UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
Departamento de Informática

Comunicação entre Grupos de Processos Utilizando o Modelo Editor-Assinante

Jorge Alexandre Cabrita Morgado
(Licenciado)

Dissertação para obtenção do grau de
Mestre em Informática

Fevereiro de 2000

Comunicação entre Grupos de Processos Utilizando o Modelo Editor-Assinante

Jorge Alexandre Cabrita Morgado

Dissertação submetida para provas

de mestrado em

Informática Informática

Departamento de Informática

Faculdade de Ciências da Universidade de Lisboa

Lisboa

Fevereiro de 2000

Dissertação realizada sob a orientação do

Doutor Luís Eduardo Teixeira Rodrigues

Professor Auxiliar

Faculdade de Ciências da Universidade de Lisboa

Resumo

Os canais de mensagens constituem uma abstracção que permite suportar a coordenação entre aplicações distribuídas sem obrigar à associação explícita entre os intervenientes. De acordo com esta abstracção, as aplicações interagem através de um terceiro componente, designado por canal, que recolhe mensagens publicadas por processos editores e as entrega a processos assinantes. Os canais de mensagens podem ser voláteis, caso em que as mensagens publicadas são entregues apenas aos assinantes activos no momento da publicação, ou persistentes, caso em que o canal armazena as mensagens para posteriormente serem recuperadas por assinantes que se encontrem inactivos no momento da publicação.

Este trabalho propõe uma arquitectura que permite acrescentar características de persistência a canais de mensagens voláteis tirando partido de mecanismos de comunicação fiável entre grupos de processos. A arquitectura executa o armazenamento de mensagens assincronamente, fora do caminho crítico das mensagens trocadas entre os editores e assinantes activos. Uma concretização da arquitectura é proposta para o iBus, um sistema de canais de mensagens voláteis desenvolvido em Java.

Abstract

A message bus is an abstraction that supports the coordination of distributed applications without enforcing an explicit binding between the participants. According to this paradigm, applications use a third-party, the message bus, to exchange data. The bus collects messages produced by publishers and delivers them to subscriber processes. Messages can be volatile, if delivered to the active subscribers and discarded afterwards, or persistent, if stored by the bus such that they can be delivered later to subscriber that are not active when the message is published.

The thesis presents an architecture that allows to add persistence characteristic to volatile message buses. The architecture makes use of reliable group communication mechanisms to add these features in a transparent manner. Messages are stored asynchronously, outside the critical path from the publisher to the subscriber, thus introducing minimal overhead to the volatile part of the channel. A prototype based on the iBus system is presented and evaluated.

Palavras Chave

Sistemas distribuídos

Modelo editor-assinante

Canais de mensagens

Comunicação em grupo

Tolerância a faltas

Keywords

Distributed systems

Publisher-subscriber model

Message bus

Group communication

Fault-tolerance

Agradecimentos

Em primeiro lugar agradeço ao professor Luís Rodrigues pelo apoio, motivação e todas as ideias que tornaram este trabalho possível.

Em seguida agradeço à HLC Telecomunicações & Multimédia pelos recursos disponibilizados sem os quais não seria possível realizar as simulações apresentadas no final desta dissertação. Agrade co ainda a todos os meus colegas em geral e um agradecimento especial ao Hélder Santos, Henrique Pais e Manuel Rodrigues.

Por último, agradeço à minha família e todos os amigos que tiveram a paciência necessária para permanecerem ao meu lado durante todos momentos.

Lisboa, Fevereiro de 2000.

Jorge Alexandre Cabrita Morgado

Este trabalho é dedicado aos meus pais.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 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 inter-operabilidade, 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[1] 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 (normalmente em disco) de modo a poderem ser mais tarde recuperadas por assinantes que não estejam ligados 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 contac-

[1] Internet Protocol.

tar 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.

Esta dissertação estuda a integração dos diversos modelos de canais de mensagens voláteis e persistentes propondo uma arquitectura que permite armazenar as mensagens publicadas no canal sem degradar o desempenho dos participantes activos ou que não necessitam de persistência. O objectivo é conciliar as vantagens em termos de eficácia dos canais de mensagens voláteis com as vantagens em termos de funcionalidades dos canais de mensagens persistentes. A tolerância a faltas é uma importante propriedade de não apenas nos sistemas críticos mas também nos sistemas distribuídos de propósito geral. Dado que o número de máquinas que compõem um sistema distribuído tende a aumentar com o crescimento desse sistema, o incremento dos componentes físicos e lógicos aumenta a probabilidade da ocorrência de falhas. As funcionalidades de uma arquitectura tolerante a faltas devem ser suportadas pelo sistema a um nível mais baixo, tornando-se transparentes para o programador. A arquitectura explora a utilização de serviços de comunicação em grupo, não só para acrescentar propriedades de fiabilidade e ordenação às mensagens trocadas no canal volátil mas também para garantir a persistência dessas mesmas mensagens com elevada probabilidade. A validação da solução apresentada será efectuada utilizando 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.

A dissertação está estruturada do seguinte modo. No Capítulo 2 faz-se uma panorâmica de alguns dos sistemas de canais de mensagens mais significativos, salientando as suas principais funcionalidades e limitações. O Capítulo 3 descreve a arquitectura proposta e o Capítulo 4 descreve o modo como esta pode ser concretizada usando o iBus, apresentando algumas medidas de desempenho que nos permitem validar a eficiência da solução e efectuamos uma análise comparativa discutindo as principais vantagens/desvantagens desta solução. Por último, as conclusões finais encontram-se no Capítulo 5.

Capítulo 2

Sistemas de Canais de Mensagens

2.1 Modelo Editor-Assinante

O crescimento recente dos sistemas de mensagens entre grupos de processos tem permitido desenvolver modelos e mecanismos capazes de responder ao aparecimento de novas aplicações que precisam de ser executadas sobre plataformas heterogéneas. A integração de recursos acrescenta valor aos sistemas de processos, desde as infra-estruturas de suporte à troca de mensagens (do inglês, "Message-Oriented Middleware" ou MOM) até às infra-estruturas de suporte à interacção entre objectos como o CORBA[1] [18] (do inglês, "Object Request Brokers", ou ORBs). Estes sistemas são constituidos por arquitecturas cliente/servidor que aumentam a inter-operabilidade, a portabilidade e a flexibilidade das aplicações de forma a serem distribuídas entre múltiplas plataformas. Oferecem ainda interfaces para programação de aplicações, as quais são portáveis e reduzem a complexidade do desenvolvimento de aplicações distribuídas sobre diferentes sistemas operativos e protocolos de comunicações. Estas interfaces são de alto nível e escondem os detalhes de concretização dos serviços.

[1]Component Object Request Broker Architecture.

2.1.1 Características Técnicas

Produtos representativos da gama de sistemas de suporte ao modelo editor-assinante são o MessageQ da DEC e o MQSeries da IBM. Este tipo de suporte também se encontra em alguns sistemas orientados por objectos, sobre a denominação de tecnologia de agentes de mensagens assíncronas, como por exemplo o CORBA Event Notification Service [19]. Tecnicamente, estes produtos apresentam-se como pacotes de *software*, concebidos para operarem em arquitecturas cliente/servidor, com componentes que se executam quer no cliente quer no servidor e que suportam chamadas assíncronas entre as aplicações [2]. As mensagens são armazenadas transitoriamente em repositórios designados por *filas de mensagens*. Estas, retêm a informação transmitida enquanto os programas de destino estão inacessíveis, eliminado por isso o problema da conectividade. Os sistemas de filas mensagem utilizam essencialmente comunicação assíncrona e ponto-a-ponto, embora também suportem mensagens síncronas. Posicionam-se entre os clientes e o servidor central, aceitando as mensagens recebidas e retransmitindo-as até as seu destino utilizando os protocolos oferecidos pelos sistemas operativos nativos, organizando as mensagens para que sejam executadas de uma forma segura (em alguns casos executando as aplicações necessárias caso não estejam activas). A sequência é repetida para propagar a resposta.

Este tipo de sistemas é mais adequado para aplicações orientadas por eventos - quando ocorre um evento, a aplicação cliente delega no sistema de mensagens a responsabilidade de notificar o servidor de que é necessário executar uma determinada acção. Outros exemplos referem-se aos sistemas orientados por objectos uma vez que fornecem um mecanismo (conceptual) para comunicação ponto-a-ponto entre objectos.

As vantagens dos sistemas síncronos e assíncronos devem ser consideradas para cada aplicação específica. Ao contrário do modelo de chamadas a procedimentos remotos (do inglês, “Remote Procedure Call” ou RPC [5]), que utiliza um mecanismo síncrono

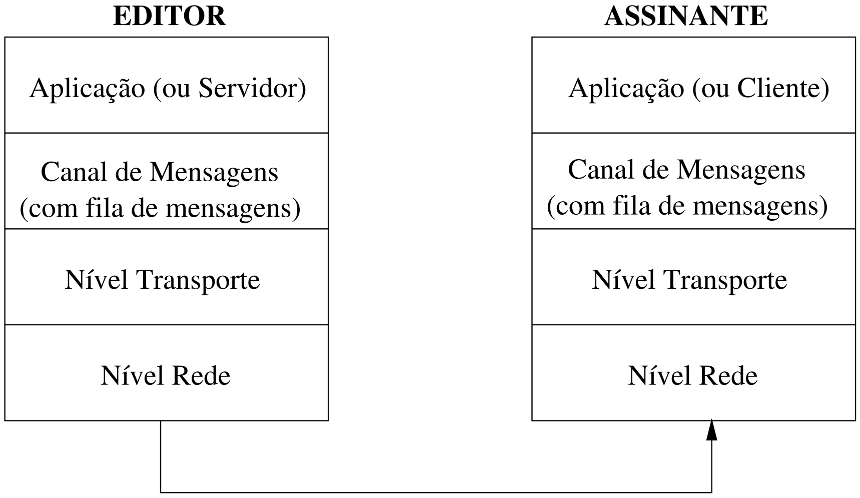

Figura 2.1: Protocolos de suporte ao modelo editor-assinante.

e bloqueante, os sistemas baseados em canais de mensagens ignoram a sobrecarga da rede de comunicações. Ou seja, um determinado processo pode enviar continuamente pedidos a um servidor mesmo que este esteja demasiado ocupado com outro tipo de tarefas. Para além disso, estes sistemas tendem a ser mais flexíveis do que as RPCs pela simples razão que um sistema pode funcionar sincronamente e em caso de falha do servidor adaptar-se à comunicação assíncrona [22].

2.1.2 Terminologia

A tendência actual no desenvolvimento de aplicações distribuídas consiste na sua decomposição em diversos módulos funcionais, que podem ser distribuídos por diferentes máquinas. Uma distribuição modular facilita do desenvolvimento das aplicações, visto que cada um dos módulos pode ser reutilizado individualmente ou em conjunto. A integração dos diversos componentes pode ser efectuada através de um canal de mensagens, oferecendo maior flexibilidade de serviços tais como mecanismos de comunicação em grupo e partilha de informação.

Os vários componentes do sistema são executados de forma concorrente e comunicam entre si em tempo real utilizando o canal de mensagens. Esses componentes podem inclusive ligar-se e desligar-se de forma dinâmica do canal.

A utilização do termo “canal” (do inglês “bus”) é análoga ao barramento de dados de um computador: os componentes do sistema são como placas ligadas ao canal e as aplicações são executadas por um interpretador análogo à unidade de processamento central. De facto, os canais de mensagens são como máquinas virtuais, construídas sobre uma arquitectura modular baseada em sistemas abertos.

O canal de mensagens representa também um mecanismo que facilita o desenvolvimento de aplicações mais estruturadas ao nível do sistema. As técnicas de estruturação clássica devem ser aplicadas no desenvolvimento de cada um dos componentes individualmente, algo que pode variar de acordo com os sistemas ou linguagens utilizadas. As aplicações são compostas a um nível mais alto de abstracção, baseando-se nos próprios componentes e nas suas principais funcionalidades. Esses componentes, independentemente da sua dimensão ou complexidade, são do ponto de vista do canal de mensagens, aplicações que partilham um mecanismo de comunicação comum.

Os clientes do canal que recebem a informação publicada designam-se de consumidores ou assinantes, enquanto os servidores que enviam essa informação são designados de produtores ou editores.

Em seguida vamos definir um conjunto de conceitos, introduzindo a terminologia normalmente utilizada neste contexto. Sempre que possível esses termos e conceitos serão utilizados neste trabalho de modo a evitar ambiguidades e a facilitar a compreensão do texto.

- Servidor, Produtor ou Editor: estes termos são normalmente utilizados para definir a entidade que publica as mensagens no canal, ou seja, a origem da informação;

- Cliente, Consumidor ou Assinante: representa a entidade que recebe as mensagens enviadas pelos editores para o canal;

- Evento: uma mensagem assíncrona que indica a ocorrência de uma actividade num componente distribuído;

- Mensagem: um conjunto de dados (ou informação) que é enviado para o canal de mensagens. Uma mensagem pode ser composta por um conjunto de mensagem mais pequenas;

- Componente ou Módulo: representa um conjunto de objectos que funcionam como uma entidade única. Os componentes conhecem os objectos que utilizam, pelo que são capazes de os fazer interagir entre si;

- Objecto: representa um tipo de classe de serviço previamente definido. Um objecto é formado por um conjunto de dados e possui métodos que permitem manipular esses dados;

- Rede de comunicações: é a estrutura física que é utilizada para a comunicação entre os diversos membros do canal de mensagens;

- Protocolo de comunicações: é a estrutura lógica que permite que todos os membros de um canal consigam comunicar entre si. Apesar de poderem ocorrer diversas conversões de protocolos entre os vários componentes do canal, o protocolo de comunicações deverá definir o formato de cada mensagem de modo a que estas possam ser percebidas pelos editores e pelos assinantes do canal;

- Filiação em grupo (do inglês, "group membership"): utilizada em sistemas que suportam a difusão de mensagens para grupos de assinantes; como veremos, este conceito é particularmente útil para desenvolver soluções com tolerância a faltas;

- Persistente: este termo aplica-se a um conjunto de dados ou informação que pode ser recuperado mesmo que ocorra uma falha no sistema;

- Falha: a falha de um sistema ocorre quando o serviço fornecido se desvia das condições previamente definidas na especificação do serviço [14];

- Erro: é a parte do estado do sistema que pode conduzir à falha, isto é, quando um erro afecta o serviço fornecido, ocorre uma falha [14];

- Falta: a causa de um erro representa uma falta no sistema. Uma falta que não produz efeitos diz-se inactiva. Quando de algum modo é activada, a falta produz um erro [14].

2.1.3 Modelos de Comunicação

Para além das facilidades de comunicação disponibilizadas, os canais de mensagens oferecem ainda serviços de filiação, execução automática de serviços e execução distribuída. O canal de mensagens pode ser iniciado pela própria aplicação que o utiliza ou pode funcionar autonomamente até que as aplicações se liguem e utilizem o canal.

Tal como ilustrado na Figura 2.2, as mensagens pode ser classificadas numa de duas categorias. A primeira, designada de modelo Pedido-Resposta, é composta por mensagens enviadas a um componente ou objecto em particular de modo a requisitar que determinada acção seja executada por esse componente (i.e., para invocar um método). Este modelo de comunicação pode ser síncrono (similar às RPCs) ou assíncrono. A segunda categoria de mensagens é baseada na comunicação por *eventos*. Os eventos são mensagens enviadas por um editor para o canal e que podem ser recebidas por nenhum ou vários assinantes desse canal. Os assinantes apenas subscrevem os assuntos que pretendem receber, delegando no canal a responsabilidade de filtrar as mensagens fora do

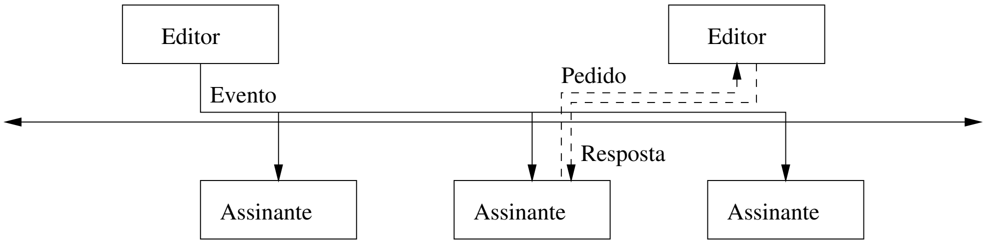

Figura 2.2: Os principais tipos de comunicação utilizados no modelo editor-assinante.

seu interesse - outra vantagem, uma vez que os assinantes apenas "consomem" aquilo que "pagam". Os eventos são por definição assíncronos.

