面向特征编程中的特征组合失效问题及其解决方法

陈志聃 沈立炜 赵文耘
(复旦大学计算机科学技术学院 上海 201203)

摘 要 软件产品线的特征之间存在依赖关系,因此在面向特征编程(FOP)中,特征模块之间在代码结构上存在密切关联。另一方面,具有可变性的特征在应用产品中的绑定与否会对依赖关系的实现造成破坏性的影响,导致FOP在实施过程中可能出现特征组合失效问题。对该问题的产生进行分析,总结出3种主要的依赖场景。另外,提出一种特征模块垂直分解方法,其核心机制在于将可变性引入特征模块内部,根据需求组装实现代码,从而可避免出现组合失效问题。最后,通过一个出版社利润考核系统产品线实例验证了方法的有效性。

关键词 面向特征编程,软件产品线,特征模块组装,特征依赖
中图法分类号 TP311.5 **文献标识码** A

Feature Composition Failures and its Solution in FOP

CHEN Zhi-dan SHEN Li-wei ZHAO Wen-yun
(School of Computer Science, Fudan University, Shanghai 201203, China)

Abstract There exist dependencies between software product line features, thus the feature modules in feature-oriented programming (FOP) is closely related in the code or structure level. On the other hand, whether the variable features are bound in the applications has destructive impact on the implementation of the feature dependencies, causing the potential problem of feature composition failures during FOP process. This paper analyzed the problem and concluded three main dependency scenarios, besides, proposed a vertical decomposition method for feature modules to solve the problem. Its key mechanism is to introduce variability into the inner part of feature modules, thus the problem can be avoided by composing the codes according to the specific requirements. Furthermore, the method was applied on a software product line of publishing-house profit evaluation systems to validate its effectiveness.

Keywords Feature-oriented programming, Software product line, Feature module composition, Feature dependency

1 引言

软件产品线方法[1]是开发一系列领域应用系统的有效途径。它通过领域工程活动创建可复用的产品线核心资产,并通过应用系统工程活动定制、生成符合特定应用需求的软件产品[2]。

在软件产品线的研究工作中,基于特征的方法已被广泛应用。特征是软件系统的功能单元,用于对系统功能进行抽象描述,是用户和开发人员都能理解的软件开发中的一阶实体[14]。基于特征的领域分析方法通常采用特征建模的方式对软件产品线的共性和可变性进行分析。因此,领域内的应用产品将共享一组通用的、必选的特征,而这些产品之间的差异则被描述成为具有可变性的特征。通常而言,可变性特征可分为可选(optional)、多选一(alternative)与多选多(or) 3种类型。"可选"意味着应用产品可以绑定或不绑定该特征。"多选一"表示存在一个特征集合,应用产品必须且只能绑定其中一个特征。"多选多"则表示应用产品至少绑定特征集合中的一个特征[7,8]。

在实现技术方面,面向特征编程(Feature-oriented Programming,FOP)可以被用来开发软件产品线[14]。FOP通过特征精化(Feature Refinement)对特征进行封装,特征精化封装的不是类的集合,而是一系列类的片段[5]。特征精化的结果通常用层(layer)来描述,每一层对应一个特征,封装与该特征相关的所有代码,形成独立的特征模块。因此,FOP是一个增量开发和逐步精化的过程,它以基程序(Base Program)为基础,将特征看作是一个个增量的模块,并以此形成软件产品线的领域核心资产;在应用系统工程阶段,FOP根据应用需求绑定产品线的可变性,选取相应的特征增量,以逐步组合的方式将特征模块组装到基程序上,从而得到最终产品。

软件产品线的特征之间存在依赖关系(Feature Dependency),使得FOP的各层次之间在代码结构上存在密切关联。另一方面,具有可变性的特征(尤其是可选特征)在应用产品中的绑定与否会对层间的依赖产生破坏性的影响。因此,面向特征编程FOP在实施过程中可能会出现特征组合失效的问题。举例而言,在特征选择时,如果一个可选特征未被绑定,并且该特征被其它已绑定的特征依赖,那么经过组装的最终产品会遗漏部分实现代码,从而引起应用系统的类型错误。

到稿日期:2012-09-20 返修日期:2012-11-30 本文受国家"八六三"高技术研究发展计划项目(2011AA010101)资助。
陈志聃(1987-),男,硕士生,主要研究方向为软件产品线,E-mail:zhidanchen@gmail.com;沈立炜(1982-),男,博士,讲师,主要研究方向为软件复用、软件产品线;赵文耘(1964-),男,教授,博士生导师,CCF高级会员,主要研究方向为软件工程、企业应用集成。

