面向复用成本优化的构件重构方法*

王忠杰+, 徐晓飞, 战德臣

(哈尔滨工业大学 计算机科学与技术学院,黑龙江 哈尔滨 150001)

Reuse Cost Optimization Oriented Component Refactoring Method

WANG Zhong-Jie+, XU Xiao-Fei, ZHAN De-Chen

(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

+ Corresponding author: Phn: +86-451-86414906, E-mail: rainy@hit.edu.cn, http://www.hit.edu.cn

Received 2004-12-11; Accepted 2005-10-08

Wang ZJ, Xu XF, Zhan DC. Reuse cost optimization oriented component refactoring method. *Journal of Software*, 2005,16(12):2157−2165. DOI: 10.1360/jos162157

Abstract: During the whole lifecycle of component reuse, it is necessary for designers to continuously re-design and refactor components to improve usability, e.g., reuse cost, reuse efficiency, etc, under the guarantee of high usefulness. For this purpose, this paper presents a reuse cost optimization oriented, locality principle and instance set decomposition based component refactoring method. A feature-oriented component model is firstly introduced, with emphasis on component reuse mechanism based on variation point. By analysis of the constituents of reuse cost, the optimization goal, i.e., increasing the proportion of fixed part of a component, is put forward. Then, the temporal and spatial locality in component reuse process is analyzed elaborately, and according to the reuse frequency of each component instance, those instances with higher reuse frequency are separated out to form (semi-)instantiated independent components, so as to decrease the instantiation cost and implementation cost during reuse process. A greedy strategy based component instance set decomposition algorithm is proposed and validated by a practical case. This method may bring some instantiation tasks forward to component design phase instead of in practical reuse phase and transform some variable features into fixed ones by decomposing the dependencies between components into dependencies between component instances, therefore avoiding multiple instantiations during frequent reuse to decrease reuse cost.

Key words: software component; component refactoring; reuse cost; component instance decomposition

摘 要: 构件需要在其复用期间进行持续的优化改进和重构,消除设计需求与复用需求之间的差异,在保证有用

* Supported by the National Natural Science Foundation of China No.60573086 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2003AA4Z3210 (国家高技术研究发展计划(863)); the National Research Foundation for the Doctoral Program of Higher Education of China under Grant No.20030213027 (国家教育部博士点基金)

作者简介: 王忠杰(1978－),男,山东龙口人,博士生,主要研究领域为软件重构与复用;徐晓飞(1962－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为企业智能计算,管理与决策信息系统,数据库,企业资源计划与供应链管理技术,电子商务与商务智能;战德臣(1965－),男,博士,教授,博士生导师,主要研究领域为数据与知识工程,软件重构与复用,虚拟企业集成与电子商务,企业资源计划系统.

性的前提下改善可用性.为此,提出一种面向复用成本优化的、基于局部性原理与实例集分解的构件重构方法.首先给出一种基于特征的构件模型,着重探讨基于可变点的复用机制,并在此基础上研究构件复用成本的构成要素、优化策略与优化目标,即通过提高构件固定部分的比例降低复用成本.探讨了构件复用过程中存在的时间/空间局部性,依据构件实例复用频度的差异,将具有高复用频度的实例分离出来形成(半)实例化构件,以降低构件复用过程中的实例化成本与实现成本.进而提出一种基于贪心策略的构件实例分解算法实现近似最优化,并通过实例验证其有效性.该方法通过将构件特征间依赖关系分解为构件实例间依赖关系,将构件的部分实例化工作由复用阶段提前到设计阶段来完成,将若干可变特征转化为固定特征,从而避免了构件频繁复用时的多次实例化,以降低复用成本.

关键词: 软构件;构件重构;复用成本;构件实例分解

中图法分类号: TP311 文献标识码: A

构件的复用性体现在两个方面[1]:有用性和可用性.前者体现了构件可能被频繁使用的程度,后者体现了复用构件的容易程度,表现为复用成本和复用效率等方面.目前的工作更多地集中在如何提高有用性,但可用性方面则较少得到关注,一般都是在复用之后再对被复用构件和复用过程的成本和效率进行评价[2,3].实际上,构件的可用性非常重要,较低的可用性会在很大程度上削弱复用所带来的优势.

构件设计是一个持续和反复进行的过程.在构件设计完成并实际复用一段时间之后,需要依据实际复用情况对构件进行优化与持续改进(称为构件重构),在构件间重新分配语义功能和责任,以获得清晰、无冗余的体系结构,同时改善构件间交互与依赖关系,并保持构件集合稳定的协作行为[4],以持续提高构件复用性能.

