# 面向对象领域设计中的变化性处理

陈兆良,王千祥,梅 宏,杨芙清
(北京大学计算机科学技术系,北京 100871)

**摘 要**: 在领域工程中识别、描述和实现变化性,对应用系统的开发具有重要的指导意义和直接作用.其中在领域设计阶段,建立比较合理、比较灵活的 DSSA,将系统的可变部分与固定部分分离开来,将系统成分在 DSSA 和构件之间进行合理的分配,把变化性实现为构件,是领域工程的核心及关键性工作.本文运用设计模式和 OO 框架,提出了面向对象领域设计中常见变化性的解决方案,并将这些方案应用于 POS 领域的领域设计.

**关键词**: 软件复用; 软件构件; 领域工程; 领域设计; 设计模式; 变化性
**中图分类号**: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2001) 11-1486-05

## Dealing with the Variability in Object-Oriented Domain Design

CHEN Zhao-liang, WANG Qian-xiang, MEI Hong, YANG Fu-qing
*(Department of Computer Science and Technology, Peking University, Beijing 100871, China)*

Abstract: Identifying, representing and implementing the variability in domain engineering have a direct and important effect on application engineering; they are the emphases of domain engineering. Especially, building the reasonable and flexible DSSA (Domain Specific Software Architecture) in domain design, separating fixed parts and variable parts of systems, assigning the system elements between the DSSA and components reasonably, and implementing the variability as components are the nucleus and key work of domain engineering. This paper discusses the corresponding solutions to the variability of object-oriented domain design, employing the research results of design pattern and object-oriented framework; and presents how to use these solutions to deal with the variability in the domain design of POS (Point-of-Sale) domain.

Key words: software reuse; software component; domain engineering; domain design; design pattern; variability

## 1 引言

领域工程对一个领域中的若干系统进行分析,识别这些应用的共同特征和可变特征,对这些特征进行抽象,形成领域分析模型(Domain Analysis Model),依据领域分析模型产生出领域中一类应用系统共同具有的构架,即特定于领域的软件构架(Domain Specific Software Architecture,缩写为 DSSA),并以此为基础识别、开发和组织可复用构件[1].这样,当开发同一领域中的新应用时,可以复用领域工程的产品,根据领域分析模型,确定新应用的需求规约,根据特定领域的软件构架形成新应用的设计,并以此为基础选择可复用构件进行组装,从而得到新系统.领域工程包括三个主要的阶段:领域分析、领域设计和领域实现[2].与领域工程对应,称开发单个应用系统的软件开发过程为应用工程.

当在整个领域而不是单个系统的范围内考虑问题时,会发现系统的一些特性是领域中所有系统共同具有的,一些特性是部分系统所具有,而其它特性只是个别系统具有.所有系统都具有的特性,是该领域中系统的本质特性,体现为该领域中系统的共同性;而部分系统和个别系统具有的特性体现为领域中系统的变化性.识别共同性与变化性是领域工程的核心内容.变化性在本文特指在特定领域中具有普遍意义的可变特性,只是单个系统具有的个性成分,由于并不具有普遍意义而缺乏复用的价值,一般不予考虑.

领域共性的部分是通用于所有应用的,一旦得到实现之后,在应用工程中便不再作为关注的重点.而选择和配置不同抽象层次的变化性将成为贯穿应用工程全程的主要活动.因此,在领域工程中识别、描述和实现变化性,对应用工程的实施具有指导意义和直接作用,是领域工程中要重点处理的问题.尤其在领域设计阶段,建立比较合理、灵活的 DSSA,将系统的可变部分与固定部分分离开来,将系统成分在 DSSA 和构件之间进行合理的分配,把变化性实现为构件,是领域工程的核心及关键性工作.

面向对象(OO)技术通过抽象、封装、继承、聚合和多态等机制增强了对软件复用的支持[3],也为变化性的处理提供了很好的途径.一个 OO 框架由一组协作类组成,阐明了整个设计、类间依赖及成员类的责任分布.这些类通常是抽象类,实现细节放在具体子类中,构成一个抽象设计,不同的子类构成对设计的不同实现[4].

