IndiQoS: um Sistema Publicação-Subscrição com Qualidade de Serviço

Nuno Carvalho
Filipe Araújo
Luís Rodrigues

DI–FCUL TR–03–13

Julho 2003

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

IndiQoS: um Sistema Publicação-Subscrição com Qualidade de Serviço*

Nuno Carvalho
Universidade de Lisboa
nunomrc@di.fc.ul.pt

Filipe Araújo
Universidade de Lisboa
filipius@di.fc.ul.pt

Luís Rodrigues
Universidade de Lisboa
ler@di.fc.ul.pt

Resumo

Os modelos de comunicação indirecta como os sistemas publicação-subscrição têm emergido como uma alternativa ao modelo de comunicação directa, ou seja, pedido-resposta. A principal vantagem do paradigma de comunicação indirecta consiste no desacoplamento entre participantes, que não necessitam de ter conhecimento quer da localização, quer do número de elementos existentes no sistema.

Uma das limitações presentes na maior parte das arquitecturas que suportam comunicação indirecta do tipo publicação-subscrição consiste na falta de suporte à expressão de parâmetros de Qualidade de Serviço (QdS), como a largura de banda ou a latência. As soluções de garantias de QdS existentes são baseadas em sistemas de comunicação directa, onde se estabelecem canais de comunicação entre emissores e receptores. Esta abordagem é de difícil utilização no modelo publicação-subscrição.

Assim, este artigo apresenta uma nova arquitectura de um sistema de comunicação indirecta – o IndiQoS *– seguindo o modelo publicação-subscrição, que fornece garantias de QdS às aplicações. Este modelo deverá fornecer às aplicações uma forma de expressão de parâmetros de QdS e remover da aplicação a tarefa de efectuar reservas dos recursos necessários. As reservas a efectuar necessitam de ser baseadas em informação dinâmica, como a localização, número e características dos participantes, assim como das características da informação que está a circular no sistema, tendo o sistema de publicação-subscrição a tarefa de processar toda esta informação de forma automática.*

Palavras Chave: Sistemas Distribuídos, Qualidade de Serviço, Middleware.

*Este trabalho foi parcialmente suportado pela FCT, através do projecto INDIQoS (POSI/-CHS/41473/2001). Partes deste relatório foram publicadas nas actas da 6ª Conferência sobre Redes de Computadores, Bragança, Portugal.

1 Introdução

Os modelos de comunicação por eventos e, em particular, o modelo publicação-subscrição, que segue um paradigma de comunicação indirecta, têm emergido nos últimos anos como uma alternativa ao modelo de comunicação pedido-resposta como a invocação remota de procedimentos. Nestes modelos, existem dois tipos de participantes: os *editores*, que efectuam a publicação de notificações, e os *assinantes* que efectuam a subscrição de notificações. A principal vantagem do paradigma de comunicação indirecta consiste no desacoplamento entre participantes, que não necessitam de ter conhecimento quer da localização, quer do número de elementos existentes no sistema. Esta propriedade não só permite que certos tipos de reconfigurações às aplicações possam ser feitas "em tempo de execução", i.e., sem que haja paragens, como simplifica a reutilização de componentes de software noutras aplicações, aumentando assim a modularidade do sistema.

Uma das limitações presentes na maior parte das arquitecturas que suportam comunicação indirecta do tipo publicação-subscrição consiste na falta de suporte à expressão de parâmetros de Qualidade de Serviço (QdS), como a largura de banda ou a latência. Esta limitação aplica-se a vários modelos, como o CORBA Event Service [12], o CORBA Notification Service [11], o Java Message Service [17], assim como a sistemas como o Cambridge Event Architecture (CEA) [1], o Scalable Internet Event Notification Architecture (SIENA) [5] ou o Hermes [13]. A falta de suporte à expressão de parâmetros de QdS neste tipo de sistemas é uma desvantagem importante, já que certo tipo de aplicações necessita de garantias de QdS para o seu bom funcionamento. Por exemplo, se uma arquitectura publicador-subscritor for usada para suportar um grupo de difusão, este grupo pode necessitar de ter garantias de largura de banda mínima entre os seus elementos. Outro exemplo ainda pode ser dado num caso em que um sistema publicador-subscritor é usado para disseminar actualizações do estado de um jogo multi-utilizador. Neste caso existem requisitos de latência mínima que necessitam de ser cumpridos.

Por outro turno, as soluções que visam oferecer garantias de QdS actualmente em uso ou em estudo não podem ser aplicadas directamente aos sistemas de publicação-subscrição, por serem essencialmente dirigidas aos sistemas de comunicação directa [4, 3, 2, 19]. Os sistemas que fornecem garantias de QdS são tipicamente baseados no estabelecimento explícito de canais ou ligações, os quais servem de suporte para efectuar as reservas dos recursos necessários para fornecer a QdS pretendida. Esta abordagem encaixa naturalmente nos sistemas de comunicação directa, onde as ligações entre elementos de um sistema são explicitamente efectuadas, mas é de difícil utilização no modelo publicação-subscrição. Neste modelo, as aplicações não devem ser forçadas a estabelecer canais de comunicação e não devem precisar de ter conhecimento acerca da localização e do

número de participantes envolvidos na comunicação. Devem apenas preocupar-se com a qualidade da informação que manipulam.