目前有关构件重构的研究得到研究人员的日渐关注.Stojanović 提出一种 9-interaction matrix 方法[4],构造两个构件的交互矩阵并通过矩阵中参数间的值关系来优化构件间关系;Vitharana 等人构造了一种基于业务策略的构件设计模型(BusCod model)[5],综合表征构件复用性能的多个指标,形成构件重构的目标函数,并通过对相关参数的调整,达到目标函数的最优化;Caballero 等人建立了形式化的构件优化框架[6],依据构件的抽象层次,将构件间的依赖关系分为 8 种类型,通过依赖类型之间的转换实现构件优化.除此之外,还包括基于模式和元模型建模的重构方法[7]、基于类关联图和类可达性的 ECR 方法[8]等.同时,研究者将面向对象方法中类和类包的设计原则[9],如开放-封闭原则、Liskov 置换原则等,应用于构件重构中,并给出多种形式化的重构规则与步骤[10].

这些方法通常依据构件设计阶段产生的语义和结构信息,分析当前设计中存在的缺陷并加以改造,但没有考虑构件设计期望与实际复用情况之间的偏差.另外,目前也尚未发现针对复用成本优化的具体重构方法.

为解决上述缺陷,本文从基于特征的构件模型及其可变点机制出发,分析复用成本优化的目标与策略,将实际复用数据作为构件重构的重要依据,并给出一种基于实例分解的构件重构算法.

1 基于特征的构件模型及其复用机制

面向特征的方法[11]在软件复用领域得到广泛应用,使用特征刻画领域内相关业务系统的共性和差异,形成领域特征模型,实现软件复用[12].另外,特征的层次性、扩展性和多维性使得特征模型比其他方法更适合于表征领域变化因素.首先介绍基于特征的构件模型,并着重探讨其可变点机制.

1.1 构件特征模型

定义 **1**(特征). 特征依赖于描述客观世界知识的本体,表现为描述特定应用领域所提供服务的术语或概念[12].特征具有层次性:父特征与子特征形成层次关系,从而成多层次的特征空间.

定义 **2**(特征项). 特征项是特征的实例,表示特征在领域中不同应用环境下可能的取值,反映了特征自身的变化性.$dom(f)$表示特征 f 所有特征项的集合,称为 f 的值域.针对特定的业务领域,$dom(f)$是一个有限的集合.特征是对所有特征项的抽象,二者之间是一种泛化/例化关系,.

定义 **3**(特征依赖). 特征集合 X 与 Y 是两个特征集合.如果 X 中特征的取值决定了 Y 中特征的取值,则称 Y 特征取值依赖于 X,记为 $X\rightarrow Y$.对 X 的一个实例,对应着一个或多个 Y 的实例.

定义 **4**(构件特征模型). 一个构件定义了特定业务领域的一个特征子空间.构件特征模型可表示为

$C\langle cid,f_{root},F,D,PS,RS\rangle$,其中 cid 为构件的唯一标识; f_{root} 为构件特征空间的根特征,是构件所有其他特征的祖先特征;F 为构件特征空间包含的除根特征外其他特征构成的集合,对$\forall f\in F,dom(f)\geq 1$;$D$ 为特征依赖的集合;PS 为构件向外提供特征的集合,RS 为构件从外界需求特征的集合,且 $PS,RS\subseteq F$.

1.2 构件可变点机制

构件的复用性是通过特征的可变性体现出来的,表现为两个方面:

(1) 特征的可选性:某一特征只有在特定情况下才是构成构件特征空间的要素,而在其他情况下不是必需的.根据可选性的不同,特征可分为必选特征、可选特征、单选特征和复选特征[12];

(2) 特征项的可选性:特征可实例化为该特征值域中包含的任意特征项.

实际上,特征的可选性可转化为特征项的可选性.对任意非必选特征 f,可以通过在 f 的值域 $dom(f)$中加入一个空特征项 $I_{\varnothing}$将其转化为必选特征.当 f 未被选中时,则认为 f 被实例化为特征项 $I_{\varnothing}$.

按照值域的大小,构件中包含的特征可分为两类:

(1) 固定特征:满足$|dom(f)|=1$ 且 $dom(f)\neq\{I_{\varnothing}\}$的特征 f 称为固定特征;

(2) 可变特征:满足$|dom(f)|>1$ 的特征 f 称为可变特征.对可变特征 f,如果 $I_{\varnothing}\in dom(f)$,则称 f 为可选的可变特征.可变特征又称可变点.

构件 C 的所有固定特征形成 C 的固定部分 $FIX\_PART(C)$,所有可变特征形成 C 的可变部分 $VAR\_PART(C)$.如果 $VAR\_PART(C)=\varnothing$,则称 C 为固定构件,否则称为可变构件.

在满足特征依赖的前提下(特征依赖 $X\rightarrow Y$ 的存在使得 X 与 Y 两个特征集合的实例化不再独立,需要依据 X 的实例化结果对 Y 进行实例化以保证特征依赖的有效性),对构件中所有特征进行特征项的选取以得到构件实例.由于固定特征仅包含一个特征项,因此不必实例化;相反,可变特征具有取值上的可变性,表征了不同构件实例之间的区别,因此构件的实例化实际上就是对可变特征进行实例化的过程.所有构件实例的集合记为 instance(C).对$\forall f\in VAR_PART(C)$, $\forall t\in$instance(C),用$\rho(f,t)$表示 f 在 t 中所对应的特征项.构件实例化一般在复用时进行.

