DEEDS – Uma Plataforma Extensível Para Disseminação de Eventos

Sérgio Duarte, José Legatheaux Martins, Henrique João Domingos e Nuno Preguiça

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
{smd, jalm, hj, nmp}@di.fct.unl.pt

Resumo— **Este artigo descreve uma plataforma de disseminação de eventos que tem como objectivo principal oferecer um elevado grau de adaptabilidade a um espectro variado de cenários de utilização, envolvendo larga-escala, ambientes heterogéneos e com requisitos distintos de qualidade de serviço. Descrevem-se os fundamentos de uma abordagem ao problema que assenta na extensibilidade da plataforma por via da incorporação de novos módulos de sistema que ofereçam novas funcionalidades, sejam reutilizáveis e, ainda, que eles próprios possam servir de ponto de partida ou de guias para a criação de uma nova geração. Apresentam-se exemplos concretos de alguns dos módulos já desenvolvidos que ilustram a expressividade do modelo seguido face aos objectivos propostos.**

I. Motivação

A programação reactiva baseada na troca e processamento de eventos é um paradigma reconhecido, amplamente utilizado em diversos domínios do desenvolvimento de aplicações informáticas, tais como: interfaces gráficas, aplicações multimédia, software de vigilância e monitorização e aplicações CSCW e *groupware* em geral.

A utilidade e a universalidade do uso de eventos são resultado da própria simplicidade do conceito e da sua conveniência na expressão de interacções entre componentes com variados graus de envolvimento entre as partes. Por isso, os eventos são uma ferramenta bastante interessante para construir aplicações complexas a partir de componentes avulsos [1] .

Apesar da importância dos eventos ser reconhecida, as soluções oferecidas pecam, em geral, por serem bastante limitadas no que toca à sua adequação a cenários envolvendo larga-escala, mobilidade ou no suporte à heterogeneidade. Na maioria dos casos, os mecanismos de troca de eventos disponibilizados apenas têm uma utilização realista nos ambientes restritos às redes locais, onde as condições de entrega dos eventos são bastante previsíveis e benevolentes. A tendência observada em ignorar ambientes mais variados e menos tolerantes, conduz, muitas vezes, ao recurso de abordagens *ad hoc* para situações específicas que poderiam ser evitadas, à partida, através de um suporte à disseminação de eventos mais completo e reutilizável. Este é um problema que permanece e que está a ganhar importância, pois a antevisão de um mundo cada vez mais povoado por toda uma panóplia de equipamentos informáticos interligados, irá, naturalmente, exigir um maior protagonismo para soluções distribuídas com suporte adequado e específico à mobilidade e à larga-escala.

A profusão de serviços baseados na Internet e de dispositivos prontos a ligarem-se a esta que é possível observar hoje, estão a tornar a Internet num meio incrivelmente diverso, bastante distante do modelo homogéneo original. As pressões crescentes, impostas por requisitos de segurança, estão a levar à imposição de fortes barreiras administrativas ao livre curso de informação. Para além disso, as enormes diferenças de capacidade computacional que é possível encontrar entre dispositivos, mostram que não será realista esperar bons resultados de modelos e soluções que se baseiam no princípio do "one size fits all", exemplificado, por exemplo, pela adopção generalizada do HTTP como a panaceia para todos os problemas de conectividade. Face a tudo isto, parece claro que a engenharia de aplicações distribuídas de larga-escala é um problema cada vez mais complexo e que torna urgente e oportuno o estudo de soluções para os problemas enumerados. Em particular, levam-nos a crer que, no domínio da disseminação de eventos, existe todo o interesse no desenvolvimento de uma plataforma de disseminação de eventos onde a ênfase se centra na extensibilidade e na programabilidade de todos os aspectos relacionados com o processamento e transporte de eventos, como forma de atacar os desafios inerentes à larga-escala, mobilidade e heterogeneidade.

Nas próximas secções deste artigo, descrevemos a plataforma DEEDS, a qual traduz os nossos esforços no desenvolvimento de um sistema de disseminação de eventos com os objectivos delineados anteriormente. Primeiramente, são descritas as principais características do modelo de disseminação e do modelo de programação adoptados, seguidos de uma discussão da arquitectura de suporte dos mesmos. O artigo prossegue com uma exemplificação da extensibilidade do sistema através de uma ilustração dos módulos desenvolvidos no âmbito de um cenário aplicacional concreto. Por fim, concluímos apresentando o trabalho relacionado na área.

II. Modelo de Disseminação de Eventos

DEEDS[1] é uma plataforma de disseminação de eventos baseada em JAVA, desenhada para ser flexível e apropriada para um largo espectro de aplicações e cenários de execução. Os princípios que têm guiado o seu desenvolvimento são a extensibilidade e a adaptabilidade das suas capacidades como forma de satisfazer os requisitos da larga-escala, mobilidade e heterogeneidade em contextos específicos. A solução oferecida é genérica no sentido que pode ser facilmente adaptada a problemas particulares ou porque simplifica, grandemente, a criação de novas soluções usando os componentes já existentes como guias exemplificativos. Nesse sentido, é usado um pequeno conjunto de conceitos simples e intuitivos que promovem uma abordagem incremental que capitaliza na experiência como forma de resolução de problemas novos.

A plataforma DEEDS segue o já bem conhecido paradigma editor/assinante, acrescido de uma operação de *retorno* ou *feedback*, a qual permite aos consumidores de eventos dialogaram com os editores individualmente. Esta operação de *retorno* recebe igual importância no modelo global de disseminação porque se antecipa que a recepção de um evento possa desencadear uma eventual necessidade de diálogo entre o editor respectivo e qualquer um dos assinantes, usando uma semântica de comunicação estritamente ponto a ponto, por oposição à operação de edição que normalmente será do tipo multi-ponto.

DEEDS expande a conhecida noção de canal de eventos, ao fornecer operações de disseminação de eventos sobre *canais de eventos activos*, um conceito que se inspira nalgumas ideias do domínio das *redes activas*[6], como será explicado mais adiante.

Tradicionalmente, os canais de eventos simbolizam endereços lógicos que servem de ponto de encontro ou de mediadores no relacionamento de editores e assinantes. Frequentemente, têm o objectivo de possibilitar interacções entre eles com variados graus de separação tanto no tempo como no espaço. O nome dado aos canais de eventos pode, também, ser indicativo do tipo ou natureza dos eventos neles transportados. Usados desta maneira, os canais de eventos podem ser vistos como uma ferramenta para estruturar os fluxos de eventos de uma aplicação.

