Introdução

Situação onde o remetente e destinatário não se conhecem mas ainda sim conseguem se comunicar. Há sempre **um intermediário** (framework ou middleware) que permite essa comunicação, isso retira a rigidez da comunicação do acoplamento direto (onde remetente e destinatário se conhecem).

- Comunicação entre entidades por meio de um intermediário, sem nenhum acoplamento direto entre o remetente e o destinatário.
- A rigidez do acoplamento direto: RMI e RPC
- Vantagens
    - Desacoplamento espacial
      Em que o remetente não tem informação sobre o destinatário e vice versa.
    - Desacoplamento temporal
      Transmissor e receptor podem existir em tempos de vida diferentes.
- Desvantagens: sobrecarga e dificuldade de gerenciamento.
  Sempre que uma entidade é inserida no processo é adicionada também uma sobrecarga. Também é mais complexo gerenciar a comunicação justamente pela falta de informação que o desacoplamento causa.

Acoplamento espacial e temporal

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

Comunicação em grupo

- Uma abstração em relação à comunicação por multicast.
  A comunicação é semelhante ao multicast, porém com garantia de **ordenação e confiabilidade**.
- A comunicação em grupo está para o multicast IP assim como o TCP está para o IP.
  O TCP garante confiabilidade para quem usa IP, da mesma forma é a comunicação em grupo que garante isso para o multicast IP.
- Áreas de aplicação:
    - Disseminação confiável de informações
      em uma transação bancária ou do mercado financeiro por exemplo.

- Aplicativos colaborativos
  Por exemplo, CS.
- Suporte a estratégias de tolerância a falhas
  Comunicação em grupo garante resposta a uma falha, para que os servidores funcionem corretamente. Ex: keepalived.
- Monitoramento e gerenciamento de sistemas
  garante monitoramento do estado de cada servidor no momento.

Modelo de programação

- Processos podem ingressar ou sair de um grupo.
- Uma mensagem enviada é entregue a todos os membros do grupo.
- Garantias de confiabilidade e ordenação.
  todas as mensagens são entregues e entregues em ordem.
- Multicast (envia para um grupo seleto, uso um consumo de banda menor) X Broadcast (envia para todos os nós) X Unicast (envia para um único nó)
  O multicast garante eficiência maior em comunicação com o grupo de nós.
- A eficiência de uma comunicação multicast quando comparado a várias mensagens separadas.

Modelo de programação

- Grupos de processos (soquetes) e grupos de objetos (rmi);
  Os grupos de processos se comunicam com soquetes.
- Grupos abertos e grupos fechados
  aberto: determinado elemento que não faz parte do grupo consegue enviar mensagem.
  fechado: nós externos não se comunicam com o grupo.
- Grupos sobrepostos e não sobrepostos
  Sobrepostos: entidades fazendo parte de mais de um grupo; **não sobrepostos:** não pode-se ter nós em comum.
- Problemas de implementação
  - **Confiabilidade:**
    validade + integridade + acordo
    validade: entregue para todos os nós do grupo.
    integridade: a mensagem não é modificada.

acordo: se uma mensagem é entregue para um nó deve ser entregue para todos.

- **Ordenação:** FIFO, Causal, Total
  FIFO: do ponto de vista de quem envia, as mensagens devem ser enviadas na ordem.
  Causal: se uma mensagem aconteceu antes de outra, a ordem deve ser preservada.
  Total: A ordem deve ser preservada em todos os processos do sistema.

Grupos Abertos X Grupos fechados

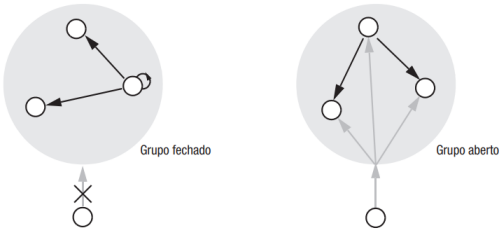

Figura 6.2 Grupos abertos e fechados.

Gerenciamento de participação

Tarefas fundamentais.