一个可复用构件除了提供规格说明之外,还需要提供相应的实现以形成可运行构件[1],才能为应用系统所复用.由于固定部分必须被复用,在构件设计阶段固定部分必须被相应的实现;对可变点来说,由于其包含了多个特征项,可在设计阶段实现,也可在复用阶段由复用者负责提供相应的实现.特征 f 的特征项 τ的实现记为 $impl(f,\tau)$,如果 τ在构件设计阶段尚未实现,则 $impl(f,\tau)=\varnothing$.

归纳起来,构件复用的基本过程分为 3 个阶段:(1) 依据特征依赖,对构件的每一个可变特征 f 进行实例化,从 $dom(f)$中选定一个特征项 τ;(2) 如果 $impl(f,\tau)=\varnothing$,实现该特征项;(3) 将构件与应用系统其他部分组合,完成复用.图 1给出了一个构件的简要示例.

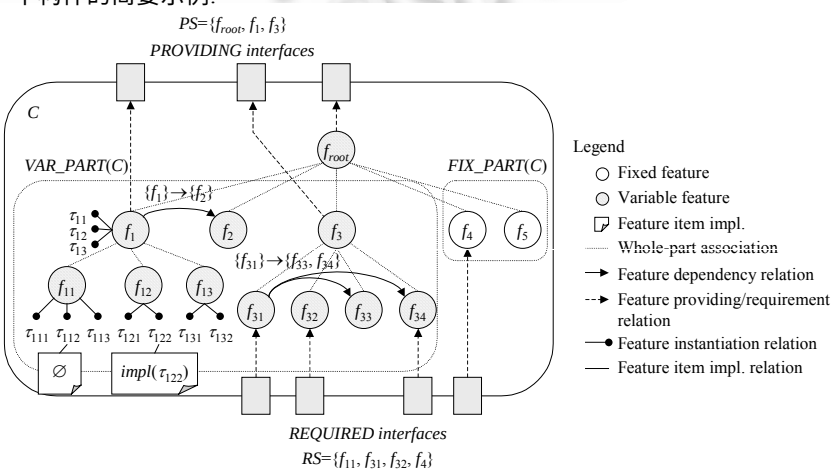

Fig.1 Feature-Based component model and its variation mechanism

图 1 基于特征的构件模型及其可变机制

2 构件复用成本优化的目标

首先研究群体构件的复用成本.由于构件是依据领域模型进行设计的,因此在针对特定的应用进行复用时,可能需要对构件进行一定程度的修改和调整才能满足需求.构成特定应用系统的构件分为以下 4 类:

- L1(直接复用):不加任何修改,直接复用;
- L2(实例化后复用):对抽象构件进行实例化,为每一个可变特征选取特定的特征项之后再加以复用;
- L3(修改后复用):现有构件无法完全满足重构需求,需要对其修改之后才能复用;
- L4(无复用):需开发新构件以满足需求.

设在使用构件集合进行系统开发中,构件 4 种复用方式所占的比率分别为 p_1,p_2,p_3 和 p_4,4 种复用方式的单位复用成本分别为 c_1,c_2,c_3 和 c_4.很容易理解,$c_1\le c_2\le c_3\le c_4$.那么为降低该构件的复用成本,可从以下目标入手:

目标 1:优化 4 类构件所占比率 p_1,p_2,p_3 和 p_4;

目标 2:优化单位复用成本 c_1,c_2,c_3 和 c_4.

针对目标 1,由于 $c_1\le c_2\le c_3\le c_4$,从变化成本角度讲,需要将构件复用时被修改的级别尽可能降低,即尽可能实现以下转化:(1) 无复用→修改后复用;(2) 修改复用→实例化/直接复用;(3) 实例化复用→直接复用.针对(1),可通过在每一次系统构造之后,将新设计与开发的构件加入构件库,通过逐渐的积累,降低日后复用过程中 L_4 构件的比例;针对(2)、(3),则需要通过对构件特征空间的改造即构件重构加以实现.

针对目标 2,首先分析一个特定构件的复用成本.构件复用成本可分为特征实例化成本 $CI(C)$、特征项的实现成本 $CP(C)$、构件变化成本 $CG(C)$和构件组合成本 $CC(C)$4 部分.对 L_1 类构件而言,$CI(C)=CP(C)=CG(C)=0$;对 L_2 类构件而言,由于固定部分无须实例化,$CI(C)$和 $CP(C)$取决于可变部分的实例化和实现成本;对 L_3 类构件而言,$CI(C)$和 $CP(C)$同样取决于可变部分的实例化和实现成本,而 $CG(C)$则取决于 C 被修改的程度.不管何类构件,$CC(C)$取决于构件之间交互关系的复杂程度.因此,构件的固定部分越大,构件的修改程度越低,与其他构件的交互越少,复用成本就越低,可通过提高可变构件中固定部分的比例、降低构件的修改程度、减少构件间交互 3 种手段来进行复用成本的优化.

