Replicação Ativa no CORBA: Padrões, Protocolos e *Framework* de Implementação*

Alysson Neves Bessani[1 †], **Joni da Silva Fraga**[1 ‡], **Lau Cheuk Lung**[2], **Eduardo Adílio Pelinson Alchieri**[1 §]

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

[2]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

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

Abstract. *This paper presents our experience in integrating the FT-CORBA and UMIOP specifications in a single middleware plataform. This integration model defines a large spectrum of middleware support for distributed objects group communication, which was used to develop an active replication model for FT-CORBA. The algorithms for reliable and atomic multicast needed by this model were developed based on FT-CORBA and UMIOP object models. These algorithms are presented using theoretical concepts for expressing their main features. At last, our FT-CORBA and UMIOP integration is compared to related experiences described in the literature and other active replication protocols.*

Resumo. *Este artigo apresenta a nossa experiência na integração das especificações UMIOP e FT-CORBA em uma plataforma única de middleware. Este modelo de integração resulta em um amplo espectro de primitivas de comunicação que sustentam um modelo de replicação ativa para o FT-CORBA. Os algoritmos para difusão confiável e atômica necessários foram desenvolvidos a partir dos modelos de objetos FT-CORBA e UMIOP. Estes algoritmos são apresentados usando conceitos teóricos fundamentais para expressar suas características básicas. Por fim, é apresentado uma comparação com experiências similares da literatura e outros protocolos para replicação ativa.*

1. Introdução

O padrão FT-CORBA surgiu de um esforço da OMG (*Object Management Group*) no sentido de especificar uma arquitetura para objetos distribuídos tolerantes a falhas [Object Management Group, 2002]. Este padrão introduz suporte à tolerância a faltas na arquitetura CORBA fazendo uso de técnicas de replicação de objetos. Este suporte é construído a partir de um conjunto de objetos de serviço usados no gerenciamento de objetos de aplicação replicados. A tolerância a faltas fornecida usando os serviços definidos pelo padrão está fundamentada em premissas de falhas de parada.

*Realizado com recursos do CNPq (projeto número 401802/2003-5).
†Bolsista PGI/CNPq.
‡Bolsista de Produtividade em Pesquisa do CNPq - Nível 2.
§Bolsista PIBIC/CNPq.

A especificação FT-CORBA define quatro tipos de replicação para os grupos de objetos: **Sem Estado** (estilo de replicação em que o estado dos objetos não é alterado pelas requisições); **Passiva Fria** (replicação com apenas um objeto ou réplica primária que grava periodicamente informações de estado em meio persistente); **Passiva Morna** (funciona de maneira semelhante ao anterior com a diferença de que os *checkpoints* da réplica primária são difundidos entre as réplicas secundárias); e, por fim, a **Replicação Ativa** (onde todas as réplicas executam as requisições do cliente).

O único tipo de replicação que é previsto e não tem seus mecanismos necessários especificados é a replicação ativa. A especificação atual do FT-CORBA permite que as abstrações definidas para a gestão de replicações possam ser também usadas com replicas ativas, desde que suportadas em suas necessidades de comunicação por ferramentas proprietárias. Assim, as requisições de método são difundidas, através de difusão atômica [Hadzilacos and Toueg, 1994, Défago et al., 2000] a todos os objetos membros do grupo (réplicas ativas) usando meios externos ao ORB. Neste caso, se considerarmos as necessidades de suporte de uma replicação ativa, do ponto de vista conceitual, diríamos que o FT-CORBA fornece os mecanismos para controle e gerência do grupo formado pelas réplicas ativas e a ferramenta proprietária fornece o suporte para comunicação de grupo.

A OMG publicou em 1999 um RFP (*Request For Proposal*), onde era definida uma série de requisitos para um serviço de difusão não confiável fundamentado nos serviços do *multicast* IP. Este processo culminou com o surgimento da especificação UMIOP (*Unreliable Multicast Inter-ORB Protocol*) [Object Management Group, 2001]. Na especificação UMIOP, o mapeamento das mensagens GIOP - protocolo genérico do CORBA independente do mecanismo de transporte - sobre a pilha UDP/*multicast* IP é definido pelo protocolo MIOP (*Multicast Inter-ORB Protocol*). A função básica do MIOP é segmentar e encapsular mensagens GIOP em um ou mais pacotes e difundí-los via UDP/*multicast* IP. Portanto, o padrão UMIOP define um mecanismo de comunicação de um para muitos sem garantias de entrega dentro da arquitetura CORBA.

Prover suportes de comunicação de grupo que oferecem garantias de entrega e de ordem de mensagens para aplicações distribuídas, envolve uma combinação de protocolos que lidam tanto com gerenciamento de grupo (*membership*, detecção de falhas, transferências de estado, etc.) como com a própria comunicação de grupo. Estas abstrações estão sendo padronizadas separadamente na OMG: o gerenciamento de grupo é tratado nas especificações FT-CORBA e a comunicação de grupo vem sendo desenvolvida em outro grupo de trabalho através do MIOP e seu modelo de objetos, definidos na especificação UMIOP. Estas experiências podem ser pensadas como blocos básicos na definição de suportes mais elaborados fornecendo serviços diferenciados (em termos da confiabilidade de entrega e de ordenação das mensagens), que seriam usados de acordo com a qualidade de serviço requerida.

Neste texto apresentamos a nossa experiência na construção de um suporte para replicação ativa baseado no modelo de objetos CORBA, fazendo uso da integração das especificações UMIOP e FT-CORBA em uma plataforma única de *middleware*. O resultado desta integração é um suporte de comunicação de grupo com primitivas básicas de difusão seletiva não confiável, onde adicionamos funcionalidades construindo primitivas que suportam qualidades de serviço mais elaboradas. Neste caso, a replicação ativa passa a ser atendida em suas necessidades de comunicação usando somente padrões definidos pela OMG, dentro do *middleware*. Como conseqüência, para suportar a replicação ativa, foi definido um protocolo de difusão atômica otimista [Pedone and Schiper, 1998] que atende as seguintes métricas: grau de latência 1, complexidade de mensagens $O(1)$ no caso ótimo e robustez máxima ($f \leq n - 1$). Este protocolo foi integrado ao FT-CORBA na forma de um *plugin* para o mecanismo SCG, que permite a integração de qualquer ferramenta proprietária de comunicação de grupo ao nosso sistema. Um artigo complementar a este [Bessani et al., 2004] descreve detalhadamente o SCG e outros mecanismos que extendem a arquitetura FT-CORBA formando a base de nossas implementações.

