

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Electrónica e Telecomunicações e de Computadores

**Sistemas de Informação**

**Message Integration Bus**

**Igor André Gaspar Cândido**

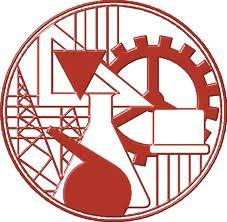
(Licenciado)

DISSERTAÇÃO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA DE COMPUTADORES

Orientador:

Fernando Miguel Carvalho

Fevereiro



INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Electrónica e Telecomunicações e de Computadores

**Sistemas de Informação**

**Message Integration Bus**

**Igor André Gaspar Cândido**

(Licenciado)

DISSERTAÇÃO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA DE COMPUTADORES

O aluno:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Igor André Gaspar Cândido)

Orientador:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Fernando Miguel Carvalho)

Índice

[Resultados obrigatórios 5](#_Toc284608191)

[Resultados opcionais 5](#_Toc284608192)

[Planeamento do projecto 5](#_Toc284608193)

[Palavras-Chave 6](#_Toc284608194)

[Recursos necessários 6](#_Toc284608195)

[Introdução 6](#_Toc284608196)

[Pesquisa 6](#_Toc284608197)

[Variações do paradigma publish/subscribe 6](#_Toc284608198)

[Sistemas publish/subscribe baseados em tópicos 6](#_Toc284608199)

[Sistemas publish/subscribe baseados no conteúdo 7](#_Toc284608200)

[Sistemas publish/subscribe baseados em tipos 7](#_Toc284608201)

[Comparação entre os vários tipos de sistemas publish/subscribe 8](#_Toc284608202)

[Diferentes paradigmas de interacção 8](#_Toc284608203)

[Message Passing 8](#_Toc284608204)

[RPC 8](#_Toc284608205)

[Notificações 8](#_Toc284608206)

[Espaços Partilhados 9](#_Toc284608207)

[Message Queuing 9](#_Toc284608208)

[Comparação de paradigmas de interacção 10](#_Toc284608209)

[Algoritmos de comunicação multicast 10](#_Toc284608210)

[Protocolo de comunicação multicast sugerido em (1) 11](#_Toc284608211)

[Algoritmo de correspondência 11](#_Toc284608212)

[Link Matching 11](#_Toc284608213)

[Protocolos 11](#_Toc284608214)

[Rest MS 12](#_Toc284608215)

[AMQP 12](#_Toc284608216)

[Conceitos envolvidos na implementação de um Message-Oriented Middleware 13](#_Toc284608217)

[Eventos 14](#_Toc284608218)

[Meio de comunicação 14](#_Toc284608219)

[Soluções existentes 17](#_Toc284608220)

[Apache QPID 17](#_Toc284608221)

[BizTalk 18](#_Toc284608222)

[Arquitectura 18](#_Toc284608223)

[Adaptadores 20](#_Toc284608224)

[Distributed PublishSubscribe (PubSub) Event System 20](#_Toc284608225)

[Subscrições 20](#_Toc284608226)

[Topologia 20](#_Toc284608227)

[Encaminhamento de eventos 21](#_Toc284608228)

[Estrutura do sistema 21](#_Toc284608229)

[Bibliografia 23](#_Toc284608230)

Espera-se como resultado final deste trabalho, uma implementação de um *Message-Oriented Middleware*. Este deverá ter uma arquitectura de servidores distribuídos, utilizando um algoritmo de comunicação *multicast* para tratar quer das subscrições, quer do encaminhamento de eventos. Deverá ser disponibilizada a garantia de entrega de mensagens e processamento transaccional. É também espectável que sejam produzidos adaptadores para as linguagens **C#** e **Java**.