本文方法拟通过对构件的重构降低目标 2 中个体构件实例化成本与实现成本,从而实现目标 1 中群体构件从“实例化复用”到“直接复用”的转化.

3 基于实例集分解的构件复用成本优化

从第 2 节的分析可以看出,应尽可能地将可变构件的可变部分转化为固定部分,或将可变构件转化为固定构件,是降低构件复用成本的途径.本节给出一种基于实例分解的构件重构方法,其基本思想是:构件不同,实例的复用频度不同,依据复用频度对所有构件实例进行分组,将具有高复用频度的实例组分离出来并形成独立构件,相当于将构件的部分实例化工作由复用阶段提前到构件设计阶段来完成,将若干可变特征转化为固定特征,从而避免了构件频繁复用时的多次实例化和特征项实现,以降低复用成本.

3.1 构件复用的局部性原理

一个构件 C 可能存在多个实例,C 被复用指的就是 C 的某一实例 $t\in\text{instance}(C)$被复用.与 CPU 访问内存地址空间时存在的局部性原理[13]类似,构件复用时也存在局部性原理,即在一定时期内,通常只有一部分构件被频繁地复用.同理,对构件的不同实例来说,也通常只有一部分实例被频繁地复用.局部性可分为以下两种类型:

(1) 时间局部性:如果一个构件或构件实例被多次复用,那么在近期它很可能还会再次被频繁复用;

(2) 空间局部性:如果一个构件或构件实例被频繁复用,那么在近期与其具有语意关联关系的其他构件或构件实例很可能也被频繁复用.

基于局部性原理对构件实例集合进行分解是有价值的,通过将可能被频繁复用的实例分离出来,建立构件实例缓冲区,用户在复用时可直接从缓冲区中获取复用实例,如果缓冲区中不存在,再对原构件进行实例化,就可以极大地降低因频繁实例化所导致的实例化成本.该过程如图 2 所示,图中标注数字表示各实例的复用频度.

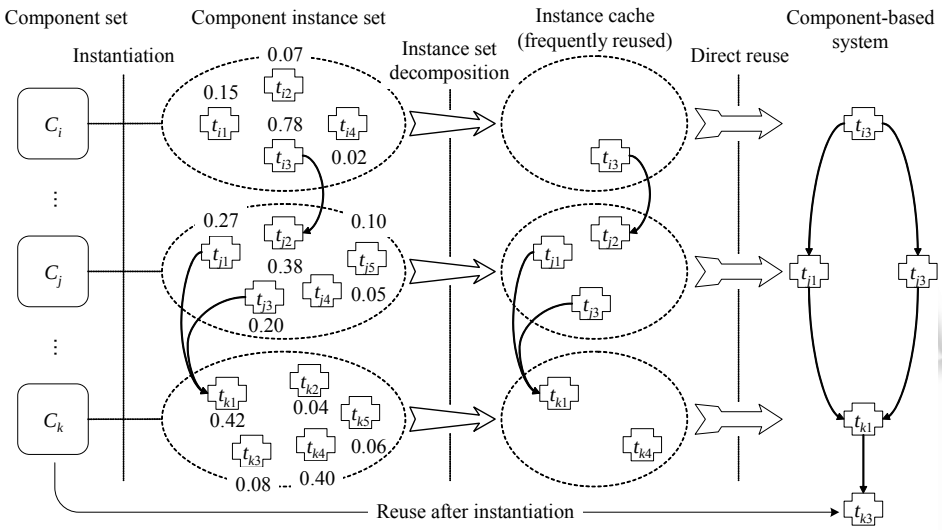

Fig.2 Principle of locality in component reuse

图 2 构件复用的局部性原理及构件实例缓存

3.2 构件实例分解

定义 5(构件实例分解).构件实例分解是指将一个构件 C 分解为一组构件 $\wp=\{C_1,C_2,\ldots,C_n\}$,并满足 $F(C_i)=F(C)$,且对任意可变特征 $f\in F(C)$,有 $dom(f|C_i)\subseteq dom(f|C)$,且 $\bigcup_{i=1}^{n} dom(f|C_i)=dom(f|C)$.另外,至少存在一个 C_i 和一个特征 g,使 $|dom(g|C)|>1$,$|dom(g|C_i)|=1$ 成立.式 $dom(f|C)$ 表示特征 f 在构件 C 内的值域.

