UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Projeto e Implementação de um Suporte para Aplicações Cooperativas do Tipo Editor Distribuído

Dissertação submetida à Universidade Federal de Santa Catarina

para obtenção do grau de

Mestre em Engenharia Elétrica

Udo Fritzke Júnior

Florianópolis, 21 de Novembro de 1995.

Projeto e Implementação de um Suporte

para Aplicações Cooperativas do Tipo Editor Distribuído

Udo Fritzke Júnior

Esta dissertação foi julgada para obtenção do título de

Mestre em Engenharia

especialidade **Engenharia Elétrica,**

área de concentração **Sistemas de Controle, Automação e Informática Industrial,**

e aprovada em sua forma final pelo Curso de Pós-Graduação.

Prof. Jean-Marie Farines, Dr. Ing.

Orientador

Prof. Enio Valmor Kassick, Dr.

Coordenador do Curso de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Joni da Silva Fraga, Dr.

Prof. Carlos Maziero, Dr.

Prof. Cirano Iochpe, Dr. rer. nat.

Aos meus pais Udo (in memoriam) e Marianne,
à Margareth,
e aos meus irmãos Roberto e Márcio.

A verdadeira existência humana consiste na vivificação mútua entre os semelhantes.

(S. Taniguchi)

AGRADECIMENTOS

Ao Prof. Dr. Ing. Jean-Marie Farines pelo trabalho de orientação e apoio em todo o desenvolvimento deste trabalho.

Aos membros da banca examinadora pela aceitação de participação e pelas críticas e comentários.

Ao engenheiro José Miguel Eyzell, por nossa amizade e pelas valiosas discussões que contribuiram muito para o trabalho.

À minha prima e acima de tudo sempre amiga, Antonieta E T Kraus, por todo o carinho e incentivo recebidos.

Aos meus amigos e companheiros nesta jornada, engenheiros João Marques Salomão, Sílvia Galvão de Souza, Gilmar Lopes Ribeiro e Fernando Passold. Também a todos os demais colegas de mestrado que não estão incluídos nesta lista apenas por motivo de espaço.

Ao estudante de Engenharia Elétrica Sr. Fabiano Bachmann pela colaboração nos testes da implementação do editor distribuído.

Aos meus amigos Sidney de M. Heusi, Edson Knippenberg, Fernando Ribeiro, Sérgio de M. Heusi, que mesmo distantes, sempre se mantiveram por perto.

Aos colegas do Laboratório de Controle e Microinformática (LCMI) da Universidade Federal de Santa Catarina (UFSC) que ofereceram todo o suporte para a realização deste trabalho.

À UFSC e à CAPES pelo suporte material e financeiro.

Mais uma vez quero agradecer aos meus pais pelo eterno exemplo de vida e de luta, aos meus irmãos pelo apoio e incentivo e à Margareth, pelo amor e carinho compartilhados em todos os momentos.

A Deus.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

A necessidade de ferramentas para serem utilizadas na construção de sistemas de Trabalho Cooperativo (CSCW - *Computer-Supported Cooperative Work*), têm despertado o interesse na utilização de tecnologias existentes em Sistemas Distribuídos. Entretanto, este novo campo de aplicação demanda ainda um esforço para a adaptação das tecnologias providas pelos sistemas distribuídos recentes, tendo em vista as funcionalidades requeridas pelas aplicações cooperativas, no suporte das interações e na coordenação de seus usuários.

Neste contexto, este trabalho tem como meta principal propor uma plataforma de serviços que atende aos requisitos para o suporte à construção de aplicações cooperativas onde os participantes interagem de forma síncrona (ao mesmo tempo), distribuída (em diferentes lugares) e através de informações de natureza discreta (texto e imagens, p. ex.). Utiliza-se nesta plataforma algumas das funcionalidades oferecidas pelo sistema ISIS. Os serviços propostos objetivam oferecer funcionalidades para o gerenciamento de grupos de processos que compõem uma aplicação cooperativa, suporte para comunicação de grupo confiável (com entrega causal e atômica de mensagens), mecanismos de votação e de transferência de arquivos multi-destino.

A aplicação para a qual esta plataforma foi construída e testada é um editor distribuído para figuras geométricas. São apresentados neste trabalho o projeto e a implementação da plataforma de suporte proposta, e a seguir avaliados os aspectos operacionais (desempenho) e funcionais dos serviços implementados.

ABSTRACT

The need of tools for the construction of CSCW (Computer-Supported Cooperative Work) systems, has raised the interest on the use of present Distributed Systems technologies. However, this new application field demands for efforts on fitting these technologies according to the requirements of cooperative systems, in the support of the interactions and coordination among their users.

In this context, this work has the main goal of proposing a support service platform which provides the functionalities required for the construction of cooperative systems in which participants interact in a synchronous (same time) and distributed (different places) way, and by means of discrete nature information (e.g. text and image). This platform has used some of the functionalities of the ISIS system. The proposed services intend to offer functionalities to the management of process groups which compose a cooperative application, support for reliable group communication (with causal and atomic delivery of messages), and voting and multi-party file transfer mechanisms.

This platform has been implemented for the support of a distributed editor for geometrical shapes, which has also been used to test the implemented services. The design and the implementation of the proposed support platform are also presented in this work, and in the following are evaluated the operational (performance) and functional aspects of the implemented services.

CAPÍTULO 1

INTRODUÇÃO

A busca de cooperação, tendo em vista o alcance de objetivos comuns, é uma tendência natural das pessoas em seu ambiente de trabalho. Com o desenvolvimento tecnológico dos sistemas computacionais, principalmente em termos de processamento e comunicação de dados, tornou-se evidente a utilidade destas tecnologias no apoio às interações e à coordenação de grupos de pessoas em contextos de colaboração. Com esta preocupação, passaram a existir sistemas computacionais cujos objetivos eram o de fornecer o suporte para a cooperação entre indivíduos, buscando tornar o trabalho em conjunto mais frutífero do que simplesmente a soma dos esforços individuais. *Groupware* (ou aplicações cooperativas) é a denominação encontrada na literatura para estes tipos de sistemas, e o estudo e pesquisa das tecnologias que permitirão a construção destas aplicações, abrange um campo multi-disciplinar chamado CSCW - *Computer-Supported Cooperative Work* (ou simplesmente Trabalho Cooperativo, como será utilizado neste trabalho) [Ellis91].

O suporte para a construção de aplicações cooperativas, tem sido também uma preocupação em sistemas de Trabalho Cooperativo. Nos níveis mais baixos das implementações de aplicações cooperativas, os sistemas distribuídos existentes são fortes candidatos para este suporte. Sistemas baseados em redes de alto desempenho, comunicação de grupo, e que permitam a manipulação de objetos multimídia (texto, áudio e vídeo, p. ex.) vêm contribuir efetivamente no projeto e implementação destas aplicações [Trevor95]. Entretanto, os sistemas distribuídos atuais não podem diretamente suprir às necessidades de suas aplicações tradicionais e ao mesmo tempo, atender aos requisitos de aplicações cooperativas, onde deve ser levado em conta essencialmente o "porque" de as pessoas trabalharem em conjunto e freqüentemente, o "como" este trabalho deve ocorrer [Trevor95]. Desta forma, em sistemas de Trabalho Cooperativo, funcionalidades adicionais às providas pelos sistemas distribuídos atuais se fazem na maioria das vezes necessárias.

1.1 O PROJETO

Neste contexto, o trabalho aqui desenvolvido foi motivado principalmente pela necessidade de serem propostos, e disponibilizados na prática, serviços que oferecessem determinadas funcionalidades desejáveis em alguns tipos de aplicações cooperativas. Dentro do amplo campo de aplicação dos sistemas de Trabalho Cooperativo, escolheu-se um problema particular, para se ter, principalmente, uma referência mais nítida das funcionalidades de suporte necessárias. Por este motivo, foi dada ênfase para aplicações do tipo Editor Distribuído. A utilização de sistemas distribuídos que mais se adequassem aos requisitos existentes, como o suporte de interações de grupos de processos, foi também uma premissa básica no projeto e implementação dos serviços propostos.

Desta forma, após um estudo a respeito das características funcionais de aplicações de Trabalho Cooperativo e do levantamento dos requisitos mais importantes para um suporte a este tipo de aplicações (considerando em especial o caso dos Editores Distribuídos) [Fritzke95a-95b], propôs-se um conjunto de funcionalidades que oferecerão ao programador mecanismos de:

- gerenciamento de grupos de processos que comporão uma aplicação cooperativa, onde os participantes atuarão simultaneamente sobre informações compartilhadas, podendo entrar e sair à vontade de uma sessão de trabalho cooperativo ;
- comunicação de grupo, que garanta interações consistentes e eficientes entre os processos que compõem uma aplicação cooperativa;
- estabelecimento de consenso a respeito de questões semanticamente ligas à aplicação, e que podem envolver, ou não, os usuários da aplicação;
- transferência de arquivos contendo informações geradas em uma sessão de edição cooperativa, a partir de um participante para o grupo;

Tendo em vista um ganho de experiência prática na implementação de aplicações cooperativas e também a possibilidade de se validar os serviços de suporte implementados, colocou-se adicionalmente como meta a construção de um Editor Distribuído. O contato com questões de implementação a nível de aplicação, proporcionariam uma real medida da necessidade dos serviços de suporte.

1.2 APRESENTAÇÃO DO TRABALHO

O texto deste trabalho, será subdividido em função dos pontos principais que foram desenvolvidos no andamento do projeto.

Assim, o Capítulo 2 fará uma introdução ao tema Trabalho Cooperativo, onde serão classificadas as aplicações cooperativas e também apresentadas as características funcionais mais importantes destas aplicações. Em seguida serão abordados aspectos sobre o suporte para a construção de aplicações cooperativas, onde serão apresentados os principais requisitos destes suportes.

No Capítulo 3 será apresentado o Editor Distribuído a ser desenvolvido em termos das suas funcionalidades e características gerais. O problema de coordenação das ações concorrentes de indivíduos sobre o estado de um documento compartilhado (controle de concorrência), será também discutido em uma seção específica.

O Capítulo 4 apresentará o conjunto de serviços propostos para o suporte para aplicações de Trabalho Cooperativo do tipo Editor Distribuído. Inicialmente, cada serviço será descrito em termos das funcionalidades que deverão ser oferecidas, exemplificando-se em cada caso a necessidade destas funcionalidades. Em seguida será apresentado uma plataforma (sistema ISIS) que foi utilizada na construção dos serviços propostos. Será também mostrado como foi feito o mapeamento destes serviços sobre as funcionalidades do sistema distribuído utilizado.

O Capítulo 5 aborda de forma detalhada o projeto e a implementação dos serviços da plataforma de suporte proposta e também dos módulos que compõem o Editor Distribuído construído.

No Capítulo 6 são apresentadas considerações finais a respeito de suporte, do ponto de vista dos aspectos operacionais e funcionais da implementação.

Por fim, as conclusões do trabalho realizado e as perspectivas de trabalhos futuros serão encontradas no Capítulo 7.

CAPÍTULO 2

TRABALHO COOPERATIVO

2.1 INTRODUÇÃO

O termo Trabalho Cooperativo refere-se aos mecanismos de coordenação dos esforços e de comunicação entre indivíduos distribuídos geograficamente, com o objetivo de se realizar uma atividade ou alcançar um objetivo comum a um grupo de pessoas. 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 de conceitos associados a Trabalho Cooperativo para um conjunto cada vez maior de aplicações de sistemas informáticos. Neste contexto, a disciplina de Trabalho Cooperativo com Suporte Computacional (*Computer-Supported Cooperative Work* - CSCW), vem estudando aspectos tecnológicos e sociais do Trabalho Cooperativo com o suporte de recursos computacionais compartilhados [Ellis91].

Neste capítulo, são apresentadas inicialmente classificações de sistemas de Trabalho Cooperativo à luz de aspectos temporais e espaciais destes sistemas. Também são apresentadas as características funcionais de sistemas de Trabalho Cooperativo. A partir destas características são então citados requisitos gerais necessários para o suporte computacional destes sistemas. São descritas algumas soluções de suporte utilizadas em sistemas de Trabalho Cooperativo, onde podem então ser vistas propostas de mecanismos que vêm a solucionar, em cada caso, parte dos requisitos citados. Ao final são feitos comentários a respeito dos suportes apresentados e colocadas conclusões gerais do capítulo.

2.2 CLASSIFICAÇÃO DOS SISTEMAS DE TRABALHO COOPERATIVO

Apesar de não se poder classificar os sistemas de Trabalho Cooperativo de uma forma nítida, algumas características deste tipo de sistema podem auxiliar nesta tarefa. Estas características

estão relacionadas, principalmente, com as formas e contextos geográficos nos quais os participantes cooperam, conforme será visto a seguir.

2.2.1 Os Aspectos Temporal e Espacial dos Sistemas de Trabalho Cooperativo

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 [Ellis91] [Rodden92].

Formas de cooperação:

As formas pelas quais os participantes 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 participantes de um sistema de Trabalho Cooperativo podem estar geograficamente distribuídos de várias maneiras: todos em um mesmo local (*co-located*), com suas estações de trabalho interligadas por uma rede local, e possivelmente compartilhando uma projeção de tela; em salas diferentes mas próximas (*virtually co-located*), interagindo por uma rede local e fazendo freqüentemente uso de comunicação em tempo-real de mídias como áudio e vídeo; em salas distintas dentro de um mesmo edifício (*locally remote*), interagindo também por uma ou mais redes locais, interligadas por uma rede do tipo *back-bone*; em lugares distantes (*remote*), suportados por serviços de telecomunicações.

2.2.2 Classificação Funcional dos Sistemas de Trabalho Cooperativo

Pode-se encontrar atualmente sistemas de Trabalho Cooperativo em diversas aplicações e ambientes. A seguir, são apresentadas inicialmente quatro classes clássicas de sistemas cooperativos [Ellis91] [Rodden92] [Williams94]: sistemas de mensagens, de conferência por computador, de tomada de decisão, e de co-autoria e argumentação. Na figura 2.1 estas aplicações são posicionadas com relação aos aspectos temporal e espacial anteriormente mencionados.

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. Os sistemas de mensagens são padronizados pelo CCITT, a partir da série de padrões X400 (*Message Handling System* - MHS). O correio eletrônico e os sistemas BBS (*Bulletin Board Systems*), são exemplos tradicionais de sistemas de mensagens.

Sistemas de Conferência por Computador

Uma conferência consiste tipicamente de um conjunto de mensagens (textual ou multimídia) trocadas de forma assíncrona por um conjunto de pessoas localizadas em ambientes remotamente distribuídos, normalmente em torno de algum assunto de interesse comum. Conversações são entendidas como conjuntos de mensagens com ligações de comentários (como p. ex., respostas ou comentários a uma mensagem prévia). O acesso a uma conferência, assim como a organização e controle de conversações (aprovação de novas mensagens, remoção das obsoletas), são normalmente controladas por um ou mais usuários com direitos especiais (moderadores). O sistema SuperKOM [Palme92] é um exemplo deste tipo de aplicação.

A atual disponibilidade de redes de alto desempenho tem possibilitado a existência de sistemas de conferência tempo-real (*real-time conferencing*) com interações síncronas entre usuários em localizações distintas, como é o caso no sistema RTCAL (*Real Time Calendar*) [Sarin85]. Adicionalmente, sistemas de conferência baseados em ambientes gráficos de estações de trabalho (*desktop conferencing systems*), como no caso do sistema de conferência MERMAID [Watabe90], permitem interações de usuários através do uso de mídias como voz, imagem, 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. A sala de reuniões do sistema Colab [Stefik87] é um exemplo deste tipo de aplicação, onde cada participante tem uma estação de trabalho, sendo estas interconectadas por uma rede local, e também podem visualizar uma ampla tela no mesmo ambiente. Uma extensão destes tipos de

sistemas para o suporte de múltiplos grupos na solução de problemas complexos e obtenção de consenso, é apresentada em [Palmer94].

Sistemas de Co-autoria e Argumentação

Estes sistemas auxiliam a autoria em cooperação de documentos (de texto ou com recursos multimídia) e, suportam a negociação e a argumentação envolvidas em trabalho cooperativo. Editores cooperativos, como os sistemas CoDraft [Kirsche93a] e GROVE [Ellis91], são sistemas de co-autoria, onde o documento final resulta de contribuições individuais e de negociações entre os participantes. O tipo de cooperação é geralmente síncrona e a localização geográfica vai de face-a-face a localmente distribuída.

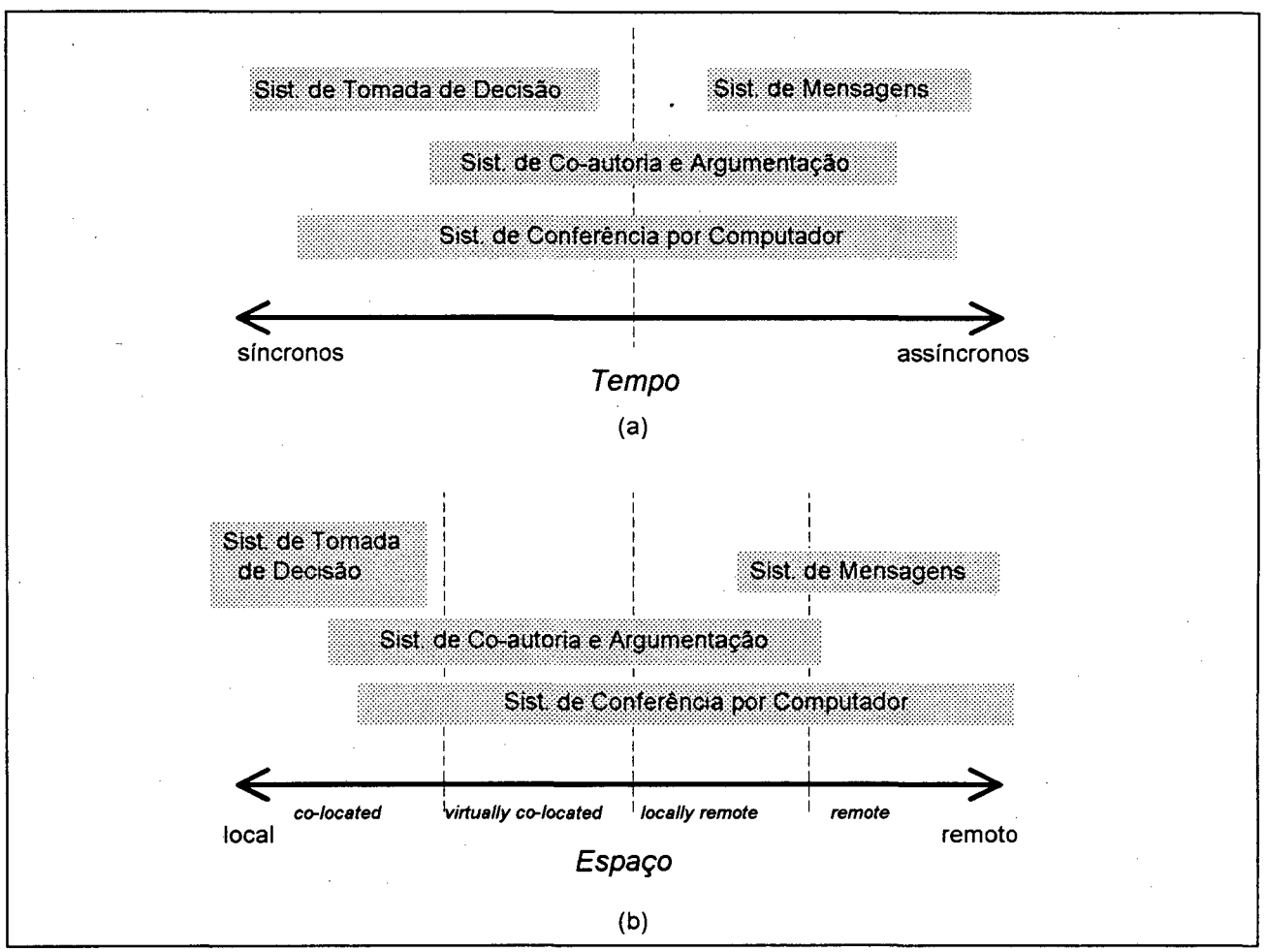


Figura 2.1 - Posicionamento temporal (a) e espacial (b) dos principais tipos de sistemas cooperativos [Rodden92].

Além destas classes de sistemas cooperativos, também podem ser mencionados alguns outros exemplos de áreas de aplicação nas quais o suporte computacional pode trazer benefícios consideráveis para a produtividade de grupos de trabalho:

Agentes Inteligentes e Sistemas de Coordenação de Trabalho Cooperativo [Ellis91]

O sistemas com agentes inteligentes baseiam-se na existência de participantes não humanos, como no caso de jogos para múltiplos usuários em computador e agentes inteligentes para monitoração de sessões de trabalho cooperativo (p. ex. o agente Liza [Gibbs89]). Os sistemas de coordenação são sistemas de suporte à integração e ao ajuste de ações dos indivíduos no sentido de se alcançar um objetivo comum. Em geral estes sistemas permitem a um usuário visualizar suas ações assim como ações relevantes de outros usuários, no contexto do objetivo comum, assim como gerar lembretes e alertas aos usuários.

Suporte a Engenharia de Software Concorrente

Este tipo de sistema tem como objetivo principal diminuir os custos devidos às interações entre os membros de equipes de desenvolvimento de software. Estas aplicações podem beneficiar-se de sistemas cooperativos do tipo síncrono e assíncrono, como sistemas de co-autoria e argumentação, sistemas de mensagens e sistemas de conferência. Como exemplo, tem-se o sistema Flecse (*Flexible Environment for Collaborative Software Engineering*) [Dewan93], que apresenta ferramentas para controle distribuído de versões de códigos fontes, teleconferência, depurador e editor de programas multi-usuários, entre outras.

Automação Industrial Integrada (CIM-*Computer Integrated Manufacturing)*

Sistemas de trabalho cooperativo vem também sendo utilizados com o objetivo de diminuir o intervalo de tempo entre a concepção inicial de um produto, até sua produção em larga escala. Neste caso, a interação entre engenheiros, especialistas, fornecedores e clientes, vem demandando ferramentas assíncronas, como correio eletrônico, e síncronas, como conferências em tempo-real, interfaces multi-usuário em software de projeto (CAD, modelagem 2D e 3D) e esboços interativos informais (sistemas de co-autoria). Como exemplo de trabalho nesta área pode-se citar o projeto CAR (*CAD/CAM in the Automotive industry in RACE - Research and Development on Advanced Communications in Europe*) [Joyner90]. Estudos de infra-estruturas de suporte baseadas em redes ISDN faixa-larga na integração de ilhas de fabricação autônomas e heterogêneas também tem sido feitos [Tschammer90]. Em [Reddy93] é apresentada uma

arquitetura hierárquica de tecnologias, que vão permitir que equipes de engenharia concorrente de produtos, interagindo localmente, colaborem de forma distribuída.

2.3 CARACTERÍSTICAS FUNCIONAIS DOS SISTEMAS COOPERATIVOS

As características funcionais dos sistemas cooperativos estão relacionadas com aspectos sociais de grupos de trabalho, tendo influência direta no comportamento e eficiência tanto dos indivíduos, como do grupo como um todo [Ellis91] [Reinhard94]. Do ponto de vista dos usuários, em geral os sistemas de Trabalho Cooperativo devem favorecer sua integração e permitir um aumento de rendimento na execução de tarefas, em relação ao trabalho individual isolado. Desta forma, estes tipos de aplicações apresentam, em conjunto com os aspectos temporais e espaciais anteriormente mencionados, os seguintes atributos:

Compartilhamento de Recursos e Coordenação

Em uma sessão de trabalho cooperativo, os participantes interagem compartilhando recursos (como p. ex., um documento, um canal de comunicação de voz, etc.) e de forma coordenada, mesmo em distâncias maiores.

Nas aplicações de Trabalho Cooperativo com interações síncronas, deve existir um compartilhamento forte (*strong sharing*) [Tou94], tanto dos dados que definem o estado de uma sessão de trabalho, como das operações sobre estes dados. Isto significa que os dados compartilhados devem estar sempre acessíveis aos usuários, assim como os usuários devem perceber imediatamente o efeito das operações de um membro do grupo sobre estes dados.

A coordenação visa minimizar as operações antagônicas nas interações entre os usuários e aumentar a produtividade do grupo. A complexidade da coordenação varia de acordo com o tamanho do grupo e a tarefa a ser desenvolvida [Reinhard94]. Normalmente grupos pequenos requerem um grau menor de coordenação, devido a menor incidência de conflitos entre operações de usuários. Sessões de *brainstorming* e conferências são exemplos de tarefas que requerem mecanismos de coordenação extremos: enquanto no primeiro exemplo os participantes podem interagir espontaneamente, no segundo caso apenas um participante se comunica enquanto os outros o ouvem passivamente.

A característica interativa e coordenada dos sistemas cooperativos é uma importante diferença com relação às aplicações de sistemas distribuídos tradicionais, como sistemas de arquivos e bases de dados [Ellis91].

Segundo [Rodden92], o controle das interações entre participantes (coordenação) de grupos em sistemas de Trabalho Cooperativo pode ainda apresentar-se de duas formas: controle explícito e controle implícito.

No primeiro caso, as informações de controle podem ser apresentadas e editadas pelos participantes, que podem então visualizar e adequar as interações para o contexto do trabalho a ser realizado. Um dos exemplos de sistemas baseados em controle explícito são os sistemas baseados em teorias de ação de linguagem (*speach act theory*), onde as pessoas coordenam seu trabalho a partir de um conjunto de ações de linguagem (pedidos, promessas, etc.) e passos de conversação pré-estabelecidos (pedidos tem de ser feitos antes da realização de uma promessa, p. ex.) [Ellis91].

Nos sistemas que apresentam controle implícito, não existem formas de representação do controle das interações no grupo. Neste caso, os participantes têm igualdade de direitos e podem atuar e usar os sistemas livremente. Em contrapartida, o sistema não acompanha as atividades do participantes, não podendo então prover suporte de coordenação mais eficiente para o processo de trabalho cooperativo. Os sistemas de co-autoria e os sistemas de conferência, que apresentam um grau reduzido de controle, normalmente efetuado por um participante apenas (moderador), são sistemas onde o controle é implícito.

Grupos Dinâmicos

Os participantes entram em uma sessão de Trabalho Cooperativo segundo políticas pré-estabelecidas, através de convites, ou de requisições explícitas de entrada em um grupo. Ao entrar, um participante deve poder obter todas as informações públicas compartilhadas, para poder participar do trabalho nas mesmas condições que seus colegas (transferência de estado). Em alguns casos é necessário o estabelecimento de níveis distintos entre os membros de um grupo com relação ao acesso as informações compartilhadas [Reinhard94]. A saída de membros ocorre espontaneamente ou por motivo de falha no sítio em que se encontram.

Tempos de Resposta Baixos

Os tempos de resposta em relação as ações dos usuários devem ser baixos. Este tempo de resposta apresenta-se como a combinação das seguintes componentes: tempo de resposta local, que é atraso entre a ação de modificação de um dado compartilhado por um determinado usuário até sua apresentação na interface local, e tempo de notificação, que é o atraso entre a ação de modificação de um objeto até sua apresentação em todos os demais usuários.

Mensagens Interrelacionadas

As mensagens de participantes podem estar interrelacionadas, como é o caso de ligações de comentários em aplicações assíncronas do tipo conferência por computador. A interrelação entre mensagens também pode apresentar-se na forma de restrições de ordenação de operações de usuários a nível do grupo. Como por exemplo, em aplicações síncronas do tipo edição distribuída, as operações de usuários sobre informações compartilhadas devem ser executadas na mesma ordem por todos os participantes, mantendo-se assim a consistência global do documento.

Interações Implícitas e Explícitas

Além de serem do ponto de vista temporal, síncronas e assíncronas, as interações entre usuários de uma aplicação cooperativa podem ser implícitas ou explícitas [Reinhard94]. O primeiro tipo trata da interação através das informações que compõem o objeto do trabalho conjunto, como no caso de um documento textual ou gráfico no processo de edição cooperativa. O segundo caso trata da comunicação direta entre os participantes de um grupo, através de canais de voz, vídeo (como em conferência em tempo-real) ou até através de gestos em reuniões face-a-face.