# Resultados obrigatórios

- Implementação de servidor de eventos

- Implementação de um algoritmo de comunicação *multicast* de eventos

- Implementação do protocolo **AMQP**

- Implementação de adaptadores para **C#** e **Java**

- Implementação de mecanismos de garantia de entrega de mensagens e processamento transaccional

# Resultados opcionais

- Implementação de adaptadores para **Python** e **Ruby**

- Implementação de protocolo **Rest MS**

- *Framework* para testes de desempenho de sistemas de mensagens

# Planeamento do projecto

- Pontos de controlo

|  |  |
| --- | --- |
| **Data** | **Designação** |
| 27 de Março de 2011 | Fim da fase de investigação sobre o contexto do problema |
| 10 de Abril | Arquitectura geral da solução |
| 12 de Abril | Arquitectura de servidores de eventos |
| 22 de Maio de 2011 | Implementação base de servidores de eventos |
| 29 de Maio de 2011 | Implementação de adaptadores para **C#** e **Java** |
| 26 de Junho de 2011 | Implementação de algoritmo *multicast* e protocolo **AMQP** |
| 24 de Julho de 2011 | Implementação de mecanismos de garantia de entrega mensagens e processamento transaccional |

# Palavras-Chave

Integração de Sistemas, Eventos, Publish/Subscribe, Message-Oriented Middleware

# Recursos necessários

Não se perspectiva a necessidade de recursos adicionais ao computador pessoal do aluno.

# Introdução

O cenário empresarial actual define que as aplicações não podem operar isoladamente, sendo que existe a necessidade de ligar várias aplicações, para que possa existir cooperação e desta forma formarem-se processos de negócio mais ricos.

As diferentes aplicações podem ser construídas em diferentes momentos no tempo, e por vezes, sobre diferentes plataformas e ainda sem conhecimento mútuo. Sendo que a determinada altura no tempo se determina que duas aplicações, sem qualquer ligação, necessitam de trocar informação. Uma possível solução para este problema de integração seria alterar as aplicações, criar um protocolo de comunicação e depois de realizada a implementação das novas funcionalidades, voltar a testar a aplicação. Esta solução tem o problema de ser demorada e ser dispendiosa.

A melhor solução, em termos financeiros e temporais, é utilizar-se uma peça de software que permita a interligação de aplicações e possa ser utilizada de forma genérica.

Um Message-Oriented Middleware (MOM) é uma peça de software que, utilizando mensagens, permite a comunicação entre aplicações. Este tipo de solução não é desenhado à medida e é suficientemente configurável para servir as necessidades de interligação de diversas aplicações. Normalmente este tipo de solução disponibiliza garantia de entrega de mensagens e processamento transaccional. São também focados aspectos como o desacoplamento no tempo, no espaço e na sincronização.

# Pesquisa

Durante estes primeiros meses de trabalho, foi feita pesquisa sobre paradigmas de comunicação, o paradigma de comunicação *publish/subscribe*, algoritmos de comunicação multicast, pormenores a ter em conta na implementação de uma solução de integração de sistemas e actuais soluções de integração de sistemas.

O paradigma *publish/subscribe* foi um ponto de foco neste estudo, por ser considerado o paradigma de comunicação de eleição, em termos de comunicações genéricas.

## 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. As diferentes formas de especificar quais os eventos em que os subscritores estão interessados levaram a vários esquemas de subscrição.

### Sistemas publish/subscribe baseados em tópicos

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**. Nestas versões do paradigma, 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 do 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 conjuntos de caracteres como chaves para dividir o espaço de eventos.

### Sistemas publish/subscribe baseados no conteúdo

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 ou meta-data associada aos eventos.

Os consumidores subscrevem eventos ao especificarem filtros, utilizando uma linguagem de subscrição. Estes filtros definem restrições na forma de pares nome – valor de propriedades, utilizando operadores de comparação básica para definir quais os eventos a considerar. As restrições podem ser combinadas para formar padrões de subscrição. 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 subscrição, possibilitando a passagem de um padrão de subscrição.

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.

