Event-Based Middleware - A New Paradigm for Wide-Area Distributed.pdf

Os clientes e servidores estão espalhados pela internet, em ambientes heterogéneos, tornando-se difícil a interligação entre eles.

Num sistema síncrono os clientes e servidores têm de estar ligados em simultâneo para que possa existir comunicação.

Para funcionar na internet os sistemas baseados em troca de eventos funcionam melhor, uma vez que permitem a comunicação muitos para muitos, esta comunicação é possibilitada por o cliente não ter de decidir qual o melhor parceiro de comunicação.

Um sistema de publish-subscribe pode ter uma funcionalidade semelhante, mas tem uma área de aplicação limitada. Um middleware possibilita type-checking de invocações, assegura entrega de mensagens, controlo de acessos e transacções.

Características importantes em middlewares e sistemas publish/subscribe:

- **Escalabilidade**: Característica crucial em aplicações Web. A escalabilidade de um sistema é determinada pela escalabilidade dos seus componentes. Evitar as dependências a serviços centralizados. Stateless, consumo eficiente de largura de banda e memória.

- **Interoperabilidade**: O modelo de eventos e linguagem de subscrição deve ser independente da linguagem e da plataforma. Não de tirar partido de particularidades da camada de rede que não sejam consideradas *standards*. Para melhorar a performance devem ser utilizados serviços disponíveis. Permitir a integração do número possível de componentes.

- **“Reliability”**: A garantia de entrega de mensagens pode constar na definição de qualidade de serviço requerida pelo cliente. Não deve ser apenas dado um nível de qualidade de serviço, uma vez que as utilizações do middleware são genéricas e terem contextos diferentes, devendo ser oferecidos diferentes níveis de garantia de entrega de mensagens, desde “o melhor esforço” até “garantido e síncrono”. A tolerância a falhas permite a garantia de serviço, mesmo em situação de falhas de clientes ou de servidores, mecanismos, como persistência de eventos em bases de dados, são exemplos de tolerância a falhas.

- **Clareza**: Subscrições com filtragem de eventos por conteúdo. *Composite event expressions* detectam padrões na passagem de eventos no middleware, permitindo aos subscritores de eventos serem mais expressivos na informação que desejam. Tradeoff Clareza vs Eficiência.

- **Usabilidade**: O middleware deve ser facilmente integrável com a linguagem da aplicação. Mapeamento de eventos e subscrições em objectos da linguagem. Type-checking de tipos de eventos e subscrições.

Composite Events ???

Uso de ODL – Object Definition Language??? ODMG

Armazenamento de eventos de forma persistente para tornar os Event Brokers mais robustos na correcta entrega de eventos a todos os subscritores.

Funcionalidades tradicionais de middlewares: QoS, segurança, transacções, etc, são essenciais.

# Motivações

Os sistemas de larga escala estão a ganhar importância pois os custos de conectividade estão cada vez mais a descer. Com base neste facto torna-se possível a criação de grandes sistemas *standards* constituídos por milhares de outros sistemas.

Fraco acoplamento entre os componentes de um sistema *standard* toma um papel principal na sua criação, por causa da natureza dinâmica e heterogeneidade dos sistemas que o compõem.

The Evolution of Publish/Subscribe Communication System

# Introdução

Nos últimos anos a importância dada ao paradigma publish/subscribe como um meio para distribuir informação através de sistemas distribuídos tem vindo a crescer. Os participantes das comunicações podem ser publicadores de informação ou subscritores de informação.

Motivações

As principais características de um paradigma de comunicação muitos para muitos, tal como este, são:

- Anonimidade – Entidades não necessitam de ter conhecimento da entidade com que comunicação.

- Desacoplamento no tempo - Emissor e receptor não necessitam de estar ligados ao mesmo tempo para poderem comunicar.

- Desacoplamento de fluxo de informação – O acto de envio de informação ou recepção não bloqueia os participantes.

Os sistemas que implementam o paradigma de comunicação publish/subscribe podem ser classificados de duas formas:

- Topic-based

- Content-based

Num sistema publish/subscribe do tipo topic-based os eventos são distribuídos de acordo com o assunto a que estão ligados, a estes assuntos dá-se o nome de tópico. Um subscritor subscreve os eventos associados a um tópico e recebe todos os eventos publicados nesta situação, desta forma cada tópico define um canal lógico com semântica de muitos para muitos. Enquanto que num sistema publish/subscribe do tipo content-based as subscrições são mais flexíveis, uma vez que se referenciam a características do conteúdo dos eventos. Desta forma cada subscrição pode ser vista como um canal lógico dinâmico.

A possibilidade de criação de números elevados de canais lógicos levou a que os sistemas publish/subscribe do tipo content-based não pudessem ser implementados com base em servidores centralizados ou em soluções a nível da camada de rede, como a capacidade epamulticast IP, uma vez que um computador não tem capacidade para processar todas as subscrições existentes e que não existem endereços IP multicaste suficientes para todos os possíveis canais lógicos.

As dificuldades na implementação de sistemas publish/subscribe levaram a crer-se que a melhor forma de implementar este tipo de sistemas seria a nível aplicacional, através de um conjunto *event brokers*, que trocam informação através de comunicações TCP/IP ponto a ponto.

As comunicações entre brokers têm de ter em conta dois problemas: encaminhamento de subscrições e encaminhamento de informação. O encaminhamento de subscrições refere-se ao processo de proliferação das subscrições por todos os brokers, por forma a criar um mapeamento entre subscrições e subscritores. O encaminhamento de informação refere-se ás operações de “matching” e reenvio. A operação “matching” refere-se ao processo de identificação dos subscritores a que um item recebido pelo broker deve ser enviado. Este processo é realizado com base no mapeamento entre subscrições e subscritores. A operação de reenvio consiste na transmissão de um evento por uma sequência de ligações TCP até chegar aos destinos finais.

A cooperação entre os brokers destina-se a optimizar algumas métricas de performance ligadas ás aplicações, tais como, o desempenho de passagem de informação ou número de ligações TCP por distribuição de informação, assim como para garantir que os sistemas publish/subscribe são de confiança e ou de distribuição atempada de informação.

# Estado da arte

Os diferentes sistemas de publish/subscribe diferente entre si pela forma como implementam os mecanismos do paradigma( encaminhamento de subscrições e informação ). Um exemplo destas diferenças é o caso do TIB/Rendezvous, que implementa o encaminhamento de informação sem ter em conta o conteúdo da informação a ser encaminhada. Neste caso, a informação enviada por um *event broker* é distribuída por toda a árvore de *brokers* do sistema. Cada *broker* guarda a informação dos subscritores presentes na sua LAN, desta forma o encaminhamento de subscrições apenas é feito através dos limites das LANs.

O sistema Gyphon aplica um algoritmo de “matching” eficiente, em que cada *broker* tem uma tabela com todas as subscrições do sistema. Estas tabelas são assumidas como estáticas. Utilizando estas tabelas de subscrições, os eventos apenas circulam pelos *brokers* necessários a chegar aos subscritores que os subscrevem.

O Siena incorpora algoritmos de encaminhamento de subscrições que especificam a forma como as tabelas de subscrição de cada *broker* devem ser mantidas, por forma a evitar a comunicação excessiva de subscrições, assim como a replicação desnecessária das mesmas. Estes algoritmos baseiam-se nas relações de confinamento entre as subscrições.

O factor em comum entre os três sistemas discutidos anteriormente consiste no facto de as operações de transporte de informação serem feitas através de uma topologia ficha de conexões TCP. Pode ser utilizada uma topologia dinâmica e auto configurável de ligações TCP por forma a melhorar o desempenho aplicacional, esta topologia permitiria reduzir o número de ligações TCP necessárias para que a informação atinja o seu destino

As redes de sobreposição (overlay network infrasturures) possibilitam a criação de topologias dinâmicas de redes TCP. O principal objectivo destas redes é criar, de forma dinâmica, uma árvore de distribuição, a nível aplicacional, onde a raiz encaminha mensagens para um grupo dinâmico de clientes. Estas redes também procuram alta performance ( através da construção das árvores com vista na optimização dos parâmetros de serviço de rede ), robustez ( através da adaptação da árvore á volta de brokers com falhas ) e heterogeneidade ( qualquer cliente ligado á internet pode ser um cliente deste tipo de rede ).