- Tarefas
  - Fornecer interface para mudanças de participação.
    as mensagens devem conseguir entrar e sair a qualquer momento.
  - Detecção de falha
    devem existir mecanismos de detecção de falha do nó para isolar ou realizar uma ação.
  - Notificar membros sobre mudanças na participação do grupo
    mecanismos para avisar caso algum nó saia ou entre.
  - Realizar expansão de endereço do grupo
    Quando uma mensagem é enviada para um grupo através de um endereço, o gerenciador deve garantir a expansão da mensagem para todos os participantes do grupo.
- O multicast IP é um caso frágil de serviço de participação
  Não há detecção de falha, nem notificação sobre os participantes.

Gerenciamento de participação

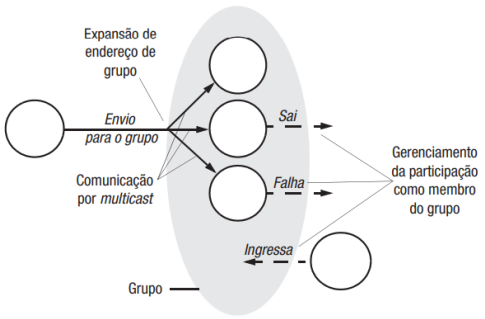

Figura 6.3 O papel do gerenciamento da participação como membro do grupo.

Estudo de caso: JGroups

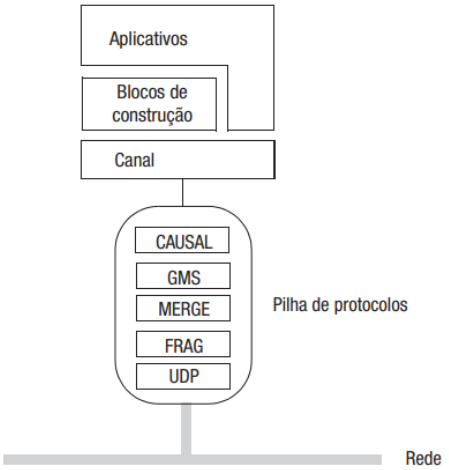

Figura 6.4 A arquitetura do JGroups.

A arquitetura do JGroups

- Canais – Interface mais primitiva para desenvolvedores.
  possibilitam interação de processo.
- Blocos de construção – Abstração de nível mais alto.
- Pilha de protocolos – Fornece um protocolo de comunicação.
  funcionalidades implementadas para garantir um protocolo de comunicação.

Canais

- Objeto que possibilita a interação de processos com grupos
- Operações: connect, desconnect e close (torna o canal inutilizável)
- Mensagens são enviadas pelos canais por multicast confiável
  garante confiabilidade na implementação do JGroups.

Gerenciamento de Participação

- getView – lista de membros atual
- getState – retorna o histórico associado ao grupo
  lista quem entra e quem sai.

Exemplo de canal

- Disparo de um alarme
  - Classe FireAlarmJG
    - Método raise()

```
FireAlarmJG alarm = new FireAlarmJG();
alarm.raise();
```

- Destinatário do alarme
    - Classe FireAlarmConsumerJG
        - Método await()

```
FireAlarmConsumerJG alarmCall = new FireAlarmConsumerJG();
String msg = alarmCall.await();
System.out.println("Alarm received: " + msg);
```

O método await bloqueia o método enquanto se espera uma mensagem.

Exemplo de canal

```
import org.jgroups.JChannel;
public class FireAlarmJG {
    public void raise( ) {
        try {
            JChannel channel = new JChannel( );
            channel.connect("AlarmChannel");
            Message msg = new Message(null, null, "Fire!");
            channel.send(msg);
        }
        catch(Exception e) {
        }
    }
}
```

```
import org.jgroups.JChannel;
public class FireAlarmConsumerJG {
    public String await( ) {
        try {
            JChannel channel = new JChannel( );
            channel.connect("AlarmChannel");
            Message msg = (Message) channel.receive(0);
            return (String) msg.GetObject( );
        }
        catch(Exception e) {
            return null;
        }
    }
}
```

Blocos de construção

- Abstrações de mais alto nível sobre a classe canal.
- Análogo a RPC e RMI
- Exemplos:
    - **MessageDispatcher:** Envia uma mensagem e bloqueia até que n respostas sejam recebidas.
    - **RpcDispatcher:** recebe um método como parâmetro e o executa em todos os membros do grupo.
      Um determinado objeto envia uma mensagem para um grupo e todos os nós que chamaram a classe RpcDispacher, ela vai disparar a execução de um método (passado por parâmetro no momento da invocação do método dessa classe) e garante que seja executada em todos os membros do grupo.