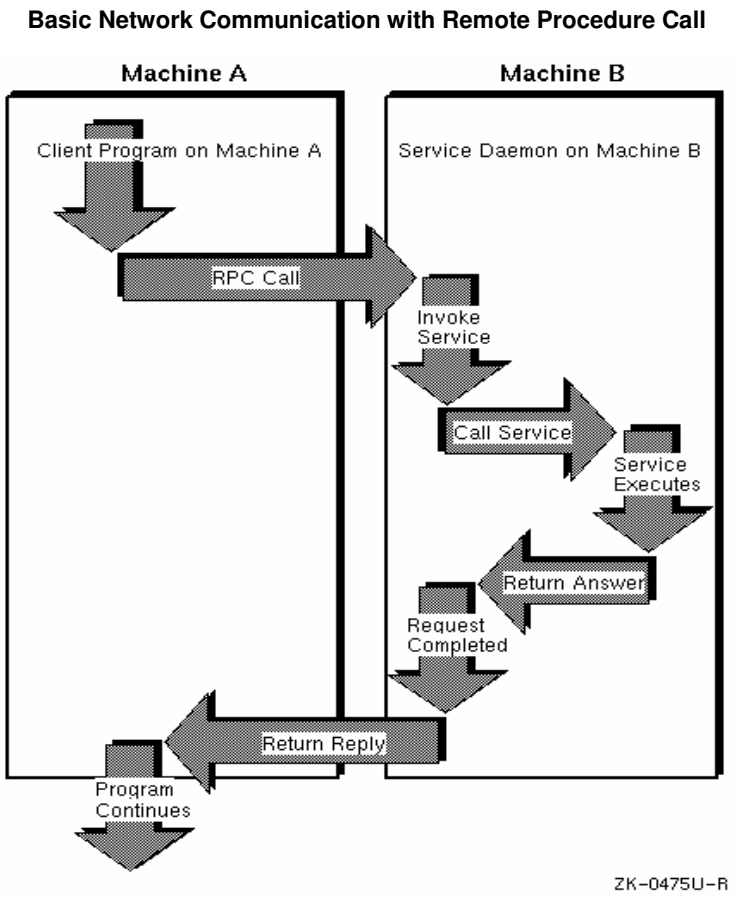
# Message Oriented Middleware & Message Brokers

Tecnologias de Middleware

Pedro Miguel Martins Nunes
Curso de Especialização em Informática 06/07
Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa

# Message Oriented Middleware (MOM)

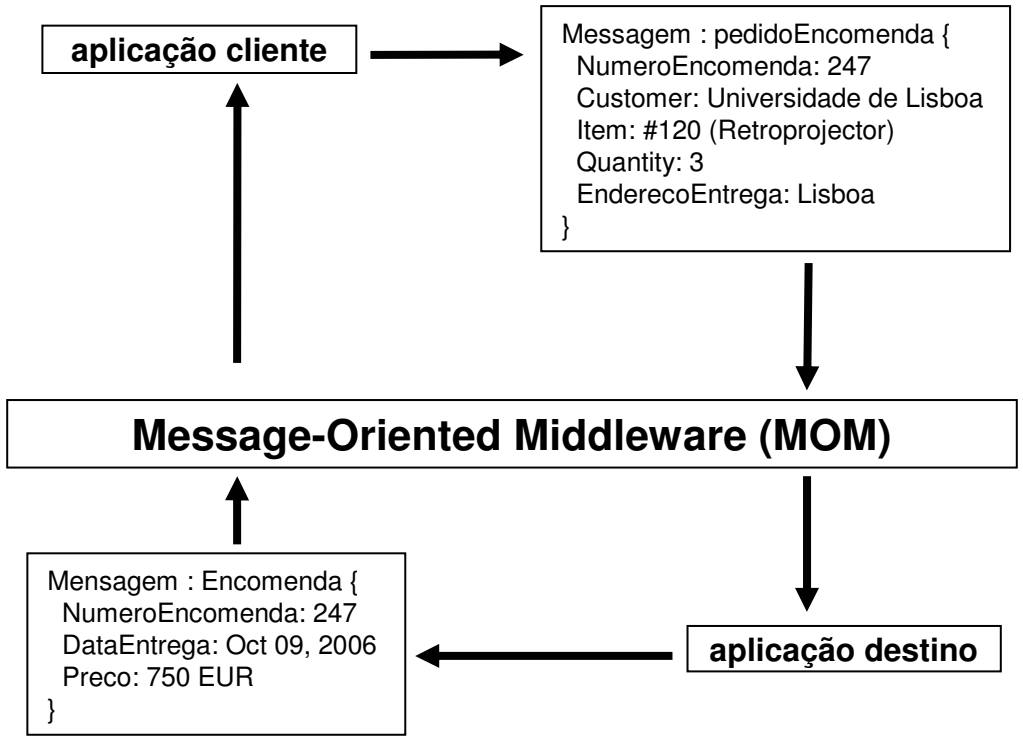
- Porque surgiu?
  - Limitações dos sistemas baseados em RPC
    - comunicação transiente
    - comunicação síncrona



# Message Oriented Middleware (MOM)

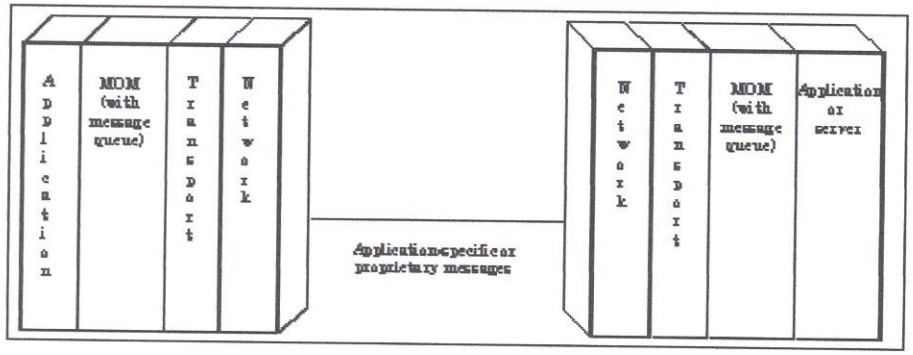
- O que é?
  - Tipo de middleware que assenta num sistema de comunicação assíncrono e persistente de troca de mensagens
  - Mensagem:
    - conjunto estruturado de informação caracterizado por
      - Tipo
      - Parâmetros

# Message Oriented Middleware (MOM)



# Message Oriented Middleware (MOM)

- No que consiste?
  - Software que reside
    - Aplicação Cliente
    - Aplicação Destino
  - ...e que possibilita
    - chamadas assíncronas entre aplicações
  - Filas de mensagens
    - Permitem armazenamento se aplicação destino não está disponível

