应用面向特征编程方法 FOP 实现软件产品线增量开发

吴元凯，彭鑫+，赵文耘
（复旦大学计算机科学技术学院， 上海 200433）

+通讯作者：pengxin@fudan.edu.cn

摘 要:面向对象编程 OOP（Object Oriented Programming）技术在实现软件产品线时存在不足，一个重要原因是 OOP 对产品线可变性的支持有限。在 OOP 的基础上，相关研究者提出面向特征编程 FOP（Feature Oriented Programming），研究特征的模块性，以及支持特征模块性的编程模型。本文对 FOP 的相关技术和模型进行了研究，提出基于 FOP 进行软件产品线的增量开发，可以实现产品线更高层次的模块化和特征的灵活配置，同时避免了维护困难。在此基础上对一个网上缴费产品线实例进行研究，实现和验证了相关方法和结论。最后基于实验对 FOP 的优缺点进行了讨论。

关键词: 面向对象编程，面向特征编程，软件产品线，模块性，可变性，增量

Using Feature-Oriented Programming to Implement Incremental Software Product Line Development

WU Yuan-kai[1], PENG Xin[1], ZHAO Wen-yun[1]

[1]（*School of Computer Science and Technology, Fudan University, Shanghai 200433,China*）

Abstract: OOP(Object Oriented Programming) has some weaknesses in software product line implementation, an important reason is that OOP provides only limited supports for variability in software product line. Based on OOP, some researchers introduce FOP which studies the feature modularity and programming models that support feature modularity to make up the weaknesses of OOP. This paper explores FOP and its models, and suggests incremental software product line development based on FOP to achieve a higher level modularity, flexibility and make maintenance easy. Then we present a case study on online fee payment software product line for validation with related methods and discussion. At last, we discuss the advantages and disadvantages of FOP based on the experiment.

Keywords: OOP, FOP, software product line, modularity, variability, incremental

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分为开发核心资产的领域工程阶段以及和基于核心资产实现应用产品的应用工程阶段。传统的面向对象 OO（Object-Oriented）技术对于实现产品线存在一些缺陷，难以对产品线可变性提供灵活有效的支持。不论是核心资产还是应用资产开发，都在源代码级别进行了过多的操作，而且 OO 技术复用的粒度细，使产品线的扩展和维护比较困难，同时程序的扩大使得易读易理解性都在下降。

特征（feature）是设计中的一阶实体，是面向用户的功能，用户根据特征区别相似问题。特征精化是实现特征模块化的方法，特征精化封装的并不是传统的方法或者类，而是类或者方法的切片，这些切片构成了整个特征[3]。面向特征编程 FOP（Feature Oriented Programming）是研究特征的模块性和支持特征模块化的编程技术，是构建产品线一种很好的辅助方法[4][13][15]。在 OOP 技术基础上结合 FOP 方法，可以在一定程度上弥补传统的面向对象开发技术在产品线可变性实现方面的不足。

产品线的开发方式是增量式（incremental）的[1]。增量式的开发是产品线技术的关键特征，它要求首先进行最基本核心资产的开发，在此基础上开发部分应用产品，随着需求的增加不断增加核心资产，并且每一阶段都在核心资产的基础上开发出需要的应用产品[1]。增量式开发对于实现产品线的复用性，灵活性和可配置性都是非常重要的。FOP 技术本身就是在基程序的基础上进行特征精化，完全可以作为产品线增量开发的实现方法，同时 FOP 对基程序进行增量开

收稿日期：2009-- 基金项目：国家自然科学基金（60703092）、国家863 计划（2007AA01Z125）、上海市重点学科建设项目资助（B114）资助.
作者简介：吴元凯，男，1984 年生，硕士生，研究方向为软件产品线；彭鑫（通讯作者），男，1979 年生，博士，讲师，CCF 会员，研究方向为软件产品线、软件体系结构、软件维护与再工程；赵文耘，男，1964 年生，教授、博导，CCF 高级会员，研究方向为软件工程、电子商务