针对该问题背景，本文将详细分析 FOP 中的特征组合失效问题，将产生该问题的场景归类为使用依赖、修改依赖与激活依赖这 3 种主要类型。其次，针对特征组合失效问题，本文提出了一种特征模块的垂直分解方法用以管理上述 3 种特征依赖，其核心机制在于将可变性引入特征模块内部，根据需求组装实现代码。使用该方法可有效解决特征选择对系统代码结构产生影响的问题，进一步增强 FOP 的可适应性、可伸缩性和可配置性。最后，本文将该方法应用于一个出版社利润考核系统的产品线实例，其结果验证了方法的有效性，避免了可能出现的特征组合失效问题。

本文第 2 节详细介绍 FOP 的背景；第 3 节分析 FOP 中的特征组合失效问题，并对其产生场景进行了分类；第 4 节是解决该问题的特征模块垂直分解方法；第 5 节将该方法用于一个出版社利润考核系统产品线，并对结果进行了评估；第 6 节是相关工作；最后是对本文的总结。

2 背景介绍

FOP 将产品线的开发看作表达式计算，用常量(constants)表示基程序，函数(functions)表示特征增量。一个系统被描述为常量和一系列作用于常量上的函数[5]。例如 B 表示基程序，X 和 Y 分别代表不同的特征增量，系统 P 的特征组装过程可通过式(1)进行描述：

$$P=X(Y(B)) \tag{1}$$

对于系统 P 而言，将一个特征增量(比如 Y)添加到基程序 B 上，可以完成下面两件事情。首先，特征 Y 可以添加新的类，或者向 B 中已存在的类引入新的属性和方法；其次，特征 Y 可以修改 B 中已有方法的实现。例如，图 1 是一个累加器系统的 FOP 实现示例，图 1(a)描述了该示例的层次结构，其中特征用大写字母表示。BASE 表示初始的基础累加器，图 1(b)为 BASE 的代码。COUNTER 表示计数特征，用于统计累加数据的个数，图 1(c)为 COUNTER 的代码，其中用 refines 表示特征精化的过程。COUNTER 除了引入了新的属性 ctr 和新的方法 size 之外，还修改了 BASE 中 add 方法的实现。可以用 Super 访问被修改方法的实现，来达到在已有实现的基础上增加新行为的目的。图 1(d)是累加器系统组装后的结果。

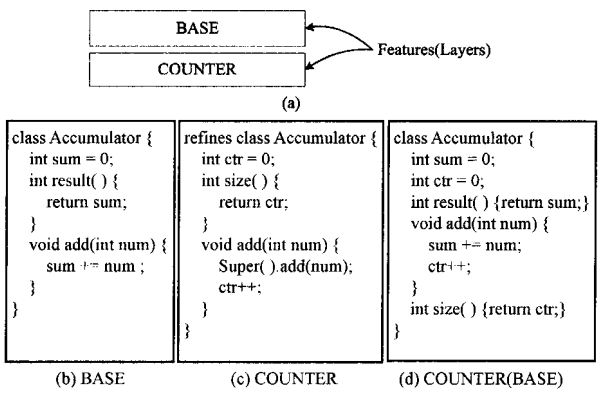


图 1 累加器系统的 FOP 实现示例

FOP 中的特征(层)可以进一步细化，一个特征通常包含一个基模块(base module)和若干派生模块(derivative modules)。基模块包含该特征新增的类以及对已有特征进行扩展而新增的属性和方法所对应的代码，每个派生模块则分别包含了该特征对某个已有特征的若干方法的修改所对应的代码部分[6]。基于该机制，使用图 2 描述了累加器系统的模块分解，其中图 2(a)是系统的模块结构。基模块用特征对应的小写字母表示，如图 2(a)中的 base 和 counter。派生模块用 $\partial b/\partial H$ 的形式表示，含义是该模块属于特征 H，并修改了特征 B 的基模块 b 中方法的实现，如图 2(a)中的∂base/∂COUNTER。因此，我们将图 1(c)中表示计数特征 COUNTER 的代码进行拆分，得到图 2(b)和图 2(c)分别对应基模块 counter 和派生模块∂base/∂COUNTER 的代码。

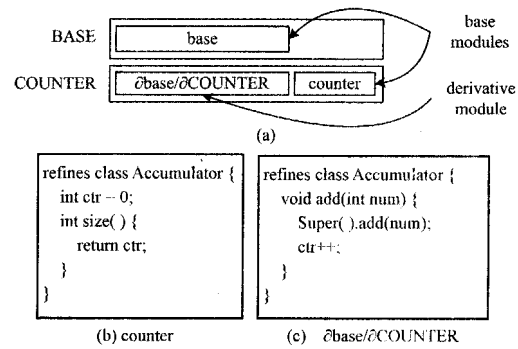


图 2 累加器系统的模块分解

另外，我们引入文献[6]所介绍的 ＋(introduction sum)和 ·(weaving)这两种标记，来形式化地表示模块组装的过程。其中，＋ 是二元操作符，用于连接模块；· 将派生模块添加到其它模块上，表示对其它模块的方法进行修改。通过＋和 · 两种操作，可以使用式(2)形式化地表示特征组装和模块表达式的关系：

