UM SUPORTE PARA APLICAÇÕES DE TRABALHO COOPERATIVO DO TIPO EDITOR DISTRIBUÍDO

Udo Fritzke Jr.
e-mail: fritzke@lcmi.ufsc.br

Jean-Marie Farines
e-mail: farines@lcmi.ufsc.br

UFSC - Universidade Federal de Santa Catarina
LCMI - Laboratório de Controle e Microinformática
Centro Tecnológico - Caixa Postal 476
Campus Universitário
88049-900 - Florianópolis - SC - Brasil

RESUMO

Este trabalho apresenta, após uma breve introdução sobre o tema CSCW (*Computer-Supported Cooperative Work)*, as características relevantes e o problema de controle de concorrência em uma aplicação cooperativa do tipo editor distribuído, onde um conjunto de usuários compartilha interativamente objetos replicados (tais como figuras geométricas e anotações de texto que compõem um documento em edição). Em seguida define uma plataforma de serviços para o suporte a este tipo de aplicação e discute uma proposta de implementação desta plataforma, com a utilização do sistema ISIS. Desta forma, o modelo de implementação proposto apresenta a utilização das facilidades de gerenciamento de grupo e *multicasts* causalmente ordenados providos pelo ISIS no suporte para aplicações cooperativas do tipo editor distribuído.

ABSTRACT

After a brief introduction about CSCW, this paper presents the meaningful characteristics and the concurrency control problem in a cooperative application as a distributed editor in which a set of users share interactively replicated objects (as geometric objects and text annotations which compose a document under edition). In the sequence it will be defined a service platform to the support for this kind of application and discussed an implementation proposal for this platform, with the utilization of the ISIS system. In this way, the proposed model presents the utilization of group management facilities and causal ordered multicasts provided by the ISIS system to implement the support of a distributed editor.

1. INTRODUÇÃO

As tecnologias recentes para comunicação (p.ex. redes de alta velocidade), manipulação, processamento e armazenamento de novos tipos de informações tais como áudio, voz, vídeo e imagem têm impulsionado o desenvolvimento e a difusão do Trabalho Cooperativo para um conjunto cada vez maior de aplicações, permitindo dessa forma o aproveitamento de esforços individuais para realizar uma atividade ou um objetivo comum a um grupo de pessoas. A realização dessa atividade compartilha habitualmente o mesmo ambiente computacional e neste caso, o termo utilizado é Trabalho Cooperativo com Suporte Computacional (*Computer-Supported Cooperative Work* - CSCW). Neste artigo, procura-se definir o suporte para uma atividade de trabalho cooperativo específica, a edição distribuída, e analisar alternativas para a implementação deste.

Neste artigo, são apresentados inicialmente alguns aspectos importantes do Trabalho Cooperativo tais como definições, classificações e características funcionais. A seguir, discute-se ao visto destes aspectos, os requisitos exigidos para o suporte de aplicações de Trabalho Cooperativo. Escolhida uma aplicação de edição distribuída como exemplo de aplicação de

Trabalho Cooperativo, são apresentadas as características funcionais do editor e os diversos métodos de controle de concorrência adequados a este tipo de aplicação. Nos itens seguintes, define-se uma plataforma de serviços para o suporte a um editor distribuído e discute-se a implementação deste suporte utilizando o sistema ISIS; a partir deste modelo de implementação, apresenta-se a utilização das características de gerenciamento e comunicação de grupo do ISIS para implementar a plataforma de serviços proposta e os mecanismos a serem introduzidos para o controle de concorrência. Concluindo este artigo, a proposta de suporte é confrontada com outras propostas existentes e são apresentadas as perspectivas futuras de trabalho.

2. TRABALHO COOPERATIVO

2.1 Os Aspectos Temporal e Espacial dos Sistemas de Trabalho Cooperativo

Com objetivo de classifica-las, as aplicações de Trabalho Cooperativo vem sendo analisadas segundo dois aspectos: o aspecto temporal que define as formas de cooperação (ou dos tipos de interação) possíveis e o espacial que corresponde a disposição geográfica dos participantes [Rodden92] [Ellis91].

Formas de cooperação: as formas pelas quais os usuários interagem em um grupo são de dois tipos: síncrona ou assíncrona. A interação síncrona requer a presença simultânea de todos os participantes em uma sessão. Já a interação assíncrona ocorre em um intervalo de tempo maior, e não requer a presença simultânea dos participantes. Existem também aplicações que misturam ambos os tipos de interações, como no caso de sistemas de conferência assíncronos que também suportam interações síncronas com alguns tipos de mídias (como voz ou texto, p. ex.).

Disposição geográfica: os usuários podem estar geograficamente distribuídos de várias maneiras. Todos no mesmo local (*co-located*), interagindo por uma projeção de tela (com o assunto principal, p. ex.), e com suas máquinas interligadas por uma rede local. Em salas diferentes mas próximas (*virtually co-located*), interagindo por uma rede local. Em salas distintas mas distantes (*locally remote*), interagindo também por uma ou mais redes locais, interligadas por uma rede do tipo *back-bone* (p. ex., um campus universitário). Em lugares distantes (*remote*), suportados por serviços de telecomunicações.

2.2 Classificação dos Sistemas de Trabalho Cooperativo

Do ponto de vista funcional as aplicações de Trabalho Cooperativo dividem-se, seguindo as classificações de [Rodden92], [Ellis91] e [Williams94], em quatro classes:

Sistemas de Mensagens: baseiam-se na troca de mensagens de texto de forma assíncrona entre usuários. Os sistemas de mensagens serviram inicialmente para interação entre usuários de uma mesma máquina, mas atualmente, com o suporte de redes de longa distância, permitem a troca de mensagens entre usuários de máquinas distintas remotas. Os sistemas de mensagens são padronizados pelo CCITT, a partir da série de padrões X400 (*Message Handling System* - MHS).

Sistemas de Conferência por Computador: As conferências correspondem a troca de mensagens (textual ou multimídia) por um conjunto de pessoas localizadas em ambientes diferentes e que cooperam de forma assíncrona. As mensagens submetidas à conferência, são controladas por um ou mais moderadores. Conversações são entendidas como grupos de mensagens com ligações de comentários (como p. ex., respostas ou comentários a uma mensagem prévia). Controle de acesso à conferência, organização e controle de conversações

(aprovação de novas mensagens, remoção das obsoletas) são algumas das atividades do moderador. O sistema SuperKOM [Palme92] é um exemplo deste tipo de aplicação.

A disponibilidade de redes de alto desempenho permitem a existência de sistemas de conferência tempo-real (*real-time conferencing*) com interações síncronas entre usuários remotos, como é o caso no sistema RTCAL (*Real Time Calendar*) [Sarin85]. Adicionalmente, sistemas de conferência (*desktop conferencing system*) como o MERMAID [Watabe90], permitem o uso de mídias como imagem, voz, gráficos e desenhos a mão livre.