A noção de rede de sobreposição é facilmente utilizável no paradigma publish/subscribe baseado em tópicos, ao considerar-se que cada tópico está associado a uma árvore, e ao implementar-se o encaminhamento de informação utilizando-se o serviço de procura das redes de sobreposição. Esta utilização não pode ser feita no paradigma publish/subscribe orientado ao conteúdo, uma vez que cada publicação poderia necessitar de uma diferente árvore de multicast.

# Ideias

- Definição de linguagens “ad-hoc” para declaração de subscrições e anúncio de publicações.

- Definição de algoritmos de encaminhamento de subscrições inteligentes que visem limitar a distribuição das subscrições, mantendo contudo um grau de disponibilidade para os mapeamentos entre os subscritores e as subscrições. Um exemplo da necessidade deste tipo de algoritmo é o SIENA, este sistema não é tolerante a falhas no sentido de garantir o mapeamento entre subscritores e subscrições. Na eventualidade de um *broker* falhar, as suas subscrições são perdidas. Uma solução para este problema passa por considerar a informação dividida em subconjuntos sobrepostos, onde cada *broker* é responsável por mapear os subscritores e as subscrições de um determinado subconjunto da informação. Desta forma as subscrições seriam partilhadas por mais do que um *broker*, sendo que na eventualidade da falha de um deles o outro continuaria a ter a subscrição.

- As ligações TCP entre os *brokers* podem ser alteradas dinamicamente, de forma a adaptarem-se á estrutura lógica das subscrições. Este mecanismo de adaptação dinâmica procura a redução de ligações feitas para a distribuição da informação, evitando que esta passe por *brokers* que não têm subscritores para a mesma. Este mecanismo de alteração dinâmica pode também ter em conta a latência das ligações TCP, por forma a dar prioridade á utilização de ligações que tenham menos latência, reduzindo desta forma o tempo de distribuição da informação.

- Garantir a animosidade dos utilizadores dos sistemas publish/subscribe implica que estes não possam disponibilizar semânticas de comunicação entre os publicadores e os subscritores. Estes sistemas apenas podem fornecer uma semântica de comunicação para cada um dos tipos de clientes, quer para publicadores, quer para subscritores.

Semântica de notificações (subscritores):

- A semântica é clara quanto á informação que o sistema de publicação/subscrição entregará aos subscritores. No entanto os pormenores das condições em que esta entrega será feita não são claros, entre os pormenores por definir estão: se será entregue, quando e quantas vezes a mesma informação será entregue a um subscritor.

- A especificação clara de uma semântica que rege durante um período de uma subscrição, que informação será entregue, quantas vezes esta será entregue durante o intervalo de subscrição é fundamental para se determinar um sistema publish/subscribe como um *standard* de paradigma de comunicação.

- Que informação será entregue – Toda a informação vista pelo *event broker* que corresponde a uma subscrição, apenas a informação que corresponda a uma subscrição e que tenha sido recebida após a chegada da subscrição.

- Quantas vezes a informação será entregue – pelo menos uma vez, exactamente uma vez ou até uma vez.

Semântica de publicações:

- A comunicação entre os produtores e os consumidores através de filas de mensagens deixa a responsabilidade de remoção de mensagens da fila ao consumidor, para o momento em que o consumidor adquire a mensagem da fila. Esta responsabilidade dada ao consumidor vai de encontro ao paradigma de publish/subscribe, uma vez que a interacção segue o estilo “pull”, o subscritor não tem direito de remover a informação.

- O tempo de vida da informação tem de ser definido pelo publicador, caso este tempo não seja definido a informação armazenada no sistema de publish/subscribe pode rapidamente ultrapassar os limites aceites pelo sistema.

An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems

# Introdução

Os primeiros sistemas publish/subscribe eram baseados em tópicos. Nestes sistemas, a unidade mínima de informação é o evento, cada evento é classificado com um conjunto de assuntos, também conhecidos por tópicos. Os publicadores têm de catalogar os eventos com um tópico e os consumidores subscrevem todos os eventos associados a um tópico.

Uma alternativa aos sistemas baseados em tópicos são os sistemas baseados em conteúdos. Estes sistemas têm a flexibilidade de permitir que os subscritores escolham os eventos a subscrever com base em filtros com tantas as dimensões, quantos os atributos dos eventos, eliminando-se a necessidade de tópicos pré-definidos.

Os sistemas baseados em conteúdos eliminam a necessidade de criar e gerir tópicos do sistema, tornando-se mais facilmente administrados.

O paradigma publish/subscribe baseado em conteúdos é o paradigma mais poderoso, no entanto ainda não surgiram implementações que sejam eficientes e escaláveis. Uma implementação eficiente deste paradigma tem de resolver dois problemas chave:

- A correspondência de um evento com um grande número de subscritores num único *broker* de eventos

- Distribuir de forma eficiente eventos pela rede de *brokers* de eventos. Este problema toma maior relevância quando:

- O sistema publish/subscribe é geograficamente distribuído e os *brokers* de eventos estão ligados com conexões lentas.

- Existe um grande número de publicadores, subscritores e eventos.

Em ambos os casos deve-se limitar a distribuição dos eventos aos *brokers* de eventosque tenham subscritores para os eventos a distribuir.

Uma das grandes vantagens dos sistemas baseados no paradigma publish/subscribe baseado em tópicos é o facto dos problemas anteriormente enumerados serem facilmente resolvidos. A correspondência é implementada á custa de uma pesquisa numa tabela; enquanto o problema de distribuição de eventos é resolvido formando grupos de distribuição para cada tópico de eventos.

Existem duas soluções simples para o problema de distribuição de eventos no paradigma publish/subscribe baseado em conteúdos:

- “Matching-first” – o evento é comparado com todas as subscrições, criando-se uma lista de destinos para o evento, de seguida este é enviado para todas as entradas dessa lista.

- “Flooding” – o evento é enviado para todos os possíveis destinos da rede, em cada um dos destinos é feita a filtragem de forma a rejeitar eventos que não sejam requisitados.

O “Matching-first” funciona melhor em pequenos sistemas, mas em grandes sistemas com um elevado número de possíveis destinos, o acréscimo á dimensão dos eventos, causado pela adição de informação de encaminhamento aos cabeçalhos dos eventos, pode tornar-se impraticável. Utilizando esta técnica, podem existir várias cópias do mesmo evento a circular no mesmo troço de rede, a caminho de diferentes destinos.

O “Flooding” tem uma menor eficiência quando num grande sistema apenas um pequeno número de clientes deseja receber um determinado evento. Esta técnica não consegue explorar a localidade dos subscritores, pois é provável que subscritores numa determinada área geográfica tenham subscrições semelhantes.

O “Link Matching” é um protocolo para a distribuição de eventos no paradigma publish/subscribe baseado em conteúdos. Este protocolo define que cada *broker* de eventos faz corresponder os eventos com os subscritores ligados aos outros *brokers* de eventos da rede, tentando determinar para que *brokers* o evento deve ser reencaminhado. Além deste trabalho de reencaminhamento entre *brokers* de eventos, cada *broker* também reencaminha os eventos para os subscritores a que está ligado, dependendo das subscrições destes. Este protocolo não sofre das mesmas desvantagens que o protocolo “Match First”, uma vez que não acrescenta informação de encaminhamento aos cabeçalhos dos eventos e cada evento apenas passa uma vez numa ligação entre *brokers*. As desvantagens do protocolo “Flooding” são evitadas ao enviarem-se os eventos apenas para os *brokers* de eventos e os clientes que os requerem, explorando-se desta forma a localidade. O protocolo “Flooding” sobrecarrega uma rede com um número de publicações bastante menor que o “Match First”.

# Algoritmo de correspondência

O algoritmo de correspondência apresentado baseia-se na ordenação e organização de subscrições numa estrutura de dados PST( Parallel Search Tree ). Nesta estrutura cada subscrição corresponde a um caminho desde o nó raiz até a um nó folha. Cada nó da estrutura contém um atributo e um valor. O processo de correspondência começa no nó raiz e consiste no percorrer de todos os caminhos que o evento satisfaça, até aos nós folha. Esta estrutura escala bem horizontalmente, uma vez que explora os aspectos em comum entre as várias subscrições, ao todas estas corresponderem a caminhos iniciais comuns desde o nó raiz.