Assim, este artigo apresenta uma nova arquitectura de um sistema de comunicação indirecta, seguindo o paradigma publicação-subscrição, que fornece garantias de QdS às aplicações. Este modelo permite: (*i*) fornecer às aplicações uma forma de exprimir os parâmetros de QdS pretendidos e (*ii*) remover da aplicação a tarefa de reservar recursos. As reservas a efectuar necessitam de ser baseadas em informação dinâmica, como a localização, número e características dos participantes, assim como das características da informação que está a circular no sistema. Toda a informação necessária para manter o sistema deverá ser processada de forma automática pelo próprio sistema de publicação-subscrição, delegando às aplicações apenas a tarefa de processar as notifacações disseminadas.

Este artigo está estruturado da seguinte forma: a Secção 2 descreve de uma forma geral quais os requisitos de um sistema publicação-subscrição que fornece garantias de QdS, a Secção 3 apresenta a arquitectura do sistema *IndiQoS*, a Secção 4 descreve o primeiro protótipo da arquitectura, a Secção 5 apresenta direcções futuras e a Secção 6 conclui este artigo.

2 Sistemas Publicação-Subscrição com QdS

Uma das grandes vantagens dos sistemas publicação-subscrição consiste no desacoplamento entre os participantes, no tempo, no espaço, e ainda no fluxo de dados [8]. Este trabalho apresenta ainda uma quarta dimensão no desacoplamento entre participantes: o desacoplamento na QdS. Este desacoplamento separa os parâmetros de QdS do tipo e conteúdo das notificações disseminadas no sistema. De seguida são apresentadas as características do modelo proposto.

As reservas necessárias para a disseminação com QdS devem ser efectuadas em tempo de execução, baseadas nas propriedades definidas pelos assinantes, nas propriedades das notificações anunciadas pelos editores e nos recursos disponíveis. Um aspecto importante a considerar no sistema é que os editores e os assinantes devem poder expressar os seus requisitos de QdS usando uma notação idêntica à que usam para definir outro tipo de requisitos, nomeadamente requisitos relacionados com o conteúdo das notificações. Para isso, editores e assinantes devem anunicar o padrão de tráfego que pretendem, respectivamente, gerar ou receber na forma de um *perfil de QdS*. Esta informação é usada em tempo de execução pelo sistema para estimar a quantidade de recursos necessários para um determinado fluxo de notificações e para verificar se os pedidos dos assinantes podem ser satisfeitos.

Para exemplificar como deverão ser efectuadas as publicações e subscrições no sistema apresentado, considere-se um edifício onde as várias divisões estão

equipadas com sensores de temperatura. Estes sensores anunciam a temperatura da sua divisão numa notificação do tipo *Temp*, com os seguintes atributos: (*i*) *sala*, que indica onde está a ser recolhida a temperatura, (*ii*) *temperatura*, que indica qual a temperatura da sala aquando da disseminação da notificação e (*iii*) *precisão*, que indica a precisão do sensor. Nos exemplos seguintes será usada uma notação baseada em [7]. Usando este modelo, as subscrições podem ser feitas do seguinte modo:

Subscriber s = **subscribe** *Temp*
where (sala = "sala1")

ou

Subscriber s = **subscribe** *Temp*
where (temperatura > 60)

A primeira expressão corresponde a uma subscrição de notificações provenientes da sala "sala1" e a segunda corresponde a uma subscrição de notificações de qualquer sala, em que a temperatura excede os 60 graus. As publicações podem ser definidas da seguinte forma:

Publisher p = **new** *Publisher*
of *Temp*
withProfile (sala="sala1",
temperatura=any,
precisão=0.01);

e = **new** *Temp* (sala="sala1",
temperatura=16,
precisão=0.01)

p.publish (e)

Publisher consiste num componente auxiliar que é usado para disseminar notificações. Permite também informar o sistema sobre o tipo de notificações a produzir. Esta informação toma a forma de anúncios no sistema. No exemplo anterior é apenas considerado o perfil do conteúdo da informação que vai ser disseminada. Esta informação pode ser usada pelo sistema para optimizar a disseminação de notificações [6]. De seguida será discutida uma forma de anunciar informação sobre QdS em conjunto com a informação sobre o tipo de notificação.

Considere-se agora que cada um dos sensores referidos tem diferentes QdS's associadas, por exemplo, um sensor dissemina notificações esporadicamente, apenas quando a temperatura atinge um determinado valor e outro sensor dissemina a temperatura periodicamente. A informação sobre QdS deverá ser incluída quer nos anúncios, quer nas subscrições. A adição de mecanismos que permitem às aplicações expressar parâmetros de QdS pode ser efectuado de várias formas.

Uma aproximação possível consistiria em codificar a informação sobre a QdS no tipo de notificação. Por exemplo, se um sensor disseminasse *esporadicamente* uma notificação com a informação da temperatura numa determinada sala e outro sensor disseminasse *periodicamente* uma notificação também sobre a temperatura de uma determinada sala, poderiam ser criados dois subtipos, um para disseminação esporádica e outro para periódica. Apesar de intuitiva, esta aproximação teria algumas desvantagens. Com vários atributos de QdS, esta aproximação levaria a uma explosão de tipos diferentes para a mesma informação. Outra razão para rejeitar esta aproximação consiste também no facto de só ser possível determinar alguns atributos de QdS em tempo de execução, como por exemplo a latência. A latência depende da localização do assinante em relação ao editor e da carga dos nós intermédios numa determinada altura.

