基于扩展的构件依赖关系图聚类的体系结构重构策略研究 *

钟林辉[1a], 姚昕凡[2], 徐 静[1b], 李 晖[3]

(1. 江西师范大学 a. 计算机信息工程学院; b. 文学院, 南昌 330022; 2. 江西省广播电视学校, 南昌 330029; 3. 南昌航空大学 国际交流处, 南昌 330063)

摘 要: 通过在软件体系结构层次实施软件重构,能够改善软件的质量、提高软件的易演化性。提出了扩展的构件依赖关系图的概念,将软件体系结构的逻辑依赖关系以及演化历史中蕴涵的演化依赖关系纳入到统一的表示中。进一步提出了基于扩展的构件依赖关系图聚类的体系结构重构策略,从而达到改善体系结构质量的目的。

关键词: 软件演化; 演化信息; 软件体系结构; 软件重构

中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2010)08-2987-04

doi:10.3969/j.issn.1001-3695.2010.08.046

Research on strategy of software architecture refactoring based on clustering of component dependency relation

ZHONG Lin-hui[1a], YAO Xin-fan[2], XU Jing[1b], LI Hui[3]

(1. *a. School of Computer Information & Engineering, b. School of Chinese Language & Literature, Jiangxi Normal University, Nanchang* 330022, *China*; 2. *School of Broadcast & Television, Nanchang* 330029, *China*; 3. *Dept. of International Cooperation & Exchange, Nanchang Hangkong University, Nanchang* 330063, *China*)

Abstract: For improving the evolvability of software architecture, this paper proposed a software architecture refactoring strategy based on clustering of component dependency relation, which consisted of logical relation and evolution relation among components. Moreover, showed an example for explaining its usability.

Key words: software evolution; evolution information; software architecture; software refactoring

0 引言

软件重构是一种在保证系统外部行为属性不发生变化的前提下,通过改善系统的结构达到改善软件质量(如易演化性)的方法。在软件体系结构层次实施软件重构,能够改善软件的质量、提高软件的易演化性。目前,在软件体系结构(包括产品线软件体系结构)层次进行重构的研究不是很多。软件体系结构层次的重构涉及到构件、构件间连接关系的修改(如增加新的连接件或者复合构件)等。例如,Philipps 等人提出了一种能够直接在用图表示的软件体系结构上运用重构规则的方法[1],这些重构规则能够保证构件之间的行为属性不发生变化。Garcia 等人提出了体系结构不良代码的概念,以及相应的体系结构层次的重构操作[2]。另一方面,产品线体系结构代表的是领域内一组具有公共属性的可复用资产,产品线体系结构的不断演化要求重构操作能够支持变化性构件与公共构件之间的相互转变[3]以及对变化性的重构[4]。特别地,Bourquin 等人研究比较了各种重构技术及其工具在实施体系结构重构后,对软件质量的影响[5]。

本文在支持演化信息的软件构件模型[6]的基础上,将体系结构看做是复合构件成员构件间的拓扑结构,并以此为基础研究一种构件化软件体系结构重构策略,将体系结构的逻辑依赖关系和演化依赖关系统一表示,通过聚类的方法优化体系结构,从而达到提高系统易演化性的目的。

1 扩展的构件依赖关系图

构件依赖关系图用来描述软件体系结构中构件之间的依赖关系。其中,将软件体系结构中的构件抽象成为一个有向图中的节点,节点之间的边表示构件间存在依赖关系,边上的权值表示节点之间的依赖密切程度。通常,构件依赖关系图中的连接关系是逻辑依赖关系。本文除考虑逻辑依赖关系外,还考虑演化依赖关系,并将包含这两者依赖关系的构件依赖关系图称为扩展的构件依赖关系图。

