一种结合扇入和概念分析技术进行 Aspect 挖掘的方法*)

张晓风　陈　平　崔伟勇
(西安电子科技大学软件工程研究所　西安 710071)

摘　要　横切关注是分布在多个模块单元的功能，其存在是对系统理解和进化的一个很大的障碍。AOP(Aspect Oriented Programming)提出了将横切关注模块化为 aspect 的方法，以解决这个问题。其中最难的是如何发现 aspect，论文提出了一种结合扇入和概念分析技术进行 aspect 挖掘的方法，并通过系统的实验验证了该方法的有效性和正确性。

关键词　AOP，Aspect 挖掘，扇入，概念分析

A Method of Aspect Mining Using Fan In and Concept Analysis

ZHANG Xiao-Feng　CHEN Ping　CUI Wei-Yong
(Software Engineering Institute, Xidian University, Xi'an 710071)

Abstract　Crosscutting concerns are functionalities that assigned to not a single modular unit. The existence of croscutting concerns is a major problem in software understanding and evolution. Aspect Oriented Programming (AOP) offers mechanisms to factor them out into a modular unit called aspect. One of the hardest problems is how to detect aspect. In this paper a method combined fan in and concept analysis for aspect mining is presented. Finally, a systematic experiment is conducted in order to verify the correctness and validity of the method.

Keywords　AOP, Aspect mining, Fan in, Concept analysis

1　引言

面向对象的系统，通过将应用领域内的真实实体映射为层次化的类，来进行软件系统的开发[1]。但是并不是所有的开发的系统的需求都需要与单个模块单元(类)相对应。某些需求的实现被分散在不同的类中，而且有可能与类中实现其它职责的功能相搀杂。这种需求与实现的不匹配，成为系统维护阶段的一个潜在的严重问题[2]。事实上很难在单元模块里找到如何实现这些需求的，不利于对程序的理解，而且如果需求改变，代码的改进就变得很困难。AOP(Aspect Oriented Programming)通过提供一种称为 aspect 的新模块来消除横向的需求(横向关注)与代码之间的不匹配。aspect 可以使需求定位到代码单元。AOP 将一个横向关注实现为一个独特单元 aspect 的一部分，而不是将其散射到多个类中并与其它功能相搀杂[3]。

将横向关注模块化为 aspect 有利于对现有软件的理解与改进。但是其中最难的一个问题是如何确定一个功能是否应该被视为一个 aspect。这个问题现在被称为 aspect 挖掘[4]。本文提出一种 aspect 挖掘的方法，该方法是将目标系统进行植入，收集目标系统的静态信息，运行植入后的目标系统，收集动态信息。通过扇入理论[5]过滤掉那些成为 aspect 种子可能性极小的方法，对那些极有可能成为 aspect 种子的方法，结合 aspect 挖掘的目的进行概念分析，将种子分类打包成 aspect，本文主要是通过用例图构造概念进行分析[6]。这种 aspect 挖掘的方法，可以先将那些大量的难以成为 aspect 种子的方法过滤掉，因此对于较大的目标系统，这种方法比单纯的概念分析更有效。

2　相关理论

2.1　扇入

扇入度量(或扇入值)是由 Henderson-sellers 定义的。扇入度量是对控制变成模块的位置进行计数。在向对象的上下文里，扇入度量的模块类型被定义为方法。本文将扇入的度量定义为对方法 M 进行调用的不同方法的数目。如果某些方法的扇入值比较高，就可以被看作是横切关注的种子，成为 aspect。

这种度量非常简单，但是由于多态的存在，问题可能稍微复杂一点。多态使一个方法调用可以影响多个方法的扇入值。对方法 M 的调用会影响到被 M 精化的方法以及将 M 精化的方法[5]。在图 1 中给出了一个例子。并在表 1 中给出了该例子中各方法 m 的扇入值以及调用方法集合。如表 1 中所示，类 D 中方法 f2 对 B 的成员方法 m 的调用使 B 的超类 A 和 B 的子类 C1，C2 中的方法 m 的扇入值都增加。