Assim, tendo em conta os aspectos anteriores, este artigo propõe uma arquitectura em que as operações de anúncio e subscrição são enriquecidas com atributos de QdS, que podem ser definidos de forma semelhante à definição de conteúdos de informação. Para tal, o editor terá de anunciar um *perfil* do tipo de informação que vai ser disseminada e também um perfil da QdS necessária para disseminar as notificações. As operações deverão ser da seguinte forma:

Publisher p = **new** *Publisher*
of *Temp*
withProfile (sala="sala1",
temperatura=any,
precisão=0.005)
withQoSProfile *Sporadic*

ou

Publisher p = **new** *Publisher*
of *Temp*
withProfile (sala="sala1",
temperatura=any,
precisão=0.01)
withQoSProfile *Periodic (periodo = 1)*

ou ainda

Publisher p = **new** *Publisher*
of *Temp*
withProfile (sala="sala1",
temperatura=any,
precisão=0.01)
withQoSProfile *Periodic (periodo = 10)*

É de notar ainda que alguma informação, como o período das notificações, ajuda a caracterizar a forma do tráfego que vai ser produzido pelo editor. Desta forma consegue-se também uma separação entre o tipo de informação e a QdS necessária para a sua disseminação.

O assinante, por seu lado, deverá expressar os requisitos de QdS usando uma condição de filtragem semelhante à que é usada para o conteúdo:

Subscription s = **subscribe** *Temp*
where (temperatura > 60)
withQoS ((*Periodic(periodo<1)*)
and (*latencia<10*))

Note-se que esta informação deverá ser interpretada pelo sistema de forma a efectuar não só as correspondências de informação, mas também as reservas de recursos necessárias. Note-se também que algumas subscrições podem não ser aceites por falta de recursos disponíveis. Em suma, deverá ser o sistema a efectuar as ligações e as reservas de recursos de forma transparente para a aplicação.

3 Arquitectura do Sistema

3.1 Aspectos a Considerar

Esta secção apresenta uma nova arquitectura – o *IndiQoS* – que visa oferecer as funcionalidades descritas na secção anterior. Na concepção da arquitectura foram considerados os seguintes aspectos: (*i*) a escalabilidade do sistema, (*ii*) os parâmetros de QdS a suportar e (*iii*) a infraestrutura de suporte à QdS (por exemplo rede IP e protocolo RSVP [3]).

Escalabilidade Uma arquitectura baseada na existência de um servidor centralizado é inerentemente não escalável. Por este motivo, estamos interessados em desenvolver arquitecturas descentralizadas. Nos parágrafos seguintes, descrevemos de modo sumário, três exemplos representativos dos vários tipos de arquitecturas distribuídas e descentralizadas para suportar o paradigma publicação-subscrição, a saber: (*i*) o CEA, que pode funcionar sem encaminhadores de notificações;

(*ii*) o SIENA, que inclui encaminhadores de notificações em hierarquia ou numa estrutura entre-pares; e (*iii*) o Hermes [13], que inclui um sistema puramente entre-pares de encaminhadores de notificações, que tal como outros sistemas (por exemplo, SCRIBE [15]) se baseia no Pastry [14], um sistema de encaminhamento e alocação de objectos em redes entre-pares de larga escala.

CEA é um sistema de comunicação assíncrona que segue o paradigma *publish-register-notify*. O editor começa por anunciar que tipo de notificações vai publicar e, após receber subscrições, envia notificações assíncronas para os assinantes interessados. A filtragem é normalmente feita pelos assinantes, mas pode também existir um mediador, cuja função consiste precisamente em aliviar os assinantes do processamento de filtragem de notificações. Note-se que na ausência de mediadores, existe uma árvore de disseminação por cada publicação, gerida pelos próprios editores, que dificilmente poderá ser partilhada com outro tipo de notificações ou com outros editores. Isto significa que o CEA apresenta limitações importantes no que diz respeito às possibilidades de efectuar optimizações dos recursos utilizados. No entanto, talvez a limitação mais decisiva advenha do facto do CEA (na versão sem mediador) não possuir todas as características de desacoplamento dos outros sistemas publicação-subscrição, uma vez que editores e assinantes comunicam directamente entre si. Isto impede, por exemplo substituições "em tempo de execução" de um editor.

SIENA consiste num serviço de notificação de eventos e é composto por editores, assinantes e por um conjunto de encaminhadores, também designados por servidores. Os servidores encontram-se interligados entre si por um protocolo "entre-servidores" e os clientes ligam-se através de um protocolo "cliente-servidor". Os servidores do SIENA podem ser dispostos de forma hierárquica, numa rede não hierárquica entre-pares ou ainda numa configuração híbrida. Uma importante desvantagem do SIENA reside na sua difícil escalabilidade, uma vez que todos os anúncios terão de ser difundidos por todos os encaminhadores de notificações do sistema, de forma a ser possível encaminhar subscrições para montante[1].