Os canais de eventos disponibilizados na plataforma DEEDS herdam todas estas características, mas o modelo proposto estipula, ainda, que cada canal, individualmente, possui uma qualidade de serviço intrínseca. Este atributo da qualidade de serviço é expresso segundo uma combinação de qualidades abstractas tais como: fiabilidade, garantias de ordenação, capacidades de persistência, tipo de endereçamento dos assinantes (*unicast*, *multicast*, *anycast*, etc), latência, modelo de falhas, etc. Esta qualidade de serviço própria de um canal é fruto de uma codificação específica e, normalmente, assenta na assumpção de que certas condições de execução serão respeitadas. Neste modelo, a confiança na qualidade de serviço oferecida por dado canal baseia-se, portanto, numa correcta e íntegra parametrização e instalação da infra-estrutura de disseminação, em justo acordo com os requisitos de execução dos canais. Na maioria das circunstâncias, este atributo não será exactamente um contrato formal mas deverá ser visto mais como uma forma de documentar o comportamento esperado de um dado canal de eventos. Por forma a oferecer um amplo espectro de possibilidades no domínio da qualidade de serviço, a plataforma DEEDS recorre a alguns dos princípios em se baseiam as *redes activas*[6]. Este conceito de redes faz parte de uma doutrina no campo das redes de computadores que, na sua forma mais pura, advoga que os pacotes de informação são transmitidos junto com o código que rege o seu encaminhamento ao longo da rota até ao seu destino. Esse código, especificado individualmente para cada pacote, deverá alojar-se nos *routers* da rede, de forma a criar algoritmos de encaminhamento feitos à medida para cada aplicação, em clara oposição à utilização mais corrente de algoritmos de encaminhamento fixos e pré-instalados. Trata-se, portanto, de um afastamento ousado da prática tradicional das redes de computadores. Por essa razão mas, também, pelos problemas relativos à segurança e ao desempenho que a podem afectar, é uma abordagem que continua a enfrentar considerável resistência.

A plataforma DEEDS, ao adoptar alguns dos princípios das redes activas, tem como principal objectivo usufruir da sua principal vantagem que reside na facilidade com que permite que novos algoritmos de encaminhamento possam ser experimentados e explorados pelas aplicações, sem haver necessidade de uma actualização concertada da infra-estrutura de disseminação. Por outro lado, evita-se a maior das críticas feitas às redes activas pelo facto de disseminação de eventos efectuar-se num contexto de uma rede virtual ou lógica, onde a segurança dos seus *routers*, apesar de importante, não põe em causa a infra-estrutura real da rede física.

Os canais activos de eventos fornecidos pela plataforma DEEDS baseiam-se no princípio que todos os eventos que flúem num dado tipo de canal partilham a mesma lógica de encaminhamento, codificada em objectos específicos denominados de *system assistants*. Estes objectos são responsáveis por materializar a qualidade de serviço publicitada pelo canal em questão, tirando partido dos recursos disponibilizados por um conjunto de servidores aplicacionais, organizados numa infra-estrutura lógica do tipo *overlay network*. O isolamento do código, que executa nos servidores da rede, em objectos específicos, pré-sancionados tem como benefícios, uma maior facilidade e eficiência na sua distribuição e elimina os riscos de execução de código arbitrário por parte dos servidores da infra-estrutura de disseminação. Além disso, como os *system assistants* intervêm ao nível do sistema, podem ser substituídos, em qualquer altura, por uma versão melhorada, sem que as aplicações precisem de preocupar-se ou, sequer, o saber. Finalmente, este modelo assenta, ainda, no pressuposto de que a qualidade de serviço oferecida por um *system assistant* é independente dos protocolos de comunicação que interligam os servidores na rede lógica. Esta qualidade é fundamental e

[1] Distributed Extensible Event Dissemination Service

pretende-se que vá ao encontro de um dos objectivos da plataforma em suportar ambientes heterogéneos. A título de exemplo, permite explorar cenários interessantes tais como, ter um grupo de aplicações a ser servido, eficientemente, através de comunicação IP multi-ponto numa rede local e, ao mesmo, tempo usar TCP ou HTTP, para alcançar um domínio remoto, onde outro arranjo de protocolos foi preferido e instituído.

III. Arquitectura da Infra-Estrutura de Disseminação

Para materializar o modelo descrito anteriormente foi desenvolvida uma arquitectura distribuída com o propósito de suportar ambientes heterogéneos de larga-escala. A infra-estrutura de disseminação consiste numa rede lógica *(overlay network)* de nós servidores e clientes, organizada em três camadas. Interligando os nós, de forma a criar a rede lógica, existem múltiplas ligações redundantes, denominadas *transportes*, cujo tipo está dependente da disponibilidade e das políticas administrativas locais. Tipicamente, estas ligações são criadas usando protocolos ao nível da camada de transporte (ou superior) tais como: TCP, UDP, UDP Multicast, HTTP, etc. A primeira camada da rede lógica forma a espinha dorsal (*backbone*) da infra-estrutura de disseminação, enquanto que a camada intermédia é formada por servidores secundários e por aplicações clientes. A última camada apenas abrange as aplicações clientes, como se pode observar na Figura 1.

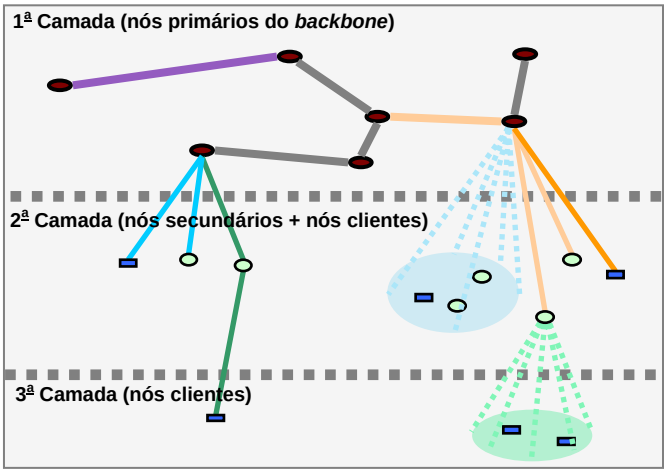


Figura 1 – Organização da rede lógica que compõe a infra-estrutura de disseminação DEEDS.

