PÓS-GRADUAÇÃO ALFA

UNIALFA
CENTRO UNIVERSITÁRIO
ALVES FARIA

Diego Américo Guedes

PROGRAMAÇÃO COM FRAMEWORKS E COMPONENTES

GRUPO JOSÉ ALVES

Eventos e Notificações
Modelo Publish-Subscribe

Prof. Me. Diego Guedes

O que há de errado com RMI/RPC?

- Comunicação ponto-a-ponto
  - Cliente para objeto remoto
  - Dificulta escalabilidade
- Acoplamento temporal (sincronismo)
  - Cliente fica bloqueado quando faz uma chamada a um objeto remoto
  - Esse acoplamento entre cliente e servidor não é natural para alguns sistemas distribuídos, por exemplo, em aplicações móveis
- Acoplamento espacial
  - Cliente possui uma referência para o objeto remoto

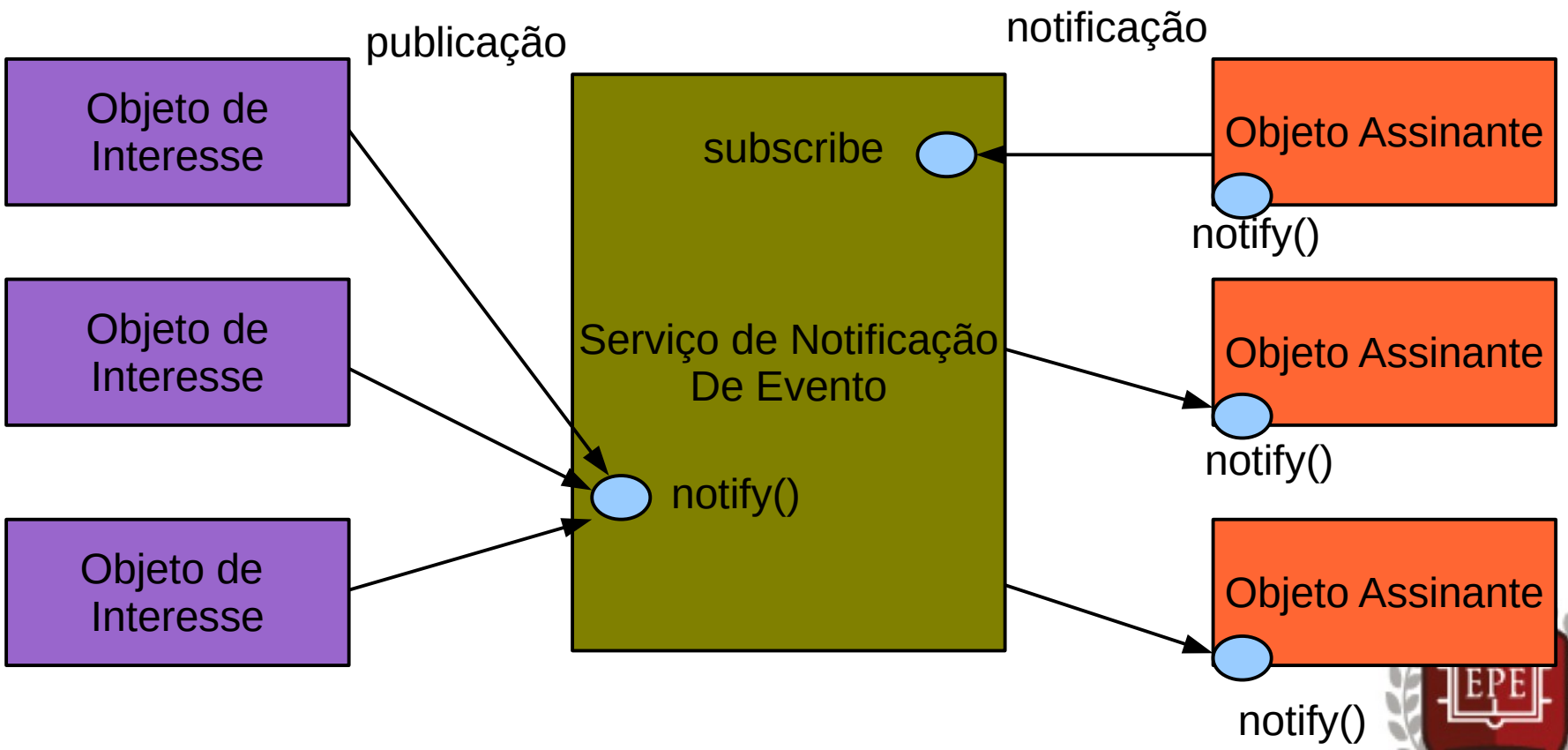
Modelo de Eventos e Notificação

- Paradigma de comunicação mais apropriado para sistemas de grande escala, heterogêneos e dinâmicos.
- Também chamado de modelo publish-subscribe
- Modelo de comunicação assíncrona entre
  - Objeto de interesse (publish): gerador de eventos
  - Objeto assinante (subscribe): consumidor de eventos