定义 1 扩展的构件依赖关系图 ECDG 是一个二元组$\langle C, E\rangle$。其中,C 表示一个系统中所有的构件集合;$E \subset C \times C$,表示构件之间存在的关系,包括逻辑和演化两种依赖关系;对任意的边 e, $e \in E$, $W(e)$ 表示边 e 的权重,$W: C \times C \rightarrow N+$。

a)逻辑依赖关系包括构件之间的接口组装关系和复合构件内部的接口委派关系;

b)演化依赖关系指的是构件在演化历史中蕴涵的依赖关

收稿日期: 2010-01-03; **修回日期**: 2010-04-22 **基金项目**: 国家自然科学基金资助项目(60773054);江西省自然科学基金资助项目(2009GQS0053,2008GQS0047);江西省教育厅科技项目(GJJ09140,GJJ09457);江西师范大学 2007 博士启动基金资助项目

作者介绍:钟林辉,副教授,博士,主要研究方向为构件化软件、软件演化(shiningto@yahoo.com.cn).

系。其中演化依赖关系定义如下：

定义 2　构件演化依赖关系 CED。将软件系统中所有的构件的集合记做 C。若其中构件 $c_1, c_2, \cdots, c_k$ 在的系统整体演化的若干个版本(记做 V)中同时发生了变化,则称这些构件具有演化依赖关系。若将软件体系结构在演化过程中的所有版本记做集合 V,那么定义这些构件之间的演化依赖权重为 $|V|$(即集合 V 的阶)。

定义 3　构件演化依赖关系图 CEDG = $\langle C, E\rangle$ 是构件演化依赖关系 CED 对应的一个带权的完全有向图。其中,C = CED. ComSet, $\forall (c_1, c_2 : c_1 \in C, c_2 \in C : (c_1, c_2) \in E \wedge (c_2, c_1) \in E)$,边上的权重 $W: V \times V \rightarrow$CED. value。即对于 CED. ComSet 中的任意两个构件 c_1 和 c_2,在构件依赖关系图 CEDG 中都存在边$\langle c_1, c_2\rangle$和$\langle c_2, c_1\rangle$。

可以将演化依赖关系分为单变化的演化依赖关系和多变化的演化依赖关系。多变化的演化依赖关系是在单变化的演化依赖关系基础上,通过数据挖掘技术得到的。下面首先定义单变化依赖关系,然后介绍如何用概念格计算多变化依赖关系。

1)单变化的演化依赖关系

单变化的演化依赖关系 $R_i = \langle \{C_1, C_2, \cdots, C_k\}, \{V_i\}\rangle$,表示软件体系结构从第 $i-1$ 个版本变化到第 i 个版本时,构件 $C_1, C_2, \cdots, C_k$ 发生了变化。

单变化的演化依赖关系可以通过考察扩充的构件描述语言 xJBCDL[6] 中成员构件的版本号得到,如果系统表示的第 i 个基线的成员构件发生了变化,则成员构件属于单变化依赖关系。计算方法如下所示：

设第 i 个单变化依赖关系记录在集合 R_i 中,初始 $R_i = \varnothing$;软件体系结构的第 i 个版本对应的 xJBCDL 记为 xJBCDL_i。

```
for all 成员构件 C in xJBCDL_i do
取 C 的版本号到变量 t1 中,
查找 C 在 xJBCDL_{i-1}的版本号,并存储到变量 t2 中
if t1 == t2 then
  Ri = Ri + C;// 将成员构件 C 加入到单变化依赖关系中
end if
eND for
```

2)多变化的演化依赖关系

给定一个变化事务序列 $R_i, \cdots, R_j$,全变化事务的演化依赖关系定义为 $R_i, \cdots, R_j = \langle \{C_1, C_2, \cdots, C_k\}, \{V_i, \cdots, V_j\}\rangle$,表示对任意的体系结构版本 V_k, $V_k \in \{V_i, \cdots, V_j\}$, 在体系结构从第 V_{k-1} 版本变化到 V_k 版本时,构件 $C_1, C_2, \cdots, C_k$ 同时发生了变化。

概念格是按照用户自定义属性的方式对对象聚类并按照偏序关系组成图结构,所以要正确使用概念格来计算构件的多变化依赖关系,需要建立相应的形式背景。在这里,将构件对应于概念格中的对象,将软件体系结构版本号对应于概念格的属性;如果在与软件体系结构版本号的单变化依赖关系中存在对应的构件,则构成概念格形式背景的一个值。