Nesta infra-estrutura de disseminação, são os nós primários que possuem a maior responsabilidade no encaminhamento e processamento de eventos, sendo neles que os protocolos implementados pelos *system assistants* terão maior impacto. Os nós secundários têm o objectivo de aliviar a carga dos nós primários, evitando, sempre que possível, que um número excessivo de nós clientes, correspondentes às aplicações finais, se ligue directamente a estes.

De forma a facilitar a coordenação da infra-estrutura de disseminação DEEDS, os nós da rede lógica suportam um repositório global de informação denominado por *system registry* ou *repositório do sistema*. Este repositório guarda informação de natureza variada, desde informação global quase-estática, até informação de carácter local colectada dinamicamente. As porções globais do repositório são replicadas entre todos os nós (não clientes) da rede, usando simultaneamente um processo pró-activo de baixa prioridade, bem como a pedido, em resultado de buscas explícitas. Como exemplo da informação com carácter global mantida neste repositório, destaca-se o directório dos canais de eventos existentes. É também neste repositório que é guardada a informação relativa aos módulos de sistema (*system assistants*) que materializam cada tipo de canal e donde estes são carregados automaticamente quando necessário.

IV. Arquitectura dos Servidores

A função principal dos servidores, quer se trate de um nó primário ou de um nó secundário, consiste em encaminhar os eventos entre os editores e os assinantes através da infra-estrutura de disseminação de um modo consistente com a qualidade de serviço esperada. Tal como foi indicado anteriormente, esta operação não é regida por um único algoritmo de encaminhamento pré-definido, mas é guiada pela lógica contida em *system assistants* indicados para efeito, específicos para cada canal. Para tal, cada servidor fornece o motor de execução destes objectos, implementado e expondo o ambiente adequado.

Para tornar a criação de novos tipos de canais, e por conseguinte de novos *system assistants*, sem ter de começar de raiz, cada servidor fornece à partida um conjunto de recursos de natureza variada. Estes recursos consistem em informação útil, global ou específica para cada canal, que é oportunamente actualizada pelo sistema. Nomeadamente, esses dados incluem, por exemplo, listas de assinantes (locais) de cada canal, nós primários activos, transportes disponíveis e várias tabelas de encaminhamento previamente calculadas. Consequentemente, para obter e manter esta informação actualizada, cada servidor é composto por um número de serviços em execução, que cooperam com as suas contrapartes noutros nós, de modo a produzir o que é, essencialmente, uma visão monitorizada do estado da infra-estrutura de disseminação. O acesso a esta informação pelos processos do servidor faz-se através do repositório de sistema, sendo ele próprio gerido por um destes serviços. Uma descrição mais fina deste, e dos demais serviços presentes no protótipo corrente da plataforma DEEDS, é feita nas subsecções seguintes. Note-se que a exposição destes serviços pretende salientar o facto do protótipo ter sido concebido, propositadamente, de uma forma recursiva, de modo a explorar os próprios conceitos propostos na plataforma e, assim, obter desde logo uma validação prévia de vários aspectos da arquitectura.

A. *Serviços de Monitorização do Backbone*

Estes são dois serviços interligados que correm, apenas, nos nós da primeira camada da rede lógica, ou seja no *backbone*. O seu objectivo é o de monitorar, continuamente, o estado do *backbone* de modo a produzir uma visão estruturada deste, da

qual possam ser derivadas árvores de cobertura (e as respectivas tabelas de encaminhamento ponto a ponto e multi-ponto) que sejam razoáveis e reflictam uma interpretação racional da topologia da rede física subjacente.

O mais simples destes serviços é o serviço *Hello*, cuja tarefa consiste em sondar uma lista de nós primários, um por um, para determinar quais estão activos e a que distâncias se encontram em termos de latência. Para tal, este serviço recorre um canal de eventos próprio, com uma semântica de endereçamento ponto a ponto, usado na troca de estampilhas temporais com os outros nós. Em cada passo, o processo de selecção do alvo da próxima sondagem é determinado por um escalonador que atribui prioridades mais altas aos nós mais próximos e mais importantes, de forma a evitar que a largura de banda atribuída ao serviço (que é fixa) não seja desperdiçada a sondar nós irrelevantes, ou seja, aqueles que estão, simultaneamente, longe em termos de latência ou em número de saltos na árvore de cobertura corrente.

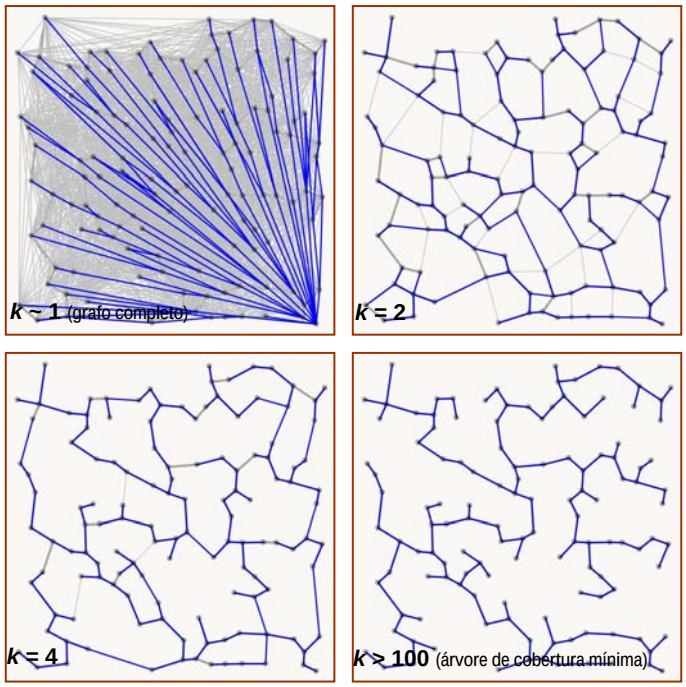


Figura 2 – Grafos de cobertura para 4 factores distintos, obtidos a partir de um grafo completo correspondente a uma rede lógica de 128 nós com latência proporcional à distância euclidiana. As linhas mais escuras/azuis correspondem à árvore de cobertura com raiz no nó do canto inferior direito.

