构件化软件演化信息建模和获取方法研究*

钟林辉[1]，谢　冰[2]
(1. 江西师范大学 计算机信息工程学院，南昌 330022；2. 北京大学 计算机信息科学技术学院，北京 100871)

摘　要：为了清晰、全面地获取构件化软件的演化历史信息，通常需要提供演化信息表示和建模的有效手段。分析了构件化软件演化信息获取所需解决的若干问题，提出了在软件构件模型基础上，增加建模元素以表示软件演化信息的方法；通过结合软件配置管理系统，实现对构件化软件变化过程的追踪和管理。基于该方法和相应的系统支持，可以为实施软件重构和后期开发提供指导。

关键词：软件演化；演化信息；软件体系结构；软件构件
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2014)02-0401-03
doi:10.3969/j.issn.1001-3695.2014.02.019

Research on evolution information modeling and acquisition for component-based software

ZHONG Lin-hui[1], XIE Bing[2]
(1. *School of Computer & Information Engineering, Jiangxi Normal University, Nanchang* 330022, *China*; 2. *School of Electronics Engineering & Computer Science, Peking University, Beijing* 100871, *China*)

Abstract: In order to clearly and comprehensively acquire the evolution information for component-based software, effective representation and modeling methods were needed to be provided. This paper analyzed the problems of acquiring the evolution information for component-based software, and proposed a method by adding the evolution information element to the component model. As a result, it could track and manage the change process of component-based software and provided guidance for the software refactoring and software post-development by the support of software configuration management system.

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

0　引言

软件构件、体系结构的演化会直接影响基于构件的软件工程中的各种活动如组装、部署等，缺乏有效的软件体系结构演化支持会导致各个活动之间的不一致性。为了有效地支持构件化软件的演化，不同的研究者从不同的角度提出了不同的解决方案。有的研究者在构件模型和体系结构语描述上增加变化的特征，提出了可配置的软件体系结构[1]、动态体系结构描述语言[2]、带版本信息和变化信息的软件构件模型[3]、在模型中增加管理变化的机制如预测变化的形式化语言 E-CAL(带动作的扩展约束语言)[4]、支持变化检查的复用契约[5]和变化契约[6]；有的研究者考虑到软件配置管理系统实际存储了软件演化的历史信息，试图用软件配置描述语言如 PCL[7]来描述组成系统模块的演化性，但是其研究中的软件配置管理系统局限在源代码级的，缺乏对构件化软件演化的支持，也难以同构件化软件开发中的活动相衔接；而有的研究者尝试建立与具体实现语言无关的软件演化建模方法，如 GÎrba 等人[8,9]提出了一个软件演化的元模型(Hismo 模型)，通过实例化得到针对不同类型语言和系统泛型的软件演化模型；Thomas 等人[10]则使用主题(topic)对软件演化进行建模，并利用主题演化模型定量分析了软件演化过程的一些重要特征。但是，这些方法不能有效地支持构件化软件的演化，亦缺乏同软件配置管理系统的有效结合。为此，本文分析了构件化软件演化需要解决的问题，提出了基于青鸟构件模型的构件化软件演化信息建模方法，介绍了支持构件化软件演化建模和获取的系统环境。

1　构件化软件演化信息获取所面临的问题

1)如何获取构件化软件演化过程的完整变化历史？

构件化软件的演化主要有两种情况：一是软件体系结构/构件在一个特定的范围内进行演化，如在一个项目组内对软件体系结构/构件不断升级、完善，形成新的版本，此时的体系结构/构件是连续变化的；另一种情况是软件体系结构/构件分布在不同项目组内进行演化，一个项目组内正在演化的体系结构/构件来源于另一个项目组中体系结构/构件的某个版本，此时的体系结构/构件可能是分布变化的。因此需要有一种机制能够分别记录体系结构/构件的集中变化和分布变化的版本信息，并以此为基础追踪体系结构/构件在不同项目组内的变化历史，最终形成完整的体系结构/构件变化历史。这种机制应该建立在构件化软件的核心技术之中，以便将演化信息贯穿构件化软件开发的全局，可见，这种机制应该建立在软件构件模

收稿日期：2013-06-19；**修回日期**：2013-08-11　　**基金项目**：国家自然科学基金资助项目(61262015)；江西省自然科学基金资助项目(2009GQS0053)；2013 年度江西省教育厅科学技术项目(GJJ13230)

作者简介：钟林辉(1974-)，男，江西赣州人，副教授，博士，主要研究方向为构件化软件、软件体系结构、软件演化(shiningto@sohu.com)；谢冰(1970-)，男，湖南湘潭人，教授，博士，主要研究方向为软件工程、形式化方法、分布式系统.