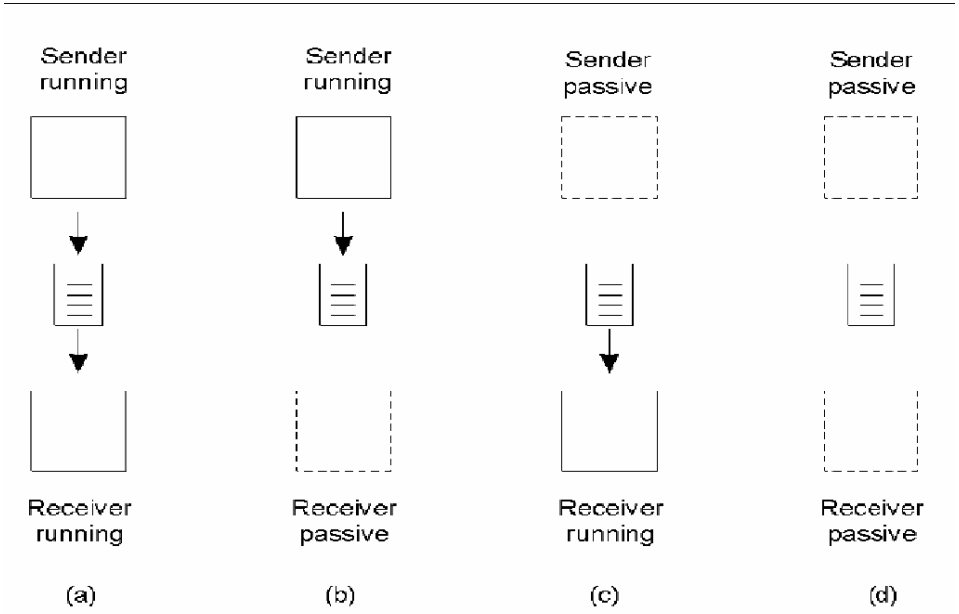


# Message Oriented Middleware (MOM)

- Como funciona?
  - Modelo de Filas de Mensagens
    - Aplicações comunicam através da inserção de mensagens em filas específicas
    - Cada aplicação tem a sua fila privada, para a qual outras aplicações podem enviar mensagens
    - A entrega das mensagens é assegurada pelo MOM
      - Mensagem é mantida na fila da aplicação destino, se esta está indisponível para receber mensagens
      - Cliente apenas tem a garantia que a mensagem será inserida na fila da aplicação destino

# Message Oriented Middleware (MOM)

- Comunicações usando filas
  - Independência entre aplicações

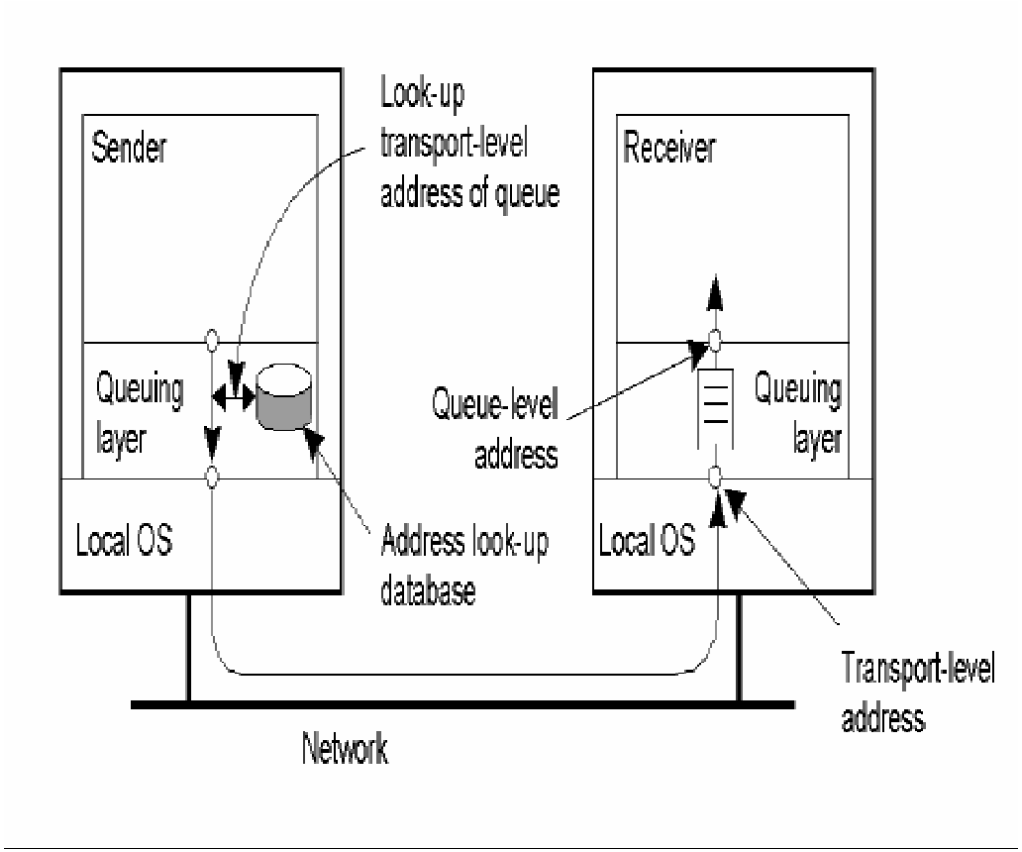


# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Arquitectura
    - Mensagens são colocadas numa fila origem
      - Cada mensagem especifica a fila destino para a qual deve ser transferida (fila destino)
      - É da responsabilidade do sistema a alocação destas filas e assegurar a transferência de mensagens entre elas
    - Mensagens são retiradas da fila destino pela aplicação destino