O texto está organizado da seguinte forma: na seção 2 temos uma breve revisão das principais especificações da OMG que utilizam a noção de grupos de objetos. A seção 3 apresenta o modelo de sistema e o suporte algorítmico para replicações que faz uso dos padrões da OMG. A seção 4 apresenta o SCG e sua arquitetura de *plugin* que permite a inclusão de serviços de comunicação de grupo em nossa implementação do FT-CORBA. Esta seção apresenta ainda uma descrição da implementação dos protocolos propostos e medidas de desempenho comparando-a a outros *plugins* para ferramentas proprietárias. Finalmente, a seção 5 relata alguns experimentos similares da literatura e a seção 6 apresenta as conclusões do trabalho.

2. Grupos nas Especificações da OMG: FT-CORBA e UMIOP

A arquitetura **FT-CORBA** [Object Management Group, 2002] define os serviços, em nível de *middleware*, responsáveis por prover as funcionalidades básicas para aplicações tolerantes a faltas através da replicação de objetos CORBA. Esses serviços estão divididos em três módulos básicos:

- **Gerenciamento de Replicação (SGR):** Este é o serviço responsável pelo ciclo de vida dos grupos. Duas funcionalidades são oferecidas por ele: gerenciamento de propriedades onde as características funcionais da replicação (tipo de replicação usada, número mínimo de réplicas, etc.) são definidas; e o gerenciamento de grupos que oferece mecanismos para a criação de membros (através de fábricas de objetos) e para o controle de *membership* dos grupos definidos;
- **Gerenciamento de Falhas (SGF):** É o serviço responsável pela detecção, notificação, análise e diagnóstico de falhas. Este serviço trabalha em conjunto com o SGR para que este último mantenha um *membership* sempre atualizado dos grupos;
- **Gerenciamento de Recuperação e Logging (SRL):** O SRL é responsável pela manutenção da consistência de estados das réplicas. Este serviço define mecanismos para a recuperação de réplicas e do próprio grupo. Entre estes mecanismos está um suporte para construção de *logs* usados no armazenamento de requisições enviadas ao grupo. Este serviço também fornece mecanismos para atualização de membros através de *checkpoints*. Além disso também é responsabilidade do SRL atuar na recuperação de réplicas faltosas através da atualização de seus estados.

Conforme já citado, a especificação FT-CORBA não define um suporte de comunicação de grupo, indispensável na replicação ativa: não existem interfaces e nem tão pouco um protocolo padronizado para tal. As especificações apenas sugerem que se utilize qualquer suporte proprietário para este fim.

Em relação a interoperabilidade com grupos onde objetos são mantidos por diferentes ORBs que implementam o FT-CORBA, a OMG criou um formato padronizado para a referência de grupo de objetos: IOGR (*Interoperable Object Group Reference*). A referência IOGR consiste basicamente em um conjunto de perfis IIOP (concretização do GIOP sobre TCP/IP) que identificam cada membro do grupo ou um *proxy* para acesso a estes.

A especificação **UMIOP** [Object Management Group, 2001] define um protocolo baseado em *multicast* IP e um modelo de objetos que permite a difusão de mensagens sem garantia de confiabilidade em grupos de objetos de aplicação. O modelo de objeto definido permite que vários objetos CORBA possam ser invocados simultaneamente através de uma única referência, contrastando com o modelo convencional do CORBA, onde uma referência de objeto corresponde a uma única implementação do mesmo.

Em termos de interoperabilidade, o padrão UMIOP também define uma referência de grupo que permite endereçar um grupo com zero ou mais objetos. Esta IOR (*Inter-Operable Reference*) de grupo utiliza um tipo de perfil diferente do perfil IIOP para difundir mensagens via UDP/*multicast* IP. Este perfil contém todas as informações necessárias para o acesso ao grupo em nível de transporte (endereço IP classe D e porta), além da identificação lógica do grupo, utilizada para acessar os objetos membros em nível de ORB.

A grande diferença entre a IOGR do FT-CORBA e a IOR de grupo do UMIOP são as informações contidas nas mesmas: enquanto a IOGR fornece a lista de membros do grupo (*membership*), a IOR de grupo contém apenas uma chave de grupo e um endereço *multicast* IP. Esta chave é resolvida pelos adaptadores de objetos (POA - *Portable Object Adapter*[1]), disponíveis nos *hosts* endereçados pelo *multicast* IP, para se chegar aos membros do grupo.

Os padrões FT-CORBA e UMIOP foram implementados pelo grupo de sistemas distribuídos do DAS/UFSC através dos projetos GROUPPAC [Lung et al., 2001] e MJACO [Bessani et al., 2002], respectivamente. Estas implementações são a base deste trabalho.

3. Protocolo para Replicação Ativa no CORBA

Conforme citado anteriormente, ferramentas de comunicação de grupo podem ser usadas para manter a consistência das réplicas quando o modelo de replicação ativa é usado no FT-CORBA. O surgimento do padrão UMIOP abre uma importante perspectiva para a construção de serviços mais elaborados, como os exigidos na replicação ativa.

Seguindo a abordagem proposta em [Hadzilacos and Toueg, 1994], construímos a difusão atômica sobre um protocolo de difusão confiável. Na seqüência descrevemos estes dois algoritmos, usados no suporte de comunicação de grupo constituído a partir da integração FT-CORBA/UMIOP no nosso sistema, que dão sustentação à técnica de replicação ativa.

3.1. Modelo de Sistema

Partindo da idéia de que o FT-CORBA preenche requisitos de gerenciamento de grupo e que o MIOP e o IIOP, disponíveis no mesmo ORB, compõem uma base concreta para a construção de diferentes protocolos de comunicação de grupo, temos que descrever o modelo de sistema adotado. Este modelo define as premissas dos algoritmos desenvolvidos como suporte para replicação ativa.

O sistema considerado é um sistema assíncrono *ad-hoc* [Défago et al., 2000] onde os processos falham apenas através de parada (acidental ou forçada) extendido com detectores de faltas perfeitos [Chandra and Toueg, 1996]. Neste sistema consideramos um conjunto $\Pi = \{p_0, p_1, ...\}$ de processos, onde são definidos grupos fechados[2] $G \subset \Pi$ que representam os serviços replicados.

