Projeto e Implementação de uma Aplicação Groupware com Editor Cooperativo de Objetos Gráficos e *Talk* Multi-usuário em Ambientes Distribuídos Heterogêneos

Udo Fritzke Jr., Jean-Marie Farines, Fabiano Bachmann, Joni da Silva Fraga

LCMI - Laboratório de Controle e Microinformática
DAS - Departamento de Automação e Sistemas
UFSC - Universidade Federal de Santa Catarina
Caixa Postal 476 - 88.040-900 - Florianópolis - S. C.
Tel.: +55 (48) 231-9202 - Fax: +55 (48) 231-9770
e-mail: {fritzke, farines, fabiano, fraga}@lcmi.ufsc.br

Resumo

Este artigo apresenta a implementação de uma aplicação cooperativa composta de um editor gráfico e de um *"talk"* multi-usuário, a partir de um modelo conceptual proposto para representar este tipo de aplicação e de um modelo de programação que adapta o modelo anterior para ambientes distribuídos e heterogêneos. Para facilitar a estruturação desta aplicação, o modelo conceptual foi utilizado na fase de projeto. A implementação foi realizada numa plataforma CORBA a partir de um modelo de programação baseado no modelo conceptual e no qual foram introduzidos conceitos e mecanismos que facilitam a flexibilidade e aumentam a confiabilidade da aplicação.

Abstract

This paper presents an implementation of a cooperative application which consists of a graphical editing tool and a multi-user talk, by using a conceptual model to represent and structure it during the design phase. A programming model which adapts the former model for a distributed and heteregeneous environment is also used during the implementation phase. The implementation has been performed on a CORBA platform. Some mechanisms to facilitate the flexibility and increase the reliability of the application have been also introduced in this proposal.

1. Introdução

Este artigo tem como objetivo discutir a implementação de aplicações cooperativas, descritas a partir de um modelo conceptual proposto e da sua tradução num modelo de programação que leva em conta o ambiente distribuído e heterogêneo e características de reutilização, flexibilidade e confiabilidade. Os modelos anteriores são ambos baseados em objetos e visam estabelecer um quadro de trabalho e uma linguagem de comunicação únicos para as diversas aplicações cooperativas

A aplicação *groupware* escolhida para avaliar os modelos e implementa-los é composta por duas atividades: uma de edição cooperativa de objetos gráficos e outra de troca de mensagens de texto

entre vários usuários. Esta implementação foi realizada no âmbito do projeto Protem-CC intitulado ASAP (Ambiente de Suporte à APlicações distribuídas baseada em objetos)[1].
Na fase de projeto da aplicação cooperativa, foi utilizado o modelo conceptual proposto que permite a estruturação desta a partir de abstrações representando os participantes, os objetos compartilhados e a coordenação de cada uma das suas atividades. O modelo de programação utilizado para a implementação é resultado do mapeamento do modelo conceptual para ambientes distribuídos heterogêneos e inclui mecanismos de reflexão computacional para introduzir maior flexibilidade no tratamento das questões de coordenação e de restrições temporais. A replicação de alguns serviços é usada neste modelo para aumentar a confiabilidade e a eficiência da aplicação. No contexto deste trabalho, foi utilizado um suporte para objetos distribuídos baseado na especificação CORBA da OMG.
A seção 2 descreve o modelo conceptual utilizado no projeto das atividades de edição cooperativa de objetos gráficos (ferramenta OCE) e de *"talk"* multi-usuário (ferramenta MUT). A seção 3 apresenta o projeto da aplicação cooperativa composta por estas duas atividades. Na seção 4, são descritos o modelo de programação e a arquitetura de implementação desta aplicação, discutindo finalmente alguns dos mecanismos utilizados tais como reflexão computacional no mecanismo de coordenação das atividades e replicação em alguns serviços.

2. Um Modelo Conceptual para a Estruturação de Aplicações *Groupware*

No modelo conceptual proposto em [FF97a, FF97b] e apresentado a seguir (segundo a técnica de modelagem OMT) uma aplicação de computação cooperativa (classe **Groupware Application**) é composta por uma ou mais atividades cooperativas independentes entre si (classe **Activity**) conforme apresentado na figura 2.1. Por sua vez, uma atividade tem um ciclo de vida e pode ser subdividida em estágios (classe **Stage**), conforme definido na plataforma COLA [TRB95].

Fig 2.1: Aplicação cooperativa

A estruturação em termos de atividades baseia-se em critérios como: tipos de interações, ferramentas, objetivo da atividade, conjunto de objetos compartilhados, etc. e facilita a definição e implementação de estratégias diferentes de coordenação para cada uma destas. A subdivisão de uma atividade em estágios possibilita representar diferentes aspectos na dinâmica da mesma, como por exemplo, numa atividade de tele-conferência pode-se ter um estágio inicial de apresentação do palestrante seguido de um estágio de debate entre os demais participantes.
Esta estrutura permite que um usuário que participa em mais de uma atividade possa assumir diferentes papeis em cada uma destas. Desta forma ainda, um participante afastado de um grupo associado a uma atividade por decisões da política de coordenação por exemplo, pode ainda permanecer em outras atividades.

2.1 Componentes e relações em cada atividade cooperativa

Segundo o modelo conceptual proposto já descrito em [FF97a], os principais elementos que uma atividade cooperativa pretende representar são: o elemento humano (denominado colaborador), a informação compartilhada (denominada objeto compartilhado), a cooperação entre estes elementos e a coordenação desta cooperação. Cada estágio de uma atividade (classe **Stage**) é

[1] www.lcmi.ufsc.br/~workasap

definido como uma agregação das classes **Collaborator Group** e **SetOfObjects**. A classe **Collaborator Group** agrega o grupo de colaboradores que atuam em cada estágio da atividade, sendo que cada colaborador será instanciado a partir da classe **Collaborator**. A classe **SetOfObjects**, por sua vez, permite agregar o conjunto de objetos compartilhados da atividade (**Shared Object**) que estarão envolvidos no estágio.

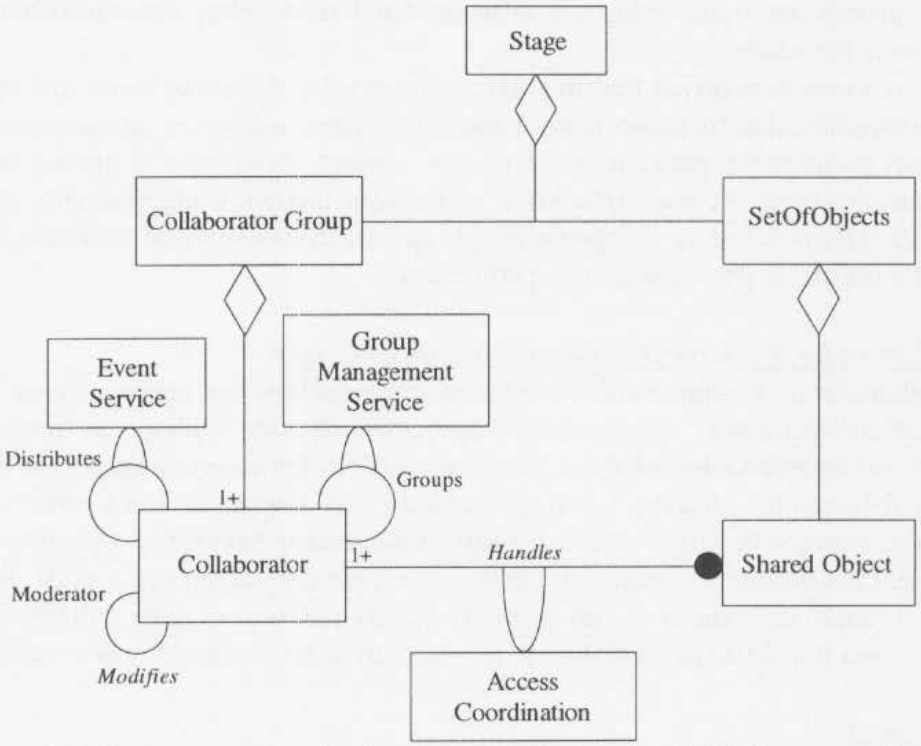

Fig 2.2: Componentes e relações num estágio de uma atividade cooperativa

As relações entre os colaboradores, e entre estes e os objetos compartilhados são modeladas por associações entre classes de objetos. As possibilidades de interações entre os colaboradores são representadas na figura 2.2, pelas seguintes associações:

- *Groups* que permite requisições de entrada e saída, convites e expulsões, e obtenção da composição do grupo de colaboradores. Esta associação tem como atributo, um serviço de gerenciamento de grupo (classe **Group Management Service**).
- *Distributes* que modela a distribuição de eventos de interesse entre os colaboradores. Um serviço de eventos (classe **Event Service**), atua como atributo desta associação.
- *Modifies* que representa a possibilidade de um colaborador, que detém o papel de moderador, poder modificar o papel assumido por outro colaborador .