$$H(B)=h+\partial b/\partial H \cdot b \tag{2}$$

累加器系统的模块表达式如下：

COUNTER(BASE) = counter + ∂base/∂COUNTER · base

3 特征组合失效问题分析

当 FOP 应用于软件产品线开发时，FOP 的层次结构中会包含具有可变性的特征(层)。由于特征之间存在依赖关系，因此在应用系统工程阶段，如果某个可变性特征未被绑定，并且该特征被其它已绑定的特征所依赖，那么最终的产品会存在类型错误，这会给系统带来破坏性的影响，从而导致了 FOP 中的特征组合失效问题。

针对这个问题，本文总结了使用依赖、修改依赖和激活依赖这 3 种导致该问题出现的主要场景。另外，由于多选一和多选多这两类可变性的语义可以被相应转换为具有约束的一组可选特征所表示的语义[16]，因此具有多选一与多选多可变性的特征在 FOP 层次中也可被转换成一组可选的层，根据应用需求决定这些层(特征变体)是否被组装在应用系统中。基于这种转换，本文所描绘的特征组合失效问题均来源于可选层是否被绑定对 FOP 组装所带来的影响。

3.1 使用依赖

使用依赖是指一个特征对其它特征所具有功能的依赖，以保证其自身功能正常的运行。如果系统中存在使用依赖，被依赖特征具有可变性，并且特征选择时没有绑定被依赖特征，而绑定的特征中存在访问该被依赖特征功能的代码，那么系统在编译时会存在类型错误。

图 3 描述了特征的可变性对使用依赖的影响。图 3(a)是该示例的模块结构，包含 A、B 和 C 3 个特征，其中 A 和 C

是必选特征，而 B 是可选特征。图 3(b)和图 3(c)分别是 B 和 C 的基模块对应的代码，其中 C 的基模块 c 中的 $f2$ 方法使用了 B 的 $f1$ 方法，表明 C 对 B 存在使用依赖。在特征选择时，如果没有绑定特征 B，那么最终的系统会存在类型错误。图 3(a)中的虚线表明特征 C 对可选特征 B 存在依赖。

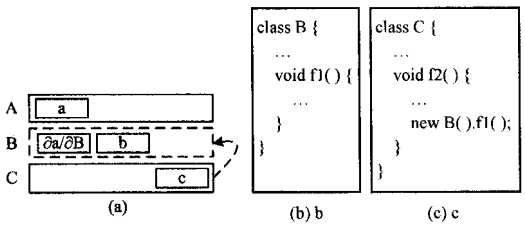


图 3　对可选特征的使用依赖

3.2　修改依赖

修改依赖是指一个特征(被修改者)的行为可以被其它特征(修改者)修改。如果系统中存在修改依赖，并且被修改特征具有可变性，则特征选择时若绑定了修改者但未绑定被修改者，由于修改特征中存在对被修改特征的行为进行调整的代码，因此最终的系统存在类型错误。

图 4 描述了特征的可变性对修改依赖的影响。图 4(a)是该示例的模块结构，包含 B、MODIFYEE 和 MODIFIER 3 个特征，其中 B 是必选特征，其余均为可选特征。图 4(b)和图 4(c)分别是 MODIFYEE 的基模块以及 MODIFIER 的派生模块对应的代码，MODIFIER 的派生模块 ∂modifyee/∂MODIFIER 表明 MODIFIER 对 MODIFYEE 存在修改依赖。特征选择时，如果绑定了 MODIFIER 而未绑定 MODIFYEE，最终的系统会存在类型错误。图 4(a)中的虚线表明 MODIFIER 对可选特征 MODIFYEE 存在依赖。

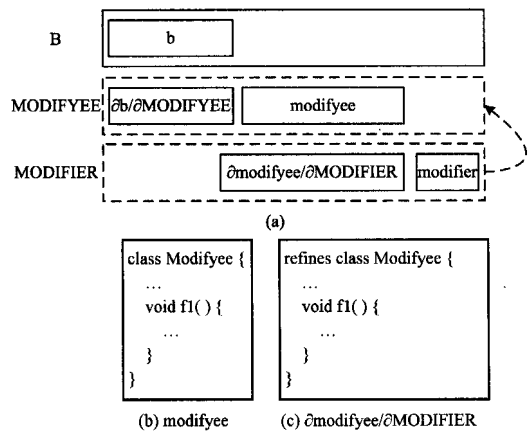


图 4　对可选特征的修改依赖

通过修改依赖的定义可知，所有派生模块均描述了特征之间的修改依赖。对于派生模块 $\partial b/\partial H$，如果特征 B 具有可变性，该修改依赖会受到特征可变性的影响，比如图 4(a)中的∂modifyee/∂MODIFIER。如果特征 B 不具有可变性，即 B 是必选特征，该修改依赖不会受到可变性的影响，比如图 4(a)中的 $\partial b/\partial$MODIFYEE。