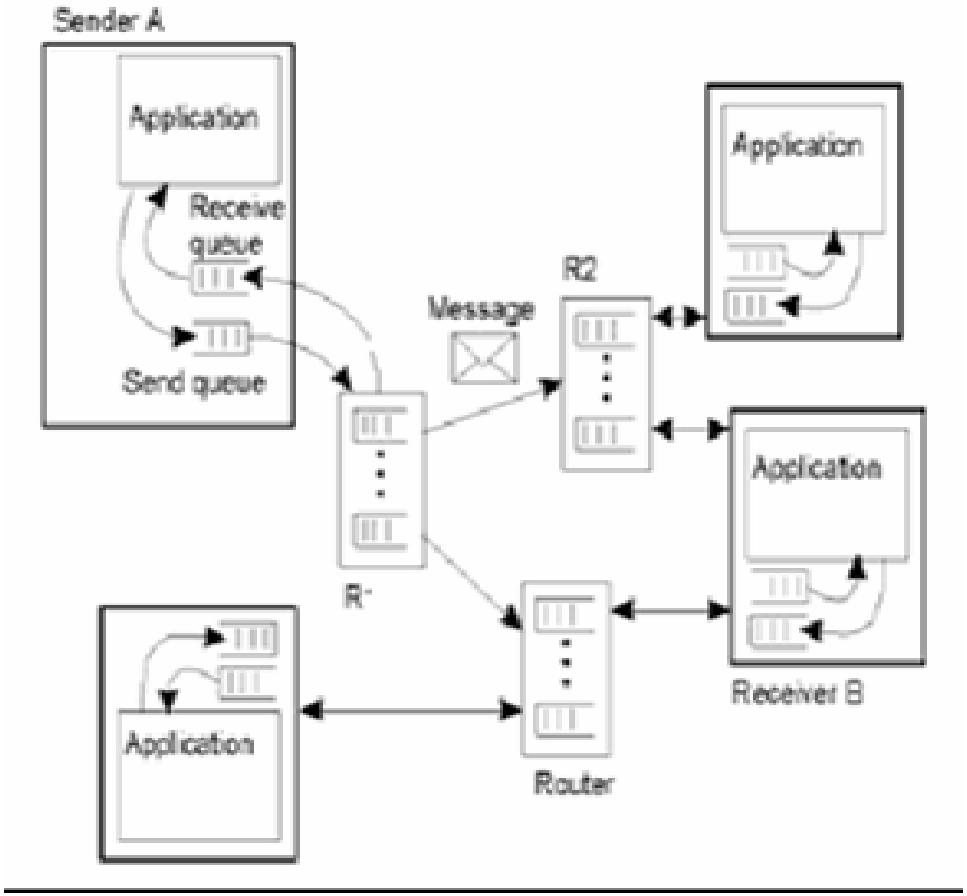
# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Arquitectura
    - Conjunto de filas distribuídas por múltiplas máquinas
    - Base de dados de mapeamento de filas



# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Arquitectura
    - Filas são mantidas por Gestores de Filas
    - Gestor de Filas:
      - Interage directamente com aplicação cliente/destino
      - Interage com outros Gestores de Filas (como *routers*)



# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Interface
    - API providenciada pelo MOM às aplicações
      - Envio de mensagens
      - Recepção de mensagens

# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Exemplo básico de interface

| Primitiva | Significado |
|---|---|
| Put | Adiciona mensagem a uma fila específica |
| Get | Remove mensagem da fila; processo bloqueado quando fila vazia |
| Poll | Verifica existência de mensagens numa fila e remove a mensagem desejada; processo não fica bloqueado |
| Notify | Instala callback function – invocada pelo MOM de cada vez que uma mensagem é colocada na fila |

# Message Oriented Middleware (MOM)

- Sistema baseado em Filas de Mensagens
  - Outras funcionalidades
    - Mensagens guardadas em fila podem ter um tempo de vida associado antes de serem descartadas
    - Priorização de mensagens
    - Filas podem ser partilhadas por múltiplas aplicações providenciadoras do mesmo serviço
      - Balanceamento de carga
      - Aumento de Performance
    - Filas transaccionais

# Message Oriented Middleware (MOM)

## Vantagens

- ✓ Permite a não-interdependência entre aplicações cliente/destino
- ✓ Assegurada fiabilidade do sistema, através de mecanismos de persistência
- ✓ Aplicações são isoladas das redes de comunicação, maior simplicidade e independência a problemas de rede
- ✓ Filas podem ser partilhadas entre aplicações, permitindo distribuição de carga
- ✓ Aplicação cliente pode definir prioridades às mensagens

## Desvantagens

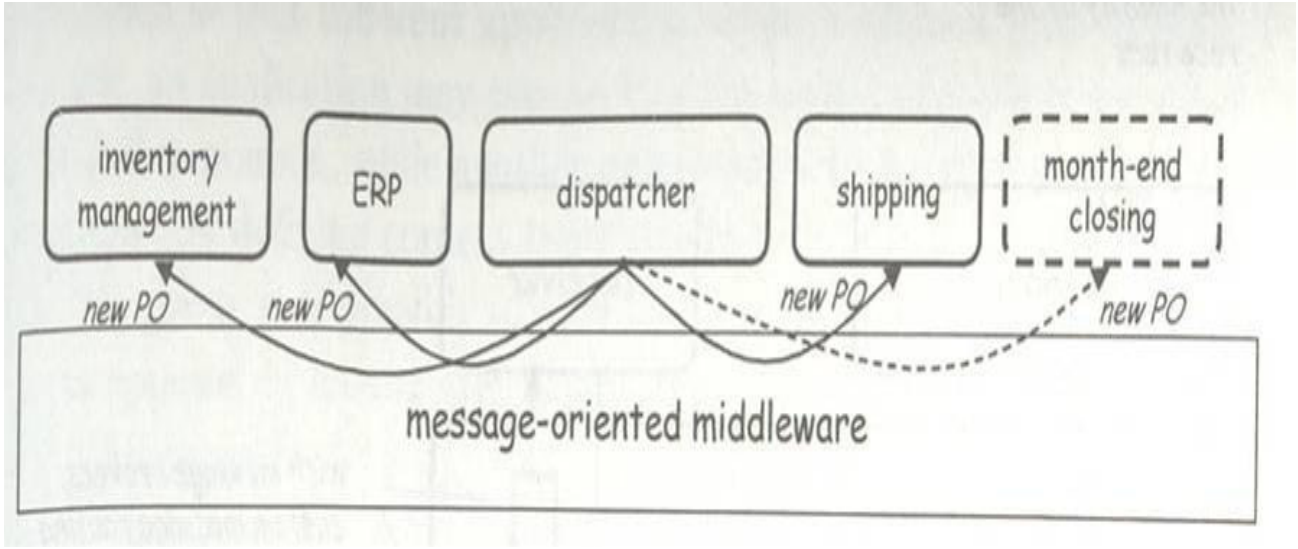
- ✗ Não existe um standard definido que regule as várias implementações
- ✗ Software MOM tem de correr em todas as aplicações
  - Quanto mais heterogéneo for o sistema, maior os custos associados
- ✗ Problemas de escalabilidade

# Message Brokers

- Porque surgiu?
  - Insuficiência dos mecanismos já existentes
    - RPC
    - MOM
      - Ambos criam ligações ponto-a-ponto entre aplicações, provocando problemas de escalabilidade
        - A responsabilidade de definir o receptor da mensagem é da aplicação cliente
      - Restrições
        - de encaminhamento
        - de conteúdo

