ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.6, June 2006, pp.1287−1297
DOI: 10.1360/jos171287

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

以体系结构为中心的构件模型的形式化语义*

楚 旺+, 钱德沛

(西安交通大学 计算机科学与技术系,陕西 西安 710049)

Formal Semantic of Architecture-Centric Component Model

CHU Wang+, QIAN De-Pei

(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

+ Corresponding author: E-mail: chuw1119@163.com

Chu W, Qian DP. Formal semantic of architecture-centric component model. *Journal of Software*, 2006, 17(6):1287−1297. http://www.jos.org.cn/1000-9825/17/1287.htm

Abstract: Existing software development approaches construct component model in an unstructured and informal way, and the dependencies among components are implicit and lack rigorous semantic. They don't support top-down component reuse effectively. This paper tries to address these problems by using category theory to define the links between components, so that the links between components and the link compositions have rigorous meaning. The morphism composition is used to trace the relationships between components at different abstraction levels, and the architecture design pattern is used to define component composition and constrains precisely. Category theory supports the diagrammatic representation of component model that visualizes the relationships between components and the structural feature. The constructed component model is understandable, traceable, and hence reusable. Formal component modeling approach is suitable to the software development for reuse.

Key words: architecture; component model; component traceability; formal semantic; software reuse

摘 要: 目前的软件开发方法采用非结构化和非形式化方式建立构件模型,构件之间的关系是隐含的,并且缺乏严格的语义,不能有效地支持自顶向下的构件重用.利用范畴论定义构件之间的关系,使得构件之间的关系以及关系组合具有严格的语义.态射合成被用来跟踪不同抽象层次的构件之间的关系,利用体系结构设计模式精确地定义构件的组合关系和应满足的条件.范畴论支持图形化建模,可以使模型中的构件关系以及结构特征可视化,有利于模型的理解、跟踪和重用.形式化的构件建模方法适用于以重用为目标的软件开发.

关键词: 体系结构;构件模型;构件可跟踪性;形式化语义;软件重用
中图法分类号: TP311 文献标识码: A

长期以来,"软件重用"被引入软件开发并一直被作为软件工程的重要目标之一."软件重用"的主要目的有3个:减少开发与维护成本;缩短开发时间;提高软件质量.但是,从早期的"Copy-Paste"方式的程序代码重用、函数库重用到目前的对象类重用和构件重用,这些重用策略在开发实践中并未达到预期的效果.我们可以从软件开

* Supported by the National Grand Fundamental Research 973 Program of China under Grant No.G1999032710 (国家重点基础研究发展规划(973))

Received 2005-12-15; Accepted 2006-03-28

发的实践中得出以下结论:一是“可重用性”并不是软件代码内在的普遍特征,人们解决问题的方法才是真正有效的可重用资产[1];二是自底向上的重用方法不适用于工程实践,对于构件集成而言,“设计重用”应先于“构件重用”,一个自顶向下的方法可以以体系结构的形式为构件重用提供环境[2,3];三是问题域与软件构件之间应具有良好的可跟踪性,这是理解、维护和重用软件构件的基础和前提,软件重用不仅包括软件构件本身,还包括构件之间的关系.

实践经验同样证明:重用策略在同一领域中可以发挥更好的作用,许多开发组织已经认识到大多数的开发项目并不是一次性的,可以为同一领域内的未来项目开发可重用的核心资产,即为重用而开发(development for reuse),开发者希望软件开发能够像硬件生产那样大规模地展开,即利用重用进行开发(development with reuse).由于软件构件的开发和具体产品的开发是由不同的组织完成,软件构件的可理解性和可跟踪性是构件重用的前提,因此,寻找适当的抽象机制描述开发经验以及建立问题域和软件构件之间的跟踪模型是该领域研究的难点.目前,可跟踪性还没有被广泛使用,跟踪模型还未被清晰地定义,可跟踪实体的类型不清晰,实体的粒度太大,而且大多数跟踪框架只是简单地识别存在关联,而没有定义关联的语义,这种语义可以帮助开发者理解所做的跟踪[4,5].

体系结构在软件开发中的重要作用已经得到广泛的认知:体系结构使得开发者将重点由代码行转向大粒度构件的设计和重用上;体系结构体现了系统的设计决策,通过体系结构可以将复杂系统进行层次化抽象,层次化的模型便于理解、管理、跟踪和重用[6,7],以体系结构为中心的软件开发方法正在成为支持软件重用的有效途径.但是,目前的软件过程仍侧重于利用非形式化的方法构建不同抽象层次的软件模型,不同开发阶段所产生的工作产品之间可跟踪性没有建立起来,不利于利用自动化工具对软件模型进行验证、管理、维护和重用[8].为了实现构件的可重用性,构件模型必须是可跟踪的和可理解的,这就要求不同抽象层次的构件及其关系应该有明确和严格的定义.本文采用范畴论描述以体系结构为中心的构件模型,使得不同抽象层次构件之间的关系具有明确的语义,而且范畴论支持图形化建模,可以提高构件模型的可理解性,为支持可视化“Drag & Drop”的构件选择和组装奠定理论基础.

1 相关研究

近年来,基于体系结构的软件重用已经成为一个十分活跃的研究领域.梅宏等人在文献[9]中提出了基于体系结构、面向构件的软件开发方法,体系结构不再局限于高层设计,而是扩展到软件开发的全过程,在软件生命周期的各个阶段间保持良好的可跟踪性.他们在文献[10]中利用面向特征(feature-orientation)的映射方法建立需求与软件体系结构的可跟踪性.

为了在不同的抽象层次上实现构件的重用,Bachmann 等人在文献[11]中提出了基于体系结构的软件设计方法(architecture-based design,简称 ABD).这种方法以软件需求作为输入,通过不断地分解,将高抽象层次的系统分解为低抽象层次的系统.每一次分解时,高抽象层的功能需求被细化并指派给低抽象层的构件,通过选择适当的体系结构风格来满足在质量和业务目标方面的要求.ABD 方法将被设计的系统看作是由构件及构件之间的关系组成的体系结构,其产生的工作产品包括一组概念构件(conceptual component)、一组描述构件之间交互关系的模板(template)以及有关实施方面的约束条件.基于体系结构的设计方法使得每个概念构件具有结构简单和功能简单的特点,而且每个构件可以独立开发.

