效率明显提高。

5　结束语

与原有算法相比，本文算法的执行效率得到明显提高。抽取查询接口的模式并对查询接口模式异构性进行处理是查询接口集成的关键。因此，如何正确抽取查询接口并快速有效地处理接口异构性，是下一步研究的方向。

参考文献

[1] 刘　伟, 孟小峰, 孟卫一. Deep Web 数据集成研究综述[J]. 计算机学报, 2007, 30(9): 1475-1478.

[2] He Bin, Chang K C C. Statistical Schema Matching Across Web Query Interfaces[C]//Proc. of SIGMOD'03. San Diego, California, USA: [s. n.], 2003: 1-10.

[3] Rahm E, Bernstein P A. A Survey of Approaches to Automatic Schema Matching[J]. VLDB Journal, 2001, 10(4): 334-350.

编辑　陈　晖