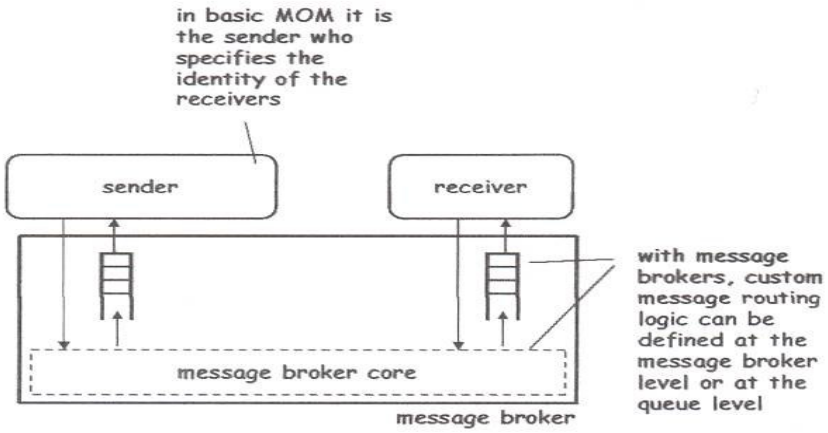
# Message Brokers

- Porque surgiu?



# Message Brokers

- No que consiste?
  - Sistema MOM evoluído, em que a lógica é:
    - retirada dos clientes
    - colocada no middleware



# Message Brokers

- No que consiste?
  - Lógica presente no middleware
    - Encaminhamento
      - identidade da aplicação cliente
      - conteúdo da mensagem
      - tipo da mensagem
    - Transformação
      - de formatos
    - Filtragem
      - que aplicações terão acesso à mensagem

# Message Brokers

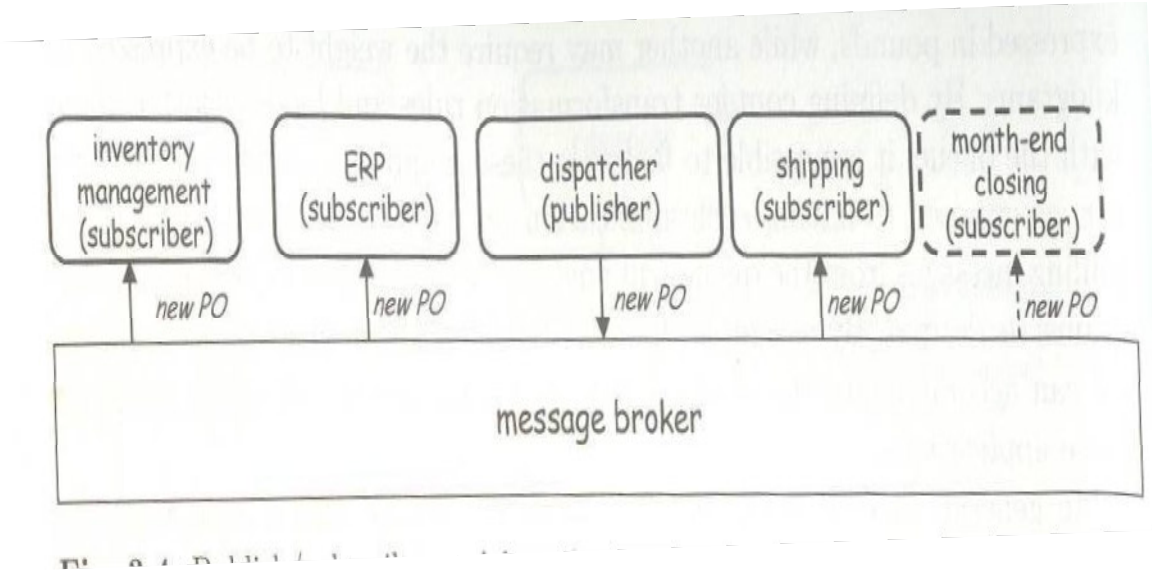
- Limitações
  - Excesso de lógica no middleware
    - dificuldades de debug e manutenção
    - quebras de performance
  - Mensagens grandes
    - performance grandemente afectada

# Message Brokers

- Publish/Subscribe
  - Modelo de interacção mais conhecido e adoptado
  - Funcionamento:
    - Publishers
      - Aplicações que publicam informação no middleware
    - Subscribers
      - Aplicações que registam o interesse junto do middleware, num determinado tipo de informação