A associação *Handles* representa a relação de manuseio dos objetos compartilhados (zero ou mais objetos) pelos colaboradores (pelo menos um). A classe **Access Coordination**, colocada como atributo da associação *Handles*, define as políticas que permitem coordenar a manipulação dos objetos compartilhados.

Na seqüência são apresentadas em maiores detalhes as diferentes classes que compõem uma atividade cooperativa.

Colaboradores

Os colaboradores, podem assumir quatro papéis (atributo *Role*): moderador, membro ativo, membro observador ou membro inativo.

O moderador tem o direito de atuar sobre a mudança de estágio de uma atividade, sobre a composição do grupo de colaboradores e sobre as políticas de controle de concorrência e de acesso sobre os objetos compartilhados. O moderador pode ainda alterar o atributo *Role* da classe **Collaborator** (através da operação *SetRole*) quando deseja-se mudar as condições de acesso de um colaborador sobre os objetos compartilhados, ou quando da transferência do papel de moderador. O papel de moderador é atribuído automaticamente ao iniciador de uma atividade cooperativa e poderá ser transferido, sob juízo do atual moderador, dinamicamente entre os participantes desta atividade.
O moderador e os membros ativos podem atuar sem restrições de acesso sobre um subconjunto dos objetos compartilhados (definido pelo atributo *SetOfDesiredObjects* que contém todos os objetos que um participante pretende utilizar). Ao membro observador é apenas atribuído o direito de leitura dos objetos compartilhados, e ao membro inativo, nenhum direito. A definição do atributo *SetOfDesiredObjects* é importante para que os mecanismos de coordenação possam definir a melhor utilização dos recursos compartilhados.

Grupo de Colaboradores e Serviço de Gerenciamento de Grupo
A classe **Collaborator Group** contêm atributos representando os colaboradores potenciais (*SetOfPotencialCollaborators)* que poderão participar de um estágio de uma atividade cooperativa e os colaboradores efetivos (*SetOfActualCollaborators)* atuando no estágio. O conjunto de colaboradores efetivos é definido a partir das interações entre estes, através do serviço de gerenciamento de grupo (classe **Group Management Service**) e a partir do universo de colaboradores potenciais. As operações desta classe permitem a entrada e saída dinâmica de membros e a obtenção da composição do grupo. O acesso à estas operações é diferenciado entre os participantes, em função do papel assumido por cada um deles em determinado momento.

Serviço de Eventos
O Serviço de Eventos permite que os colaboradores efetivos enviem e recebam do grupo, através da operação *NotifyEvent*, mensagens descrevendo eventos de interesse, como por exemplo, indicações de "novo estágio", de "participante saindo do grupo", etc. Estas mensagens, além de conter a descrição do evento, indicam o colaborador que o disparou. Para isto, no momento da criação da atividade, são registrados todos os eventos que poderão ser difundidos durante seus estágios (operação *RegisterEvents* da classe **EventService**). Cada colaborador pode ainda, durante um estágio, registrar seu interesse em determinados eventos, através da operação *RegisterInterests*.

Objetos Compartilhados e Conjunto de Objetos
As interações entre participantes de uma atividade cooperativa ocorrem através dos recursos que eles compartilham, sejam simples arquivos, bases de dados, recursos de comunicação, implementações de objetos em ambientes distribuídos ou até mesmo conjunto de canais para transferência de fluxos de mídias contínuas (áudio e vídeo, p. ex.) em tempo-real (abstração *stream* [CBDW92]). Cada um destes recursos é representado no modelo proposto como um objeto compartilhado da classe **Shared Object**. As formas e as características de implementações destes objetos compartilhados dependem das funcionalidades da aplicação e do tipo de interações utilizadas pelos participantes. A classe **SetOfObjects**, permite definir o conjunto de objetos que serão compartilhados pelos colaboradores em um determinado estágio de uma atividade.

Coordenação de Acesso
A coordenação das operações de acesso dos colaboradores sobre os objetos compartilhados é representada no modelo conceptual pela classe **Access Coordination**. O controle de acesso e o controle de concorrência sobre objetos compartilhados são os dois aspectos levados em conta nesta classe. O controle de acesso baseado na definição do papel de cada colaborador em cada

estágio de uma atividade, permite garantir diferentes níveis de privacidade (segurança) para os objetos compartilhados. O controle de concorrência permite a manutenção da consistência dos objetos compartilhados, a despeito de acessos concorrentes dos participantes ativos sobre estes.

3. Projeto da Aplicação Cooperativa

Nesta seção, é apresentado o projeto de uma aplicação cooperativa composta de duas atividades: um editor cooperativo de objetos gráficos OCE e um "*talk*" multi-usuário MUT, segundo o modelo conceptual proposto.

3.1 A atividade editor cooperativo de objetos gráficos OCE

A ferramenta OCE ("*graphic Object Cooperative Editor*", editor cooperativo de objetos gráficos) permite desenhar figuras geométricas simples (como linhas, polígonos, círculos e elipses) e fazer anotações de texto. As operações sobre os objetos são realizadas de forma síncrona, de tal forma que todos os usuários possam perceber imediatamente as operações de edição executadas por outros usuários.
O OCE foi projetado segundo o modelo conceptual descrito anteriormente e representado na figura 2.2, no qual: os objetos de desenho fazem parte da classe **Shared Object**; cada participante é representado por sua janela de edição que será considerada como um objeto da classe **Collaborator**; o gerenciamento de grupo dos participantes da edição é feito a partir de objetos da classe **Group Management Service**; informações sobre o estado do documento global e notificações são fornecidas aos participantes através de um serviço de eventos da classe **Event Service**; o controle de concorrência e o controle de acesso sobre os objetos de desenho são realizados pela classe **Access Coordination**.

Objetos de Desenho
Cada objeto de desenho, objeto geométrico ou texto editado de forma cooperativa, é uma instância de uma classe do tipo **Shared Object**. Estes objetos são compartilhados por todos os participantes da edição que têm acesso para leitura ou escrita aos atributos destes. Por exemplo, um retângulo criado como um objeto de desenho por algum participante poderá ter posteriormente algum de seus atributos modificado (posição na tela, p. ex.) ou ainda removido pelos participantes habilitados.
A possibilidade de escrita concorrente de valores sobre os atributos dos objetos, requer mecanismos de garantia de consistência destes objetos. A criação de objetos sobrepostos (relação *on-top-of* [KLL93]) é também possível desde que o controle de concorrência de objeto seja estendido a nível de documento. Os mecanismos de garantia de consistência dos objetos são tratados neste modelo em objetos da classe **Access Coordination** e serão discutidos com mais detalhes numa próxima seção.

Janela de Edição Comum
Cada participante é representado por sua janela de edição comum, objeto da classe **Collaborator**. Através desta janela, na qual é apresentado o documento em edição (ou seja, o conjunto de objetos de desenho), os participantes podem modificar o documento e ter uma visão WYSIWIS (*What You See Is What I See*) das alterações dos demais usuários sobre o mesmo documento.
O estado do documento apresentado na janela de edição comum de cada participante corresponde ao estado de uma cópia local do documento global. Esta cópia local é atualizada automaticamente (através do serviço de eventos), sempre que o documento global é modificado por algum participante. A janela de edição comum apresenta também menus para interface com o serviço de gerenciamento de grupo de participantes.