Com a disponibilidade no mesmo ORB das pilhas MIOP/UDP/*multicast* IP e

[1] Componente do ORB que atua nos servidores e que é responsável por localizar e ativar objetos, e também por direcionar aos mesmos mensagens recebidas.

[2] Grupos fechados são aqueles em que somente membros do grupo podem difundir mensagens neste.

IIOP/TCP/IP, podemos assumir que os processos no sistema são interligados por um suporte de comunicação que provê dois serviços básicos:

- **Comunicação ponto a ponto confiável:** Definido pelas primitivas $send(p, m)$ e $receive(m)$, este serviço define canais de comunicação ponto a ponto confiáveis: se um processo correto p envia uma mensagem m a um outro processo q não faltoso então este último acabará por receber m;
- **Difusão seletiva não confiável:** Definido pelas primitivas $U\text{-}multicast(G, m)$ e $receive(m)$. A primeira primitiva difunde a mensagem m no grupo G, sem garantia nenhuma com relação ao recebimento de m pelos membros deste grupo. A primitiva $receive(m)$ é usada pelos membros de G para receber uma mensagem m enviada ao grupo. As únicas propriedades deste serviço são a não criação de mensagens espúrias e a garantia de que processos corretos pertencentes ao grupo acabam por receber alguma mensagem difundida no mesmo.

Estes serviços modelam diretamente os protocolos IIOP (sobre TCP/IP) e MIOP (sobre UDP/*multicast* IP), respectivamente, especificados para ORBs CORBA.

No modelo são assumidos detectores de falhas perfeitos formando, então, as bases para a construção de um serviço de *membership* de partição primária com propriedades fortes como as definidas em [Ricciardi and Birman, 1991]. A inclusão destes detectores e, por conseguinte, do *membership* de partição primária no nosso modelo é justificada pelo fato de que o próprio padrão FT-CORBA prevê serviço de detecção com a semântica de detectores perfeitos (classe $\mathcal{P}$). Detectores mais fracos (como o $\diamond\mathcal{W}$, por exemplo) são complexos demais para serem implementados em protocolos práticos, tendo inclusive sua utilidade questionada [Delporte-Gallet et al., 2002], e não atingem a robustes dos perfeitos que é máxima (suportam até $n - 1$ faltas).

3.2. Difusão Confiável

O serviço de **difusão confiável** é definido em termos de duas primitivas: $R\text{-}multicast(G, m)$ e $R\text{-}deliver(m)$. A primitiva $R\text{-}multicast(G, m)$ é usada para a difusão da mensagem m para todos os processos pertencentes ao grupo G enquanto a primitiva $R\text{-}deliver(m)$ é usada para liberar a mensagem m para a camada superior. Estas primitivas devem satisfazer as propriedades de validade[3], acordo[4] e integride[5] [Hadzilacos and Toueg, 1994] que caracterizam uma difusão confiável.

3.2.1. Algoritmo

O algoritmo de difusão confiável proposto é fundamentado no uso dos dois serviços de comunicação disponiveís no nosso modelo e, incorpora técnicas de recuperação (NACK) e confirmação (ACK) de mensagens de maneira semelhante a protocolos como o Trans [Melliar-Smith et al., 1990], o Psync [Peterson et al., 1989] e o usado pelo sistema Transis [Amir et al., 1992].

Protocolos que se baseiam em NACKs são chamados "controlados pelo receptor" [Levine and Garcia-Luna-Aceves, 1998], pois em geral é este o responsável por identificar perdas e pedir retransmissões. Apesar de escalável e eficiente em ambientes onde a

[3]**Validade:** Se um processo correto difundiu m em G, então algum processo correto pertencente à G entregará m ou nenhum processo do grupo está correto.

[4]**Acordo:** Se um processo correto pertencente a G entrega a mensagem m, então todos os processo corretos pertencentes a G entregarão m.

[5]**Integridade:** Para qualquer mensagem m, cada processo correto pertencente a G entrega m no máximo uma vez e apenas se m foi previamente difundida em G.

taxa de perda de mensagens é baixa, esta classe de protocolo sofre do problema do *buffer* infinito: o acordo só pode ser provado se considerarmos que os processos armazenam todas as mensagens recebidas para poderem responder a eventuais NACKs. O uso de ACKs como confirmações de recebimento de mensagens, apesar de limitar a escalabilidade do protocolo (devido a necessidade de um serviço de *membership*), evita o problema do *buffer* infinito, já que os processos podem saber quais mensagens foram recebidas por todos os membros do grupo, tornando possível a retirada das mesmas de seus *buffers*. A necessidade do serviço de *membership* não é um problema quando consideramos uma implementação do padrão FT-CORBA. Outro problema decorrente do uso de ACKs no protocolo é a elevada quantidade de confirmações que será enviada: $\#G$ ACKs para cada mensagem. Este problema é minimizado no protocolo proposto fazendo com que os processos enviem ACKs a cada N mensagens recebidas ou a cada T unidades de tempo. N e T são parâmetros do protocolo ajustados de acordo com o ambiente de execução.

Algoritmo 1 Difusão Confiável (processo p)

1: {Inicialização}
2: $buffer \leftarrow \emptyset$ {*Buffer* com as mensagens recebidas.}
3: $ack\_i \leftarrow \emptyset$ {Conjunto de ACKs recebidos para a mensagem com identificador i.}
4: $received \leftarrow 0$ {Quantidade de mensagens recebidas.}
5: $last\_ack \leftarrow getTimestamp()$ {Instante de tempo em que foi enviado o último ACK.}
Require: $R\text{-}multicast(G, m)$ {Difusão da mensagem no grupo.}
6: $U\text{-}multicast(G, m)$
Require: $receive(m)$ {Recebimento de uma mensagem.}
7: **if** $m.id > maxStable(buffer) \wedge m \notin buffer$ **then** {Mensagem não recebida.}
8: $R\text{-}deliver(m)$
9: $buffer \leftarrow buffer \cup \{m\}$
10: $received \leftarrow received + 1$
11: **end if**
Require: $receive(< ACK, q, stable >)$ {Recebimento de um ACK.}
12: **for all** $m \in buffer : m.id \leq stable$ **do**
13: $ack_{m.id} \leftarrow ack_{m.id} \cup \{q\}$
14: **end for**
15: $buffer \leftarrow buffer \setminus \{m \in buffer | G \subseteq ack_{m.id}\}$
16: **repeat** {Verificação e envio de NACKs.}
17: $missing \leftarrow \{i | i \leq stable \wedge \nexists m \in buffer : m.id = i\}$
18: $send(q, < NACK, p, missing >)$
19: $q \leftarrow select(G)$
20: **until** $\forall i \in missing : \exists m \in buffer : i = m.id$
Require: $receive(< NACK, q, missing >)$ {Recebimento da NACKs}
21: **for all** $m \in buffer : \exists i \in missing : m.id = i$ **do**
22: $send(q, m)$
23: **end for**
Require: $viewChanged(V^i)$ {Mudança no *membership*.}
24: $G \leftarrow V^i$
25: $buffer \leftarrow buffer \setminus \{m \in buffer | G \subseteq ack_{m.id}\}$
26: $R\text{-}deliver(V^i)$
Require: $received \bmod N = 0 \vee last\_ack - getTimestamp() \geq T$ {Gatilho para envio de ACK.}
27: $stable \leftarrow maxStable(buffer)$
28: $U\text{-}multicast(G, < ACK, p, stable >)$
29: $last\_ack \leftarrow getTimestamp()$