3.3　激活依赖

特征只有处于活跃状态时才能运行其所具有的功能。一个特征的激活可能要依赖于其它特征的状态，这种依赖称作激活依赖。激活依赖分为互斥激活依赖、主从激活依赖、并发激活依赖和顺序激活依赖 4 类[3]。如果系统中的两个特征之间存在激活依赖，则需要通过代码来控制和检测这两个特征之间的激活状态。倘若其中一个特征具有可变性，在特征选择时，如果没有绑定该特征，由于另一个特征的模块中包含和未绑定特征存在激活依赖相关的代码，因此最终的系统会存在类型错误。

图 5 描述了特征的可变性对互斥激活依赖的影响，互斥激活依赖是指一些特征不能同时处于活跃状态，即不能同时执行这些特征的功能[3]。图 5(a)是该示例的模块结构，包含 B、F 和 G 3 个特征，其中 B 是必选特征，其余均为可选特征。图 5(b)和图 5(c)分别是特征 F 和 G 的基模块对应的代码，由于 F 和 G 之间存在互斥激活依赖，因此 F 的方法执行之前要判断 G 是否处于活跃状态，只有当 G 处于非活跃状态时，F 的方法才继续执行。同样地，G 所包含的方法亦是如此。如果特征选择时只绑定了 F 和 G 中的一个特征，最终的系统会存在类型错误。图 5(a)中的虚线表明了 F 和 G 之间相互的依赖关系。

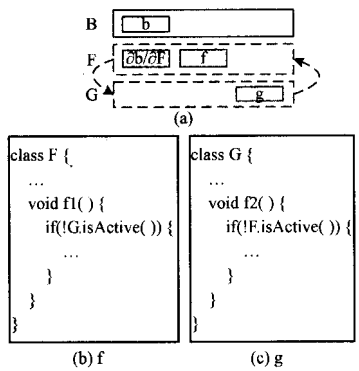


图 5　可选特征之间的互斥激活依赖

主从激活依赖、并发激活依赖以及顺序激活依赖所产生的特征组合失效问题与图 5 描述的场景类似，因此可以参考互斥激活依赖中存在的特征组合失效问题。

4　特征组合失效问题的解决方法

对于上述特征依赖所引起的特征组合失效问题而言，它们具有类似的模式，即一个特征绑定与否，决定了该特征所对应的层是否被集成进最终系统。然而，与特征依赖相关的代码耦合在特征模块内部，被依赖特征的绑定与否无法直接控制这部分代码的存在性，因此导致了在组装过程中存在依赖关系而遗漏依赖对象的问题。本节提出对特征模块进行垂直分解的机制，在一个层中将受到可变性影响的特征依赖所对应的代码从特征的核心功能代码中抽取出来作为独立的层次与模块。这种机制可以支持特征组装的一致性，避免特征组合失效问题的产生。

4.1　特征模块的垂直分解模式

图 2 描述了特征的模块结构，将一个特征分解为基模块和派生模块，文献[6]将这种分解看作特征的水平分解。为了解决特征组合失效问题，需要对特征的模块结构进行重新划分，在特征的水平分解基础上，对特征进行垂直分解。将一个特征对应的层划分为多个子层，第一个子层是核心子层，只包含该特征核心功能对应的代码，即对其它可变性特征不存在依赖的部分。其余的每个子层为非核心子层，用子层 n 表示(n 等于 1,2,3…)，分别对应一种该特征和其它可变性特征存在依赖相关的代码。通过垂直分解，将一个特征的基模块和

派生模块垂直划分为多个子模块，属于核心子层的部分称作核心基模块和核心派生模块。

定义 1 核心基模块是本特征的基模块中不会产生特征组合失效问题的代码对应的模块。

定义 2 核心派生模块是本特征的派生模块中不会产生特征组合失效问题的代码对应的模块。

对于子层 n，将原属于基模块中受依赖特征影响的代码水平分解为非核心基模块和非核心派生模块。由于派生模块只包含修改方法的代码，因此原属于派生模块中受依赖特征影响的代码作为非核心派生模块存在，而不包含非核心基模块。

定义 3 非核心基模块是在非核心子层中存在的、因与某个可变性特征存在依赖而新增的类以及对已有特征进行扩展而新增的属性和方法所对应的代码模块。

定义 4 非核心派生模块是在非核心子层中存在的，原属于基模块的方法或派生模块的代码中，和受可变性影响的特征依赖相关的代码对应的模块。

在子模块的代码实现层次上，可以通过 refines 表示子模块之间的精化关系。比如，非核心派生模块通过 refines 将受可变性影响的依赖相关的代码精化到核心基模块或者核心派生模块中。而非核心基模块通过 class 来创建自身的类，并且通过 refines 在核心基模块中添加新的属性或者方法。