Existem diversas razões que justificam a utilização de um protocolo de comunicações assíncrono. Em geral os canais de mensagens assíncronos são mais eficientes em termos de desempenho. Num sistema que utilize o modelo de comunicações pedido-resposta, o processo que inicia a comunicação pode executar outras tarefas enquanto aguarda por uma resposta ao seu pedido. Por outro lado, essas respostas nem sempre são necessárias: os processos que produzem eventos não têm de esperar que as suas mensagens sejam processadas pelos consumidores do canal. Nos canais persistentes, outra vantagem refere-se ao facto de os assinantes não necessitarem de estar activos em simultâneo com os editores. Neste caso, possuir um mecanismo de comunicação assíncrona é particularmente importante uma vez que a comunicação síncrona iria falhar.

Outro modelo possível é a comunicação Ponto-a-Ponto, apesar de este mecanismo ser normalmente utilizado nos casos onde é necessário transferir grandes volumes de informação uma vez que apenas dois membros do canal estão envolvidos.

A um nível mais baixo, o canal de mensagens é apenas responsável por classificar as mensagens pelo seu tipo e encarregar-se da sua disseminação e entrega. O actual conteúdo da mensagem, ou seja, a informação contida na própria mensagem, é determinado pelo protocolo utilizado entre os editores e assinantes do canal.

2.1.4 Principais Vantagens

Os canais de mensagens representam uma tecnologia alternativa à comunicação síncrona, personificada pelo RPC, que pode ser utilizada para desenvolver aplicações que necessitam de comunicar, através de uma rede, e em geral, com outras aplicações noutros tipos de plataformas. Este tipo de tecnologia não está disponível para os utilizadores finais, mas pelo contrário deve ser aproveitada pelas aplicações desses utilizadores quando necessitam de comunicar. Algumas das suas principais vantagens são:

- Oferecer uma plataforma padrão para comunicação entre diferentes ambientes (heterogeneidade);

- Separar as componentes lógica e funcional dos próprios ambientes de processamento, garantindo transparência quanto à localização e independência do protocolo utilizado;

- Permite que os responsáveis pelas aplicações se concentrem no modelo do negócio, abstraindo-se da componente "rede de comunicações"t'. Este aspecto é particularmente importante uma vez que simplifica o desenvolvimento de sistemas distribuídos reduzindo o nível de conhecimentos necessários para os conceber;

- Aumenta as possibilidades de operação entre aplicações oferecendo modelos de comunicação assíncrona;

- Reduz a vulnerabilidade dos sistemas, particularmente às falhas na rede;

- Permite o desenvolvimento simultâneo para várias plataformas dado que a maior parte dos produtos utiliza o mesmo interface de programação para todos os sistemas onde é suportado.

Este tipo de sistemas oferece um elevado desempenho às aplicações: as mensagens recebidas são mantidas numa fila de espera enquanto não puderem ser enviadas. Isto significa que o destinatário não necessita de estar disponível para responder - enquanto isso a aplicação pode continuar a operar até obter a resposta, oferecendo por isso um modelo de comunicações não bloqueante.

2.1.5 Principais Limitações

De um modo geral os sistemas de filas mensagens são produtos proprietários, o que significa que na maior parte dos casos são incompatíveis entre si. A utilização de um único sistema poderá resultar numa maior dependência do seu fabricante ou vendedor relativamente ao suporte técnico ou futuras actualizações. Este aspecto poderá ter um impacto negativo na flexibilidade do sistema, na sua manutenção, portabilidade e interoperabilidade com outras aplicações instaladas.

O canal de mensagens deverá estar activo em cada uma das plataformas de rede instaladas. As consequências desta imposição depende das características dos sistemas existentes:

- Nem todos os produtos existentes no mercado suportam todos os sistemas operativos e protocolos de comunicações. As possibilidades de escolha dependem das aplicações a suportar e respectivos protocolos, ou vice-versa.

- As tarefas de manutenção tendem a aumentar para os administradores das aplicações, em especial em grandes arquitecturas distribuídas com sistemas heterogéneos.

2.2 Sistemas representativos

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 a 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). Em qualquer um dos casos, o modelo de comunicações utilizado é assíncrono, embora os vários produtos existentes possuam extensões que suportam o envio de mensagens síncronas.

Na última década um razoável número de sistema distribuídos têm oferecido um estilo de comunicação baseado em quadros de mensagens (do inglês "bulletin board") direccionado para programas. Nesta tecnologia, com nomes tão variados como comunicação em canais de mensagens ("message bus"), filas de mensagens ("message queues"), endereçamento baseado no assunto ("subject-based addressing") ou endereçamento por grupos de processos, as aplicações registam-se em canais de mensagens divididos por assuntos, subscrevendo-os. A mesma ou outras aplicações podem então publicar mensagens sobre um ou mais assuntos garantindo que apenas os processos que subscrevem a mesma fila de mensagens, o mesmo assunto ou que pertencem ao mesmo grupo do editor receberão as mensagens publicadas (ver Figura 2.3).

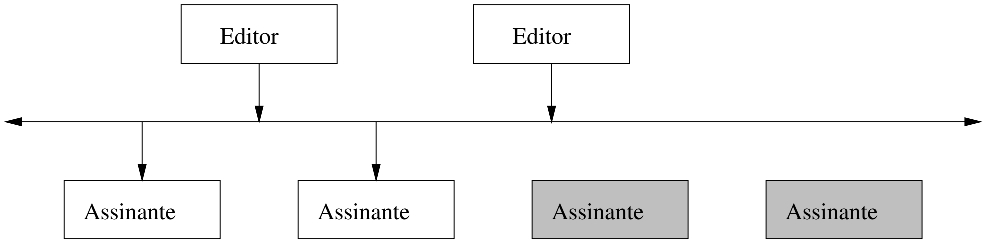

Figura 2.3: Publicação num canal de mensagens.

2.2.1 Sistema V

O Sistema V [7] 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 dos primeiros 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. Um editor pode enviar as mensagens para o grupo sem conhecer a sua filiação e um assinante pode simplesmente juntar-se ao grupo para começar a receber as mensagens publicadas, introduzindo uma abordagem onde a origem e o destino das mensagens estão conceptualmente separados, tal como referido anteriormente.

2.2.2 Isis

Apesar de o estilo de comunicações utilizado no sistema V não ter sido concebido com o objectivo de tolerar faltas nos membros que subscrevem um canal, pela simples razão de que as primitivas de comunicação em grupo não ofereciam uma forma segura de conhecer todos os assinantes de um determinado assunto, introduziu um novo conceito para disseminação da informação onde os membros ou serviços do canal poderiam ser alterados de uma forma dinâmica. Este paradigma voltou a ser utilizado por outros sistemas nos anos seguintes adicionando-lhe novas funcionalidades em termos de fiabilidade, desempenho e tolerância a faltas. Um dos aspectos chave do sistema Isis [4] 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 sistema 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. O Isis consegue garantir que mesmo na presença de falhas, a sequência de mensagens não será comprometida: todos os assinantes que permaneçam ligados ao canal deverão ver as mesmas mensagens, na mesma ordem.

2.2.3 TIB (Teknekron Information Bus)

O TIB [17] tornou-se um dos primeiros sistemas comerciais 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. Extremamente popular no suporte de transacções, o TIB oferece garantia de entrega de mensagens na mesma ordem em que foram publicadas. Os serviços fundamentais incluem comunicação fiável e ordenação de mensagens usando a semântica "exactamente uma vez" na ausência de falhas e garantia de entrega utilizando um repositório de mensagens persistentes no caso de falhas. À semelhança do Isis este sistema introduziu ainda um espaço de armazenamento temporário de mensagens associada à facilidade de retransmissão. Em ambos os sistemas 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 funcionalidade 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 ausências ou duplicações.

2.2.4 Horus

Baseado no sistema Isis, a arquitectura do Horus [23] consiste numa solução modular com o objectivo de permitir o desenvolvimento de aplicações escaláveis compostas por um elevado número de grupos de processos. O seu modelo é baseado em filiação entre grupos de processos e primitivas de comunicação para suportar tolerância a faltas, balanceamento de carga, replicação de dados coerente e segurança. O Horus é um sistema modular composto por uma pilha de protocolos - a comunicação entre membros de um grupo de processos pode ser optimizada incluindo ou excluindo dinamicamente determinados módulos na pilha de protocolos de comunicação. O Horus oferece suporte para uma grande quantidade de protocolos ao nível das suas camadas como por exemplo, ordenação total, garantia de entrega, fragmentação de mensagens e segurança da informação através de criptografia, entre outros. A sua flexibilidade capaz de responder aos mais variados requisitos de cada aplicação, estende-se ao interface com o sistema, às várias propriedades oferecidas por cada um dos níveis do protocolo e até mesmo à própria configuração do canal.

2.2.5 IBM MQSeries

Outro exemplo de uma gama de produtos que utilizam uma arquitectura idêntica, mas concebida como um sistema de mensagens de mais alto nível para aplicações de rede é o MQSeries da IBM cujo primeiro objectivo era o de desenvolver um produto que tornasse mais simples o acesso aos sistemas de *mainframes* via rede de comunicações.

O MQSeries é um produto direccionado para o problema do acesso a sistemas centrais a partir de aplicações cliente, embora o sistema da IBM possa também ser útil em outras áreas. Tecnologias deste tipo, tipicamente apresentam o *mainframe* através de um interface abstracto, o qual permite ao programador utilizar uma arquitectura cliente/servidor para desenvolver aplicações distribuídas. O MQSeries possui ainda a vantagem de poder ser executado sobre mais de vinte sistemas operativos diferentes para além de contar com inúmeros produtos e serviços que lhe acrescentam novas funcionalidades. A tecnologia utilizada situa-se ao nível do protocolo, entre o sistema operativo e a aplicação criando uma rede transparente através do seu sistema de filas de mensagens. O programador necessita apenas de saber o nome da fila (e não o seu endereço) para enviar uma mensagem. Depois de colocada na fila, a mensagem é retirada pela aplicação de destino utilizando um protocolo assíncrono. A representação física de uma fila de mensagens pode ser feita por um ou vários ficheiros, assim como pode servir a uma ou várias aplicações. Também uma aplicação pode obter mensagens de mais do que uma fila. Estas capacidades podem ser utilizadas para balancear a carga de todo o ambiente de comunicações.

2.2.6 BEA MessageQ 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 MessageQ da BEA (anterior DEC MessageQ) e o MSMQ da Microsoft [8].

Neste tipo de produtos é apresentado ao utilizador a abstracção da caixa-de-correio para onde as mensagens podem ser enviadas pelas aplicações da rede, à semelhança do correio-electrónico entre utilizadores embora as mensagens possuam informação binária. Mais tarde, as aplicações autorizadas processam as mensagens recebidas, respondendo às mesmas em conformidade ou simplesmente analisando-as sem executarem uma resposta. Tal como seria de esperar, estes produtos possuem um extenso suporte para opções tais como níveis de prioridade (para que as mensagens mais urgentes possam passar à frente de outras menos críticas), controlo de fluxo (para que as filas de espera de mensagens não cresçam indefinidamente no caso de os processos demorarem demasiado tempo a levantar as mensagens), segurança, balanceamento da carga (quando vários processos utilizam a mesma fila-de-espera), persistência dos dados e tolerância a faltas (para aplicações críticas).

O produto da Microsoft apresenta uma elevada capacidade de integração entre as aplicações Windows e um aumento da disponibilidade do sistema já que a comunicação entre os serviços se baseia modelo de fila de mensagens: comunicações rápidas exigem que as mensagens sejam apenas mantidas em memória, enquanto que os serviços mais críticos deverão utilizar mensagens persistentes aumentando assim a capacidade de recuperação do sistema em caso de falha. As mensagens persistentes podem ainda ser do tipo transaccional. Embora a introdução de transacções torne o sistema globalmente mais lento devido à natureza do protocolo utilizado (protocolo de confirmação atómica em duas fases [12]), introduz um mecanismo que garante a consistência do sistema mesmo na presença de falhas [9].

Embora limitado ao sistema operativo Windows NT Server, o MSMQ oferece várias alternativas para transmissão de mensagens. Entretanto outro género de aplicações, como por exemplo o FlaconMQ da Level 8 Systems [1], têm vindo a reduzir a incompatibilidade entre os sistemas da Microsoft e da IBM permitindo uma maior integração entre ambos e oferecendo a possibilidade de clientes Windows acederem a outras plataformas sem aumentar a complexidade de desenvolvimento. O FlaconMQ é composto um conjunto de produtos que estendem as funcionalidades oferecidas pela API[2] do MSMQ Server para ambientes não-Windows, tais como Unix, VMS, AS/400 e *mainframes* IBM através da ligação com o IBM MQ Series.

2.2.7 CORBA Event Notification Service

A programação orientada por objectos facilita o desenvolvimento de interfaces adicionais capazes de oferecer novas propriedades de comunicação. Esta será talvez a principal razão pelo qual os diversos sistemas se procuram adaptar a este novo paradigma oferecendo novos interfaces de desenvolvimento para este tipo de linguagens. O serviço de eventos do CORBA (CORBA Event Notification Service) oferece já a abstracção do produtor e do consumidor de forma a decompor a comunicação entre os objectos utilizando canais de mensagens. Enquanto que os primeiros produzem os eventos de dados, os segundos processam esses dados. O CORBA introduz ainda o conceito de canal de eventos ("event channel") - um objecto que permite que múltiplos fornecedores comuniquem com múltiplos consumidores de uma forma assíncrona. Tal objecto é por si só um consumidor e fornecedor de eventos. Este serviço surge devido ao facto de o CORBA utilizar unicamente o modelo pedido-resposta síncrono. No CORBA um pedido é sempre dirigido a um objecto em particular e requer que ambos o cliente e o servidor estejam disponíveis. A solução consiste em introduzir no sistema um objecto do tipo canal, capaz de interceptar todos os eventos que ocorrem entre o produtor e o consumidor, recolhendo-os para que na presença de falhas as mensagens nunca se percam e possam alcançar o seu destino após a execução de um algoritmo de recuperação. Desta forma o CORBA consegue oferecer um serviço de mensagens que na perspectiva dos produtores e consumidores funciona assincronamente baseado no modelo editor-assinante. De notar, no entanto, que o CORBA não possui nenhum me-

[2] Application Programmer Interface.

canismo para classificação de mensagens baseado no seu assunto, uma das principais facilidades deste tipo de arquitectura. Cada nova mensagem enviada pelos produtores deve ser classificada utilizando um conjunto de assuntos ou atributos previamente definidos. Separadamente, cada um dos assinantes que deseja ser informado quando ocorre um determinado evento deverá subscrever esse assunto para receber as mensagens automaticamente, ou procurar verificar se existem novas mensagens relacionadas com esse assunto. O principal objectivo do ENS consiste em fornecer um serviço de difusão de mensagens confiável mesmo que os produtores e consumidores não se conheçam. Infelizmente as possibilidades de utilização do ENS são limitadas uma vez que nem todas as implementações CORBA possuem este serviço.

O CORBA possui ainda outro serviço designado de "Persistent Object Service" (POS) que acrescenta persistência às mensagens enviadas. Este serviço permite manter várias colecções de objectos para que possam ser utilizados pelas aplicações à *posterióre*. Os objectos são organizados de uma forma eficiente em termos de consistência, persistência e controlo de acessos de acordo com o serviço para o qual se destinam. Estas funcionalidades facilitam a criação de bases de dados onde os objectos idênticos são organizados em tabelas relacionais, por oposição aos mecanismos de armazenamento baseados em sistemas de ficheiros que contém objectos de tipos irregulares.

2.2.8 iBus

O iBus [15] é um produto 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. O iBus não requer qualquer tipo de bibliotecas adicionais para além da própria máquina virtual Java . A distribuição de eventos em larga escala é suportada através de IP multicast[3] [10] e TCP[4] [20] utilizando a abstracção do editor e do assinante. Um evento pode ser qualquer objecto Java concebido a partir do 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. Para além das facilidades de comunicação (difusão em grupo, ponto-a-ponto, bloqueante, não-bloqueante e pedido-resposta) o iBus oferece ainda um conjunto de funcionalidades 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 [11]. A arquitectura modular utilizada nos protocolos de comunicação do iBus é inspirada no Horus, um sistema que constitui uma evolução do sistema Isis referidos anteriormente. O conjunto de protocolos de comunicação suportados e as várias qualidades de serviço oferecidas podem ser extendidas pelos programadores à medida das suas necessidades.