Hermes é um sistema publicação-subscrição baseado em tipo (i.e., as notificações são tipificados e a subscrição das notificações é feita primeiramente pelo tipo, sendo também possível adicionar critérios de filtragem) constituído por três entidades: editores, assinantes e encaminhadores de notificações. Os encaminhadores de notificações comunicam através de uma rede

[1]Note-se que para reduzir o problema é possível efectuar fusões entre anúncios, sempre que haja uma relação de inclusão entre estes. O mesmo se passa com as subscrições.

lógica sobre a rede normal IP. É sobre essa rede lógica que circulam as notificações e as mensagens de controlo. Estas últimas têm como objectivo distribuir anúncios de publicações e subscrições. Quando um editor ou um assinante deseja ligar-se ao sistema, estabelece uma ligação com um dos encaminhadores. Para cada tipo de notificação, um dos encaminhadores é deterministicamente nomeado o *rendez-vous point* — esta possibilidade é inerente ao próprio sistema entre-pares Pastry que serve de suporte ao Hermes. Desta forma, quando um assinante deseja subscrever um determinado tipo de notificação, comunica com o *rendez-vous point* para que seja definido um caminho entre o editor e o(s) assinante(s). O caminho é sempre definido sobre a rede lógica construída inicialmente. Uma vantagem desta arquitectura consiste no facto de ser escalável. Em contrapartida a complexidade de concretização é elevada e deve-se essencialmente ao número de funções delegadas aos encaminhadores de notificações.

Dadas as razões apontadas, o modelo a adoptar para o desenvolvimento do *IndiQoS* será baseado numa arquitectura entre-pares escalável.

Parâmetros de QdS Outro dos aspectos chave a considerar no desenvolvimento da arquitectura é o conjunto de parâmetros de QdS a suportar. De entre o conjunto de parâmetros possível o *IndiQoS* irá garantir, nesta fase inicial, a largura de banda e a latência, devido essencialmente à simplicidade e ao suporte oferecido pelas redes de dados subjacentes. Numa fase posterior, o *IndiQoS* irá suportar mais parâmetros de QdS, como o *jitter*, a disponibilidade ou a taxa de perdas.

Infraestrutura de Suporte à QdS Ao nível da rede de dados, a satisfação dos requisitos de QdS deve ser assegurada recorrendo a serviços como os oferecidos por exemplo pelo RSVP [3] ou pelos serviços diferenciados [2]. No entanto, a utilização destes mecanismos é encapsulada através da utilização de componentes de rede abstractos, designados por *Infopipes*. A utilização destes componentes esconde dos restantes níveis da arquitectura *IndiQoS* as particularidades de cada rede concreta, assegurando, não só uma maior modularidade, como a portabilidade das camadas superiores para um conjunto diversificado de redes de dados. Os aspectos mais importantes desta camada serão resumidos na Secção 3.4.

3.2 Descrição Geral da Arquitectura

Como referimos anteriormente, a arquitectura *IndiQoS* é baseada numa rede entre-pares que interliga editores e assinantes. Os nós da rede entre-pares têm

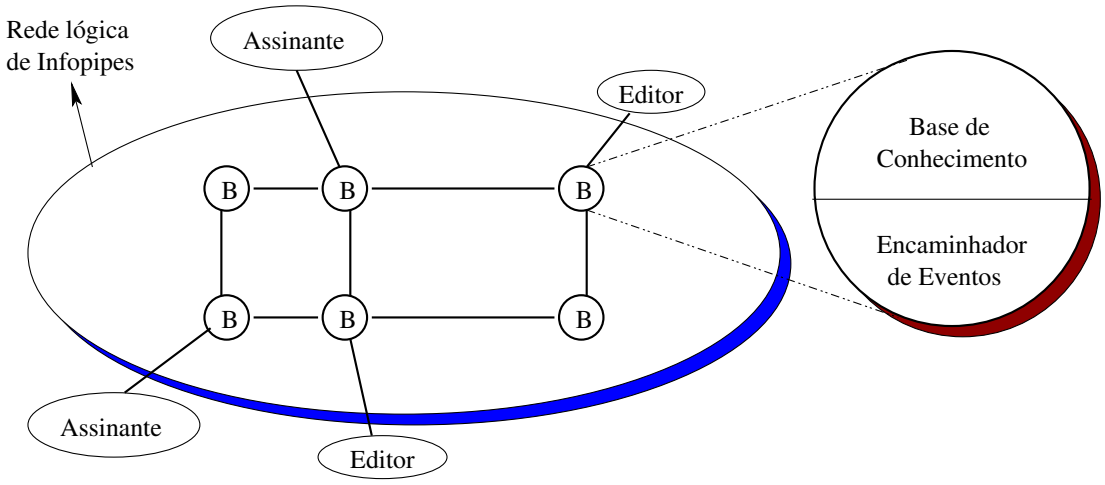

Figura 1: Ilustração da arquitectura proposta.

dois componentes: um componente de encaminhamento e um componente designado por *Base de Conhecimento*. O componente encaminhador de notificações tem a função de efectuar o encaminhamento das notificações que vai recebendo e terá de manter informação suficiente para tal. A base de conhecimento tem como função definir caminhos entre os editores e os assinantes, cujo processamento só pode ser efectuado tendo conhecimento sobre (i) a rede lógica definida sobre a rede IP, (ii) as publicações e as subscrições que incluem os recursos de QdS necessários e (iii) os recursos do sistema ainda disponíveis. Os encaminhadores de notificações recebem instruções da base de conhecimento sobre como encaminhar as notificações. Esta separação de conceitos é importante para diminuir a complexidade do sistema e para o tornar mais modular. A arquitectura está ilustrada na Figura 1.

