RAQUEL DE JESUS PINHEIRO

SISTEMAS DISTRIBUÍDOS BASEADOS EM EVENTOS: DISSEMINAÇÃO DESCENTRALIZADA DE NOTIFICAÇÕES

Dissertação apresentada ao Programa de Pós Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

CURITIBA
2013

RAQUEL DE JESUS PINHEIRO

SISTEMAS DISTRIBUÍDOS BASEADOS EM EVENTOS: DISSEMINAÇÃO DESCENTRALIZADA DE NOTIFICAÇÕES

Dissertação apresentada ao Programa de Pós Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de Mestre em Informática.

Área de concentração: Ciência da Computação
Orientador: Prof. Dr. Luiz Augusto de Paula Lima Jr.

CURITIBA
2013

Pinheiro, Raquel
SISTEMAS DISTRIBUÍDOS BASEADOS EM EVENTOS: DISSEMINAÇÃO DESCENTRALIZADA DE NOTIFICAÇÕES. Curitiba, 2013.

Dissertação - Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática.

1. DEBS - Roteamento Baseado em Conteúdo. 2. *Publish-Subscribe* – Roteamento Baseado em Conteúdo. 3. DEBS – Disseminação de Eventos. I. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e Tecnologia. Programa de Pós-Graduação em Informática.

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

PUCPR
GRUPO MARISTA

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

DEFESA DE DISSERTAÇÃO Nº 12/2013

Aos 05 dias do mês de Setembro de 2013 realizou-se a sessão pública de Defesa da Dissertação **"Sistemas Distribuídos Baseados em Eventos – Disseminação e Descentralização de Notificações"** apresentada pela aluna **Raquel de Jesus Pinheiro,** como requisito parcial para a obtenção do título de Mestre em Informática, perante uma Banca Examinadora composta pelos seguintes membros:

Prof. Dr. Luiz Augusto de Paula Lima Junior PUCPR (Orientador)	(assinatura)	Aprovado (Aprov/Reprov)
Prof. Dr. Alcides Calsavara PUCPR	(assinatura)	APROVADO (Aprov/Reprov)
Prof. Dr. Carlos Alberto Maziero UTFPR	(assinatura)	aprovado (Aprov/Reprov)

Conforme as normas regimentais do PPGIa e da PUCPR, o trabalho apresentado foi considerado aprovado (*aprovado/reprovado*), segundo avaliação da maioria dos membros desta Banca Examinadora. Este resultado está condicionado ao cumprimento integral das solicitações da Banca Examinadora registradas no Livro de Defesas do programa.

Prof. Dr. Mauro Sérgio Pereira Fonseca
Diretor do Programa de Pós-Graduação em Informática

Programa de Pós-Graduação em Informática
PPGIa
PUCPR

Agradecimentos

Gostaria de agradecer a minha família pelo apoio, incentivo, compreensão, motivação e carinho durante esse tempo (e tantos outros momentos), que passei no PPGIA. Aos meus pais, Adélio e Rose, por sempre me incentivarem nos estudos. À minha irmã Monique, pelo carinho, compreensão e incentivo. Aos meus irmãos, Leonardo e Marcelo, também pelo incentivo dado durante esse tempo.

Um grande agradecimento ao meu orientador, prof. Dr. Luiz Augusto de Paula Lima Jr., pela disposição e dedicação que sempre teve durante meu trabalho de pesquisa. Muito obrigada.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

P_S	Publicador de um conjunto S
S	Conjunto de tipos de eventos
τ	Tipo de evento
I	Inscrição
$I_i(x)$	Inscrição de índice *i* de um assinante
B	Broker
x	Assinante
Ep	Evento gerado por um Publicador P
N	Notificação
$N(P_s)$	Notificação de um Publicador
TR_τ	Tabela de roteamento de um tipo de evento
$E\tau$	Evento de um tipo
A_τ	Árvore de inscrição de um tipo de evento
$A_\tau(x)$.	Árvore de inscrição de um tipo de evento de interesse de um Assinante
Q	Mensagem de inscrição
C	Mensagem de cancelamento de inscrição
M_P	Mensagem de mudança: broker para publicador
M_B	Mensagem de mudança: publicador para broker

Lista de Abreviaturas

DEBS	Sistemas Distribuídos Baseados em Eventos (*Distributed Event Based System*)
DBR	Roteamento Baseado em Documento (*Routing Document-Based*)
CBR	Roteamento Baseado em Conteúdo (*Routing Content-Based*)
DF	Documento de "inundação" (*Documento Flooding*)
CAI	*Construção da Árvore de Inscrição*
CI	*Cancelamento de Inscrição*
RFID	Identificação por Radio Frequencia (*Radio-Frequency Identification*)
SII	Sistema de Invocação Implícita
XML	Linguagem de marcação extensível (*extensible Markup Language*)
REDS	Sistema Reconfigurável de Encaminhamento (*Reconfigurable Dispatching System*
OMG	Object Management Group
IGMP	Protocolo Internet de Gerenciamento de Grupo (*Internet Group Management Protocol*)
LAN	Rede Local (*local Area Network*)
SOA	Arquitetura Orientada a Serviço (*Service-oriented architecture*)
FIFO	Primeiro a entrar, primeiro a sair (*first-in-first-out*)
MANETs	*Mobile ad hoc Network*
ID	Identificador único (*identificator distinct*)
CAI	Criação da Árvore de Inscrição
CI	Cancelamento de Inscrição
NE	Notificação de Eventos

Resumo

Em sistemas Baseados em Eventos, onde clientes se inscrevem conforme seu interesse para receber eventos, a forma como é realizada a entrega de eventos, para os assinantes, é uma das funcionalidades de maior abordagem, visto que a o objetivo final desses sistemas é a entrega de eventos para clientes que registraram inscrições. No modelo clássico de sistemas Baseados em Eventos, para registrar interesse em determinado evento, um assinante envia sua inscrição para uma entidade intermediária, denominada *Broker*, onde esta realiza a comunicação entre um assinante e um publicador. Sendo assim, o destino para a entrega de notificações de eventos se torna conhecido explicitamente pelo *Broker*. Para não centralizar a entrega de notificações, nossa ideia é seguinte: fazer com que cada assinante e publicador desempenhe o papel de *Broker*, repassando inscrições e eventos para as partes interessadas, onde não haja o conhecimento explícito dos destinatários. Uma entidade pode ser *Broker* (somente repassa mensagens), Assinante e Cliente. Para toda entidade do sistema, temos o seguinte pressuposto: ao receber um evento, se a entidade não tem interesse, mas conhece outra entidade tenha interesse, então o evento é repassado. Definimos que para cada inscrição registrada referente a um Tipo de Evento, que a inscrição seja conhecida por todos os publicadores do mesmo tipo de evento da inscrição. É construída uma rota até a inscrição chegar a um publicador, onde cada entidade tem a informação de vizinhos que repassaram a inscrição. Dessa forma, quando um publicador emitir um evento, tal evento é entregue para as entidades vizinhas, onde cada entidade repassa o evento para seus vizinhos sucessivamente, até que o evento chegue ao assinante interessado. Sendo assim, nossa proposta é que eventos sejam disseminados somente dentro das rotas definidas no momento do registro de inscrições, evitando que todos os *Brokers* repassem eventos para assinantes que não tem interesse registrado. Com base em análises realizadas, temos um ganho no número de mensagens propagadas na operação de notificação de eventos.

Palavras-chave: DEBS – Protocolo de Roteamento Baseado em Conteúdo. Sistemas Distribuídos Baseados em Eventos. Roteamento Baseado em Conteúdo. DEBS – Notificação de Eventos.

Abstract

In events-based systems, clients subscribe according to their interest to receive events. The way that the delivery of events to subscribers is performed is one of the functionality of a better approach, once the ultimate goal of these systems is to deliver events to clients that registered interests. On the classical model of events-based systems, to register interest in a particular event, the subscriber sends your enrollment for an intermediate entity, called Broker, where it performs the communication between a subscriber and a publisher. Thus, the destination for deliveries of events notifications becomes explicit for the Broker. For not centralize the notifications delivery, our idea is the following: make each subscriber and publisher performs Broker's role, passing subscriptions and events for the interested parts, where it doesn't exist explicit knowledge of the recipients. An entity can be Broker (it only forwards messages), subscriber and client. For all systems entity, we have the following presupposition: at receiving an event, if the entity doesn't have interest, but it knows another entity that has interest, then the event is passed. We define that for each enrollment registered referring to an event kind that this enrollment is known by all publishers from same kind of the enrollment's event. Is built a route until the enrollment reach at a publisher, where each entity has information of the neighbors that passed the enrollment. Thusly, when a publisher issue an event, this event is delivered to neighbors entities, where each entity passes this event for its neighbors successively, until that the event reaches at the interested subscriber. Thusly, our proposal is that events can be disseminated only in routes defined upon registration from subscriptions, avoiding that every brokers passing events for subscribers that don't have registered interest. Based on analyses made, we get a gain in the number of propagated messages at the operation of the events notifications.

Keywords: DEBS - Routing Protocol Based on Content. Distributed Systems Based Events. Content Based Routing. DEBS - Event Notification.

Capítulo 1

Introdução

Com o desenvolvimento da computação, da interoperabilidade entre sistemas e da miniaturização de dispositivos, observa-se nos últimos anos uma crescente necessidade de sistemas ativos [GAL;HAD10-Cap.1]. Estes sistemas reagem a certos estímulos e podem também agir proativamente predizendo certos fenômenos. É o caso dos seguintes sistemas encontrados em [HINZE;BUCHMAN;SACHS09]:

- Gerenciador de Bagagens em aeroportos: O objetivo principal é fazer o transporte da bagagem para o destino correto. A bagagem é marcada no *Check-In*, com uma etiqueta *RFID*. Quando a bagagem passa por um leitor de etiquetas, um evento que contém a posição e tempo é gerado. O encaminhamento da bagagem é especificado e atualizado em tempo real. Por exemplo, só é permitido guardar a bagagem no avião, se a mesma tiver passado por todas as verificações de segurança com êxito.
- Detecção de Fraudes: Clonagem de cartão de crédito ou de celular são exemplos de fraude. Uso indevido é detectável sempre que o verdadeiro usuário tem um padrão de uso bastante regular. Um telefone celular que está sendo usado para fins comerciais com um padrão de chamada de um a cinco minutos durante o horário comercial, e que de repente mostra chamadas de longa duração à noite, mostra um comportamento fora

do comum. Eventos de registros de ações, como por exemplo, ligações para celular, podem ajudar a detectar as fraudes.

- Blogs[2]: Mineração e divulgação de informações conforme interesse em postagens de blogs. Por exemplo, um "*post*" sobre a fusão de duas empresas será complementado com informações sobre as duas empresas. A análise de texto pode ser processada para extrair outros eventos de alto nível e trazer os resultados de forma agregada para assinantes.

Tais sistemas ativos, que foram apresentados no parágrafo anterior, são baseados em "eventos". Qualquer acontecimento de interesse que pode ser observado a partir de um sistema é considerado um evento. Este pode ser um evento físico, como a presença de uma pessoa detectada por sensores, ou em geral, uma mudança de estado arbitrária detectável em um sistema [PIETZUCH06-01]. Eventos podem ser gerados externamente ao sistema ou podem ser inferidos pelo próprio sistema e devem de alguma forma chegar a componentes que têm interesse naquele tipo de evento. Quando "sistema" é citado, referência-se então um Sistema Distribuído Baseado em Eventos - *Distributed Event Based Systems* (DEBS). Este é definido como sendo um Sistema de Invocação Implícita (SII), composto por componentes distribuídos que interagem uns com os outros usando apenas eventos. Os eventos são gerados por componentes chamados *publicadores (publishers)*. Componentes interessados nos eventos que foram gerados, são chamados de *assinantes (subscribers)*. Um assinante é notificado quando um acontecimento de seu interesse é gerado por um publicador [HINZE;BUCHMANN10-Cap.2]. Um sistema baseado em eventos é constituido dos seguintes elementos: Eventos e notificações como meio de comunicação, publicadores e assinantes como componentes que interagem, inscrições significando interesse do assinante em determinadas notificações e o serviço de notificação de eventos, que é o responsável por entregar notificações entre publicadores e assinantes [PIETZUCH06-01].

Alguns provedores de busca na Internet tem implementado instalações (geralmente são servidores alocados juntamente com servidores que centralizam e disseminam notícias) baseados em conteúdo, que podem entregar mensagens de alerta aos usuários, contendo mensagens que atendam a um conteúdo recém publicado a usuários que tenham determinadas

---

2 Página de Internet com características de diário, atualizada regularmente.

inscrições definidas. A entrega do conteúdo em geral é via e-mail ou mensagens de celular [LEGATHEUX;DUARTE10]. Obviamente, o método que consiste na divulgação de eventos no ambiente distribuído para todos os "assinantes" que previamente registraram o seu interesse, possui vantagens sobre a entrega via e-mail ou celular, onde o destino é conhecido e implícito.

Um trabalho que consideramos para acrescentar na abordagem em DEBS, apesar de que não incluímos em nosso trabalho, é o modelo de Campos Magnéticos Virtuais proposto em [LIMA;CALSAVARA10]. Consideramos que tal modelo possui características que podem ser empregadas em DEBS. A principal semelhança encontrada é que a disseminação de mensagens é realizada por nós de origem que não tem conhecimento explícito dos nós destinatários. Outra característica é que os nós recebem mensagens conforme sua força de atração e no contexto de DEBS, assinantes poderiam enviar suas inscrições para Publicadores que tenham maior força de atração. O magnetismo é uma idéia para ser analisada e avaliada posteriormente para ser aplicada no contexto de nosso trabalho.

A fim de propor soluções em uma rede estática para Sistemas Baseados em Eventos, este trabalho tem como objetivo propor métodos de disseminação de mensagens na Notificação de Eventos, juntamente em operações comuns de DEBS (estabelecimento de "*links"* de conexões na rede, criação e cancelamento de inscrição, troca de papel - publicador e *broker*). Tais métodos propostos neste trabalho, consistem na adaptação de idéias e conceitos de métodos existentes ([MAYER10], [CUGOLA06], [BALDONI05], [KOOSHA10], [YOO09] e [SANTORO06-cap.2]) de disseminação de eventos, a fim de propor vantagem na forma de roteamento para entrega de mensagens na rede em comparação com abordagens já existentes.

1.1 Desafio

No contexto do modelo de *Inscrições de Eventos*, encontrado em [HINZE;BUCHMANN10-cap.2], o qual representa "*Publish-Subscribe*", o objetivo deste trabalho é de fazer com que ocorra melhor desempenho na disseminação, troca e entrega de mensagens na rede. O intuito é aplicar novos algoritmos de disseminação de mensagens na concepção de DEBS, aonde os Publicadores de eventos ou nós intermediários responsáveis

pela entrega, não sejam implícitos para os assinantes. O objetivo é de melhorar a rota para a entrega de mensagens aos destinatários, trazendo possivelmente uma redução no número de troca de mensagens durante a chamada de todas as operações abordadas neste trabalho: criação e cancelamento de inscrições, notificação de eventos e manutenção da rede (conforme mudança de papel definida).

1.2 Motivação

Como muitos dos trabalhos sobre DEBS utilizam a noção de Roteamento Baseado em Conteúdo (CBR) e alguns destes trabalhos tem o objetivo de melhorar os métodos de Correspondência e Roteamento, vimos uma oportunidade de estudar a disseminação de eventos distribuídos com possíveis ganhos de desempenho da montagem e percurso de rota de destinatários, juntamente com ganho no número de troca de mensagens. Observou-se uma série de potenciais cenários onde eventos podem ser aplicados:

- Acidentes em rodovias: Uma ambulância de emergência pode disponibilizar um serviço de alerta enquanto está a caminho do local da ocorrência da emergência. Outros veículos podem assinar este serviço, a fim de receberem avisos de ambulâncias que estão próximos para que os mesmos tenham ciência (colaborem, tenham uma reação ao evento) que durante seu percurso uma ambulância irá cruzar seu caminho [MAYER10].
- Controle de tráfego aéreo: Difusão de eventos relevantes, tais como localizações periódicas de aviões, pouso e decolagens. Interessados em vôos podem solicitar avisos de vôos de determinada companhia, horário, destino e preço [LEGATHEUX;DUARTE10].
- Disseminação de notícias: Notícias que são geradas por agências de notícias locais são distribuídas para empresas interessadas que desejam receber apenas informações relevantes [PIETZUCH06-01].

1.3 Proposta

As abordagens dos trabalhos relacionados, descritos no capítulo três, envolvem soluções de roteamento. Alguns dos objetivos abrangem: minimizar a taxa de entrega de mensagens, diminuir o número de troca de mensagens, otimizar a manutenção da rede e disponibilizar um protocolo para a realização de roteamento. Com base nos trabalhos relacionados, a ideia é levantar os pontos positivos e falhos de cada protocolo, avaliando a necessidade da proposta em si, juntamente com os conceitos de cada um, para incorporar uma solução que trate falhas em específico, como o número de troca de mensagens, manutenção e roteamento.

Sendo assim, este trabalho propõe um protocolo para a disseminação de eventos, onde todo nó na rede deve ter o papel de disseminador [CAO;SINGH04] e [KOOSHA10], sendo capaz de encaminhar a mensagem até que os nós interessados recebam a mesma. A arquitetura da rede é de um grafo não-dirigido formado por árvores referentes a inscrições, onde os nós são conectados a outros nós que tenham correspondência entre o conteúdo de cada inscrição e o modelo de notificação. O conceito é semelhante a abordagem vista em [YOO09], onde os nós de mesmo interesse ficam próximos uns dos outros, formando um grupo, para que mensagens sejam encaminhadas dentro de cada grupo. Para definição do protocolo que será apresentado no capítulo 4, será necessário propor as seguintes operações na rede:

- Conectividade entre nós – criação de árvores referentes a cada inscrição;
- Criação e cancelamento de inscrições;
- Notificações de eventos;
- Mudança de papel: *Broker* para Publicador e vice-versa.

1.4 Contribuição

Este trabalho visa às seguintes contribuições:

- Proposta de algoritmos para que o menor número possível de mensagens seja trocado na entrega nas operações de DEBS;

- Adaptação do algoritmo de Árvore de Abrangência [SANTORO06-cap.2] para a criação de conexões na rede sobreposta (“*overlay*”);
- Avaliação de custo e desempenho na troca de mensagens na rede. Comparação com resultados de trabalhos relacionados.

1.5 Organização

O conteúdo desta dissertação está organizado da seguinte forma: no capítulo 2 é apresentada uma visão geral de Sistemas Distribuídos Baseados em Eventos, tais como conceitos, terminologia, arquiteturas, modelos, métodos de correspondência, onde todos os tópicos são necessários para abordagem de DEBS. No capítulo 3 são apresentados, de forma específica, conceitos e métodos que realmente fazem parte do foco do trabalho: o roteamento de mensagens. São detalhados algoritmos básicos vistos na literatura, assim como algoritmos de trabalhos relacionados. No capítulo 4 é apresentada a proposta deste trabalho, onde são detalhados os todos os algoritmos propostos. No capítulo 6 apresentamos análises realizadas sobre os algoritmos propostos, com base nas teorias de grafos e árvores. No capítulo 6 apresentamos o “*framework*”, onde os algoritmos são implementados, juntamente com os resultados de experimentos realizados. Por fim, no capítulo 7 apresentamos as conclusões e sugestões, bem como a continuidade em pontos específicos deste trabalho.

Capítulo 2

Sistemas Baseados em Eventos

Sistemas Distribuídos Baseados em Eventos constituem-se de Eventos e Notificações desses Eventos por meio de comunicação, entre duas entidades principais, que compõem o sistema: Publicador e o Assinante. O Publicador produz eventos e o Assinante recebe uma notificação (via mensagem) da ocorrência de tais eventos, conforme seu interesse registrado. Fazendo-se uma analogia, assinantes se inscrevem para receber novidades de publicadores, semelhante à noção de inscrições para receber notícias de uma revista, por exemplo. O principal componente do sistema é o Serviço de Notificação de Eventos[1], sendo o intermediador (*middleware*) entre Publicador e Assinante, conforme a figura 2.

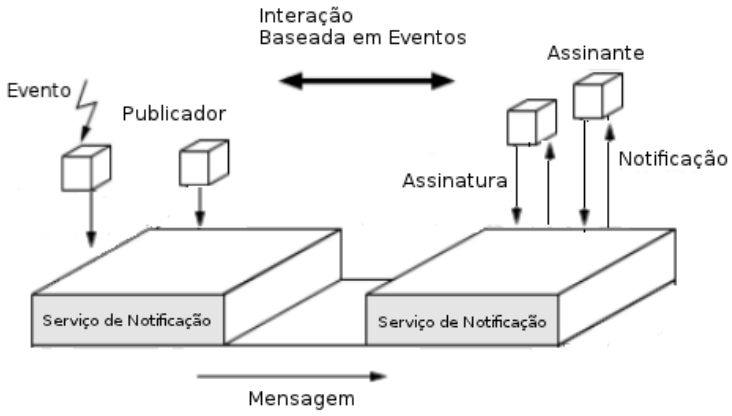

Figura 2.1: Interação em Sistemas Baseados em Eventos. Figura adaptada [PIETZUCH06-01].

[1] Também definido como *Distribuidor* (“*Dispatcher*”) [PIETZUCH06], [PICCO08], [BALDONI05].

O Serviço de Notificação de Eventos é o responsável por encaminhar notificações de eventos (que foram emitidos por publicadores) para assinantes que enviaram inscrições registrando interesse em determinado evento.

Nas seções seguintes, para formalização, são detalhados alguns fundamentos sendo em seguida apresentadas definições adotadas com base em terminologias, modelos e mecanismos de Sistemas Baseados em Eventos. O intuito deste capítulo é apresentar noções básicas necessárias para a definição de Sistemas Distribuídos Baseados em Eventos.

2.1 Terminologia

Em um contexto de software de um Sistema Distribuído Baseado em Eventos, os componentes constituintes são Publicadores, Assinantes e o Serviço de Notificações, os quais atuam na emissão e recebimento de Notificações.

2.1.1 Publicador e Assinante

Publicadores são componentes que emitem notificações da ocorrência de seus próprios eventos. O Publicador é quem observa seus próprios Eventos sendo sua principal função a de enviar notificações para o Serviço de Notificação, sem ter o conhecimento de quais são os Assinantes interessados. Um Assinante recebe as notificações que são entregues pelo Serviço de Notificação. Estes enviam inscrições para o Serviço de Notificação, apenas indicando seus interesses em determinadas classes de Eventos. Tanto o envio de inscrições como no recebimento de Notificações, o Assinante não tem conhecimento do respectivo Publicador.

2.1.2 Filtros e Inscrições

Uma inscrição[2] é uma estrutura de dados que define quais são as notificações que um assinante está interessado em receber. Assinantes registram seu interesse em receber certos tipos de notificações, através do envio de inscrições para o Serviço de Notificação. Inscrições são compostas por filtros, que consistem basicamente em funções booleanas com valores para o Serviço de Notificação testar se os filtros da inscrição são verdadeiros em comparação com a estrutura da Notificação. No entanto, as inscrições podem ser compostas por outros dados (como *metadados*, por exemplo) para complementação de informações referentes ao interesse de um Assinante.

2.1.3 Serviço de Notificação de Eventos

O Serviço de Notificação de eventos é o mediador que separa Publicadores de Assinantes, o qual disponibiliza uma interface para interação entre Publicadores e Assinantes como operações de *publicação/inscrição*. O serviço é responsável por entregar notificações recebidas para todos os Assinantes que tenham registrado assinaturas correspondentes, ou seja, recebendo uma notificação, este tem a tarefa de identificar os interessados e assim entregar a notificação. Sendo assim, o serviço avalia as assinaturas recebidas, fazendo a entrega de notificações que correspondam a inscrições dos Assinantes.

2.2 Mecanismos de Filtragem em Notificações

Nesta seção, são descritos os mecanismos de comunicação vistos na literatura [PIETZUCH06], os quais são modelos de comunicação em DEBS. Cada mecanismo é ideal em contextos nos quais a necessidade seja atendida por estes. Nosso objetivo nessa seção, não é de comparar ou indicar qual é o mecanismo mais adequado, mas sim o de apenas mostrar a

[2] O termo *inscrição* também é definido, na literatura [PIETZUCH06], como “Assinatura”.

definição e funcionamento de cada um. Este trabalho abordará aplicações de disseminação de eventos Baseada em Conteúdo.

2.2.1 Canais

Um Canal é a forma mais simples de identificar conjuntos de notificações. Neste modelo, os Publicadores escolhem um determinado canal no qual as notificações serão enviadas e os Assinantes enviam suas inscrições para um determinado canal conforme seu interesse. Notificações publicadas em um canal são entregues a todos os assinantes deste canal.

2.2.2 Filtragem Baseada em Assunto

Na filtragem por assunto, cada notificação é parte de uma hierarquia de assuntos, que forma uma árvore. É aplicado o uso de cadeia de caracteres (“*strings)*” para seleção da notificação. Publicadores armazenam a “*string*” do assunto de cada notificação que representa o caminho na árvore de assuntos. Por exemplo, uma aplicação de bolsa de valores publica novas cotações da empresa 'XXX' no tema de tecnologia. A inscrição para receber notificações da empresa ‘XXX’ seria a seguinte: '/Exchange/Europe/NewYork/Technology/XXX’. Assim, um Assinante que registra uma inscrição com o conteúdo '/Exchange/Europe/NewYork/Technology/*', este recebe todas as notificações de cotações do tema de tecnologia independente da empresa (utiliza ‘*’ para indicador ‘*todos’*). Conforme mais profundo é o nível da árvore, mais restrito é o critério de seleção da notificação.

2.2.3 Filtragem Baseada em Conteúdo

No modelo Baseado em Conteúdo, assinantes registram seu interesse especificando condições sobre o conteúdo das notificações que deseja receber. Sendo assim um filtro de

uma inscrição é uma consulta formada por um conjunto de restrições sobre os valores dos atributos da Notificação. Na entrega de notificações para assinantes, as notificações são entregues de acordo com a comparação entre os filtros da inscrição e o conteúdo da notificação [BALDONI; VIRGILLITO05].

2.3 Modelos Baseados em Conteúdo e Correspondência

Modelos de correspondência, aplicados para comparações entre notificações e inscrições de eventos, estão diretamente ligados a função de roteamento em DEBS. Informalmente, um modelo de dados define a forma como o conteúdo das notificações está estruturado, enquanto o modelo de filtro define como as inscrições podem ser especificadas. A forma como é avaliado se um filtro corresponde à notificação é comparação de predicados do filtro sobre o conteúdo da notificação. O modelo de filtro sempre depende do modelo de dados da notificação [PIETZUCH06]. Nesta seção são apresentados modelos de correspondência de forma resumida, a fim de se ter a noção de tais métodos de correspondência, visto que o foco do trabalho não é de otimização de correspondências entre inscrições e notificações, e sim o roteamento.

2.3.1 Tuplas

Uma notificação é uma tupla, onde esta é um conjunto ordenado de atributos, sendo cada atributo uma “string”, indicando um *valor*. Notificações e inscrições seguem um “*template*”, onde a ordem de atributos sempre será a mesma. Os atributos da notificação e da inscrição são comparados uns com os outros de acordo com sua posição. Por exemplo:

- A notificação *(“Vôo”, “PR”, 10:45)* corresponde à inscrição *(“Vôo”, “PR”, *)*

A deficiência encontrada neste modelo é a comparação por posição, pois pode ser considerada inflexível para os assinantes, devido ao fato de que nenhum atributo pode ser opcional. Em contrapartida, o caractere '*' (asterisco) significa “todos”, análogo ao “*” na linguagem SQL em Banco de Dados. Nenhum atributo na tupla pode ser opcional, mas é possível incluir um

atributo com o significado “traga qualquer valor, em Notificações, que tenha na posição desse atributo.

2.3.1 Registros Estruturados

No modelo de Registros Estruturados, uma notificação *n* é um conjunto não-vazio de atributos, onde cada atributo é único. Uma notificação *n* é um conjunto de atributos *{a1, ... , an}* onde cada *ai* é um par com *nome/valor*: *(ni, Vi)* de nome *ni* e valor *Vi*.Assume-se que cada atributo da notificação é único: $ai \neq aj \Rightarrow ni \neq nj$ e que existe uma função que mapeia cada notificação *ni* para um Tipo *Tj*, onde *Tj* corresponde ao valor *Vi*.

Um filtro *F* é uma função booleana, sem estado, que é aplicada a uma notificação *n.* Geralmente *F* se constitui de uma expressão booleana que consiste em predicados que são combinados por operadores booleanos.

- Uma notificação *n* corresponde a *F* se *F(n)* retornou *verdadeiro*.
  $F \rightarrow \{verdadeiro, falso\}$
- Um atributo de um filtro é um simples filtro que impõe uma limitação sobre o valor de um único atributo. Por exemplo: *{companhia = GOL}*.
- A função de Filtro é definida por um conjunto de três atributos *Ai* = (*ni, Opi, Ci*), onde *ni* é o nome do atributo, *Opi* é o operador de teste e *Ci* é um conjunto de constantes que pode ser vazio. O atributo *ni* determina a que atributo da notificação a restrição se aplica. Se a notificação não contém um atributo com o nome *ni*, em seguida, *Ai* retorna *falso*. Caso contrário, o operador *Opi* é avaliado usando o valor do atributo e o conjunto especificado de constantes *Ci*. O resultado de *Ai* pelo resultado de *Opi* (que também avalia *verdadeiro* ou *falso*). Por exemplo, temos a notificação e inscrição seguintes:

Notificação = {(compra = “Passagem”) Λ *(empresa = “GOL”)* Λ *(destino = “PR”)}*
Inscrição = {(compra = “Passagem”) Λ *(empresa = “GOL”)* Λ *(destino = “PR”)}*

O teste de todos os filtros na notificação retorna verdadeiro para o conjunto de filtros aplicados: *F = A1 Λ A2 Λ ... Λ An.* Portanto, uma notificação *n* corresponde a um filtro *F* se todos os atributos filtros de *F* forem equivalentes aos atributos de *n.*

2.3.2 Registros semi-estruturados

O modelo semi-estruturado se distingue do modelo de registros estruturado principalmente pelo seguinte fato: no modelo estruturado, os nomes de atributos são únicos e portanto, um nome de atributo representa exclusivamente um único atributo. Ao contrário, no modelo semi-estruturado cada atributo pode ter o mesmo nome o qual define os nomes de atributos. Uma notificação é um documento XML, que consiste em um conjunto de elementos, que são organizados em uma hierarquia onde um único elemento é o elemento raiz, chamado de "notificação". Cada elemento que consiste em um conjunto de atributos cujos nomes devem ser únicos, e um conjunto de elementos filhos que não são necessariamente distintos. A figura 2.1 mostra um exemplo de modelo de notificação de um leilão de peças de um computador.

```
<notificacao>
    <leilao data_fim="01-02-2013 18:00" preco_minimo="900,00" >
    <vendedor nome="xxx" identificador="1234" />
    <item>
        <processador fabricante="AMD"
            modelo="Phenom II"
            memoria_cache="1 MB"
            clock="3.0" />
    </item>
    </leilao>
</notificacao>
```

Figura 2.2 – Modelo de notificação baseado em XML. Figura adaptada [PIETZUCH06-02].

Um atributo *A* é um par (*ni*, *vi*) com nome *ni* e valor *vi*. Os nomes dos atributos devem ser únicos em relação aos elementos. Cada um dos filtros seleciona um subconjunto de elementos

de uma notificação. Para tal seleção, restrições sobre os atributos dos elementos selecionados são aplicadas, o qual consiste de um conjunto de filtros de atributo.

2.4 Conclusão

Este capítulo apresentou conceitos e definições para possibilitar uma visão global de todos os componentes constituintes de um Sistema Distribuído Baseado em Eventos. Com base em tais conceitos e definições, adotamos para nosso trabalho as seguintes abordagens:

- Filtragem Baseada em Conteúdo: adotamos esse método de filtragem, pois em comparação com os demais mecanismos vistos, este método possibilita uma forma dinâmica de comunicação na rede;
- Correspondência: devido ao fato de que o foco do nosso trabalho é o Roteamento de mensagens, os métodos de correspondência entre inscrições e notificações não são abordados. No entanto, como correspondência é essencial no funcionamento da Filtragem Baseada em Conteúdo, utilizamos a estratégia de correspondência disponível no "*framework*" REDS [CUGOLA; PICCO; 05].
- Modelos Baseados em Conteúdo e Correspondência: Para a troca de mensagens na rede, optamos pelo modelo de Tuplas. Em comparação com os modelos de Registros Estruturados e Semi-Estruturados, este modelo não é a melhor solução para ser aplicada, pois não há flexibilidade para preenchimento de atributos em inscrições e notificações de eventos. Como o objetivo do nosso trabalho é propor algoritmos voltados para o roteamento e disseminação de mensagens e não a o Conteúdo e Correspondência, aplicamos em nosso trabalho a definição do modelo de Tuplas.

Capítulo 3

ROTEAMENTO DISTRIBUÍDO DE NOTIFICAÇÕES

A estratégia do roteamento distribuído é determinar, por um algoritmo baseado em conteúdo, quais são as notificações que serão encaminhadas e como a filtragem de notificações é realizada para a disseminação de notificações de eventos [MÜHL02]. Neste capítulo serão apresentados algoritmos aplicados em operações de inscrições e notificação de eventos [PIETZUCH06-cap.2], como o roteamento baseado em Identidade, baseado em Abrangência e roteamento baseado em "*Merge*", onde tais algoritmos são aplicados em sistemas de redes "*Broker*" (sistemas que tenham a mesma arquitetura e características do sistema Rebeca [PIETZUCH06-09]). Após a definição de algoritmos para operações de inscrições e notificação de eventos, procuramos apresentar a abordagem de protocolo *multicast* para disseminação de eventos. Em seguida apresentamos serviços de notificações existentes definidos e desenvolvidos em projetos de pesquisa. Por último, apresentados trabalhos de artigos relacionados onde os algoritmos de disseminação e manutenção da rede são aplicados em redes de arquiteturas diferentes de redes de arquitetura "*Broker*".