2.3 Análise Comparativa

Os sistemas anteriormente descritos requerem a existência de um elevado número de processos o que implicitamente aumenta o risco de falhas. Apesar de os sistemas tolerantes a faltas serem difíceis de construír, várias propostas têm sido apresentadas nomeadamente no que se refere a serviços de comunicação em grupo, tais como filiação e difusão de mensagens entre grupos de processos. De um modo geral, todos os sistemas distribuídos tolerantes a faltas oferecem mecanismos para ordenação de mensagens (causal e/ou total), acordo distribuído e replicação. Este tipo de serviços garantem que todos os processos correctos deverão receber a mesma sequência de mensagens ou no-

[3] Baseado nos protocolos IP/UDP permite a disseminação de eventos a partir de um editor para vários assinantes. Uma vez que se trata de um serviço datagrama, podem ser adicionadas facilidades para retransmissão de mensagens que se perdem.

[4] Transmission Control Protocol.

tificações de falhas. Por outro lado, é necessário prevenir todos os pontos de falha do sistema, nomeadamente o editor, os assinantes ou mesmo a própria rede de comunicações. Este último aspecto é particularmente importante uma vez que uma partição desfavorável poderá conduzir à formação de dois grupos que apesar de pertencerem ao mesmo canal poderão ficar incoerentes.

2.3.1 Replicação

A replicação dos vários componentes do sistema é uma das principais técnicas utilizadas para minimizar as possibilidades de falhas no canal. O Isis, é o exemplo de um sistema que oferece elevada fiabilidade através do suporte à replicação. Este sistema utiliza uma tecnologia baseada em grupos de processos, na qual diversos protocolos de acordo são utilizados para garantir que as mensagens serão entregues a todos os processos que subscreveram um determinado assunto. Esta aproximação também permite a replicação activa dos editores, de modo a que uma réplica possa fornecer exactamente a mesma sequência de mensagens que o editor primário. O Isis possui um módulo para gestão de falhas que garante que a sequência de mensagens não será comprometida: todos os assinantes que permaneçam ligados ao canal verão as mesmas mensagens, na mesma ordem.

O TIB é um exemplo de outro sistema que utiliza uma abordagem semelhante. Este sistema oferece a capacidade de recuperação a faltas se existirem pelos menos dois editores equivalentes para uma determinada classe de assuntos. Nesta configuração os assinantes estão inicialmente conectados a um editor primário; o segundo editor (réplica) funciona como se publicasse informação no canal mas o TIB inibe a transmissão de quaisquer dados. No entanto, se o primário falhar, o TIB acabará por detectar a ocorrência - nessa altura o sistema reconfigurar-se-á automaticamente de modo a que os assinantes comecem a receber dados da outra fonte. Desta forma o sistema TIB é capaz de garantir que as mensagens serão entregues normalmente na ordem que foram enviadas, e que de um modo geral, não ocorrerão momentos de ausência de mensagens ou recepções fora de ordem. Estas propriedades podem, no entanto, ser violadas se a rede de comunicações ficar bastante sobrecarregada ou um assinante ficar temporariamente particionado da rede. Nestas situações, existirá um vazio na sequência de entrega de mensagens durante o período de tempo necessário para a recuperação do sistema.

2.3.2 Canais Voláteis

Essencialmente concebidos para aplicações que requerem difusão de mensagens em larga escala, como por exemplo jogos em rede ou a distribuição de vídeo e áudio em tempo real, este tipo de canais apresentam um elevado desempenho uma vez que, ao contrário dos canais persistentes, eliminam o custo associado à gravação das mensagens. Na realidade, os canais voláteis representam o modelo mais simples deste paradigma, daí que sejam suportados por quase todos os sistemas acima referidos. Uma vez que as mensagens enviadas não são tornadas persistentes por nenhum dos componentes do canal, bastará utilizar um protocolo de comunicações não fiável, isto é, sem retransmissões para conceber um canal volátil na sua forma mais simples. Nas redes locais, alguns sistemas aproveitam ainda as facilidades de difusão da rede de comunicações (por exemplo, redes *Ethernet*) para enviar a mesma mensagem para o maior número de assinantes activos sem perdas no desempenho global do sistema (ver Figura 2.4). Outros sistemas oferecem ainda semânticas de fiabilidade suportando mecanismos de retransmissão sobre os protocolos de difusão em grupo.

De qualquer forma, os canais voláteis não suportam um conjunto de aplicações muito alargado. Dado que este tipo de canais de mensagens não tolera a ocorrência de faltas nos assinantes, a única solução consiste em recorrer aos mecanismos de replicação referidos atrás. No entanto, este tipo de solução poderá anular as principais propriedades

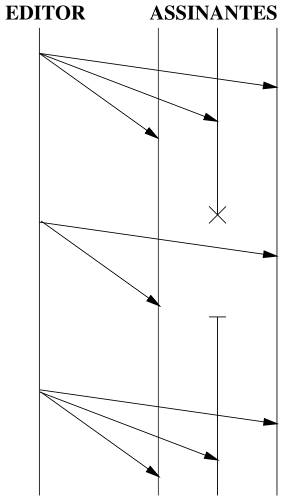

Figura 2.4: Canal de mensagens volátil.

destes canais em termos de desempenho, uma vez que a replicação de assinantes pode revelar-se demasiado complexa por um lado, e fazer aumentar a latência[5] das comunicações por outro.

2.3.3 Canais Persistentes

Os sistemas que suportam canais persistentes possuem a capacidade de armazenar as mensagens enviadas para o canal, para posterior entrega aos assinantes que não se encontrem activos no momento da publicação. Este tipo de semântica faz sentido, numa lógica de "melhor esforço" (do inglês, "best-effort") uma vez que se assume que os assinantes falhados deverão recuperar em algum instante e procurar obter as mensagens não recebidas.

De um modo geral, existem duas formas de introduzir persistência nos canais de mensagens. Uma das abordagens consiste em fornecer uma semântica de persistência baseada no editor. Neste caso, as mensagens são guardadas numa unidade de armazenamento não-volátil antes de serem enviadas para o canal. Na presença de uma falha

[5]Tempo que decorre entre o envio e a recepção de uma mensagem.

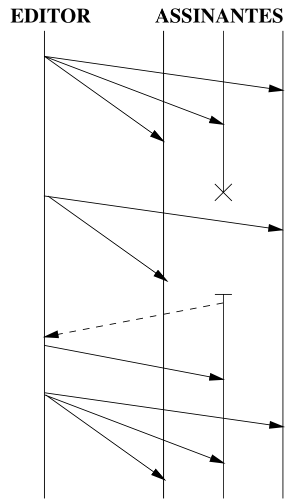

Figura 2.5: Canal de mensagens persistente baseado no editor.

de um assinante, existe a garantia de que as mensagens serão entregues "pelo menos uma vez". O editor é responsável por retransmitir essas mensagens em intervalos de tempo pré-definidos até que seja obtida uma confirmação da entrega ou em alternativa os próprios assinantes contactam o editor para receber essas mensagens. Na ausência de falhas, as mensagens serão entregues "exactamente uma vez" na mesma ordem em que foram enviadas pelo mesmo editor (ver Figura 2.5). Neste caso é importante salientar que as mensagens publicadas por diferentes editores possuem uma sequência de entrega aleatória, embora alguns sistemas ofereçam mecanismos para ordenação total de mensagens verificando-se, no entanto, uma degradação no desempenho do sistema [16].

Esta solução não é difícil de concretizar pois apenas requer que o editor possua uma forma de guardar as mensagens para que possam ser novamente retransmitidas. Isto limita a diversidade de dispositivos em que uma aplicação editor pode ser executada. Por outro lado, a semântica do canal permanece inalterável, ou seja, as mensagens continuam a ser publicadas pelo mesmo editor. O principal problema desta solução é o facto de não tolerar falhas no editor. Mais uma vez será necessário recorrer aos mecanismos de replicação para conceber um canal de mensagens tolerante a faltas, o que como já foi dito, poderá tornar a solução demasiado complexa num canal com um elevado número de editores.

Este tipo de solução foi inicialmente introduzida no sistema Isis (actualmente Orbix+Isis) para suportar replicação utilizando comunicação fiável baseada no modelo de sincronia virtual e numa primitiva de difusão em grupo que oferecia ordem casual de mensagens (através de vectores de tempo) entre grupos que se podem particionar. Do mesmo modo também o sistema TIB oferecia mecanismos de fiabilidade, e tal como o Isis baseia a persistência do canal numa unidade de armazenamento não volátil instalada no editor. Este tipo de soluções é particularmente útil tornando o sistema tolerante a faltas por um lado, e introduzindo mecanismos para balanceamento de carga por outro. Tratando-se de uma falha na rede de comunicações ou no assinante do canal, as mensagens são posteriormente reenviadas prevalecendo a ordem pela qual foram publicadas pelo editor.

Outra solução proposta pelo MQSeries da IBM e pelo Microsoft Message Queue Server consiste em basear a persistência num terceiro componente do canal que funciona simultaneamente como assinante e editor. Este componente, designado por "servidor de mensagens" recebe as mensagens publicadas pelos editores e retransmite-as pelos assinantes depois de as tornar persistentes. Esta solução é compatível com a maior parte dos sistemas já existentes, uma vez que representa apenas uma evolução do paradigma. Ao invés de existir um canal de mensagens, passamos a ter dois canais: um canal para publicação de mensagens onde participam os editores e outro canal para recepção de mensagens composto pelos assinantes. Uma das principais vantagens desta solução é a facilidade com que se podem concretizar diferentes serviços no servidor de mensagens tais como ordem total, conversão de protocolo entre o editor e o assinante, detecção de falhas centralizada, transacções, prioritização e encaminhamento de mensa-

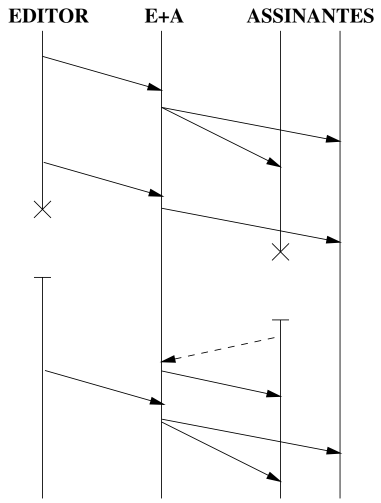

Figura 2.6: Canal de mensagens persistente baseado numa entidade intermédia.

gens entre rede distintas só para citar alguns exemplos. No entanto, esta solução possui desvantagens em termos de desempenho uma vez que obriga ao encaminhamento das mensagens por uma terceira entidade (ver Figura 2.6). Para além disso, o servidor de mensagens pode tornar-se um ponto de estrangulamento do sistema.

2.4 Serviços de Comunicação em Grupo

Como vimos nem todos os sistemas conseguem oferecer a melhor semântica em termos de persistência e eficiência. Por outro lado, a quantidade e qualidade dos serviços oferecidos pode justificar por si só a escolha de um determinado sistema. A maior parte dos canais de mensagens existentes não oferecem mecanismos de "difusão fiável" - em particular, se um processo assinante receber uma mensagem e em seguida falhar, isso não significa que todos os outros assinantes tenham recebido essa mensagem.

A aplicação de um modelo de comunicação fiável a um sistema síncrono pode ser conseguida obrigando o editor a aguardar por uma confirmação de entrega de todos os assinantes do canal. Se essa confirmação não for recebida após um período de tempo previamente definido, o editor assume que o assinante em causa está falhado e remove-o do grupo. No entanto, dado que os canais de mensagens utilizam essencialmente comunicação assíncrona, na ausência de uma confirmação de entrega, o processo editor deverá retransmitir a mensagem. Neste cenário, apesar de as mensagens serem entregues pelo menos uma vez, os editores não podem falhar enquanto não receberem a confirmação de entrega de todos os assinantes. Em termos operacionais, se esta propriedade for violada, o sistema poderá ficar incoerente: os assinantes que não receberam uma mensagem ficarão incoerentes em relação aos restantes. Caso não exista qualquer tipo de comunicação entre assinantes, os elementos que perderam mensagens não possuem mecanismos para actualizar o seu estado, acabando também por serem considerados falhados pelo sistema.

Dado que este tipo de semântica não é aceitável para muitas aplicações, torna-se necessário introduzir no canal de mensagens o conceito de "difusão fiável uniforme" [16]. Neste caso, existe a garantia de que mesmo quando um assinante recebe uma mensagem e falha em seguida, todos os outros assinantes correctos também recebem essa mensagem. Deste modo, o assinante falhado ao recuperar poderá ter perdido mensagens em relação aos restantes membros activos do canal, mas não possuirá um estado incoerente.

Este conceito aplicado aos canais de mensagens pressupõe a existência de persistência. Uma mensagem pode ser considerada persistente quando o canal oferece mecanismos para a sua recuperação (esta propriedade é normalmente designada de "durabilidade"). Uma vez que os assinantes que falham devem poder recuperar o atraso sem contactarem o editor, a persistência deve ser fornecida por uma terceira entidade. No entanto, ao contrário de algumas das soluções apresentadas anteriormente esta entidade não pode constituir um ponto único de falha ou de estrangulamento do próprio canal.

No capítulo seguinte, apresentaremos o nosso modelo de persistência colocando o armazenamento de mensagens fora do caminho crítico das mensagens trocadas entre editores e assinantes activos de forma a aproveitar a eficiência dos canais voláteis.

Capítulo 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 obriga 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. Apesar disso, mesmo nos casos onde é necessário contemplar semânticas como garantia de entrega, este tipo de protocolos consegue ser mais eficiente dado que só é enviada uma mensagem para todos os assinantes do canal.

A grande maioria dos sistemas existentes não oferecem uma semântica forte no que se refere à fiabilidade do canal. Quando oferecem elevada fiabilidade, esta é geralmente obtida à custa da introdução de dispositivos adicionais no canal, tal como o servidor de mensagens, que diminuem o desempenho do sistema. Em alternativa, alguns sis-

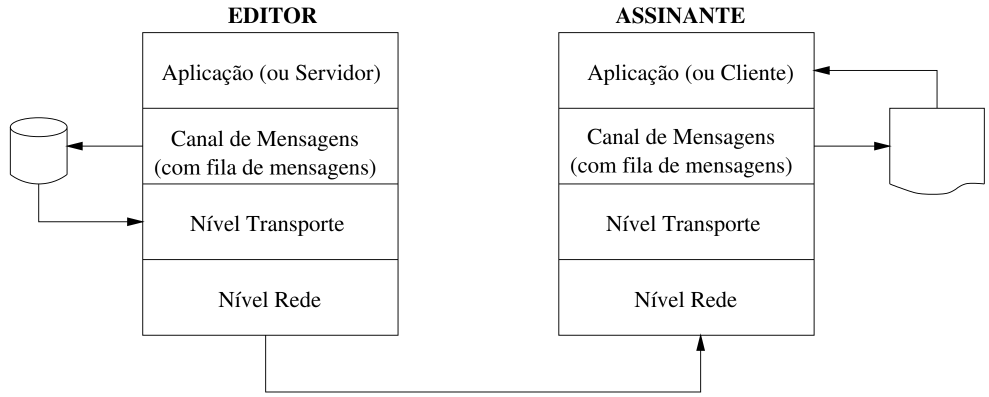

Figura 3.1: Protocolos de suporte à persistência.

temas resolvem este problema introduzindo novas qualidades de serviço ao protocolo utilizado. As arquitecturas modulares dos canais de mensagens facilitam a criação de novas camadas que podem ser inseridas na pilha de protocolos do canal (ver Figura 3.1). Estas componentes têm como função garantir a persistência do canal por forma a garantir que mesmo na presença de uma falha as mensagens poderão ser facilmente recuperadas. No caso do editor, uma mensagem só é enviada para o canal depois de ter sido efectuada a sua cópia persistente; em relação aos assinantes, as mensagens são igualmente salvaguardadas e só então entregues às respectivas aplicações e, no caso de um canal fiável, enviada a confirmação de recepção.

