基于面向对象程序的一种波动分析方法及其实现

A Kind of Ripple Analysis Approach and It's Implementation Based on Object-Oriented Program

董志宏　李必信　郑国梁

（南京大学计算机软件新技术国家重点实验室　南京210093）

（南京大学计算机科学与技术系　南京210093）

Abstract　Ripple Analysis is a means to aid programmer for programming and program understanding. New feature of Object-Oriented language brings new ripple effect as well as new object to be analysed. In this paper, ripple effect based on level of member-method and an approach for ripple analysis are discussed. Then, the principle and implementation of our analysis tool are briefly introduced.

Keywords　Ripple, Ripple analysis, Method dispatch table, Class relation diagram, Ripple diagram, RAT

"波动"一词从程序设计出现就已经产生。在程序中，各个语句之间不是孤立的，一个语句执行后的不同结果可能使另一个语句执行时得到不同的结果，或者决定了另一个语句执行或者不执行，这两种情况都称前一个语句影响了后一个语句。那么，当修改程序的某个部分（语句）时，可能潜在地影响到程序的其它部分（语句），这就是程序中存在的波动效应[1]，而找出对某个部分的修改影响波动到的程序范围也就是波动分析。在 Weiser 提出程序切片的概念以后，程序切片方法不断扩充和发展，成为波动分析的主流。本文所讨论的波动分析与程序切片有所不同，是根据面向对象语言的特点、针对类及其成员而提出，以图在程序修改的前后，找到相应方法所能波动到的类和成员，减少语法语义错误和帮助程序理解。

一、面向对象与波动分析

面向对象程序设计语言引入了类、对象、封装、继承、多态等概念，使得程序与语句之间出现了新的语法单位；同时在语义上，程序不再是由抽象的语句组成，而是由具有独立功能的类与对象构成，过程和函数也被成员方法所取代。继承关系和允引关系是类间的两种重要关系，也是软件结构化的重要机制。继承作为一种模块扩充机制，能通过增加或修改已有类的特征去定义新类；作为一种类型精化机制，能通过例化已有类型去定义新类型。允引关系则是通过引用类的变量说明定义的，使一个类可以通过另一个类的接口去使用该类的特征。这两种关系具有共同之处，就是一个类定义时使用了另一个已定义的类。分析程序中各个部分的相互影响时，就不能不把这些新机制考虑进来。面向对象的程序切片方法增加了三种依赖关系：选择依赖、同步依赖和通信依赖，把切片技术应用于面向对象的系统相关图和过程相关图中[3]，实现了对类方法中的语句的波动分析，但没有把类及其成员方法和成员变量单独作为研究对象，实际上对这些部件的分析是很有必要的。我们以一段简单的 C++ 程序为例（成员方法具体实现略）：

例1

```
classA
{
  void f1(){…};
};
class B:public A
{
  void f2(){…};
}
class C:public B
{
  void f1(){…};
  void f3(){…};
}
class D
{
  void f4();
};
class E:public D
{
};
void D::f4()
{
  B::f1();
}
```

根据类继承关系，我们可以建立程序执行时类成员方法的分派表[2]（即可施于各类的方法）如下：

	A	B	C	D	E
f1	A:f1	A:f1	C:f1		
f2		B:f2	B:f2		
f3			C:f3		
f4				D:f4	E:f4

如表中的 f1方法，可以在类 A、B、C 中调用，如果 A 中的 f1方法被修改了，A 的子类 B 中的 f1方法也随之更改，类 C 则因为重新定义覆盖了父类继承的方法 f1而不受影响。对调用了 B::f1和 D::f4而言，若修改的是 A::f1的实现，则 A::f1的作用可能与定义 D::f4时已有所不同，可能与程序员的最初设计不符，导致程序员必须重新考虑D::f4的实现；若修改的是A::f1

的程序头，即方法名、形式参数或返回值，则 D::f5处调用接口不一致，必定产生一个语法错误，又如类 B 中的 f2方法，即便从未被调用过、不会对程序的结构和结果产生影响、也可能因其修改而使该方法不再适用于子类 C，甚至改变了类 B 的性质而使其在语义上不能作为 C 的父类。这些情况就是我们所研究的面向对象程序中的波动效应。

