Canais de Mensagens Persistentes para Sistemas Distribuídos Tolerantesa Faltas*

Jorge Morgado, Luís Rodrigues
Dept. de Informática da Faculdade de Ciências
da Universidade de Lisboa
Edifício C5 - Piso 1, Campo Grande
1700 Lisboa
Tel.: +351-1-8440100
jmorgado@hlcmm.pt, ler@di.fc.ul.pt

27 de Outubro de 1999

Resumo

Os canais de mensagens constituem uma abstracção que permite suportar a coordenação entre aplicações distribuídas sem obrigar à associação explícita entre os intervenientes. Os canais de mensagens podem ser voláteis, caso em que as mensagens publicadas são entregues apenas aos assinantes activos, ou persistentes, caso em que o canal armazena as mensagens para que possam ser lidas posteriormente à sua publicação. Este artigo propõe uma arquitectura que permite acrescentar características de persistência a canais de mensagens voláteis tirando partido de mecanismos de comunicação em grupo fiável.

*Partes deste relatório foram publicadas na 2ł Conferência sobre REDES de COMPUTADORES - Tecnologias e Aplicações, Évora, Portugal, Outubro 1999

1 Introdução

A vulgarização da utilização de aplicações distribuídas tem motivado o desenvolvimento de vários modelos de comunicação entre processos. Estes modelos pretendem aumentar a interoperabilidade, a portabilidade e a flexibilidade das aplicações de forma a facilitar a sua configuração, nomeadamente no que se refere à colocação dos componentes nas múltiplas plataformas que suportam a aplicação. Para além disso, os modelos de comunicação tentam esconder do programador as especificidades das redes de computadores e dos sistemas de codificação, permitindo que estes se concentrem no modelo do negócio em que a aplicação se insere.

Dos vários modelos existentes, dois emergem como particularmente úteis: o modelo de chamadas a procedimentos remotos e o modelo editor-assinante. O primeiro modelo, particularmente adaptado a aplicações com uma arquitectura cliente-servidor, pretende fornecer transparência à distribuição ao emular as chamadas a procedimentos locais. Este modelo requer uma associação explícita do cliente ao servidor, tornando algo complexa a reconfiguração da aplicação. Pelo contrário, no modelo editor-assinante, a associação entre os componentes é estabelecida em tempo de execução por um "canal de mensagens", uma abstracção que permite aos participantes trocarem informação de modo anónimo. O modelo editor-assinante facilita a reconfiguração da aplicação, uma vez que é possível acrescentar em qualquer momento novos produtores ou consumidores de informação ao canal. Os canais de mensagens podem distinguir-se em duas grandes categorias: os canais de mensagens voláteis e os canais de mensagens persistentes.

Nos canais de mensagens voláteis, as mensagens produzidas pelos editores são entregues aos assinantes que nesse momento estiverem associados ao canal após o que são descartadas. A vantagem deste tipo de canal é a sua eficiência. Em redes que permitam a difusão da informação de modo eficiente, por exemplo redes locais ou redes IP com suporte para difusão, o canal pode ser suportado directamente por um protocolo de transporte. Tipicamente, diferentes qualidades de serviço podem ser fornecidas (em termos de fiabilidade e pontualidade) consoante o tipo de protocolo de transporte utilizado.

Nos canais de mensagens persistentes, todas as mensagens enviadas são armazenadas em memória estável (tipicamente em disco) de modo a poderem ser mais tarde recuperadas por assinantes que não estejam li-

gados ao canal no momento da publicação. Este tipo de canal permite a comunicação diferida entre editores e assinantes, que deste modo não necessitam de estar simultaneamente activos para coordenarem as suas actividades. Por outro lado, a necessidade de armazenar as mensagens acarreta uma penalização no desempenho do sistema.

Existem diversos meios de concretizar a abstracção do canal de mensagens persistente. Uma possibilidade consiste em delegar no editor o ónus de armazenar as mensagens que publica e de contactar periodicamente todos os assinantes até que estes recebam as mensagens publicadas (técnica por vezes designada por "empurrar", do inglês, "push"). A alternativa contrária, consiste em delegar no assinante a responsabilidade de contactar periodicamente o editor para recolher as mensagens que tenham sido publicadas deste o último contacto (técnica designada por "puxar", do inglês, "pull"). Outra alternativa possível consiste em delegar num servidor especializado (ou num conjunto de servidores cooperantes) o papel de armazenar as mensagens do canal e de as disponibilizar para os seus assinantes. Esta última técnica é por exemplo usada no serviço de "news" da Internet, um dos mais populares sistemas editor-assinante.