发时，绑定特征具有灵活性，特征增量作为产品线资产，可以与基程序选择性绑定，对于产品线可变性的实现提供了极大支持。

本文针对 FOP 技术和软件产品线的可变性进行了研究。OOP 技术实现的产品线系统对于可变性的实现主要依赖继承，重载等技术，对于可变性的支持存在不同程度的问题，比如继承方式必须修改类名，重载技术过分依赖于参数配置，使得特征交互变得复杂。基于以上原因，我们尝试把 FOP 方法应用于一个已有的网上缴费产品线实例，利用 FOP 的方法和工具来辅助产品线的增量开发。文章第二部分介绍了面向特征编程的思想和当前主流技术，及其增量开发形式。第三部分提出FOP对可变性增量的支持技术。第四部分应用FOP方法和工具，对一个产品线实例进行增量开发，其优势和一些不足之处将在第五部分进行讨论。最后，对全文总结并且对下一步工作进行展望。

2 背景介绍

FOP 是通过特征组合来进行编程的一种方法，其主要目标在于将问题分解为多个特征，每个特征成为在功能上的一个增量[2]。初始的基程序仅包含最基本的用户需求和功能，特征作为选择性的功能精化基程序。这种增量开发、逐步精化的开发方法对于软件产品线可变性实现及演化都提供了较好的支持。

2.1 FOP 模型、语言以及工具支持

GenVoca 模型是为了实现 FOP，早期提出的一种方法，通过把用户要加入和删除的特征模块化来实现用户定制[2]。它提出层（layer）的概念，层是程序精化的部分（refinement），也就是某个特征。这个特征可以是只影响了单个类，也很可能横切了多个类。GenVoca 通过层的逐步增加，达到逐步精化。这个模型实现了最基本的 FOP 思想。图 1 描述了一个包中含有的 3 个类 c1-c3，层（layer）r1 横切了这些类，封装了 3 个类的切片，r2 和 r3 也是一样。当进行特征精化组合层 r1-r3 以后，类 c1-c3 达到了精化。

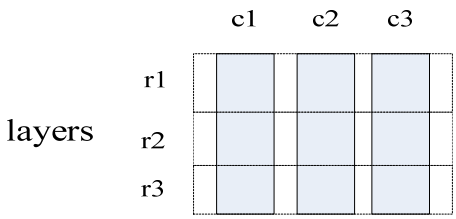

图 1：类（classes）和层（layers）

Fig1: classes and layers

AHEAD（*Algebraic Hierarchical Equations for Application Design*）模型是在早期的GenVoca模型基础上扩展而来[5]，同样也是基于逐步精化实现FOP的一种方法[2]。AHEAD模型的基本思想是：基程序被定义为常量（constant），精化部分被定义为函数（functions），函数的功能是把特征加入到基程序中。如下所示：

App = i(j(f)) //App中包含基程序f，特征i，j。

AHEAD模型可以解释以上代数式，完成把特征i，j加入到基程序中的工作。同时AHEAD中允许含有多个特征的单元（unit）出现，并且支持单元之间的精化，也就是实现多个特征集合的精化。

ATS（*AHEAD Tool Suite*）[6]是当前支持AHEAD模型的主要工具，它提供了一系列的小工具来支持AHEAD方法开发系统。目前大部分工具仅提供命令行操作方式，还没有集成开发环境支持。ATS用Jakarta语言编写，所以ATS也仅支持Jakarta语言格式的代码，Jakarta是java语言的扩展。ATS最主要的工具是组合器（composer），使用它来组合基程序和特征，生成目标代码。如图2所示，我们把特征扩展表示成Jakarta语言的语法，通过composer调用jampack或者mixin工具组合特征，其中jampack把特征插入到源代码中，并保持原来类名不变，而mixin采用OO的继承方式添加特征，修改被继承类的类名，把原来的类名用作最终类名。然后通过转换工具jak2java把jak文件转换为纯java文件，完成特征精化的整个过程。[6]中提供了更多的信息和示例。

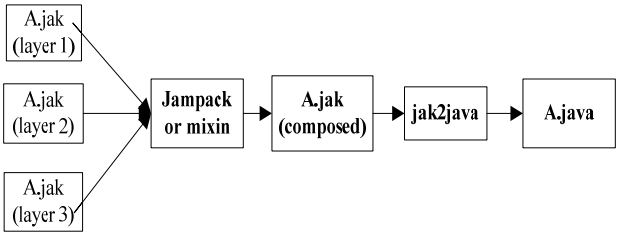

图2：AHEAD工具及使用

Fig2: AHEAD tools and usage

2.2 基本的FOP增量开发形式

FOP技术是通过对基程序或者基类增量的方式进行开发，采用FOP增量开发产品线是建立在AHEAD工具基础上的，用到Jakarta语言（Java的扩展）[7]。典型的FOP增量形式包括：新变量新方法引入，原方法精化，类或者包的添加。