表 1　图 1 所示代码的 Fan-In

Method	Callerset	Fan-In
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

2.2　概念分析

概念分析(FCA)是集合论中格理论的分支，它提供了识别具有共同属性的对象的最大组。给定上下文 $C(O,A,R)$ 其中 O 是对象的有限集合，A 是属性集合，R 是对象集合 O 和

*)基金支持：国家自然科学基金(项目编号：60473063)，国家教育部博士点基金(项目编号：20030701009)及"十五"国防预研项目(项目编号：41306060106)。张晓风　硕士研究生，主要研究领域为逆向工程，面向对象技术。陈　平　博士，教授，博士生导师，主要研究领域为电子信息系统软件开发理论与技术，面向对象技术，软件工程，逆向工程。崔伟勇　硕士研究生，主要研究领域为逆向工程，面向对象技术。

属性集合 A 上的二元关系，$R \in O \times A$，概念 c 被定义为集合对(X,Y)且满足：

$$X=\{o \in O \mid a \in Y: (o, a) \in R\} \quad (1)$$

$$Y=\{a \in A \mid o \in X: (o, a) \in R\} \quad (2)$$

其中集合 X 被称为概念 c 的 extent(Ext[c])，Y 被称为概念 c 的 intent(Int[c])[6]。

```
interface A{
      public m();}
class B implements A{
     public void m();}

class C1 extend B{
     public void m();}
class C2 extend 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(); }}
```

图 1　多态代码实例

从概念 c 的定义可以看出在每一个概念 c 中每一个元素都具有概念中的属性，每一个属性所有的元素都具有。而在该概念外没有一个元素具有该概念内的那些相同属性，每一个属性都不可能被概念内的所有元素具有。在图 2 所示的左上表格中给出了对象 o1，o2，o3 与属性 a1，a2，a3，a4 之间的关系，* 表示该行的对象具有该列的属性。图 2 中间给出了由表格中的属性对象关系构造的概念。由构造的概念可以看出如果将概念的 extent(对象集合)或者 intent(属性集合)排序，构造出的概念成偏序关系。这种偏序关系可以表示为图 2 右边所示的格。从格中可以看出越靠近底层的节点具有的对象数越少，越靠近顶层的节点包含的对象数越多，同时当对象数增多时对象所共有的属性即集合 Y 的元素个数越少。节点 $n1$ 是节点 $n2$ 的特化当且仅当 $n1$ 的对象集合 X1 是 $n2$ 的对象集合 X2 的超集。反之称 $n2$ 是 $n1$ 的一般化。

对于格 L 中的一个节点 n 的完整表示应该是一个集合对$(\mathrm{Ext}[n],\mathrm{Int}[n])$。然而，表示同样的信息可以用更简洁更易读的方法。节点 n 的信息可以用 Ext[n]中的一个对象 o 表示，当且仅当该节点是包含对象 o 的最特化的节点。同样节点 n 的信息可以用 Int[n]中的一个属性表示当且仅当该节点是包含属性 a 的最一般化的节点。用数学的方法表示可以用单个对象或属性表示的概念集合分别为公式(3)和(4)：

$$\gamma(o)=\inf\{n \in L \mid o \in \mathrm{Ext}[n]\} \quad (3)$$

$$\mu(a)=\sup\{n \in L \mid a \in \mathrm{Int}[n]\} \quad (4)$$

	a1	a2	a3	a4
o1	*	*	*	
o2	*	*		*
o3	*		*	

bot = ({},{a1,a2,a3,a4})
c0 = ({o2}, {a1,a2,a4})
c1 = ({o1}, {a1,a2,a3})
c2 = ({o1,o2}, {a1,a2})
c3 = ({o1,o3}, {a1,a3})
top = ({o1,o2,o3}, {a1})