型的基础上。

2)如何在构件模型层次建立起对构件化软件演化的支持?

软件构件模型是构件本质特征及构件间关系的抽象描述。构件化软件的演化也是其本质特征之一。但是增加对演化性的支持需要考虑各类相关的演化信息,如新版本的修改人员、修改原因、版本命名、版本号编制机制、版本之间的关系等。软件构件模型中只应纳入最本质的特性,也就是说,只应该将最重要的信息纳入构件模型中,那么如何取舍这些信息呢?

3)如何建立起支持演化的软件构件模型和实施管理的软件配置管理模型之间的关联机制?

支持演化的软件构件模型记录了构件化软件的演化,而软件配置管理系统实际存储了系统的演化历史信息。构件化软件开发中,需要将体系结构和构件的演化映射到软件配置管理系统中去。因此,在软件构件模型和配置管理模型两者之间需要提供一种自动的映射机制,可以提高实施演化管理的效率。

2 构件化软件演化信息建模方法

2.1 扩充的构件模型

构件化软件的演化可能分别在不同范围内进行。例如,不同的项目组对同一个构件化软件进行修改,每个项目组的修改历史相应地对应着一个版本变化空间。在这个版本变化空间内,软件发生的变化是集中的。而不同的版本变化空间之间,软件的变化可能是分布的,也就可能采取了不同的版本命名规则,实施了不同的修改方针、策略,实现了不同的修改要求。

而在每个版本变化空间,构件化软件的演化可以发生在两个基本的方面:一是软件体系结构的组成构件之间的连接关系发生变化;二是软件体系结构的连接关系不发生变化,只是其中的组成构件本身发生变化。在此基础上,软件体系结构可以发生更加复杂的变化,如软件体系结构中组成构件以及组成构件之间的连接关系同时发生变化。

综上所述,软件体系结构可以用复合构件的配置来描述,相应地,软件体系结构的演化可以用复合构件的配置变化来描述。同时,为了记录体系结构集中和分布的变化,需要提供不同的处理机制。

为此本文扩充了构件模型,提出了支持软件演化的构件模型 xJBCOM,如图1所示。

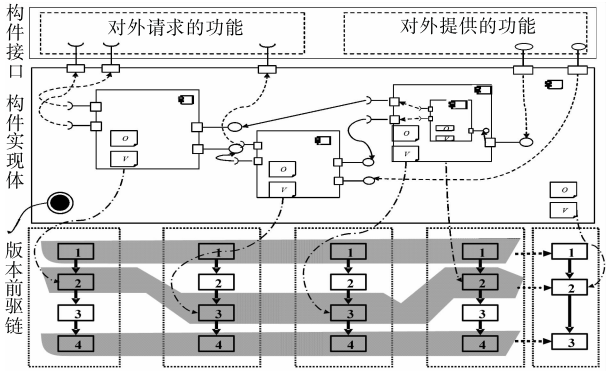

图1 支持演化的软件构件模型 xJBCOM

扩充的构件模型 xJBCOM 包含了带版本信息的原子构件、带版本信息的复合构件,并通过定义构件的版本族和前驱版本链等信息来支持复合构件本身以及成员构件的演化历史的定义。其中的相关定义如下:

定义1 软件构件 component。软件构件的定义在原有的接口 interface 和实现体 implement 两部分之外,增加版本信息。定义如下:

component = ⟨comname, interface, implement, versioInfo⟩

其中:comname 是构件名称;interface 是构件的接口;implement 是构件的实现体;versionInfo 是构件的版本信息。

定义2 构件接口 interface。接口是函数的集合,用 Z 语言规约表示为:P method。其中 method 表示函数,method = CNAME × METHOD;METHOD:P DATA→P DATA ;这些函数集合可以区分为对外请求的函数集合 requestedFunSets 和对外提供的函数集合 providedFunSets 两类,满足以下条件:

(requestedFunSets ∩ providedFunSets = ∅) ∧ (requestedFunSets ∪ providedFunSets = interface)

定义3 构件实现体 implement。对于原子构件,其实现体直接对应某一段代码,或是某一个程序实体,构件实现体表现为一个指向其代码的指针。对于复合构件,其实现体定义为 implement = ⟨compSet, constrain ⟩。其中,compSet 表示组成复合构件的成员构件集合,可以由原子构件和复合构件组成。约束 constrain 决定了 compSet 一个配置关系即成员构件间的联结关系。定义为 constrain: P(com1↔com2),其中 com1、com2 是原子构件或复合构件,满足构件对任意一个 com1 对外提供的函数,在构件 com2 中都存在一个与之函数基调相同的对外请求的函数。