1） 引入新变量和方法。特征的增加，往往需要在类中插入新变量和方法，是对类内容的增量。ATS使用refines关键字，把变量和函数直接插入到类中。根据AHEAD规则，如果引入的变量或方法在基程序同一个类中不存在，则把变量和方法，直接插入到该类中，否则覆盖原变量或方法[6]。

2） 精化原方法。新功能的增加可能需要在原方法中加入更多功能性语句，是对方法的增量。ATS使用super()函数替代原方法的内容，在其之前或者之后添加新代码。如果要

精化内容在原方法的中间位置，由于ATS缺乏对精化部分代码的定位功能，在精化原方法时需覆写已有代码，在需要的位置插入新的内容（类似于OOP中的重写override）。

3） 添加新类或程序包。新特征模块可能以整个类或者数个类构成的包的形式出现，是对程序的增量。可以直接拷贝新类或者包到基程序中，或者应用ATS中的composer工具把新类或者整个包添加到基程序上[6]。

3 FOP对产品线可变性增量的支持

对产品线的增量开发，主要工作是可变性增量。产品线可变性增量就是可变点的增加，同时增加新的变体。典型的产品线可变性表示为可选（optional）、多选一（alternative）和“或”关系（or）三种。可选指软件制品可以选择绑定或者不绑定这个变体。多选一表示软件制品必须要在一组变体中选择一个进行绑定。“或”关系表示软件制品可以在一组变体中选择0个或者多个进行绑定[8][9]。

在产品线开发过程中，对于各种可变性增量的演化，FOP方法都提供了不同程度的支持。下文列举了常见的可变性增量在FOP中的各种实现方法。

1）不变点（即必选点Mandatory）演化为可选可变点（**M2Op**）。

即在不变点增加可选新变体。对于可选择绑定的特征，使用FOP方法和规则，将需要精化的特征代码集中在一个单元中（文件或者文件夹），作为产品线的资产。 产品发布时，根据用户是否需求该功能，选择在基程序上是否绑定这个特征。

2）不变点演化为多选一可变点（**M2A**）。

即在不变点增加多个新变体，必须在原变体与新变体中选择一个绑定到基程序。对于增加的新变体，同样将精化特征代码集中到一个独立的单元中。由于产品发布时，原变体可能不被绑定，所以需要在基程序中提取出原变体，并且对基程序相关部分重构，使之适合FOP工具自动化绑定。FOP方法要求把多个变体都表示单元形式，在生成制品时，选择一个特征单元与基程序进行组合，即完成了这种情况下的FOP实现。

3）不变点演化为“或”类型可变点（**M2Or**）。

即在不变点增加多个新变体，在原变体与新变体中选择0个或者多个绑定到基程序。利用FOP，将精化特征代码集中到一个独立的单元中。同时在基程序中提取原变体，同样表示到独立单元中，并且对基程序相关部分重构。FOP方法选择需要的几个特征模块进行绑定，完成用户定制。

4）多选一可变点变体增加（**A+**）。

即在多选一可变点下增加更多的变体提供绑定。因为是多选一可变点，所以新增变体仅是产品发布之前可选绑定对象之一，FOP只需把此特征按上文同样的方法表示成模块形式，作为产品线资产，与此可变点下其他特征模块一样待绑定。

5）“或”类型可变点变体增加（**Or+**）。

即在“或”类型可变点增加更多的变体提供绑定。FOP处理方法与4）中情况类似，把特征表示成模块，与此可变点下其他特征一起待选。应用制品发布时，选择0个或者多个特征绑定到基程序。

由分析可见，FOP方法对于常见可变性增量形式支持是非常好的，充分实现了特征模块化，根据用户需求绑定特征单元形成完整的可发布应用，系统提供的功能非常清晰。在进行FOP增量开发产品线的同时，需要对基程序进行少量重构。比如在2）3）情况下，须在基程序中提取出原必选特征，作为一个可选变体，同时对基础程序进行必要的重构，以适应FOP增量开发。

4 实例研究

4.1 实例介绍

本文用到的实例是一个网上缴费系统，用于高校学生网上缴纳学费等各项费用。图 3 的特征模型是对此系统共性和可变性的模型化描述。基本功能包括，学生登录，获取费用信息，选择付费方式，系统生成付费清单，银行网上平台缴费，银行返回结果（成功或者失败），最后系统与银行对账，将缴费记录录入数据库。这是个原始的网上缴费产品线系统，包含了最基本的用户需求。下文是对于阶段性用户需求，应用 FOP 方法辅助产品线增量开发的过程，图 4 是最终演化后该系统的特征模型。