Nesta arquitectura, uma das funcionalidades essenciais da base de conhecimento é a de determinar as árvores de distribuição óptimas para as notificações dos editores, sendo os nós da rede os únicos pontos de bifurcação possível. A dificuldade desta tarefa reside no facto de os percursos óptimos dependerem dos recursos pedidos pelos assinantes e pelos recursos ainda disponíveis em cada encaminhador de notificações.

A base de conhecimento estará distribuída pelos nós da rede. O conhecimento necessário para a tomada de decisões deve ser parcial e obtido através de informação local e de comunicação com outras partes da base de conhecimento. Assim, existem dois tipos de canais de comunicação distintos no sistema: (i) canais por onde circulam notificações e (ii) canais de controlo.

3.3 Caracterização de Tráfego

Como foi referido na secção 2, caberá às próprias aplicações fazerem a caracterização do tráfego que pretendem publicar ou subscrever. Esta caracterização deverá corresponder a um compromisso possível entre dois objectivos antagónicos. Por um lado, deverá ser possível às aplicações gerarem tráfego com um padrão que seja suficientemente flexível de forma a corresponder às suas necessidades. Por outro lado, a caracterização deverá fornecer informação suficiente ao sistema de forma a permitir a criação de reservas de recursos com dimensão adequada, tendo em vista a desejável optimização na utilização desses mesmos recursos.

Deste modo, optámos por utilizar o modelo de *balde de testemunhos* (do inglês, "token bucket") para caracterizar o tráfego de dados gerado pelas aplicações. Um balde de testemunhos é um contador não negativo que acumula testemunhos a uma taxa constante r até atingir a capacidade máxima b. Para caracterizar o tráfego de dados usando este modelo, é necessário definir valores apropriados para r e b de forma a que todas as notificações sejam enviadas, satisfazendo os requisitos da aplicação. Este modelo permite um tratamento matemático relativamente simples por parte do sistema, permitindo simultaneamente a geração de padrões de tráfego suficientemente complexos por parte das aplicações. Outros exemplos da utilização do balde de testemunhos para caracterizar o tráfego das aplicações podem ser encontrados em [18, 19].

Os anúncios e as subscrições devem estar acompanhados com informação sobre a modelação de tráfego, para além do tipo de informação que vai ser publicada ou recebida. Mais precisamente, o editor deverá anunciar (i) o tipo de informação que vai publicar e (ii) uma modelação do tráfego que vai ser gerado para essa publicação. O assinante, por outro lado, deve subscrever (i) o tipo apropriado de informação que deseja receber, (ii) uma condição de filtragem da informação, (iii) o tráfego esperado, usando o modelo de balde de testemunhos e, (iv) opcionalmente, a latência máxima de propagação de cada notificação.

3.4 Infopipes

Uma forma de tornar o sistema mais modular consiste em encapsular a concretização da QdS em componentes de software genéricos, que podem ser materializados usando diferentes soluções. Estes componentes, que serão descritos de seguida, são denominados *Infopipes*[2] [10] e quando interligados entre si, possibilitam o fluxo de dados desde uma origem até determinado(s) destino(s). Existem vários tipos de Infopipes, mas neste artigo apenas vão ser tomados em conta os seguintes:

[2]Dado que o nome destes componentes foi proposto por um grupo de investigação de *Georgia Institute of Technology*, optou-se por não traduzir as designações originais.

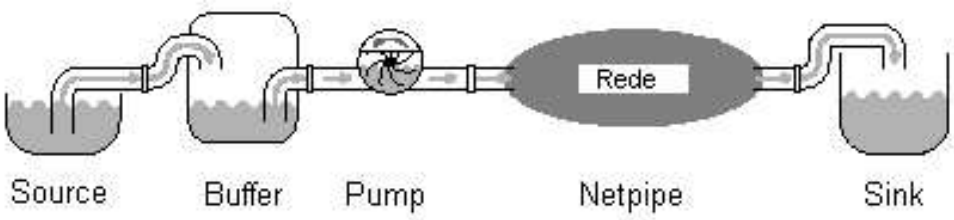

Figura 2: Exemplo de um Infopipeline que interliga dois nós.

Buffer Tem como função armazenar os items de informação que lhe chegam, reencaminhando-os para o ponto de saída quando pedido pelo Infopipe que se segue, por ordem FIFO.

Pump Tem como função pedir items de informação ao Infopipe anterior, reencaminhando-os para o seguinte a um débito constante, previamente definido, mas que pode ser alterado em tempo de execução.

Netpipe Consiste num *Pipe* em que a origem e o destino estão localizados em máquinas diferentes. É o responsável por fazer passar a informação pela rede e é sobre este Infopipe que vamos fazer incidir os parâmetros de QdS.

Source Estabelece uma ponte entre a origem da informação e os outros Infopipes.

Sink Estabelece uma ponte entre os Infopipes e o destino dos items de informação.

Split Tee É constituído por um ponto de entrada e n pontos de saída. Cada item de informação que entra neste Infopipe é replicado por todos os pontos de saída, concretizando assim a noção de *difusão*.