Este trabalho propõe uma arquitectura que tenta conciliar as vantagens em termos de desempenho dos canais de mensagens voláteis com as vantagens em termos de funcionalidade dos canais de mensagens persistentes. A arquitectura explora a utilização de serviços de comunicação em grupo, não só para oferecer propriedades de fiabilidade e ordenação às mensagens trocadas no canal volátil mas também para garantir a persistência dessas mesmas mensagens com elevada probabilidade. Finalmente propõe-se o desenvolvimento de um protótipo da arquitectura usando um canal de mensagens volátil desenvolvido na linguagem Java, o iBus, o qual suporta já a utilização de diferentes primitivas de comunicação em grupo.

O artigo está organizado do seguinte modo. Na Secção 2 é feita uma panorâmica de alguns dos canais de mensagens mais significativos, quer a nível académico quer a nível comercial. A Secção 3 oferece uma descrição da arquitectura proposta e a Secção 4 descreve o modo como esta pode ser concretizada usando o iBus. As conclusões finais e trabalho futuro encontram-se na secção 5.

2 Trabalho Relacionado

Nesta secção apresentamos uma panorâmica sobre vários sistemas de canais de mensagens. Estes sistemas distinguem-se pelo modo de difusão das mensagens do editor para os assinantes, pelas qualidades de serviço oferecidas em termos de fiabilidade, pontualidade e persistência, etc. Na sua grande maioria, todos os canais de mensagens escondem do utilizador os mecanismos relacionados com a rede de dados, tais como o envio, encaminhamento e recepção de mensagens. O canal de mensagens é responsável por ligar o editor ao assinante de uma forma eficiente e transparente para ambos: o editor, tipicamente, não conhece o conjunto de assinantes e vice-versa (a menos que o desejem).

2.1 Sistema V

O sistema V foi o primeiro a utilizar um canal de mensagens para comunicação entre grupos de processos. O modelo utilizado foi concebido para suportar um conjunto de serviços distribuídos independentemente da sua localização. Apesar de não possuir uma semântica forte no que respeita à fiabilidade dos canais, uma vez que não oferecia garantia de entrega, este sistema foi o primeiro a suportar o paradigma editor-assinante, no qual as mensagens sobre um determinado assunto eram transmitidas para um grupo de processos cujo nome correspondia a esse assunto.

2.2 Isis

O ISIS [1] expandiu os objectivos do sistema V, adicionando-lhe novas funcionalidades em termos de fiabilidade, desempenho e tolerância a faltas. Um dos aspectos chave do sistema Isis foi o de tornar patente a inter-relação que existe entre a noção de fiabilidade e a noção de filiação em sistemas com número variável de participantes. O Isis introduziu o conceito de sincronia virtual, que define que as alterações à filiação do canal devem ser ordenadas em relação ao fluxo de mensagens, de modo a que os participantes tenham uma perspectiva coerente de quem recebeu uma dada mensagem. O sistema oferece vários tipos de garantia de entrega, ordenação de mensagens, replicação e sincronização de várias acções entre os membros do grupo.

2.3 TIB

Com uma arquitectura idêntica ao sistema Isis, o TIB tornou-se um dos primeiros sistemas concebidos para adaptar a sua configuração na presença de máquinas e aplicações que falham dinamicamente, redes que se particionam ou durante períodos de manutenção e actualização dos sistemas. À semelhança do Isis, o TIB oferece também um espaço de armazenamento temporário de mensagens. Esta funcionalidade toma a forma de um assinante que guarda no disco todas as mensagens de um assunto especificado, reenviando-as mais tarde quando solicitadas. Esta tecnologia preserva a ordem e a fiabilidade do mecanismo de publicação e está cuidadosamente sincronizado com a entrega de novas mensagens de modo a que um assinante possa recuperar obtendo o mínimo de mensagem perdidas e seguindo a entrega de novas mensagens na ordem correcta, sem omissões ou duplicações.