设计模式描述了如何利用面向对象的基本概念和机制解决可扩展的软件设计中经常出现的问题,针对反复出现的设计问题给出可复用的解决方案.文[5]介绍了 OO 设计中的 23 个设计模式,每一个设计模式(除 Singleton、Facade、Flyweight、Memento 模式外)都提供了对设计中的某种变化性的解决方

收稿日期:2001-01-03;修回日期:2001-06-07
基金项目:国家自然科学基金(No. 60043002);教育部高等学校骨干教师资助计划

案.在领域设计中,需要将系统的可变部分与固定部分分离开来,实现各种变化性,而设计模式正是提供如何针对不同的问题分离可变部分的解决方案,为进行这项活动提供了可供借鉴的思路.

本文结合设计模式和 OO 框架的研究成果,对面向对象领域设计中面临的常见变化性给出了相应的解决方案,并且采用这些方案解决了 POS 领域的领域设计中的变化性问题.

## 2　领域的变化性

下面从需求的变化性,及在现有系统中的表现形式两个维度来讨论领域设计中的变化性.

### 2.1　需求的变化性

变化性是领域中部分系统所具有的特性,即领域中系统可选的特性.按照在不同系统中出现的情况,可以将变化性分为以下三类:

(1)单个可选的

部分现有系统具有这类特性,但并非全部系统都具有.未来的系统可能具有这一特性,也可能不具有这一特性.例如,POS 领域里部分 POS 软件支持进行残损销售(将商品作为残品销售,销售价格由收款员输入),部分 POS 软件支持商品的时段优惠(某种商品在某个时间段内销售可以有优惠价格),但并非全部系统都有这些特性.

(2)成组可选的

可选的变化性有时以成组的方式出现.例如 POS 领域中,时段价优惠、会员价优惠、打折价优惠和积分优惠都是可选的,可以将它们组成一组"优惠方式"的变化性.一组可选的变化性作为整体,又可以是必须的(例如,必须至少一种基本的销售方式)或者是可选的(例如,可以不支持任何优惠方式).

(3)多选一的

这是一组互相之间存在互斥关系的特性,一个特定的系统必须具有,且只能具有其中的一项特性.例如,销售一件商品时,可能有多种优惠方式同时可选,这时就需要依据一定的优惠原则从中选择最终采用的优惠方式.而优惠原则又有三种:"最低价原则",选择使得商品价格最低的优惠方式;"最高价原则",选择使得商品价格最高(除零售价以外)的优惠方式;"优先级原则",为不同的优惠方式赋予不同的选用优先级.一个 POS 软件必须采用某种优惠原则,而且只能采用一种.多选一的变化性取值范围还可能是可变的.

变化性之间还可能存在依赖、互斥关系.依赖关系是指在变化性 $p$ 存在的情况下,才能存在变化性 $q$,这时称变化性 $q$ 依赖于变化性 $p$.互斥关系是指变化性 $p$ 和 $q$ 不能同时存在于一个系统中,是上面讨论的多选一变化性的一种特殊情况.

### 2.2　变化性在现有系统中的表现形式

面向对象领域设计中的元素是类、属性/服务、代码段、协作类、关系等.因此,分析阶段得到的变化性在现有系统的设计中往往具有不同的粒度,既可以是类、类簇(一组相互协作的类),也可能是一个或者多个相关的属性、服务,甚至可以是服务代码中的一个步骤、一个分支.

## 3　领域变化性的处理

### 3.1　变化性的实现方式

领域工程将领域中的共性抽象和实现为构架,而变化性则被分析、分类、组织和实现,最终体现为模型和实现中的变化性成分 —类构件、纯虚方法重载和模板等.