No entanto, não é normal encontrarmos um canal de mensagens que coloque a persistência simultaneamente no editor e no assinante. Tal opção tornaria o canal demasiado lento para além dos recursos que seria necessário dispor em cada um dos seus membros. O exemplo serve apenas para ilustrar como, através da introdução de novas camadas na pilha de comunicações, é possível oferecer os mais diversos tipos de qualidade de serviço. Esta flexibilidade é importante, dado que os vários domínios aplicacionais que requerem um estilo de comunicação orientado por eventos têm diversificado os seus requisitos de fiabilidade, desde a semântica de "pelo menos uma vez" até à garantia de "exactamente uma vez" (situação ideal), requisitos de escalabilidade, disponibilidade, débito e latência.

Como é óbvio, não existe uma única solução capaz de optimizar um conjunto tão vasto de requisitos técnicos. Por esta razão, são necessárias várias concretizações do mesmo serviço concebidas para diferentes ambientes. Diferentes concretizações deverão suportar diferentes qualidades de serviço de acordo com as diferentes necessidades de cada aplicação. Por exemplo, uma concretização que favoreça o desempenho mas que não seja capaz de prevenir a duplicação de mensagens poderá ser útil para algumas aplicações. No entanto, uma concretização de um canal que elimine todas as mensagens não fornecerá um serviço útil.

3.1 Solução Proposta

Nesta dissertação propomos uma arquitectura que pretende integrar as funcionalidades dos canais de mensagens voláteis, em particular no que se refere à sua eficiência, com as vantagens oferecidas pelos canais persistentes nomeadamente a capacidade de tolerar falhas nos membros do canal.

A arquitectura explora mecanismos de comunicação em grupo para tornar eficiente a difusão de mensagens, a detecção de faltas e a recuperação de elementos do sistema. A solução 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 semelhante aquele que é utilizado 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 arquivá-las para que possam ser novamente enviadas para o canal quando solicitadas. A solução proposta não considera os problemas provocados pelas partições na rede de dados ou perdas de mensagens.

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 como se de um canal volátil se tratasse, isto é, sem quaisquer custos de desempenho no seu funcionamento. As aplicações que pretendam utilizar um canal persistente deverão apenas requisitar uma nova qualidade de serviço. Os consumidores deverão 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. Na ausência de falhas os consumidores recebem as mensagens directamente do produtor, tal como acontece nos canais sem persistência.

A persistência do canal é oferecida à custa de um membro desse canal designado de "Arquivo de mensagens". Em alternativa pode ser criado um grupo de arquivos de modo a aumentar a disponibilidade do sistema. A persistência do canal baseia-se no pressuposto de que o arquivo recebe todas as mensagens publicadas no canal, daí que, em canais com um elevado débito de mensagens, o desempenho dos membros do arquivo deve ser suficientemente elevado para não perder nenhuma mensagem. Este valor deve considerar o número de mensagens enviadas num determinado intervalo de tempo, contra a capacidade do arquivo para armazenar essas mensagens nesse período, tal como será explicado mais adiante. Poderá existir um mecanismo coordenado entre os membros do arquivo para o armazenamento das mensagens recebidas, assim como poderemos basear o critério de armazenamento em valores probabilísticos. Se uma mensagem for enviada para todos os arquivos, quanto maior for o desempenho dos membros do grupo, menor será a probabilidade de se perderem mensagens devido à ocorrência de uma falha no arquivo, uma vez que consideramos que a rede de comunicações não perde mensagens.

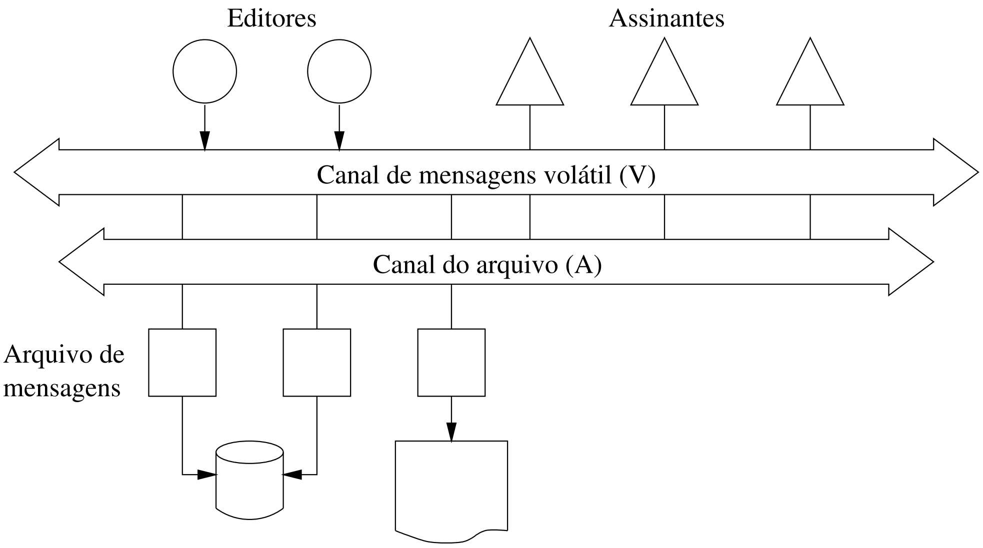

Figura 3.2: Canal de mensagens persistente baseado no arquivo de mensagens.

O arquivo subscreve o canal V na mesma perspectiva dos consumidores, ou seja, com objectivo de receber as mensagens publicadas. O arquivo não é uma propriedade de configuração automática do canal, mas pelo contrário deve ser disponibilizado pelo administrador do sistema. Ao contrário dos restantes membros do canal, o arquivo deverá possuir um mecanismo para armazenamento não volátil de mensagens, como por exemplo um disco rígido ou uma base de dados como apresentado na Figura 3.2.

Na realidade, quando uma aplicação se junta ao canal está a subscrever dois assuntos: o canal volátil (V) e o canal que lhe permite comunicar com o arquivo de mensagens (A). A publicação de mensagens no canal V continua a ser efectuada através de uma só operação - as mensagens são “empurradas” para o canal no sentido de um editor para vários assinantes. Os elementos do arquivo recebem a mensagem e gravam-na para que possa 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 V. 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 pode ou não continuar a receber as mensagens publicadas no canal V.

Se a falha ocorrer no editor, os assinantes permanecerão activos embora se verifique uma quebra na transmissão. Os assinantes que também estejam 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 - se esta propriedade fosse violada, uma falha no editor significaria também uma falha no arquivo de mensagens comprometendo a persistência do canal (ver Figura 3.3). Uma vez que não é possível garantir que um editor depois de falhar irá recuperar com a mesma identificação que possuía antes de abandonar o canal (endereço IP, porto de ligação, número de ordem de mensagens enviadas, etc.) assumimos que estes nunca falham ou, pelo contrário, quando falham e recuperam serão considerados por todos os membros como um novo editor. Esta semântica é semelhante aquela que é utilizada num canal sem persistência.

De seguida abordaremos os diversos componentes do sistema e protocolos propostos com mais pormenor, explicando cada uma das suas funcionalidades.

3.2 Produtores de Informação

Os produtores da informação registam-se no canal mensagens volátil, acima designado de canal V. Apesar de a conjugação dos canais V e A tratar de um canal persistente, esta informação não tem de ser conhecida pelos produtores já que esta funcionalidade é oferecida pelo arquivo independentemente da origem da informação. Do mesmo modo, os produtores não necessitam de saber se existem arquivos uma vez que estes actuam

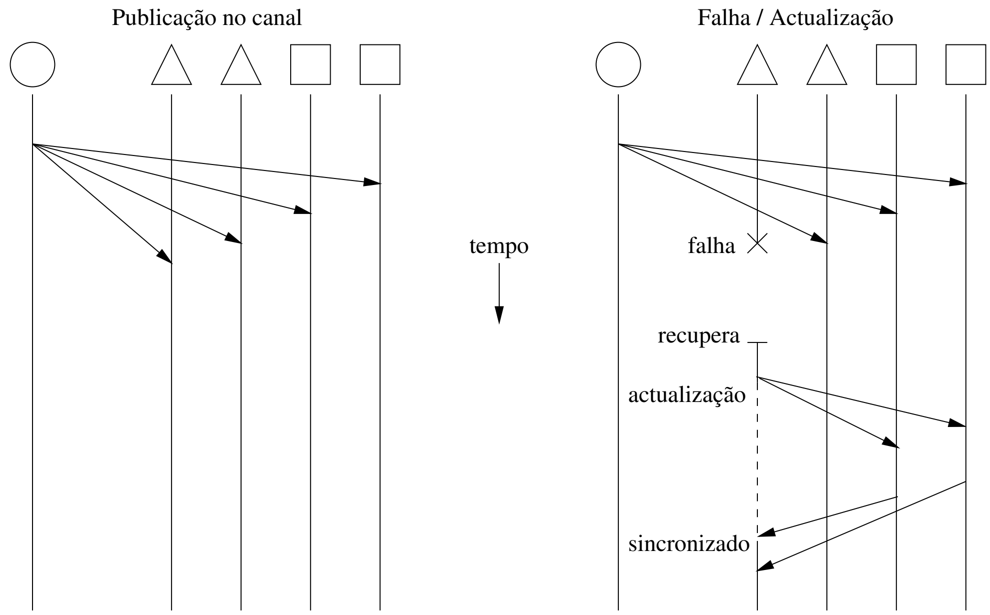

Figura 3.3: Publicação no canal e recuperação após falha de um editor/assinante.

como consumidores do canal persistente - esta é semântica é utilizada nos canais de mensagens voláteis.

Depois de registado no canal os processos produtores limitam-se a enviar as mensagens utilizando diferentes tipos de serviço de acordo com os requisitos do sistema em termos de fiabilidade e desempenho. As semânticas disponíveis em termos de qualidade de serviço são uma característica do canal e determinam os mecanismos de recuperação de faltas tal como falaremos mais adiante. Nesta dissertação vamos considerar dois tipos de primitivas normalmente oferecidas pelos sistemas para comunicação em grupo: sincronia virtual e ordem total.

O modelo de sincronia virtual é normalmente utilizado nos sistemas distribuídos que requerem coordenação entre grupos de processos sem afectar de forma significativa o desempenho do sistema. Este modelo pressupõe que o sub-sistema de comunicação informa os participantes da composição actual do grupo, na forma de uma *vista de grupo*. Adicionalmente, o sistema ordena a entrega de mensagens em relação às vistas.

Este modelo permite que os membros do grupo mantenham uma perspectiva coerente do sistema a partir do fluxo de mensagens recebido. Quando uma mensagem é enviada para um grupo de processos, esta será recebida por todos os membros activos do grupo entre o mesmo par de vistas - ou seja, a mensagem é totalmente recebida antes ou depois de todos os participantes receberem a mesma vista do grupo Se um processo produtor difundir uma mensagem quando o conjunto dos membros do grupo for $\{p_0, ..., p_n\}$, então a mensagem será entregue quando a vista do grupo é ainda $\{p_0, ..., p_n\}$, garantindo que todos os processos receberão essa mensagem nessa mesma vista e consequentemente possam coordenar as suas acções.

Nos casos onde a ordenação oferecida pela sincronia virtual é insuficiente, é possível executar algoritmos para ordenação total das mensagens difundidas para o grupo de processos. Este tipo de ordenação garante que se dois processos p e q receberem ambos as mensagens m_1 e m_2, então ambos deverão entregar m_1 antes de m_2 ou vice-versa. Existem diversas formas de obter ordem total. A mais comum consiste em utilizar um contador sequencial ou um testemunho (do inglês "token") [6]. O contador sequencial é concretizado por um processo eleito para o efeito, enquanto que o testemunho é uma forma de mover esse contador pelos membros do grupo. Enquanto possui o testemunho, o produtor deve incrementar o contador nas mensagens que publica. Os consumidores, ordenam as mensagens de acordo com o número sequência atribuído pelo emissor. Garantindo que o testemunho é único e não se perde, a simples ordenação do número de sequência resulta numa ordem total das mensagens difundidas dentro do grupo. No entanto, enquanto um produtor possuir o testemunho, nenhum outro pode publicar mensagens. Apesar de existirem outras formas de conseguir ordem total, devemos estar conscientes que todas aumentam a latência da comunicação, resultando num pior desempenho das aplicações.

3.3 Consumidores

À primeira vista os consumidores são os principais responsáveis pela necessidade de o canal oferecer mecanismos de persistência. Quando falham e recuperam, será expectável que consigam recuperar as mensagens não recebidas durante o período em que estiveram desligados do canal. No entanto, a arquitectura de persistência aqui apresentada permite a coexistência de dois tipos de consumidores: os interessados na persistência e os consumidores que apenas necessitam de um canal volátil. Este facto representa uma solução flexível uma vez que não obriga a que os produtores possuam algum conhecimento específico do tipo de consumidor.

Os consumidores que não pretendam utilizar a persistência do canal devem apenas registar-se no canal V para receberem as mensagens publicadas pelos produtores. Em caso de falha não existe qualquer mecanismo para recuperação das mensagens não recebidas uma vez que V se trata de um canal volátil.

No caso de aplicações críticas onde a sequência de mensagens não pode ser comprometida, a solução consiste em subscreverem também o canal de mensagens A a partir do qual podem comunicar com o arquivo de modo a obterem as mensagens não recebidas em V. A distinção entre consumidores persistentes e os restantes consumidores do canal deve ser feita pelo arquivo de mensagens a partir da filiação dos canais V e A. Ou seja, assumindo que todos os assinantes registados em A pretendem utilizar o canal persistente, basta identificar esses processos no conjunto de assinantes registados em V. Deste forma é possível ao arquivo conhecer os consumidores que utilizam uma semântica persistente. Por outro lado, o arquivo deverá ser capaz de distinguir os membros do seu grupo dos restantes consumidores em A através das suas funções dentro deste canal, uma vez que todos os membros do arquivo são simultaneamente produtores e consumidores em A (ver Figura 3.4).

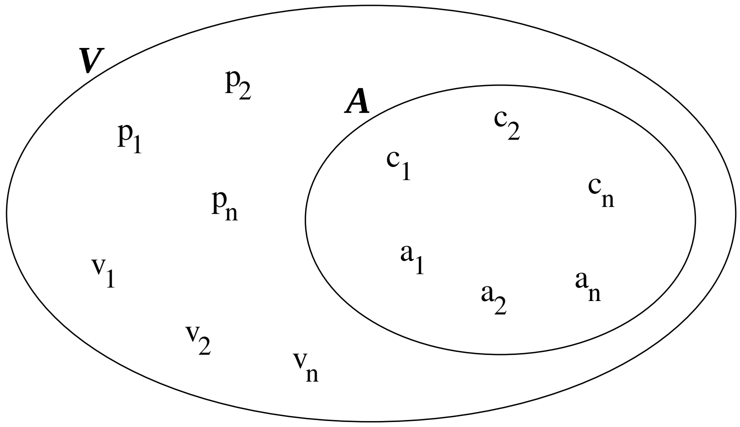

Figura 3.4: Composição dos canais V e A.

- Produtores: Todos os processos que publicam mensagens no canal P.

$$\{p_1,\ ...,\ p_n\}$$

- Arquivos: Todos os processos que publicam mensagens no canal A.

$$\{a_1,\ ...,\ a_n\}$$

- Consumidores Persistentes: Todos os processos que consomem mensagens em A, excluindo os arquivos.

$$\{c_1,\ ...,\ c_n,\ a_1,\ ...,\ a_n\} - \{a_1,\ ...,\ a_n\} = \{c_1,\ ...,\ c_n\}$$

- Consumidores Voláteis: Todos os processos que consomem mensagens em P, excluindo os consumidores persistentes e os arquivos.

$$\{v_1,\ ...,\ v_n,\ c_1,\ ...,\ c_n,\ a_1,\ ...,\ a_n\} - (\{c_1,\ ...,\ c_n\} + \{a_1,\ ...,\ a_n\}) = \{v_1,\ ...,\ v_n\}$$

A qualidade de serviço oferecida pelo canal determina a sua fiabilidade e pode variar de acordo com as necessidades específicas de cada aplicação. No caso de um canal não fiável (isto é, sem mecanismos de retransmissão onde se podem perder mensagens) ou em caso de falha dos consumidores, estes podem ou não iniciar o processo de recuperação com o arquivo utilizando dois tipos de semânticas distintas como veremos em seguida.