O algoritmo 1 apresenta o protocolo para difusão confiável[6] Neste algoritmo, para a difusão de mensagens, o emissor utiliza diretamente o serviço de difusão não confiável (linha 6). Os membros receptores só aceitam mensagens não recebidas (linha 7) que são entregues a aplicação e colocadas em um *buffer* de recepção (linhas 8 e 9). As linhas 27-29 descrevem os passos para o envio de ACKs. Basicamente, um ACK contém o identificador da maior mensagem estável do *buffer*, isto é, a mensagem de maior número de sequência tal que todas as anteriores foram recebidas, obtido através da função $maxStable(buffer)$. Quando um receptor recebe o ACK, ele acrescenta o emissor deste em todas os ack_i onde $i \leq stable$ (linas 12-14) e depois remove do *buffer* as mensagens que foram confirmadas por todos (linha 15).

[6] O algoritmo considera apenas um emissor por questões de legibilidade.

A verificação de mensagens perdidas é feita sempre que um ACK é recebido: na linha 17 são calculadas as mensagens que faltam, na linha 18 é enviado um NACK pedindo a reposição das mesmas. Estes NACKs são enviados sempre através do serviço de comunicação ponto a ponto confiável a um dos membros do grupo. Inicialmente, o pedido é feito ao processo emissor do ACK, em caso de falha deste, outro processo é escolhido aleatóriamente através da função $select(G)$. As linhas 21-23 representam o comportamento de um membro ao receber um NACK: ele envia todas as mensagens solicitadas presentes em seu *buffer*. Para que o *buffer* não cresça indefinidamente devido a falhas de processos (ACK de processo falhos não chegam e mensagens ficam no *buffer* sem perspectiva de retirada), o algoritmo recorre ao serviço de *membership*, que distribui visões atualizadas do grupo sempre que este se altera em sua composição. Um processo recebe notificações de alterações de *membership* através de uma chamada $viewChanged(V^i)$, quando então a visão local do grupo é atualizada e todas as mensagens que esperam o recebimento de confirmações para serem apagadas são verificadas novamente segundo esta nova visão (linha 25).

Corretude do algoritmo (Rascunho): Esta prova é obtida a partir da prova das três propriedades da difusão confiável.

Prova de Integridade: A não duplicação é garantida pelo algoritmo que só entrega mensagens não recebidas (linha 7). A não criação de mensagens é consequência direta das propriedades do serviço de difusão não confiável.

Prova de Validade: Esta propriedade é garantida pelas características de grupos fechados e pela premissa de que, se um processo difunde uma mensagem em um grupo, pelo menos ele vai recebê-la.

Prova de Acordo: Se um processo correto p executa $R\text{-}deliver(m)$, então $m \in buffer_p$ (*buffer* do processo p). Por construção, m não será removida deste *buffer* enquanto p não receber ACKs de todos os membros do grupo para essa mensagem ($G \subseteq ack_{m.id}$). Suponha que um processo correto q não receba m. Como q é correto ele terminará por receber um ACK com $stable \geq m.id$. Desta forma, q acabará por descobrir que perdeu m e enviará um NACK, pedindo sua retransmissão, através do serviço de comunicação ponto a ponto confiável, para algum outro processo correto que deu evidências de ter recebido m. Um processo acabará por repor m para q, visto que esta mensagem não é removida dos *buffers* dos processos que a receberam enquanto estes não receberem ACKs para ela de todos os membros do grupo (e q não enviou tal ACK). Este mesmo raciocínio pode ser estendido a todos os membros de G, demonstrando que todos os processos corretos acabam por receber (e entregar) todas as mensagens que foram liberadas por algum processo correto de G.

3.3. Difusão Atômica

A **difusão atômica** é uma primitiva de comunicação de grupo mais elaborada que garante que todos os processos corretos membros de um grupo entregarão a mesma seqüência de mensagens difundidas neste. Esta primitiva é a base para replicação ativa, pois garante as propriedades fundamentais para o determinismo de réplicas [Schneider, 1990]. Formalmente, a difusão atômica é definida através das primitivas, $A\text{-}multicast(G, m)$ e $A\text{-}deliver(m)$, de maneira análoga a difusão confiável, que satisfazem todas as propriedades da difusão confiável e a propriedade de ordenação total local[7] [Hadzilacos and Toueg, 1994]. A comunicação de grupo construída pela adição desta propriedade de ordem à difusão confiável é identificada como "difusão atômica local". Este

[7]**Ordenação Total Local:** Se dois processos corretos p e q entregam as mensagens m e m' endereçadas ao grupo G, então p entrega m antes de m' se e somente se q entregar m antes de m'.

serviço é suficiente para a replicação ativa do FT-CORBA, visto que, pela especificação, cada objeto pertence a um e somente um grupo. Este é o sentido da ordenação total local, que diferente da ordenação total global, é definida sobre grupos que não se sobrepõem.

3.3.1. Algoritmo

O algoritmo de difusão atômica é baseado no paradigma do ordenador fixo [Défago et al., 2000]: os processos recebem as mensagens difundidas no grupo através de um serviço de difusão confiável e cabe a um deles, o ordenador, impor a ordem de entrega destas mensagens.