通过对特征模块进行垂直分解，将每个特征对其它可变性特征存在的依赖保存在非核心子层中。非核心子层具有自身的可变性，在其所属特征是必选或被绑定的情况下，它的绑定性与其所依赖的特征的绑定性保持一致。因此，非核心子层对相应特征的依赖关系可以保证代码结构的完整性和类型安全，从而在特征选择时不会出现特征组合失效问题。

图 6 描述了特征模块的垂直分解示例，系统包含 B、F、G、H 和 K 5 个特征，其中 B 和 G 是必选特征，其余均为可选特征，K 对 F 和 H 存在依赖。由于 K 受 F 和 H 可变性的影响，因此 K 垂直分解为 3 个子层，其中子层 1 包含对可选特征 F 的依赖部分，子层 2 包含对可选特征 H 的依赖部分。作为约定，核心基模块用特征对应的小写字母标以下脚标 0 表示，如 k_0。K 的核心子层中的 ∂b 和 ∂g 用于标记核心派生模块。在子层 1 中，$\partial k_1/\partial F$ 表示非核心基模块，$\partial k_0/\partial F$ 表示非核心派生模块。派生模块中的非核心派生模块形式化表示可以通过式(3)进行简化：

$$\partial^2 g/\partial F=\partial(\partial g)/\partial F \tag{3}$$

因此子层 1 中的 $\partial^2 g/\partial F$ 用来标记非核心派生模块。

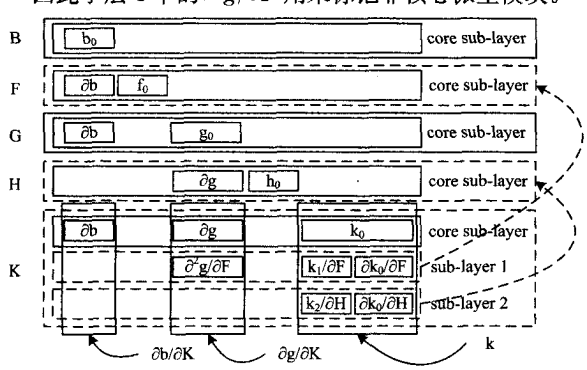


图 6 特征模块的垂直分解示例

4.2 基于可变性绑定的模块组装过程

在应用系统工程阶段，根据用户的特征选择完成模块组装过程，将该过程分为特征的模块组装和系统的模块组装两个阶段。

特征的模块组装过程中要考虑到非核心子层的可变性。在图 6 的示例中，如果最终系统绑定了 H 和 K 而未绑定 F，由于 K 的子层 1 依赖 F，因此在 K 的模块组装过程中不组装子层 1 中的模块。每个特征组装完成后分别得到一个完整的基模块和若干派生模块。特征的模块组装过程可以通过＋和·两种标记进行形式化表示，比如 K 中的基模块和各个派生模块的模块组装表达式分别为 $k=\partial k_0/\partial H \cdot k_0+k_2/\partial H$，$\partial b/\partial K=\partial b$，$\partial g/\partial K=\partial g$。

在特征的模块组装过程基础上，完成系统的模块组装过程。由于特征的模块组装完成后得到了每个特征完整的基模块和派生模块，因此系统的模块组装过程同 FOP 的背景介绍中所描述的模块组装过程是一致的。对于图 6 的示例而言，系统的模块组装表达式为：

$$K(H(G(B)))=\partial b/\partial K \cdot \partial b/\partial G \cdot b+\partial g/\partial K \cdot \partial g/\partial H \cdot g+h+k$$

4.3 3 种特征依赖场景的处理方式

针对特征组合失效问题分析中的各个问题场景，本节说明了如何通过特征模块的垂直分解来解决这些问题。

4.3.1 使用依赖

针对图 3 描述的特征可变性对使用依赖的影响，图 7 描述了该示例垂直分解的模块结构和部分模块对应的代码。将对 B 存在使用依赖的代码独立为非核心派生模块 $\partial c_0/\partial B$，$\partial c_0/\partial B$ 同 B 保持一致的绑定性。特征选择和组装过程中，即使没有绑定 B，也不会导致最终的系统出现类型错误。

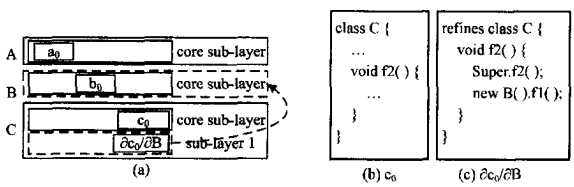


图 7 对可选特征的使用依赖管理

4.3.2 修改依赖

针对图 4 描述的特征可变性对修改依赖的影响，图 8 描述了该示例垂直分解的模块结构和部分模块对应的代码。将 MODIFIER 对 MODIFYEE 的修改表示为非核心派生模块 ∂^2modifyee/∂MODIFYEE，在 MODIFIER 被绑定的情况下，∂^2modifyee/∂MODIFYEE 同 MODIFYEE 保持一致的绑定性。特征选择和组装过程中，即使没有绑定 MODIFYEE，也不会导致最终的系统出现类型错误。