图 1 表示一个系统的软件体系结构及其构件的演化。其中白色的方框表示构件的一个版本,如果这个构件发生了演化,则用黑色的方框表示。

构件的形式背景如表 1 所示。

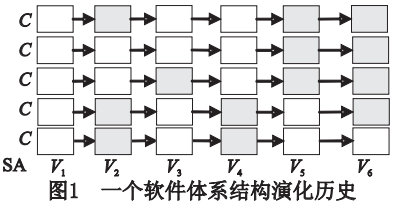

图1　一个软件体系结构演化历史

表 1　体系结构上下文

componen	version					
	V_1	V_2	V_3	V_4	V_5	V_6
C_1		√			√	√
C_2					√	√
C_3			√		√	√
C_4		√		√		√
C_5		√		√		

生成的概念格如图 2 所示。

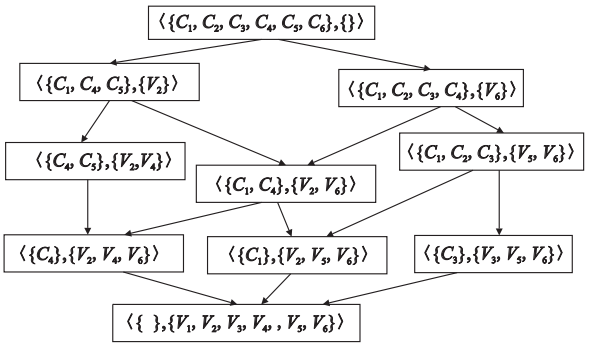

图2　体系结构对应的格

概念格中每个概念表示的是一个构件演化依赖关系。摈弃概念格中那些构件集合个数或者版本集合个数为 1 的概念,剩下的概念即是候选的构件演化依赖关系。基于度量的启发性策略能够辅助用户选择演化依赖强度较高的构件演化依赖关系,一旦构件演化依赖关系确定,就可以将构件演化依赖关系转换成相应的构件演化依赖图,纳入到扩展的构件依赖关系图中。为了从概念格中选择合适的概念,使得概念所代表的构件演化依赖关系具有较强的依赖关系,本文使用启发信息来选择合适的概念。

启发信息计算如下:设构件演化依赖关系表示为$\langle \{C_1, C_2, \cdots, C_m\}, \{V_1, V_2, \cdots, V_n\}\rangle$,所有构件个数为 M,所有版本个数为 N,则启发信息度量值定义为 $\sqrt{(m/M)^2 + (n/N)^2}$。

这个启发信息的度量公式是经验性的,使用这个启发信息选择概念时,会导致所选的概念具有“在较多的版本中包含较多的构件”的特征。启发信息度量值越高表示构件的演化依赖关系越强,这些构件纳入已有的复合构件中或者生成新复合构件的概率也越高。

2　基于扩展的构件依赖关系图聚类的体系结构重构

软件“高内聚,低耦合”的特性与软件的易演化性密切相关。软件体系结构设计中,满足构件内部高内聚、构件之间低耦合的特征,一方面使得构件变化的波动效应能够尽可能地局限在构件内部,避免变化的大范围传播;另一方面,软件开发人员在开发的过程中能够清楚地知道变化涉及的构件集合。

2.1　聚类标准

软件体系结构重构相当于在体系结构层次上各组成成分的重新划分。本文在扩展的构件依赖图上实施图的聚类算法,聚类的结果是产生若干子图,这些子图在内部具有高内聚、子

图之间具有低耦合的特点。由此可以为软件体系结构重构提供指导和依据。

现在已有许多图的聚类算法,如模式驱动的 ACDC 算法[7]、基于爬山算法的最优解算法 NAHC 和 SAHC[8],以及基于遗传算法的最优解算法[9]。本文采用的是文献[8]中提出的最优化标准及其相应的支持工具集(Bunch)作为工作的研究基础。在 Bunch 工具集中用 MQ 表示聚类后子图代表的模块的质量,MQ 是在 -1 ~ 1 的一个值,MQ 的值越大,表示聚类的效果越符合"高内聚、低耦合"的软件工程原则。

$$MQ = \begin{cases} \dfrac{1}{k}\sum_{i=1}^{k} A_i - \dfrac{1}{\frac{k(k-1)}{2}}\sum_{i,j=1}^{k} E_{i,j} & \text{if } k > 1 \\ A_1 & \text{if } k = 1 \end{cases}$$