我们在文献[12,13]中将体系结构的构建引入软件构件开发(从业务建模到软件构件实现)的全过程,利用模式(software pattern)描述软件构件开发过程中的设计决策,并依此建立软件构件的跟踪模型,这一过程是随着构件的开发过程自动进行的.

目前的研究侧重于以体系结构为中心的设计方法的应用以及对体系结构自身的形式化描述,而对不同抽象层次体系结构中构件之间的依赖关系及其组合的语义缺乏深入的研究.这不仅影响了体系结构设计过程中对设计结果的验证,而且影响了在构件选择和组装过程中对构件的理解.

尽管范畴论在一些计算机科学的研究领域得到广泛重视,但在以体系结构为中心的构件建模方面尚未得

到应用.本文将利用范畴论对体系结构中的构件关系进行描述,并给出构件模型的形式化语义.

2 以体系结构为中心的构件模型的范畴论语义

要重用一个构件,首先要回答 3 个问题:“它能做什么?”、“它为什么存在?”、“它需要与哪些构件相互协同?”.构件规范可以回答第 1 个问题,其他问题则需要理解构件之间的关系及其组合.在基于体系结构的构件开发过程中会产生大量的不同抽象层次的构件(如业务构件、需求构件、设计构件和软件构件),这些构件之间存在着不同的直接关系和间接关系.明确定义构件关系有 3 方面的作用:一是有助于在构件开发过程中对构件之间的关系进行验证,以保证构件的正确性.对于以重用为核心的软件开发而言,如果构件关系存在错误,也会被“重用”到产品中,修改它们将会付出很大的代价.二是可以通过构件关系及其组合跟踪业务目标是如何实现的,支持自顶向下的重用策略.三是在构件维护过程中,可以通过构件关系确定构件修改所影响的范围和相应的测试目标.为了支持构件重用和维护,对构件关系的研究具有十分重要的意义.

构件的开发和重用往往是由多个不同的开发小组完成的.在构件开发的各个阶段,一些重要的构件关系是否得以保持?构件组合后是否满足要求,即构件之间的关系及其组合是否具有所需的语义?要回答这些问题就需要一种理论,它既支持构件关系的形式化定义(有利于利用计算机辅助工具对构件关系进行处理),又支持构件关系的图形化表示(有利于构件关系的直观理解和选择).范畴论正是这样一种数学理论.范畴论又被称为箭头理论,它主要应用于对象关系及其组合的研究.

范畴论是一种抽象的数学理论,用于描述各种数学系统的外部结构,并被用于描述软件规范之间的关系和软件结构[8,14].范畴论的一个重要特点是:它侧重于对象之间的关系,而不是对象自身的描述.态射(morphism)对构件模型来说十分重要,因为它决定了构件之间的关系,允许构件的组合,并从它们的配置中推断模型的特性.此外,范畴论还支持图形化表示(对象为结点,态射为有向边),有利于模型的理解.

本文利用范畴论定义构件之间的关系.在一个范畴内,对象之间的关系组合是有定义的,只要各个开发阶段所产生的工作产品符合范畴的定义,态射合成就可以用来跟踪不同抽象层次的构件之间的关系.下面首先介绍本文所涉及的范畴论概念.

2.1 基础知识

定义 **2.1.1**(范畴:**Category**). 一个范畴由以下 3 部分组成:

- 多个对象(objects)组成的集合;
- 多个态射(morphisms)组成的集合;
- 态射合成法则:对于态射 $f:A\to B,g:B\to C$,如果 $cod(f)=dom(g)$,那么 $g\circ f:A\to C$ 就有定义.

范畴论中的定义和证明大多是利用图(diagram)来进行的,通过图的可视化特点使一些复杂问题变得直观、易于理解,下面给出范畴图的定义.

定义 **2.1.2**(范畴图). 一个范畴图就是一个元组 $G=(V,E,L,l,s,t)$.其中:V,E 分别是节点(对象)和边(态射)集合;L 是对象和态射的标识集合;l 是映射函数,$l:(V\cup E)\to L$,返回给定对象或态射的标识符;$s,t:E\to V$ 返回给定态射的源或目标对象.

定义 **2.1.3**(共限:**Colimit**). 对包含有多个对象 A_i 和态射 a_i 的图来说,该图的共限(colimit)就是对象 L 及一组态射 l_i,满足:对每一组态射 $l_i:A_i\to L$, $l_j:A_j\to L$ 和 $a_x:A_i\to A_j$, $l_j\circ a_x=l_i$ 成立.图 1 给出了共限的图形化表示.

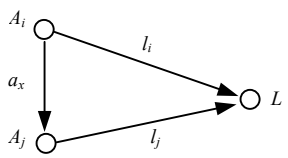

Fig.1 Colimit

图 1 共限

Colimit 为描述体系结构中的构件组合提供了理论基础.

定义 **2.1.4**(交换图:**Commutative diagram**). 在一个含有对象与态射的图中,若从图中任一对象出发到另一对象有两条或更多的(由同向箭头连接的)路径相通,则沿着这些路径(沿箭头方向)的态射合成都相等,这样的图就称为交换图.

交换图可用于描述体系结构中的构件关系的特征以及用于验证构件开发过程中构件关系的正确性.

2.2 构件模型的范畴论语义

以体系结构为中心的软件开发过程包括一系列体系结构的设计步骤:目标分解、体系结构设计和验证[12].

目标分解:该活动将系统目标或需求分解为一系列子目标并指派给构件,系统目标与构件之间形成“责任-指派”关系,本文称其为γ-关系.

体系结构设计:该活动的主要工作是确定构件之间的协同规则.构件与体系结构之间形成“部分-整体”关系,本文称其为β-关系.

验证:该活动的主要工作是确认所设计的体系结构是否满足系统目标,系统目标与体系结构之间形成“目标-满足”关系,本文称其为λ-关系.

