基于构件的软件版本管理系统

李 欣,张 路,谢 冰,杨芙清
(北京大学软件工程研究所,北京 100871)

摘 要: 传统的软件配置管理建立在文件版本控制的基础之上,现代大型软件系统的开发要求在更大粒度上进行版本控制.同时,基于构件的软件开发是当前的发展趋势,也需要适应其特点的配置管理支持工具.本文提出了基于构件的软件版本控制模型,采用将版本控制与并发控制单位分离的策略,从而在大粒度版本控制的同时,保证了并行开发的灵活性.

关键词: 配置管理; 软件构件; 版本控制; 并发控制
中图分类号: TP311 **文献标识码**: A **文章编号**: 0372-2112 (2000) 11-0119-03

Component-based Software Version Management System

LI Xin, ZHANG Lu, XIE Bin, YANG Fu-Qing
(Software Engineering Institute, Peking University, Beijing 100871, China)

Abstract: The unit of traditional software version control is a file. Version control of large unit is required while developing large software systems that contain a great number of files. Furthermore, it is needed to support component-based software development, which is the trend of software industry. In this paper, it is proposed that the basic version control unit should be a component, and that the concurrency control unit should be different from version control unit, thus it can improve the capability of concurrency control while using a large version control unit.

Key words: configuration management; software component; version control; concurrency control

1 引言

配置管理系统[3,9]是软件开发的关键支撑工具之一,是一种管理软件开发和维护过程以及其中各中间软件产品的系统,是 ISO 与 CMM 质量保证体系的核心支持工具.配置管理研究怎样在不同时刻标识软件系统的配置,以便系统化地控制配置的改变,并在整个软件系统的生命周期内维护配置的完整性和可追踪性[1].其中,版本管理是基础和核心.

传统的版本管理系统以文件作为管理的基本粒度.版本管理系统记录、维护每个文件的演化历史.在大型软件开发中,系统往往包含较多文件,这使得传统方式版本管理的工作量很大,而且不易于描述文件间内在的组合关系.目前,基于构件的软件开发方法已成为发展趋势[7,8].构件作为系统的有机构成成分,在物理上可以表现为多个文件的集合体,而在开发过程中是作为一个原子单位使用的.系统的开发者关心的是构件整体的开发、演化,组装和维护.这种大粒度的开发方法,对版本管理提出了新的要求[3].这些要求包括:

应能有效存储和管理构件演化历史.

操作模型应有利于体现构件的整体性,降低系统开发的复杂程度.

需要保证并行开发构件时的正确性,同时不减少项目组协同工作的灵活性.

本文研究了构件的版本控制策略,提出了基于构件的版本管理模型.针对并行开发问题,又提出了分别在构件和文件粒度上进行版本管理和并发控制的方法.在此基础上,设计实现了一个产品化的配置管理系统 JBCM.该系统既提升了管理的粒度,又能确保团队开发具有较好的并行性.

2 以构件为粒度的版本管理

2.1 版本管理系统中的构件定义

在基于构件复用的青鸟软件生产线中,软件构件定义如下:"构件是可以被多个软件系统复用的具有独立功能的系统构成成分"[8].构件在实际形态上可表现为通过目录结构组织起来的一些文件的集合,并且是系统中可以明确辨识的构成成分.需要指出的是,在以前许多有关版本管理的文献中都出现了构件的概念[4],但其中的构件一般指的就是文件.本文中的构件则是应用系统中多个相关文件构成的一个逻辑整体,例如一个类的定义及其实现,一个完整的功能模块等.

构件版本是构件组成文件版本的集合.构件版本的变化不仅体现了组成文件的版本变化,同时也反映了构件中文件组成的变化.也就是说,组成文件发生版本演化,或者增加和删除构件中的文件,都会引起构件版本的演化.在基于构件的系统中,文件版本由系统内部控制,用户只关注构件版本,从而提升了管理层次.图 1 反映了构件版本与文件版本的关系:

收稿日期:1999-12-21;修回日期:2000-07-26

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

图中虚线箭头表示构件和文件与其不同版本的关系,实线箭头表示构件版本由文件版本组成的关系. 从图中可以看出构件的版本 2 比版本 1 增加了一个文件 3,而且文件版本也发生了演化. 构件版本的演化与文件版本的演化同步进行,并且随着文件的版本演化自动产生.