Algoritmo 2 Difusão Atômica (processo p)

1: {Inicialização}
2: $unordered \leftarrow \emptyset$ {Conjunto das mensagens recebidas e não ordenadas.}
3: $undelivered \leftarrow \emptyset$ {Conjunto das mensagens ordenadas e não entregues.}
4: $next\_deliver \leftarrow 1$ {Ordem da próxima mensagem a ser liberada.}
5: $next\_order \leftarrow 1$ {Próxima ordem a ser enviada (usada pelo ordenador).}
Require: $A\text{-}multicast(G, m)$ {Quando da chamada de uma difusão atômica.}
6: $R\text{-}multicast(G, m)$
Require: $R\text{-}deliver(m)$ {Quando do recebimento de uma mensagem.}
7: $unordered \leftarrow unordered \cup \{m\}$
8: **if** $getSorter(G) = p$ **then** {Eu sou o ordenador.}
9:     $R\text{-}multicast(G, < m.id, next\_order >)$ {Envia a ordem para esta mensagem.}
10:     $next\_order \leftarrow next\_order + 1$
11: **end if**
Require: $R\text{-}deliver(< id, ord >)$ {Quando do recebimento da ordem de uma mensagem.}
12: $undelivered \leftarrow undelivered \cup \{< id, ord >\}$
13: **for all** $m \in unordered, < id, ord > \in undelivered : m.id = id$ **do**
14:     **if** $ord = next\_deliver$ **then**
15:         $A\text{-}deliver(m)$
16:         $next\_deliver \leftarrow next\_deliver + 1$
17:         $unordered \leftarrow unordered \setminus \{m\}$ {Apaga mensagem já liberada.}
18:         $undelivered \leftarrow undelivered \setminus \{< id, ord >\}$ {Apaga a ordem já usada.}
19:     **end if**
20: **end for**
Require: $viewChanged(V^i)$ {Quando do recebimento de uma nova visão do grupo.}
21: $G \leftarrow V^i$
22: $send(getSorter(G), < next\_deliver >)$
23: **if** $getSorter(G) = p$ **then** {Sou o ordenador.}
24:     **wait until** $receive(< q, ord_q >), \forall q \in G$ {Espera até receber os $next\_deliver$ dos membros.}
25:     $next\_order \leftarrow max(\{ord_q | q \in G\})$
26:     **wait until** $next\_order = next\_deliver$ {Espera até receber as ordens já definidas.}
27:     $pending\_messages \leftarrow unordered$ {Pega as mensagens pendentes...}
28:     **for all** $m \in pending\_messages$ **do**
29:         $R\text{-}multicast(G, < m.id, next\_order >)$
30:         $next\_order \leftarrow next\_order + 1;$
31:     **end for**
32: **end if**

O protocolo de difusão atômica proposto é apresentado no algoritmo 2. No emissor, a difusão atômica é implementada usando difusão confiável (linha 6). Quando um membro do grupo recebe uma mensagem m ela é colocada no *buffer* $unordered$ (linha 7). Se o receptor é o ordenador, ele define uma ordem para m e difunde uma mensagem relacionando esta ordem (representada pelo contador $next\_order$) a $m.id$ (linhas 8-11). A mensagem m é efetivamente liberada para a aplicação quando um processo do grupo recebe uma mensagem do tipo $< id, ord >$ relacionando $m.id$ a uma ordem (linhas 12-20). Note que o recebimento de uma ordem atrasada (devido ao assincronismo do sistema) pode acarretar a liberação de um conjunto de mensagens que estavam bloqueadas (laço **for all** nas linhas 13-20).

A parte mais complexa do algoritmo 2 consiste na troca da visão de grupo (linhas 21-32). Devido ao caráter assíncrono do ambiente considerado, cada um dos membros do grupo pode ter um valor diferente para $next\_deliver$, assim a primeira coisa que cada membro do grupo faz é enviar o seu valor de $next\_deliver$ ao ordenador do grupo, escolhido através da função $getSorter(G)$ (linha 22). O ordenador, define seu valor para $next\_order$ como o valor máximo entre todos os recebidos, e então espera entregar as mensagens com identificador menor a $next\_deliver$ (que vão ser recebidas devido as propriedades da difusão confiável - linha 26). Depois disso, o ordenador transfere as mensagens pendentes para o *buffer* $pending\_messages$ e as ordena, difundindo esta ordem no grupo (linhas 27-31).

Corretude do algoritmo (Rascunho): As propriedades de integridade, acordo e validade podem ser obtidas diretamente do algoritmo de difusão confiável. Resta então provar a ordenação total local de mensagens.

Prova de Ordenação total local: Esta prova é construída por contradição. Suponha que existam dois processos corretos p e q que entregam a mensagem m com ordens diferentes ($ord_p \neq ord_q$). Quando não existe falha no ordenador esta situação é claramente impossível, visto que cada mensagem tem um id único e o ordenador, pela própria construção do algoritmo, só envia a ordem de uma mensagem uma única vez através de difusão confiável a todos os membros de G, assim $ord_p = ord_q$. Considere agora o caso em que o ordenador falha enquanto uma mensagem m esta sendo engajada no grupo. Temos então uma das duas situações a seguir:

- O ordenador falhou **depois** de enviar a ordem de m: Se algum processo correto recebeu esta ordem para m, pela propriedade de acordo da difusão confiável, todos os processos corretos acabam por recebe-la também. Assim, nesse caso, as ordens de entrega dos processos devem ser iguais ($ord_p = ord_q$). Se nenhum processo recebeu a ordem de m enviada pelo ordenador então caímos no caso abaixo.
- O ordenador falhou **antes** de enviar a ordem de m: Então nenhum processo tem a ordem para a mensagem difundida no grupo; o novo ordenador então determinará esta ordem (linhas 27-31).

3.4. Considerações e Otimizações sobre os Algoritmos

Muitas das características dos algoritmos apresentados foram determinadas pelos conceitos e mecanismos definidos pelos padrões usados. Mas mesmo assim, algumas otimizações podem ser implementadas para melhorar o desempenho dos protocolos em condições favoráveis, caracterizando ainda mais o caráter otimista dos mesmos.

Podemos, por exemplo, utilizar mecanismos de *piggybacking* para o envio de ACKs concatenadas com as mensagens de aplicação difundidas através do protocolo de difusão confiável, quando possível. Esta técnica, muito usada em outros protocolos [Melliar-Smith et al., 1990, Amir et al., 1992], diminui o *overhead* provocado por mensagens de controle, aumentando o desempenho do protocolo em termos de mensagens de aplicação por mensagens de controle.

Na difusão atômica, podemos diminuir o número de mensagens difundidas no grupo (numa relação de 2 para 1) quando o emissor da mensagem é o ordenador, bastando apenas que o emissor/ordenador envie a mensagem já com sua ordem definida, permitindo que os processos receptores a entreguem imediatamente.