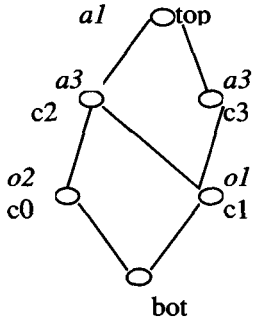

图 2　概念格实例

公式(3)表示的是用对象 0 表示的节点，其中 inf 表示概念集合的下确界，公式(4)表示的是用属性 a 表示的节点，其中 sup 表示概念集合的上确界。由公式(3)(4)表示的节点标记法，只能标记最特化或者最一般化的节点。在图 2 的各种用斜体标记了可以用单个对象或属性标记的节点。

3　结合扇入和概念分析进行 aspect 挖掘的方法

3.1　基本思想

通过对目标程序进行植入，收集目标程序信息。对信息进行扇入分析，提取可能成为横切关注的方法。通过概念分析对这些方法分类打包成 Aspect。本文主要通过分析用例图(use-case)中的方法来将扇入分析得到的方法进行分类打包。这种方法不仅可以通过分析用例图来构造概念格，还可以通过将扇入分析的结果通过其他的功能分析构造概念格进行 aspect 挖掘。

3.2　具体实现细节

3.2.1　对目标系统进行植入收集信息

通过反射原理对目标系统进行植入，运行目标程序，在这过程中收集目标系统的方法调用信息包括被调用的方法，调用方法，被调用的总次数以及生成 use-case 所需要的信息。以调用方法建立结构体，其中调用方法和被调用方法以 C. M()的形式表示，其中 C 为方法 M 所属的类，M 为调用方法或被调用方法的名字。而调用的总次数实际就是被调用方法的 Fan-In。

3.2.2　对收集得到信息做扇入分析

通过对以往关于横向关注的研究，可以发现横向关注可以发生在不同的层次，比如类，方法等。在方法的层次上横向关注大多隐含在对方法的调用而不是调用方法本身的逻辑，也就是说横向关注主要是发生在控制变成模块的地方。典型的例子包括日志，跟踪和条件检查和异常处理等。

AOP 通过一种称为 advice 的机制在方法的层次上减少横向关注。这种机制主要是获取程序执行控制，并将横向的功能添加到这些控制中。这种机制将横向关注孤立为一个模块。在本文所述的方法中，将该机制逆向运用，通过扇入分析提取具有散射特征的方法。而横向关注则体现为分布在代码不同位置的对某一实现横向功能的方法的调用。因此我们可以被调用方法的扇入值作为横向关注的一个标志。对收集的信息作扇入分析主要通过以下几步：

- 除去 Fan-In 小于某一固定值的方法的记录，该固定值一般取 10。
- 除去名字与 get * /set * 模式相匹配的方法的记录，因为这些方法只是设定或返回一个值。

·除去公用方法像 ToString(),以及集合方法(collection method)的记录。

通过以上三步获得的方法信息大多可以作为 aspect 的种子。

3.2.3 对扇入分析的结果做概念分析生成 aspect

use-case 是一个连贯的功能性单元,由一个用消息顺序表示的类元(系统、子系统、或类)提供,这些消息与系统执行的动作在系统和一个或多个外部用户(表现为参与者)间交换。一个 use-case 反映了系统的一个功能。当一个功能由多个分布在源代码不同位置的方法实现时,这个功能往往有可能会成为一个 aspect。基于这个原理,可以通过分析 use-case 来获取 aspect。具体流程如下:

(1)通过对源代码进行植入,生成一系列 use-case。并将扇入分析阶段得到的 aspect 种子的方法在这些 use-case 中进行概念分析。

(2)构造概念 c,其中将方法作为 intent 即 attribute 集合,将 use-case 剧情作为 extent 即 object 集合,二者之间的关系 $R=(o,a)$是指方法 a 在 use-case 剧情 o 中出现。