3.3.1 Impacto no Desempenho

Em geral, uma solução de persistência deverá operar entre vários editores e vários assinantes. No entanto, a filosofia do protocolo apresentado consiste dissociar as fontes da informação dos consumidores do canal, tornando o canal de mensagens persistente totalmente transparente para as entidades que não utilizam esta semântica. Assim, apesar de os produtores de mensagens e os consumidores voláteis poderem utilizar o canal sem prejuízo no seu desempenho, existe um custo associado aos consumidores persistentes e aos membros do arquivo que são também consumidores do canal V.

Relativamente ao arquivo de mensagens, tal como foi referido atrás, teremos de considerar o custo necessário ao armazenamento das mensagens. Em relação aos consumidores persistentes, os mecanismos de recuperação apresentados em seguida, requerem que estes mantenham um estado persistente acerca das mensagens que recebem, como por exemplo, a identificação da última mensagem correctamente processada antes de ocorrer a falha. Naturalmente, estes factores poderão produzir um impacto negativo no débito do canal.

O desempenho global do sistema é afectado não só pelo volume de mensagens transmitidas, mas também pelo número de assinantes que falham em relação à quantidade de mensagens enviadas durante os períodos de falha.

3.3.2 Recuperação

O processo de recuperação de assinantes baseia-se na utilização do canal A para comunicar com o arquivo de mensagens tal como referido anteriormente. O assinante, depois de voltar ao canal, pode iniciar a recuperação de mensagens utilizando comunicação em grupo com os vários membros do arquivo. Utilizando difusão em grupo não é necessário que o consumidor conheça o número de membros do arquivo e dado que o canal de recuperação utiliza comunicação fiável existe a garantia que todos os membros do arquivo irão receber os pedidos do consumidor.

No entanto, apesar de todos os arquivos activos possuírem condições para participarem no processo de sincronização do consumidor, qualquer solução que envolvesse todos os arquivos consumiria demasiados recursos do canal, com prejuízos óbvios para o desempenho global do sistema. Desta forma torna-se necessário definir um conjunto de critérios que permitam aos membros do arquivo decidir entre si qual é o elemento do grupo que possui melhores condições para responder ao consumidor. Alguns desses critérios podem ser:

Antiguidade: O arquivo mais antigo do grupo será aquele que necessariamente possui o maior número de mensagens arquivadas. Os restantes arquivos podem não possuir todas as mensagens uma vez que são mais recentes. O mecanismo de sincronização de arquivos será explicado mais adiante, embora este critério pressuponha que mesmo os arquivos que não estão totalmente sincronizados com os restantes membros do grupo podem responder aos pedidos dos consumidores. No caso do arquivo mais antigo estar já a responder a pedidos de outro consumidor será seleccionado o próximo elemento mais antigo do grupo.

Circular: Baseado no algoritmo "round-robin" este mecanismo permite acima de tudo partilhar a carga entre arquivos sem recorrer a técnicas demasiado sofisticadas, tais como ocupação do processador, memória disponível e sobrecarga da rede de comunicações. Os arquivos deverão organizar entre si a lista de membros disponível criando, por exemplo, uma tabela ordenada dos respectivos endereços IP, e a partir daí responder aos pedidos dos consumidores de uma forma circular.

Desempenho: O arquivo com maior capacidade de processamento será seleccionado para sincronizar o consumidor. Em alternativa pode ser considerado o desempenho do arquivo em relação ao consumidor a sincronizar, ou seja, deve ser escolhido um arquivo cujas características em termos de desempenho se assemelham ao consumidor, visto que não faz sentido utilizar um arquivo demasiado rápido com um consumidor demasiado lento ou vice-versa. Sempre que um arquivo detecte a entrada de um novo processo no canal A, deve obter as suas características físicas, organizando em seguida esta informação utilizando um algoritmo pré-definido e repetindo este processo em intervalos de tempo regulares.

Proximidade: Este critério baseia-se na distância média entre arquivo e consumidor medida, por exemplo, em milisegundos. O objectivo é diminuir a latência das comunicações e consequentemente o tempo necessário para sincronizar o consumidor. No caso de se tratar de um consumidor localizado numa rede distinta, deve ser considerado o número de "saltos" entre cada um dos arquivos e o consumidor.

Existem diversos mecanismos que podem ser utilizados para seleccionar o arquivo com maior capacidade para sincronizar o consumidor. Outros critérios possíveis seriam, por exemplo, o número de mensagens em arquivo, fiabilidade ou débito da rede de comunicações ou mesmo escolha aleatória. No entanto, optou-se por não apresentar aqui uma descrição extensiva de todas as técnicas descritas na vasta bibliografia disponível sobre o assunto [21]. Naturalmente, deve ser seleccionado o mecanismo que melhor se adequa ao sistema em questão sendo que, em alguns casos podem ser combinados diversos critérios de forma a optimizar o processo de recuperação dos consumidores.

Relativamente à forma como as mensagens podem ser recuperadas do arquivo, propomos duas abordagens distintas que serão apresentadas em pormenor mais adiante: *i)* recuperação explícita baseada num mecanismo de navegação (do inglês "browsing") que permita ao assinante seleccionar as mensagens em falta; *ii)* um processo de recuperação automático capaz de identificar as mensagens não recebidas dependendo da qualidade de serviço utilizada.

3.4 Arquivos

Uma das principais vantagens desta solução é o facto de tolerar faltas em todos os intervenientes do canal persistente, incluindo o próprio arquivo de mensagens (desde que permaneça pelo menos um arquivo activo). Apesar de a persistência do canal ser baseada no arquivo, é possível recorrer a mecanismos de replicação de modo a que mesmo na ausência de um dos elementos do arquivo, os restantes consigam receber e armazenar as mensagens enviadas.

A sincronização do arquivo pode ser conseguida utilizando o mesmo mecanismo de recuperação dos assinantes. Uma vez que o arquivo é também um assinante do seu próprio canal de mensagens (canal A), onde são publicadas as mensagens perdidas, o processo de reintegração com os restantes elementos do grupo é trivial. A mesma abordagem deve ser utilizada no caso de ocorrer uma falha no arquivo. Ao recuperar, o elemento do arquivo poderá estar atrasado em relação aos restantes e deverá recuperar utilizando o mesmo mecanismo dos assinantes do canal. A replicação do arquivo deve basear-se em critérios bem definidos, tais como a proximidade do produtor e consumidores ou volume de mensagens a arquivar.

Relativamente a este último aspecto deve ser tido em consideração o volume de mensagens do canal em oposição à capacidade do arquivo para receber e tornar essas mensagens persistentes. Não é possível definir uma regra abstracta que nos permita calcular o número de servidores que deverão constituir o arquivo uma vez que esse valor dependerá não só da capacidade de processamento dos produtores e dos membros do arquivo, mas também do volume de mensagens a tornar persistentes. Este parâmetro de configuração pode ser obtido de modo empírico, através da avaliação de desempenho da arquitectura final. No entanto, de modo a simplificar a solução apresentada, consideramos que todos os membros do arquivo recebem e tornam persistentes todas as mensagens enviadas desde que permaneçam correctos.

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 política de reciclagem automática de memória terá que se basear noutros critérios. Este tipo de critério pode ser configurado e aplicados 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 ou ainda a eliminação explícita de mensagens no caso de existir a garantia de que todos os assinantes (activos e restantes) já receberam as mensagens a apagar.

Os mecanismos de sincronização de arquivos apresentados nos parágrafos seguintes serão baseados numa máquina de estados - *arquivos sincronizados* e *arquivos não sincronizados* - coordenada entre os diversos membros do grupo de arquivos. Tal como no caso dos algoritmos para gestão de dados replicados, as técnicas para concretizar a manutenção dos estados dos arquivos dependem das características do sistema alvo e dos critérios de desempenho que se pretendam satisfazer. Isto é, mesmo que um arquivo não esteja sincronizado com os outros elementos do seu grupo, pode ainda assim participar no processo de reintegração dos assinantes dependendo do mecanismo de recuperação a utilizar. No entanto, como veremos em seguida, qualquer algoritmo para gerir este tipo de situações de uma forma automática tornaria a solução apresentada demasiado complexa e difícil de conceber. Assim, os *arquivos não sincronizados* são excluídos do grupo durante os processos de recuperação automática, podendo apenas participar nos processos de recuperação explícita.

3.5 Mecanismos de Recuperação

O sistema apresentado baseia-se na semântica FIFO como a qualidade de serviço mínima. Um canal não ordenado, apesar de ser possível de concretizar, tornaria impraticável identificar de modo eficiente quais as mensagens já entregues - outro problema que nos desviaria da persistência do canal. Poderiam ser considerados diversos mecanismos para recuperação de mensagens dependendo do protocolo de comunicação utilizado. Diferentes tipos de ordenações requerem políticas diferentes para descobrir quais as mensagens que não foram recebidas enquanto o assinante esteve falhado. Para uma melhor compreensão, veja-se os diversos algoritmos existentes: ordem casual [13, 3] ou ordem total [16] só para citar alguns exemplos.

Nesta dissertação iremos considerar três métodos para recuperação de mensagens. No primeiro caso o processo de recuperação é controlado pela própria aplicação utilizando um mecanismo para consulta/recuperação das mensagens contidas no arquivo. Os restantes métodos baseiam-se em algoritmos para comunicação entre grupos de processos onde a recuperação de assinantes é gerida automaticamente pelo canal. Os factores enunciados para esses algoritmos são pois também relevantes neste contexto.

3.5.1 Navegação no Arquivo

Uma vez que este mecanismo de recuperação é iniciado pelo assinante, não existe a necessidade de verificar qual o membro do arquivo mais adequado para retransmitir as mensagens pretendidas. Sendo um processo explícito, a ligação é estabelecida utilizando comunicação ponto-a-ponto entre o assinante e o arquivo usando o canal de mensagens A. Dependendo das mensagens a obter, podem ser efectuadas várias ligações a diversos membros do arquivo. Neste processo não existe qualquer garantia de que esses membros estejam activos no momento da ligação ou podem simplesmente não estar disponíveis para responder ao assinante. No entanto, é um mecanismo extremamente simples de conceber e permite acima de tudo recuperar individualmente as mensagens do passado ou de diferentes produtores sem produzir uma carga excessiva sobre o arquivo.

Tal como foi referido acima, usando este mecanismo não é necessário que o arquivo consultado esteja sincronizado com os restantes membros do seu grupo (estado *arquivo não sincronizado*). Mesmo neste caso, o arquivo pode possuir as mensagens pretendidas pelo assinante, ou seja, o arquivo pode estar atrasado em relação aos restantes mas não estará incoerente em relação ao seu conteúdo.

Apesar de este ser um mecanismo de recuperação de mensagens simples de fornecer, qualquer processo de navegação pelo conteúdo do arquivo pressupõe que os objectos não são estanques, isto é, o arquivo é capaz de aceder ao conteúdo total ou parcial das mensagens de modo a que estas possam ser consultadas. Depois de seleccionadas as mensagens pretendidas pelo assinante, decorre o processo de envio. O exemplo mais flagrante deste tipo de mecanismo de consulta é-nos fornecido pelo protocolo NNTP, utilizado pelo serviço de "news" da Internet. Também neste caso o cliente pode consultar diversos servidores, que podem ou não possuir os artigos desejados. Na primeira consulta, apenas o cabeçalho dos artigos é enviado para o cliente; em seguida apenas os artigos seleccionados são enviados.

3.5.2 Recuperação Automática

O processo de sincronização automática é uma propriedade do canal de mensagens persistente utilizado para manter os assinantes actualizados após recuperarem a uma falha. Este processo pode ser iniciado quer pelo assinante quando retorna ao canal, quer pelo arquivo de mensagens quando detecta a sua reentrada utilizando os serviços de filiação. Em qualquer dos casos, tal como no processo anterior, deve ser utilizado o canal do arquivo, embora neste caso quando a comunicação é iniciada pelo assinante seja preferível utilizar os mecanismos de difusão em grupo do canal para que todos os arquivos sejam contactados numa só operação.

No entanto, ao contrário do mecanismo de recuperação explícito, sendo um processo automático é necessário que todos os membros do arquivo estejam sincronizados entre si (estado *arquivo sincronizado*) para que possam participar na recuperação do assinante. Dado que este processo pretende actualizar os assinantes que recuperaram, desde a última mensagem recebida antes de falharem até à mensagem mais recente publicada no canal V, é condição mínima necessária que o arquivo utilizado na recuperação esteja completamente actualizado em relação a este canal.

Nesta dissertação vamos considerar dois mecanismos de recuperação automática, que dependem das primitivas semânticas oferecida pelo canal em termos de ordenação.

Ordem Total

Apesar de a maior parte dos sistemas que oferecem ordenação total de mensagens possuírem piores desempenhos em termos de escalabilidade e latência da comunicação, dadas as suas especificidades, algumas aplicações requerem este tipo de serviço. Em particular, se um processo entregar um par de mensagens por uma determinada ordem e posteriormente falhar, qualquer processo correcto que venha a entregar esse par de mensagens deve entregá-las na mesma ordem. Um serviço que satisfaça esta propriedade pode ser designado por serviço de *ordenação total forte*. No entanto, este serviço pode ser enfraquecido, por exemplo, oferecendo uma *ordenação total fraca*, a qual garante a ordem de entrega apenas para processos que permaneçam correctos. Infelizmente, quanto mais forte é a ordenação mais dispendiosa é a sua concretização. Na base dos protocolos conhecidos, as propriedades de ordenação forte tendem a exigir um maior número de mensagens trocadas entre o grupo de processos, sujeitando o sistema a maiores tempos de espera até que as mensagens sejam entregues às aplicações.

No entanto, apesar de prejudicarem o desempenho global do sistema, qualquer semântica de ordenação total beneficia o algoritmo de recuperação de mensagens. O algoritmo proposto funciona do seguinte modo: cada assinante mantém a identificação da última mensagem recebida; depois de recuperar de uma falha, o assinante informa o arquivo acerca da última mensagem recebida; a partir daí, todas as mensagens seguintes serão entregues pelo arquivo (ver Algoritmos 1 e 2). Utilizando um canal de mensagens totalmente ordenado, existe a garantia que a mensagem n terá sempre sido enviada depois da mensagem $n-1$ e antes da mensagem $n+1$.

Algoritmo 1 Assinante em operação normal.

```
Em caso de falha/recuperação executar Algoritmo 2
Para cada mensagem recebida {
  Guardar informação sobre id da mensagem
}
```

Algoritmo 2 Sincronização do assinante após recuperar de uma falha.

```
Para cada editor conhecido {
  Informar o arquivo sobre da id da mensagem
}
Retomar Algoritmo 1
```

Sincronia Virtual

Uma vez que todos os algoritmos utilizados para obter ordem total são dispendiosos e dado que nem todos os sistemas necessitam ou suportam este tipo de semânticas, podemos conceber um canal de mensagens parcialmente ordenado recorrendo ao modelo de sincronia virtual introduzido no sistema Isis [4]. Neste modelo, os processos podem juntar-se ao grupo dinamicamente e são automaticamente excluídos quando falham. Por outro lado, quando um processo se junta ao grupo, é efectuada uma transferência de estado do grupo a partir de um ou vários membros do grupo. Assim, todos os processos observam sequências idênticas acerca da filiação do grupo, as quais são designadas de *vistas* de grupo. Qualquer par de processos que pertença a duas vistas consecutivas, recebe o mesmo conjunto de mensagens durante o período entre ambas as vistas, ou seja, em caso de falha, se uma mensagem m_j é entregue a esses processos, então, qualquer mensagem m_i, que o sistema tenha de entregar antes de m_j, também terá sido entregue[1].

O mecanismo de recuperação proposto baseia-se pois nesta semântica, através da qual os membros do arquivo correctos (estado *arquivo sincronizado*) conseguem descobrir com algum grau de certeza as mensagens que não foram recebidas pelos assinantes enquanto estiveram falhados. Dado que as vistas funcionam como um ponto de sincronização, os arquivos, sendo assinantes do canal de mensagens volátil (canal V), utilizam os serviços de filiação em grupo para observarem entradas e saídas de assinantes persistentes. Esta característica pode ser aproveitada como a seguir se descreve.

Dado 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 ocorreu a falha/desconexão do assinante. Deste

[1] Alguns sistemas consideram que um processo pode ter recebido mensagens que tenham sido enviadas depois de o processo falhar, caso a sua falha não seja reportada de imediato.