Interface Compartilhada

É desejável que as informações públicas compartilhadas sejam apresentadas de forma consistente aos usuários. O comprometimento com esta propriedade, definida em [Stefik87] como WYSIWIS (*What You See Is What I See*), pode ser verificada em diferentes graus: desde o WYSIWIS estrito, onde todos os participantes visualizam informações idênticas, até formas mais relaxadas, onde as informações sendo visualizadas por um participante não precisam ser as mesmas dos demais. Neste último caso pode-se ter, por exemplo, situações de autores lendo páginas diferentes de um mesmo documento, em uma sessão de edição cooperativa, ou de ter-se diferentes vistas de um mesmo documento (como posições e formatos independentes dos ponteiros do mouse de cada usuário, diferentes contextos de cores, etc).

2.4 SUPORTE PARA APLICAÇÕES DE TRABALHO COOPERATIVO

Neste trabalho, define-se como suporte para aplicações de trabalho cooperativo, o sistema computacional que vem oferecer facilidades para a implementação de uma determinada classe de sistema cooperativo, ou até de um conjunto delas, segundo suas características funcionais. A arquitetura geral de sistemas cooperativos adotada é similar ao modelo apresentado em

[Rodden92] (figura 2.2), onde o suporte procura implementar as funcionalidades não oferecidas pelos sistemas de comunicação subjacentes.

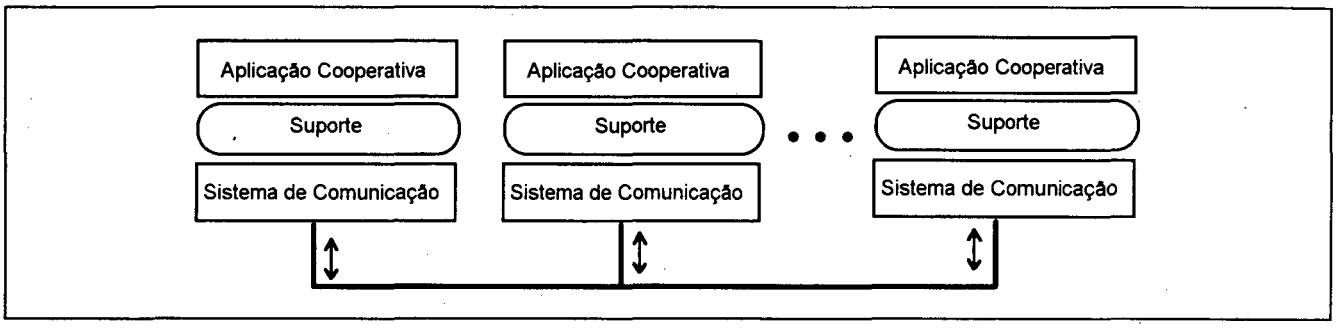


Figura 2.2 - Arquitetura geral dos sistemas cooperativos.

Desta forma, a definição e a complexidade resultante de um suporte está fortemente ligada aos aspectos funcionais da aplicação em questão. Diferentes formas de cooperação e distribuições geográficas, vão demandar diferentes tecnologias de coordenação e comunicação. Como por exemplo, sistemas onde as interações entre os usuários são síncronas e explícitas (como em *desktop conferencing*), requerem tempos de resposta do sistema de comunicação mais estritos do que aqueles requeridos por aplicações onde os usuários interagem de forma assíncrona (como em conferência por computador).

2.4.1 Requisitos para o Suporte a Aplicações Cooperativas

De uma forma geral, as características funcionais dos sistemas cooperativos anteriormente apresentadas demandam as seguintes características do suporte:

Comunicação de Grupo

Aspectos como as formas de coordenação das interações entre usuários, a natureza dinâmica dos grupos compostos por estes usuários, e possíveis relações entre as mensagens de difusão[1] trocadas por estes, evidenciam a necessidade de mecanismos de comunicação de grupo que permitam:

- o uso de endereçamento de grupo, livrando o programador de detalhes de endereçamento dos processos individuais que compõem o grupo (transparência de nomeação). A natureza dinâmica dos grupos que compõem sessões de trabalho cooperativo requer que o

[1] Difusão, no contexto deste trabalho, refere-se ao envio de uma mensagem para todas as entidades que compõe um grupo em uma aplicação cooperativa.

endereçamento de grupo seja também flexível no que diz respeito às entradas e saídas de participantes, e não estático com relação à configuração dos sítios de trabalho [Pendergast93].

- a entrega confiável das mensagens para os processos de um grupo (sem perda ou duplicação).

- a entrega atômica de mensagens para o grupo, ou seja, ou todos os processos do grupo recebem uma mensagem, ou nenhum deles a recebe. Este requisito deve-se à natureza interativa dos sistemas cooperativos, onde atividades de um usuário podem depender de atividades anteriormente executadas por outros usuários, e também da certeza de que estas atividades foram executadas globalmente.

- a entrega assíncrona de mensagens, isto é, não bloqueando o emissor até que a operação de envio seja completada. Esta característica, em conjunto com a habilidade da aplicação de receber mensagens também de forma assíncrona (sem executar operações de recepção explicitamente), tende a diminuir os tempos de resposta de operações de usuários [Pendergast93].

- tornar transparente para a aplicação as operações de inicialização e controle de canais de comunicação de grupo [Pendergast93], como por exemplo a manutenção de conexões individuais de transporte e controle de restrições de recursos de comunicação;

- a entrega de mensagens emitidas por membros de um grupo respeitando relações de ordenação.

- 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*). O conhecimento da composição do grupo (saber quem entrou ou saiu do grupo, p. ex.) é importante para o andamento do trabalho cooperativo. As alterações de *membership* porém, devem ser percebidas pelos usuários de forma consistente, isto é de forma ordenada com relação às mensagens de difusão. Por exemplo, o envio de uma mensagem de difusão para um grupo com alteração de *membership* em andamento, deve ser entregue para os processos que compõem o grupo antes da alteração ou para aqueles restantes após a alteração, mas não para um conjunto diferente de processos.

Suporte de Tempo-Real

Os requisitos de tempo-real aumentam principalmente com o grau de sincronismo nas interações entre os usuários e com a utilização de mídias contínuas nessas interações.

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

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 (textos, gráficos, etc). As mídias contínuas são mais sensíveis a atrasos de transporte entre sua geração e apresentação em sítios remotos. Para o transporte de informações de natureza contínua, são necessários mecanismos de sincronização intra-mídia que procuram minimizar os efeitos destes atrasos na qualidade de apresentação dessas mídias.

Estes mecanismos, por sua vez, baseiam-se no controle das características de Qualidade de Serviço (*QoS - Quality of Service*) que são requeridas para o transporte de mídias contínuas. A noção de QoS ocorre verticalmente, a partir da interpretação de parâmetros a nível de usuário (qualidade de apresentação de vídeo ou de reprodução de áudio, p.ex.) até o nível de transporte e inferiores. As características de QoS a nível de transporte, podem ser expressas através de parâmetros como vazão *(throughput)*, faixa-passante (*bandwidth*), atraso fim-a-fim, variação de atraso fim-a-fim (*jitter*), taxa de perdas de pacotes ou bits e prioridade, e variam amplamente segundo o tipo de mídia envolvida [Blair93]. Por exemplo, o transporte de vídeo requer alta vazão e garantias de altas faixas-passantes, enquanto conexões de áudio não requerem faixas-passantes tão elevadas (p. ex. 220 Mbps para vídeo não comprimido e 64 Kbps para áudio [Coulson92]); também, áudio e vídeo podem suportar certas percentagens de perdas, mas não atrasos (semântica *better-never-than-late*).

Os requisitos de tempo-real dos fluxos[2] isócronos[3] em sistemas multimídia, devem ser suportados também a nível de sistema operacional, através de garantias de alocação de recursos de

[2] Fluxo: informação sendo transferida fim-a-fim e que requer sincronização [Escobar91]. Normalmente está associado à transferência de dados de uma determinada mídia, também em conexões multi-ponto.

processamento que atendam às restrições temporais na manipulação de amostras, na sua geração e recepção [Coulson92].

A passagem de mídias contínuas através do sistema, requer também que o escalonamento do processamento de tarefas seja fortemente coordenado [Blair93], de tal forma a não prejudicar o comportamento de tempo-real desejado. Por exemplo, tarefas de leitura de informações de vídeo em disco, devem cooperar com módulos de transporte da informação, sem violação da integridade temporal do vídeo.

Em algumas aplicações ainda, é necessário que haja sincronização de mídias contínuas relacionadas entre si[4]. Como exemplo, tem-se a sincronização de lábios (*lip sync*) requerida em sistemas de conferência em tempo-real devido às relações temporais que devem ser mantidas entre os quadros de voz e de vídeo, desde sua produção até sua apresentação em sítios remotos.

Desta forma, o suporte deve oferecer recursos de comunicação e sincronização intra-fluxo e inter-fluxo compatíveis com as características dinâmicas das mídias contínuas utilizadas.

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 que processos em um grupo percebam falhas de outros processos de forma consistente. 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 da falha (com um outro sítio assumindo as vezes do sítio que falhou e continuando a execução da difusão interrompida, ou não entregando a mensagem para nenhum processo, p. ex.).

Suporte à Heterogeneidade

As diferenças entre os sistemas que podem estar interligados por redes de comunicação (principalmente a longa distância) e a variedade de tipos especializados de hardware para o suporte de mídias contínuas [Nicolaou90], são fatores que contribuem para a heterogeneidade dos sistemas cooperativos. Porém, as soluções para o problema de heterogeneidade, como a adição de

---

[3] Segundo Nicolaou [Nicolaou90], fluxos isócronos são fluxos de amostras de tamanho finito que são geradas, transmitidas e recebidas em intervalos fixos de tempo, impondo um conjunto de restrições temporais que não devem nunca ser excedidas.

[4] Estudos de sincronização temporal de mídias contínuas podem ser encontrados em [Leung90], [Little90], [Nicolaou90], [Escobar91], [Coulson92] e [Yavatkar92].

camadas de software que permitam abstrações homogêneas dos sistemas de comunicação subjacentes, devem respeitar compromissos com o desempenho do sistema [Coulson92], tendo em vista principalmente os requisitos de tempo-real dos sistemas cooperativos.

2.4.2 Algumas Propostas de Suporte Existentes

Suportes para sistemas de Trabalho Cooperativo têm sido largamente discutidos na literatura. A seguir são apresentadas algumas propostas de mecanismos de suporte, que vêm de encontro aos requisitos delineados na seção 2.4.1.

Objetos Canais para Sistemas de Suporte de Grupos

Em [Pendergast93] é proposto um modelo de suporte baseado na representação de canais de comunicação de grupo como Objetos Canais (GSS-CO - *Group Support Systems Channel Objects*). Os Objetos Canais facilitam a construção de aplicações cooperativas provendo conexões virtuais de mensagens de difusão (1:N) confiáveis e gerenciamento dinâmico destas conexões em função da entrada e saída de participantes em uma sessão de Trabalho Cooperativo.

Três tipos de objetos (figura 2.3) atuam na interface entre aplicações e o sistema de comunicação, mantendo os Objetos Canais: *Network Base Object - NBO*, *Network Interface Object - NIO*, e *Name Server Object - NSO*. O NBO é responsável pela conexão das aplicações com os Objetos Canais, assim como pelo uso destes objetos para transmissão de mensagens e requisição de lista de estações onde um determinado Objeto Canal está ativo. O NIO manuseia a comunicação entre estações e encapsula as interações entre o NBO e o sistema de transporte (p. ex., IPX, NETBIOS ou TCP/IP - *Transmission Control Protocol/Internet Protocol*) utilizado. A comunicação pode ser feita através de difusão de datagramas, para situações onde a confiabilidade da comunicação não é requerida (e é desejado maior desempenho), e conexões de sessão[5] para difusão confiável, para os casos onde não são toleradas perdas. O NSO encarrega-se de manter uma lista atualizada de Objetos Canais e estações onde existem instâncias de Objetos Canais ativos, de tal forma a permitir ao NIO manter as conexões de difusão.

[5] Por questões de redução do *overhead* na criação e manutenção de múltiplas conexões de transporte, todas as conexões de sessão são multiplexadas sobre uma única conexão de transporte confiável, independentemente do número de Objetos Canais utilizados em cada estação.

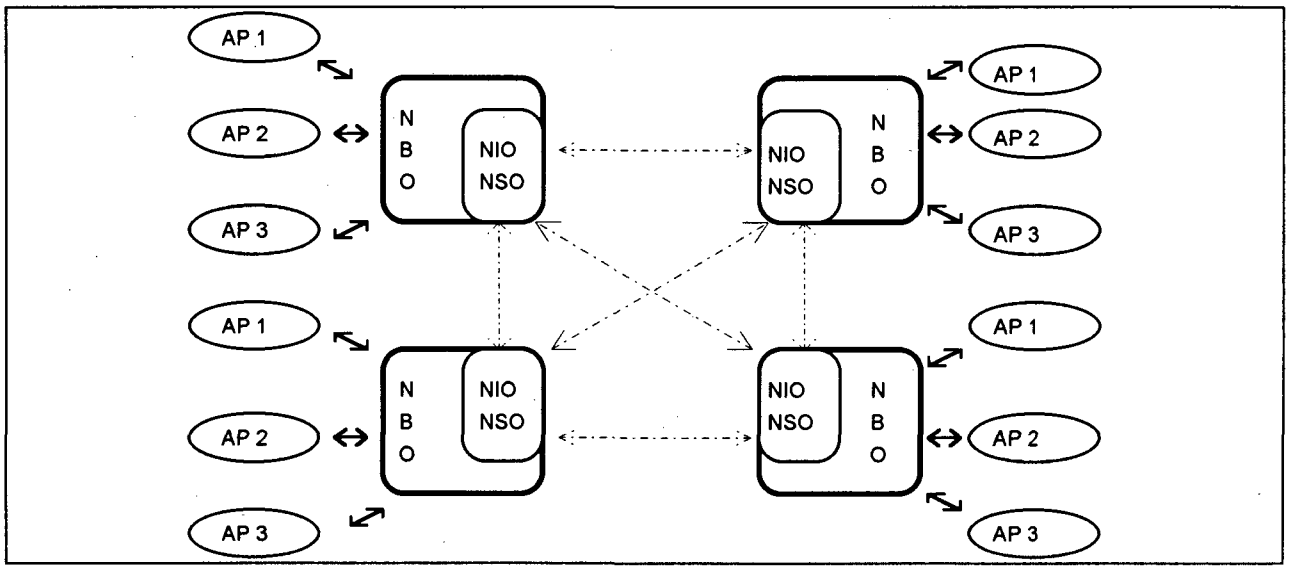


Figura 2.3 - Comunicação utilizando Objetos Canais [Pendergast93]

A estrutura de comunicação baseada nos Objetos Canais tem sido utilizada para o suporte de editores gráficos e de esboços distribuídos, sistemas de tomada de decisão e sistemas cooperativos para projeto de base de dados .

Plataforma de Comunicação de Grupo

A plataforma de comunicação de grupo (*Multiparty Communication Platform*) apresentada em [Kirsche93a] e [Kirsche93b], define os seguintes serviços de suporte para a implementação de aplicações cooperativas: serviço de Gerenciamento de Grupo, serviço de Difusão de Mensagens, serviço de Voto e serviço de Transferência de Arquivo Multi-destino.

O serviço de Gerenciamento de Grupo oferece as funcionalidades necessárias à criação e manutenção de grupos de aplicações. Estas funcionalidades são apresentadas através de primitivas de: criação de grupo, convite de novas instâncias de aplicações para participar de um grupo, requisição de entrada em um grupo, requisição de lista atual de aplicações participantes em um grupo, saída de um grupo e expulsão de alguma aplicação participante de um grupo.

O serviço de Difusão de Mensagens permite a difusão de mensagens a todos os membros de um grupo com apenas uma operação de chamada, de forma confiável (evitando erros de bits, duplicação, perda e erro de seqüenciamento de pacotes, a menos de uma falha permanente do sítio ou da rede) e com transparência de nomeação. O envio das mensagens pode ocorrer segundo três modos: modo imediato, onde as unidades de dados do serviço são entregues aos receptores com o menor atraso de transporte possível; modo *stream*, no qual provedor do serviço decide a

quantidade de unidades de dados de serviço que são entregues aos destinatários e o instante no tempo em que os dados são transferidos; modo concatenado, que funciona de forma similar ao modo *stream*, porém mantendo os limites entre as unidades de dados de serviço na entrega. O uso de uma primitiva *flush* permite que as mensagens ainda não recebidas pelos destinatários o sejam garantidamente, sincronizando a continuação da aplicação que difundiu a mensagem com a recepção destas. A utilização dos diferentes modos de envio provê uma otimização da relação entre o fator de utilização de pacotes de camadas inferiores e o tempo de resposta para eventos disparados pela aplicação.

A percentagem mínima de destinatários que deverão receber a mensagem enviada pode ser especificada pelo parâmetro "confiabilidade-k". Quando este parâmetro for igual a 0%, especifica-se que o serviço é bem sucedido mesmo que nenhum membro de um grupo destinatário receba a mensagem e, quando for igual a 100%, que o serviço é bem sucedido apenas quando todos os membros do grupo recebem as mensagens de forma correta.

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

O mecanismo de votação provido pelo serviço de Voto funciona em três fases, sendo as duas primeiras obrigatórias e a última opcional. Na primeira fase, uma pergunta é distribuída pelo 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.

O serviço de Transferência de Arquivo Multi-destino permite que imagens armazenadas em arquivos sejam recuperadas localmente e distribuídas ao grupo de aplicações, sem sobrecarregar a aplicação com operações de empacotamento/desempacotamento e translação de formatos de dados e utilizando protocolos adequados para transferência de volumes maiores de dados. Todos os destinatários enviam então respostas para o iniciador do serviço de transferência de arquivo, indicando sucesso ou falha nesta transferência. O suporte confirma de forma única o sucesso ou a falha para a aplicação iniciadora que não precisa então se preocupar com confirmações individuais de cada destinatário.

Como no serviço de Difusão de Mensagens, este serviço proporciona o transporte de arquivos da forma mais confiável possível e provê a transparência de nomeação de membros de um grupo. Além disto, este serviço permite também ao iniciador especificar uma percentagem *k* de confiabilidade na execução de uma operação de transferência de arquivo multi-destino.

A plataforma de comunicação de grupo foi utilizada no suporte de uma aplicação do tipo editor gráfico distribuído (CoDraft), que permite a edição cooperativa, entre usuários remotos, de desenhos baseados em figuras geométricas pré-definidas, traços a mão livre, assim como arquivos de imagens.

Paradigma de Conjunto de Eventos

Em [Ravindran92] é apresentada uma estrutura de comunicação, da mesma forma que as propostas anteriores, baseada em grupo de processos. A estrutura de comunicação de Ravindran e Prasad baseia-se em um "paradigma de conjunto de eventos", onde as atividades de usuários de uma aplicação cooperativa, são representadas como um conjunto (*cluster*) de eventos podendo envolver diferentes tipos de dados (como texto e áudio, p. ex.). As atividades dos usuários atuam sobre o estado de um objeto compartilhado (como um documento cooperativo, p. ex.) através da difusão dos eventos que compõem cada atividade para o grupo, e da conseqüente alteração das cópias locais do objeto compartilhado (réplicas), segundo os eventos difundidos. A manutenção da consistência do estado do objeto entre os participantes, ocorre através de mecanismos de ordenação total da entrega de mensagens ou mecanismos de coordenação implementados a nível de aplicação.

O suporte para o paradigma do conjunto de eventos ocorre através da primitiva de difusão com ordenação causal [Lamport78] O_send (*Ordered Send*). A ordenação causal provida por esta primitiva garante que se a iniciação de uma mensagem *A* ocorre antes da iniciação de uma outra mensagem *B*, *A* é entregue antes de *B* em todos os sítios receptores em comum (notação: $A \rightarrow B$). A relação causal entre mensagens pode ser expressa na primitiva O_Send, da seguinte forma:

```
O_Send (x, nome_do_grupo, Occurs_After(y)),
```

onde `x` é a mensagem a ser enviada, `nome_do_grupo` identifica o grupo para o qual a mensagem `x` deve ser entregue, e a relação `Occurs_After(y)` indica que a mensagem `x` deve ser entregue somente após a entrega da mensagem `y`. Se `y` é setada como `NULL`, a mensagem `x` pode ser entregue sem nenhuma restrição de ordenação.

Desta forma, através da primitiva O_Send, o programador pode inferir diferentes graus de ordenação entre mensagens de acordo com a semântica das atividades a nível de aplicação. Como exemplo, pode-se imaginar um participante (P_1) de um grupo de edição cooperativa difundindo uma anotação textual (TEXT) e um trecho de voz (VOICE), e também um segundo participante (P_2) enviando um comentário (COM) à atividade do primeiro. Considerando-se que no conjunto de eventos (TEXT, VOICE) não existam restrições de ordenação na entrega em qualquer membro P_i (i = 1, 2, ... N; onde N é número de participantes do grupo), sendo porém necessário primeiramente o recebimento deste conjunto no membro que faz o comentário antes do envio do evento COM, estas atividades podem ser expressas por:

em P_1: `O_Send(TEXT, grupo_edição, Occurs_After(PEN))`

e

`O_Send(VOICE, grupo_edição, Occurs_After(PEN))`

em P_2: `O_Send(COM,grupo_edição,Occurs_After(TEXT^VOICE)),`

onde o evento PEN representa, por exemplo, a seleção em um menu da aplicação de uma caneta para a escrita. A possível intercalação na entrega do conjunto de eventos (TEXT, VOICE), ou seja, tanto TEXT pode preceder VOICE, como VOICE pode proceder TEXT (ou ainda combinações de ambas), e a relação de causalidade entre este conjunto de eventos e o evento COM, demonstra a expressão de diferentes graus de concorrência admissíveis entre eventos em uma aplicação cooperativa.

O gerenciamento de grupo provido pelo suporte apresentado por Ravindran e Prasad, permite que a difusão de conjuntos de eventos respeite a propriedade de atomicidade de entrega, onde se garante que todas as mensagens de um conjunto de eventos, ou são recebidas por todos os membros de um grupo, ou não são recebidas por nenhum membro. Adicionalmente, também pode prevenir que entradas e saídas dinâmicas de participantes sejam percebidas nos demais participantes em meio a entregas de eventos que compõem uma atividade. Isto impede que, no exemplo anterior, um membro ingressante em meio à difusão do conjunto de eventos (TEXT, VOICE), apenas receba o trecho de voz. Impede também que uma falha no sítio emissor da atividade (TEXT, VOICE) faça com que os demais membros recebam esta atividade de forma incompleta. Estas propriedades do sistema de comunicação proposto por Ravindran e Prasad são implementadas através de um mecanismo de envelopamento e de um protocolo específico de entrega (*commit*) das mensagens que compõem um conjunto de eventos.

Protocolo de Conversação Multi-Fluxo

Em [Yavatkar92] é proposto um protocolo de transporte chamado MCP (*Multiflow Conversation Protocol*), que leva em conta, além do uso de mídias discretas nas interações entre participantes de uma sessão de Trabalho Cooperativo, a utilização de fluxos distintos de mídias contínuas relacionadas (como voz e vídeo, p. ex.). Cada fluxo é representado por uma conexão (um-para-um ou multi-ponto) entre os participantes. No protocolo MCP, os fluxos de mídias distintas relacionados compõem uma conversação multi-fluxo, que provê sincronização temporal na entrega das mensagens emitidas através das conexões de cada fluxo.

Para a obtenção de sincronização temporal em uma conversação multi-fluxo, é proposto o uso da noção de Δ-causalidade, que associa à causalidade estrita (ver proposta anterior) as limitações de atraso das mídias utilizadas nos fluxos que constituem a conversação. Por exemplo, se um participante emite, através de duas conexões um-para-vários distintas, respectivamente, um fluxo de voz e outro informando eventos do seu mouse, os participantes receptores deverão receber estes fluxos de forma não apenas causalmente ordenada, mas também respeitando os limites de atraso da entrega dos pacotes de voz. Pacotes de voz atrasados além do limite (300 ms, p.ex.) devem ser descartados, e os eventos de mouse relacionados não precisam ter sua entrega postergada mais do que este limite.

Duas mensagens m_1 e m_2 estão em relação Δ-causal se:

1) $m_1 \rightarrow m_2$ e

2) m_1 é enviada no máximo Δ unidades de tempo antes de m_2.

O valor de Δ, para uma determinada conversação multi-fluxo, é calculado considerando a dinâmica das mídias de cada fluxo que compõe a conversação. A cada um destes fluxos é associado um atraso desejável, que é o máximo atraso fim-a-fim tolerável antes que a qualidade do serviço deteriore, e um atraso suportado, que é o limite máximo de atraso fim-a-fim a partir do qual a entrega da informação perde a utilidade. O valor de Δ deve estar entre o maior atraso desejável, e o valor do menor atraso suportado, encontrados entre os fluxo da conversação.

Desta forma, a Δ-causalidade garante a entrega causalmente ordenada de mensagens de uma conversação multi-fluxo em um sítio A, apenas para as mensagens transmitidas em um passado recente, ou seja, entre T e T-Δ (onde T é o instante em que a última mensagem entregue a A foi transmitida). Para a implementação deste esquema, as mensagens são unicamente identificadas na conversação e marcadas com carimbos de tempo na sua emissão.

O protocolo MCP oferece também mecanismos de coordenação dos participantes através de controle de concorrência baseado em passagem de bastão. A cada fluxo criado é associado um bastão. Este bastão oferece uma autorização para transferência de dados sobre o fluxo. As aplicações que participam de um fluxo utilizam primitivas de transferência, replicação e deleção do bastão. Estas primitivas permitem diferentes níveis de coordenação. Em um extremo estão as aplicações onde cada participante, só utiliza o fluxo para emissão de mensagens quando em posse do bastão único (*strict floor concurrency control*). Em outro extremo, como no caso das aplicações do tipo *brainstorming*, não existe controle de concorrência sobre o acesso ao fluxo. Neste último caso, o bastão é replicado entre todos os participantes do fluxo.

Extensões à Arquitetura ANSA para o Suporte de Sistemas Cooperativos Multimídia

Em [Coulson92] são propostas extensões à arquitetura ANSA (*Advanced Networked Systems Architecture*) [ANSA87]. As extensões tem como objetivo principal a adequação dos modelos computacional e de engenharia[6] da arquitetura ANSA ao suporte de sistemas cooperativos multimídia.

Com relação ao modelo computacional, são propostas formas de representação da transmissão de mídias contínuas, mecanismos para expressão de sincronização de comunicação de mídias contínuas e de eventos discretos, capacidade de especificação e alteração dinâmica da qualidade de serviço na comunicação de mídias contínuas, habilidade de alocação de recursos de forma atômica (tudo-ou-nada) e suporte à comunicação de grupo, tanto em modelos de invocação (RPC, *fork/join*, e envio assíncrono) como em transmissão de mídias contínuas.

As funcionalidades adicionais ao modelo computacional propostas baseiam-se em uma plataforma básica de serviços que consiste de dois tipos básicos de objetos: dispositivos e *streams*[7]. Estes serviços podem ser vistos na arquitetura ANSA como serviços complementares.

Os objetos dispositivos provêm uma abstração dos dispositivos físicos de produção de informação multimídia (como câmeras, p. ex.), assim como de consumo (como *displays*, p. ex.) e também intermediários (como dispositivos de compressão de vídeo, p. ex.). Os objetos dispositivos são acessados pelo usuário através de dois tipos de interfaces: *chain interface*, que oferece as funcionalidades genéricas de uma fonte ou de um consumidor de fluxos de mídias contínuas

[6] O modelo computacional especifica um modelo de linguagem de programação de objetos potencialmente distribuídos e suas interações. O modelo de engenharia apresenta as especificações, orientações e conceitos através dos quais um modelo computacional abstrato pode ser realizado a nível de sistema. No modelo ANSA (e também ODP) estes vários modelos representam as diferentes visões do problema.