### Sistemas publish/subscribe baseados em tipos

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 a verificação de tipo assegurada em tempo de compilação, uma vez que a abstracção resultante é parametrizada com o tipo dos eventos, resultando em código sem utilização de conversão 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 a conversão de tipos explícita.

Os sistemas *publish/subscribe* baseados em tipo podem ser implementados através de sistemas baseados em conteúdo que utilizem filtragem por membros públicos que sejam considerados o tipo do evento.

### 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. 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.

## Diferentes paradigmas de interacção

Os padrões *Messaga Passing*, invocações remotas, notificações, espaços partilhados e filas de mensagens, são paradigmas de comunicações alternativos ao *publish/subscribe*. Estes paradigmas situam-se a níveis de abstracções diferentes.

### 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 enviando e recebendo mensagens

A passagem de mensagens é assíncrona para o produtor, enquanto geralmente é 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. O receptor de uma mensagem tem de conhecer o emissor desta.

### RPC

Uma das formas de interacção distribuída mais utilizadas é a 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.

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.

### Notificações

O paradigma de notificações consiste na separação das invocações remotas em duas invocações assíncronas, esta separação é 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 publicador, podendo sobrecarrega-lo à medida que o sistema cresce em dimensão.

### Espaços Partilhados

O espaço de memória distribuído (DSM – *distribuited shared memory*) providencia alojamento num sistema distribuído com a acessibilidade dum espaço comum, sendo que este é partilhado através de espaços de endereçamento disjuntos. Um exemplo de espaço de memória distribuído é um *tuple space*.

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

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.

### 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 designadas de *message-oriented middleware* (MOM).

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

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

A escolha do elemento a ser recolhido da pilha não é baseada na estrutura do elemento, mas na ordem com que os elementos são colocados na pilha, normalmente seguindo a filosofia FIFO ( *first in first out* ) ou baseada num 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 dar garantias transaccionais e 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.

## Algoritmos de comunicação multicast

O paradigma *publish/subscribe* baseado em conteúdos é o paradigma mais poderoso, no entanto as suas implementações eficientes e escaláveis acartam problemas difíceis de contornar. Uma implementação eficiente deste paradigma tem de resolver dois problemas chave:

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

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

- O sistema *publish/subscribe* é geograficamente distribuído e os servidores 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 servidores de eventos que 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 à 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 protocolo “Flooding” sobrecarrega uma rede com um número de publicações bastante menor que o “Match First”.

## Protocolo de comunicação multicast sugerido em (1)

O documento An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems (1) caracteriza um protocolo de comunicação *multicast*, definido pelos autores do mesmo, que visa melhorar a comunicação *multicast* entre os clientes e servidores de eventos. Este protocolo é constituído por um algoritmo de correspondência de eventos a subscrições e por uma estratégia de distribuição de eventos pela rede de servidores de eventos.

### 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.

### Link Matching

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

## Protocolos

Nesta secção do documento são apresentados os protocolos, já estudados, que são frequentemente utilizados em soluções **Message-Oriented Middleware**.

### Rest MS

O **RestMS** é um serviço de mensagens **RESTful** que providencia comunicação assíncrona através duma interface **REST**, utilizando HTTP/HTTPS. O **RestMS** assenta sobre **RestTL** e é extensível através de especificações de perfis.

#### Domains

Os *domains* são *namespaces* para *pipes* e *feeds*. Um servidor pode ter vários *domains* e cada um deles pode ter credenciais de segurança diferentes. Um *domain* contém os *profiles* que nele podem ser utilizados

#### Profiles

Contém um conjunto de semânticas que o servidor implementa, estas semânticas podem definir o tipo de *feeds*, *pipes*, códigos de resposta, utilizações dos objectos definidos.

#### Feeds

As *feeds* são o ponto de entrada de mensagens no sistema. Vários clientes podem escrever mensagens numa *feed*, a ordem das *feeds* deve ser mantida, relativamente a cada cliente. As *feeds* podem ser criadas dinamicamente por clientes.