3.1 Algoritmos de Roteamento: Inscrição e Notificação

Em uma rede em que nós específicos, chamados "*Brokers*", são responsáveis por repassar mensagens, cada nó *Broker* realiza o gerenciamento de criação e cancelamento de inscrições. Para este gerenciamento, cada Broker conta com uma tabela onde armazena os filtros das respectivas inscrições que nós Assinantes registram ou cancelam conforme seu interesse. Nas próximas seções são detalhados algoritmos para disseminação de mensagens entre Brokers, para operações de inscrições. A seguir, algumas definições são apresentadas, as quais são citadas nas seções seguintes:

- *No:* É o nó que está realizando a operação de criação/cancelamento de inscrição;
- T_B: Tabela de roteamento de um nó Broker;
- *S ("subscribe"):* É o conjunto de inscrições em T_B;
- *U ("unsubscribe")*: Conjunto de cancelamentos de inscrições em T_B;
- *F*: Função de filtro (que consiste de um conjunto de atributos vistos no capítulo 2 seção 2.1.2).

3.1.1 Flooding

"*Envio de mensagem para todos os nós"* é o que ocorre na disseminação por "*Flooding*". A tabela de roteamento de cada Broker B é inicializada com o conjunto $\{(F_T, U) \mid U \in N_B\}$ na inicialização do sistema, onde $F_T(n) = true$ para todo n ∈ N. Como $N(F_T) = N$, isto implica que um *Broker* ao receber uma notificação de um cliente local, ou de um *Broker* vizinho, este encaminha a notificação para todos os seus vizinhos. Pelo fato da topologia ser um grafo acíclico e conectado, esta estratégia garante que cada notificação seja processada exatamente uma vez por cada Broker. O algoritmo segue abaixo:

1. Procedimento administra (*No_Solicitante Y, Conjunto S, Conjunto U*)
2. Início
3. $T_B \leftarrow T_B \cup \{(F, Y) \mid F \in S\}$;

4. $T_B \leftarrow T_B \setminus \{(F, Y) \mid F \in U\}$;
5. Retornar (Ø, Ø);
6. Fim

Um *Broker* ao receber uma notificação atualiza sua tabela de roteamento:

- Se o nó Y emitiu uma inscrição *F*, então o broker adiciona (*F*, *Y*) em sua tabela de roteamento (linha 3).
- Se um nó Y cancelou uma inscrição *F*, o Broker exclui (*F*, *Y*) da sua tabela de roteamento (linha 4).

A disseminação por *Flooding* é a maneira mais simples de propagação de mensagens, onde todos os nós do grafo recebem todas as mensagens de interação (inscrição, notificação de eventos e de manutenção da rede).

3.1.2 Roteamento Simples

No Roteamento Simples, todos os Brokers têm conhecimento de todas as inscrições registradas na rede. Nessa estratégia é realizada a propagação, por *Flooding*, de filtros de inscrições, na criação e cancelamento de inscrições, para todos os Brokers. Assim, cada Broker mantém sua tabela de roteamento atualizada. Para a entrega de notificações, á aplicado o Roteamento Baseado em Conteúdo, onde uma notificação é entregue somente para assinantes que tenham inscrições registradas correspondentes ao conteúdo da notificação. Conforme mostrado na figura 3.1, o relacionamento entre Assinantes e Brokers é de um para um.

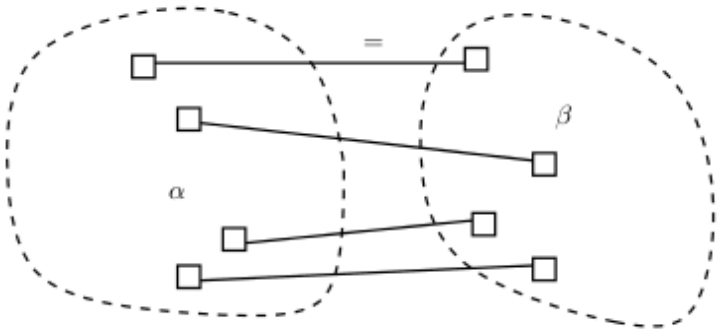

Figura 3.1 – Associação entre assinantes e brokers. Figura adaptada [PIETZUCH06-04].

Inicialmente, a tabela de roteamento T_B de cada Broker B é inicializada com *vazio* Ø.
O algoritmo funciona da seguinte forma:

Procedimento admin(No_Solicitante *Y*, *Conjunto S*, *Conjunto U*)
Início
$T_B \leftarrow T_B \cup \{(F, Y) \mid F \in S\}$;
$T_B \leftarrow T_B \setminus \{(F, Y) \mid F \in U\}$;
$M_S \leftarrow \{(F, H) \mid H \in NB \setminus \{Y\} \wedge F \in S\}$;
$M_U \leftarrow \{(F, H) \mid H \in NB \setminus \{Y\} \wedge F \in U\}$;
Retornar (M_S, M_U);
Fim

Quando um cliente realiza uma operação de inscrição *sub*(*Y*, *F*) ou uma operação de cancelamento de inscrição *unsub*(*Y*, *F*), as operaçãos *administrar*({F}, Ø) e *administrar*(Ø, {F}) são chamadas respectivamente entre Brokers, conforme a figura 3.2.

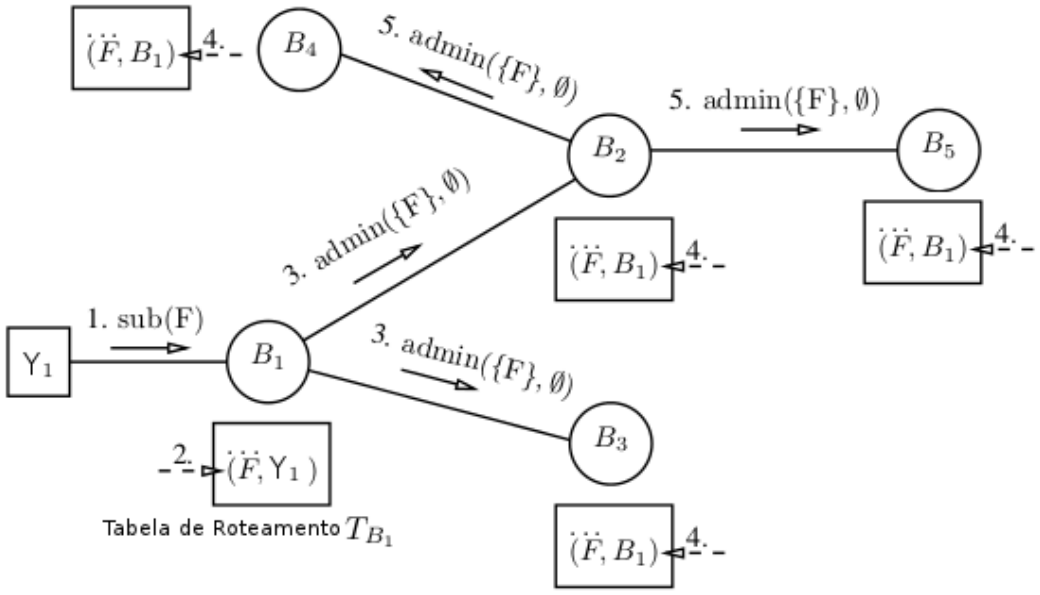

Figura 3.2 – Disseminação de nova inscrição. Figura adaptada [PIETZUCH06-04].

- Se o nó *Y* emitiu uma inscrição *F* então o broker adiciona (F, Y) na tabela de roteamento (linha 3).

- Se o nó *Y* emitiu o cancelamento da inscrição *F* então o broker remove (F, Y) na tabela de roteamento (linha 4).
- Para cada vizinho *H* exceto o nó solicitante *Y*, uma tupla (*F*, *H*) é retornada para cada criação e para cada cancelamento de inscrição *F* (linhas 5 e 6). Então a inscrição/cancelamento é enviada para todos os Brokers vizinhos, exceto o *Y* que é o nó solicitante.

3.1.3 Roteamento Baseado em Identidade

A fim de evitar o conhecimento global de inscrições ("*Flooding*" e Roteamento Simples*)* e encaminhamento desnecessário de inscrições/cancelamentos entre *Brokers*, o algoritmo baseado em identidade repassa inscrições para outros *Brokers* levando em conta as semelhanças entre estas. A ideia é a seguinte: o conjunto de notificações que um *Broker Bi* encaminha para um *Broker Bj*, é o conjunto de todas as notificações que são correspondidas por qualquer registro de roteamento (*F*, *Bj*) na tabela de roteamento T_{Bi}. Podem existir dois registros de encaminhamento (*F*, *Bi*) e (*G*, *Bi*) com *N*(*F*) = *N*(*G*) sendo um desses registros suficiente para a entrega de notificações tanto para *F* quanto para *G*.

Identidade conjuntiva de filtros: Verifica quais filtros são iguais, referente a uma notificação *n*, realizando a junção desses filtros onde estes são iguais. Por exemplo: os filtros $F1 = A1i \Lambda ... \Lambda An$ e $F2 = A2j \Lambda ... \Lambda Am$ são conjunções pelo fato de *F1* e *F2* terem o mesmo número de filtros e de atributos e que $\forall A1i \exists A2j$, onde $A1i \equiv A2$.

- A criação e cancelamento de uma inscrição são encaminhados somente para um vizinho se não houver inscrição idêntica em sua tabela de roteamento;
- Um cancelamento de inscrição não é encaminhado para um Broker vizinho se houver outra inscrição de um cliente local ou outro vizinho que é idêntica a inscrição a ser cancelada.

Procedimento admin(*No_Solicitante Y, S, U*)
Início
$M_S \leftarrow \emptyset$;
$M_U \leftarrow \emptyset$;
Para todos os filtros $F \in S \cup U$ faça
  Se $Y \in N_B$ então
    $T_B \leftarrow T_B \setminus CB(F, Y)$;
  Senão
    $T_B \leftarrow T_B \setminus \{(F, Y)\}$;
  FimSe
  $A \leftarrow \{(F, H) \mid H \in D_B(F) \setminus \{Y\}\}$;
  Se $F \in U$ então
    $M_U \leftarrow M_U \cup A$;
  Senão
    $M_S \leftarrow M_S \cup A$;
    $T_B \leftarrow T_B \cup \{(F, Y)\}$;
  FimSe
FimPara
Retorna (M_S, M_U);
Fim

Um Broker *B* recebendo uma criação ou cancelamento de inscrição, verifica o seguinte:
Atualiza sua tabela de roteamento:

- Se *Y* for um Broker vizinho, então o broker B remove todos os regisros de roteamento nos quais os filtros F sejam idênticos ao filtro do nó *Y* (linha 7)
- Se *Y* é um cliente local, B remove apenas (F, *Y*) correspondente ao nó *Y* (linha 9)

B repassa *F* para todos os seus vizinhos, exceto para o nó *Y* que é o nó solicitante (linha 11).
Por último, se *F* é uma inscrição, o broker *B* adiciona o filtro (*F*, *Y*) em sua tabela de roteamento (linha 16). Na figura 3.4 o broker *B1* recebe uma nova inscrião *F* do nó *S*. O broker *B1* adiciona (*F*, *S*) em sua tabela de roteamento, encaminha a inscrição para os brokers *B2* e *B3* e registra em sua tabela para quais Brokers enviou a inscrição para posteriormente verificar se já enviou ou não enviou uma inscrição idêntica.

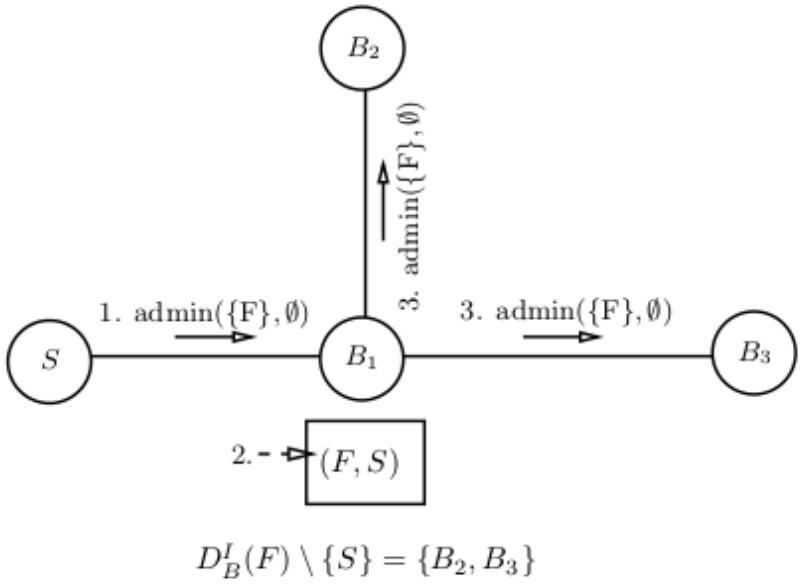

Figura 3.4 – Processamento uma nova inscrição de um vizinho. Figura adaptada [PIETZUCH06-05].

Já na figura 3.5, o broker *B1* também adiciona (*F*, *S*) porém só encaminha a nova inscrição *F* para *B3* que é seu único vizinho com F ≡ F.

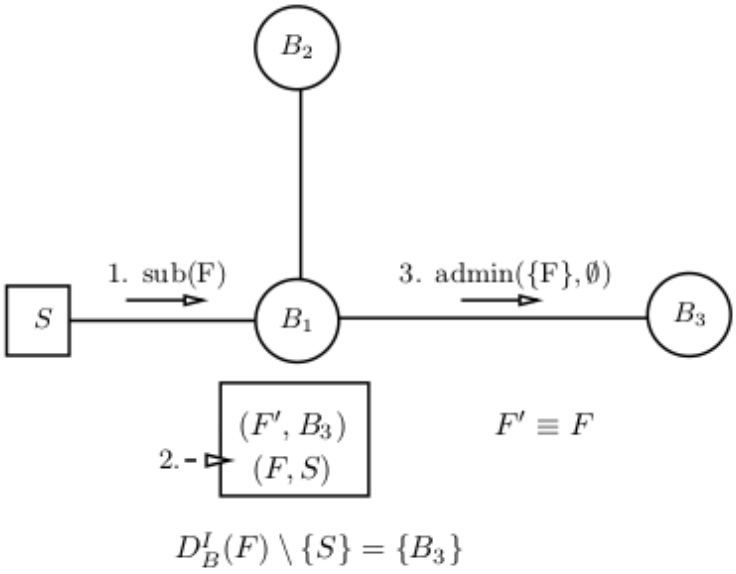

Figura 3.5 - Processamento uma nova inscrição de um cliente - Figura adaptada [PIETZUCH06-05].

3.1.4 Roteamento Baseado em Abrangência

A abordagem no algoritmo de encaminhamento baseado em Abrangência é de reduzir (ainda mais do que no encaminhamento baseado em Identidade) os registros redundantes e desnecessário no encaminhamento de criação ou cancelamento de inscrições. A verificação de abrangência é conforme o seguinte: os filtros $F1 = A1i \wedge ... \wedge An$ e $F2 = A2j \wedge ... \wedge Am$ são conjunções, mantendo $\forall i \; \exists j$ onde $A1i \sqsupseteq A2j$ implica $F1 \sqsupseteq F2$. Se uma notificação *n* corresponde a *F2*, isso implica que todos os filtros de *F1* também serão correspondentes. Se os filtros de *A2j* não tiverem abrangência, a notificação é entregue somente para os filtros de *A1i*. Por exemplo, $\{x \in [5, 8]\}$ cobre $\{x \in [4, 7] \wedge x \in [6, 9]\}$, embora $\{x \in [5, 8]\}$ não faz abrangência de $\{x \in [4, 7]\}$ e nem de $\{x \in [6, 9]\}$. O encaminhamento da notificação é decidido conforme *F2* e não *F1*.

- Uma inscrição não é encaminhada para um vizinho se já existe uma inscrição em sua tabela de roteamento que tenha cobertura e que já tenha sido encaminhada anteriormente.
- Um broker recebendo uma inscrição, este remove todos os registros de roteamento de sua tabela que tem cobertura/abrangência da inscrição recebida.
- Um cancelamento não é encaminhado para um vizinho se houver uma inscrição de um cliente local ou outro vizinho que cobre a inscrição que será cancelada.
- Se um cancelamento for enviado a um vizinho, o *Broker* que enviou também encaminha um subconjunto vazio possivelmente de inscrições que não estão na abrangência da inscrição. Isto é feito para garantir a entrega de notificações que correspondem a essas inscrições existentes.

Nova inscrição

Se um broker *B* recebe uma nova inscrição *F* de um vizinho ou de um cliente local *Y*, primeiro o broker *B* atualiza sua tabela de roteamento: se *Y* é um vizinho então *B* remove todos os registros da tabela de roteamento de filtro que são iguais a *F* e que se referem a *Y*, para evitar filtros obsoletos. Se *Y* é um cliente local, o broker remove apenas (*F*, *Y*). Em seguida B encaminha *F* para todos os seus vizinhos exceto *Y*. Por último *B* inclui (F, *Y*) em sua tabela de roteamento.

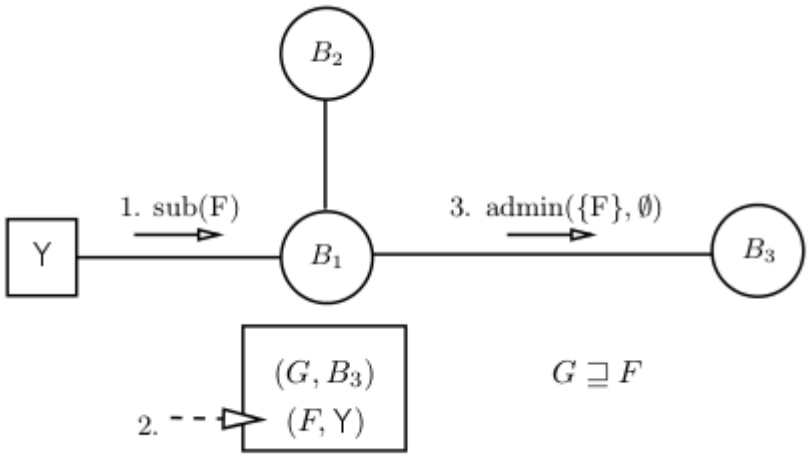

Figura 3.6 – Nova inscrição de um cliente. Figura adaptada [PIETZUCH06-06].

Como exemplo de processamento de nova inscrição enviada por um cliente, no cenário da figura 3.6 o cliente *Y* envia uma inscrição para o Broker *B1*, no qual este encaminha *F* somente para *B3* que é o único Broker que tem inscrição na qual tem abrangência pela inscrição recém enviada a *B1*. Na figura 3.7 é demonstrado um cenário onde um vizinho Broker envia uma inscrição para *B1*.

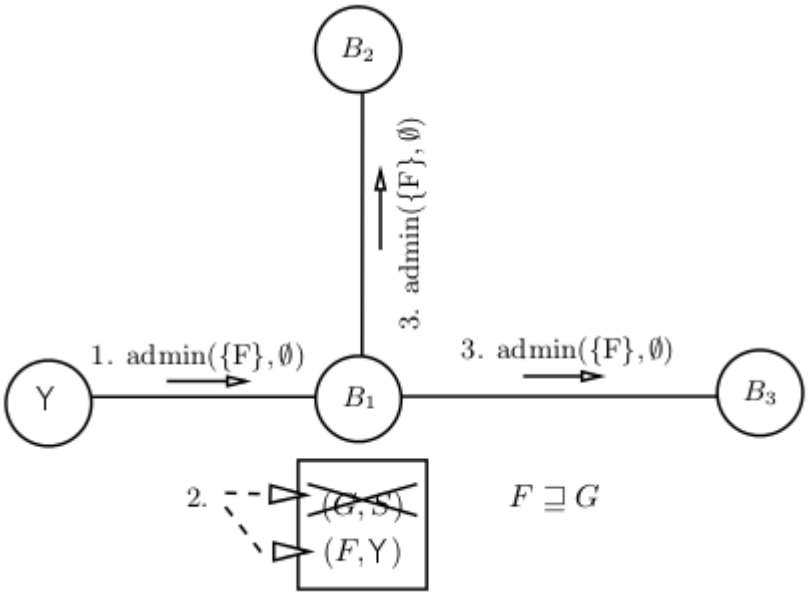

Figura 3.7 – Nova inscrição de um Broker vizinho. Figura adaptada [PIETZUCH06-06].

Primeiramente, quando *B1* recebe *F* este exclui de sua tabela de roteamento a entrada correspondente ao vizinho *Y*, a fim de evitar duplicidade de filtros referentes a um mesmo

vizinho. *B1* repassa F para todos os seus vizinhos *Brokers*, pois nenhum dos vizinhos recebeu *F* vindo de *B1* anteriormente.

Cancelamento de inscrição

Para garantir que um Broker receba todas as notificações correspondentes a um filtro *F*, no momento do cancelamento de uma inscrição, esta também é enviada para cada vizinho do Broker, onde um subconjunto de filtros esteja na abrangência de *F*. O tratamento de um cancelamento é semelhante ao tratamento de uma inscrição. Primeiramente, a tabela de roteamento é atualizada e os destinos para os quais o cancelamento é transmitido são determinados. Conforme exemplo da figura 3.8, quando um broker recebe um cancelamento de inscrição de um vizinho *Y*, este verifica o seguinte:

- Se *Y* é um vizinho, o broker B remove todos os registros de roteamento onde filtro tenha abrangência de F e que se refira ao destino *Y*.
- O Broker B envia o filtro para todos os seus vizinhos, exceto para *Y*.

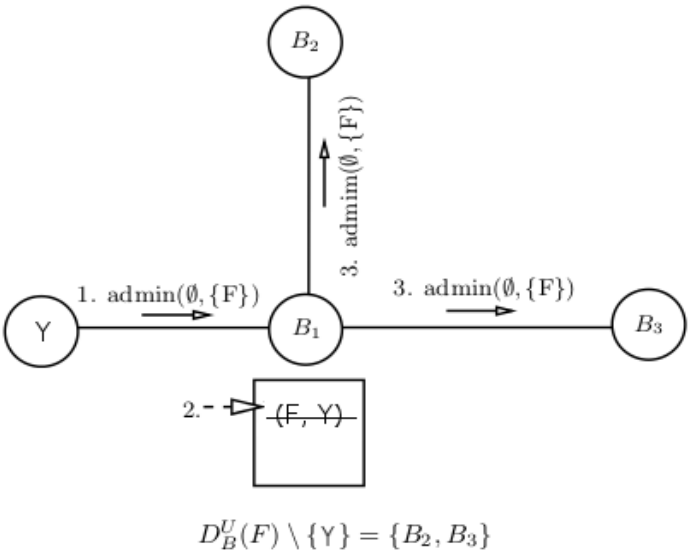

Figura 3.8 – Cancelamento de inscrição. Figura adaptada [PIETZUCH06-06].

Quando o nó que envia a mensagem é um cliente local, então o processamento ocorre conforme o cenário da figura 3.9.

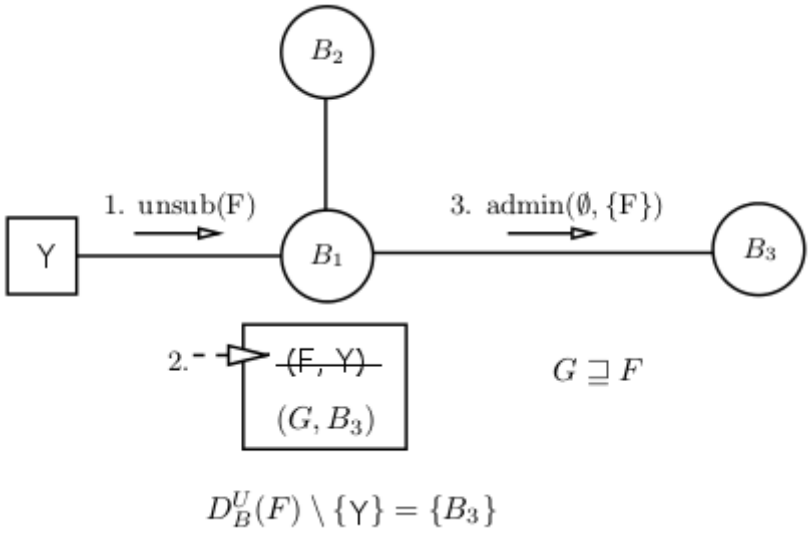

Figura 3.9 – Cancelamento de inscrição - cliente local. Figura adaptada [PIETZUCH06-06].

- Se *Y* é um cliente local o broker B remove apenas *(F, Y)* para não existir múltiplas inscrições ativas de um cliente.

Nova inscrição sem abrangência

Primeiramente todas as inscrições sem abrangência, da tabela de roteamento do Broker, são recebidas do nó Y. Cada entrada de roteamento (F, U) ∈ P, representa uma inscrição *F* sem abrangência no nó de destino U. Então o Broker *B* faz o seguinte:

- Determina o número de destinatários *K* em *P* com inscrições idênticas.
- Todos os registros idênticos são removidos de *P*

Assim todas as inscrições dos nós de destino são adicionadas como destinatários.

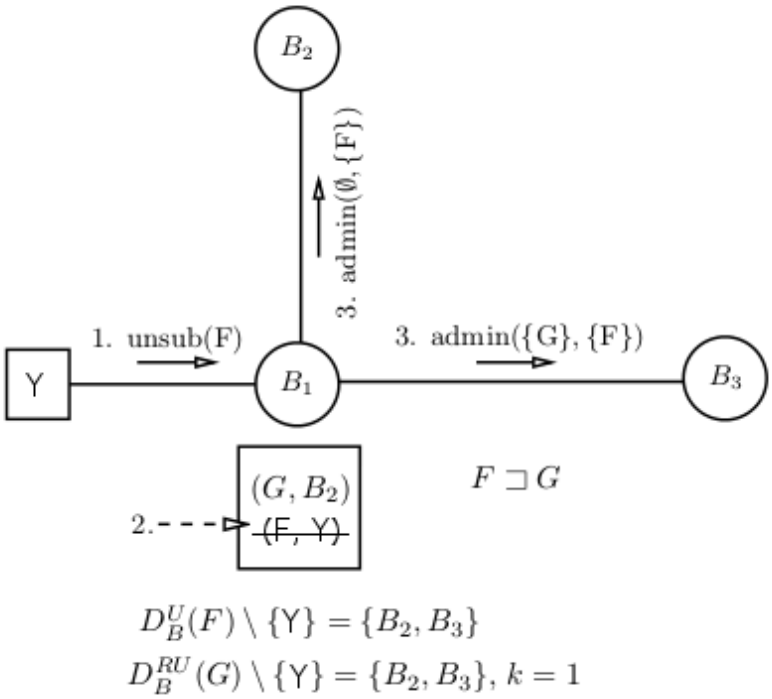

Figura 3.10 – Cancelamento de inscrição de um cliente local [PIETZUCH06-06].

No cenário da figura 3.10 o Broker *B1* recebe um cancelamento de *F*. Este remove o registro de *F* (referente à *Y)* de sua tabela de roteamento. O filtro *G* que já está na tabela de *B1*, não tem abrangência de *F*. Enquanto o cancelamento *F* é encaminhado para *B2* e *B3*, a inscrição *G*, que não tem abrangência, é encaminhada somente para *B3* devido à variável *k* ter valor igual a um.

3.1.5 Roteamento Baseado em “merge”

O objetivo do “*merge*” é determinar um novo filtro a partir da junção de filtros existentes. Essa estratégia pode ser utilizada para reduzir a quantidade de inscrições que um *Broker* armazena. A junção pode ser realizada para um conjunto de filtros conjuntivo, com no máximo um filtro de atributo para cada atributo, testando se o valor do atributo (de um dos conjuntos em questão) pode ocorrer no outro conjunto (candidato a junção). Por exemplo:
F1 = {x = 5 ∧ y ∈ {2, 3}} e *F2* = {x = 5 ∧ y ∈ {4, 5}} pode ser feito o “*merge*” resultando em F = {x = 5 ∧ y ∈ {2, 3, 4, 5}}.

Esta estratégia é executada por cada *Broker*, realizando a união de filtros com abrangência em comum, onde o *Broker* encaminha os filtros resultantes para seus destinatários no lugar dos filtros originais. O merge é aplicado onde o destinatário seja o mesmo. O intuito é o de diminuir o repasse de criação e cancelamento de inscrições para um *Broker* vizinho, evitando repassar a mesma mensagem mais de uma vez. Um algoritmo de roteamento baseado em “merge” que garante que os filtros gerados sejam transmitidos de tal forma que apenas as notificações de interesse sejam entregues a um Broker, este é chamado perfeito, caso contrário, o algoritmo é imperfeito. Um Broker ao construir um novo filtro baseado em filtros de sua tabela de roteamento, este remove os registros de inscrições que possuem abrangência em sua tabela referente aos nós assinantes, deixando apenas o filtro resultante da junção.

Processamento de criação de Inscrição

- Se o NoSolicitante $\in N_B$ (se está na tabela de roteamento do broker), os filtros deste que são cobertos pela nova inscrição, são removidos da tabela de roteamento.
- Depois disso, é verificado se a nova inscrição tem abrangência por qualquer junção existente em relação ao mesmo destino. Se pelo menos um filtro é encontrado, a nova inscrição é adicionada e o filtro do *NoSolicitante* é removido.
- Se a nova inscrição não tem abrangência por qualquer junção existente, então é verificado se é possível gerar uma nova junção da inscrição de filtros existentes (que não são junções) em relação ao mesmo destino. Se a nova junção for gerada, os outros filtros que constituem a nova inscrição (que foi recebida pelo broker) são removidos da tabela de roteamento. Se uma nova junção foi gerada, é verificado se os filtros (ou outras junções) em relação ao mesmo destino são abrangidos por esta junção. Os filtros cobertos (junções) são removidos da tabela de roteamento (e seus filtros constituintes) são adicionados à nova junção.

3.1.6 Roteamento Hierárquico

No roteamento hierárquico, um Broker é definido como o nó raiz da topologia. Cada notificação publicada sempre é encaminhada para um nó raiz, que é um Broker. Conforme a figura 3.11, cada criação e cancelamento de inscrição são enviados no sentido do nó raiz da árvore, ou seja, para o nó Broker. No roteamento hierárquico, cada Broker é responsável por disseminar cada notificação que recebe na sua respectiva árvore, porém, sua tabela de roteamento contém apenas filtros de inscrições originados de sua árvore própria árvore ao invés do Broker ter em sua tabela todas as inscrições de Brokers na rede. Comparado ao encaminhamento ponto-a-ponto (*peer-to-peer*), o roteamento hierárquico reduz o tamanho das tabelas de roteamento.

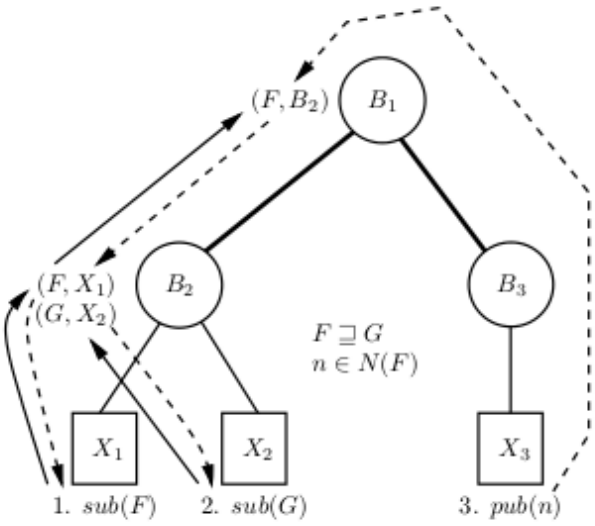

Figura 3.11 – Roteamento hierárquico. Figura adaptada [PIETZUCH06-07].

A quantidade de registros da tabela de roteamento de cada Broker é corresponde ao seu nível na hierarquia Broker. Para o nó da raiz, a dimensão de suas tabelas de roteamento não é reduzido, embora tenha de lidar com todas as notificações publicadas no sistema. Assim, este nó pode, eventualmente, ser sobrecarregado. Uma solução possível para isto pode ser o de replicar o nó de raiz e de alguns dos seus nós filhos para conseguir assim uma carga de trabalho superior. Para dar suporte ao roteamento hierárquico, os algoritmos de roteamento detalhados na seção 3.1 são alterados pontualmente, com exceção do Roteamento Baseado em Merge. O Broker raiz é definido por *R*, sendo *B* um Broker que não é raiz.

3.2 Principais Estratégias e Serviços de Notificações de Eventos Existentes

Apresentamos nessa seção uma estratégia *multicast* para roteamento em DEBS, os principais serviços de notificação de eventos (serviços das organizações OMG e Apache), projetos de pesquisa e principais artigos de trabalhos relacionados ao roteamento de eventos.

3.2.1 *Multicast*: IGMP – Protocolo Internet de Gerenciamento de Grupo (*Internet Group Management Protocol*)

O Protocolo Internet de Gerenciamento de Grupo - IGMP[3] é um protocolo *multicast* participante do protocolo TCP/IP, onde a finalidade principal é notificar roteadores sobre *hosts* que entram e saem de grupos *multicast* em uma rede local. Conforme especificação [CAIN;DEERING;KOUVELAS;FENNER02], o protocolo é utilizado por *hosts* e roteadores, em uma LAN. Um *host* utiliza o protocolo para a fim de notificar os roteadores sobre sua entrada/saída de grupos. Já os roteadores *multicast* utilizam IGMP para saber quais grupos têm membros em cada uma de suas redes, para atualizar suas listas de endereços de grupos *multicast* para cada uma de suas redes.

A versão mais recente IGMPv3 dá suporte a "filtragem de fontes" para informar que *hosts* informem, no momento da entrada em um grupo, o interesse em receber pacotes de apenas endereços de origem específicos. Essa informação pode ser usada por protocolos de roteamento *multicast* para evitar a entrega de pacotes para grupos onde não existem *hosts* interessados. A interação entre *host* e roteador ocorre através da troca de mensagens IGMP, as quais são encapsuladas em datagramas IP.

Mensagens IGMP

Existem três tipos de mensagens de consulta para a interação entre host e roteador:

[3] IGMPv3 - versão mais atual [CAIN;DEERING;KOUVELAS;FENNER02].

- Consulta Geral (envio de mensagem para 224.0.0.1) usado para saber quais grupos têm membros;
- Consulta de Grupo - especifico, utilizado para saber se um determinado grupo tem membros;
- Consulta de grupo e fonte específica, onde são informados também *hosts* específicos como fonte de mensagens, conforme interesse do *host*.

A figura 3.12 mostra o formato de mensagens de consulta.