A correspondência começa na raiz, tomando em consideração um atributo a1. Em cada um dos nós não folha, encontra-se o valor vj para o aj em consideração no nó, nos termos de pesquisa, e percorre-se a ligação da árvore que tenha o valor vj, caso exista. Caso nenhuma das ligações do nó não folha corrente tenha o valor vj, então é percorrida a ligação que assumo o valor \*, significando “don’t care” para o atributo aj. Inicia-se o processo de correspondência em cada um dos nós atingidos pelas ligações percorridas. Quando alguma das pesquisas atinge um nó folha, as subscrições ligadas a este são acrescentadas á lista de subscrições com correspondência á pesquisa efectuada. A ordenação dos atributos de pesquisa na árvore é arbitrária, no entanto as pesquisas efectuadas levam a crer que se consegue um melhor desempenho, quando os nós próximos da raiz representam um atributo que tenha menos subscrições com “don’t care”, relativo ao respectivo atributo.

O algoritmo de correspondência pode ser optimizado, de seguida sugerem-se algumas optimizações:

- “Factoring” – Alguns dos passos de pesquisa podem ser evitados ao factorizarem-se alguns atributos, existindo alguns custos de espaço. Esta optimização consiste na escolha de alguns atributos, de preferência aqueles que as subscrições raramente tenham “don’t care” relativamente a eles, para se tornarem índices. São construídas subárvores para cada valor possível dos índices.

- “Trivial Test Elimination” – Simplificação da árvore, ao eliminarem-se todos os nós que apenas são atingidos por uma ligação com o valor \*.

- “Delayed Branching” – Por vezes pode-se retardar o seguimento de ligações \* até que um conjunto de testes seja realizado. Esta optimização poda as ligações da árvore provenientes da ligação \* retardar-se o seguimento, sendo que são eliminadas as ligações inconsistentes com os testes.

# Link Matching

O “Link Matching” é uma estratégia para distribuir eventos sem usar listas de destino. Aquando a recepção de um evento cada *broker* apenas faz a correspondência necessária para determinar quais dos seus vizinhos devem receber esse mesmo evento, sendo que estes vizinhos podem ser outros *brokers* ou subscritores. Isto é, em vez de determinar qual a lista de todos os subscritores do sistema que deve receber o evento, cada *broker* determina por quais das suas ligações cada evento deve ser encaminhado. Intuitivamente esta abordagem é mais eficiente, uma vez que o número de ligações de cada *broker* é tendencialmente menor que o número de subscritores de todo o sistema.

O processo “link matching”utiliza a estrutura de dados PST, em que cada caminho desde o nó raiz até um nó folha representa uma subscrição. Esta estrutura é expandida com um vector de números ternários, sendo que pode assumir os valores “Sim” (S), “Não” (N), “Talvez” (M). O primeiro passo deste processo é anotar os nós folha com um vector com dimensão igual ao número de ligações de saída do *broker*. Cada posição deste vector determina se devem ser enviados eventos por a ligação correspondente, sendo que esta decisão baseia-se no facto de existirem subscrições atingíveis por aquela ligação. As anotações feitas nos nós folha são propagadas para os seus nós pais, de baixo para cima (bottom-up).

O significado dos valores ternários presentes nos vectores associados aos nós da PST é o seguinte:

- “Sim” - baseado nos testes feitos até ao momento, o evento terá correspondência com um subscritor que é mais facilmente atingível através da ligação correspondente á posição do vector, onde o valor está inserido.

- “Não” - o evento não será correspondido com algum subscritor atingível através da ligação correspondente á posição do vector, onde o valor está inserido.

- “Talvez” – é necessária mais pesquisa para determinar se existe algum subscritor atingível através da ligação correspondente á posição do vector, onde o valor está inserido.

O algoritmo “link matching”consiste nos seguintes três passo.

- Em cada *broker*, a PST é anotada com vectores de valores ternários, que codificam o encaminhamento de informação das subscrições pela rede de *brokers*.

- É gerado uma máscara de inicialização de valores ternários, em cada um dos *brokers*, para as ligações utilizadas para encaminhamento de informação. A junção de todas as máscaras de ligações dos *brokers* forma a rede de ligações do sistema.

- Em tempo de correspondência de eventos com subscrições, a máscara de inicialização para um conjunto de ligações, baseada no publicador, é refinada até que um *broker* consiga determinar se deve, ou não, utilizar enviar eventos por cada uma das ligações, isto é, até que todos os valores da máscara sejam “Sim” ou “Não”.

## Anotação da PST

Cada um dos *brokers* do sistema tem uma cópia de todas as subscrições organizada numa PST.

Cada *broker* anota cada um dos nós da PST, com um vector de valores ternários. Estas anotações contêm *n* valores ternários, sendo que cada um representa uma ligação de saída do *broker*. Os valores ternários podem assumir o valor “Sim”, que representa que a procura a chegar aquele nó tem garantia que seguindo aquele nó chegará a um subscritor; “Não”, que significa que uma pesquisa a atingir aquele nó, seguindo aquela ligação, não atingirá algum subscritor; “Talvez”, não existem certezas quanto á utilização daquela ligação.

O processo de anotação é recursivo e inicia-se nos nós folha da PST, que representam as subscrições. A posição *l*, do vector de valores ternários associado a um dos nós folha da PST, assume o valor S se um dos subscritores, correspondente ao nó, é atingível através da ligação que *l* representa, caso contrário assume o valor N. Quando todas as estão anotadas começa o processo de propagação, em que as anotações são propagadas desde os nós folha até ao nó raiz da PST, esta propagação é feita utilizando dois operadores: *Alternative Combine* e *Parallel Combine*.

O operador *Alternative Combine* é utilizado para combinar as anotações dos nós atingíveis pelas ligações que não correspondam a “don’care” \*. *Parallel Combine* é utilizado para combinar o resultado do operador *Alternative Combine* com a anotação do nó atingível através da ligação “don’t care” \*.

O operador *Alternative Combine* escolhe o valor menos significativo, dos valores a combinar, isto é, “Talvez” domina os valores “Sim” e “Não”. O operador *Parallel Combine* escolhe o valor mais liberal, dos valores a combinar, isto é, “Sim” domina “Talvez” e “Talvez” domina “Não”.

|  |  |  |  |
| --- | --- | --- | --- |
| *Alternative* | Sim | Talvez | Não |
| Sim | Y | M | M |
| Talvez | M | M | M |
| Não | M | M | N |

|  |  |  |  |
| --- | --- | --- | --- |
| *Combine* | Sim | Talvez | Não |
| Sim | Y | Y | Y |
| Talvez | Y | M | M |
| Não | Y | M | N |

O cálculo da anotação de um nó é feito utilizando o operador *Alternative Combine* em todos os nós filhos, atingíveis por ligações não “don’t care” \*. Caso não exista uma ligação “don’t care”, é adicionada tendo como nó destino um com o vector de valores ternários todos assumindo o valor “Não”. Finalmente é aplicado o operador *Parallel Combine* entre o resultado final do operador *Alternative Combine* e o vector de valores ternários do nó atingível pela ligação “don’t care” \*.

## Geração da máscara de inicialização

Os *brokers* conhecem a estrutura da rede de *brokers*. A partir deste conhecimento cada *broker* constrói uma tabela de encaminhamento, que mapeia os possíveis destinos das suas ligações.

Assume-se que cada *broker* conhece o conjunto de árvores de ligações, cada publicador usará apenas uma destas árvores. Quando a rede de ligações de *brokers* não tem ciclos, a construção de árvores de ligações é simples. Quando a topologia de ligações de *brokers* não é uma árvore, o esquema de ligações entre *brokers* é mais complexo. No entanto, mesmo nestes casos, existirá um pequeno número de árvores de ligações. Sendo que no pior dos casos, existirá uma árvore de ligações por cada *broker* que tenha ligações com publicadores, e no caso mais prático, em que a topologia de ligações de *brokers* é em árvore, existiram ainda menos árvores de ligações.

A partir dos melhores caminhos e das redes de distribuição, cada *broker*, gera os destinos descendentes possíveis para cada rede de distribuição. Um destino é descendente quando é um descendente do *broker* numa das redes de distribuição. São geradas máscaras de inicialização com um valor ternário para cada ligação do *broker*, uma para cada rede de distribuição, sendo esta geração baseada na análise descrita anteriormente. O valor ternário representante da ligação *l* assume o valor “Talvez”, quando pelo menos um dos destinos alcançável através da ligação *l* é um descendente do *broker*, numa das árvores de distribuição. Quando não é possível atingir algum destino descendente do *broker* através de uma ligação *l*, então o valor ternário correspondente a essa ligação assumirá o valor “Não”. A máscara de inicialização tem como objectivo apenas permitir o encaminhamento de eventos através de ligações do *broker* que tenham destinos descendentes deste, sendo estas ligações marcadas com o valor ternário “Talvez”, podendo ser refinadas para o valor “Sim”, caso o processo de correspondência determine que o evento deve ser entregue a um subscritor atingível por a ligação correspondente.

