

基于版本复制技术的版本树管理

张小勇^{1,2}, 洪刚^{1,2}

(1 南京师范大学 计算机科学与技术学院, 江苏 南京 210097;
2 江苏省信息安全保密技术工程研究中心, 江苏 南京 210097)

[摘要] 针对协同图形编辑系统中基于版本复制技术的多版本协同技术, 深入研究了版本树的管理问题, 包括版本树优化、版本之间冲突和操作意愿保证, 给出了版本之间冲突操作的实现算法. 实验表明, 该方案满足协同系统的要求.
[关键词] 协同图形编辑系统, 版本复制, 多版本协同, 版本树管理
[中图分类号] TP391 [文献标识码] A [文章编号] 1672-1292(2010)02-0074-05

Management of Version Tree Based on Version Replication

Zhang Xiaoyong^{1,2}, Hong Gang^{1,2}

(1 School of Computer Science and Technology, Nanjing Normal University, Nanjing 210097, China
2 Jiangsu Research Center of Information Security & Privacy Technology, Nanjing 210097, China)

Abstract Concurrency control is one of key techniques in cooperative graphics editing systems. This paper focuses on the multi-versioning cooperation based on version replication in the cooperative graphics editing systems and makes a deep research on the management of the version tree including optimization of the version tree, conflict treatment among versions and intention preservation of operations. Then an algorithm of treatment of conflict operations among versions is implemented. The experiment shows that the schemes in this paper can satisfy the requirements of the cooperative editing systems.
Key words cooperative graphics editing system, version replication, multi-versioning cooperation, management of version tree

协同图形编辑系统的对象可通过一组属性来表现, 这些属性从空间关系的角度看, 分成几何属性 (例如位置、大小等) 以及非几何属性 (例如颜色、线型等). Sun 等人提出的对象复制方法虽然在一定程度上支持自然、自由的交互, 但还不能有效解决非几何属性操作引起的冲突问题^[1-5]. 三维绘图系统中, 由于几何求解的不可逆性, 复杂的图形对象难以恢复到冲突操作执行前的状态. 由于冲突产生的复制对象不能共存于同一个对象中, 这会失去原始设计的意义. 如果两个复制对象位置很近必然会相交, 经过布尔计算会产生复杂的图形对象, 因此对象复制难以实现. 版本复制技术依据文档版本在冲突操作发生时直接复制文档, 使得冲突的操作分别作用于不同文档版本, 从而解决冲突问题, 保证操作的意愿^[3-5]. 本文研究了基于版本复制的多版本协同系统, 对其中的关键技术进行了深入讨论.

1 版本复制技术

基于版本复制的多版本技术, 当冲突操作发生时, 将对目标物理文档进行版本复制, 产生另外两个物理子版本, 并将冲突的两个操作在新产生的两个不同文档版本上分别执行^[1]. 设 G 是版本 V 上的一个对象, $EC(O)$ 为操作 O 的执行上下文. 若在 $EC(O)$ 中存在一个操作与 O 冲突, 则从原始版本 V 派生出两个复制版本 V_a 和 V_b . 操作 O 及其冲突操作分别应用于 V_a 和 V_b .

收稿日期: 2009-12-28
基金项目: 江苏省高校自然科学基金 (07KJD520112).
通讯联系人: 张小勇, 实验师, 研究方向: 智能控制与协同系统. E-mail: zhangxiaoyong@njnu.edu.cn

存在作用于同一对象 G 上的两个 mov 并发操作, 一个操作试图将 G 移动到文档的右上方, 另一个则试图将 G 移动至文档左下方, 系统为了实现这两个冲突操作, 先将初始版本 V 返回至原先状态后, 复制出两个新版本 $V1$ 、 $V2$ 再将冲突操作分别作用于新版本 $V1$ 、 $V2$ 此时系统文档由 3 个文档版本构成: 版本 V 、版本 $V1$ 和版本 $V2$ 如图 1 所示。