Interação de objetos no modelo de eventos e notificação

- Objeto assinante (***subscribe***) expressa seu interesse em uma informação.
- Quando a informação está disponível, objeto de interesse publica (***publish***) a informação em um bus ou canal
- Essa informação é chamada de evento e o ato de entregar a informação é chamada de notificação
- Inscrever-se em um tipo particular de evento é chamado de ***registrar interesse*** no evento
- Um **serviço de notificação de evento** representa um mediador entre o objeto de interesse e o objeto assinante, provendo:
  - Armazenamento e gerenciamento de inscrições
  - Entrega eficiente de notificações

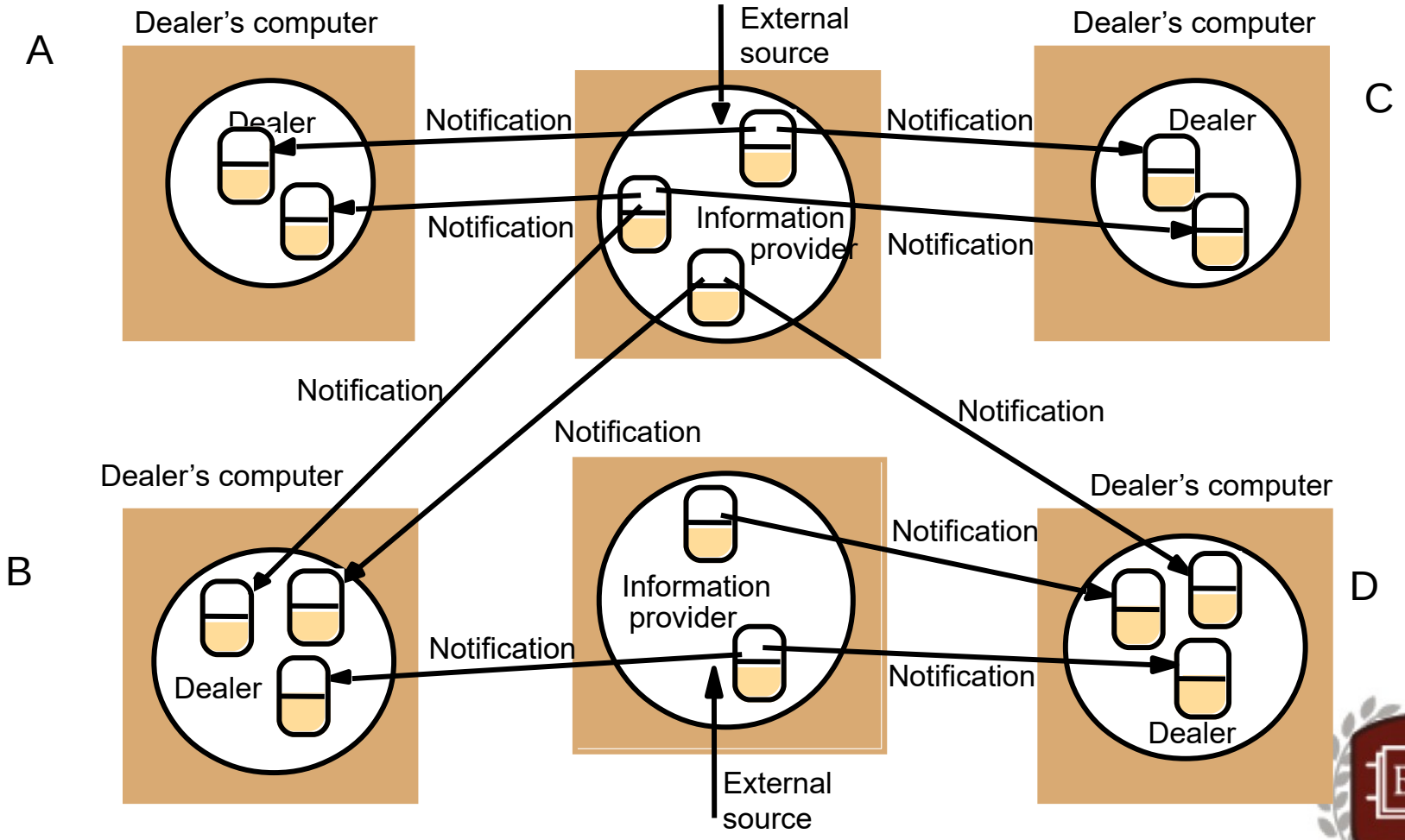
Interação de objetos no modelo de eventos e notificação



