Integração das Especificações ROMIOP e ETF para Difusão Atômica no CORBA*

Daniel Borusch[1], Lau Cheuk Lung[1], Alysson Neves Bessani[2], Joni da Silva Fraga[2]

[1]PPGIA Programa de Pós-Graduação em Informática Aplicada
PUC-PR Pontifícia Universidade Católica do Paraná
R. Imaculada Conceição, 1155 Prado Velho CEP 80215-901 Curitiba PR

[2]DAS Departamento de Automação e Sistemas
UFSC Universidade Federal de Santa Catarina
Campus Universitário, Caixa Postal 476 CEP 88040-900 Florianópolis SC

{dborusch,lau}@ppgia.pucpr.br, {neves,fraga}@das.ufsc.br

Abstract. *OMG published a specification of a reliable ordered multicast inter-orb protocol for distributed applications developed in CORBA (ROMIOP). That specification was made to attend the demand of applications that need more restritive guarantees on reliability and ordenation, since already existed a specification without these resources (UMIOP). This article presents how ROMIOP was implemented as well as modifications that were made on the specification to make possible the creation of a better protocol. Perfomance measures were made comparing ROMIOP with others protocols, like UMIOP, to show it´s caracteristics and benefits against the rest.*

Resumo. *A OMG publicou uma especificação de um mecanismo de difusão confiável e ordenada para aplicações distribuídas desenvolvidas em CORBA (ROMIOP). Essa especificação foi criada para atender a demanda de aplicações que necessitam de garantias mais restritivas de acordo e ordenação, visto que já havia uma especificação sem esses recursos (UMIOP). Este artigo apresenta como foi implementado o ROMIOP bem como alterações que foram realizadas na especificação para possibilitar a criação de um protocolo melhor. Foram feitas medidas de desempenho comparando o ROMIOP com outros protocolos, como o UMIOP, para demonstrar as características e benefícios do mesmo em relação aos demais.*

1. Introdução

A arquitetura CORBA (*Common Object Request Broker Architecture*) [OMG, 2002a], criada pela OMG (*Object Management Group*), tem como principal componente o ORB (*Object Request Broker*). Ele possibilita que os objetos recebam e realizem invocações de modo transparente em um sistema distribuído, sendo considerado a base para a

* Este trabalho faz parte do projeto GroupPac/MJaco financiado pelo CNPq (Edital Software Livre 401802/2003-5).

interoperabilidade entre aplicações em ambientes heterogêneos. Para realizar a troca de mensagens entre os ORBs, há um elemento que especifica uma sintaxe de transferência padrão além de um conjunto de formatos de mensagens conhecido como GIOP (*General Inter-ORB Protocol*). A implementação do GIOP para o protocolo TCP/IP é conhecida como IIOP (*Internet Inter-ORB Protocol*), que usa como base à comunicação ponto a ponto, ideal para aplicações do tipo cliente/servidor. Entretanto, diversas outras áreas de aplicação necessitam disseminar a mesma mensagem para uma infinidade de *hosts*. Uma das formas de fazer isso é utilizando o *multicast* IP, que compreende um conjunto de extensões ao protocolo IP que o habilita na concretização de comunicações multiponto [Deering, 1986].

Visto que não havia nenhuma especificação que descrevia a utilização da comunicação multiponto na arquitetura CORBA, em 2001 a OMG publicou a especificação do UMIOP (*Unreliable Multicast Inter-ORB Protocol*) [OMG, 2003a]. O UMIOP foi proposto para prover um mecanismo comum de entrega de requisições sem garantias via *multicast*. O protocolo de transporte padrão definido para o UMIOP foi o *multicast* IP sobre UDP/IP, que diferentemente do TCP/IP, não é orientado à conexão, com isso apenas invocações *one-way* podem ser realizadas. Não tendo nenhuma garantia, o MIOP (protocolo definido na especificação UMIOP) pode ser caracterizado como sendo de alto desempenho, considerado ideal para aplicações como *streaming* de áudio e vídeo, em que a perda de alguns pacotes pode ser tolerada. Contudo, diversas aplicações não podem tolerar perdas de pacotes, necessitando de garantias mais restritivas como acordo e ordenação. Para isso, como a OMG não havia publicado até o momento nenhuma especificação que apresentasse uma solução, o nosso grupo de pesquisa propôs o protocolo ReMIOP (*Reliable Multicast Inter-ORB Protocol*) [Bessani et al., 2003] para atender a essa demanda.

Entretanto, no final de 2002, a OMG publicou uma especificação preliminar (*draft*), planejada para ser padronizada em 2005, que apresenta uma solução utilizando para isso um protocolo de comunicação multiponto com garantia de entrega e ordenação total (ou seja, difusão atômica [Défago et al., 2004]). Esta especificação foi denominada ROMIOP (*Reliable, Ordered, Multicast Inter-ORB Protocol*) [OMG, 2003b]. O ROMIOP, assim como o UMIOP, também utiliza o *multicast* IP sobre UDP/IP, porém invocações com retorno de resposta (*two-way*) são suportadas. Fazendo um estudo detalhado do ROMIOP é possível verificar que essa especificação fornece uma série de interfaces IDL, algumas ainda confusas, dando uma larga margem de interpretações. Ou seja, a especificação não dá muitos detalhes de como as interfaces devem ser implementadas e que algoritmos de ordenação total adotar [Défago et al., 2004].