## Correspondência de eventos

O processo de correspondência de eventos é iniciado quando um evento é publicado e chega a um *broker*. Este processo consiste nos seguintes passos:

- É criada uma máscara que é inicializada com a máscara de sinalização, correspondente à árvore de distribuição do publicador.

- Começando no nó raiz, a máscara é refinada pela anotação em cada um dos nós que o processo de correspondência percorre. Durante o processo de refinamento qualquer valor ternário da máscara que assuma o valor “Talvez” é substituído pelo valor da anotação. Caso a máscara esteja completamente refinada, ou seja, não tenha nenhum valor “Talvez”, a pesquisa termina retornando a máscara refinada.

- Caso a máscara não esteja refinada, continua-se a pesquisa nos nós filhos atingíveis por ligações que estejam de acordo com os valores associados aos atributos do evento a corresponder. É executada uma pesquisa em cada um dos filhos seleccionados, utilizando uma cópia da máscara corrente. No retorno de cada pesquisa, todos os valores ternários “Talvez”, da máscara corrente, são transformados em “Sim”, caso exista na posição correspondente da máscara retornada da pesquisa um valor “Sim”. Após todas as pesquisas nos nós filhos terem sido executadas, as posições da máscara corrente que assumam o valor ternário “Talvez” são afectados com o valor “Não” e esta máscara é retornada.

- A pesquisa sobre o nó raiz termina e envia uma cópia do evento para todas as ligações cuja posição da máscara de valores ternários retornada assuma o valor “Sim”.

## Conclusões

Uma rede de *brokers* a utilizar o protocolo “flooding” sobrecarrega a rede com um número de publicações de eventos significativamente menor que utilizando o protocolo “link matching”, independentemente do número de subscrições. A utilização do protocolo “link matching” tem um maior grau de desempenho quando comparada com o protocolo “flooding”, quando um evento tem como destino uma pequena percentagem dos clientes do sistema. No caso em que os eventos têm de ser distribuído para a maior parte do sistema, a diferença de desempenho entre os dois protocolos discutidos não é muito significativa, uma vez que a maior parte das ligações da rede serão utilizadas para distribuir os eventos no protocolo “link matching”. Estes resultados mostram que o protocolo “link matching” é mais apropriado para distribuir eventos para partes da rede, sendo este o caso dos típicos sistemas que seguem o paradigma publish/subscribe sobre uma WAN.

Casos em a lista de destinos pode crescer para incluir centenas ou milhares de destinos, o algoritmo “match first” torna-se impraticável.

Distribuir um evento para toda a rede e filtrá-lo nos receptores também tem as suas desvantagens. O algoritmo “flooding” mostra que este tipo de abordagem de distribuição de eventos tem problemas de escalabilidade e inabilidade de explorar a localidade. O algoritmo “flooding” é uma boa aproximação da distribuição *broadcast*, uma vez a maior parte das técnicas de destruição em WAN requerem o uso duma série de *routers* ou *bridges* a ligarem ligações LAN.

O “link matching” é um algoritmo de distribuição de eventos baseado numa correspondência distribuída, parcial, de eventos publicados. Este algoritmo permite a distribuição de eventos para um grande número de consumidores distribuídos pela WAN sem colocar uma carga exagerada sobre a rede. Este algoritmo também explora a localidade de subscrições.

Demonstra-se que uma rede de *brokers* a utilizar o algoritmo “link matching” permanece em execução, enquanto que uma rede de *brokers* que utilize o algoritmo “flooding” fica sobrecarregada, este facto deve-se ao maior número de eventos que a rede que usa o “flooding” tem de processar.

## Protótipo

O sistema desenvolvido é constituído por uma rede de *brokers* e clientes, sendo que cada *broker* é consiste nas seguintes camadas: motor de correspondência, protocolos de cliente e *broker*, gestor de conexões e camada de transporte.

De seguida é feita uma descrição de cada uma das camadas de um *broker*:

- Motor de correspondência – Implementa um algoritmo de correspondência e está dividido nos seguintes componentes:

- Gestor de subscrições – Recebe uma subscrição de um cliente, interpreta a expressão da subscrição e adiciona a subscrição à árvore de subscrições.

- Interpretador de eventos – Começa por interpretar o evento recebido, após este passo, descodifica o evento de acordo com as regras do esquema do evento. Finalmente, é utilizado o algoritmo de correspondência de eventos, para determinar os subscritores interessantes no evento descodificado.

- Protocolos de cliente e *broker* – Este componente é constituído por objectos que são suficientemente robustos para tratarem de falhas de conexões, ao mantarem registo de eventos por cliente. Este componente é constituído por outros dois, sendo eles:

- Objecto de protocolo de cliente – Implementa o protocolo *broker* para cliente. O protocolo de cliente entrega os eventos recebidos enquanto o cliente estava desligado, quando este se volta a conectar após uma falha. O registo que armazena estes eventos é limpo periodicamente.

- Objecto de protocolo de *broker* – Implementa o protocolo *broker* para *broker*.

- Gestor de conexões – Gere as conexões para os clientes e os outros *brokers* na rede.

- Camada de transporte – Envia e recebe mensagens de e para clientes e outros *brokers* na rede. A escalabilidade é melhorada ao implementar-se o método de envio de forma assíncrona, para tal é mantida uma pilha de mensagens de saída por conexão. Uma *thread* do *broker* envia uma mensagem ao empilha-la na respectiva pilha de mensagens. Existe um conjunto de *threads* que têm como único objectivo o envio das mensagens das pilhas de mensagens de saída, para o seu destino, utilizando o protocolo de rede.

# Implementações

O Active Networks é um mecanismo para eliminar a forte dependência de arquitecturas de encaminhamento em *standards* da Internet. Este mecanismo permite a inclusão dinâmica de código nos *routers* ou através da substituição de pacotes passives por código activo.

O projecto SwitchWare permite a inclusão dinâmica de código através de carregamento de módulos assinados nos *routers* da rede.

The many faces of publish/subscribe

# Introdução

A internet tem mudado a dimensão dos sistemas distribuídos. Nos dias que correm, os sistemas distribuídos envolvem milhares de entidades, sendo que estas podem estar distribuídas pelo mundo, e as suas localizações e comportamentos podem variar ao longo do tempo de vida do sistema. Estas condições mostram a necessidade de modelos de comunicação mais flexíveis e de sistemas que reflictam a natureza dinâmica e desacoplada das aplicações.

Ligações individuais que sejam ponto a ponto e síncronas, levam a aplicações estáticas e rígidas, e tornam o desenvolvimento de aplicações de larga escala dinâmicas incómodo e pouco elegante. É utilizada infra-estrutura de *middleware* dedicada como a “cola” entre as diferentes entidades destes ambientes de larga escala uma, para reduzir as dificuldades dos arquitectos de aplicações, baseada num esquema de comunicação adequado.

O paradigma de interacção publish/subscribe tem recebido um crescendo de atenção e acredita-se que pode providenciar comunicação desacoplada, sendo esta característica fundamental para sistemas de larga escala.

Os subscritores podem expressar o seu interesse num evento ou num padrão de eventos, sendo subsequentemente notificados de eventos gerados por publicadores, que estejam de acordo com os seus interesses registados.

Um evento é propagado de forma assíncrona para todos os subscritores que tenham registado interesse nele.

A vantagem deste modelo de interacção baseado em eventos consiste no total desacoplamento no tempo, espaço e sincronização entre publicadores e subscritores.

Os sistemas baseados no paradigma publish/subscribe podem ser caracterizados da seguinte forma: topic-based, content-based e type-based.

Produtores publicam informação num bus software( gestor de eventos ) e consumidores subscrevem a informação que desejam receber do bus. Esta informação normalmente é conhecida como evento e o acto da sua entrega é conhecido como notificação.

Os subscritores subscrevem eventos sem saberem a origem dos eventos. As subscrições são armazenadas mas não são propagadas para os publicadores.

O sistema de eventos pode ser visto como um proxy para os subscritores, uma vez que propaga os eventos para os subscritores relevantes. Os subscritores vão ser notificados da ocorrência de todos os eventos que sejam de seu interesse (em caso de falha, alguns eventos podem não ser recebidos). Os publicadores podem anunciar a natureza dos seus eventos futuros. Este tipo de informação pode ser útil para o sistema de eventos se ajustar á circulação de eventos e para os subscritores descobrirem novos tipos de eventos.