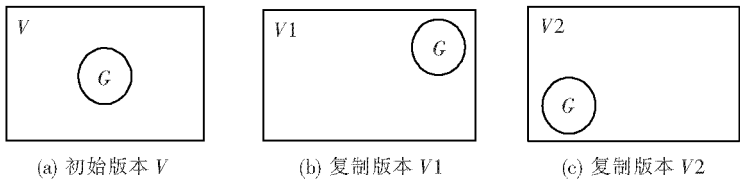


图 1 版本复制技术

Fig.1 The technique of version replication

由图 1 可以发现, 使用版本复制技术可以很好地解决非几何属性操作冲突时对象复制所不能解决的情况. 因此基于版本复制的协同多版本技术存在研究的意义. 随着协同编辑的进行, 版本的层次结构将不断延伸, 形成一个树状结构, 称之为版本树. 版本树的每个节点代表一个版本, 存储了这个版本上执行过的所有操作. 协同多版本技术的关键是使用版本递增创建算法对一组并发操作进行分析和版本创建, 算法的原理是每产生一个不冗余的相容组, 就代表一个版本, 相容组中的操作就是这个版本应该存储的操作. 版本复制的前提是有冲突操作存在, 而版本复制后, 各子版本中存储的操作都包含了父版本中的操作; 且各子版本中存储的操作的差别是那些引起冲突的操作. 抽取出各个子版本中那些相同的操作存储于父版本中, 只将冲突的操作分别存储于各个子版本中, 最大程度上避免同一个操作出现在不同的版本中, 实现操作的分层次存储^[6-7].

2 版本树管理

2.1 版本树优化选择

如图 2 所示, 节点 2 中, 其操作执行顺序为 O_2, O_3, O_4, O_b , 则最大冲突组 (Max Conflict Set) 依次为: $MCS_1 = \{O_2\}$, $MCS_2 = \{O_3, O_4\}$, $MCS_3 = \{O_2, (O_3, O_4)\}$, $MCS_4 = \{(O_1, O_2), (O_3, O_4)\}$. 可以发现, 节点 1 和节点 2 的版本树中子版本的视图效果一样, 但父版本中操作不一样, 节点 1 的两个父版本的操作分别为 O_3, O_4 而节点 2 上的为 O_1, O_2 , 出现了各站点间版本树不一致性的情况. 解决方法是对冲突操作集合中的冲突组进行排序, 按每个组中操作的全序关系对组排序, 冲突组中全序关系小的操作所在组, 排在集合前面, 以此类推.

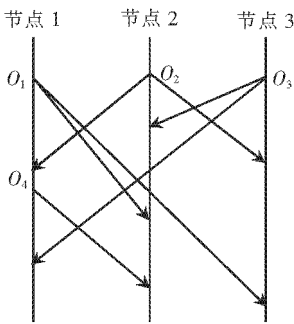


图 2 一个应用实例

Fig.2 An application scenario

对节点 1 上的最大冲突组 MCS_3 排序后结果为 $MCS_3 = \{(O_b, O_2), (O_3, O_4)\}$, 和节点 2 的最大冲突组 MCS_3 一致. 排序后, 各站点版本树的一致性得到保证. 若存在并发操作 O_b, O_2, O_3, O_4, O_5 , 且 $O_1 \times O_2 \times O_5, O_3 \times O_4$ (“ \times ”代表操作冲突). 根据改进的多版本增创算法^[5]和冲突组排序, 得到 $MCS_3 = \{(O_1, O_2, O_5), (O_3, O_4)\}$, 其产生的版本数目为 9 个; 但当最大冲突组 $MCS_3 = \{(O_3, O_4), (O_b, O_2, O_5)\}$, 其产生的版本数目为 8 个; 当冲突组中操作较多时, 冲突组就占用较多的空间. 解决方法是将冲突组集合中操作个数少的组排在冲突组集合的前面, 对相同操作个数的组再排序. 上例中排序后的最大冲突组 $MCS_3 = \{(O_3, O_4), (O_b, O_2, O_5)\}$, 版本总的数目压缩到最少.