Sistemas de Tomada de Decisão: Estes sistemas visam aumentar a produtividade no processo de tomada de decisão. A forma de cooperação é síncrona e a localização pode ser do tipo reunião face-a-face ou em sítios diferentes. Mecanismos de controle de acesso e de votação de cada usuário são necessários nestes sistemas. O sistema Colab [Stefik87] é um exemplo deste tipo de aplicação.

Sistemas de Co-autoria e Argumentação: Estes sistemas permitem auxiliar a autoria de documentos cooperativos (de texto ou com recursos multimídia) e, suportar a negociação e a argumentação entre envolvidos em trabalho cooperativo. Editores cooperativos (como p. ex. os sistemas CoDraft [Kirsche93] e GROVE [Ellis91]), podem ser considerados como sistemas de co-autoria. O tipo de cooperação é geralmente síncrona e a localização geográfica vai de face-a-face a remota; são ainda utilizados mecanismos de controle e votação dos usuários.

2.3 Características Funcionais das Aplicações Cooperativas

As aplicações cooperativas, segundo as classes abordadas, apresentam as seguintes características funcionais:

- Os grupos de usuários cooperam de forma estreita. Em uma sessão de trabalho cooperativo (como p. exemplo, no caso de uma conferência por computador, reunião eletrônica, e co-autoria de um documento), os participantes interagem intimamente e de forma **coordenada** (não destrutiva) sobre recursos compartilhados (como p. exemplo, um canal de comunicação de voz, um documento compartilhado, etc.), mesmo em distâncias maiores.

- Os grupos de usuários são dinâmicos. Os participantes entram em uma sessão de trabalho cooperativo segundo políticas pré-estabelecidas (convites, requisições de entrada, etc.) e saem espontaneamente ou por motivo de falha no sítio em que se encontram. Ao entrar, um participante deve poder obter todas as informações consideradas importantes sobre a sessão, para poder participar do trabalho nas mesmas condições que seus colegas (transferência de estado).

- Os tempos de resposta em relação as ações dos usuários devem ser baixos. Este tempo de resposta apresenta-se como a combinação dos seguintes atrasos: tempo de resposta local (i.e. atraso entre a ação de modificação de um objeto até sua apresentação na interface local) e tempo de notificação (i.e. atraso entre a ação de modificação de um objeto até sua apresentação em todos os outros sítios).

- As mensagens de participantes diferentes podem estar interrelacionadas. É o caso por exemplo em conferência por computador (mesmo em conferências assíncronas) de mensagens enviadas por dois participantes que podem estar semanticamente ligadas (como p.ex., uma resposta/comentário a uma mensagem prévia).

Estas características levam à definição de requisitos a serem atendidos pelo suporte das aplicações cooperativas.

3. SUPORTE PARA APLICAÇÕES DE TRABALHO COOPERATIVO

Neste trabalho, adota-se uma arquitetura geral de um sistema cooperativo em três níveis (Figura 1), conforme definido em [Rodden92] e onde cada nível oferece serviços ao superior. O Suporte para o Trabalho Cooperativo, objeto de nosso estudo, contem um conjunto de serviços e facilidades que permitam atender aos requisitos exigidos para que as aplicações cooperativas possam se executar sobre um sistema distribuído.

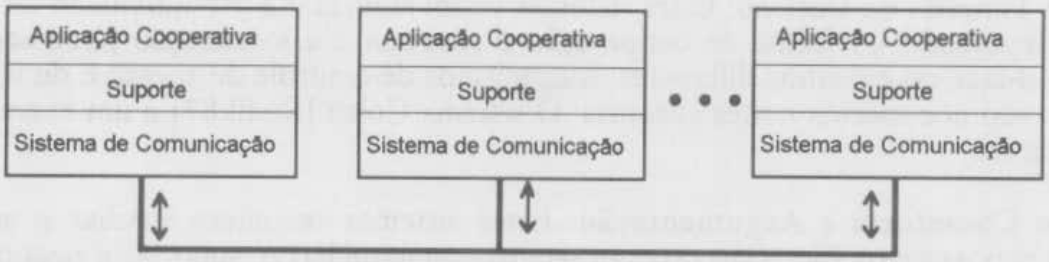

Figura 1 - Arquitetura geral de sistemas cooperativos

Como já foi visto, nas aplicações de trabalho cooperativo, diversas formas de interação entre os usuários são possíveis desde a atuação assíncrona sobre objetos compartilhados (como uma base de dados de agenda de reuniões, p. ex.), até sistemas altamente síncronos onde a comunicação se dá através de múltiplos canais de comunicação de mídias contínuas e discretas (como em *real-time conferencing*). A combinação desses diversos aspectos define a complexidade da aplicação cooperativa, e conseqüentemente do suporte desta.

Em particular o suporte deverá conter serviços que permitirão atender as exigências das aplicações nas questões que seguem.

Comunicação de Grupo: As aplicações de trabalho cooperativo impõem que o suporte tenha facilidades para comunicação de grupo tais como:

- o uso de endereçamento de grupo, descarregando o programador de detalhes de endereçamento dos processos individuais que compõem o grupo;
- a liberação única das mensagens para o grupo (sem perda ou duplicação);
- em algumas aplicações, a liberação ordenada das mensagens para o grupo;
- a manutenção de históricos de eventos de uma sessão de trabalho cooperativo, para recuperar o trabalho em caso de faltas ou para informar algum novo participante da sessão a respeito do trabalho até então realizado;
- obtenção de informações da composição do grupo (*membership*), pois este tipo de informação pode ser útil para o andamento do trabalho cooperativo (como por exemplo, saber quem entrou ou saiu do grupo). As alterações de *membership* porém, devem ser percebidas pelos usuários de forma consistente, isto é de forma ordenada com relação às mensagens de difusão (p. ex., o envio de uma mensagem de difusão para um grupo com alteração de *membership* em andamento, deve ser liberada para a composição antes da alteração ou após a alteração, mas não para um conjunto intermediário de processos).

Requisitos de tempo real: os requisitos de tempo-real aumentam preponderantemente com o grau de sincronismo nas interações entre os usuários, e com a utilização de mídias contínuas nestas interações.

Em interações síncronas, requer-se que o trabalho cooperativo não torne o ritmo individual de cada membro mais lento, do que se eles estivessem trabalhando de forma não integrada. Conseqüentemente, o sistema deve reagir de forma suficientemente rápida, aos eventos globais disparados pelas atividades dos usuários. Por exemplo, cada usuário de um editor de textos cooperativo, ao ser modificado um texto ou um desenho, deve perceber essa modificação no contexto do documento global o mais rápido possível, para que cada um possa prosseguir com seu trabalho de criação.