一种特殊情况是按照某一特征 f 进行实例分解,将 $dom(f)$ 中的每一个特征项分解到不同的构件中,而其他特征的值域则保持不变.设 $dom(f)=\{\tau_1,\tau_2,\ldots,\tau_m\}$,则分解得到 m 个构件 $\{C_1,\ldots,C_m\}$,对每一个 C_i,有 $dom(f|C_i)=\{\tau_i\}$,且对 f 之外的其他任何特征 g,有 $dom(g|C_i)=dom(g|C)$.

构件实例分解要满足完备性、正确性和不相交性.完备性是指分解得到的构件 $C_1,C_2,\ldots,C_n$ 实例集合的并集能够覆盖 C 的所有实例,即 $\bigcup_{i=1}^{n}\text{instance}(C_i)\supseteq\text{instance}(C)$;正确性是指 $C_1,C_2,\ldots,C_n$ 实例集合的并集不包含非 C 的实例集合中的实例,即 $\bigcup_{i=1}^{n}\text{instance}(C_i)\subseteq\text{instance}(C)$.二者结合起来,要求 $\bigcup_{i=1}^{n}\text{instance}(C_i)=\text{instance}(C)$,这保证了在使用 $C_1,C_2,\ldots,C_n$ 来代替原构件 C 后,构件的复用度不受影响.不相交性是指任意两个构件 C_i,C_j 的实例集合不相交,$\text{instance}(C_i)\cap\text{instance}(C_j)=\varnothing$,以避免冗余.

在实例分解 $\wp$ 中,对某一 C_i,如果 C 的任一可变特征 f 均满足 $|dom(f|C_i)|=1$,则称 C_i 为 C 的一个实例构件;如果 C_i 中一部分可变特征满足 $|dom(f|C_i)|=1$,而其他可变特征满足 $|dom(f|C)|>1$,则称 C_i 为 C 的一个半实例构件.实例分解后得到原构件的一组半实例构件和实例构件,比原构件的可变点数目减少,固定部分比例增加.

在图 3 中,从构件 C 分解出一个半实例化构件 C_1 和构件 C_2.可以看出,C_1 中特征 f_2 的特征项数目为 1,可看作固定特征.

实例分解存在的一个问题是:所有固定特征在各子构件中重复出现,可变特征的若干特征项也在各子构件中重复出现.这实际上是通过特征(项)的冗余来达到降低复用成本的目的.

3.3 构件实例分解算法

构件的可变点可能有多个,每个可变点的特征项也较多,因此构件实例分解的方案必有很多种.如何选择恰当的分解方案是需要解决的一个问题.按照局部性原理提出以下分解规则:

规则 **1**. 如果 $p(t,C)\geq\psi$(ψ为复用频度阈值),那么将 t 分解出来,形成实例化构件;

规则 **2**. 如果 $p(t,C)\geq\psi$且存在其他构件 C'的实例集合 T 使得 $t\rightarrow T$,那么对 C',将 T 分解出来形成实例化构件.

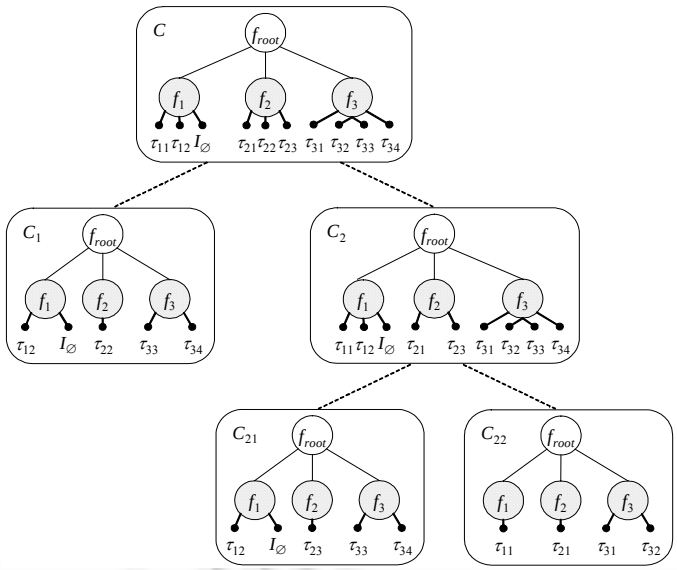

Fig.3 Example of instance-based component decomposition

图 3 构件实例分解

上述规则是直接应用局部性原理得到的分解规则,但如上节所述,这种直接的分解可能会造成特征项的大量冗余,从而对构件的维护造成较大影响.为此,我们将构件实例的复用频度转化为特征项的复用频度,通过对特征项的分解,在提取高复用频度的构件实例的同时,尽可能降低特征项冗余的程度.