Implementando estas modificações temos um protocolo de difusão atômica com complexidade de mensagens de $O(1)$ (uma única difusão não confiável) nos casos que o ordenador é o emissor (a maioria, ver próxima seção) e que não existem perdas de mensagens. O grau de latência (uma métrica baseada na idéia de relógios lógicos usada em [Pedone and Schiper, 1998]) é igual a 1 para o nosso protocolo.

4. Integração de Ferramentas de Comunicação de Grupo ao FT-CORBA

O modelo de replicação ativa, proposto para o FT-CORBA, baseia-se na idéia de um grupo fechado de réplicas acessado por clientes leves através do mecanismo de comunicação usual do CORBA (chamada de métodos remotos sobre IIOP). A especificação FT-CORBA também prevê a utilização de ferramentas de comunicação de grupo proprietárias para difusão de mensagens com diferentes níveis de qualidade de serviço [Object Management Group, 2002]. Entretanto, estas mesmas especificações não definem como tais ferramentas devem ser integradas na arquitetura FT-CORBA.

Diante disto foi criado o ***framework*** **SCG** (Suporte de Comunicação de Grupo). O SCG é parte integrante do GROUPPAC e define um mecanismo através do qual a infraestrutura FT-CORBA interage com a ferramenta de comunicação de grupo. Através deste mecanismo é possível a integração de qualquer suporte de comunicação de grupo a arquitetura do FT-CORBA através de *plugins*. A figura 1 apresenta este modelo.

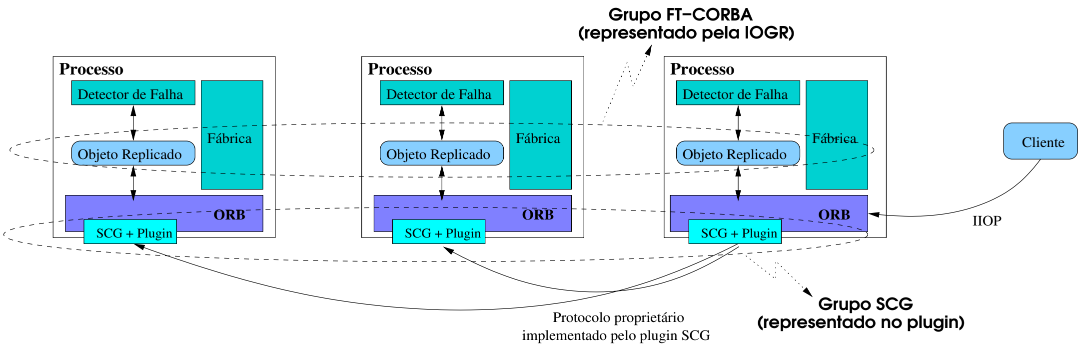

Figura 1: Replicação Ativa no FT-CORBA.

Nesta figura, o cliente de posse da IOGR do grupo de réplicas envia uma requisição ponto a ponto (usando IIOP) a uma das réplicas do serviço, selecionada através da função $getSorter(G)$ (por isso, em geral, o emissor é o ordenador). Esta réplica, serve de "ponte" para acesso aos protocolos de comunicação de grupo. As requisições são interceptadas na réplica e passadas para o SCG, que difunde as mesmas fazendo uso da ferramenta de comunicação de grupo encapsulada no *plugin* carregado. Cada membro do grupo que recebe a requisição, através do SCG, a executa. As respostas, se houverem, são enviadas de volta, através do SCG/*plugin*, à réplica "ponte" que repassa o resultado ao cliente, usando IIOP. Portanto, é através destes elementos "ponte" e de *plugins* que o grupo fechado definido pelo FT-CORBA fica acessível a clientes IIOP.

A descrição completa do SCG e de outras estruturas usadas na implementação de replicação ativa no GROUPPAC pode ser encontrada em [Bessani et al., 2004].

4.1. Plug-in MJACO: Integrando as especificações FT-CORBA e UMIOP

Os protocolos apresentados neste texto foram implementados como um *plugin* SCG no GROUPPAC. Esta implementação foi baseada no ORB MJACO [Bessani et al., 2002]. A idéia básica para a construção deste *plugin* foi refletir o grupo FT-CORBA, representado pela IOGR, em um grupo UMIOP, usado no contexto do SCG, para então fazer uso dos serviços de difusão não confiável (MIOP) e comunicação ponto a ponto confiável (IIOP).

A implementação foi feita a partir de duas interfaces definidas em IDL: `UMIOPAdaptorReceiver` e `ReceiverResponses`. A primeira é implementada por todas as instâncias do *plugin*, que são registradas como membros do grupo UMIOP que reflete o grupo de réplicas. Esta interface fornece métodos de recebimento de mensagens como `receiveRequest()`, de confiabilidade `ack()` e `nack()` e de ordenação como `setGlobalOrder()`[8]. A interface `ReceiverResponses` tem sua implementação ativada no *plugin* do processo "ponte" visando vencer uma limitação do modelo de objetos UMIOP: apenas métodos sem resposta são chamados via MIOP. Desta forma, apesar das requisições serem difundidas no grupo através do método `receiveRequest()` da interface `UMIOPAdaptorReceiver`, as respostas são enviadas de volta ao *plugin* "ponte" via IIOP através da chamada do método *receiveRequest()* da interface `ReceiverResponses`. A referência para o objeto receptor (que implementa esta interface) é passada como um parâmetro em `receiveRequest()`, implementando assim um mecanismo de *callback*.

4.2. Desempenho e Escalabilidade da Replicação Ativa

A fim de validar a implementação do nosso protocolo de difusão atômica foram realizados alguns experimentos com o *plugin* SCG implementado comparando-o com outros dois *plugins* (já implementados) para ferramentas proprietárias: ENSEMBLE [van Renesse et al., 1998] e JGROUPS [JGroups, 2004]. A aplicação exemplo utilizada foi um serviço de armazenamento remoto simples que provê uma operação para o armazenamento de um bloco de dados retornando um resumo deste (*hash*). O estilo de replicação empregado nos testes foi a ativa com votação, onde a resposta retornada ao cliente é o valor majoritário escolhido a partir do conjunto de respostas retornadas por todas as réplicas. As ferramentas de comunicação de grupo foram configuradas de maneira equivalente para prover um serviço de difusão atômica para o GROUPPAC. O cenário dos testes foi uma rede local onde não ocorreram falhas nos objetos. A figura 2 apresenta o resultado destes experimentos.