2.4 IBM MQSeries e Microsoft Message Queue Server

Outro gama de produtos que utilizam uma arquitectura idêntica, mas concebida como um sistema de mensagens de mais alto nível para aplicações de rede são o MQSeries da IBM e o MSMQ Server da Microsoft. O MQSeries é um produto direccionado para o acesso a sistemas centrais a partir de aplicações cliente, embora possa também ser útil em outras áreas. O MSMQ Server aumenta a capacidade de integração entre as aplicações Windows, assim como a disponibilidade do sistema já que a comunicação entre os serviços se baseia modelo de fila de mensagens. Os produtos da Microsoft e da IBM têm convergido para soluções integradas através de sistemas como o FlaconMQ da Level 8 Systems [2] que oferece uma maior integração entre ambos.

2.5 iBus

O iBus [3] é um sistema recente, disponível desde 1996, que utiliza o Java como linguagem de desenvolvimento nativa. As principais vantagens desta opção prendem-se com a portabilidade das aplicações desenvolvidas relativamente ao sistema operativo, embora também possam existir outros benefícios na utilização de uma linguagem optimizada para funcionar sobre uma rede de comunicações utilizando o protocolo IP. A dis-

tribuição de eventos em larga escala é suportada através de IP multicast [4] e TCP utilizando a abstracção editor-assinante. Um evento pode ser qualquer objecto Java que implemente o interface java.io.Serializable para que possa ser transmitido através de um canal iBus, tornando a solução o mais genérica possível para um maior conjunto de aplicações. Além das facilidades de comunicação o iBus oferece ainda um conjunto de funcionalidade de coordenação tais como detecção de falhas e sincronização entre grupos de processos que subscrevem ou deixam o canal através do modelo de sincronização virtual forte [5]. A arquitectura modular utilizada nos protocolos de comunicação do iBus é inspirada no Horus [6], um sistema que constitui uma evolução do sistema Isis referido anteriormente.

3 Integração de canais voláteis e persistentes

Dado que uma das vantagens dos canais voláteis é o seu bom desempenho, estes requerem a utilização de protocolos de comunicação eficientes. A utilização de protocolos de transporte orientados à ligação ponto-a-ponto não é uma solução eficaz para sistemas com elevado número de editores e assinantes dado que obrigaria a estabelecer um elevado número de ligações. Por outro lado, a utilização de um protocolo de transporte para difusão sem garantias de fiabilidade não satisfaz os requisitos de um vasto leque de aplicações onde se espera que todos os assinantes activos recebam as mensagens produzidas pelos editores.

Deste modo, a arquitectura aqui proposta pressupõe a utilização de serviços de comunicação em grupo, que ofereçam serviços de filiação e primitivas de comunicação com diferentes propriedades. Este tipo de aproximação é consistente com a utilizada em sistemas como o TIB e o iBus. Dado que este tipo de protocolos não aborda o problema da persistência das mensagens, a nossa arquitectura propõe a utilização de um "arquivo de mensagens". O arquivo é composto por um conjunto de membros do canal persistente cuja missão consiste em receber as mensagens transmitidas e arquiva-las para que possam ser novamente enviadas para o canal quando solicitadas.

Os restantes aspectos relacionados com a solução apresentada prendem-se com a forma como produtor e consumidores interagem com o canal. O produtor da informação deve poder continuar a publicar as mensagens no canal através de uma única operação. Por outro lado, os consumidores de-

verão obter as mensagens do canal de uma forma natural. Na presença de uma falha, um consumidor depois de recuperar poderá requer o histórico das mensagens transmitidas no canal enquanto esteve inactivo.

A persistência do canal é oferecida à custa de um membro desse canal designado de "arquivo". Em alternativa pode ser criado um grupo de arquivos de modo a aumentar a disponibilidade do sistema. A persistência baseia-se no pressuposto de que o arquivo recebe todas as mensagens publicadas no canal. O arquivo não é uma propriedade de configuração automática do canal, mas pelo contrário deve ser disponibilizado pelo administrador do sistema. O arquivo deverá possuir um mecanismo para armazenamento não volátil de mensagens, como por exemplo um disco rígido ou uma base de dados (ver figura 1).

