基于OOP和AOP的软件产品线实现技术研究

祝家意 彭 鑫 赵文耘
(复旦大学计算机科学技术学院 上海 200433)

摘 要 作为目前最为主流的软件开发技术,面向对象的编程OOP(Object-Oriented Programming)对于软件产品线可变性的实现提供了一定的支持。但OOP对于具有横切特性的产品线特征以及可选的特征交互关系仍然难以提供灵活、有效的支持,因此一些相关研究者将面向方面的编程AOP(Aspect-Oriented Programming)引入到产品线实现方法中。AOP不仅能分离横切的关注点,而且还通过依赖关系分离的方式为可选交互关系的灵活配置提供了有力的支持。显然,结合OOP和AOP这两种技术实现的产品线系统将具有更高的可复用性、灵活性和可配置性。在相关问题分析的基础上对基于OOP和AOP的产品线实现技术进行了研究,并通过一个酬金发放系统产品线的实例分析对相关方法进行了验证和分析。

关键词 软件产品线,可变性,实现,面向对象编程,面向方面编程
中图法分类号 TP311.5 **文献标识码** A

Combining Object-oriented Programming and Aspect-oriented Programming for Software Product Line Implementation

ZHU Jia-yi PENG Xin ZHAO Wen-yun
(School of Computer Science, Fudan University, Shanghai 200433, China)

Abstract As one of the most popular software development technology, OOP(Object-oriented Programming) does provide certain mechanisms for the implementation of software product line variabilities. However, OOP does not support crosscutting features and optional feature interactions well. Therefore, some researchers introduced AOP(Aspect-oriented Programming) to the implementation of software product line. AOP can not only separate crosscutting concerns, but also provide flexible supports for configuration of optional feature interactions through separation of dependencies. Therefore, combining OOP and AOP in product line implementation can greatly promote the reusability, adaptability, and configurability of product line assets. This paper explored the OOP and AOP combined implementation method for software product line on the analysis of related problems, and then presented a case study on a reward offering software product line for validation with related analysis and discussion.

Keywords Software product line, Variability, Implementation, OOP, AOP

1 引言

软件产品线是共享一组受控的公共特征并且在一组预定义的公共核心资产基础上开发而成的一系列软件应用系统[1]。软件产品线工程主要包括领域工程和应用系统工程两大阶段,其中领域工程是产品线核心资产的开发阶段,而应用系统工程阶段的任务则是在产品线核心资产的基础上实现应用产品的复用式开发。由此可见,在领域核心资产的基础上实现高质量和高效的应用产品生产是软件产品线工程的基本目标。

实现软件产品线开发目标的基础是面向产品线范围内所有应用产品的全面性的共性和可变性分析及设计。所谓软件产品线的可变性是对产品线范围内不同应用产品间差异性进行抽象的结果。在目前已经存在的一些产品线开发方法中,需求级的可变性分析、建模以及面向应用的定制已经得到了较好的支持,例如最典型的是基于特征的领域分析和建模方法。变化点和它们之间组合的规则在实现层次必须要同样反映出来,并且在创建一个应用产品时对它们进行实例化[2]。然而,由于问题空间和解空间之间所存在的巨大鸿沟,如何将需求层面(例如特征模型)上的可变性分析结果映射为可变性的体系结构设计和代码实现仍然是一个难题[3]。

在当前的软件开发中,主流的开发技术仍然是面向对象的编程OOP(Object-Oriented Programming)。然而,OOP在横切特征以及可变性的实现上存在许多不足。而面向方面的编程AOP(Aspect-Oriented Programming)则由于其在横切特征以及适应性和可配置性上的优势,在作为软件产品线实现技术方面得到了越来越多的关注[3]。OOP对于可变性提供了继承、重载以及反射等实现机制,而且软件产品线的主要部

到稿日期:2009-01-16 返修日期:2009-03-20 本文受国家863计划(2007AA01Z125),国家自然科学基金(60703092)资助。
祝家意(1986-),男,硕士生,主要研究方向为软件产品线;彭 鑫(1979-),男,博士,讲师,主要研究方向为软件产品线、软件体系结构等,E-mail:pengxin@fudan.edu.cn;赵文耘(1964-),男,教授,博士生导师,主要研究方向为软件工程。

分还是依赖于 OOP 进行实现。AOP 则能够在可变性实现的灵活性和模块化程度等方面对 OOP 进行良好的补充。因此，OOP 与 AOP 结合的实现技术是目前较为现实的一种软件产品线实现方法。本文通过一个酬金发放系统的产品线开发实例对基于 OOP 和 AOP 的产品线实现技术进行研究。本文的后续部分将首先介绍产品线可变性、典型的可变性实现技术以及仍然存在的问题。然后，本文将对 OOP 与 AOP 结合的产品线实现技术进行探讨，并通过酬金发放系统产品线开发实例对相关方法和技术的应用进行分析和讨论。