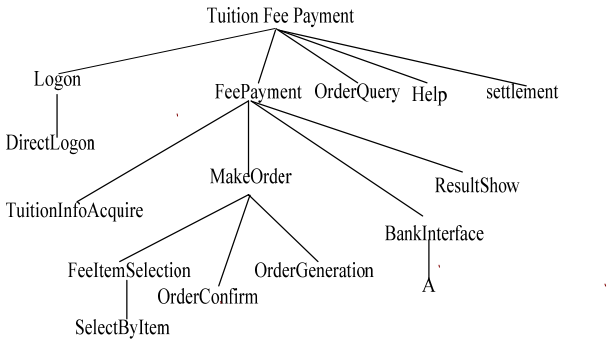

图 3 初始的网上缴费系统特征模型

Fig3: the initial feature model of online fee system

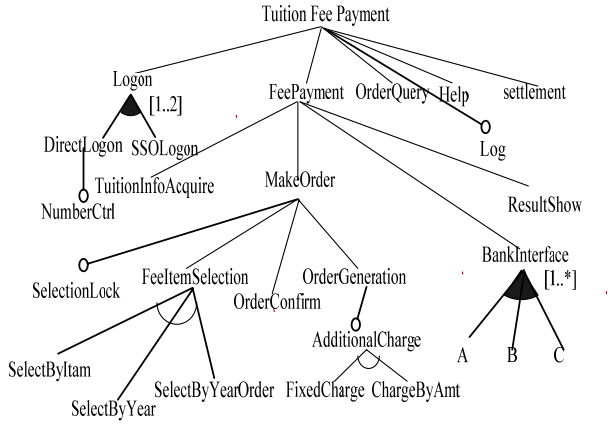

图 4 最终演化后的特征模型

Fig4: the final feature model of online fee system

4.2 产品线增量开发过程

此产品线的增量开发过程分为四个阶段，分别根据用户不同时期的需求变化演化产品线，表 1 给出了四阶段概览。

	阶段一	阶段二	阶段三	阶段四
特征增量	“单点登录”	“选项锁”,“按年缴费”,“按年顺序缴费”	“手续费”,“银行 B”,“银行 C”	“日志”,“登录人数控制”

表 1 增量开发阶段概览

Tab1: Overview of stages of incremental development

●阶段一。在登录验证中，用户需要在“直接登录”的基础上增加“单点登录”（Single Sign On）方式。

“直接登录”就是用户直接使用用户名密码登录。“单点登录”指的是用户从校园网其他地方登录以后跳转到缴费系统进行缴费操作。于是“登录”（Logon）从不变点变为“或”类型的可变点，它包含“直接登录”和“单点登录”2 个变体，可以选择绑定 1-2 个，属于情况 3）。

对于要增加的特征， FOP 把“单点登录”相关的代码组织在单独的 jak 文件中。把这个特征用 Jakarta 语言编码如图 5。第 1 行使用 refines 关键字，表示把该特征添加到此类，引入变量和函数。第 2 行定义变量 fromSSO，代表是否从网上查询登录，此变体还包含 2 个与之相关的函数，分别在第 3 行和第 4 行。

```
1.  refines class FeeUser{
2.      private boolean fromSSO;//是否来自网上查询
3.      public boolean checkAddCode(…) {… }
4.      public boolean ifFromSSO() {…}
5.  }
```

图 5： 单点登录特征

Fig5: feature of SSO

基程序和特征表示成目录的形式，目录 base 包含基程序。以“单点登录”为例，目录 SSOrefine 包含特征“单点登录”，base 目录结构是：base\com\wingsoft\webfee\common\FeeUser.java ，SSOrefine 的目录结构：SSOrefine\com\wingsoft\webfee\common\FeeUser.jak。使用 AHEAD 工具中的 composer 进行增量开发：

composer --target=project base SSOrefine （1）

Composer 遍历目录中所有文件夹，遇到名字相同的文件就调用 jampack 组装工具对基程序精化，精化的结果保存在 project 目录中。再把其中的 jak 文件转成 java 文件即完成“单点登录”功能的添加。同时 SSOrefine 文件夹作为核心资产保存下来，对未来应用产品的开发同样适用。