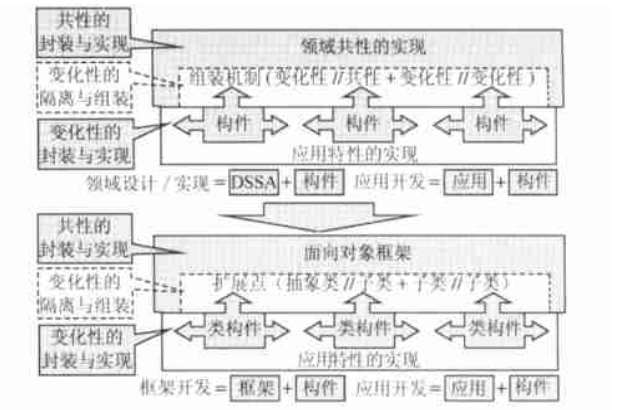


图 1　领域共性实现为构架,领域变化性实现为构件

设计一个灵活的 DSSA 是实现变化性的关键.将领域中的共性和变化性成分合理地分配到构架和构件中,使 DSSA 与构件间的接口尽可能地明确和简单,在实现上尽可能正交,Coplien 称这种方法为"变化性的规整化"[6].进行规整化时,一方面需要将变化性成分聚类,提炼变化性中的共性,向外提供一致而简单的接口;另一方面要封装可变的成分及其实现方式,保证在应用工程时选择/添加不同的变化性取值时不会破坏共性实现的结构.

在青鸟构件模型[7]中,构架是系统级的框架,DSSA 是领域实现的构架,也就是实现了领域变化性的框架.图 1 说明了如何实现变化性:将领域共性实现为面向对象框架,将规整化的领域变化性封装和实现为类或者类簇构件,在框架中用抽象类定义变化性的接口,变化性取值体现为抽象类的不同子类,固定变化性意味着将特定的子类增加到类层次结构中.

### 3.2　各种变化性的处理

不同类型的变化性,其处理方式往往也是不同的.

#### 3.2.1　多选一的变化性

多选一的变化性常常是某个比较抽象的、必须的系统成分的一组具体的实现方式.可以使用抽象父类和设计模式 Singleton 来实现多选一的变化性:抽象父类是多选一变化性的公共接口;Singleton 模式保证了在系统中只使用一个变化性选项.

(1)多选一的类

通常可以为一组多选一的类识别出一个公共的抽象父类,将这一组多选一的类作为该抽象父类的具体子类.子类中共同的部分移入抽象父类中,子类给出抽象父类的不同实现.这样就可以比较自然地将这些具体子类作为构件,将 DSSA 中需要与具体子类有关的成分作为构件的"使用指南",抽象父类是 DSSA 中的构件规约.在应用工程中,只要根据需求选择一个具体子类(构件)即可.

理想情况下,具体子类的对外接口是完全相同,只有内部实现方式不同,DSSA 中的其它成分只是访问抽象父类的接口.这样无论选择哪个具体子类,DSSA 中的其它成分都可以

不受影响.可以将 Factory Method 与 Singleton 结合起来使用,将实例的创建和引用封装在一致的、唯一的接口中.如果是多个多选一的变化性相互关联,必须匹配使用,则可以用 Abstract Factory 模式.图 2 是一个小例子,由原来的 Client 直接与 UserA 或者 UserB 打交道,变成与 Factory、User 打交道,从而实现了隔离和封装.

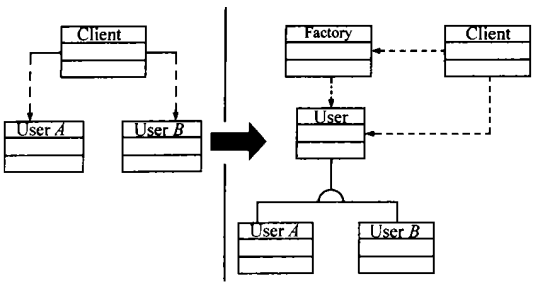


图 2 多选一的一组类的处理

(2) 多选一的属性/服务

如果在一个类中有一组逻辑上相关的多选一的属性,通常这个类中也会有与之相关的多选一的服务.有两种方法处理这种变化性:

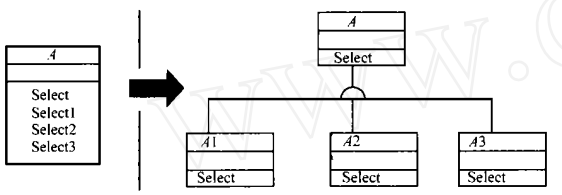


图 3 多选一的服务的处理(方法 1)

方法 1:将多选一的属性/服务抽象为虚方法,变化性选择体现为具体子类中对虚方法的不同实现.再利用新增的虚方法实现原有类的其他服务(参见 Template Method 模式),多选一的属性/服务就与系统的其他部分分离开了.如图 3 的例子,原来的 Select 是根据需求选择调用多选一的服务 Select1、Select2、Select 3 其中之一.