2 背景介绍

2.1 产品线可变性

典型的产品线的可变性类型分为“可选”(optional)、“多选一”(alternative)和“或”关系(or)3 种。可选指应用产品可以选择绑定或者不绑定这个变体。“多选一”表示应用产品必须要在一组变体中选择一个进行绑定。“或”关系表示应用产品可以在一组变体中选择 0 个或者多个变体进行绑定。可变性在软件模型以及开发制品(例如特征模型、体系结构以及构件等)中是以可变点和变体的形式出现的。可变点是现实世界中的可变性在各种软件开发制品中的具体体现，而变体则是软件开发制品中各个可变点上各种特定的可变性实例[4]。

例如图 1 所示的特征模型是对酬金发放领域共性和可变性需求的模型化描述。这个产品线实例将贯穿本文的各个部分。酬金发放系统是企事业单位为职工和其他人员(例如学生和临时聘用人员等)进行工资之外的酬劳发放的系统，基本功能包括酬金录入、系统审核、转卡发放等。由于财务管理模式上的不同，各个用户单位在酬金录入方式、发放控制等方面存在差异。该产品线的特征模型如图 1 所示，其中“发放清单提交”是系统的公共特征，而“酬金录入”、“安全设置”、“项目关联”和“项目余额控制”是系统的可变点。

1) “酬金录入”是多选一类型的可变点，系统提供两个变体：一种是“单笔录入”，即每次只录入一条酬金记录；一种是“Excel 成批导入”，录入人员将酬金记录写入到 excel 表格当中，导入时系统提取所有记录并提交。

2) “安全设置”是“或”类型的可变点，其下面包含“非自动显示姓名”、“隐藏卡号”和“禁止修改卡号”3 个变体，任何一个应用产品都可以从中选取 0～3 种进行绑定。“非自动显示姓名”是指在“单笔录入”时，发放人员输入发放对象工号后相应的姓名不自动显示，而是与输入的姓名进行比对，比对一致才允许发放。“隐藏卡号”是为了保护职工的卡号信息，当出现职工卡号时隐藏卡号的后 6 位。“禁止修改卡号”指禁止录入人员修改职工的卡号，修改卡号是一个危险的操作，但是在某些特别的情况可能会需要。

3) “项目关联”是可选类型的可变点，是指酬金是否要与特定的项目经费挂钩。而“项目余额控制”则会进一步根据项目经费的当前余额控制酬金发放，如果酬金发放总额超过经费余额则不允许。

特征模型中的可变特征并不是完全独立的。它们之间存在着约束关系，这种约束是一种静态依赖关系[5]。基本的约束关系包括 requires 和 excludes 两种，前者表示正向的依赖关系，而后者表示互斥(排他)关系。除了静态的约束关系，特征之间还可能存在体现特征运行时交互的动态依赖关系[5]。这种交互关系如果与可变特征相关，那么一定要加以明确的描述，因为它们会影响可变性定制后的程序级组装。在图 1 所示的特征模型中，可选的“非自动显示姓名”会影响“单笔录入”的运行轨迹：如果绑定了“非自动显示姓名”这个变体，则在“单笔录入”时会跳过，取被发放酬金的员工的姓名操作。可选的“项目余额控制”会影响“发放清单提交”：如果项目余额检查不通过，则本次清单提交将被禁止。

2.2 典型的可变性实现技术

作为当前主流的开发技术，OOP 对于产品线可变性的实现提供了一系列的支持。属于 OOP 范畴的可变性实现技术包括继承、重载、代理、参数化、反射和设计模式等[2]。继承和重载是面向对象程序设计的主要特征。通过继承，可变性的部分能够从基础程序中分离出来。而重载则可以根据参数类型不同选择不同的操作。代理使得一个对象利用代理对象实现某个方法，但这种方法在实现多选一可变点时会出现困难，因为会涉及到代理对象方法选择的问题[2]。参数化的方式几乎能实现所有类型的可变点，生成产品线应用产品的时候只需要通过定制就能自动生成应用产品。但是在现实中产品线系统往往很庞大，单纯使用参数化方式实现系统可变点将使参数的规模变得很庞大，程序可读性和可维护性差，系统行为间交互变得很复杂。反射是指程序能在运行时动态决定它的行为，因此可变点能够在运行时实现动态的绑定。设计模式强调重用重复出现的结构设计主题。比如说设计模式中的工厂方法模式就能够根据程序的需要创建不同的对象。