●阶段二。用户提出系统存在重复缴费的极少数情况，并且用户希望缴费方式在原有“按项目缴费”的基础上增加“按年缴费”和“按年顺序缴费”。“按项目缴费”即根据费用项目缴纳，“按年缴费”是根据每年产生的费用缴费，“按年顺序缴费”指必须先缴清之前每年的费用才能缴纳今年的费用。

由于我们的系统跟银行系统通信会有延迟，用户可能已经缴费但系统却还没收到银行的反馈信息，缴费结果将暂时反映不到系统中，为了避免重复缴费的发生，我们在“制作订单”（makeOrder）特征下加入“选项锁”这个可选特征，在用户缴费之后的一段时间内锁定该缴费项目。“制作订单”从不变点演化为可选可变点，属于情况 1）。

“缴费方式选择”这个不变点演化为多选一可变点，它包含 2 个变体，“按项目缴费”和“按年缴费”属于情况 2）。

此后，多选一可变点继续增加变体“按年顺序缴费”，属于 4）中情况。

通过 FOP 方法，分别完成上面的可变性增量，被精化的类和方法已经在表 2 中表示出来。“按年缴费”和“按年顺序缴费”是在同一个不变点增加的 2 个不同变体，我们还在基程序中提取出变体“按项目缴费”。

●阶段三。银行 A 提出对于有的用户网上缴费要收取手续费，并且随着与学校关联的银行的增多，必须让每个用户可以选择自己需要的银行进行缴费。

在特征“订单生成”（OrderGeneration）之下增加可选特征“手续费”，不变点演化为可选可变点，属于情况 1）。“手续费”指银行为每笔网上缴纳收取的费用。如果绑定了这个变体，那么必须在“固定手续费”和“按金额收费”两个变体中选择一个。

银行增多，必须把特征“银行接口”这个不变点演化为

“或”类型可变点，属于情况 3)，其下包含的变体是每个银行的相关信息。再次增加关联的银行则属于情况 5)，在“或”类型可变点增加变体，用户可以绑定 1 个或多个银行。这里依次添加银行 B 和 C，同时提取出银行 A 的信息为原变体。具体修改参数已经表示在表 3 中。由于“或”类型可变点可能绑定多个变体，而变体间的交互使得绑定顺序是敏感的，本例中，银行A,B,C的特征均对 initBankInfo(…)方法精化，存在特征交互，这里的交互比较简单，我们采用重构 initBankInfo(…)的方法，就可以使得每次精化相对独立，并且特征的添加顺序是任意的。特征交互的复杂情况，在本文第五部分进行了更深入的讨论。

●阶段四。用户对系统安全性和运行性能提出了较高要求，相比前三阶段功能性需求的 FOP 实现，此阶段为非功能性需求的增加。系统增加可选特征“日志”,“登录人数控制”，属于情况 1)。“日志”特征对浏览用户和登录用户都进行记录，“登录人数控制”使系统同时在线人数保持在一个阈值之下，从而确保系统性能和响应速度。精化的类和方法在表 4 中列出。可以看出，FOP 对于用户非功能性可变性增量支持仍然是非常好的。

	可变性增量类型	影响的类	新变量	新方法	方法精化	增加类
“选项锁”	1）	FeeInfo,FeeOrder	lockTags	ifLocked(…)	addFeeInfoAt(…), initByItem(...)	N
“按年缴费”	2）	FeeOrder, FeeUser	payMode	initByYear(…), getPayMode()	init(…)	N
“按年顺序缴费”	4）	FeeOrder, FeeUser	payMode	initByYearS(…), getPayMode()	init(…)	N
“按项目缴费”	基程序中提取	FeeOrder, FeeUser	payMode	initByItem(…), getPayMode()	init(…)	N

表 2 第二阶段增量开发

Tab2: The second phase of the incremental development

	可变性增量类型	影响的类	新变量	新方法	方法精化	增加类
“手续费”	1）	AAgent, FeeOrder	rate	setRate(..),getRate() calculateRate(…)	checkMaxAmt(..) init(…)	N
“银行 B”	3）	Configuration	N	readBConfig(…), updateBInfo(…)	initBankInfo(…)	BAgent， BConfig，BReturn
“银行 C”	5）	Configuration	N	readCConfig(…), updateCInfo(…)	initBankInfo(…)	CAgent， CConfig，CReturn
“银行 A”	基程序中提取	Configuration	N	readAConfig(…), updateAInfo(…)	initBankInfo(…)	AAgent， AConfig，AReturn