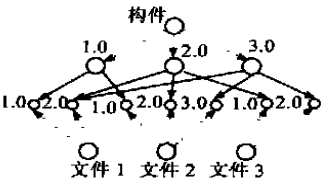


图 1 构件版本与文件版本

基于上述构件与文件关系模型,提出并实现了基于构件的软件版本管理系统.

2.2 构件的版本管理

(1) 以构件为粒度的版本管理特点

与基于文件的版本管理相比,基于构件的版本管理有以下主要特点: ①构件的抽象级别比文件高. 构件是应用系统中可以明确辨识的构成成分. 记录、维护构件的版本比文件的版本管理更有意义. ②构件的粒度可以比文件大很多. 一个项目中可能有诸多分布的逻辑单元,这些逻辑单元与构件相对应. 构件的数量较少,而且整体逻辑意义明显,可以更清晰地体现项目的演化历史. ③在构件基础上,可以体现出系统的层次性、构造性等特征. 同时,构件版本管理也可以满足对文件版本的管理需求,使版本管理既有大粒度,又有灵活性.

(2) 构件版本管理的基本模式

基于构件的版本管理系统采用"检出(Check Out)、修改、检入(Check In)"的基本操作模型,操作的基本单位是构件. 使用者需要先将构件从版本库检出到工作区,随后在工作区中完成对构件的修改,最后将修改的结果检入版本库. 构件组成文件的增删以及其中任何一个文件的修改都被视为对整个构件的修改. 因此,作为检入操作的结果,版本管理系统会自动生成构件的一个新版本. 以构件版本为粒度的版本管理系统记录和管理了开发人员对构件修改的历史.

(3) 构件版本树

在基本模式之上,使用者可以构件的某个版本进行分支,建立一个新的开发流,以适应不同的开发需求. 构件多个分支还可以进行合并. 由此,一个构件的所有版本构成了一棵版本树. 构件版本树是系统对构件进行版本管理的基础.

构件的版本树是由版本管理系统维护,系统需要提供比较完善的版本名自动生成与管理机制. 其基本原则是: ①无冲突地表示整棵版本树; ②有效的区分版本名与分支名; ③有利于体现构件的开发过程.

图 2 是采用的一种比较实用有效的版本树命名方法. 版本 1.0 为构件的初始版本,实心节点为分支点. 版本 1.2.1.2.2.0 与版本 1.2.2.3 合并后形成版本 1.2.2.4.

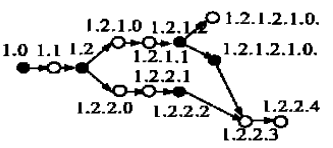


图 2 构件版本树

3 以文件为基本粒度实现并发控制

3.1 版本控制与并发控制单位的分离

版本管理系统应能对项目组共同开发软件系统提供管理支持. 多个开发人员可以分工开发不同构件,也可以同时开发同一构件. 为了保证协同开发的安全性和正确性,必须解决构件开发过程中的并发控制问题.

在基于文件的版本管理系统中,版本控制与并发控制的基本单位都是文件. 使用者可以在检出时对文件加锁,防止其它用户对该文件进行修改;检入时,在生成文件新版本的同时,要对文件解锁. 而在基于构件的版本管理系统中,如果把并发控制的单位定为构件,在需要修改时对构件加锁,会造成其他使用者无法同时修改构件和构件中的任何文件. 因此,提出并实现了以构件为版本控制单位,以文件作为并发控制单位的管理方法.

3.2 文件的操作模式与并发控制

在实现基于构件的版本管理系统中,构件是版本控制单位,而并发控制则在构件内部的文件级别上进行管理. 对于一个构件,使用者可设定其中文件的操作模式,由此来控制相关文件的加锁活动.