Além disso, a natureza de tempo-real das mídias contínuas (voz, vídeo, áudio, p. ex.), reflete-se em requerimentos temporais do suporte mais estritos que no caso de mídias discretas. As mídias contínuas são mais sensíveis a atrasos de transporte entre sua geração e apresentação (normalmente remotas); são necessários a nível de suporte, mecanismos de sincronização intra-mídia que procuram minimizar os efeitos destes atrasos na qualidade de apresentação dessas mídias. Em algumas aplicações ainda, torna-se necessário sincronizar mídias contínuas relacionadas entre si, como p. ex. num sistema de teleconferência em tempo-real no qual a sincronização da voz e da imagem (sincronização de lábios - *lip sync*) é necessária. Desta forma, o suporte deve oferecer recursos de comunicação e sincronização compatíveis com as características temporais das mídias contínuas, normalmente quantificadas através de parâmetros negociáveis de qualidade de serviço (*Quality of Service - QoS*) tais como vazão, atraso fim-a-fim, taxas de perdas de pacotes ou bits, e prioridade.

Tolerância a faltas: Além de voluntária, a saída de um membro de um grupo de trabalho cooperativo pode ser devido a falha do seu sítio. O suporte deve ter mecanismos que permitam recuperar as falhas. Por exemplo, quando o emissor de uma mensagem de difusão falha antes que todos os receptores no grupo a tenham recebido, é necessário que se garanta que todos tenham a mesma visão global (com um outro sítio assumindo as vezes do sítio que falhou e continuando a execução da difusão interrompida).

4. EDITOR DISTRIBUÍDO

Neste trabalho, pretende-se estudar e discutir os suportes para aplicações cooperativas. Tendo em vista a abrangência desse assunto, limitou-se na especificação e implementação de um suporte para uma aplicação especifica de trabalho cooperativo da classe de sistemas de coautoria: um editor distribuído. Três razões levaram a esta escolha: permitir uma definição clara dos serviços de suporte necessários para a aplicação escolhida; ganhar experiência com as características e requisitos de aplicações cooperativas; avaliar o desempenho dos serviços de suporte e mecanismos de coordenação implementados.

4.1 Características de um Editor Distribuído

No editor distribuído escolhido, mais de um usuário pode editar um mesmo documento (textos, gráficos e desenhos) simultaneamente e de forma coordenada (cooperação síncrona), independentemente da localização geográfica em que se encontra cada um deles, através de uma interface compartilhada.

As principais funcionalidades apresentadas por esse editor, relacionadas com aspectos de cooperação são:

- controle de uma janela de edição comum (*common drawing board*), que consiste em uma área de tela compartilhada por um grupo de usuários de instâncias distintas do editor, na qual se realiza o trabalho de edição cooperativa;

- controle de um cursor de mouse distribuído (*telepointer*), o qual quando invocado, permite a um usuário replicar o movimento do seu mouse para as janelas de edição comum dos demais usuários participantes da sessão cooperativa;

- mecanismos de garantia da consistência dos objetos da janela de edição comum, em casos de tentativas de acesso concorrente sobre um mesmo objeto, como p. ex., alteração de uma mesma figura geométrica por mais de um membro do grupo simultaneamente (controle de concorrência);

- opções de convite, expulsão, pedido de participação e pedido de saída de uma sessão de trabalho cooperativo;

- manipulação de documentos armazenados em arquivos para a edição na janela de edição comum;

- modo de edição cooperativa ou individual (*stand-alone*).

Em aplicações desta classe, a simetria deve ser mantida. Todos os membros devem manter réplicas consistentes dos objetos (p.ex. figuras geométricas) que compõem o documento sendo editado, sendo que o acesso a esse documento não é restringido por algum usuário de maior hierarquia (como um moderador, p. ex.). Além do mais, todos os usuários devem ter os mesmos direitos de acesso, podendo entrar e sair de uma sessão, expulsar alguém e convidar alguma nova instância do editor a participar dessa.

Outrossim, os membros de uma sessão atuam de forma autônoma no acesso ao documento compartilhado, desde que operações conflitantes de membros distintos não levem o documento a um estado inconsistente. Isto significa que qualquer usuário pode escrever/desenhar em qualquer parte da janela de edição comum, a qualquer momento.

4.2 Controle de Concorrência

Em editores distribuídos, os usuários interagem de forma síncrona a partir de sítios distintos, possivelmente fazendo alterações concorrentes sobre um mesmo objeto (Figura 2). Neste caso, para que se garanta a consistência global dos objetos sendo editados, é necessário algum mecanismo de controle de concorrência como forma de coordenação do trabalho cooperativo.

Porém, a coordenação de atualizações concorrentes sobre os objetos deve ser feita, na medida do possível, sem que os usuários se sintam impedidos ou embaraçados na sua cadência de trabalho. Em outras palavras, os sistemas devem ter tempos de resposta aceitáveis para as ações dos usuários.

A implementação de um editor distribuído, deve então estabelecer um compromisso entre formas de coordenação das atividades concorrentes e o tempo de resposta imposto pelo sistema. Tanto uma aplicação sem coordenação (i.e. mais responsiva), onde objetos são modificados sem nenhum compromisso com a semântica das ações executadas sobre os mesmos, como um sistema estritamente coordenado (porem muito lento), são indesejáveis.

4.2.1 Formas de Controle de Concorrência em Editores Distribuídos

Existem diversos métodos de controle de concorrência para aplicações do tipo editor distribuído, desde métodos sem nenhuma coordenação das atividades dos usuários até controle de concorrência estrito (*strict floor control*). A seguir são abordadas as principais classes de métodos de controle de concorrência propostos para este tipo de aplicação.

Métodos sem nenhum controle de concorrência. Nestes métodos apresentados em [Stefik87], as atualizações em uma cópia de um objeto compartilhado são simplesmente difundidas para os sítios que contém as outras cópias, e imediatamente executadas por ocasião das recepções. Se dois usuários difundem atualizações simultaneamente, podem ocorrer situações de disputa para ver qual atualização será executada primeiro (*race conditions*) (Figura 2), o que pode levar a cópias inconsistentes entre os objetos.

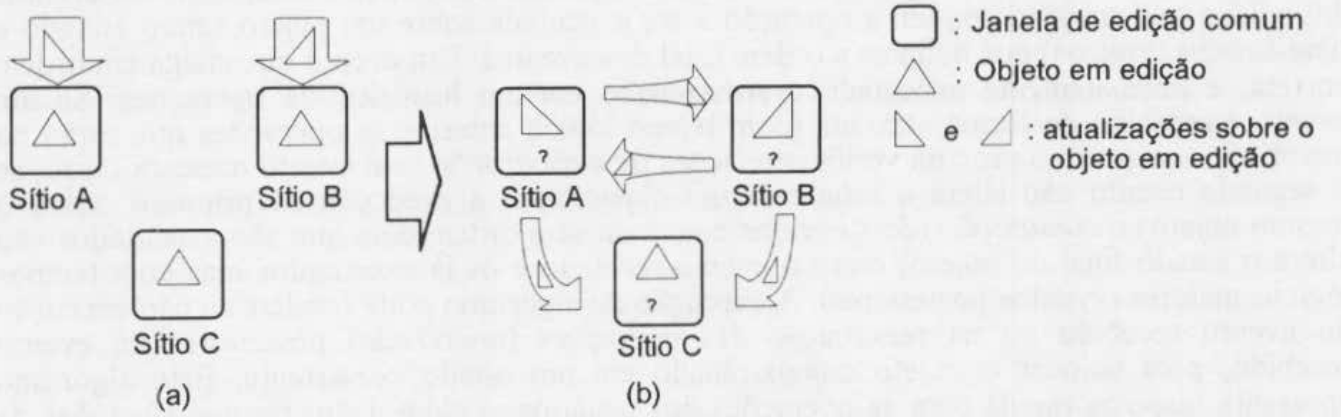

Figura 2 - Atualizações concorrentes de um objeto replicado: (a) fase de atualização das cópias locais; (b) fase de difusão das atualizações para os sítios restantes, onde ocorre conflito entre atualizações concorrentes sobre um mesmo objeto.