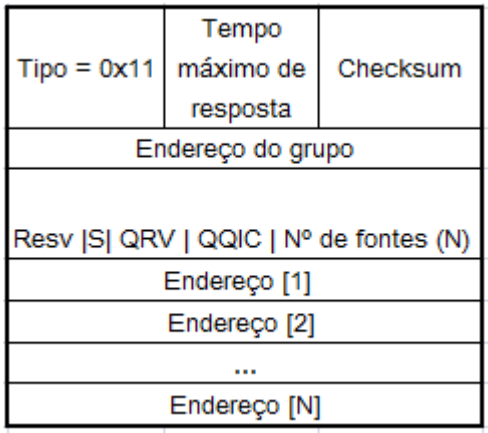

Figura 3.12 – Formato de mensagens de consulta. Figura adaptada [CAIN;DEERING;KOUVELAS;FENNER02].

A tabela 3.1 detalha o conteúdo de cada campo da mensagem.

Tipo	Possíveis conteúdos: 0x11 = Consulta de “adesão” 0x22 = Relatório de “adesão” (“Membership Report”) 0x17 = Saída do grupo
Tempo Maximo de Resposta	Especifica o tempo máximo permitido antes do roteador enviar um relatório de adesão do grupo para o grupo todo
Checksum	Usado para somatória no controle de ocorrências de erros
Endereços de grupos	Em mensagens de Consulta de “adesão”, o campo de endereço do grupo é definido como zero, e quando a mensagem é de consulta específica de um grupo o endereço do grupo é informado
Resv	Valor zero em transmissão e ignorado no recebimento da mensagem

Flag S	Indica para os roteadores *multicast* eliminarem o tempo imite para receber mensagens de consultas
QRV	Nível de robustez do conteúdo da consulta
QQIC	Intervalo de consulta, em segundos, usado pelo *host* remetente
Nº de fontes	Especifica quantos endereços de origem estão presentes na consulta. O valor é zero em uma consulta geral e diferente de zero para consultas específicas de grupos e fontes

Tabela 3.1: conteúdo de mensagens de consulta IGMP.

Roteadores

Cada roteador pode assumir um dos dois papéis na rede: consultante ou não-consultante. No papel de consultante, o roteador é responsável por transmitir solicitações de entrada ou saída de *hosts* para grupos, onde armazena uma lista de todos os endereços de grupos. Já no papel não-consultante, o roteador é responsável somente por repassar mensagens. Todos os roteadores *multicast* iniciam como consultante na rede, onde este deixa de ser consultante somente se receber uma mensagem de consulta de outro roteador que tenha um endereço IP mais baixo que o dele.

Conforme *timeout* definido na mensagem de consulta, *hosts* enviam mensagens de relatórios para informar aos roteadores seu estado atual em grupos (entrada, saída ou permanência). Esses relatórios são utilizados, pelo roteador, para atualizar sua lista de endereços de grupos *multicast*. A figura 3.13 mostra a estrutura de um relatório.

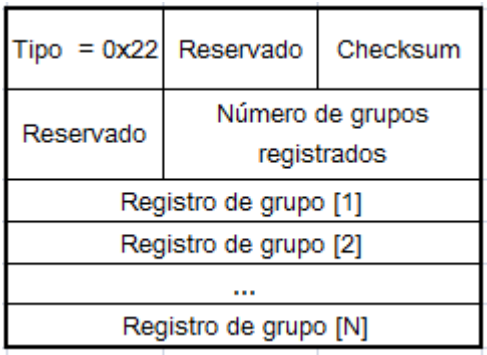

Figura 3.13 – Formato de mensagens de relatório. Figura adaptada [CAIN;DEERING;KOUVELAS;FENNER02].

Entrada em grupos:

- Quando um *host* quer entrar em novo grupo, este envia seu pedido para o *host* responsável em que está conectado;
- O *host* irá adicionar o nome do *host* solicitante e o nome do grupo requisitado em sua lista. Se for o primeiro pedido para entrar no grupo, o *host* envia um "*report"* para o roteador de *multicast*.

Saída de grupos:

- Se o *host* de um grupo recebeu apenas mensagens de saída e não existem mais *hosts* para aguardar respostas no grupo, então o host envia um "*report*" para o roteador *multicast*, onde este atualiza a lista de grupos, excluindo o grupo de quem recebeu o "*report*".

IGMP no roteamento de Sistemas Baseados em Eventos

Conforme soluções apresentadas em [LEGATHEUX;DUARTE10], o roteamento *multicast* é aplicado em redes de topologias hierárquicas, onde a disseminação de notificações de eventos é realizada em níveis conforme o interesse de assinantes. Para tal, nada impede a utilização do protocolo IGMP em DEBS, onde *hosts* solicitam participação em grupos conforme seu interesse, tendo semelhança com a solução de canais de eventos [OMG04-01].

3.2.2 CORBA

A plataforma CORBA - *Common Object Request Broker Architecture* [OMG12] é extensível por meio de serviços em diferentes abordagens no ambiente de computação distribuída. O Serviço de Eventos [OMG04-01] e o Serviço de Notificação de Eventos [OMG04-02] atendem a funcionalidade "*Publish/Subscribe*".

3.2.2.1 Serviço de Eventos

Conforme documento de especificação [OMG04-01], o Serviço de Eventos separa a comunicação entre dois papéis de objetos: fornecedor e consumidor. Os fornecedores produzem os dados de eventos e consumidores fazem o processamento desses dados. Fornecedores e Consumidores não têm associação entre si, sendo um serviço intermediário chamado Canal de Eventos que permite a comunicação entre Fornecedor e Consumidor de forma assíncrona. Um canal de eventos é ao mesmo tempo um consumidor e um fornecedor de eventos, onde esse é responsável por determinar como mudanças são propagadas entre fornecedores e consumidores. O tipo de comunicação entre Consumidores e Fornecedores com o canal de eventos é divido entre dois modelos: "*push*" e "*pull*". Um canal de evento pode ser de um tipo específico, genérico ou misto. Em um canal de evento genérico, consumidores e fornecedores enviam e recebem dados de eventos genéricos, ou seja, todos os dados de eventos produzidos. Em um canal de um tipo específico, fornecedores e consumidores enviam e recebem dados de eventos referentes ao tipo de evento do canal. O canal é o responsável por realizar a correspondência do tipo de evento para inscrição e entrega de dados eventos entre fornecedor e consumidor.

Comunicação – modelo "Pull"

No modelo "*pull*" são os fornecedores que iniciam a transferência dos dados de eventos para os consumidores invocando operações da interface *PullConsumer*. Para estabelecer comunicação no modelo "*pull*", consumidores e fornecedores trocam referências de objetos *PullSupplier* e *PullConsumer*. O canal de eventos, por sua vez, encaminha os dados de eventos para os consumidores.

Conforme a figura 3.14, a interação entre consumidor e fornecedor é realizada da seguinte forma:

- Consumidores e Fornecedores solicitam comunicação através do canal de eventos;
- Ocorrendo a produção de um evento, o fornecedor envia dados do evento para o canal de eventos;
- O canal de eventos recebendo dados de eventos do fornecedor encaminha os dados para os consumidores com comunicação estabelecida.

Figura 3.14 – Modelo de comunicação "*Pull*" – adaptação [OMG04-03].

Neste modelo de comunicação, o consumidor apenas informa interesse, onde fica somente aguardando o recebimento dos dados de eventos. O fornecedor é que o responsável por enviar dados de eventos para o canal de eventos, e este é quem mapeia a entrega dos dados do evento para os consumidores.

Comunicação – modelo "*Push*"

No modelo "*push*" são os consumidores que "retiram/buscam" dados de eventos dos fornecedores invocando operações da interface *PushSupplier* do Canal de Eventos. Para estabelecer comunicação, consumidores e fornecedores trocam referências de objetos *PushSupplier* e *PushConsumer*. O canal de eventos, por sua vez, solicita dados de dados de eventos para os fornecedores.

Figura 3.15 – Modelo de comunicação "*Push*" – adaptação [OMG04-03].

Conforme a figura 3.15 a interação funciona da seguinte forma:

1. Fornecedores e Consumidores solicitam comunicação com o canal de eventos;

2. O fornecedor produz um evento e retém os dados de eventos para si mesmo;
3. Consumidores solicitam dados de eventos para o canal de eventos;
4. O canal de eventos solicita dados de eventos para Fornecedores;
5. Os fornecedores enviam dados de eventos para o canal de eventos;
6. O canal de eventos recebendo os dados de eventos de fornecedores, este encaminha os dados para os consumidores que tenham comunicação estabelecida.

3.2.2.2 Serviço de Notificação de Eventos

Conforme documento de especificação [OMG04-02], o Serviço de Notificação de Eventos é sucessor do Serviço de Eventos com a inclusão de novos recursos:

- Transmitir eventos em uma estrutura de dados bem definida ao invés de utilizar somente Tipos de Evento.
- Permitir para cada cliente a realização de especificação exata de interesse em eventos, anexando filtros para cada *proxy* em um canal de eventos.
- No registro do interesse de tipos de eventos, permitir que fornecedores encontrem canais de eventos aonde os consumidores registraram interesse. Assim os fornecedores podem produzir eventos sob demanda, ou evitar a transmissão de eventos em que os consumidores não têm interesse.
- Na disponibilidade de um canal de eventos de um tipo de evento, permitir que consumidores descubram esses canais de eventos quando tais canais estiverem ativos.

Eventos Estruturados e Filtragem de Eventos

No Serviço de Eventos dois tipos de eventos foram definidos: eventos genéricos e eventos “tipados”. Já no Serviço de Notificação de Eventos outra forma de representação de eventos é definida: uma estrutura de dados. A representação através de uma estrutura de dados possibilita que a correspondência do registro de interesse possa ser mapeada de maneira mais eficiente. Na figura 3.16 mostra a estrutura definida de um evento.

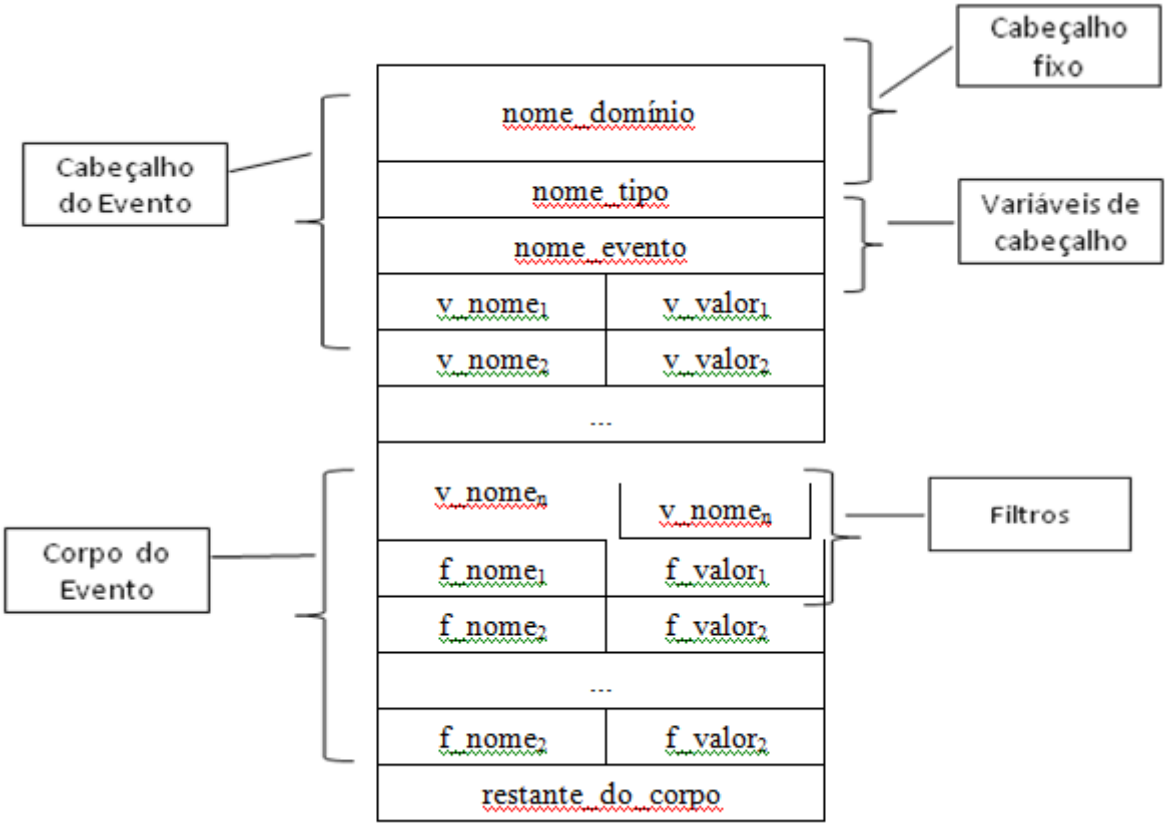

Figura 3.16 – Estrutura de Evento – adaptação [OMG04-02].

Cada evento é composto por dois componentes principais: um cabeçalho e um corpo. O cabeçalho é divido entre uma parte fixa e uma parte opcional. A parte fixa do cabeçalho do evento é composta por três domínios de caracteres:

- Nome_domínio: identifica o domínio de negócio em que o tipo de evento é definido (por exemplo, telecomunicações, finanças, saúde, etc.)
- Nome_tipo: categoriza o tipo de evento exclusivamente dentro do domínio (por exemplo: alertas, cotação, sinais vitais).
- Nome_evento: identifica a instância específica do evento que está sendo transmitido.

A parte opcional do cabeçalho é composta por pares de nome/valor de configuração, onde cada nome é uma *string* e cada valor é de um tipo qualquer. Os seguintes campos podem ser pares de valores: ConfiabilidadeEvento, Prioridade, HorárioInício, HorárioTérmino e TempoLimite.

Na segunda divisão da estrutura, que é o corpo do evento, também é composta por duas partes: filtros e informações complementares de interesse. A parte de filtros também é definida por pares de nome/valor, onde estes representam o interesse específico do assinante.

3.2.3 JINI

A tecnologia Jini é uma Arquitetura Orientada a Serviço (SOA) mantida atualmente pela Apache [ASF-13] (anteriormente mantida pela SUN *Microsystems*). Um sistema Jini é um sistema distribuído que disponibiliza recursos ("*hardware*", "*software*", ou ambos) para grupos de usuários com interesses em comum. Interfaces de Eventos Distribuídos são especificadas na arquitetura para atender Eventos e Notificação de Eventos Distribuídos. O objetivo é permitir que objetos registrem interesse em eventos de outros objetos, e na ocorrência de um evento esses objetos recebam uma notificação de evento [SUN01].

Em um sistema Jini um evento é algo que acontece em um objeto, que corresponde a alguma mudança no seu estado abstrato. Para permitir que entidades recebam notificações de seus eventos, um objeto exporta uma identificação de um Tipo de Evento para que outros objetos indiquem interesse. Na mudança de estado o próprio objeto é responsável por identificar e gerar um objeto *RemoteEvent* para ser enviado como notificação para outros objetos que registraram interesse. Os objetos que compõe o sistema distribuído de eventos são os seguintes:

- O objeto que registra o interesse em um evento
- O objeto gerador de evento
- O objeto destinatário de notificações de eventos ("*Remote Event Listener*" - ouvinte remoto de eventos).

Um objeto ouvinte registra interesse em um tipo específico de evento e o gerador do tipo de evento envia uma notificação quando há ocorrência de um evento. A figura 3.17 mostra a interação entre os objetos do sistema distribuído de eventos.

Figura 3.17 – Interação de objetos – adaptação [ASF-RIVER13].

A classe *RemoteEventListener* representa o objeto ouvinte, a qual é definida por uma interface que contém somente o método *"notify"*, que que informa os ouvintes interessados que ocorreu um evento. As classes *RemoteEvent* e *EventRegistration* são objetos que o ouvinte consome. O objeto *RemoteEvent* representa um evento, onde este é composto pelos os seguintes atributos: um identificador para o tipo de evento em que o interesse tenha sido registrado; uma referência do objeto em que o evento ocorreu; um número de seqüência que identifica o tipo de evento; um objeto que foi enviado como parte do registro de interesse pelo ouvinte. A figura 3.18 mostra uma implementação da interface *RemoteEventListener*.

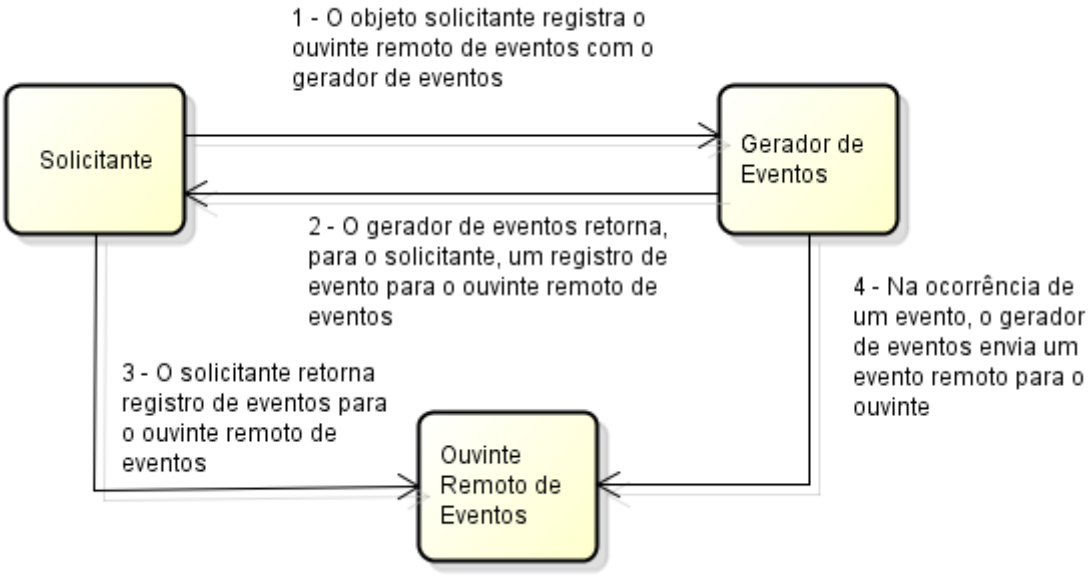

Figura 3.18 – Interface *RemoteEventListener* – adaptação [ASF-RIVER13].

O objeto *RemoteEvent* é passado como parâmetro para o método "*notify*" do objeto *RemoteEventListener*. A classe *EventRegistration* define um objeto que retorna para o ouvinte as informações do tipo de evento registrado do gerador de eventos. Outra tarefa que o ouvinte remoto de eventos também executa, é a filtragem de notificações de eventos. No registro de interesse, o solicitante pode incluir restrições para receber notificações de eventos, onde tais restrições são aplicadas no tipo de evento de interesse.

3.2.4 JEDI

A infra-estrutura JEDI ("*Java Event-based Distributed Infrastructure*") é uma infra-estrutura orientada a objetos para desenvolver sistemas distribuídos baseados em eventos [CUGOLA;NITTO;FUGGETTA01]. Um evento é simplesmente uma mensagem e o modelo de mensagem adotado é o modelo de tuplas [GEHANI; JAGADISH;SHMUELI;92]. A arquitetura é composta por Publicadores e Assinantes, definidos como "Objetos Ativos", onde interagem entre si por meio de eventos. O componente chamado "*Event Dispatcher*" é o responsável por fazer o intermédio entre objetos ativos publicadores e assinantes, recebendo registro de interesse e enviando notificações de eventos correspondentes a cada interesse registrado. A figura 3.19 mostra a interação entre objetos ativos e o serviço de envio de notificações de eventos ("*dispatcher*").

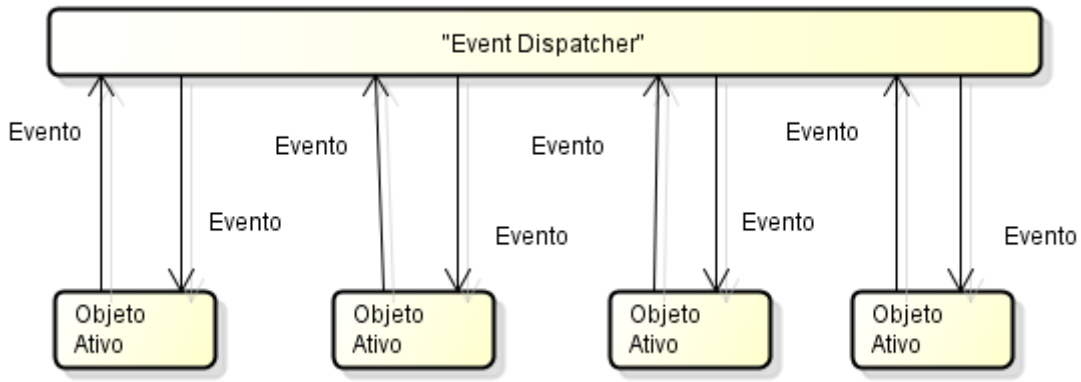

Figura 3.19 – Interface *RemoteEventListener* – adaptação [CUGOLA;NITTO;FUGGETTA01].

Cada evento gerado por um Objeto Ativo é enviado para o "*event dispatcher"*, e este repassa o evento para os Objetos Ativos (assinantes) interessados.

Serviço de Notificação ("Event Dispatcher")

O "*Event Dispatcher*" é um componente logicamente centralizado, uma vez que deve ter um conhecimento global de todos os eventos que são gerados e de todas as inscrições registradas. Na infra-estrutura JEDI, duas estratégias para disseminação de eventos (serviço de "*Event Dispatcher*") são implementadas: centralizada e distribuída. A implementação centralizada atende um único processo (sistema operacional) e foi desenvolvida para atender aos requisitos de um sistema simples, composto por alguns Objetos Ativos. Já a versão distribuída foi implementada para atender vários processos, chamados de servidores de "*dispatcher*", que são interligados na rede. Em ambas as estratégias, a estrutura da rede é em árvore. A figura 3.20 mostra da estratégia hierárquica, onde cada servidor ("*Server Dispatcher*") é um nó na rede.

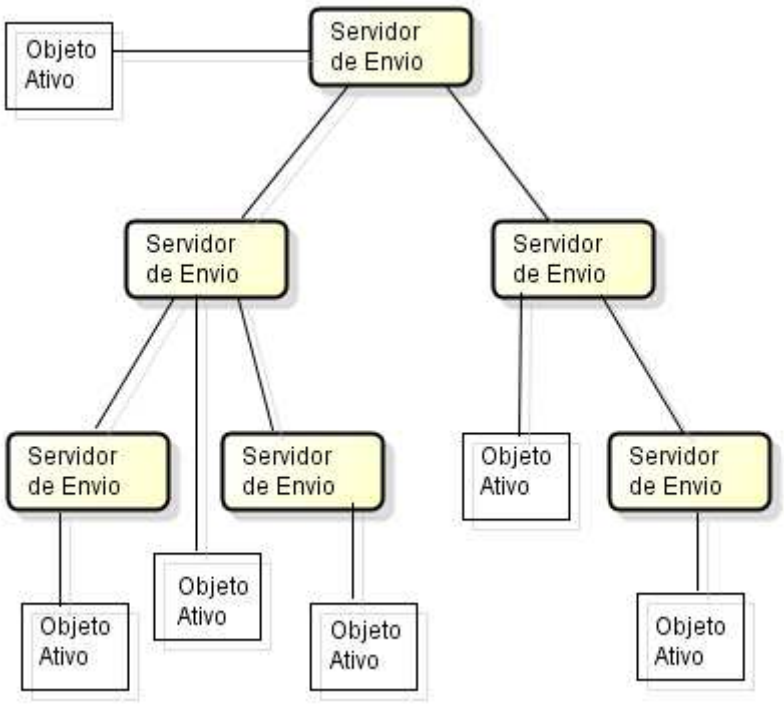

Figura 3.20 – Estrutura da rede – servidores "*Event Dispatcher*" – adaptação [CUGOLA;NITTO;FUGGETTA01].

As operações de criação e cancelamento de inscrições são gerenciadas de formas diferentes nas estratégias centralizadas e distribuídas. Quando a rede é local (centralizada), cada "*server dispatcher*" armazena localmente todas as inscrições registradas por objetos ativos que são filhos deste na árvore. Quando um evento é gerado, este é distribuído para todos os "*servers dispatcher*" na árvore e cada um decide para quais objetos ativos que o evento será entregue. Na estratégia distribuída o gerenciamento de inscrição é realizado de uma forma diferente: as inscrições registradas são propagadas para todos os "*servers dispatcher*" na árvore. Quando um "*server dispatcher*" recebe uma solicitação de criação ou cancelamento de inscrição, este repassa a solicitação somente para o nó de nível acima dele na árvore. Já no recebimento de evento, todo "server dispatcher" propaga o evento para todos os nós que estão conectados a ele em um nível acima e em um nível abaixo na árvore. Sendo assim, cada "*server dispatcher*" ao receber um evento envia o evento para as seguintes entidades:

- Para o nó de nível acima dele na árvore não levando em consideração o interesse do nó, pois o nó de nível acima pode ser um "*server dispatcher*" ou um objeto ativo (pois todos os nós interessados devem receber o evento);
- Para o subconjunto de nós de um nível abaixo na árvore (nós filhos) se inscreveram para um padrão de evento que corresponde com o evento a ser enviado.

Outra questão que é abordada no serviço de notificação é a ordenação causal. Os eventos são entregues de acordo com uma política de ordenação causal, por exemplo: se o evento E1 foi causado pela geração de evento E2, então qualquer Objeto Ativo registrado os eventos E1 e E2, receberão primeiramente E1 depois o evento E2.

3.2.5 Hermes

Hermes é um *middleware* baseado em eventos desenvolvido por um grupo de pesquisa na Universidade de Cambridge [PIETZUCH;BACON;03]. Na arquitetura definida, dois componentes consistem um sistema baseado em eventos: clientes e *brokers*. Clientes de eventos podem ser publicadores ou assinantes e os *brokers* são os componentes que representam o *middleware* propriamente dito. Dessa forma, a interação entre clientes ocorre através de operações disponibilizadas por *brokers*. A figura 3.21 mostra a arquitetura do um sistema sobre o *middleware*.

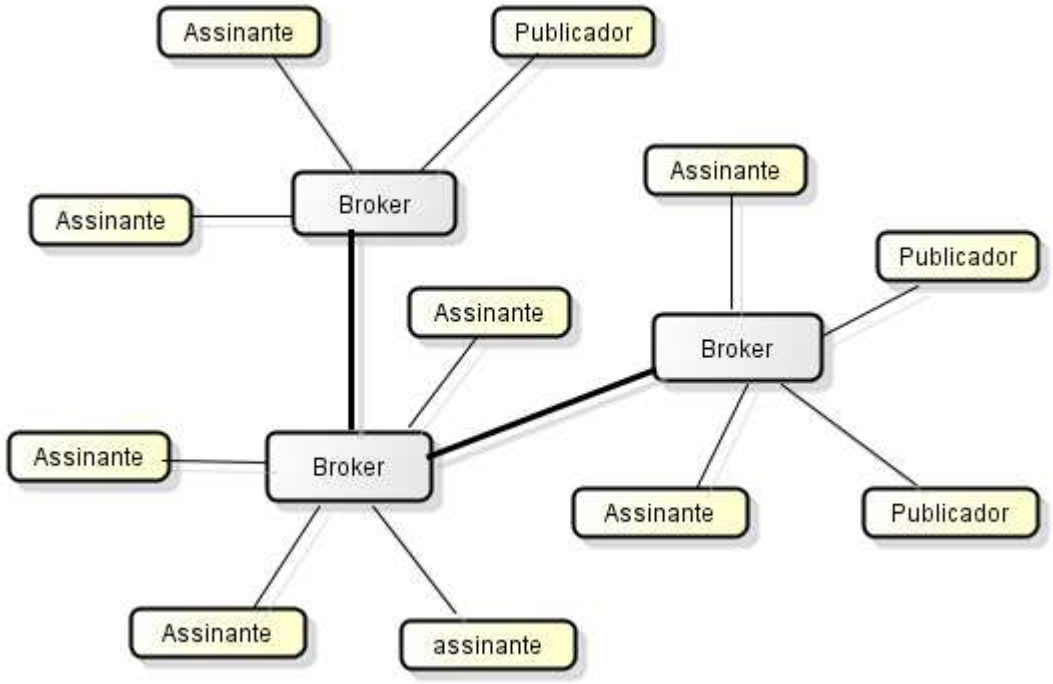

Figura 3.21 – Arquitetura de um sistema Hermes - adaptação [PIETZUCH;BACON;03].

A execução do *middleware* ocorre em uma rede "*overlay*" de topologia em árvore. Cada sobreposição da rede corresponde a uma árvore onde o nó raiz é nó de encontro de um tipo de evento e os nós filhos são os *brokers* que tem conexões de assinantes e publicadores conectados. Cada árvore é construída da seguinte forma: antes de emitir um evento, um publicador envia uma mensagem de "anúncio" para o *broker* em que está conectado. Cada *broker* ao receber a mensagem de anúncio, repassa a mesma para seu *broker* vizinho até que a mensagem chegue ao *broker* de encontro do tipo de evento, armazenando o identificador do nó de quem recebeu a mensagem, para que posteriormente todo *broker* conheça o caminho para enviar a publicação de eventos. Para não sobrecarregar o *broker* com todos os anúncios emitidos, todo broker executa o algoritmo de abrangência (visto na seção 3.1.4 - "Roteamento Baseado em Abrangência"). Sendo assim, dois algoritmos podem ser aplicados para a entrega de publicação de eventos: baseado em tipo de evento e baseado em tipo e atributo de evento.
Para a disseminação de publicações, cada *broker* faz o repasse da mensagem para o nó de encontro e para todos os seus vizinhos que tenham interesse registrado.

Roteamento baseado em tipo (disseminação genérica)

Nesse roteamento todos os anúncios e publicações de eventos devem chegar ao nó de encontro. No decorrer da entrega de publicações os *brokers*, que são caminho para o nó de encontro, enviam a mensagem de publicação para vizinhos (*brokers* ou assinantes) que registraram interesse no tipo de evento de publicação. Assim quando a mensagem chega ao nó de encontro, este repassa a mensagem somente para seus vizinhos com interesse exceto de quem recebeu a mensagem. Na Figura 3.22 mostra um exemplo de interação entre assinantes, publicadores e *brokers*. Neste cenário, temos dois assinantes $S_{\{1,2\}}$, dois publicadores $P_{\{1,2\}}$, seis *brokers* $B_{\{1 \ldots 5\}}$ e o *broker* de encontro R.

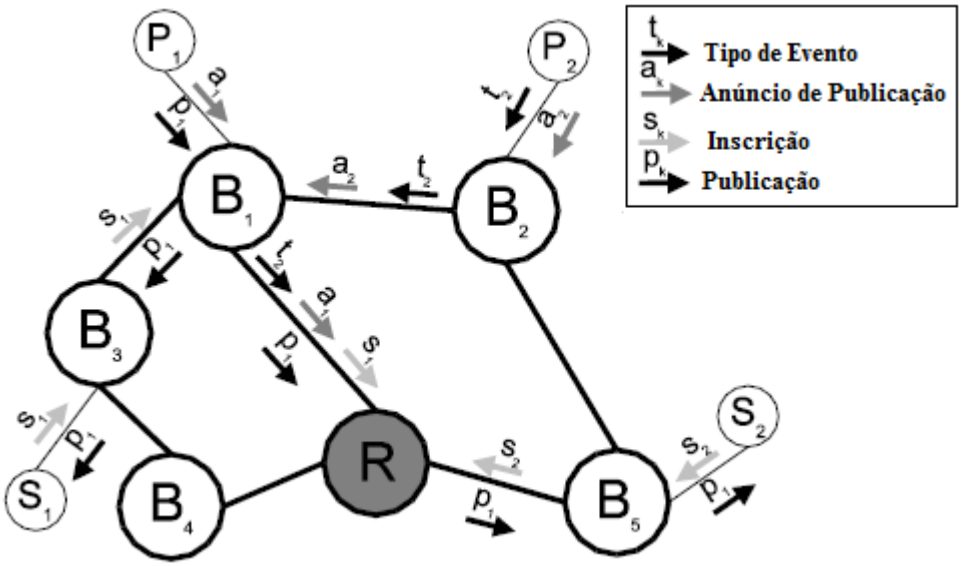

Figura 3.22 – Roteamento baseado em tipo – adaptação [PIETZUCH;BACON;03].

Os anúncios emitidos pelos publicadores P1 e P2 são enviados até o nó de encontro R, onde no decorrer do caminho os *brokers* B1 e B2 armazenam em suas tabelas de roteamento os identificadores dos nós P1 e P2. No registro de inscrição, o assinante S1 envia uma inscrição para o *broker* B3 e este como não têm nenhum publicador conectado a ele, repassa a inscrição para o *broker* vizinho de um nível acima na árvore. B3 repassa a inscrição para o B1. Quando recebe uma publicação de P1, este repassa a publicação do evento para o nó de encontro R e para B3 (que tem interesse registrado). Por último, o nó R repassa a publicação para todos os

seus vizinhos com interesse, exceto do vizinho de quem recebeu a mensagem. Assim, o nó R repassa a mensagem somente para B5.

Roteamento baseado em tipo e atributo (disseminação restrita)

O roteamento baseado em atributo estende o roteamento baseado em tipo, aplicando a filtragem de inscrições para a disseminação de notificação de evento. A diferença é que o no encaminhamento da mensagem de inscrição, além de armazenar o tipo de evento, o *broker* também armazena a expressão de filtro da inscrição. A ideia é que a filtragem seja feita o mais próximo possível do publicador para evitar propagação desnecessária. O encaminhamento de mensagens de publicação é controlado pelo Estado de inscrições em tabelas de roteamento. Mensagens de publicação seguem o caminho inverso de correspondência inscrições, onde as publicações de eventos são progressivamente filtragem de acordo com interesses de assinantes. A Figura 3.23 mostra um cenário de interação com expressões de filtros. Primeiramente, os publicadores P1 e P2 anunciam o tipo de evento enviando as mensagens de anúncio A1 e A2. Essas duas mensagens de geram registros nas tabelas de roteamento dos *brokers* B1, B2, e R. Quando o assinante S1 registra interesse, a mensagem de inscrição é encaminhada para o nós B3, B1 e R.