Dois tipos de experimentos foram realizados medindo o tempo entre o envio da requisição ao serviço replicado e a recepção da resposta pelo cliente. A figura 2 apresenta o resultado destes experimentos.

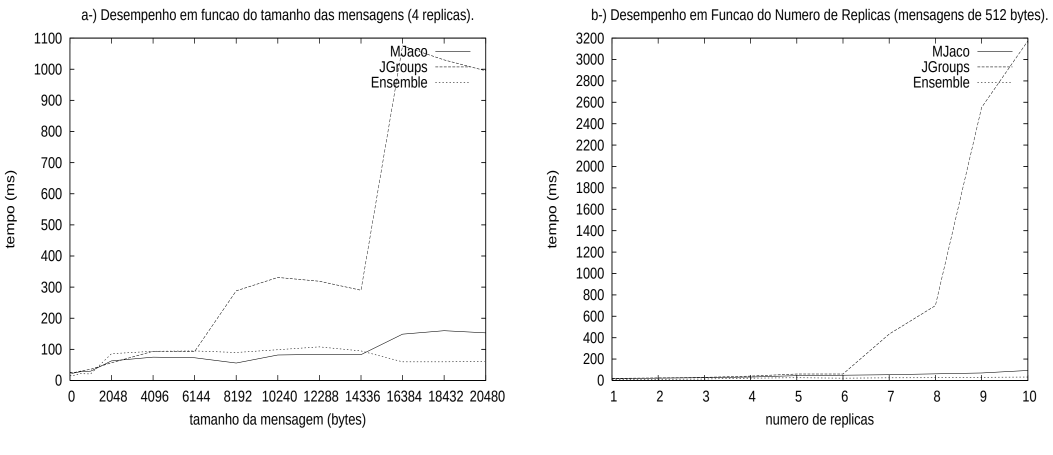

Figura 2: Desempenho do mecanismo de replicação ativa.

Na figura 2-a temos o tempo de resposta do serviço replicado considerando diferentes tamanhos de mensagens. Já na figura 2-b temos este tempo considerando um número variável de réplicas. Em ambos os casos temos resultados equivalentes em média entre a solução que implementa nossos protocolos e a baseada na ferramenta ENSEMBLE. A ferramenta de comunicação JGROUPS obteve resultados bastante inferiores, principalmente em termos de escalabilidade.

Os resultados apresentados na figura 2 são bastante positivos, principalmente se levarmos em consideração que nossa implementação é um primeiro protótipo enquanto o

[8]Note que estes métodos mapeiam diretamente os tipos de mensagens usadas pelos algoritmos 1 e 2.

ENSEMBLE é uma ferramenta consagrada e bastante estável. Outro ponto a ser ressaltado é que o *plugin* implementado utiliza apenas mecanismos de comunicação do ORB, que apesar de garantir portabilidade, impoêm um *overhead* natural do *middleware*.

5. Trabalhos Relacionados

Existem vários trabalhos que apresentam soluções para replicação ativa no CORBA, entretanto apenas dois destes sistemas seguem as especificações FT-CORBA. O sistema Eternal [Moser, L. E. et al, 1999] utiliza interceptação em nível de sistema operacional para integrar a ferramenta de comunicação de grupo Totem [Moser et al., 1996]. Esta abordagem, além de não ser interoperável prende a infraestrutura a apenas uma ferramenta. O projeto IRL (*Interoperable Replication Logic*) [Baldoni et al., 2002] tem por finalidade implementar uma infraestrutura FT-CORBA completamente portável que possa ser integrada a ORBs já existentes. A abordagem do IRL para replicação ativa consiste em dividir o sistema em três camadas instaladas em ambientes diferenciados em termos de sincronismo: cliente (sistema assíncrono), camada intermediária (parcialmente síncrono) e réplicas (assíncrono). Neste contexto, o IRL implementa *gateways* replicados na camada intermediária, onde os protocolos de acordo podem ser implementados, pois não estão sujeitos a impossibilidade FLP [Fischer et al., 1985]. Estes *gateways* recebem chamadas dos clientes via IIOP e as reenviam (também via IIOP) as réplicas do objeto invocado. Além desta abordagem ter um custo maior em termos de desempenho, pois existe um passo a mais de comunicação (duas chamadas IIOP e uma execução do protocolo de acordo para a atualização das réplicas do *gateways* na camada intermediária), ela sofre do problema de concentração dos *gateways* em redes especiais. Segundo os próprios autores do trabalho, devido as propriedades de sincronismo, este mecanismo só pode ser implementada em redes locais ou controladas, que obviamente não podem estar espalhadas por grandes áreas.

O protocolo de difusão confiável que mais se assemelha ao apresentado no algoritmo 1 é o Trans [Melliar-Smith et al., 1990] e sua versão melhorada, usada no sistema Transis [Amir et al., 1992]. Estes protocolos, assim como o Psync [Peterson et al., 1989], utilizam confirmações positivas e negativas (ACKs e NACKs) para garantir propriedades da difusão confiável, fazendo uso de *piggybacking* para diminuir o *overhead* de mensagens de controle. Entretanto, a preocupação maior nestes protocolos é com as relações de causa/efeito entre mensagens, não necessárias na implementação de replicação ativa. Um ponto que pode determinar limitações nos mesmos é a utilização apenas de serviços de difusão não confiáveis. Neste caso, as recuperações ficam sujeitas as taxas de perda de mensagens de seus ambientes, o que em determinadas situações pode atrasar bastante o recebimento das mensagens perdidas. No algoritmo proposto neste texto isto não acontece, uma vez que o mesmo usa canais ponto a ponto confiáveis para a recuperação de mensagens perdidas.