Métodos com detecção de operações conflitantes. Estes métodos baseiam-se na alteração local e difusão assíncrona de atualizações de objetos, de forma semelhante aos métodos sem controle de concorrência. Desta forma, a qualquer momento um usuário pode alterar um determinado objeto e notificar aos demais usuários suas operações. A consistência do objeto replicado, porém, é garantida posteriormente, através de mecanismos de detecção de operações conflitantes (que levam o objeto replicado a um estado inconsistente) apropriados, e a correção do estado do objeto.

Em [Sarin85], propõe-se que as atualizações recebidas sejam imediatamente executadas, porém de uma forma reversível, ou seja, de maneira que possam ser desfeitas posteriormente (em caso de conflito). Para detecção de conflito, é sugerido um sequenciador centralizado, que ordene globalmente as atualizações, ou algum esquema baseado em carimbos de tempo (*time-stamps*) nas mensagens de atualização. A execução imediata das atualizações recebidas permite que o sistema tenha uma reação rápida às operações dos participantes; porém a possibilidade de reversão da execução de operações previamente executadas, pode ser desconfortável para os participantes.

Em [Stefik87] o processo de atualização dos objetos também permite a reversão de execução de operações. A cada objeto replicado é associado um carimbo, com o autor e o tempo da última atualização, sendo que toda vez que um objeto é modificado, um novo carimbo lhe é associado. A cada difusão de uma atualização, são enviados além do carimbo atual, o carimbo da versão anterior no sítio de origem. O conflito entre atualizações é detectado quando o sítio que recebe a alteração verifica que o carimbo da versão anterior não é igual ao carimbo da sua cópia local. Isto significa que o sítio que fez a atualização (ou o que a recebeu) não está com sua cópia do objeto em um estado consistente (p.ex. algum deles, ou ambos, ainda não recebeu uma atualização anterior). A resolução de conflito é então realizada com intervenção humana, por exemplo suspendendo temporariamente as atividades, até que o conflito seja resolvido. Este método apresenta também baixos tempos de resposta para operações não conflitantes, entretanto a correção manual de conflitos faz com que a garantia de integridade dos objetos compartilhados seja vulnerável a erros dos usuários.

Em [Ellis91], é proposto um método similar ao anterior, porém com a resolução automática do conflito entre as operações. A difusão de uma atualização é acompanhada por um vetor de

estado indicando quantas operações (recebidas de outros sítios) o sítio de origem executou anteriormente. Cada sítio possui um vetor de estado local que é comparado com o recebido. Se forem iguais, a operação é imediatamente executada, senão, é transformada antes de sua execução, de forma a não levar a cópia local a um estado inconsistente. Esta transformação é função do tipo de operação e do histórico de operações já executadas.

[Karsenty93] apresenta um algoritmo (ORESTE -*Optimal RESponse TimE*) onde os eventos difundidos por um sítio contém a operação a ser executada sobre um objeto sendo editado e *time-stamps* (lógicos) que definem a ordem total dos eventos. Um evento que chega em ordem correta, é imediatamente executado e armazenado em um histórico de operações. Se um evento é recebido de forma atrasada (com tempo lógico anterior às operações que estão no histórico), o algoritmo procura verificar relações de *mascaração* (um evento mascara outro, se o segundo evento não altera o estado de um objeto após a execução do primeiro, sobre o mesmo objeto) e *comutação* (dois eventos comutam se a ordem com que são executados não altera o estado final do objeto) entre o evento recebido e os já executados mas com tempos lógicos maiores (eventos posteriores). A execução do algoritmo pode resultar na não execução do evento recebido ou na reexecução das operações (*undo/redo*) posteriores ao evento recebido, para colocar o objeto compartilhado em um estado consistente. Este algoritmo apresenta resposta rápida para as operações dos usuários, e além disto, resolve situações de conflito de forma automática. Segundo [Karsenty93], do ponto de vista algorítmico este método é mais simples do que as transformações de operações realizadas em [Ellis91].

Métodos de controle de concorrência estrito. Com os métodos que se baseiam no controle de concorrência estrito, cada participante ao atualizar um determinado objeto da janela de edição comum, bloqueia primeiramente o acesso dos outros participantes sobre este objeto, de forma a impedir a execução de atualizações concorrentes sobre o mesmo. Após o bloqueio do objeto, as atualizações são difundidas de forma a manter todas as réplicas do objeto em um estado consistente.

Em [Sarin85], um protocolo de resincronização garante o controle de concorrência sendo que em um dado instante, um único sítio é a fonte de atualizações sobre um objeto e que todos os sítios processam todos os comandos deste antes de começar a receber os comandos de um outro sítio. Em [Stefik87], são apresentados dois modelos para o controle: o modelo bloqueio-centralizado, onde existe um servidor de travas (*locks*) centralizado, e o modelo travas-errantes (*roving-locks*), onde as travas são distribuídas por vários sítios, concedendo-se o controle desta pelo último usuário que atualizou o objeto (diminuindo o tempo de alocação de travas, em situações onde um mesmo usuário acessa com maior frequência um determinado objeto). No CoDraft [Kirsche93], o controle de concorrência se baseia em um mecanismo de votação, através do qual um determinado participante procura obter inicialmente um consenso entre todos os participantes, com relação ao bloqueio de um objeto, para então efetuar as alterações sobre o mesmo.

A seguir, analisam-e as vantagens e desvantagens das três classes de métodos acima apresentadas. O baixo tempo de resposta apresentado pelas duas primeiras classes, é uma característica fortemente desejada, porém o excesso de conflitos entre operações concorrentes sobre objetos compartilhados pode aumentar substancialmente esse tempo e tornar a execução da aplicação confusa, devido a freqüente requisição dos usuários para a resolução de conflitos ou às reversões automáticas de operações já executadas. Nestes casos, os métodos baseados em controle de concorrência estrito tornam as operações dos participantes mais coordenadas. Além disto, nesta classe, o bloqueio de um objeto por um determinado participante, evita que sequências interrelacionadas de atualizações de objetos sejam interrompidas por atualizações de outros participantes (como p. ex., diminuição/destruição de uma figura enquanto outro participante a está aumentando). Em contrapartida, o processo de exclusão mútua sobre um objeto pode se extender indefinidamente, devido a liberdade que os participantes têm em retê-la, enquanto estiverem atualizando um determinado objeto.