Um *Infopipeline* é formado interligando os Infopipes acima descritos. Com Infopipelines é possível suportar fluxos de informação, formando uma rede virtual sobre a rede de dados subjacente e escondendo as propriedades da rede de dados que não são relevantes para as restantes camadas da arquitectura *IndiQoS*. A Figura 2 exemplifica uma possível ligação entre dois nós do sistema.

Para além da API que permite interligar os Infopipes, existe uma API que possibilita a consulta/alteração de alguns parâmetros de cada Infopipe, destinada à aplicação que os usa. Por exemplo, o Netpipe tem uma API com a qual se podem verificar os parâmetros de QdS. Com esta API é possível criar mecanismos de retroacção (*feedback*) entre Infopipes localizados em máquinas diferentes. A Figura 3, retirada de [9], ilustra um componente que consulta um Infopipe do tipo

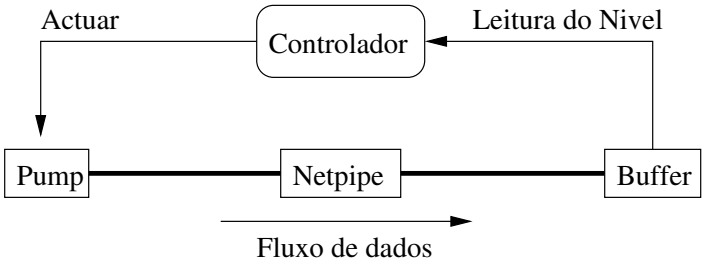

Figura 3: Exemplo de um mecanismo de retroacção para controlo de fluxo.

Buffer e actua sobre outro do tipo Pump. Quando o Buffer atinge um determinado nível, o mecanismo de retroacção reduz a taxa de débito de items de informação. É ainda possível alterar a concretização do Netpipe sem ser necessário alterar a concretização do resto do sistema.

4 Concretização da Arquitectura Proposta

Dada a complexidade de concretizar uma arquitectura desta natureza, foi efectuado um protótipo inicial que disponibiliza às aplicações as funcionalidades mais importantes. A modularidade intrínseca à arquitectura definida neste artigo permite que a construção do protótipo seja efectuada de forma incremental. De seguida, serão descritos os pontos já concretizados nesta fase inicial do trabalho.

4.1 Distribuição 0.1 do *IndiQoS*

Para a primeira distribuição da arquitectura *IndiQoS* houve a preocupação de criar um protótipo de cada um dos componentes da arquitectura, de modo a poder validar e demonstrar a interacção entre eles. Para ter este protótipo disponível numa fase relativamente precoce do projecto, foi necessário simplificar a concretização de alguns dos seus componentes. Esta distribuição materializa os seguintes componentes:

- API para as aplicações editor e assinante.
- Rede de nós entre-pares com mecanismos rudimentares para encaminhamento e manutenção da base de conhecimento. Em particular, esta primeira distribuição inclui uma concretização simplificada deste último componente, recorrendo a um servidor centralizado que é acedido através de invocação remota por todos os nós. Naturalmente, distribuições futuras irão incluir uma concretização descentralizada deste componente.

- API para instanciação e gestão de Infopipelines. Sublinhe-se que nesta versão, apenas uma concretização do componente Netpipe está disponível.

Todo o protótipo foi desenvolvido usando a linguagem Java. Nas secções seguintes, abordamos de modo abreviado a concretização de cada um dos componentes listados acima.

4.2 API para Editores e Assinantes

Para definir um novo tipo de informação, deverá ser criada uma notificação de um evento que estenda a classe abstracta `Event`, i.e., todos os tipos que circulem na rede deverão ser derivados da classe base `Event`.

```
public abstract class Event {
    // ...
}
```

A classe `Advertisement` apresentada de seguida deverá ser utilizada pelos editores para anunciar e posteriormente publicar a informação.

```
public class Advertisement {
    public Advertisement(Class eventType, QoS qos) throws RemoteException{}
    public void advertise() throws CouldNotAdvertise {}
    public void unadvertise(){}
    public Class getEventType() {}
    public TokenBucket getTokenBucket() {}
    public void setTokenBucket(TokenBucket tokenBucket) {}
    public void publish(Event event) throws CouldNotPublishException {}
}
```

O assinante, por outro lado, tem uma API para subscrever e receber um determinado tipo de informação. A subscrição é efectuada usando o método `activate()` da classe `Subscription`. Quando o assinante deseja parar de receber informação, basta invocar o método `deactivate()`. Quando é criada uma subscrição, é também especificado um filtro para a informação e um objecto onde serão recebidas as notificações. Estes dois últimos objectos são uma concretização das interfaces `Filter` e `Notifiable` e são definidas pela aplicação. O assinante recebe assincronamente as notificações do tipo que subscreveu, no método `handler()` da classe que concretiza a interface `Notifiable`.

```
public interface Filter {
        public boolean testFilter(Event event);
}
```