O serviço *LinkState* completa este par de serviços. Em cada nó, a sua função consiste em distribuir, eficientemente, os dados colectados pelo serviço *Hello* aos demais nós e recolher os anunciados por estes. Deste modo, cada nó primário obtém uma visão global do estado da infra-estrutura de disseminação, podendo daí derivar uma árvore de cobertura com características desejáveis, valendo-se por exemplo, de algoritmos da teoria de grafos. Na disseminação da informação, este serviço recorre ao seu próprio canal de eventos, tratando-se, neste caso, de um canal que implementa uma difusão total. A lógica de encaminhamento utilizada é peculiar, na medida em que os dados a disseminar são publicados formatados de maneira a, também, incluírem a mais recente árvore de cobertura do *backbone* calculada. Com um modesto aumento do tamanho das mensagens, esta formatação especial permite que elas possam ser difundidas de forma eficiente a todos os nós do *backbone*, ao executar-se o encaminhamento em cada nó em função da topologia da árvore de cobertura embebida na própria mensagem. Este esquema é especialmente vantajoso porque, esta utilização de encaminhamento baseado na origem garante a ausência de ciclos e, portanto, dispensa o tratamento de mensagens duplicadas. Evita-se qualquer necessidade de coordenação explícita entre as várias instâncias do serviço e, apesar disso, consegue-se melhorar, globalmente, a informação de encaminhamento usada pelo próprio canal de uma forma progressiva e recursiva.

No presente protótipo, a árvore de cobertura usada pelo canal do serviço *LinkState* pode ser obtida por dois algoritmos alternativos. O mais simples, consiste no cálculo da árvore de cobertura mínima do grafo completo ou (quase completo) que inclui todos os nós primários. Este algoritmo tem a vantagem de oferecer um custo computacional baixo. Porém, tem o defeito decorrente das árvores de cobertura mínimas terem, frequentemente, elevada profundidade e tenderem a formar meandros, que frequentemente levam a que pares de nós que estão relativamente próximos possam não o estar segundo a árvore. Esta particularidade pode afectar negativamente o desempenho do serviço pois qualquer perda de pacotes ao longo do trajecto de difusão joga em contra do objectivo de que nós próximos tenham uma visão da rede o mais consistente possível. Pode-se minorar este problema obtendo a árvore de cobertura a partir de um grafo de cobertura (*spanner*), construído de modo a garantir que o custo entre qualquer par de nós não exceda num dado um factor a seu custo directo.

Este algoritmo alternativo, para factores suficientemente grandes, produz a árvore de cobertura mínima. Para factores progressivamente menores, vão sendo adicionadas ligações (transformando a árvore de cobertura mínima em grafo de cobertura) que funcionam como atalhos que encurtam as distâncias entre nós que de outra maneira excederiam o factor limite. Para factores próximos de 1, o resultado do algoritmo degenera no grafo de partida (neste caso um grafo completo ou quase completo), como se pode observar na Figura 2. Este algoritmo apresenta, no entanto, um custo computacional significativamente superior.

B. Serviço de Gestão do Repositório do Sistema

Como o seu nome indicia, este serviço tem a seu cargo as tarefas relacionadas com a gestão do repositório de informação do sistema (*system registry*). Este serviço executa em todos os nós, independentemente do tipo, mas nos nós clientes a sua operação é restrita, ao apenas suportar replicação volátil (*caching*).

A actualização do repositório é feita de duas maneiras distintas. Existe um método de replicação que se limita a disseminar os itens do repositório, usando uma política de

varrimento circular. A prioridade deste processo é muito baixa e reflecte a largura de banda atribuída a esta tarefa. Mais frequentemente, as actualizações são o resultado das consultas realizadas num nó que, não podendo ser resolvidas localmente, resultam numa pesquisa noutros nós. As respostas com sucesso a estas consultas não só actualizam o repositório que gerou a consulta mas, também, os repositórios de todos os nós por onde passou a resposta à consulta. Ambas formas de actualização assentam na utilização pelo serviço de um canal próprio para enviar e receber informação. O tipo de encaminhamento específico deste canal propaga os eventos um salto de cada vez, disseminando cada consulta ou item para os nós vizinhos e para longe da origem e enviando (ponto a ponto) a resposta a uma consulta de volta ao destino. Em cada salto, o repositório do nó em questão é consultado ou actualizado, dependendo da natureza da operação.

V. Modelo de Programação

O modelo de programação da plataforma DEEDS está definido para um ambiente de execução JAVA standard. As interfaces de programação estão divididas em duas categorias. Uma dedicada ao desenvolvimento de aplicações e a outra utilizada para estender a plataforma com novos tipos de canais, novos tipos de transportes e serviços de suporte adicionais.

A noção de evento usada em DEEDS é flexível e corresponde, genericamente, a uma notificação de pequena dimensão, estanque e composta por um terno de valores, dois fornecidos pela aplicação e um terceiro retornado pelo sistema, como resultado de uma operação de envio. Os dois componentes do evento especificados pela aplicação são, respectivamente, o *envelope* e a *carga*. A *carga* pode ser qualquer objecto *serializável*[2] e pretende-se que seja a parte mais volumosa do evento e aquela que normalmente é opaca para o sistema. O *envelope* acompanha a *carga* e deve ser visto como uma forma de descrever o evento de uma forma abreviada ou de passar informação de controlo arbitrária à infra-estrutura de disseminação. O *recibo* completa o evento e é gerado, automaticamente, como resultado de uma operação de *edição* ou de *retorno*. Os *recibos* não podem ser fabricados e agregam informação variada, produzida pelo sistema acerca de um evento, tal como o seu número de sequência, a sua dimensão depois de empacotado e o identificador da entidade que o gerou. Por conter este último, o *recibo* é um dos argumentos da operação de *retorno* como forma de especificar o seu alvo.

DEEDS recorre, ainda, a outro tipo importante de objectos, o *critério*, usado como argumento nas operações de assinatura de canais e serve de contraparte ao *envelope*. Estes objectos são especificados pelos assinantes de um canal para instruírem o sistema relativamente ao padrão de eventos dos quais querem ser notificados. Os *critérios* correspondem, portanto, a filtros de eventos que operam sobre os *envelopes* dos mesmos. Uma assinatura baseada num *critério* pressupõe que o *envelope* dos eventos contém informação em que se baseia a filtragem, sendo o normal que essa informação seja uma descrição grosseira do resto do evento. Assim, no contexto da filtragem, torna-se evidente a vantagem da separação entre *envelope* e *carga*, pois em qualquer ponto da infra-estrutura de disseminação é possível filtrar eventos sem ter de descodificar a sua parte potencialmente mais volumosa correspondente à *carga*, algo que seria imperativo caso ambos estivessem agregados numa só estrutura.