[7] Aqui o termo *stream* não se refere ao modo de envio de mensagens apresentado na seção 2.4.2, mas ao fluxo fim-a-fim de dados de um determinado tipo de informação multimídia.

(como operações de *start* ou *stop* de uma câmera de vídeo, p. ex.) e a *end-point interface*, que oferece uma abstração de todos os aspectos relacionados com o transporte de mídias contínuas (como a leitura/escrita de dados de um fluxo, p. ex.). Adicionalmente, *end-point interfaces* podem ser agrupadas de tal forma a representar grupos de fontes ou consumidores de fluxos de mídias contínuas.

Os objetos *stream* são abstrações de conexões de transporte (uni-direcionais) e provêm os serviços através dos quais os objetos dispositivo são conectados entre si. O controle das conexões de transporte dos objetos *stream* se dá através da *stream interface*, que oferece serviços de conexão/desconexão de dispositivos fontes e consumidores, alocação de recursos de rede segundo uma política de melhor esforço, e leitura e alteração das características do *stream* (como p. ex., vazão, taxas de erro, etc.).

Pode ser visto na figura 2.4 um exemplo de configuração destes serviços básicos, onde se têm um grupo de dispositivos fonte produzindo um fluxo de mídias contínuas e um grupo de dispositivos consumindo este fluxo (conexão tipo M:N), como no caso de uma conferência telefônica multi-usuário.

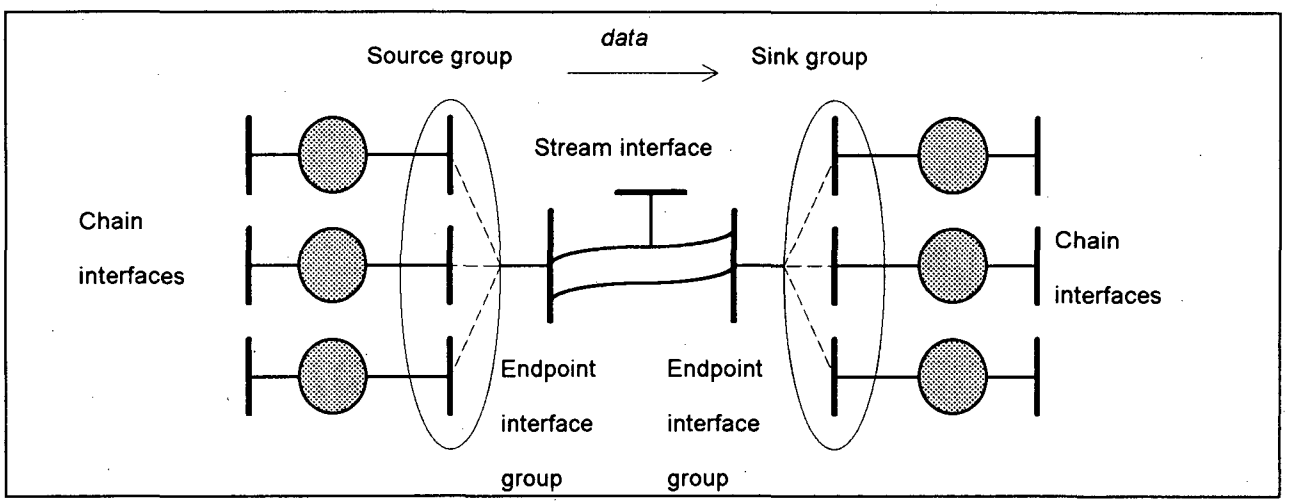


Figura 2.4 - Exemplos de serviços básicos em uma configuração de grupo [Coulson92]

Com relação ao modelo de engenharia, a extensão à arquitetura ANSA propõe meios de suporte para pilhas de protocolos eficientes e reconfiguráveis, suporte de alocação dinâmica de recursos para conexões envolvendo comunicação de mídias contínuas, e suporte a heterogeneidade em sistemas distribuídos. Para resolver o compromisso entre os requisitos de suporte a heterogeneidade e desempenho do sistema em relação à comunicação de mídias contínuas, foi proposta a inserção de uma interface de rede multimídia (*MNI - Multimedia Network Interface*) [Blair93] baseada em *transputers*. A MNI oferece à aplicação na estação de trabalho hospedeira

as funcionalidades do suporte ANSA extendido e facilidades adicionais como sistema X Window[8] com suporte de janelas de vídeo, conversores A/D e D/A, algoritmos de compressão de vídeo, etc.

2.4.3 Comentários a Respeito dos Suportes Apresentados

A partir da descrição dos modelos de suporte anteriormente apresentada, pode-se observar que eles atendem de maneira distinta aos requisitos de suporte à comunicação de grupo. Os suportes apresentados em [Pendergast93] e [Kirsche93a-93b] apresentam mecanismos de endereçamento de grupos dinâmicos, entrega confiável e assíncrona de mensagens, gerenciamento de canais de comunicação de grupo e obtenção de informações de composição de grupos. Entretanto, estes suportes não provém garantias de entrega ordenada, nem atômica, de mensagens. Estas duas últimas características podem ser conferidas no suporte de [Ravindran92].

Com relação aos requisitos de tempo-real, os suportes descritos em [Pendergast93], [Kirsche93a-93b] e [Ravindran92] suportam apenas interações síncronas baseadas em mídias discretas. Já em [Yavatkar92] e [Coulson92], são apresentados mecanismos que estendem as interações síncronas entre participantes de um grupo cooperante com a possibilidade de utilização de fluxos de mídias contínuas.

Aspectos de tolerância a faltas não são apresentados em nenhum dos modelos descritos.

O suporte à heterogeneidade de sistemas distribuídos e de dispositivos de processamento de mídias contínuas, sem o sacrifício dos requisitos de tempo-real, é apresentado em [Coulson92] através da implementação de interfaces de hardware específicas. Estas interfaces têm o objetivo de aumentar a capacidade de processamento de pilhas de protocolos e de manipulação de mídias contínuas, sem sobrecarregar as estações de trabalho e os sistemas operacionais utilizados.

De uma forma geral, observa-se então que as plataformas de suporte apresentam diferentes mecanismos de suporte, não havendo uma específica que atenda a todos os requisitos desejáveis para o suporte de sistemas de Trabalho Cooperativo. Isto demonstra a não trivialidade da questão de suporte aos sistemas de Trabalho Cooperativo, evidenciando também que estes sistemas abrangem uma ampla gama de aplicações com diferentes requisitos temporais.

---

[8] X Window System *is a trademark of the Massachusetts Institute of Technology.*

2.5 CONCLUSÕES

Neste capítulo foi apresentada inicialmente uma classificação dos sistemas de Trabalho Cooperativo, principalmente sob o enfoque dos aspectos temporais e espaciais destes sistemas. Cabe observar que esta é uma classificação aproximada, uma vez que não existe uma fronteira nítida entre os diversos tipos de aplicações cooperativas.

A seguir foram apresentadas características funcionais que, adicionadas aos aspectos temporal e espacial, definem atributos tecnológicos importantes relacionados com os sistemas de Trabalho Cooperativo.

Ao final, são discutidos requisitos gerais relacionados com o suporte computacional para aplicações cooperativas, e apresentadas algumas soluções de suporte encontradas na literatura.

CAPÍTULO 3

EDITOR DISTRIBUÍDO

3.1 INTRODUÇÃO

Conforme visto no capítulo anterior, os mecanismos requeridos para o suporte de aplicações de Trabalho Cooperativo são definidos essencialmente a partir das características funcionais específicas de cada aplicação cooperativa em questão.

Desta forma, para permitir uma definição clara dos serviços necessários a uma plataforma de suporte para sistemas de Trabalho Cooperativo, foi escolhido como objeto de estudo e implementação um modelo específico de aplicação cooperativa da classe de sistemas de co-autoria: um editor distribuído. A necessidade de obter-se uma experiência inicial com características funcionais e requisitos de aplicações cooperativas e também de serem efetuadas avaliações dos serviços de suporte e mecanismos de coordenação, motivaram também a definição desta aplicação específica.

Assim, neste capítulo abordar-se-á inicialmente as características funcionais de um editor distribuído. Em seguida é apresentado o problema de controle de concorrência em aplicações deste tipo e descritas soluções encontradas na literatura. Esta discussão a respeito de controle de concorrência faz-se necessária, devido ao fato de que as formas de interação encontradas em aplicações de edição cooperativa são conceitualmente diferentes das encontradas em sistemas distribuídos clássicos. Sistemas de gerenciamento de base de dados, por exemplo, procuram privilegiar aspectos como transparência de concorrência, enquanto as aplicações de Trabalho Cooperativo são construídas com a intenção de manter a presença e o estado dos participantes visíveis [Ellis89] [Rodden92]. São feitos também comentários a respeito das formas de controle de concorrência. Finalmente, é descrita a interface com o usuário do editor distribuído que será implementado e, ao final, o capítulo é concluído.

3.2 CARACTERÍSTICAS DE UM EDITOR DISTRIBUÍDO

Um editor distribuído (ou editor cooperativo) é uma ferramenta de suporte ao trabalho de edição cooperativa de documentos compostos de combinações de textos e gráficos (figuras e imagens). Neste trabalho será enfocada a problemática relacionada com os aspectos cooperativos para edição distribuída de documentos gráficos (cujas ferramentas são também denominadas *shared drawing tools* [Cosquer95]), em particular no que diz respeito ao seu suporte computacional. Tipicamente, este tipo de aplicação apresenta as seguintes características gerais:

- os participantes estão geograficamente distribuídos, porém próximos (*co-located* e *virtually co-located*);
- as interações entre os participantes são síncronas e implícitas, ou seja, estes atuam simultaneamente compartilhando informações relacionadas ao documento em edição, através de uma interface comum;
- a ordem de grandeza dos grupos de participantes é considerada na literatura [Cosquer95] como limitada a dez.

As principais funcionalidades relacionadas com aspectos de interação entre participantes apresentadas pelo editor distribuído 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 é apresentado um documento em edição. Através desta janela os participantes atuam modificando o documento e podem visualizar as alterações dos demais usuários sobre o mesmo documento.
- interface WYSIWIS (*What You See is What I See*) relaxada [Stefik87], através da qual o documento pode ser visualizado em sua totalidade de forma consistente por todos os participantes, porém, ao contrário do WYSIWIS estrito, não se fazendo necessário que as mesmas informações com respeito ao documento (mesmos objetos, p. ex.) sejam apresentadas simultaneamente a todos os participantes. A interface WYSIWIS relaxada permite um grau maior de autonomia nas ações dos participantes.

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

- mecanismos de garantia da consistência dos objetos da janela de edição comum, nos 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 dinâmicas de uma sessão de trabalho cooperativo;

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

- modos de edição cooperativo ou individual (*stand-alone*).

Das funcionalidades acima mencionadas decorrem duas importantes características relacionadas com o projeto deste tipo de aplicação [Kirsche93a]:

- simetria: todos os participantes devem manter réplicas consistentes dos objetos que compõem o documento sendo editado, sendo que o acesso a este documento não é restringido por algum usuário de maior hierarquia (como um moderador, p. ex.). Além do mais, todos os participantes 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 desta, sem que haja uma forma explícita de controle das interações entre os participantes [Rodden92].

- autonomia: 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 desenhar em qualquer parte da janela de edição comum, a qualquer momento.

3.3 CONTROLE DE CONCORRÊNCIA EM EDITORES DISTRIBUÍDOS

Em editores distribuídos, o fluxo de informação entre os participantes sobre objetos compartilhados é do tipo $M \times (1 \rightarrow N)$, onde M é o número de participantes tentado acessar concorrentemente um objeto e N é o número de réplicas do objeto distribuídas pelo grupo [Cosquer95].

As atualizações efetuadas por um participante sobre um objeto que compõem um documento, ocorrem em duas fases básicas:

(a) atualização da cópia local do objeto replicado, através da janela de edição comum;

(b) difusão da cópia local atualizada para os demais participantes (incluindo aqui sua recepção e apresentação nas janelas de edição comum destes participantes).

Como os participantes interagem de forma síncrona e a partir de sítios distintos, podendo ocorrer, como visto acima, alterações concorrentes sobre um mesmo objeto, podem ser detectadas situações de conflito na apresentação destas modificações nos receptores. Na figura 3.1, tem-se por exemplo dois participantes em sítios distintos (A e B) efetuando modificações concorrentes sobre um mesmo objeto. Após a difusão destas atualizações, todos os sítios envolvidos terminam por não ter como definir um estado consistente do objeto alvo das modificações.

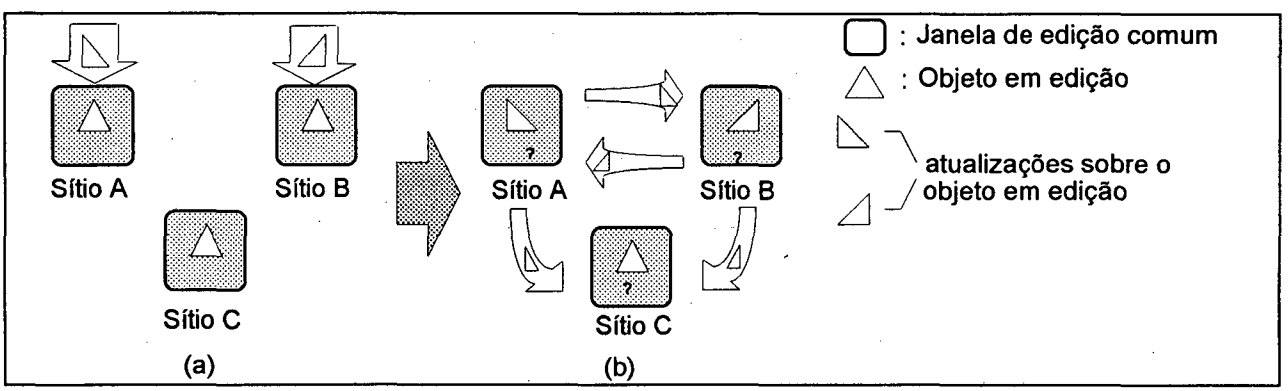


Figura 3.1 - Atualizações concorrentes de um objeto replicado: (a) fase de atualização das cópias locais; (b) fase de difusão das atualizações para os demais sítios, onde ocorre conflito devido a atualizações concorrentes.

Percebe-se então a necessidade de mecanismos de controle de concorrência como forma de coordenação do trabalho cooperativo, para que seja garantida a consistência global do documento em edição.

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 sobre o estado do sistema.

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.

3.3.1 Formas de Controle de Concorrência em Editores Distribuídos

Existem na literatura diversas abordagens para a coordenação das interações dos participantes em aplicações do tipo editor distribuído, desde métodos sem nenhuma forma de controle de concorrência das atividades dos participantes, até aqueles baseados em controle de concorrência estrito (*strict floor control*). Conforme será apresentado a seguir, estas soluções para controle de concorrência são agrupadas, neste trabalho, em três principais classes: sem controle de concorrência, com detecção de operações conflitantes e com controle de concorrência estrito.

Sem controle de concorrência

Nos métodos desta classe, apresentada em [Stefik87] como "modelo cooperativo", 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 participantes difundem atualizações simultaneamente, podem ocorrer situações de disputa para ver qual atualização será executada primeiro (*race conditions*), o que pode levar a cópias inconsistentes do objeto (como mostrado na figura 3.1), dependendo da semântica destas atualizações. Este tipo de mecanismo pode ser empregado em situações onde o acesso simultâneo sobre um mesmo objeto é muito pouco provável, e também onde as situações de conflito resultantes possam ser resolvidas por interações explícitas entre os participantes (através de protocolos sociais, p. ex.).

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 nos receptores, através de mecanismos de detecção de operações conflitantes 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 seqüenciador 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.

No "modelo de detecção de dependência", apresentado também 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 recebido 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 receberam 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 humanos.

Em [Ellis89][Ellis91], propõe-se um método similar ao anterior, porém com a resolução automática do conflito entre as operações (algoritmo dOpt - *distributed Operation transformation algorithm*). 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 da semântica da operação (inserir ou deletar um caracter em um texto, p. ex.) e do histórico de operações já executadas. Este algoritmo foi utilizado no editor de textos cooperativo GROVE.

[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[1] e comutação[2] 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*)

[1] Um evento mascara outro, se o segundo evento não altera o estado de um objeto após a execução do primeiro, sobre o mesmo objeto.

[2] Dois eventos comutam se a ordem com que são executados não altera o estado final do objeto.

posteriores ao evento recebido, para colocar o objeto compartilhado em um estado consistente. Similarmente ao algoritmo dOpt, o algoritmo ORESTE apresenta respostas ágeis 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 [Ellis89].

Com controle de concorrência estrito

Com os métodos que baseiam-se 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[3], de forma a impedir a execução de atualizações concorrentes sobre o mesmo. Após o bloqueio do objeto, as modificações são difundidas de forma a manter todas as réplicas do objeto em um estado atualizado.

Em [Sarin85], um protocolo de re-sincronização garante o controle de concorrência sendo que em um dado instante, um único sítio é a fonte de atualizações sobre um objeto e todos os sítios processam todos os comandos deste antes de começar a receber os comandos de um outro sítio. Em [Stefik87], são apresentados também dois modelos para o controle de concorrência estrito: 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 para o 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 [Kirsche93a], o controle de concorrência baseia-se 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.

3.3.2 Comentários sobre as Classes de Controle de Concorrência

A seguir, analisam-se as vantagens e desvantagens das três classes de controle de concorrência 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

[3] O que equivale a introduzir uma fase anterior à fase de atualização da cópia local (fase (a), seção 3.3), responsável pela garantia de exclusão mútua sobre o objeto replicado.

compartilhados pode aumentar substancialmente esse tempo e tornar a execução da aplicação confusa, devido a freqüente requisição dos usuários para a resolução de conflitos (como no "modelo de detecção de dependência" de Stefik) ou às reversões automáticas de operações já executadas (como nos algoritmos dOpt e ORESTE). 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 de uma figura enquanto outro participante a está aumentando). Em contrapartida, o processo de exclusão mútua sobre um objeto pode estender-se 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 (*"sharable objects"* [Cosquer95]). Uma vez definidos estes objetos compartilhados, não poderão mais ser acessadas partes internas destes (objetos gráficos que o compõem), e o controle de concorrência será garantida somente a este nível [Cosquer95]. Os métodos baseados em controle de concorrência estrito, tornam-se menos restritivos a medida que a granularidade dos objetos diminui (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.

3.4 CARACTERÍSTICAS DO EDITOR DISTRIBUÍDO IMPLEMENTADO

O editor distribuído desenvolvido neste trabalho (FDD - Ferramenta de Desenho Distribuída), tem como objetivo permitir a edição cooperativa de desenhos compostos de objetos geométricos básicos (linhas, retângulos, polígonos, círculos e elipses), e que definem também a granularidade[4] das informações compartilhadas pelos co-autores dos documentos .

[4] Opções de agrupamento de objetos (não previstas nesta versão do editor), comuns em editores gráficos comerciais, podem requerer níveis de granularidade variáveis (definidas pelos usuários), durante uma sessão de trabalho cooperativo. Desta forma, atualmente a granularidade dos objetos que compõem um desenho é fixa.

Este editor apresenta as principais funcionalidades deste tipo de aplicação cooperativa, conforme mencionado na seção 3.2. Através dos botões e menus disponíveis na interface com o usuário (figura 3.2), um participante pode escolher entre ferramentas para o desenho de objetos na janela de edição comum, requisitar serviços de gerenciamento de grupo (para entrar ou sair de um grupo, conhecer quais os participantes estão cooperando, e convidar e expulsar outros participantes), invocar o *telepointer* do sistema, salvar ou ler um desenho de um arquivo, e sair do editor.

Se algum usuário trabalhando individualmente for convidado para ingressar em algum grupo, ele pode salvar o desenho que estava editando e então ingressar no grupo convidante, ou mesmo recusar o convite. Da mesma forma, se um participante de um grupo for convidado a se retirar, ele tem a opção de aceitar ou não este pedido. Se aceitar, ele perde todo o contato que vinha sendo mantido com o grupo. Caso contrário, sua participação é mantida da mesma forma como acontecia antes da tentativa de expulsá-lo.

A garantia de consistência dos objetos compartilhados através da janela de edição comum é realizada por um mecanismo de controle de concorrência baseado em passagem de bastão entre os participantes que manipulam as cópias destes objetos. Este mecanismo vai ser apresentado em maior detalhe no Capítulo 5.

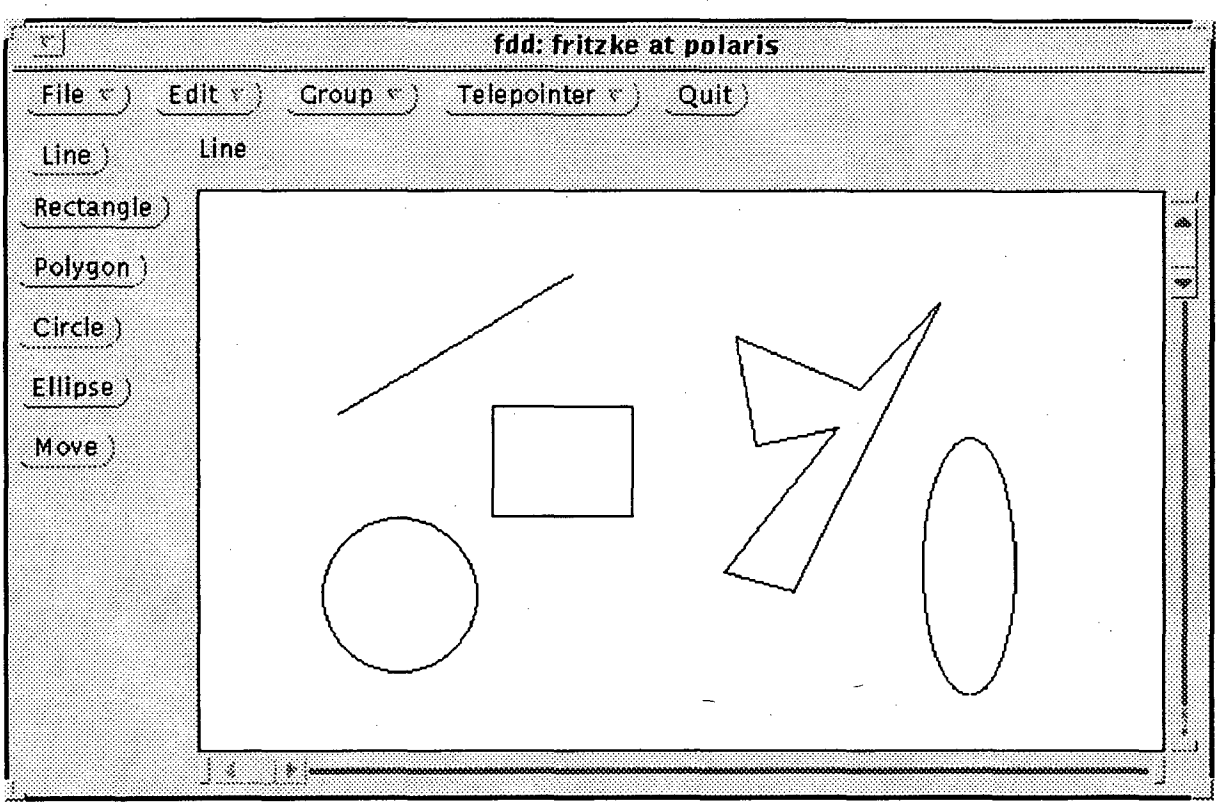


Figura 3.2 - Interface com o usuário do editor distribuído implementado.

3.5 CONCLUSÕES

Este capítulo apresentou as funcionalidades relevantes do ponto de vista da cooperação numa aplicação do tipo editor distribuído e que têm influência no projeto destes sistemas, definindo atributos como simetria e autonomia. Estas características serão importantes na definição de uma plataforma de serviços de suporte para este tipo de aplicação.

Foram discutidas também formas de controle de concorrência nestes sistemas de Trabalho Cooperativo. Os métodos baseados em detecção de operações conflitantes apresentam menores tempos de resposta, devido ao fato de não imporem nenhum tipo de bloqueio ao acesso dos participantes no estado compartilhado de uma sessão de edição. Nesta classe, os algoritmos de Ellis (dOpt) e de Karsenty (ORESTE) mostraram ser os mais adequados, pelo fato de apresentarem correção automática de estados conflitantes dos dados replicados. Os métodos com controle de concorrência estrito são úteis quando é importante garantir a continuidade de ações interrelacionadas iniciadas por um determinado participante, evitando também as sequências de correções no estado do documento em edição que podem ocorrer na classe anterior. Em geral, com o uso de controle de concorrência estrito, as atividades dos participantes são mais coordenadas.

CAPÍTULO 4

UMA PROPOSTA DE SUPORTE PARA EDITORES DISTRIBUÍDOS

4.1 INTRODUÇÃO

Como foi visto no capítulo 2, são conhecidas diversas soluções para o suporte de aplicações cooperativas. O caso específico de suporte dos editores distribuídos também tem sido tratado em vários lugares, enfocando aspectos distintos como comunicação de grupo [Ravindran92] [Kirsche93a] [Pendergast93] e controle de concorrência [Sarin85] [Stefik87] [Ellis91] [Karsenty93]. Apesar disto, não são conhecidos trabalhos que apresentem um conjunto de serviços que englobem estes aspectos e também façam uso de ferramentas existentes para a programação distribuída adequadas ao suporte de grupos de aplicações cooperativas e ainda ofereçam vantagens adicionais aos programadores destas aplicações (como p. ex., replicação de dados, tolerância a falhas, etc.).

Neste sentido, este capítulo vem propor uma plataforma de serviços de suporte que proverá importantes facilidades para o desenvolvimento de aplicações cooperativas síncronas baseadas em mídias discretas, como é o caso de editores distribuídos. Esta plataforma foi projetada e implementada com a utilização do sistema ISIS [Birman93], fazendo uso das funcionalidades apresentadas para programação de sistemas distribuídos baseados em grupos de processos, e adicionando outras não providas por este sistema.

Este capítulo apresentará inicialmente o conjunto de serviços de suporte proposto. Na continuação é apresentado o sistema ISIS e também sua utilização na construção da plataforma de suporte. Finalmente são discutidas diferenças entre o suporte proposto e outros estudados, e apresentadas as conclusões gerais do capítulo.

4.2 UMA PROPOSTA DE SUPORTE PARA EDIÇÃO DISTRIBUÍDA

As características funcionais de um editor distribuído, conforme apresentado no capítulo anterior, permitem identificar os principais requisitos para um suporte a este tipo de aplicação cooperativa. Resumidamente, um sistema subjacente de suporte deverá oferecer:

- mecanismos de comunicação de grupo, que permitam implementações de aplicações com atributos de simetria e de autonomia;
- tempos de resposta dos serviços de comunicação compatíveis com as interações em tempo-real dos participantes de uma sessão de edição cooperativa;
- tolerância a falhas em sítios onde existam instâncias de aplicações e participantes engajados na edição cooperativa.

O suporte a ser apresentado tem o objetivo de atender a estes requisitos, provendo funcionalidades favoráveis ao projeto e à implementação de aplicações do tipo editor distribuído. Estas funcionalidades são providas por quatro serviços que compõem a plataforma de suporte: serviço de Gerenciamento de Grupo, serviço de Comunicação de Grupo, serviço de Voto e serviço de Transferência de Arquivo Multi-destino.

A proposta de plataforma de comunicação apresentada em [Kirsche93a] serviu como base na definição das características deste suporte, haja visto que os serviços desta pareceram ser os mais adequados para a aplicação cooperativa que serve de exemplo neste trabalho. Ao final deste capítulo, discutir-se-ão as diferenças entre o suporte proposto e o de Kirsche et al, assim como também em relação a outros suportes propostos na literatura.

Nas subseções seguintes os serviços mencionados acima são apresentados em maiores detalhes.

