# 协同图形图像编辑系统中对象锁与区域锁的并行

杨志军[1] 杨建旭[2] 徐向华[2] 惠怀海[2]

(中国电子科技集团公司第五十二研究所 杭州 310012)[1] (浙江大学计算机学院 杭州 310027)[2]

**摘 要** 在分布式协同编辑系统中,为了防止并行操作中出现冲突,常常用到加锁的机制。分布式协同图形图像编辑系统是协同系统中一个特殊的类,因此有特殊的加锁方式。本文提出一种新的不但能够给对象加锁,而且能够给区域加锁的锁机制。在这种锁机制下,用户可以自由地选择给一个图案对象加锁或者给一个工作区域加锁。这是一种细粒度和粗粒度的结合,既有利于减少加锁时系统的负载,又有利于方便用户的操作。这个机制已经应用在我们的CoDesign系统。

**关键词** CSCW,协同图形图像编辑,区域锁,对象锁,伙伴策略锁

## Combination of Object Lock and Region Lock in Object Based Collaborative Graphics/Image Editing Systems

YANG Zhi-Jun[1] YANG Jian-Xu[2] XU Xiang-Hua[2] HUI Huai-Hai[2]

(No. 52 Institute, China Electronics Technology Group Corporation, No. 30, Macheng Road, Hangzhou 310012)[1]

(College of Computer Science, Zhejiang University, Hangzhou 310027)[2]

**Abstract** Locking is a common technique in distributed editing system used to prevent conflicting, maintain user intention and maintain result consistency. Object-based distributed cooperative graphics/image editing systems are a special class, thus have them special locking scheme. In this paper, a novel hybrid scheme for object lock and region lock is proposed. In the proposed scheme, user could freely lock a graphics object or lock a region. Therefore, this is a combination of fine-grain locking and coarse-grain locking, thus not only saving the burden of system used to transfer unnecessary or un-sensitive data, but also making a convenience for users to select. The proposed scheme has been implemented in our CoDesign system.

**Keywords** CSCW, Cooperative graphics/image editor, Region lock, Object lock, Buddy scheme lock

## 1 引言

随着社会分工的深化,人们的协同行为也必然会越来越广泛,而计算机与信息技术的发展,为这些协同提供了交互的平台,这个平台的作用是使得一个群体可以通过互联网在异地协同完成一项共同的任务[3],它给这个群体一种同在一个会议室工作的感觉,其目标是传输各种明显或不明显的参与者的动作[2],例如:声音,手势等等。

CSCW有着广泛的应用领域和市场前景,研究协同工作的机理与体系不仅是有必要的,而且将以一种革新的姿态引领科学与技术的前进,CSCW已经应用到的领域有:军事、工业、协同计算机辅助设计、办公自动化和管理信息系统、医疗、远程教育、电子商务与商业、贸易、金融的应用、电子政务……。

这就是基于Internet的分布式实时协同编辑系统,它可以允许一组用户通过Internet同时观看和编辑同一份文档/图形/图像/多媒体文档[3~5]。

基于Internet的分布式协同编辑系统中,解决并行冲突的方式有些采用了锁,有些不需要用户申请锁[4,6,7]。这些不需要锁的系统大多是被应用到文本编辑系统[4,6],或者在实现过程中仍然存在一些不易解决的问题[7]。而在分布式协同图案编辑系统中,由于图形图像本身的特殊性,为了防止并行操作中出现冲突,锁被广泛的应用。

加锁的目的是为了更好地协同,所以加锁必须从用户的角度考虑操作的方便性和实用性。Chengzheng Sun提出了可选和细粒度的加锁方式,有效地降低了用户锁定的工作量,但这种方式可能更适合文本编辑系统,它不足以满足基于图形图像的编辑系统中用户对于灵活性的要求,也不足以处理图形图像编辑中的复杂情况。

因此,有必要应用一种同时可以处理对象锁[1,8,12]和区域锁[8]的机制。本文提出了把截然不同类型的粗粒度区域锁和细粒度对象锁统一提供给用户,并使用统一的接口的机制。这种机制下对象和区域之间复杂的关系将被细致的讨论。

## 2 CoDesign协同图案编辑背景

### 2.1 CoDesign图形文档数据模型

一个图形文档即一个图案由一系列图层组成,每个图层由一系列图形对象组成,每个图形对象都有一个唯一的标识符(ID)。每个图形对象都有它对应的位置,以及记录它形状、颜色等属性的数据。