Além dessas especificações, a OMG publicou recentemente a especificação ETF (*Extensible Transport Framework*) [OMG, 2003] que define um arcabouço (*framework*) que permite a terceiros projetar e implementar *plugins* adicionais de protocolos de transporte de mensagens GIOP no ORB. Esta especificação, que já é implementada na grande maioria dos ORBs disponíveis, torna possível a extensão de um ORB/CORBA através da adição de novos protocolos de transporte sem que haja modificações significantes na sua estrutura. No entanto, a ETF foi concebida visando apenas protocolos de transporte ponto-a-ponto, para multiponto, extensões na especificação são necessárias.

Este artigo propõe, como principal contribuição, um conjunto de extensões para efetivar a integração das especificações ROMIOP e ETF, englobando aspectos arquiteturais, conceituais e algumas decisões de projeto. A solução proposta é completamente interoperável, sem o uso de interfaces proprietárias, e totalmente de acordo com as especificações da OMG. Nesse sentido, podemos considerá-la bastante próxima ao que seria uma solução definitiva em termos de comunicação de grupo com ordem total no CORBA. Definimos um algoritmo de difusão atômica para ser implementado, na forma de um *plugin* ETF, dentro ROMIOP como forma de validação da arquitetura proposta, algumas medidas mostrando o custo da ordenação total dentro do ORB também são mostradas.

Este artigo está organizado da seguinte forma: na seção 2 são apresentados alguns trabalhos relacionados. Na seção 3, a arquitetura do MJaco e um conjunto de extensões à especificação ROMIOP é apresentada. Na seção 4, é introduzido o algoritmo de ordenação total utilizado. A seção 5 apresenta algumas considerações a respeito da implementação. Alguns resultados obtidos com o ROMIOP podem ser visualizados na seção 6 e finalmente, a seção 7 apresenta conclusões deste trabalho.

2. Trabalhos Relacionados

Comunicação de grupo no CORBA foi e continua sendo um tema de bastante interesse. Os primeiros trabalhos sobre o assunto utilizavam ferramentas comerciais de comunicação de grupo. Esses trabalhos podem ser identificados na literatura dentro três soluções básicas: as abordagens de integração [Maffeis, 1995], de serviço [Felber, 1998] e de interceptação [Moser et. al., 1998, Lau, 2001].

A abordagem de integração consiste na construção ou na modificação de um ORB existente, adicionando mecanismos de processamento em grupo. A idéia principal nesta abordagem é que o processamento de grupo seja suportado por um sistema de comunicação de grupo abaixo do núcleo do ORB. Já na abordagem usando objetos de serviço é prover o suporte a grupos de objetos como um conjunto de serviços no topo do ORB, e não como parte do próprio ORB. Finalmente, a abordagem de interceptação prevê que as mensagens enviadas aos objetos servidores devem ser capturadas e mapeadas em um sistema de comunicação de grupo, de maneira transparente para a aplicação.

Em [Bessani et. al., 2004] é proposta uma implementação de difusão atômica sobre o MJaco. Esta implementação utiliza os protocolos MIOP e IIOP no desenvolvimento de um sistema ótimo em vários aspectos. Um ponto negativo desse trabalho em relação ao ROMIOP é que ele depende da pesada infra-estrutura do FT-CORBA, implementada através do sistema GroupPac.

A presente proposta tem a virtude de ter disponível as últimas especificações OMG relacionadas à comunicação de grupo. O uso dessas permitiu-nos alcançar a completa interoperabilidade e portabilidade do ORB, requisitos fundamentais de qualquer especificação da OMG.

3. Arquitetura MJaco

O MJaco (http://grouppac.sourceforge.net) é um *middleware* CORBA com suporte para comunicação de grupo baseado nas especificações UMIOP, padronizadas pela OMG.

Este *middleware* permite a difusão de mensagens de forma não confiável, de acordo com o padrão UMIOP, ou confiável, implementados pelos protocolos ReMIOP [Bessani et. al., 2003a] e ROMIOP, todos os três baseados na pilha UDP/*multicast* IP.

Na figura 1, temos o ORB com as duas pilhas de protocolos: uma para a comunicação ponto a ponto, baseada em IIOP, utilizando os serviços do TCP/IP, e outra para a comunicação multiponto formada pelo MIOP e ROMIOP, que utiliza UDP/*multicast* IP. O modelo de integração apresenta os vários elementos definidos na especificação que compõem o suporte para os dois modelos de comunicação.