2.3 基于 OOP 的可变性实现问题

传统的 OOP 设计方法有很多优点，如可读性强、易维护、可扩展性强等，能够为可变性的实现提供一些支持。例如图 1 所示特征模型中的“酬金录入”这个可变点就可以通过面向对象设计模式来实现(见 4.1.1 节)。但是也有一些产品线可变性在使用 OOP 实现时会出现问题。例如可选的“隐藏卡号”对于“单笔录入”和“excel 成批导入”两种酬金录入方式都有影响。如果使用 OOP 实现的话，“隐藏卡号”这个变体的实现将会分散到“单笔录入”和“excel 成批导入”两个变体中。此外，对于可选的动态特征交互，OOP 只能以固化的条件判断语句实现。例如“项目余额控制”时如果酬金发放的金额大于项目的余额会中止基础程序提交的操作，用 OOP 实现这个变体只能有两种方法：一种是在“发放清单提交”这个必选的公共特征中加入条件判断语句然后通过读取配置文件等其它信息来实现；另一种是通过对基础程序进行入侵式的修改实现。但是无论是加入条件判断语句还是对基础程序入侵式修改，都会影响产品线的可重用性和可移植性。

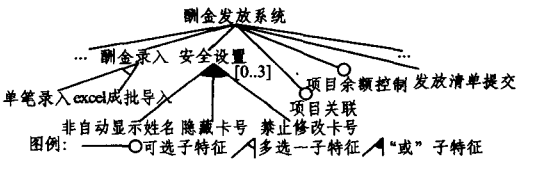

图 1 酬金发放系统的特征模型图

出现上述这些问题的主要原因是由于 OOP 缺乏对系统横切属性模块化的能力以及与可变性相关的动态依赖关系分离的机制。AOP 正是在这些情况下对 OOP 的良好补充。AOP 技术大大提高了软件开发的模块化(尤其是针对横切方面)程度，而这一改进主要源于 AOP 中的两大特性：多量化

(Quantification)和不知觉性(Obliviousness)[6]。

3 OOP 和 AOP 结合的产品线可变性实现方法

由于 OOP 对于产品线可变性的实现能够提供一定的支持,而且 AOP 目前在很大程度上仍然被认为是 OOP 的一种补充,因此,我们认为产品线可变性实现仍然应该以 OOP 为主,并在某些特定情况下采用 AOP 进行补充。一般而言,当产品线可变点属于以下情况时,可以用 OOP 中的继承、重载、反射、设计模式、代理等机制来实现。

1)对于多选一类型的可变点,往往可以使用工厂模式等面向对象设计模式来实现。因为对于多选一类型可变点所在的实现单元而言,它不需要了解多选一可变点具体绑定了哪一个变体。具体的实现绑定可以由工厂对象借助 OOP 中的反射等机制来实现。这种信息隐藏技术将使二者的依赖关系能够最小化。

2) 变体属于可选的或"或"类型的可变点时适合于用代理的机制实现,当一个对象使用可选的或者或类型变化点中的变体时,代理机制隐藏了变体绑定的细节。

同时,当产品线变体属于以下情况时,可以通过 AOP 来实现:

1) 变体横切系统或者变体的实现涉及系统的多个模块。这是由于 AOP 内在属性决定的。用 AOP 实现这种变化性能使系统更加模块化,一个变体的实现不会影响基础程序或者分散到其它变体中。

2) 变体与其它变体或基础程序有体现运行时交互的动态依赖关系并且在依赖关系中是源变体。有变体参与的动态依赖关系由于会影响到应用产品定制后的程序级组装,因此这种依赖关系应该被明确地描述。同时一个依赖关系总是关联到两个对象。在图 1 特征模型图中,变体"项目余额控制"和"发放清单提交"就有一个动态的依赖关系。我们称"项目余额控制"为源对象,"发放清单提交"为目标对象。目标对象可以保持对依赖关系的不知觉性。比如在这个依赖关系中,"发放清单提交"可以对此关系保持不知觉性。因此考虑将这种依赖关系在源变体的方面中实现,这样依赖关系更加明确化。同时当依赖关系的目标对象是基础程序时,由于依赖关系实现在源变体的方面里,这样避免了绑定此变体时对基础程序入侵式的修改。当依赖关系的双方都为变体时,也避免了绑定一个变体对另一变体入侵式的修改。

3) 变体的实现需要向已有的基础程序类中引入新的简单变量或聚合对象。AOP 所提供的 Introduction 机制使得程序可以在变体被绑定的时候通过编织机制将相关的变量或对象加入到目标类中。这种实现方式保持了相应基础程序类的"不知觉性",即基础程序类不需要了解这些与特定变体相关的实现单元。