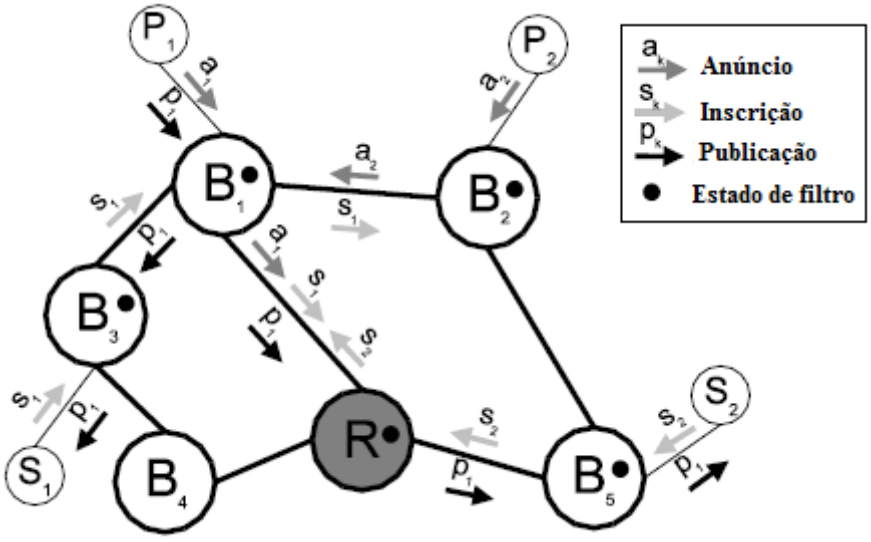

Figura 3.23 – Roteamento baseado em tipo e atributo - adaptação [PIETZUCH;BACON;03].

O broker B1 encontra correspondência entre o filtro da mensagem A2 e a inscrição de S1, onde assim a inscrição de S1 é encaminhada para o B2. A inscrição S1 também é enviada para R por conta do roteamento para o nó de encontro. A inscrição do assinante s2 é encaminhada da mesma forma, exceto que é descartada por B1 devido que a inscrição S1 tem cobertura para a inscrição S2, não sendo necessário repassar inscrições com a mesma abrangência (existem atributos de S1 que também existem em S2). Por último, a publicação P1 é encaminhada ao longo do caminho inverso da inscrição de S2, ou seja, a mensagem é encaminhada para os nós B1, R, e B5.

Comparado com o roteamento baseado em tipo, o roteamento baseado em atributo aumenta o tamanho de tabelas de roteamento por conta dos filtros de inscrições. Isto é porque as publicações de evento são filtradas o mais perto possível dos publicadores. Uma vantagem é que as publicações, do mesmo tipo e filtro de eventos, não precisam chegar ao nó de encontro mais uma vez. Assim, o nó de encontro pode não ser uma sobrecarga para a disseminação de notificações de eventos.

3.2.6 Rebeca

O sistema Rebeca é um modelo de Serviço de Notificação de Pesquisa proposto em [PIETZUCH06]. Apresentamos nessa seção as principais características e a arquitetura do sistema Receba.

Modelo do Serviço de Notificação

O modelo adotado neste sistema consiste de um conjunto de nós físicos interligados, onde cada nó executa um ou mais processos. A interação entre processos é realizada por troca de mensagens. Cada nó na rede é conectado a outro nó por um link, sendo a conexão ponto-a-ponto. As mensagens que são trocadas entre os nós, são trocadas de forma assíncrona (ocorre certa demora entre o envio e recebimento de uma mensagem). A ordenação na entrega de mensagens obedece à ordem FIFO. Assume-se que falhas não ocorram durante o fluxo de

troca de mensagens, pois a camada de transporte, sendo sob TCP / IP, compensa possíveis falhas de comunicação.

Arquitetura

Os componentes do sistema estão ilustrados na figura 3.25. Cada componente é um processo separado, o qual está ligado a um processo do Serviço de Notificação. O serviço é conceitualmente centralizado, mas sua implementação é distribuída entre vários nós e processos [PIETZUCH06-08].

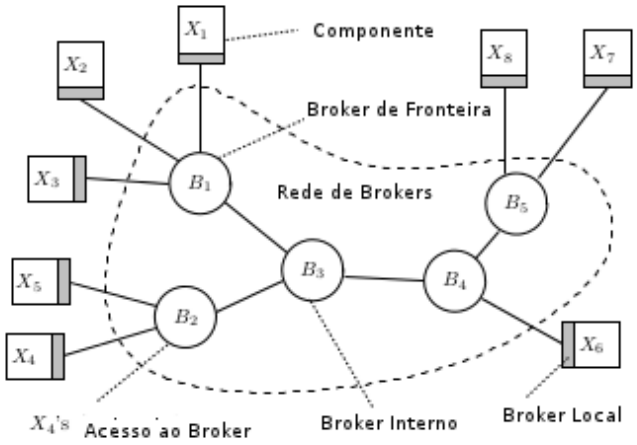

Figura 3.25: Roteamento na rede *Broker* - Rebeca. Figura adaptada [PIETZUCH06-08].

O Serviço de Notificação se constitui sob uma rede "*overlay*". Formalmente, a topologia da rede é modelada como grafo acíclico G = (V, E), com o conjunto de nós V = (B1,..., Bn) correspondem aos Brokers e um conjunto de arestas E ⊆ {(Bi, Bj) | 1 ≤ i <j ≤ n} onde representam ligações bidirecionais [MÜHL02]. Três tipos de nós são definidos: Broker Local, Broker de Fronteira e Broker Interno. Os Brokers Locais são pontos de acesso para o *middleware*, sendo normalmente parte da biblioteca de comunicação, onde são utilizados apenas para questões de implementação. Os Brokers de Fronteira são responsáveis por fazem a ponte entre o *middleware*, juntamente com Brokers Locais (componentes do serviço). Já os Brokers Internos estão ligados a outros Brokers Internos ou Brokers de Fronteira, a fim de

ajudar no encaminhamento de mensagens. Brokers locais implementam as operações de interface "*publish/subscribe*" entre o serviço de notificação. Os Brokers Internos e de Fronteira encaminham as mensagens recebidas para seus vizinhos de acordo com as tabelas de roteamento baseadas em Filtro, juntamente com as respectivas estratégias de roteamento. Como resultado final, as mensagens são enviadas para os Brokers Locais dos Assinantes.

Roteamento de Notificação Distribuído

A função do encaminhamento de notificação distribuído é bastante simples: combinação de notificações com todas as inscrições e entregar a notificação a todos os Assinantes e Brokers vizinhos que tenham uma inscrição correspondente. A estratégia mais simples de roteamento é a entrega por "*flooding*", onde todos os brokers recebem as notificações de eventos. Uma alternativa para substituir *flooding* é o roteamento Baseado em Conteúdo, onde cada broker tem uma tabela de roteamento que é usada para decidir a rota para onde notificações serão entregues, com base em seu conteúdo, para os clientes locais e vizinhos. Comparado com *flooding* o roteamento baseado em conteúdo reduz o número de notificações que são enviadas, mas dificulta o encaminhamento de notificação e introduz a necessidade de atualizar as tabelas de roteamento se as inscrições mudarem. Em geral, as tabelas de roteamento são mantidas por troca de informação entre Brokers, toda vez que houver nova inscrição ou cancelamento de inscrição [MÜHL02]. Três estratégias para o roteamento Baseado em Conteúdo são adotadas no sistema Rebeca: roteamento baseado em Identidade, roteamento baseado em Abrangência e roteamento baseado em Merge [PIETZUCH06], conforme detalhado nas seções 3.1.3, 3.1.4 e 3.1.5.

3.3 Principais trabalhos na área de Roteamento Distribuído de Notificações de Eventos

Na seção seguinte, vamos apresentar alguns trabalhos que abordam soluções de roteamento onde o principal foco é a disseminação de notificações de eventos. Em tais trabalhos encontrados, temos dois objetivos em comum: a redução do número de mensagens trocadas na rede e o tempo de entrega de mensagens.

3.3.1 Pub/Sub Baseado em Conteúdo com Grupos Virtuais

No trabalho de [ZHANG;HU05] a rede de *Brokers* baseada em conteúdo é uma rede "*overlay*", que está organizada em uma única árvore. Os nós assinantes são conectados a um nível da árvore (L_0,..., L_n) que servem como "*servidores*" para inscrição e notificação de evento. Cada nó da árvore mantém uma tabela de inscrição, que registra as inscrições de cada nó. Inscrições de assinantes são agregadas e propagadas a partir das folhas da árvore. Quando uma mensagem é publicada, esta é encaminhada para o nível mais baixo da árvore N através da comparação com as tabelas de inscrição em cada nível. No exemplo de figura 3.26, se a mensagem contém o valor três, esta vai ser transmitida em primeiro lugar para os nós K1 e K0 em seguida, as os nós folhas L0 e L2.

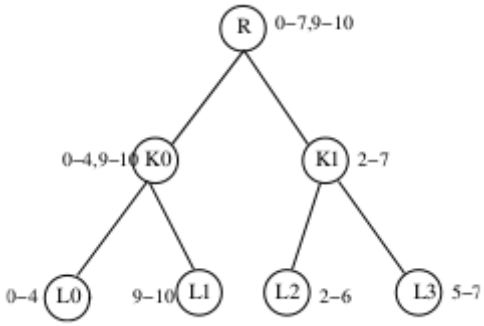

Figura 3.26: Árvore *Publish-Subscribe* [ZHANG;HU05].

Um grupo virtual G_i é uma sub-árvore da árvore original N. Uma sub-árvore consiste de nós folhas L_{i0},..., L_{in} que são nós "servidores" no grupo virtual e de todos os nós intermediários da árvore nos caminhos de folhas até a raiz. Na figura 3.27 temos o grupo G_1 que inclui o nó raiz R de N, onde K_0 e K_1 são nós internos e L_0 e L_1 são nós folhas.

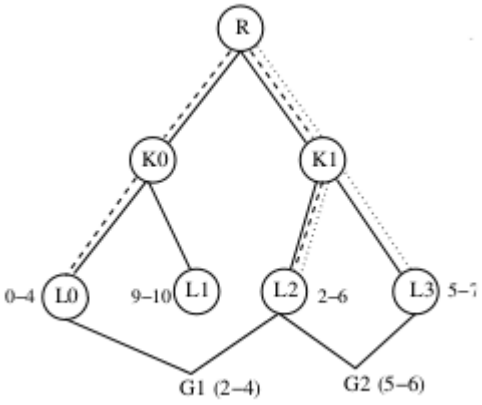

Figura 3.27: Grupos hierárquicos na árvore *Publish-Subscribe* [ZHANG;HU05].

Quando uma mensagem é publicada pelo nó raiz R, está é enviada para os grupos virtuais. Se a mensagem pertence a um grupo G_i por exemplo, a mesma é encaminhada para o nível mais baixo da sub-árvore T_i sem ter correspondência com a inscrição de nenhum nó. Caso contrário, a mensagem é entregue pelo serviço de notificação da árvore original N, onde a inscrição dos nós tenham correspondência em cada nível da árvore.

3.3.2 Roteamento Hierárquico em Grupos

No trabalho de [YOO09] o termo *documento* é utilizado ao invés de *mensagem*. O artigo resume que métodos *de* comunicação no modelo *publish*/*subscribe* para MANETs podem ser definidos como: *Flooding* de documentos (DF), roteamento baseado no destino (DBR) e roteamento baseado em conteúdo (CBR). Os métodos de comunicação são descritos a seguir:

- *Flooding* de documentos: Corresponde à disseminação de eventos para todos os brokers independente do interesse. Cada broker determina destinatários do documento e faz o envio do mesmo. Nenhuma técnica de roteamento especial é necessária visto que é feito broadcast do documento.
- DBR: Inscrições de nós interessados em um evento são registradas e replicadas para todos os brokers. Um broker recebendo uma notificação envia para os seus vizinhos interessados, pois têm armazenado suas inscrições e conhece para quais nós deve enviar a notificação. O custo para o broadcast de inscrições, realizando correspondência de inscrições e o custo para manter a tabela de roteamento para os assinantes, pode ser considerável dependendo do tamanho da rede.
- CBR: Cada nó conhece a inscrição de seus nós vizinhos. Quando um nó recebe uma mensagem, este é encaminha para o nó que tem interesse, conforme a correspondência de inscrição. A topologia de rede para CBR deve ser em árvore, o que resulta em alto custo para manutenção da rede quando há muitos nós na rede.

O modelo proposto é um modelo hierárquico combinando DF e CBR. A comunicação intergrupos é feita por CBR e a comunicação intergrupos é feita por DF. Conforme a figura 3.28, o modelo proposto permite várias árvores na rede. Cada árvore corresponde a um grupo

que é representado por um nó. O tamanho do grupo é limitado para a manutenção da árvore não ficar com um custo elevado. Para a comunicação intergrupos há duas opções. Na primeira, inscrições são encaminhadas para cada nó representante do grupo. A raiz da árvore torna-se o nó que resume inscrições e transmite o resumo para todos os nós representantes dos demais grupos via broadcast.

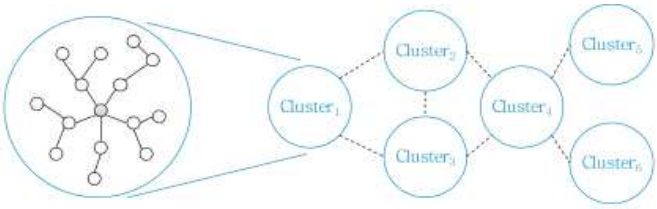

Figura 3.28: Topologia da rede – agrupamento por interesse [YOO09].

Quando um nó raiz recebe um resumo de inscrições de outro grupo, este extrai informações de roteamento do cabeçalho da mensagem, armazena informações do resumo e envia a mensagem aos nós representantes de outros grupos. Quando um documento é publicado, este é entregue aos assinantes no grupo ao longo do caminho de acordo com o método CBR. O grupo raiz verifica o documento com as inscrições, determina grupos destinatários e envia o documento para os grupos selecionados na forma de DBR. No entanto, se aumentar a quantidade de nós na rede, os problemas originais do método DBR permanecem. A segunda opção consiste em usar o método DF. O nó raiz de cada grupo mantém informações sobre grupos vizinhos e transmitem os documentos publicados os seus vizinhos, independentemente de seu interesse. O documento publicado é entregue aos assinantes do grupo ao longo do caminho de acordo com o método CBR. Sendo assim, este método não tem sobrecarga para manter informações de caminho. Isto implica que a taxa de perda documentos será baixa. Portanto, o método de DF é adotado para comunicação de intergrupos. Os mecanismos de inscrições e publicações são descritos a seguir:

- Mecanismo de Inscrições: Quando um nó quer registrar uma inscrição, é feito um broadcast da sua mensagem para todos os nós ao longo da árvore. Cada nó armazena a inscrição em sua tabela de roteamento.
- Mecanismo de Publicação: Quando um nó publica um documento, o documento é encaminhado ao longo do caminho para nós conforme inscrições correspondentes. Cada documento publicado deve ser entregue ao grupo *raiz* para ser transferido para assinantes de

outros grupos. O nó raiz escolhe os grupos vizinhos que devem receber o documento, em seguida acrescenta seu ID do grupo para o documento e envia a mensagem para os grupos selecionados. A mensagem sendo encaminhada ao nó raiz de cada grupo, este escolhe outro nó, que é o representante do grupo, que tem o maior valor de respostas positivas (se está ativo ou não, se há conexão ou desconexão) e o menor número de saltos para chegar a esses grupos. Em seguida, cada nó seleciona o próximo nó e transfere a mensagem do documento a este até que a mensagem chegue a um nó do grupo de destino final.

3.4 Conclusão

Neste capítulo apresentamos os principais trabalhos na abordagem de Roteamento Distribuído de Notificações de Eventos, onde identificamos características para serem aproveitadas na proposta do nosso trabalho. Nos algoritmos de roteamento de operações de inscrição e notificação, primeiramente temos a propagação de mensagens por *Flooding*. No contexto de notificação de eventos, a principal desvantagem de *Flooding* é que notificações podem ser entregues desnecessariamente aos nós do grafo, pois toda notificação é enviada para todos os *Brokers* independentemente da existência de um nó vizinho ter interesse correspondente. Todos os outros algoritmos, o Roteamento Simples, Baseado em Abrangência e Merge, também aplicam *Flooding*, porém com otimizações. No Roteamento Simples a propagação de mensagens para todos os brokers é realizada somente na operação de criação e cancelamento de inscrições. No Roteamento Baseado em Identidade é estendido do Roteamento Simples, porém a entrega de notificações é propagada somente para nós vizinhos onde a inscrição registrada seja correspondente com o conteúdo da notificação. No entanto, ocorre o problema de crescimento de tabelas de roteamento que crescem exponencialmente conforme o número de Assinantes existentes na rede. Já o Roteamento Baseado em Abrangência busca aborda o problema de crescimento de tabelas de roteamento. A propagação de criação e cancelamentos é realizada de uma vez onde existam semelhança e cobertura de filtros, fazendo com que as tabelas tenham o tamanho reduzido.

O conceito do protocolo IGMP [CAIN;DEERING;KOUVELAS;FENNER02] é semelhante aos conceitos de JEDI [CUGOLA;NITTO;FUGGETTA01], CORBA no Serviço de Eventos [OMG04-01] e [YOO09], onde servidores (roteadores no caso do IGMP)

centralizam um tipo de evento para agrupar assinantes que tenham os mesmos interesses, onde a forma da disseminação de notificação de eventos é por *multicast*. Em termos de arquitetura, os serviços de notificação de Hermes [PIETZUCH;BACON;03] e Rebeca [PIETZUCH06] as características de arquitetura são as mesmas: rede *broker*. No entanto, o roteamento em [PIETZUCH;BACON;03] é feito sobre redes "*overlay*" onde um nó é o "ponto final" para que uma mensagem chegue. Apesar de que uma rede *broker* é centralizada conceitualmente, a estratégia de agrupamento para propagar mensagens também é utilizada no trabalho de [YOO09], onde agrupamentos de tipos são definidos. Já no trabalho [ZHANG;HU05] a ideia é que mensagens sejam propagadas em árvores de interesse, onde cada nó de um nível da árvore é responsável por receber e repassar mensagens.

Aproveitamos em nosso trabalho o conceito de "nó final" [PIETZUCH;BACON;03], para que mensagens não sejam repassadas para nós que não tenham ou que não conheçam vizinhos com interesse em tais mensagens. Verificamos também que uma rede sobreposta pode constituir agrupamentos [ZHANG;HU05] ou árvores de interesse [YOO09], para que notificações de eventos sejam entregues somente em cada árvore. Na estratégia de árvores de interesse, a disseminação de notificações pode ser realizada somente dentro de cada árvore, onde somente os nós de um contexto de interesse recebam notificações. Também partimos do pressuposto que em árvores de interesse, a única situação que pode exigir *flooding* é a operação para criar cada árvore. Sendo assim, com base nos trabalhos aqui citados, apresentamos no próximo capítulo nossa proposta.

Capítulo 4

PROPOSTA

Em DEBS, cada registro de inscrição é direcionado para *n* "fontes" (por exemplo, para um *Broker* ou para um publicador) que repassem notificações de eventos, conforme citado no capítulo 2. A forma do estabelecimento de conexão entre assinantes, *brokers* e publicadores reflete diretamente na manutenção de conexões entre nós e nas operações de criação e cancelamento de inscrições, onde também ocorre influencia na eficiência da rota estabelecida para a disseminação de mensagens na rede.

Este capítulo descreve a proposta de algoritmos distribuídos que correspondem à adaptação de conceitos e modelos de disseminação de eventos vistos no capítulo 3. O intuito é propor métodos para o roteamento de mensagens, de tal forma que a cada Notificação de Evento a mensagem seja disseminada somente entre nós que tenham conexões que correspondam ao Evento da notificação, evitando que os nós que não tenham interesse recebam a notificação. Para evitar a centralização, propomos que todo nó na rede tem o papel de *broker*.

Para tal, os algoritmos propostos neste trabalho abordam as seguintes operações:

- Criação e cancelamento de inscrições
- Notificações de Eventos
- Mudança de papéis: Publicador para *Broker* e *Broker* para Publicador.

Neste trabalho, assumiremos que:

- A rede é estática (i.e., não haverá mudança na sua topologia) sobre a camada TCP/IP;

- Os *links* de comunicação entre os nós são bidirecionais e o grafo da topologia da rede é conexo;
- Não há perda de mensagens na comunicação entre os nós.

4.1 Definições

Para o acompanhamento da proposta desenvolvida neste trabalho, antes é necessário que algumas definições sejam apresentadas.

4.1.1 Publicadores e Brokers

- Designamos um Publicador sendo P_S, onde S é um conjunto de tipos de eventos S = $\{\tau_1, \tau_2, ..., \tau_n\}$
- Designamos $\{P_S\}$ o conjunto de todos os Publicadores.

Um publicador é o nó responsável por emitir e publicar notificações de eventos, as quais devem ser entregues para os nós interessados em tais tipos de eventos. Um nó publicador também realiza o papel de um nó Broker.

- Designamos um Broker sendo *B*

Definimos que um *Broker* é um nó que não é um Assinante e nem um Publicador, ou seja, não tem interesses e também não publica eventos, sendo somente um transmissor de mensagens. Este também pode assumir a troca de papel tornando-se um nó Publicador ou vice-versa.

4.1.2 Assinantes e Inscrições

Designamos um Assinante sendo *x*. Um assinante se inscreve, através de inscrições, para receber notificações de vários tipos de eventos.

Designamos uma Inscrição *I* de um assinante *x* sendo $I_i(x)$. Como todo assinante pode registrar várias inscrições de diferentes tipos de eventos, cada inscrição é única com a identificação de um índice *i*, como por exemplo: $I_1(x)$, $I_2(x)$, $I_3(x)$,..., $I_n(x)$. A inscrição $I_i(x)$ é uma *tupla* composta por:

$<\{\tau i\}, \{ExpReg\}>$, onde

- τi é uma sequência de caracteres que formam um nome, o qual identifica o tipo do *Evento*.
- *ExpReg* é uma expressão regular que permite que o nó Assinante informe restrições de interesse no tipo de evento. Neste atributo, são especificados os valores de interesse no Evento. As restrições informadas na expressão regular podem ser do maior até o menor nível de especificação de interesse. Conforme visto na seção 2.2.2, o caractere '*' indica “todos”, análogo a “recupere todas as notificações, independente deste atributo”. Ou também a expressão regular pode conter restrições de todos os grupos disponíveis para determinado tipo de evento. Por exemplo, se na inscrição para obter notícias o assinante desejar somente receber somente notícias de um determinado jornal e de uma determinada coluna de tal jornal, este deve informar a expressão regular para que a restrição seja identificada na notificação do evento. Para o assinante especificar tal interesse, teríamos os seguintes conjuntos disponíveis para serem utilizados na expressão regular:

[Jornal], *[Coluna]*, [Assunto], [Autor]

Outro exemplo seriam conjuntos de expressões regulares para obter notificações de eventos sobre voos em aeroportos, onde os possíveis conjuntos seriam estes:

[Companhia], *[Horário]*, *[Origem]* e [Destino]

Exemplos de inscrições:

1. Um assinante *x*' registra duas inscrições, sendo uma para receber notificações de eventos do tipo “notícias” e a outra para receber notificações do tipo de evento “voos”. As inscrições de *x*' seriam compostas pelo seguinte conteúdo:

I_1(x') = <{“Notícias”}, {[“Jornal-*X*”], [“*Coluna*-*Y*”], [“*Assunto*-*A*”], [“*Autor*-*”]}>

I_2(x') = <{“Voo”}, {[“Companhia-*”], [“Horário-15:00”], [“Origem-A”], [Destino-B”]}>

4.1.3 Eventos e Notificações

Um evento e é qualquer acontecimento no escopo presente.

- Designamos E_τ um evento o qual é identificado pelo tipo de evento τ.
- Designamos um evento emitido por um Publicador P_S sendo $E_\tau(P_S)$, o qual identifica o tipo de evento τ que pertence ao conjunto S de P_s.

Na ocorrência de um Evento uma notificação, que é simplesmente uma mensagem, é criada pelo Publicador que emitiu o evento (observador do evento). O processo de divulgação da notificação para todos os interessados é chamado de publicação. Conforme visto no capítulo 2, o modelo de estrutura (“*template*”) de uma Notificação deve seguir o mesmo modelo da Inscrição para que a correspondência de conteúdo possa ser realizada. Para tal, designamos:

- Uma Notificação gerada por um publicador é definida por $N(P_s)$
- Uma Notificação é uma tupla designada por $N(P_s)$ = <{τi}, {$E\tau(P_s)$}>, onde τi é um valor que identifica o Tipo de Evento τ publicado.

$E\tau(P_s)$ é um conjunto de valores que corresponde a um evento, sendo o evento de um tipo τ. Cada valor que compõe o evento $E\tau(P_s)$, corresponde aos possíveis grupos de expressões regulares disponíveis para o tipo de evento, dos quais foram especificados em inscrições registradas. Como exemplo, as inscrições dos assinantes x' e x'' para receber eventos de notícias, são correspondentes com a notificação $N(P_s)$:

$Ii(x')$ = <{“Notícias”}, {[“Jornal-*X*”], [“*Área-X*”], [“*Assunto-X*”]}>
$Ii(x'')$ = <{“Notícias”}, {[“Jornal-*X*”], [“*Área*-*], [“Assunto-*”]}>

$N(P_s)$ = <{“Notícias”}, {[“X”], [“X”], [“X”]}>

Outro exemplo que podemos ter é uma inscrição para obter notificações de vôos em um aeroporto. Sendo a inscrição I_j do assinante x'', temos:

$I_j(x'')$ = <{“*Voo*”}, {[“Companhia Aérea-*X*”], [“*Origem-CTBA*”], [“*Destino-MG*”]}>

$N(P_s)$ = <{“*Voo*”}, {[“*X*”], [“CTBA”], [“MG”]}>

Um ponto a ser considerado, é a questão da ordem em que notificações de eventos são recebidas por assinantes. Pode ocorrer a situação em que dois eventos, com origem de publicadores diferentes, sejam entregues a um mesmo assinante onde a mensagem tenha passado por caminhos diferentes (um caminho com maior quantidade de saltos no grafo que o outro). Isto significa que a segunda notificação de evento pode ser entregue antes que a primeira, ou vice-versa. Em nossa proposta, consideramos a necessidade da abordagem de notificações causais, no entanto tal abordagem é sugerida no capítulo 7.

4.2 Arquitetura

Em nossa proposta, a conexão entre assinantes e publicadores se dá sob árvores de interesse de assinantes onde todos os publicadores, do mesmo tipo de evento de interesse do assinante, são abrangidos nestas árvores de interesse.

A arquitetura proposta neste trabalho não é uma arquitetura de “rede *Broker*” como definido em [PIETZUCH06-Cap2], [ZHANG;HU05], [YOO09] e [CUGOLA06]. Na nossa proposta, todo nó na rede deve ser capaz de repassar uma mensagem que tenha recebido, ou seja, todo nó deve desempenhar o papel de um Broker. Se somente os nós publicadores forem *brokers*, o repasse da mensagem seria diretamente de publicador para assinante ocorrendo à centralização de disseminação de eventos e conhecimento de publicadores por parte de assinantes conforme ocorrência em “*redes Brokers*”. Os componentes presentes na arquitetura do sistema são ilustrados na figura 4.1.

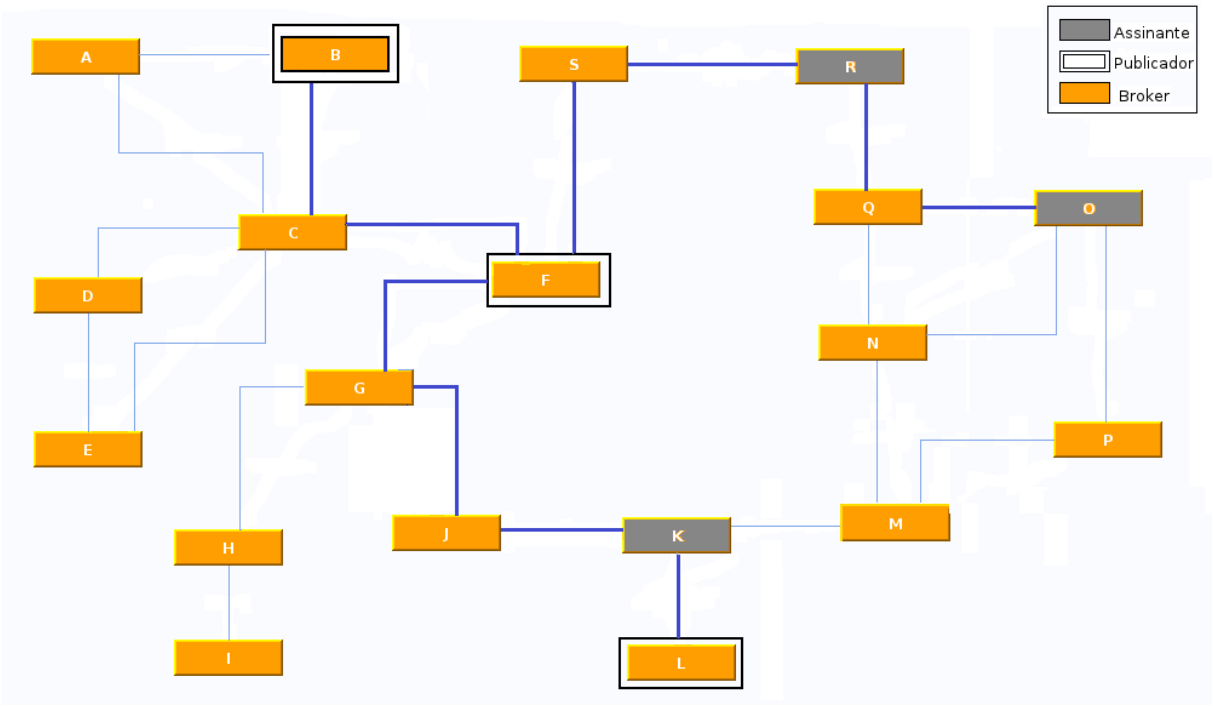

Figura 4.1: Um exemplo de Arquitetura da rede.

Conforme a figura 4.1, os três tipos de nós podem ter conexão direta entre si, no qual um Assinante pode ter conexão com um *Broker* e com um Publicador, assim equivalente para esses três componentes. Os nós que tenham interesse em comum são interligados, para que notificações de eventos sejam entregues somente para os nós com mesmo interesse. Para tal, a ideia é criar um grafo formado por árvores referentes a inscrições registradas. Cada árvore corresponde a uma inscrição, onde cada Assinante é o nó raiz dessa árvore, os nós Brokers e Publicadores podem ser nós intermediários, e todos os nós que são folhas são Publicadores. Os nós Publicadores quando publicam uma notificação de eventos, propagam a notificação somente dentro da árvore de inscrição em que estão inseridos. Sendo x um Assinante de eventos do tipo τ, designamos a "árvore de inscrição" assim criada, com raiz em x e abrangência de todos os publicadores de eventos do tipo τ por $A_\tau(x)$. Qualquer nó pode fazer parte de mais de uma árvore de inscrição, pois cada árvore está sobreposta na rede.

O encaminhamento de eventos e demais mensagens, se dá sobre redes "*overlay*", onde cada nó Assinante é o nó raiz de uma árvore de abrangência de publicadores sobreposta na qual é criada a partir de cada inscrição registrada. Cada assinante encontra um conjunto de publicadores conforme seu interesse, onde o tipo de evento de tais publicadores é correspondente ao tipo de evento de interesse do assinante. O conjunto de publicadores

encontrados por assinantes compõe a árvore de abrangência de publicadores, a qual é definida por “árvore de inscrição”. A cada inscrição recebida por um nó (onde a inscrição foi emitida para encontrar os publicadores correspondentes), este conhece o vizinho que lhe entregou a inscrição. Assim, quando houver notificação de um evento, a mensagem é repassada para o nó do qual a inscrição foi recebida. Se a partir de um nó, os caminhos dessas árvores até os assinantes divergirem, o nó será denominado um *“fork node”*, pois deverá propagar a mensagem por dois ou mais caminhos diferentes. Por exemplo, o nó F é um “*fork node”*, *pois* deverá duplicar as mensagens vindas do publicador B, para serem entregues aos assinantes R e K, seguindo suas árvores de inscrição $A_\tau(K)$ e $A_\tau(R)$. Por fim, os nós assinantes estão presentes na rota de entrega dos publicadores que emitem o mesmo tipo de evento de interesse. Cada árvore $A_\tau(x)$ tem as seguintes características:

- Todo *x*, que é raiz da árvore, é um Assinante;
- Todo nó folha é um Publicador, mas nem todo nó Publicador é necessariamente uma folha.

Sendo $G = (V, E)$ o grafo da topologia de nós, onde V é um conjunto não vazio de vértices e E um conjunto de pares de vértices ($E \subseteq (V \times V)$), a árvore $A_\tau(x)$ é definida por (V', E') onde $V' \subseteq V$ contendo os nós (assinantes, publicadores e *brokers*) nos caminhos ligando todos os Publicadores de eventos do tipo τ ao Assinante *x*, e onde $E' \subseteq V' \times V' \subseteq E$.

Sendo *S* o conjunto de todos os assinantes de eventos de um tipo τ, então $\Sigma_\tau = \{A_\tau(x), \forall\, x \in S\}$. Sendo assim, temos $|\Sigma_\tau| = |S|$ árvores sobrepostas.