Interação de objetos no modelo de eventos e notificação

- Desacoplamento Espacial: Objeto de interesse e objeto assinante não precisam conhecer um ao outro
  - Nenhuma das entidades possui referência para a outra
- Desacoplamento Temporal: objeto de interesse e objeto assinante não precisam estar ambos ativos (conectados) no momento da comunicação
- Comunicação assíncrona:
  - Objeto de interesse não é bloqueado enquanto produz um evento
  - Objeto assinante não é bloqueado enquanto consome um evento

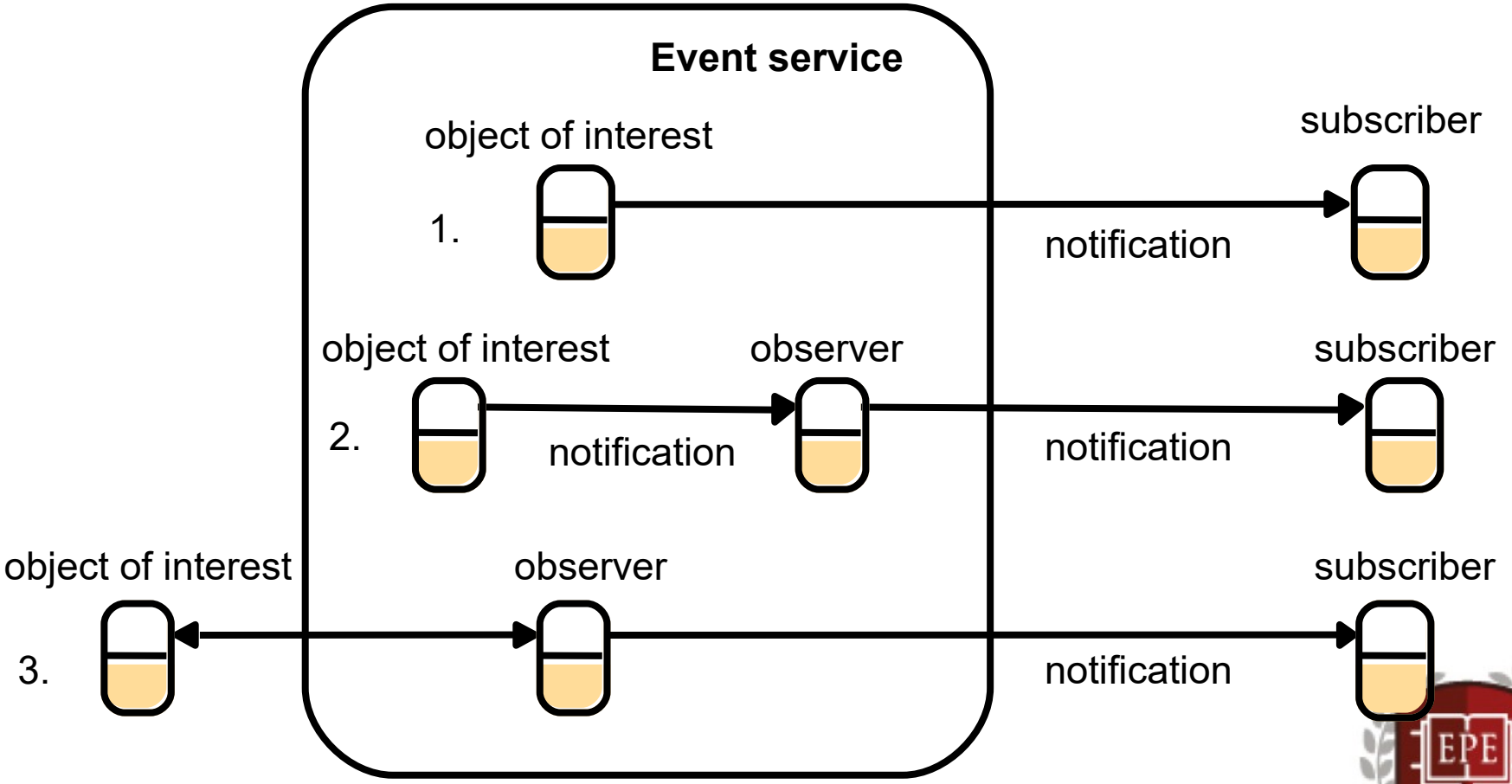
Exemplo de Aplicação
Corretora de ações



Serviço de Evento

- Serviço de Notificação de Evento
  - Armazena e gerencia as inscrições dos objetos assinantes
  - Armazena os eventos para que possam ser entregues mesmo quando há desconexão do objeto assinante
  - Realiza filtragem de eventos
  - Detecta correlação de eventos

Participantes da notificação de eventos distribuída



Participantes da notificação de eventos distribuída

- Objeto de interesse: trata-se de um objeto que sofre mudanças de estado, como resultado da invocação de seus métodos.
- Evento: ocorre em um objeto de interesse como resultado da conclusão da execução de um método
- Notificação: é um objeto que contém informações sobre um evento.
- Assinante: é um objeto que se inscreveu em algum tipo de evento em outro objeto. Ele recebe notificações sobre esses eventos.