A primeira fase do projeto do MJaco foi a integração do UMIOP no ORB e a sua implementação. Dando continuidade a este projeto, foi implementado o protocolo ReMIOP, que estende as especificações UMIOP com propriedades de difusão confiável, oferecendo garantias do tipo "melhor esforço" de que mensagens difundidas serão entregues por todos os membros corretos de um grupo. Por fim, o ROMIOP foi adicionado ao rol de protocolos, sendo este último o único que, além de garantir a difusão confiável, atende a garantia de ordenação total das mensagens, ou seja, todos os membros do grupo entregam todas as mensagens na mesma ordem.

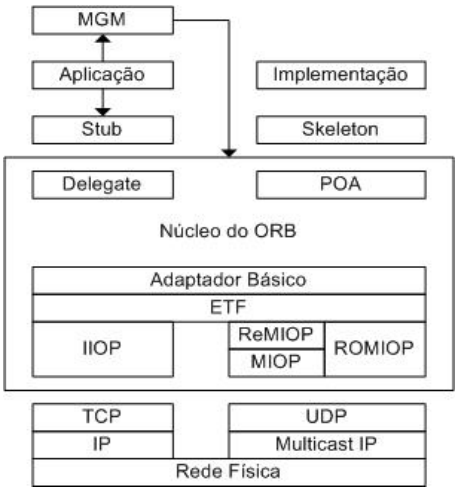

Figura 1: Arquitetura do MJaco.

É importante notar a forma em que o ROMIOP foi introduzido no MJaco. Ao invés de colocá-lo em uma camada acima do ReMIOP, ou até mesmo no mesmo nível do ReMIOP utilizando como base o MIOP, ele foi implementado a partir da base. Isso, como será visto mais adiante, foi feito devido a diferenças consideráveis no formato da especificação desses protocolos, sendo preferível iniciar sua implementação sem tomar como base nada do MIOP.

3.1. A Especificação ETF

A ETF é a especificação de uma plataforma que permite a terceiros projetar e implementar *plugins* de transportes de mensagens. Com ela, os diversos *middlewares* que implementam o CORBA tornam-se muito mais flexíveis devido ao fato de que há na especificação todas as classes e métodos que devem (ou podem, no caso dos opcionais) ser implementados e quais funcionalidades cada um deve ter, tornando desnecessário modificar o código do ORB em si. Isto assegura as propriedades de portabilidade e interoperabilidade do ORB.

O grande problema desta especificação é que ela define apenas como adicionar *plugins* de transporte ponto a ponto, o que a torna deficiente para protocolos multiponto, tal como o ROMIOP. Devido a esse fato, o *middleware* escolhido teve de ser levemente alterado, adicionando a funcionalidade de envio de mensagens para grupos.

Basicamente, a especificação define quatro interfaces obrigatórias: *connection, profile, listener e factories*. A primeira separa a camada de mensagens (GIOP) da camada da ETF, criando um canal de interação entre as mensagens e as conexões (tanto de clientes quanto de servidores). A segunda armazena todas as informações relativas ao protocolo, como métodos para enviar (*marshalling*) e receber informações através da IOR. A terceira provê a iniciativa de "ser conectada" a uma requisição feita por um cliente, direcionando essa requisição para um servidor. A quarta e última interface é a responsável por criar instâncias de clientes, sendo ela a responsável pela ligação do ORB com o *plugin* (a figura 4, na seção 5, apresenta a implementação do ROMIOP como um *plugin* ETF).

3.2. A Especificação ROMIOP

O ROMIOP (atualmente um *draft* [OMG, 2003b]) define um conjunto de interfaces para provisão de comunicação multiponto com garantia de entrega e ordem total para todos os membros não faltosos de um de grupo de objetos. Ele suporta tanto requisições que necessitam de resposta (*two-way requests/replies*) quanto as que não precisam (*oneway requests*). Esta especificação foi projetada para que os protocolos que a implementam coexistam tanto com o IIOP, quanto com MIOP e qualquer outro protocolo de comunicação *multicast*, não podendo interferir no funcionamento destes.

A especificação define uma interface de configuração dos vários possíveis métodos para consolidação de respostas de requisições e qualidade do serviço de ordenação. Em relação ao primeiro fator há dois meios distintos para realizar a consolidação: **votação simples**, onde há três possibilidades para definição da resposta (primeira recebida, última recebida e a primeira que satisfaça o parâmetro de consistência de dados); **votação por quorum**, onde há duas possibilidades para definição da resposta (número de membros que enviam a mesma resposta e porcentagem de membros), ambas dependentes do parâmetro de consistência.

O parâmetro de consistência de dados determina três possibilidades: todas as respostas devem ser iguais; todas as respostas devem ser diferentes (aparentemente sem utilidade); e a padrão, em que a maioria das respostas iguais prevalece. É interessante notar que com essa última opção é possível fornecer um suporte limitado à tolerância de faltas, utilizando-se da idéia de replicação da máquina de estados [Schneider, 1990]. Além disso, há mais um parâmetro relativo à consolidação de respostas que acaba sobrepondo-se a todos os outros. Pode-se configurar um tempo limite para o recebimento das respostas, e mesmo se a opção escolhida ainda não tiver sido atendida, o processo de consolidação é forçado com os resultados que já foram obtidos.