上述 3 种重要的构件关系直接影响对构件模型的理解和重用,需要进行严格的定义和验证.本文中,一个体系结构的设计被记为 $R_0\rightarrow_{\lambda}Z\{R_i|i\leq n\}$,其中,$R_0$,$R_i$ 和 Z 分别代表“系统需求”、“构件”和“体系结构”,“系统需求”来自于更高抽象层次的体系结构.

定义 **2.2.1**(体系结构). 一个体系结构是一个元组(R_{Set},L).其中:R_{Set} 是构件集合;L 是构件关系集合(包括γ-关系、β-关系和λ-关系.其中,λ-关系用于说明构件配置的合法性).

由于范畴论关注的是对象之间的关系而不是描述对象的逻辑,而且本文主要讨论构件之间的关系定义,所以我们在本文中不详细讨论构件的描述逻辑,只涉及不同构件之间的标识映射.下面首先给出与构件规范有关的定义.

定义 **2.2.2**(构件标识:**Component signature**). 一个构件标识是一个四元组$\theta=(\Sigma,A,\Gamma,D)$,其中:

Σ是一个集合,包含构件规范中的所有标识符;

A 是构件的属性集合;

Γ是构件的接口集合;

$D:\Gamma\rightarrow 2^A$,对于 $r\in\Gamma$, $D(r)$是 r 所影响的属性集合.

定义 **2.2.3**(构件标识态射:**Component signature morphism**). 设$\theta_1=(\Sigma_1,A_1,\Gamma_1,D_1)$和$\theta_2=(\Sigma_2,A_2,\Gamma_2,D_2)$是两个构件标识,构件标识态射$\delta:\theta_1\rightarrow\theta_2$就是一个标识符映射:$\delta_{\Sigma}:\Sigma_1\rightarrow\Sigma_2$,并满足下列条件:

$\delta_{\Sigma}(\Sigma_1)\subseteq\Sigma_2$;

对每一个 $a\in A_1$,$sort(\delta_{\Sigma}(a))=sort(a)$,$sort$ 确定属性的类型;

对于每一个 $r\in\Gamma_1$ 和 $p\in P$, P 是 r 的参数集合,$sort(\delta_{\Sigma}(p))=sort(p)$;

对于每一个 $r\in\Gamma_1$,$\delta_{\Sigma}(D_1(r))=D_2(\delta_{\Sigma}(r))$.

上述条件要求构件标识的映射不能改变构件中属性的类型和接口的参数类型,而且每个接口所影响的属性集合保持不变.

命题 **2.2.1**. 设$\theta_1=(\Sigma_1,A_1,\Gamma_1,D_1)$,$\theta_2=(\Sigma_2,A_2,\Gamma_2,D_2)$和$\theta_3=(\Sigma_3,A_3,\Gamma_3,D_3)$为构件标识,如果存在态射$\delta_1:\theta_1\rightarrow\theta_2$ 和$\delta_2:\theta_2\rightarrow\theta_3$,则存在态射$\delta_2\circ\delta_1:\theta_1\rightarrow\theta_3$.

根据定义 2.2.3,命题 2.2.1 显然成立.

命题 2.2.1 用于跟踪不同构件标识之间的映射关系.

定义 **2.2.4**(构件描述).构件描述是一个三元组$\mathcal{R}=(\theta,\Phi,C)$.其中:$\theta$是构件标识;$\Phi$是功能和非功能目标集合;$C$包含实现构件目标$\Phi$所需要的条件及应满足的约束(如应该服从的全局目标,相互协同的子构件及协同规则,行为约束等等).

构件描述中目标分为功能和非功能目标,功能目标描述构件的能力/职责,即说明构件能做什么(包括其输入、输出、前置条件和后置条件等);非功能目标描述性能、时态等方面的要求.

构件功能不仅涉及输入输出变换,还对其环境产生影响(如构件状态).本文利用状态转移过程解释构件功能.

定义 **2.2.5**(θ-解释). 对于一个构件描述$\mathcal{R}=(\theta,\Phi,C)$,$\theta$-解释是一个二元组 $I=(M,V)$,其中:

M 是一个状态转移系统$(W,w_o,\rightarrow,\Gamma)$:$w_o\in W$,$\rightarrow\subseteq W\times W$; Γ是构件的接口集合;

$V:F\to W\to S$ 是取值函数:S 是给定表达式的类型;F 是基于 θ 的表达式;$V(\varphi)(w)$返回表达式 φ 在状态 w 的取值.状态是由构件属性的取值所构成的,状态转移是根据开发人员对构件接口功能的解释产生的.尽管可以利用接口描述构件的功能,但当不同的开发人员实施或重用该构件时,仅仅通过接口(名字、参数)很难准确地理解该构件如何发挥作用,包括构件功能对构件状态的影响、输入输出的变换关系以及与其他构件如何交互等等.此外,要测试所设计的体系结构,也需要提供构件接口的可操作性解释.定义功能需求并作出详细的解释是软件设计人员的责任,传统的开发方法只强调了前者而忽略了后者,导致测试设计变得非常困难.

对于业务环境来说,构件的 θ-解释描述了业务过程的一系列具体实例;对于需求模型来说,构件的 θ-解释描述了用户与软件系统交互过程的一系列具体情景.

定义 **2.2.6**(构件规范).构件规范是一个二元组 $R=(\mathcal{R},I)$,$R=(\theta,\Phi,C)$,I 是 θ-解释,对于每一个目标 $g(g\in\Phi)$,都有 $I\models g$.

$\models$ 的定义是:设 $\varphi:a(I,O)\psi$ 为一个构件接口,φ 和 ψ 分别为 $a(I,O)$ 的前置条件和后置条件,我们说 $M=(W,w_o,\to,\Gamma)$ 满足 $\varphi:a(I,O)\psi$,记为 $I\models\varphi:a(I,O)\psi$,如果存在路径集合 $\{p_k|k\in N\}$,且对于 $p_k:w_1w_2...w_n,\langle w_i,w_j\rangle\in\to,i,j\le n,V(\varphi)(w_1)$ 为真,$a(x,y)$执行后,构件状态最终变为 w_n,且 $V(\psi)(w_n)$为真.由于受篇幅限制,其他逻辑表达式(如 temporal logic)在本文中不作讨论.