方法 2:将多选一的属性/服务从原来的类中分离出来,定义成一个抽象父类的多个具体子类,使用新增的抽象类实现原有的服务,作为原来的类的部分类.具体的设计方式可以参考 Strategy、Iterator 等设计模式.Strategy 模式支持将服务封装为类,Iterator 模式支持子类属性的封装和遍历.这样,多选一的属性/服务的问题就变成处理多选一的类的问题了.按这个方法,图 3 左边的类 *A* 可以如图 4 所示来处理.

### 3.2.2 单个可选的变化性

可选部分与系统其他部分之间可能存在双向的依赖关系.如果只有可选部分对其他部分的单向访问,则只需将可选部分实现为类或类簇即可.而当系统其他部分也访问可选部分时,就必须将系统中实现和使用可选变化性的成分组织到一起,与系统的其他部分分离,一起实现为类或类簇.

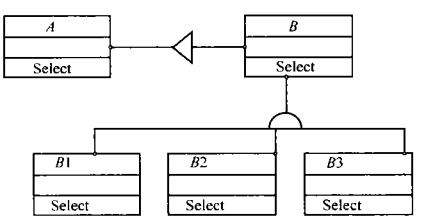


图 4 多选一的服务的处理(方法 2)

(1) 可选的类

对于单个的可选的类 *C*,一种自然的实现方式是将这个类作为构件.但类 *C* 通常不会孤立地存在,类 *C* 与其客户类 *D*1、*D*2 之间存在实例连接和消息连接.在相关的类 *D*1、*D*2 中通常会包含类 *C* 的实例或引用作为属性,在服务中也会访问类 *C* 的服务.由于类 *C* 是可选的,这些配合 *C* 而存在的属性是可选的,访问类 *C* 的服务也是可变的.对于这些属性和服务有两种可能的处理方式:

方法 1:将这些属性/服务之外的特性定义为 *D*1 等客户的父类,将可选属性/服务移到子类中.这样父类和子类就区别于是否使用可选变化性的两种情况,在子类中增加对可选构件的引用,即可实现可选的变化性.类 *C* 和相应的子类构成类簇构件,所有相关子类必须配合使用,这一点可以用 Abstract Factory 模式或用工具辅助实现.这种方法的缺点是,如果一个客户类用到多种可选的变化性,随着类层次的增长,子类个数呈指数增长.

方法 2:参考 Template Method、Strategy 等设计模式,将这样的属性/服务从原先的类中分离出来,定义为新的类,通过嵌套类或对象组装的方式实现,可以避免方法一造成的子类个数膨胀问题.当多个新类协作实现一个可选部分时,即构成类簇构件.在实现方式上,原有的类是新类的友元,新类是原有类的成员.

具体采用哪种处理方式,需要根据具体情况进行选择.选择的原则是使构件内部保持较高的内聚,构件与 DSSA 之间保持较低的耦合.

如果可选的是协作的一组类,则可以实现为类簇构件.协作类之间存在的依赖关系和互联关系是构件的内部实现,构件的客户只关心类簇对外的接口.

(2) 可选的属性/服务

当一个类中有可选的属性时,通常这个类中会有与这个属性相关的可选的或具有内部变化性的服务.处理这种变化性的基本方式是将可选的属性/服务分类从原有类中分离出来,参考 Strategy 等设计模式,定义为原有类的子类或部分类.如果可选成分影响到原有类中其他服务的实现,则可以将可变的服务分拆为若干个私有的子服务,再结合 Template Method 等模式封装变化性的影响.子服务对应于不同的可变代码段,在父类中实现为空,到子类实现中再填入代码.于是,可选的属性/服务就变成了可选的类,处理方法参见"可选的类".有时候,在区分类可选的属性/服务时,很难将完全屏蔽可选成分的影响,这时只能在设计中保留共性的代码,将可变代码留到组装之后手工编写.