### 2.2 CoDesign元操作

文档可以通过以下几种编辑操作来改变状态:1. 新建,2. 移动,3. 改变大小,4. 改变边框颜色,5. 填充颜色,6. 删除。

每个站点维护一张历史列表,记录所有站点的操作。各个操作在不同的站点以不同的次序执行可能产生结果不一致

杨志军 双学位,主要研究方向为图像处理,分布式计算;杨建旭 硕士生,主要研究方向为CSCW,移动数据库,移动搜索引擎等领域;徐向华 博士生,主要研究方向为CSCW,并行计算等领域;惠怀海 硕士生,主要研究方向为CSCW,软件工程等。

的情况，而事实上不同站点的执行结果应该是一致的，不同站点执行操作的历史列表也应该是一致的[3,11]。所以 CoDesign 系统采用以下规则确定列表的顺序：

**定义 1**(状态向量) 给定一个站点 $i$ 和它们状态向量(一个计数器 $C$)，当且仅当在站点 $i$ 的用户做一个元操作的时候，计数器加 1。

**定义 2**(全序关系"=>") 设操作 $O_a$ 和 $O_b$ 分别由站点 $i$ 和站点 $j$ 产生，与操作相关的状态向量分别为 $SV_{O_a}$ 和 $SV_{O_b}$。$SV_{O_a} => SV_{O_b}$，当且仅当，(1) $\mathrm{sum}(SV_{O_a}) < \mathrm{sum}(SV_{O_b})$ 或 (2) $i < j$ 且 $\mathrm{sum}(SV_{O_a}) = \mathrm{sum}(SV_{O_b})$。

CoDesign 系统不同站点执行操作的历史列表中前一项和后一项一定是满足全序关系的。

除了编辑操作，CoDesign 系统还可以进行加锁，解锁的操作。

1. 锁定对象：锁定单个对象或锁定多个对象。

2. 锁定区域：锁定圈定好的矩形区域，包括锁定区域圈定的对象(除去已经被其他用户以区域锁锁定的对象)。

3. 解开对象锁：解开单个或多个对象，或解开区域锁已经锁定的对象。

4. 解开区域锁：解开矩形区域锁以及区域锁定的对象。

## 3 锁的类型与作用

锁的作用是用来防止并行操作中出现冲突，防止用户意图的破坏以及数据的破坏。而又正是"锁"妨碍了协同，一用户加锁，其他用户则不能操作锁定的部分。

锁的类型由不同的分类标准可以有不同的划分。

### 3.1 强制锁与可选锁

传统的方式是，为了避免冲突的发生，用户在操作之前要得到将要编辑的对象的锁，这样用户对任何对象的编辑都由系统强制加锁的方式我们称之为强制锁，强制锁的优点是它能有效地避免冲突的发生，缺点是耗用很大的系统资源，造成响应慢以及用户操作的序列化，延长了等待时间，降低协同编辑的效率。

当多用户在协同编辑器中同时编辑不同区域时，锁的作用是体现不出来的。锁机制在系统里可以是可选的。可选锁[1,8]一定程度上克服了强制锁的缺陷，但增加了系统控制的难度，以及冲突的数量。

CoDesign 系统采用可选锁，这样，当用户真正认为编辑的对象很重要，并且不能被其他用户修改时才去上锁。这样既有利于减少用户的操作负担，又有利于减少系统的负载。

### 3.2 立即锁与非立即锁

传统的锁机制是用户对一个对象加锁后，要等待系统的响应，只有系统确认上锁后才能对试图锁定的对象进行操作。而立即锁[8]，则是不等待系统的响应，上锁后直接对对象进行操作。立即锁是一种高响应的机制，对于一些网络不通畅，系统响应比较慢，或者编辑的数据量比较大的情况能够起到较好的效果。

CoDesign 系统采用立即锁。这是由 CoDesign 系统的结构决定的。CoDesign 系统采用分布复制结构，任何操作都是先在本地执行，之后发送到其他远程站点。当锁定操作没有冲突存在，用户的操作将继续进行下去。当锁定操作存在冲突，这次锁定可能被取消。如果锁的操作被取消，与之相关的操作也相应的被取消。

### 3.3 前锁与后锁

前锁是在对对象操作之前锁定对象。后锁[9]也叫冲突控制锁，在操作一个对象之前，不需要请求锁，如果冲突发生，系统自动上锁。

CoDesign 系统采用前锁与后锁并用，即如果用户对区域或对象加锁，使用的是前锁；如果用户不加锁而操作，发生冲突的时候，由系统来加后锁。这样的复合锁机制综合考虑了方便用户的操作以及降低系统的负载。