Serviço de gerenciamento de grupo

A classe **Group Management Service** permite a gestão do grupo de participantes efetivos da atividade OCE, definidos a partir da classe **Collaborator Group.** A entrada no grupo ocorre através de duas operações: o convite pelo moderador (*Invite*) a algum outro participante potencial (também definido a partir da classe **Collaborator Group**) e a requisição (*Request*) ativada por um participante potencial. O convite é registrado pelo sistema, garantindo a entrada do convidado caso este faça uma requisição de entrada, utilizando a segunda operação. O participante, ao se tornar efetivo, recebe o desenho no seu estado corrente e passa a participar do grupo com os privilégios definidos pelo seu papel no grupo.

A saída de participantes pode ocorrer também através de duas operações: uma requisição de expulsão (*Expel*) ativada pelo moderador e uma requisição voluntária de saída (*Leave*). Em ambos os casos, após a saída, os participantes perdem o direito de acesso aos objetos de desenho. Quando necessário, o conjunto dos membros atuais de uma sessão de edição pode ser obtido através da operação *GetGroupMembers*.

O acesso à essas operações é feito a partir da verificação das credenciais do participante. As credenciais são fornecidas ao serviço de gerenciamento de grupo quando o participante ingressa no grupo.

Serviço de eventos

O serviço de eventos (da classe **Event Service**) permite que cada participante do OCE envie e receba (através operação *NotifyEvent(...)*), de maneira não previsível, eventos indicando a alteração do estado do documento global. Para isto, cada participante ao registrar-se no serviço de eventos informa seu interesse em receber estas notificações (*RegisterInterest(...)*). Um procedimento de *call-back*, responsável pela atualização do estado da cópia local do documento global, é ativado a cada recepção de uma notificação.

Coordenação

A coordenação no OCE, através de objetos da classe **Access Coordination** tem como objetivo o controle de concorrência e o controle de acesso sobre os objetos de desenho compartilhados.

O controle de acesso dos participantes aos objetos de desenho é feito com base nos seus respectivos papéis (definidos estaticamente no conjunto de participantes potenciais mas alteráveis pelo moderador): **moderador**, com total direito de acesso sobre os objetos; **participante ativo**, com acesso aos objetos de desenho; **participante observador**, com apenas visualização dos objetos desenhados; e **participante inativo**, sem nenhum direito.

O controle de concorrência aos objetos de desenho é executado através de políticas bem definidas de bloqueio. Várias dessas políticas são mencionadas a seguir.

3.2 Atividade "*talk*" multi-usuário MUT

A ferramenta MUT (*Multi-User Talk*) tem como objetivo suportar uma conferência baseada em mensagens de texto entre um grupo de usuários distribuídos em diferentes máquinas.

A atividade MUT foi projetada segundo o mesmo modelo conceptual que a atividade OCE. As mensagens são objetos que fazem parte da classe **Shared Object**. Cada participante desta atividade tem disponível uma janela para edição e outra para a visualização das mensagens, cada uma sendo considerada como objeto da classe **Collaborator**. O gerenciamento de grupo dos participantes da conferência é feito a partir de objetos da classe **Group Management Service**. As notificações sobre o estado do documento global são fornecidas aos participantes através de um serviço de eventos da classe **Event Service**. A coordenação de acesso às janelas é realizada através da classe **Access Coordination**.

Mensagens

Cada mensagem trocada entre os usuários em uma sessão de conferência é uma instância da classe **Shared Object**. As mensagens são criadas e acessadas através de operações básicas para criação e acesso às mensagens (como p. ex. *Add*, *Remove*, *ReadAll*, *Read*, etc.) oferecidas por esta classe. O acesso ao conjunto de mensagens é controlado por objetos da classe **Access Coordination**. Duas mensagens podem estar vinculados através de uma relação do tipo *reply*; esta relação é apresentada para todos os participantes de forma consistente (i. e., um *reply* não pode aparecer em uma janela de apresentação de mensagens antes da mensagem original).

Janela de edição de mensagens

Cada participante pode editar as mensagens, a serem enviadas, numa janela de edição de mensagens, objeto da classe **Collaborator**. A operação de envio (sem ser um *reply)* pelo participante, comporta procedimentos de seleção do nome do destinatário, ativação do modo de edição e envio da mensagem a partir desta janela. Este objeto tem funcionalidades específicas: indicação dos participantes remetente e destinatário da mensagem, facilidade de anexar documentos, arrastando arquivos de texto a partir de um *file manager*.

Janela de visualização de mensagens

As mensagens submetidas e aceitas pela coordenação são visualizadas nas janelas de apresentação de mensagens (objetos da classe **Collaborator)** de cada participante do grupo. Este objeto apresenta algumas funcionalidades diferentes do anterior. Por exemplo, é a partir de uma mensagem apresentada na janela de visualização, que poderá ser enviado um *reply*. O procedimento associado corresponde a seleção desta operação na janela de apresentação de mensagens e, em seguida a edição de sua mensagem na janela de edição.

Serviço de gerenciamento de grupo e Serviço de eventos

Os serviços de gerenciamento de grupo (classe **Group Management Service)** e de eventos (classe **Event Service)** da atividade MUT são similares àqueles utilizados na atividade OCE e por esta razão não serão descritos aqui.

Coordenação

De maneira similar ao que ocorre no OCE, na atividade MUT a coordenação tem dois objetivos: controle de acesso e controle de concorrência sobre o conjunto de mensagens. O controle da palavra (*floor*), que faz parte do controle de acesso, regulamenta o fluxo de mensagens da conferência, em função do estágio corrente no qual esta se encontra. Uma conferência pode assumir os seguintes estágios:

- **Apresentação**. Neste estágio, apenas um participante tem o direito de palavra. Somente suas mensagens são aceitas para serem incluídas no conjunto de mensagens.
- **Discussão**. Neste estágio, os usuários, cada um por sua vez, poderão enviar mensagens para o usuário que detém o controle da palavra. Este último poderá responder à cada uma destas mensagens enviando um *reply* correspondente.
- ***Brain-storming***. Neste estágio, todos os usuários poderão emitir mensagens (ou *replies*) para todos os demais. Ou seja, todas as mensagens submetidas pelos participantes serão incluídas no conjunto de mensagens.

A transição entre cada estágio da conferência é definida pelo moderador, alterando dinamicamente o papel dos demais participantes conforme o caso, através do serviço de gerenciamento de grupo de participantes. Por exemplo, no estágio de apresentação, o moderador poderá fazer com que todos os demais participantes, exceto um (que assumira o papel de participante ativo), assumam o papel de observadores. No estágio de discussão, o moderador atribui o papel de participante ativo ao colaborador a quem ele deseja passar o direito de palavra. Num estágio de *brain-storming*, todos os participantes assumem o papel ativo.

4. Implementação da Aplicação Cooperativa

4.1 Modelo de programação

A implementação da aplicação *Groupware* de edição cooperativa de objetos gráficos OCE e de *"talk"* multi-usuário MUT é baseada num modelo de programação orientado-a-objetos que mapeia os componentes do modelo conceptual apresentado. Este modelo utiliza ainda o mecanismo de reflexão computacional [Mae87] para facilitar aspectos de flexibilidade relacionados à coordenação, tais como a implementação de diferentes políticas de controle de concorrência e a alteração dinâmica dos níveis de direito de acesso durante uma sessão de trabalho cooperativo.

O modelo de programação utilizado neste trabalho denomina-se modelo RTR-C, modelo reflexivo tempo-real para aplicações cooperativas. Ele é uma extensão do modelo RTR [FFFS95, FFF97] e sua descrição mais detalhada se encontra em [FF97a]. Os componentes deste modelo de programação permitem a implementação das classes de objetos e associações que definem, para o modelo conceptual, um estágio de uma atividade cooperativa.

Neste modelo de programação, os objetos da classe **Collaborator** e os da classe **Shared Object** do modelo conceptual são implementados como objetos distribuídos com uma relação de cliente-servidor entre eles (correspondente a associação *Handles* do modelo conceptual). A coordenação de acesso (objeto da classe **Access Coordination** do modelo conceptual) é implementada no modelo de programação como um meta-objeto **Coordinator MetaObject** associado ao objeto da classe **Shared Object**. Os objetos das classes **Group Management Service** e **Event Service** do modelo conceptual são também representados como objetos distribuídos, e as associações *Groups* e *Distributes* entre objetos da classe **Collaborator** são vistas como relações cliente-servidor. As informações e funcionalidades das classes **Collaborator Group** e **SetOfObjects** do modelo conceptual são incluídas no modelo de programação respectivamente nas classes **Group Management Service** e **Coordinator MetaObject**.