如果 R 中的功能目标没有相应的实施策略,那么 R 称为抽象规范/高层规范,否则称其为具体规范(相对于某一抽象规范)或低层规范.

定义 **2.2.7**(δ-逆射). 设 θ 和 θ_1 是两个构件标识,并存在态射 $\delta:\theta\to\theta_1$.对于 θ_1-解释 I_1,其沿着 δ 的逆射就是从 I_1 中获取关于 θ 的解释,且基于 θ 的表达式 f 在其上的取值与 $\delta(f)$ 在 I_1 上的取值相同,记为 $I_1\restriction_\delta\delta$-逆射的核心是从 I_1 的状态转移系统中获取属于另一个构件的状态转移系统.

δ-逆射的定义是为了描述不同构件规范的 θ-解释之间的关系.

基于上述定义,下面给出构件关系的定义.

定义 **2.2.8**(γ-关系:"责任-指派"关系). 设 $R_1=(\mathcal{R}_1,I_1)$和 $R_2=(\mathcal{R}_2,I_2)$是两个构件规范,R_1 是给定的构件规范(包含待分解的全局目标),R_2 是在体系结构设计过程中产生的构件(包含子目标).其中:$\mathcal{R}_1=(\theta_1,\Phi_1,C_1)$,$\mathcal{R}_2=(\theta_2,\Phi_2,C_2)$,$\mathcal{R}_1$ 包含全局目标.$\gamma:R_1\to R_2$ 是一个构件标识态射 $\delta:\theta_1\to\theta_2$,并且满足:$I_1\models\Phi_1\Rightarrow I_2\models\Phi_2$, $I_2\not\models\Phi_2\Rightarrow I_1\not\models\Phi_1$.

基于体系结构的构件开发过程是以功能目标的分解为基础的,将功能划分为一系列步骤,并将其作为子目标指派给多个构件,被分解的目标称为全局目标.在划分子目标时,目标之间不能产生矛盾,如一个目标的实现不能导致其他目标的失败(例如使其他目标的前置或后置条件不成立).将子目标指派给构件后,要确定这些构件的协同规则,利用给出的全局目标的解释,验证协同规则的应用是否满足全局目标的后置条件.

定义 2.2.8 的条件说明子目标不能与全局目标相矛盾,即如果全局目标实现则意味着子目标实现,但如果子目标没有实现,则全局目标就无法实现.γ-关系回答"构件为什么存在"的问题,R_1 是 R_2 要服从的全局目标,是 R_2 存在的原因.

定义 **2.2.9**(β-关系:"部分-整体"关系). 给定两个构件规范 $R_1=(\mathcal{R}_1,I_1)$和 $R_2=(\mathcal{R}_2,I_2)$,$\mathcal{R}_1=(\theta_1,\Phi_1,C_1)$,$\mathcal{R}_2=(\theta_2,\Phi_2,C_2)$,$I_1$ 和 I_2 分别是 θ_1 和 θ_2 的 θ-解释.$\beta:R_1\to R_2$ 是一个构件标识态射 $\delta:\theta_1\to\theta_2$ 且满足下面的条件:

条件 1: $I_2\models_{\theta_2}\delta(\varphi),\varphi\in\Phi_1$,当且仅当 $I_1\models_{\theta_1}\varphi$;

条件 2:对于 $g_2\in\Phi_2,g_2$ 是 $\mathcal{R}_2$ 的一个目标,如果 $g_1\in\Phi_1,g_1$ 是 g_2 的一个子目标,那么就有 $I_2\models g_2\Rightarrow I_1\models g_1,I_1\not\models g_1\Rightarrow I_2\not\models g_2$.

条件 1 要求 R_1 中参与实现 R_2 全局目标的功能解释必须经过验证与功能描述相一致,而且在 R_2 中仍能保持该一致性.条件 2 说明了"部分-整体"关系不是简单的组合关系,R_1 作为 R_2 的组成部分,它需要完成部分子目标,且不能与全局目标产生矛盾.

β-关系有助于理解"一个构件如何与其他构件相互协同"的问题,R_2 给出了 R_1 的上下文,其中的协调规则描述了 R_1 与其他构件的协作关系.β-关系的定义有利于产品的组装和验证.

定义 **2.2.10**(λ-关系:"目标-满足"关系). 设 $R_0=(\mathcal{R}_0,I_0)$为高层构件规范,$R=(\mathcal{R},I)$是 R_0 的低层构件规

范,$\mathcal{R}_0=(\theta_0,\Phi_0,C_0)$,$\mathcal{R}=(\theta,\Phi,C)$.$\lambda:R_0\to R$ 是一个构件标识态射$\delta:\theta_0\to\theta$,对于需求$\varphi_0\in\Phi_0$,$I\vDash\delta(\varphi_0)$,且 $I_0=I\restriction_\delta$.

定义 2.2.10 中的高层构件规范是指包含待分解的全局目标的构件规范.低层构件规范是相对高层构件规范的抽象程度而言的,它包含在体系结构设计中形成的子构件以及协调规则.定义中的条件要求对于高层规范中的所有目标(包括功能目标和非功能目标)都应被满足.λ-关系反映了高层目标与实施策略的关系,R_0 是较高抽象层次的构件视图,可以看作是一个“黑盒”,它只说明了“构件是什么”和“构件能做什么”.R 则描述了目标的实施策略,可以看作是一个“白盒”,是较低抽象层次的构件视图.

定义 **2.2.11**(体系结构设计模式). 设 $R_0=(\mathcal{R}_0,I_0)$为系统目标或需求,$\{R_i=(\mathcal{R}_i,I_i)|i\in N\}$是为实现 R_0的全局目标而指派的构件集合,$R=(\mathcal{R},I)$为 R_0 的设计规范.如果存在下列构件关系:$\{\gamma_i:R_0\to R_i|i\in N\}$,$\{\beta_i:R_i\to R|i\in N\}$和$\lambda:R_0\to R$,那么,$R_0$,$\{R_i\}$,$R$ 以及上述构件关系构成一个体系结构设计模式,记为 $R_0\to_\lambda R\{R_i|i\in N\}$.图 2 给出了体系结构设计模式的范畴图.