Participantes da notificação de eventos distribuída

- Objetos observadores:
  - o principal objetivo é desvincular um objeto de interesse de seus assinantes, já que um objeto de interesse pode ter muitos assinantes diferentes, com interesses distintos;
  - Um ou mais observadores podem ser inseridos entre um objeto de interesse e os assinantes.
- Gerador de eventos (*publisher*): objeto que declara que vai gerar notificações de tipos de eventos em particular. Um gerador pode ser um objeto de interesse ou um observador.

Esquemas de Inscrição

- Num sistema distribuído ocorrem milhares de eventos
- Nem todos os eventos são de interesse de todos os assinantes
- As diferentes maneiras de se especificar os eventos de interesse deram origem a diferentes esquemas de inscrição:
  - Baseado em tópico
  - Baseado em conteúdo

Esquemas de Inscrição

- Modelo de Inscrição baseado em tópicos
  - Classifica o conteúdo dos eventos por tópico, identificado por uma palavra chave
  - Um objeto de interesse publica um evento em um tópico.
  - Um Objeto assinante se inscreve em um ou mais tópico
  - A ideia de tópico se assemelha à ideia de canal de evento. Um canal é criado para cada tópico distinto
  - Cada canal é identificado por um nome e consiste em um serviço de evento próprio

Esquemas de Inscrição

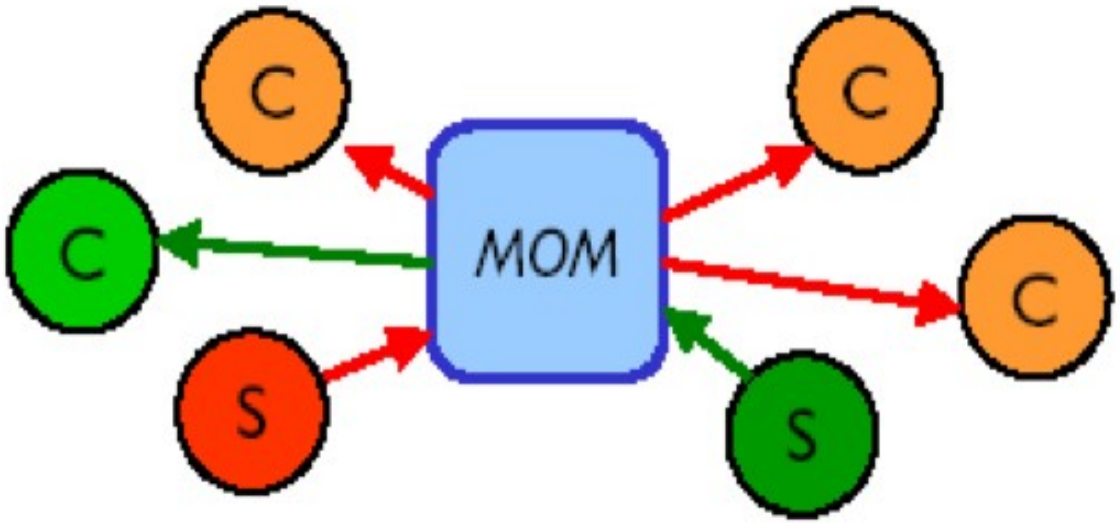
- Modelo baseado em conteúdo
  - Classifica um evento pelo seu conteúdo real em tempo de execução
  - Um evento é classificado de acordo com seus atributos (dados internos ou metadados)
  - Um objeto assinante expressa seu interesse por um determinado evento através de expressões lógicas do tipo <atributo> <operação> <valor>

Middleware Orientado a Mensagem

Prof. Me. Diego Guedes

Introdução

- Middleware orientados a mensagem (MOM) proveem comunicação distribuída em uma arquitetura peer-to-peer com serviço centralizado de armazenamento e repasse de mensagem
- Processos/Componentes enviam e recebem mensagens de canais administrados por um provedor central (MOM)



C = consumidor de mensagem

S = produtor de mensagem

Introdução

- Característica do modelo de comunicação
  - Comunicação assíncrona
    - Emissor pode continuar sua execução após o envio da mensagem
  - Comunicação desacoplada
    - O receptor não precisa estar ativo no momento do envio da mensagem
    - Não há garantias de quando a mensagem será entregue pelo serviço central
    - Assemelha-se a um serviço postal.