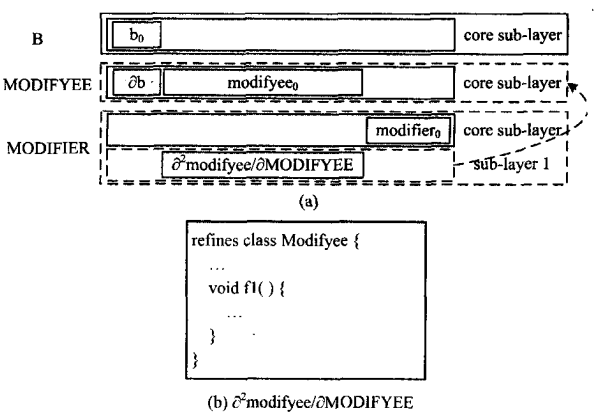
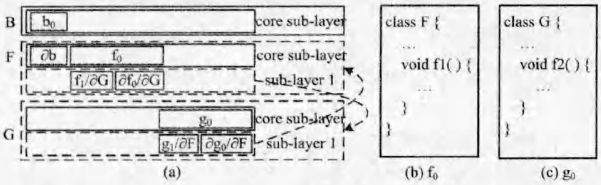


图 8 对可选特征的修改依赖管理

4.3.3 激活依赖

针对图5描述的特征可变性对互斥激活依赖的影响，图9描述了该示例垂直分解的层次结构和部分模块对应的代码。非核心基模块 $f_1/\partial G$ 和 $g_1/\partial F$ 以及非核心派生模块 $\partial f_0/\partial G$ 和 $\partial g_0/\partial F$ 用于控制和检测 F 和 G 之间的互斥状态，只有当 F 和 G 同时绑定时，这些子模块才会存在于最终系统中。




```
refines class F {
  static Boolean FExclusiveWithGState =
false;
  static Boolean getFExclusiveWithGState()
{
    return FExclusiveWithGState;
  }
}
```

(d) $f_1/\partial G$

```
refines class G {
  static Boolean GExclusiveWithFState
= false;
  static Boolean getGExclusiveWithF-
State() {
    return GExclusiveWithFState;
  }
}
```

(e) $g_1/\partial F$

```
refines class F {
  void f1() {
    if(!G.getGExclusiveWithFState( )) {
      Super.f1();
    }
  }
}
```

(f) $\partial f_0/\partial G$

```
refines class G {
  void f2() {
    if(!F.getFExclusiveWithGState( )) {
      Super.f2();
    }
  }
}
```

(g) $\partial g_0/\partial F$

图9 互斥激活依赖的管理

4.4 方法的局限性

在执行特征模块垂直分解时，要按照分解模式将和特征依赖相关的代码与特征核心功能代码进行解耦，将其独立为非核心子层中的非核心基模块和非核心派生模块。非核心派生模块中的代码是原基模块或派生模块的方法中和依赖相关的部分，如果这部分代码位于原方法的首尾两端，可以通过FOP的refines和Super机制对代码进行有效的分解与封装，并保证以后组装的正确性。然而，如果这部分代码位于原方法内部(夹杂在方法体中间)，由于目前的FOP缺少将代码编织到方法内部的有效机制，因此很难将代码进行良好的分离，这也是该方法存在的一个局限。

5 实例研究与评估

本文的实例是一个出版社利润考核系统产品线，用于考核出版社的图书销售利润以及编辑的利润分配，图10是该产品线的特征模型，基程序(BASE)主要包括图书基本信息提取、销售数据提取和成本数据提取等功能。其它功能管理(OTHERMANAGEMENT)用于提取其它收入和费用。利润考核(PROFITEVALUTION)用于结算、分配以及审核利润，主要包括结算(BALANCE)、取消结算(CANCELBALANCE)、利润分配(PROFITDISTRIBUTION)、分配审核(CHECKEDITORPROFIT)和取消审核(CANCELCHECKEDITORPROFIT)。系统设定(SYSTEMSETTING)包括结算设定(BALANCESETTING)、日志记录(LOGGING)、数据获取模式(DATAACQUISITIONMODE)等。