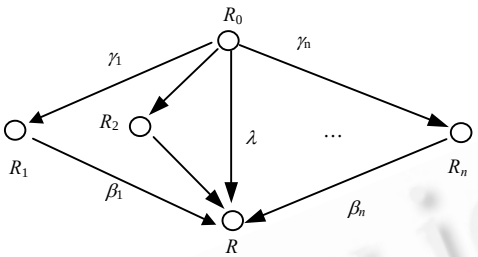

Fig.2 Architecture design pattern

图 2 体系结构设计模式

命题 2.2.2 对于构件规范 R_1,R_2 和 R_3,如果存在$\gamma_1:R_1\to R_2$,$\gamma_2:R_2\to R_3$,那么存在$\gamma:R_1\to R_3$.对于构件规范 R_1,R_2 和 R_3,如果存在$\gamma:R_1\to R_2$,$\beta:R_2\to R_3$,且 R_1 的目标在 R_3 中有相应的实施策略(协调规则),那么存在$\lambda:R_1\to R_3$;否则,如果 R_3 的目标只是 R_1 的子目标,则存在$\gamma:R_1\to R_3$.对于构件规范 R_1,R_2 和 R_3,如果存在$\gamma:R_1\to R_2$,$\lambda:R_2\to R_3$,那么存在$\gamma:R_1\to R_3$.

根据命题 2.2.1、定义 2.8、定义 2.9 和定义 2.10,该命题显然成立.

以体系结构为中心的软件开发过程包括 4 个开发阶段:业务建模、需求分析、构件设计和构件实现,分别产生 4 种工作产品:业务模式、需求模式、设计模式和实施模式,这些模式构成一个可跟踪、可理解的构件模型[13].

定义 **2.2.12**(构件模型). 构件模型是一个包含不同抽象层次体系结构设计模式的集合$\{R_0^j\to_\lambda R^j\{R_i|i\in N\}|j\le n\}$,其中 n 为体系结构设计模式的数量.

层次化的软件构件模型有助于软件构件的跟踪,可以有效地支持自顶向下的构件重用.

命题 **2.2.3**. 构件模型就是一个由构件规范(作为对象)及其关系(作为态射)构成的范畴.

根据命题 2.2.2,构件模型中构件关系的组合是有定义的,所以命题成立.

命题 2.2.3 说明:只要在体系结构的设计过程中形成的构件之间的关系成立,不同抽象层次构件之间的关系就有定义,具有可跟踪性和可理解性.图 3 给出了一个由构件模型构成的范畴的例子.从图中可以看出,软件构件 R'_{11} 是构件规范 R_{11} 的一个实现,R_{11} 包含了 R_1(需求)的子目标. R'_1 由 R_{11} 和 R_{12} 构成,并满足 R_1(需求).范畴图中的γ关系有助于回答“构件为什么存在?”的问题,λ关系和β关系有助于回答“为了实现系统目标,哪些构件需要相互协同?”的问题,这都是理解构件模型所必需的.

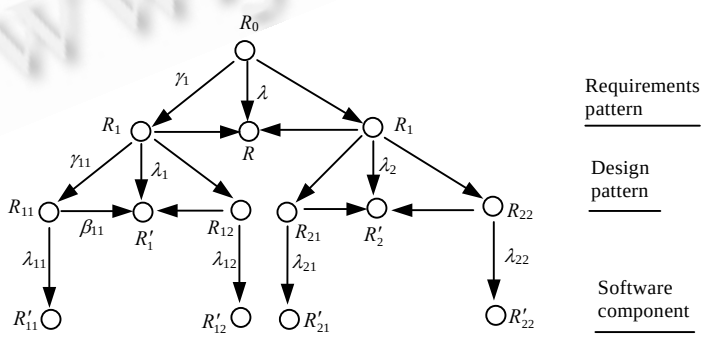

Fig.3 An example of category

图 3 一个范畴的例子

从语义上看,一个体系结构设计模式 $R_0\rightarrow_{\lambda}R\{R_i|i\in N\}$就是由构件 $R_0,R,\{R_i|i\in N\}$和构件关系$\lambda,\{\gamma_i\},\{\beta_i\}$构成的 colimit,对于$\lambda:R_0\rightarrow R,\beta_i:R_i\rightarrow R$ 和$\gamma_i:R_0\rightarrow R_i$而言,$\beta_i\circ\gamma_i=\lambda$成立,这是构件组合应满足的条件(范畴图的可交换性).

定义 **2.2.13**(构件模型实例). 设 F 为一个由构件及其关系构成的范畴,G 是构件模型构成的范畴.如果存在态射:$M:G\rightarrow F$,那么 F 被称为 G 的实例范畴.

图 4 是一个构件模型实例.R_0,R_1,R_2,R 以及它们之间的态射构成软件构件模型的范畴, R'_0 , R'_1 , R'_2 ,R'以及它们之间的态射构成实例范畴.从图 4 中可以看出,以体系结构为中心的软件构件模型不仅支持构件的重用还支持构件关系的重用.

通过由构件关系构成的需求模式,设计模式和实施模式可以将业务目标映射到具体的软件构件,从而支持自顶向下的构件重用.通过提高构件的可跟踪性,不仅有助于构件的维护,还有利于构件的重用.图 5 是一个简单的例子.