文件的操作模式可以分为三种:"只读"、"排它写"和"共享写". "只读"表示对文件不加锁,使用者也不能对文件进行修改,使用者可以使用"只读"模式来浏览文件;"排它写"表示对文件加锁,使用者可以对文件进行修改,但不允许其他使用者同时进行修改,这样可以保证对同一文件的修改是串行化的;"共享写"则表示对文件不加锁,使用者可以对文件进行修改,而且允许其他使用者同时对同一文件进行修改. 虽然"共享写"模式可以提高并发程度,但会带来多个用户修改结果互相冲突的问题. 版本管理系统可以分析这种冲突,并提示用户进行相应的合并处理,由此解决了文件内容的一致性问题.

在构件的修改过程中,文件的操作模式是可以随时改变的. 例如,使用者先用只读模式检出构件中的各个文件,而在要修改文件时,可将需修改文件的操作模式改为"排它写"或"共享写",然后修改. 完成了对构件中所有相关文件的修改,也就完成了对构件的修改. 这时检入修改后的构件,系统会自动产生该构件的一个新版本. 如果在使用者检入之前另一使用者已检入了构件的另一新版本,该使用者就需要先进行更新操作,系统会将构件最新版本拿到工作区,与该使用者修改过的版本进行合并后才能检入. 使用者对文件操作模式的修改必须满足一定的条件. 具体转换条件见图 3.

3.3 文件的操作权限管理

操作权限用于管理特定用户对构件中文件的使用权限,分为"只读"和"可写"两种. 它控制的是用户对文件的操作能力. 用户操作模式是用户检出构件时对文件的实际操作. 用户操作模式不能超过操作权限. 例如,拥有"只读"权限的用户只能用"只读"模式检出文件,拥有"可写"权限的用户才可以用"只读"、"排他写"和"共享写"模式修改文件.

操作权限与操作模式一样,都是针对构件中的文件. 通过操作权限的管理,可以划分不同用户对同一构件所承担的工作任务. 操作模式和操作权限相结合,有效地解决了多个用户协同开发同一个构件时的并发控制问题.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

转换前操作模式	转换后操作模式	转　换　条　件
只读	排它写	①该版本是文件的最新版本、②没有其它使用者对该文件的操作模式为“排它写”或“共享写”
只读	共享写	没有其它使用者对该文件的操作模式为“排它写”
排它写	只读	无条件
排它写	共享写	无条件
共享写	只读	无条件
共享写	排它写	①该版本是文件的最新版本、②没有其它使用者对该文件的操作模式为“共享写”

图 3　各种文件操作模式间的转换

4　基于构件的版本管理系统的实现

在上述研究基础之上,研制了青鸟配置管理系统 JBCM 的核心部分 ——基于构件的版本管理系统 JBVM. JBVM 对基于构件的软件开发方法提供了充分支持,其最主要的特点是将构件与文件版本管理分布于两个不同层次,在系统设计方面也就具有相应的层次性. 本节主要讨论设计与实现中的关键问题及其解决策略.

4.1　版本库的组织结构

版本库是构件版本演化历史的存储区. 版本库中记录的数据分为版本构件和元数据两类. 版本构件是用户交付的各类构件,由于同时具有版本信息,故称为版本构件. 元数据是管理系统维持自身运行所需的各种控制文件与完成各种版本管理功能所需的信息,包含日志、用户管理、并发控制、权限控制、操作模式、统计与审计等信息.

版本管理系统设计的核心内容包括版本库的组织、结构与维护. 版本库的组织结构分为两个层次:项目层和构件层. 对于系统开发人员来说,版本库中的项目可以与系统开发中的工作划分相对应,一个软件系统的整体或子系统都可以作为一个项目. 而构件对应于软件系统结构中最低层的不可再分的基本逻辑单元,可以是系统开发过程中需要作为一个整体的文件集合,例如用于实现一个 C + + 类的所有. cpp 和. h 文件. 下面是版本库的一个简化结构示例图. 在项目 a(用 a. Prj 表达)下,有 c 和 d 两个构件(用 c. Comp 和 d. Comp 表达),而 c 构件由 e. cpp、f. h 和一个 g 目录组成. TempFile 是版本库的临时文件存储区.