Esta discussão deve considerar também a granularidade com que se definem os objetos compartilhados. Os métodos baseados em controle de concorrência estrito, tornam-se menos restritivos a medida que a granularidade dos objetos aumenta (com a subdivisão de objetos, p. ex.), pois diminui a probabilidade de dois ou mais participantes necessitarem atualizar o mesmo objeto; desta forma, em uma mesma figura, pode-se permitir acessos concorrentes sobre atributos que são independentes semanticamente (p.ex. o movimento do objeto na tela e a alteração de sua cor). O algoritmo ORESTE, quando verifica as propriedades de comutação e mascaração entre operações, considera implicitamente, para editores gráficos, uma granularidade a nível de atributos dos objetos.

5. UMA PROPOSTA DE SUPORTE PARA UM EDITOR DISTRIBUÍDO

Neste item, são apresentadas as funcionalidades do suporte para aplicações de trabalho cooperativo do tipo editor distribuído. A proposta de plataforma de comunicação apresentada em [Kirche93] serviu como fonte de inspiração para definir as características do suporte proposto, haja visto que os serviços desta nos pareceram os mais adequados para o editor gráfico distribuído que nos serve de exemplo neste trabalho. Num próximo item, discutir-se-á a implementação do suporte proposto.

As principais funcionalidades providas pelo suporte proposto podem ser agrupadas da seguinte forma: serviço de gerenciamento de grupo, serviço de comunicação de grupo, serviço de transferência de arquivos multi-destino, e serviço de voto.

5.1 Serviço de Gerenciamento de Grupo

O serviço de gerenciamento de grupo provê as funcionalidades para manutenção dos grupos de instâncias de aplicações. Abaixo são apresentadas as primitivas do serviço de gerenciamento de grupo:

- *CreateGroup:* Cria um novo grupo, adicionando automaticamente o iniciador do serviço. Retorna um identificador único do grupo e associa uma senha para autenticação dos membros do grupo em operações de gerenciamento subseqüentes.
- *Invite:* Permite a um membro de um grupo já criado, convidar novas instâncias para participar da sessão. Se a instância convidada aceita o convite, ela é inserida como membro do grupo e o membro que a convidou lhe repassa o estado atual da sessão de trabalho cooperativo.
- *JoinRequest:* Permite a instâncias que não fazem parte de nenhum grupo, solicitar a permissão de entrada em um grupo a algum membro deste grupo. Se o membro solicitado aceita a inserção da instância, este transfere o estado atual da sessão à instância requisitante.
- *GetGroupMembers:* Permite a algum membro de um grupo requisitar a lista atual de membros do grupo a que pertence.
- *Leave:* Permite a qualquer membro de um grupo abandonar este grupo de forma unilateral.
- *Expel:* Permite a um membro solicitar a saída de outro membro do grupo.

5.2 Serviço de Comunicação de Grupo

O serviço de comunicação de grupo (*multicast*) deve permitir a difusão de mensagens a todos os membros de um grupo com apenas uma operação de envio, de forma confiável (evitando erros de bits, duplicação, perda e sequenciamento de pacotes, a menos de uma falha permanente do sítio ou da rede) e com transparência de nomeação. Este serviço permite liberações de mensagens ordenadas causalmente (*causal delivery ordering*) [Birman93]; esta ordenação pode permitir a implementação de um controle de concorrência estrito.
A percentagem de destinatários que deverão receber a mensagem enviada é especificada na forma de um parâmetro chamado confiabilidade-k; quando este parâmetro é igual a 0%, especifica que o serviço é bem sucedido mesmo que nenhum membro de um grupo destinatário receba a mensagem e, quando é igual a 100%, que todos os membros do grupo deverão receber as mensagens de forma correta. Neste ultimo caso, o uso de uma primitiva específica (*Flush)* permite que as mensagens *multicasts* ainda não recebidas pelos destinatários, o sejam garantidamente, sincronizando a continuação da aplicação com a recepção de todas as mensagens.

5.3 Serviço de Transferência de Arquivos Multi-destino

Este serviço permite que documentos em arquivos sejam recuperadas localmente e distribuídas aos demais membros da sessão de edição (para serem apresentadas na janela de edição comum), sem sobrecarregar a aplicação com operações de empacotamento/desempacotamento e translação de formatos de dados. Todos os destinatários enviam então respostas para o iniciador do serviço de transferência de arquivo, indicando sucesso ou falha nesta; o suporte confirma de forma única, o sucesso ou a falha para o usuário iniciador que não precisa então se preocupar com confirmações individuais de cada destinatário.

Como no serviço de *multicast*, este serviço proporciona o transporte de arquivos da forma mais confiável possível e provê a transparência de nomeação de membros de um grupo. Também de forma similar ao caso de *multicast*, este serviço permite ao iniciador especificar uma porcentagem *k* de confiabilidade na execução de uma operação de transferência de arquivo multi-destino.

5.4 Serviço de Voto

O serviço de voto implementa operações necessárias para o estabelecimento dinâmico de consenso entre os membros de uma sessão de trabalho cooperativo, através de interações entre instâncias do editor ou entre usuários. A obtenção de uma identificação única de um objeto e o controle de concorrência através do acesso mutuamente exclusivo sobre objetos compartilhados são exemplos do primeiro tipo de situação. A execução de comandos que possam causar consequências desastrosas aos participantes da sessão, como a destruição de todos os objetos da tela exemplifica o segundo tipo de situação citada.

O serviço de voto funciona em três fases, sendo as duas primeiras obrigatórias e a última opcional. Na primeira fase, uma pergunta é distribuída do iniciador para todos os demais membros. Na segunda, cada membro envia seu voto para o iniciador que pode eventualmente informar aos demais membros o resultado da votação, durante a terceira fase. As primitivas do serviço de voto são descritas abaixo:

- *SendIssue*. Permite ao seu iniciador fazer uma pergunta e enviá-la para o grupo.
- *ReceiveIssue*. Permite que cada membro de um grupo receba uma pergunta.
- *SendVote*. Permite que um membro respondedor envie seu voto ao iniciador.

- *ReceiveVote*. Implementa a recepção dos votos no iniciador do processo de votação, a quem é retornado apenas o resultado final da votação.
- *SendIssueResult*. Difunde o resultado da votação para o grupo.
- *ReceiveIssueResult*. Permite aos membros votantes, receber o resultado da votação.