#### Pipes

Os *pipes* consistem na forma de recepção de mensagens dos clientes. Sobre os *pipes* apenas se podem realizar leituras e são ordenados para um leitor. São criados de forma dinâmica e são privados para os clientes que os criam. O servidor pode limpar *pipes* que não sejam utilizados ou que estejam a ficar sobrecarregados de mensagens.

#### Joins

Os *joins* definem o encaminhamento que deve ser feito das mensagens recebidas nas *feeds* para os *pipes* subscritos por clientes. São criados de forma dinâmica e devem sempre ser privados. A criação de *joins* deve ser feita sempre para o URI do *pipe* respectivo. Se a *feed* ou o *pipe* a que o *join* diz respeito forem destruídos, o *join* também o será.

### AMQP

Os sistemas de mensagens das empresas permitem que os programas comuniquem através da troca de mensagens, tal como as pessoas comunicam ao trocarem *emails*. Mas ao contrário dos *emails*, os sistemas de mensagens das empresas providenciam garantias de entrega, velocidade, segurança e liberdade de spam. Até ao surgimento do **AMQP**, não havia nenhum *standard* aberto para sistemas de mensagens de empresas, portanto os programadores tinham de escrever os seus próprios, ou utilizar sistemas proprietários que são dispendiosos.

O **AMQP** é o primeiro sistema aberto para mensagens de empresas. Foi desenhando para suportar trocas de mensagens para qualquer aplicação distribuída. O encaminhamento pode ser flexivelmente configurável, facilmente suporta paradigmas de mensagens comuns, tais como: ponto a ponto, *fanout*, *publish-subscribe* e pedido-resposta.

O **AMQP** define não só a interface, mas também o comportamento dos clientes e servidores, por forma a possibilitar uma verdadeira interoperabilidade entre diferentes fabricantes.

Define o formato das mensagens a serem enviadas através do canal de comunicação.

Opera segundo dois ideias:

- Message Queues: Podem dar diferentes tipos de qualidade de serviço, exemplo: garantia de persistência de dados elevada, ou armazenamento apenas em RAM. Cliente pode especificar qual a qualidade de serviço que deseja, no acto da criação da *queue*.

- Exchanges: Encaminham as mensagens para *queues*. Têm semânticas: one-to-one, one-to-many, one-to-one-of-many.

- Bindings: Os bindings são conjuntos de regras que definem o comportamento dos Exchanges, podendo definir que todas as mensagens são encaminhadas para uma queue, ou, que o encaminhamento é baseado no conteúdo das mensagens.

As mensagens circulam entre servidores de eventos e clientes através de métodos assíncronos e síncronos.

Entidades relevantes, do ponto de vista de comunicação:

- Servidor de eventos: Entidade a que os utilizadores se conectam. Pode ser distribuído por várias replicadas organizadas em *cluster* (rede de brokers).

- Utilizador: Autenticações de acesso ao *middleware*

- Conexão: Ligação física, exemplo: **TCP/IP**

- Canal: Ligação lógica associada a uma conexão, através desta consegue-se manter estado numa comunicação.

## Conceitos envolvidos na implementação de um Message-Oriented Middleware

Nesta secção são discutidos alguns aspectos da implementação de um sistema de integração de aplicações. Entre estes aspectos estão: os eventos, o meio e a qualidade de serviços. Esta discussão é guiada pelas preocupações causadas pela flexibilidade, confiabilidade, escalabilidade e desempenho.

### 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.

#### Invocações

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.

### 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

A arquitectura de um sistema de eventos caracteriza a topologia e a delegação de responsabilidades das diferentes entidades do sistema.

##### 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. 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.

##### Distribuída

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. Estas arquitecturas estão bem adaptadas para entregas rápidas e eficientes de dados.