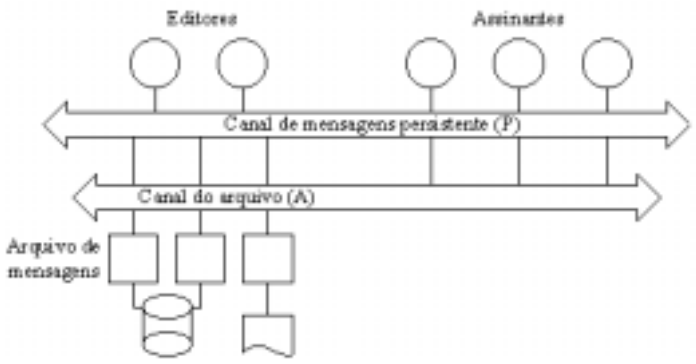

Figura 1 - Canal de mensagens persistente

Na realidade, quando uma aplicação se junta ao canal está a subscrever dois assuntos: o canal persistente (P) e o canal que lhe permite comunicar com o arquivo de mensagens (A). A publicação de mensagens no canal P continua a ser efectuada através de uma operação "push". Os elementos do arquivo recebem a mensagem e gravam-na para que possa ser novamente transmitida se necessário.

Em caso de falha de um assinante o editor poderá continuar a enviar informação para o canal sem prejuízo para o elemento inacessível que, após recuperar, contacta o arquivo através do canal A solicitando-lhe as mensagens não recebidas em P. Em alternativa o arquivo poderá iniciar o processo de actualização do assinante utilizando comunicação ponto-a-ponto, aumentando o grau de confiança no sistema. Enquanto recupera, o assinante continua a receber as mensagens publicadas em P.

Se a falha ocorrer no editor, os assinantes permanecerão activos embora se verifique uma quebra na transmissão. Os assinantes falhados poderão recuperar antes do editor e ainda assim iniciar o processo de actualização, razão pelo qual os membros do "arquivo" não podem funcionar simultaneamente como editores do canal persistente (ver figura 2). Uma vez que não é possível garantir que um editor depois de falhar irá recuperar com a mesma identificação que possuia antes de abandonar o canal (endereço IP, porto de ligação, número de ordem de mensagens enviadas, etc.), assumimos que estes, ao recuperarem após uma falha, são considerados por todos os membros como um novo editor. Esta semântica é semelhante aquela que é utilizada num canal sem persistência.

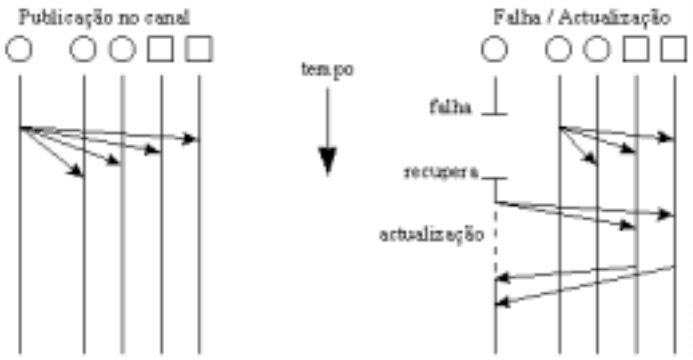

Figura 2 - Exemplo de publicação no canal e recuperação após falha de um editor / assinante

Existem ainda funcionalidades de coordenação e sincronização intrínsecas aos grupos de processos que derivam da utilização do modelo de sincronia virtual, tais como a possibilidade de o arquivo detectar a entrada ou saída de consumidores. Esta característica pode ser aproveitada do seguinte modo:

Os arquivos, sendo assinantes do canal de difusão de mensagens, utilizam os serviços de filiação em grupo para observarem entradas e saídas de assinantes. Uma vez que a sincronia virtual ordena de modo total as mensagens em relação à informação de filiação, os arquivos podem armazenar no histórico esta informação e posteriormente, recuperar qual a vista em que a falha / desconexão ocorreu. Deste modo, quando um determinado assinante se volta a ligar ao canal, e considerando que o assinante P(n) falhou na vista V(i), as mensagens do histórico podem ser classificadas em

três categorias:

- Mensagens que foram entregues de certeza ao participante: todas as mensagens da vista V(i-1), caso P(n) pertença a V(i-1).
- Mensagens acerca das quais existe incerteza de entrega ao participante: todas as mensagens da vista V(i).
- Mensagens que foram não entregues de certeza ao participante: todas as mensagens da vista V(i+1) até à vista corrente.