Seguindo a estrutura adotada no modelo RTR, o modelo de programação (RTR-C) utiliza um meta-objeto gerenciador **Manager MetaObject** que gerencia em cada sítio, os pedidos concorrentes de ativação de operações dos objetos compartilhados (da classe **Shared Object**) e, se necessário, trata eventuais restrições de tempo. Este meta-objeto **Manager MetaObject** interage com o meta-objeto **Coordinator MetaObject** que é responsável pela implementação das políticas de coordenação (controle de acesso e de concorrência) adotadas.

Ainda, como no modelo RTR, do qual é uma extensão, a questão do tempo (restrições temporais, *time-out*) é tratada a nível meta por meta-objetos específicos (**Scheduler MetaObject**, **Clock MetaObject** que interagem com o **Manager MetaObject**) que permitem manter flexibilidade deste ponto de vista. O tratamento da questão temporal não é abordada neste artigo mas é apresentado em [FFFS95, FFF97].

4.2 O uso do modelo de programação nas atividades OCE e MUT

A implementação de cada atividade OCE e MUT segue uma arquitetura cliente-servidor distribuída. O suporte à distribuição dos objetos clientes e servidores é oferecido por uma plataforma Orbix+Isis da IONA [II96], compatível com a especificação CORBA (*Common Request Broker Architecture*) da OMG (*Object Group Management*) [OMG93] que define um padrão para a implementação de ORB (Object Request Broker) em ambientes distribuídos heterogêneos. Nesta plataforma, além do ORB da IONA (Orbix), temos disponível o suporte ISIS [Bir93] para tratar o gerenciamento de grupo e a difusão de mensagens com ordenação causal. Na figura 4.1 pode ser vista a arquitetura distribuída baseada no modelo de programação RTR-C e utilizada na implementação do OCE e do MUT.

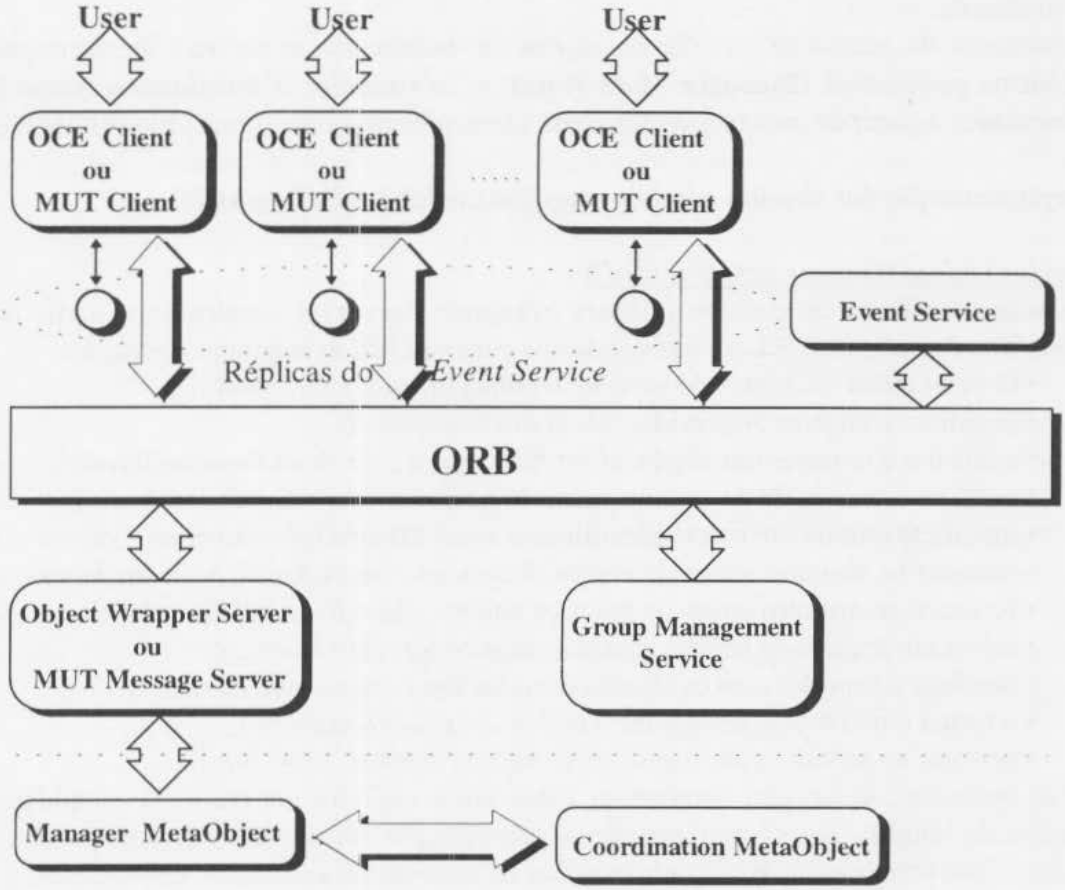

Fig. 4.1: Arquitetura da Implementação

No caso da atividade OCE, os vários participantes de um estágio desta atuam como clientes (classe **OCE Client**), através do ORB, de um servidor distribuído de objetos de desenho (**Object Wrapper Server**). Neste servidor é armazenado o desenho em edição e são disponibilizadas aos clientes operações básicas de edição (adição, remoção e alterações de atributos de um objeto de desenho, p. ex.), através de sua interface IDL-CORBA. No caso da atividade MUT, clientes MUT (classe **MUT Client**) interagem de forma similar com o servidor de mensagens (**MUT Message Server**) que armazena e exclui dinamicamente as mensagens trocadas entre os clientes utilizando uma lista de mensagens; este servidor oferece também a estas, operações básicas de escrita e leitura sobre a lista de mensagens, através de interfaces IDL-CORBA. Os objetos das classes **OCE Client** e **MUT Client** apresentam as características dos objetos da classe **Collaborator** dos modelos conceptual e de programação; os objetos servidores **Object Wrapper Server** e **MUT Message Server** são da classe **Shared Object** destes mesmos modelos.

Os clientes OCE e MUT, assim como a implementação da coordenação (correspondente a classe **Access Coordination** do modelo conceptual e implementada através do meta objeto **Coordination MetaObject**), podem acessar o serviço de Gerenciamento de Grupo (**Group Management Service**) que lhes oferece, através de sua interface IDL, as operações de gerenciamento de grupo de participantes, conforme definido no modelo conceptual.

O Serviço de Eventos (**Event Service**) foi implementado de maneira replicada utilizando o suporte para grupo de objetos oferecidos pela plataforma Orbix+Isis e associando uma réplica a cada um dos clientes OCE e MUT. Desse modo, as necessidades de acordo e ordenação causal são garantidas em cada réplica do grupo. O servidor de objetos de desenho (**Object Wrapper Server**) e o servidor de mensagens (**MUT Message Server**) atuam como cliente do Serviço de Eventos, invocando uma operação de notificação de eventos sempre que necessário. Desta forma, foi possível implementar a associação de uma *thread* a cada notificação de um evento de interesse

em cada processo OCE e MUT, permitindo a notificação assíncrona de eventos por parte dos servidores de objetos de desenho e de mensagens a todos os seus clientes OCE e MUT, respectivamente.
A coordenação do acesso ao servidor de objetos de desenho ou ao servidor de mensagens usa meta-objetos gerenciador (**Manager MetaObject**) e coordenador (**Coordination MetaObject**) implementados a partir de recursos de filtragem oferecidos pela plataforma Orbix [ION95].