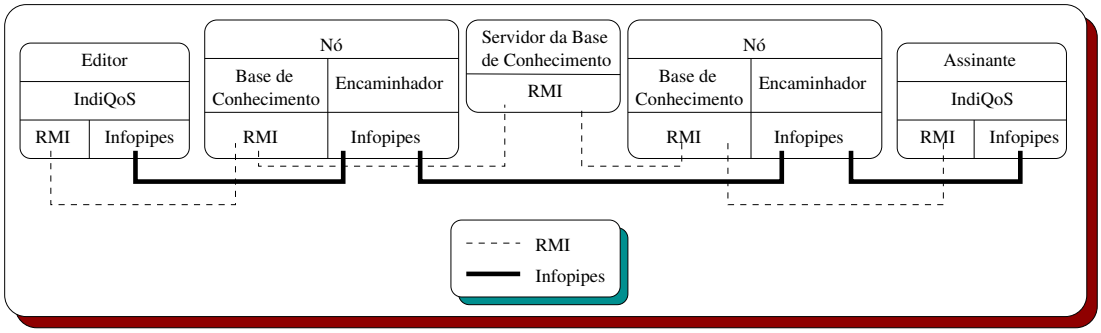

Figura 4: Interacção entre os componentes do sistema.

```
public interface Notifiable {
        public void handler(Event e);
}

public class Subscription {
    public Subscription(Class et, Filter f, Notifiable n, QoS qos, int localPort)
        throws RemoteException {}
    public void activate()
        throws CouldNotActivateSubscription {}
    public void deactivate(){}
    public Class getEventType() {}
    public Filter getFilter () {}
    public Notifiable getNotifiable () {}
    public QoS getQos() {}
    public void setFilter( Filter filter ) {}
    public void setNotifiable( Notifiable notifiable ) {}
    public void setQos(QoS qos) {}
}
```

4.3 Rede de Nós Entre-Pares

Como referimos anteriormente, a troca de informação entre os vários componentes do sistema faz-se através de canais distintos. Para trocar informação de controlo é usada a invocação remota de procedimentos, mais concretamente o pacote *Remote Method Invocation* (RMI) do Java. Para efectuar o envio de notificações, caso em que é necessário garantir QdS, são usados os Infopipes. A Figura 4 apresenta uma visão geral da arquitectura. A figura ilustra também as interacções existentes entre os componentes e os tipos de canais utilizados nessa interacção, que passamos a descrever mais pormenorizadamente.

- Quando um nó da rede entre-pares se inicia, a sua base de conhecimento local regista-se no sistema. Como referimos, nesta primeira versão, a base de conhecimento baseia-se num servidor centralizado, pelo que toda a interacção entre as bases de conhecimento locais é feita através deste servidor. No caso concreto do protótipo, esta interacção é materializada por invocações remotas (RMI).

- De um modo similar, os editores e os assinantes interagem com o servidor da base de conhecimento, de modo a registar os seus anúncios e subscrições. Note-se que um editor não necessita de enviar notificações para o sistema enquanto não existirem assinantes interessados.

- A base de conhecimento, ao emparelhar uma subscrição com um ou mais anúncios, determina o percurso para o fluxo de dados. Quando existem vários assinantes para a mesma informação, este percurso encontra-se organizado na forma de uma árvore de disseminação. Em consequência, a base de conhecimento instrói os encaminhadores de notificações dos nós da rede entre-pares envolvidos no percurso para criarem os Infopipes necessários à transferência da informação. Caso se trate da primeira subscrição para um determinado editor, e se o passo anterior for concluído com sucesso, o editor é instruído para começar a emitir notificações. Caso se trate de uma segunda ou posterior subscrição, será nalguns casos possível, ao construir a árvore de disseminação, partilhar alguns dos Netpipes já criados anteriormente.

  A base de conhecimento parametriza os Infopipes que participam no Infopipeline de modo a assegurar que os parâmetros de QdS requisitados pela aplicação são satisfeitos. Exemplos desta parametrização são: cálculo dos tamanho dos *buffers*, restrições à largura de banda e latência dos Netpipes.

Tal como pretendido pela nossa arquitectura, todas as operações de controlo descritas são efectuadas pelo protótipo do *IndiQoS*. As aplicações que usam o protótipo limitam-se a efectuar anúncios ou subscrições com as QdS's pretendidas, usando a API definida anteriormente.

Uma característica interessante do *IndiQoS*, consiste no facto de a base de conhecimento poder criar um Infopipeline distinto para cada fluxo de dados, ou optar por multiplexar diversos fluxos num único Infopipeline. Esta última estratégia é possível se, ao criar um Infopipeline, for efectuada uma sobre-reserva de recursos de modo a permitir a sua posterior partilha. Deste modo é possível estabelecer diferentes compromissos entre a utilização da rede e a sinalização necessária para estabelecer novos fluxos [16]. Esta estratégia justifica também porque optámos por criar Infopipelines com base numa rede de Netpipes ponto-a-ponto ao invés de Infopipes suportando difusão directamente entre o emissor e

os assinantes (embora este tipo de Netpipes possa ser utilizado entre um nó da rede entre-pares e diversos assinantes a ele associados).

4.4 Infopipelines

Foi realizada uma concretização em Java dos Netpipes enumerados na Secção 3.4, seguindo a sua especificação.

Cada Infopipe, à excepção do Source e do Sink, é constituído por três componentes distintos: (*i*) um porto de entrada, (*ii*) um núcleo e (*iii*) um ou mais portos de saída. O Source e o Sink apenas têm um núcleo e um porto de saída/entrada, já que consistem nos pontos de partida e chegada da informação que circula no Infopipeline. Os Infopipes interligam-se através dos portos, ou seja, um porto de saída de um Infopipe liga-se a um porto de entrada do Infopipe seguinte, formando Infopipelines. É através destas ligações que a informação é encaminhada entre Infopipes. O núcleo de cada um é responsável por efectuar o processamento inerente ao tipo de Infopipe em questão.