表 2 扇入分析结果

序号	方法	Fan-In
a1	Centerctrl. park()	13
a2	RoadBlock. realeased()	5
a3	Car. Run_Ahead()	8
a4	Ctime. GetCurrentTime()	15
a5	Time. Format()	17
a6	ESystemTime()	12
a7	FILE. fputs()	28
a8	CSTring. Format()	26
a9	CEmulatorView. OnTimer()	14
a10	CView. OnTimer()	27
a11	Ctime. SetTimer()	5
a12	ASSERT()	23
a13	CEmulatorView. CreateCompatibleDC()	14
a14	CBitMap. CreateCompatibleBitmap()	14
a15	CEmulatorView SelectObject()	14
a16	CEmulatorView SetTextColor()	14
a17	CEmulatorView SetBkMode()	14

(3)通过构造概念格对构造的概念分析。在概念格中查找那些可以用一个剧情 o 标记的概念,即属性最特化的概念。如果该概念对以下两个条件成立,该概念中的方法可以被认为是一个 aspect。

·标记一个概念的方法属于的类不少于一个。

·一个类的方法所标记的概念不少于一个。

4 实例分析

为了验证方法的有效性和正确性,论文采用自主开发的有完备的设计文档的停车场仿真程序作为实例实现论文所述的 aspect 挖掘的方法。该仿真程序的所完成的功能主要是模拟停车场的运行情况并将运行情况以日志的形式存储到文本文件中。对该程序作扇入分析得到如表 2 所示的结果,该表中所列的方法是扇入值大于或等于 5 的方法。从表中可以看出经过扇入分析后的方法为只占原程序中方法的很小一部分。大量的实验证明经过扇入分析后,如果取 Fan-In 大于 10 的方法,只有 1%左右而大于 5 的有 5%左右[5]。

跟踪程序运行的轨迹,获得了以下的用例:初始化 O_1,汽车在停车场内 O_2,写日志 O_3,画图 O_4。用例中与表 2 中的方法的关系如表 3 所示,其中标记表示该列的方法出现在该行的用例中。

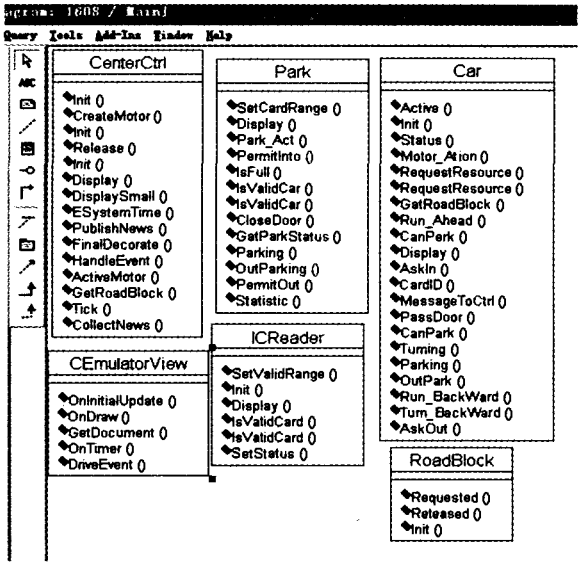

图 3 停车场仿真类图

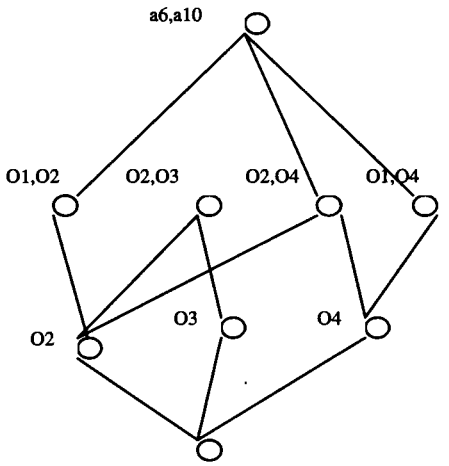

图 4 构造的概念格

表 3 用例与调用方法的关系

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11	a12	a13	a14	a15	a16	a17
O1						*				*	*						
O2	*	*	*	*		*											
O3			*	*			*	*									
O4					*	*			*	*	*		*	*	*	*	*