6. UMA IMPLEMENTAÇÃO DO SUPORTE PROPOSTO

O suporte descrito no item anterior está sendo implementado a partir dos recursos de comunicação de grupo do sistema ISIS [Birman89] [Birman91] [Birman93]. O sistema ISIS é um conjunto de ferramentas para programação de sistemas distribuídos, provendo mecanismos de alto nível para implementação de *software* baseado em grupo de processos.

A escolha do ISIS como ferramenta para implementação do suporte para aplicações cooperativas, deu-se principalmente pelo fato deste prover protocolos mais adequados para interação de grupos, em oposição aos padrões um-para-um encontrados no modelo cliente-servidor convencional [Rodden92]. As características relevantes do ISIS, aspectos de sua utilização no suporte de aplicações do tipo editor distribuído, e a arquitetura do sistema em curso de implementação, serão apresentados nas subseções seguintes.

6.1 Características Funcionais do ISIS

6.1.1 Grupos de Processos

No ISIS são permitidos quatro estilos de grupos de processos: grupos pares, onde os membros do grupo cooperam entre si para algum fim, como no caso de aplicações cooperativas simétricas; grupos cliente-servidor, onde um certo número de clientes interage com um grupo par de servidores (sendo que os clientes de um grupo par não podem monitorar a passagem de mensagens entre o grupo de servidores); grupos de difusão que são similares aos grupos cliente-servidor porém implementando uma forma especial de *multicast*, o *multicast* de difusão, que é enviado de um processo servidor para seu grupo e para todos os clientes; grupos hierárquicos, onde um grupo maior é implementado por subgrupos menores, sendo que um cliente é limitado a um subgrupo que aceita requisições sob sua responsabilidade.

Para o gerenciamento de grupos de processos, o ISIS apresenta funcionalidades que permitem a criação e a destruição de grupos; a entrada (com apresentação de credenciais) e saída dinâmica de processos; a monitoração da entrada (com autenticação de credenciais) e da saída de processos; a manutenção de réplicas de dados, com transferência de estado para processos ingressantes no grupo.

Além disto, o ISIS apresenta facilidades para implementação de aplicações de computação redundante e com recuperação automática de falhas e, para a comunicação a longa distância confiável.

6.1.2 *Multicasts* Ordenados

O sistema de comunicação do ISIS oferece primitivas de *multicast* para grupos de processos com duas formas de ordenação da liberação das mensagens: ordenação total (primitiva ABCAST) e causal (primitiva CBCAST).

- Na ordenação total (*total ordering*), duas mensagens (*multicasts*) quaisquer A e B, são liberadas nos sítios receptores em comum na mesma ordem relativa (ou A antes de B, ou B

antes de A). Este tipo de ordenação é normalmente chamada de ordenação de liberação atômica (*atomic delivery ordering*). Este tipo de ordenação permite que objetos compartilhados, como no caso de documentos em editores cooperativos, sejam atualizados de forma concorrente sem prejuízo da sua consistência, porém sem garantir que a seqüência de atualizações destes objetos seja a que ocorreu realmente a partir das ações dos usuários. No ABCAST, a comunicação se dá de forma síncrona, onde o processo emissor é bloqueado até que todos os sítios recebam a mensagem e entrem em um acordo em relação a ordem de liberação desta.

- Na ordenação causal (*causal delivery ordering*), segundo [Lamport78], uma mensagem A, cuja iniciação ocorre antes da iniciação de uma outra mensagem B, é liberada antes de B em todos os sítios receptores em comum (notação: A $\rightarrow$ B). No ISIS, um evento **E** ocorre antes de um outro evento **F** se: (a) a execução de **F** segue a execução de **E** no mesmo processo; (b) **E** é a iniciação de um CBCAST e **F** é a liberação do mesmo CBCAST em um processo destino; (c) há fechamento transitivo de (a) e (b). Duas mensagens em relação causal estão, desta forma, relacionadas por uma cadeia de transmissão e recepção de mensagens. O CBCAST do ISIS permite uma comunicação assíncrona (*pipeline*), onde o processo emissor só é bloqueado se a capacidade de armazenamento intermediário de comunicação ("bufferização") é excedida. Por esta razão, o CBCAST é entre duas e vinte e cinco vezes mais rápido que o ABCAST, dependendo do grau em que a aplicação é bem sucedida na comunicação assícrona. As taxas de transmissão com o CBCAST são também similares às obtidas com *streams* (comunicação com conexão) do UNIX e TCP.

As primitivas de *multicast* do ISIS são ordenadas com relação a alteração de configuração do grupo de processos, o que garante que um *multicast* iniciado concorrentemente a saídas/entradas de processos, é sempre liberado para os processos do grupo após a alteração, ou pelo grupo anterior, e não por uma configuração intermediária.

6.2 Utilização do Ambiente ISIS para o Suporte de Aplicações Cooperativas

6.2.1 Suporte a Editores Distribuídos

A plataforma de serviços que compõe o suporte da aplicação cooperativa escolhida pode ser mapeada, em grande parte, diretamente a partir das funcionalidades apresentadas pelo ISIS. A tabela abaixo apresenta resumidamente este mapeamento.

	Serviço de Gerenciamento de Grupos	Serviço de Comunicação de Grupo	Serviço de Transferência de Arquivos Multi-destino	Serviço de Voto
Gerenciamento de grupos de processos	criação e destruição de grupos, entrada (com transf. de estado) e saída de processos de um grupo, monitoração de entrada/saída de processos (com verif. de credenciais)			informação do número de processos no grupo
Multicast de grupo	CBCAST, recepção e resposta a mensagens	CBCAST, recepção e resposta a mensagens	CBCAST, recepção e resposta a mensagens	CBCAST, recepção e resposta a mensagens

Tabela 1 - Utilização das primitivas do ISIS no suporte a aplicações cooperativas do tipo editor distribuído.

A escolha da primitiva de *multicast* CBCAST em relação a primitiva ABCAST se deve ao seu melhor desempenho e no fato desta ultima não garantir a seqüência real de operações dos usuários.

6.2.2 Controle de Concorrência

Com as primitivas de gerenciamento de grupos de processos e de *multicast* de grupo do ISIS, pode-se implementar duas políticas de controle de concorrência estrito: através do serviço de voto e através de um mecanismo de passagem de bastão (*token*), que utiliza o serviço de comunicação de grupo do suporte apresentado:

- Com o serviço de voto, um participante de uma sessão de edição cooperativa deve, antes de acessar um objeto, verificar se alguém já o esta fazendo com exclusão mútua. Neste caso, o serviço de voto retorna ao usuário a não possibilidade de acesso ao objeto (i.e. não existe consenso quanto ao direito de exclusão mútua no acesso sobre um objeto). Em caso contrário (i.e. em caso de haver consenso), o participante ganha o direito de acesso com exclusão mútua sobre o objeto.