面向对象程序中引起此类波动的元素包括类成员方法、类成员变量、类继承关系等类定义层次的部件，其中最主要的是类成员方法。类成员变量和类继承关系等没有程序体实现，不会有因调用其它部件而引起的波动问题。所以对它们的波动分析是类成员方法波动分析的子集。在这里我们主要以类成员方法为研究对象讨论波动分析。

为便于描述波动问题，我们定义一些符号与谓词，以给出基于面向对象语言类成员方法的波动的形式化定义：

A:a：对类 A 可合法调用的成员方法 a。

S(a,A)：a 是在类 A 中定义并实现的成员方法。

I(A,B)：类 A 是类 B 的直接子类。

C(A:a,B:b)：类 A 的成员方法 a 被类 B 的成员方法 b 所直接调用。

定义1 令集合 SC 为程序中的所有类的集合，集合 SR 为类 A 的成员 a 改动后被波动影响的类成员方法的集合，集合 SR 中的元素满足这样的条件：i)A:a∈SR；ii)∀ E:e(E:e∈SR)，∀ D(D∈SC∧I(D,E)∧¬S(e,D))⇒D:e∈SR；iii)∀ E:e(E:e∈SR)，∀ D(D∈SC∧C(E:e,D:d)⇒D:d∈SR。

二、波动分析方法与实现

面向对象程序中容易引起波动的主要是继承关系和允引关系。C++中有单继承和多继承，单继承在程序的类结构中可以用树表示，多继承则表现为无环有向图。在面向对象语言中对引用基本没有限制，甚至可以循环引用，只能用没有规律的多重有向图表示允引关系。前面例1中各个类的相互关系组成了类关联图1，方法 A::f1的波动范围则显示在图2中。

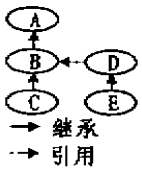


图1 类关联图

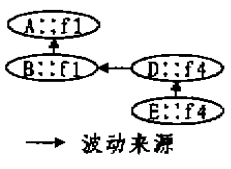


图2 A::f1的波动图

继承关系和允引关系往往交替出现，在类与方法数量较多的时候很难用一种简单有效的模型分析其波动效应，只能对它们采取分别处理、结果综合、多次求解、递增构图的策略。

对程序中的继承和允引关系分别用继承和引用图来表示，但图形只适合人工波动分析，在计算机辅助分析中，继承图和引用图可以转化为继承关系表和引用关系表中的数据。另外，由于波动图[1]中的类只是类关联图的子图，完整的类关系图对构造波动图是没有作用的，并且从待分析程序得到的是直接类继承关系和直接方法引用关系，我们没有必要保存整个类继承层次和引用关系数据，只须建立类直接继承关系表和直接方法引用关系表以分别对应前面定义的谓词 I(A,B)和 C(A:a,B:b)。在类直接继承关系表中，"1"表示该单元所在行对应的类是该单元所在列对应类的直接子类，"0"表示该单元所在行对应的类不是该单元所在列对应类的直接子类。在方法直接引用关系表中，"1"表示该单元所在行对应的类成员方法被该单元所在列对应类成员方法所直接调用，"0"表示该单元所在行对应的类成员方法从未被该单元所在列对应类成员方法直接调用。例1中的类直接继承关系表和方法直接引用关系表如表1和表2。

表1 类直接继承关系表

	A	B	C	D	E
A	0	0	0	0	0
B	1	0	0	0	0
C	0	1	0	0	0
D	0	0	0	0	0
E	0	0	0	1	0

表2 方法直接引用关系表

	B:f1	D:f4
B:f1	0	1
D:f4	0	0

另外，方法覆盖(OverRide)的引入，使沿继承层次的波动可能被子类重定义的方法所屏蔽，如例1中的类 C，它是被修改方法 f1所在类 A 的间接子类，本应包括在波动图(图2)中，但由于类 C 中重新定义了方法 f1，故对类 C 调用方法 f1时调用的是重新定义的方法而不是从父类中继承到的方法，从而将类 C 排除在 A::f1的波动范围之外。可见必须考察方法的多重定义点，以便在构造波动图时对搜索路径进行剪枝，这就需要建立方法定义点分布表。方法定义点分布表对应了前面定义的谓词 S(a,A)，其中，"1"表示该单元所在行对应的方法在该单元所在列所对应的类中有定义(实现)，"0"表示该单元所在行对应的方法在该单元所在列所对应的类中无定义(实现)。