Fila de Mensagem

- Fila de Mensagem (Message Queue)
  - Conceito fundamental em MOM
  - Provê a estrutura de dados que armazena mensagens no MOM
  - As mensagens são armazenadas em FIFO

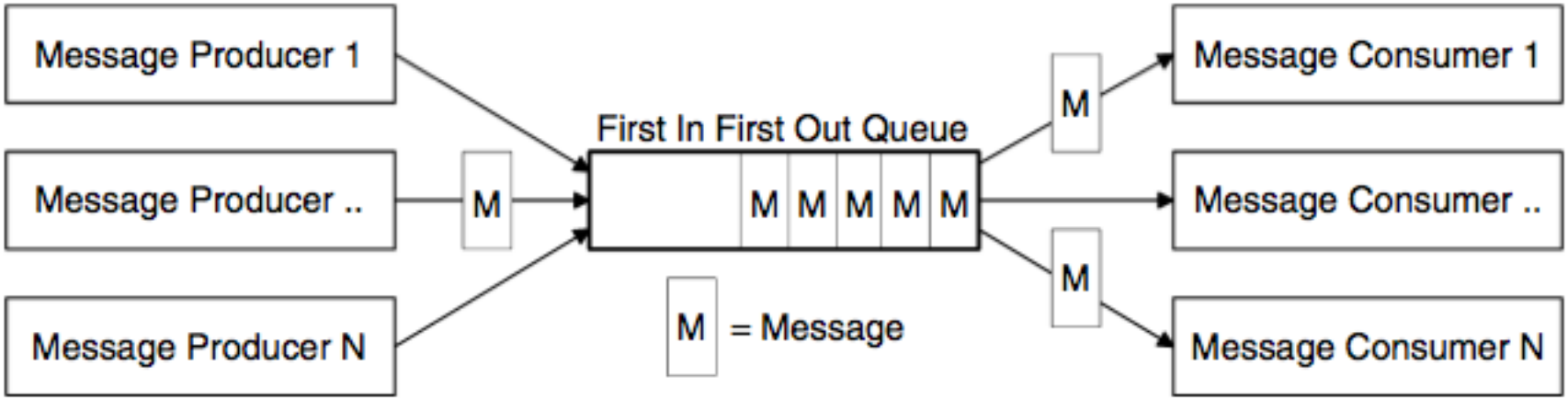


Figure 1.5 Message queue

Fila de Mensagem

- É possível configurar
  - O nome da fila
  - O tamanho da fila
  - A política de ordenação
  - Um limiar para ativar gravação da fila em um armazenamento persistente
- Cada aplicação/componente pode ter sua própria fila ou uma fila pode ser compartilhada entre várias aplicações/compontes

Fila de Mensagem

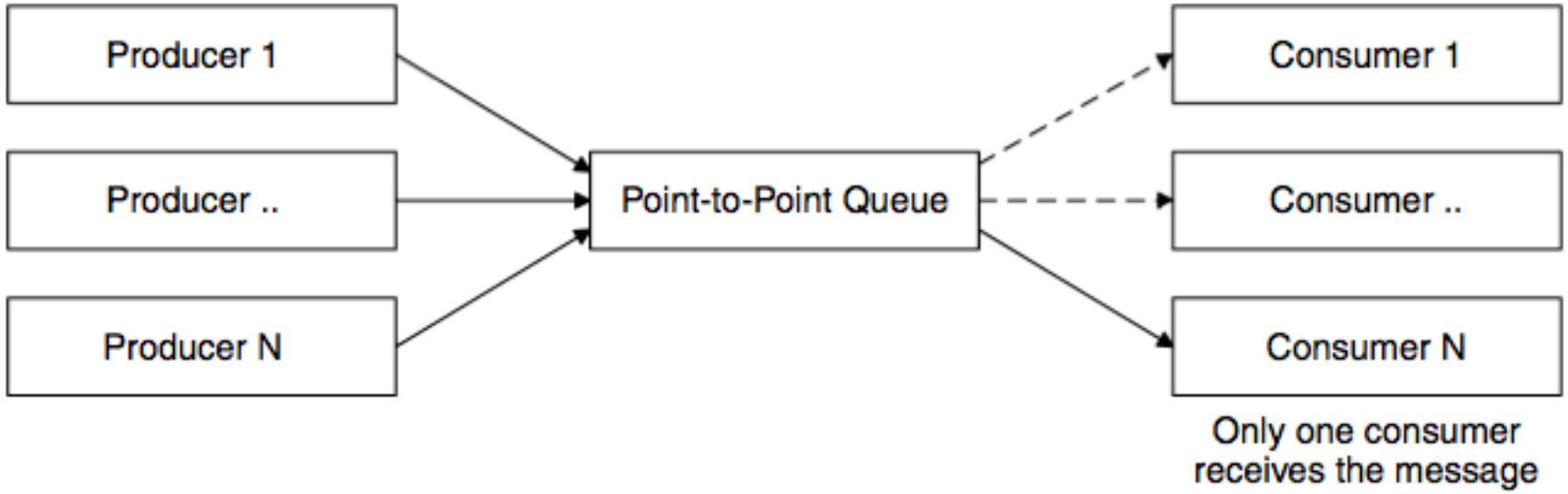
- Tipos de Fila
  - Fila pública: não exige autenticação de acesso
  - Fila privada: exige autenticação de acesso
  - Fila temporária: existe apenas temporariamente
  - Fila persistente: persiste cada mensagem na fila
  - Fila de mensagem morta: armazenam mensagens cujo tempo de vida expirou ou que não foram entregues