表 3 第三阶段增量开发

Tab3: The third phase of the incremental development

	可变性增量类型	影响的类	新变量	新方法	方法精化	增加类
“日志”	1)	FeeUser	N	doLog()	Login()	SessionCreateLog
“登录人数控制”	1)	FeeUser	loginNum	isOverFlow()	Login()	N

表 4 第四阶段增量开发

Tab4: The fourth phase of the incremental development

4.3 产品线演化

随着用户需求的增加，产品线和应用产品都必须不断演化。用户要增加更多的特征或者想在现有的特征上使用最新的技术是导致产品线演化的动力之一[11]。核心资产和应用资产的演化构成了产品线的演化。对于网上缴费产品线的开发，我们通过 4 个阶段的用户需求，实现了使用 FOP 辅助产品线增量开发的目标。通过用户需求，逐渐增加新的功能，也就是增加新特征，这是产品线演化的一个方面。通过产品线各个应用资产各自版本的对比分析，更新可变点或者可变点的增加或者减少，也是演化的一个方面。在实例中，“付

费方式选择”演化为多选一可变点，而“登录”和“银行接口”都已经演化为“或”类型可变点。“选项锁”特征是可选择绑定的，但是几乎所有的用户都选择，会逐渐演化为必选特征。

5 讨论与分析

在这部分，结合产品线增量开发实验的经验，通过将 FOP 与主流 OOP 技术比较，对构建产品线的作用的分析，以及 FOP 对系统开发维护的益处的讨论，体现了 FOP 独特的优势。同时讨论了 FOP 对交互特征模块的处理，也指出了 FOP 在横切特性上面的一些不足之处。

5.1 FOP 与 OOP 的比较

FOP实际上是对OOP方法的一种扩展，OOP方法通过子类继承来增量开发，FOP通过特征组合来开发程序，粒度更大，层次更高。FOP提出了层（layer）的概念，一个特征就是一层，可能横切了多个类，比如“选项锁”这个特征，横切了2个类中的多种方法。如果采用OOP方法，我们必须创建子类继承FeeInfo和FeeOrder 2个类，并且把涉及到的方法无一例外的覆写（Override），工作量大而且操作多个类多个方法，添加的代码比较散乱，容易出错，不利于系统的维护。使用FOP方法把横切的几个类集中到一个特征模块中，修改代码集中，每次精化基程序都是把整个特征用一个命令添加上去，自动修改多个类的代码，比如命令（1），把SSOrefine这个目录下的精化文件全部添加到基程序上。若特征精化的模块已经存在，开发者甚至并不需要知道具体修改的类的细节。FOP通过refine关键字把要修改的内容直接添加到原来类中，并不改变原来的类名，这对于系统的增量开发非常有利，因为被修改的类可能横切了许多其他类，类名若改变会导致很多对基程序的修改，不利于软件制品的维护。相比于传统的OOP技术，FOP提供了更高层次的模块化和灵活性，代码复用变的更加容易实现。

5.2 FOP 对构建产品线的作用

FOP 增加了产品线资产的可复用性。复用性是构建产品线的非常重要的目标，要求产品线大部分的核心资产能在应用产品中得到复用。FOP 把特征模块化，减少了复用时代码级别的操作，在把相应的特征文件组织到一定的目录以后，复用该特征只需执行一行命令。并且 FOP 采用的是增量式开发，逐步精化基程序，是一种在基程序上线性增加功能的开发模式，遵循了产品线开发的原理。

FOP 支持产品线的可变性。特征作为精化基程序的层，可以灵活的组合。基程序可以和变体任意的绑定，这种机制能很好的实现产品线的可变性。把 FOP 用作增量开发，同样能很好的支持产品线可变性增量，这点在实验中已经验证，对于列出的典型的产品线可变性增量，FOP 都提供了不错的实现，同时作为 FOP 的特长，实现了变体特征的模块化。值得一提的是，AHEAD 工具支持特征绑定的配置，把相关基程序和需要绑定的特征写在配置文件里，调用 composer 的时候就能根据配置文件的绑定顺序进行程序精化[6]。