通过表 3 中的数据可以构造出图 4 所示的概念格,通过分析概念格可以看出用剧情 O 标记的节点为 O_2,O_3,O_4,其中 O_2,O_3,O_4 所标记概念的方法集合可以作为潜在的 aspect 因为 O_4 中的方法属于的类不止一个,而且 O_2,O_3,O_4 中的方法在多个概念中出现。

通过实验可以发现这种先对目标系统进行静态的扇入分析然后进行概念分析的方法能有效地剔除大量的不能成为 aspect 种子的方法。该实验只是用例图来构造概念格,由于用例图是目标系统动态运行收集到的信息,很难收集到目标系统完整的运行路径,因此也存在一定的缺陷。对于构造概

念格使用用例图只是一种方法，对于 aspect 挖掘可以根据 aspect 挖掘的目标使用其它的构造概念格的方法。

结束语 论文提出一种结合 Fan-In 理论和概念分析的技术进行 aspect 挖掘的方法，这种方法比单纯的对目标系统做概念分析更有效。单纯的概念分析对于较大的系统，收集的信息过多，构造的概念格变得非常庞大，构造的过程和对概念格分析的过程费时费力。该方法先通过 Fan-In 分析，将那些难以成为 aspect 种子的方法过滤掉，实验证明这种先作扇入分析的方法可以剔出 95% 左右的非 aspect 种子的方法。但是该方法由于是通过收集动态信息来生成用例，因此由于程序运行很难遍历所有的分支，所以收集的信息不能完全反映目标系统，因此对 aspect 的挖掘也不可能非常准确，但是这种方法还是能将大多数的 aspect 挖掘出来。同是该方法现在还没有完全的自动完成，对于概念分析仍然需要手工完成，论文下一步的工作是要将该方法能自动实现。

参 考 文 献

1 Fowler M. Refactoring: Improving the design of existing code. Addison-Wesley Publishing Company, Reading, MA, 1999
2 Tonella P, Ceccato M. Migrating interface implementation to aspect oriented programming. In: Proc. of the International Conference on Software Maintenance (ICSM), page (to appear), Chicago, Illinois, USA, September 2004. IEEE Computer Society Press
3 Hannemann Jan, Kiczales Gregor. Design Pattern Implentation with Java and AspectJ [M]. JavaWorld, 2003. 9
4 Loughran N, Rashid A. Mining aspects. In: Proc. Of the Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design (with AOSD), Enschede, The Netherlands, April 2002
5 Marin M, van Deursen A, Moonen L. Identifying aspects using fan-in analysis. Technical Report SENR0413, CWI, 2004
6 Tonella P, Ceccato M. Aspect Mining through the Formal Concept Analysis of Execution Traces. WCRE, 2004. 9
7 Ganter B, Wille R. Formal Concept Analysis. Springer-Verlag, Berlin, Heidelberg, New York, 1996
8 Kiselev I. Aspect-Oriented Programming with AspectJ. Sams Publishing, Indianapolis, Indiana, USA, 2002

（上接第 248 页）

如果 node 对应的是顺序进程，则如果 node 的输入名字集合中存在 x，则{goto 3；}否则{node= 它的长兄节点（即包含该进程的 ambient 节点），goto 2}；

如果 node 对应的是 ambi 进程，则如果 node 是根节点，则{goto 4；}否则如果 node 的输入名字集合中存在 x，则{goto 3；} 如果 node 的输入名字集合中不包含 x，则{node= 它的父节点，goto 2；}；

3）找到对应的定义性出现在节点 node 上，确定它的出现路径：从根节点到 node 的路径上的内部编号序列确定了名字的出现路径 path，return path；

4）没有找到对应的定义性出现，表明出错，retrun −1。

算法 FindAmbiDefPath（AmbiTree：TREE，x：NAME，flag：BOOLEAN）：PATH

参数：Ambi 语法树 AmbiTree，一个使用性 ambient 名字 x；

输出：绑定该出现 x 的定义性出现的路径。如果 ambient 名字 x 的定义点不是根节点，则 flag=true；否则 flag=false。

1）找到名字为 x 的 ambi 类型节点 node：对给定的 AmbiTree 从根节点开始进行部分先序搜索，即搜索顺序为节点、它的所有儿子节点，不包括它的弟节点。若找到名字为 x 的 ambi 类型节点 node，则转 2，否则转 6。