4.2.1 Serviço de Gerenciamento de Grupo

O serviço de gerenciamento de grupo provê as funcionalidades para manutenção dos grupos de processos[1]. As primitivas deste serviço permitem que os grupos de processos que constituem um conjunto de aplicações de Trabalho Cooperativo sejam dinamicamente configuráveis e ainda, que

[1] Neste capítulo, o termo "processo" refere-se a uma aplicação de trabalho cooperativo em execução em um determinado sítio (como uma instância do editor distribuído, p. ex.).

estas aplicações tenham conhecimento atual dos participantes de uma sessão de trabalho cooperativo. Abaixo são apresentadas as primitivas do serviço de gerenciamento de grupo:

JoinGroup

Permite a inserção de um processo em um grupo existente ou a criação de um novo grupo. Tanto o processo chamador como o grupo são identificados por nomes simbólicos. Se um grupo que se deseja criar já existe, é retornado um erro. Quando um processo entra no grupo, um dos membros existentes transfere-lhe o estado atual da sessão de trabalho cooperativo (no caso do editor, a última versão do documento). Esta primitiva associa também uma senha para autenticação dos membros do grupo em operações de gerenciamento subseqüentes e retorna um identificador único do grupo.

Leave

Permite a qualquer membro de um grupo abandonar este grupo de forma unilateral.

GetGroupMembers

Permite a algum membro de um grupo requisitar a lista atual de membros do grupo a que pertence.

Invite

Permite a um membro de um grupo já criado, convidar novos processos para participar da sessão. Se o processo convidado aceita o convite, ele é inserido como membro do grupo e um dos membros lhe transfere o estado atual da sessão de trabalho cooperativo.

Expel

Permite a um membro solicitar a saída de outro membro do grupo. Devido à característica simétrica de editores cooperativos, onde não existe um participante com maior prioridade, considera-se que tanto o pedido de expulsão de um membro assim como a aceitação do pedido por parte deste, podem ser feitas de forma autônoma. As implicações sociais deste tipo de atividade não estão sendo estudadas neste trabalho.

4.2.2 Serviço de Comunicação de Grupo

O serviço de comunicação de grupo (*multicast*) permite 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 erro de seqüenciamento de pacotes, a menos de uma falha permanente do sítio ou da rede). Este serviço provê também a entrega de mensagens ordenadas causalmente [Lamport78] onde, se o envio de uma mensagem de difusão A ocorre antes de uma outra B, esta primeira é entregue aos receptores comuns de A e B antes de B ($A \rightarrow B$). Um evento E ocorre antes de um evento F, se:

1) A execução de F segue a execução de E em um mesmo processo;

2) E é a iniciação de uma difusão e F é a entrega desta difusão em um processo destino;

3) Existe o fechamento transitivo de 1) e 2).

A entrega de mensagens é também atômica [Birman91], onde ou todos os destinatários operacionais recebem certamente a mensagem, ou, e somente se o emissor falha, nenhum deles a recebe.

O endereçamento dos processos que compõem o grupo é feito através da identificação do grupo e não pelo endereçamento individual de cada processo (transparência de nomeação). Isto é permitido pelo fato do suporte encarregar-se da sincronização entre alterações na formação do grupo (*membership*) e a comunicação de grupo. Isto garante que uma difusão iniciada de forma concorrente a saídas/entradas de processos, seja sempre entregue para os processos do grupo após a alteração, ou antes da mesma, e não por uma configuração intermediária (atomicidade de endereçamento do grupo).

O uso de uma primitiva específica (*Flush*) permite que as mensagens de difusão ainda não recebidas pelos destinatários o sejam garantidamente, sincronizando a continuação da aplicação com a recepção destas mensagens.

No editor distribuído implementado, o serviço de comunicação de grupo é utilizado para difundir informações geradas pelos participantes ao grupo, mantendo as janelas de edição comum de todos os participantes atualizadas e consistentes. Atualmente, estas informações são os atributos dos objetos geométricos que compõem o documento em edição (dimensões, coordenadas na janela de edição comum, etc.) e as coordenadas assumidas pelo *telepointer*. A entrega causalmente ordenada das mensagens de difusão é utilizada para garantir que operações relacionadas difundidas por um usuário não sejam recebidas em ordens diferentes, e também na implementação

do mecanismo de controle de concorrência que garantirá a consistência global do documento (conforme será visto no próximo capítulo).

4.2.3 Serviço de Voto

O serviço de voto implementa as operações necessárias para o estabelecimento dinâmico de consenso entre os membros de uma sessão de trabalho cooperativo, através de interações entre os processos do grupo, ou através de interações diretas entre os participantes do sistema cooperativo. O primeiro tipo de situação pode ocorrer, por exemplo, na implementação de métodos de obtenção de identificação única de objetos geométricos sobrepostos em editores gráficos (como a relação *is_on_top_of* apresentada em [Kirsche93a]). O controle de concorrência sobre objetos replicados é outro exemplo do serviço de voto envolvendo processos. A execução de comandos que possam causar consequências desastrosas aos participantes da sessão, como a destruição de todos os objetos da tela em um editor gráfico, exemplifica o segundo tipo de situação.

O serviço de voto funciona em três fases, sendo as duas primeiras obrigatórias e a última opcional. Na primeira fase, uma questão é distribuída pelo requisitor para todos os demais membros. Na segunda, cada membro envia seu voto para o requisitor. O processo de votação termina quando ocorre consenso, ou seja, após todos os membros votantes do grupo terem respondido positivamente à questão, ou no momento em que algum deles responda de forma negativa. Na terceira fase o iniciador pode eventualmente informar aos demais membros o resultado da votação.

As questões são marcadas por um identificador único no processo requisitor (*issue_id*), que permite ao serviço de voto gerenciar a votação de uma ou mais questões iniciadas por um mesmo processo simultaneamente. As questões são identificadas globalmente através do par (*issue_id*, requisitor). Além disto o requisitor pode enviar com a questão argumentos que definam a semântica da questão a ser votada a nível de aplicação, como por exemplo "deletar todos os objetos". Desta forma os processos votantes podem distinguir diferentes votações concorrentemente em execução, e ainda obter informações a respeito do objetivos de cada uma destas votações.

As primitivas do serviço de voto são descritas abaixo:

- *SendIssue*. Permite ao requisitor enviar assincronamente uma questão a ser votada ao grupo. O processo requisitor pode definir uma "função de avaliação", que será invocada pelo suporte quando a votação requisitada terminar.

- *ReceiveIssue*. Permite que cada membro de um grupo receba uma questão a ser votada. Esta primitiva é definida pela aplicação na inicialização do serviço de voto e atende a todas as questões recebidas. A aplicação é notificada a cada recepção de uma questão, recebendo o par (*issue_id*, requisitor) e os argumentos que definem a semântica da questão.

- *SendVote*. Permite que um membro respondedor envie seu voto de concordância ou discordância ao requisitor.

- *ReceiveVote*. Implementa a recepção dos votos no requisitor do processo de votação, que é notificado a respeito do resultado final da votação através da "função de avaliação" definida na primitiva *SendIssue*.

- *SendIssueResult*. Difunde o resultado da votação para o grupo.

- *ReceiveIssueResult*. Permite aos membros votantes, receber o resultado da votação.

Conforme citado em [Kirsche93a], o desacoplamento do serviço de voto da aplicação oferece os benefícios de:

(1) livrar a programação da aplicação do gerenciamento das votações. Isto é útil quando existem múltiplas votações simultaneamente em andamento, e contribui para a simplificação da programação da aplicação;

(2) refletir automaticamente a entrada e saída de processos votantes no estado da votação, onde por exemplo, um processo que sai do grupo é automaticamente retirado da lista de votos esperados;

(3) descartar silenciosamente as mensagens após a recepção de um voto de discordância (suficiente para impedir o consenso). Por exemplo, em uma votação para a deleção de todos os objetos desenhados na janela de edição comum, basta que um participante (mesmo que seja o primeiro a votar) discorde da deleção para que a votação possa ser terminada.

O serviço de voto, da mesma forma que o serviço de comunicação de grupo, implementa a entrega de questões e votos de forma confiável e atômica, e com expansão da lista dos destinatários.

4.2.4 Serviço de Transferência de Arquivos Multi-destino

Este serviço permite que documentos armazenados em arquivos sejam recuperados localmente e distribuídos aos demais membros da sessão de edição (para serem apresentadas na janela de

edição comum), sem sobrecarregar a aplicação com operações de empacotamento e desempacotamento. Todos os destinatários enviam 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 preocupar-se com confirmações individuais de cada destinatário. Este serviço deve prover protocolos adequados à transferência de informações volumosas (como imagens, p. ex.).

Como no serviço de comunicação de grupo, este serviço proporciona o transporte de arquivos de forma confiável e atômica e provê a transparência de nomeação de membros de um grupo.

4.3 O SISTEMA ISIS

O suporte descrito no item anterior foi implementado a partir de recursos de comunicação de grupo do sistema ISIS [Birman89-91-92-93]. O sistema ISIS é um conjunto de ferramentas para programação de sistemas distribuídos, e que provê 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 a programação de grupos de processos interativos, em oposição aos padrões um-para-um encontrados no modelo cliente-servidor convencional [Rodden92]. A seguir são apresentadas as características relevantes do sistema ISIS, e a sua utilização no suporte de aplicações do tipo editor distribuído (seção 4.4). A arquitetura do sistema implementado é apresentada no Capítulo 5.

4.3.1 Características Funcionais do Sistema ISIS

O sistema ISIS apresenta, em essência, ferramentas para implementação de grupos de processos virtualmente síncronos e uma coleção de protocolos de difusão confiável com garantias de entrega causal ou total de mensagens. A confiabilidade dos protocolos de difusão do sistema ISIS inclui, além das restrições de ordenação, a atomicidade de entrega de mensagens, e atomicidade de endereçamento do grupo.

Com o sistema ISIS, uma aplicação distribuída pode ser projetada segundo um modelo síncrono, onde o sistema deve escalonar um evento distribuído (como alterações de *membership*, entregas de mensagens e falhas) por vez, garantindo as propriedades de ordenação de eventos e atomicidade acima e assim facilitando significativamente sua programação. Entretanto, o

"sincronismo virtual" apresentado pelo ISIS permite que uma aplicação faça melhor uso do paralelismo inerente aos sistemas distribuídos, do que aquele permitido em implementações puramente síncronas. Com o sincronismo virtual, o escalonamento de eventos tem aparência síncrona, mas as execuções são normalmente concorrentes e as difusões de mensagens podem ser assíncronas (sem bloqueio do emissor).

A seguir são apresentadas as características do sistema ISIS com relação aos grupos de processos que podem ser implementados e os protocolos de comunicação de grupo oferecidos.

Grupos de Processos

No ISIS são permitidos quatro estilos de grupos de processos, que diferem na forma pela qual os processos interagem com o grupo (figura 4.1):

- grupos pares, onde os membros do grupo cooperam proximamente, como no caso de manutenção de réplicas de dados em aplicações cooperativas simétricas.

- grupos cliente-servidor, onde um certo número de clientes interage com um grupo par de servidores, através de padrões *request/reply* para um determinado servidor (RPC) ou para todo o grupo. Apesar do ISIS permitir que processos interajam diretamente com um grupo, ao registrar-se como cliente a comunicação cliente/servidor torna-se mais eficiente (sendo que os clientes de um grupo par não podem monitorar a passagem de mensagens no interior do grupo de servidores).

- grupos de difusão, que são similares aos grupos cliente-servidor porém implementando uma forma especial de comunicação de grupo, o *multicast* de difusão, que é enviado de um processo servidor para todos os clientes. Estes últimos atuam apenas como receptores passivos destas mensagens.

- grupos hierárquicos, onde um grupo maior é implementado por subgrupos menores, cada qual normalmente contendo partições dos dados do sistema. Um cliente, inicialmente interage com um subgrupo de maior hierarquia (grupo raiz), e em seguida é redirecionado para um subgrupo que responderá por requisições sob sua responsabilidade. Também é permitida a comunicação de um cliente com todos os membros da hierarquia.

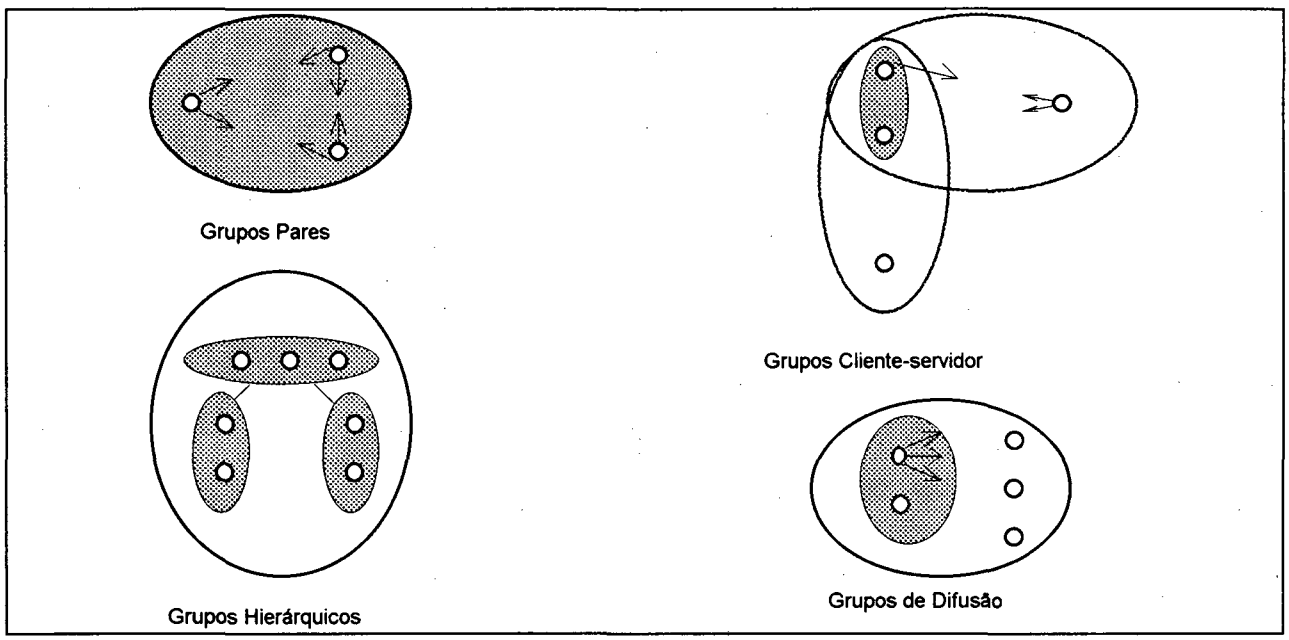


Figura 4.1 - Estilos de grupos no sistema ISIS [Birman91-93]

Para o gerenciamento dos grupos de processos mencionados, o ISIS apresenta funcionalidades que permitem: criação e a destruição de grupos; entrada dinâmica de processos com controle de acesso (através de apresentação/autenticação de credenciais) e transferência de estado; saída dinâmica de processos; monitoração da alteração de *membership*; vigilância de processos; sincronização através de mecanismos de bloqueio e passagem de bastão.

Além disto, o ISIS apresenta facilidades para implementação de aplicações de computação redundante e com recuperação automática de falhas (por armazenamento de históricos do estado do sistema) e, para a comunicação a longa distância confiável.

Comunicação de Grupo

O sistema de comunicação do ISIS oferece primitivas de difusão de mensagens para grupos de processos com diferentes restrições de ordenação de entrega, e que são compatíveis com o modelo de execução virtualmente síncrono. Abaixo são apresentadas estas primitivas em ordem crescente de garantias de ordenação[2]:

[2] Também estão disponíveis no ISIS as primitivas MBCAST e FBCAST. A primeira não oferece nenhuma garantia de confiabilidade (ordenação e entrega atômica, p. ex.) e não pode ser utilizada com o modelo de sincronismo virtual. A segunda oferece ordenação FIFO (*First In First Out*) ponto-a-ponto e apresenta entrega atômica de mensagens.

- CBCAST: oferece garantias de ordenação causal na entrega de mensagens. Com esta primitiva, duas mensagens emitidas por processos em sítios distintos, mas que estejam relacionadas por uma cadeia de transmissão e recepção de mensagens, terão suas entregas ordenadas causalmente. 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.

- ABCAST: oferece garantias de entrega totalmente ordenada de mensagens. Na ordenação total, duas mensagens de difusão quaisquer *A* e *B*, são entregues nos sítios receptores em comum na mesma ordem relativa (ou *A* antes de *B*, ou *B* antes de *A*). A ordenação de mensagens enviadas de forma concorrente, ao contrário do CBCAST, requer que os processos comuniquem-se de maneira síncrona. Cabe observar que apenas mensagens ABCAST concorrentes são entregues totalmente ordenadas, ou seja, mensagens enviadas através CBCASTs e ABCASTs concorrentes, não tem nenhuma garantia de ordenação (causal ou total) na sua entrega. A primitiva ABCAST permite que objetos compartilhados, como no caso de documentos em editores cooperativos, sejam atualizados de forma concorrente sem prejuízo da sua consistência.

- GBCAST: oferece garantias de entrega totalmente ordenada de mensagens, com respeito a GBCASTs, ABCASTs e CBCASTs. Desta forma, esta primitiva é similar ao ABCAST, porém ainda resolve o problema da inexistência da garantia de ordenação total na entrega de difusões executadas através de CBCASTs e ABCASTs concorrentes.

O custo em termos de desempenho dinâmico destas primitivas cresce com as restrições de ordenação apresentadas. O CBCAST é de duas a vinte e cinco vezes mais rápido que o ABCAST, dependendo do grau em que a aplicação é bem sucedida na comunicação assíncrona. As taxas de transmissão com o CBCAST são similares às obtidas com *streams* (comunicação com conexão) do UNIX[3] e TCP (*Transmission Control Protocol)*. O GBCAST é ainda mais caro que o ABCAST, e seu uso é aconselhado apenas em situações onde coexistem CBCASTs e ABCASTs em uma mesma aplicação. De uma forma geral, o uso do CBCAST oferece o melhor desempenho sem comprometer a implementação de aplicações virtualmente síncronas, onde é suficiente apenas garantir-se a causalidade nas mensagens de difusão.

[3] UNIX *is a registered trademark of AT&T.*

4.4 UTILIZAÇÃO DO SISTEMA ISIS NO SUPORTE PROPOSTO

Nesta seção é apresentada a utilização das funcionalidades do sistema ISIS nos serviços do suporte para aplicações do tipo editor distribuído apresentados. No projeto dos serviços a serem implementados, houve a preocupação em aproveitar-se todas funcionalidades já oferecidas pelo ISIS, apenas adicionando os mecanismos não oferecidos, como será visto a seguir. Este mapeamento pode ocorrer devido ao fato de aplicações de Trabalho Cooperativo com as características do editor distribuído apresentado no capítulo anterior, atenderem aos requisitos de aplicações que podem utilizar o modelo de sincronismo virtual [Birman90]:

- os processos não utilizam relógios que mostram tempo real;
- os processos apenas interagem através do ISIS;
- os dados são armazenados em memória volátil.

Detalhes maiores a respeito da implementação do suporte (assim como da aplicação) serão apresentados no próximo capítulo.

4.4.1 Serviço de Gerenciamento de Grupo

A) O Gerenciamento de Grupo no ISIS

No serviço de gerenciamento de grupo, são utilizados os mecanismos providos pelo ISIS para: criação de grupos; entrada e saída dinâmicas de processos no grupo; transferência de estado para processos ingressantes; verificação de credenciais de processos que solicitam entrada no grupo; monitoração da composição do grupo e de processos individuais que compõem o grupo, com relação a entrada e saída de processos. As funcionalidades acima são oferecidas basicamente pelas primitivas "pg_join"[4], e "pg_leave", que permitem respectivamente a entrada e saída de um processo em um grupo. Complementos necessários para a implementação do serviço de gerenciamento de grupo e que envolvem interações entre os membros do grupo são feitos através das primitivas de *multicast* "gbcast" e "cbcast".

[4] Os termos entre aspas (e em caracteres em estilo terminal) são utilizados como identificações das primitivas de interface com o sistema ISIS e indicam o nome da função correspondente na biblioteca do *toolkit* do ISIS [Birman90] [ISISMP86].

As primitivas "pg_join", e "pg_leave" proporcionam a sincronização necessária entre a mudança de *membership* de um grupo e entrega de mensagens de *multicast* de todos os tipos de ordenação, garantindo que eventos como a recepção de um *multicast*, a alteração de *membership* do grupo, a leitura da formação do grupo quando uma mensagem é recebida, sejam percebidos por todos os membros do grupo na mesma ordem. As alterações de *membership* são também percebidas individualmente. Ou seja, se por exemplo dois processos abandonarem o grupo concorrentemente, o grupo perceberá dois eventos distintos, cada um relacionado com a partida de um processo, e ainda, a ordem de saída dos processos será a mesma em todos os membros que permaneceram no grupo.

B) As Primitivas do Suporte Proposto

JoinGroup

A primitiva *JoinGroup* apoia-se basicamente sobre as primitivas do ISIS "pg_join" e "gbcast" conforme o quadro abaixo.

pg_join	Entrada de um processo em um grupo e criação de grupo, transferência e recepção de estado, monitoração da composição do grupo e verificação de credenciais de processos ingressantes.
gbcast	Difusão da identificação do membro ingressante.

Quadro 4.1 - Uso do ISIS na primitiva *JoinGroup*.

O mecanismo de transferência de estado para membros ingressantes é ativado pela primitiva do ISIS "pg_join". Quando um novo membro entra no grupo através da primitiva do suporte *JoinGroup*, este recebe o estado em duas fases:

- primeiramente lhe é transferida uma lista de membros com a identificação de todos os participantes atuais no grupo, e que pode ser lida pela aplicação através da primitiva do suporte *GetGroupMembers*;
- em seguida, lhe são transferidas as informações de estado relacionadas com a aplicação cooperativa.

A identificação de cada membro que compõe a lista de membros, associa às informações do processo ingressante (como PID do UNIX, número e nome do sítio em que se encontra), um nome simbólico (o *login name* do usuário, p. ex.), que é útil para a identificação de participantes de uma sessão de Trabalho Cooperativo.

O "gbcast" é utilizado, após a transferência de estado ter-se completado, para difundir ao grupo a identificação do novo processo. Ao receber esta mensagem de difusão, os membros existentes (inclusive o ingressante) atualizam sua lista de membros. Adotou-se aqui o "gbcast" para que a lista de membros esteja ordenada totalmente, e também para garantir que a atualização desta lista esteja sincronizada com a entrega de qualquer outra mensagem enviada ao grupo.

Assim como a transferência de estado, a monitoração do grupo e a verificação de credenciais de processos ingressantes também são ativadas pela primitiva "pg_join".

Leave

A saída de um membro de um grupo ocorre através da primitiva do ISIS "pg_leave". Após a execução desta primitiva, o mecanismo de monitoração de grupos do ISIS (ativado pela primitiva "pg_join") informa a todos os outros membros da alteração de sua composição, que por sua vez atualizam suas listas de membros, retirando desta o processo que saiu. Com o uso deste mecanismo de monitoração, os membros do grupo são informados mesmo nos casos em que processos saem do grupo de forma involuntária (por falha, p. ex.), ou seja, sem que seja invocada a primitiva *Leave* explicitamente.

GetGroupMembers

Um processo pode verificar a composição atual do grupo a que pertence a partir da informação contida na lista de membros construída com a entrada (através de *JoinGroup*) e saída (através de *Leave* ou involuntária) de processos do grupo.

Invite

O convite para o ingresso de algum processo em um determinado grupo é feito através da difusão de uma mensagem "cbcast" para um "grupo básico" de processos, identificando simbolicamente o usuário a ser convidado. Este grupo básico é criado pelo suporte (através da primitiva "pg_join") quando este é inicializado pelo processo de aplicação e compõe-se de todos os processos que têm interesse em vir eventualmente a ser membros de algum outro grupo de aplicações cooperativas

(por iniciativa própria, através da primitiva *JoinGroup*) ou do grupo no qual encontra-se o processo convidante (através do recebimento de um convite, como na figura 4.2). Somente o processo pertencente ao grupo básico e cuja identificação coincide com a do processo convidado recebe o convite. A decisão em aceitar ingressar, ou não, no grupo é feita a nível de aplicação (podendo envolver o participante convidado).

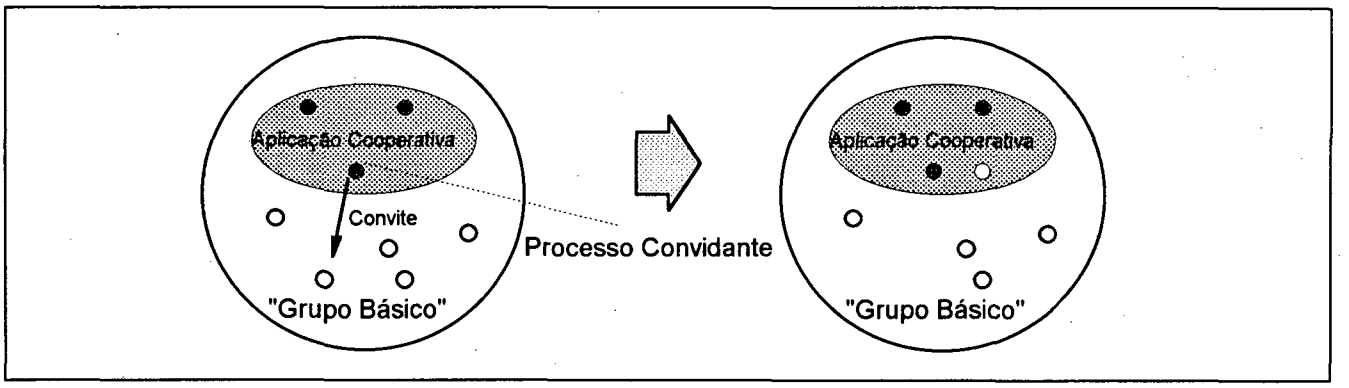


Figura 4.2 - Convite para o ingresso de um processo em uma aplicação cooperativa.

Expel

De forma similar à primitiva *Invite*, o pedido de expulsão de um processo existente em um determinado grupo é feito através da difusão de uma mensagem "cbcast" identificando simbolicamente o usuário a ser expulso. Entretanto esta mensagem é enviada para o grupo ao qual pertencem ambos os processos (o requisitor da expulsão e o a ser expulso) (figura 4.3). O suporte verifica se a identificação do processo coincide com a recebida. Caso coincida, envia então este pedido ao processo para que este reaja saindo, ou não, do grupo. Caso contrário, o suporte ignora a mensagem recebida.

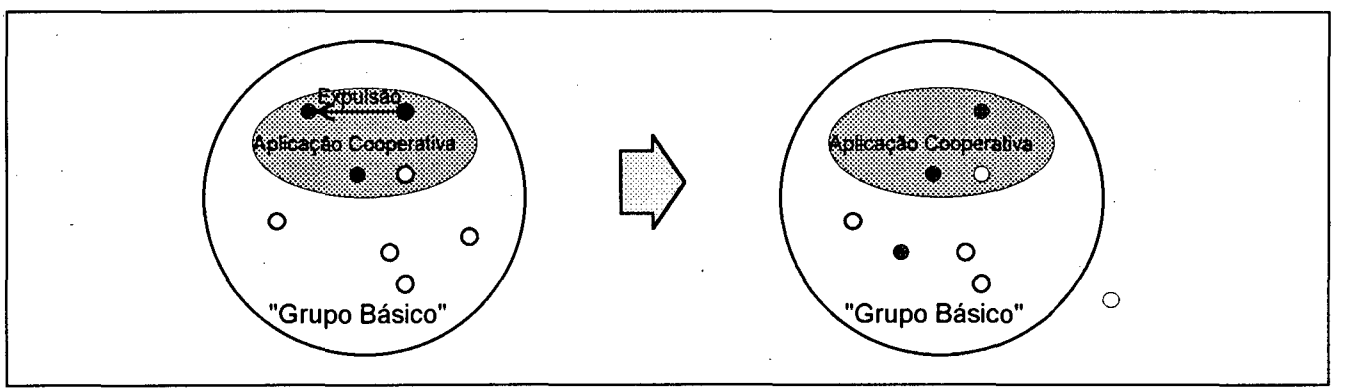


Figura 4.3 - Pedido de expulsão de um processo de uma aplicação cooperativa.