设 $C_1,C_2,\ldots,C_m$ 是 C 经过某一实例分解之后得到的构件集合,那么对其中任一构件 C_i,其相对于 C 的复用频度 $p(C_i,C)$定义为 C_i 的每一个实例(同时也是 C 的实例)的复用频度之和,即

$$p(C_i,C)=\sum_{t\in \mathrm{instance}(C)} p(t,C) \tag{1}$$

C 的可变特征 f 的特征项 τ的复用频度 $p(\tau,f,C)$定义为所有包含该特征项的构件实例的复用频度之和,即

$$p(\tau,f,C)=\sum_{t\in T_\tau} p(t,C),T_\tau=\{t|t\in \mathrm{instance}(C)\wedge \tau=\rho(f,t)\} \tag{2}$$

由上式可知,特征项的复用频度分布较之构件实例的频度分布更加集中.

定义 **6**(固定部分改善程度). 构件 C_i 相对于 C 的固定部分改善程度 $v(C_i,C)$定义为满足$|dom(g|C)|>1$ 且 $|dom(g|C_i)|=1$ 的特征 g 的总数目与 C 的可变特征的数目的比值,即

$$v(C_i,C)=\left|\left\{g\,\middle|\,dom(g|C)>1\wedge dom(g|C_i)=1\right\}\right| \Big/ \left|VAR\_PART(C)\right| \tag{3}$$

定义 **7**(特征项冗余度). 特征 f 的特征项冗余度 $r(f)$定义为 f 的每一个特征项在分解后所有构件中重复出现的次数之和,可通过下式计算得到:

$$r(f)=\sum_{\tau\in dom(f)}\left|\left\{C_i\,\middle|\,\tau\in dom(f|C_i)\wedge C_i\in\wp\right\}\right|-1 \tag{4}$$

归纳起来,基于实例的构件分解要针对两个目标进行优化:(1) 尽量增加构件的固定部分;(2) 尽量减少构件分解后的数目以及特征项冗余.这两个目标是相互约束的:增加固定部分要求对构件进行更深入的实例化,但同时必然造成分解后构件数目过多,这些构件中特征项重复出现的次数过多.使用下式描述构件实例分解 $\wp$ 的水平:

$$G(\wp)=\frac{\sum_{i=1}^{n}\left(p(C_i,C)\times v(C_i,C)\right)}{\sum_{j=1}^{m}r(f_j)} \tag{5}$$

其中 $p(C_i,C)$为 C_i相对于 C 的复用频度,$v(C_i,C)$为 C_i相对于 C 的固定部分改善程度,$r(f_j)$为 f_j的特征项冗余度,n为最终得到的(半)实例化构件的数目,m 则为 C 中包含特征的数目.

在上述定义的基础上,给出一种构件实例分解的贪心算法.该算法的基本思想是:每次均选取具有最大复用频度的特征项τ,并将τ所属的可变特征 f 的值域按$\{\tau\}$和 $dom(f)-\{\tau\}$进行分解,形成半实例化构件 C_1,C_2.将 C_1加入最终的构件集,并针对 C_2重复上述过程,直到分解的水平不再有改善为止.

算法. 构件实例分解.

输入:构件 C 及其每一个实例 t 的复用频度 $p(t,C)$.

输出:一组半实例化构件 $\wp=\{C_1,C_2,\ldots,C_m\}$.

Step 1. 令 $C_0=C$, $\wp=\{C_0\}$, $\wp'=\{C_0\}$,并设 $G(\wp)=-\infty$;

Step 2. 依据式(3)计算 C_0 中每一个可变特征 f 的每一个特征项τ的复用频度 $p(\tau,f,C_0)$;

Step 3. 在 C_0所有可变特征的所有特征项中,选取具有最大复用频度的特征项τ_f,τ_f属于特征 f.将 f 的值域分解为$\{\tau_f\}$和 $dom(f)-\{\tau_f\}$两部分,并将 C_0 按特征 f 实例分解为两个构件 C_1 和 C_2,满足 $dom(f|C_1)=\{\tau_f\}$,$dom(f|C_2)=dom(f)-\{\tau_f\}$,且对 f 之外的其他任一可变特征 g,有

$$dom(g|C_1)=\left\{\tau_g\middle|\tau_g\in dom(g)\wedge\exists t\in \text{instance}(C),\tau_g=\rho(g,t),\tau_f=\rho(f,t)\right\},$$

$$dom(g|C_2)=\bigcup_{\tau_{f'}\in dom(f)-\{\tau_f\}}\left\{\tau_g\middle|\tau_g\in dom(g)\wedge\exists t\in \text{instance}(C),\tau_g=\rho(g,t),\tau_{f'}=\rho(f,t)\right\}.$$

Step 4. 在 $\wp'$中使用 C_1,C_2 替换 C_0,并依据式(1)、式(2)、式(4)、式(5),计算 $G(\wp')$;

Step 5. 如果 $G(\wp')>G(\wp)$,则令 $\wp=\wp'$.此时 $\text{instance}(C_2)=\text{instance}(C)-\{t|t\in\text{instance}(C)\wedge\tau_f\neq\rho(f,t)\}$,且对 $\forall t\in\text{instance}(C_2)$,$p(t,C_2)=p(t,C)$.令 $C_0=C_2$,并返回 Step 2;如果 $G(\wp')\leq G(\wp)$,算法结束,$\wp$即为最终的子构件集合.