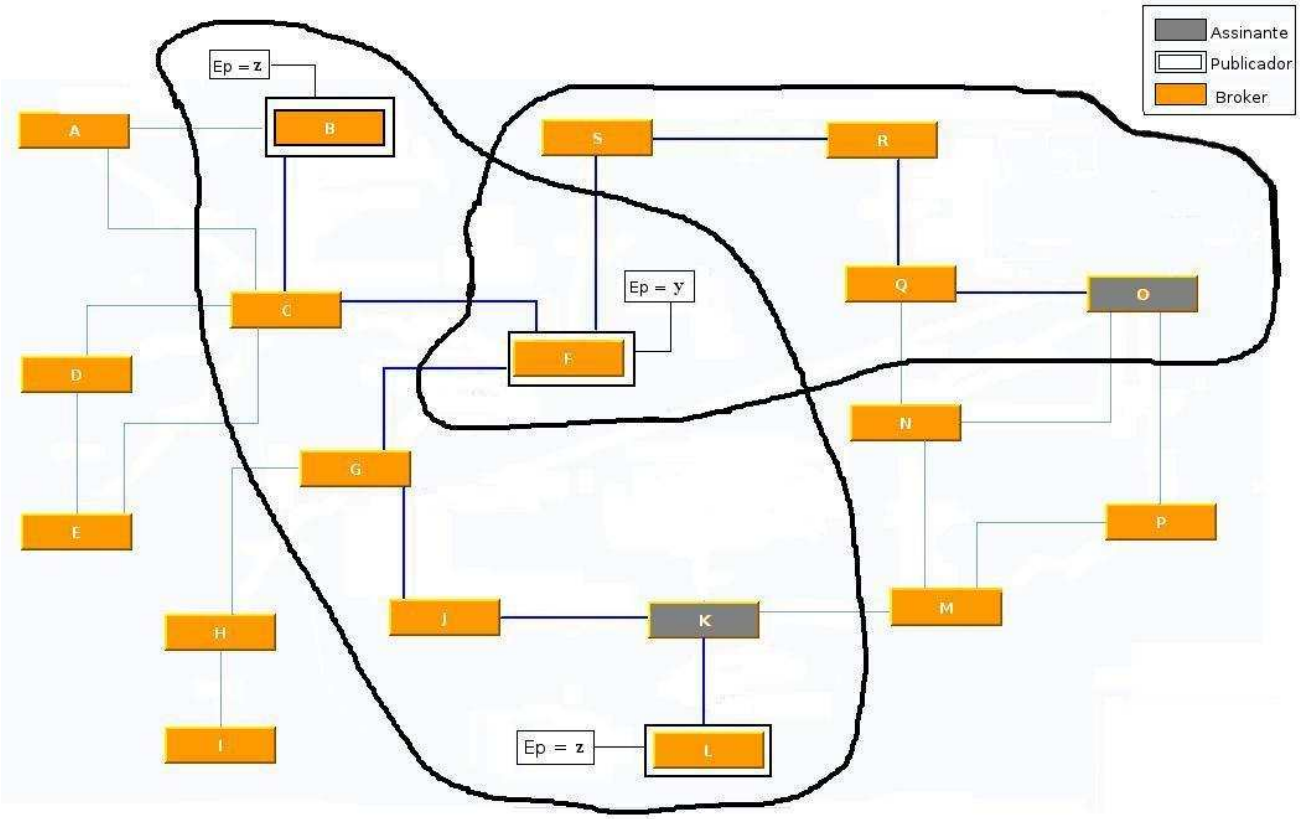

Figura 4.2 - Árvores de Inscrição – $A_y(O)$ E $A_z(K)$.

Na figura 4.2 temos um exemplo de um grafo formado por duas árvores sobrepostas $A_y(O)$ e $A_z(K)$. A primeira árvore é referente à inscrição do nó O, na qual o interesse do nó O é o tipo de evento *y*. Entre os nós publicadores B, F e L, o único nó que publica eventos do tipo *y* é o publicador F. Portanto para receber notificações do tipo de evento *y*, o nó O precisa ter a abrangência do nó F. Sendo assim, o caminho do nó O que leva à conexão para o F, fica sendo os seguintes nós: O, Q, R, S e F. A segunda árvore de inscrição se refere à inscrição do assinante K, na qual o Evento de interesse é do tipo *z*. Os nós que publicam eventos do tipo *z* são os publicadores B e L, portanto para receber notificações é necessário que o nó K esteja na abrangência de tais publicadores. Então, a árvore de inscrição do nó K é formada pelos nós que levam aos publicadores sendo os seguintes caminhos: K, L; K, J, G, F, C, B.

- Dessa forma, $A_y(O)$ corresponde à inscrição do nó *O*, onde é incluída a abrangência de todos os publicadores de eventos do tipo *y*.

$A_y(O) = (V, E)$

V = {O, Q, R, S, **F**}, onde **F** é um publicador (único nesse exemplo) do tipo de evento *y*.

E = {(O, Q), (Q, O), (Q, R), (R, Q), (R, S), (S, R), (S, F), (F, S)}.

A_z(K) sendo a árvore de inscrição correspondente a inscrição do nó *K:*

$A_z(K) = (V, E)$, onde

V = {L, K, J, G, F, C, B}, onde L, F e B são publicadores, sendo somente os nós L e F publicadores do tipo de evento *z*.

E = {(L, K), (K, L), (K, J), (J, K), (J, G), (G, J), (G, F), (F, G), (F, C), (C, F), (C B), (B, C)}.

Dessa forma, buscamos diminuir a quantidade de troca e do tempo na entrega de tais mensagens, pois o percurso de cada mensagem é somente dentro de uma árvore de inscrição, onde outros nós que estão fora da árvore de inscrição não têm conhecimento das mensagens trocadas. Como $A_\tau(x)$ só é construída no momento da criação de inscrições, o custo da construção é compensado pela redução significativa posterior na disseminação de notificações de eventos.

4.3 Estrutura de dados

O roteamento para entrega de mensagens na rede, o qual é realizado por cada nó, é desempenhado dentro da respectiva árvore de inscrição $A_\tau(x)$ em que o nó *x* está inserido. A disseminação de mensagens é realizada para cada nó que é vizinho de *x*, onde cada nó vizinho seja um caminho para a entrega da mensagem. Para tal, é realizado o armazenamento e gerenciamento de informações em uma estrutura de dados que corresponde a uma tabela de roteamento. Cada nó inserido na árvore $A_\tau(x)$ tem a informação de nós vizinhos interessados no tipo de evento τ, onde cada par de nós é referente a quem *x* deve receber e repassar mensagens. Esta informação de nós vizinhos é armazenada na tabela de roteamento. Sendo assim, definimos a estrutura de tabelas de roteamento.

4.3.1 Tabelas de Roteamento

- Designamos TR_τ sendo a Tabela de Roteamento, a qual é referente a um tipo τ de evento, para todos os tipos de nós da rede (Assinante, Publicador e Broker). Cada nó pode ter várias tabelas de roteamento, sendo cada tabela referente a um tipo de evento τ.

Inscrição	Proximo	Origem

Tabela 4.1: Tabela de roteamento TR_τ

Conforme a tabela 4.1, temos a seguinte estrutura para cada tabela de roteamento:

- A coluna ‘inscrição’ representa cada inscrição distinta referente ao tipo evento de TR_τ.
- A coluna ‘próximo’ e ‘origem’ representam os nós de quem um nó *x* pode receber e enviar mensagens.

Sendo assim, a coluna 'Inscrição' armazena todas as inscrições registradas por Assinantes, sendo o campo chave da tabela. A coluna 'próximo' armazena um conjunto de nós para quem uma notificação de evento deve ser repassada e a coluna 'Origem' armazena o conjunto de nós dos quais as mensagens são recebidas. Como exemplo, a figura 4.3 mostra as tabelas de roteamento de cada nó que está inserido nas árvores de inscrição $A_y(O)$, $A_z(K)$ e $A_z(E)$, onde o nó Assinante O é *raiz* da árvore A_y(O). Sendo F o único publicador do tipo de evento *y*, os nós de *O* até F têm a inscrição $I_{i(o)}$ na tabela de roteamento. Para o evento do tipo *z*, existem dois publicadores: os nós B e L. Cada assinante do tipo de evento *z* tem uma inscrição registrada que chega a tais publicadores. O assinante K, que é raiz de $A_z(K)$, e todos os demais nós que chegam até L e B, têm em suas tabelas de roteamento a inscrição do nó K, assim o mesmo ocorre para o nó E.

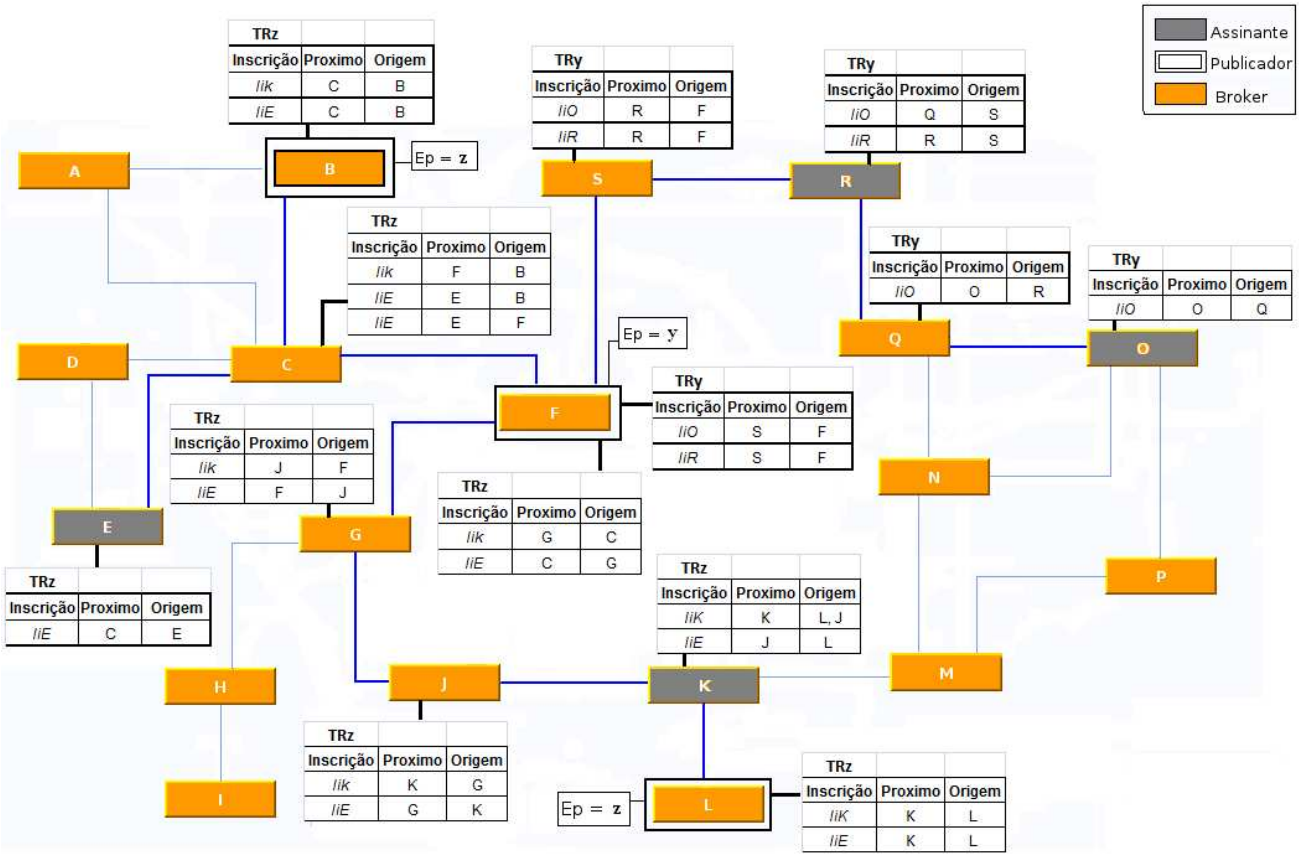

Figura 4.3: Tabelas de roteamento de todos os nós das Árvores de Inscrição $A_z(E)$, $A_z(K)$, $A_y(R)$ e $A_y(O)$.

As informações das colunas ‘próximo’ e ‘origem’ são utilizadas para finalidades diferentes. Por exemplo, quando um nó recebe uma notificação, este deve repassar a notificação para os nós que constam na coluna ‘próximo’ do registro da tabela onde o conteúdo da coluna ‘inscrição’ for correspondente ao *template* da notificação. Quando um nó recebe uma mensagem de cancelamento de inscrição, a mensagem de cancelamento deve ser encaminhada para os nós que constam na coluna ‘origem’, pois estes são os nós de rotas que levam aos nós publicadores. Seguindo o exemplo da figura 4.3, a tabela de roteamento do assinante *O* seria definida por $TR_y(O)$. O nó *Q* estaria na coluna ‘origem’ e o próprio nó *O* estaria na coluna ‘próximo', visto o seguinte: *O* é nó que emitiu a inscrição, sendo *O* a ‘origem’ da disseminação da inscrição. Já o nó *Q* é o ‘próximo’ nó vizinho de *O* na árvore $A_y(O)$, o qual está na rota que chega aos publicadores de tipos de eventos onde *O* tem interesse. Assim, na tabela 4.2 temos a ilustração da tabela de roteamento de nó $TR_y(O)$.

TRy		
Inscrição	**Proximo**	**Origem**
IiO	O	Q

Tabela 4.2: Tabela de roteamento $TR_y(O)$

Outro exemplo é a tabela $TR_z(K)$ do assinante K. Conforme a tabela 4.3, existem dois assinantes do tipo de evento z, que são os assinantes E e K. Assim, os publicadores de tipos de evento z devem ter conhecimento das inscrições registradas. Na tabela de roteamento, para a inscrição do nó K, os nós L e J estariam na coluna 'origem', pois estes são rota para os publicadores do evento *z*, os quais estão na árvore $A_z(K)$, e o nó K estaria na coluna 'próximo', pois este mesmo que emitiu o registro de inscrição. Para a inscrição de E, o nó L estaria na coluna 'origem', pois L é um publicador do tipo de evento z, e na coluna 'próximo' teria o nó K, pois K tem conexão com J que é o próximo nó da rota que leva a E que é outro assinante.

TRz		
Inscrição	**Proximo**	**Origem**
IiK	K	L, J
IiE	J	L

Tabela 4.3: Tabela de roteamento $TR_z(K)$.

O exemplo da tabela de roteamento do nó *F*, o qual é um publicador que não é *folha* (após F existem outros publicadores), onde este também é responsável por repassar mensagens. Portanto o publicador F teria duas tabelas de roteamento. Uma tabela seria referente às inscrições do tipo de evento *z* registradas pelos assinantes E e K, conforme a tabela 4.4.

TRz		
Inscrição	**Proximo**	**Origem**
IiK	G	C
IiE	C	G

Tabela 4.4: Tabela de roteamento TRz(F).

A segunda tabela do nó F seria referente às inscrições do tipo de evento *y* registradas pelos assinantes O e R, conforme a tabela 4.5.

TRy		
Inscrição	**Proximo**	**Origem**
IiO	S	F
IiR	S	F

Tabela 4.5: Tabela de roteamento TRy(F).

Com base nas tabelas de roteamento, a rota para enviar notificações aos assinantes interessados é estabelecida conforme os registros referentes à correspondência entre uma determinada ‘inscrição’ e do modelo de notificação do evento. A partir da busca na tabela pela inscrição que corresponda ao modelo da notificação, os nós que estão na coluna ‘*próximo’* são incluídos no caminho a ser percorrido para enviar tal mensagem. No exemplo das tabelas 4.4 e 4.5, o nó F recebendo uma notificação do tipo de evento y, este enviaria a notificação do evento de tipo y para o nó S. Caso o nó F receba uma notificação do tipo de evento z, então a o envio da notificação seria para o nó G e/ou C, dependendo do nó de quem recebeu a notificação. Portanto, o conteúdo das tabelas de roteamento é a base para o fluxo de encaminhamento de mensagens entre todos os nós de árvores de inscrição.

4.4 Algoritmos propostos

As próximas seções abordam as operações de DEBS executando sobre a rede de arquitetura apresentada na seção anterior. Nas próximas seções detalhamos as operações referentes a Inscrições, Notificação e mudança de papel entre Publicadores e *Brokers*.

4.4.1 Criação de Inscrições

O primeiro passo na rede é o estabelecimento da criação de conexões entre nós, formando um Grafo não direcionado. A conexão dos nós é estabelecida no momento da operação de Criação de Inscrição, onde conexões entre nós referentes a uma inscrição formam uma “árvore de inscrição”. Mesmo que existam inscrições idênticas, o assinante de cada

inscrição forma uma árvore distinta. Todos os nós que estão no caminho do assinante até os publicadores, precisam ter o conhecimento de todas as inscrições registradas. Isso evita que em casos onde um nó de um nível maior na árvore cancele sua inscrição, que este tenha a responsabilidade de alocar todos os nós que estão em um nível abaixo.

Para a formação de cada árvore no grafo, tomamos como base o algoritmo de Árvores de Abrangência [SANTORO06-Cap.2], porém fazendo com que a abrangência seja de nós Publicadores apenas e não de todos os nós do grafo. Na formação de cada árvore, todo nó assinante registra sua inscrição, a qual é entregue para os nós Publicadores que emitem eventos correspondentes ao evento que consta na inscrição registrada. A ordem da criação de uma árvore obedece à ordem "*first-in-first-out*" (FIFO), ou seja, os publicadores recebem inscrições na mesma ordem em que as mesmas foram emitidas. Um Assinante *x* ao emitir uma inscrição, repassa uma mensagem Q[4] para todos os seus vizinhos, na qual consta a seguinte pergunta: "é meu vizinho na árvore de inscrição e é um nó publicador do mesmo tipo de evento da minha inscrição?".

A mensagem *Q* é definida com sendo a seguinte tupla:

$$Q = <\{I_i(x)\}>$$

onde $I_i(x)$ é a inscrição a ser registrada.

Para enviar a mensagem Q, o nó *x* deve identificar um subconjunto de vizinhos N(*x*), onde os quais são vizinhos na árvore de inscrição de *x*:

$$\text{vizinhos\_inscrição}(x) \subseteq N(x)$$

Um nó vizinho ∈ N(*x*) ao receber a mensagem *Q*, este responde *x* e repassa a inscrição para seu subconjunto N(*x*). Dessa forma, todos os nós da rede devem receber *Q* para a criação da árvore de inscrição de *x*. Como exemplo de disseminação da mensagem *Q*, a figura 4.9 ilustra um cenário de registro da inscrição do nó K, onde $I_i(k)$ é definida pela seguinte tupla:

[4] Terminologia utilizada em [SANTORO06-Cap.2].

I_i(k) = <{“X”}, {[“Jornal-*A*”], [“*Area-B*”], [“*Assunto-C*”]}>

Na figura 4.4, cada número dentro de um círculo é a sequência em que a mensagem foi transmitida. Onde houver números repetidos, significa que o processamento da mensagem foi executado de forma concorrente, no qual a mensagem pode ter chegado antes ou depois para o nó em questão (por exemplo, as mensagens trocadas entre os nós A e B ou D e E).

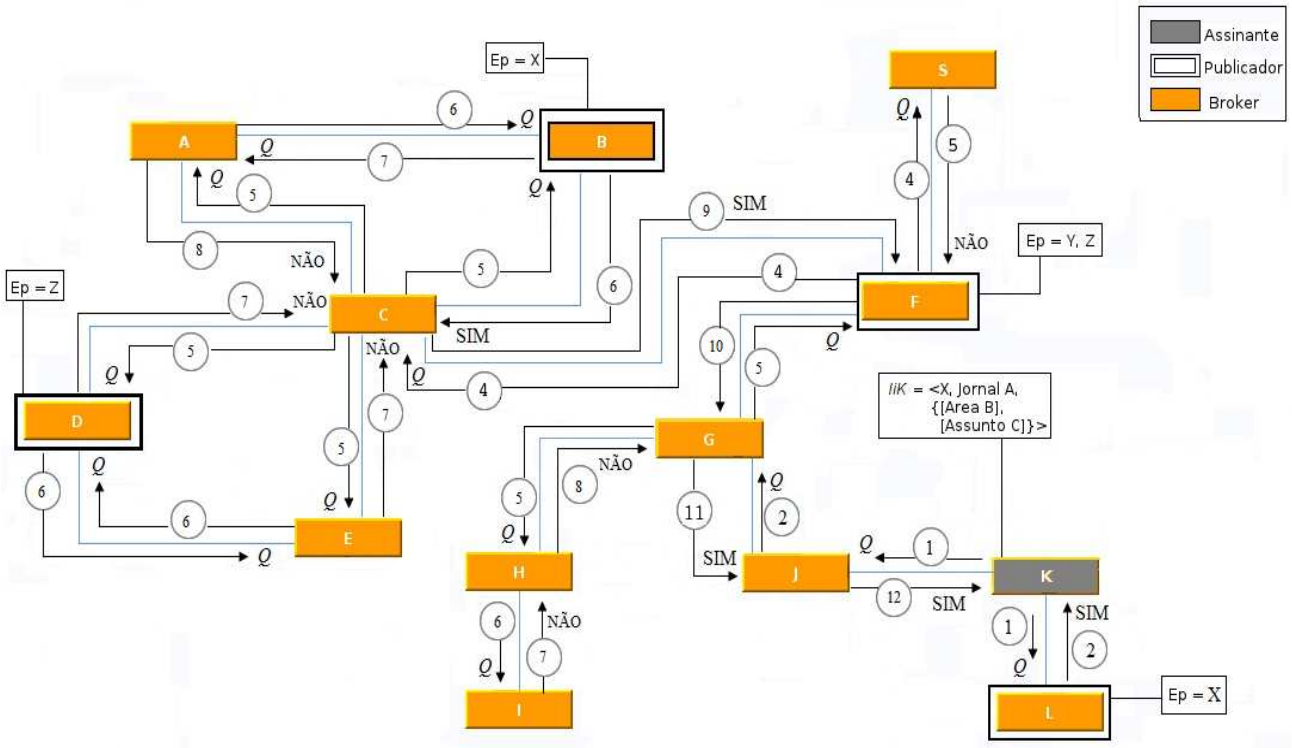

Figura 4.4: Nova inscrição - disseminação da mensagem *Q*.

Cada nó recebendo a mensagem Q repassa a mensagem conforme os passos abaixo.

Se for a primeira vez que o nó está recebendo Q de um vizinho *x*, este verifica:

1. Se o nó é Publicador, então:

Este responde “SIM” imediatamente para *x*, porém continua repassando *Q* para seus vizinhos, pois podem existir outros Publicadores, os quais também devem receber *Q*. Por outro lado, se o nó já respondeu “SIM” (independente da próxima resposta que irá receber de seus vizinhos), este não repassará mais respostas para *x*, pois já tinha respondido “SIM” anteriormente.

2. Se o nó não é um Publicador, então:

Este repassa Q para seus vizinhos, onde fica aguardando a resposta de todos os vizinhos.

- Se o nó recebe uma resposta “SIM”, então repassa “SIM” imediatamente para x. Após o envio do “SIM” este não repassará mais respostas para x;
- Se o nó recebe um “NÃO”, então este só enviará “NÃO” para *x* após receber a resposta de Q de todos os seus vizinhos ou também se ocorrer o seguinte:

1. Se não for a primeira vez que o nó está recebendo Q, então o nó faz soma a contagem de vezes que recebeu Q. Se o número da contagem for igual ao número de vizinhos, então o nó considera a resposta um “NÃO” e repassa a resposta “NÃO” para *x*.

Cada nó atualiza sua tabela de roteamento $TR_{\tau}(x)$ após enviar e receber a resposta de Q. A única coluna que pode modificar após um nó ter enviado a resposta de Q, é a coluna 'origem'. Somente após receber a quantidade de respostas igual o número de vizinhos para os quais enviou Q, é que o nó conclui a atualização de conteúdo da coluna 'origem' de $TR_{\tau}(x)$:

Envio de resposta:

- Se o nó enviou “SIM”, mas não recebeu “SIM” de nenhum dos seus vizinhos, então o seu identificador permanece na coluna 'origem' (pois este é um nó Publicador *folha*).
- Se o nó enviou e recebeu um “SIM”, então o identificador dos nós de quem recebeu a resposta “SIM” é atualizado na coluna 'origem' de $TR_{\tau}(x)$.

Recebimento de resposta:

- Se o nó recebeu um “SIM” e tem um vizinho para repassar a resposta, então a coluna 'próximo' recebe o identificador do nó para quem está respondendo. Caso contrário significa que o nó é um Assinante e foi ele mesmo que disparou a mensagem Q pela primeira vez. Neste caso, a coluna 'próximo' recebe o identificador do próprio nó.

Assim, um nó considera o nó de quem está recebendo a resposta como sendo um nó que é ‘origem’ de notificações, e o nó para quem está repassando a resposta de Q (que é o nó de quem recebeu Q) como sendo o ‘próximo’ nó.

- O identificador do nó de quem recebeu a resposta “SIM” (pode ser ele mesmo), é armazenado na coluna ‘origem’;
- O identificador do nó para quem está repassando a resposta “SIM” (é o nó de quem recebeu *Q*) é armazenado na coluna 'próximo'.

A figura 4.5 mostra a árvore de inscrição construída referente ao resultado da disseminação de mensagens *Q*, conforme o exemplo da figura 4.9.

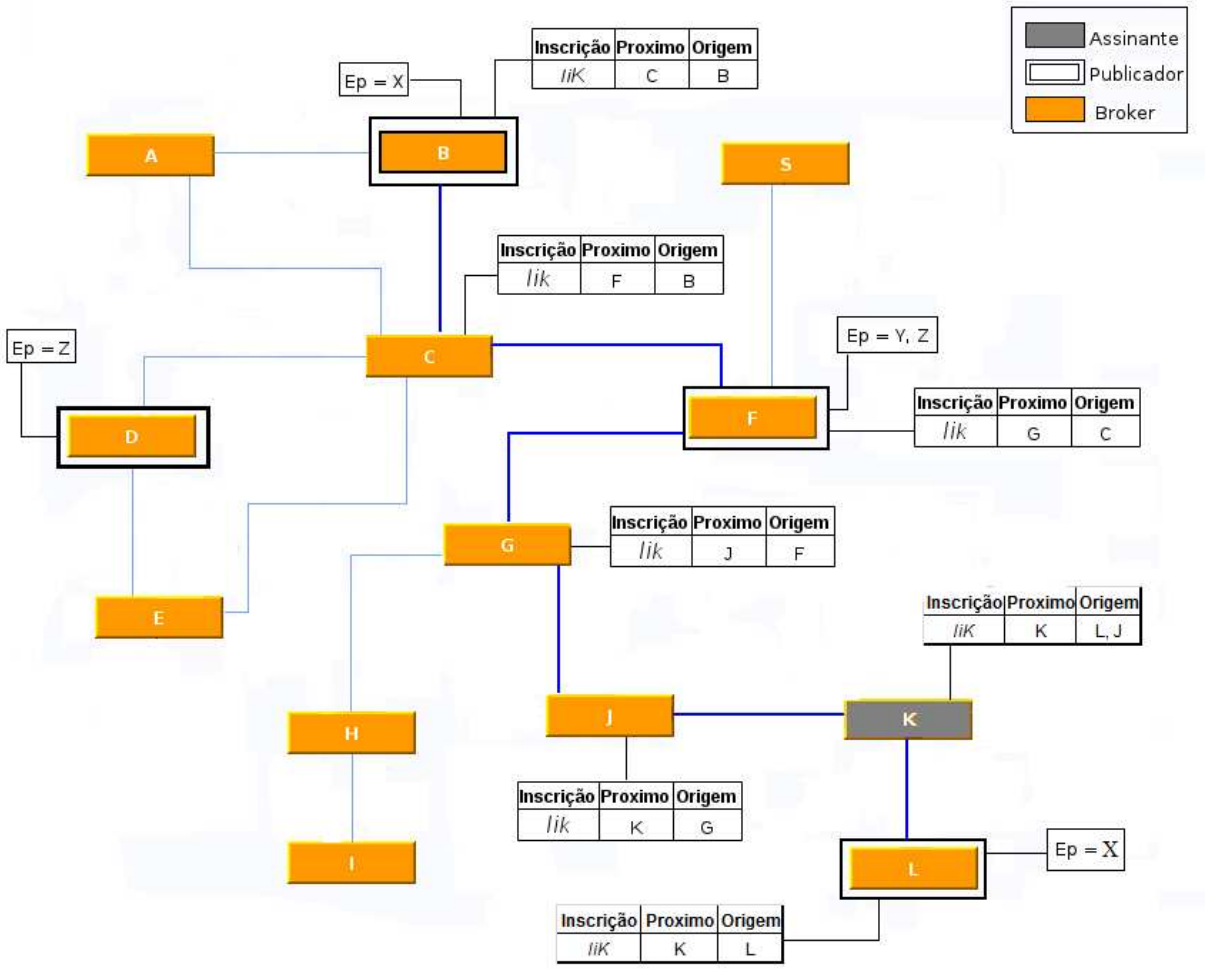

Figura 4.5: Árvore de inscrição formada após a disseminação da mensagem *Q*.

A seguir o algoritmo para ***Construir_Árvore_Inscrição*** é a formalizado. Este procedimento é chamado por cada Assinante no momento do registro de nova inscrição.

Antes, algumas pressuposições precisam ser explicitadas:

Após a conexão de nós na rede *overlay*, todo nó na rede tem seus *status* como OCIOSO, o qual significa que o nó está pronto para receber mensagens. Assim, para registrar uma inscrição, cada nó Assinante altera seu status *S* para *ASSINANTE*. Após isto, o nó irá fazer chamada para o procedimento *Construir_Árvore_Inscrição*.

- Status: *S* = {ASSINANTE, ATIVO, CONCLUÍDO, OCIOSO} (estados possíveis)

S_{INIT} = {ASSINANTE, OCIOSO} (estados iniciais)
S_{TERM} = {CONCLUÍDO} (estado final)

1. Inicialmente, um nó está no estado ASSINANTE e os demais estão com no estado OCIOSO.
2. No fim do algoritmo, todos os nós estarão com no estado CONCLUÍDO.

Assim, temos o seguinte:

- Um nó, ao fazer a chamada de *Construir_Árvore_Inscrição*, tem seu status S_{INIT} = ASSINANTE
- N(*x*) é o conjunto de todos os vizinhos de *x* na rede overlay.

Início *Construir_Árvore_Inscrição(x)*

```
ASSINANTE
Espontaneamente
Início
    nó_raiz := verdadeiro;
    arvore_inscricao := ∅
    Enviar("Q") para N(x);
    tornar_status (ATIVO);
Fim

ATIVO
```

```
Receber(MENSAGEM)
Início
    Recebeu_resposta := Recebeu_resposta + 1;
    Enviou_resposta_SIM = Enviou_resposta_SIM + 1;

    Se (MENSAGEM = “Q_SIM”) então
        arvore_ inscricao := {origem};
        TR_t(x)^Origem := {origem};
        TR_t(x)^Próximo := Nó_que_enviou_Q;
    FimSe

    Se (nó_raiz != verdadeiro)
        Se (MENSAGEM = “Q_SIM”) então
                Se (Enviou_resposta_SIM = 1) então
                    Enviar("Q_SIM") para Nó_que_enviou_Q;
                    Enviou_resposta_SIM = Enviou_resposta_SIM + 1;
                FimSe
        Senão
                Se (Recebeu_resposta = |N(x) - Nó_que_enviou_Q|) então
                  Enviar("Q_NÃO") para Nó_que_enviou_Q;
                FimSe
        FimSe
    FimSe

    Se (Recebeu_resposta = |N(x) - Nó_que_enviou_Q|) então
        tornar_status (CONCLUÍDO);
    FimSe
Fim

OCIOSO
Receber(Q)
Início
    Nó_que_enviou_Q := {origem};
    nó_raiz := falso;
    Recebeu_mensagem := recebeu_mensagem + 1;
```

```
Se (recebeu_mensagem = 1) então
    Se (x ∈ {P} ^ Eτ(x) ∈ Inscricao(Q)) então
        TRτ(x) Origem:= x;
        TRτ(x) Próximo := Nó_que_enviou_Q;
        Enviar("Q_SIM") para Nó_que_enviou_Q;

        Se (|N(x) - Nó_que_enviou_Q | > 0) então
            Enviar("Q") para {N(x) - Nó_que_enviou_Q)};
            tornar_status (ATIVO);
        Senão
            tornar_status (CONCLUIDO);
        FimSe
     Senão
        Se (|N(x) - Nó_que_enviou_Q | = 0 || recebeu_mensagem = |N(x)|)
            Enviar("Q_NÃO") para Nó_que_enviou_Q;
            tornar_status (CONCLUIDO);
        Senão
            Enviar("Q") para {N(x) - Nó_que_enviou_Q)};
            tornar_status (ATIVO);
        FimSe
    FimSe
FimSe

Fim
```

Fim *Construir_Árvore_Inscrição*

No exemplo da figura 4.9 onde o nó K registra uma nova inscrição, o procedimento ***Construir_Árvore_Inscrição(x)*** é executado da seguinte forma:

1. Primeiramente, o nó K envia a mensagem Q para seus vizinhos J e L.
2. O nó L ao receber Q, responde imediatamente "SIM" para o nó K, pois L é publicador do mesmo tipo de evento da inscrição recebida em Q. Até este momento o nó K já tem o primeiro nó filho de sua árvore.

3. O nó J como não é publicador, não responde nada para o nó K e envia Q para seu vizinho G, aguardando a resposta do nó G, para assim responder o nó K.

4. G repassa Q para seus vizinhos F e H.

5. O nó H repassa Q para seu vizinho I, e o nó F repassa a mensagem Q para os nós C e S. Então o nó I verifica que não é um publicador e que também não tem vizinhos para repassar Q.

6. Assim o nó I responde “NÃO” como resposta para o nó H. O nó H recebe a resposta do nó I, onde verifica que recebeu a resposta de todos os seus vizinhos, assim repassando “NÃO” para o nó G.

7. O nó G ainda não repassa a resposta “NAO” para J, pois ainda não recebeu a resposta de F (que pode ser um “SIM” ou outro “NÃO”).

8. O nó S faz as mesmas verificações que o nó I (que não é publicador e não tem vizinhos para enviar Q), repassando “NAO” para o nó F. Nesse momento, o nó C repassa Q para seus vizinhos B, A, D e E.

9. O nó A recebe Q primeiramente do nó C e depois recebe Q do nó B. Quando A recebe Q pela segunda vez (vindo do nó B), o nó A considera Q como uma resposta “NÃO”, pois significa que um vizinho seu também não é Publicador. O mesmo acontece com o nó B, que receber Q através do nó C, este responde “SIM” imediatamente para C. Quando B recebe Q pela segunda vez do nó A, o nó B não responde nada para C, pois já tinha respondido “SIM” anteriormente para C.

10. Assim como A e B, os nós D e E também recebem duas vezes a mesma mensagem Q, sendo a diferença que D e E respondem “NÃO” para o nó C, pois nenhum dos dois é um publicador.

11. Então o nó C repassa a resposta “SIM” para o nó F. Assim, F repassa “SIM” para o nó G, que responde “SIM” para J.

12. O nó J responde K que agora passa a ter mais um filho (que é o nó J). Dessa forma, a árvore de inscrição do nó K está completa.