Modelos de Mensagem

- Existem dois modelos de mensagem em MOM
  - Modelo ponto-a-ponto (1:1)
  - Modelo publish-subscribe (1:N)

Modelos de Mensagem

- Ponto-a-Ponto
  - Um emissor envia uma mensagem para a fila que é consumida por um único receptor
  - Usado em balanceamento de carga

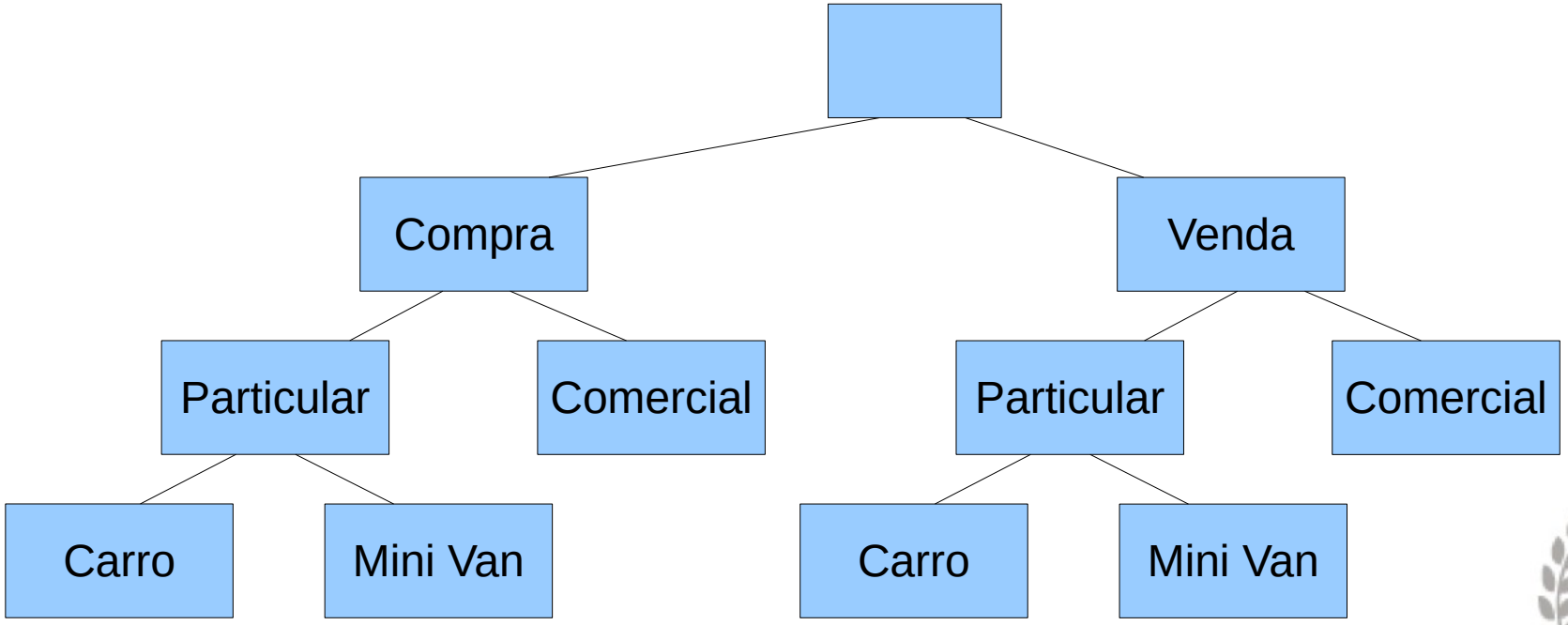


Modelos de Mensagem

- Publish-subscriber
  - Um emissor envia uma mensagem para a fila que é consumida por inúmeros receptores

Modelos de Mensagem

- Publish-subscriber
  - Hierarquia de canais/tópicos



Serviços Comuns

- Filtragem de mensagem
  - Permite que um receptor/consumidor de mensagem selecione as mensagens que deseja receber
  - Usado no modelo de mensagem pub-sub
  - O modelo de subscrição pode ser por tópico ou conteúdo.