4.3 **Implementação dos objetos componentes das atividades OCE e MUT**

*O Servidor **Object Wrapper Server** do OCE*

O servidor de objetos de desenho (**Object Wrapper Server**) é construído a partir de uma interface IDL do padrão CORBA, oferecendo aos clientes OCE as seguintes operações:

- destruir todos os objetos do servidor e reinicializa-lo: *short New(...);*
- registrar os clientes do servidor: *short RegisterClient(...);*
- adicionar e remover um objeto: *short AddObject(...)* e *short RemoveObject (...);*
- verificar a existência de alguma operação pendente: *short GetOperation(...);*
- modificar atributo do objeto identificado: *short ModifyObjectAttribute(...);*
- fornecer os atributos atuais do objeto identificado: *short ReadObjectAttributes(...);*
- fornecer os atributos atuais de todos os objetos: *short ReadAllObjectAttributes(...);*
- salvar em arquivo os objetos contidos no servidor: *void Save(...);*
- atualizar o servidor com os objetos contidos em arquivo: *void Read(...);*
- retornar o número de objetos do servidor: *long GetItemsNr(...);*
- permitir ao servidor conectar-se ao serviço de eventos: *short BindToES(...).*

Todas as operações do servidor apresentam, entre outros omitidos por razões de simplificação, o parâmetro de entrada *my_id* que permite a identificação no servidor do cliente que fez a invocação. Esta identificação é utilizada para fins de controle de acesso e de concorrência.

*O Servidor **Message Server** do MUT*

O servidor de mensagens da aplicação MUT foi também especificado através de uma interface IDL. O servidor oferece operações para: criar e adicionar novo objeto *Message*: *void Add(...);* ler a próxima mensagem: *short Read_Next(...);* salvar uma lista de mensagens em arquivo: *void Save(...);* ler uma lista de mensagens a partir de um arquivo *void Read(...).*
Como no caso do **Object Wrapper Server** do OCE, todas as operações do servidor de mensagens do MUT possuem um parâmetro *my_id* que identifica o participante que requisita a operação, e que é utilizado pelos mecanismos de controle de acesso.

O Cliente OCE

O cliente OCE implementa a interface gráfica (menus, botões e uma tela de desenho onde são desenhados os objetos já editados e em edição) que permite ao usuário da atividade OCE requisitar operações do serviço de gerenciamento de grupo, como por exemplo, um *JoinRequest* para entrar em um grupo, e também ter acesso ao estado do conjunto de objetos de desenho armazenados pelo servidor.
Cada cliente OCE possui uma cópia do conjunto de objetos contidos no servidor de objetos de desenho. A partir destas cópias locais é atualizada a tela de desenho. Da mesma forma, cada vez que é criado um desenho ou modificado um atributo de algum objeto já desenhado através de tela de desenho, é executada uma atualização do conjunto de objetos. Em seguida a cada atualização é invocada, seguindo o modelo de programação anterior, uma operação no servidor (neste caso *AddObject(...)* ou *ModifyObjectAttribute(...)*). Se a operação invocada não puder ser executada pelo servidor, por razões de coordenação ou falha de comunicação, é retornada uma exceção ao cliente. No caso de uma exceção, é desfeita a atualização sobre a cópia local do conjunto de objetos de desenho. A figura 4.2 apresenta uma janela do Editor OCE.

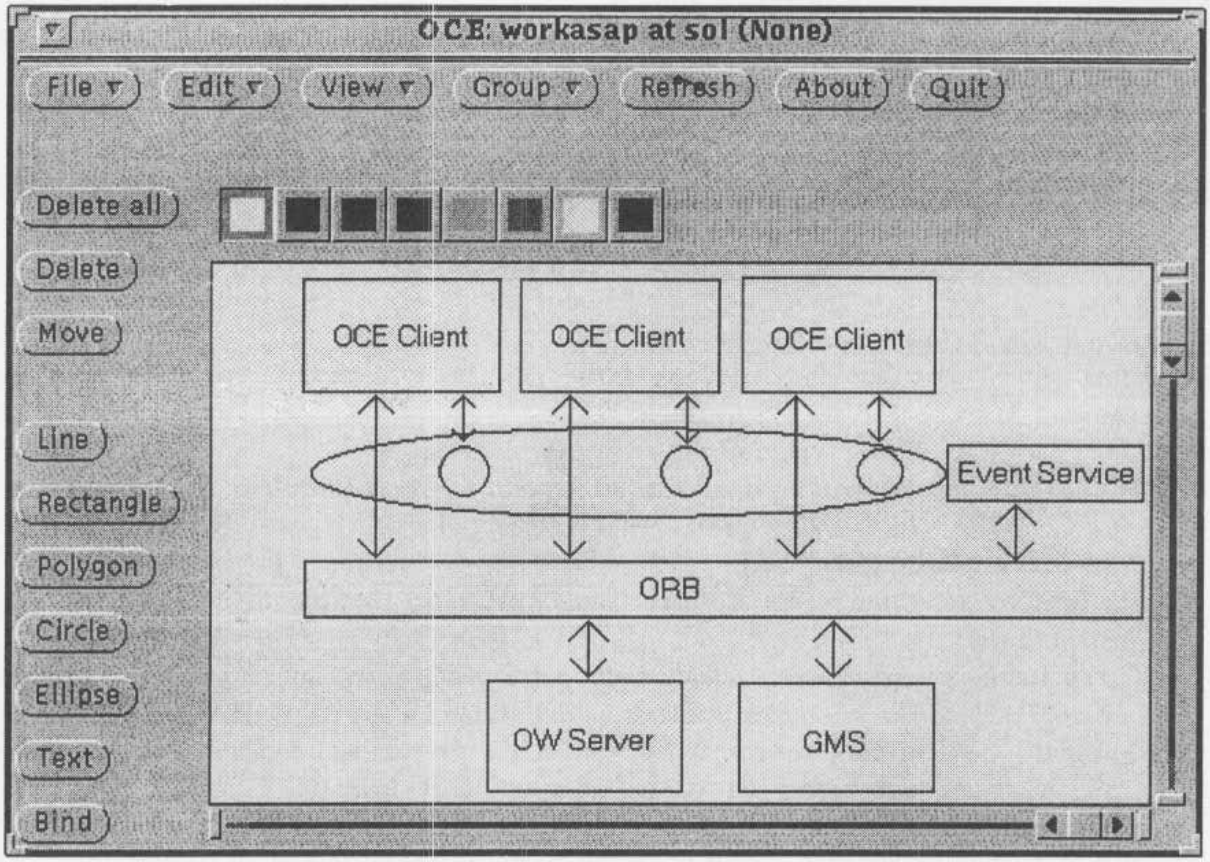

Fig 4.2: Janela da atividade OCE

O Cliente MUT

Os clientes MUT implementam interfaces gráficas que fornecem acesso às operações básicas oferecidas pelo serviço de gerenciamento de grupo e também permitem que os clientes possam visualizar a entrada e saída dinâmicas, papel, sítio e domínio dos participantes de um grupo. Além disso, os clientes MUT também são providos de áreas destinadas à edição das mensagens a serem enviadas ao grupo e à visualização das mensagens que circulam pelo grupo.

Os clientes MUT possuem uma cópia local do conjunto de mensagens difundidas e do grupo de participantes. A partir destas cópias locais é feita a atualização das telas destinadas a visualização das mensagens e do grupo de participantes.

Após a mensagem ter sido editada e enviada, é executada uma operação do servidor de mensagens que a inclui na lista de mensagens daquele servidor. Em seguida o servidor de mensagens notifica todos os clientes MUT participantes do grupo a respeito da alteração do conteúdo do conjunto de mensagens.

Na figura 4.3 são apresentadas a janela principal de diálogo do cliente MUT, a de edição das mensagens e a de visualização das mensagens difundidas no grupo.