4 实例分析

这一部分将通过酬金发放系统产品线(简化的特征模型如图 1 所示)的实现对基于 OOP 和 AOP 的产品线实现方法进行实例分析。该产品线属于基于 Java 的 Web 系统,AOP 则使用 AspectJ 实现。

4.1 基于 OOP 和 AOP 的酬金发放产品线实现

4.1.1 酬金录入

"酬金录入"属于多选一类型的可变点,相关实现单元与其它部分的交互关系确定不变,可变的仅仅是"酬金录入"本身的实现方式(单笔还是批量)。此外,酬金录入功能在可预见的未来也不太可能会增加新的变体,即变化模式基本上是确定的。因此,这个可变点可以完全借助于 OOP 中的继承和反射等机制实现,这里用工厂模式实现。如图 2 和图 3 所示,分别实现"单笔录入"和"excel 成批导入"方式的 NormalInput 和 ExcelInput 类定义为 Input 类的子类。外界通过 InputFactory 类的 getInput 方法获得实际的酬金录入类对象,而该方法将利用 Java 反射机制判断输入参数的类型并返回对应的酬金录入类对象。如果是文件类型,则返回 ExcelInput 类对象,否则返回 NormalInput 类对象。

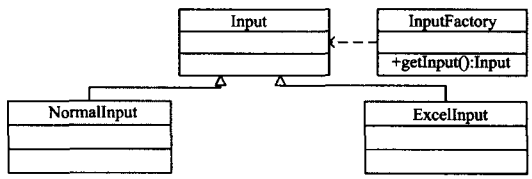

图 2 基于工厂模式的酬金录入可变点实现类图

```
1. public class InputFactory{
2.          ……
3.          public Input getInput(Object entry){
4.                    if(entry instanceof File)
5.                      return new ExcelInput(entry );
6.                    else
7.                      return new NormalInput(entry );
8.          }
9. }
```

图 3 基于工厂模式的酬金录入可变点实现代码

4.1.2 安全性设置

1)非自动显示姓名

可选的"非自动显示姓名"会影响"单笔录入":如果绑定了"非自动显示姓名"这个变体,则在"单笔录入"时会跳过,取被发放酬金的员工的姓名操作。因此"非自动显示姓名"与"单笔录入"有动态依赖关系并且是源变体。所以根据第 3 节的方法应将其实现为一个方面。系统默认为自动显示姓名,所以这个方面如果绑定的话将会影响基础程序的执行轨迹,即跳过基础程序里 getNameById()方法。其 pointcut 和 advice 的定义如图 4 所示。第 4 行表示这个方面只横切在"单笔录入"执行轨迹上对 getNameById()的调用。第 5 行表示其横切的方式为 around,因此在 advice 里可以决定是否继续执行 getNameById()方法,这里返回空值表示跳过 getNameById()方法即姓名显示为空,需要用户输入进行验证。

```
1. Pointcut getName(String id):
2.          call( * getNameById(String))
3.                    &&args(id)
4. &&cflow(execution( * NormalInput. NormalInput(..)));
5. String around(String id ):getName(id){
6.                    return "";
7.                    //返回空值,表示姓名不会自动显示
8.                          }
```

图 4 非自动显示改名变体 Pointcut 和 Advice

2) 禁止修改卡号

当“禁止修改卡号”绑定时，所有对卡号修改操作将不会成功，因此“禁止修改卡号”会更改基础程序的执行轨迹，根据第3节的方法应将这种动态依赖关系的源变体实现为一个方面。如图5第5行所示，对修改卡号的函数调用将不会执行并且返回为False。

```
1. aspect ForbidModifyCardNo{
2   pointcut chgCardNo();
3.                call(* chgCardNo(..));
4.   Boolean around():chgCardNo(){
5.                    return False;
6.                }7.}
```

图5 禁止修改卡号方面具体实现

3) 隐藏卡号

“隐藏卡号”对于“单笔录入”和“excel成批导入”两种酬金录入方式都有影响，如果使用OOP实现的话，“隐藏卡号”这个变体的实现将会分散到“单笔录入”和“excel成批导入”两个变体中。根据第3节的方法，这种实现涉及多个模块的变体应使用方面来实现；这里略去具体实现细节。

4.1.3 项目关联

1) 项目关联

“项目关联”表示每一条酬金数据将与一个具体的项目号关联，这样可以方便进行项目余额检查以及经费管理。这个变体需要在Input类中引入变量projectNo，同时在每一次发放清单提交时，增加一个保存项目号的操作；这里略去实现细节。