4.2　构件的增量存储

构件的不同版本间具有较大的内容相似性,不同版本的存储有必要使用增量存储方式,以减少存储冗余. 在传统的文件版本管理系统中,已较深入地研究了文件的增量存储技术,有实际的算法及实现,如最长公共子序列算法[5]等. 根据基于构件的版本管理的特点,提出了按构件和文件两个层次分别进行增量存储的设计方案:

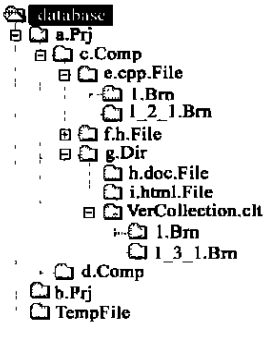


图 4　版本库的层次结构

(1) 对于构件中具体文件的修改,使用传统的文件逆向增量存储技术[2],保存最新文件版本与增量部分.

(2) 构件的版本由附属目录及文件的版本组成. 构件的增量存储体现在版本中记录的是相比前一版本组成成分的不同部分.

(3) 对于文件和构件存在多个分支的情况,对各分支独立进行增量式存储. 保存每个分支上的最新版本和其他版本相对于各自后一版本的增量.

4.3　构件版本的比较与合并

构件演化过程中会产生多个分支,各分支都会实现一些不同于其他分支的新特性. 有时需要将这些特性都合并到一个版本中. 为了保证在多个用户协同工作时的正确性,也经常需要合并多个用户分别修改过的构件版本. 构件不同版本的合并采用如下策略:不同版本添加的文件有选择地添加到合并版本中,修改过的文件逐一进行合并.

构件版本的比较是版本合并的基础. 在合并之前,可以通过构件版本的比较得到不同版本间的差异. 除此之外,为了查看构件的演化历史,实现对构件的变化与过程控制,也需要比较构件的新老版本;在基于构件的版本管理系统中,构件版本的比较分为两个层次. 底层是文件级的比较,通过比较不同版本文件的具体内容,得到文件内容的差异. 上层是构件级的比较,通过比较构件不同版本组成成分,来获取构件整体的变化情况. 在 JBVM 系统中,通过构件级别上的比较则可分析出构件的不同版本各自包含哪些文件,以及分别对哪些文件进行了修改.

5　结束语

本文着重讨论了基于构件的软件版本管理模型,以及相应系统的设计与实现. 基于构件的思想不仅可以应用于版本控制,配置管理中的其它功能,例如配置支持、构造支持、审计控制、统计报告、过程支持和团队支持等方面都可以建立在构件的基础上. 我们将进一步研究以构件为粒度配置支持、构造支持和变化控制方法.

作者简介:



李　欣　1975 年出生. 北京大学信息科学中心硕士研究生. 1997 毕业于武汉大学计算机系,获学士学位. 主要从事软件工程和软件工程环境、软件复用和软件构件技术等方面的研究.



张　路　1973 年出生. 北京大学计算机科学技术系博士研究生. 主要从事软件工程和软件工程环境、软件复用和软件构件技术等方面的研究.

(下转第 131 页)

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

体相互平行,层间间距与同层间距均为 1μm,且每根导体长为 $(2x-1)$μm.

表 1　Lcap 与 Fastcap 的速度、精度比较

例子名称	Fastcap 变量数	Fastcap 电容	Fastcap 时间	Lcap 变量数	Lcap 电容	Lcap 时间	精度差别(%)	加速比
Pb33	2148	974	38	4015	937	4.2	3.8	9.0
Pb44	3546	1125	122	6669	1071	3.0	4.8	40.7
Pb55	5402	1519	60	10381	1463	7.5	3.7	8.0
Pb333	1196	1033	8	2392	1011	2.5	2.1	3.2
Pb444	1880	1370	22	3685	1369	3.7	0.07	5.9
Pb555	9944	1605	171	19802	1515	10.2	5.6	16.8

注:电容单位 10^{-18}F(法拉),时间单位 s(秒).