对于协作的一组类中的可选属性/服务的处理,可以按照前面的方法将每个类中的可选属性/服务抽象为类,再以"可选的类"同样的方法实现为类簇构件.

### 3.2.3 成组可选的变化性

将成组的可选变化性作为整体实现时,对外应该提供一致的接口,对内应该支持成员构件的组装.

(1) 成组可选的类

用抽象父类作为这样一组可选的类的公共接口,用数组/链表/表格等动态数据结构保存具体类的对象,客户方利用多态性一致地访问它们.这样的可选类往往被用来封装业务策略的实现,用 Singleton 模式保证每种策略只有一个实现体.

(2) 成组可选的属性/服务

可以采用以下两种方法来实现成组的属性/服务的变化性.一种方法是:每个属性/服务对应于一个可选的变化性,可以将共性的成分定义为父类,将可选成分作为子类的特性,增加新的可选特性时,定义新的子类.另一种方法是,将一组属性抽象为一个用动态数据结构表示的共性属性,将相应的一组服务改为作用于这个共性属性的一个服务.例如,可以将"顾客卡"中的"是否打折卡?""是否会员卡?""是否储金卡?"等服务抽象为一个作用于"卡种类"属性的服务 —"是否一种(特定的卡)?".

### 3.2.4 具有内部变化性的服务

对于具有内部变化性(代码段)的服务,可以将服务中具有变化性的部分从服务中分离出来,定义为该类中可选的或多选一的服务,然后就可以使用可选的或多选一的服务的实现方式了.

### 3.2.5 小结

领域设计中变化性的情况是千变万化的,进行软件设计的方式也是千变万化的,因此,变化性的处理方式是不可能穷尽的.以上仅对常见的一些基本的情况进行了讨论.可以将上述方法归纳为"将服务中可变的内部实现封装为子服务"、"将可变的属性/服务抽象为类"、"将单个类实现为类构件"、"将一组协作类实现为类簇构件"等几条原则.此外,还可以结合条件编译等其他实现变化性的方法.

## 3.3 变化性的绑定时间

在单个系统的生命周期中,需要在某个时间将需求上的变化性固定下来,这个时间称为变化性的绑定时间(binding time).典型的绑定时间包括开发时、启动和运行时.开发时绑定意味着在系统开发的过程中(分析、设计、实现、编译、链接)固定变化性;启动时绑定意味着在系统提交用户后,在系统启动时刻通过读取配置信息,固定变化性;运行时绑定意味着在系统已经投入运行后,通过设置参数等手段,固定变化性.

变化性的绑定时间对领域设计有显著的影响.变化性的绑定时间越晚,对应用系统适应变化性的能力要求就越高.如果变化性在编译时刻及其之后绑定,就必须实现所有预计的变化性.如一组多选一的需求的绑定时间为运行时,就要求系统能够对这一组需求都进行支持,而且要提供在系统投入运行后从这一组需求中选择一个的功能.

# 4 POS 领域的实例

在 POS 软件中,优惠原则是多选一的变化性.代表一笔销售的类 CSellRecord,其中有一个选择优惠方式的服务 ChooseDiscount,是具有变化性的,属于 3.2.4 的情况.应用设计模式 Strategy,将优惠原则这一变化性从 CSellRecord 类中分离出来,如图 5 所示.定义抽象类 CDiscountManager,其中定义服务 ChooseDiscount.对应于每种优惠选择原则,定义 CDiscountManager 的一个具体子类,在子类的服务 ChooseDiscount 中实现这种优惠选择原则.变化性实现为三个类构件.