### 3.4 对象锁与区域锁

在一个图案设计过程中，用户感兴趣的是本用户的任务部分，用户不期望其他用户更改的也只是自己认为敏感的图形对象，用户将选择关心的图形对象进行加锁。对象锁[1,8,12]是用来锁定图案中的图形对象的锁机制。采用对象锁使得加锁的粒度细化。提高了系统的响应，并且，使得协同用户的聚焦度增强。

如果用户需要同时编辑的对象比较多，或者，用户感兴趣的是一个图案的区域，并且不期望其他用户涉及到这些对象或者区域，用户将选择关心的区域加锁。区域锁[8]是用户对其指定的图案区域加锁，在一个基于图形对象的图案中，锁定一个区域意味着同时锁定了区域包含的对象。其他用户不能进入该区域进行操作，也不能操作区域包含的对象。

CoDesign 系统采用对象锁和区域锁并行的方式。这样细粒度与粗粒度的结合使得用户的选择更为多样，尽管由这种设计增加了系统设计的复杂度，但有效地缓解了锁的独占性与协同的共享性之间的矛盾，以及用户使用的方便性与系统的高负载之间的矛盾。

## 4 处理对象锁和区域锁并行

在 CoDesign 系统中，用户可以自由地选择，选择锁定图形对象或者锁定图案的区域。系统维护一张锁列表，列表里既可以存放对象锁，也可以存放区域锁。而上层提供给用户的接口是统一的加锁，解锁。

### 4.1 半包围协议

**定义 3**(半包围锁定) 给定一个图形对象和一个区域锁，当区域锁的轮廓包围对象的一部分时(如图 1)，图形对象与区域锁二者共同所处的状态叫做半包围锁定。

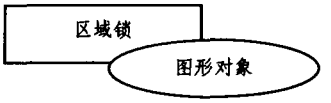


图 1

**定义 4**(全包围锁定) 给定一个图形对象和一个区域锁，当区域锁的轮廓包围对象的全部时(如图 2)，图形对象与区域锁二者共同所处的状态叫做全包围锁定。

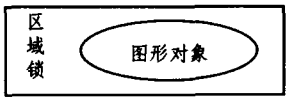


图 2

在一个存在图形对象的图案中，当用户锁定一个区域时，必然要锁定相关的图形对象。那么，锁定哪些相关的对象就成为首先遇到的问题。

处于全包围锁定状态的图形对象很明显是一定要被锁定的，处于半包围锁定状态的对象同样也是与区域锁相关的对象，是否要锁定它们呢？

1. 如果不锁定：锁定区域的效果不能完全体现出来，因

为至少有一部分区域是被一些对象占据，这些对象或者处于未锁定状态，或者被其他用户锁定。这样其他用户就可以操作该对象入侵本区域。

2. 如果锁定：这种方式可能会使得用户锁定的范围过分扩大，妨碍其他用户的操作。并且，也可能将要锁定其他用户或本用户已经锁定的对象。这时的处理参见本文 4.2，4.4 节。

3. 如果只锁定对象的一部分：当一个区域锁，只锁定它包围对象的那一部分，这势必会导致不同用户之间操作的混乱。

**定义 5**(相关包围锁定(区域锁定协议)) 当用户锁定一个区域时，用户不但同时锁定处于全包围锁定状态的对象，并且也同时锁定处于半包围锁定状态的对象，而不是只锁定其一部分。这样的锁定叫做相关包围锁定。

当用户锁定一个区域时，处理相关的图形对象的问题时，CoDesign 系统采用相关包围锁定。

**定义 6**(相关包围对象集合 REOS(Relative Enclosed Objects Set)) 给定一个用户的区域锁，所有与该区域锁处于相关包围锁定状态的图形对象都属于 REOS。

例如，图 3 中区域锁 $Lock^{region1}$ 的相关包围对象集合是：

$\{Object_a, Object_b, Object_c\}$

其中 $Object_a$ 是处于全包围锁定状态的对象，$Object_b$，$Object_c$ 是处于半包围锁定状态的对象。

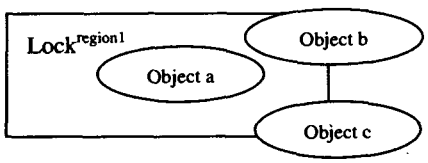


图 3

User $k$ 拥有的锁权限是一个二元组$\{LockSet_k^{region}, LockSet_k^{object}\}$，其中 $LockSet_k^{region}$ 是 User $k$ 拥有的区域锁集合，该集合中的每一个元素由两部分组成，其一是 User $k$ 拥有的区域本身，其二是该区域的相关包围对象集合 REOS；$LockSet_k^{region}$ 的表示形式如下：