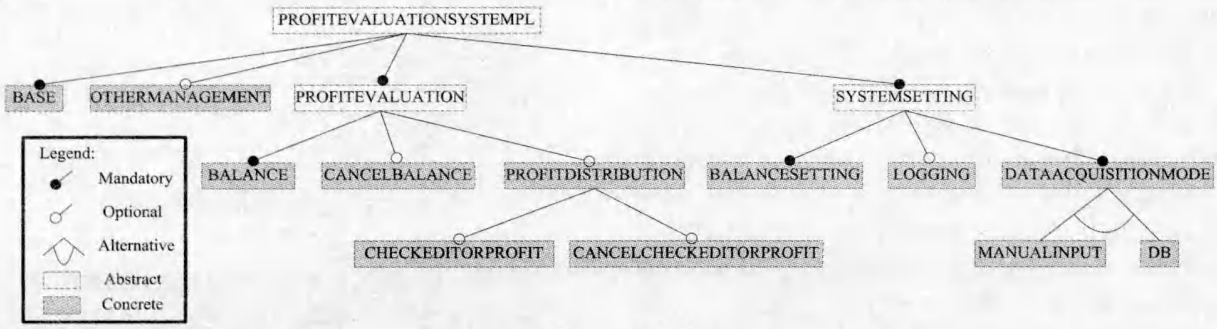


图10 利润考核系统的特征模型

利润考核系统产品线中受可变性影响的部分特征依赖如下：

(1)使用依赖：OTHERMANAGEMENT使用DATAACQUISITIONMODE(多选一)，BALANCE使用OTHERMANAGEMENT(可选)。

(2)修改依赖：PROFITDISTRIBUTION修改CANCELBALANCE(可选)，LOGGING修改CALCELBALANCE(可选)，LOGGING修改CHECKEDITORPROFIT(可选)，LOGGING修改CANCELCHECKEDITORPROFIT(可选)。

(3)互斥激活依赖：BALANCE与CANCELBALANCE(可选)，CHECKEDITORPROFIT(可选)与CANCELCHECKEDITORPROFIT(可选)。

(4)顺序激活依赖(多个特征按顺序逐个被激活)：BALANCE与CHECKEDITORPROFIT(可选)，CANCELCHECKEDITORPROFIT(可选)与CALCELBALANCE(可选)(业务需求规定只有结算后才能审核编辑分配，取消全部编辑审核后才能取消结算)。

图11描述了该利润考核系统产品线部分特征的模块结构，由于篇幅限制，主要以结算特征的模块垂直分解作为实例。通过特征的垂直分解，受可变性影响的特征依赖相关的代码独立为单独的子模块，可以避免特征组合失效问题。比如结算(BALANCE)对可选的其它功能管理(OTHERMANAGEMENT)存在使用依赖，如果没有对结算特征模块进行垂直分解，那么特征选择时如果没有绑定其它功能管理特征，则必须选择结算特征，这将导致系统在编译时存在类型错误。通过将结算特征中和其它功能管理特征存在的依赖相关的代码独立为非核心派生模块 $\partial bm_0/\partial OM$，当其它功能管理特征没有绑定时，只要不绑定非核心派生模块 $\partial bm_0/\partial OM$，就不会出现上述问题。

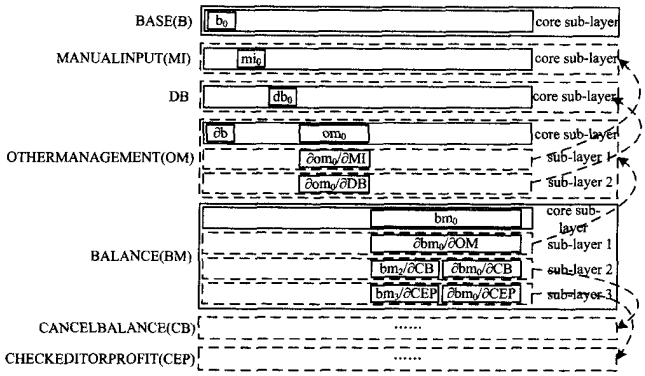


图 11 利润考核系统的模块结构

6 相关工作

Lee 和 Kang[3] 阐述了在基于特征的共性和可变性分析基础上增加特征依赖分析的重要意义，并提出了一种面向对象的方法管理特征依赖。Cho 和 Lee 等人[11] 在文献[3]的基础上，提出了一种通过面向方面编程(Aspect-oriented Programming，AOP)的机制管理软件产品线中的特征依赖的方法。Zhang 和 Mei 等人[10] 通过特征建模需求之间的依赖，提出了一种模块化的方式来分析需求，以合适的粒度分析需求依赖，并研究了需求依赖之间潜在的联系。以上对特征依赖的研究针对基于构件的软件产品线开发过程(CBSE)，受这些研究的启发，本文对 FOP 的软件产品线开发过程中和特征依赖相关的问题进行了研究。