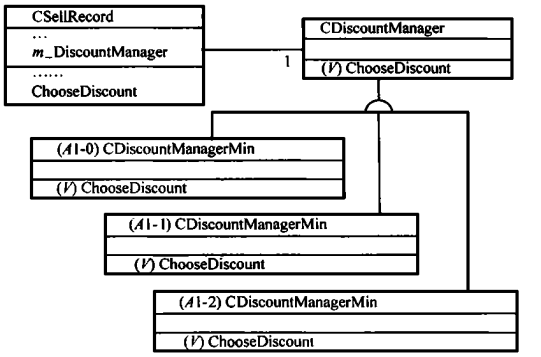


图 5

POS 系统还可以支持多种优惠方式:会员价、前台单个优惠、前台整体优惠等,这些是可选的变化性.这样,CDiscountManager 中的服务 ChooseDiscount 仍然具有变化性,即每个系统所支持的优惠方式集合是不同的,需要对优惠方式集合中的各种方式进行遍历,计算每种优惠方式的单价和折扣,从中选择最终采用的优惠方式.为了实现这种变化性,仍然可以采用设计模式 Strategy,将优惠方式也作为 Strategy 从 CDiscountManager 中分离出来.这样就得到如图 6 所示的解决方案.这里定义了抽象类 CDiscountMode,其中定义了服务 CalcPriceAndDiscount.对应于每种优惠方式,定义 CDiscountMode 的一个具体子类,变化性实现为六个类构件.

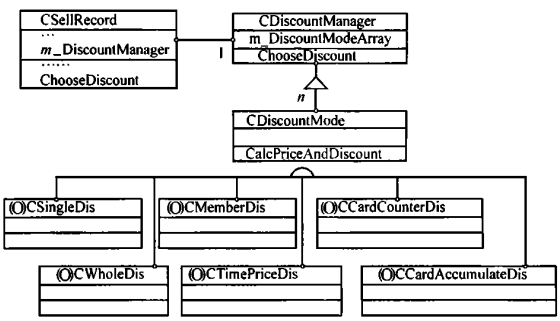


图 6

POS 领域的实践表明,在领域设计中对领域变化性的处理为应用系统的快速开发和演化提供了较好的支持.用户提出新的需求时,通常可以开发新的构件予以实现.只要领域设计能够容纳新的构件,就可以直接将构件组装到系统中.对共性和变化性成分实现方式的划分,很好地支持了现有构架的演化、新构件的开发,减小了开发新的系统所需的代价.在 POS 领域的构件化过程中,我们得到了以下一些数据:

® 开发"贴花销售"所需的构件并完成组装,时间为 2.5 小时;

® 开发"残损销售"所需的构件并完成组装,时间为 0.5 小时;

® 开发一种优惠方式所需的构件(如时段优惠)并完成组

装,时间平均为 15 至 20 分钟.

与通常增加相应功能的过程相比,上述采用领域设计结果进行的增加功能时间大大减少.

## 5 相关研究

Parnas 首次提出了用共性和变化性分析构造程序家族的方法,也就是"首先研究集合中成员的共同特征,然后确定单个成员的特殊特征"[8]. Parnas 还介绍了如何用信息隐蔽的方法适应系统在功能、结构、数据、算法和控制过程等方面的变化. 变化性封装的思想影响了面向对象方法中的封装机制和最近兴起的"设计模式"[5]——"识别和封装易变的成分".

在文[6]中,Coplien 对如何控制软件设计中的复杂性、如何处理设计中所遇到的变化性问题进行了深入的研究和介绍.

文[9～11]中介绍了在设计一个制造业框架的时候,如何利用设计模式来对框架的设计进行逐步的、一系列的改造,处理框架中的变化性,从而达到降低框架的复杂性、提高框架的灵活性和复用性的效果. Schmid 所采用的处理变化性的方法与本文的一些做法是基本一致的.

Yannis 提出了一种"基于协作的设计"(Collaboration-Based Designs)的方法来处理设计中的变化性,并将变化性实现为构件[12]. Yannis 的方法是一种分层次(layer)的方法,将一组类中相互协作、共同完成某个任务(变化性)的部分(类的属性、服务)分离出来,组成一个 Collaboration,用嵌套类、继承、参数化的方式来实现为构件,在构造具体系统的时候,再根据需求,选择这些构件进行组装. 这种变化性处理的方式类似于本文 3.2.2 的(1)所介绍的处理可选的类的方法 2.