$\{[Region_1^k, REOS_{region1}^k(Object_{11}^k, Object_{12}^k, Object_{13}^k \cdots,)], [Region_2^k, REOS_{region2}^k(Object_{21}^k, Object_{22}^k, Object_{23}^k \cdots)] \cdots\}$ 每一项的上标表示这个区域或对象的所有者。

$LockSet_k^{object}$ 是 User $k$ 拥有的独立对象锁集合，这里的独立对象锁所指的是这个对象锁不属于任何一个区域锁。

例如，在图 4 中，User $A$ 拥有以下对象和区域的锁权限，其中 $LockSet_A^{region}$ 是$\{[Region_1^A, REOS_{region1}^A(Object_a^A, Object_b^A, Object_c^A)]\}$，$LockSet_A^{object}$ 是$\{Object_b^A\}$。

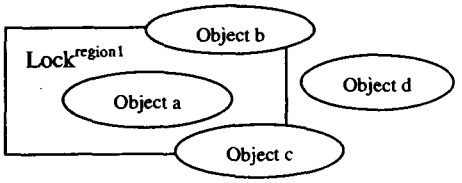


图 4

### 4.2 重复锁定协议

1)当用户锁定一个对象之后又锁定一个区域，而这个区域全包围或半包围这个锁定的对象：系统将提示用户是否把该对象的锁权限加入该区域。这种方式进一步增加了用户操作的灵活度。

2)当用户锁定一个区域之后又锁定另一个区域，而这两个区域是有重叠的部分：允许同一用户锁定重叠的区域。

3)当用户锁定一个区域及其对象，之后用户释放了其中一些对象，一段时间后用户又试图对它们加锁：系统将提示用户或者把该对象的锁权限加入该区域，或者独立为一个对象锁。

4)当用户锁定一个对象之后又锁定该对象：忽略锁定。

5)当用户锁定一个区域之后又锁定该区域：忽略锁定。

### 4.3 限制用户操作协议

锁定就意味着对没有拥有锁权限的用户的操作的拒绝。

1)当一用户锁定一个对象：其他用户不可以对该对象进行 CoDesign 的元操作。

2)当一用户锁定一个区域：其他用户不可以移动对象进入该区域，其他用户新建的对象与该区域不能处于的半包围锁定或全包围锁定状态。其他用户不可以对区域锁定的对象进行 CoDesign 的元操作。其他用户锁定的区域不可以与该区域有相交的部分。

### 4.4 局部释放策略与伙伴协议

用户锁定一个区域，不但锁定了区域本身，也锁定了区域中的对象。这是一种整体锁定的形式。但这种整体锁定会给其他用户带来不必要的阻碍，如果在区域锁中可以释放图形对象的锁权限，将有效地融通不同用户之间的请求。

4.4.1 *伙伴协议 I* 存在这样一种情况，如图 4，用户 $A$ 的区域锁锁定一系列对象(包括 $Object_a$)，用户 $B$ 此时试图编辑 $Object_a$，而 $Object_a$ 的锁权限在 $A$ 的手中，$B$ 期望 $A$ 释放锁。如果 $A$ 同意把 $Object_a$ 的编辑权限交给 $B$，但并不希望释放所有的锁权限。此时，$A$ 可以只释放 $Object_a$ 的权限。

释放后，用户 $A$ 的 $LockSet_A^{region}$ 是$\{[Region_1^A, REOS_{region1}^A(Object_b^A, Object_c^A)]\}$，$A$ 的 $LockSet_A^{object}$ 是$\{Object_b^A\}$。

假设用户 $B$ 在此之前没有拥有任何锁权限，此时用户 $B$ 的 $LockSet_B^{region}$ 是$\{\}$，$B$ 的 $LockSet_B^{object}$ 是$\{Object_a^B\}$，$Object_a$ 的上标由 $A$ 变为 $B$ 表示 $Object_a$ 的所有者以及由 $A$ 变成 $B$。