##### Servidores distribuídos

Existe uma abordagem intermédia, adoptada por diversos sistemas actuais, 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 o encaminhamento.

Os servidores de eventos do sistema *Distributed PublishSubscribe Event System* 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.

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

#### Disseminação

A disseminação caracteriza a forma como a informação é distribuída. Sendo que, os dados podem ser enviados utilizando primitivas de comunicação ponto a ponto, ou utilizando mecanismos *multicast*, 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 servidor de eventos 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. 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.

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 *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.

##### Prioridades

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.

##### Confiabilidade

O desacoplamento da sincronização entre produtores e consumidores de informação torna a implementação de propagação confiável de eventos (entrega garantida) 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 servidor centralizado de eventos apenas causa um atraso na entrega.

Sistemas baseados numa rede de sobreposição de servidores de eventos normalmente utilizam protocolos confiáveis para propagar eventos para todos os servidores 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 servidores de eventos, uma vez que estes são resistentes a falhas de alguns dos servidores da rede.

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

# Soluções existentes

Neste capítulo é feita uma descrição de algumas soluções, para integração de software empresarial, a serem utilizadas no mercado.

## Apache QPID

O Apache **Qpid** é um sistema de mensagens de empresas multiplataforma que implementa o protocolo **AMQP**, providenciando servidores de eventos escritos em **C++** e **Java**, juntamente com clientes para **C++,** **Java JMS**, **.Net**, **Python** e **Ruby**.

Ao implementar a ultima especificação **AMQP**, o **Qpid** providencia gestão de transacções, filas de mensagens, distribuição mensagens, segurança, *clustering*, federação e suporte heterogéneo para multiplataforma.

O **Apache** **Qpid** é um esforço conjunto da **Red Hat**, **Iona** e outros para construir software que utilize **AMQP**.

O **Apache Qpid** é constituído pelos seguintes componentes: SocketIo Thread Pool, Event Processor Thread Pool e Message Pump Thread Pool, sendo as suas funções as seguintes.

SocketIo Thread Pool:

* Ler informação dos Sockets
* Asynchronous Read Filter faz passar as mensagens pelo FilterChain para serem processadas como eventos, num outro protocolo.

Event Processor Thread Pool

* Processar eventos de filas associadas a sessões
* Escrever eventos em *buffers* associados a sessões
* Descodificar eventos
* Encaminhar eventos através de “Exchange”
* Criar “*frame*” e cabeçalho de conteúdo
* Escrever evento no *buffer* de sessão (envio directo) ( Pode ser feita uma conversão de protocolo )
* Colocar evento numa fila para ser posteriormente entregue

Message Pump Thread Pool

* Armazenamento de eventos, enquanto a fila está em modo de persistência

## BizTalk

Nenhuma aplicação é uma “ilha”. Quer queiramos, ou não, ligar sistemas tornou-se a norma, chegando ao ponto de a ligação de várias peças de software tornar-se mais de que simples trocas de bytes. À medida que as organizações se movem na direcção de uma realidade orientada a serviços, o verdadeiro objectivo (criar processos de negócio eficientes que consigam unir diferentes sistemas num todo coerente) torna-se mais atingível.

O **BizTalk** suporta este objectivo, ao permitir a conexão entre diferentes peças de software, permitindo ainda a criação e modificação, gráfica, da lógica de processamento desse software. O **BizTalk** também permite que os processos em execução sejam monitorizados, se interaja com os parceiros de negócio e se realizem outras tarefas orientadas ao negócio

O **BizTalk Server** é a solução de integração e conectividade da **Microsoft**. Providencia uma solução que permite que organizações se conectem, mais facilmente, a diferentes sistemas. Inclui mais de 25 adaptadores para diferentes plataformas e uma infra-estrutura de mensagens robusta.

### Arquitectura

O **BizTalk** integra uma gama de tecnologias para cumprir as suas funções, sendo que a Ilustração 1 mostra os principais componentes deste.