由于改进的多版本技术采用操作分层次存储的结构, 某个版本上执行过的操作, 分为两部分: 一部分是存储在此版本上的操作, 另一部分是分布在其所有父版本中的操作. 当用户在系统视图中激活某一个版本时, 系统需要遍历此版本的所有父版本中操作, 并结合自己版本中的操作, 得到作用在此版本上的所有操作.

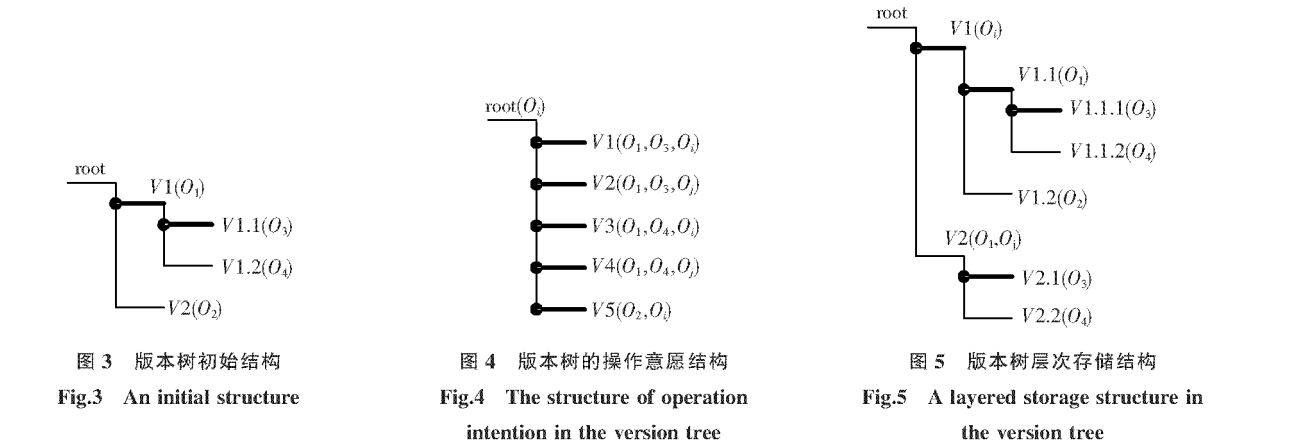
2.2 版本之间冲突与操作意愿保证

在基于版本复制的协同编辑过程中存在多个版本后, 会有并发操作分别作用于父版本和子版本. 改变父版本和子版本中的同一对象的同一属性到不同值时, 系统需处理版本间的冲突操作. GMCS表示整个版本树的结构, 以及版本树中的所有操作, 由狭义 MCS的嵌套定义得到. 狭义的 MCS只能表示执行在一个版本上的所有操作, 故 GMCS是 MCS的扩展. 具体定义为: 当版本树深度小于等于 2 层时, GMCS即为 MCS; 深度大于 2 层时, GMCS表示为本节点的相容操作组和各个子版本的 GMCS组成的集合, 以此类推得到 GMCS

例如, 根版本 root有两个子版本 $V1(O_1)$ 和 $V2(O_2)$, $O_1 \neq O_2$, $V1$ 版本存在两个子版本 $V1.1(O_3)$ 和 $V1.2(O_4)$, $O_3 \neq O_4$. 版本树如图 3 所示. 则此时要得到整个版本树的 GMCS 需先得到子版本 $V1$ 的 $GMCS_{V1}$ 即为 $V1$ 的 $MCS_{V1} = \{O_1(O_3, O_4)\}$; 以及 $V2$ 的 $GMCS_{V2} = \{O_2\}$; 故 $GMCS = \{O_c(GMCS_{V1}, GMCS_{V2})\} = \{O_c(O_1(O_3, O_4), O_2)\}$. 其中 O_c 代表 root版本中相容的操作如创建图形操作.