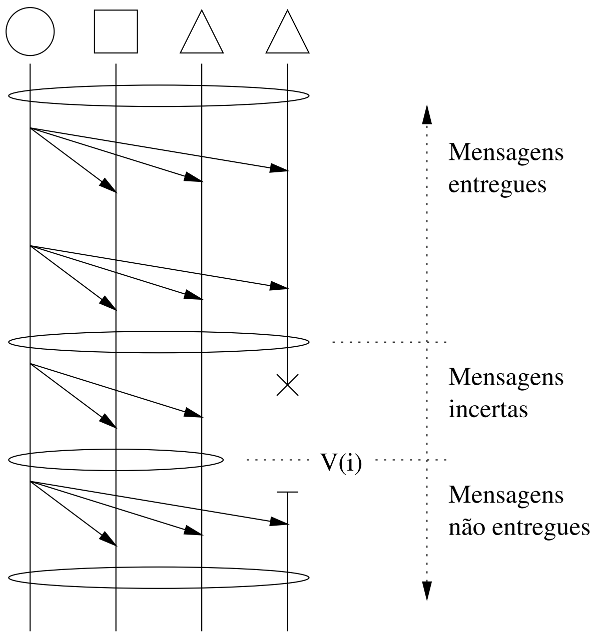

Figura 3.5: Classificação de mensagens efectuada pelo arquivo com base nas vistas do grupo.

modo, quando um determinado assinante se volta a ligar ao canal, e considerando que o assinante $A(n)$ falhou na vista $V(i)$, as mensagens do histórico podem ser classificadas em três categorias (ver Figura 3.5):

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

O processo de reintegração pode ser iniciado quer pelos assinantes quando recuperam, quer pelo arquivo de mensagens quando detecta uma reentrada no 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 da última mensagem recebida e processada antes de falhar. A partir desta informação o arquivo identifica as mensagens em falta, ou seja, todas as mensagens desde $P(m+1)$ inclusive, até à mais recente mensagem arquivada.

Em alternativa, se o processo de recuperação automática for iniciado pelo arquivo, podemos considerar duas abordagens distintas para o caso das mensagens acerca das quais existe incerteza de entrega ao participante, utilizando o conhecimento obtido a partir das vistas de grupo acima descrito. A primeira, mais "optimista", consiste 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 ao arquivo 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). Por último, todas as mensagens que foram não entregues de certeza ao assinante serão enviadas pelo arquivo.

Como vimos o conhecimento obtido a partir das vistas 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. As vistas do grupo funcionam pois como um ponto de sincronização entre os participantes do canal. No entanto, uma vez que entre duas vistas consecutivas as mensagens poderão chegar desordenadas, qualquer semântica de ordenação deve ser oferecida pelo canal dependendo da qualidade de serviço utilizada. 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 possui 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 [16]. 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. Se um assinante falhar depois de ter recebido uma mensagem existe a certeza que todos os restantes assinantes também receberam essa mensagem. Quando o assinante falhado regressar ao canal pode estar atrasado em relação aos restantes mas o seu estado não está incoerente.

A difusão fiável uniforme extende o modelo de "difusão fiável". Apesar de a sua concretização não ser trivial um dos mecanismos mais utilizados consiste em cada assinante comunicar ao editor a recepção das mensagens enviadas (ACK); quando o editor receber a confirmação de entrega de todos os assinantes, responde positivamente à recepção tornando a entrega efectiva. No entanto, este tipo de solução implicará um conjunto de modificações no editor dado que se baseia num protocolo de acordo em duas fases. Será necessário que o editor conheça todos os assinantes activos (utilizando vistas de grupo) e seja capaz de confirmar a recepção da mensagem. Apesar de o desempenho do sistema ficar necessariamente afectado, visto que os assinantes mais rápidos terão de esperar pelas respostas dos mais lentos, a solução poderá servir um conjunto de aplicações mais exigentes.

Outro aspecto a considerar é a cadência de mensagens entre cada vista. Se o canal trocar a informação da filiação com alguma frequência, o processo de recuperação dos assinantes poderá ser optimizado. Uma vez que, as mensagens acerca das quais existe incerteza de entrega estão contidas entre duas vistas consecutivas, a recuperação dos assinantes será mais eficiente se o total de mensagens incertas for reduzido, ou seja, por outras palavras, se as vistas mudarem com frequência. No limite, se a vista mudar a cada mensagem enviada para o canal, teremos o equivalente a uma semântica de ordem total. O balanço entre o tamanho de cada "saco" de mensagens e a quantidade de pontos de sincronização depende naturalmente, quer do número de membros do canal, quer do volume de mensagens publicadas.

3.6 Acerca dos Eventos (mensagens)

Regra geral, os sistemas de difusão em grupo baseiam-se nas primitivas de comunicação oferecidas e tipos de semânticas disponíveis. Estas por sua vez utilizam a informação transmitida em cada um dos eventos para assegurar capacidades de tolerância a faltas. Frequentemente encontramos sistemas que incluem números de ordem ou estampilhas de tempo nas mensagens enviadas. Como vimos, dadas as características da nossa solução de persistência, os eventos disseminados no canal não podem ser opacos, uma vez que tanto os assinantes persistentes com os membros do arquivo utilizam a informação do editor e a sequência da mensagem para iniciarem o processo de sincronização automaticamente. Se for utilizado o mecanismo de recuperação baseado na navegação pelo conteúdo do arquivo, poderemos extender estas referências a outros tipos de campos reconhecidos pelos assinantes, que podem ou não ser oferecidos pelo próprio canal de mensagens, tais como o assunto ("subject") da mensagem, informação de prioridade, data/hora de envio, etc.

As modificações propostas para responder às especificações referidas consistem em introduzir em cada mensagem enviada os seguintes campos:

1. identificação unívoca do editor (obtida, por exemplo, a partir do seu endereço lógico dentro da rede);

2. um contador sequencial que possibilite ordenar uma mensagem em relação às restantes (calculado, por exemplo, a partir de um relógio físico para evitar a duplicação de mensagens).

Estas referências, apesar de serem fornecidas pelos editores, não requerem nenhum tipo de modificação especial nestas entidades - apenas significa que as mensagens enviadas deverão obedecer a determinados critérios.

Na próxima secção descrevemos um protótipo da solução proposta para um canal de mensagens persistentes utilizando o paradigma do editor-assinante. Os mecanismos de persistência foram introduzidos ao sistema iBus considerando algumas das técnicas apresentadas atrás. Em seguida será apresentada uma análise do desempenho do sistema considerando a solução com e sem suporte para persistência.

Capítulo 4

Protótipo “p-iBus”

O conceito de “Orientação por Objectos” tem-se revelado particularmente útil no contexto da concepção e concretização de programas de elevada dimensão. Este método de programação conduz a sistemas com uma arquitectura derivada dos objectos que são manipulados e não das funções a executar. Uma das principais vantagens deste paradigma é a sua modularidade desde que aplicada correctamente. Os módulos, além de reduzirem um problema complexo a vários problemas mais simples, tornam-se reusáveis e adaptáveis. Tais características, revelam-se particularmente interessantes no contexto dos sistemas distribuídos já que estes tendem a tornar-se demasiado complexos.

A estrutura de classe de um sistema ou linguagem orientada por objectos traduz a existência de relações entre os vários objectos (classes) desse sistema ou linguagem, das quais as mais importantes são as relações de herança e associação. Estes conceitos facilitam o desenvolvimento de aplicações já que permitem ao programador estender as capacidades de cada objecto (por exemplo, utilizando herança) de modo a obter as funcionalidade pretendidas sem a necessidade de reescrever código adicional.

4.1 O sistema iBus

O iBus [15] é um sistema de mensagens relativamente recente que utiliza o Java como linguagem de desenvolvimento, o que, para além de oferecer maior portabilidade relativamente ao sistema operativo, aproveita as facilidades desta linguagem para conceber de aplicações que comunicam entre si utilizando o protocolo IP. O iBus é na verdade uma plataforma para o desenvolvimento de aplicações orientadas por objectos utilizando canais de mensagens baseados em eventos suportados em canais TCP ou grupos de difusão IP. O iBus oferece uma API intuitiva, utilizando o modelo editor-assinante, a qual pode ser facilmente estendida com novas qualidades-de-serviço. De entre as várias funcionalidades de comunicação oferecidas encontram-se a comunicação ponto-a-ponto, difusão em grupo (bloqueante e não-bloqueante) e modelo pedido-resposta. Relativamente às funcionalidades de coordenação destacam-se a detecção de falhas, gestão de membros e tolerância a faltas.

O sistema oferece transparência à localização permitindo que as aplicações sejam transferidas de uma máquina para outra sem afectar os restantes membros do canal. A arquitectura do iBus não possui um ponto de falhas único e não requer nenhum serviço complementar, como por exemplo, serviços de nomes ou páginas-amarelas.

Estas foram as principais razões pelo qual o iBus foi escolhido como plataforma para o desenvolvimento da solução de persistência apresentada nesta dissertação. Dado que se trata de um sistema de canais de mensagens voláteis que utiliza protocolos de difusão modernos, pareceu-nos o mais adequado para adicionar às mensagens transmitidas no canal algumas das propriedades de durabilidade já referidas. Por outro lado, graças à sua modularidade, portabilidade e semânticas de comunicação oferecidas o iBus permite-nos experimentar as técnicas de persistência sugeridas sem nos preocuparmos com os pormenores de concretização do sistema.

O protótipo desenvolvido, a que chamámos “persistent iBus”, ou simplesmente “p-iBus”, foi construído expandindo a arquitectura de classes do sistema iBus original.

4.1.1 Qualidade de Serviço (QoS)

O iBus utiliza um modelo de comunicação fácil de perceber, capaz de acomodar vários tipos de aplicações que utilizem serviços de difusão em grupo. A principal abstracção consiste no *canal de comunicação* no qual os editores publicam as suas mensagens (objectos Java), e aos quais os assinantes se juntam para receber essas mensagens. Os canais são organizados com base no seu nome, um assunto e versão da aplicação. O iBus assume a responsabilidade de filtrar e entregar os eventos publicados num determinado canal aos assinantes que subscreveram esse canal - as aplicações apenas “pagam” os serviços que “consomem”.

O iBus suporta as seguintes primitivas de comunicação em grupo:

Difusão-Assíncrona: Os eventos são transmitidos através de difusão em grupo IP, TCP ou ambos. Tipicamente, é utilizado o protocolo de difusão em grupo IP nos casos onde as aplicações funcionam sobre uma rede local como a Ethernet, de modo a aproveitar as capacidades de difusão inerentes a estas redes. Nos outros casos, os eventos são encaminhados entre redes distintas através de uma aplicação iBus utilizando TCP (“iBus TCP gateway”). O iBus permite ainda simular a difusão de eventos em grupo sobre o protocolo TCP.

Recepção-Síncrona: A recepção síncrona de eventos funciona quer sobre TCP, quer sobre difusão IP, utilizando um estilo de comunicação baseado no mecanismo pedido-resposta à semelhança de outros sistemas tais como Java-RMI ou CORBA. No entanto, ao contrário destes sistemas, o iBus oferece maior tolerância a faltas já que permite que um pedido seja enviado e executado por mais de um assinante.

O mesmo tipo de primitivas pode ser oferecido com diferentes qualidades de serviço de modo a satisfazer um vasto leque de aplicações com diferentes necessidades. Por exemplo, para satisfazer aplicações onde o débito seja o factor determinante é possível conceber um canal que não seja capaz de prevenir a duplicação de mensagens, favorecendo o desempenho do sistema; em contrapartida também é possível introduzir mecanismos de confirmação para evitar a retransmissão de mensagens para todos os assinantes do canal. Outros exemplos são a criptografia de mensagens para sistemas onde prevaleça a segurança do canal ou a fragmentação dos dados no caso de sistemas que necessitem de transmitir grandes blocos de informação. 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.

Esta será a estratégia seguida para concretizarmos o serviço de de canais de mensagens persistentes descrito no capítulo anterior. 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.

4.2 Os Componentes do iBus

Para publicar mensagens no canal, os editores deverão compor o protocolo de comunicações baseado nas diversas primitivas oferecidas pelo sistema. Em seguida, deverão "empacotar" o objecto a transmitir (*post*) e "empurra-lo" para o canal através da pilha de protocolos inicialmente composta. As aplicações assinantes funcionam de um modo muito idêntico - quando recebem um novo evento, é invocado o método `dispatch-Push` a partir do qual o objecto recebido é processado. Os editores e os assinantes deverão utilizar a mesma qualidade de serviço para compor o protocolo de comunicações.

Os componentes do iBus são essencialmente:

- **URL (Uniform Resource Locator):** Representa o canal de comunicações através do qual as aplicações enviam e recebem as mensagens, no formato

$$ibus : // < IP\,address > [:< port >] < subject >$$

  Os canais iBus podem ser automaticamente criados pelo sistema a partir do assunto e versão da aplicação ou, pode ser especificado o endereço de destino nos casos onde se pretenda utilizar comunicação ponto-a-ponto. Ex:

```
iBusURL url = iBusURLFactory.create("Tick-Channel", "1.5", "/msg");
iBusURL url = new iBusURL("ibus://224.118.34.37/Tick-Channel/1.5/msg");
```

- **Pilha de Protocolos:** Representam a QoS[1] do canal de mensagens, composta por um lista linear de pequenos objectos que em conjunto garantem as semânticas oferecidas pelo canal, tais como difusão fiável, sincronia virtual ou ordenação de mensagens (ver Figura 4.1[2]).

  Se necessário, cada aplicação pode utilizar mais do que uma pilha de protocolos (em inglês "stack") para enviar ou receber mensagens no mesmo canal. Ex:

```
Stack stack = new Stack("Reliable");
```

[1]Qualidade de Serviço (Quality of Service

[2]Imagem obtida em [15], usada com permissão do autor.

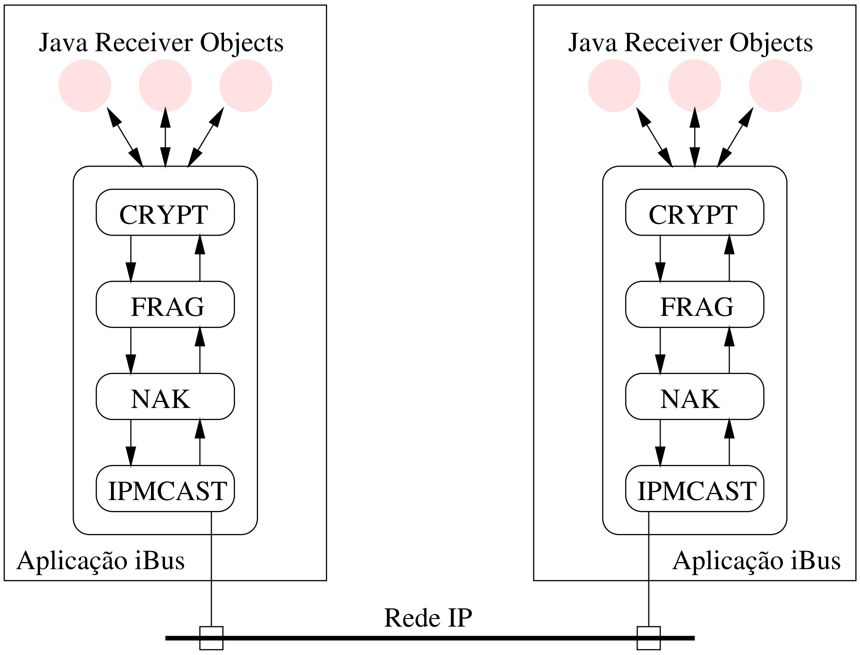

Figura 4.1: Exemplo de comunicação entre duas aplicações iBus através da pilha de protocolos.

```
Stack stack = new Stack("SVS:FRAG:NAK(hbeat=3000):IPMCAST");
```

- **Comunicação no Canal:** A pilha de protocolos do iBus não define apenas a qualidade de serviço e semântica do canal, mas serve também para interagir com este. É através desta pilha que as aplicações transmitem e recebem as mensagens. A comunicação pode ser baseada no modelo editor-assinante utilizando os mecanismos para publicar/subscrever um determinado canal, ou dessiminado as mensagens utilizando comunicação ponto-a-ponto (ver Figura 4.2[3]). No primeiro caso, o fluxo de comunicação ocorre no sentido dos editores para os assinantes através de primitivas não bloqueantes. Em contrapartida a comunicação utilizando o modelo pedido-resposta ocorre nas duas direcções e o processo que efectua o pedido fica bloqueado até receber a primeira resposta - as respostas seguintes serão "si-

[3]Imagem obtida em [15], usada com permissão do autor.