A interpretação do *envelope* como fonte de informação de controlo permite que a disseminação do evento possa ser controlada finamente. Para além do papel na filtragem como foi descrito, o *envelope* pode ser, também, usado para controlar o encaminhamento do evento. Se, por exemplo, num canal com requisitos de pontualidade, o envelope incluir uma estampilha temporal, esta pode ser analisada em qualquer ponto da rota para descartar eventos demasiado atrasados e, assim, libertar mas cedo recursos de comunicação valiosos. Se, por outro lado, o envelope conter uma descrição do conteúdo da *carga*, então pode ser usado em canais que disseminam eventos segundo políticas de encaminhamento baseadas no conteúdo. Ou pode, ainda, funcionar como critério de indexação numa base de dados destinada ao suporte de canais que ofereçam formas de persistência de eventos. De um modo geral, a utilização conjunta de *envelopes* e *critérios* aumenta a flexibilidade, em número e em tipo, das soluções de disseminação de eventos que é possível explorar na plataforma DEEDS.

```
import deeds.api.*;
public class Publisher implements EventFeedbackSubscriber {
    EventChannel c ;
    public Publisher() {
        Deeds.Directory().clone( “prototype”, “channel_name”);
        c = Deeds.Directory().lookup(“channel_name”);
        c.subscribeFeedback( criteria, ..., this);
        while(...)  c.publish( envelope, payload );
        c.unsubscribe(...);
    }
    void nofifyFeedback( Receipt r, Envelope e, MarshalledEvent m ) {
        Object  payload = m.getEvent();
        ...
        c.feedback( r, envelope, payload2) ;
    }
}
```

```
import deeds.api.*;
public class Subscriber implements EventSubscriber, EventFeedbackSubscriber{
    EventChannel c ;
    public Subscriber() {
        c = Deeds.Directory().lookup(“channel_name”);
        c.subscribe( criteria, ..., this) ;
        c.subscribeFeedback( criteria2, ..., this);
    }
    void nofify( Receipt r, Envelope e, MarshalledEvent m ) {
        Object  payload = m.getEvent() ;
        ...
        c.feedback( r, envelope, payload2) ;
    }
    void nofifyFeedback( Receipt r, Envelope e, MarshalledEvent m ) {
        Object  payload = m.getEvent();
        ...
        c.feedback( r, envelope, payload2) ;
    }
}
```

O excerto de código apresentado acima exemplifica a utilização dos objectos descritos anteriormente e dos

[2] Qualquer instância de uma classe que implemente a interface java.io.Serializable ou que derive desta.

principais interfaces de programação na criação de aplicações segundo o padrão *editor/assinante*. Por uma questão de brevidade e simplicidade apenas são mostradas listas parciais de argumentos.

Este exemplo ilustra as diferenças do DEEDS, relativamente ao padrão, que consistem nas particularidades associadas ao mecanismo de *retorno* ou *feedback* e no modo como novas instâncias de canais são criadas. Especificamente a este respeito, a adição de novos canais ao directório é feita por clonagem, tendo como base um protótipo de canal já existente e previamente sancionado. Pretende-se desta forma enfatizar que a activação da infra-estrutura de disseminação não opera sobre código aplicacional arbitrário mas, sim, sobre código de sistema que terá passado pelo escrutínio adequado.

VI. Desenvolvimento de Novos Módulos de Sistema

A extensibilidade das capacidades da plataforma passa, necessariamente, pelo desenvolvimento de novos módulos de sistema, ou seja pela criação de novos tipos de *system assistants*.

Na sua essência, estes objectos implementam um protocolo de encaminhamento, suportado pelos nós da rede lógica, que tem o propósito de fazer chegar os eventos com origem nos editores aos assinantes interessados, observando certas garantias específicas de qualidade de serviço. Concretamente, um *system assistant* tem de lidar com dois fluxos de eventos distintos e separados. O fluxo multi-ponto, produzido pela operação de edição e o fluxo ponto a ponto (opcional) desencadeado pelas operações de retorno. Para além destes dois fluxos normalizados, podem ainda ser trocadas mensagens de controlo, cuja formatação e interpretação é livre.

Para ajudar na criação *system assistants*, é possível capitalizar em certos recursos oferecidos pelo sistema com esse objectivo. Caso contrário, o desenvolvimento de um novo tipo de canal seria uma tarefa bastante mais complexa, em especial se a qualidade de serviço desejada fosse minimamente elaborada. O acesso a esses recursos pré-existentes é feito através do repositório de dados do sistema e é apresentado sob a forma de objectos dinâmicos, conhecidos por *contentores*, devidamente identificados e que outros processos mantêm actualizados. Estes contentores têm a responsabilidade de detectar alterações na informação que guardam e de notificar os interessados dessas ocorrências. Este esquema permite que os módulos de sistema sincronizem o seu estado ou parte dele, em reacção a alterações ocorridas nos contentores que monitorizam. Como foi dito, o protótipo corrente da plataforma DEEDS, já fornece alguns destes contentores tais como, a lista dos nós de primários do *backbone*, os transportes que podem ser utilizados para os contactar, listas dos assinantes locais de cada canal, uma visão corrente do estado das ligações da rede lógica, uma árvore de cobertura de baixo custo e as tabelas de encaminhamento correspondentes. Estes recursos são uma grande ajuda na elaboração de novos tipos de canais. Tal será notório na próxima secção, onde se descrevem vários protótipos de canal que foram desenvolvidos no âmbito da validação do modelo de activação da rede de disseminação da plataforma DEEDS.

VII. Exemplos de Canais Activos Desenvolvidos

Com o objectivo de validar conceitos fundamentais propostos na plataforma DEEDS, foi implementado um cenário aplicacional paradigmático, onde se destaca uma clara ênfase na troca de eventos com variadas garantias de qualidade de serviço. Em termos gerais, trata-se de uma aplicação de vídeo-conferência em grupo, estruturada segundo uma suite de ferramentas complementares mas autónomas. Detalhes mais concretos acerca dessas ferramentas podem ser encontrados em [7].

Um princípio importante guiou a concretização deste cenário e consistiu em conceber as várias ferramentas tentando encontrar para os canais de comunicação utilizados a qualidade de serviço "natural" que conduzisse a soluções simples mas eficazes, relegando para o canal a responsabilidade de a implementar. Nesse sentido, o desafio não se centrou nas ferramentas propriamente ditas mas, sim, em mostrar que o conceito de canais activos, proposto na plataforma DEEDS, é uma abordagem viável para fornecer soluções de disseminação de eventos com variadas qualidades de serviço e que essas soluções auferem de vários atributos desejáveis, como o de serem simples, eficazes e, sobretudo, promoverem a extensibilidade da plataforma através da reutilização e concepção incremental de novo código.