Uma outra interface prevista na especificação é o serviço de grupo, que fornece operações básicas como adição e remoção de membros, além de criação de grupos.

Por fim, em termos de protocolos de comunicação, o ROMIOP define apenas formatos para vários tipos de mensagens. Este formato consiste em um cabeçalho que possui um tamanho fixo e trás informações relativas aos dados contidos no pacote (como seu tipo, número de identificação, posição do pacote dentro de um conjunto de mensagens, dentre outros) e uma área de dados. Os tipos de pacotes definidos são: *request*, *reply*, ACK, NAK, *keepalive*, *cancel* e *memberchange*. Cada tipo possui uma estrutura diferente na área de dados, de acordo com a semântica do pacote. Dentre os

campos podemos citar: para qual membro o pacote é direcionado, endereço de ACK/NAK, ID do pacote já recebido, status do membro, dentre muitos outros.

Cada um dos tipos de pacotes acima citados vem sempre acompanhado inicialmente por um cabeçalho padrão (*PacketHeader*) que informa, além do tipo do pacote seguinte, informações complementares, como se há a necessidade de envio de ACK, o tamanho do identificador único do pacote (que pode variar entre 4 valores distintos: 4, 8, 16 ou 32 bytes), versão do protocolo dentre outras. Abaixo pode ser vista justamente essa definição acima descrita.

```
module ROMIOP {
     typedef octet PacketType;
     const PacketType Request = 0;
     const PacketType Reply = 1;
     const PacketType ACK = 2;
     const PacketType NAK = 3;
     const PacketType KeepAlive = 4;
     const PacketType Cancel = 5;
     const PacketType MemberChange = 6;
     const PacketType MsgOrder = 7;
     struct PacketHeader_1_0 {
        char magic[4];
        octet version;
        octet flags;
        short packet_type;
        unsigned long packet_size;
    }; };
```

3.3. Extensões Propostas no ROMIOP e ETF

Visto que até a presente data, a especificação do ROMIOP ainda não foi concluída, diversas considerações tiveram de ser tomadas e adicionadas para que a implementação desta especificação fosse realizável. A mais importante foi a criação de mais um tipo de mensagem, a que contém a ordem das mensagens enviadas pelo receptor líder (*MsgOrder*). Sua definição é bem simples: há um identificador de quem enviou a mensagem e uma lista com uma estrutura também criada que contém o identificador da mensagem e a identificação do membro emissor que enviou a mensagem.

Outro ponto crucial é em relação à consolidação das respostas. Somente é dito quais modelos devem ser implementados (votação simples ou por quorum), porém não é informado aonde deve ocorrer a consolidação em si (no emissor ou nos receptores). Neste modelo, o algoritmo de consolidação é processado no cliente que requisitou as respostas, tirando dos servidores essa carga extra de processamento. Essa questão pode ser considerada a mais trabalhosa para ser desenvolvida.

Devido ao fato de não haver nenhum módulo relacionado à consolidação de respostas na especificação da ETF, bem como a ausência de material relacionado ao assunto em outros trabalhos, toda uma modelagem teve de ser desenvolvida para solucionar o problema. Diversas abordagens foram analisadas, implementadas e testadas antes de se chegar ao método definitivo que atendesse a especificação incompleta do ROMIOP. Justamente devido a esse fato que o protocolo não utilizou como base nenhum outro previamente existente (ver figura 1 na seção 3). O controle das respostas recebidas não passa pelo protocolo, sendo função do *middleware* definir a quem pertence a resposta. Essa abordagem teve de ser contornada, visto que desse

modo apenas a primeira resposta seria utilizada, descartando todas as outras (fato que ocorreria na existência de mais de 1 servidor). A solução escolhida foi criar um desvio dessa informação para o protocolo do ROMIOP, após o envio da mesma para o *middleware*, atendendo dessa forma ambas as especificações (ETF e ROMIOP).

Apesar de aparentar ser um método lento (a mesma informação passa duas vezes pelo protocolo) os testes realizados (ver seção 6) comprovaram que a perda de performance foi insignificante. Além disso, esse método possui a vantagem de modularizar as funções: mensagens que necessitam ser consolidadas passam por esse processo enquanto as que não, seguem o caminho direto. Dessa forma cada cliente processa a consolidação de suas requisições enquanto os servidores ficam com a tarefa de responder as requisições e definir a ordem das mensagens.

Justamente devido à necessidade de consolidar as respostas é que foi implementado um serviço de membros (*membership*) com mais funcionalidades do que o existente no MJaco. É preciso saber exatamente a quantidade de membros funcionais para que o cliente saiba quantas respostas esperar. Todo esse módulo foi projetado sem nenhum tipo de especificação. Visto que o protocolo necessita manter uma lista com os membros atualizada, e o mesmo também deve ser capaz de lidar com falhas de omissão, falhas de *crash* e problemas com a rede física (cabo de rede cortado, roteador mal configurado), foi implementado um algoritmo executado pelo servidor líder. Nele, a cada intervalo de tempo pré-determinado, é enviada uma mensagem para o grupo perguntando se estão vivos (*alive*). Todos os que não enviarem um ACK para esta mensagem são excluídos do grupo, ou seja, é assumida a abstração de detector perfeito.