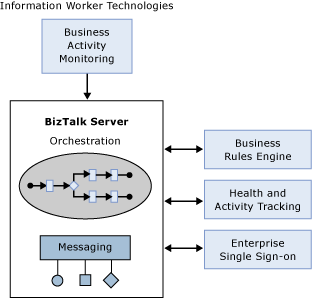


Ilustração 1 - Arquitectura de BizTalk Server, retirada de http://msdn.microsoft.com/en-us/library/aa547058(v=BTS.70).aspx

Tal como a ilustração sugere, o coração do **BizTalk** é o **BizTalk Server Engine**. Este é dividido em dois componentes:

- Um componente de mensagens que providencia a habilidade de comunicar com diversos componentes de software. Ao utilizar adaptadores, para os diferentes tipos de comunicação, o **BizTalk Server Engine** consegue suportar vários protocolos e formatos de dados, incluindo Web *services*.

- Suporte para criar e executar orquestrações, que são processos definidos graficamente. As orquestrações são construídas sobre os componentes de mensagens do **BizTalk Server Engine** e implementam a lógica que coordena toda, ou parte, da lógica de negócio.

Além do **BizTalk Server Engine**, podem ser utilizados outros componentes do **BizTalk**, tais como:

- Um motor de regras de negócio que avalia conjuntos de regras complexas

- Um ponto central que permite aos programadores e administradores monitorizarem e gerirem o **BizTalk Server Engine**, assim como as orquestrações que este executa.

- Um ponto mecanismo de identificação único à empresa que disponibiliza a possibilidade de mapear a informação de autenticação entre sistemas Windows e não Windows.

O **BizTalk Server** ainda incluí uma ferramenta de monitorização de actividades de negócio (BAM), que permite a monitorização de processos de negócio, sendo que a informação disponibilizada contextualizada ao negócio em vez de termos técnicos.

#### Comunicação

O **BizTalk** é construído sobre uma arquitectura *publish/subscribe*, baseada no conteúdo dos eventos. As mensagens são publicadas para o sistema e são recebidas por um ou mais subscritores activos.

O **BizTalk** torna o processamento de mensagens seguro, ao serializa-las para uma base de dados, enquanto espera por eventos externos, prevenindo-se assim a perda de dados. Esta arquitectura liga o **BizTalk** ao **Microsoft SQL Server**.

### Adaptadores

O **BizTalk** utiliza adaptadores para comunicações com diferentes protocolos e produtos de software específicos, como o SharePoint. Alguns dos adaptares incluídos na versão Server 2006 são: **Base EDI**(Covast), File, http, FTP, SMTP, POP3, SOAP, SQL, **MSMQ**, Web Services Enhancements(WSE) 2.2 e os adaptadores **Windows SharePoint Services** (WSS). O adaptador **WCF** foi adicionado na versão 2006 R2.

## Distributed PublishSubscribe (PubSub) Event System

O sistema de eventos Distributed PublishSubscribe (PubSub) Event System é sistema de eventos *publish/subscribe* distribuído. É distribuído por a publicação e subscrição de eventos poder ser intra-máquina ou inter-máquina. Os publicadores e subscritores não necessitam de saber da existência uns dos outros, sendo contudo possível que um publicador possa escutar os eventos de subscrição, para determinar se existem subscritores interessados nos seus eventos, antes da sua publicação.

### Subscrições

Os eventos são tipificados através da propriedade **EventType**. Esta propriedade é um *GUID* que permite que sejam definidos tipos eventos sem que exista a preocupação de conflitos com tipos de evento criados por outras utilizações. As aplicações subscrevem tipos de evento, sendo que existe a opção de subscrição de múltiplos tipos de eventos.

As subscrições têm associados identificadores de subscrição. Na criação de uma subscrição é criado um evento de subscrição, que é publicado pela rede, contendo o respectivo identificador de subscrição e tipo de evento.