As próximas subsecções apresentam os vários tipos de canais desenvolvidos, onde são descritos pormenores sobre os protocolos de encaminhamento utilizados para conseguir as diferentes qualidades de serviço. Importa salientar que toda a discussão se centra ao nível dos nós primários que compõem o *backbone* da infra-estrutura de disseminação. Os nós restantes, nos tipos de canais em questão, limitam-se a transferir eventos entre as camadas da rede lógica e, portanto, têm, nestes exemplos, um papel menos elaborado e interessante.

A. *UnreliableBroadcast*

Este tipo de canal é, certamente, o mais simples. O seu objectivo é o de implementar uma difusão total e um mecanismo de retorno, não fiáveis, segundo uma ordenação de eventos do tipo FIFO individualmente para cada origem.

A simplicidade deste canal advém do facto de tirar total partido das tabelas de encaminhamento fornecidas pelos serviços de monitorização do *backbone*. Como se trata de uma difusão total não há necessidade gerir listas de assinantes, sendo o encaminhamento sempre feito de modo a enviar os eventos para jusante na árvore de cobertura relativamente ao nó de origem.

Não há qualquer tentativa de detecção de duplicados ou ciclos. Ocorrência de ciclos, em resultado inconsistências temporárias nas tabelas de encaminhamento, é tratada pelo simples mecanismo de TTL incorporado no empacotamento e desempacotamento dos eventos. Por esse facto, a utilidade deste canal na presente forma é limitada. Quanto ao mecanismo de retorno, consiste em enviar os eventos empacotados directamente para o nó alvo, sem passar por terceiros.

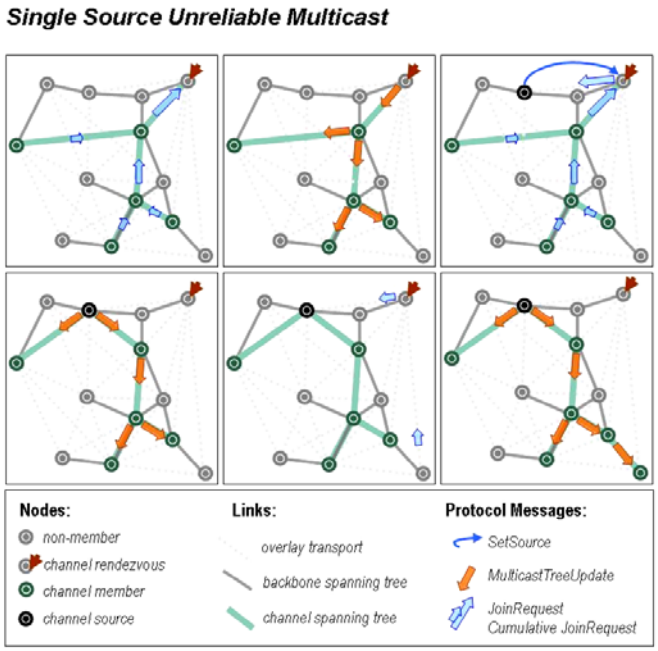
Este canal, não implementa qualquer mecanismo de fiabilidade ou retransmissão de eventos perdidos e, por isso, o seu desempenho a este respeito é influenciado pela natureza dos *transportes* que formam a rede lógica. Porém, é de notar que a utilização de *transportes* fiáveis não se traduz, de modo algum, num canal fiável mas, sim, numa instância de canal mais fiável.

B. *SingleSourceUnreliableMulticast*

Este canal possui duas diferenças importantes relativamente ao anterior, na medida em que fornece uma difusão parcial e assume que apenas um editor estará activo num dado momento, sem proceder a uma verificação ou coacção nesse sentido. Relativamente ao tipo de ordenação de eventos e mecanismo de retorno permanece igual.

Como a difusão implementada é parcial, este tipo de canal faz uso das listas de assinantes para determinar o conjunto de nós primários que participam no protocolo de encaminhamento. Os demais nós primários não participam, com excepção de um, denominado de nó *rendezvous*, que serve de ponto de encontro até o *editor* aparecer e o nó que o serve ser conhecido pelos demais. Este nó *rendezvous* é determinado de forma independente para cada instância deste tipo de canal, através de uma função que faz um mapeamento globalmente consistente do identificador de instância de canal num identificador de nó, sendo o nó primário, de entre os conhecidos localmente, com o identificador mais próximo que é seleccionado para nó *rendezvous*. Devido a partições ou inconsistências nas percepções individuais dos nós do estado da rede virtual, pode haver num dado momento, mais do que um nó *rendezvous*. No entanto, em circunstâncias normais, o funcionamento dos serviços de monitorização do *backbone* garante a convergência e a selecção de um único ponto de encontro.

Assim, periodicamente, cada nó que deseje incluir-se ou manter-se no grupo de difusão (porque o *contentor* relevante indica que possui assinantes) tem de avisar o nó indicado desse facto, e assim entrar ou permanecer na árvore de cobertura do grupo de difusão. De início, esse nó é o nó de *rendezvous* mas, depois, passa a ser o nó que serve o editor activo. Essa transferência faz-se através do envio de uma mensagem do tipo *SetSource* assim que um nó detecta que tem o editor. Concretamente, para entrar ou permanecer no grupo, cada nó interessado tem de enviar uma mensagem do tipo *JoinRequest* em direcção à raiz da árvore de difusão (ou por defeito para o nó *rendezvous*). Para evitar que a raiz seja inundada com muitas destas mensagens, cada nó funde os *JoinRequests* que recebeu dos seus descendentes directos com o seu, antes de o encaminhar para o seu pai. Em resultado desta política, a raiz tende a não receber muitas mensagens individuais mas, sim, receber poucas mensagens de maior dimensão que acumulam vários pedidos de adesão ou manutenção ao grupo de difusão. A raiz, em reposta ao um pedido de adesão, se for oportuno, difunde uma actualização da árvore de difusão (*MulticastTreeUpdate*), enviando-a para os seus descendentes directos. Estes propagam-na aos seus descendentes directos e assim por diante, já de acordo com a árvore actualizada que já deverá incluir eventuais novas adesões. Um nó abandona o grupo de difusão enviando, insistentemente, uma mensagem do tipo *LeaveRequest* directamente para a raiz, até receber uma confirmação explícita da mesma. A raiz, em resposta de um abandono, actualiza a árvore de difusão pelo mesmo procedimento usado no processo de adesão/manutenção. A figura seguinte ilustra o funcionamento deste canal.