表3 方法定义点分布表

	A	B	C	D	E
f1	1	0	1	0	0
f2	0	1	0	0	0
f3	0	0	1	0	0
f4	0	0	0	1	0

建好上述几个表，就可以按照下面的步骤开始构造指定类成员方法的波动图了：

步1 定义空集合S，并将给出或扫描程序代码得到的待分析成员加入集合中；

步2 根据集合S中新增加且未经步2的成员方法A:f，对照方法直接引用关系表中的数据，将该成员方法所对应的行中为"1"的单元所在的列所对应的类成员方法加入集合中；

步3 根据集合S中新增加且未经步3的成员方法A:f所属的类A，对照直接类继承关系表中的数据，找到该类所对应的行中为"1"的单元所在的列所对应的类B，如果在方法定义分布表中，f所对应的行与B所对应的列确定的单元数据为1，则转至步4，否则将成员方法B:f加入集合；

步4 如果集合S不再增加元素，转至步5，否则继续步2～步4；

步5 波动分析结束，输出集合中的成员方法。

至此，我们就得到了指定成员方法的波动范围——集合S。

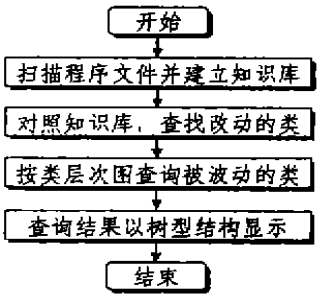
前面提到，面向对象程序波动分析的研究对象不止于类成员方法，也包括类成员变量和类继承关系。不难发现，从对成员方法的波动分析路径中抽取有关继承和被调用的内容，就可以得到对其它部件的分析方法，因此这里不再对其它内容做详细讨论。

三、波动分析工具简介

为了帮助用户理解和维护面向对象的程序，我们开发了工具集OOPSE(面向对象程序的理解与支撑环境)，用于理解基于标准C++语言的面向对象程序设计软件理解工具，波动分析工具RAT就是其中具有相对独立功能的辅助开发工具，用Visual C++语言在Windows操作系统上实现，所研究的对象也是针对C++语言的语法规则，把标准C++语言程序文件作为输入，经过扫描、分析，最终生成体现该程序波动分析结果的图表和报告。

RAT工具可细分为波动分析和波动检测两个部分，波动分析属于对用户动作的事前分析，针对用户给出的特定类与方法，分析波动可能影响的类与方法，具有指导性作用；波动检测属于事后分析，它通过对程序代码扫描，自主定位已被修改的类与方法，并找到相应的波动范围，为用户了解自己行为的效果提供帮助，具有监督作用。

RAT的基本思想是：从程序中抽取出类定义信息，存入知识库，根据知识库中类的继承部分和引用部分，将整个系统类关系构造成类似于"格"的层次结构，基类或允引类位于上层，衍类或引用类位于下层；从新旧知识库信息的对照中，找到发生变动的类及变动成员，根据变动的不同类型，分别沿继承链或调用链列出所有受到影响的类与成员，并用成员的重定义点对搜索路径树进行剪枝；最终结果以树型图的形式显示给用户(即以引起波动的类为根，受到波动的类为枝叶)。波动检测工具的程序实现简要流程如下：



结束语 波动分析在程序的设计与维护中起着重要的作用，但波动依赖于编程语言的语法结构，故无法对所有的语言用单一的方法来解决，而只能针对不同语言的特点分别处理，有时也要与其它的程序分析工具相结合。本文所讨论的波动分析就是在做C++语言的程序理解时提出并实现的。波动分析工具RAT目前还属于原型工具，它没有把影响范围精确到每个语句，也没有实现自动修改或给出建议修改方案，用户仍然要在分析结果提示的类中手工纠正错误，这是此工具的不足之处，也是我们这个项目的一个发展方向。

参考文献

1 杨洪，徐宝文. PSS/Ada程序切片与波动系统的设计与实现. 计算机研究与发展，1997，34(3)：217～222

2 Holst W, Szafron D. Incremental Table-Based Method Dispatch for Reflective Object-Oriented Languages. (TOOLS23). Santa Barbara, California, 1997 63～74

3 李必信，郑国梁. 软件理解研究与进展. 计算机研究与发展，1999，36(8)：897～906

4 Pressman R S. Software Engineering: a practitioner's approach, 1992

5 Ducasse S, Blay-Fornarino M. A Reflective Model for First Class Dependencies. (OOPSLA'95). New York: ACM Press, 1995. 265～280