4.4.2 Cancelamento de Inscrições

O principal componente do sistema que precisa ter conhecimento sobre o cancelamento de inscrições é o Publicador. Cada notificação gerada é repassada para os nós que estão na coluna 'próximo' da tabela de roteamento, até que todos os Assinantes que tenham inscrição registrada equivalente a notificação, recebam a mesma. Se um assinante registra um cancelamento de inscrição, o Publicador e os nós intermediários precisam ter o conhecimento para atualizar sua tabela de roteamento e para que notificações não cheguem aos Assinantes que não tenham mais interesse, ou seja, que tenham cancelado uma inscrição.

Para cancelar uma inscrição, o assinante x envia uma mensagem C para todos os seus vizinhos de sua árvore de inscrição $A_{\tau}(x)$ para que a mensagem chegue aos nós publicadores do tipo de evento τ.

A mensagem C é definida com sendo a seguinte tupla:

$C = <\{I_i(x)\}>$

onde $I_i(x)$ é a inscrição a ser cancelada.

No Cancelamento de Inscrições, a coluna ‘origem’ da tabela de roteamento é utilizada na rota para entrega de mensagens de cancelamento, onde a informação da coluna 'origem' indica para qual nó a mensagem deve ser repassada. Esse procedimento é o contrário da Notificação de Eventos, onde cada nó repassa a notificação para os nós que estão na coluna ‘próximo’.

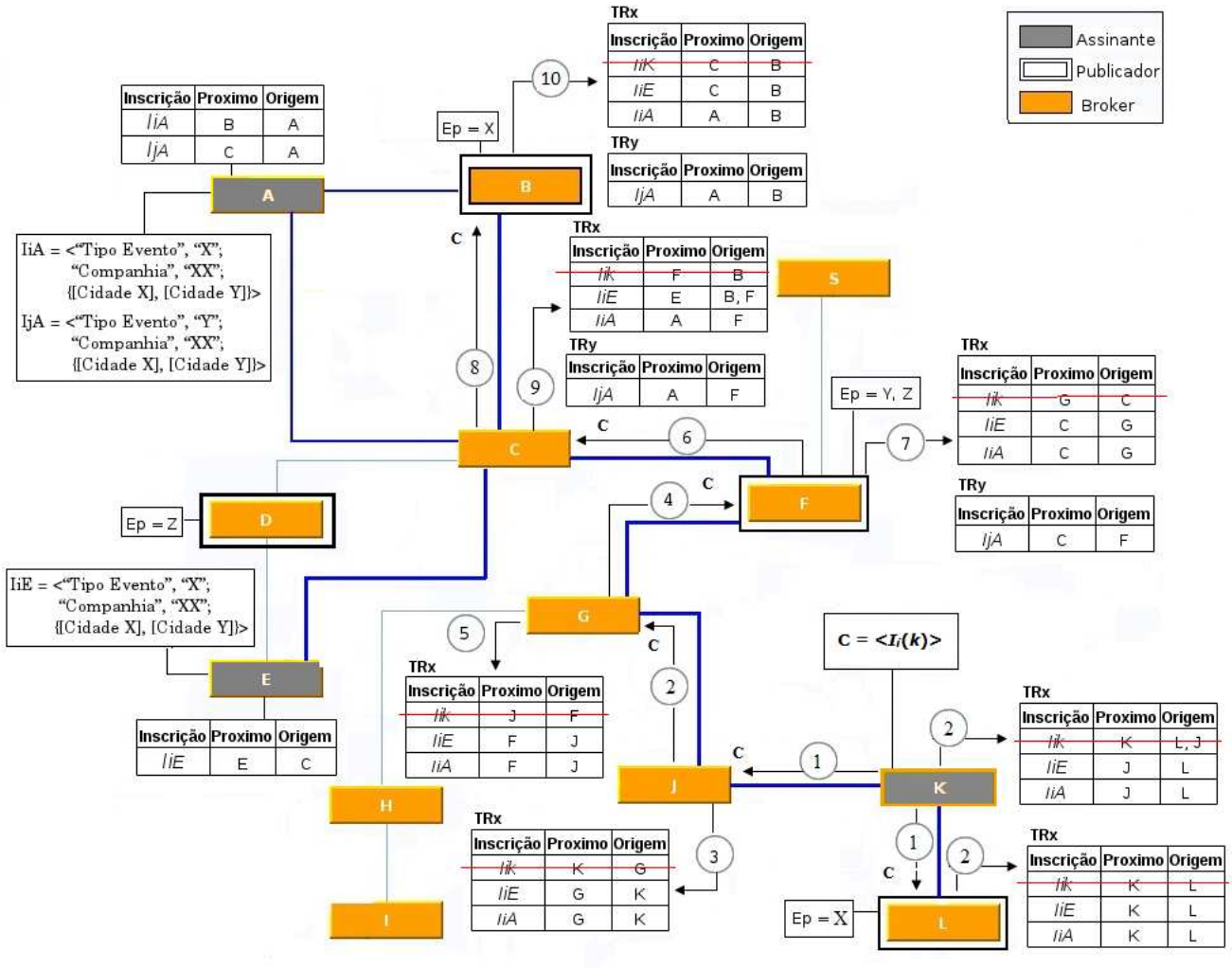

Figura 4.6: Cancelamento de Inscrições

A figura 4.6 mostra um exemplo de um cancelamento de inscrição. Cada número dentro de um círculo é a sequência que a mensagem *C* foi recebida/processada. Neste exemplo, o que ocorre é o seguinte:

1. O nó *K* registra o cancelamento da inscrição *Ii*(*x*), onde envia a mensagem *C* para seus vizinhos L e J, que são os nós que estão na coluna 'origem' referente ao registro da inscrição *Ii*(*k*).

2. O nó L verifica que na coluna 'origem' ele mesmo é o único nó, então L exclui o registro de sua tabela de roteamento. O nó J repassa a mensagem *C* para seu vizinho G e, após isso, também exclui o registro de sua tabela. Os mesmos passos são realizados pelos nós F e C.

3. Finalmente o nó B, que é o publicador interessado, recebe do nó C a mensagem *C*. Como B é o nó de origem de notificações, pois na coluna 'origem' consta ele mesmo, então B somente atualiza sua tabela de roteamento, não repassando a mensagem *C*.

Antes da formalização do algoritmo, algumas pressuposições precisam ser explicitadas:

Para registrar o cancelamento de uma inscrição, cada nó Assinante altera seu status *S* para *ASSINANTE.* Após isto, o nó irá fazer chamada para o procedimento ***Cancelar_Inscrição***.

Estados possíveis:

- Status: *S* = {ASSINANTE, CONCLUÍDO, OCIOSO}

S_{INIT} = {ASSINANTE, OCIOSO}
S_{TERM} = {CONCLUÍDO}

1. Inicialmente, um nó está no estado ASSINANTE e os demais estão com no estado OCIOSO.
2. No fim do algoritmo, todos os nós estarão com no estado CONCLUÍDO.

A seguir, o algoritmo é formalizado.

Início *Cancelar_Inscrição(x)*

ASSINANTE
Espontaneamente
Início
indice := Indice($TR_\tau(x)^{\text{Inscrição}} \Leftrightarrow$ *Inscrição*(*C*));
Enviar(*C*) para {$TR_\tau(x)$[indice]$^{\text{Origem}}$};
Remover_Registro($TR_\tau(x)$[indice]);
tornar_status (CONCLUÍDO);
Fim

OCIOSO

Receber(*C*)

Início

Nó_enviou_C = {origem};

indice := Indice((*TR*$_\tau$(*x*)$^{\text{Inscricao}}$ ⇔ *Inscrição*(*C*));

Se (|{*TR*$_\tau$[indice]$^{\text{Origem}}$}| > 1) então

Enviar(*C*) para {*TR*$_\tau$(*x*)[indice]$^{\text{Origem}}$} - Nó_enviou_C;

FimSe

Remover_Registro(*TR*$_\tau$(*x*)[indice]);

tornar_status (CONCLUÍDO);

Fim

Fim *Cancelar_Inscricao*

Cada nó ao receber a mensagem *C*, este executa o algoritmo ***Cancelar_Inscrição*** onde a execução é realizada da seguinte forma:

1. O nó recupera as linhas da tabela TR$_\tau$(*x*) em que um dos nós da lista da coluna 'origem', seja igual ao nó que enviou *C*. Após isso, é localizada somente a linha onde a inscrição seja igual à inscrição recebida em *C*.
2. Então, para cada nó da lista de nós da coluna 'origem', referente ao índice da tabela (recuperado no passo anterior), é verificado:

- Se a quantidade de nós da lista é igual a um, e se o nó *x* é igual ao nó da coluna origem, então o registro é removido da tabela TR$_\tau$(*x*), pois *x* é um Publicador, sendo ele mesmo o nó de 'origem';
- Se a quantidade de nós da lista é maior que um, então é removido somente o nó que enviou *C* da coluna 'próximo', referente à linha do índice recuperado de TR$_\tau$(*x*).

2. Se o status do nó *x* é CONCLUÍDO (entrou na condição se *x* é igual ao nó de 'origem'), a mensagem *C* não é mais repassada. Caso contrário, cada nó 'origem' continua repassando C até que o STATUS se torne CONCLUIDO.
3.

4.4.3 Notificações de Eventos

Na ocorrência de um evento $E_\tau(P_s)$, o publicador P_s só irá publicar a notificação $N(P_s)$ se existe pelo menos uma árvore $A_\tau(x)$, onde há pelo menos um registro em sua tabela $TR_\tau(x)$ de inscrições correspondentes à $E_\tau(P_s) \in N(P_s)$. Ou seja, se inscrições referentes ao tipo de evento τ não foram registradas para P_s, então o publicador não tem porquê publicar uma notificação.

Quando uma publicação é efetuada, o nó Publicador (bem como também todos os outros nós da rede) desempenha o papel de *Broker*, sendo responsável por enviar as notificações de eventos para os nós interessados. Cada nó, para repassar uma notificação $N(P_s)$, deve identificar a tabela de roteamento $TR_\tau(x)$ de inscrições registradas referentes à τ. Após isto, o nó deve identifica os registros da tabela que são referentes a inscrições que correspondam à $N(P_\tau)$. A partir disso, o trabalho do nó x é de repassar $N(P_s)$ para os nós distintos que estão na coluna 'próximo' de $TR_\tau(x)$. Para tal, definimos o conjunto R onde $R = R_\tau(x)$ o qual é composto por registros de $TR_\tau(x)$ onde as inscrições são correspondentes ao conteúdo da notificação.

- Designamos $R_\tau(x) = \{TR_\tau(x)^{\text{inscrição}} \leftrightarrow N(P_s)\}$

Sendo assim, todo nó x de uma árvore $A_\tau(x)$ deve receber notificações onde o conteúdo de $I_i(x)$ seja correspondente ao conteúdo da notificação. Por exemplo: se um publicador P_s emite uma notificação $N(P_s)$ e este está inserido em várias árvores de um tipo de evento τ, então todo nó assinante x que têm inscrição correspondente à notificação deve receber $N(P_s)$.

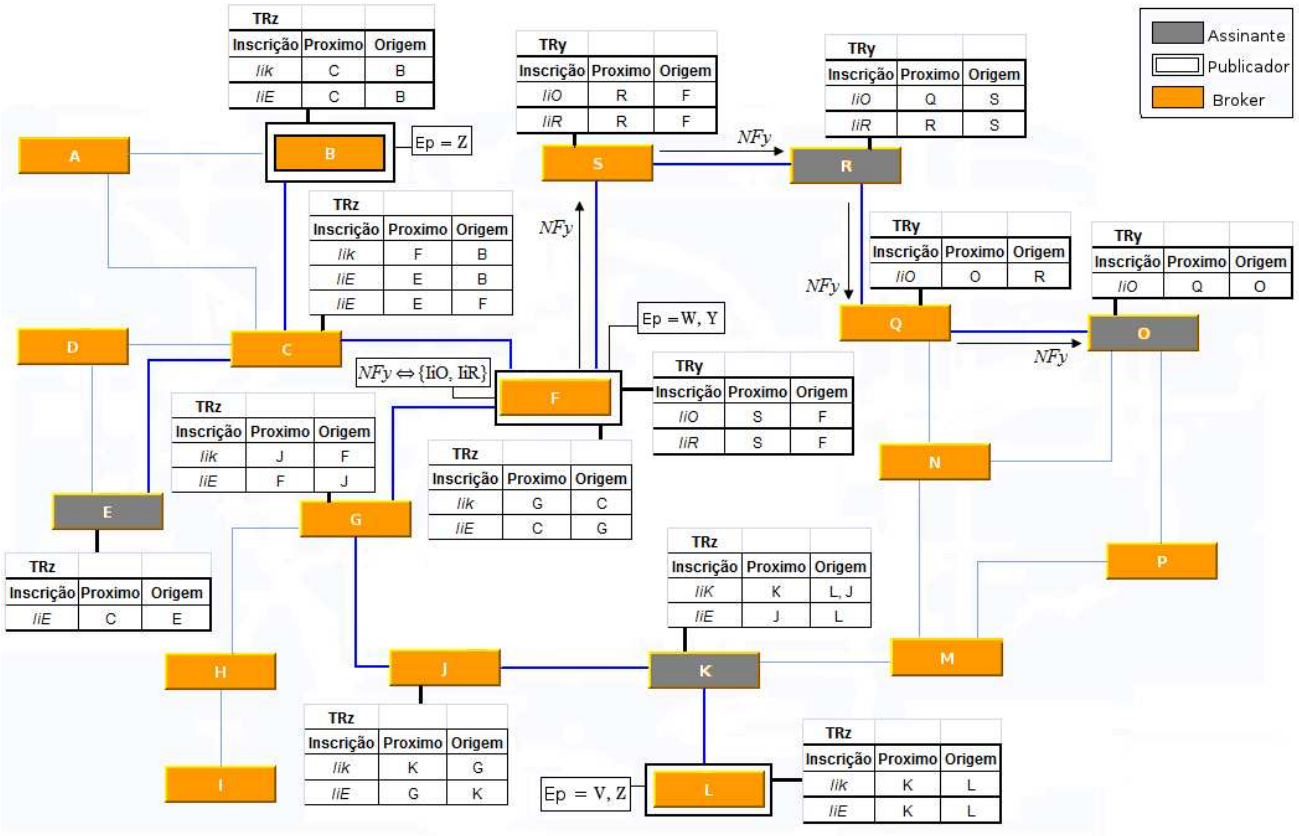

Figura 4.7: Disseminação da notificação N(P_y) para os nós interessados no evento E_y.

No exemplo da figura 4.7, uma notificação é emitida pelo nó F, a qual é correspondente a inscrição $I_i(o)$ e a inscrição $I_i(R)$.

Para repassar uma notificação recebida, todo nó faz a chamada do procedimento ***Notificar_Evento***. No momento da chamada do procedimento, o status do nó é alterado para ATIVO e cada nó ao receber $N(P_\tau)$ tem seu status alterado para OCIOSO.

- Status: S = {ATIVO, CONCLUÍDO, OCIOSO}

S_{INIT} = {ATIVO, OCIOSO}

S_{TERM} = {CONCLUÍDO}

A seguir, o algoritmo é formalizado.

Início *Notificar_Evento*

ATIVO

Espontaneamente

Início

$R_\tau(x) := \{TR_\tau(x)^{\text{Inscrição}} \leftrightarrow N(P_\tau)\}$;

Enviar($N(P_\tau)$) para $\{TR_\tau(x)^{\text{Próximo}} \in R_\tau(x)^{\text{Próximo}}\}$;

tornar_status(CONCLUÍDO);

Fim

OCIOSO

Receber($N(P_\tau)$)

Início

Nó_enviou_notificação := {origem};

$R_\tau(x) := \{TR_\tau(x)^{\text{Inscrição}} \leftrightarrow N(P_\tau)\}$ - Nó_enviou_notificação;

Enviar($N(P_\tau)$) para $\{\{TR_\tau(x)^{\text{Próximo}} \in R_\tau(x)^{\text{Próximo}}\}$;

tornar_status(CONCLUÍDO);

Fim

Fim *Enviar_Notificacao*

No exemplo da figura 4.7, a execução do algoritmo ***Enviar_Notificacao*** seria da seguinte forma:

1. O nó F publica a notificação $N(P_y)$ onde envia a notificação para todos os nós da coluna 'próximo' de $R_\tau(f)$, que é o nó S. A mensagem é repassada somente uma vez, pois F seleciona um nó distinto da coluna ‘próximo’;
2. O nó S recebe $N(P_y)$ e repassa para o nó R, que está na coluna ‘próximo’;
3. O nó R recebe $N(P_y)$ e repassa para a notificação para o nó Q. O nó R é o próximo nó da inscrição de *Ii(R)*, porém em sua tabela existe a inscrição I_i(o), onde o próximo nó é Q;

4. O nó Q recebe $N(P_y)$ onde $N(P_y)$ para o nó O que é o próximo nó;

O nó O recebe $N(P_y)$ onde para repassar $N(P_y)$ verifica que não existem mais registros de inscrições correspondentes à $N(P_y)$ que não seja a inscrição I_i(o) dele mesmo. Então $N(P_y)$ não é repassada para nenhum outro nó, tornando o status de O CONCLUÍDO, finalizando a execução do algoritmo.

4.4.4 Broker e Publicador: mudança de papel

Todo nó que é somente Broker pode assumir o papel de um nó Publicador, e todo nó que é Publicador pode deixar de ser um Publicador para ser um Broker. Outra possibilidade também é um Broker se tornar Assinante, o qual o procedimento é simples: o nó deve registrar uma inscrição, conforme visto na seção 4.2.1, onde é criada uma árvore de inscrição. No entanto, para a mudança de papel de um nó Publicador para Broker e vice-versa, os procedimentos são outros. Para registrar a mudança de papel, o nó deve enviar uma mensagem *M* para seus vizinhos, onde M_B é a mensagem que indica a mudança de papel do nó para Broker, e M_P indica a mudança de papel do nó para Publicador.

Na mudança de Publicador para Broker, uma questão levantada é a de Tolerância a Falta: *"E se existir somente um Publicador de E_τ e justamente esse Publicador mudar seu papel para Broker? Os nós que registraram inscrições de E_τ ficarão sem Publicador?"*

Para tal questão, assumimos o seguinte: como o objetivo deste trabalho é o roteamento de mensagens, não vamos abordar a tolerância a falta em cada mudança de papel. O que definimos são algoritmos para o encaminhamento e manutenção de cada árvore de abrangência, considerando que exista mais de um Publicador de determinado evento E_τ ou que cada Assinante crie uma árvore de inscrição na operação de um novo nó Publicador no grafo.

Como sugestão para trabalhos futuros, as questões de tolerância a falta de nós Publicadores, são apresentadas no capítulo 7. A seguir são apresentados os procedimentos para mudança de papel: Broker para Publicador e Publicador para Broker.

4.4.4.1 Broker para Publicador

Todo Publicador P_s tem conhecimento das inscrições de nós assinantes *x* onde o tipo de evento das inscrições é um subconjunto de *s*. Da mesma forma, cada novo Publicador P_s que entrar na rede, deve ter conhecimento de inscrições já registradas referentes a $E_\tau\{Ps\}$, para que cada $x_\tau \epsilon s$ seja incluído na abrangência do novo publicador P_s. Cada novo Publicador deve conhecer o 'próximo' nó, o qual tem conexões com nós assinantes de eventos $E_\tau(P_s)$, para que a notificação seja entregue e para que o P_s tenha em sua tabela de roteamento a inscrição $I_i(x)$ equivalente ao "*template*" da notificações $N(P_s)$ emitidas. Para tal, a idéia é fazer com que os novos nós Publicadores sejam incluídos em todas as árvores de inscrições de $E_\tau\{Ps\}$ já existentes. Para registrar a mudança de papel para Publicador, o Broker envia uma mensagem M_P.

A mensagem M_P é definida com sendo a seguinte tupla:

$M_P = <\{E_t\}>$

onde E_t é tipo de Evento E_t que o nó passará a emitir.

A figura 4.8 mostra um cenário de mudança de papel onde o nó I envia uma mensagem M_P para mudar seu papel de *Broker* para Publicador.

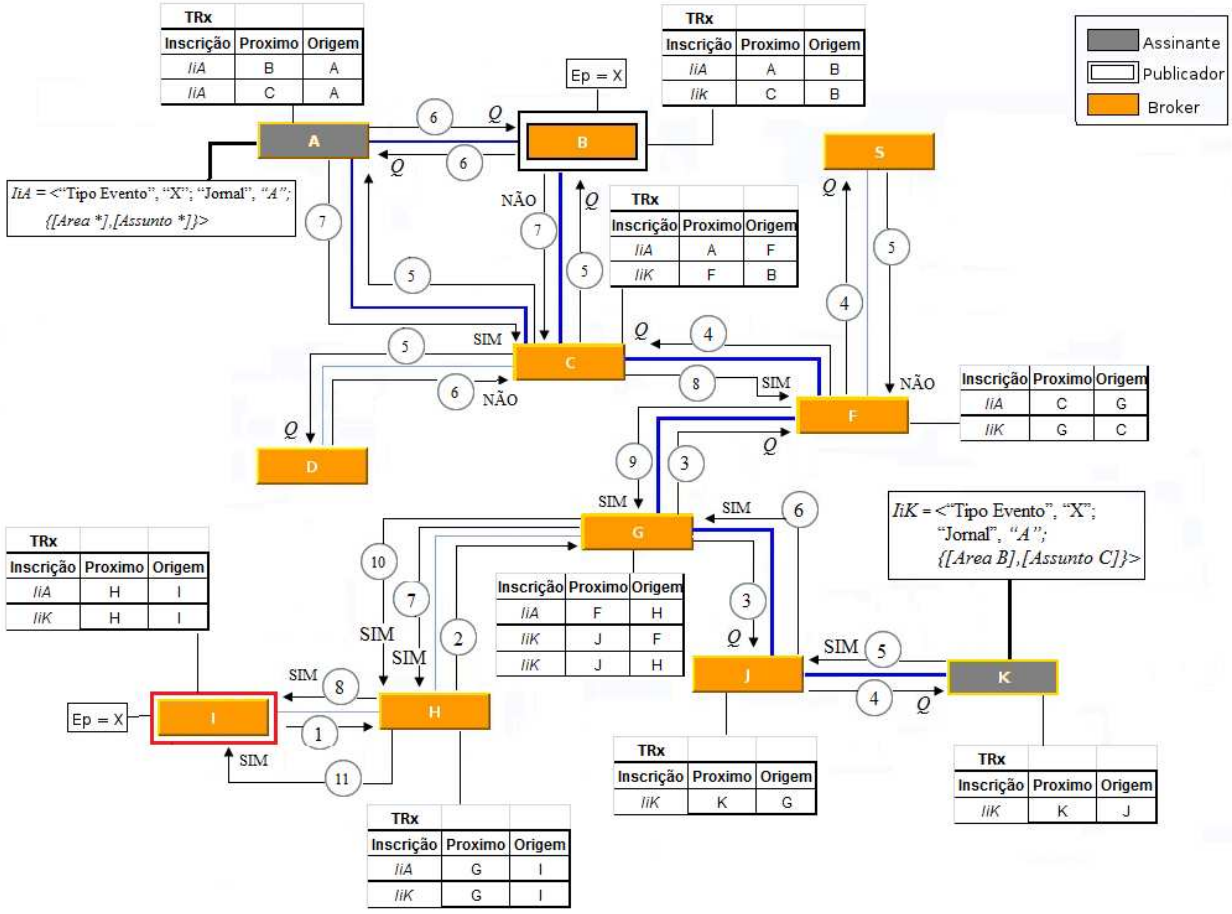

Figura 4.8: Registro de mudança de papel – Broker para Publicador.

Para registrar seu novo papel como Publicador, o nó Broker envia uma mensagem M_P com a seguinte pergunta: "é um Assinante do mesmo tipo de evento E_τ que estou publicando?". Cada nó ao receber a mensagem, repassa e responde M_P conforme os passos abaixo.

Se for a primeira vez que o nó *x* está recebendo M_P de um vizinho *p*, este verifica o seguinte:

1. Se o nó é Assinante onde o tipo de evento τ de interesse é igual ao tipo Eτ(P_s) então:

- Este responde "SIM" imediatamente para P_τ e continua repassando M_P, pois podem existir outros assinantes que devem ter abrangência do novo publicador.

2. Se o nó é um Broker, Publicador ou Assinante onde o tipo de evento τ é diferente de $E_\tau(P_s)$ então:

Este repassa M_P para seus vizinhos, onde fica aguardando a resposta de todos os vizinhos.

- Se o nó recebe uma resposta "SIM", então repassa "SIM" imediatamente para P_τ.

- Se o nó recebe um “NÃO”, então este só enviará “NÃO” para P_τ após receber a resposta de M_P de todos os seus vizinhos ou também se ocorrer o seguinte: se não for a primeira vez que o nó está recebendo M_P, então o nó faz soma a contagem de vezes que recebeu M_P. Se o número da contagem for igual ao número de vizinhos, então o nó considera a resposta um “NÃO” e repassando a resposta “NAO” para P_τ.

A atualização da tabela de roteamento $TR_\tau(x)$ é realizada no envio M_P e no recebimento da última resposta de M_P. Se não existir um registro em $TR_\tau(x)$ onde na lista de nós da coluna ‘origem’ não tenha o nó que enviou a resposta, então um novo registro é incluído na tabela $TR_\tau(x)$. Caso contrário, as colunas 'origem' e 'próximo' são atualizadas somente. Em ambos os casos, a tabela é atualizada conforme o seguinte:

Coluna 'origem'

- Se o nó somente enviou um “SIM” e não recebeu outro “SIM” como resposta de seus vizinhos (é um publicador *folha*), estão este armazena seu próprio identificador na coluna 'origem'
- Se o nó enviou “SIM” e também recebeu um “SIM” de seus vizinhos, então o identificador dos nós de quem recebeu a resposta “SIM” é atualizado na coluna 'origem' de $TR_\tau(x)$.

Coluna 'próximo'

- Se o nó recebeu um “SIM” e tem um vizinho para repassar a resposta, então a coluna 'próximo' recebe o identificador do nó para quem está repassando a resposta.
- Se o nó recebeu um “SIM”, mas não tem um nó vizinho para enviar a resposta, ou seja, se o nó é o Assinante que disparou a mensagem M_P pela primeira vez, então a coluna 'próximo' recebe o identificador do próprio nó.

Assim, um nó considera o nó de quem está recebendo a resposta como sendo um nó que é ‘origem’ de notificações, e o nó para quem está repassando a resposta de M_P (que é o nó de quem recebeu M_P) como sendo o ‘próximo’ nó para quem deve repassar mensagens.
Para repassar uma mensagem M_P para mudança de papel de Broker para Publicador, o nó Broker faz a chamada do procedimento ***Mudar_Papel_Para_Publicador***.

Os possíveis estados de cada nó são os seguintes:

- Status: *S* = {ATIVO, CONCLUÍDO, OCIOSO, PUBLICADOR}

S_{INIT} = {PUBLICADOR, OCIOSO}

S_{TERM} = {CONCLUÍDO}

1. Todo nó é iniciado com o *status* OCIOSO.

2. Na chamada do procedimento *Mudar_Papel_Para_Publicador*, o status do nó é alterado para PUBLICADOR.

3. Cada nó ao repassar M_P para todos os seus vizinhos, este tem seu status alterado para ATIVO.

4. Após o nó receber a resposta de M_P de todos os seus vizinhos, seu status é alterado para CONCLUÍDO.

A seguir o algoritmo é detalhado, juntamente com a seqüência de execução para o exemplo da figura 4.8.

Início *Mudar_Papel_Para_Publicador(x)*

```
PUBLICADOR
Espontaneamente
Início
    contador := 0;
    Enviar(M_P) {N(x)};
    tornar_status (ATIVO);
Fim

ATIVO
Receber(RESPOSTA)
Início
    recebeu_sim := recebeu_sim + 1;
    recebeu_resposta := recebeu_resposta + 1;
```

Se (*RESPOSTA* = “SIM”) então
Recebeu_sim := recebeu_sim + 1;
FimSe

Se (recebeu_sim > 0) então
Se (|N(*x*) - *NoQueEnviou*(*RESPOSTA*)| ≡ recebeu_sim) então
TR(*x*) ^{Proximo} := *x*;
FimSe
FimSe
tornar_staus (CONCLUIDO);
Fim

OCIOSO
Receber(M_P)
Início
Recebeu_*Q* := recebeu_Q + 1;
Se (*x* ∈ {*A*}) então
Se ($\forall \tau \in x_\tau \mid \tau \in E_\tau \leftrightarrow (\tau \in M_P)$) então
Enviar(*SIM*) para *NoQueEnviou*(M_P);
$TR_\tau(x)^{origem}$:= *NoQueEnviou*(M_P);
FimSe
Senão
Se (|N(*x*) - *NoQueEnviou*(M_P)| > recebeu_M_P)então
Para cada *vizinho* ∈ N(x) - {*NoQueEnviou*(M_P)} faça
Enviar(M_P) para *vizinho*;
Senão
Enviar(*NÃO*) para *NoQueEnviou* (M_P);
FimSe
FimSe
Tornar_status (CONCLUIDO);
Fim

Fim *Mudar_Papel_Para_Publicador*

Conforme exemplo da figura 4.8, a execução do algoritmo, no registro de mudança de papel do nó I, é a seguinte:

1. O nó I altera seus status para PUBLICADOR e envia a mensagem M_P para todos os seus vizinhos, que nesse caso é o nó H;
2. O nó H tem seu status OCIOSO quando recebe M_P, onde envia a mensagem M_P para todos os seus vizinhos, que nesse caso é o nó G;
3. O nó G que também está com status OCIOSO, repassa M_P para seus vizinhos F e J;
4. O nó J repassa a mensagem M_P para K; O nó F repassa a mensagem M_P para C e S.
4. O nó F repassa M_P para C e S.

5. O nó K recebendo M_P, este verifica que é um assinante do mesmo tipo de evento recebido em M_P. Então K responde “SIM” para J e não repassa M_P para nenhum nó, pois K é 'próximo' dele mesmo por se um nó assinante. Assim, K fica com seu status CONCLUIDO.
5. O nó S verifica que não é um assinante e que também não tem mais vizinhos para repassar M_P. Então S responde "NÃO" para F.
5. O nó C repassa a mensagem M_P para os nós D, B e A.

6. O nó J responde “SIM” para G.
6. O nó B repassa M_P para seu vizinho A e o nó A repassa M_P para seu vizinho B.

7. O nó G responde "SIM" para H e inclui um novo registro em sua tabela de roteamento, onde J na coluna 'próximo' e H como 'origem'.
7. Os nós A e B verificam que receberam M_P duas vezes. Sendo assim, estes só respondem M_P para o nó de quem receberam M_P pela primeira vez. Então o nó B responde "NÃO" para C e o nó A responde "SIM" para C, pois A é um assinante do mesmo tipo de evento recebido em M_P.

8. O nó H responde “SIM” para o nó I e inclui um registro em sua tabela de roteamento onde G é o 'próximo' nó e o nó I é o nó de 'origem'.
8. O nó C responde "SIM" para F e atualiza sua tabela de roteamento incluindo o 'próximo' sendo A e 'origem' sendo F.

9. O nó F responde "SIM" para G e atualiza sua tabela sendo C o 'próximo' nó e G sendo 'origem'.
10. G responde "SIM" para H e atualiza sua tabela sendo F o 'próximo' nó e H um nó de 'origem'
11. O nó H responde "SIM" para I e atualiza sua tabela sendo o 'próximo' nó G e o I sendo o nó de 'origem'. Assim o algoritmo é concluído.

4.4.4.2 Publicador para Broker

Para um Assinante a mudança de papel de um nó Publicador para *Broker* significa a saída de uma ‘origem’ de notificação de eventos de sua árvore de inscrição. Na mudança de papel, cada Assinante que tenha inscrições registradas de E_τ deve ter conhecimento da saída do publicador P_s para assim atualizar sua tabela de roteamento $TR_\tau(x)$ excluindo os registros ou atualizando a coluna ‘origem’ referentes a P_s. Para registrar seu novo papel como Broker, o nó Publicador envia uma mensagem M_B informando sua mudança de papel, a qual deve ser entregue para o um outro publicador de E_τ (caso exista) ou para todos os assinantes onde $x_\tau \in \{P_s\}$. Caso exista mais de um publicador do tipo de evento do publicador P_s, então o publicador que recebe a mensagem M_B não repassa a mesma para seus vizinhos, atualizando somente sua tabela de roteamento, pois para os demais nós conectados a ele não fará diferença se um publicador anterior mudou de papel. Para tal, a mensagem deve ser propagada nas árvores $A_\tau(x)$ onde o tipo de evento τ seja igual ao tipo de evento do publicar P_s que registra a mudança de papel.

A mensagem M_B é definida com sendo a seguinte tupla:

$M_B = <\{Et\}>$
onde E_t é tipo de Evento que o nó deixará de emitir.

Para que um nó repasse M_B para todos os assinantes de todas as árvores de inscrições, este deve identificar os nós para quem deve repassar M_B, em sua tabela de roteamento. Para

tal, definimos o conjunto R_A onde $R_A = R_\tau(x)$ o qual é composto por registros de TR_τ onde as inscrições são correspondentes ao tipo de evento de P_s.

- Designamos $R_A(\tau) = \{TR_\tau^{\text{inscrição}} \leftrightarrow \tau \equiv (s \subseteq P_s)\}$

Todo nó que recebe a mensagem M_B verifica o seguinte para repassar a mensagem:

1. Verifica os registros do conjunto R_A, onde o tipo de evento $\tau \in P_s$ (se existem outros nós que recebem o mesmo tipo de evento). Então o nó repassa M_B para os nós da coluna 'próximo' de cada registro da tabela TR_τ. O nó só não irá repassar M_B se ele mesmo for o próximo nó para quem deve ser enviado, ou seja, se o nó for um assinante e se não existem conexões com outros nós que tenham inscrições do mesmo tipo de evento. Para atualiza a tabela de roteamento $TR_\tau(x)$, todo nó realiza as seguintes verificações:

1. Se na lista de nós de 'origem' de sua tabela de roteamento existir mais de um nó origem referente a inscrições do tipo de evento informado em M_B, então o nó retira o publicador da lista de 'origem'.
2. Caso contrário, se o Publicador de τ é o único nó de 'origem' na lista de $TR_\tau(x)$, então o nó exclui o registro da tabela onde as inscrições sejam correspondentes ao tipo de evento τ informado na mensagem M_B.

Como exemplo, a figura 4.9 mostra um cenário de mudança de papel onde o publicador B registra a mensagem M_B para mudar seu papel para Broker. A seguir o algoritmo é detalhado.