- O mecanismo de passagem de bastão, proposto em [Birman89] como forma de sincronização de acessos concorrentes em dados replicados, baseia-se na requisição e passagem de um bastão entre processos que acessam as cópias destes dados, de forma exclusiva quando detiverem o bastão; as requisições e passagens de bastões são implementadas com *multicasts* com ordenação causal (CBCASTs). As requisições enviadas para o grupo são armazenadas no topo de uma fila de requisições pendentes, em todos os processos; o processo que detém o bastão, verifica o requisitante mais antigo na sua fila, a quem então o transfere. Desde que um processo envie um bastão somente após tê-lo requisitado, recebido e efetuado as operações de escrita sobre as réplicas (também com *multicasts* com ordenação causal), pode-se garantir um comportamento de 1-cópia com relação as réplicas guardadas pelo bastão, mesmo utilizando-se um *multicast* assíncrono. Porém, como as requisições de bastão são concorrentes, a ordenação causal na sua difusão não seria suficiente para garantir que as filas de requisição estejam globalmente ordenadas; os processos não poderiam então determinar consistentemente o novo detentor do bastão. Entretanto, esta decisão pode ser tomada pelo atual detentor do bastão com a consulta a sua própria fila de requisições. Com a mensagem de passagem do bastão identificando o novo detentor, os demais processos podem remover a requisição correspondente de suas filas. O funcionamento deste mecanismo é similar ao do protocolo de resincronização apresentado em [Sarin85].

A exclusão mútua com mecanismo de bastão, ao contrário da implementada com o serviço de voto, não obriga que um processo aguarde por todas as respostas de todos os processos (consenso) para obtenção do acesso sobre uma réplica, quando esta estiver livre. Neste caso o bastão lhe seria repassado imediatamente por seu último detentor. No caso da réplica estar sendo modificada por um outro processo, com o mecanismo do bastão, o requisitante não necessita repetir várias vezes o pedido de acesso à réplica, como acontece quando se usa o serviço de voto; basta que ele espere por sua vez na fila de requisições. De uma forma geral, a implementação do controle de concorrência estrito com o mecanismo de passagem do bastão, apresenta-se mais eficiente que o baseado no serviço de voto.

6.2.3 Limitações do ISIS no Suporte de Aplicações Cooperativas

Existem aplicações de trabalho cooperativo que fazem uso de comunicação de mídias contínuas (como p. ex. voz, vídeo e áudio) para interações entre os usuários, como é o caso de sistemas de teleconferência em tempo-real (*distributed desktop conferencing systems*) [Watabe91]. Este tipo de comunicação requer suportes que permitem a comunicação em tempo-real. O ISIS não contém facilidades para este tipo de comunicação.

Em alguns casos de aplicações cooperativas, mesmo apenas utilizando mídias discretas, pode-se questionar ainda o uso de *multicasts* causalmente ordenados. Determinadas mensagens de atualização podem ser liberadas de forma desordenada, conforme a semântica das atualizações para a aplicação. Por exemplo, na difusão de atualizações sobre objetos diferentes e independentes, é irrelevante a ordem relativa em que estes dois eventos são recebidos e executados pelos demais participantes.

6.3 Arquitetura

A arquitetura proposta, decomposta em dois níveis funcionais hierárquicos (Figura 3), compõe-se da aplicação cooperativa (editor distribuído), e do suporte a aplicações cooperativas.

No nível da aplicação cooperativa são implementadas todas as funcionalidades relacionadas com o trabalho de editoração cooperativa (gerenciamento de entrada e saída de participantes, interface WYSIWIS - *What You See Is What I See* -, recursos de edição, etc.) e controle de concorrência sobre os objetos compartilhados.

O suporte, implementando os serviços apresentados no item 5, torna transparente à aplicação os aspectos relacionados com o gerenciamento de grupo de processos, a comunicação um-para-vários confiável e com liberação causalmente ordenada de mensagens. Adicionalmente, prevê-se a implementação de um serviço de *multicast* sem ordenação, para implementação de mecanismos de controle de concorrência baseados na detecção de operações conflitantes, para posterior estudo comparativo com os mecanismos baseados no controle estrito.

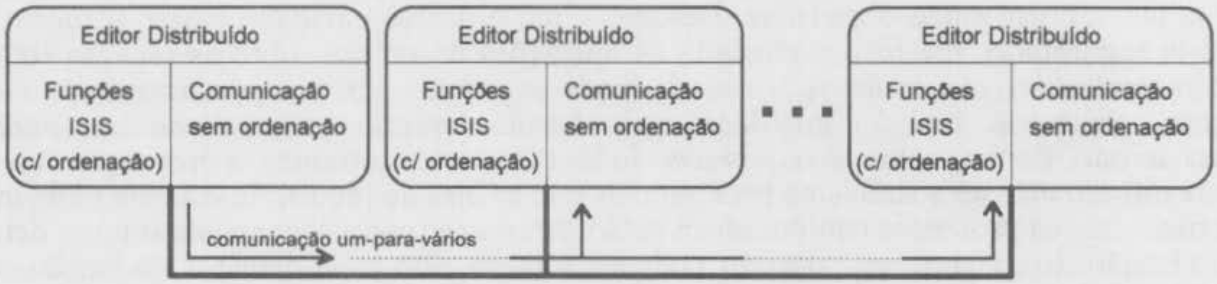

Figura 3 - Níveis funcionais da arquitetura proposta

7. COMPARAÇÕES COM OUTROS TRABALHOS RELACIONADOS

Conforme mencionado anteriormente, o suporte aqui proposto baseia-se nos serviços oferecidos pela plataforma de comunicação de grupo apresentada em [Kirsche93], utilizada no editor de desenhos CoDraft. O trabalho aqui apresentado diferencia-se deste a partir do modelo de implementação adotado, sendo aproveitadas apenas as interfaces dos serviços apresentadas por Kirsche *et al*. A implementação aqui proposta faz uso, no serviço de comunicação de grupo, serviço de transferência de arquivo e serviço de voto, das primitivas de *multicast* causal e dos mecanismos de recuperação de falhas apresentados pelo sistema ISIS. Na implementação de Kirsche *et al*, ao contrário, não existe a preocupação em ordenar causalmente *multicasts* enviados por uma instância do editor. Em aplicações do tipo editor distribuído, porém, é comum encontrar-se situações onde *multicasts* de alteração de objetos em edição estejam relacionados causalmente com outras operações do usuário. Como exemplo desta situação, pode-se citar que *multicasts* causalmente ordenados permitem evitar que um *telepointer* seja apresentado aos demais usuários apontando para um objeto, após a deleção deste objeto.

Além do mais, na plataforma de Kirsche *et al*, dispõe-se apenas do serviço de voto para a implementação do controle de concorrência. No modelo adotado neste trabalho, o uso de primitivas de *multicast* com ordenação causal permite também a implementação do mecanismo de controle de concorrência, baseado na passagem de bastão, que nos parece mais adequado para o tipo de aplicação em questão.