O desacoplamento entre os publicadores e subscritores num sistema de eventos pode ser decomposto nas seguintes dimensões:

Desacoplamento no espaço – As entidades que interagem não necessitam de ter conhecimento de uma da outar. Os publicadores publicam eventos através do sistema de eventos e os subscritores recebem estes eventos, indirectamente, através do sistema de eventos. Os publicadores não guardam referências para os subscritores, nem têm noção de quantos subscritores participam nas interacções. Da mesma forma, os subscritores não têm referências para os publicadores, nem têm conhecimento do número de publicadores que participam nas interacções.

Desacoplamento no tempo – As entidades participantes na interacção não necessitam de estar activas em simultâneo para que ocorra uma interacção. O publicador pode publicar um conjunto de eventos enquanto que os subscritores interessados estão desconectados, e o subscritor pode ser notificado de um evento quando o publicador deste está desconectado.

Desacoplamento na sincronização – Os publicadores não são bloqueados enquanto publicam eventos e os subscritores são notificados da ocorrência de eventos de forma assíncrona. A produção e consumo de eventos não ocorrem no fio de execução principal dos publicadores e subscritores, portanto não ocorrem de forma síncrona.

O desacoplamento da produção e consumo de eventos aumenta a escalabilidade ao remover a dependência explícita entre as entidades envolvidas nas interacções. Esta eliminação de dependências reduz a necessidade de coordenação e assim a sincronização entre as entidades, fazendo com que a infra-estrutura de comunicação esteja bem adaptada a ambientes distribuídos, sendo estes assíncronos por natureza, os ambientes móveis são um exemplo destes.

Frankli e Zodnik classificaram sistema de distribuição de eventos de acordo com os seus mecanismos de entrega de dados, sendo estas caracterizações: push vs pull, não periódico vs periódico e um para um vs um para muitos. Os sistemas de informação baseados em push têm sido extensamente estudados. São utilizadas caracterizações semelhantes em engenharia de *software* e modelos de coordenação.

# Paradigmas alternativos de comunicação

# Message Passing

O message passing pode ser visto como o antecessor das interacções distribuídas. Este paradigma representa uma comunicação de baixo nível, na qual os intervenientes comunicam ao enviar e receber mensagens. Embora existam esquemas de interacção complexos a trabalhar sobre este paradigma, raramente é utilizado directamente para desenvolver aplicações distribuídas, uma vez que o endereçamento físico e a seriação de dados, e por vezes a lógica de controlo de comunicação, tornam-se visíveis para a camada aplicacional.

A passagem de mensagens é assíncrona para o produtor enquanto é geralmente síncrona para o consumidor. O produtor e o consumidor estão acoplados no tempo e no espaço, uma vez que têm de estar activos ao mesmo tempo e o receptor de uma mensagem tem de conhecer o emissor desta.

# RPC

Uma das mais utilizadas formas de interacção distribuída é a interacção remota, sendo esta uma extensão da noção de invocação de uma operação para um contexto distribuído. Ao expor as interacções remotas da mesma forma que as interacções locais, o modelo de comunicação RPC e as suas derivações distribuídas facilitam a programação distribuída. Contudo a distribuição não pode ser exposta de forma totalmente opaca à aplicação, caso assim não fosse dar-se ia aso a potenciais falhas que têm de ser tratadas especialmente para este tipo de operações.

O paradigma de comunicação RPC diferencia-se do publish/subscribe em termos de acoplamento, nomeadamente: a natureza síncrona deste paradigma introduz um acoplamento no tempo, sincronização na parte do consumidor, e no espaço, uma vez que os objectos invocadores têm uma referência remota para os objectos a serem invocados.

Foram feitas várias tentativas para remover o acoplamento de sincronização na comunicação remota e evitar o bloqueio da *thread* invocadora, enquanto esta espera pela resposta da invocação remota.

A primeira tentativa passou por providenciar um tipo especial de assincronismo em que as invocações remotas não tinham valor de retorno. O CORBA disponibiliza um tipo especial de modificador de um só sentido, que pode ser utilizado para especificar este tipo de métodos. Esta tentativa leva a invocações com poucas garantias de fiabilidade, uma vez que o emissor não recebe notificações de sucesso ou falha, designando-se este tipo de abordagem como *send-and-forget*.

A segunda variação do paradigma RPC é menos restritiva pois suporta valores de retorno, mas não os disponibiliza directamente à *thread* invocadora. Estes valores são disponibilizados através de um mecanismo que é acedido quando os verdadeiros valores de retorno são necessários. Através desta solução do problema, conhecida como *future* ou *future type message*, a *thread* invocadora pode continuar a processar e requisitar o valor de retorno mais tarde, graças ao mecanismo referido anteriormente.

# Notificações

O paradigma de notificações consiste na separação das invocações remotas em duas invocações assíncronas, esta separação foi feita com o objectivo de conseguir o desacoplamento de sincronização. Neste paradigma as duas invocações assíncronas são sequenciais, sendo que a primeira é enviada pelo cliente para o servidor, transportando os argumentos da invocação e uma referência para o cliente, e a segunda é enviada pelo servidor para o cliente, consistindo numa resposta. Este esquema pode ser facilmente estendido para o servidor passar a fazer várias chamadas de resposta para o cliente.

O paradigma de notificações, onde os subscritores registam o seu interesse directamente nos publicadores, sendo que estes fazem a gestão das subscrições e enviam eventos, corresponde ao famoso padrão de desenho *observer*. Este paradigma é normalmente implementado utilizando invocações assíncronas para conseguir o desacoplamento de sincronização. No entanto, apesar dos publicadores notificarem os subscritores de forma assíncrona, ambos estão acoplados no tempo e no espaço.

A gestão de comunicação é feita pelo publicando, podendo sobrecarrega-lo à medida que o sistema cresce em dimensão.

# Espaços Partilhados

O espaços de memória distribuídos (DSM – distribuited shared memory ) providencia alojamento num sistema distribuído com a acessibilidade dum espaço comum, partilhado através de espaços de endereçamento disjuntos.

Um *tuple space* é composto por uma colecção ordenada de *tuplos*, que são igualmente acessíveis pelos clientes do sistema distribuído. A comunicação entre os clientes do sistema é feita através da inserção e remoção de *tuplos* do *tuple space*. Os *tuples spaces* normalmente disponibilizam três tipos de operações, sendo eles: *out* que cola um *tuplo* no *tuple space*; *in* que lê e remove um *tuplo* do *tuple space*; *read* que lê um *tuplo* do *tuple space*.

O modelo de interacção providenciado pelo paradigma espaços partilhados permite o desacoplamento no tempo e no espaço, uma vez que os produtores e consumidores de *tuplos* permanecem anónimos entre eles.

Uma operação do tipo *in* implementa uma interacção com semântica um de muitos, uma vez que apenas um consumidor lê um *tuplo*, enquanto uma operação do tipo *read* implementa uma interacção com semântica um para muitos, em que um *tuplo* pode ser lido por vários consumidores.

Ao contrário do paradigma *publish/subscribe*, o paradigma de espaços partilhados não providencia o desacoplamento da sincronização pois os consumidores recolhem novos *tuplos* do *tuple space* de forma síncrona. Esta característica limita a escalabilidade do paradigma, uma vez que desta forma os consumidores se sincronizam com os produtores.

Sistemas que seguem este paradigma, como o JavaSpaces, compensam o acoplamento na sincronização com notificações assíncronas.

O sistema Internet Indirection Infrastructure ( I3 ) implementa um conceito semelhante ás notificações assíncronas, chamado *rendezvous*. Este conceito implica que em vez de se realizar o envio de pacotes explicitamente para um destino, cada pacote é associado com um identificador. Este identificador é utilizado pelo receptor para obter o pacote. Este nível de indireção permite o desacoplamento entre o acto de envio e o acto de recepção.

# Message Queuing

Os paradigmas *Message Queuing* e *publish/subscribe* estão ligados, uma vez que sistemas que implementam o paradigma *Message Queuing* normalmente integram alguma forma de interacção *publish/subscribe*.

Aproximações tão centradas em mensagens são chamadas de message-oriented middleware (MOM).