### Topologia

As máquinas na rede de eventos estão organizadas em hierarquia. Cada máquina tem conhecimento do seu pai e cada pai tem conhecimento dos seus filhos, à medida que estes se conectam. As comunicações entre máquinas são feitas através de **TCP**. A Ilustração 2 mostra um exemplo desta organização.



Ilustração 2 - Imagem retirada de WSP Event System

Caso os eventos sejam enviados entre irmãos, o encaminhamento é feito através da máquina pai, até ao nó destino. Independentemente do número de subscritores atingíveis através do nó pai, um evento será enviado apenas uma vez da máquina filho para a máquina pai, ou de um pai para um filho. O pai encaminha o evento para o seu pai e/ou seus filhos, caso seja apropriado.

### Encaminhamento de eventos

As tabelas de encaminhado são construídas e mantidas dinamicamente, de forma semelhante a como os *routers* de rede mantêm as suas tabelas para *multicast* **IP**. As subscrições são tratadas ao serem enviados eventos através da rede. Cada nó utiliza os eventos de subscrição para manter as suas tabelas de encaminhamento. Quando são publicados eventos numa máquina ou encaminhados para uma máquina, o sistema de eventos encaminha-os através da rede de ligações físicas onde existem subscrições.

### Estrutura do sistema

A arquitectura dos servidor de eventos é composta por: **Event Router**, **Publishing Applications**, **Subscribing Applications** e **Shared Memory**. A Ilustração 3 mostra a mostra as ligações entre estes componentes.



Ilustração 3 - Imagem retirada de WSP Event System

O **Receiver** e **Fowarder** são as interfaces de comunicação para interagir com as máquinas pai e filho. Quando um evento chega ao **Receiver**, este envia o evento para o RePublisher, que publica o evento no sistema local. Publicar o evento, neste sentido, significa colocar o evento no *buffer* de memória partilhada. Os subscritores interessados copiam o evento da memória partilhada, tal como o **Listener**. Este último reencaminha o evento para outros componentes, caso seja apropriado fazê-lo. Caso um evento seja de subscrição, o **Listener** envia-o para **Subscriptions**, para que sejam actualizadas as tabelas de encaminhamento. Se o tipo de evento for para ser persistido, o **Listener** encaminha o evento para o **Persister**. Quando as tabelas de encaminhado indicam que o evento precisa de ser encaminhado, o **Listener** encaminha o evento para o **Forwarder**, que por sua vez envia o evento para outra máquina.

As aplicações de publicação limitam-se a colocar eventos no *buffer* de memória partilhada. Quando um publicador e subscritor estão na mesma máquina, o evento é colocado na memória partilhada, pelo publicador, e recolhido desta, pelo subscritor. Nesta situação nunca existem custos acrescidos do evento passar pelo servidor de eventos. Uma vez que podem existir N subscritores à escuta de eventos colocados na memória partilhada, é possível que todos os N subscritores recolham o evento ao mesmo tempo e de forma eficiente.

# Bibliografia

1. *An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems.* **Gurudth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao.**

2. *Event-Based Middleware: A New Paradigm for Wide-Area Distributed Systems?* **Pietzuch, Peter R.**

3. *The Evolution of Publish/subscribe Comunication Systems.* **Roberto Baldoni, Mariangela Contenti, Antonio Virgillito.**

4. *The Many Faces of Publish/Subscribe.* **Patrick TH. Eugster, Pascal A. Felber, Rachid Guerraoui, Anne-Marie Kermarrec.**

5. *Web Solutions Platform Event System: A Distributed Publish/Subscribe Event System.* **Hamilton, Keith.**

6. Introducing BizTalk Server 2010. *MSDN.* [Online] http://msdn.microsoft.com/en-us/library/aa547058(v=BTS.70).aspx.

7. *Apache Qpid.* [Online] http://qpid.apache.org/.