假设此时接收到远程的 2 个并发操作, O_i 执行在版本 root上, 改变图形 G 的颜色为红色, O_j 执行在版本 $V1$ 上, 改变图形 G 的颜色为绿色. 根据定义 1 可以分析出 $O_i \neq O_j$. 由于 O_i 作用在版本 root上, 所以 O_i 操作应该执行于 root版本的所有子版本中, 即执行于版本 root, $V1, V1.1, V1.2, V2$ 上. 同理 O_j 执行于版本 $V1, V1.1, V1.2$. 根据版本复制原则, 版本复制后在保证用户操作意愿的基础上, 版本树的结构应该如图 4 所示, 此结构没有采用存储优化.

为了保证操作意愿, 实现其操作的层次存储时需要注意以下情况: O_i, O_j 相互冲突, 则这两个操作必须分离, 一种有效的方式是在 root版本下分离出新操作. 图 5 表示了版本间冲突操作分离后的版本树的操作层次存储结构. 该方式虽然产生了较多的中间版本, 但在满足用户操作便利性、操作意愿保证性上都取得了较好的效果, 符合现实环境中的开发设计.



2.3 版本间冲突操作的实现

设并发操作 O_i 目标版本为 V_i , O_j 目标版本为 V_j 且 $O_i \neq O_j$ 则子树构造法为:

- (1) 当 O_i 为待执行操作, O_j 为已执行的操作, 且 V_i 为 V_j 的父版本. 则 O_i 的执行过程如下: 遍历版本树得到版本 V_j 上执行过的所有操作, 作为 V_i 版本的一个新的子版本节点, 其子树为原 V_j 的子树的复制; 版本 V_i 的子版本 (不含上一步中创建的新版本) 删除操作 O_j 后作为只含操作 O_i 的新版本的子树; 将 O_i 所在版本作为 V_i 另一个新的子版本.
- (2) 当 O_j 为待执行操作, O_i 为已执行的操作, 且 V_i 为 V_j 的父版本. 则 O_j 的执行过程如下: 删除 V_i 中操作 O_i . 遍历版本树得到版本 V_j 上执行过的所有操作与 O_j 合并作为 V_i 版本的一个新的子版本节点, 其子树为原 V_j 的子树的复制; 版本 V_i 的子版本 (不含上一步中创建的新版本) 作为只含操作 O_i 的新版本的子树; 将 O_i 所在版本作为 V_i 另一个新的子版本.

根据广义最大冲突组和子树构造法给出一个实现算法, 解决版本间冲突操作 (Conflict operation between Multiple Versions, COBMV) 的问题.

版本间冲突操作实现算法 COBMV (O_i) 如下:

function COBMV(O_i)

Input O_i : 待执行的操作.

Output GMCS 整个版本树的结构.

{

- 1 得到现有的整个版本树的结构 GMCS
- 2 记录 O_i 执行的版本 V_i
- 3 O_i 与 V_i 及其所有子版本和父版本的广义最大冲突组相比较:
 - 1 若都不存在冲突, 则 O_i 加入至 V_i 的 GMCG 的相容操作组中.
 - 2 若与 V_i 的某个子版本的 GMCG _{V_i} 中操作 O_j 冲突时:
 - 1 GMCG _{V_i} 复制为 V_i 的一个新的子版本的 GMCG' _{V_i} , 即 $\text{GMCS}^+ = \text{GMCG}'_{V_i}$;
 - 2 $\text{GMCS}_V = \{\{O_i\} + (\text{GMCS}_V - O_j)\}$;
 - 3 若与 V_i 的父版本中的相容操作 O_j 冲突:
 - 1 V_i 父版本中的相容操作组 $\text{GMCG}_V = \{O_j\}$;
 - 2 V_i 操作的 GMCS_V 与 O_i 合并成新的 GMCG' _{V_i} , 即 $\text{GMCS}^+ = \{O_i + \text{GMCG}'_{V_i}\}$;
 - 3 $\text{GMCS}^+ = \{\{O_j\} + \text{父版本的所有子版本的 GMCS}\}$;