Cada Infopipe dispõe também de uma interface para a respectiva aplicação. Esta interface permite alterar/consultar configurações dos Infopipes, por exemplo, a aplicação pode alterar a taxa de um *Pump* ou alterar o tamanho de um *Buffer*.

5 Trabalho Futuro

Este trabalho está na sua fase inicial, tendo sido já concretizados alguns dos componentes mais importantes do sistema. A modularidade intrínseca da arquitectura permite que a concretização do sistema seja efectuada de forma incremental. Alguns dos módulos do sistema foram construídos de uma forma simplista de forma a testar a interacção entre os componentes. Apesar do protótipo existente já permitir lançar pequenas aplicações – editores e assinantes – com QdS, ainda é cedo para extrair uma avaliação definitiva acerca dos méritos da presente arquitectura. O trabalho futuro passa por construir versões completas dos módulos da arquitectura.

O primeiro passo será, naturalmente, substituir a concretização centralizada da base de conhecimento por uma versão descentralizada, em que a informação de controlo estará distribuída pelos vários nós da rede entre-pares. Isto irá obrigar a desenvolver e concretizar algoritmos que permitam definir os Infopipelines de forma localizada (mas não necessariamente óptima).

Outro passo, será desenvolver outras concretizações dos Infopipes, em particular dos Netpipes, de modo a permitir utilizar a arquitectura sobre diferentes mecanismos de QdS ao nível da rede de dados subjacente. É importante sublinhar que a satisfação de um conjunto concreto de requisitos de largura de banda

e latência pode ser satisfeito de várias maneiras, através de diferentes combinações de Infopipes e, inclusive, diferentes configurações dos mesmos Infopipes (tais como largura de banda e tamanho dos *buffers* à entrada dos Netpipes). Deste modo, o desenvolvimento de algoritmos que optimizem a utilização dos recursos disponíveis será fundamental para o sucesso da arquitectura. Outro aspecto importante a ter em conta consiste na utilização das APIs que os Infopipes colocam à disposição das aplicações. Estas API's permitem construir controladores baseados em retroacção (*feedback*) no sistema. Com estes mecanismos é possível efectuar, por exemplo, uma reconfiguração dos caminhos definidos para certos tipos de notificações, de forma a optimizar o consumo de recursos.

6 Conclusões

Este artigo apresenta uma nova arquitectura de comunicação para sistemas distribuídos que segue o paradigma publicação-subscrição, enriquecida com funcionalidades que garantem QdS às aplicações. Esta arquitectura foi definida de forma a ser escalável, modular e também de forma a fornecer garantias de QdS às aplicações de forma automática. O artigo faz também uma descrição da concretização desta arquitectura. Este protótipo permitiu validar a interacção entre os diversos componentes da arquitectura.

Referências

[1] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *IEEE Computer*, March 2000.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differenciated services, December 1998. RFC 2475.

[3] Ed.R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (RSVP) — version 1 functional specification, September 1997. RFC 2205.

[4] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview, June 1994. RFC 1633.

[5] Antonio Carzaniga. *Architectures for an Event Notification Service Scalable to Wide-area Networks*. PhD thesis, Politecnico di Milano, December 1998.

[6] Antonio Carzaniga, David S. Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[7] P. Th. Eugster, R. Guerraoui, and Christian H. Damm. Linguistic support for large-scale distributed programming. In *In 16th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001)*, pages 131–146, October 2001.

[8] Th. Eugster Felber. The many faces of publish/subscribe. Technical report, Swiss Federal Institute of Technology in Lausanne (EPFL), 2001.

[9] Jie Huang, Andrew Black, Jonathan Walpole, and Calton Pu. Infopipes - an abstraction for information flow, 2001. http://www.cse.ogi.edu/ walpole/papers/eco-opw2001.pdf.

[10] R. Koster, A. P. Black, J. Huang, J. Walpole, and C. Pu. Infopipes for composing distributed information flows. In *Proceedings of the International Workshop on Multimedia Middleware*, pages 44–47. ACM, October 2001.

[11] Object Management Group, OMG Headquarters, 250 First Avenue, Suite 201, Needham, MA 02494, USA. *Notification Service Specification*, June 2000.

[12] Object Management Group, OMG Headquarters, 250 First Avenue, Suite 201, Needham, MA 02494, USA. *Event Service Specification*, March 2001.

[13] P. Pietzuch and J. Bacon. Hermes: A distributed event-based middleware architecture. In *22nd IEEE International Conference on Distributed Computing Systems Workshops (DEBS '02)*, 2002.

[14] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[15] Antony I. T. Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.

[16] Rute Sofia, Roch Guérin, and Pedro Veiga. An investigation of inter-domain control aggregation procedures. DI/FCUL TR 02–8, Department of Informatics, University of Lisbon, July 2002. Superseded by report 02-14.

[17] Sun Microsystems, 901 San Antonio Road, Palo Alto, CA 94303, USA. *Java Message Service*, November 1999.

[18] Puqi Perry Tang and Tsung-Yuan Charles Tai. Network traffic characterization using token bucket model. In *INFOCOM (1)*, pages 51–62, 1999.

[19] J. Wroclawski. The use of RSVP with IETF integrated services, September 1997. RFC 2210.