4.4.2 Serviço de Comunicação de Grupo

O serviço de comunicação de grupo baseia-se na primitiva de difusão de mensagens "cbcast" do sistema ISIS. Além de garantir a ordenação causal na entrega de mensagens que respeitem a relação ocorre antes e permitir o envio assíncrono destas mensagens, conforme apresentado anteriormente, a primitiva "cbcast" oferece difusão confiável, entrega atômica de mensagens (*delivery atomicity*) e transparência de nomeação de grupos, que permitem sua utilização na implementação de aplicações virtualmente síncronas.

A ordenação total oferecida pelo "abcast", oferece às aplicações cooperativas garantias de atualizações consistentes de réplicas dos dados, como ocorre com os objetos que compõem um documento em edição cooperativa. Com o uso do "cbcast" no serviço de comunicação de grupo, a consistência de objetos atualizados concorrentemente deve ser garantida por mecanismos de coordenação (controle de concorrência) implementados a nível de aplicação, através dos serviços de voto ou mesmo através do serviço de comunicação de grupo. Mesmo considerando a complexidade adicionada na implementação da aplicação que decorre deste fato, o menor custo em desempenho da primitiva "cbcast" prevaleceu na escolha da primitiva mais adequada.

4.4.3 Serviço de Voto

O envio da questão a ser votada, os votos de cada participante, e a divulgação do resultado da votação são também executados através da primitiva "cbcast". Estes envios são assíncronos. Desta forma, por exemplo, o processo emissor da questão não é bloqueado a espera dos votos dos participantes, que são por sua vez recolhidos pelo suporte. Apenas quando a votação termina o emissor é notificado de seu resultado através da invocação da função de avaliação definida pelo processo requisitor da votação, que recebe como parâmetros os números de votos a favor e contra. O uso do "cbcast" é responsável pela entrega confiável e atômica, e pela expansão de endereços oferecidas pelas primitivas do serviço de voto.

Apesar da primitiva "cbcast" apresentar propriedades de endereçamento virtualmente síncronas, onde todos os receptores de uma mensagem de difusão tem a mesma informação de *membership* ao receber esta mensagem, permanece uma questão a ser resolvida pelo serviço de voto. Devido ao fato do envio dos votos ser assíncrono, podendo ainda depender de interações entre o processo receptor e o usuário (o que pode tomar um tempo indeterminado), pode ocorrer a entrada (ou a saída) de um novo processo ao grupo, antes que a votação seja encerrada (figura 4.4-a). Com entrada de um processo, mesmo que todos os processos votantes existentes quando

foi iniciada a votação enviem seu votos, o requisitor poderá ficar esperando indefinidamente pelo voto do novo processo, dado que este sequer recebera a questão, resultando em não finalização da votação.

A forma mais simples de contornar-se este problema, seria apenas ignorar a entrada do novo processo na contabilização do consenso, permitindo que o serviço de voto termine a votação com os votos até então recolhidos. Esta abordagem não é aconselhável por motivos de simetria, dado que o processo que juntou-se ao grupo já recebeu o estado da sessão de trabalho cooperativo, tendo conseqüentemente o direito de influir em votações que dizem respeito a esta sessão.

Para resolver este problema, utilizou-se a funcionalidade do ISIS para monitoração de processos, através da primitiva "pg_monitor". Esta primitiva permite que o serviço de voto seja informado a cada entrada (ou saída) de um novo processo. Com esta informação o serviço de voto pode então enviar a questão para o processo ingressante, que deverá eventualmente enviar seu voto, permitindo assim a correta finalização da votação. Da mesma forma, poderá também excluir da lista de processos votantes o membro que saiu.

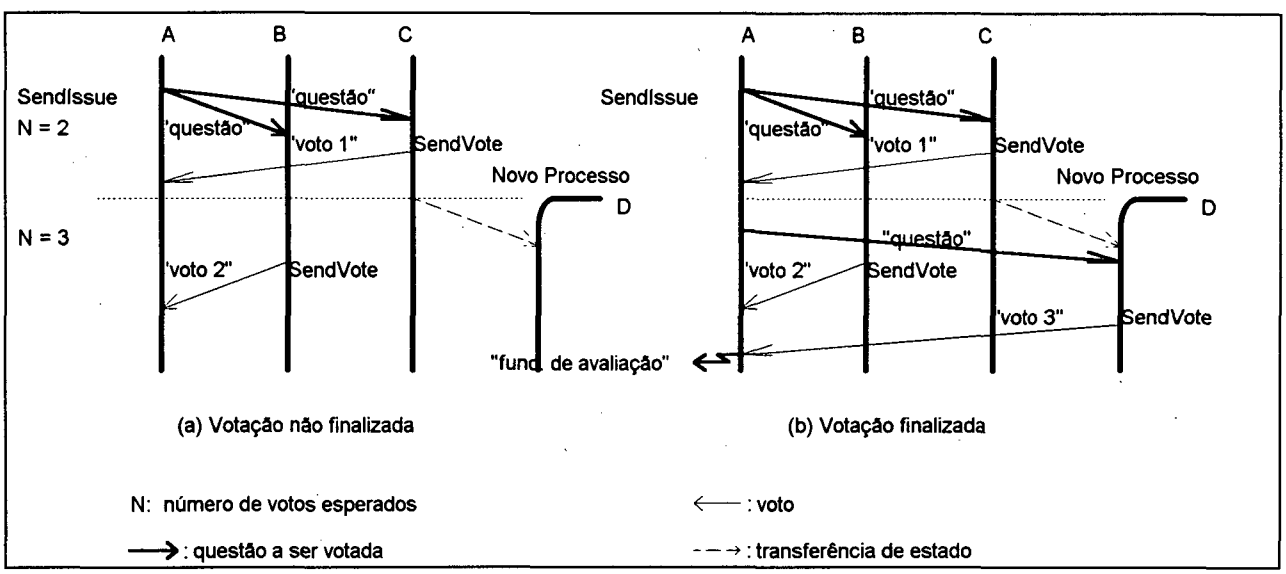


Figura 4.4 - Entrada de um processo antes do término de uma votação: (a) caso onde o requisitor não pode terminar a votação; (b) caso onde o novo membro é requisitado também para votar, permitindo assim a finalização da votação.

4.4.4 Serviço de Transferência de Arquivos Multi-destino

Devido ao fato da versão atual do editor distribuído não suportar edição de imagens, este serviço não está ainda sendo utilizado por não ser necessário. A transferência de documentos em edição

para os processos ingressantes está sendo feito através do mecanismo de transferência de estado provido pelo serviço de gerenciamento de grupo.

4.5 COMPARAÇÕES COM TRABALHOS RELACIONADOS

Neste item são discutidas as diferenças relevantes entre a proposta de suporte deste trabalho e as propostas apresentadas em [Kirsche93a], [Pendergast93] e [Ravindran92], descritas no Capítulo 2. Estas propostas foram escolhidas para esta discussão por serem as que mais se enquadram dentro do contexto de aplicações cooperativas baseadas em mídias discretas, em especial para o caso de editores distribuídos.

4.5.1 Plataforma de Comunicação de Grupo de Kirsche

Conforme mencionado anteriormente, o suporte proposto neste trabalho baseia-se na proposição de serviços da plataforma de comunicação de grupo apresentada em [Kirsche93a], utilizada no editor de desenhos CoDraft. Entretanto, estas propostas diferenciam-se basicamente pelos modelos de sistema de comunicação adotados. A implementação aqui proposta faz uso, no serviço de comunicação de grupo, 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 se ordenar causalmente *multicasts* enviados por instâncias do editor, oferecendo apenas a garantia de ordenação FIFO ponto-a-ponto nos envios provenientes de uma mesma fonte. 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 mesmo usuário e também de usuários distintos.

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 (conforme será apresentado no Capítulo 5), que nos parece mais adequado para o tipo de aplicação em questão.

Enfim, os mecanismos de recuperação de falhas apresentados pelo ISIS, permitem também a entrega atômica de mensagens, 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 [Kirsche93a] 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 suporte de aplicações do tipo editor distribuído.

4.5.2 Objetos Canais para Sistemas de Suporte de Grupos

A arquitetura baseada em Objetos Canais para Sistemas de Suporte de Grupos proposta em [Pendergast93], resolve também o problema de comunicação de grupo confiável, no sentido de que provê mecanismos para implementação de conexões 1:N. A expansão da lista de destinatários é feita no momento em que um Objeto Canal é criado, através da comunicação deste evento para um objeto servidor de nomes (NSO) que controla as conexões de sessão entre os participantes (onde existem Objetos Canais ativos). A comunicação de grupo oferecida, entretanto, também não oferece mecanismos de entrega atômica de mensagens, nem permite a mudança de listas de endereços por razões de falha. Além disto, a entrega de mensagens é feita no máximo com garantias de ordenação FIFO ponto-a-ponto.

Uma característica adicional deste método é a interface baseada em objetos apresentada pelos Objetos Canais, que permite o encapsulamento das funcionalidades dos sistemas de comunicação sobre os quais são implementados as aplicações cooperativas. Esta característica é importante pois vem contribuir na diminuição da complexidade da programação das aplicações e permite que atualizações no suporte impliquem em poucas modificações no código destas aplicações. Com o intuito de fazer-se uso destas características, escolheu-se implementar o suporte proposto neste trabalho segundo a mesma tendência deste caso, encapsulando parte de seus serviços em classes de objetos, conforme será discutido no próximo capítulo.

4.5.3 Primitiva O_Send

A estrutura de comunicação apresentada em [Ravindran92] permite que 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, possam ser ordenadas de forma total (no caso de atividades concorrentes) ou causal, através da primitiva O_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 evitar-se que entradas e saídas dinâmicas de participantes sejam percebidas nos demais participantes em meio a entrega de eventos que compõem uma atividade (entrega atômica de *clusters*).

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. Utilizando-se o CBCAST, todas as mensagens enviadas por um mesmo processo tem sua entrega ordenada causalmente, o que não é sempre necessário em aplicações cooperativas. Entretanto o ISIS oferece a primitiva FBCAST que oferece entrega atômica e com ordenação FIFO ponto-a-ponto das mensagens, que poderia ser utilizada nos casos onde se pretende obter custos em desempenho ainda menores que os obtidos com o CBCAST.

4.6 CONCLUSÕES

Neste capítulo foram apresentados os serviços de uma plataforma de suporte para aplicações cooperativas do tipo editor distribuído. O suporte proposto também pode ser aplicado em sistemas com características similares aos editores distribuídos, ou seja, aplicações baseadas principalmente em interações síncronas e implícitas (utilizando mídias discretas) entre grupos dinâmicos de participantes. Em seguida foi apresentado o sistema ISIS que foi utilizado na implementação dos serviços e as funcionalidades deste sistema que foram aproveitadas. Ao final o suporte proposto foi comparado com outras propostas de suporte disponíveis na literatura (e apresentadas no Capítulo 2).

As comparações feitas permitem verificar-se dois aspectos importantes. Primeiramente, o confronto dos serviços propostos com as soluções já implementadas anteriormente, permite concluir que as primitivas dos serviços propostos satisfazem plenamente aos principais requisitos da aplicação cooperativa em questão (editor distribuído), que são: o gerenciamento de grupo de processos e a interação de participantes através de mecanismos de voto e de comunicação de grupo, em ambientes onde os participantes podem entrar ou sair de forma autônoma de sessões de Trabalho Cooperativo. Além disto, a utilização do sistema ISIS na implementação do suporte proposto, por oferecer mecanismos que garantem a entrega com ordenação causal e atômica de mensagens (tolerância a faltas), permite que as aplicações sejam implementadas a partir de serviços mais adequados e que vão diminuir efetivamente a complexidade da sua programação.

CAPÍTULO 5

IMPLEMENTAÇÃO DO EDITOR DISTRIBUÍDO E DO SEU SUPORTE

5.1 INTRODUÇÃO

Este capítulo tem como meta apresentar as implementações dos serviços da plataforma de suporte e do editor distribuído propostos neste trabalho.

Os projetos da plataforma de suporte e do editor distribuído que serão apresentadas, foram realizadas a partir do paradigma de orientação a objetos [Coad92]. Os serviços da plataforma de suporte e a aplicação cooperativa foram concebidas segundo este paradigma visando uma definição mais clara dos seus módulos e de suas interações, conduzindo adicionalmente de forma natural a uma codificação destes módulos em uma linguagem de programação orientada a objetos. Entretanto, como será discutido posteriormente neste capítulo, nem todos os módulos foram implementados na prática usando a abordagem orientada a objetos, pois para os mecanismos de suporte para a interface gráfica e para a comunicação utilizou-se facilidades existentes (nos sistemas X Window[1] e ISIS) que foram implementadas para serem utilizadas em linguagens procedurais. A nível de projeto, as classes de objeto são representadas conforme a notação apresentada no Anexo A. Esta notação será utilizada neste documento para representar a utilização das respectivas classes. As classes implementadas (codificadas na linguagem de programação C++ [C++a] [C++b]) estão apresentadas nos Anexos B e C.

5.2 ARQUITETURA DA IMPLEMENTAÇÃO

Conforme visto no Capítulo 2, foi adotada neste trabalho a arquitetura em três níveis para sistemas de Trabalho Cooperativo que é apresentada na figura 5.1. A plataforma de suporte implementada localiza-se em um nível intermediário, fornecendo funcionalidades a aplicações

---

[1] X Window System *is a trademark of the Massachusetts Institute of Technology.*

cooperativas (em especial o editor distribuído), e utilizando mecanismos de comunicação e gerenciamento de grupo do sistema ISIS.

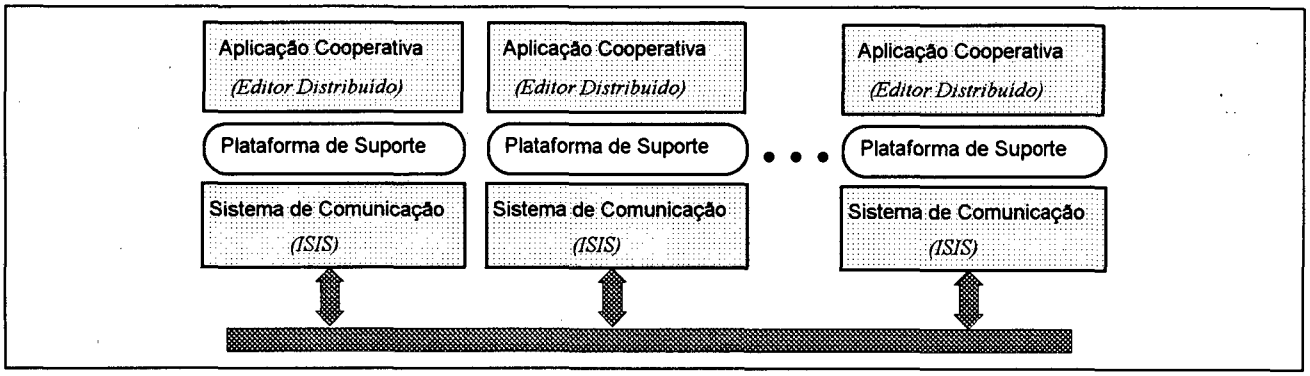


Figura 5.1 - Arquitetura dos Sistemas de Trabalho Cooperativo

5.3 IMPLEMENTAÇÃO DA PLATAFORMA DE SUPORTE

5.3.1 Uso da Orientação a Objetos na Implementação

O objetivo principal do uso de orientação a objetos na implementação dos serviços da plataforma de suporte foi o encapsulamento das funcionalidades do sistema de comunicação. Este encapsulamento deve permitir que o suporte seja implementado sobre diferentes sistemas de comunicação, sem que o código da aplicação sofra alterações significativas, em resultado a escolhas de diferentes primitivas de comunicação. Adicionalmente, com uso de orientação a objetos seria possível obter-se os benefícios da modularidade oferecida por este modelo de programação.

Entretanto, por razões de adequação da implementação com as bibliotecas e o sistema de execução[2] do *toolkit* do sistema ISIS utilizado, algumas características da interface do suporte foram mantidas na forma procedural, conforme é oferecido pelo ISIS. Também nos casos em que as primitivas do ISIS ofereciam de forma direta as funcionalidades requeridas, deixou-se de implementar os serviços a partir da abordagem orientada a objetos.

Desta forma, a camada correspondente à aplicação cooperativa poderá acessar os serviços de suporte de duas formas:

[2] Que baseia-se essencialmente em chamadas a funções (interface procedural) de uma biblioteca ligada estaticamente à aplicação, e notificação de eventos através de um mecanismo de escalonamento de *lightweight tasks* [Birman90].

1) Através de métodos de classes de serviços, implementadas para o Serviço de Gerenciamento de Grupo e Serviço de Voto;

2) Diretamente a partir das funcionalidades do ISIS, no caso do Serviço de Comunicação de Grupo. O *toolkit* do sistema ISIS oferece suas funcionalidades através de uma interface procedural a uma biblioteca de funções, a partir da qual uma aplicação pode requisitar operações (como o envio de uma mensagem de difusão, p. ex.), e definir manipuladores de funções que são escalonados pelo ISIS por ocasião da recepção de eventos originados dentro do grupo de processos ao qual a aplicação pertence (como a recepção de uma mensagem de difusão, p. ex.).

5.3.2 Arquitetura do Suporte Implementado e Interfaceamento com a Aplicação

Na figura 5.2 mostra-se como objetos que compõem uma aplicação cooperativa, como p. ex. módulos de controle de um *telepointer*, atualizações da janela de edição comum e controle de concorrência, podem fazer uso de objetos instanciados a partir das classes de serviços de suporte da plataforma proposta, assim como de funcionalidades do sistema ISIS.

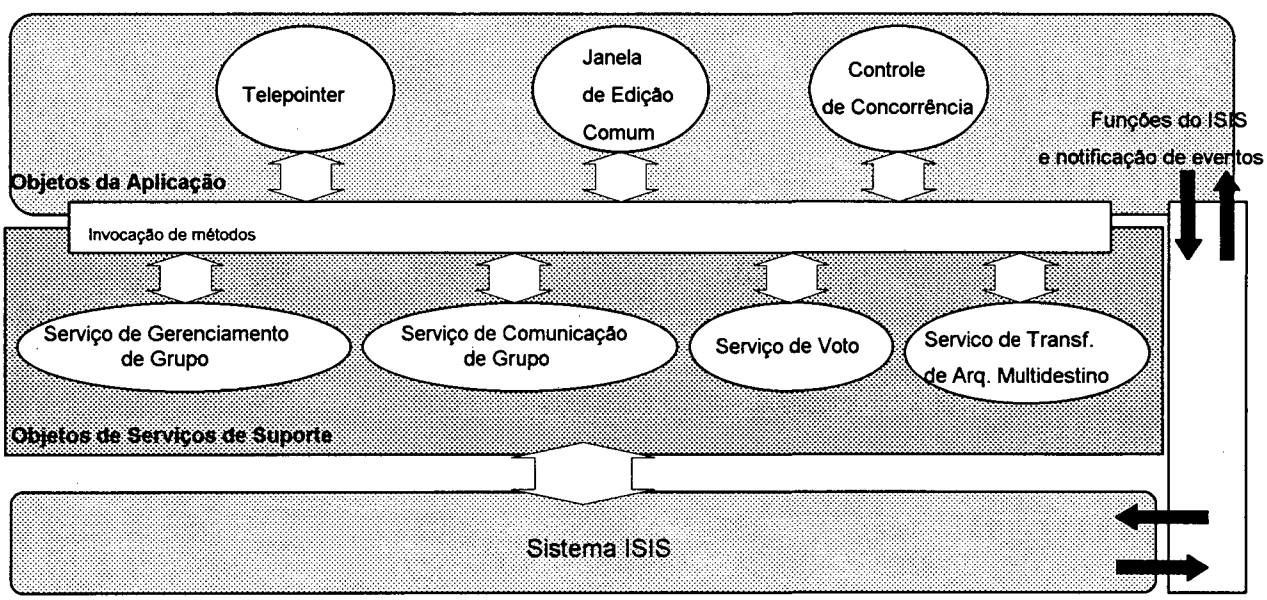


Figura 5.2 - Arquitetura da implementação de uma aplicação cooperativa sobre a plataforma de suporte proposta.

Nos itens a seguir estas formas de interfaceamento entre uma aplicação cooperativa e o suporte serão apresentadas em maior detalhe.

A) Classes de Objetos

Cada um dos serviços da plataforma de suporte é representado, a nível de projeto, por uma classe de objeto, conforme é mostrado na figura 5.3. As classes Serviço de Gerenciamento de Grupo e Serviço de Voto foram codificadas em C++ (Anexo B) e oferecem, através de seus métodos, as funcionalidades do serviço correspondente (como as primitivas *Leave* e *Expel*, p. ex.).

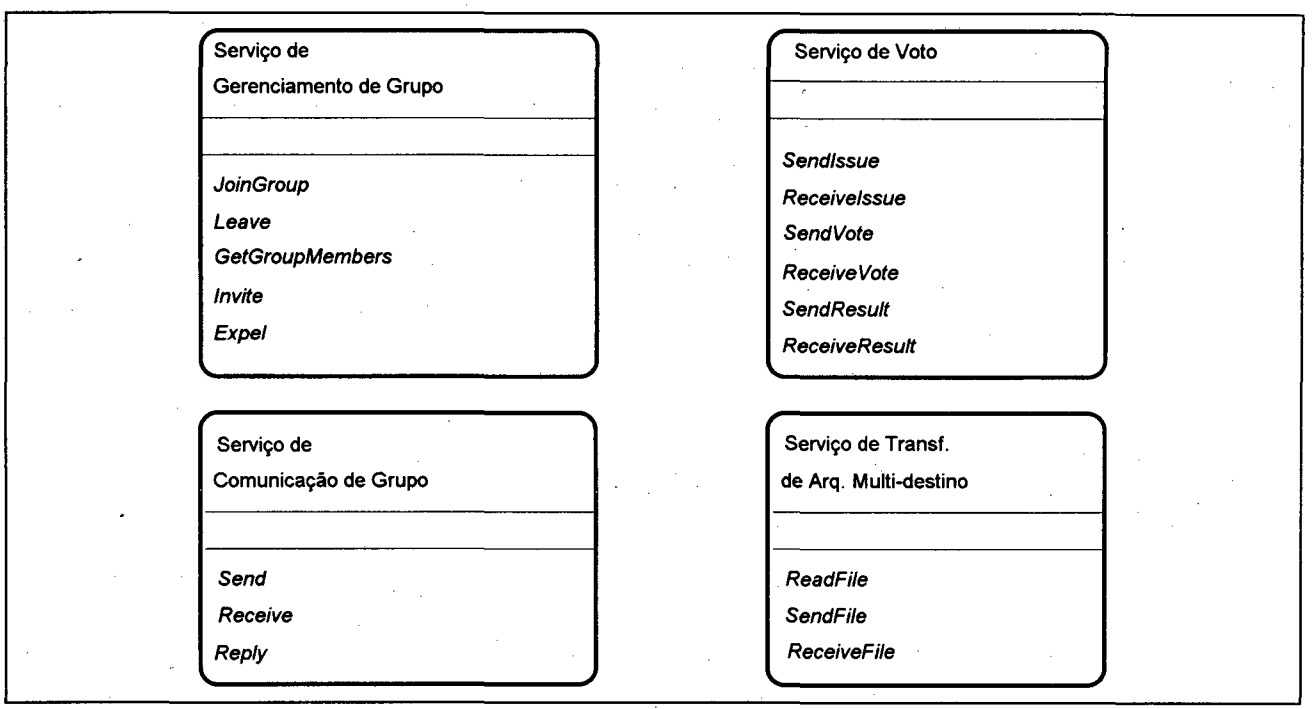


Figura 5.3 - Representação das classes dos serviços da plataforma de suporte.

B) Notificação de Eventos no ISIS

No ISIS, toda a execução ocorre através de um mecanismo de escalonamento de *lightweight tasks* [Birman90][3].Os eventos que podem ocorrer dentro de um grupo de processos, como alteração dinâmica de configuração de um grupo, entrega de mensagens de *multicast*, recepção de estado, podem ser tratados através de manipuladores de funções ("*callback routines*" ou "funções de retorno de chamada") definidos pela aplicação e que são ativados como *lightweight tasks* sempre que um destes eventos ocorre. Estes manipuladores podem receber como argumentos informações relacionadas com o evento que ocorreu (como p. ex., a nova configuração do grupo,

[3] O mecanismo de escalonamento de *lightweight tasks* do ISIS V2.1 é completamente externo ao sistema operacional (UNIX). Assim, mesmo considerando que múltiplas tarefas (*tasks*) possam coexistir em uma mesma área de endereçamento de um processo, o escalonamento é feito por processo. Desta forma, se uma tarefa bloquear (por operações de I/O, p. ex.), todo o processo bloqueia-se também.

a mensagem recebida ou o estado atual da aplicação distribuída). A figura abaixo ilustra o funcionamento da notificação de eventos no ISIS onde, inicialmente os processos que compõem um grupo registram uma função de retorno de chamada "msg_proc" para o tratamento de mensagens recebidas (através da primitiva "isis_entry" do ISIS) e em seguida, estas funções são ativadas pelo recebimento de uma mensagem "cbcast". Como pode-se observar, por motivos de simetria esta função é também ativada no emissor do "cbcast".

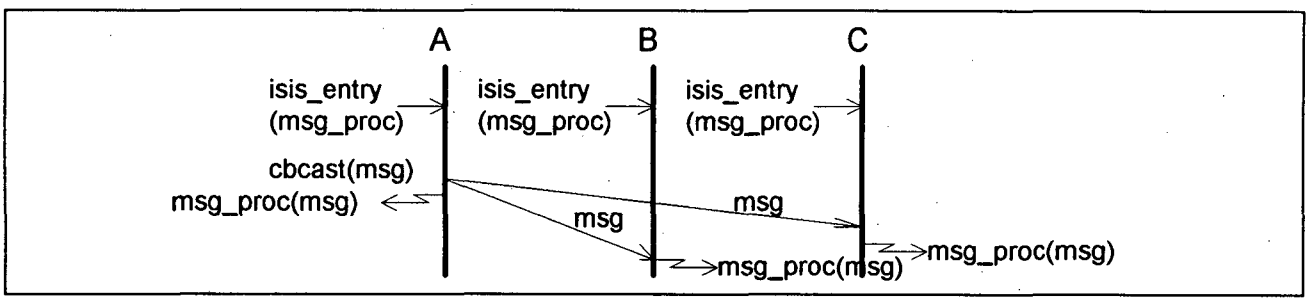


Figura 5.4 - Notificação de eventos no ISIS.

No suporte implementado, esta interface de notificação de eventos através de manipuladores foi mantida, e é utilizada nos serviços conforme é apresentado a seguir.

5.3.3 Implementação dos Serviços da Plataforma de Suporte

5.3.3.1 Serviço de Gerenciamento de Grupo

Para utilizar as primitivas do Serviço de Gerenciamento de Grupo é necessário inicialmente criar um objeto desta classe. Nesta implementação cada objeto pode gerenciar um grupo de processos e uma aplicação deve estar em um grupo de processos apenas[4]. Na criação de um objeto da classe Serviço de Gerenciamento de Grupo, é verificada a existência do "grupo básico" que é utilizado para envio de convites de união ao grupo a outros processos (ver Capítulo 4). Se este grupo existir, o processo une-se ao grupo. Caso contrário ele cria um "grupo básico" novo. Ainda na criação do objeto são registradas as funções de retorno de chamada para recepção de convites, pedidos de expulsão e recepção de identificação de membros ingressantes em um grupo de Trabalho Cooperativo.

Após a criação deste objeto, podem então ser acessadas as primitivas do serviço através de mensagens aos métodos que as implementam.

---

[4] Esta é uma restrição que poderá ser eliminada posteriormente dado que o sistema ISIS permite sobreposição de grupos de processos.

Os mecanismos de envio e recepção de estado, e autenticação de credenciais providos pela primitiva *JoinGroup* são implementados também através de funções de retorno de chamada, e são escalonadas pelo sistema de *lightweight tasks* do ISIS quando ocorrem os eventos a elas associados. Por exemplo, quando algum processo que está em um grupo recém criado percebe a entrada de um novo processo, é ativada neste processo a função de retorno de chamada que recebe como argumento as credenciais que foram submetidas pelo processo ingressante. Dependendo do valor retornado por esta função (definido pelo suporte em função das credenciais submetidas) é permitida ou não a entrada do novo processo no grupo.