C. *UnreliableMulticast*

Relativamente ao anterior, este tipo de canal foi estendido de modo a suportar múltiplos editores. As modificações feitas nesse sentido são relativamente simples e centram-se na natureza da árvore de disseminação que, agora, é única; incorpora todos os nós com editores ou assinantes e que passa a ficar alojada sempre no ponto de encontro. Para disseminar os eventos, cada nó interpreta a árvore de disseminação de modo a colocar a origem do pacote na raiz, encaminhando os eventos na direcção oposta. Os processos de adesão, saída do grupo de difusão e actualização da árvore de difusão são idênticos aos descritos para o canal anterior e utilizam, também, os mesmos tipos de mensagens.

Nos termos descritos, este canal pode ser muito ineficiente dependendo da proporção de editores e assinantes bem como a sua distribuição. Por essa razão, foram ensaiadas alternativas mais adequadas para os casos em que o número de assinantes é reduzido comparativamente ao número de editores. Nestas condições, é contraproducente que os nós onde há editores mas não há nenhum assinante façam parte da árvore utilizada na disseminação dos eventos, sendo melhor que esta apenas inclua os nós com assinantes. Foram ensaiadas duas alternativas segundo este esquema revisto, com a diferença entre elas a residir no modo como os eventos chegam dos editores à árvore de disseminação. Na mais simples, um editor envia simplesmente os seus eventos directamente para a raiz da árvore de disseminação. Na outra, para evitar problemas

decorrentes da raiz poder encontrar-se num nó muito distante, cada nó editor procura enviar os eventos para o nó assinante mais próximo de si, o qual depois os encaminha como se fosse ele a raiz da árvore de disseminação. Em qualquer dos casos, o nó de *rendezvous* tem de gerir duas árvores de cobertura distintas correspondentes aos dois conjuntos de nós, uma relativa apenas ao grupo dos assinantes e outra que engloba também os editores. Consequentemente, houve necessidade de introduzir mais alguns tipos de mensagens para esse efeito.

D. *SingleSourceReliableMulticast*

Os canais descritos anteriormente estão sujeitos a várias situações que conduzem à perda de eventos, como por exemplo, pela a utilização de *transportes* não fiáveis, durante fases de reorganização do *backbone* ou em alturas de instabilidade na rede física. Para resolver esse problema e recuperar os eventos perdidos, foi desenvolvido um novo tipo de canal, baseado no canal descrito em (B), que introduz em cada nó da árvore de disseminação uma pequena fila com tamanho fixo onde são guardados os últimos pacotes recebidos. Periodicamente, sempre que são detectadas falhas na sequência de numeração dos eventos é enviado de um pedido de retransmissão (*NakPacket*) para o nó imediatamente superior, o qual o tentará satisfazer com base na sua própria fila. Neste esquema, de tempos a tempos, cada nó está, também, instruído de enviar para o seu ascendente uma mensagem (*AckPacket*) indicando qual foi o último evento que recebeu dentro da sequência. Estes pacotes vão são fundidos em cada nó da árvore, à semelhança do que acontece com os *JoinRequests*, de modo a evitar o conhecido fenómeno de implosão. A raiz, ou seja, o nó que serve o editor avança a fila de acordo com o *AckPacket* com o menor dos números de sequência recebidos. E para evitar ficar bloqueado devido a nós falhados, expulsa do grupo de difusão todo o nó incapaz de avançar o seu número de sequência durante demasiado tempo.

E. *ReliableMulticast*

A mesma técnica descrita anteriormente para conseguir a fiabilidade foi adaptada à versão *UnreliableMulticast* para produzir este tipo de canal. A adaptação consistiu em ter uma fila separada para cada fonte individual de eventos e dos pacotes de controlo usados para gerir a fila serem, neste caso, encaminhados em direcção ao nó primário que serve o editor a que diz respeito a fila.

F. *TotalOrderReliableMulticast*

Este tipo de canal expande a qualidade de serviço oferecida pela solução descrita na secção anterior, oferecendo, para além da fiabilidade, a garantia suplementar de uma ordenação total dos eventos. Ou seja, assegura que estes são notificados aos assinantes por uma ordem que é globalmente consistente. A forma de o conseguir não é nova e passa por introduzir um sequenciador que tem o papel de ordenar os eventos segundo a política desejada. Concretamente, a nossa solução consiste em dar ao nó *rendezvous* esse papel e impor aos demais que entreguem os eventos pela mesma ordem por ele observada. Para que todos os nós tenham acesso à sequência observada pelo nó de *rendezvous*, este embebe o fluxo de eventos com um outro composto por mensagens que fazem o mapeamento entre a sequência individual de cada editor e a sequência de ordenação total. Neste processo, o nó *rendezvous* pode ser visto como uma nova fonte de eventos e, por isso, os mesmos mecanismos que regulam a fiabilidade garantem que as mensagens de mapeamento perdidas são recuperadas. Os eventos são entregues aos assinantes assim que as filas de espera do nó em causa contêm os eventos que respeitam tanto a sequência na origem como a sequência total do grupo.

VIII. Trabalho Relacionado

A inexistência de suporte nativo para difusão multi-ponto fiável à escala da Internet tem alimentado o interesse na procura de soluções para o problema de difusão de informação. Horus[8] e o iBus[9] são dois exemplos paradigmáticos de *middleware* de *messaging* que atacaram o problema da comunicação orientada ao grupo com garantias de qualidade de serviço configuráveis. Nestes sistemas, a qualidade de serviço é oferecida pela composição de protocolos em camadas, sob a forma de pilhas de protocolos extensíveis. Nestes sistemas, o modelo de comunicação está fortemente orientado para a computação *peer-to-peer* entre aplicações cliente, sem intervenção de servidores de suporte, ou então onde estes têm um papel muito restrito. O nosso trabalho difere destes sistemas tanto no âmbito como na abordagem. Nós propomos uma solução que inclui suporte para cenários de larga-escala, enquanto estes sistemas estão, essencialmente, vocacionados para ambientes de rede local. Nós também atacamos o problema da qualidade de serviço de modo radicalmente diferente, que não se baseia composição e que tem preferência por protocolos que não sejam do tipo “end to end”, de acordo com princípios das redes activas adaptados para o caso específico das redes lógicas.