# Message Brokers

- Publish/Subscribe

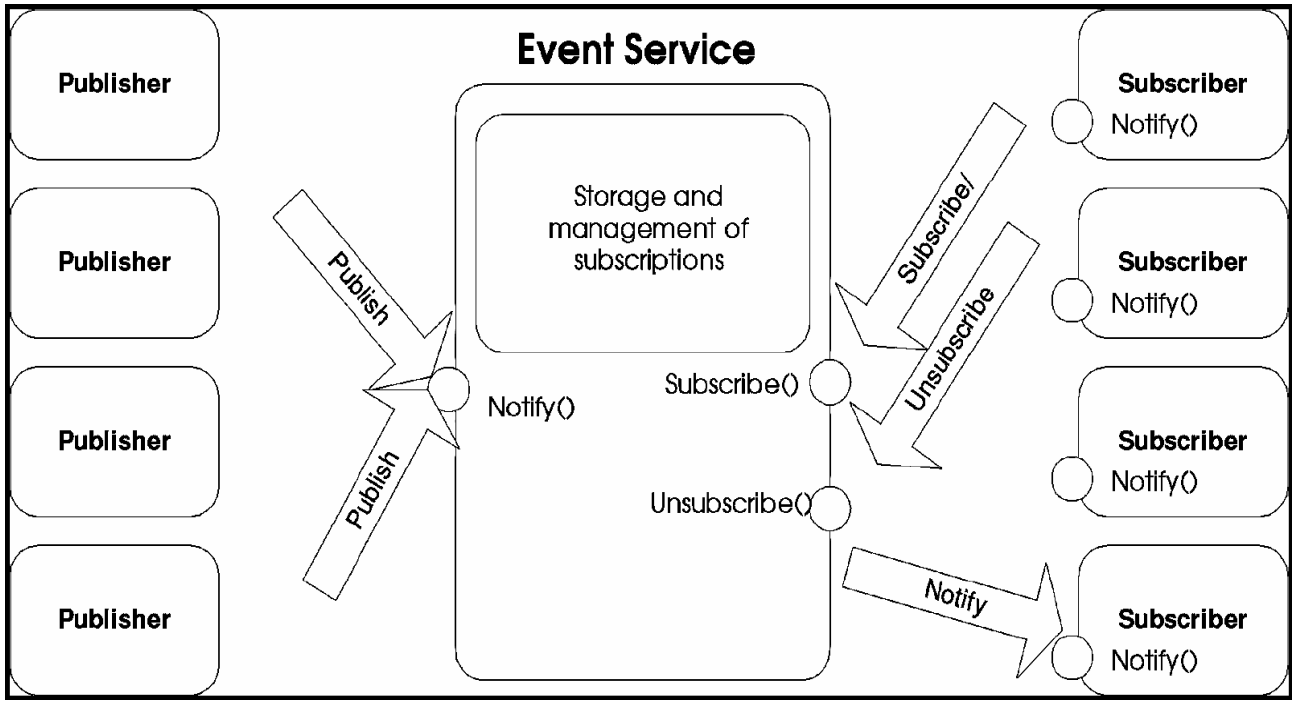


# Message Brokers

- Publish/Subscribe
  - Terminologia
    - Evento: informação disponibilizada pelos *publishers* e consumida pelos *subscribers*
    - Notificação: acto de entrega de evento
    - Gestor de Eventos: software bus onde a informação é mantida

# Message Brokers

- Publish/Subscribe



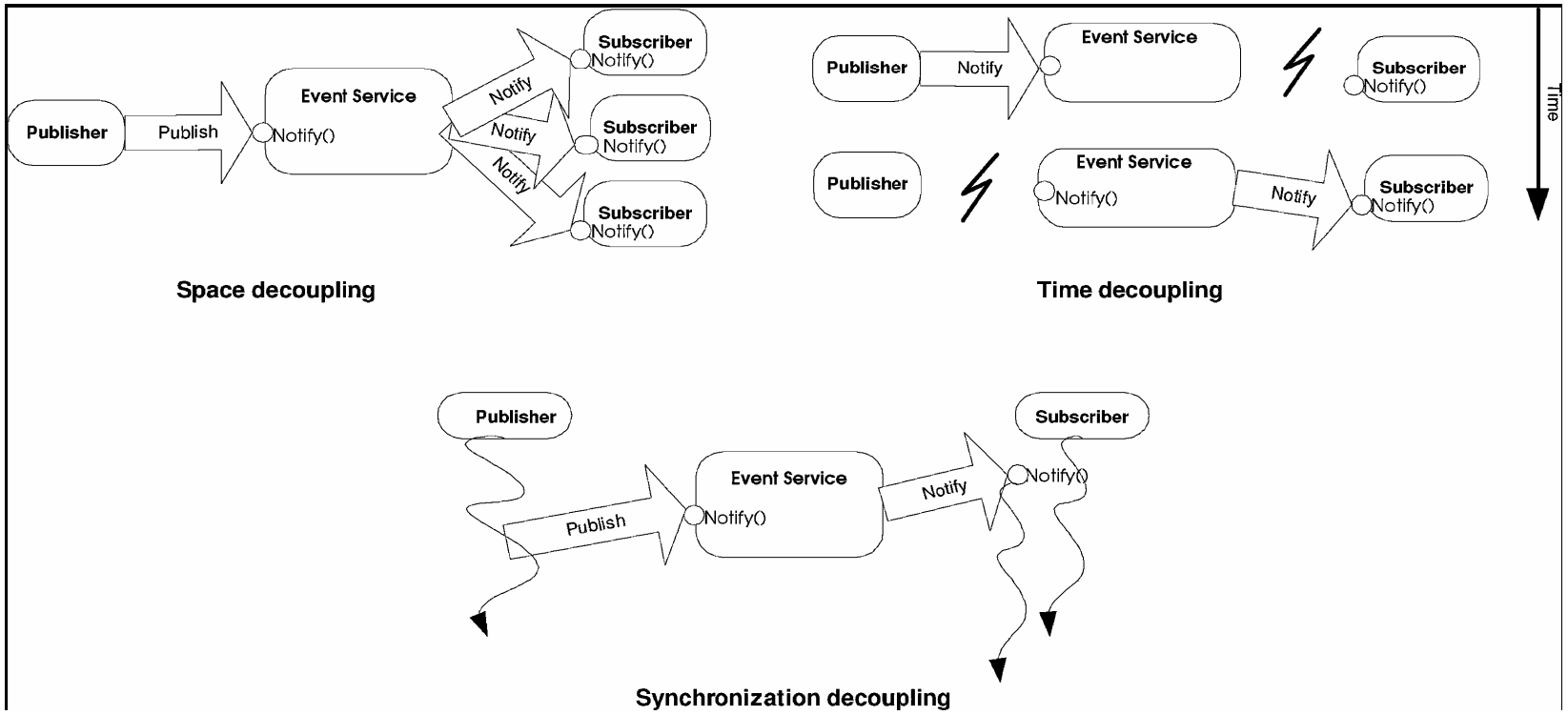
# Message Brokers

- Publish/Subscribe
  - Este modelo providencia independência:
    - em termos de  tempo
    - em termos de espaço
    - em termos de sincronização

    ... entre *Publishers/Subscribers*

# Message Brokers

- Publish/Subscribe



# Message Brokers

- Publish/Subscribe
  - Classificação
    - Baseado em tópicos
      - Eventos são definidos por assunto, expresso através de uma *keyword*
      - Cada tópico fica associado a um canal e é visto como um serviço próprio

```
public class StockQuote implements Serializable {
   public String id, company, trader;
   public float price;
   public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
   public void notify(Object o) {
      if (((StockQuote)o).company == 'TELCO' && ((StockQuote)o).price < 100)
         buy();
   }
}
// ...
Topic quotes = EventService.connect("/LondonStockMarket/Stock/StockQuotes");
Subscriber sub = new StockQuoteSubscriber();
quotes.subscribe(sub);
```

# Message Brokers

- Publish/Subscribe
  - Classificação
    - Baseado em conteúdo
      - Eventos são definidos pelas suas próprias propriedades
        - Atributos internos
        - Meta-data
      - Aquando da subscrição, é definido um padrão de eventos a subscrever
        - Filtros

```
public class StockQuote implements Serializable {
   public String id, company, trader;
   public float price;
   public int amount;
}
public class StockQuoteSubscriber implements Subscriber {
   public void notify(Object o) {
      buy();       // company == 'TELCO' and price < 100
   }
}
// ...
String criteria = ("company == 'TELCO' and price < 100");
Subscriber sub = new StockQuoteSubscriber();
EventService.subscribe(sub, criteria);
```