5.3.3.2 Serviço de Comunicação de Grupo

Na implementação, o Serviço de Comunicação de Grupo foi mapeado diretamente sobre as funcionalidades de comunicação de grupo com ordenação causal do sistema ISIS (primitiva CBCAST). Não houve um encapsulamento deste serviço em uma classe específica devido ao fato desta primitiva do ISIS já prover as funcionalidades pretendidas, ou seja: envio de forma confiável e atômica de mensagens para um grupo de processos, com transparência de nomeação e com garantias de ordenação causal na entrega desta mensagem.

Para a recepção de uma mensagem é necessário que seja associada uma função de retorno de chamada a um identificador de uma entrada do ISIS (pela primitiva "isis_entry()" do ISIS). Esta função é ativada em todos os processos do grupo (conforme foi ilustrado na figura 5.4) quando da recepção de uma mensagem referenciando este identificador de entrada.

5.3.3.3 Serviço de Voto

O Serviço de Voto, de forma similar ao Serviço de Gerenciamento de Grupo, foi implementado como uma classe de objetos. A classe que implementa este serviço encapsula as funcionalidades definidas no Capítulo 4, a partir da primitiva CBCAST (para o envio de questões, votos e resultado da votação), e dos mecanismos de ativação de entradas para recepção de mensagens e de monitoração de processos oferecidos pelo ISIS.

Com a utilização destas funcionalidades do ISIS, o Serviço de Voto oferece a possibilidade de estabelecimento de consenso baseado em primitivas de envio de questões e votos assíncronas, no sentido de que o processo votante não precisa emitir seu voto imediatamente após a recepção de uma questão, nem o processo iniciador fica bloqueado a espera dos votos. Adicionalmente, uma nova votação pode ser iniciada mesmo que alguma outra (iniciada pelo mesmo processo) ainda não tenha sido finalizada. O envio de questões e de votos de forma assíncrona, apesar de permitir

situações onde uma votação possa consumir um período de tempo demasiadamente longo, faz-se necessário nos casos onde o envio de votos envolva interações com usuários humanos (cujas durações poderiam causar bloqueios indesejáveis no processo iniciador).

As primitivas de monitoração de processos do ISIS, permitem também que o processo que iniciou uma votação considere a saída e entrada de processos de forma concorrente, na contabilização do número de votos esperados para obtenção de consenso.

A figura abaixo ilustra o funcionamento do Serviço de Voto (onde o número de processos no grupo permanece constante) mostrando o uso dos métodos da classe Serviço de Voto para o envio da questão (*SendIssue*), dos votos (*SendVote*) e do resultado (*SendResult*) e, a ativação de funções de retorno de chamada definidas pela aplicação para o tratamento destas mensagens (*ReceiveIssue* e *ReceiveResult*)[5]. Como pode ser visto, a "função de avaliação" é apenas acionada após o término da votação. A *issue_id* contém toda informação com relação a votação iniciada, como identificador local (que permite a iniciação de votações simultâneas), e a definição semântica da questão a ser votada (como "deletar todos os objetos", p. ex.).

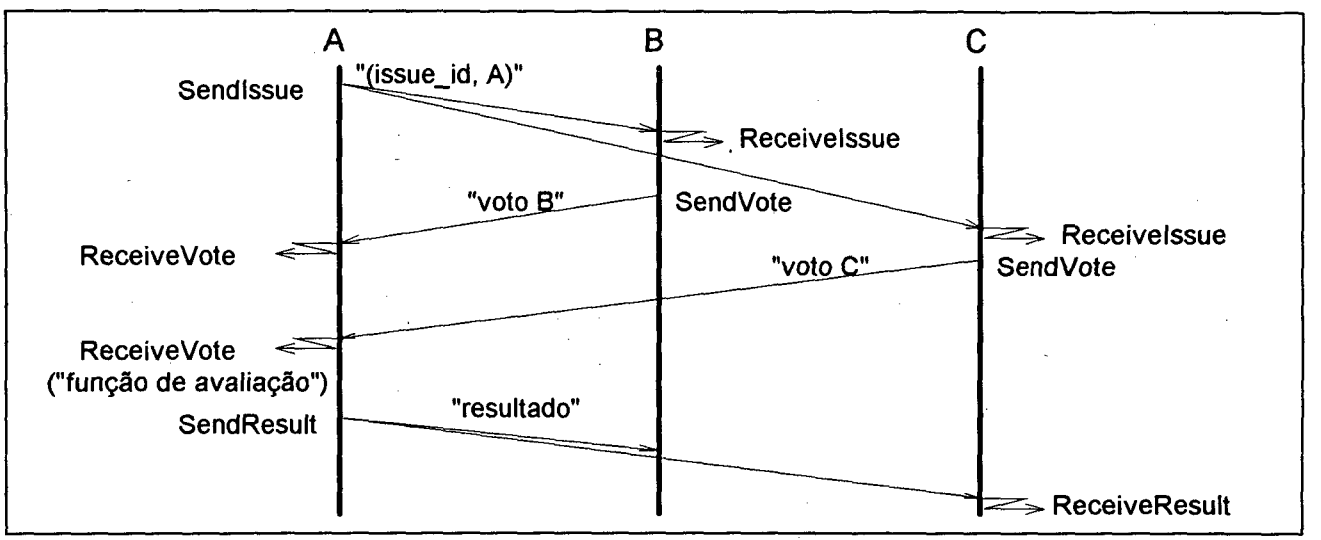


Figura 5.5 - Serviço de Voto

5.3.3.4 Serviço de Transferência de Arquivo Multi-destino

O Serviço de Transferência de Arquivo não está disponível na versão atual do suporte. O principal motivo para isto foi o fato de não se estar manipulando grandes volumes de informações

[5] A ativação de forma simétrica das funções de retorno de chamada implementada pelo ISIS (conforme mostrado na figura 5.4) é mascarada pelo Serviço de Voto. Assim, as primitivas para o tratamento de questões e de votos, não são ativadas nos emissores das questões e dos votos, respectivamente.

no editor, como seria o caso no uso de imagens (padrão GIF, p. ex.). A transferência de documentos está sendo feita através do mecanismo de transferência de estado provido pelo ISIS e ocorre apenas na entrada de um processo em um grupo.

5.4 IMPLEMENTAÇÃO DO EDITOR DISTRIBUÍDO

A aplicação cooperativa implementada baseou-se nas características funcionais do editor distribuído de figuras geométricas apresentado no Capítulo 3. Os módulos que compõem esta aplicação foram projetados como classes de objetos, que interagem entre si para compor de forma modular as funcionalidades do editor, conforme apresentado no diagrama de Coad-Yourdon[6] [Coad92] da figura 5.6.

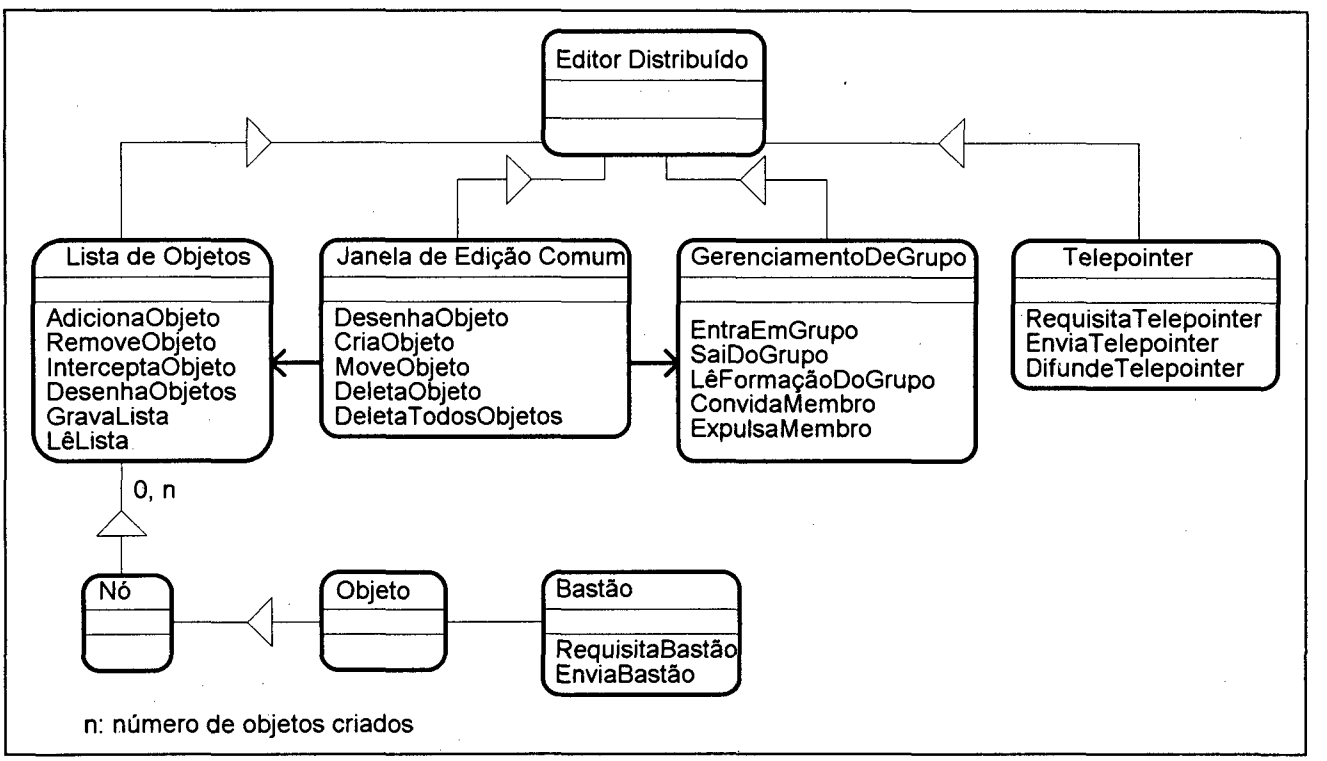


Figura 5.6 - Representação Coad-Yourdon das classes de objetos que compõem o Editor Distribuído.

Assim, cinco principais classes de objetos constituem a aplicação implementada: Lista de Objetos que compõem o desenho em edição; Bastão (*token*) para controle de concorrência no acesso aos

[6] A notação utilizada nos diagramas de Coad-Yourdon estão apresentados no Anexo A.

objetos que compõem a Lista de Objetos; Janela de Edição Comum; Ponteiro de *mouse* distribuído (*telepointer*); e Gerenciamento de Grupo.

Na implementação dos módulos baseados nas classes Lista de Objetos, Bastão e Telepointer, foi utilizada também a abordagem de orientação a objetos, através da codificação destas classes na linguagem de programação C++ (Anexo C). Entretanto, na implementação da Janela de Edição Comum adotou-se a abordagem procedural da linguagem C pelo fato de procurar-se seguir as características de programação utilizadas pelo ambiente XView[7] [Heller90] no que se refere ao tratamento de eventos da aplicação (movimentos e aperto de botões do *mouse*, seleções de menus, etc.). O tratamento de eventos no XView dá-se através de um Notificador de Eventos (*Notifier*), que permite que uma aplicação seja notificada da ocorrência de eventos através da invocação de funções de retorno de chamada previamente registradas. A classe de Gerenciamento de Grupo foi implementada com as primitivas oferecidas pelo suporte, através de um objeto da classe Serviço de Gerenciamento de Grupo.

5.4.1 Lista de Objetos

No editor distribuído implementado, um documento cooperativo é o resultado da edição (através de operações de criação, deleção e movimento) de um conjunto de objetos que representam figuras geométricas básicas, como segmentos de reta, retângulos, polígonos, círculos e elipses. Cada um destes tipos de figuras é definido por uma classe específica, que define métodos para: criação e destruição de objetos; desenho, apagamento e movimentação de figuras geométricas na Janela de Edição Comum; leitura e escrita de atributos dos objetos criados (como p. ex., largura, altura, posição relativa na Janela de Edição Comum, etc.).

À medida que uma sessão de edição transcorre e são acrescidas, retiradas ou modificadas figuras geométricas do documento, são, respectivamente, criados objetos partir das classes correspondentes, e deletados ou modificados objetos já instanciados. A cada objeto criado a partir de uma figura desenhada, é associado um identificador globalmente único[8]. Isto faz-se necessário para evitar que objetos distintos criados em diferentes instâncias do editor tenham eventualmente a mesma identificação.

Para o gerenciamento destas operações sobre o documento cooperativo foi então definida uma classe Lista de Objetos.

---

[7] XView *is a trademark of Sun Microsystems, Inc.*

[8] A identificação globalmente única de um objeto é conseguida compondo um identificador como uma tripla <identificador local, identificador do processo UNIX, número do sítio>.

Esta classe oferece métodos para adição e remoção de uma figura na lista, detecção de intersecção do ponteiro do *mouse* com a borda de alguma figura do documento (para seleção de um objeto), desenho das figuras na Janela de Edição Comum, e gravação/leitura dos itens da Lista de Objetos em um arquivo (conforme figura 5.6). Além disto, a classe Lista de Objetos oferece o acesso aos métodos das classes de figuras geométricas que a compõem.

A classe Lista de Objetos implementa a semântica de uma lista encadeada simples [Sobelman89], o que permite uma manipulação eficiente das figuras geométricas desenhadas, a partir de alocação/liberação dinâmica de espaço em memória para os nós e itens da lista (figura 5.7).

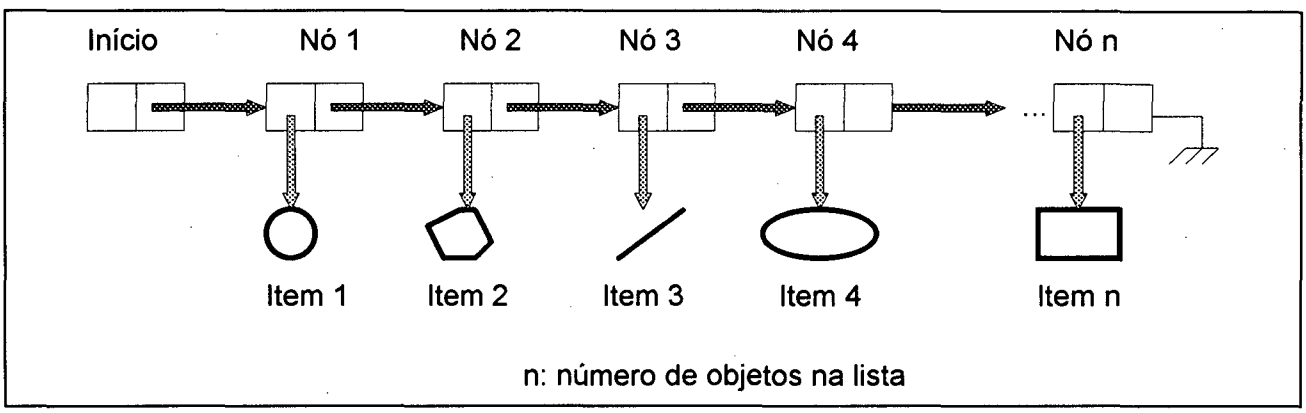


Figura 5.7 - Lista encadeada de objetos.

Todas as instâncias do editor contém uma cópia da Lista de Objetos. Os itens desta lista podem ser criados e atualizados pelos usuários de cada instância de forma autônoma através das Janelas de Edição Comum.

5.4.2 Controle de Concorrência

A autonomia dos usuários, citada anteriormente, pode levar o documento cooperativo a um estado inconsistente, caso ocorram modificações concorrentes e conflitantes sobre um mesmo objeto. Desta forma faz-se necessário um mecanismo que garanta que operações deste tipo sejam percebidas por todos os participantes de forma totalmente ordenada.

O mecanismo de controle de concorrência adotado no editor distribuído implementado baseia-se em uma proposta apresentada em [Birman89] e [Birman90] para sincronização do acesso sobre réplicas de dados. Este mecanismo associa aos objetos compartilhados um bastão que é transferido entre os processos que desejam atualizá-los. Um processo apenas poderá modificar algum objeto quando estiver de posse do bastão, em caso contrário deverá requisitar inicialmente o bastão enviando uma mensagem de difusão ao grupo. As mensagens de requisição e passagem

de bastão, assim como de atualizações das réplicas, são enviadas através de difusões com entrega causalmente ordenada. Todos os processos que recebem a requisição de bastão a adicionam em uma fila de requisições[9], segundo sua ordem de chegada. Quando o processo que detém o bastão não tem mais interesse em retê-lo, verifica em sua fila de requisições o próximo detentor, enviando-lhe em seguida o bastão. Este processo se repete enquanto houver mais de um processo interessado em atualizações sobre os objetos compartilhados e garante um comportamento de 1-cópia com relação as réplicas guardadas pelo bastão.

Entretanto, como as requisições de bastão podem ser concorrentes, a ordenação causal na entrega destas mensagens não seria suficiente para garantir que as filas de requisição estejam globalmente ordenadas. Isto poderia levar a uma inconsistência com relação a escolha do próximo detentor do bastão. Apesar disto, 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].

Algoritmo de Controle de Concorrência Implementado

O algoritmo[10] de controle de concorrência baseado na passagem de bastão é apresentado nas figuras 5.8 e 5.9.

Os procedimentos de requisição e de passagem do bastão (figura 5.8) são implementados através da primitiva do Serviço de Comunicação de Grupo para a difusão das mensagens (procedimento `BCast`). Os procedimentos de recepção de mensagens de requisição e de passagem do bastão (figura 5.9), são implementadas como funções de retorno de chamada registradas pela aplicação e invocadas pelo ISIS quando as respectivas mensagens são entregues.

Adicionalmente, através de primitivas de observação de processos, verificação da pertinência de um processo com relação a um grupo e de obtenção do posto de algum processo em um grupo oferecidas pelo sistema ISIS, que são representadas no algoritmo pelas funções `Pertence`, `Observa`, e `MaisAntigo`, respectivamente (figura 5.9), pôde-se implementar o mecanismo de controle de concorrência tolerante a falhas em dois aspectos importantes:

---

[9] Poderia-se também pensar em enviar as requisições de bastão apenas para o atual detentor deste. Entretanto, com cópias replicadas da fila de requisição pode-se garantir, por exemplo, que o procedimento de passagem de bastão continue mesmo em caso de falha de seu atual detentor.

[10] No Anexo D encontram-se definições dos tipos de dados, variáveis, e procedimentos utilizados no algoritmo apresentado nas figuras 5.8 e 5.9.

- falha de um processo que requisitou o bastão e ainda não o recebeu: neste caso a função `Pertence` retornaria falso, evitando que o bastão fosse transferido para um processo que não mais participa do grupo;

- falha do atual retentor do bastão: a função `Observa` permite que os processos que recebem a mensagem de passagem do bastão ativem o mecanismo de observação de processos do ISIS, que invocará o procedimento `NotificaFalha` (em todos os processos do grupo) todas as vezes que o atual processo detentor do bastão falhar. A função `MaisAntigo` permite que apenas o processo mais antigo reaja a falha ocorrida, assumindo então o bastão e enviando-o para um eventual requisitor em sua fila de requisições.

```
RequisitaBastão()
{
MSG_REQUISIÇÃO msg_req;
        SE (esperando_bastão==FALSE) {
                SE (retém_bastão)
                        RETORNA;
                BCast (grupo, msg_req, ASYNC);
        esperando_bastão=TRUE;
        }
}
```

(a)

```
EnviaBastão()
{
BOOL            bastão_transferido = FALSE;
MSG_REQUISIÇÃO msg_req;
MSG_PASSAGEM    msg_passagem;
        ENQUANTO( (FilaVazia(Fila_de_requisições, requisitor)==FALSE)
                  E(bastão_transferido==FALSE))  {
                msg_req = RemovePrimeiro(Fila_de_requisições);
                msg_passagem.novo_detentor = Emissor(msg_req);
                BCast( grupo, msg_passagem, SYNC,
                        bastão_transferido);
        }
        SE (bastão_transferido)
                retém_bastão = FALSE;
}
```

(b)

Figura 5.8 - Procedimentos de requisição (a) e de passagem (b) do bastão.

```
RecebeRequisiçãoBastão(MSG_REQUISIÇÃO msg_requisição)
{
        Insere(Fila_de_requisições, msg_requisição)
        SE (retém_bastão E usando_bastão==FALSE)
                EnviaBastão();
}
```

(a)

```
RecebePassagemBastão(MSG_PASSAGEM msg_passagem)
{
ENDEREÇO        novo_detentor;
BOOL            bastão_transferido = FALSE;
        novo_detentor = msg_passagem.novo_detentor;
        SE (Pertence(grupo, novo_detentor) {
                detentor = novo_detentor;
                SE (meu_endereço == detentor) {
                        detém_bastão = TRUE;
                        esperando_bastão = FALSE;
                }
                SENÃO {
                        Observa(detentor, NotificaFalha);
                }
                bastão_transferido = TRUE;
        }
        SE (Emissor(msg_passagem) == meu_endereço) {
                Reply(bastão_transferido);
        SENÃO
                RemoveEndereço(Fila_de_requisições, novo_detentor);
}
```

(b)

```
NotificaFalha()
{
        SE (MaisAntigo(grupo, meu_endereço)) {
                retém_bastão = TRUE;
                EnviaBastão();
        }
}
```

(c)

Figura 5.9 - Procedimentos de recepção da requisição (a) e da passagem (b) do bastão e de notificação de falha do atual detentor (c).

O algoritmo de requisição e passagem do bastão é oferecido para a aplicação através da classe Bastão. A cada figura geométrica desenhada durante uma sessão de edição, é associado um objeto da classe Bastão (ver figura 5.6), que vai oferecer o mecanismo de controle de concorrência sobre o objeto correspondente a esta figura. Desta forma obtêm-se uma granularidade de controle de concorrência a nível de figuras.

Considerações com Respeito ao Controle de Concorrência Proposto

Uma outra alternativa para o controle de concorrência baseado nos serviços de suporte implementados, é a obtenção de exclusão mútua através do Serviço de Voto, como ocorre no editor de desenhos CoDraft [Kirsche93a]. Neste caso, sempre que algum usuário pretender alterar um objeto, e não estiver com o direito de acessá-lo, é iniciado um processo de votação para verificar se alguém se opõem a que este usuário receba este direito. O usuário só poderá prosseguir com as alterações se o processo de votação resultar em consenso.

Entretanto, o mecanismo de passagem de bastão mostra-se mais adequado para o controle de concorrência por duas razões:

- 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.

5.4.3 Janela de Edição Comum

Na implementação da Janela de Edição Comum, um usuário pode atuar sobre o espaço de objetos compartilhados através de operações de desenho (criação), movimentação e deleção de objetos. Estas operações utilizam diretamente os métodos da classe Lista de Objetos, e são ativadas por menus e botões disponíveis a nível de interface com o usuário e através da manipulação do *mouse* sobre uma área de desenho (*canvas*).

Desenho de Objetos

O desenho de um objeto na Janela de Edição Comum ocorre através das seguinte etapas. Primeiramente, utiliza-se uma versão local provisória deste objeto para o armazenamento de seus atributos, enquanto o usuário não termina de defini-los completamente[11]. Em seguida o objeto é

[11] A escolha do objeto a ser desenhado é feito através de um botão disponível na interface com o usuário, e a apresentação dos estados intermediários do objeto na tela ocorre simultaneamente às ações do usuário.

criado, ou seja, a sua versão provisória é incluída na Lista de Objetos e difundida, através do Serviço de Comunicação de Grupo, para as demais instâncias do editor em execução no mesmo grupo. Nestas instâncias, ao ser recebido o objeto que fora criado, este é também incluído nas respectivas cópias locais da Lista de Objetos.

Movimentação de Objetos

Na operação de movimento o usuário intercepta um objeto, através do *mouse*, alterando localmente suas coordenadas relativas no plano definido pela tela de pintura[12]. Entretanto, a operação de movimento de um objeto deve considerar o aspecto de autonomia dos participantes com relação a modificação de atributos dos objetos, já que este encontra-se replicado como um item da Lista de Objetos. Poderiam existir, por exemplo, dois ou mais usuários desejando movimentar simultaneamente o mesmo objeto em direções opostas. Neste caso é preciso, antes de iniciar o movimento, obter a exclusão mútua de acesso sobre a réplica. Para isto verifica-se primeiramente se o bastão associado ao objeto está em poder da cópia local. Em caso afirmativo, o movimento do objeto é iniciado, e a cada modificação de posição esta é difundida para o grupo por meio do Serviço de Comunicação de Grupo. Em caso negativo, o usuário é informado de que o objeto encontra-se bloqueado por outro processo, a quem também é oferecida a opção de requisitar o bastão, através do método de requisição (da classe Bastão) associado à cópia local do objeto. A cópia local é então liberada para atualização, apenas após ter sido recebido o bastão requisitado.

Deleção de Objetos

A deleção de objetos pode ocorrer de duas formas: deleção individual ou de toda a Lista de Objetos (operação *delete all*). A deleção de um objeto individual ocorre de forma similar à operação de movimento de um objeto: primeiramente a aplicação assegura que a cópia local contém o bastão associado; só então deleta objeto localmente e em seguida difunde ao grupo (pelo Serviço de Comunicação de Grupo) a operação de deleção do objeto. Os receptores desta operação deletam imediatamente suas cópias.

Devido aos efeitos desastrosos que podem decorrer da operação de deleção de toda a Lista de Objetos, é necessário que se estabeleça anteriormente um consenso entre os participantes do grupo com relação a sua execução. Para obter este consenso utilizou-se o Serviço de Voto provido pelo suporte. Na fase de distribuição da questão, é enviada ao grupo a operação de deleção da Lista de Objetos. Em seguida, cada um dos membros responde, concordando ou não

[12] Onde o par (x=0; y=0) corresponde ao canto superior esquerdo da tela de pintura.

com esta operação. Somente no caso de todos os participantes responderem positivamente à questão é que a cópia local da Lista de Objetos é destruída. Em uma terceira fase, o iniciador da operação envia o resultado da votação para que as demais cópias da Lista de Objetos sejam também destruídas (se houve consenso).

5.4.4 *Telepointer*

O *telepointer* (ou cursor de mouse distribuído) permite a replicação dos movimentos do ponteiro de mouse do sistema, possibilitando a algum usuário do editor distribuído apontar para cópias remotas de objetos que estão sendo apresentados na Janela de Edição Comum.

O ambiente gráfico de janelas sobre o qual foi implementado o editor distribuído (Sistema X Window [Nye] [Nye92]) não suporta, além do ponteiro de *mouse* do sistema, a criação de nenhum outro ponteiro [Nakajima93]. Por este motivo foi necessária a implementação do *telepointer* a nível da aplicação.

As funcionalidades do *telepointer* foram encapsuladas na classe *Telepointer*. Nesta classe, o *telepointer* foi implementado como um recurso único do sistema, e que é compartilhado por todas instâncias do editor. Apenas a instância que possui o *telepointer* tem os movimentos do seu ponteiro de sistema replicados. O *telepointer* é gerenciado de forma similar ao mecanismo de passagem de bastão citado anteriormente (seção 5.4.2). Qualquer usuário que desejar obtê-lo envia uma requisição ao grupo. Esta requisição é armazenada em uma fila de requisição em todos os participantes do grupo. Quando o atual detentor quiser liberá-lo, ele o envia ao mais antigo requisitor em sua fila de requisições. Inconsistências nesta fila são aqui também resolvidas através da mensagem de passagem do *telepointer*.

As mensagens de requisição e passagem do *telepointer* basearam-se no Serviço de Comunicação de Grupo. A recepção de mensagens de requisição e passagem do bastão são implementadas também através de funções de retorno de chamadas registradas no ISIS. Adicionalmente, o mecanismo de gerenciamento do *telepointer* foi implementado de forma tolerante a falhas do processo que requisitou o bastão (e ainda não o recebeu) e do atual retentor do bastão.