但是，当 $B$ 真正操作(改变大小，移动)$Object_a$ 的时候却出现了问题。根据限制用户操作协议，无论 $B$ 是否对 $Object_a$ 加锁，$B$ 的操作(改变大小，移动)由于入侵 $A$ 的区域将被限制操作。

针对这种情况，采用伙伴协议 $I$：使 $Object_a$ 成为 $A$ 的伙伴，允许 $Object_a$ 在 $A$ 的区域里自由出入。在 $A$ 的区域锁的属性里面存放一张伙伴列表 BuddyList，当 $Object_a$ 的权限从 $A$ 移交到 $B$ 的时候，把 $Object_a$ 加入 BuddyList。$B$ 操作(改变大小，移动)$Object_a$ 的时候，当检测到操作与现存的区域锁冲突，就在区域锁的 BuddyList 里查找是否有 $Object_a$，如果有，操作被允许。

```
/* Objecta 的权限从 A 移交到 B 的伪代码 */
if(A->RegionLock->RmLock(Object_a) != 0)
{
if (B->ObjectLock->AddLock(Object_a) != 0)
{
  A->RegionLock->BuddyList->Add(Object_a);
}
}
/* 当 B 移动或 Resize Object_a 的伪代码 */
if (B->Move(Object_a)||B->Resize(Object_a))
{
  int result = 0;
  result = Search_BuddyList(Object_a)
      /* 在所有非 B 的区域锁的 BuddyList
         里面搜索 Object_a */
```

```
if(result == 1) /* 如果搜到 */
    pass(B's operation on Object_a);
else
    block(B's operation on Object_a);
```

4.4.2 伙伴协议 II 用户 $A$ 的区域锁已经锁定一系列对象(包括 $Object_a$),用户 $B$ 此时试图锁定另一个区域,而这个区域与 $Object_a$ 处于相关包围锁定状态,如图 5。

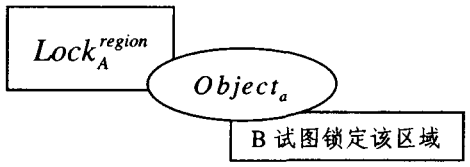


图 5

这种情况存在的问题是:是否允许用户 B 的锁定操作?

a. 不允许。不允许的理由是:$A$ 已经锁定了 $m$,$B$ 的势力范围就不应该涉及到 $m$。

b. 允许。允许的理由是:有可能与 $A$ 处于相关锁定状态的对象很多很大,这样,$A$ 的区域锁过分妨碍了其他用户的操作,这不是真正意义上的协同,应该允许 $B$ 的锁定。

综合以上情况的分析,允许 $B$ 的锁定是比较合理的,然而这就需要处理一个问题:如果 $B$ 能够锁定,$m$ 的锁权限归哪个用户?

a. 如果 $Object_a$ 归 $B$,那么 $A$ 对 $Object_a$ 的锁定形同虚设。

b. 如果 $Object_a$ 被 $A$ 与 $B$ 共享,这将会严重影响用户的意图保持[3,10,11]。

c. 由于 $Object_a$ 首先被 $A$ 锁定,$Object_a$ 的锁权限归属于 $A$ 是比较合理的选择。

系统选择 $c$:$Object_a$ 的锁权限归属于 $A$。此时需要处理另一个问题:如果 $Object_a$ 归 $A$,当 $A$ 操作 $Object_a$ 移动时,必然会触犯到 $B$ 的区域,那么 $Object_a$ 是否可以移动?由于 $A$ 是第一个对 $Object_a$ 加锁的用户,同时也是当前对 $Object_a$ 加锁的用户,$A$ 拥有 $Object_a m$ 的优先级应该是最高的。这个分叉点上合理的选择是:允许 $A$ 操作 $Object_a$ 在 $B$ 中移动。

如何让 $A$ 操作 $Object_a$ 在 $B$ 的区域移动?因为此时 $Object_a$ 侵犯了 $B$ 的区域(即违反限制用户操作协议)。针对这个问题,采用伙伴协议 II:在伙伴协议 I 的基础上(即令 $Object_a$ 成为 $B$ 的 $Buddy$)加入对 $Object_a$ 的可等待机制。

由于其他用户对于被 $A$ 锁定的 $Object_a$ 是处于等待锁定状态,当 $A$ 释放锁的时候,有必要把 $Object_a$ 的锁权限交给下一个申请 $Object_a$ 的锁权限的用户。可以使用的策略有两种:

1)在 $Object_a$ 的属性里设置等待队列,当 $B$ 锁定区域时,虽然锁定 $Object_a$ 没有成功,但是,系统已经把 $B$ 加入了 $Object_a$ 的等待队列。当 $A$ 释放锁的时候,只要 $Object_a$ 的等待队列不空,就把 $Object_a$ 的锁权限转移给下一个对待的用户。当 $B$ 解锁的时候,如果此时 $B$ 还没有拥有 $Object_a$ 的锁权限,在 $Object_a$ 的等待队列里删除 $B$。

2)由于系统中记录这种对象锁和区域锁的杂交的链表是按时间顺序记录的,因此,当 $A$ 解锁之后,在链表中查找位于 $A$ 的区域锁之后的锁节点,找到第一个与 $Object_a$ 处于相关包围锁定状态的区域锁(如果存在的话),把 $Object_a$ 加入到这个区域锁的伙伴列表。用这种策略不存在以下情况的处理,即 $B$ 解锁时 $A$ 还没有解锁。而这种情况在策略 1 中是需要后续处理的。

综上,策略 1、2 都是可行的方式,策略 1 会增加网络的负担,而策略 2 会增加本地查找的时间。CoDesign 系统采用了策略 2。

**结束语** 本文中,阐述了一种新的基于对象的协同图案编辑系统中处理并行冲突的加锁机制,即同时应用对象锁与区域锁,并提出了相应有效实用的解决方法。我们从方便用户操作的角度开始,提出了把截然不同的粗粒度区域锁和细粒度对象锁统一提供给用户,并使用统一的接口的机制。接着,一种处理区域与对象之间存在的锁定冲突的伙伴策略被提出,这种策略从本质上不同于共享锁策略。

协同的图案编辑在纺织印染以及机械,建筑等领域有广泛的前景。然而传统的处理并行冲突的方式却显得过于单调,这种在 CoDesign 系统中的杂交锁的机制的应用对于分布式协同的其他领域有着借鉴的作用。

## 参 考 文 献


1 Sun Chengzheng. Optional and Responsive Fine-Grain Locking in Internet-Based Collaborative Systems. Proc of the IEEE trans. on parallel and distributed systems, SEP 2002,13(9)

2 Greenberg S, Roseman M, Webster D, et al. Issues and experiences designing and implementing two group drawing tools. In: Proc. of Hawaii Intl. Conf. on System Sciences, Kuwaii, Hawaii, IEEE Press, VOL. 4, January, 1992. 138～150

3 Sun C, Jia X, Zhang Y, Chen D. Achieving convergence, causality- preservation, and intention- preservation in real-time cooperative editing systems. Proc. of ACM Trans. on Computer-Human Interaction, March 1998,5(1):63～108

4 Ellis C A, Gibbs S J. Concurrency Control in Groupware Systems, In: Proc. of ACM SIGMOD Conf. Management of Data, 1989. 399～407

5 Ellis C A, Gibbs S J, Rein G L. Groupware: Some Issues and Experiences. Proc. of Comm. ACM, 1991, 34(1):39～58

6 Ionescu M, Dorohonceanu B, Marsic I. A Novel Concurrency Control Algorithm in Distributed Groupware. In: Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, NV, June 2000. 1551～1557

7 Sun Chengzheng, Chen D. A Multi-version Approach to Conflict Resolution in Distributed Groupware Systems. In: Proc. of the 20th IEEE Intl. Conf. on Distributed Computing Systems, April 10-14, 2000. 316～325

8 Chen D, Sun Chengzheng. Optional and responsive locking in distributed collaborative object graphics editing systems. Proc. of ACM SIGGROUP, 1999, 20(3): 17～20

9 Xue L, Zhang K, Sun C. Conflict control locking in distributed cooperative graphics editors. In: Proc. of the 1st Intl. Conf. on Web Information Systems Engineering (WISE 2000), HongKong, IEEE CS Press, June 2000. 401～408

10 Ellis C A, Gibbs S J. Concurrency control in groupware systems. In: Proc. of the ACM SIGMOD Conf. on Management of Data, Seattle, WA, USA, May 1989. 399～407

11 Li D, Zhou L, Muntz R R. A new paradigm of user intention preservation in real-time collaborative editing systems. In: Proc. of the Seventh Intl. Conf. on Parallel and Distributed Systems, Iwate, Japan, July, 2000

12 Newman-Wolfe R E, Webb M L, Montes M. Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor. In: Proc. of ACM on CSCW 92, November 1992, Toronto, Ontario, Canada, 1992. 265～272