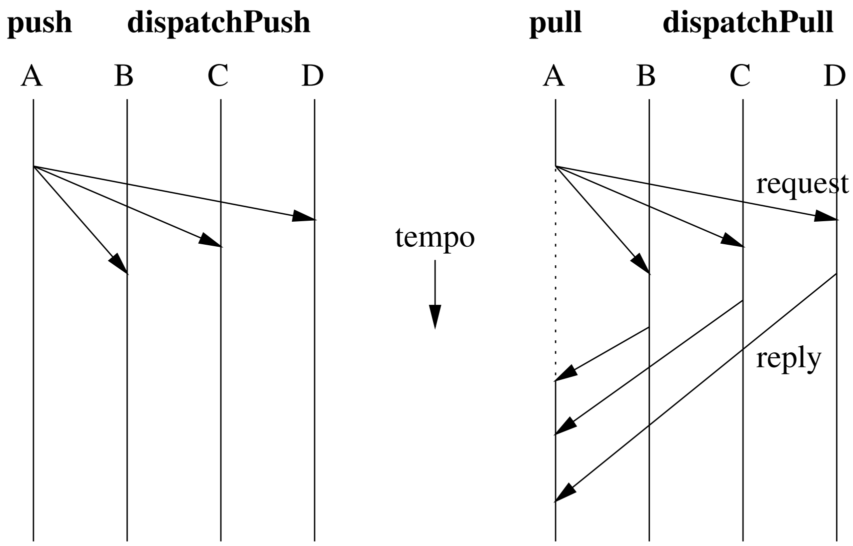

Figura 4.2: Dois modelos de comunicação oferecidos pelo iBus - difusão em grupo e pedido-resposta.

lenciosamente" descartadas.

- **Mensagens:** Todos os dados enviados ou recebidos no canal derivam da classe `iBus.Posting`, uma abstracção de uma lista dinâmica de tuplos compostos por uma cadeia de caracteres alfa-numéricos (classe `java.lang.String`) e um objecto Java (classe `java.lang.Object`). As aplicações podem construír uma mensagem colocando vários objectos nesta lista apenas especificando o seu índice ou nome. Todos os objectos devem ser instâncias dos tipos de dados fundamentais do Java ou qualquer objecto definido pelo programador que implemente o interface `java.io.Serializable` para que possa ser transformado numa cadeia de caracteres.

4.3 Concretização do Canal de Mensagens Persistentes

Sendo o iBus um sistema desenvolvido em Java que utiliza o paradigma das linguagens orientadas por objectos, procurou-se sempre que possível aproveitar as facilidades oferecidas por este novo conceito de programação. O iBus, à semelhança de outros sistemas idênticos, permite introduzir novas camadas (níveis) na pilha de protocolos utilizados, daí que idealmente, a semântica de persistência poderia ser uma opção do programador baseado na qualidade de serviço do canal. Ou seja, as aplicações que desejassem utilizar o canal persistente deveriam simplesmente adicionar à sua pilha de protocolos as respectivas camadas. Este tipo de facilidades permitiria-nos, por exemplo, desenvolver uma camada para o arquivo de mensagens capaz de interceptar as mensagens recebidas e arquivá-las em memória estável.

Porém, verificou-se que para conceber este tipo de arquitectura era necessário conhecer em detalhe alguns dos protocolos que compõem a pilha de modo a separar a mensagem enviada da informação de controle introduzida por esses protocolos. Dado que o código fonte do iBus não é do domínio público, não foi possível utilizar esta solução, optando-se por desenvolver os mecanismos de persistência ao nível da própria aplicação.

4.3.1 Diagrama de Classes

O diagrama de classes do sistema desenvolvido é apresentado na Figura 4.3 de onde se destacam a estrutura de classes do assinante persistente e arquivo de mensagens. O editor do canal consiste apenas num objecto que utiliza o canal iBus, capaz de instanciar eventos a serem transmitidos no canal. Como foi referido atrás, esta entidade não utiliza nenhuma funcionalidade do canal persistente.

Em termos gerais todas as entidades que participam no canal utilizam a pilha de

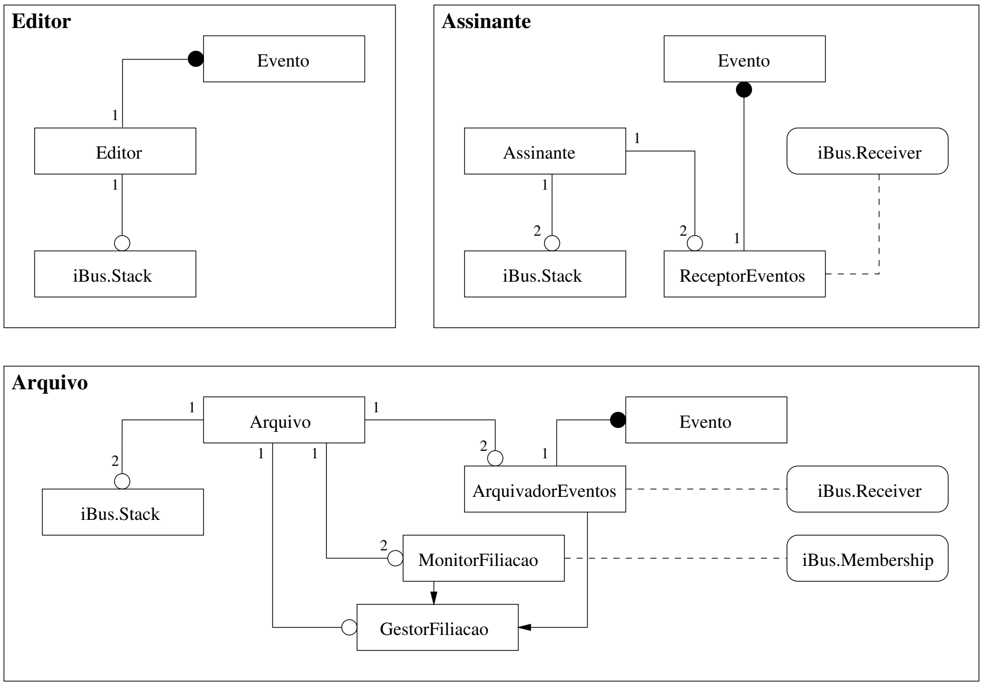

Figura 4.3: Diagrama de classes dos componentes do canal de mensagens persistente.

protocolos iBus para comunicarem entre si, enviando e recebendo mensagens (eventos). Em seguida faremos uma breve descrição das classes mais significativas utilizadas por cada um dos membros do canal persistente.

4.3.2 Classes *ReceptorEventos* e *ArquivadorEventos*

Estas classes têm como principal função receberem os eventos transmitidos no canal de mensagens. Cada entidade instância dois objectos desta classe para cada um dos canais V e A. Estas classes são idênticas na sua essência embora c objecto ArquivadorEventos execute ainda funções relacionadas com a salvaguarda das mensagens recebidas em V.

Em relação ao assinante persistente, não existe qualquer distinção entre as mensagens recebidas nos canais V e A. Em qualquer dos casos o assinante assume que se tratam de mensagens que foram transmitidas por um determinado editor já que a sua identificação está contida no cabeçalho de cada mensagem.

4.3.3 Classe *MonitorFiliação*

Esta classe é utilizada para receber a informação de filiação do canal de mensagens (vistas do grupo) uma vez que implementa o interface `iBus.Membership`. Somente o arquivo de mensagens necessita de conhecer os membros dos canais V e A, e com base nesta informação actualizar um conjunto de vectores contendo a lista de assinantes persistentes e voláteis.

4.3.4 Classe *GestorFiliação*

Finalmente esta classe utiliza a informação dos vectores referidos para determinar quando um assinante persistente retornou ao canal após um falha comparando a filiação actual do canal com a filiação das vistas anteriores. Utilizando os critérios referidos acima é então possível determinar com segurança quais as mensagens em falta e a partir daí retransmiti-las pelo canal A directamente para o assinante.

4.4 O Demonstrador

O sistema desenvolvido para concretizar um protótipo consiste num canal para difusão de mensagens de informação financeira, designadas de *ticker*[4]. No protótipo consideramos também a transmissão de outros tipos de mensagens tais com o título de uma notícia ou uma mensagem com a informação da data e hora (esta não é uma mensagem especial do canal, mas apenas um simples objecto Java que varia a cada mensagem enviada). Todas as mensagens (objectos) transmitidas no canal podem ser agrupadas (descendem) num único tipo de dados a classe Ticker (ver Figura 4.4).

[4]Um ticker é uma unidade financeira, como por exemplo, a cotação de uma moeda num determinado instante.

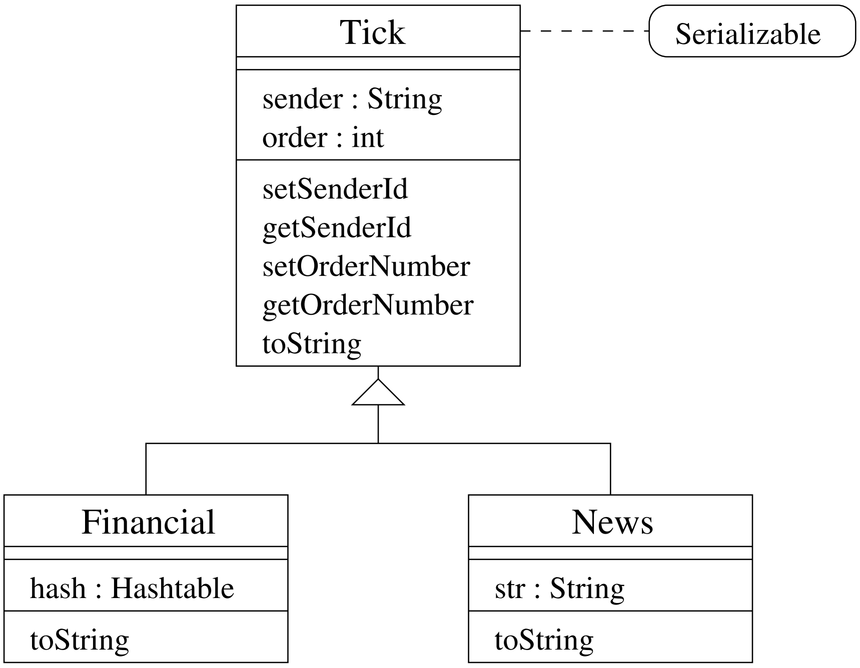

Figura 4.4: “Package Ticker”.

Para efeitos de demonstração, dado que o número de servidores disponíveis para experimentar a solução era reduzido, apenas será considerado o número médio de mensagens transmitidas no canal por intervalo de tempo utilizado dois assinantes com e sem persistência de modo a comparar a eficácia do canal volátil em relação ao canal persistente. Por esta razão, optou-se por executar apenas um processo de cada tipo, em particular:

- um editor;
- um assinante volátil;
- um assinante persistente;
- um arquivo de mensagens.

Por outro lado, visto que em funcionamento normal, a eficiência do canal depende naturalmente do sistema de mensagens utilizando (no nosso caso, o iBus) - uma

Processo	**Canal V**	**Canal A**
Editor	produz mensagens	(nenhuma função)
Assinante Volátil	consome mensagens	(nenhuma função)
Assinante Persistente	consome mensagens	consome mensagens
Arquivo	consome mensagens	produz mensagens

Tabela 4.1: Funções desempenhadas por cada um dos membros dos canais V e A.

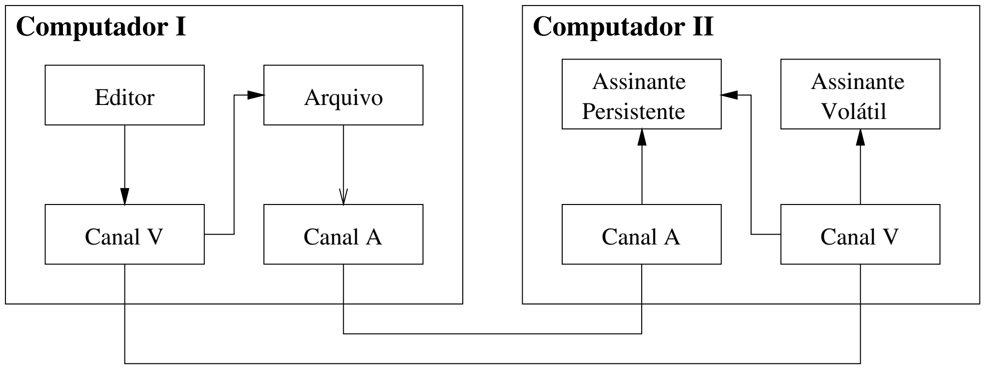

Figura 4.5: Arquitectura do sistema p-iBus.

vez que V se trata de um canal volátil - interessa apenas verificar a solução em termos de eficiência do canal para receber e arquivar todas as mensagens publicadas, assim como, recuperação de mensagens perdidas na presença de uma falha. Os aspectos relacionados com a capacidade física da rede de comunicações, assim como, do arquivo e consumidores do canal não foram considerados.

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 (ver Tabela 4.1).

Tanto o editor como assinante volátil são aplicações vulgares que utilizam um canal de mensagens iBus, razão pela qual a sua concepção não merece ser apresentada neste trabalho.

A arquitectura do protótipo desenvolvido é apresentada na Figura 4.5 e a Tabela 4.2

Processo	**Canal** V	**Canal** A
Editor	difusão	(sem comunicação)
Assinante Volátil	subscritor	(sem comunicação)
Assinante Persistente	subscritor	pedido/resposta
Arquivo	subscritor/ filiação	ponto-a-ponto/ filiação

Tabela 4.2: Semânticas de comunicação utilizadas.

resume os modelos de comunicação utilizados por cada um dos membros do canal. Dado que o iBus não possui primitivas para ordenação total de mensagens, e visto que um canal totalmente ordenado tornaria os mecanismos de recuperação propostos demasiado simples, optou-se por conceber um canal de mensagens persistentes utilizando os mecanismos de recuperação automática baseado no modelo de sincronia virtual. De todos os membros do canal persistente, apenas o arquivo necessita de observar as entradas e saídas do canal daí que utilizem a informação das vistas para detectarem quando os assinantes persistentes falham e recuperam. As vistas recebidas no canal A servem para detectar as entradas e saídas de consumidores e membros do arquivo que falham, enquanto que as vistas do canal V servem para detectar qual a última mensagem recebida antes de um assinante ou membro do arquivo falhar.

Existe um custo associado à recuperação automática utilizando o modelo de sincronia virtual. Uma vez que o arquivo utiliza a informação de filiação dos canais V e A, tem de esperar a entrada do assinante em ambos os canais - esta informação é necessária para identificar o processo como um assinante persistente. Ou seja, quando o consumidor se liga ao canal V o arquivo determina as mensagens em falta utilizando o algoritmo descrito no capítulo anterior; quando o arquivo detecta a entrada do consumidor no canal A, inicia o processo de recuperação enviando as mensagens não recebidas. Se o arquivo receber a informação de filiação na ordem inversa (primeiro a entrada do consumidor no canal A) terá de esperar que o consumidor se ligue ao canal V para calcular as mensagens não recebidas. No entanto, neste caso, se se tratar de um canal totalmente ordenado será mais eficiente se o processo de sincronização for iniciado pelo assinante persistente informando o arquivo acerca da última mensagem recebida (recuperação automática baseado na informação das mensagens recebidas pelo assinante).

4.5 Avaliação

Para avaliar o desempenho do canal baseamos-nos no débito de mensagens transmitidas e distância entre o produtor e os restantes membros do canal verificando o impacto nos consumidores com e sem persistência. Os componentes do sistema desenvolvido são:

- 1 PC, processador Pentium 166 MMX, 64 Mb RAM (S.O. Linux, kernel 2.0.34);
- 1 PC, processador Pentium Pro 200 MHz, 64 Mb RAM (S.O. Linux, kernel 2.0.34);
- rede Ethernet 100 Mbps;
- iBus^{TM} versão 0.5;
- JDK^{TM} versão 1.1.6;
- relógio externo para sincronização das máquinas via NTP[5].