Finalmente, a especificação em seu estágio atual não define o algoritmo de ordenação total utilizado. Assim foi definido um algoritmo baseado em ordenador fixo [Défago et. al., 2004], apresentado na seção 4, no sentido de verificar as potencialidades da arquitetura proposta.

4. Algoritmo de Ordenação Total

Um algoritmo de ordenação total com difusão confiável é aquele que garante que todos os processos não-faltosos pertencentes a um grupo entreguem o mesmo conjunto de mensagens na mesma ordem [Défago et. al., 2004]. Esse tipo de algoritmo também é chamado de difusão atômica porque a entrega da mensagem ocorre como se fosse uma primitiva indivisível: a mensagem é entregue para todos ou para nenhum, e se entregue, todas as outras mensagens são ordenadas antes ou depois dessa.

4.1. Modelo de Sistema e Premissas

Em relação a falhas de processos, assumimos um modelo de falhas de parada (*crash*). As falhas de processos são toleradas na medida em que o módulo de gerência de grupos (*membership*) mantém uma lista dos membros atualizada.

Em relação aos canais, assumimos uma semântica de entrega perfeita para mensagens. Esta semântica é implementada a partir da retransmissão periódica das mensagens até que todos os processos receptores acusem o recebimento da mesma (através de uma mensagem ACK). Mensagens duplicadas são detectadas através de seu identificador e descartadas, porém ACKs para estas mensagens são enviados antes de seu descarte uma vez que o ACK pode ter sido perdido pelo seu emissor.

Suposições em relação ao tempo tiveram de ser realizadas para implementar o protocolo do ROMIOP. Para determinar o reenvio das mensagens devido ao não recebimento de ACKs suficientes foi adotado que a soma dos tempos de computação e comunicação é síncrono, ou seja, há um limite superior conhecido para que ambos ocorram. Outra suposição em relação ao tempo tomado foi o de um detector de falha perfeito. Se o tempo levado para chegar a resposta da mensagem perguntando se o processo está vivo (*keepalive*) ultrapassar um certo valor, aquele processo será considerado com problema e será excluído do grupo.

Se algum processo travar e não retornar (*crash*) não há problemas, visto que o módulo de membership mantém uma lista da quantidade de membros atualizada. Uma outra possibilidade é o travamento de um processo e depois seu retorno. Nesse caso, dependendo do tempo que o processo permanecer sem responder ele poderá ser removido do grupo, porém ao retornar, ele será re-incluído na lista de membros. O único problema relacionado a essa última possibilidade é que qualquer resposta oriunda de uma requisição recebida anteriormente ao travamento será descartada pelos receptores.

Por fim, foi implementado um algoritmo de eleição de líder em que o primeiro processo a entrar no grupo é considerado o líder. Isso é realizado enviando-se uma mensagem e esperando por um certo tempo. Se ninguém o responder, ele se considera o líder e avisa a todos que dali pra frente entrarem no grupo da existência do líder. É importante frisar que todos os parâmetros de *timeout* relacionados a tempo são configuráveis para que o protocolo trabalhe da melhor forma possível em cada ambiente de rede.

4.2. Algoritmo

O protocolo adotado para difusão atômica no ROMIOP é baseado no paradigma do ordenador fixo [Défago et al. 2004]. Basicamente, o protocolo funciona da seguinte forma: o emissor difunde uma mensagem solicitando aos membros do grupo sua entrada nesse mesmo grupo. Em seguida, envia sua requisição para o endereço do grupo e fica aguardando uma quantidade de ACKs iguais à quantidade de receptores presentes no grupo. Se a quantidade de ACKs for menor do que o esperado, a requisição é reenviada para o grupo. O algoritmo simplificado do ROMIOP é apresentado na figura 3.

Inicialmente, todos os buffers são inicializados com valores vazios (linha 1). Todas as mensagens a serem enviadas, tanto por clientes quanto servidores, precisam, antes, ser armazenadas (linha 8) e um *timer* precisa ser criado (linha 9) para cada uma delas, menos as do tipo ACK. Isso é necessário, pois todos esses tipos de mensagens necessitam de uma confirmação de que a mensagem enviada efetivamente chegou no destino, sinalizado com o envio de um ACK. Se o ACK não chegar ao emissor da mensagem, essa é reenviada após o término do *timer*. Quando uma quantidade suficiente de ACKs é recebida (linha 42), o vetor que armazenava a mensagem é esvaziado (linha 44), e o *timer* de reenvio da mesma é parado (linha 43).