图 2 显示了 Pb444 的结构,导体到介质表面的距离均为 1μm,衬底位于平面 $z=0$,介质相对介电系数为 3.4554,所求为深颜色导体的电容.

从表 1 可看出,Lcap 同 Fastcap 计算结果符合的比较好,最大误差为 5.6%.在速度上,Lcap 明显优于 Fastcap.对规模最大的例子 Pb555,Lcap 有 19802 个变量,仅用 10.2 秒,而 Fastcap 的变量数 9944 大约是它的一半,却用了 171 秒.可以看出,随着问题规模的增加,加速比也在加大,图 3 直观表明了这一点.

图 3 是表 1 中各例在不同划分、不同变量数情况下的速度比较,它利用远多于表 1 的数据,并使用二次曲线拟合得到比较曲线.可以粗略看出,二者计算复杂度都约为 $O(n)$,但 Lcap 要快许多,原因主要是以线电荷大量代替面电荷,使近场积分的计算速度大大加快了.

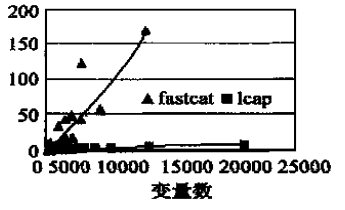


图 3　Fastcap 与 Lcap 计算时间比较

4　结论

本文在间接边界元素法求解 VLSI 寄生电容的过程中,引入了导体表面使用线边界元、自然边界使用面边界元的混合边界元思想,并采用多级加速法求解方程组,使得计算速度提高很多,取得了速度与精度的良好平衡.

参考文献:

[1] Sharad Kapar, David E. Long. IES^3: A Fast Integral Equation Solver for Efficient 3-Dimensional Extraction [M]. Proc. of ICCAD, 1997: 448 - 455.

[2] K. Nabors, J. White. Fastcap: a multipole accelerated 3-D capacitance extraction program [J]. IEEE Trans. CAD, Nov. 1991, 10(11): 1447 - 1459.

[3] Zhen-qiu Ning, Patrick M. Dewilde. SPIDER: capacitance modeling for VLSI interconnections [J]. IEEE Trans. on Computer-aided Design, December 1988, 7(12): 1121 - 1228.

[4] Z. Wang, Y. Yuan, Q. Wu. A paralled multipole accelerated 3-D capacitance simulator based on an improved Model [J]. IEEE Trans. CAD, 1996, 15(12): 1441 - 1450.

作者简介:



杨钊志　清华大学计算机科学与技术系 98 级博士研究生.



王泽毅　清华大学计算机科学与技术系教授、博士生导师.主要研究方向为 VLSI-CAD 中的器件模拟,互连分析及其中的串并行方法.

(上接第 121 页)

参考文献:

[1] E. H. Bersoff. Elements of software configuration management [J]. IEEE Trans. on Software Engineering, 1984. 10(1).

[2] Randal. C. Burns, Darrell D. E. Long. A Linear Time, Constant Space Differencing Algorithm [M].

[3] Carma M. Clure. Software reuse techniques [J]. Prentice Hall PTR, New Jersey, ISBN 0-13-661000-5, 1997.

[4] S. Dart. Concepts in configuration management systems [A]. In Proceedings of the Third International Workshop on Software Configuration Management [C]. ACM SIGSOFT, 1991: 1 - 18.

[5] Daniel. S. Hirschberg. Algorithms for the longest common subsequence problem [J]. ACM, October 1977, 24: 664 - 675.

[6] Yi-Jing Lin, Stewen P. Reiss. Configuration management with logical structures [A]. Proceedings of ICSE-18, 1996: 298 - 307.

[7] H. Mili, F. Mili, A. Mili. Reusing software: issues and research directions [J]. IEEE Transaction on Software Engineering, June 1995: 21(6).

[8] 杨芙清,梅宏,李克勤. 软件复用与软件构件技术 [J]. 电子学报, 1999, 27.

[9] 张路,李欣,梅宏,杨芙清. 基于复用的软件开发过程中的配置管理 [J]. 计算机科学, 1999, 26(5).

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net