Zhang、Godil 和 Jacobsen[13,15] 在 AOP 的软件产品线开发过程中首次提出了水平分解(Horizontal Decomposition，HD)的概念。HD 将特征水平分解为包含新增加类的模块和包含方面(Aspect)的模块，方面包含引入的新属性和新方法以及对已有方法的修改。Liu 和 Batory 等[6] 借鉴文献[13，15]，将 FOP 中的层水平分解为基模块和派生模块，并增加了模块分解的形式化表示，基模块不仅包含新增加的类，还包含对已有类引入的新的属性和方法，派生模块只包含对已有方法的修改。本文在文献[6]的基础上，增加了特征模块的垂直分解。

Apel 等[9] 提出了面向特征的设计，作为领域分析和领域实现之间的一个桥梁，该范例将一个产品线的设计按照特征进行分解，在设计阶段可以检测特征之间的依赖和交互。Liu 和 Batory 等在文献[6]中提到了可选特征问题，问题描述类似本文提到的可选特征对修改依赖的影响。Apel 和 Hutchins[12] 提出了一个用于源代码和其它非代码制品的统一的 FOP 特征组装模型，该模型包含一个类型系统，用于检测特征组装过程中存在的类型错误。相比以上这些研究，本文对 FOP 实现阶段可变性对特征依赖的影响所导致的代码结构问题进行了系统、全面的分析，并提出了解决方法。

结束语 特征模型中可变性特征之间的依赖会引起 FOP 中的特征组合失效问题，这类问题在 FOP 设计与编码时往往被忽略，从而影响软件产品线应用产品定制时的正确性。因此，本文对这个关键问题进行了系统、全面的分析，归纳了 3 种引发组合失效问题的特征依赖场景，并针对这些场景提出了一种特征模块垂直分解的方法以解决该问题。该垂直分解方法避免了特征选择对系统代码结构的影响，增强了 FOP 的可适应性、可伸缩性和可配置性。

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，2001

[2] Pohl K, Böckle G, van der Linden F. Software Product line Engineering: Foundations, Principles, and Techniques[M]. Heidelberg New York: Springer Berlin, 2005

[3] Lee K, Kang K C. Feature Dependency Analysis for Product Line Component Design[C]// Proceedings of 8th the International Conference on Software Reuse, ICSR. 2004

[4] Peng Xin, Zhao Wen-yun, Xue Yun-jiao, et al. Ontology-Based Feature Modeling and Application-Oriented Tailoring[C]// Proceedings of 9th the International Conference on Software Reuse, ICSR. 2006

[5] Batory D, Sarvela J N, Rauschmayer A. Scaling Step-Wise Refinement [J]. IEEE Transactions on Software Engineering, 2004, 30(6): 355-371

[6] Liu Jia, Batory D, Lengauer C. Feature Oriented Refactoring of Legacy Applications[C]// Proceedings of 28th the International Conference on Software Engineering, ICSE. 2006

[7] Kang K C, Cohen S G, Hess J A, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study[R]. CMU/SEI-90-TR-21. Pittsburgh: Software Engineering Institute, Carnegie Mellon University, 1990

[8] Kang K C, Kim S, Lee J, et al. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures[J]. Annals of Software Engineering, 1998, 5(1): 143-168

[9] Apel S, Scholz W, Lengauer C. Detecting Dependences and Interactions in Feature-Oriented Design[C]// Proceedings of 21st International Symposium on Software Reliability Engineering, ISSRE. 2010

[10] Zhang Wei, Mei Hong, Zhao Hai-yan. A Feature-Oriented Approach to Modeling Requirements Dependencies[C]// Proceedings of 13th International Conference on Requirements Engineering, RE. 2005

[11] Cho H, Lee K, Kang K C. Feature Relation and Dependency Management: An Aspect-Oriented Approach[C]// Proceedings of 13th International Software Product Line Conference, SPLC. 2008

[12] Apel S. A Calculus for Uniform Feature Composition[J]. ACM Transactions on Programming Languages and Systems, 2010, 32(5): 19

[13] Zhang C, Jacobsen H-A. Resolving Feature Convolution in Middleware Systems[C]// Proceedings of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA. 2004

[14] Apel S, Kästner C. An Overview of Feature-Oriented Software Development[J]. Journal of Object Technology, 2009, 8(5): 49-84

[15] Godil I, Jacobsen H-A. Horizontal Decomposition of Prevayler [C]// Proceedings of 2005 Conference of the Centre for Advanced Studies on Collaborative Research, CASCON. 2005

[16] Czarnecki K, Eisenecker U. Generative Programming-Methods, Tools, and Applications[M]. Addison-Wesley, 2000

[17] 谢仲文，李彤，代飞，等. 基于特征组合的软件需求建模[J]. 计算机科学，2012，39(1)：130-133，141

[18] 吴元凯，彭鑫，赵文耘. 应用面向特征编程方法 FOP 实现软件产品线增量开发[J]. 小型微型计算机系统，2010，31(8)：1613-1618