FOP 设计方法有助于产品线的演化。产品线各个应用资产的演化往往差距很大，客户需求的多样性导致了这个结果。例如在已有网上缴费产品线上，有的客户需要添加“邮件通知”的特征，有的客户并不需要。到一定的阶段，各个应用资产与核心资产需要一定程度上的同步，需要在产品线可控范围内继续演化。FOP 方法中特征模块化，不但提高了系统的易理解性，而且对于提取各个应用资产的相同模块，更新核心资产很有帮助，甚至可以实现这个过程的自动化。该网上缴费系统的初始基程序是最原始的产品线核心资产，经过不断演化，像“选项锁”等特征都成为了核心资产的一部分。

5.3 FOP 的应用对系统开发维护的影响

目前的系统开发主要还是采用的面向对象 OOP 方法，如果在网上缴费产品线的增量开发中不应用 FOP 方法，系统的开发依赖于为每个客户保存的一个独立的版本，根据各个客户的需求修改其程序，特征分散于各个版本的代码中，而且对于相同的功能性需求，必须在不同的版本中重复相同的代码。例如三个客户需要“单点登录”功能，那么就需要在这 3 个客户独立的版本中拷贝相同的类似于图 5 的代码。当有客户需要删除一项已绑定的功能，如客户与某银行的协议终止，在“银行接口”上要删除一个已经绑定的银行，那么开发者必须定位特征，找到所有相关类和方法甚至变量，删除有关代码，这本身是一个繁琐的过程。随着客户的增加和产品演化的复杂性增加，软件的维护和扩展变得非常困难。

采用 FOP 辅助开发产品线的整个开发过程中，从基程序开始对于用户的每个功能性需求都采用特征模型描述，各个特征的相关代码表示在各自模块中，然后使用 AHEAD 工具添加特征代码到基程序上。未来有新功能的增加，采用同样的处理方式。对于不同应用资产的各种功能的增加，只要把特征模块添加到该应用的基程序上。对于客户需要删除已绑定特征，可以根据该特征模块很快的定位特征代码在基程序中的位置，手动删除。或者可以从基程序重新组合各个特征（除去要删除的），在这个意义上，程序可以恢复到任意一个早期版本。同时，特征模块化使得代码的易读性加强，系统维护者可以迅速理解系统，系统的可维护性也增强了。

5.4 有交互的特征间的绑定分析

特征之间的交互使得特征组合的灵活性受到一定的限制，在本文的例子中，“银行接口”可变点下的变体“A”“B”和“C”实际是存在一定的交互的，在精化的过程中都修改 initBankInfo(…)函数，他们的精化顺序直接影响到精化代码的修改，在这个例子中，我们通过重构这个函数，实现了特征交互下的 FOP 方法的应用。但是并不是所有的特征交互都可以通过简单重构相关代码来支持。

对于特征交互粒度比较大的情况，比如类，方法间的交互，相关研究者已经提出了较好的解决方法：把特征之间的交互代码单独提取出来作为一个依赖于两者的特征，把原来的两个交互的特征分解为 3 个特征模块（2 个各自独立的模块和一个交互模块），当单个特征被用来精化基程序时，仅使用其独立模块，另一个特征再次精化时，则添加交互模块和另一个独立模块[10][14]。

特征交互粒度小的情况，比如语句级的交互，很难将交互代码模块从特征中分离出来。例如有“选项锁”特征和没有“选项锁”的“按年缴费”功能代码是有区别的，有“选项锁”的“按年缴费”功能代码包含语句判断该年的费用项是否被锁定，而这样的语句使用 AHEAD 工具是不能提取出来单独作为一个模块的。对于这种情况，有两种处理方法，一是通过观察分析，看是否能重构部分函数来适应 AHEAD 工具操作，这个方法对代码依赖性高，局限性比较大；二是将“按年缴费”对应于是否有“选项锁”功能构建 2 个不同的特征模块，这种方法理论上能处理所有的情况，但是对于每对交互特征都必须至少构建 4 个特征模块，而如果有 N 个特征具有交互关系，那么此方法需要构建的特征数将达到 2^N，随着 N 的增大，处理这些特征变的非常复杂。对于这种小粒度的特征交互情况，FOP 方法的支持还很有限。

5.5 缺陷和弥补

FOP 方法的特征虽然可能横切多个类，但是对于类中方法的修改却很局限。除非要添加部分是方法头或者方法尾，可以使用 super()函数代替原方法的相同代码，否则必须覆写整个方法。为了添加“选项锁”特征，需要对 FeeOrder 类中方法修改，实际上我们已经覆写了 initByItem(...)整个方法。AHEAD 工具对于插入代码的位置难以准确定位，所以无法直接在方法中间插入要添加的代码。