O modelo de interacções das *message queues* tem algumas semelhanças com os *tuple spaces*, uma vez que as *queues* podem ser vistas como espaços globais que são preenchidos pelos publicadores. Do ponto de vista funcional, as *message queues* também possibilitam as temporizações transaccionais e dão garantias de ordem.

Nos sistemas que implementam o paradigma *message* *queuing*, as mensagens são recolhidas pelos consumidores de forma concorrente com a semântica um de muitos, de forma semelhante ao tipo de operação *in* dos *tuple spaces*. Este modelo de interacção é também referenciado como ponto a ponto (PTP – point to point ) *queuing*.

Qual o elemento recolhido da pilha não é determinado pela estrutura do elemento, mas pela ordem com que os elementos são colocados na fila, normalmente seguindo a filosofia FIFO ( first in first out ) ou com um esquema de prioridades.

Os produtores e consumidores estão desacoplados no tempo e no espaço. As *message queues* não disponibilizam desacoplamento no sincronismo, uma vez que os consumidores recolhem mensagens de forma síncrona.

Alguns sistemas que implementam o paradigma *message queuing* disponibilizam suporte limitado para entrega de mensagens de forma assíncrona. No entanto, estes mecanismos assíncronos não são escaláveis para grandes números de consumidores, por causa das interacções adicionais necessárias para garantir a temporização transaccional e as garantias de ordem.

# Comparação de paradigmas de interacção

Os paradigmas de interacção tradicionais diferem do *publish/subscribe* pelas suas limitações em suportarem desacoplamento no espaço, no tempo e na sincronização.

# Variações do paradigma *publish/subscribe*

Os subscritores normalmente estão interessados em eventos particulares ou em padrões de eventos, não estão interessados em todos os eventos.

# Topic-based publish/subscribe

As primeiras versões do paradigma *publish/subscribe* eram baseadas nas noções de tópico ou assunto e foram implementadas por várias soluções industriais, tais como, Altherr, Talarian Skeen e TIBCO.

Os participantes podem publicar eventos e subscrever tópicos particulares que são identificados por palavras-chave.

Subscrever o tópico T pode ser visto como tornar-se membro de um grupo T. Publicar um evento com o tópico T, consiste no *broadcast* desse evento para todos os membros do grupo T.

Na prática, os sistemas *publish/subscribe* baseados em tópicos introduzem uma abstracção programática que mapeia tópicos individuais a canais de comunicação distintos.~

Cada tópico é visto como um serviço de eventos, sendo identificado por um nome único, tendo uma interface que permite publicar eventos no próprio e subscrever eventos associados ao tópico.

A abstracção de tópicos é fácil de entender e força a interoperabilidade entre plataformas ao utilizar apenas *strings* como chaves para dividir o espaço de eventos.

Foram propostos vários melhoramentos ao esquema de interacção orientado ao tópico, sendo que o mais útil consiste no uso de hierarquias para orquestrar tópicos. Quase todos os motores baseados em tópicos oferecem uma forma de endereçamento hierárquico, que permite que os programadores organizem os tópicos de acordo com o confinamento de relações. Uma subscrição feita a um nó da hierarquia implica as subscrições de todos os subtópicos daquele nó. Os nomes dos tópicos são geralmente representados utilizando a notação URL e introduzem uma hierarquia semelhante às redes de notícias USENET. A maior parte dos sistemas permitem que os nomes dos tópicos contenham *wildcards*, esta característica foi introduzida pelo sistema TIBCO Rendezvous. Os *wildcards* nos nomes dos tópicos permitem subscrever e publicar para vários tópicos cujos nomes correspondam a um conjunto de palavras-chave, tal como uma subárvore ou um nível específico da hierarquia.

# Content-based publish/subscribe

Apesar de todas as tentativas de melhoramentos, os sistemas *publish/subscribe* baseados em tópicos representam um esquema estático com expressividade limitada. Os sistemas *publish/subscribe* baseados em conteúdo, ou propriedades, melhoram os esquemas baseados em tópicos ao introduzirem um esquema de subscrições baseado no conteúdo dos eventos. Os eventos não são classificados de acordo com um critério previamente definido, como por exemplo o nome de um tópico, mas de acordo com as propriedades do próprio evento. Estas propriedades podem ser atributos internos das estruturas dos eventos, como acontece no Gryphon, Siena, Elvin e Jedi, ou meta-data associada aos eventos, como acontece no Java Message Service.

Os consumidores subscrevem eventos ao especificarem filtros, utilizando de uma linguagem de subscrição. Estes filtros definem restrições na forma de pares nome – valor de propriedades utilizando operadores de comparação básica, que definem os eventos a considerar. As restrições podem ser combinadas para formar padrões de subscrição. Alguns sistemas, tais como o Cambridge Event Architecture (CEA), também disponibilizam correlação de eventos, nestas os participantes subscrevem combinações de eventos e são notificados quando ocorrem o conjunto de eventos. Os padrões de subscrição são utilizados para identificar os eventos de interesse de um determinado subscritor e propaga-los. É disponibilizada uma variante da operação *subscribe*, possibilitando a passagem de um padrão de subscrição. Este padrão pode ser representado das seguintes maneiras:

- String: Os padrões de subscrição são muitas vezes especificados utilizando gramáticas textuais, armazenadas em *strings*, tais como SQL, Oracle, OMG Default Filter Constraint Language, XPath, ou outras gramáticas proprietárias. A descrição textual, na forma de *string*, é descodificada por o mecanismo apropriado.

- Template object: O sistema JavaSpace, inspirado no sistema de correspondência baseado em *tuplos*, adoptou um mecanismo de correspondência também baseado em *tuplos*. Utilizando este mecanismo, no acto de subscrição de eventos, um participante fornece um objecto *t*, sendo que este objecto indica que o participante está interessado em todos os eventos que estão de acordo com o tipo de *t* e cujos atributos correspondem aos atributos correspondentes de *t*, excepto aqueles que contêm o *wildcard null*.

- Código executável: Os subscritores disponibilizam um capaz de filtrar eventos em tempo de execução. A implementação deste objecto é deixada a cabo do programador da aplicação. Uma aproximação alternativa ao problema, baseada numa biblioteca de filtros implementados utilizando reflecção, foi descrita por Eugester e Guerraoui. Este método de definição de padrões de subscrição não é muito utilizado pois os filtros resultantes são extremamente difíceis de optimizar.

Um sistema baseado no conteúdo dos eventos providencia uma granulosidade de subscrição superior a sistemas baseados em tópicos. Para a mesma funcionalidade ser atingida utilizando tópicos, o subscritor teria de filtrar eventos irrelevantes, ou os tópicos teriam de ser divididos em vários subtópicos. A primeira possibilidade leva a um uso ineficiente de largura de banda, enquanto a segunda resulta num elevado número de tópicos e consequentemente um risco acrescido de eventos redundantes.

# Type-Based publish/subscribe

Os tópicos normalmente agrupam eventos que apresentam aspectos em comum, não só no seu conteúdo, mas também na sua estrutura. Esta observação levou à ideia de substituir o modelo de classificação baseado em nomes de tópicos, por um modelo que filtra os eventos de acordo com o seu tipo.

A noção de espécie de evento é directamente correspondida com o tipo do evento. Esta correspondência permite uma integração entre a linguagem e o *middleware*. Esta característica permite o type safety assegurado em tempo de compilação, uma vez que a abstracção resultante é parametrizada com o tipo dos eventos, resultando em código sem utilização de casts de tipo. Enquanto as abordagens baseadas em *templates*, como a utilizada no JavaSpaces, consideram os tipos dos eventos uma propriedade dinâmica, resultando em interfaces que exigem o cast de tipos explícito.

Os sistemas publish/subscribe baseados em tipo podem ser implementados através de sistemas baseados em conteúdo que utilizem filtragem por membros públicos considerados o tipo do evento, encapsulando estes eventos. Este encapsulamento pode ser conseguido armazenando o tipo do evento em membros privados, que são acedidos através de métodos públicos.

# Comparação entre os vários tipos de sistemas publish/subscribe

A abordagem baseada em tópicos é estática e primitiva mas pode ser implementada de forma eficiente. Por outro lado, a abordagem baseada em conteúdo é altamente expressiva mas requer protocolos sofisticados com custos de execução elevados. Estes custos adicionais levam a que se devam escolher esquemas estáticos em situações em que as possibilidades de “grupos” de eventos sejam limitadas. Como é referenciado por Eugster e Guerraoui, a expressividade pode ser atingida ao aplicarem-se filtros baseados no conteúdo, no contexto de tópicos configurados estaticamente. Esta técnica é utilizada em casos em que os valores possíveis da propriedade em questão não estão dentro de intervalos discretos.