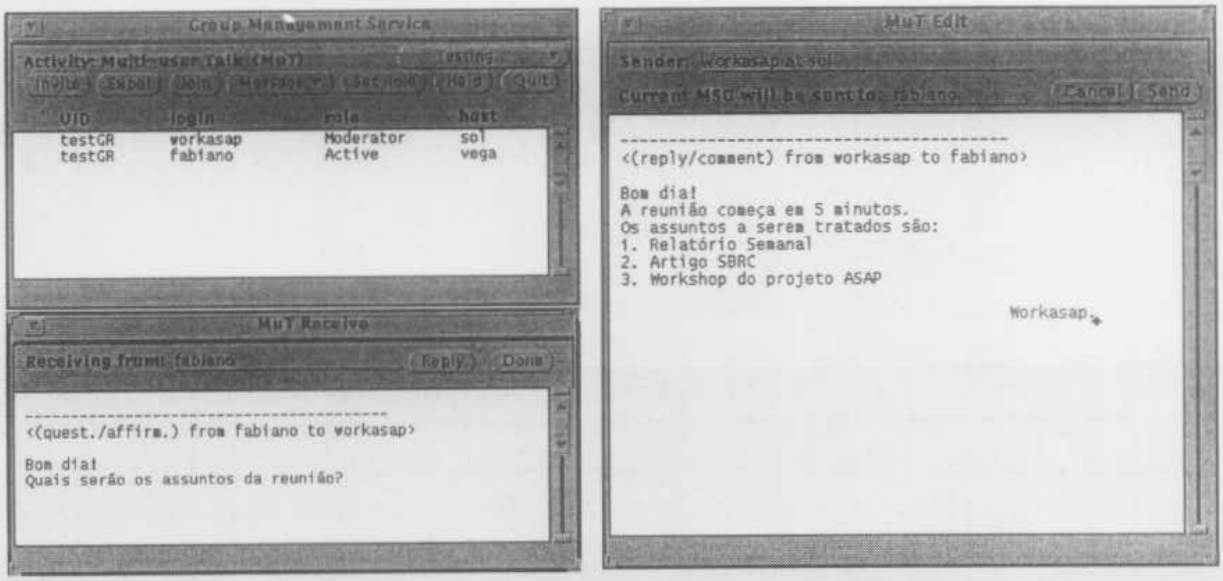

Figura 4.3: Janelas da atividade MUT

O Serviço de Gerenciamento de Grupo

De forma similar aos servidores de objetos de desenho e de mensagens, o serviço de gerenciamento de grupo (**Group Management Service**) foi construído como uma implementação de uma interface IDL . Este servidor oferece as seguintes operações:

- definir o papel do participante identificado por *c_id*: *short SetRole(in CollabID my_id, in CollabID c_id, in string role)*; apenas o colaborador com o papel de moderador pode invocar esta operação.
- devolver ao cliente o papel do participante *c_id: short GetRole(in CollabID c_id, out string role);*
- permitir ao moderador convidar um participante potencial (contido no atributo *SetOfPotencialCollaborators* da classe **Collaborator Group** do modelo conceptual) a se tornar participante efetivo (que passa a pertencer também ao atributo *SetOfActualCollaborators* da mesma classe): *short Invite(in CollabID my_id, in CollabID c_id);*
- permitir ao moderador, retirar um participante do grupo de participantes efetivos (ou seja de *SetOfActualCollaborators*): *short Expel(in CollabID my_id, in CollabID c_id);*
- permitir a algum participante potencial, solicitar a entrada no grupo de participantes efetivos: *short JoinRequest(in CollabID c_id);*
- permitir que o participante saia do grupo de participantes efetivos: *short Leave(in CollabID c_id);*
- devolver através de parâmetros, os nomes, sítios, domínios e papéis, de cada um dos participantes efetivos: *short GetGroupMembers(...).*

O controle sobre a execução das operações *SetRole*, *Invite* e *Expel*, é feito através da verificação das credenciais associadas ao parâmetro *my_id* destas operações.

O Serviço de Eventos

Ao contrário do que ocorre com os servidores anteriores e para poder oferecer um mecanismo efetivo de notificação de eventos assíncronos por parte de um objeto compartilhado (**Object Wrapper Server** no caso do OCE p.ex.) aos seus clientes (clientes OCE p.ex.), optou-se por implementar o serviço de eventos, cujas características foram estabelecidas nos modelos conceptual e de programação, como um servidor replicado e onde cada cópia localiza-se no mesmo processo dos clientes da aplicação. Para isto utilizou-se a noção de grupo de objetos

oferecida pela plataforma Orbix+Isis. A seguir é apresentada a implementação do servidor de eventos para a atividade OCE, sendo que procedimento similar é feito para a atividade MUT.
Cada cliente OCE ao ser iniciado, registra uma réplica de uma implementação do serviço de eventos. O grupo de réplicas oferece aos objetos compartilhados (por sua vez clientes deste serviço de eventos) uma operação *oneway void NotifyEvent(in short ev, in EvShortSeq data)* que lhe permite notificar um evento identificado pelo parâmetro *ev*. O parâmetro *data* permite que sejam passados dados relacionados ao evento. Desta forma, a cada operação de edição disparada por um cliente OCE e processada pelo **Object Wrapper Server**, este servidor informa aos demais clientes OCE a operação de edição executada através de uma invocação a *NotifyEvent(...)*.
Um cliente OCE, ao inicializar uma implementação do serviço de eventos, registra um conjunto de eventos dos quais quer ser notificado e, ao mesmo tempo, informa ao serviço funções de retorno de chamada (*call-back*) para o tratamento de cada um destes eventos (operações *RegisterEvents(...)* e *RegisterInterests(...)* do modelo conceptual). Com estas funções de retorno de chamada, cuja programação é de responsabilidade do programador da aplicação, o tratamento da notificação fica isolado da implementação da operação *NotifyEvent(...)*.
As invocações à operação *NotifyEvent(...)* são tratadas nos processos clientes OCE por *threads* da plataforma Isis (ver [II96]), o que garante a ordem causal das notificações nos diversos clientes OCE com relação às operações de edição recebidas e executadas pelo **Object Wrapper Server**.

5 Discussão sobre a Implementação da Aplicação Cooperativa

5.1 Reflexão computacional como mecanismo de implementação da coordenação

A utilização da reflexão computacional no modelo de programação proposto em [FF97a] tem como objetivo oferecer maior flexibilidade com relação à escolha das políticas de coordenação na implementação das atividades OCE e MUT. Desta forma, a alteração das políticas de controle de acesso e de concorrência sobre os objetos compartilhados (**Object Wrapper Server** no caso do OCE e **Message Server** no caso do MUT) pode ser feita sem a necessidade de se conhecer nem alterar a implementação destes objetos. A discussão que segue é apresentada para a atividade OCE, sendo que ela é também válida para a atividade MUT.

Implementação do meta-objeto gerenciador
O meta-objeto gerenciador (**Manager MetaObject**) é implementado na forma de pré-filtros e pós-filtros, através dos filtros *por-objeto* oferecidos pela plataforma Orbix [ION95]. Esses filtros permitem que uma invocação a um servidor Orbix seja interceptada pelo processo que implementa este servidor sem que o cliente e a implementação do próprio servidor percebam que esta intercepção ocorreu. No caso do pré-filtro, poder-se-á inibir a execução da operação invocada se o pré-filtro levantar uma exceção informando ao cliente da não execução desta operação, num procedimento transparente à implementação do servidor. No caso do pós-filtro, poderá ser executado algum processamento posterior ao da operação invocada e executada.
Para a implementação do **Manager MetaObject** do servidor **Object Wrapper Server** foram definidos um pré-filtro (classe **OW_mmo**) e um pós-filtro (classe **OW_pmmo**). O pré-filtro realiza a interação com a implementação do meta-objeto coordenador (**Coordination MetaObject**) para verificar as condições de acesso à cada operação invocada do **Object Wrapper Server**. Conforme o resultado desta interação com o **Coordination MetaObject**, pode ser, ou não, disparada uma exceção no **Manager MetaObject**. Neste caso a operação invocada não é executada e o cliente será informado do motivo (p. ex., participante sem as credenciais adequadas). Caso contrário, a operação invocada é executada.

O pós-filtro é utilizado pelo **Manager MetaObject** para sinalizar a finalização da execução da operação pelo servidor. Esta sinalização é utilizada no **Coordination MetaObject** para fins de controle de concorrência.