} //end function

以上只给出了算法的语言描述, 其主体部分就是用广义最大冲突组实现子树构造法. 对于上例中, 若节点先执行 O_i , 则 $\text{GMCS} = \{O_c(O_i(O_1(O_3, O_4), O_2))\}$, 再执行 O_j 时, 发现 O_j 与父版本中的 O_i 冲突, 则从 GMCS 取出操作 O_i 成为新版本 $\{O_i\}$, 并复制其所有的子版本的 GMCG _{V_i} 到新版本下, 再将 $\{O_1(O_3, O_4)\}$ 的相容操作组与 O_j 合并中形成新的 GMCS_{new} , 故最后的 $\text{GMCS} = \{O_c(O_i(O_1(O_3, O_4), O_2)), O_j(O_1(O_3, O_4))\}$; 同理执行顺序为 O_j, O_i 时应用算法 COBMV 将得到相同的版本树的结构 GMCS

可将该算法集成到已有的协同图形编辑系统 CoWork 中. 以下给出一个版本冲突和执行过程, 操作具体为: $O_1 = \text{Create}(V_{\text{root}}, G)$, $O_2 = \text{Move}(V_{\text{root}}, G, X)$, $O_3 = \text{Move}(V_{\text{root}}, G, Y)$, $O_2 \times O_3$ 发生版本复制得到子版本 $V_1(O_1, O_2)$, $V_2(O_1, O_3)$, 然后执行 $O_4 = \text{Color}(V_1, G, \text{red})$, $O_5 = \text{Color}(V_2, G, \text{blue})$, 且 $O_4 \times O_5$, 导致 V_1 再发生版本复制得到版本 $V_{1.1}$ 和 $V_{1.2}$ 其中 V_{root} 对应系统的根节点版本 $V_{\text{rootVersion}}$.

操作 O_1 执行后, 在版本 $V_{\text{rootVersion}}$ 上执行. 图 6 显示了节点 0 (对应用户 mary) 的用户界面. 随后此操作被发送到节点 1 上, 节点 1 上的图形编辑区将出现相同的图形 A1. 节点 0 上的操作 O_2 将版本 $V_{\text{rootVersion}}$ 上的对象 G 向下平移, 图 7 显示了平移操作正在进行时视图的交互状态. 图 8 表现的是节点 1 上的操作 O_3 将版本 $V_{\text{rootVersion}}$ 上的对象 G 向右平移的交互过程.

操作 O_2, O_3 并发执行时, 节点 0 执行 O_2 操作时如图 7 所示. 节点 1 执行 O_3 时操作如图 8 所示.



图 6 O_1 执行后节点 0 的用户界面图

Fig.6 The interface in site 0 as O_1 is executed



图 7 O_2 执行时节点 0 上的交互界面

Fig.7 The interface in site 0 as O_2 is executed

操作 O_2 和 O_3 相互冲突, 因此版本 $V_{\text{rootVersion}}$ 发生复制, 产生 $V_{\text{rootVersion}}$ 的两个复制版本 $V_{\text{rootVersion}_1}$ 和 $V_{\text{rootVersion}_2}$, 而版本 $V_{\text{rootVersion}}$ 变为中间版本, 记录冲突操作执行前的版本状态. 图 9 和图 10 分别显示了并发操作 O_2, O_3 执行后节点 0 和节点 1 上两个子版本的用户视图.