算法在每一次循环中,将构件 C_0 分解为两个构件 C_1 和 C_2,由 Step 3 可知,C_1 中包含的实例均满足 $dom(f|C_1)=\{\tau_f\}$,而由 Step 5 可知 C2 的实例集合满足 $\text{instance}(C_2)=\text{instance}(C)-\{t|t\in\text{instance}(C)\wedge\tau_f\neq\rho(f,t)\}$,保证了不相交性.另有 $\text{instance}(C_1)\cup\text{instance}(C_2)=\text{instance}(C)$,即保证了分解的完备性和正确性.

算法中每次循环均将具有最大复用频度的特征项分离出来,直到分解的水平不再改善为止.由于构件可变特征的数目以及每一个可变特征的值域都是有限的,算法必然会终止.该算法执行的最坏情况是将 C 的每一个实例均分解出来,形成实例化构件,故其可能的最大循环次数为 $n=|\text{instance}(C)|$.而 Step 1~Step 5 的时间复杂性分别为 $O(1),O(m),O(m),O(1),O(1)$,其中 $m=\sum_{f\in VAR\_PART(C)}\left|dom(f)\right|$ 为 C 中所有可变特征的特征项数目之和,故整个算法的时间复杂性为 $O(mn)$.

3.4 应用实例

表 1 给出了图 3 中构件 C 的所有实例复用频度的示例数据,表 2 上半部分则是依据表 1 得到的该构件可变特征的每一个特征项的复用频度.

按照特征项复用频度的大小,首先选择 f_2 的特征项 τ_{22},分解得到 $\wp=\{C_1,C_2\}$,其分解水平为 $G(\wp)=-0.49$.计算得到 C_2的各特征项复用频度见表 2 下半部分,接着选择 f_2的特征项 τ_{23}对 C_2进一步分解,得到 $\wp=\{C_1,C_{21},C_{22}\}$,其分解水平为 $G(\wp)=-0.13$.此时如对 C_{22}再次分解,计算得到的构件分解水平下降,故算法停止,最终得到的构件实例划分集合为 $\wp=\{C_1,C_{21},C_{22}\}$,如图 3 所示.

表 3 给出了分解过程中各相关构件内包含的特征项的复用频度的变化情况;表 4 则给出了在分解前后构件复用及实例化的相关统计数据,从中可以看出,分解构件 C 与分解后的半实例化构件集合$\{C_1,C_{21},C_{22}\}$平均实例化次数从 3 下降到 1.93,从而实例化成本得到较大降低.

Table 1 Reuse frequency for component instances
表 1 构件实例的复用频度

Component instance	f_1	f_2	f_3	$p(t,C)$
t_1	τ_{11}	τ_{21}	τ_{31}	0.03
t_2	τ_{11}	τ_{21}	τ_{32}	0.04
t_3	τ_{12}	τ_{22}	τ_{33}	0.25
t_4	τ_{12}	τ_{22}	τ_{34}	0.32
t_5	τ_{12}	τ_{23}	τ_{33}	0.02
t_6	τ_{12}	τ_{23}	τ_{34}	0.09
t_7	$I_\varnothing$	τ_{22}	τ_{33}	0.08
t_8	$I_\varnothing$	τ_{22}	τ_{34}	0.06
t_9	$I_\varnothing$	τ_{23}	τ_{33}	0.07
t_{10}	$I_\varnothing$	τ_{23}	τ_{34}	0.04

Table 2 Reuse frequency for variable feature's items
表 2 构件可变点的特征项的复用频度

f_1	$p(\tau, f_1, C)$	f_2	$p(\tau, f_2, C)$	f_3	$p(\tau, f_3, C)$
τ_{11}	0.07	τ_{21}	0.07	τ_{31}	0.03
τ_{12}	0.68	τ_{22}	0.71	τ_{32}	0.04
$I_\varnothing$	0.25	τ_{23}	0.22	τ_{33}	0.42
				τ_{34}	0.51

Reuse frequency for C_2's feature items after the first decomposition

f_1	$p(\tau, f_1, C_2)$	f_2	$p(\tau, f_2, C_2)$	f_3	$p(\tau, f_3, C_2)$
τ_{11}	0.07	τ_{21}	0.07	τ_{31}	0.03
τ_{12}	0.11	τ_{23}	0.22	τ_{32}	0.04
$I_\varnothing$	0.11			τ_{33}	0.09
				τ_{34}	0.13