O problema do tratamento da qualidade de serviço no contexto específico do paradigma *editor/assinante* foi também discutido em [10]. Neste trabalho, a notificação com garantias de qualidade de serviço é exposta ao nível da linguagem de programação, usando um enquadramento de “colecções assíncronas” de eventos. Este trabalho permite o processamento de eventos usando abstracções familiares de programação orientadas ao objecto como as listas, conjuntos, vectores, conjuntos ordenados, etc. Pouco ou nada nos é disto sobre os detalhes concretos da infra-estrutura de suporte subjacente a este trabalho.

Siena[11], Elvin[12] e Gryphon[13] são outros exemplos importantes de sistemas de disseminação de eventos que oferecem assinatura baseada no conteúdo. Nestes sistemas, os eventos são consumidos de um manancial global fornecendo sofisticadas expressões de filtragem que operam sobre a estrutura dos eventos. Em [12], o sistema Elvin é descrito como uma solução centralizada e não escalável, mas que oferece suporte para a desconexão dos assinantes. Tanto o Siena como o Gryphon atacam o problema da escalabilidade através de arquitecturas distribuídas envolvendo múltiplos servidores e, simultaneamente, tentam migrar as expressões de filtragem associadas às assinaturas de modo a optimizar a

disseminação. Estas plataformas enfatizam a sua componente relacionada com o processamento algébrico dos eventos e optam por basear o transporte de eventos em protocolos de encaminhamento fixos. Na qualidade de plataforma genérica, DEEDS não incorpora qualquer álgebra de eventos ou impõe uma estrutura particular para os eventos, na qual se deverão basear eventuais políticas de filtragem. Porém, como aposta na extensibilidade oferece um maior potencial na gama de cenários de disseminação que pode atacar.

[14][15] e [16] são exemplos de sistemas que também atacam o problema da disseminação de informação em infra-estruturas do tipo rede lógica. Estes sistemas exploram diferentes opções nos protocolos de encaminhamento que empregam e no modo como as redes lógicas respectivas são organizadas. Porém, têm em comum de em cada caso representarem um esforço em desenvolver um algoritmo particular para resolver um problema de âmbito bastante específico. DEEDS difere neste respeito porque oferece uma solução que aspira a generalidade e a extensibilidade por via de novos módulos de sistema. Nesta medida, estes trabalhos, mais do que alternativas, podem ser vistos como uma fonte de inspiração ou um desafio à capacidade da plataforma DEEDS de incorporar os conceitos e os algoritmos desenvolvidos nestes trabalhos e apresentá-los sob a forma de novos tipos de canais de eventos.

IX. CONCLUSÕES

Neste artigo descrevemos uma plataforma de disseminação de eventos que tem como objectivo principal oferecer um elevado grau de adaptabilidade a um largo espectro de cenários de utilização, segundo um modelo que pretende capitalizar na experiência anterior. O solução encontrada para realizar esses objectivos, assenta na extensibilidade da plataforma por via da incorporação de novos módulos de sistema que ofereçam novas funcionalidades, sejam reutilizáveis e, ainda, que eles próprios possam funcionar como ponto de partida, fonte de inspiração ou guias para a criação de uma nova geração de novos módulos.

Com o propósito de ilustrar os fundamentos deste trabalho, apresentámos um conjunto variado de módulos que representam exemplos concretos do conceito de *canal activo de eventos* e que foram desenvolvidos de uma forma incremental, no âmbito de um cenário aplicacional específico e paradigmático. Apesar de estes módulos não terem, ainda, sido alvo de uma afinação ou de um estudo formal no que toca ao seu desempenho ou correcção, estamos convencidos que na presente forma já são um importante testemunho à expressividade do modelo de *canais activos de eventos* e à sua capacidade de endereçar problemas de disseminação com variadas exigências de qualidade de serviço.

X. REFERÊNCIAS

[1] J. Bacon, K. Moody, J. Bates, R. Hayton, C. Ma, A. McNeil, O. Seidel, M. Spiteri, “Generic Support for Distributed Applications”, In Journal “Computer”, Volume 33, Number 3, p. 68-76, March 2000.

[2] Object Management Group. “Common Object Request Broker Architecture (CORBA)”, http://www.omg.org

[3] Sun Microsystems, “Java Beans(TM) Manual”, July 1997. http://java.sun.com/beans/

[4] Microsoft Corporation, “Microsoft COM Technologies”, http://www.microsoft.com/com/tech/DCOM.asp

[5] J. Waldo, “The Jini Architecture for Network-centric Computing”, CACM, July 1999, p. 76-82, http://www.sun.com/jini

[6] J M. Smith, et al. “Activating Networks: A Progress Report”. IEEE Computer, Vol. 32, No. 4, p. 32-41, April 1999.

[7] S. Duarte, J. Legatheaux Martins, H. J. Domingos, N. Preguiça. “A Case Study on Event Dissemination in an Active Overlay Network Environment”. In Proceedings of Second International Workshop on Distributed Event-Based Systems (DEBS’03). June 2003.

[8] R. van Renesse, K. P. Birman, and S. Maffeis. “Horus: A flexible group communication system”. Communications of the ACM, 39(4):76--83, April 1996.

[9] M. Altherr, M. Erzberger and S. Maffeis. “iBus - A Software Bus Middleware for the Java Platform”. In International Workshop on Reliable Middleware Systems, p. 43-53, October 1999.

[10] P. Eugster, R. Guerraoui, J. Sventek. “Distributed Asynchronous Collections: Abstractions for Publish/Subscribe Interaction”. ECOOP, pp. 252-276, 2000.

[11] A. Carzaniga, D. S. Rosenblum and A. Wolf. “Achieving Scalability and Expressiveness in an Internet-scale Event Notification Service”. In Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC-00), July 2000.

[12] B. Segall, D. Arnold. “Elvin has left the building: A publish/subscribe notification service with quenching”. In Proceedings of AUUG97, Brisbane, 1997.

[13] G. Banavar et al. “An efficient multicast protocol for content-based publish-subscribe systems”. In the 19th IEEE International Conference on Distributed Systems (ICDCS’99), May 1999

[14] Yang-Hua Chu, Sanjay Rao, and Hui Zhang. “A case for end system multicast”. In Proceedings of ACM Sigmetrics, Santa Clara, CA, 2000

[15] J. Jannotti, D. Gifford, K Johnson, M.Kaashoek and J. O’Toole Jr.”Overcast: Reliable Multicasting with an Overlay Network”. In Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, pp 197-212. USENIX Association October 2000.

[16] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. “Almi: An application level multicast infrastructure.” In Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS), pages 49--60, 2001.