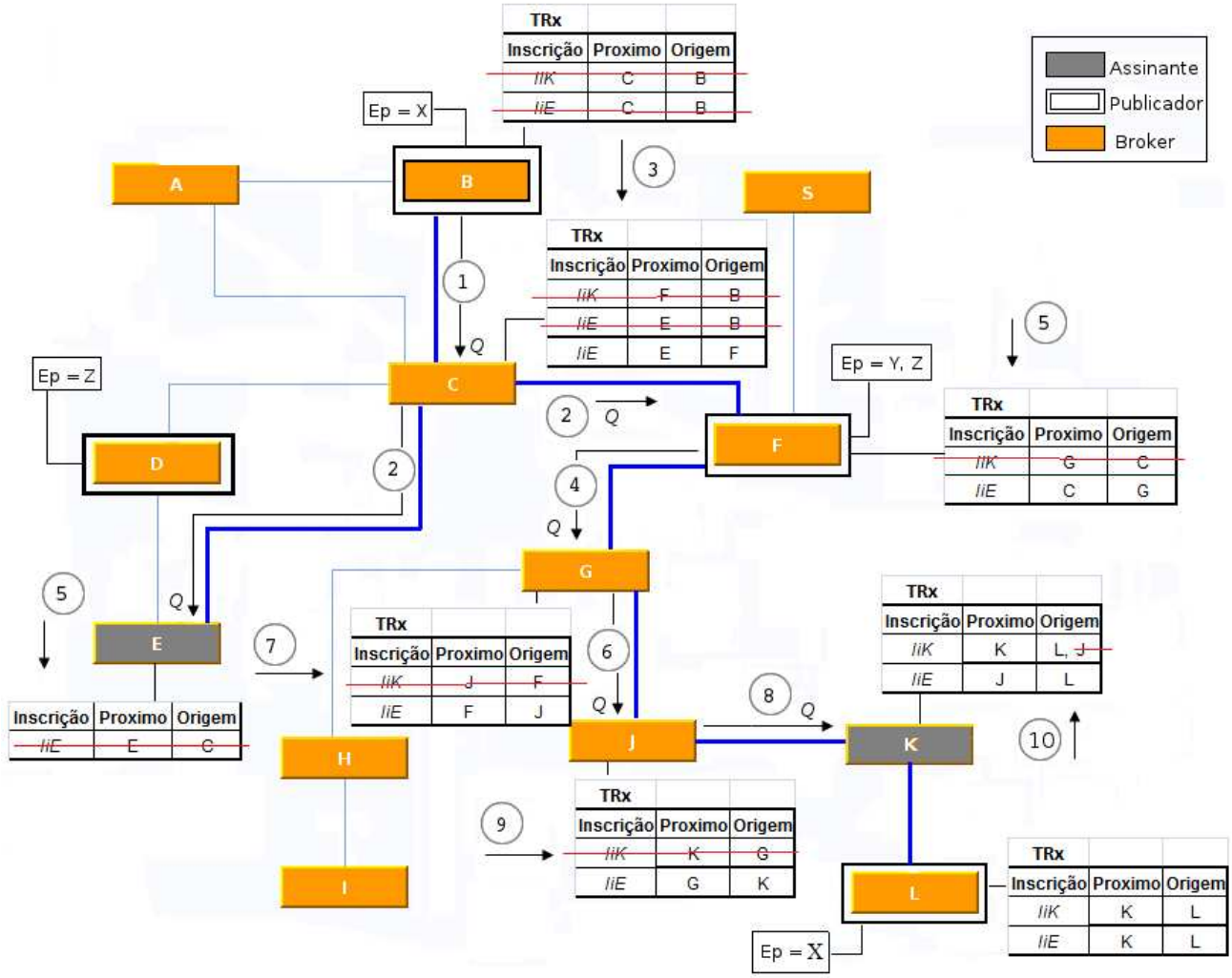

Figura 4.9: Mudança de papel: Publicador para Broker

O procedimento ***Mudar_Papel_Para_Broker*** é chamado e executado pelo nó publicador que esta registrando a mensagem M_B para a mudança de papel de Publicador para Broker. Cada nó que recebe a mensagem M_B também executa o mesmo procedimento para repassar e responder M_B.

Na chamada e execução do algoritmo, os possíveis estados de cada nó são os seguintes:

- Status: S = {ATIVO, BROKER, CONCLUÍDO, OCIOSO}

S_{INIT} = {BROKER, OCIOSO}

S_{TERM} = {CONCLUÍDO}

1. Todo nó é iniciado com o status OCIOSO.
2. Na chamada do procedimento *Mudar_Papel_Para_Broker*, o status do nó é alterado para BROKER.
3. Cada nó ao repassar M_B para seus vizinhos, este tem seu status alterado para ATIVO.
4. Após o nó receber a resposta de M_B de todos os seus vizinhos, seu status é alterado para CONCLUÍDO.

Segue a formalização do algoritmo.

Início *Mudar_Papel_Para_Broker (x)*

```
BROKER
Espontaneamente
Início
    Repassar(M_B) para {N(x)};
    tornar_status (ATIVO);
Fim

ATIVO
Receber(RESPOSTA)
Início
    Recebeu_Resposta := recebeu_ Resposta + 1;

    Se (recebeu_ Resposta = 1) então

        Se (x ∈ {P_τ}) então
            Se (e(x) ↔ e(M_B)}) então
                TR_τ(x)^origem := TR_τ(x)^origem – {Identificador_No_Recebeu(M_B)}
                tornar_status (CONCLUIDO);
            Senão
                Repassar(M_B);
                Atualizar_Roteamento(M_B);
                tornar_status (CONCLUIDO);
```

```
            FimSe
        Senão
            Repassar(M_B);
            Atualizar_Roteamento(M_B);
            tornar_status (CONCLUIDO);
        FimSe
    FimSe

Fim

Repassar(M_B)
Início
    Para cada proximo ∈ {Inscricao(TR_τ(x)) ⊂ e(M_B)} faça
        Repassar(M_B) para vizinho;
    FimPara
    tornar_status (ATIVO);
Fim

Atualizar_Roteamento(M_B)
Início
    Se (|{TR_τ(x)^origem ∈ e(M_B)}| > 1) então
        TR_τ(x)^origem := TR_τ(x)^origem – {Identificador_No(M_B)}
    Senão
        TR_τ(x) := TR_τ(x)  - {TR_τ(x) ⊂ e(M_B)};
    FimSe
Fim
```

Fim *Mudar_Papel_Para_Broker*

Como exemplo da figura 4.9, a execução do algoritmo ***Mudar_Papel_Para_Broker*** é a seguinte:

Para tal, o nó B consulta em sua tabela de roteamento os registros referentes à *s*, que é um conjunto de tipos de evento, sendo S = {τ_1, τ_2, ..., τ_n}. Os registros encontrados são armazenados no conjunto R_A. Assim, a seqüência de passos é a seguinte:

1. O nó *B* envia a mensagem M_B para os nós que estão na lista de 'próximo' referente $E(P_\tau)$. Assim, a mensagem é enviada para o nó *C*;
2. O nó *C* ao receber a mensagem M_B *e* verifica os próximos nós para quem deve repassar a mensagem recebida. Então o nó *C* repassa M_B para os nós E e F. Após isto, *C* atualiza sua tabela de roteamento excluindo os registros da tabela referentes ao tipo de evento τ = x.
3. O nós E ao receber M_B, verifica que não tem vizinhos para repassar M_B. Então E somente atualiza sua tabela de roteamento, tornando seu status como CONCLUÍDO.
4. O nó F repassa M_B para o nó G. Após isto atualiza sua tabela de roteamento excluindo o registro referente à $\tau = x$. Os nós G e J realizam os mesmos passos que o nó F executou. Finalmente quando a mensagem chega ao nó K, que é o um nó assinante, este verifica que ele mesmo é o 'próximo'. Então K não repassa a mensagem M_B, onde somente atualiza sua tabela de roteamento excluindo da lista de nós da coluna ‘origem’ o nó que lhe enviou M_B, que é o nó J.

4.5 Conclusão

A operação de Mudança de Papel Broker - Publicador tem a mesma finalidade da operação de Criação de Inscrição, onde o objetivo é fazer com que Assinantes tenham abrangência do novo Publicador. A diferença é que na mudança de papel, o novo Publicador é quem dissemina o tipo de evento, para que assim todos os assinantes do mesmo tipo de evento incluam o novo publicador em suas árvores de inscrições. Os custos de disseminação de mensagens nessas duas operações são compensados na Notificação de Eventos, Cancelamento de Inscrições e Mudança de Papel – Publicador para Broker, pois com árvore de inscrição formada, toda mensagem é propagada dentro da respectiva árvore.

Na notificação de Eventos, um Publicador emite uma notificação se existem Assinantes que tenham interesse no tipo de Evento a ser publicado. Sugerimos que seja incluído um *buffer* onde tais notificações sejam armazenas e enviadas a cada nova publicação. Para tal, é

necessário definir as necessidades do sistema, como temporalidade e periodicidade por exemplo. Com tais definições, os procedimentos para adotar o buffer de notificações podem ser estabelecidos em nossa proposta. Outro ponto para ser abordado é a mudanças de Assinante para Broker e Assinante para Publicador. Não iremos realizar tal abordagem, ficando essas operações para continuidade deste trabalho.

Capítulo 5

ANÁLISE

Conforme os algoritmos apresentados como proposta deste trabalho, buscamos a otimização no fluxo de troca de mensagens na realização de operações em DEBS. Assim, neste capítulo vamos detalhar análises realizadas das operações de Criação e Cancelamento de Inscrições, Notificações de Eventos e Mudança de papel *Broker*/Publicador.

5.1 Criação da Árvore de Inscrição - CAI

A construção de uma árvore de inscrição $A_\tau(x)$ é a função mais importante do sistema, pois todas as demais operações realizadas na rede são realizadas sobre $A_\tau(x)$. Para toda inscrição registrada, uma árvore de inscrição é construída, ou seja, conforme o número de inscrições registradas cresce o fluxo de troca de mensagens na rede "*overlay*" para a formação de cada $A_\tau(x)$. Outro ponto a ser levado em consideração, é que na construção de toda $A_\tau(x)$, ocorre *flooding* para a disseminação da mensagem Q, pois todos os nós Publicadores do grafo devem receber Q. Ou seja, o número de mensagens propagadas varia de acordo com o tamanho do grafo, e o tempo para que um Assinante *x* receba uma resposta de um Publicador variam de acordo com a distância que *x* está de um nó Publicador. Para estimar o custo e tempo na propagação de mensagens para a construção de cada $A_\tau(x)$, definimos o seguinte:

- O número de mensagens propagadas pelo nó *x*, sendo M[*CAI*], onde M representa o número de mensagens para a operação *CAI*.

- Sendo $G = (V, E)$ o grafo da topologia de nós, temos a representação do número de vértices e arestas:

$$m = |E|$$
$$n = |V|$$

- Definimos o tempo para propagação de mensagens no grafo G sendo T[*CAI*] onde T representa a variável de tempo sobre *CAI*.

A quantidade de mensagens trocadas entre os nós do grafo, para a criação de $A_{\tau}(x)$ é a seguinte:

M[*CAI*] = 2*m*, pois em cada aresta haverá duas mensagens (Q, Q_{SIM}, $Q_{NÃO}$, Q_Q).

Em questão de tempo T que um assinante *x* encontra todos os publicadores, temos variação de acordo com a distância entre publicador e assinante. A seguir, temos a avaliação do tempo de chegada de resposta de um publicador P_s para um assinante *x*.

(1)
Pode ser considerado o melhor caso onde todos os publicadores são vizinhos diretos de *x*. Este é o caso se G é um grafo completo. Nesta condição, T = 2.

Na figura 5.1, temos um exemplo de um nó Assinante *k* que registra uma inscrição onde o único publicador do tipo de evento τ onde *k* tem interesse, é o nó vizinho *l*.

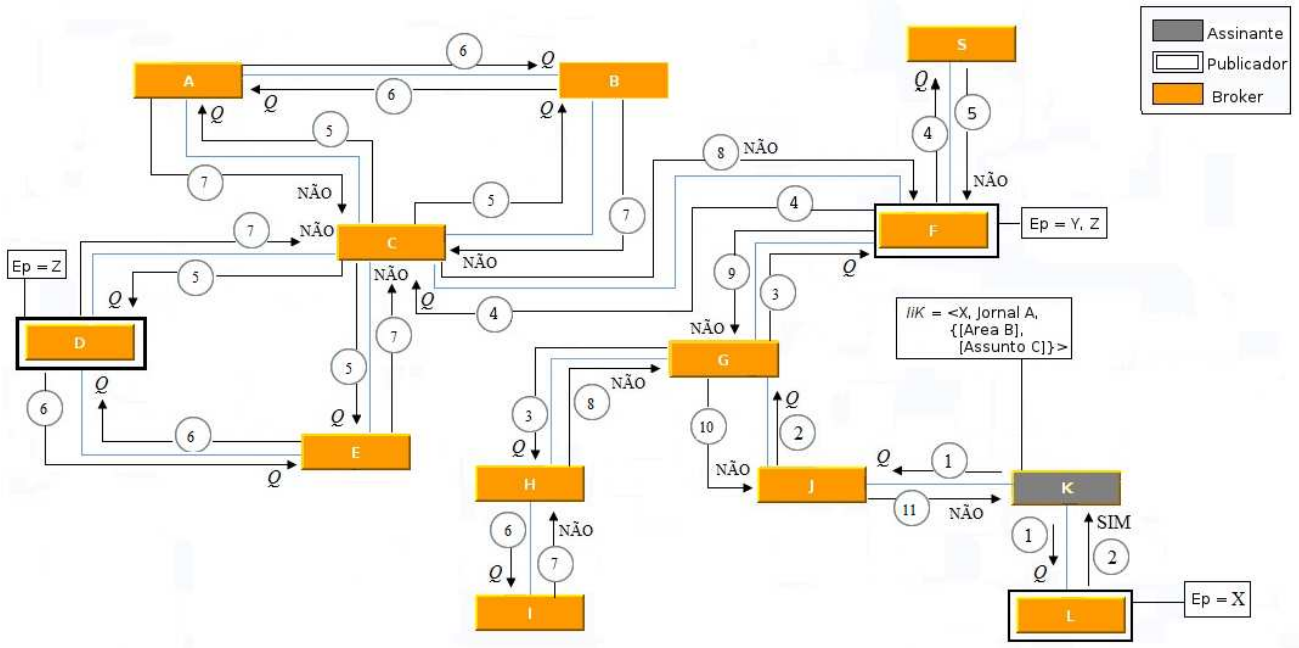

Figura 5.1: *CAI* - abrangência de um publicador vizinho.

- No caso geral, porém, o tempo de propagação da mensagem Q para a criação da árvore de inscrição (CAI) é o seguinte:

$$T[CAI] \leq 2.d(G) \leq 2n-2$$

No exemplo da figura 5.1 temos M[*CAI*] = 26 e T[*CAI*] = 11.

5.2 Cancelamento de Inscrição - CI

No cancelamento de uma inscrição $I_i(x)$, todo nó publicador P_s da árvore $A_\tau(x)$, sendo τ pertencente ao conjunto S, sendo equivalente a τ de $I_i(x)$, deve receber a mensagem de cancelamento. Sendo $A_\tau(x) = (V', E')$ e sendo $n' = |V'|$, $m' = |E'|$, temos os seguintes custos:

- M[*CI*] = m', pois todos os vértices de $A_\tau(x)$ recebem a mensagem.
- T[*CI*] $\leq$ n' - 1, pois como $A_\tau(x)$ é uma árvore, esta é a maior profundidade possível com n' nós.

No exemplo da figura 5.2, temos o nó K que registra o cancelamento de inscrição $I_i(k)$. A mensagem deve ser repassada para todos os nós de $A_\tau(K)$ até que a mensagem C chegue aos nós publicadores P_τ de $A_\tau(K)$ que são os destinatários finais da mensagem C de cancelamento.

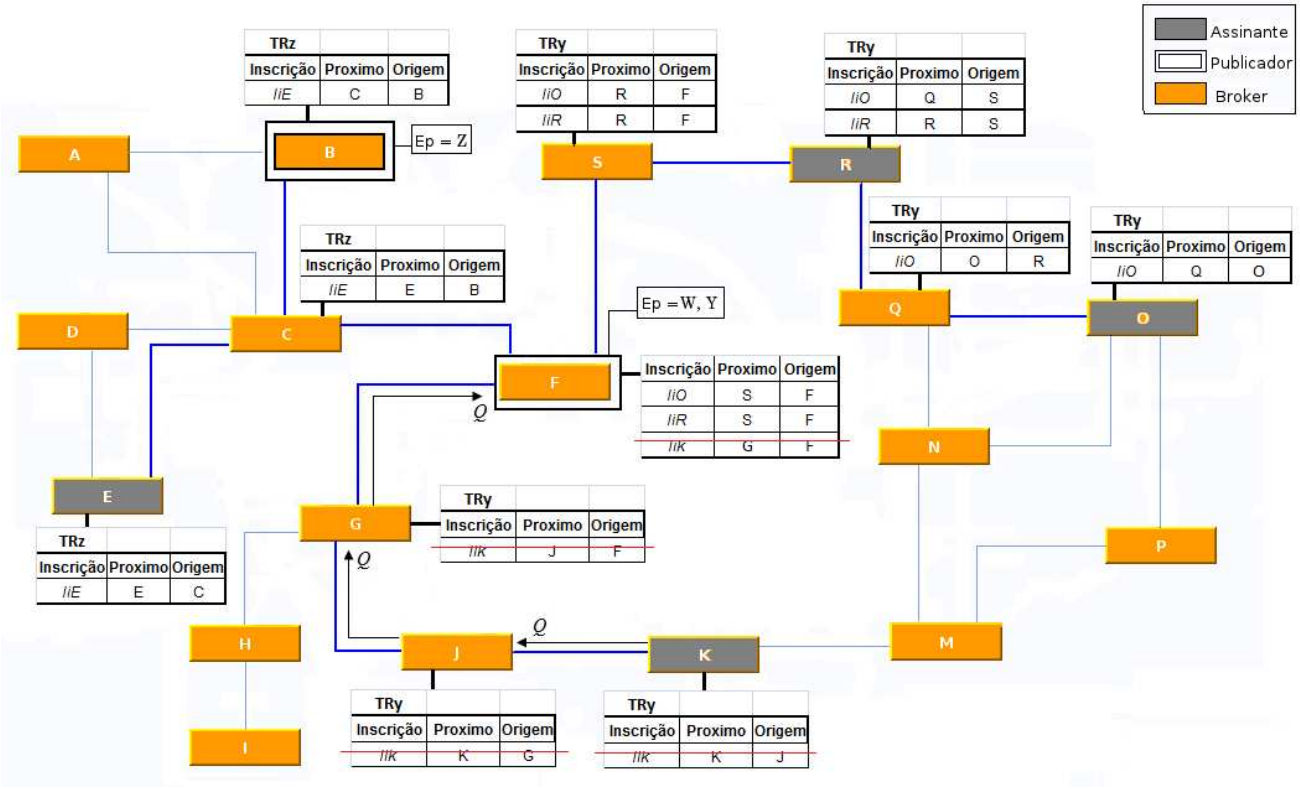

Figura 5.2: Custo de cancelamento de inscrição.

Portanto, no exemplo da figura 5.2, a mensagem C é propagada para os nós J e G, para que assim chegue ao nó F. O custo de M é M[CI] = 3 e o custo de tempo para C chegar ao nó F é T = 3.

5.3 Notificação de Eventos - NE

Toda notificação de eventos é encaminhada para os nós interessados dentro da árvore de inscrição de um assinante x. Como cada árvore de inscrição se refere a uma inscrição distinta de cada assinante, onde o nó assinante x a raiz de $A_\tau(x)$. Sendo S o conjunto de todos os assinantes de eventos de um tipo $\tau \in N(P_s)$, temos o conjunto de árvores de inscrição desses

assinantes, onde A = $\{A_\tau (x), \forall x \in S\}$. Portanto A é o conjunto de todas as árvores de inscrição de tipo de evento τ.

Sendo assim, na publicação de um evento E_τ, a notificação de E_τ é disseminada somente dentro de árvores $A_\tau(x)$ onde referentes ao tipo de evento τ. Portanto, o custo de propagação de mensagens para a notificação $N(P_\tau)$ é variável, de acordo com o número de árvores em que o publicador P_s está inserido.

- Definimos A sendo o conjunto da somatória de todas as arestas de árvores A do grafo:

$$A = \Sigma_{a \epsilon A} |E^{(a)}|$$

- Definimos a maior profunidade de uma árvore A sendo:

$$Max\{d(a) : a \epsilon A\} \leq n - 1$$

A disseminação de uma notificação $N(p_\tau)$ é realizada sobre o conjunto A, onde temos os seguintes custos de mensagem e tempo:

- $M[N(P_\tau)] \leq |E(A)| - 1$, ou seja, M é menor ou igual ao tamanho de todas as arestas de A, exceto o nó raiz de $A_\tau(x)$.
- $T[N(P_\tau)]$ é menor ou igual a *Max*.

O pior caso é quando as árvores do grafo não têm sobreposição. No exemplo da figura 5.4 temos duas árvores sem sobreposição: $A_\tau(K)$ e $A_\tau(R)$. O publicador F emitindo uma notificação, a mensagem é disparada para sentidos diferentes, sem aproveitamento de caminho que temos em $A_\tau(O)$, onde a árvore de O é sobreposta de $A_\tau(R)$.

Ainda na figura 5.3, temos três árvores de inscrição: $A_y(O)$, $A_y(R)$ e $A_y(K)$. Neste cenário, o publicador F emitiu uma notificação NF_y e todo nó raiz de cada árvore de inscrição, deve receber NF_y.

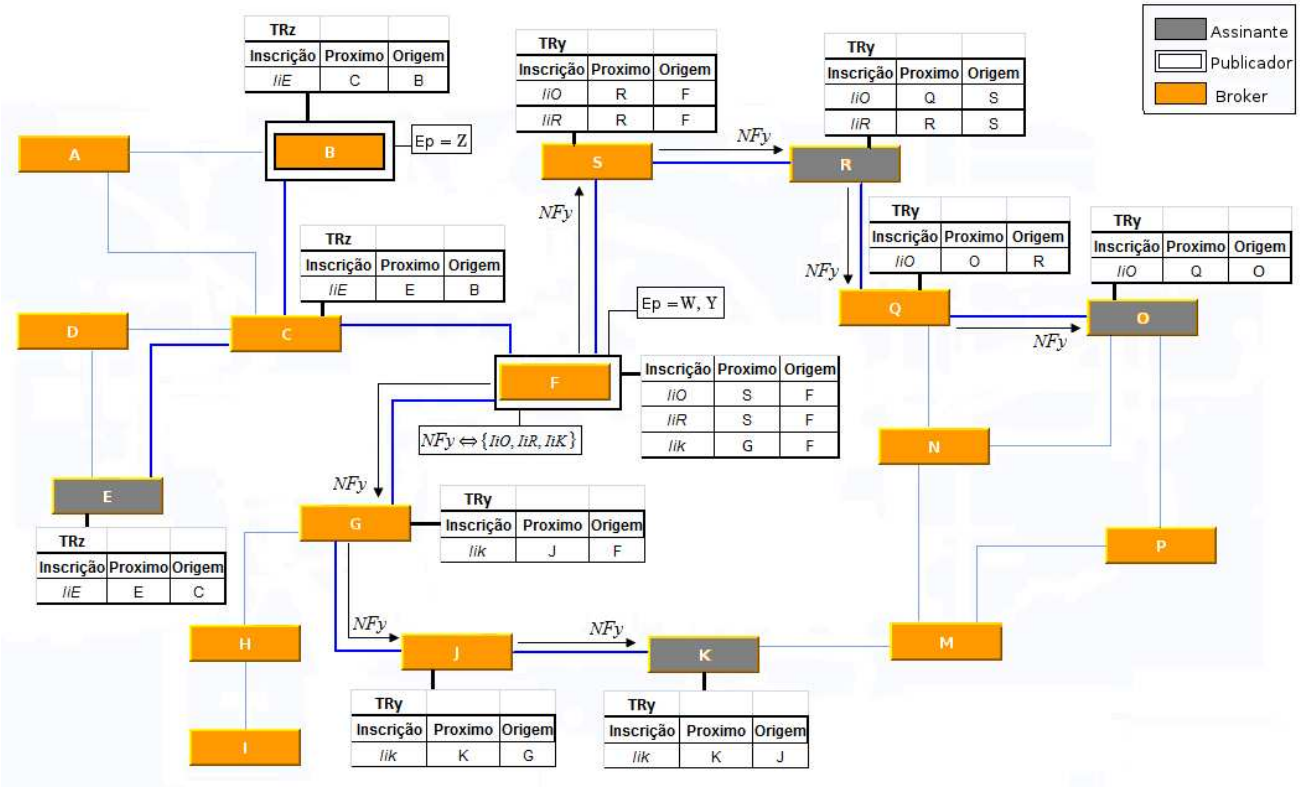

Figura 5.3: Propagação de Notificação de Evento.

Assim, o custo de M é M[Np_τ] = 7 e o tempo para cada nó raiz receber a mensagem é T = 4.

5.4 Mudança de Papel – Broker e Publicador

Os custos de quantidade e tempo de mensagens propagadas na mudança de papel de *Broker* para Publicador são iguais aos custos obtidos para a criação de inscrições, pois todos os Assinantes, que já tenham inscrições registradas do mesmo tipo de evento do novo publicador, devem receber a mensagem Q de mudança de papel. Podemos comparar as figuras 4.9 e 4.13 do capítulo 4, onde uma árvore de inscrição é criada e onde um nó *Broker* faz a mudança de papel para Publicador. A entrega de mensagens no grafo é igual para esses dois casos, sendo a disseminação por *flooding*, onde é construída a abrangência de Publicadores. Assim, para um publicador "descobrir" os nós Assinantes de inscrição correspondente ao conjunto de evento que publica, a mensagem Q deve ser entregue para todos os nós do grafo, sendo $x_\tau \epsilon$ V(G).

Na mudança de papel de Publicador para *Broker*, os custos de quantidade de mensagens e tempo na propagação de mensagens são iguais aos custos obtidos na operação *CI* só que no sentido contrário. Podemos comparar a figuras 4.11 e 4.14 do capítulo 4, onde um cancelamento de inscrição é registrado e onde um Publicador faz a mudança de papel para *Broker*. Nesses dois casos, dos exemplos das figuras 4.11 e 4.14, a mensagem Q é disseminada somente dentro da árvore de inscrição. No cancelamento de inscrição, todos os Publicadores devem receber a mensagem Q, para atualizar suas tabelas de roteamento e excluir a rota referente ao nó Assinante que está cancelando a Inscrição. Já na mudança de publicador para *Broker* é o contrário: todos os Assinantes devem receber Q para que o nó que o Publicador (que está mudando seu papel) seja excluído de sua árvore de inscrição, sendo assim também excluído da rota de 'origem' de notificações de eventos. Portanto, todo Assinante deve receber a mensagem Q quando $x_\tau \forall \tau \in P_s \mid S = \{\tau_1, \tau_2, ..., \tau_n\}$.

5.5 Conclusão

Conforme análises realizadas, podemos concluir que na operação de registro de inscrições, o custo de propagação de mensagens aumenta com o aumento de nós em uma rede. No entanto ocorre um ganho significativo na quantidade de mensagens e de tempo de propagação na notificação de eventos, pois a área do grafo para disseminação estará com uma quantidade de nós mais restrita. Independente da quantidade de nós em árvores de inscrição, os custos de notificação de um evento serão menores em comparação com custos de redes *broker*. A notificação de eventos é uma das operações de DEBS que tem maior fluxo de solicitações e conforme nosso algoritmo, esta é a operação com menores custos.

No cancelamento de inscrições, pode ser considerado que o custo e tempo de propagação em redes *broker* tem certa vantagem em relação ao nosso algoritmo de Cancelamento de Inscrições, pois somente os nós *brokers* recebem a mensagem de cancelamento. Verificamos que no nosso algoritmo proposto, ocorre um custo maior para que mensagens não sejam propagadas sem necessidade. No entanto, em sistemas onde é requisito que não ocorra "*spans*", os valores de custos são aceitáveis em relação a esta funcionalidade.

O dinamismo proposto na mudança de papel de *Broker* e Publicador foi um novo desafio em nossa abordagem, pois a maioria dos sistemas apresentados no capítulo 3, deste

trabalho, pressupõe papéis já definidos. Não há custos diferentes dos já relatados, pois as mudanças de *Broker* para Publicador e de Publicador para *Broker*, tem os mesmos custos, de quantidade e tempo de propagação, que o cancelamento de inscrição criação de inscrição respectivamente.

Podemos concluir que em nossos algoritmos propostos, temos custos maiores para estabelecimento de conexões na rede, no entanto os custos diminuem gradativamente na manutenção da rede. No entanto há um ganho de desempenho em operações realizadas sobre a rede estabelecida, que acaba sendo umas das operações realizadas com maior freqüência em DEBS.

Capítulo 6

EXPERIMENTOS

Para avaliar os algoritmos propostos no capítulo 4, adotamos a API de REDS [CUGOLA; PICCO; 05] que disponibiliza todas as funcionalidades de comunicação, independente da topologia de rede. REDS é uma infraestrutura, desde a camada de mais baixo nível, como a camada de transporte, até a camada de interface "*publish-subscribe*". Este capítulo aborda as principais características de REDS, implementação dos algoritmos de Criação de Árvore de Inscrição e Notificação de Eventos detalhados no capítulo 4, e cenários iniciais de experimentos. Consideramos que nossa implementação está em um nível inicial, no entanto relatamos nesse capítulo os módulos ainda faltantes nas questões de implementação.

6.1 REDS – Sistema Reconfigurável de Encaminhamento (*Reconfigurable Dispatching System*)

Segundo [CUGOLA;PICCO05], a estrutura aberta do *framework* REDS é destinada para o desenvolvimento de sistemas de pesquisa "*publish-subscribe*". O framework foi projetado principalmente para soluções que aplicam configurações dinâmicas em infra-estrutura de topologias para redes móveis e *peer-to-peer*. O diferencial que caracteriza a plataforma é a arquitetura modular onde, por exemplo, o formato de mensagens e filtros, os mecanismos para o gerenciamento de tabelas de roteamento, estratégia de roteamento e o protocolo de transporte de rede sobreposta, são módulos separados. A plataforma disponibiliza uma estrutura de classes, na linguagem Java, que definem o seguinte: 1- API

para acessar os serviços de "*Publish/Subscribe*", 2 - Arquitetura de um *Broker* genérico com implementação de interfaces "*Publish/Subscribe*".

6.1.2 API Cliente

O termo "cliente" é a definição dos nós que podem fazer a chamada de operações de Assinantes como também fazer chamadas operações de Publicador. Cada nó *cliente* pode ser um Assinante e Publicador ao mesmo tempo. Através da interface do cliente, um nó se conecta a um *Broker*, para o qual envia mensagens na chamada de operações de inscrição e publicação. Conforme a figura 6.1, a interface *DispatchingService* define as operações "*Publish" e* "*Subscribe*" que são implementadas pelo cliente.

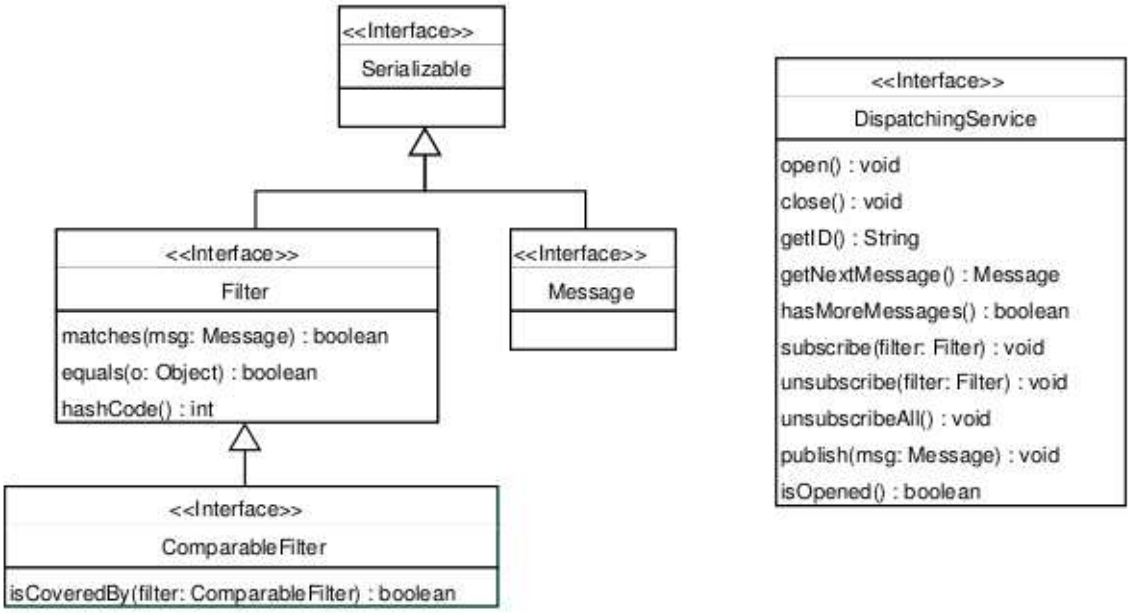

Figura 6.1: REDS – Interface de um Cliente [CUGOLA;PICCO05].

Mensagens trocadas na rede entre componentes, não tem restrição de formato onde a interface *Message* implementa somente a interface *Serializable*, sendo uma mensagem qualquer objeto serializado. O fato de não ter restrição de formato possibilita inclusão da implementação de formato de inscrição como XML, tuplas e registros estruturados por exemplo. Já a interface "*Filter"*, que define uma inscrição, também tem o mínimo de restrições, onde implementa *Serializable*. Essa interface disponibiliza a função de "merge"

(visto no capítulo 3) na operação "*isCoveredBy*", evitando que filtros que tenham a mesma cobertura de notificações, sejam propagados sem necessidade.

6.1.2 Arquitetura

A arquitetura de um *Broker* é estruturada em duas camadas: transporte e roteamento, conforme a figura 6.2. Um *Broker* ao entrar na rede este fica aguardando requisições de Clientes ou de outros *Brokers*, que podem ser requisições de solicitação de conexão, inscrição ou publicação de eventos.