# Pormenores de implementação

# Eventos

Os eventos podem assumir duas formas, sendo estas mensagens ou invocações. As mensagens são eventos entregues a subscritores através de uma operação genérica, por exemplo notificação, enquanto as invocações são eventos que despoletam a execução de operações específicas no subscritor.

## Mensagens

Ao mais baixo nível, quaisquer dados que circulem pela rede são encapsulados em mensagens. A maior parte dos sistemas implementam a notificações através de mensagens, sendo estas explicitamente criadas pelas aplicações.

As mensagens são compostas por um cabeçalho, que contém informação específica à mensagem, num formato genérico, e dados de conteúdo, que contêm informação específica aos utilizadores. O cabeçalho das mensagens tem como campos comuns os seguintes: identificador da mensagem, emissor, prioridade e data de expiração; estes campos podem ser interpretados pelo sistema, ou serem simplesmente informação para os consumidores. O tratamento dos dados de conteúdo duma mensagem varia de sistema para sistema, sendo que existem três tipos de tratamento comuns. O primeiro não faz qualquer interpretação destes dados considerando-os como um *array* de bytes. O segundo tratamento encontrado providencia um conjunto de tipos de mensagem que permitem identificar o conteúdo dos dados de conteúdo contidos nestas. Finalmente, o terceiro tratamento encontrado permite que as mensagens sejam auto descritivas quanto ao seu tipo.

O sistema TIBCO Rendezvous define um formato de mensagem que não tem cabeçalho, permite no entanto que o programador crie a sua própria estrutura de mensagem baseada num conjunto de tipos básicos, que podem ser estruturados hierarquicamente. Sendo que o tipo de mensagem pode ser interrogado em tempo de execução.

Os sistema Distribuited Asynchronous Collections(DAC) e Java Message Service (JMS) suportam objectos mensagem, em que o evento pode ser qualquer objecto Java seriável.

Muitas vezes as mensagens são vistas como registos com vários campos.

## Invocações

Os eventos são diferenciados, a alto nível, em duas categorias, sendo elas, invocações e mensagens. As invocações são direccionadas para um tipo específico de objecto e têm uma semântica bem definida. O sistema assegura que todos os consumidores têm uma interface compatível para processar a invocação. A interface tem o mesmo propósito que um contracto entre o invocador e os invocados.

Os sistemas que disponibilizam interacções baseadas em invocações, disponibilizando múltiplas semânticas e esquemas de endereçamento, são normalmente chamados sistemas de mensagens. Estes sistemas incorporam lógica adicional aos esquemas *publish/subscribe* ou filas de mensagens, por forma a transformar as mensagens de baixo nível em invocações em métodos dos subscritores, sendo que todos têm de ser do mesmo tipo.

Enquanto alguns sistemas tomam em consideração os valores de retorno das invocações, os modelos publish/subscribe baseados em tipos do COM+ e o CORBA Event Service, tipicamente apenas consideram invocações num sentido, não existindo resposta do chamado.

Os produtores invocam operações num objecto intermediário, por exemplo um canal de eventos, sendo que este exibe a mesma interface que os consumidores. O objecto intermediário reencaminha os eventos para todos os consumidores registados.

O COM+ ainda disponibiliza uma forma de filtragem baseada no conteúdo dos eventos, ao possibilitar que sejam especificados valores para os argumentos de invocação para restringir as possíveis invocações.

# Meio de comunicação

A transmissão de dados entre os produtores e os consumidores é da responsabilidade do meio do *middleware*. O meio pode ser classificado de acordo com características como: arquitectura ou garantias que providencia para os dados, tais como persistência e confiabilidade.

## Arquitecturas

### Centralizada

O papel dos sistemas *publish/subscribe* consiste na disponibilização da troca de eventos entre produtores e consumidores de eventos, de forma assíncrona. O assincronismo pode ser implementado ao os produtores enviarem mensagens para uma entidade específica, que os armazena e os encaminha para os consumidores. Esta abordagem é chamada arquitectura centralizada, por causa da entidade central que armazena e encaminha as mensagens. Esta abordagem é utilizada por sistemas de filas, tais como IBM MQSeries e Oracle Avanced Queuing, sendo cada um deles construído sobre uma base de dados centralizada. As aplicações baseadas neste tipo de sistemas têm fortes requisitos em termos de confiabilidade, consistência de dados e suporte transaccional, não necessitando de um elevado desempenho do meio de comunicação.

### Distribuida

O assincronismo também pode ser implementado ao utilizarem-se primitivas de comunicação inteligentes que implementem mecanismos de armazenamento e encaminhamento, tanto nos processos produtores como nos consumidores, para que a comunicação aparente ser assíncrona e anónima para as aplicações, sem a necessidade da utilização de uma entidade intermediária. Esta abordagem é conhecida como arquitectura distribuída, uma vez que não existe uma entidade central no sistema. O sistema TIBCO Rendezvous utiliza esta abordagem, onde nenhum processo age como ponto de estrangulamento de desempenho ou ponto único de falha. Estas arquitecturas estão bem adaptadas para entregas rápidas e eficientes de dados.

### Servidores distribuídos

Existe uma abordagem intermédia, adoptada pelos sistemas Gryhon, Siena e Jedi, que consiste na implementação do serviço de notificações de eventos como uma rede distribuída de servidores. Em contraste com os sistemas completamente descentralizados, esta abordagem alivia a carga dos processos participantes ao utilizar servidores dedicados a executar os protocolos complexos para a persistência, confiabilidade e alta disponibilidade, tal como para a filtragem baseada no conteúdo dos eventos e para a o encaminhamento.

Os servidores de eventos do sistema Jedi estão organizados numa estrutura hierárquica, onde os clientes podem-se ligar a qualquer nó. As subscrições são propagadas no sentido ascendente da árvore de servidores. Esta topologia hierárquica tende a causar uma elevada carga de trabalho nos servidores raiz, e uma falha de um servidor pode desligar toda uma subárvore.

No sistema Gryphon existe um grafo que sumariza os interesses dos subscritores. Este grafo é imposto ao grafo de *brokers* de mensagens, por forma a evitar comparações redundantes. O sistema Siena utiliza encaminhamento das subscrições e da publicação dos eventos para fixar os caminhos das notificações.

Os servidores de eventos mantêm registo de informação relevante para fazer a correspondência de eventos com subscrições de uma forma eficiente.

## Disseminação

Forma como a informação é distribuída.

Os dados podem ser enviados utilizando primitivas de comunicação ponto a ponto, ou utilizando mecanismo multicast implementados por hardware tais como IP multicast. Sendo que a escolha do mecanismo de comunicação a utilizar depende de vários factores, tais como: ambiente de utilização ou arquitectura do sistema.

As abordagens centralizadas, tais como alguns sistemas de filas de mensagens, utilizam primitivas de comunicação ponto a ponto entre os produtores/consumidores e o *broker* centralizado. Estes sistemas focam-se em dar garantias além do alto desempenho de comunicação e escalabilidade.

Os sistemas *publish/subscribe* baseados em tópicos podem beneficiar dos vários estudos feitos sobre comunicação de grupo e dos protocolos resultantes para disseminar eventos para os subscritores.

Para garantir um bom desempenho relativo ao número de mensagens da comunicação, são utilizados os protocolos IP multicast ou um dos muitos protocolos de multicast.

A utilização eficiente de eventos multicast em sistemas publish/subscribe baseados em conteúdo é ainda um problema. Os sistemas Gryphon e Siene utilizam algoritmos que entregam a servidores de rede lógicos, de uma forma em que os eventos são propagados apenas para os servidores que gerem os subscritores interessados nos eventos em questão. O desempenho destes sistemas baseados em disseminação é afectado pelo custo da filtragem dos eventos em cada um dos servidores, que depende do número de subscrições no sistema.

Foram propostos algoritmos de filtragem de dados em sistemas publish/subscribe, que são altamente eficientes e escaláveis.

O problema de agregar subscrições com o objectivo de aumentar a velocidade de filtragem em cada servidor, com o preço de uma pequena perda na precisão, tem sido estudado é estudado por Chan C.Y.

Independentemente das técnicas de filtragem empregues, o encaminhamento de eventos selectivo dos sistemas publish/subscribe baseados em conteúdo dificulta a exploração das primitivas de multicast, ao nível de rede.