2) 项目余额控制

“项目余额控制”是在酬金与项目关联的时候，当预发放的酬金数据超过项目余额时不允许发放。由于余额检查影响基程序的运行轨迹，因此应用方面来实现。Advice的实现如图6所示。第4行表示当酬金数据没有超过项目余额时，则继续提交当前酬金数据。

```
1. void around(Inputer p):call(* submit(..)&&target(p)){
2.         /*检查项目金额是否超支*/
3.         If(! result)
4.           Proceed(p); //如果没有超支则继续提交
5.         else
6.             /*提示项目超支*/
7.             //不调用Proceed(p)则会终止submit()调用
8. }
```

图6 项目余额检查的Advice具体实现

4.2 实现分析

本节将对本文提出的方法进行评价。复用性和可移植性是软件产品线非常重要的两个目标。具体到产品线来说，复用性表示产品线大部分的核心资产能在应用产品中得到复用，可移植性表示产品线的可变点本地化而且是一种附加的方式而不是入侵的方式对产品线公共部分资产进行更改[7]。本方法首先将产品线的公共部分和可变部分分离出来；同时将动态依赖关系实现在源变体中，这样依赖关系将直接与源变体关联，目标对象可以对这种依赖保持不知觉性。因此当目标对象是基础程序时，依赖关系的实现可以不通过对基础程序一种入侵式的修改来实现。当目标对象是变体时，依赖关系对这个变体没有任何约束。至于可配置性，AOP技术不仅能使产品线的可变部分和不变部分彻底分离，而且将产品线的各种动态依赖关系的影响范围降到最小，因此增加了产品线系统的可配置性。

4.3 讨论

本方法之所以能够使得基础程序与可变点之间彻底分离，主要是利用到了AOP的一些特殊的机制，在不修改基础程序的情况下对基础程序状态和行为进行修改。如introduce引入新的变量或方法，around类型的advice能够决定基础程序的运行轨迹等。因此在不修改基础程序代码情况下，改变基础程序的能力决定了能多大程度上实现产品线的灵活性。在本方法的实现中，虽然AOP提供以上这些良好的机制，但还是不能完全满足产品线实现的需要。比如说在实现“禁止修改卡号”变体时，基础程序有一个Text类型的变量编辑卡号，所以需要横切修改卡号这个函数禁止编辑卡号。更合理的解决方案应该是方面里定义一个Label类型变量来覆盖基础程序的Text变量，这样用户看到的将是灰色的不可编辑框。但是当前的AOP并没有提供这种机制。

结束语 由于良好的可读性、可维护性，OOP技术必然构成产品线实现技术的重要部分。但是在产品线实现过程中，可变性的实现对实现技术提出了一些新的要求。比如说要使横切的可变性横块化、一个变体的实现不能对另一变体或者基础程序进行入侵式的修改，即依赖关系影响最小化等。这些要求在OOP中很难实现。于是AOP技术由于其横切属性和其它机制的优势成为OOP在以上这些情况下的一个补充。因此我们提出了一种结合OOP和AOP的产品线实现方法来提高其可读性、可维护性、可配置性。最后我们使用酬金发放的产品线系统成功地验证了此方法。

参考文献

[1] Clements P, Northrop L. Software Product Lines: Practices and Patterns[M]. 张莉，王雷，译. 北京：清华大学出版社，ISBN 7-302-07932-3/TP.5757

[2] Gacek C, Anastasopoules M. Implementing product line variabilities[C]//Proceedings of the 2001 symposium on Software reusability: putting software reuse in context. SSR2001

[3] Peng Xin, Shen Liwei, Zhao Wenyun. Feature Implementation Modeling based Product Derivation in Software Product Line[C]//Proceedings of the 10th International Conference on Software Reuse. ICSR2008

[4] Pohl K, Bockle G, van der Linden F. Software Product Line Engineering: Foundations, Principles, and Techniques[M]. Springer Berlin Heidelberg New York, ISBN-10 3-540-24372-0

[5] Zhang Wei, Mei Hong, Zhao Haiyan. Feature-driven requirement dependency analysis and high-level software design[J]. Requirements Eng, 2006, 11: 205-220

[6] 莫倩，王恺，刘冬梅，等译. Aspect-oriented Software Development[M]. 机械工业出版社，ISBN 7-111-17533-6

[7] Lee K, Kang K C, Minseong Kim, et al. Combining Feature-Oriented Analysis and Aspect-Oriented Programming for Product Line Asset Development[C]//Proceedings of the 10th International on Software Product Line Conference. SPLC2004