Implementação do meta-objeto coordenador

O meta-objeto coordenador (**Coordination MetaObject**) é implementado como uma classe C++ que oferece ao **Manager MetaObject** as funcionalidades necessárias para a verificação da possibilidade de acesso de um cliente OCE ao servidor **Object Wrapper Server**. Para isto, em cada função membro da classe correspondente ao pré-filtro **OW_mmo**, é feita uma chamada à função *short Coordination MetaObject:: GetAccessRight(const OWCollabID& cid, ObjectID& oid, OpType& op)*. Esta função é dividida em duas partes, responsáveis respectivamente pela verificação das condições de controle de acesso e pelo controle de concorrência. Para que uma operação do servidor seja executada, o direito de acesso deve ser garantido pelas duas partes:

- Controle de acesso:

A sessão do **Coordination MetaObject,** que controla o acesso ao **Object Wrapper Server**, analisa o direito de acesso do colaborador em função dos parâmetros indicando o papel do colaborador *c_id* e o tipo de operação *op* a ser realizada. Inicialmente, o **Coordination MetaObject** interage com o serviço de gerenciamento de grupo através da invocação da operação *GetGroupMembers()* para saber se o participante está no grupo de participantes efetivos, e qual é o seu papel.

- Controle de concorrência:

Na versão atual do OCE é implementada uma política de controle de concorrência simplificado[2] , baseado no controle de execução das operações de atualização e leitura do servidor **Object Wrapper Server**. Operações de leitura (como p. ex., leitura de atributos de algum objeto de desenho) podem ser executadas a qualquer momento, enquanto que operações de atualização (como p. ex., adição, remoção ou alteração de um atributo de um objeto de desenho) só podem ser executadas se não existir alguma outra operação de escrita em andamento. A indicação da existência de alguma operação em execução é feita com a ajuda do pós-filtro implementado pelo **Manager MetaObject**, que indica a ocorrência deste fato. Neste caso a função *GetAccessRight()* do **Coordination MetaObject** retorna um erro ao **Manager MetaObject**.

O modelo de *threads* da plataforma Orbix+Isis utilizado (TPR - *Thread Per Request*) é não preemptivo. Uma vez que um *thread* entra em execução para atender uma invocação de operação do servidor, um segundo pedido de ativação só pode ser executado se o servidor ceder explicitamente o controle da execução do *thread* (isto pode ocorrer se a operação em execução fizer uma invocação à outro servidor. p. ex.). No caso do servidor **Object Wrapper Server** isto ocorre apenas ao final de cada operação de atualização, quando este invoca o servidor **Event Service** para notificar a atualização realizada aos demais clientes. Neste momento, as informações cuja consistência deve ser garantida já foram atualizadas e o controle de execução já pode ser passado a outro *thread* (para atendimento de uma invocação eventualmente pendente).

Entretanto, o controle de concorrência sobre operações de atualização foi mantido para que o servidor **Object Wrapper Server** possa ser portado para ambientes que suportem *threads* preemptivos, como no caso da versão *multi-threaded* da plataforma (Orbix-MT). Por outro lado, as operações de leitura não estão sujeitas ao controle de

[2] Mesmo que a implementaçao do servidor *Object Wrapper Server* seja centralizada, faz-se necessário ter uma política de controle de concorrência, já que é utilizado o modelo de *Threads Per Request* (TPR) da plataforma Orbix+Isis; no caso de utilizar o modelo *Thread Per Process* (TPP) isto não seria necessário.

concorrência para que os clientes OCE, que recebem a notificação da atualização realizada e desejam atualizar suas telas (por um *refresh*), possam faze-lo, requisitando a operação *ReadAllObjectAttributes(...)*, mesmo que o *thread* que esteja atendendo a atualização não tenha ainda terminado.

5.2 Considerações sobre a replicação dos objetos compartilhados e seu suporte

Uma característica importante em aplicações groupware é a possibilidade de manuseio de múltiplas cópias dos objetos compartilhados (replicação), tendo em vista aspectos de desempenho e tolerância a falhas. Neste item discute-se a forma de levar em conta a existência de objetos compartilhados replicados nos modelos apresentados e na sua implementação.

Replicação de Objetos Compartilhados

No modelo conceptual anterior, os objetos compartilhados são instâncias da classe **Shared Object**, que se abstrai de aspectos de localização geográfica das implementações destes, assim como da eventual existência de réplicas destes objetos em diferentes sítios.
O modelo de programação, que provê uma implementação destas classes do modelo conceptual, oferece transparência de localização através do uso de uma plataforma CORBA para o suporte à distribuição de objetos. Entretanto, no modelo de programação RTR-C a implementação dos objetos compartilhados é apresentada de forma centralizada, ou seja, cada objeto compartilhado é implementado como um servidor em um determinado sítio onde diversos clientes podem acessá-lo de maneira concorrente. As principais razões para esta escolha são: a simplificação no tratamento de problemas de controle de acesso e concorrência; e a não disponibilidade na especificação CORBA de mecanismos de replicação.
Arquiteturas replicadas, onde as informações compartilhadas e as funcionalidades das aplicações groupware são simétricas em todos os sítios onde se encontram participantes, têm sido entretanto largamente utilizadas [EG89, KBL93, CV95, FF95]. A replicação numa aplicação groupware oferece as vantagens de diminuir o tempo de resposta na leitura de informações compartilhadas e de aumentar o grau de tolerância a falhas da aplicação. O número de cópias de um objeto compartilhado (da classe **Shared Object**), em uma aplicação groupware com N participantes distribuídos em N sítios distintos, pode variar de 1 a N. Considera-se aqui o caso de N cópias, onde existe uma cópia de cada objeto compartilhado em cada sítio onde pode ser encontrado um participante.

Replicação no nível "meta" do Modelo de Programação RTR-C

O controle da replicação dos objetos compartilhados (da classe **Shared Object**) em N sítios reflete-se também no nível meta. Para cada réplica de objeto existe uma réplica correspondente, no mesmo processo do **Manager MetaObject** e do **Coordination MetaObject**.
É preciso, também, que as garantias de ordenação das invocações nos objetos compartilhados sejam também mantidas quando ocorrerem as invocações dos métodos em "nível meta". Ou seja, as invocações que são interceptadas pelas réplicas dos **Manager MetaObject** (e por conseqüência pelas réplicas do **Coordination MetaObject**), deverão ser recebidas na mesma ordem em que ocorre sua recepção pelos objetos compartilhados da classe **Shared Object**. Só assim pode-se fazer uso dos mecanismos de ordenação do suporte de comunicação na implementação de políticas de controle de concorrência sobre os objetos compartilhados.
A utilização da replicação no modelo RTR-C deve ainda levar em conta aspectos de escalonamento de *threads* que não comprometam as relações de ordenação na entrega de invocações. Por exemplo, para implementar uma aplicação virtualmente síncrona sobre os sistemas Isis [Bir93] e Orbix+Isis [II96], é necessário que o tratamento de recepção de mensagens/invocações seja não-preemptivo. Os meta-objetos escalonador e relógio do modelo RTR-C devem então considerar esta questão.

Suporte de comunicação

O suporte de comunicação, subjacente à plataforma CORBA utilizada para a implementação distribuída de objetos, deverá oferecer mecanismos de comunicação de grupo que permitam:

- a expansão de uma invocação efetuada por um cliente a um objeto para todas as N cópias deste, de forma transparente;
- a recepção da invocação de forma atômica em cada cópia, (ou seja, todas as cópias recebem a invocação, ou nenhuma delas a recebe);
- a realização das invocações ao grupo de objetos formado pelas N cópias de forma síncrona ou assíncrona;
- a alteração dinâmica do grupo de objetos, e a sincronização desta alteração com a entrega de invocações;
- o tratamento de invocações pelo grupo de objetos segundo critérios de ordenação. No caso particular de aplicações groupware, ao contrário do que ocorre em plataformas que oferecem critérios de ordenação mais fortes (total [Bir93], p. ex.), as políticas de ordenação podem levar em conta a semântica da operação sendo invocada. No modelo de programação proposto, o algoritmo de ordenação da invocação pode ser implementado no "nível meta" de cada cópia do objeto invocado. Entretanto, para descarregar a implementação em "nível meta", da responsabilidade de ordenação, pode ser necessário que o suporte de comunicação já ofereça um determinado nível de ordenação na entrega das invocações (FIFO, causal, etc.).
- a transferência de estado entre cópias de objetos para atualização daqueles que ingressam nos grupos.
- a utilização de *threads* não preemptivos, no caso de implementações de servidores *multi-threaded*, para garantia de relações de ordenação em algumas situações (sincronismo virtual [Bir93], p. ex.).