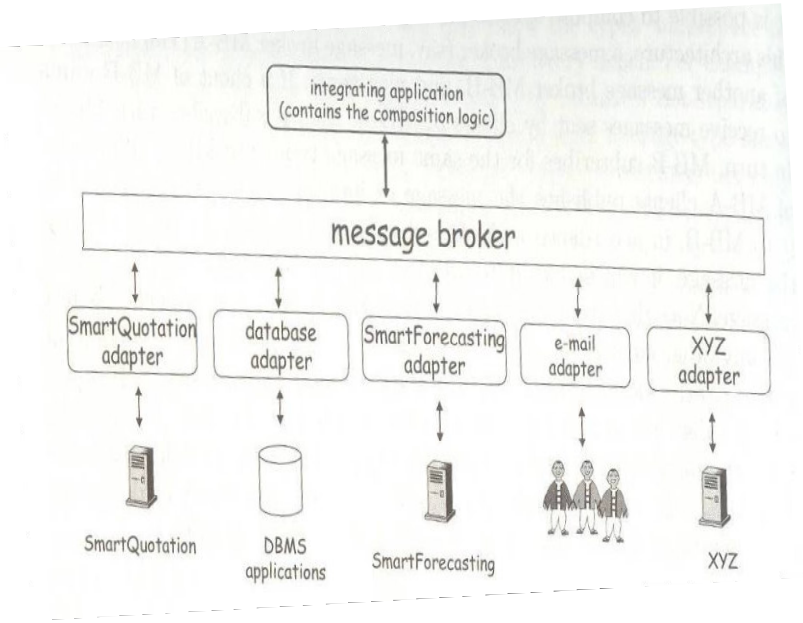
# Message Brokers

- Publish-Subscribe
  - Classificação
    - Baseado no tipo
      - Existe um conjunto pré-definido de tipos de eventos
      - Na subscrição é indicado qual o tipo desejado pelo *subscriber*

```
public class LondonStockMarket implements Serializable {
   public String getId() {...}
}
public class Stock extends LondonStockMarket {
   public String getCompany() {...}
   public String getTrader() {...}
   public int getAmount() {...}
}
public class StockQuote extends Stock {
   public float getPrice() {...}
}
public class StockRequest extends Stock {
   public float getMinPrice() {...}
   public float getMaxPrice() {...}
}
public class StockSubscriber implements Subscriber<StockQuote> {
   public void notify(StockQuote s) {
      if (s.getCompany() == 'TELCO' && s.getPrice() < 100)
         buy();
   }
}
// ...
Subscriber<StockQuote> sub = new StockSubscriber();
EventService.subscribe<StockQuote>(sub);
```

# Message Brokers

- Arquitectura
  - Adaptadores
    - Interagem com sistemas heterogéneos
    - Cada aplicação necessita de um adaptador
  - Message Broker
    - Facilita a interacção de
      - Adaptadores
      - Aplicações
    - Efectua a gestão de
      - Filas internas
      - Regras



# Message Brokers

## Vantagens

- ✓ Baixo custo de desenvolvimento
- ✓ Maior capacidade de desenvolvimento
  - derivado ao menor custo de integração
- ✓ Baixo custo de manutenção
  - adaptadores

## Desvantagens

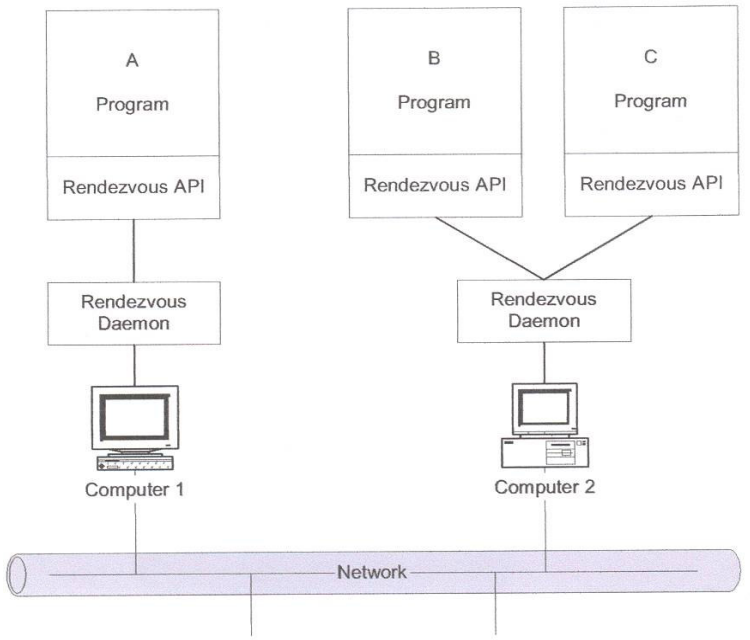
- ✗ Grandes custos comerciais
  - licenças
  - infraestrutura
  - desenvolvimento
  - ▸ Limitativo para pequenas/médias empresas
    - menores problemas de integração
    - demoram a recuperar o investimento

# Message Brokers

- TIBCO Rendezvous
  - Componentes
    - *communications daemon*
      - Background process - rvd
    - API

another.

*Figure 1 Rendezvous Operating Environment*



# Message Brokers

- TIBCO Rendezvous
  - Mensagens e Dados
    - cada mensagem tem um tópico (*subject name*), que define a quem se destina
    - dados são encapsulados nos campos da mensagem
      - cada campo contém dados de um específico tipo de dados
      - programas acedem aos campos da mensagem através do nome ou por um identificador numérico

# Message Brokers

- TIBCO Rendezvous
  - Uso de metainformação
    - dados definidos pelo *producer*, que permitem aos *subscribers* interpretar e usar os dados

*Table 6 Self-Describing Data*

| Element | Description |
|---|---|
| **Field Annotations** | |
| type | Producers must designate the type of every message field. Rendezvous software uses a set of wire-format datatype designations to characterize data by type and size. For example, th type `TIBRVMSG_I32ARRAY` denotes an array of 32-bit integers. |
| count (number of elements) | Producers must specify the length of all array data—that is, the number of elements in an array. |
| field name | Producers can label the fields of a message with names. Consumer use field names to select specific fields from messages. |
| field identifier | Producers can label the fields of a message with numeric identifier Consumers can use field identifiers to select specific fields from messages. All field identifiers in a message must be unique within that message. |
| **Message Annotations** | |
| subject name | Producers must label every outbound message with a send subject name (also called the *send subject name)*, which describes its conten and destination set. |
| reply subject name | Producers can label an outbound message with a reply subject nam to which consumers can send reply messages. |

# Message Brokers

- TIBCO Rendezvous
  - Endereçamento baseado em tópicos
    - abstrai os programadores dos detalhes relativos a endereços de rede, protocolos, hardware, OS,...
    - *producers* colocam em cada mensagem um nome definidor do tópico e enviam essas mensagens
    - *subscribers* que procurem por esse tópico, recebem todas as mensagens com esse nome
    - Tópico
      - *string* que especifica o destino da mensagem
      - descritivo do conteúdo da mensagem

# Message Brokers

- TIBCO Rendezvous
  - Endereçamento baseado em tópicos

These examples illustrate the syntax for subject names.