A literatura sobre protocolos de difusão atômica é extremamente vasta (veja [Défago et al., 2000] para um *survey* bastante completo da área), entretanto alguns sistemas influenciaram de maneira crucial nosso trabalho. O protocolo apresentado em [Pedone and Schiper, 1998] explora a alta probabilidade de ordem total intrínsica aos mecanismos de difusão utilizados na prática e desenvolve um algoritmo otimista que é rápido em casos onde esta ordenação total espontânea se verifica e relativamente lento nos casos em que ela não é válida. Nossos algoritmos seguem essa filosofia, agindo de maneira rápida (difusão atômica em apenas uma difusão não confiável) nos casos onde o ordenador é a "ponte" (emissor no grupo fechado de réplicas) e a rede não perde mensagens, e recorrendo a canais ponto a ponto confiáveis para NACKs e reparos quando o sistema perde mensagens. O algoritmo desenvolvido em [Ezhilchelvan et al., 2003] também baseia-se no paradigma do ordenador fixo para garantir ordem total das mensagens. Entretanto ele se utiliza de detectores de falha $\diamond\mathcal{W}$ e portanto requer pelo menos $n/2$ processos corretos para sua execução. A grande diferença desse algoritmo para o nosso está na forma como o sistema se recupera de falhas no ordenador: nós utilizamos o serviço de *membership* disponível no modelo enquanto eles utilizam um algoritmo de consenso para definir se o líder é faltoso ou não.

6. Conclusão

Este trabalho apresenta nossos esforços na construção de um suporte para replicação ativa no modelo de objetos CORBA. O objetivo traçado neste caso era a integração das especificações UMIOP e FT-CORBA em uma plataforma única de *middleware*. Como conseqüência, para suportar a replicação ativa, foi definido um protocolo de difusão atômica com as seguintes características: grau de latência 1, complexidade de mensagens $O(1)$ e robustez máxima ($f \leq n-1$). Este protocolo faz uso das pilhas MIOP/UDP/*multicast* IP e IIOP/TCP/IP definidas para ORBs CORBA. Esta base algoritmíca é integrada à arquitetura FT-CORBA na forma de um *plugin* para o *framework* SCG. Experimentos realizados demonstram um bom desempenho e escalabilidade deste *plugin*.

O resultado prático desta integração é um suporte de comunicação de grupo com primitivas básicas de difusão seletiva (ainda sem garantia e sem ordenação), que é extendido para prover qualidades de serviço mais elaboradas em termos de garantias e ordem. Neste caso, a replicação ativa passa também a ser atendida em suas necessidades de comunicação usando somente padrões definidos pela OMG, dentro do próprio *middleware*.

Nossos trabalhos futuros deverão considerar a evolução das implementações dos protocolos apresentados. Novas versões estão sendo projetadas com a finalidade de torná-los ainda mais eficientes. Estão nos nossos planos também a melhoria do serviço de *membership* e dos detectores de faltas do GROUPPAC, considerando trabalhos recentes e técnicas adaptativas.

Referências

Amir, Y., Dolev, D., Kramer, S., and Malki, D. (1992). Transis: A communication subsystem for high availability. In *FTCS-22: 22nd International Symposium on Fault Tolerant Computing*, pages 76–84, Boston, Massachusetts. IEEE Computer Society Press.

Baldoni, R., Marchetti, C., and Termini, A. (2002). Active software replication through a three-tier approach. In *Prooceedings of the 21st Symposium on Reliable Distributed Systems (SRDS'02)*, pages 109–118, Osaka, Japão. IEEE.

Bessani, A. N., Alchieri, E. A., Lung, L. C., and da Silva Fraga, J. (2004). GroupPac 3: Extendendo o FT-CORBA para gerenciamento e replicação ativa. Submetido ao WTF 2004 (V Workshop de Testes e Tolerância a Falhas).

Bessani, A. N., da Silva Fraga, J., and Lung, L. C. (2002). MJaco: Integração do multicast IP na arquitetura CORBA. In *Anais do XX Simpósio Brasileiro de Redes de Computadores - SBRC'2002*, Buzios, RJ.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2).

Delporte-Gallet, C., Fauconnier, H., and Guerraoui, R. (2002). A realistic look at failure detectors. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)*, Washington - D.C. - USA.

Défago, X., Schiper, A., and Urban, P. (2000). Totally ordered broadcast and multicast algorithms: a comprehensive survey. Technical Report TR DSC/2000/036, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland.

Ezhilchelvan, P., Palmer, D., and Raynal, M. (2003). An optimal atomic broadcast protocol and an implementation framework. In *Proceedings of the 8th IEEE Int. Workshop on Object-Oriented Real-Time Dependable Systems*, Guadalajara - México.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382.

Hadzilacos, V. and Toueg, S. (1994). A modular approach to the specification and implementation of fault-tolerant broadcasts. Technical report, Department of Computer Science, Cornell University, New York - USA.

JGroups (2004). JGroups: A toolkit for reliable multicast communication. Disponível em http://www.jgroups.org.

Levine, B. N. and Garcia-Luna-Aceves, J. J. (1998). A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348.

Lung, L. C., da Silva Fraga, J., Padilha, R., and Souza, L. (2001). Adaptando as especificações FT-CORBA para redes de larga escala. In *Anais do XIX Simpósio Brasileiro de Redes de Computadores - SBRC'2001*, Florianópolis, SC.

Melliar-Smith, P., Moser, L., and Agrawala, V. (1990). Broadcast protocols for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(1).

Moser, L. E., Melliar-Smith, P. M., Agarwal, D. A., Budhia, R. K., and Lingley-Papadopoulos, C. A. (1996). Totem: A fault-tolerant multicast group comunication system. *Communications of the ACM*, 39(4):54–63.

Moser, L. E. et al (1999). The eternal system: an architecture for enterprise applications. In *Proceedings of the 3rd International Enterprise Distributed Object Computing Conference (EDOC'99)*, Mannhein - Alemanha.

Object Management Group (2001). Unreliable multicast inter-orb protocol specification v1.0. OMG Standart ptc/03-01-11.

Object Management Group (2002). The common object request broker architecture: Core specification v3.0. OMG Standart formal/02-12-06.

Pedone, F. and Schiper, A. (1998). Optimistic atomic broadcast. In *Proceedings of the 12th International Symposium on Distributed Computing (DISC'98)*.

Peterson, L. L., Buchholz, N. C., and Schlichting, R. D. (1989). Preserving and using context information in interprocess communication. *ACM Transactions Computing Systems*, 7(3):217–246.

Ricciardi, A. M. and Birman, K. (1991). Using process groups to implement failure detection in asynchronous environments. In *ACM Symposium on Principles of Distributed Computing*, pages 341–353, Montreal - Quebec - Canada.

Schneider, F. B. (1990). Implementing fault-tolerant service using the state machine aproach: A tutorial. *ACM Computing Surveys*, 22(4):299–319.

van Renesse, R., Birman, K. P., Hayden, M., Vaysburd, A., and Karr, D. (1998). Building adaptative systems using ensemble. *Software - Prat. and Exp.*, 28(9):963–979.