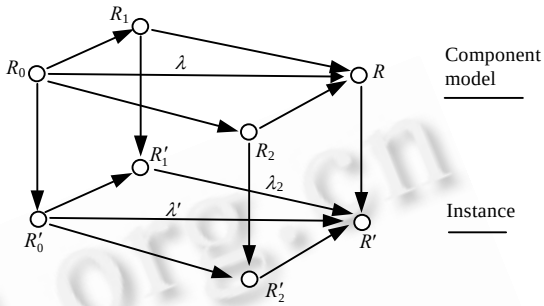

Fig.4 Instance of component model

图 4 构件模型实例

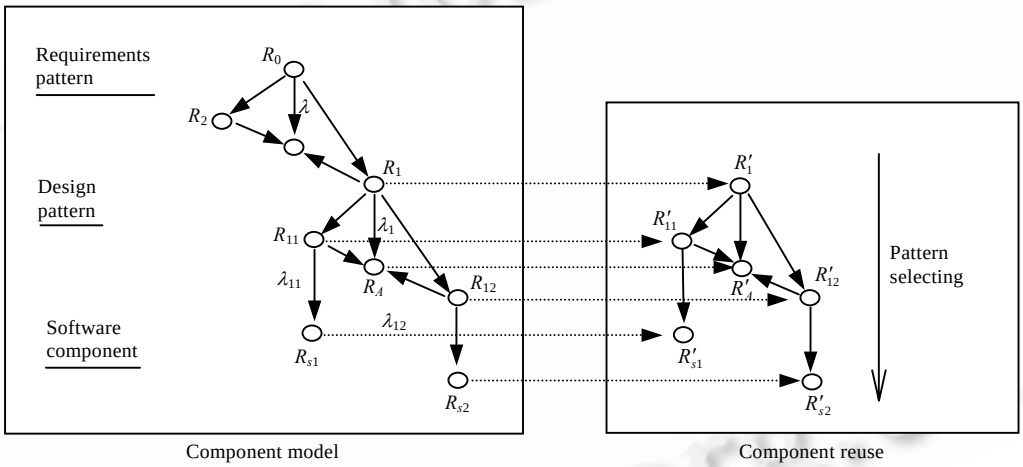

Fig.5 Component reuse by pattern selecting

图 5 通过模式选择支持构件重用

在图5中,如果一个产品的需求 R'_1 与构件模型中的需求 R_1 相同,那么,可以选择 $R_1\rightarrow_{\lambda_1}R_A\{R_{11},R_{12}\}$作为产品的设计模式,再通过$\lambda_{11}$和$\lambda_{12}$选择构件 R_{S1}和 R_{S2}.

利用设计模式自顶向下地选择软件构件比直接在构件库中选择更有效.这是因为软件构件属于解空间,对于同样的需求,不同的人会有不同的解决方案.在同一领域内,各个业务环境具有相同的目标、相同或相近的业务规则和相似的资源分配方式,具有相同或相似的软件需求.在同一领域内,各个业务环境的软件需求比实现这些需求的软件构件更有可能相同或相似,需求重用可以导致设计规范、软件构件、测试用例的重用,这种自顶向下的重用策略可以促进软件构件的重用[15].

3 实例研究

我们以面向新闻出版领域的软件开发作为例子,讨论构件模型的可跟踪性问题.受篇幅所限,本文只讨论需求的可跟踪性.“目标满足”问题一直是传统需求工程方法的主要缺陷之一,提供软件需求和业务目标的可跟踪性是需求工程需要解决的重要问题[16].本文利用需求模式明确地描述了需求与业务目标之间的关系,可以较好地解决这一问题.在利用模式开发具体产品时,开发者重用业务模式构建具体的业务环境,通过“业务目标”和

“需求模式”可以检索相应的“软件需求”,并将其组装为具体的产品需求.此外,业务模式为软件需求的描述提供了上下文.下面给出一个简化的需求模型(如图 6 所示).

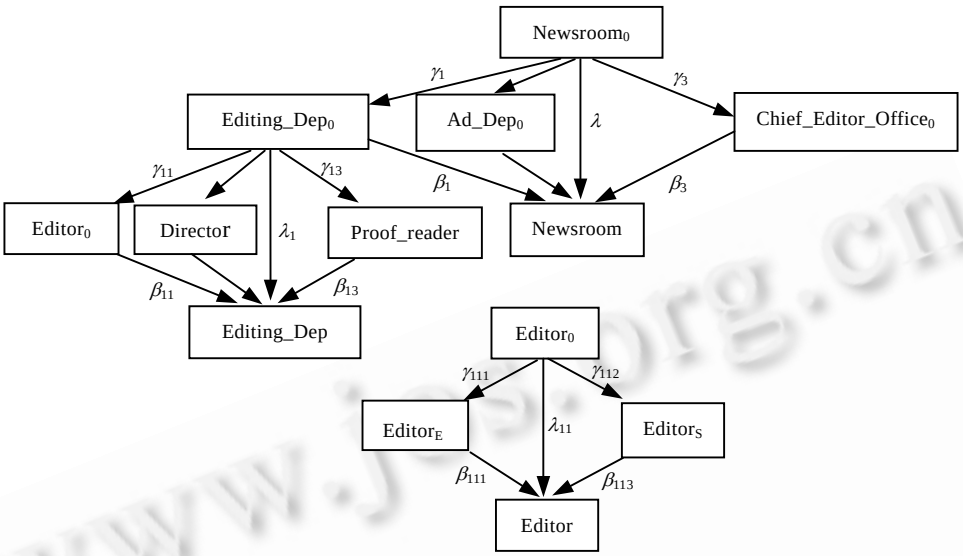

Fig.6 Requirements model (excerpt)

图 6 需求模型(部分)

模型中的构件包括报社(Newsroom)、编辑部(Editing_Dep)、广告部(Ad_Dep)、总编室(Chief_Editor_Office)、编辑(Editor)、部主任(Director)、校对(Proof_reader)等等.定义中的“0”下标表示构件是高层次抽象.

$Newsroom_0$ 是最高抽象层次的实体,它只包含自身的功能目标以及前提条件和对排版结果的约束,反映了 Newsroom 作为一个整体在更高层次的环境中的视图.

$Newsroom_0$ 的全局目标是由 $Editing\_Dep_0$,Ad_Dep_0,$Chief\_Editor\_Office_0$ 共同协作完成的,即 $Newsroom_0$ 是它们的共同目标.$Newsroom_0$,$Editing\_Dep_0$,Ad_Dep_0,$Chief\_Editor\_Office_0$,Newsroom 以及上面的关系就构成了一个高层次的组织模式.

受篇幅限制,下面只简单给出 Editing_Dep 所涉及的构件描述,其中的描述逻辑不在此讨论.