# Qualidade de serviço

As garantias que o meio providencia a todas as mensagens variam entre os diferentes sistemas. As características da qualidade de serviço mais consideradas são: persistência, garantias transaccionais e prioridades.

## Persistência

As mensagens podem ser enviadas sem gerar qualquer resposta, assim como podem ser processadas horas após terem sido enviadas. Os intervenientes da comunicação não controlam como as mensagens são transmitidas, nem como são processadas. Assim, o sistema de mensagens tem de providenciar garantias, não só ao nível de confiabilidade, mas também ao nível de durabilidade da informação.

Não é suficiente saber que uma mensagem chegou ao sistema de mensagens entre os produtores e consumidores, também é necessário ter garantias que a mensagem não será perdida em caso de falhas do sistema de mensagens.

A persistência é normalmente implementada nos sistemas publish/subscribe que têm uma arquitectura centralizada e guardam as mensagens até que os consumidores as possam processar.

Os sistemas de filas, tais como IBM MQSeries e o Oracle Advanced Queuing, oferecem persistência utilizando uma base de dados.

Os sistemas publish/subscribe distribuídos não costumam oferecer persistência, uma vez que as mensagens são directamente enviadas pelo produtor para todos os subscritores. Apenas nos casos em que os produtores guardam uma cópia de cada mensagem, os subscritores com falhas podem receber mensagens perdidas, após a recuperação.

O sistema TIBCO Rendezvous oferece uma aproximação mista, em que um processo está receptivo a um conjunto de tópicos, este processo armazena de forma persistente as mensagens destes tópicos e reenvia mensagens perdidas a subscritores em recuperação de falhas.

O sistema Cambridge Event Architecture providencia um repositório de eventos, potencialmente distribuído, para armazenamento e recolha de eventos eficiente. Este repositório fornece mecanismos que facilitam a procura de eventos simples e compostos, permitindo a resposta de sequências de eventos armazenados.

## Proridades

Pode ser desejável ordenar, pela sua prioridade, as mensagens à espera para serem processadas pelo consumidor. Por exemplo, um evento real-time pode requerer uma resposta imediata, como por exemplo notificação de falha, e deve ser processado antes das outras mensagens.

As prioridades afectam as mensagens que estão em trânsito, ou seja, não estão a ser processadas. As prioridades de execução são geridas pelo *scheduler* da aplicação e não pelo sistema de mensagens. Este facto leva a que dois subscritores dos mesmos tópicos, possam processar mensagens por ordens diferentes, uma vez que processam mensagens com ritmos diferentes, apesar dos canais de comunicação serem FIFO ( First In Firt Out ).

As prioridades devem ser consideradas uma qualidade de melhor esforço, ao contrário da persistência.

A maioria dos sistemas de mensagens publish/subscribe, centralizados ou distribuídos, providenciam mecanismos de prioridades, apesar de disponibilizarem números diferentes de prioridades e diferirem na forma como as aplicam.

## Transacções

As transacções são geralmente utilizadas para agrupar múltiplas operações em blocos atómicos, que ou são executados na totalidade, ou não são executados de todo.

Em sistemas de mensagens, as transacções são utilizadas para agrupar mensagens em grupos atómicos, em que ou a totalidade do grupo de mensagens é enviada, ou nenhuma mensagem é enviada. Um caso de utilização de transacções é o seguinte, um produtor publica várias mensagens semanticamente relacionadas e não deseja que os consumidores vejam parcialmente a sequência de mensagens, formando um conjunto inconsistente de mensagens, no caso de falha no envio do conjunto de mensagens. Semelhantemente, uma aplicação vital para o negócio pode desejar consumir uma ou várias mensagens, processá-las e apenas depois fazer commit da transacção. Caso o consumidor falhe antes de fazer o commit da transacção, todas as mensagens envolvidas estarão disponíveis para serem reprocessadas, após a recuperação da aplicação.

A forte integração com bases de dados faz com que os sistemas IBM MQSeries e Oracle Advanced Queuing disponibilizem um leque alargado de mecanismos transaccionais. Os sistemas JMS e TIBCO Rendezvous também disponibilizam suporte para agregação transaccional de mensagens no contexto de uma sessão. O sistema JavaSpaces disponibiliza mecanismos transaccionais simples para garantir a atomicidade da produção e consumo de eventos. Um evento publicado num JavaSpace, no contexto de uma transacção, não é visível fora da transacção, até que esta se faça commit da transacção. Semelhantemente, um evento consumido não é removido do JavaSpace, até que se faça commit da transacção que engloba a operação. Vários eventos podem ser produzidos e consumidos no contexto da mesma transacção.

## Confiabilidade

O desacoplamento da sincronização entre produtores e consumidores de informação torna a implementação de propagação confiável de eventos (entrega garantia) desafiadora.

Os sistemas publish/subscribe centralizados normalmente utilizam canais ponto a ponto confiáveis para comunicar com os publicadores e os consumidores, e mantêm cópias dos eventos em locais de armazenamento estáveis. Sendo que os eventos são entregues de forma confiável a todos os subscritores. Uma falha do broker centralizado de eventos apenas causa um atraso na entrega.

Sistemas baseados numa rede de sobreposição de brokers de eventos normalmente utilizam protocolos confiáveis para propagar eventos para todos os brokers ou para um subconjunto deles. Podem ser utilizados protocolos baseados em comunicação em grupo e mecanismos de multicast confiáveis do nível aplicacional, em sistemas baseados numa rede de sobreposição de brokers de eventos, uma vez que são resistentes a falhas de alguns dos brokers da rede.

Os publicadores e subscritores comunicam com o broker mais próximo, geralmente utilizando canais de comunicação ponto a ponto.

Sistemas que permitem que os publicadores e os subscritores comuniquem directamente, tais como TIBCO Rendezvous, também utilizam protocolos multicast confiáveis simples. Uma vez que os eventos ficam residuais no sistema para subscritores que tenham falhado ou se tenham atrasado (desacoplamento temporal), a entrega garantida tem de ser implementada por um processo dedicado a guardar eventos e reenvia-los para os subscritores que os requisitem.

# Conclusões

O publish/subscribe é um paradigma de interacção distribuída bem adaptado para o desenvolvimento de sistemas escaláveis e desacoplados. Para pesquisar e comparar sistemas distribuídos baseados em eventos, foi introduzida uma classificação baseada em três dimensões de desacoplamento entre produtores e consumidores de informação: tempo, espaço e sincronização.

O descacoplamento é uma propriedade desejável, pois impõe a escalabilidade, permitindo que os participantes operem de forma independente uns dos outros. Ao nível de implementação, contudo, a escalabilidade é um assunto delicado, uma vez que a interacção publih/subscribe pode ser construída sobre várias camadas de comunicação e pode ser facilmente dificultada por uma arquitectura desapropriada, especialmente em casos em que sistemas publish/subscribe são construídos sobre infra-estruturas que não são desenhadas com a escalabilidade em mente.

A escalabilidade é uma propriedade que colide com outras propriedades desejáveis. Por exemplo, a alta expressividade e subscrição selectiva requere uma filtragem e um encaminhamento, complexos e dispendiosos, desta forma a escalabilidade é limitada. De forma semelhante, as garantias de forte confiabilidade envolvem custos significativos, uma vez que os eventos têm de ser registados, e eventos perdidos têm de ser detectados e retransmitidos. Mesmo os protocolos desenvolvidos especialmente para redes de larga escala, tais como o protocolo de envio confiável Reliable Multicast Transport Protocol (RMTP), não escalam bem para grandes números de subscritores, uma vez que é gerado uma grande quantidade de trafego para as mensagens de confirmação.

Os protocolos probabilísticos têm sido alvo de uma atenção crescente, uma vez que este tipo de protocolo combina a natureza o desacoplamento e a comunicação baseada em participantes dos sistemas publish/subscribe. Em vez de serem disponibilizados protocolos determinísticos (garantidos) confiáveis, são disponibilizados protocolos multicast probabilísticos que garantem, com uma elevada probabilidade, que um determinado evento chega a todos os subscritores. A integração deste tipo de protocolos probabilísticos em sistemas publish/subscribe baseados em conteúdos permanece uma dificuldade.

Enquanto programar abstracções para publish/subscribe pode ser esmagador, desenhar algoritmos apropriados para serem utilizados neste tipo de sistemas em larga escala é ainda um problema em aberto. As contrapartidas têm de ser tratadas para lidarem com a escalabilidade, expressividade e a qualidade de serviço.