Com a existência de N réplicas de objetos e N clientes, torna-se importante a possibilidade de implementação dos objetos das classes **Collaborator** e **Shared Object**, que são encontrados em uma mesma máquina, em um mesmo processo (objetos *co-located*). Entretanto, o sistema de comunicação deverá poder distinguir entre os casos onde a comunicação se dá entre objetos *co-located* e onde esta ocorre entre objetos remotos, de forma a otimizá-la no primeiro caso.

Controle de concorrência sobre objetos replicados numa aplicação de groupware

Existem na literatura diversas abordagens para o controle de concorrência em aplicações groupware, em especial para o caso de editores distribuídos, e que podem ser agrupadas em três classes [Fri95]: sem controle de concorrência; com detecção de operações conflitantes; e com controle de concorrência estrito.

No primeiro caso, não existe por parte da aplicação nenhum suporte que garanta a consistência dos objetos compartilhados frente a atualizações concorrentes. Normalmente, situações conflitantes são deixadas ao encargo dos usuários e de interações explícitas entre os mesmos. O terceiro caso diz respeito aos algoritmos de passagem de bastão, estritos mas com limitações de desempenho.

No segundo caso, a aplicação se encarrega de manter a consistência dos objetos compartilhados a partir da detecção de operações conflitantes disparadas pelos colaboradores. Os métodos destas classes baseiam-se na alteração local das cópias dos objetos compartilhados e na difusão assíncrona destas atualizações. A consistência das réplicas é garantida posteriormente nos receptores, através de mecanismos de detecção de operações conflitantes apropriados e correção dos seus estados. Alguns algoritmos são apresentados em [SG85, Ste87, EG89, KBL93]. Os mais interessantes, por apresentarem resolução automática de conflitos e serem completamente distribuídos são os algoritmos dOpt (*distributed Operation transformation*) [EG89] e ORESTE (*Optimal REsponse TimE*) [KBL93]. Em [Fri95] pode ser encontrada uma apresentação mais detalhada dos algoritmos de controle de concorrência, assim como uma discussão comparativa a respeito.

6. Conclusão

Neste artigo, foram apresentados o projeto e a implementação de uma aplicação cooperativa composta por um editor de objetos gráficos e de um *"talk"* multi-usuário, com o intuito de ilustrar uma abordagem de cunho geral para este tipo de aplicação.
Na fase de projeto desta abordagem utiliza-se um modelo conceptual que permite estruturar a aplicação em termos de classes de objetos que representam para cada atividade, abstrações dos participantes, dos objetos compartilhados por estes e da coordenação; as relações entre estas e os componentes que representam serviços de gerenciamento de grupo e de eventos são também levadas em conta. Este modelo mostrou suas potencialidades em termos de expressividade das abstrações propostas e das informações associadas para o projeto de várias atividades cooperativas, em particular as do tipo das encontradas na aplicação implementada neste trabalho. Este modelo pode ainda ser utilizado em aplicações cooperativas que envolvam mídias contínuas desde que se adote uma abstração adequada para o objeto compartilhado, por exemplo, um canal de transferência de fluxo de mídia contínua (*"stream"*).
Na fase de implementação desta abordagem, utilizou-se um modelo de programação que transporta o modelo conceptual anterior para o ambiente de implementação. No nosso caso, este ambiente permite suportar distribuição e heterogeneidade segundo a especificação definida na plataforma CORBA-OMG. Para atender requisitos de flexibilidade, o modelo de programação introduz o uso do conceito de reflexão computacional para tratar os problemas da coordenação através do controle de acesso, do controle de concorrência e das eventuais questões associadas às restrições temporais. Desta forma é oferecida transparência aos participantes, tanto em termos de localização dos objetos compartilhados quanto da coordenação e controle de concorrência destes.
A aplicação cooperativa citada foi implementada e testada na plataforma CORBA ORBIX+ISIS [ION95] e posteriormente na ChorusCool da SunSoft. Os resultados alcançados nos permitiram validar os modelos conceptual e de programação da abordagem proposta e antever sua utilização para implementar aplicações cooperativas que envolvam mídias contínuas.

Agradecimentos

Agradecemos ao CNPq pelo financiamento (Protem-CC) do projeto ASAP do qual este trabalho fez parte.

Bibliografia

[Bir93] K. P. Birman. The process group approach to reliable distributed computing. *Communications of ACM*, 36(12), December 1993.
[CBDW92] G. Coulson, G. S. Blair, N. Davies, and N. Williams. Extensions to ansa for multimedia computing. *Computer Networks and ISDN Systems*, (25):305–323, 1992.
[CV95] F. J. N. Cosquer and P. Verissimo. Survey of selected groupware applications and supporting platforms. Technical report, INESC, 1995.
[EG89] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. In NY ACM, editor, *Proceedings of the ACM SIGMOD'89 Conference on the Management of Data*, Seattle, Wash., May 1989.
[FF95] U. Fritzke Jr. and J-M. Farines. Um suporte para aplicações de trabalho cooperativo do tipo editor distribuído. In *Anais do 13o. Simpósio Brasileiro de Redes de Computadores - SBRC'95*, pages 443–459, Belo Horizonte, MG, Brasil, Maio 1995.
[FF97a] U. Fritzke Jr. and J-M. Farines. Modelagem e implementação de aplicações de computação cooperativa em ambientes distribuídos heterogêneos. In *Proceedings of the XXIV Brazilian Hardware and Sofware Seminars*, Brasilia-DF, Brasil, August 2-8 1997.

[FF97b] U. Fritzke Jr. and J-M. Farines. Un modèle pour les applications coopèratives dans un environment réparti hétérogène. Technical report, LCMI-UFSC, Février 1997. Publication Interna LCMI-UFSC.

[FFF97] J.Fraga, J-M. Farines, O Furtado. RTR model: Na approach for dealing with real-time programming in open distributed systems. In *Proceedings of 3th IEEE Workshop on Object-oriented Real-time Dependable Systems (WORDS'97)*, New Port Beach, USA, Feb. 5-7 1997.

[FFFS95] J. Fraga, J-M. Farines, O Furtado, F. Siqueira. A programming model for real-time applications in open distributed systems. In *Proceedings of the 5th IEEE Workshop on Future Trends in Distributed Computing Systems* , Korea, August 1995.

[Fri95] U. Fritzke Jr. Projeto e implementação de um suporte para aplicações cooperativas do tipo editor distribuido. Dissertação Mestrado PG-EEL, Universidade Federal de Santa Catarina, Florianópolis, SC, Novembro 1995.

[II96] Isis Distributed Systems and IONA Technologies, ltd. *Orbix+Isis programmer's guide*. Marlboro, MA, USA, March 1996. Part Number D071.

[ION95] IONA Technologies, ltd. *Orbix Advanced Programmer's guide*. Iona Technologies Ltd., Dublin, Ireland, July 1995.

[KBL93] A. Karsenty and M. Beaudouin-Lafon. An algorithm for distributed groupware applications. In *Proceedings of the 13th. International IEEE Conference on Distributed Computing Systems*, pages 195–202, Pennsylvania, USA, May, 25-28 1993.

[KLL93] T. Kirsche, R. Lenz, H. Lürsen et al. Communication support for cooperative work. *Computer Communications*, 16(9), September 1993.

[Mae87] P. Maes. Concepts and experiments in computacional reflection. In *Proceedings of the OOPSLA'87*, pages 147-155, october 1987.

[OMG93] Object Management Group OMG. *The Common Object Request Broker Architecture*, December 1993. OMG Document 93-12-43, Revision 1.2.

[SG85] S. Sarin and I. Greif. Computer-based real-time conferencing systems. *IEEE Computer*, pages 33–45, October 1985.

[Ste87] M. Stefik. Beyond the chalkboard: Computer support for collaboration and problem solving in meetings. *Communications of the ACM*, 30(1):32–47, January 1987.

[TRB95] J. Trevor, T. Rodden, and G. Blair. COLA: A lightweight platform for CSCW. *Computer Supported Cooperative Work (CSCW)*, pages 197–224, 1995.