XVCL 是一种基于 XML 的可变性配置语言[12]，它提出对于一些难以分离的特征或者是分离后程序维护反而更加复杂的特征，可以把这种特征保留在基程序中。XVCL 中使用<select>关键字在基程序中表示可变点，<break>关键字可以准确定位要添加代码的位置。这种机制可以很大程度上弥补 FOP 横切特性的不足。

6 总结与展望

特征在软件制品中占有重要地位，客户的需求变更直接导致特征的增加或减少。面向特征编程是在面向对象的基础上发展而来。由于传统 OOP 技术对于特征的支持不足，特征代码散乱，不利于软件复用和维护，同时对于实现软件产品线可变性存在不足。于是 FOP 方法作为 OOP 的扩展被提出用于弥补这些缺陷。在用 OOP 实现产品线的基础上，运用 FOP 方法把特征代码模块化，如此增量开发产品线系统，可以使得整个产品线的模块性更高。作为实验，根据用户需求，我们分阶段从基础程序开始增量开发网上缴费产品线系统，实现了 FOP 方法在辅助产品线开发上的实际应用。最后在验证了该方法的有效性的基础上进行了讨论。

到目前为止，FOP 技术还不是很成熟，同时 FOP 工具缺少集成开发环境的支持，并且 FOP 已有的工具难以实现界面级特征的组合（html，jsp 等），使得 FOP 思想和技术难以被工业界广泛利用，我们未来的很多工作将关注于 FOP 工具的集成与改进，以及如何在更普遍的产品线系统中广泛应用 FOP 思想。

References:

[1] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns（软件产品线实践与模式），张莉，王雷译。清华大学出版社，ISBN 7-302-07932-3/TP.5757.

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

[3] D. Batory, R.E. Lopez-Herrejon, and J.P. Martin, “Generating Product lines of Product-Families”, Proceedings of the 17th International Conference on Automated software Engineering, 2002, 81-92.

[4] C. Prehofer, “Feature-Oriented Programming: A Fresh Look at Objects”, Proceedings of the 11th European Conference on OOP, 1997, 419-443.

[5] Batory Don, Geraci Bart J. Composition validation and subjectivity in GenVoca generators[J]. IEEE Transactions on Software Engineering, 1997, 23(2):67-83.

[6] AHEAD Tool Suite, homepage: www.cs.utexas.edu/users/schwartz/ATS.html, Accessed October 2008.

[7] D. Batory, B. Lofaso, and Y. Smaragdakis, “JTS: Tools for Implementing Domain-Specific Languages”, Proceedings of Fifth International Conference on Software Reuse, June 1998, 143-153.

[8] K. Pohl, G. Bockle, and F.J. van der Linden, Software Product Line Engineering: Foundations, Principles and

Techniques, Springer Berlin Heidelberg New York, ISBN-10 3-540-24372-0.

[9] T.Asikainen, T.Mannisto, T.Soininen, “A Unified Conceptual Foundation for Feature Modelling”, Proceedings of the 10th International Software Product Line Conference, August 2006, 31-40.

[10] J. Liu, D. Batory, C. Lengauer, “Feature Oriented Refactoring of Legacy Applications”, Proceeding of the 28th international conference on Software engineering, 2006, 112-121.

[11] John D. McGregor, “The Evolution of Product Line Assets”, TECHNICAL REPORT, CMU/SEI-2003-TR-005, pp. 43-44.

[12] H. Zhang and S. Jarzabek, 2004. XVCL: A mechanism for handling variants in software Product Lines. Science of Computer Programming, volume 53(3), pp. 381–407, Elsevier.

[13] S.Trujillo, D.Batory, O.Diaz, "Feature Oriented Model Driven Development: A Case Study for Portlets", Proceeding of the 29th international conference on Software engineering, 2007, 44-53.

[14] C.Kim, C.Kastner, D.Batory, "On the Modularity of Feature Interactions", Proceedings of the 7th International Conference on Generative Programming and Component Engineering, 2008, 23-34.

[15] G.Freeman, D.Batory, and G.Lavender, "Lifting Transformational Models of Product Lines: A Case Study", Proceedings of the 25th International Conference on Model Transformations (ICMT), 2008, 16-30.