2）确定 node 的出现路径：从根节点到 node 的路径上的内部编号序列，确定了名字 x 的一个可能定义路径 path1。

3）逆向搜索找到对应的定义性出现：如果 node 是根节点，则{flag=false；goto 4；}；否则如果 node 的输入 ambient 名字集合中包含 x，则{flag= true；goto 5；} 否则{node= node 的父亲节点，goto 3；}。

4）找到名字 x 的定义性出现路径：retrun path1。

5）确定 node 的出现路径：从根节点到 node 的路径上的内部编号序列，确定了名字 x 的出现路径 path. return path。

6）没有找到对应的定义性出现，表明出错，retrun −1。

（2）对于一个名字 x 的定义性出现，确定它所绑定的所有使用性出现

算法 FindVarBoundPaths（AmbiTree：TREE，x：NAME，path：PATH）：set of PATH；

参数：Ambi 语法树 AmbiTree，一个定义性变量名字 x 及其出现路径；

输出：定义性名字 x 所绑定的所有使用性出现的路径集合。

1）初始化 set={}；

2）根据 path 在 AmbiTree 中找到对应定义性变量名字 x 出现的节点 node；

3）如果 node 对应的是一个顺序进程，则 return path；

4）如果 node 对应的是一个 ambi 进程，则

如果 node 有弟节点，则

{ 对于 node 的每个弟节点 brother，做：如果 brother 的输入名字集合中没有 x，则

{ sibpath= path 加上对应 brother 的编号；

set= set∪{sibpath}；}}

如果 node 有子节点，则

{ 对于 node 的所有子节点 son，做：如果 son 的输入名字集合中没有 x，则

{ sonpath= path 加上对应 son 的编号；

set= set ∪{sonpath}；}

node=son；

goto 3；}}

return set；

算法 FindAmbiBoundPaths（AmbiTree：TREE，x：NAME，path：PATH）：set of PATH；

参数：Ambi 语法树 AmbiTree，一个定义性 Ambient 名字 x 及其出现路径；

输出：定义性名字 x 所绑定的所有使用性出现的路径集合。

1）初始化 set={}；

2）在 AmbiTree 中找到对应定义性 ambient 名字出现的节点 node 及出现路径 path：

path= FindAmbiDefPath(AmbiTree, x, flag)；

如果 flag= true && node 有弟节点，则

{ 对于 node 的所有弟节点 brother，做：

{ sibpath= path 加上对应 brother 的编号；

set= set ∪sibpath；}

return set}

3）如果 flag= false，则 return path；

结论 本文针对 Ambient 演算的层次定义，提出了 Ambient 演算语法结构树的概念，并且在 Windows XP 环境下用 Visual C++ 实现了 Ambient 演算的数据流分析程序，该程序以 ambient 的语法结构树为输入，对于名字的使用性出现，可以给出绑定它的定义性出现路径，以及对于名字的定义性出现，可以给出它所绑定的所有使用性出现路径。为进一步研究 Ambient 演算的性质和实现技术提供了有效的手段和环境。

参 考 文 献

1 Cardelli L, Gordon A D. Mobile Ambients In: Proc. FoSSaCS'98, Lecture Notes in Computer Science, Springer Verlag, 1998, 1378: 140～155
2 金英，金成植，郑爽. Action 演算中动作内部数据流分析方法及其实现. 吉林大学学报（理学版），2003，41：94～98
3 Cardelli L, Gordon A D. Types for mobile ambients. In: Proc. 26th Annual ACM Symposium on Principles of Programming Languages, 1999. 79～92
4 Cardelli L, Gordon A D. Mobility types for Mobile Ambients. In Proc. Of ICALP, Lecture Notes in Computer Science, Springer Verlag, 1999, 1644: 230～239
5 张丽翠. 具有并行性和移动性的演算语言的形式语义研究：[吉林大学博士学位论文]. 2005. 4