定义4 版本信息 versionInfo = ⟨versionId, locationPath, projectInfo, preLink⟩。其中,versionInfo 由版本号 versionId、构件源路径信息 locationPath 以及项目信息 projectInfo 构成。preLink 表示当前构件在演化过程中的前驱构件,用前驱构件的版本号 versionId 表示;对于新构件的第一个版本,该项置为空串。由于可能修改分布在不同的开发组中,各开发组的版本命名规则可能不同,preLink 编码要求后继版本号一定要大于前驱版本编号,这实际也是一般开发中约定俗成的惯例。即满足:

preLink = component. versionInfo ∧ preLink. versionInfo. versionId⟨component. versionInfo. versionId ∧ preLink. comname = component. comname

另外,考虑构件化软件的演化历史信息捕获和表示,还可以定义构件版本族的概念。

定义5 构件版本族 comVersionGroup = ⟨*CV*,*CE*⟩

构件版本族表示一个构件在演化过程中对应的版本变化空间,其中 *CV* 是构件版本的集合,边集 *CE* = {(*cv*1,*cv*2)} 表示 *cv*1 和 *cv*2 具有版本演化关系当且仅当 {⟨*cv*1,*cv*2⟩ | *cv*1 ∈ *CV*, *cv*2 ∈ *CV*, *F*(*cv*1,*cv*2) = true}。其中函数 *F* 表示构件的演化关系。同时,用 *CV*. Label 表示构件版本族的版本号集合。

在一个开发组中,可以采用不同的版本编码策略,这里给出一种方法,如图2所示。具体方法是:版本号为一个由“.”分割的字符串,按“.”的分割从左到右排列,奇数位的数字表示分支名,偶数位的数字表示该分支下的子版本号。版本名的最后一位数字表示版本在其分支上的序号,前面的所有数字表示版本所在分支。如版本名1.2.2.0表示该版本是分支1.2.2的第0个版本。分支的最后一位表示它是对同一版本的第几个分支,前面所有数字表示此分支从哪个版本分支而来,如分支名1.2.1.2.2表示该分支是版本1.2.1.2的第二个分支。

据此方法可以保证:a)无冲突地表示整棵版本族;b)有效

区分版本名与分支名;c)清晰体现构件演化过程。

在分布的版本演化过程中,若能够获得所有的构件版本,则通过 preLink 所表示的当前构件在演化过程中的前驱构件,可以形成整体的演化历史,表达为如图 2 类型的版本树。

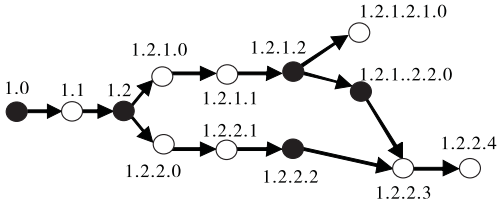

图 2　构件版本族的版本号编码规则

2.2　扩充的语法支持

与上述对应的扩充后的构件描述语言 xJBCDL 如下:

a)扩充后的构件接口规约可表示成如下形式

Component_Interface:: = Component 〈component_name〉 "〈"〈VersionInfo〉[,〈LocationInfo〉][,〈ProjectTeamInfo〉]"〉" is
[PreLink:〈component_name〉"〈"〈VersionInfo〉 "〉"]
[Provides:〈Function_Spec_list〉
Requires:〈Function_Spec_list〉]|
[Services: [dual] Service 〈service_name〉
{ , [dual] Service 〈service_name〉 }
]
Description:〈text_docu〉 semiformal document| Formal document
]
End Component_Interface;
〈VersionInfo〉:: = Version = 〈VersionNo〉;
〈LocationInfo〉:: = Location = 〈Path〉
〈ProjectTeamInfo〉:: = ComponentManager = 〈Name〉, [ComponentUsers = 〈NameList〉,][ComponentDesigner = 〈NameList〉]

b)扩充后的复合构件实现体规约成如下形式

Compound_component:: =(复合构件的实体)
Reference:(包含的成员构件)
〈component_name〉 { , 〈component_name〉 }
Instance: (成员构件的实例化)
〈instance_name〉{ ,〈instance name〉} :〈component name〉〈VersionInfo〉;
{〈instance_name〉{ ,〈instance name〉} :〈component name〉〈VersionInfo〉;}
Connection:(成员构件实例间的连接关系)
〈connection_spec〉 { 〈connection_spec〉 }
Mapping:(外部接口到内部成员构件实例的功能映射)
〈mapping_spec〉 { 〈mapping_spec〉 }
〈VersionInfo〉:: = Version = 〈VersionNo〉;