$Editor_0$ {**Globalgoal** $Editing\_Dep_0$

Localgoal (SelectedArticles≠∅):Typesetting(*O*=News_with_types) //新闻稿排版

}

Director {**Globalgoal** $Editing\_Dep_0$

Localgoal (Articles≠∅): Articlesselecting() (SelectedArticles≠∅) //选择稿件

(Is_Signed(Newsfile)=*F*∧Is_Proofed(Newsfile)=*T*):

Newssectionsigning(*I*=Newsfile,*O*=News_section)(Is_Signed(News_section)=*T*) //签发版面

}

Proof_reader {**Globalgoal** $Editing\_Dep_0$

Localgoal (Is_Proofed(News_with_types)=*F*):

Proofing(*I*=News_with_types,*O*=Newsfile) (Is_Proofed(Newsfile)=*T*) //校对版面

}

$Editor_0$,Director,Proof_reader 需要相互协作以共同完成 $Editing\_Dep_0$ 的业务目标,下面给出 Editing_Dep 的定义.

Editing_Dep {**Attributes** SelectedArticles:set //被选新闻稿

Globalgoal $Newsroom_0$

Localgoal //新闻稿件组版

(SelectedArticles≠∅):Newsediting(*O*=News_section)(Is_Signed(News_section)=*T*)

Components $Editor_0$,Director,Proof_reader

Coordination-rules Newsediting(*O*=News_section)

{Director.Articlesselecting(); //部主任选择稿件

$Editor_0$.Typesetting(*O*=News_with_types); //编辑人员排版

Proof_reader.Proofing(*I*=News_with_types,*O*=Newsfile); //校对人员校对版面

Director.Newssectionsigning(*I*=Newsfile,*O*=News_section) //部主任签发

}

}

$Editing\_Dep_0$,$Editor_0$,Director,Proof_reader,Editing_Dep 以及它们之间的关系构成了一个低层次的组织模式.图 6 的上半部分给出了业务环境模型(部分)的图形化表示,它包括两个组织模式:$Newsroom_0 \rightarrow_{\lambda} Newsroom$ {$Editing\_Dep_0$,Ad_Dep_0,$Chief\_Editor\_Office_0$}和 $Editing\_Dep_0 \rightarrow_{\lambda 1} Editing\_Dep$ {$Editor_0$,Director,Proof_reader}.

在层次化的模型中,当前层的实体(如 $Editing\_Dep_0$)定义了“做什么”,上一层的实体(如 $Newsroom_0$)描述了“为什么”,下一层的实体(如 Editing_Dep)则定义了“如何做”.所以,层次化的模型结构有利于对业务环境的理解和跟踪.

构件模型的范畴图可以直观地表示构件之间的关系,UML 的协作图(collaboration diagrams)和顺序图(sequence diagrams)可用于理解构件之间的协作和交互.图 7 给出了 Editing_Dep 中 Newsediting(*O*=News_section)的协作图.

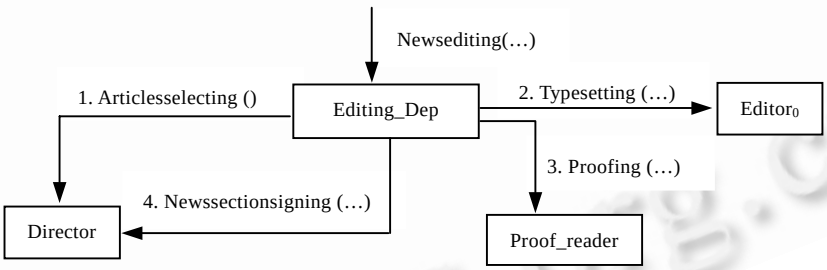

Fig.7 Collaboration diagram of Editing_Dep

图 7 Editing_Dep 中协调规则的协作图

建立业务环境模型的目的有两个:一是使用户和开发者对业务环境有共同的理解;二是为需求规范提供上下文.传统的软件开发方法采用用例(use case)来描述需求模型,用例与业务环境之间、用例之间的关系都没有给出明确的说明.本文利用需求模式解决这一问题,下面以 $Editor_0$ 为例进行需求划分.在需求建模阶段,$Editor_0$ 中的 Typesetting()可以进一步细化为一系列操作步骤,由用户和软件系统分别完成:指派给用户的操作构成环境需求 $Editor_E$;指派给软件系统的操作构成软件需求 $Editor_S$.最后要验证该划分的合理性,并形成一个需求模式 $Editor_0 \rightarrow_{\lambda 11} Editor${$Editor_E$,$Editor_S$}:$Editor_E$ 描述用户所进行的操作;$Editor_S$ 描述软件需求;Editor 是包含用例的构件规范.图 6 的下半部分给出了需求划分模式的图形化表示.下面给出 $Editor_E$,$Editor_S$ 和 Editor 的构件描述.

$Editor_E${**Globalgoal** $Editor_0$

Localgoal

Plandivision(*O*=Division_Border) //确定分区边界

Choosearticle(*O*=Selected_Item) //选择稿件

…

```
        }
Editor_S{Globalgoal Editor_0
        Localgoal
            Displaypagespace() //显示页面空间
            Displayarticlelist()  //显示稿件列表
              …
        }
Editor{Globalgoal Editing_Dep_0
        Localgoal (SelectedArticles≠∅): Typesetting(O=News_with_types)
        Comps Editor_E, Editor_S
        Coordination-rules Typesetting(O=News_with_types)
          {Editor_S.Displaypagespace(); //软件系统显示页面空间
           Editor_E.Plandivision(O=Division_Border); //用户确定分区边界
                    …
          }
        }
```

$Editor_0$ 的描述中给出了 Typesetting()的前提条件,Editing_Dep 描述了 Typesetting()与其他构件的协同关系.在应用工程中,开发人员利用组织模式构建面向具体报社的出版环境模型并验证其正确性,一旦获得认可,模型中 $Editor_0$ 对应的需求模式就可以被重用.通过组织模式 $Editing_Dep_0 \rightarrow_{\lambda 1} Editing_Dep\{Editor_0, Director, Proof_reader\}$和需求划分模式 $Editor_0 \rightarrow_{\lambda 11} Editor\{Editor_E, Editor_S\}$,可以很容易地理解软件需求 $Editor_S$ 与业务目标 $Editor_0$ 的关系.Editing_Dep 中的协调规则 Newsediting(*O*=News_section)描述了不同用户(Editor,Director,Proof_reader)的不同用例(Typesetting,Articlesselecting,Proofing,Newssectionsigning)之间的关系.