IBM 的 San Francisco 框架是一个基于 Java 的构件集,它使得开发者不再从最底层开始构造商业应用系统,而是利用已经有的构件对服务器端的商业应用系统进行组装式的开发[13]. San Francisco 框架采用了设计模式以及扩展点、扩展技术来处理商业领域中的变化性.

设计重构需要花费大量的人力和时间,文[14]对面向对象设计重构的自动化支持方面进行了研究.

## 6 结束语

领域设计阶段是进行构件/构架开发的关键阶段. 在领域工程中,面对的是一个领域中具有变化性的多个系统,识别变化性是必须进行的,而且是比较自然的工作. 对于在领域分析阶段识别的领域中系统间的共同性和变化性,需要在领域设计阶段给出灵活的解决方案,从而使得开发出的 DSSA 和可复用构件在应用工程中可以通过选择和复用,形成领域中具有不同需求的应用系统. 基于这样的情况,我们在领域设计中参考设计模式中给出的对不同问题的解决方案,结合 OO 框架的研究成果,将系统的固定部分和可变部分在 DSSA 和构件间进行分配,处理领域的变化性,从而形成了灵活的 DSSA.

本文较为详细地阐述了领域变化性的情况,给出了在领域设计中处理这些变化性的手段,虽然采用这些手段可以基本上处理 POS 软件领域中出现的问题,但这些变化性的情况还是比较简单、比较常见的情况,我们还将继续对领域中变化性的情况进行进一步的总结与归纳.

**参考文献**:


[1] Ruben Prieto-Diaz. Domain analysis: An introduction [J]. ACM SIGSOFT Software Engineering Notes, April 1990, 15(2): 47 - 54.

[2] 青鸟工程. 青鸟领域工程指南 [Z]. 青鸟工程技术资料, 2000 年 5 月.

[3] 邵维忠,杨芙清. 面向对象的系统分析 [M]. 清华大学出版社,广西科学技术出版社, 1998 年 12 月.

[4] 杨芙清,梅宏,吴穹,朱冰. 基于异质构件复用的软件开发技术及其支持系统 [J]. 中国科学, 1997, 27(3): 275 - 281.

[5] E Gamma, R Helm, Ralph Johnson, J Vlissides. Design Patterns-Elements of Reusable Object-Oriented Software [M]. Addisom-Wesley, 1995.

[6] Jim O Coplien. Multi-Paradigm Design for C++ [M]. Addisom-Wesley Publishing Company, 1999.

[7] 青鸟工程. 青鸟构件模型 [Z]. 青鸟工程技术资料, 2000 年 1 月.

[8] David L Parnas. On the design and development of program families [J]. IEEE Transactions on Software Engineering, March 1976, SE-2(1): 1 - 9.

[9] Schmid H A. Creating the architecture of a manufacturing framework by design patterns [A]. In Proceeding of OOPSLA'95, ACM, NY 1995 [C], 370 - 384.

[10] Schmid H A. Fachhochschule konstanz, creating applications from components: A manufacturing framework design [J]. IEEE SOFTWARE, Nov. 1996, 13(6): 67 - 75.

[11] Schmid H A. Systematic framework design by generalization [J]. COMMUNICATIONS OF THE ACM, Oct. 1997, 40(10): 48 - 51.

[12] Yannis Smaragdakis, Don Batory. Implementing reusable object-oriented components [Z].

[13] K A Bohrer. Architecture of the san francisco frameworks [R]. San Francisco Frameworks, 0018 - 8670/98, 1998 IBM, 37(2).

[14] Lance Tokuda, Don Batory. Evolving objected-oriented designs with refactorings [A]. Automated Software Engineering 1999 [C].

[15] 李克勤. 面向对象的领域工程方法研究 [D]. 北京大学博士论文, 2000. 7.

[16] 常继传. 特定于领域的构件组装技术研究和实践 [D]. 北京大学硕士论文, 2000. 6.


**作者简介**:


**陈兆良** 男. 1975 年 9 月出生于广东省茂名市. 北京大学计算机科学技术系博士研究生,主要从事软件工程、软件复用和软件构件技术等方面的研究工作.