Os assinantes volátil e persistente foram executados no primeiro computador, enquanto que os processos editor e arquivo de mensagens foram executados no computador mais rápido. Estas escolhas prendem-se com o facto de se pretender que as mensagens sejam enviadas do produtor para os consumidores utilizando a rede de comunicações ao invés de as mensagens serem trocadas entre duas pilhas de protocolos

[5]Network Time Protocol

executadas na mesma máquina - um cenário para o qual o iBus possui algumas optimizações. Em relação ao arquivo, dado que se trata de um processo que executa diversas operações em ambos os canais não deveria estar limitado à capacidade de processamento do equipamento. Por outro lado, um consumidor persistente que recupere de uma falha, também recebe as mensagens em falta através da rede de comunicações.

4.5.1 Resultados

Foram efectuados diversas simulações dos canais volátil e persistente, aumentando gradualmente o débito dos canais e calculando a distância entre a transmissão e a recepção dessas mensagens. O objectivo era perceber qual o impacto em termos de desempenho desta solução de persistência. Não foram considerados os tempos de recuperação do consumidor persistente, já que estes valores dependem do número de mensagens transmitidas enquanto o consumidor esteve desligado do canal. Por outro lado, tentou-se também perceber qual o impacto da solução sobre o arquivo de mensagens já que este processo necessita de gravar as mensagens recebidas e determinar as mensagens em falta para cada consumidor persistente através da informação de filiação dos canais V e A.

Nesta avaliação as mensagens enviadas pelo editor eram compostas por um objecto do tipo *ticker*, ou seja, podem conter aleatoriamente um objecto simples (*News*) ou um objecto composto (*Financial*) de modo a simular um ambiente real onde as mensagens possuem tamanhos e conteúdos variáveis. A técnica utilizada consistiu em programar o editor para debitar 100 mensagens sequencialmente suspendendo o processo entre cada mensagem durante períodos graduais de entre 1000 até 1 milisegundo e calculando no final o tempo médio para cada conjunto de 100 mensagens enviadas. Não foi possível efectuar esta simulação com tempos inferiores a 1 ms já que este é o valor mínimo para o qual se pode suspender um processo (*thread*) em Java. Existe ainda um custo associado

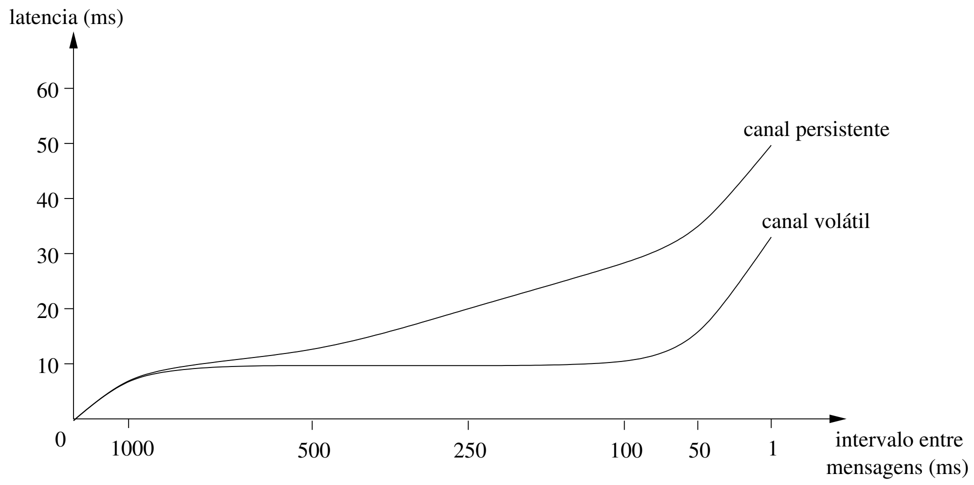

Figura 4.6: Débito máximo de difusão - relação entre os canais volátil e persistente.

ao facto de o editor ter de compor as mensagens antes de as enviar, embora este valor não possa ser mensurável. Quer os consumidores volátil e persistente, quer o arquivo de mensagens receberam e processaram correctamente todas as mensagens.

No canal volátil o sistema deixou de ser eficiente quando o editor passou a debitar mensagens a cada 50 ms. A partir deste momento a latência cresceu exponencialmente com o número de mensagens. Em contrapartida, no canal persistente, a latência das comunicações teve um crescimento linear a partir do momento que o editor começou a publicar mensagens com tempos inferiores a 1000 ms, ou seja, a degradação no desempenho do sistema ocorreu bastante mais cedo do que no canal volátil. No entanto, também neste caso o crescimento exponencial da latência só ocorreu quando o intervalo entre mensagens passou a ser inferior a 50 ms.

O facto de a latência aumentar mais prematuramente no canal persistente é uma consequência natural deste canal, uma vez que exige uma maior disponibilidade de recursos, pela necessidade de existir um canal adicional (o canal de mensagens de arquivo), e ser utilizado o modelo de sincronia virtual o que obviamente aumenta o número de mensagens transmitidas. Dado que o crescimento exponencial da latência ocorreu no mesmo ponto em ambos os canais poderemos concluir que esse crescimento não é uma consequência do canal persistente mas sim, do próprio iBus.

No gráfico apresentado na Figura 4.6 podemos observar a relação do desempenho do sistema entre os canais volátil e persistente. Comparando o desempenho de ambos os canais poderemos afirmar que o canal persistente é sensível ao volume de mensagens transmitidas, pelo que esta solução poderá não ser adequada a sistemas com um elevado número de mensagens. Quantificando esta afirmação, poder-se-á dizer que a solução não é eficiente em sistemas com um débito superior a uma mensagem por segundo. No entanto, em sistemas com um débito de transmissão inferior, esta solução verificou-se bastante eficaz. Em alternativa, poderão ser efectuadas diversas optimizações no sistema, tais como enviar vários objectos na mesma mensagem reduzindo assim a informação de controle transmitida. Outra solução consiste em aproveitar essas mensagens de controle do canal para transmitir as mensagens a publicar - técnica designada de *piggybacking*.

Os principais pontos de estrangulamento da solução de persistência apresentada resultam da utilização de um canal alternativo e da sobrecarga introduzida pelo modelo de sincronia virtual, o que aumenta o volume de mensagens a enviar em cada um dos canais V e A. Dado que esta solução de persistência foi comparada com um canal de mensagens volátil "puro", isto é, sem qualquer mecanismo para detecção de faltas seria espectável que o sistema apresentasse resultados menos animadores.

Tentou-se ainda executar a mesma simulação com o objectivo de verificar o impacto do tipo de mensagens transmitidas nesta solução de persistência. O teste efectuado consistiu em publicar no canal um conjunto de mensagens simples (*News*) e um conjunto de mensagens compostas (*Financial*) utilizando o modelo pedido-resposta ao invés de difusão em grupo. Tal como na simulação anterior foram publicadas 100 mensagens reduzindo gradualmente o intervalo de tempo entre cada mensagem. Dado que o mecanismo de comunicação utilizado pressupõe uma resposta dos assinantes, pretendia-se calcular no editor a distância (em milisegundos) entre o envio e a recepção da mesma mensagem por, pelo menos, um dos assinantes activos. No entanto, os resultadas da simulação não foram conclusivos dado que, por limitações da própria linguagem Java, não foi possível calcular com uma precisão inferior a 1 segundo a referida distância, sendo os valores obtidos muito semelhantes em ambos os canais.

4.6 Discussão

A realização do sistema de informação financeira utilizando a solução de canais persistentes permitiu-nos validar a arquitectura proposta nesta dissertação para a concretização de canais de mensagens persistentes. Globalmente, os resultados são muito animadores embora existam alguns problemas de eficiência na solução de persistência devido ao facto de o iBus ser desenvolvido em Java. A experiência com esta arquitectura permite também identificar algumas limitações do sistema iBus.

Em primeiro lugar, o modelo de composição da pilha do iBus não permite introduzir novas camadas à pilha de comunicação sem acesso ao código fonte. Isto pode ser uma limitação para utilizadores que pretendam refinar a arquitectura através da criação de novas camadas. Este problema refere-se ao facto de todas a operações para tornar as mensagens persistentes, assim como detectar e transmitir as mensagens não recebidas pelo consumidor serem executadas ao nível da aplicação. Se possível, a solução de persistência deverá ser fornecedida por outros níveis do protocolo diminuindo o tempo necessário para transportar a mensagem pela pilha, assim como para simplificar a criação de um arquivo e consumidor persistente.

Para além disso, apesar de possuir suporte para comunicação em grupo, na versão disponível este suporte é ainda bastante rudimentar. Em particular, os mecanismos de difusão em sincronia virtual e ordenação total não se encontram concretizados. Isto, não sendo impeditivo para desenvolver um protótipo que demonstra a exequibilidade da solução apresentada, representa uma severa limitação para utilização em ambientes operacionais com requisitos de fiabilidade.

Por outro lado, o protótipo do arquivo recorre a diversos vectores para organizar a informação das vistas de ambos os canais e as mensagens transmitidas entre cada vista. Dado que este tipo de objectos exerce uma carga adicionar sobre o arquivo de mensagens, o processo de recuperação é mais lento.

Verificou-se ainda que ao reduzir o tempo entre duas vistas consecutivas no arquivo de mensagens de modo a tornar a detecção de falhas mais eficaz e simultaneamente diminuir o número de mensagens a transmitir em caso de falha de um consumidor, todo o sistema se torna demasiado lento devido à quantidade de mensagens com informação de controle que é necessário transmitir. O iBus baseia a detecção de falhas num tipo de mensagem que é trocado periodicamente entre os membros do canal com base em dois parâmetros configuráveis[6]. Estas propriedades são importantes para um correcto funcionamento do sistema de detecção de falhas, daí que devem ser efectuados alguns ajustes nestes parâmetros dependendo do volume de mensagens do canal. Nas simulações de persistência realizadas, dado que se tratava de um canal com um elevado débito de transmissão, os valores utilizados foram relativamente elevados de modo a diminuir o volume de mensagens referentes as vistas do grupo. Apesar de necessárias, estas modificações possuem o inconveniente de fazer aumentar o número de mensagens entre vistas, tornando mais lento o sistema de recuperação de assinantes, uma vez que também aumenta o número de mensagens a receber.

O iBus apresenta ainda outro problema relacionado com a identificação dos mem-

[6]Quando é utilizado um canal fiável, um dos níveis da pilha de protocolos baseia a detecção de falhas numa mensagem designada de "heartbeat" e num tempo máximo (*timeout*) para determinar se um membro está falhado. Se um *heartbeat* de um membro do canal não for recebido durante o tempo máximo previamente estabelecido, o iBus assume que esse membro está falhado.

bros do canal quando se aumenta demasiado os valores do *heartbeat* e do *timeout* em canais que utilizam o modelo pedido-resposta. Uma vez que a informação de filiação do canal é recebida demasiado tarde, os membros do canal são identificados quando o editor recebe a primeira resposta, pelo que a latência da primeira mensagem é sempre bastante superior às mensagens seguintes.

No entanto, julga-se que algumas estas limitações poderão vir a ser corrigidas no futuro. Durante o período de desenvolvimento desta dissertação continuaram a ser concretizados desenvolvimentos ao iBus, pelo que se prevê um amadurecimento deste produto. É importante assinalar que estão anunciadas extensões de persistência para as próximas versões do iBus embora, tanto quanto sabemos, estas venham a ser concretizadas num modelo baseado num servidor centralizado.

Capítulo 5

Conclusões

Esta dissertação propôs uma arquitectura de suporte ao modelo editor-assinante que integra os canais de mensagens voláteis com canais persistentes. Esta arquitectura permite que, após recuperação, os consumidores de um canal possam obter a sequência de mensagens transmitidas para o canal enquanto estiveram falhados ou inactivos.

A solução proposta consiste na criação de um arquivo para o armazenamento persistente de mensagens. Um arquivo pode ser composto por um ou vários assinantes do um canal, do mesmo modo que vários canais persistente podem utilizar o mesmo arquivo. O arquivo armazena as mensagens assincronamente, fora do caminho crítico das aplicações que não necessitam de persistência. Outro dos aspectos importantes da soluções refere-se à transparência do arquivo quanto à sua localização e utilização.

Foi desenvolvido um protótipo usando o sistema iBus. O protótipo permitiu demonstrar a exequibilidade da arquitectura proposta. O sistema iBus revelou-se uma escolha adequada, dada a portabilidade da mesma, o facto de a distribuição binária ser gratuita e da facilidade de desenvolvimento sobre as interfaces oferecidas. A experiência permitiu também identificar algumas limitações da versão do iBus utilizada, em particular no que se refere a dificuldade de desenvolver novos protocolos para acrescentar à pilha de comunicação.

A concretização foi avaliada através de uma aplicação de demonstração que simula a difusão de informação financeira. Num protótipo de pequena dimensão foi possível verificar que o canal persistente desenvolvido é até dez vezes mais lento do que um canal volátil "puro", ou seja, sem qualquer mecanismo para detecção de falhas. No entanto, este valor só é atingido em sistemas com um elevado débito de mensagens, não comprometendo a eficácia da solução.

Este trabalho levantou algumas questões interessantes que poderiam ser exploradas no futuro:

- Seria interessante acrescentar mecanismos de reciclagem automática de memória nos arquivos persistentes, recolhendo para isso informação de quais os assinantes que já receberam as mensagens.

- Os mecanismos de reconciliação após partições na rede poderiam ser melhorados de modo a minimizar o número de ligações ponto-a-ponto a estabelecer. Em particular, quando mais que um consumidor ou arquivo necessitam de ser actualizados, poderia ser utilizada comunicação em grupo para realizar esta actualização.

- A dissertação não aborda a publicação atómica de uma sequência de eventos, organizada sobre a forma de uma transacção. Este seria também um aspecto onde o sistema poderia ser extendido.

- Poderiam ser criados grupos de arquivos dedicados a cada editor do canal de modo a optimizar o armazenamento das mensagens, assim como o número de arquivos que participam nos processos de reconciliação.

Para terminar, importa sublinhar que a utilização de comunicação em grupo facilitou bastante a introdução transparente da persistência do canal, revelando-se um mecanismo particularmente adequado para suportar o modelo editor-assinante.

Os protótipos desenvolvidos para demonstração da solução de persistência podem ser obtidos em http://www.di.fc.ul.pt/˜ler/students/jorgemorgado.html.

Bibliografia

[1] Level 8. Hiperligação: http://www.level8.com/falcon.htm, 1999.

[2] K. Birman. *Building Secure and Reliable Network Applications*. Manning Publications Co., 1996.

[3] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, February 1987.

[4] K. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. New York: IEEE Computer Society Press, 1994.

[5] A. Birrel and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, February 1994.

[6] M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Transactions on Computer Systems*, 2(3):251–273, August 1984.

[7] D. Cheriton and W. Zwaenepoel. Distributed process groups in the V-kernel. *ACM Tran. on Computer Systems*, 3(2), May 1985.

[8] Microsoft Corporation. A guide to reviewing microsoft message queue server release 1.0 - the message queuing service of windows nt, microsoft message queuing services. 1997.

[9] Microsoft Corporation. Optimizing performance in a microsoft message queue server environment. 1998. Microsoft White Paper.

[10] D. Deering. RFC 1112: Host extensions for IP multicasting, August 1989.

[11] R. Friedman and R. van Renesse. Strong and weak virtual sincrony in horus. In *1996 IEEE Symposium on Reliable Distributed Systems*. IEEE Press, October 1996.

[12] J. Gray and A. Reuter. *Transaction processing: concepts and techniques*. Series in Data Management Systems. Morgan Kaufmann, 1993.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), July 1978.

[14] R. Lemos and P. Veríssimo. Confiança no funcionamento: Proposta para uma terminologia em português. 1989. Publicação conjunta INESC e LCMI/UFSC.

[15] S. Maffeis. ibus - the java intranet software bus. February 1997. SoftWired AG.

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

[17] B. Oki, M. Pfluegl, A. Siegel, and D. Skeen. The information bus - an architecture for extensible distributed systems. In *Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles*, pages 58–68. New York: ACM Press, December 1993.

[18] OMG. The Common Object Request Broker: Architecture and Specification, 1997.

[19] OMG. CORBAservices: Common Object Services Specification, 1997.

[20] J. Postel. RFC 793: Transmission control protocol, September 1981.

[21] M. Spiteri and J. Bates. An architecture to support storage and retrieval of events. In *Proceedings of MIDDLEWARE 1998*, Lancaster, UK, September 1998. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing.

[22] S. Steinke. Middleware meets the network. *LAN: The Network Solutions Magazine 10*, (13):56, December 1995.

[23] R. van Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.