Este conhecimento pode ser utilizado para facilitar o processo de reintegração de assinantes que tenham estado desligados do canal, uma vez que restringe o número de mensagens para as quais existe incerteza na entrega. Claro que se o canal só usar comunicação totalmente ordenada, basta ao assinante memorizar em memória estável o número de sequência da última mensagem recebida. No entanto, como existe um custo associado à ordenação de mensagens, é importante optimizar a recuperação em canais que usem ordenações mais fracas.

Utilizando as diferentes primitivas do sistema de comunicação em grupo, podem também obter-se diferentes semânticas em relação à persistência das mensagens. Se for utilizada uma qualidade de serviço fiável, e dado que o armazenamento de mensagens é feito assincronamente, um assinante não possuí no momento da entrega, garantia de que essa mensagem venha a ser tornada persistente. Uma sequência de falhas desfavorável, incluindo a falha desse assinante assim como do processo editor pode levar a que todas as cópias voláteis da mensagens se percam antes desta poder ser registada no arquivo. Para aplicações onde este comportamento não é aceitável, é possível usar uma primitiva de comunicação mais forte, mas com menos desempenho, denominada de difusão uniforme [7]. Esta primitiva garante que se uma mensagem é entregue a um assinante do canal será entregue a todos os assinantes, pelo que a mensagem só não será tornada persistente se todas as cópias do arquivo falharem.

O sistema apresentado baseia-se na semântica FIFO como a qualidade de serviço mínima (um canal não ordenado tornaria impraticável identificar de modo eficiente quais as mensagens já entregues). A recuperação pode ser controlada pela aplicação ou ser gerida de modo automático pelo canal. No primeiro caso, as aplicações solicitam as mensagens não recebidas utilizando um pedido baseado num vector de mensagens com posições do tipo P(m) onde "P" representa o Produtor da mensagem e "m" o

respectivo número de ordem. A recuperação automática pode seguir duas abordagens distintas. A primeira, mais "optimista", consite em não entregar nenhuma mensagem uma vez que se assume que nenhuma mensagem se perdeu enquanto a aplicação esteve falhada. Na segunda abordagem, "pessimista", todas as mensagens enviadas para o canal serão entregues garantindo que nenhuma mensagem será perdida (esta metodologia, apesar de parecer excessiva, justifica-se pelo simples facto de não ser possível garantir que uma determinada mensagem entregue à aplicação foi processada, uma vez que a própria aplicação pode falhar depois de ter recebido a mensagem e antes de a processar).

Dado que se considera um sistema aberto, em que o número de assinantes não está pré-definido, não é possível determinar quando uma mensagem já foi entregue a "todos" os potenciais assinantes. Deste modo, qualquer mecanismo de reciclagem automática de memória terá que se basear noutros critérios. Este tipo de critérios pode ser configurado e aplicado ao nível dos arquivos. Mensagens arquivadas podem ser descartadas usando critérios como a antiguidade, número de mensagens já arquivadas, percentagem de utilização da memória disponível, etc.

4 Persistência no iBus

Sendo o iBus uma plataforma de desenvolvimento orientada por objectos utiliza um método de programação que conduz à manipulação das suas classes funcionais de modo a estender as capacidades de cada objecto por forma a obter as funcionalidades desejadas. Uma das principais vantagens deste paradigma é a sua modularidade que permite introduzir novas qualidades de serviço ao iBus sem alterar ou retirar as características daquelas já existentes.

Apesar de estender as funcionalidades dos canais de mensagens do iBus, o mecanismo de comunicação persistente proposto poderá ser aplicado a outros sistemas do mesmo género, embora nos exemplos apresentados prevaleça a semântica funcional do iBus.