其中:A_i 表示第 i 个子图的内部关联度,子图 i 的内部关联度由组成子图的 N_i 个节点以及节点之间的边关联度 μ_i 决定;$E_{i,j}$ 表示第 i 个子图以及第 j 个子图之间的关联度,$E_{i,j}$ 由节点个数 N_i、N_j 以及第 i 个子图和第 j 个子图的边关联度 $\varepsilon_{i,j}$ 决定。

计算 A_i 和 $E_{i,j}$ 的公式如下:

$$A_i = \mu_i / N_{2i}$$

$$E_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \dfrac{\varepsilon_{i,j}}{2N_iN_j} & \text{if } i \neq j \end{cases}$$

Bunch 工具的输入是一组三元关系组〈v1,v2,weight〉,表示 v1 和 v2 存在依赖关系,weight 表示 v1 和 v2 依赖关系的权重,weight 可选。输出是一个若干节点构成的子图。

2.2　基于聚类的软件体系结构重构算法

按照"高内聚、低耦合"的度量,软件体系结构中的构件(包括复合构件和其成员构件)会被重新划分。下面给出基于聚类的软件体系结构重构算法[10]。

输入:给定一个软件体系结构的描述 xJBCDL,扩展的构件依赖关系图 ECDG。

输出:重构后的软件体系结构描述 xJBCDL。

begin

通过"高内聚、低耦合"的度量,扩展的构件依赖关系图 xCDG 被划分为若干子图,设子图的集合为 Gset。

a)首先考察 Gset 中的所有子图 G,对于 G 中的所有边 e = (v1, v2),如果 e 是"演化依赖"关系,则

如果 v1 和 v2 是同级的,则调用"同级构件合并操作";

如果 v1 和 v2 分别是两个同级复合构件的成员构件,则调用"同级构件合并操作";

如果 v1 和 v2 不是同级的,则调用"跨级构件合并操作"

b)对任意的两个子图 G1,G2,G1 ∈ Gset,G2 ∈ Gset;

begin

对任意 G1 和 G2 连接的有向边 e = 〈v1, v2〉,v1 ∈ G1,v2 ∈ G2

begin

如果边 e 的类型是组装关系,则对 v2 做提升操作,同时将 v2 标记为"需要做合并操作"的节点

如果边 e 的类型是委派关系,则对 v1,v2 做提升操作

end

end

将所有 Gset 中的子图 G 中被标记为"需要做合并操作"的节点做"同级构件合并操作"

end

通过聚类的方法实现软件体系结构重构,并不能保证每次重构后的软件体系结构在语义上是合乎情理的,需要结合领域知识来判断最终软件体系结构的合理性。在构件依赖图中增加构件演化依赖关系,可以丰富聚类中若干概念之间的关联性。可以将演化依赖关系理解为一种增加聚类准确性的启发信息。

2.3　例子

在上面的例子中,编译器使用的是 LR1 的文法状态机来解析具体的程序语言。在本节中,更改编译器的体系结构使之能够使用不同的文法状态机来编译程序,更改后的体系结构如图 3 所示。其中构件 LL、LR1、LR2 是支持不同文法的三个状态机;构件 SemLR1、SemLR2 和 SemLL 则是相应的语义变换处理规则;构件 TLRTurning 负责配置和调用合适的文法状态机和相应的语义变化处理规则,如构件 LL 必须与 SemLL 配合使用。

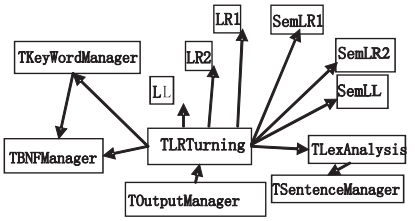

图3　多种文法的编译器体系结构

这个最初的编译器体系结构在经过若干的版本变化后,按照逻辑关系的计算方法会得到如下多变化的演化依赖关系:

〈{LR1, SemLR1},{v1,v2,v3}〉;

〈{LR2,SemLR2},{v1,v2,v3,v4}〉;

〈{LL, SemLL},{v2,v3,v5}〉;

加入演化依赖关系后的编译器体系结构如图 4 所示(短划线表示演化依赖关系)。

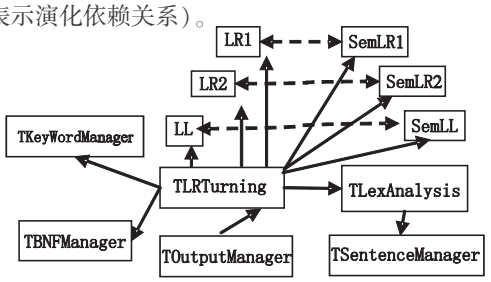

图4　增加了演化依赖关系的多种文法编译器体系结构

经过聚类以后,编译器体系结构会被聚类成如图 5 所示的情况,虚线围起来的构件是那些可能在语义上联系紧密的构件集合。

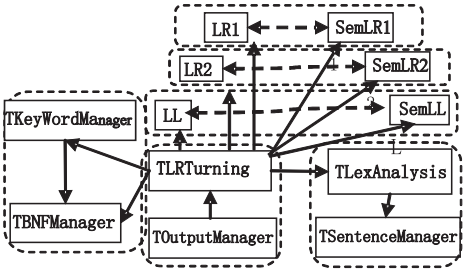

图5　实施扩展构件依赖图聚类的体系结构重构策略后的编译器体系结构

结合领域知识,LR1 和 SemLR1 都是用来处理 LR 文法的,将其重构成为一个新的构件更能体现其含义,新生成的复合构件 TLR1 规约如下所示。类似的 LR2 和 SemLR2、LL 和 SemLL 都将重构成新的复合构件 TLR2Manager 和 TLLManager。

下面是新生成的复合构件 TLR1Manager 的一种可能的 xJBCDL 表示。

```
component TLR1 〈version = 3, location = \\SCM\\TLR1〉
  // 版本没有变化,还是已有的版本
  // 接口和实现体规约省略
end TLR1
component TSemLR1 〈version = 3, location = \\SCM\\TSemLR1〉
  // 版本没有变化,为已有的版本
  //接口和实现体规约省略
ewnd TsemLR1
component TLR1Manager 〈version = 1〉{ //新生成的一个版本
provides:
  getLr1States; getLr1SematicRules;
references:
  TLR1, TSemLR1
instances:
  lr1:TLR1〈version = 3〉;
  semLr1:TSemLR1〈version = 3〉;
end TLR1Manager}
```

3 结束语

构件化软件开发中,对软件体系结构实施重构能改善体系结构的质量,如易演化性。本文研究了在构件化软件开发中利用构件间的演化依赖关系以及构件本身的逻辑依赖关系,通过聚类辅助软件体系结构实施重构的方法,并以编译器为例说明了本方法的有效性。

参考文献:

[1] PHILIPPS J, RUMPE B. Refinement of information flow architectures [C]//Proc of the 1st International Conference on Formal Engineering Methods. 1997:203-212.

[2] GARCIA J, POPESCU D, EDWARDS W, *et al*. Identifying architectural bad smells[C]//Proc of the 13th European Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2009:255-258.

[3] CRITCHLOW M, DODD K, CHOU J, *et al*. Refactoring product line architectures[C]//Proc of the 1st International Workshop on Refactoring: Achievements, Challenges, and Effects. 2003:23-26.

[4] MÁRCIO R. Recommending refactorings when restructuring variabilities in software product lines[C]//Proc of the 2nd Workshop on Refactoring Tools, in conjunction with the Conference on Object Oriented Programming Systems Languages and Applications. 2008.

[5] BOURQUIN F, KELLER R K. High-impact refactoring based on architecture violations[C]//Proc of the 11th European Conference on Software Maintenance and Reengineering. Washington DC: IEEE Computer Society, 2007:149-158.

[6] 钟林辉,谢冰,邵维忠. 扩充 CDL 支持构件演化模型的方法研究[J]. 软件学报,2002,13(增刊):138-142.