Serviços Comuns

- Controle de Transação
  - É usado quando se deseja que um conjunto de mensagens seja entregue atomicamente, ou seja, ou todas as mensagens são entregues ou nenhuma nenhuma mensagem é entregue.

Produtor/Emissor
1.O emissor envia uma mensagem ou conjunto de mensagens para o broker
2.Ao receber um commit, o broker persiste as mensagens e então as envia
3.Ao receber um rollback, o broker descarta as mensagens

Consumidor/Receptor
1.O receptor se inscreve para receber uma mensagem ou um conjunto de mensagens
2.Ao receber um commit, o broker descarta as mensagens
3.Ao receber um rollback, o broker reenvia as mensagens

Serviços Comuns

- Garantia de entrega
  - O nível de confiabilidade em que o MOM opera é configurável:
    - Serviço com garantia de entrega: o MOM persiste a mensagem indefinidamente, até que a mesma seja entregue
    - Serviço sem garantia de entrega: o MOM não persiste a mensagem ou persiste enquanto a mensagem for válida (Time-to-live TTL)
- Replicação do Servidor MOM

Estudo de Caso

- Java Message Service (JMS)
  - Provê uma interface (API) Java para comunicação baseada em mensagem (ponto a ponto ou pub-sub)
  - É uma especificação que permite que aplicações Java criem, enviem, recebam e leiam mensagens através de um MOM
  - JBossMQ, OpenJMS, JORAM, **GlassFish**