其中增加的信息可以分为三类:

a)版本控制信息,用关键字 version 和 preLink 来表示。关键字 version 用来辅助系统在组装过程中从配置库中选择合适的构件版本,不同版本组合代表系统不同演化;关键字 preLink 用来跟踪当前构件的版本是否来自于其他版本变化空间的某个版本,通过 preLink 可以有效地追踪构件的完整演化历史。

b)构件的源路径信息,用关键字 location 表示,既可以是一个逻辑路径,也可以对应构件库,甚至可以是用 URL 定义的路径信息。

c)人员管理信息,用关键字 componentDesigner、componentManger 和 componentUsers 可以表达对应软件体系结构的开发者、复合构件和成员构件对应的构件管理员以及构件的使用者列表。

3　系统支持

为了能实际地支持构件化软件演化信息的获取,需要将建模后的系统同软件配置管理模型相互映射。本文在基于构件软件配置管理系统[11]的基础上,实现了一个构件化软件演化信息建模和获取原型系统,如图 3 所示。

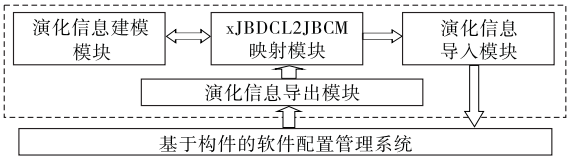

图 3　系统支持环境

该系统包括演化信息建模、xJBCDL2JBCM 映射、演化信息导入和演化信息导出等模块。其中演化信息建模模块支持用户基于文本或者图形方式描述构件化软件的演化;xJBCDL2JBCM 映射模块实现将 xJBCDL 中关于演化信息的操作映射为 JBCM 所支持的操作如检入、检出操作;演化信息导入负责根据映射结果生成相应的系统版本或者基线,而导出模块则可以逆向从软件配置系统中生成演化信息的描述信息。

4　结束语

本文分析了构件化软件开发过程中演化信息获取面临的主要问题,提出了在软件构件模型中增加与软件演化相关信息的方法,并在语法层次提供相应的支持,以实现对构件化软件演化信息的建模;同时,通过与基于构件的软件配置管理系统的无缝对接,实现对构件化软件演化信息的获取。这为今后构件化软件的演化分析和系统维护奠定了基础。

致谢　感谢北京大学软件工程研究所邵维忠教授、张路教授对本文工作的支持和帮助。

参考文献:

[1] MAGEE J, DULAY N, EISENBACH S, *et al*. Specifying distributed software architectures[C]// Proc of the 15th European Software Engineering Conference. 1995:137-153.

[2] ALLEN R J. A formal approach to software architecture[D]. Pittsburgh, PA: Carnegie Mellon University, 1997.

[3] Van OMMERING R, Van der LINDEN F, KRAMER J, *et al*. The Koala component model for consumer electronics product software[J]. IEEE Computer,2002,33(3): 78-85.

[4] AJIL A, SAMUEL A. E-CAL: a formal language for software model evolution[C]//Proc of IEEE International Conference on Information Reuse and Integration. 2011:212-217.

[5] MENS T. A formal foundation for object-oriented software evolution [C]//Proc of IEEE International Conference on Software Maintenance. 2001:549-552.

[6] QI Da-wei, YI J Y, ABHIK R. Software change contracts[C]//Proc of the 20th ACM SIGSOFT International Symposium on the Foundations of Software Engineering. 2012:1-4.

[7] GULLA B, GORMAN J. Experience with the use of a configuration language[C]//Proc of the 6th SCM Workshop of the ICSE. 1996: 198-219.

[8] GÍRBA T, DUCASSE S. Modeling history to analyze software evolution:research articles[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2006,18(3): 207-236.

[9] GÎRBA T. Modeling history to understand software evolution [D]. Berne:Fakultät der Universität, 2005.

[10] THOMAS S W, ADAMS B, HASSAN B. Validating the use of topic models for software evolution[C]// Proc of the 10th IEEE International Working Conference on Source Code Analysis and Manipulation. 2010:55-64.

[11] 张路. 基于构件的软件配置管理技术研究[D]. 北京:北京大学, 2000.