Enfim, os mecanismos de recuperação de falhas apresentados pelo ISIS, permitem também entrega atômica de mensagens (*delivery atomicity*), onde ou todos os processos receptores operacionais recebem uma mensagem, ou se o emissor falhar, nenhum deles a recebe [Birman91]. Isto é desejável para que se garanta que as réplicas dos objetos estejam em estados consistentes em caso de falha do emissor. Em [Kirche93] este problema não é abordado, sendo porém reconhecido como objeto de trabalhos posteriores.

Outrossim, estudos realizados a respeito do desempenho das primitivas CBCAST do ISIS [Birman91], indicam que o preço pago pela ordenação e confiabilidade oferecidos pelo ISIS, pode ser semelhante ao do uso de múltiplas conexões ponto-a-ponto do protocolo TCP sobre UNIX. Desta forma, acredita-se que questões de desempenho não venham a restringir o uso destas facilidades oferecidas pelo ISIS, no tipo de aplicação em discussão neste artigo.

A seguir, a proposta de suporte para aplicações cooperativas feita neste artigo é comparada com trabalhos anteriores que apresentam soluções para o controle de concorrência [Karsenty93], e primitivas de *multicast* ordenadas [Ravindran92].

O algoritmo ORESTE proposto em [Karsenty93] nos parece o mais adequado entre os mecanismos de controle de concorrência com detecção de conflito aqui estudados, para aplicações do tipo editor distribuído, por apresentar baixos tempos de resposta às ações dos usuários sobre as informações compartilhadas. O algoritmo ORESTE baseia-se em regras de mascaração e comutação entre eventos gerados pelos usuários, e propõe relações de dependência menos restringentes que a relação de causalidade utilizada neste trabalho. Porém, como mencionado acima, o uso da relação de causalidade se faz necessária na representação de algumas atividades dos usuários de um editor distribuído. Além do mais, questões como gerenciamento de grupo e transferência de arquivos não são tratadas em [Karsenty93]. Porém, com o objetivo de avaliar melhor este algoritmo e de compara-lo a nossa proposta, prevê-se na continuação deste trabalho a sua implementação, usando os serviços de comunicação sem ordenação, previstos no suporte apresentado neste artigo.

Em [Ravindran92] é apresentada uma estrutura de comunicação baseada também em grupo de processos, onde atividades relacionadas (compostas de interações pelo teclado, mouse, ou canais de áudio, p. ex.) de um, ou entre mais de um usuário, difundidas para o grupo, podem ser ordenadas de forma total (no caso de atividades concorrentes) ou causal, através de uma primitiva específica: *O_Send -Ordered Send*. Com esta primitiva, também os eventos que compõem uma atividade de usuário podem ser relacionados causalmente. A questão de gerenciamento de grupo também é resolvida, com a capacidade de se evitar que entradas e saídas dinâmicas de participantes sejam percebidas nos demais participantes em meio a liberações de eventos que compõem uma atividade. Em comparação com a primitiva de ordenação causal do ISIS, a primitiva *O_Send* apresenta, com a noção de atividades compostas, uma capacidade maior de representação nas mensagens de eventos da aplicação e suas relações causais.

8. CONCLUSÕES

Este artigo apresentou, após uma introdução na problemática de CSCW, as características relevantes e o problema de controle de concorrência de réplicas de objetos em uma aplicação do tipo editor distribuído. Em seguida definiu-se uma plataforma de serviços para o suporte a este tipo de aplicação e discutiu-se uma proposta de implementação desta plataforma, com a

utilização de algumas funções do sistema ISIS. Finalmente apresentou-se alguns trabalhos relacionados com o problema de suporte a aplicações do tipo editor distribuído, comparando-os com a proposta de suporte feita neste trabalho.

As facilidades do ISIS para a programação de grupos de processos e a disponibilidade de *multicasts* assíncronos com ordenação causal na liberação, indicam que este sistema pode ser utilizado satisfatoriamente no suporte para as interações de grupos de aplicações utilizando mídias discretas. Com o objetivo de uma avaliação final da plataforma de serviços descrita, está sendo desenvolvido atualmente no LCMI (Laboratório de Controle e Microinformática) da UFSC, um protótipo de editor gráfico distribuído (para edição cooperativa de figuras geométricas e anotações de texto) e o seu suporte baseado no modelo de implementação da plataforma de serviços apresentado neste artigo.

9. REFERÊNCIAS BIBLIOGRÁFICAS

[Birman89] Birman, K., Joseph, T.; *Exploiting Replication in Distributed Systems.* In Sape Müllender, editor, *Distributed Systems*, New York, 1989. ACM Press, Addison-Wesley.

[Birman91] Birman, K. P. et al; *Lightweight Causal and Atomic Group Multicast.* ACM Transactions on Computer Systems, Vol. 9, No. 3, August 1991.

[Birman93] Birman, K. P.; *The Process Group Approach to Relieble Distributed Computing.* Communications of the ACM, Vol. 36, No. 12, december 1993.

[Ellis91] Ellis, C.A.; Gibbs, S.J.; Rein, G.L.; *Groupware - Some Issues and Experiences.* Communications of the ACM. Vol. 34, No. 1, January 1991.

[Kirsche93] T. Kirsche et al; *Communication Support for Cooperative Work.* Computer Communications, vol. 16, no. 9, september 1993.

[Karsenty93] Karsenty, A.; Beaudouin-Lafon, M.; *An Algorithm for Distributed Groupware Applications.* Proceedings of The 13th. International Conference on Distributed Computing Systems, Pittsburg, Pennsylvania, may 25-28 1993.

[Lamport78] Lamport, L.; *Time, Clocks, and the Ordering of Events in a Distributed System.* Communications of the ACM. Vol. 21, No. 7, July 1978.

[Palme92] Palme, J.; Tholerus T.; *SuperKOM -Design Considerations for a Distributed, Highly Structured Computer Conferencing System.* Computer. Communications., vol. 15, no. 8, october 1992.

[Ravindran92] Ravindram, K.; Prasad, B.; *Communication Structures and Paradigms for Distributed Conferencing Applications.* Proceedings of The 12th. International Conference on Distributed Computing Systems, Yokohama, Japan, June 9-12, 1992.

[Rodden92] Rodden, T.; Blair, G. S.; *Distributed Systems Support for Computer Supported Cooperative Work.* Computer Communications, vol. 15, no. 8, october 1992.

[Stefik87] Stefik, M. et al; *Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings.* Commun. ACM 30, 1 (jan 1987), 32-47.

[Sarin85] Sarin, S. et al; *Computer Based Real-time Conferencing Systems.* IEEE Computer 18, 10 (oct 1985), 33-45.

[Watabe90] Watabe K. et al; *Distributed Desktop Conferencing System with Multiuser Multimedia Interface.* IEEE Journal on Select. Areas in Communic., Vol 9. No. 4, May 1991.

[Williams94] Williams, N.; Blair, G. S.; *Distributed Multimedia Applications: A Review.* Computer Communications, vol. 17, no. 2, February 1994.