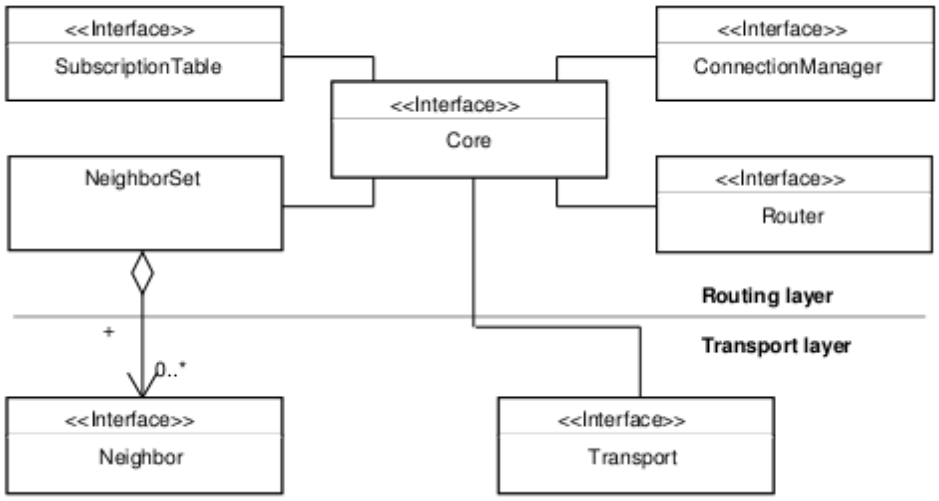

Figura 6.2: REDS – Arquitetura de um Broker [CUGOLA;PICCO05].

Núcleo ("core")

A interface do *núcleo* do sistema abstrai o componente central de um *Broker*: realiza a mediação da comunicação entre os componentes do sistema, para que assim um Cliente não tenha conhecimento dos Publicadores, dos quais há interação sem conhecimento. A interface *núcleo* espelha a maioria dos métodos fornecidos por outras interfaces. Com isto, ocorre a dissociação entre os componentes, que precisam estar cientes apenas da existência do *núcleo*. Outra responsabilidade do *núcleo* é enviar mensagens internas da interface *InternalMessage*,

como mostrado na Figura 6.3.

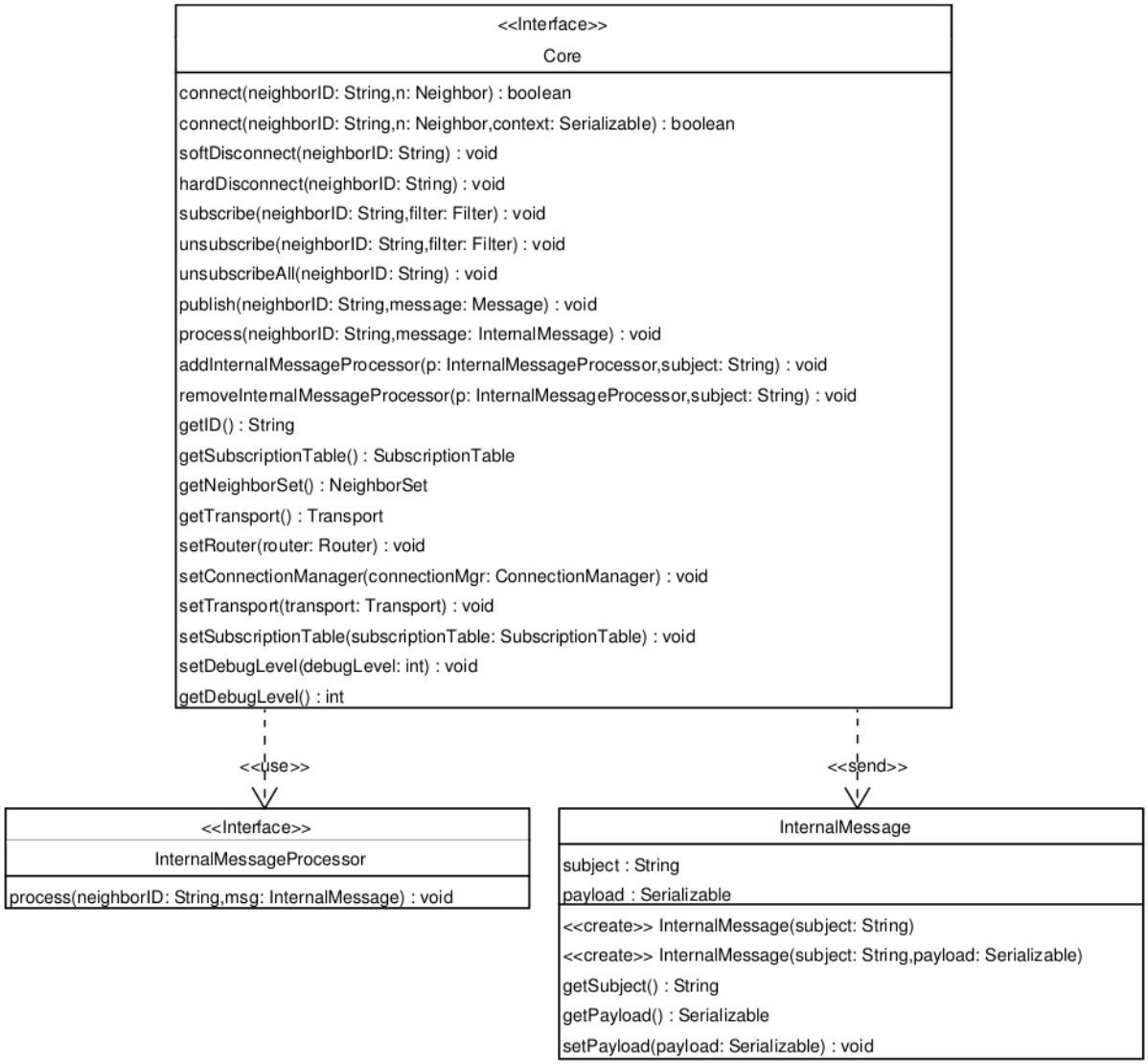

Figura 6.3: REDS – Interface Núcleo [CUGOLA;PICCO05].

Mensagens internas são mensagens de controle, sendo mensagens trocadas entre *Brokers* para ajudar no encaminhamento cooperativo. Exemplos disso são mensagens que contêm o número de nós conectados a um *Broker* para evitar rotas congestionadas ou mensagens que contém informações de localização em esquemas de roteamento "*location-aware*". Mensagens internas são caracterizadas por um assunto ("*string*" simples) e um objeto serializável. Objetos que implementam a interface *InternalMessageProcessor* podem ser registrados para

assuntos específicos. Quando o núcleo recebe uma mensagem interna da camada de transporte, este repassa a mensagem para os processadores registrados com base no assunto.

Transporte

A camada de transporte fornece os mecanismos utilizados para o transporte de mensagens de qualquer componente (*broker* ou cliente) na rede, onde disponibiliza interfaces *Transport* e *Neighbor*, mostrado na Figura 6.4. A interface *Transport* disponibiliza métodos para abrir e fechar links de conexões entre *Brokers*, onde tais métodos podem ser usados para definir a topologia inicial da rede ou também alterar a topologia em tempo de execução.

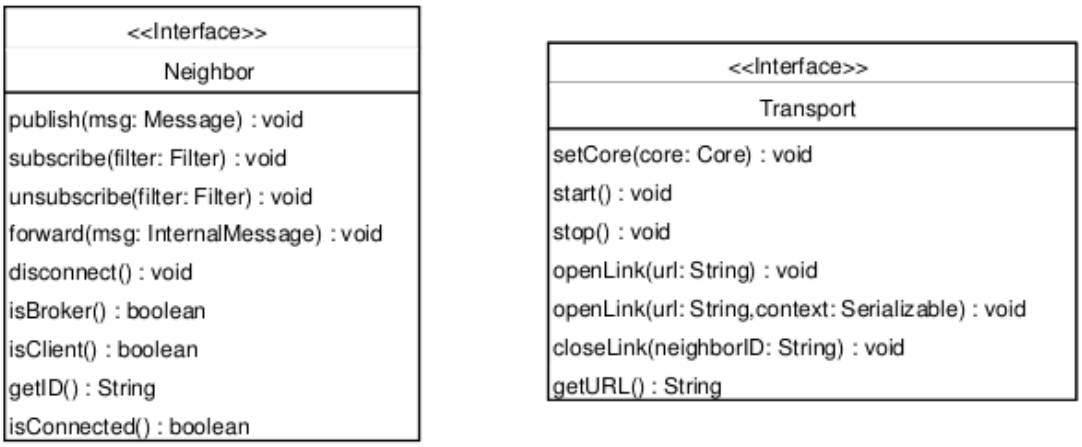

Figura 6.4: REDS – Interfaces da camada de Transporte [CUGOLA;PICCO05].

A interface "*Neighbor*" é implementada por todos os nós e está ligada diretamente à interface do Transporte. Tal interface define métodos como "*isBroker"* e "*isClient*" que verificam qual é o papel do nó que está conectado a um Broker ou a outros clientes.

Roteamento

As principais interfaces da camada de roteamento são as interfaces "*Core*", "*Router*", "*ConnectionManager*" e "*SubscriptionTable*" que é a interface que representa a estrutura de dados que armazena as inscrições para fins de roteamento.

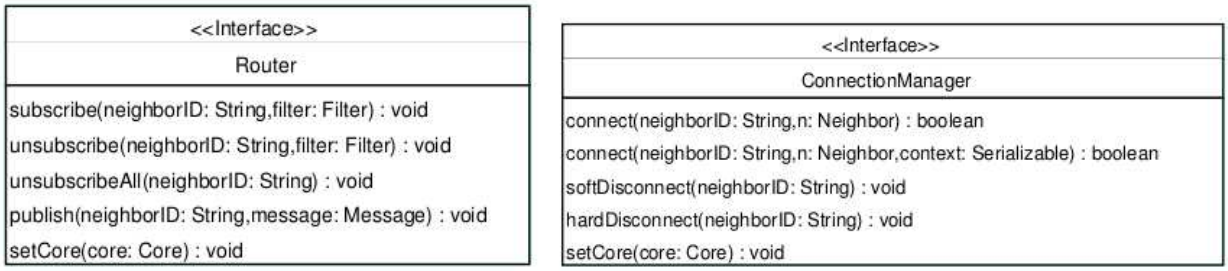

Figura 6.5: REDS – Interfaces da camada de Roteamento [CUGOLA;PICCO05].

Conforme a figura 6.5, a interface *Router* abstrai o componente Broker que implementa uma estratégia de roteamento específica. Os métodos desta interface são invocados pela camada de transporte, através do *Núcleo*, para notificar o componente *Router* que a criação ou cancelmento de inscrição foi recebida de um dos vizinhos do Broker, e precisa ser encaminhado de acordo com a estratégia de roteamento encapsulada no componente.

6.2 Implementação

Conforme detalhado na seção anterior, os componentes do sistema são divididos entre *Cliente* e *Broker*, sendo o *Broker* o componente centralizado no sistema. Todo cliente implementa a interface *DispatchingService*, que nada mais é do que uma interface para o cliente enviar e receber mensagens de um *Broker*. Para permitir que todo cliente desempenhe o papel de *Broker* é necessário alterar a arquitetura, fazendo com que todo nó tenha acesso a duas interfaces: *DispatchingService* e *TCPTransport*. Dessa forma, todo nó fica aguardando mensagens na *Thread* do objeto *TCPTransport* e cada objeto *TCPTransport* tem acesso ao serviço de "*Dispatcher*", para que a cada mensagem recebida seja também repassada conforme o algoritmo de Construção de Árvore de Inscrição. A figura 6.6 mostra os componentes do sistema na arquitetura proposta, onde o objetivo é fazer com que cada nó seja capaz de receber e repassar mensagens.

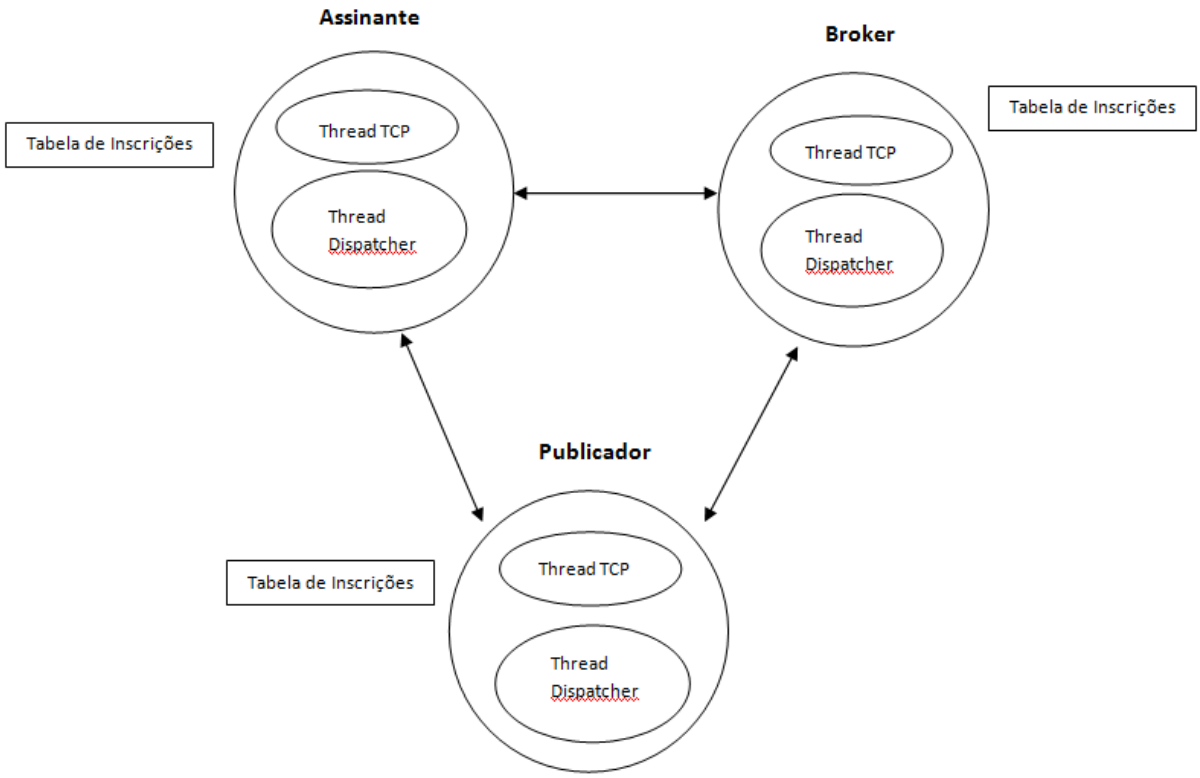

Figura 6.6: REDS – Alterações da arquitetura de componentes do sistema.

Para a implementação de componentes onde todos realizam o papel de *broker*, foram realizadas alterações conforme as tabelas 6.1 e 6.2. Na tabela 6.1 temos a lista de classes incluídas na versão original de REDS para a Criação de Árvores de Inscrição.

RecordRouting	Representa a estrutura de um registro da tabela de roteamento, onde há uma inscrição (*Filter*), um nó de origem (NodeDescriptor) e um Próximo nó (NodeDescriptor).
Inscricao	Representa a estrutura de uma inscrição, onde um objeto TipoEvento e uma expressão regular são atributos da inscrição.
Notificacao	Representa o modelo de uma Notificação emitida por Publicadores. Os atributos da classe são correspondentes aos atributos da classe *Inscricao*.
TipoEvento	Representa os tipos de Eventos que os Publicadores emitem e que Assinantes podem registrar interesse.

Tabela 6.1: Lista de novas classes na versão original de REDS.

Na tabela 6.2 temos a lista de classes incluídas na versão original de REDS.

SubscriptionTable	Alterada a estrutura da tabela para incluir objetos da classe *RecordRouting*. Incluido as colunas ‘inscrição’, ‘origem’ e ‘próximo’. Alteração de assinaturas de métodos.
GenericTable	Adicionado métodos para adicionar registros na tabela de roteamento utilizando a classe *RecordRouting*.
TCPDispatchingService	Envia e recebe requisições TCP.
DispatchingService	Interface do serviço de “*dispatcher*”. A classe *TCPDispatchingService* implementa *DispatchingService*.
NodeDescriptor	Representa um objeto nó no grafo.
SubscriptionForwardingRoutingStrategy	Classe que inclui as funcionalidades da classe *SubscriptionForwardingRoutingStrategy* para um nó enviar uma mensagem registrada para todos os seus vizinhos. As operações “*publish-subscribe*” são representadas no objeto de Tipos de Mensagens (Q, C, MP e MB) para as operações CAI, CI, MPB e MPP. Essa classe acessa os métodos *send* e *receive* da classe TCPProxy (envia mensagens por Sockets) e é responsável pela construção e atualização de Árvores de Inscrição.
AbstractTopologyManager	Incluídos os métodos *subscribe* e de criação de inscrição com assinatura para receber um objeto Inscrição como parâmetro.
AbstractTransport	Retirado o tratamento de envio de inscrições somente para *Broker*.

Tabela 6.2: Lista de classes alteradas na versão original de REDS.

Todas as classes incluídas e alteradas que constam nas tabelas 6.1 e 6.2 fazem parte do módulo de estabelecimento de conexões na rede. Listamos na tabela 6.3, de modo geral, classes e métodos necessários para implementação das demais funcionalidades da proposta.

Funcionalidade	Classe	Métodos
Notificação de Eventos	GenericTable	addSubscription(RecordRouting registroRoteamento); getAllFilters(NodeDescriptor n); getAllFiltersExcept(boolean duplicate, NodeDescriptor n); getSubscribedNeighbors(Filter f)
	SubscriptionTable	getAllFilters(NodeDescriptor n); matches(Message message, NodeDescriptor excludedDestination)
	SubscriptionForwardingRoutingStrategy	subscribe(NodeDescriptor neighbor, Filter filter); publish(NodeDescriptor sourceID, Message message)
	TCPTransport	accept(), openLinkHelper(NodeDescriptor oNode, String url)
	AbstractTopologyManager	getAllNeighborsExcept(NodeDescriptor excludedNeighbor)
	TCPDispatchingService	forward, run(), reply(Message, MessageID); subscribe(Inscricao oInscricao); unsubscribe(Filter filter)
Criação e cancelamento de Inscrições	SubscriptionTable	(RecordRouting registroRoteamento); removeSubscription(RecordRouting registroRoteamento);
Mudança de papel	AbstractTopologyManager	addNeighbor(NodeDescriptor oNodeDescriptor); confirmConnection(NodeDescriptor id, Transport transport); removeNeighbor(NodeDescriptor removedNeighbor)

Tabela 6.3: Lista de classes e métodos para alteração na versão original de REDS.

Após a conclusão da implementação, a API poderá ser utilizada importando o arquivo de extensão .jar do projeto REDS.

6.3 Conclusão

Verificamos que a API que foi adotada para implementação é flexível, onde as alterações foram e podem ser realizadas nos pacotes existentes no projeto, sem necessidade de criação de novas extensões em todas as camadas definidas nos pacotes do projeto REDS. Para simulação completa das funcionalidades propostas, é necessário dar continuidade na implementação. Com base na análise realizada no capítulo 5 deste trabalho, podemos concluir por indução que na principal operação na rede, que é a Criação da Árvore de Inscrição, os custos de quantidade de troca de mensagens não variam muito com a versão original comparada de REDS, sendo somente o tempo na propagação de mensagens que pode ter variação, apresentando um custo maior. Conforme as alterações realizadas na API, a disseminação de mensagens no grafo poderá ser realizada dentro de árvores de inscrição, onde acreditamos que o custo de tempo de entrega de mensagens é compensado na disseminação de notificação de um evento e demais operações que constam na proposta do nosso trabalho. No entanto, é necessário que haja continuidade na fase de implementação para que dados de simulação possam ser relatados e comparados com as análises realizadas neste trabalho.

Capítulo 7

CONCLUSÃO

Este trabalho apresentou uma pesquisa que abrange abordagens para roteamento de mensagens em Sistemas Distribuídos Baseados em Eventos. Tivemos o objetivo de detalhar soluções específicas de roteamento, visando relacionar tais soluções a ideia da nossa proposta. Realizamos análises de custos referentes ao tempo e número de mensagens, durante a propagação de mensagens, com base na teoria de grafos. Conforme análises realizadas, podemos concluir que a abordagem de Árvores de Inscrições (baseadas em Árvores de Abrangência) para a disseminação de mensagens é eficiente dentro dos parâmetros adotados como indicativos: custos de tempos e número de mensagens propagadas.

7.1. Continuidade da Proposta

Conforme detalhamos no capítulo 6, a implementação e os experimentos foram realizados de forma sucinta, a fim de se ter uma amostra inicial da proposta, onde realizamos alterações na plataforma de REDS para a implementação do algoritmo de criação de árvores de inscrição. No entanto, a implementação de todos os algoritmos propostos é importante, para que avaliações completas possam ser visualizadas. Por tanto, esse trabalho de pesquisa pode ser melhorado em termos de experimentos e avaliações, contando com o seguinte:

1. Implementação dos algoritmos de Notificação de Eventos, Cancelamento de Inscrições e Mudança de papel Broker - Publicador.
2. Aplicação de experimentos em todos os algoritmos citados no passo anterior, comparando com a plataforma REDS original.

3. Comparação de resultados de experimentos com outros trabalhos relacionados, onde os parâmetros de análise sejam os mesmos de nosso trabalho: quantidade de mensagens propagadas e tempo de propagação de mensagens.

7.2. Considerações e trabalhos futuros

Nas seções seguintes, vamos abordar algumas considerações para continuidade do nosso trabalho, para aplicação no grafo formado da rede sobreposta, a fim de aperfeiçoar proposta de notificação de eventos descentralizada.

7.2.1. Grupos de interesse: estabelecimento de conexões

Em nossa proposta, árvores de interesse são construídas, formadas por assinantes e publicadores do mesmo tipo de evento. Dessa forma, a entrega de notificações de eventos é sempre realizada dentro de árvores de interesse, evitando a disseminação para nós que não tenho interesse. Porém, pode ocorrer que uma árvore de determinado tipo de evento tenha a maior parte de nós assinantes na rede, ocasionando sobrecarga de publicadores. Isso pode gerar uma demora significativa para que notificações de eventos sejam entregues a assinantes.

Para a solução do problema de balanceamento, uma possibilidade a ser avaliada é a aplicação do modelo de Campos Magnéticos Virtuais [LIMA;CALSAVARA10]. Aplicando tal modelo, a construção de árvores de inscrições na rede pode ser realizada por atração magnética. Um Publicador de um tipo de evento pode “atrair” assinantes que tenham inscrições correspondentes ao tipo evento que este produz. A força de atração de cada Publicador poderia ser o número de níveis de cada árvore em que este tem abrangência.

7.2.2. Notificações de Eventos

No contexto de “redes Broker”, todos os nós que são *Brokers* recebem notificações de eventos, independente se existem ou não existem vizinhos Assinantes com interesse no evento publicado. A principal diferença da nossa proposta é que todos os nós de uma árvore de inscrição recebem a notificação de evento, independente de seu interesse, sendo filtrado somente o tipo de evento de interesse. Ocorre semelhança com a disseminação de notificações de eventos entre *Brokers*, porém com o ganho de que somente os nós da mesma árvore recebem a notificação, e não todos os nós *Brokers* do grafo. Um ponto a ser melhorado é fazer com que cada nó realize a disseminação de notificação fazendo a correspondência entre a expressão regular que consta na inscrição de sua tabela de roteamento. Não fizemos a abordagem de Correspondência de Inscrições neste trabalho, porém a aplicação da ideia dos algoritmos de Roteamento Baseado em Identidade, “Merge” e Abrangência, conforme foi apresentado no capítulo 3 pode aperfeiçoar significativamente os custos de propagação de notificação de eventos. Outra questão que sentimos necessidade de abordar é a causalidade de notificações de eventos. Em contextos onde assinantes precisam receber notificações em ordem, é necessário que se proponha uma solução, semelhante à proposta apresentada em [PIETZUCH;BACON;03].

7.2.3. Publicador para Broker: mudança de papel

Consideramos na proposta do nosso trabalho, que todo nó que deixa de ser Publicador para se tornar apenas *Broker*, não têm a tarefa de verificar se os nós Assinantes de sua abrangência ficarão sem um nó Publicador como fonte de notificações de eventos, ou seja, não realizamos a abordagem de tolerância a falta de um Publicador. Nossa intenção inicial foi a de aplicar o modelo de Campos Magnéticos em nossa proposta, porém não incluímos nas operações de DEBS. Partimos do pressuposto que a força de atração de cada nó seria aplicada para eleger um novo Publicador em árvores de abrangência.

Referências Bibliográficas

[LIMA;CALSAVARA10] LIMA JUNIOR, Luiz Augusto de Paula; CALSAVARA, Alcides . Autonomic Application-Level Message Delivery Using Virtual Magnetic Fields. Journal of Network and Systems Management, v. 18, p. 97-116, 2010.

[PIETZUCH06] PIETZUCH, P. GERO, M., FIEGE, L. Distributed Event-Based Systems. New York, Springer, 2006.

[GAL;HAD10-Cap.1] GAL A., HADAR E. Generic Architecture of Complex Event Processing Systems. In: HINZE, M. A., BUCHMANN A. Principles and Applications of Distributed Event-Based Systems. New York: Information Science Reference, 2010. P. 1.

[HINZE;BUCHMANN10-Cap.2] BLANCO R., ALENCAR P. In: HINZE, M. Annika., BUCHMANN A. Event Models in Distributed Event Based Systems.
New York: Information Science Reference, 2010. P. 19-20.

[LEGATHEUX;DUARTE10] LEGATHEUX M. J.; DUARTE S. *Routing Algorithms for Content-based Publish/Subscribe Systems.* Journal: IEEE Communications Surveys & Tutorials, 2010.

[BALDONI05] BALDONI, R.; BERALDI, R., QUERZONI L., CUGOLA G., MATTEO M. *Content-Based Routing in Highly Dynamic Mobile Ad Hoc Networks.* International Journal of Pervasive Computing and Communications, vol. 1, no 4, dezembro/2005 p. 277-288.

[CUGOLA;PICCO05] Cugola, G., Picco, G.P.: Reds: A reconfigurable dispatching system. Relatório Técnico 15-05, Departamento de Informática, Universidade de Roma "La Sapienzia", Roma, Itália, 2005.

[BALDONI;QUERZONI;VIRGILLITO05] Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Relatório Técnico 15-05, Departamento de Informática, Universidade de Roma "La Sapienzia", Roma, Itália, 2005.

[MÜHL02] Muhl, G. Large-Scale Content-Based Publish/Subscribe Systems. PhD thesis, Darmstadt University of Technology, Darmstadt, Germany, 2002. http://elib.tu-darmstadt.de/diss/000274/.

[LEGATHEUX;DUARTE10] LEGATHEUX M. J.; DUARTE S. *Routing Algorithms for Content-based Publish/Subscribe Systems.* Journal: IEEE Communications Surveys & Tutorials, 2010.

[CUGOLA;NITTO;FUGGETTA01] G. Cugola, E. Di Nitto, and A. Fuggetta. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE Transactions on Software Engineering*, 2001.

[CUGOLA; PICCO; 05] Cugola, G., Picco, G.P.: Reds: A reconfigurable dispatching system. In: Technical Report, Politecnico di Milano. (2005)

[EUGSTER; GUERRAOUI; DAMM 01] P. T. Eugster, R. Guerraoui, and C. H. Damm. On objects and events. In L. Northrop and J. Vlissides, editors, Proceedings of the OOPSLA
’01 Conference on Object Oriented Programming Systems Languages and Applications, páginas 254–269, 2001. ACM.

[BALDONI; VIRGILLITO 05] R. Baldoni, A. Virgillito. Distributed event routing in publish/subscribe communication systems: a survey. Technical Report 15-05, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienzia", Rome, Italy, 2005.

[KOOSHA10] KOOSHA, P., VANROMPAY, Y., BERBERS Y. *Fadip: Lightweight Publish/Subscribe for Mobile Ad hoc Networks*. OTM'10 Proceedings of the 2010 international conference on On the move to meaningful internet systems: Part II. Springer-Verlag Berlin, Heidelberg 2010.

[YOO09] YOO, S. SON, J. H., KIM, H. M. *A scalable publish/subscribe system for large mobile ad hoc networks.* Journal of Systems and Software - Elsevier, vol. 82, fevereiro / 2009, p. 1152-1162.

[CAO; SINGH 04] CAO F., SINGH Jaswinder P. *Efficient Event Routing in Content-based Publish-Subscribe Service Networks.* INFOCOM. Vigésima Terceira Conferência Anual da IEEE e Sociedades de comunicação, vol. 2, março/2004 p. 929-940.

[CUGOLA;NITTO;FUGGETTA01] Cugola G., Nitto E., Fuggetta A. *The JEDI event-based infrastructure and its application to the development of the OPSS WFMS.* Jornal IEEE Transactions on Software Engineering, *vol. 27 questão 9, setembro/2001 p.* 827-850.

[SANTORO06-Cap. 2] Nicola Santoro. Design and Analysis of Distributed Algorithms. Wiley-Interscience, 2006.

F. Cao and J. Singh. Medym: Match-early with dynamic multicast for content-based publish-subscribe networks. Proceedings of the ACM/IFIP/USENIX 6th International Middleware Conference (Middle-ware 2005), 2005.

[ZHANG;HU05] R. Zhang and Y. Hu. HYPER: A Hybrid Approach to Efficient Content-Based Publish/Subscribe. Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference, páginas 427–436, 2005.

[CUGOLA06] CUGOLA G.; MURPHY A.; PICCO G. Content-based Publish-Subscribe in a Mobile Environment. *Mobile Middleware,* A. Corradi and P. Bellavista eds., pp. 257-285, Auerbach Publications, 2006.

[PIETZUCH06-01] PIETZUCH, P. GERO, M., FIEGE, L. Basics: Terminology - Constituents of Event-Based Systems. New York, Springer, 2006. p. 11-12.

[PIETZUCH06-02] PIETZUCH, P. GERO, M., FIEGE, L. Content-Based Models and Matching: Semi estructured Records. New York, Springer, 2006. p. 54.

[PIETZUCH06-03] PIETZUCH, P. GERO, M., FIEGE, L. Content-Based Models and Matching: Matching Algorithms - Counting Algorithm. New York, Springer, 2006. p. 60.

[PIETZUCH06-04] PIETZUCH, P. GERO, M., FIEGE, L. Distributed Notification Routing: Content-Based Routing Algorithms - Simple Routing. New York, Springer, 2006. p. 84.

[PIETZUCH06-05] PIETZUCH, P. GERO, M., FIEGE, L. Distributed Notification Routing: Content-Based Routing Algorithms - Identity-Based Routing. New York, Springer, 2006. p. 88.

[PIETZUCH06-06] PIETZUCH, P. GERO, M., FIEGE, L. Distributed Notification Routing: Content-Based Routing Algorithms - Covering-Based Routing. New York, Springer, 2006. p. 93-95.

[PIETZUCH06-07] PIETZUCH, P. GERO, M., FIEGE, L. Distributed Notification Routing: Extensions of the Basic Routing Framework - Hierarchical Routing Algorithms. New York, Springer, 2006. p. 112.

[PIETZUCH06-08] PIETZUCH, P. GERO, M., FIEGE, L. Basics: A Model Distributed Notification Service - Architecture. New York, Springer, 2006. p. 21.

[PIETZUCH06-09] PIETZUCH, P. GERO, M., FIEGE, L. Basics: A Model Distributed Notification Service - Architecture. New York, Springer, 2006. p. 20 - 23.

[PIETZUCH06-10] PIETZUCH, P. GERO, M., FIEGE, L. Existing Notification Services: Standards. New York, Springer, 2006. p. 305 - 313.

[PIETZUCH06-11] PIETZUCH, P. GERO, M., FIEGE, L. Existing Notification Services: Research Prototypes. New York, Springer, 2006. p. 324 - 338.

[CAIN;DEERING;KOUVELAS;FENNER02] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan. IGMP - *Internet Group Management Protocol*, Versão 3, RFC 3376, 2002. Acesso em 20 de Setembro de 2013. Disponível em: http://www.ietf.org/rfc/rfc4604.txt.

[PIETZUCH;BACON;03] PIETZUCH P., BACON J.: Hermes: A distributed event-based middleware architecture. Workshop Internacional sobre Sistemas Distribuídos Baseados em Eventos (International Workshop on Distributed Event-Based Systems - DEBS), 2003.

[OMG12] Object Management Group (OMG). CORBA - The Common Object Request Broker: Arquitetura e Especificação, versão 3.3. OMG documento formal/2012-11-12, formal/2012-11-14, formal/2012-11-16 - novembro de 2012. Acesso em 28/10/2013. Disponível em: http://www.omg.org/spec/CORBA/3.3/

[OMG04-01] Object Management Group (OMG). Corba Event Service: Arquitetura e Especificação, versão 1.2. OMG documento formal/2004-10-02 – outubro de 2004. Acesso em 28/10/2013. Disponível em: http://www.omg.org/spec/EVNT/

[OMG04-02] Object Management Group (OMG). Corba Notification Service: Arquitetura e Especificação, versão 1.1. OMG documento formal/2004-10-11 – outubro de 2004. Acesso em 28/10/2013. Disponível em: http://www.omg.org/spec/NOT/

[OMG04-03] Object Management Group (OMG). Corba Event Service: Modules and Interfaces: Event Channels, p. 2-5. Arquitetura e Especificação, versão 1.2. OMG documento formal/2004-10-02 – outubro de 2004.

[ASF-RIVER13] Apache Software Foundation (ASF). Jini Distributed Events Specifications, versão 1.0. Acesso em 29/10/2013. Disponível em: http://river.apache.org/doc/specs/html/event-spec.html

[ASF-RIVER13] Apache Software Foundation (ASF). River Wiki. Acesso em 29/10/2013. Disponível em: http://wiki.apache.org/river/JiniOrgWiki

[SUN01] SUN Microsystems Inc. Jini Architecture Especification, versão 1.2. Califórnia, San Antonio Road, 901 - Palo Alto. Sun Microsystems, Inc, 2001. Acesso em 29/10/2013. Disponível em: http://river.apache.org/doc/specs/html/jini-spec.html

[GEHANI ; JAGADISH SHMUELI;92] GEHANI J., JAGADISH H. V., SHMUELI O. Event Specification in an Active Object-Oriented Database. ACM New York. SIGMOD International Conference on Management of Data, páginas 81–90, 1992.