As requisições (*requests*) recebidas por cada servidor são armazenadas em seu buffer temporário (linha 17). Assim que o receptor líder (o primeiro a entrar no grupo) receber uma requisição qualquer, um *timer* configurável é inicializado. Após o término

desse *timer* é enviado para todos os membros servidores do grupo uma mensagem do tipo *msgOrder* com a ordenação de todas as mensagens de requisição (*request*) recebidas. Todos os servidores ao receberem essa mensagem (linha 33), comparam os identificadores com os armazenados no buffer de mensagens já recebidas (linha 35). Todas as mensagens que o receptor em questão possuir, partindo-se da primeira e indo seqüencialmente até a ultima, serão entregues (delivered, linha 36). Se por alguma razão esse receptor não possuir uma das mensagens, todas as subseqüentes não serão entregues, até que a faltante seja recebida.

```
1.  {Inicialização}
2.  buffer <- ϕ {buffer de mensagens enviadas}
3.  received <- ϕ {buffer de requisições recebidas e não entregues}
4.  order <- ϕ {buffer com a ordem das requisições a serem entregues}
5.  members <- myself.id {lista dos membros do grupo}

6.  {To R-multicast(m)} {difusão de uma mensagem para o grupo}
7.  if m.type = REQUEST | REPLY | MEMBERCHANGE | MSGORDER then {m.type:
    tipo da mensagem m}
8.        buffer <- buffer U {m}
9.        timeout(m, 2000) {inicia um timeout para reenvio se necessário}
10.       send(m) {envia a mensagem}
11. end if
12. else if m.type = ACK then
13.       send(m)
14. end if

15. {To R-receive(m)} {recebimento de uma mensagem}
16. if m.type = REQUEST then
17.       received <- received U {m}
18  end if
19. else if m.type = REPLY then
20.       consolidate(m) {consolida as respostas recebidas}
21. end if
22. else if m.type = MEMBERCHANGE then
23.       if m.status = added then {m.status: tipo da mudança do membro}
24.             members <- members U {m.id}
25.       end if
26.       else if m.status = removed then
27.             members <- members \ {m.id}
28.       end if
29.       else if m.status = online then
30.             R-multicast(members)
31.       end if
32. end if
33. else if m.type = MSGORDER then
34.       order <- order U {m}
35.       while m.id e received.id do
36.             deliver(received) {entrega a mensagem na ordem correta}
37.             order <- order \ m
38.             received <- received \ m
39.       end while
40. end if
41. else if m.type = ACK then
42.       if enoughACK(m) = OK then {se recebeu quantidade suficiente de
    ACKs}
43.             timeout(m).stop {pára o processo que reenvia mensagem}
44.             buffer <- buffer \ m
45.       end if
46. end if
```

Figura 3: Algoritmo simplificado do ROMIOP.

Logo após o processamento da mensagem pelo servidor, se a mensagem necessitar de retorno (*two-way*), uma mensagem de resposta (*reply*) será enviada para o endereço do grupo (linha 7). O cliente emissor que enviou a mensagem de request permanece aguardando uma certa quantidade de respostas pré-determinadas, dependendo de como a consolidação foi configurada (linha 19).

Por fim, sempre que um objeto deseja entrar no grupo, inicialmente é enviada uma mensagem com o status *added*. Todos os membros que recebem essa mensagem com esse status adicionam o membro a sua lista de membros (linha 23). Após o membro receber a confirmação de que entrou no grupo, ele envia uma mensagem solicitando a lista atualizada de membros, indicando-a com o status *online*. Todos os membros, ao receber essa mensagem (linha 29), retornam a sua lista de membros do grupo.

5. Implementação do ROMIOP sobre a ETF

Para melhor apresentar o protocolo e a forma de implementação que foi adotada para satisfazer os requisitos da especificação ETF, segue na figura 4, um diagrama de classes apenas com os métodos e atributos extremamente essenciais, além de serem mostradas apenas as relações entre classes de importância vital para o correto entendimento.

Basicamente, na chegada de qualquer pacote de mensagem, através da classe ROMIOPConnection , o fragmento, após diversas verificações, é adicionado a um objeto da classe FragmentedMessage . Quando uma mensagem estiver completa ela então é entregue ao algoritmo do ROMIOP, para realização da ordenação.