A visualização do ponteiro é feita através da alteração da forma do ponteiro do sistema (que na versão atual do editor assume a forma de um pequeno helicóptero sobre a área de desenho), e que passa então a ser comandado pelos movimentos originados pelo processo que detém o *telepointer*. O usuário têm a opção de ativar ou desativar a apresentação do *telepointer* localmente na sua Janela de Edição Comum (sem interferir na eventual apresentação em outros

sítios). Assim o usuário pode facilmente escolher entre visualizar o ponteiro local clássico do sistema ou o *telepointer*.

A classe *Telepointer*, na forma como foi implementada, atende aos requisitos mencionados em [Nakajima93][13], no sentido de que um ponteiro distribuído criado a partir desta classe:

- permite o redesenho das áreas por onde o ponteiro passa (isto é obtido a partir do uso do ponteiro do sistema como *telepointer*);
- tem movimento rápido (o que é obtido com a comunicação assíncrona oferecida pelo Serviço de Comunicação de Grupo);
- é independente de outras aplicações (no sentido de só replicar os movimentos gerados dentro da Janela de Edição Comum da aplicação que possui o recurso do *telepointer* e também por voltar à forma convencional sempre que o usuário remoto retira o ponteiro da Janela de Edição Comum);
- é distinto do ponteiro ordinário do sistema (pois assume uma forma diferente do ponteiro do sistema e do ponteiro definido para a Janela de edição Comum);
- não dispersa a atenção dos usuários (pois, além de haver sempre no máximo apenas um ponteiro visível (o ponteiro do sistema ou o *telepointer*), qualquer usuário tem a opção de desativar, localmente e a qualquer momento, a apresentação do *telepointer*);
- sua posição na tela é independente das coordenadas absolutas de sua posição na aplicação que o detém, o que garante que o *telepointer* sempre apareça dentro das Janelas de Edição Comum, independente da posição das janelas da aplicação com relação a tela do computador nos diversos sítios (decorrente também do fato do *telepointer* apenas replicar os movimentos em relação à Janela de Edição Comum).

5.4.5 Gerenciamento de Grupo

Para o gerenciamento do grupo de instâncias do editor, utilizou-se diretamente as funcionalidades oferecidas pelo Serviço de Gerenciamento de Grupo do suporte implementado. Desta forma, as funcionalidades deste módulo são implementadas através da ativação de métodos definidos pela classe Serviço de Gerenciamento de Grupo e do mecanismo de notificação de eventos do ISIS utilizado neste serviço.

[13] O requisito que obriga o ponteiro a não ser escondido por outras janelas, também citado em [Nakajima93], não foi considerado nesta implementação.

5.5 CONCLUSÕES

Este capítulo apresentou, inicialmente, a implementação do conjunto de serviços propostos no capítulo anterior para o suporte a aplicações cooperativas do tipo editor distribuído. Na implementação destes serviços fez-se uso intenso de funcionalidades do sistema ISIS para comunicação de grupo e gerenciamento de processos, e também no uso de suas ferramentas para implementação de mecanismos tolerantes a falhas.

Como pôde ser visto na descrição das implementações realizadas, houve um esforço inicial de encapsulamento através de classes de objetos, ao nível da plataforma de suporte, das funcionalidades do sistema ISIS. Este esforço existiu de forma completa na fase de concepção dos serviços (como mostra a figura 5.2). Na implementação, entretanto, procurou-se efetivá-lo apenas quando as funcionalidades do sistema subjacente não ofereciam a abstração necessária, compatível com as necessidades do programador da aplicação. Apesar disto, tendo em vista a independência da interface dos serviços do suporte com relação às maneiras como eles possam ser implementados, acredita-se ser válida a disponibilização no futuro de um conjunto de serviços realizados seguindo plenamente a abordagem de orientação a objetos.

Em seguida foi apresentada a implementação de um editor distribuído de figuras geométricas. Os módulos que compõem o editor foram também concebidos como classes de objetos. Na implementação, foram encapsulados em classes as funcionalidades e informações que definem a lista de figuras que compõem um documento, o mecanismo de controle de concorrência baseado na passagem de bastão, e o *telepointer*. Com estas classes de objetos pôde-se usufruir dos benefícios oferecidos pela programação orientada a objetos, principalmente em termos de modularidade da implementação e disponibilização de peças de código reutilizáveis.

A construção dos serviços de suporte propostos em conjunto com o editor distribuído, permitiu também que fosse adquirida uma experiência importante no sentido de avaliar a real contribuição dos serviços do suporte (e do sistema ISIS) na tarefa de programação de aplicações cooperativas. O Capítulo seguinte trará considerações com respeito a esta experiência.

CAPÍTULO 6

CONSIDERAÇÕES FINAIS SOBRE O SUPORTE IMPLEMENTADO

6.1 INTRODUÇÃO

Este capítulo tem como objetivo principal mostrar de que forma os serviços de suporte para aplicações cooperativas oferecem uma real contribuição para a construção destas aplicações. Para isto os serviços que foram implementados são analisados segundo aspectos operacionais (em especial os ligados a seu desempenho), funcionais e também a partir de sua utilização na construção do editor distribuído.

Desta forma, serão inicialmente apresentados alguns resultados obtidos através de medições do tempo de resposta do Serviço de Comunicação de Grupo feitas em experimentos realizados neste trabalho e também de informações existentes na literatura. Em seguida são apresentados aspectos qualitativos das funcionalidades presentes nos serviços implementados e feitas considerações a respeito destes serviços a partir da utilização prática destes na construção e utilização do editor distribuído.

6.2 ASPECTOS DE DESEMPENHO DO SERVIÇO DE COMUNICAÇÃO DE GRUPO

Tendo em vista a necessidade de suporte às interações em tempo-real das aplicações cooperativas síncronas, como é o caso do editor distribuído implementado, foram efetuadas algumas medições de desempenho do Serviço de Comunicação de Grupo, implementado a partir da primitiva CBCAST do sistema ISIS. Também foram levadas em conta medições similares existentes em [Birman91].

6.2.1 Contexto da Avaliação

O desempenho do Serviço de Comunicação de Grupo foi obtido a partir de medições do tempo médio de execução de envios de mensagens com resposta (padrão RPC - *Remote Procedure Call*).

Estas medições foram feitas variando-se:

- o comprimento das unidades de dados do usuário: 1 Kbyte e 7 Kbytes. Estes comprimentos de dados de usuário foram escolhidos para comparação com resultados obtidos em [Birman91], e representam valores superiores à quantidade típica de dados envolvidos em interações de atualização de objetos na janela de edição comum e de replicação dos movimentos do *telepointer*.

- o número de destinatários: para si mesmo, ou seja, apenas o processo emissor recebe a mensagem, e em seguida adicionando-se sucessivamente de um a cinco processos destinos em sítios distintos remotos. O número máximo de seis sítios representa um limite razoável para um grupo em edição cooperativa.

As medições foram executadas no seguinte contexto:

- Ambiente: estações de trabalho Sun-4[1], dos modelos SPARCstation[2] 10+ (uma), SPARCstation 1+ (três), SPARCstation ELC (uma), Server 630MP (uma) (inseridas nas medições nesta seqüência), utilizando o sistema operacional SunOS[3] 4.1.3 e interconectadas por uma rede local Ethernet (10 Mbps) em situação normal de carga.

- Sistema ISIS V2.1 (versão não comercial);

- Tempo médio: média do tempo total gasto em 10000 requisições sucessivas do serviço. Para ter-se a garantia da entrega das mensagens nos destinatários, as mensagens foram enviadas em seqüência, intercaladas por uma chamada à primitiva *Flush*, e com todos os destinatários ativos respondendo com uma mensagem nula.

---

1 Sun-4 *is a trademark of Sun Microsystems, Inc.*

2 SPARCstation *is a trademark of Sun Microsystems, Inc.*

3 SunOS *is a trademark of Sun Microsystems, Inc.*

6.2.2 Limites da Avaliação

Para a análise dos dados obtidos, tanto nas medições executadas no Laboratório de Controle e Microinformática (LCMI) da Universidade Federal de Santa Catarina (UFSC), quanto nas encontradas em [Birman91], deve ser considerado que a carga sobre o sistema computacional (rede, processadores e memória) influenciam diretamente na dinâmica das interações entre os processos. Desta forma os dados obtidos não podem ser considerados como absolutos.

6.2.3 Resultados da Avaliação

A tabela abaixo apresenta os tempos médios de resposta do Serviço de Comunicação de Grupo em milisegundos, obtidos neste trabalho.

	Tempos de resposta (mseg)					
Tamanho dos dados do usuário	Para si mesmo	1	2	3	4	5
1 Kbyte	6.2	20.7	25.5	26.25	27.6	29.2
7 Kbytes	9.2	35.7	45.83	55.73	64.67	92.30

Tabela 6.1 - Desempenho do Serviço de Comunicação de Grupo do suporte implementado (CBCAST no ISIS V2.1).

A tabela abaixo reproduz alguns dos resultados obtidos em [Birman91], que correspondem às avaliações com os mesmos parâmetros utilizados para obter-se a tabela 6.1. Neste levantamento foi utilizado o sistema ISIS V3.0 (disponível comercialmente), e as estações eram Sun[4] 4/60 interconectadas por uma rede local Ethernet (10 Mbps) levemente carregada e executando SunOS 4.1.1.

	Tempos de resposta (mseg)					
Tamanho dos dados do usuário	Para si mesmo	1	2	3	4	5
1 Kbyte	1.52	7.86	10.1	12.2	15.2	18.8
7 Kbytes	1.51	17.0	23.4	34.9	43.4	54.1

Tabela 6.2 - Desempenho do CBCAST no ISIS V3.0 [Birman91].

Nas figuras 6.1 e 6.2 são apresentadas comparações entre os tempos obtidos neste trabalho, utilizando o ISIS V2.1, e os obtidos em [Birman91] onde foi utilizado o ISIS V3.0, para pacotes de 1 Kbyte e 7 Kbytes de dados da aplicação, respectivamente.

---

[4] Sun *is a trademark of Sun Microsystems, Inc.*

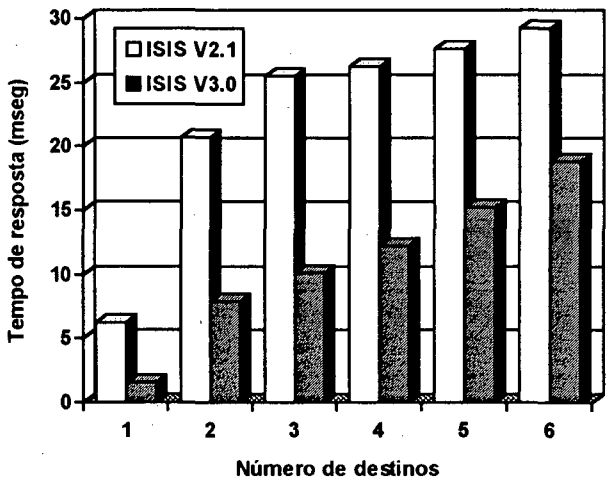


Figura 6.1 - Comparação dos desempenhos da primitiva CBCAST no ISIS V2.1 e V3.0 para 1 Kbyte de dados de usuário.

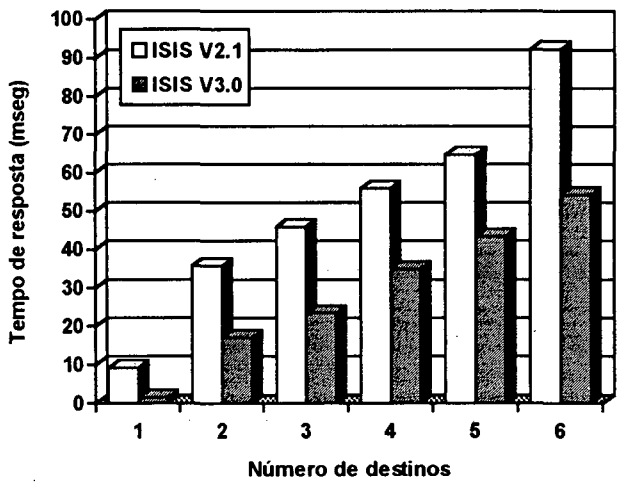


Figura 6.2 - Comparação dos desempenhos da primitiva CBCAST no ISIS V2.1 e V3.0 para 7 Kbytes de dados de usuário.

Mesmo considerando os resultados obtidos para o ISIS V2.1, pode-se afirmar que os tempos de resposta envolvidos entre um envio e o retorno de uma mensagem endereçada para um grupo pequeno, são aceitáveis do ponto de vista do desempenho, uma vez que as informações que são difundidas pelo Serviço de Comunicação de Grupo são ainda mais reduzidas (apenas os atributos que definem o objeto são transmitidos) e os grupos em editores distribuídos de desenho

compõem-se tipicamente de no máximo dez participantes [Cosquer95] envolvidos diretamente em uma sessão de desenho.

Na figura 6.3 é mostrada uma comparação entre o desempenho da primitiva CBCAST do sistema ISIS V2.1, em função dos diferentes tamanhos de pacotes de dados do usuário que foram utilizados.

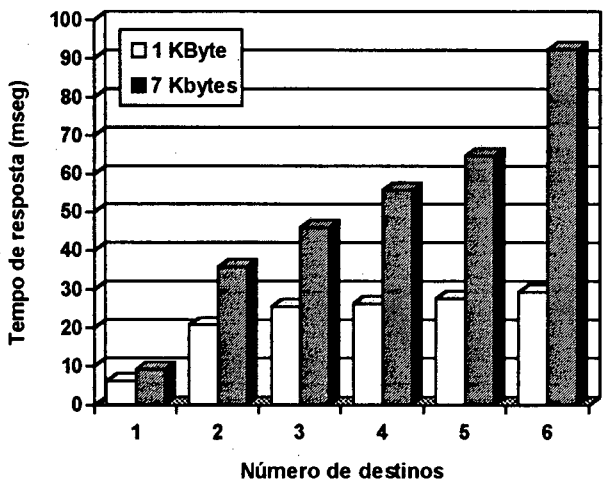


Figura 6.3 - Comparação entre os desempenhos da primitiva CBCAST para 1 Kbyte e 7 Kbytes de dados do usuário no ISIS V2.1.

Com os dados obtidos com o CBCAST do ISIS V2.1, e que são comparados na figura 6.3, pode-se verificar que o tempo de resposta desta primitiva para pacotes menores (1 Kbyte) apresenta, a partir de três até seis processos destinos, uma modificação mínima com o aumento do número de destinos (25.5 a 29.2 mseg). Por outro lado, para pacotes de 7 Kbytes e para o mesmo número de processos destino, têm-se uma perda acentuada do desempenho (45.83 a 92.30 mseg).

Como é visto nas figuras 6.2 e 6.1, o uso do ISIS V3.0 oferece um comportamento ainda melhor e com o custo em desempenho variando de forma praticamente linear com relação ao número de processos em um grupo. Comparações encontradas em [Birman91] mostram que a execução de um RPC através do CBCAST, com uma mensagem de 7 Kbytes para um destinatário remoto, atinge a mesma taxa que no caso de uma conexão sobre o TCP (*Transmission Control Protocol*) provido pelo sistema operacional subjacente: 550 Kbytes por segundo. Difusões para grupos maiores, utilizando o CBCAST atingem taxas pouco mais elevadas do que através de múltiplas

conexões TCP. Adicionalmente, as primitivas de difusão do ISIS poderiam ser mais eficientes através da utilização de mecanismos de *multicast* implementados em *hardware*[5].

6.3 ASPECTOS FUNCIONAIS DOS SERVIÇOS IMPLEMENTADOS

6.3.1 Serviço de Comunicação de Grupo

Do ponto de vista das características funcionais, o Serviço de Comunicação de Grupo, atende aos principais requisitos de comunicação de grupo delineados no Capítulo 2, quais sejam:

- transparência de nomeação;
- entrega de mensagens respeitando ordenação causal;
- entrega de mensagens de forma confiável e atômica (que inclui a noção de tolerância a falhas);
- difusões assíncronas, que permitem um maior paralelismo entre a execução das tarefas em uma aplicação e a comunicação (menores tempos de resposta);
- não exigência da manutenção de conexões a nível da aplicação.

Estas funcionalidades oferecem uma considerável redução no esforço de implementação de aplicações cooperativas.

6.3.2 Serviço de Gerenciamento de Grupo

O Serviço de Gerenciamento de Grupo implementado atende a requisitos importantes para o suporte das aplicações cooperativas em questão, já apresentados no Capítulo 2, tais como, comunicação de grupo e de tolerância a falhas.

Com relação à comunicação de grupo, as funcionalidades do ISIS utilizadas para entrada e saída dinâmica de processo de grupos, oferecem a sincronização requerida com relação à entrega de mensagens, mesmo quando estas operações são concorrentes. A facilidade de obter-se a composição atualizada e consistente do grupo, também exprime uma importante qualidade do suporte com relação ao problema de comunicação de grupo.

---

[5] A partir do ISIS V2.0 é oferecida uma facilidade ("*bypass communication*") que permite que sejam definidos pela aplicação novos protocolos de transporte, onde pode-se por exemplo fazer uso do *multicast* em *hardware* oferecido por algumas interfaces de rede Ethernet.

A capacidade de transferência de estado da aplicação cooperativa para participantes ingressantes em um grupo permite construir-se aplicações simétricas, no sentido de que informações replicadas (como um documento em edição cooperativa, p. ex.) podem ser transferidas de forma ordenada com relação às mensagens difundidas pelos participantes do grupo.

Adicionalmente, as primitivas de convite e expulsão de participantes implementadas no suporte oferecem funcionalidades necessárias para a coordenação do envolvimento dos participantes em uma sessão de trabalho cooperativo.

Apesar de não estarem encapsulados pela classe Serviço de Gerenciamento de Grupo, uma aplicação cooperativa poderá fazer uso de funcionalidades do sistema ISIS para monitoração do grupo e observação de processos. Estas funcionalidades vão fornecer uma base para implementação de mecanismos de controle de concorrência tolerantes a falha a nível de aplicação (como é o caso das classes Bastão e *Telepointer* do editor implementado). Também pode-se utilizar estas funções para notificação à aplicação cooperativa de processos que abandonam o grupo por motivo de falha.

6.3.3 Serviço de Voto

O Serviço de Voto acrescenta a capacidade de se implementar operações que requerem consenso em uma aplicação cooperativa. Como visto no Capítulo 4, a disponibilidade deste serviço contribui na construção de aplicações cooperativas [Kirsche93a], descarregando a programação da aplicação de ter que realizar o gerenciamento de votações simultâneas, refletindo automaticamente a entrada e saída de processos votantes no estado da votação e descartando silenciosamente as mensagens após a recepção de um voto de discordância (suficiente para impedir o consenso). Além disto, o Serviço de Voto beneficia-se dos mecanismos de tolerância a faltas inerentes ao uso de primitivas de difusão atômica do ISIS, o que garante, por exemplo, que o processo de resposta a uma questão não é iniciado em nenhum membro que a tenha recebido, sem que todos os outros também a tenham (mesmo no caso de falha do iniciador).

6.4 ASPECTOS DA UTILIZAÇÃO DOS SERVIÇOS NO EDITOR DISTRIBUÍDO IMPLEMENTADO

O editor distribuído implementado foi projetado com o objetivo principal de avaliar funcionalmente os serviços da plataforma de suporte proposta, e também de verificar aspectos dinâmicos destes serviços.

Assim, foi possível verificar com a utilização dos serviços do suporte proposto no editor implementado, as seguintes características globais do sistema:

- baixos tempos de resposta com relação às atividades dos usuários: a execução de sessões de desenho (realizados na forma de teste do editor com grupos de até 4 participantes), permitiu observar resultados qualitativamente satisfatórios com relação a propagação de eventos gerados pelos usuários (como o movimento de uma figura na janela de edição comum, p. ex.);

- conhecimento de todos os participantes a respeito das operações dos demais membros sobre o estado da sessão de trabalho (*collaboration awareness* [Stefik87] [Cosquer95] [Domingos95]): isto é obtido através da propagação eficiente dos eventos gerados pelos usuários e a partir da obtenção de informações de *membership* atualizadas e consistentes;

- trabalho cooperativo coordenado: obtido a partir do mecanismo de controle de concorrência, através do uso do *telepointer*, e com a utilização das primitivas de gerenciamento de grupo, em especial o convite e expulsão de participantes, e do Serviço de Voto em questões que requerem consenso.

As funcionalidades projetadas no editor visavam também oferecer uma experiência realista com respeito a complexidade da construção de aplicações distribuídas de Trabalho Cooperativo do tipo editor distribuído. Neste sentido, o envolvimento com a implementação dos módulos que requerem alto nível de interação, como a janela de edição comum e as operações a ela relacionadas, e o mecanismo de controle de concorrência, tornou ainda mais clara a necessidade de serviços que adicionem às tecnologias atuais de sistemas distribuídos, as ferramentas que propiciem a programação destas interações. A abordagem orientada a grupos de processos, em termos de comunicação e gerenciamento de grupo (com possibilidade de expulsão e convite de participantes), e os mecanismos de votação oferecidos pelo suporte implementado, contribuem neste sentido.

6.5 CONCLUSÕES

Os resultados obtidos pelas medições efetuadas com o Serviço de Comunicação de Grupo, baseado na primitiva CBCAST do ISIS, mostram que seu desempenho permite que sejam construídas aplicações compostas de grupos de até seis participantes, sem que o sistema se torne demasiadamente lento. Outras aplicações que demandem em menor intensidade o sistema de comunicação poderão ainda ser compostas de um número maior de participantes, sem comprometer os tempos de resposta. Como é de se esperar, a utilização de versões posteriores do

sistema ISIS, e também a possibilidade de se aproveitar mecanismos de *multicast* do *hardware* da rede local utilizada, pode beneficiar o suporte com um comportamento dinâmico ainda melhor.

Os resultados obtidos em relação à avaliação da dinâmica e das funcionalidades dos serviços implementados, com a construção e utilização inicial do editor distribuído implementado, foram encorajadores em ambos os aspectos. Em essência, isto significa que a união entre os serviços propostos e as facilidades oferecidas pelo ISIS, resultam em um conjunto de ferramentas que certamente vão auxiliar programadores na construção de aplicações cooperativas, em especial aquelas onde os participantes interagem de forma síncrona e através de mídias discretas (como desenhos, textos e imagens, p. ex.).

De uma forma geral, a experiência com a implementação do editor distribuído sobre a plataforma de serviços proposta, enfatiza também a importância das funcionalidades oferecidas por este suporte (ou por outros suportes similares), no sentido de prover-se ferramentas que facilitem a construção de aplicações cooperativas e que muitas vezes não são oferecidas diretamente pelas facilidades de sistemas distribuídos existentes [Rodden92].

CAPÍTULO 7

CONCLUSÕES E PERSPECTIVAS

A construção de sistemas computacionais para o apoio ao Trabalho Cooperativo, vem sendo atualmente amplamente discutida na literatura, devido à importante meta à qual estes sistemas se propõem: permitir que grupos de indivíduos possam colaborar e obter resultados melhores e de maneira mais rápida que aqueles obtidos por esforço individual, estimulando assim um ambiente de trabalho com maior sinergia. Entretanto, a tarefa de construir aplicações deste tipo demanda suportes que permitam ao programador preocupar-se essencialmente com a semântica da aplicação, e não com os mecanismos empregados para obter-se os resultados pretendidos [Trevor95].

Este trabalho apresenta um esforço no sentido de projetar e implementar uma plataforma de serviços para o suporte à construção de aplicações cooperativas. Foi dado ênfase ao suporte de aplicações onde os participantes cooperam de forma síncrona e compartilhando objetos de natureza discreta (como gráficos, imagens e textos), em especial aplicações do tipo editor distribuído. Escolheu-se uma aplicação específica (editor distribuído) pois os requisitos de suporte variam fortemente com as características funcionais das aplicações cooperativas, e a definição de um contexto de Trabalho Cooperativo permitiu uma visão mais nítida dos atributos de uma plataforma de suporte.

Editores distribuídos são construídos segundo requisitos de simetria e autonomia. Estes requisitos refletem-se em um suporte de serviços de natureza distribuída, que vão permitir principalmente a replicação das informações compartilhadas, as interações entre participantes e o controle descentralizado destas interações.

Desta forma, tendo em vista os requisitos das aplicações deste tipo, foram propostos quatro serviços para a plataforma de suporte destas aplicações: Serviço de Gerenciamento de Grupo, Serviço de Comunicação de Grupo, Serviço de Voto, e Serviço de Transferência de Arquivo Multi-destino.

Com a utilização do sistema ISIS na implementação da plataforma de serviços, obteve-se benefícios que os distinguem de outras propostas de suporte, principalmente em termos de:

- expansão automática de endereços de mensagens enviadas para grupos com *membership* dinâmico;
- entrega de mensagens de forma atômica e com ordenação causal;
- possibilidade de obtenção de informações sobre a composição do grupo *(membership)* de forma sincronizada com a entrega de mensagens a este grupo;
- ferramentas para implementar a monitoração de processos que integram um grupo;
- mecanismos de transferência de estado para processos ingressantes em um grupo, onde o estado constitui-se de informações definidas pela aplicação, como p. ex. os objetos que compõem um documento em edição cooperativa.

Estas facilidades permitirão implementar políticas mais eficientes de controle da cooperação a nível de aplicação (controle de concorrência, p. ex.). Outrossim o programador poderá colocar-se a um nível de abstração onde ele possa observar um grupo como uma entidade concisa e mantendo informações compartilhadas de forma consistente.

Com a experiência obtida através da implementação de um editor distribuído sobre os serviços propostos, e também por meio de medidas de desempenho realizadas, verificou-se, na prática, que a plataforma de serviços implementada, em especial o Serviço de Comunicação de Grupo, oferece um desempenho adequado para implementação das difusões de interações em um grupo de processos (baixos tempos de resposta), mesmo considerando o *overhead* dos protocolos que garantem a entrega causal e atômica de mensagens da primitiva CBCAST do sistema ISIS. Desta forma, este serviço foi largamente utilizado na aplicação, como por exemplo na notificação de atualização de objetos na janela de edição comum, e na difusão das coordenadas do *telepointer*.

A sincronização oferecida pelos Serviços de Comunicação de Grupo, de Gerenciamento de Grupo e de Voto, que baseia-se no modelo de sincronismo virtual do sistema ISIS, contribui para que sejam implementadas aplicações "*collaboration aware*"[1] [Stefik87] [Domingos95] (ou "*group aware*" [Trevor95]), no sentido de que sempre se poderá obter a composição atualizada e consistente do grupo, o que permite que os participantes possam estar informados de quem está atuando sobre o estado do sistema.

Adicionalmente, a utilização do sistema ISIS, tanto na implementação dos serviços do suporte proposto como na aplicação construída, demonstra que este tipo de plataforma, baseada em

[1] Em contraste às aplicações distribuídas tradicionais, onde procura-se implementar transparência sob vários aspectos (concorrência, falha e localização, p. ex.), nas aplicações cooperativas, os usuários muitas vezes precisam estar conscientes das atividades de cada qual para poderem colaborar de forma produtiva.

facilidades de programação de grupos de processos e que oferece diferentes níveis de qualidade de serviço em termos de ordenação[2] e tolerância a falhas, é de grande interesse quando se procura soluções para a camada "Sistema de Comunicação" na arquitetura de sistemas de Trabalho Cooperativo.

A partir dos resultados que foram obtidos, tanto no projeto e na implementação dos serviços de suporte, como na construção do editor distribuído, acredita-se que no futuro novos trabalhos poderão ser desenvolvidos, contribuindo em termos de complementação e de evolução do esforço atual. Em resumo, pode-se mencionar esforços em quatro níveis:

Nível Estrutural da Implementação:

- Complementar o encapsulamento dos Serviços de Gerenciamento de Grupo e de Voto, de forma a isolar completamente o código da aplicação das funcionalidades do sistemas de comunicação (neste caso o sistema ISIS), como por exemplo, através de objetos adaptadores (a exemplo do que ocorre no sistema ELECTRA [Maffeis93]).
- Expandir a abordagem orientada a objeto para o Serviço de Comunicação de Grupo.