Editor 中的 Typesetting(*O*=News_with_types)可以通过顺序图直观地描述 $Editor_E$ 和 $Editor_S$ 之间的交互过程.受篇幅所限,本文不再讨论该图.

构件的形式化描述有利于利用计算机辅助工具进行构件的管理、组装和维护,将构件模型的范畴图与 UML 模型相结合可以促进对构件模型的理解.

由于构件模型中明确地描述了构件之间的关系,若构件发生变化,则 γ,β 和 λ可用于评估变化的影响范围.

4 总 结

本文利用范畴论定义了构件模型的形式化语义,在以体系结构为中心的软件开发过程中所构建的不同抽象层次构件之间的关系以及关系组合具有精确的含义.β-关系有助于实现构件的水平跟踪(同一抽象层次),γ-关系和λ-关系有助于实现构件的垂直跟踪(不同抽象层次).特别是,形式化的体系结构设计模式精确地定义了构件的组合关系和应满足的条件,从而有利于构件的选择和组装.利用范畴论定义软件构件模型的形式化语义可以使模型中的构件关系以及结构特征可视化,有利于模型的理解、维护和重用.

在基于体系结构的构件开发过程中(如 ABD 方法),本文讨论的构件关系是存在的.本文明确地给出了它们的语义,其目的是通过这些严格的定义可以在体系结构设计中实现可测试性以及在开发过程的各个阶段所形成的构件之间保持正确的关系.其次,通过构件关系构成的设计模式支持构件的理解、选择和重用.此外,范畴论主要研究对象之间的关系而不涉及对象自身的描述逻辑,所以具有很好的适用性,本文讨论的方法是可行的.本文讨论的构件模型适合于为支持构件重用而进行的开发,对于一些业务和需求比较成熟和稳定的应用领域,通过建立可跟踪的构件模型可以提高构件的可重用性和可维护性,在进行产品的批量开发时提高效率.但这种方法不适用于面向单一客户的一次性开发,也不适用于业务和需求极不稳定的应用领域.

References:

[1] Rothenberger MA, Dooley KJ, Kulkarni UR, Nada N. Strategies for software reuse: A principal component analysis of reuse practices. IEEE Trans. on Software Engineering, 2003,29(9):825−837.

[2] Baum L, Becker M, Geyer L, Molter G. Using software architecture as a catalyst for reuse. 1998. http://www.agss.informatik,uni-kl.de/mitarbeiter//baum/publications/papers

[3] Han J. An approach to software component specification. In: Proc. of the Int'l Workshop on Component-Based Software Engineering. 1999. 97−102. http://www.it.swin.edu.au/personal/jhan/jhanPapers/cbse99.pdf

[4] Knethen AV, Peach B. A survey on tracing approaches in practice and research. Technical Report, No.095.01/E (Version 1.0), 2002.

[5] Stout GA. Requirements traceability and the effect on the system development lifecycle (SDLC). Technical Report, DISS 725, Spring Cluster, 2001.

[6] Clements PC, Northrop LM. Software architecture: An executive overview. Technical Report, CMU/SEI-96-TR-003, Carnegie Mellon University, 1996.

[7] Baum L, Becker M, Geyer L, Molter G. A process view on architecture-based software development. 1999. http://www.ece.utexas.edu/~perry/prof/wicsal/final

[8] Guo J. Using category theory to model software component dependencies. In: Proc. of the 9th Annual IEEE Int'l Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS 2002). IEEE Computer Society, 2002. 185−192.

[9] Mei H, Chen F, Feng YD, Yang J. ABC: An architecture based, component oriented approach to software development. Journal of Software, 2003,14(4):721−732 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/721.htm

[10] Liu DY, Mei H. Mapping requirements to software architecture by feature-orientation. In: Proc. of the 2nd Int'l Software Requirements to Architectures Workshop. 2003. 69−76. http://se.uwaterloo.ca/~straw03/finals/LiuMei.pdf

[11] Bachmann F, Bass L, Chastek G, Donohoe P, Peruzzi F. The architecture based design method. Technical Report, CMU/SEI-2000-TR-001, Carnegie Mellon University, 2000.

[12] Chu W, Qian DP, Liu CD. Architecture-Centric software process for software reuse. In: He JH, Zeyu GJ, Xiao CB, eds. Proc. of the 8th Int'l Conf. for Young Computer Scientists. Beijing: Int'l Academic Publishers/Beijing Word Publishing Corp., 2005. 217−222.

[13] Chu W, Qian DP. Pattern oriented software development for software reuse. Journal of Nanjing University (Natural Sciences), 2005, 41(Computer Issue):743−748 (in Chinese with English abstract).

[14] Fiadeiro JL, Maibaum T. A mathematical toolbox for the software architect. 1996. http://www.fiadeiro.org/jose/papers

[15] Roudiès O, Fredj M. A reuse based approach for requirements engineering. In: Proc. of the IEEE Int'l Conf. on Computer Systems and Applications. Los Alamitos: IEEE Computer Society, 2001. 448−450.

[16] Svetinovic D. Architecture-Level requirements specification. In: Proc. of the 2nd Int'l Software Requirements to Architectures Workshop. 2003. 14−19. http://se.uwaterloo.ca/~straw03/finals/Svetinovic.pdf

附中文参考文献:

[9] 梅宏,陈锋,冯耀东,杨杰.ABC:基于体系结构,面向构件的软件开发方法.软件学报,2003,14(4):721−732. http://www.jos.org.cn/1000-9825/14/721.htm

[13] 楚旺,钱德沛.支持软件重用的面向模式的软件开发方法.南京大学学报(自然科学),2005,41(计算机专辑):743−748.

楚旺(1966－),男,山东济南人,博士生,主要研究领域为软件工程,形式化方法.

钱德沛(1952－),男,教授,博士生导师,CCF高级会员,主要研究领域为高性能计算机,网格技术,主动网络,计算机网络管理,性能测量.