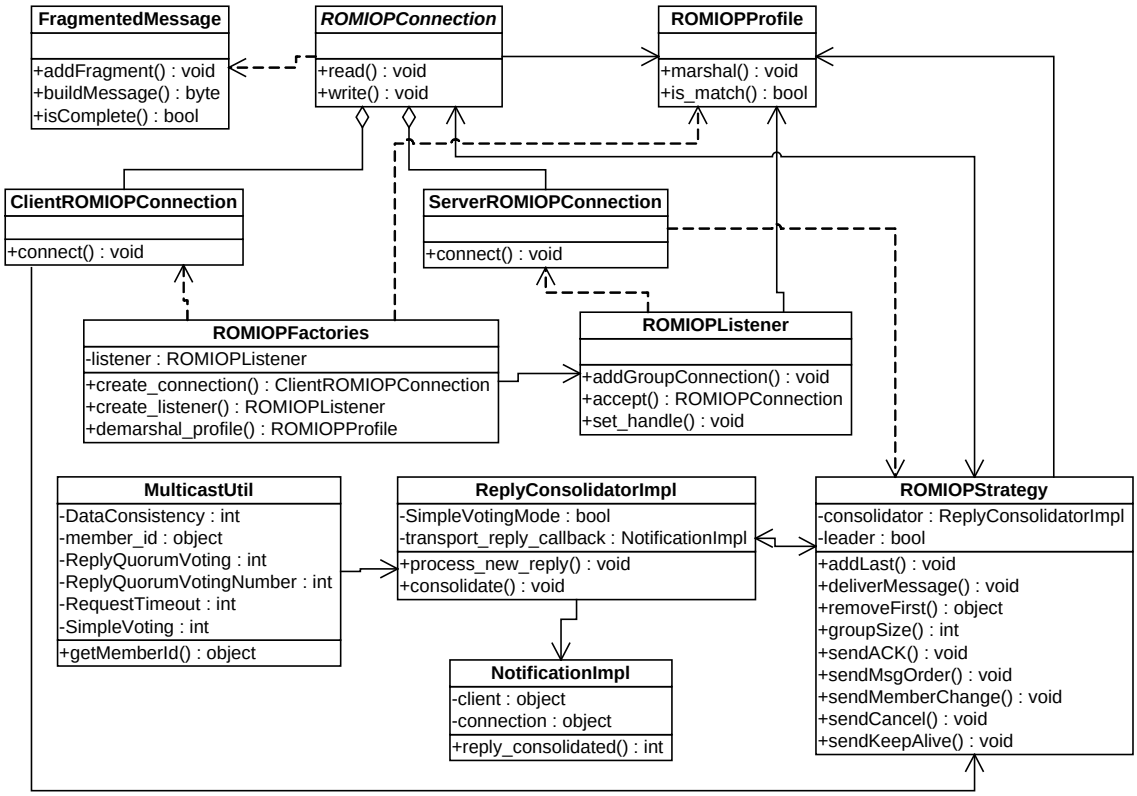

Figura 4: Diagrama de classes simplificado do *plugin* ROMIOP.

Praticamente todas as classes do protocolo utilizam-se das classes base MultciastUtil e ROMIOPProfile , sendo cada uma responsável por guardar

informações específicas. A primeira relaciona-se com todo o protocolo, guardando configurações do protocolo, como método de consolidação e tempos limites. A segunda está unicamente ligada a uma conexão, sendo responsável por guardar informações desta, como o endereço do grupo.

Para mensagens que necessitam de resposta, as classes ReplyConsolidatorImpl e NotificationImpl são utilizadas, sendo a segunda controlada pela primeira e esta controlada pela classe ROMIOPStrategy . Enquanto a classe ReplyConsolidatorImpl é acionada na criação da mensagem que necessita de resposta, bem como na chegada de qualquer uma das respostas, além de efetivamente consolidar a resposta, a classe NotificationImpl só é usada para notificar o ORB da resposta escolhida. Por fim, a classe ROMIOPStrategy mantém todos os outros processos necessários para o correto funcionamento do protocolo, sendo por isso considerada a mais complexa e com mais funcionalidades. Algumas de suas funções estão o envio dos ACKs, controle de membership e envio da ordem das mensagens.

6. Avaliação de Desempenho

Com o intuito de analisar o desempenho MJaco, bem como as opções feitas na implementação do mesmo, foram realizados diversos testes. O ambiente em que os testes foram realizados foi um conjunto de máquinas rodando o Windows XP. O cliente foi executado em uma máquina Athlon 2600+, com 512MB de memória RAM. O servidor líder foi executado em um Pentium 4 de 2.6GHz com 1,5GB de memória RAM. Os outros dois servidores foram executados em máquinas Athlon de 1.47GHz com 248MB de memória RAM cada. O teste a seguir foi realizado para analisar o algoritmo de consolidação, onde foi comparada a opção atômica (onde o cliente aguarda a resposta de cada servidor) com a de primeira resposta. A figura 5 abaixo mostra os resultados com 2 e com 3 servidores.

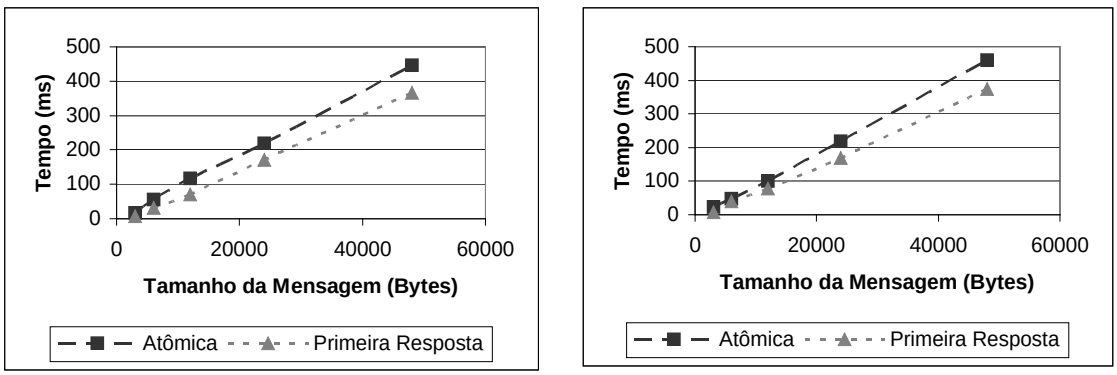

Figura 5: Comparação de dois tipos de consolidação com 2 e 6 servidores.