Nível de Funcionalidades do Suporte:

- Inserir no Serviço de Gerenciamento de Grupo a capacidade de gerenciamento de grupos sobrepostos. Isto permitiria que uma aplicação cooperativa incluísse mais de um grupo, para implementar, por exemplo, níveis diferentes de privacidade sobre dados replicados [Domingos95].
- Implementar, a exemplo dos demais serviços, uma classe de objetos correspondente ao Serviço de Transferência de Arquivos Multi-destino.
- Adicionar a semântica de "k-consenso" no Serviço de Voto, através da qual uma votação é bem sucedida após a obtenção de uma percentagem "k" (entre o número total de membros votantes) de votos de concordância.
- Projetar e implementar serviços adicionais que suportem aplicações que utilizem interações explícitas baseadas em mídias contínuas, como voz e vídeo.

---

[2] Outros trabalhos em comunicação de grupo com ordenação causal e total de mensagens são apresentados em [Ravindran92] [Macedo95a-95b].

Nível de Funcionalidades do Editor Distribuído:

- Implementar no editor disponível, diferentes formas de controle de concorrência, a partir do mecanismo de bastão, e de algoritmos baseados em detecção de operações conflitantes, com o objetivo de se realizar avaliações e comparações entre diferentes algoritmos.

- Aprimorar o editor distribuído implementado, tendo em vista sua utilização em outros contextos de Trabalho Cooperativo, como: suporte a reuniões face-a-face, nas fases de *brainstorming*, organização e avaliação de idéias (de forma similar ao utilizado no Cognoter (*cognition noter*) apresentado em [Stefik87]); e em ambientes de ensino a distância (como é oferecido no projeto EDUBA [Pastor95], p. ex.), como ferramenta para esboços informais do tipo quadro negro.

- Incluir na Janela de Edição Comum a possibilidade de edição de texto e imagens (no formato GIF, p. ex.).

- Incluir canais de comunicação informal entre os participantes, como janelas para diálogos em forma de mensagens de texto ("*Talk*"), ou canais de voz e vídeo.

- Adicionar mais facilidades para a edição, como agrupamento de objetos, ordenação espacial de objetos (relação *on_top_of* de [Kirsche93a]) e atributos adicionais aos objetos (cores, fontes, etc.).

Nível de Trabalho Cooperativo:

- Construir novas aplicações cooperativas sobre o suporte implementado, com o objetivo de se estudar aspectos sociais do Trabalho Cooperativo.

REFERÊNCIAS BIBLIOGRÁFICAS

[ANSA87] Advanced Networked Systems Architecture; *ANSA Reference Manual Release 00.03 (Draft)*. ESPRIT Technical Week Edition, ANSA Project, June 1987.

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

[Birman90] Birman, K.; Cooper, R.; Joseph, T.; Marzullo, K.; Makpangon, M.; Kanc, K.; Schmuck, F.; Wood, M.. *The ISIS System Manual, Version 2.1.* Cornell University, September 17, 1990.

[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.

[Blair93] Blair, G.; Campbell, G.C.; Garcia, F.; Hutchison D.; Scott A.; Shepherd D.; *A Network Interface Unit to Support Continuos Media*. IEEE Journal on Selected Areas in Communications. Vol. 11 no. 2, February 1993.

[Coad92] Coad, P.; Yourdon, E.; *Análise Baseada em Objetos*. Tradução da segunda edição americana. Editora Campus (Série Yourdon Press), Rio de Janeiro, 1992.

[Cosquer95] Cosquer, F. J.; Veríssimo, P. J. *Survey of Selected Groupware Applications and Supporting Platforms".* Esprit Project BR 6360 (Broadcast) Technical Report 1995.

[Coulson92] Coulson, G.; Blair, G. S.; Davies, N. and Williams, N.; *Extensions to ANSA for Multimidia Computing.* Computer Networks and ISDN Systems 25 (1992)305-323, North-Holland.

[C++a] *Sun C++ Programmer's Guide*, Sun Microsystems, Inc., Revision A of 20, October 1989.

[C++b] *AT&T C++ Language System Reference Manual*, Sun Microsystems.

[Dewan93] Dewan, P.; Riedl, J.; *Toward Computer-Supported Concurrent Software Engineering*. IEEE Computer, January 1993.

[Ellis89] Ellis, C. A; Gibbs, S. J.; *Concurrency Control in Grouware Systems*. Proceedings of the ACM SIGMOD'89 Conference on the Management of Data, Seattle, Wash., May 2-4 1989, ACM, NewYork.

[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.

[Escobar91] Escobar, J.; Deutsch, D.; Partridge C.; *A Multi-service Flow Synchronization Protocol*. March 14th, 1991.

[Fritzke95a] Fritzke Jr, Udo; Farines, Jean-Marie; *Um Suporte para Aplicações de Trabalho Cooperativo do Tipo Editor Distribuído*. 13º Simpósio Brasileiro de Redes de Computadores, Universidade Federal de Minas Gerais, Depto. de Ciência da Computação, Belo Horizonte, de 22 a 26 de maio de 1995.

[Fritzke95b] Fritzke Jr., Udo; Farines, Jean-Marie. *A Support Platform for Distributed Editing*. Proceedings of the 1st. CYTED-RITOS International Workshop on Groupware, Lisbon, Portugal, September 18-20, 1995.

[Gibbs89] Gibbs, S. J.; *LIZA: An Extensible Groupware Toolkit*. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (Austin, Tex., April 30 - May 4) ACM, New York, 1989.

[Heller90] Heller, D. *XView Programming Manual - An OPENLOOK Toolkit for X11*. The Definitive Guides to the X Window System, Vol. 7, 2nd. Edition, O'Reilly & Associates, Inc., 1990, USA.

[ISISMP86] ISIS *"Man pages"*. Cornell University, Ithaca, NY. Last change: 1st. February 1986.

[Joyner90] Joyner, S.; Parker, C.; Powrie, S.; Siemieniuch, C.; *Multi-media Collaborative Working in the Automotive Industry - The Role for Broadband Communication Systems*. Proceedings of the Sixt Computer Integrated Manufacturing Europe Annual Conference. 15 - 17 May, 1990, Lisbon, Portugal (Springer-Verlag).

[Kirsche93a] Kirsche, T et al; *Communication Support for Cooperative Work.* Computer Communications, vol. 16, no. 9, september 1993.

[Kirsche93b] Kirsche, T.; Lenz, R.; Lührsen, H.; Meyer-Wegner, K., Wedekind, H.; *Codraft - Eine Verteilte Architektur zur Unterstützung von Gruppen durch Multimediale Objekte.* Verteilte Multimediale Systeme. Proc. GI/ITG Arbeitstreffen (Stuttgart, 18./19., Februar 1993), München u. a.: K. G. Saur, 1993, S. 160-173.

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

[Leung90] Leung, W.-H. F.; Baumgartner, T.J.; Hwang Y. H.; Morgan M.J.; Tu S.-C.; *A Software Architecture for Workstations Supporting Multimedia Conferencing in Packet Switching Networks.* IEEE Journal on Selected Areas in Communications. Vol. 8 no. 3, April 1990

[Little90] Little T.D.C; Ghafoor, A.; Network Considerations for Distributed Multimedia Object Composition and Communication. IEEE Network Magazine. November 1990.

[Macêdo95a] Macêdo, R. J. de A.; *Causal Order Protocols for Group Communication.* Anais do 13º Simpósio Brasileiro de Redes de Computadores, Universidade Federal de Minas Gerais, Belo Horizonte, MG, 22 a 26 de Maio de 1995.

[Macêdo95b] Macêdo, R. J. de A.; Shrivastava, S. K. *The Implementation and Performance Analysis of a Total Order Delivery Protocol for Group Communication.* Anais da PANEL'95 - XXI Conferência Latino-Americana de Informática/XXII Seminário Integrado de Software e Hardware, Canela, RS, 1995.

[Maffeis93] Maffeis, S.; *A Flexible System Design to Support Objetc-Groups and Object-Oriented Distributed Programming.* Proceedings of the ECOOP'93 Workshop on Object-Based Ditributed Programming, Lecture Notes in Computer Science 791, Springer Verlag 1994.

[Nakajima93] Nakajima, A. *Telepointing Issues in Desktop Conferencing Systems.* Computer Communications, vol. 16, no. 9, september 1993.

[Nicolaou90] Nicolaou, C. *An Architecture for Real-Time Multimedia Communication Systems*. IEEE Journal on Selected Areas in Communications. Vol. 8 no. 3, April 1990.

[Nye] Nye, A. *Xlib Programming Manual (for version 11)*.The Definitive Guides to the X Window System, Vol 1, O'Reilly & Associantes, Inc., USA.

[Nye92] Nye, A. *Xlib Reference Manual (for version 11)*. The Definitive Guides to the X Window System, Vol 2, 3rd. Edition, O'Reilly & Associantes, Inc., 1992, USA.

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

[Palmer94] Palmer, J.D.; Fields, N. A.; Brouse, P. L.; *Multigroup Decision-Support Systems in CSCW*. IEEE Computer, May 1994.

[Pastor95] Pastor, E.; Fernández, D.; Bellido, L.; *Cooperative Learning over Broadband Networks*. Proceedings JENC6.

[Pendergast93] Pendergast, M.O.; *Multicast Channels for Collaborative Applications: Design and Performance Evaluation*. ACM SIGCOMM. Computer Communication Review, Volume 23, Nr. 2 April 1993.

[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.

[Reddy93] Reddy, Y. V. R.; Srinivas, K.; Jagannathan, V.; Karinthi, R.; *Computer Support for Concurrent Engineering*. IEEE Computer, January 1993.

[Reinhard94] Reinhard, W.; Schweitzer, J.; Völksen, G.; Weber, M.; *CSCW Tools: Concepts and Architectures*. IEEE Computer, May 1994.

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

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

[Sobelman89] Sobelman, G . E.; Krekelberg, D. E.; *C Avançado - Técnicas e Aplicações.* Editora Campus Ltda., Rio de Janeiro, 1989.

[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.

[Tou94] Tou, I.; Berson, S.; Estrin, G.; Eterovic, Y.; Wu, E.; *Prototyping Synchronous Group Applications.* IEEE Computer, May 1994.

[Trevor95] Trevor, J.; Rodden, T; Blair, G. *COLA: A Lightweight Platform for CSCW.* Computer Supported Cooperative Work (CSCW), 3, 197-224, 1995.

[Tschammer90] Tschammer, V; Bradenburg, U. W.; Hall, J.; Strick, D. *Support for Cooperative Work in Distributed CIM-Structures.* Proceedings of the Sixt Computer Integrated Manufacturing Europe Annual Conference. 15 - 17 May, 1990, Lisbon, Portugal (Springer-Verlag).

[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.

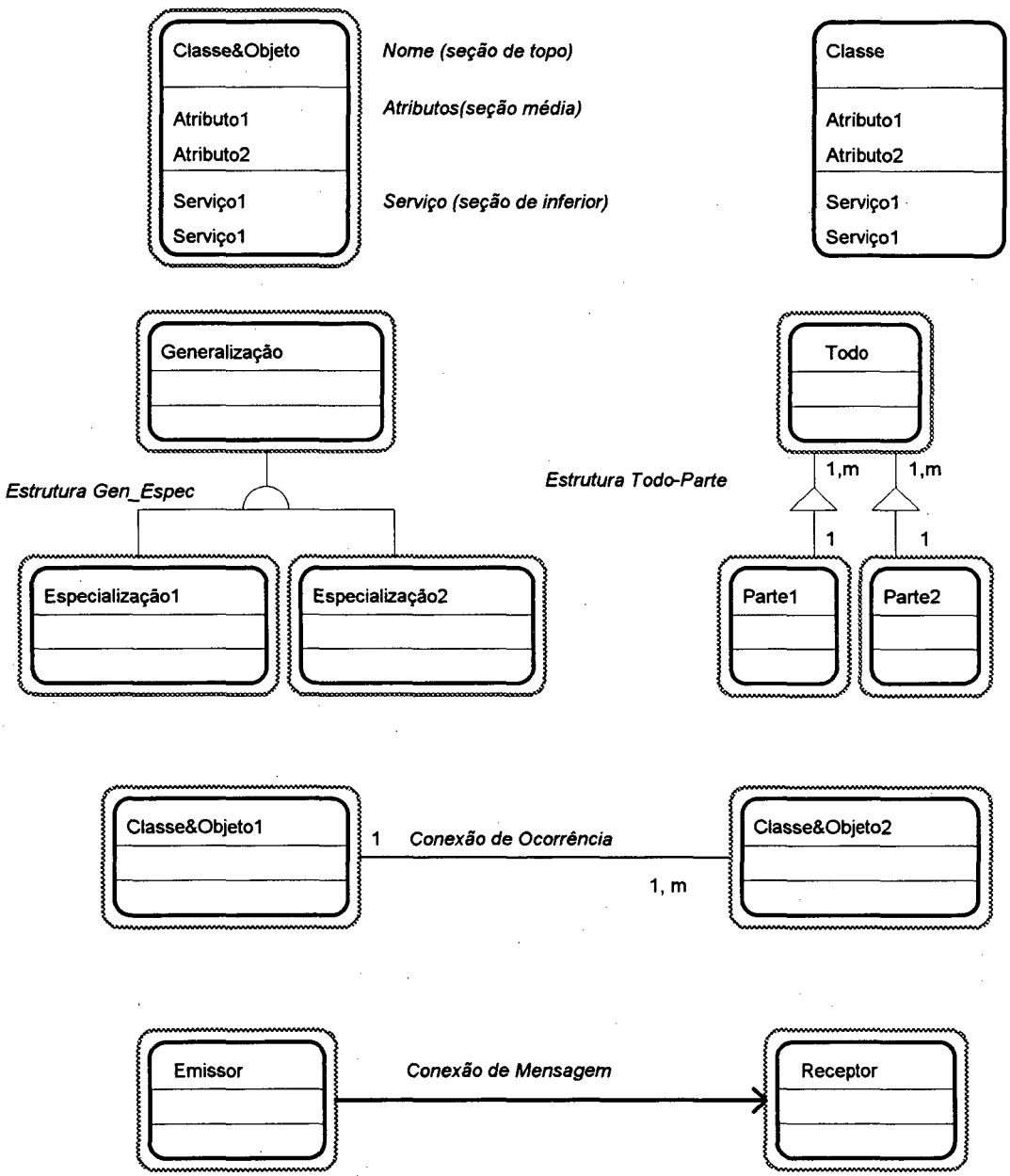
[Williams94] Williams, N.; Blair, G. S.; *Distributed Multimedia Applications: A Review.* Computer Communications, vol. 17, no. 2, February 1994.

[Yavatkar92] Yavatkar, R.; MCP: *A Protocol for Coordination and Temporal Synchronization in Multimedia Collaborative Applications.* Proceedings of the 12th International Conference on Distributed Computing Systems. Yokohana, Japan. June 9-12, 1992.

ANEXO A

NOTAÇÃO COAD-YOURDON PARA A REPRESENTAÇÃO DE CLASSES E OBJETOS

Neste anexo é apresentada a notação de Coad e Yourdon [Coad92] para representação de classes e objetos:



ANEXO B

CLASSES EM C++ DOS SERVIÇOS DE GERENCIAMENTO DE GRUPO E DE VOTO

Neste anexo são apresentadas as classes (em C++) do Serviço de Gerenciamento de Grupo (classe GwGroup) e do Serviço de Voto (classe GwVote). Algumas funções (métodos) e variáveis (atributos) membras destas classes foram omitidas por questões de clareza.

Serviço de Gerenciamento de Grupo

```
class GwGroup {
protected:
     char          gw_newmber_cred[CRED_LEN+1];     //credencial para
                                                    //autenticacao
                                                    //de novos membros
     char          gw_gname[NAME_LEN+1];   //nome do grupo
     char          gw_pname[NAME_LEN+1];   //nome do processo
     Gw_process_id *gw_pid;                //identificacao do processo
     Gw_groupview  *gw_gview;              //composicao do grupo
     char          gw_primaryname[NAME_LEN+1];// nome do grupo primario
     address       *gw_primary_p;          //endereco do "grupo basico"
     address       *gw_newgroup_p;         //endereco do grupo criado
     int           (*gw_send_state_proc)(int, address *);     //funcao envio
                                                             //de estado
     int           (*gw_receive_state_proc)(int, message *); //funcao recep
                                                             //de estado
     short         gw_unix_process_id;     //id do processo UNIX
     u_char        gw_sitenr;              //numero do sitio
     char          gw_host[HOSTNAME_LEN+1];//nome da maquina
public:
                        GwGroup(Gw * , char *pname, int svc_id); //Construtor
     virtual address *  GwJoinGroup
          (    Gw   *,
               char *    groupname,     //nome do grupo a ingressar
               Boolean   create,        //criar/entrar em um grupo
               char *    processname,   //processo ingressante
               int       (*InitFct)(address *),  //(sem uso)
               int       (*SendStateFct)(int, address *), //Fc.de envio.
                                                          //de estado
               int       (*ReceiveStateFct)(int, message  *),//Fc. de recep.
                                                             //de estado
               int       (*CompleteReceiveStateFct)(address *), //Fc. finaliz.
                                                                //da recepcao
                                                                // de estaddo
               char *    new_member_credentials,      //credencial a submeter
               int       (*AuthenticationFct)(char *) //fun.autent.de cred.
          );
     virtual   int       GwLeaveGroup(Gw *, address *gaddr_p);
     virtual   Gw_groupview * GwGetGroupMembers(Gw *);
     virtual   int       GwInvite(Gw *, char *pname_tp\o_invite);
     virtual   int       GwExpel(Gw *, char *pname_to_expel);
};
```

Serviço de Voto

```
#include "gwissue.h"

class GwVote{
private:
     void              (*user_receive_issue_func)(int, gw_issue_def, address*
                                                                          );
```

```
    void                (*user_receive_result_func)(int, gw_vote);
    int                 gw_local_issue_id;
    Gw_issue_list *     GwIssueList;
    int                 monitor_id;
    address         *   gaddr_p;
public:
    GwVote(   Gw *      Gw_p, address *gaddr_p,
              void      (*user_rcv_issue_func)(int, gw_issue_def, address *),
              void      (*user_rcv_res_func)(int, gw_vote)
          );
    virtual   int       GwSendIssue(
                            Gw          *   Gw_p,
                            int             (*eval_func)(int, int, int),
                            int             n_agrees_exp,
                            gw_issue_def    def)
                                );
    virtual   int       GwSendVote (
                            Gw          *   Gw_p,
                            gw_vote         vote,
                            int             issue_id,
                            address     *   requester
                                );
    virtual   int       GwSendResult(
                            Gw          *   Gw_p,
                            int             issue_id,
                            gw_vote         result,
                            address     *   gaddr
                                );
};
```

ANEXO C

CLASSES EM C++ UTILIZADAS NO EDITOR DISTRIBUÍDO

Neste anexo são apresentadas as classes (em C++) dos módulos componentes da aplicação implementada, ou seja:

- as classes `Location` e `Point` que são as classes básicas que definem os objetos (figuras geométricas) que compõem o documento cooperativo. As classes que definem os objetos linha, retângulo, polígono, círculo e elipse, são especializações da classe `Point`;
- a classe `List` (Lista de Objetos);
- a classe `EToken` (Bastão);
- e a classe `ETelepointer` *(Telepointer)*;

Algumas funções (métodos) e variáveis (atributos) membras destas classes foram omitidas por questões de clareza.

Figuras Geométricas

```
class Location {
     protected:
          Coordinate Coord;
     public:
          Location (int InitX, int InitY) {Coord.X=InitX; Coord.Y=InitY;}
          int GetX(){return Coord.X;}
          int GetY(){return Coord.Y;}
};

class Point: public Location {    // Deriva da classe Location
     protected:
          Boolean       Visible;
          State         DrawState;
          State         MoveState;
          ObjectType    ObjType;
          ObjectID      ObjID;
          int           InitDrag;
          Boolean       Selected;
          ObjectLimits  Limits;
          unsigned int  LineWidth;
          EToken*       Obj_token;
     public:
                              Point(ObjectID OI, int InitX, int InitY);
                              ~Point();
          ObjectType          GetObjType(){return ObjType;};
          virtual   void      Draw(State S, Xv_Window window, Event * event){};
          virtual   void      Move(State S, Point * It, Xv_Window window,
                                                        Event * event){};
          virtual   Coordinate GetCoord(int WichPoint){return Coord;};
          virtual   void      SetCoord(int WichPoint, Coordinate C)
                                                        {Coord = C;};
          virtual   int       GetWidth(){};
          virtual   int       GetHeight(){};
          virtual             void SetWidth(int W){};
          virtual   void      SetHeight(int H){};
```

```
        virtual   int       GetHorRadius(){};
        virtual   int       GetVerRadius(){};
        ObjectID            GetObjID(){return ObjID;};
        void                SetObjID(ObjectID OI){ObjID = OI;};
        virtual   int       GetRadius(){};
        virtual   int       GetSidesNr(){};
        virtual   void      SetSidesNr(int SN){};
        virtual   Coordinate *GetCoordP(){};
        virtual   void      SetRadius(int){};
        virtual   void      SetVerRadius(int){};
        virtual   void      SetHorRadius(int){};
        void                SetSelected(unsigned int width){Selected=true;
LineWidth = width;};
        void                ResetSelected(){Selected=false;};
        Boolean             GetSelected(){return Selected;};
        unsigned  int       GetLineWidth(){return LineWidth;};
        virtual   void      AdjustObjLimits(Coordinate){};
        virtual   ObjectLimits GetObjLimits(){return Limits;};
        virtual   void      SetObjLimits( int maxx, int maxy,
                                    int minx, int miny)
                                    {Limits.maxX=maxx; Limits.maxY=maxy;
                                     Limits.minX=minx; Limits.minY=miny;};
        virtual   void      Erase(Xv_Window window){};
};
```

Lista de Objetos

```
struct Node{
     Point     *    Item;
     Node      *    Next;
};

class List {
          Node *    Nodes;
     public:
                              List();
                              ~List();
                    void      Add(Point *NewItem);
                    int       Remove(Point *Item);
                    Point *   InterceptAObject(Event *event);
                    void      DrawObjects(Display *dpy, Window xwin, GC gc);
          virtual   fstream   &Save (fstream &); // gravacao em arquivo
          virtual   fstream   &Load (fstream &); // leitura de arquivo
};
```

Bastão

```
class EToken {
     protected:
          Boolean   holds_token;
          Boolean   using_token;
          Boolean   waiting_for_tok_req;
          address   holder;
          address    *    gaddr;
          FIFOqueue *    Req_queue;
     public:
          EToken();
          ~EToken();
          Request(ObjectID OID);
          Release(ObjectID OID);
};
```

Telepointer

```
class ETelePointer {
     protected:
          Server_image   Svr_image;
          Xv_cursor      Cursor;
          Canvas         TPCanvas;
          short     *    CursorBits;
          Boolean        CursorSet;
          Display   *    Display_p;
          int            HotX, HotY;
          Boolean        hidden;
          ETPtok    *    TPtoken;
     public:
                         ETelePointer(address * group_addr, Display * display,
                                      Canvas canvas);
                         ~ETelePointer();
          int            Show(Window xwindow, int destX, int destY);
          int            Hide(Boolean hidden);
          Boolean        GetHidden(){return hidden;};
          int            ResetCursorShape(Window xwindow, Xv_cursor cursor);
          int            TelepointerBCast(address * gaddr, Event *event);
};
```

ANEXO D

TIPOS, VARIÁVEIS E PROCEDIMENTOS UTILIZADOS NO ALGORITMO DE CONTROLE DE CONCORRÊNCIA

Com o objetivo de facilitar a compreensão do algoritmo de controle de concorrência baseado na passagem de bastão, apresentado no Capítulo 5, a seguir são apresentados os tipo de dados, as variáveis globais e os procedimentos que são utilizados naquele algoritmo.

`BOOL`	booleano (1 ou 0) (*inteiro*)
`FILA`	fila de mensagens do tipo `MSG_REQUISIÇÃO` (*classe FIFOQueue*)
`ENDEREÇO`	endereço de um processo ou de um grupo, que é utilizado na emissão de uma mensagem (*address*)[1]
`MSG_REQUISIÇÃO`	mensagem de requisição de bastão (*message*)
`MSG_PASSAGEM{ ENDEREÇO}`	mensagem de passagem de bastão combinada com um campo de endereço do novo detentor do bastão (*message*)
`MENSAGEM`	generalização dos tipos `MSG_REQUISIÇÃO` e `MSG_PASSAGEM` (*message*)
`VALOR`	valor que é enviado em resposta a uma mensagem (*inteiro*)
`PROC`	referência a um procedimento (*apontador para uma função*)

Quadro D.1 - Tipos de dados utilizados no algoritmo de controle de concorrência.

`BOOL usando_bastão = FALSE`	Indica se o bastão está em uso ou não pelo processo.
`BOOL retém_bastão = FALSE`	Indica se o processo está ou não com o bastão.
`BOOL esperando_bastão = FALSE`	Indica se o processo requisitou ou não o bastão e está a espera do mesmo
`FILA Fila_de_requisições`	Fila de mensagens de requisição de bastão
`ENDEREÇO detentor`	Endereço do processo que detém atualmente o bastão (inicialmente é o criador do objeto replicado)
`ENDEREÇO grupo`	Endereço do grupo que mantém o objeto replicado que está sendo guardado
`ENDEREÇO meu_endereço`	Endereço do processo

Quadro D.2 - Variáveis globais utilizadas no algoritmo de controle de concorrência.

[1] *"address"* e *"message"* são tipos definidos pelo ISIS.

`RequisitaBastão()`	Requisita o bastão, enviando para o grupo uma mensagem de requisição de bastão.
`EnviaBastão()`	Envia o bastão através de uma mensagem de passagem de bastão emitida ao grupo.
`RecebeRequisiçãoBastão (MSG_REQUISIÇÃO)`	Trata a recepção de uma mensagem de requisição de bastão.
`RecebePassagemBastão (MSG_PASSAGEM)`	Trata a recepção de uma mensagem de passagem de bastão.

Quadro D.3 - Funções que implementam o algoritmo de controle de concorrência.

As funções apresentadas abaixo, ou fazem parte da biblioteca do *toolkit* do sistema ISIS [Birman90], ou são métodos da classe FIFOQueue que implementa a fila de requisições de bastão.

`BCast(ENDERECO, MENSAGEM, MODO [,VALOR])`	Envia mensagem para processo ou grupo endereçado, de dois modos: modo síncrono (SYNC), onde o processo emissor bloqueia até que uma resposta (de algum processo do grupo) lhe seja enviada, repassando-lhe um valor; ou modo assíncrono (ASYNC), onde o emissor não é bloqueado e não recebe nenhum valor em resposta. (*cbcast()*)
`Reply(VALOR, MENSAGEM)`	Responde a uma mensagem enviando um valor ao emissor desta. (*reply()*)
`BOOL Pertence ( ENDEREÇO, ENDEREÇO)`	Retorna 1 ou 0, depedento se o processo especificado (segundo parâmetro) pertence, ou não, ao grupo especificado (primeiro parâmetro). (*pg_rank()*)
`ENDEREÇO Emissor( MENSAGEM)`	Retorna o endereço do processo emissor da mensagem especificada. (*msg_getsender()*)
`BOOL MaisAntigo( ENDEREÇO, ENDEREÇO)`	Verifica se o processo definido pelo segundo parâmetro é o processo mais antigo do grupo definido pelo primeiro parâmetro.
`Observa(ENDEREÇO, PROC)`	Observa processo definido por END e ativa o procedimento de retorno de chamada definido por PROC caso o processo falhe. (*pg_watch()*)
`Insere(FILA, MSG_REQUISIÇÃO)`	Insere na file especificada uma mensagem de requisição de bastão. (*método da classe FIFOQueue*)

`BOOL FilaVazia(FILA)`	Retorna 1 ou 0, caso a fila especificada esteja respectivamente vazia ou com algum elemento.(*método da classe FIFOQueue*)
`MSG_REQUISIÇÃO RemovePrimeiro( FILA)`	Remove da fila especificada a mensagem de requisição bastão mais antiga. (*método da classe FIFOQueue*)
`RemoveEndereço( FILA, ENDEREÇO)`	Remove da fila especificada uma mensagem cujo emissor corresponda ao especificado. (*método da classe FIFOQueue*)

Quadro D.4 - Funções auxiliares utilizadas no algoritmo de controle de concorrência.