Dado que o modelo apresentado alarga o conjunto de elementos que interagem com o canal aos membros do arquivo, além dos produtores e dos consumidores, será conveniente examinar separadamente as funções de cada um. De uma forma simplificada podemos resumir essas funções na seguinte tabela:

	Canal Persistente	Canal do Arquivo
Produtor	editor	(nenhuma função)
Consumidor	assinante	pedido / resposta
Arquivo	assinante	ponto-a-ponto

Tabela 1 - Principal tipo de função / comunicação desempenhada pelos vários membros do canal

Se observado isoladamente, o canal persistente funciona como se de um canal volátil se tratasse. Os produtores editam a informação no canal, os consumidores e o arquivo assinam essa informação. Esta solução não requer nenhum tipo de modificações no produtor dado que este elemento não interage com o canal do arquivo. Em relação aos consumidores e arquivo deverão compor a sua pilha de protocolos baseados na introdução de novas camadas (do inglês "layers"). Em relação aos consumidores e arquivo, deverá ser introduzido uma nova camada de "Sincronização" responsável pela integração dos consumidores que falham e recuperam com os restantes membros do canal. O arquivo deverá ainda possuir uma camada de "Arquivo" cuja principal função consiste em guardar as mensagens enviadas numa unidade de armazenamento não volátil como no seguinte exemplo:

```
// exemplo de uma aplicacao do tipo Arquivo
public class Archiver {
  // canal de mensagens


iBusURL url = iBusURLFactory.create("SystemChannel", ...);

// pilha de protocolos (QoS) com as camadas de integracao e arquivo


Stack s = new Stack("DISPATCH:SYNC:ARCHIVER:PULL:FRAG:FIFO:NAK:REACH:IPMCAST");
  // etc...
}
// exemplo da camada de integracao
public class SYNC extends ProtocolObject {
  // canal de integracao


iBusURL a_url = iBusURLFactory.create("SyncChannel", ...);
  // Qos fiavel para deteccao de falhas
  Stack stackArc = new iBus.Stack("Reliable");
  // etc...
```

```
}
```

A camada de integração deverá registar o estado do canal (isto é, para cada produtor conhecer o número de ordem da última mensagem enviada para a aplicação) e integrar os vários consumidores do canal depois de recuperarem de uma falha executando diversas acções tais como responder às solicitações das aplicações sobre mensagens não recebidas. A qualidade de serviço fiável (reliable) adiciona detecção de falhas ao protocolo de difusão do canal. Neste caso as aplicações poderão ser notificadas quando outras aplicações se juntam ou abandonam o canal, um aspecto particularmente importante para que os arquivos possam iniciar o processo de sincronização dos consumidores quando estes retornam ao canal. Outros tipos de qualidades de serviços podem ser definidas de acordo com as especificidades de cada sistema.

5 Conclusões

Neste artigo apresentamos uma arquitectura para adicionar canais persistentes ao iBus, uma plataforma de desenvolvimento para aplicações distribuídas orientadas por objectos. A solução proposta consiste na criação de um arquivo para o armazenamento persistente de mensagens, sendo a sua utilização transparente para os editores. As aplicações cliente subscrevem o canal requisitando uma nova qualidade de serviço designada de "persistência".

A arquitectura descrita corresponde a trabalho em curso. Como exemplo de aplicação está a ser preparado um protótipo de um sistema de informação financeira para distribuição de cotações em tempo real. A solução proposta não considera os problemas provocados pelas partições na rede. No futuro serão desenvolvidos mecanismos para suportar a sincronização entre grupos que se dividem.

Referências

[1] Birman, K. P. e Van Renesse, R., "Reliable Distributed Computing with the Isis Toolkit", IEEE Computer Society Press, 1994.

[2] Hiperligação: http://www.level8.com/falcon.htm.

[3] Maffeis, S., "iBus - The Java Intranet Software Bus", SoftWired AG (Fevereiro 1997).

[4] Deering, Steven E., "Host Extensions for IP Multicasting", RFC 1112, Request for Comments (Agosto 1989).

[5] Friedman, R., e Van Renesse, R., "Strong and Weak Virtual Sincrony in Horus", 1996 IEEE Symposium on Reliable Distributed Systems, IEEE Press (Outubro 1996).

[6] Van Renesse, R., Birman, K. P. e Maffeis, S., "Horus: A Flexible Group Communication System", Communications of the ACM 39, 4 (Abril 1996).

[7] Moser, E., Melliar-Smith, M., Agarwal, A., Budhia, R. e Lingley-Papadopoulos, C., "Totem: A Fault-Tolerant Multicast Group Communication System", Communications of the ACM 39, 4 (Abril 1996).