*Table 9 Valid Subject Name Examples*

| |
|---|
| NEWS.LOCAL.POLITICS.CITY_COUNCIL |
| NEWS.NATIONAL.ARTS.MOVIES.REVIEWS |
| CHAT.MRKTG.NEW_PRODUCTS |
| CHAT.DEVELOPMENT.BIG_PROJECT.DESIGN |
| News.Sports.Baseball |
| finance |
| This.long.subject_name.is.valid.even.though.quite.uninformative |

*Table 10 Invalid Subject Name Examples*

| |
|---|
| News..Natural_Disasters.Flood (null element) |
| WRONG. (null element) |
| .TRIPLE.WRONG.. (three null elements) |

# Message Brokers

- TIBCO Rendezvous
  - Tópicos
    - constituídos por elementos separados por pontos '**.**'
    - estes elementos são usados para implementar uma hierarquia de tópicos
    - é permitido o uso de *wildcards*
      - '*****' representa um elemento
      - '**>**' representa um ou mais elementos
      - *subscribers* podem procurar por tópicos com *wildcard* para aceder a um conjunto alargado de informação dentro da mesma subscrição

# Message Brokers

- TIBCO Rendezvous
  - Tópicos

*Table 12 Semantics of Listening to Wildcard Subjects*

| Listening to this wildcard name | Matches messages with names like these: | But does not match messages with names like these (reason): |
|---|---|---|
| RUN.* | RUN.AWAY | RUN.Run.run (extra element) |
| | RUN.away | Run.away (case) |
| | | RUN (missing element) |
| Yankees.vs.* | Yankees.vs.Red_Sox | Giants.vs.Yankees (position) |
| | Yankees.vs.Orioles | Yankees.beat.Sox (vs≠beat) |
| | | Yankees.vs (missing element) |
| *.your.* | Amaze.your.friends | your (missing elements) |
| | Raise.your.salary | Pick.up.your.foot (position) |
| | Darn.your.socks | |
| RUN.> | RUN.DMC | HOME.RUN (position) |
| | RUN.RUN.RUN | Run.away (case) |
| | RUN.SWIM.BIKE.SKATE | RUN (missing element) |

# Message Brokers

- TIBCO Rendezvous
  - Endereçamento por tópicos
    - Define modo de entrega das mensagens
      - *inbox name* (_INBOX como primeiro elemento do tópico)
        - mensagens ponto-a-ponto
      - outro qualquer nome
        - mensagens *multicast*

# Message Brokers

- TIBCO Rendezvous
  - Endereçamento por tópicos
    - Define modo de entrega das mensagens

*Figure 2 Point-to-Point Message*

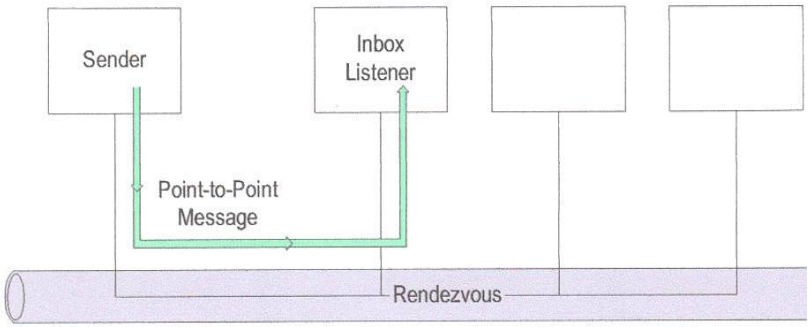
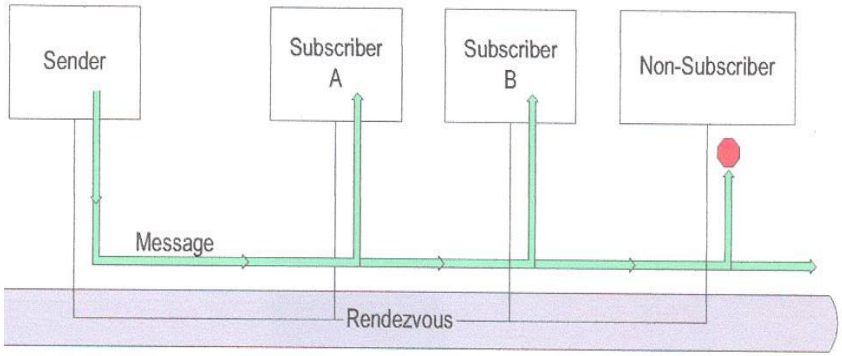



*Figure 3 Reliable Multicast Message*



# Message Brokers

- TIBCO Rendezvous
  - Metadados de uma mensagem

| | |
|---|---|
| Data | Dados |
| Type | Indicador de como tratar os dados |
| Size | Número de bytes que os dados ocupam |
| Name | Nome do campo |
| Id | Opcional. Identifica campo dentro da mensagem |
| Count | Número de elementos num array |

# Message Brokers

- TIBCO Rendezvous
  - Metadados de uma mensagem

| Field Name | Id | Type | Size | Count | Data |
|---|---|---|---|---|---|
| Name | 1 | TIBRVMSG_STRING | 11 | 1 | Jane Smith |
| Address | 2 | TIBRVMSG_STRING | 30 | 1 | 1234 Home Street<br>Anytown, USA |
| Phone | 3 | TIBRVMSG_STRING | 13 | 1 | 415-123-4567 |
| Age | 4 | TIBRVMSG_U8 | 1 | 1 | 46 |
| Scores | 5 | TIBRVMSG_U8ARRAY | 1 | 5 | 91, 99, 97, 99, 92 |

# Message Brokers

- TIBCO Rendezvous
  - Eventos
    - programas criam *event objects* para registar interesse num conjunto de ocorrências
    - a cada *event object* é fornecido uma *callback function* de cada vez que um evento ocorre

*Table 17 Event Classes*

| | |
|---|---|
| Message | An inbound message has arrived. Additional creation parameters specify the subject name and transport. See also, Listener Event Semantics on page 91. |
| Timer | A timer interval has elapsed. An additional creation parameter specifies the interval. See also, Timer Event Semantics on page 96.<br>The Rendezvous .NET API does *not* support this kind of event. |
| I/O | An I/O socket is ready. Additional creation parameters specify the socket, and the I/O condition. See also, I/O Event Semantics on page 94.<br>The Rendezvous Java, COM and .NET APIs do *not* support this kind of event. |