A análise dos gráficos trás, além do esperado que o tempo de consolidar somente a primeira resposta ser menor que o de todas, o fato de que para pequenas mensagens (aproximadamente até 3000 bytes) não há praticamente qualquer ganho de performance, sendo nesse caso mais vantajoso usar a consolidação atômica, que trás mais segurança. Com esse resultado, nota-se que para grandes tamanhos de mensagens é crucial a escolha do método de consolidação: se a aplicação necessita suportar faltas bizantinas ou apenas deseja-se ter certeza da resposta obtida (como sistemas de suporte a vida ou

aplicações militares) haverá um custo considerável. É importante notar que mesmo com o incremento do número de servidores, o tempo para se obter a resposta oriunda da consolidação via primeira resposta praticamente não aumenta.

7. Conclusão

Este artigo apresentou um estudo sobre a implementação da especificação preliminar ROMIOP utilizando os princípios da especificação ETF. A primeira especificação visa padronizar interfaces e formatos de mensagens para difusão de mensagens com garantias como confiabilidade e ordenação total, enquanto a segunda define métodos para integração de novos protocolos em sistemas já existentes. A partir da finalização desta especificação, teremos finalmente mecanismos interoperáveis de comunicação de grupo em *middleware* aberto (padronizado). Com o ROMIOP agora concluído é possível analisar cada módulo minuciosamente em busca de um melhor desempenho tornando a implementação mais eficiente e com mais funcionalidades.

Mais informações relativas à testes de desempenho, ao próprio algoritmo desenvolvido, bem como códigos fonte podem ser encontradas na internet, na página web do grupo de desenvolvimento (http://grouppac.sourceforge.net).

Referências Bibliográficas

Bessani A. N., Fraga J. S., Lung L. C. (2003a). ReMIOP: Projeto e Implementação de um Mecanismo de Difusão Confiável no CORBA. Anais do XXI SBRC 03, Natal RN.

Bessani A. N., Fraga J. S., Lung L. C., et. al.(2003b). Integrating the unreliable multicast inter-orb protocol in mjaco. In Proc. of the 4th IFIP WG 6.1 Inter. Conf. on Distributed Applications and Interoperable Systems - IFIP DAIS 03, LNCS v2893, Paris - France.

Bessani, A. N., Fraga, J. S., Lung, L. C., Alchieri, E. A. B (2004). Active Replication in CORBA: Standards, Protocols and Implementation Framework. In: 6th Int. Symposium on Distributed Objects and Applications, 2004, Larnaca, Cyprus. Proc. of DOA'04.

Deering, S. E. (1986). Host extensions for ip multicasting (rfc 988). IETF RFC.

Défago X., Schiper A. e Urbán P. (2004) Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372-421, Dec. 2004. ACM Press.

Felber, P. (1998), "The CORBA Object Group Service A Service Approach to Object Groups in CORBA", PhD. Thesis, École Polytechnique Fédérale de Lausanne, Lausanne,

Lau C. L., J. S. Fraga, J. Farines, M. Ogg, A. Ricciardi, CosNamingFT A Fault-Tolerant CORBA Naming Service 18th IEEE Symposium on Reliable Distributed Systems SRDS 99, Lausanne, Suiça, October 1999.

Maffeis, S., Run-Time Support for Object-Oriented Distributed Programming, Ph.D. Thesis University of Zurich. Zurich, 1995.

Moser, L. E., P. M. P. Melliar-Smith, Narasimhan, P., Consistent Object Replication in the Eternal System, Theory and Practice of Object Systems, 4(2): 81-92, 1998.

OMG (2002). The Common Object Request Broker Architecture v3.0. OMG Doc. 02-06-33.

OMG (2000).Object Management Group, Fault-Tolerant CORBA Specification V1.0, OMG document: ptc/2000-04-04, April, 2000. www.omg.org.

OMG (2002a). *The Common Object Request Broker Architecture* v3.0. OMG Doc. 02-06-33.

OMG (2003). Extensible transport framework specification v1.0. OMG Standard.

OMG (2003a). Unreliable multicast inter-orb protocol spec v1.0. OMG Doc. ptc/03-01-11.

OMG (2003b). Reliable, ordered, multicast inter-orb protocol. Revised Submission OMG Document realtime/2003-10-04. October, 2003.

Schneider, F. B. (1990) Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4): 299-314, Dec. 1990. ACM Press.