[7] TZERPOS V, HOLT R C. ACDC: an algorithm for comprehension driven clustering[C]//Proc of the 7th Working Conference on Reverse Engineering. Washington DC: IEEE Computer Society, 2000: 258-267.

[8] MANCORIDIS S, MITCHELL B, CHEN Y, *et al*. Bunch: a clustering tool for the recovery and maintenance of software system structures [C]//Proc of the 15th International Conference on Software Maintenance. Washington DC: IEEE Computer Society, 1999:50-59.

[9] DOVAL D, MANCORIDIS S, MITCHELL B. Automatic clustering of software systems using a genetic algorithm[C]//Proc of International Conference on Software Technology and Engineering Practice. Washington DC: IEEE Computer Society, 1998:73-91.

[10] 钟林辉. 构件化软件开发中演化信息的获取和应用技术研究[D]. 北京:北京大学,2007.

(上接第 2986 页)信息。当 ERP 系统需要该基本型柴油机的 MBOM 时,可直接由集成化 BOM 系统调用并获取。某基本型柴油机的集成化 BOM 信息的一个示例如图 8 所示。

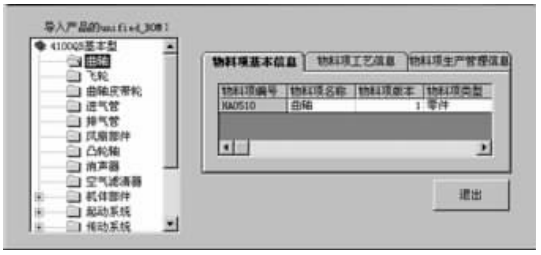

图8 某CIMS项目中集成化BOM信息示例图

3 结束语

本文针对企业实践中突出存在的 BOM 不统一问题,提出了面向并行工程的统一产品 BOM 模型。通过变革组织结构和重构 BOM 流程,建立了面向并行工程的组织体系,构建了并行的 BOM 应用架构,各 BOM 在计算机网络和分布式技术的协同环境支持下共享集成化 BOM 信息模型,并从物料项信息模型、产品结构信息模型、零件信息模型三方面详细阐述了集成化 BOM 信息模型。通过某 CIMS 应用示范工程实际应用,说明本文提出的面向并行工程的统一产品 BOM 模型可行、有效,可以较好地解决 BOM 数据正确性、完整性、一致性问题。

参考文献:

[1] 闫崇京,廖文和,宋燕,等. 基于多色图的物料清单建模研究[J]. 信息与控制,2009,38(1):121-122.

[2] 刘晓冰,黄学文,马跃,等. 面向产品全生命周期的 XBOM 研究[J]. 计算机集成制造系统 CIMS, 2002,8(12):983-987.

[3] 魏志强,王先逵,吴丹,等. 基于单一数据源的产品 BOM 多视图映射技术[J]. 清华大学学报:自然科学版,2002,42(6):802-805.

[4] 黄学文,范玉顺. BOM 多视图和视图之间映射模型的研究[J]. 机械工程学报, 2005,41(4):97-102.

[5] 赵岩,莫蓉,常智勇,等. 扩展型制造物料清单视图构建及其演绎机制[J]. 中国机械工程, 2007,18(19):2334-2339.

[6] 袁平鹏,陈刚,董金祥. BOM 一致性维护[J]. 计算机辅助设计与图形学学报,2002,14(1):83-86.

[7] MATIAS J C H, GARCIA H P, GARCIA J P, *et al*. Automatic generation of a bill of materials based on attribute patterns with variant specifications in a customer-oriented environment[J]. Journal of Materials Processing Technology, 2008, 199(1):431-436.

[8] XIONG M H, TOR S B, KHOO L P, *et al*. A Web-enhanced dynamic BOM-based available-to-promise system[J]. International Journal of Production Economics, 2003, 84(2):133-147.

[9] 熊光楞. 并行工程的理论与实践[M]. 北京:清华大学出版社, 2001.

[10] 郁鼎文,陈恳. 现代制造技术[M]. 北京:清华大学出版社,2006.