# Message Brokers

- TIBCO Rendezvous
  - Eventos

*Table 16 Event System Components*

| Component | Description |
|---|---|
| Event Object | Represents program interest in a set of events, and the occurrence of a matching event. See Events on page 81. |
| Event Driver | The event driver recognizes the occurrence of events, and places them in the appropriate event queues for dispatch. Rendezvous software starts the event driver as part of its process initialization sequence (the *open* call). See Event Driver on page 83. |
| Event Queue | A program creates event queues to hold event objects in order until the program can process them. See Event Queues on page 84. |
| Event Queue Group | Programs can organize event queues into groups for fine-grained control of dispatch priority.<br>See Queue Groups and Priority on page 85. |
| Event Dispatch Call | A Rendezvous function call that removes an event from an event queue or queue group, and runs the appropriate callback function to process the event.<br>See Dispatch on page 84. |
| Callback Function | A program defines callback functions to process events asynchronously.<br>See Callback Functions on page 89. |
| Dispatcher Thread | Programs usually dedicate one or more threads to the task of dispatching events. Callback functions run in these threads. |

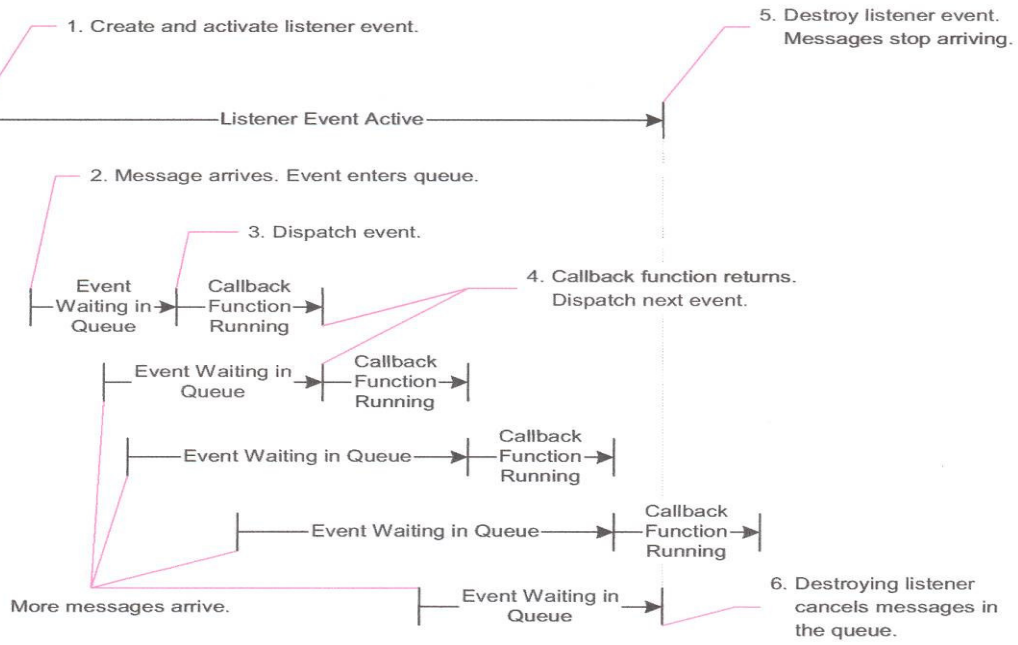
# Message Brokers

- TIBCO Rendezvous
  - Evento Listener
    - programas procuram por mensagens, criando *listener events objects*
      - com base no tópico das mensagens
      - procuram mensagens até o programa os destruir
    - programas devem definir callback functions para processar mensagens
    - mensagem chega → Event → Event queue → Dispatch Event → Callback function

# Message Brokers

- TIBCO Rendezvous
  - Evento Listener

*Figure 10 Listener Activation and Dispatch*



# Message Brokers

- Funcionalidades

*Table 2 Overview of Functionality*

| Facility | Description |
|---|---|
| Message | • Translate between universal wire format and local data formats<br>• Manipulate messages and fields |
| Event | • Listen for messages by subject name<br>• Register interest in timers and I/O events |
| Transport | • Define delivery scope, delivery mechanism and protocols<br>• Connect to the network<br>• Send messages |
| Queue | • Create and manipulate event queues<br>• Dispatch events |
| Queue Group | • Customize event dispatch by combining queues |
| Certified Message Delivery | • Confirm delivery of each message to each registered recipient<br>• Deliver messages despite process termination and restart |
| Distributed Queue | • Distribute a service over several processes |
| Fault Tolerance | • Coordinate redundant processes to achieve application-level fault tolerance |

- Arquitectura API

*Table 4 Architecture Summary*

| Element | Description |
|---|---|
| Message | Messages carry data among program processes or threads.<br>Messages contain self-describing data fields. Programs can manipulate message fields, send messages, and receive messages. |
| Event | Programs create event objects to register interest in significant conditions. For example, dispatching a listener event notifies the program that a message has arrived; dispatching a timer event notifies the program that its interval has elapsed.<br>Programs define event callback functions to process events. |
| Event Queue | Programs create event queues to organize events. A queue holds a sequence of event objects that are ready for dispatch. |
| Event Queue Group | Programs create event queue groups to prioritize event processing. |
| Event Dispatch | Programs dispatch events from queues or queue groups, processing each event with the corresponding callback function. |
| Transport | Programs use transport objects to send messages and listen for messages. A transport determines three aspects of message delivery:<br>• Delivery scope—the potential range of its messages<br>• Delivery mechanism—the path that its messages travel<br>• Delivery protocol—the ways in which programs cooperate and share information concerning message delivery<br>Transport objects of various types combine these aspects to yield different qualities of service—for example, intra-process delivery, network delivery, reliable delivery, certified delivery, distributed queue delivery. |
| Event Driver | Rendezvous software includes an event driver that places events in event queues. (Program code cannot access the event driver.) |

# Referências


- G.Alonso, F.Cassati, H.Kuno, V.Machiraju,
  **Web Services – Concepts, Architectures and Applications**

- A.Tanenbaum, M.Steen,
  **Distributed Systems – Principles and Paradigms**

- P.Eugster, P.Felber, R.Guerraoui, A.Kermarrec,
  **The Many Faces of Publish-Subscribe**
  *ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp 114-131*

- **TIBCO Rendezvous – Concepts**
  *Software Release7.4 – July 2005*