Table 3 Reuse frequency of feature items in each component
表 3 分解过程中各构件包含特征项的复用频度

	f_1			f_2			f_3			
	τ_{11}	τ_{12}	$I_\varnothing$	τ_{21}	τ_{22}	τ_{23}	τ_{31}	τ_{32}	τ_{33}	τ_{34}
C	0.07	0.68	0.25	0.07	0.71	0.22	0.03	0.04	0.42	0.51
C_1	0	0.57	0.14	0	0.71	0	0	0	0.31	0.38
C_2	0.07	0.11	0.11	0.07	0	0.22	0.03	0.04	0.09	0.13
C_{21}	0	0.11	0.11	0	0	0.22	0	0	0.09	0.13
C_{22}	0.07	0	0	0.07	0	0	0.03	0.04	0	0

Table 4 Component reuse data before and after CIS-decomposition
表 4 分解前后构件复用情况分析

Component instance	Before CIS-decomposition			After CIS-decomposition		
	The component to be reused	Number of variable features	Reuse frequency	The component to be reused	Number of variable features	Reuse frequency
t_1	C	3	0.03	C_{22}	1	0.03
t_2	C	3	0.04	C_{22}	1	0.04
t_3	C	3	0.25	C_1	2	0.25
t_4	C	3	0.32	C_1	2	0.32
t_5	C	3	0.02	C_{21}	2	0.02
t_6	C	3	0.09	C_{21}	2	0.09
t_7	C	3	0.08	C_1	2	0.08
t_8	C	3	0.06	C_1	2	0.06
t_9	C	3	0.07	C_{21}	2	0.07
t_{10}	C	3	0.04	C_{21}	2	0.04
Average instantiation times	3			1.93		

4 结束语

本文算法依据构件在实际复用中不同实例的复用频度的差异,将特征之间抽象的、高层次上的依赖关系分解为特征实例之间的一组具体的、低层次上的依赖关系.这种变换不是永久性的.这是因为构件的每一个具体实例的复用频度并非固定不变,在某一时期,一个实例可能被频繁地复用;在其他时期,该实例可能不再被频繁复用.因此这种变换也可看作是"领域业务需求"+"特定时期复用需求"的复合.该方法适用于大型软件复用组织内针对大量抽象构件的长期复用,依据不同时期不同软件产品的开发需求对构件进行重构.

如第 3.2 节所述,构件实例分解后得到的多个构件实例给构件管理带来了问题.为此可以将某一时期较少复用的构件实例重新合并为半实例化或抽象构件,可看作本文方法的逆过程.

本文方法的不足之处在于:由于针对面向特征的构件模型,在应用时不得不按照某种具体构件模型(如实体构件、过程构件)的特性对算法进行派生与定制.

References:

[1] Mili H, Mili A, Yacoub S, Addy E. Reuse-Based Software Engineering: Techniques, Organization, and Controls. New York: John Wiley and Sons, 2002.

[2] Mili A, Chmiel SF, Gottumukkala R, Zhang L. Managing software reuse economics: An integrated ROI-based model. Annals of Software Engineering, 2001,11:175−218.

[3] Rothenberger MA, Hersbauer JC. A software reuse measure: Monitoring an enterprise-level model driven development process. Information & Management, 1999,35(5):283−293.

[4] Stojanović Z. A method for component-based and service-oriented software systems engineering [Ph.D, Thesis]. Delft University of Technology, 2005.

[5] Vitharana P, Jain H, Zahedi FM. Strategy-Based design of reusable business components. IEEE Trans. on Systems, Man, and Cybernetics—Part C: Applications and Reviews, 2004,34(4):460−474.

[6] Caballero R, Demurjian S. Towards the formalization of a reusability framework for refactoring. In: Gacek C, ed. Proc. of the 7th Int'l Conf. on Software Reuse. LNCS 2319, Berlin: Springer-Verlag, 2002. 293−308.

[7] France R, Ghosh S, Song E, Kim DK. A metamodeling approach to pattern-based model refactoring. IEEE Software, 2003, 20(5):52−58.

[8] Washizaki H, Fukazawa Y. Automated extract component refactoring. In: Proc. of the 4th Int'l Conf. XP 2003. LNCS 2675, Berlin: Springer-Verlag, 2003. 328−330.

[9] Martin RC. Agile Software Development: Principles, Patterns, and Practices. New York: Prentice Hall, 2002.

[10] Quan L, He JF, Liu ZM. Refactoring and pattern-directed refactoring: A formal perspective. Tokyo: Int'l Institute for Software Technology, the United Nations University, 2005.

[11] Kang KC, Kim S, Lee J, Kim K, Shin E, Huh M. FORM: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 1998,5:143−168.

[12] Jia Y. The Evolutionary component-based software reuse approach [Ph.D. Thesis]. Beijing: Graduation School, the Chinese Academy of Sciences, 2002 (in Chinese with English abstract).

[13] Lee JH, Lee JS, Kim SD. A new cache architecture based on temporal and spatial locality. Journal of Systems Architecture, 2000, 46(15):1451−1467.

附中文参考文献:

[12] 贾育.基于演化计算构件的软件复用方法[博士学位论文].北京:中国科学院研究生院,2002.