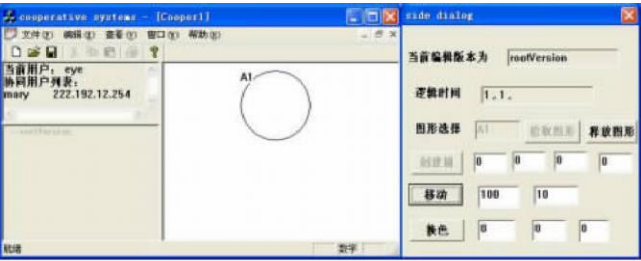


图 8 O_3 执行时节点 1 上的交互界面

Fig.8 The interface in site 1 as O_3 is executed



图 9 O_2, O_3 执行后节点 0 上版本 $V_{rootVersion.1}$ 的用户界面

Fig.9 The interface of $V_{rootVersion.1}$ after O_2 and O_3 are executed

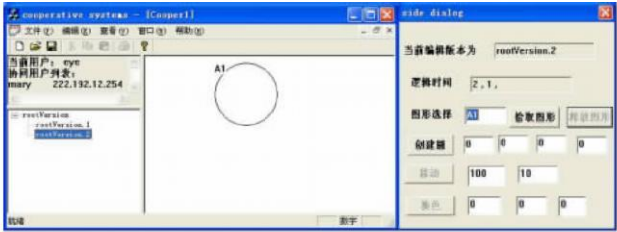


图 10 O_2, O_3 执行后节点 1 上版本 $V_{rootVersion.2}$ 的用户界面

Fig.10 The interface of $V_{rootVersion.2}$ after O_2 and O_3 are executed

3 结语

并发控制是 CSCW 研究中关键技术之一. 本文针对协同图形编辑系统中基于版本复制策略的多版本协同技术, 重点研究了版本树的管理问题, 涉及到版本树优化、版本之间冲突和操作意愿保证, 以及版本之间冲突操作的实现算法. 提出的解决方案已经在原型系统 C o W o r k 中得到了实现. 今后的工作是如何将版本复制并发控制技术应用于现有的 CAD 系统中, 以支持协同设计活动.

[参考文献] (References)

[1] Dou W an feng, Zhu M ing Shen Q i Cooperative multi-versioning technique based on version replication[C] // The 10th International Conf on Computer Supported Cooperative Work in Design, USA: IEEE Press, 2006: 159-164

[2] 沈奇, 窦万峰. 协同多版本技术中原始版本及其表示和效果显示 [J]. 金陵科技学院学报, 2005, 21(1): 19-23
Shen Q i Dou W an feng Expression and effect show of original version in cooperative multi-versioning technique[J]. Journal of Jinling Institute of Technology, 2005, 21(1): 19-23. (in Chinese)

[3] 沈奇, 窦万峰. 协同多版本技术中的中间版本 [J]. 计算机与现代化, 2005, 10: 47-50
Shen Q i Dou W an feng M id version in cooperative multi-version technique[J]. Computer and Modernization, 2005, 10: 47-50. (in Chinese)

[4] Sun C, Chen D. Consistency maintenance in real-time collaborative graphics editing systems[J]. ACM Transactions on Computer-Human Interaction, 2002, 9(1): 1-41.

[5] Xia S, Sun D, Sun C, et al Leveraging single-user applications for multi-user collaboration: the coword approach[C] // Proc of ACM Conf on CSCW. New York: ACM Press, 2004: 162-171.

[6] 朱鸣, 窦万峰. 协同图形编辑系统中改进的多版本技术 [J]. 小型微型计算机系统, 2007, 28(7): 1 318-1 321.
Zhu M ing Dou W an feng An improved multi-versioning technique in the cooperative graphic editing systems[J]. Journal of Chinese Computer Systems, 2007, 28(7): 1 318-1 321. (in Chinese)

[7] 杨君, 窦万峰. 一种新的多版本增创算法 [J]. 计算机学报, 2008, 31(4): 702-710
Yang Jun Dou W an feng A new multiple versions incremental creation algorithm [J]. Chinese Journal of Computer, 2008, 31(4): 702-710. (in Chinese)

[责任编辑: 严海琳]