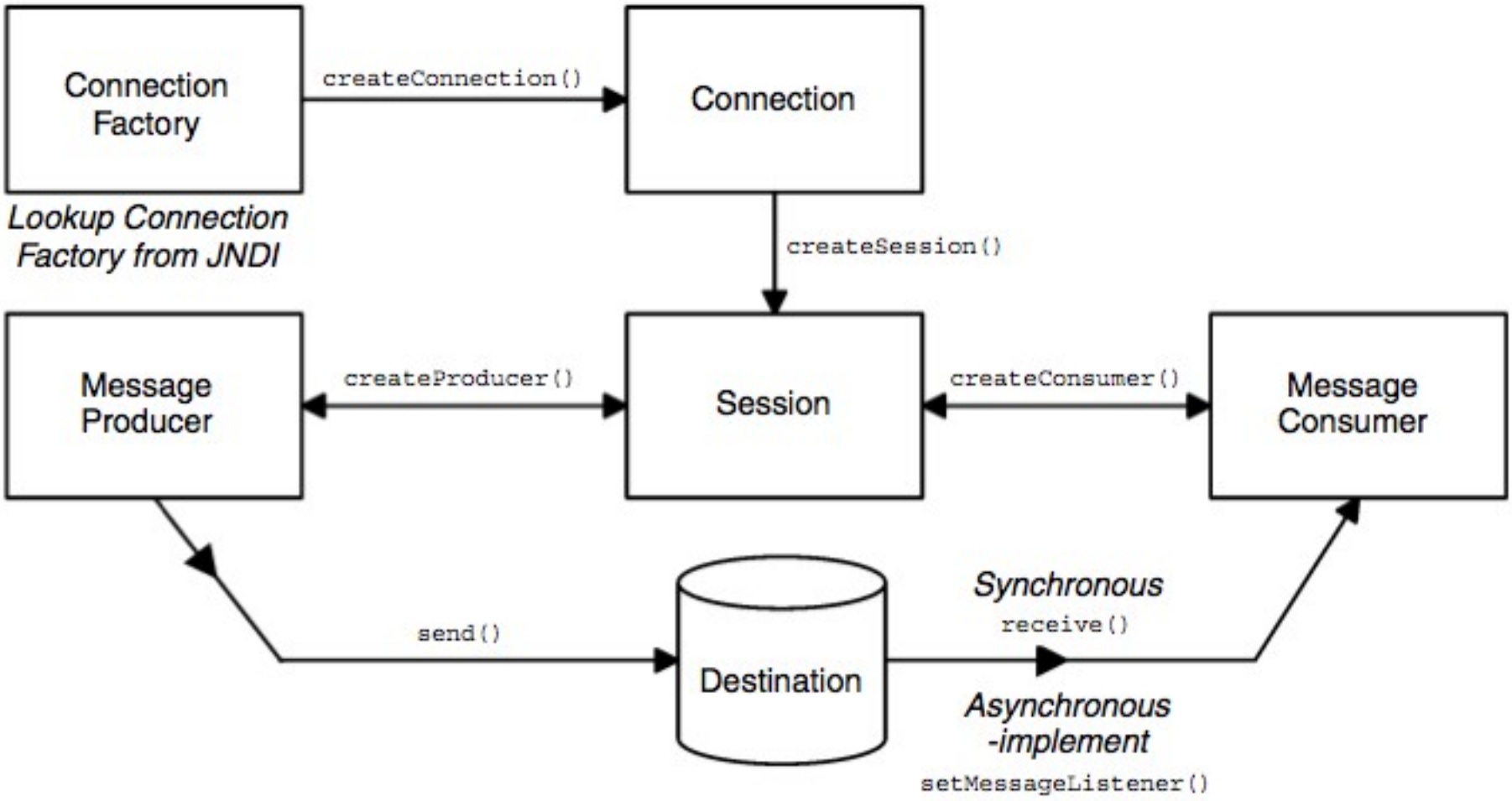
JMS

- Principais características
  - Comunicação fracamente acoplada
  - Semântica de invocação “no máximo uma vez”
  - Modelo ponto-a-ponto e pub-sub
  - Modo de entrega com persistência (default) e sem persistência
  - tempo de vida e prioridade
  - Suporte a transação
  - Roteamento seletivo através da filtragem de mensagens

JMS

- Primitivas de comunicação
  - Send: envia uma mensagem para uma fila específica
  - Receive bloqueante: bloqueia se a fila está vazia
  - Receive não bloqueante: não bloqueia se a fila estiver vazia

JMS


Connection Factory
createConnection()
Connection
Lookup Connection Factory from JNDI
createSession()
Message Producer
createProducer()
Session
createConsumer()
Message Consumer
send()
Destination
Synchronous
receive()
Asynchronous
-implement
setMessageListener()

JMS

- localhost:4848
- Vá primeiro em Recursos > Recursos JMS > Fábricas de Conexões > Novo; entre com Nome do Grupo = “jms/CF”, Tipo de Recurso = javax.jms.QueueConnectionFactory; confirme com OK.
- Acione Recursos > Recursos JMS > Recursos de Destino > Novo; entre com Nome JNDI = “jms/Quotes”; Physical Destination Name = “Quotes”; Tipo de recurso = javax.jms.Queue.
- Você tem agora uma Queue, a fila que será usada para o tráfego de mensagens.

JMS - Produtor

```
package teste;

import javax.annotation.Resource;
import javax.inject.Named;
import javax.enterprise.context.Dependent;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
@Named(value="JMSBean")
@Dependent
public class JMSBean {
  @Resource(name="jms/CF") ConnectionFactory qcf;
  @Resource(name="jms/Quotes") Destination dest;
  static int counter;
  public void send () throws JMSException {
    Connection conn = null;
    try {
      conn = qcf.createConnection();
      Session session = conn.createSession(true, Session.AUTO_ACKNOWLEDGE);
      MessageProducer prod = session.createProducer(dest);
      TextMessage msg = session.createTextMessage("^BVSP = $" + (counter++));
      prod.send(msg);    }
    finally {
      if(conn != null) conn.close();  }
```

JMS - Consumidor

```
package teste;
import javax.ejb.ActivationConfigProperty;
import javax.ejb.MessageDriven;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.TextMessage;

@MessageDriven(mappedName = "jms/Quotes", activationConfig =  {
  @ActivationConfigProperty(propertyName = "acknowledgeMode",
   propertyValue = "Auto-acknowledge"),
  @ActivationConfigProperty(propertyName = "destinationType",
   propertyValue = "javax.jms.Queue")
})
public class QuotesConsumerBean implements MessageListener {

  @Override
  public void onMessage (Message message) {
    try {
      TextMessage msg = (TextMessage)message;
      System.out.println(msg.getText());
    } catch (final JMSException e) {
      System.out.println("Deu pau! - " + e);
    }
  }
}
```

JMS

```
// Synchronous Receive
javax.jms.Message msg = queueReceiver.receive();
if (msg instanceof TextMessage) {
  javax.jms.TextMessage txtMsg = (TextMessage) msg;
  System.out.println("Reading message: " + txtMsg.getText());
```

JMS

- Formato de mensagem
  - Text: objeto da classe String
  - Byte: sequência de bytes
  - Stream: sequência de tipos primitivos
  - Object: objeto serializado