Utilizando o JGroups para Comunicação de Grupo em Dispositivos Móveis

Matheus T. Mendes, Emerson R. de Oliveira Jr.

Ciência da Computação Instituto de Ciências Exatas e Geociências
Universidade de Passo Fundo (UPF)
Caixa Postal 611 99001-970 Passo Fundo RS Brasil

bigodines@gmail.com, emerson@upf.br

Resumo. *JGroups-ME é a versão do JGroups para o ambiente móvel. Esta necessita de um roteador de mensagens executando em um desktop com o JGroups tradicional. Este artigo descreve a implementação de uma solução alternativa que elimina tal necessidade. A solução portou o roteador de mensagens para a plataforma Java Micro Edition (Java ME) e criou uma aplicação (midlet) que dispara o roteador. Tal medida, possibilitou o uso do JGroups para a criação de um sistema de comunicação em grupos totalmente móvel.*

1. Introdução

Comunicação em grupo é um paradigma de programação muito utilizado em sistemas distribuídos [Coulouris et al. 2004] e possui diversos *frameworks* que auxiliam na criação de sistemas que fazem uso desta técnica como por exemplo o JGroups [Ban, 2006]. Dentre as inúmeras vantagens da utilização deste paradigma está a transparência para o usuário da entrega de mensagens para o grupo, o que o afasta dos detalhes e da complexidade inerentes aos sistemas distribuídos e provê a possibilidade da cooperação entre os integrantes do grupo.

Nos últimos anos, a proliferação de dispositivos móveis como PDAs e *smartphones* criou um novo tipo de computação, denominada computação móvel. As linguagens, sistemas operacionais e os próprios programadores tiveram de se adaptar a dispositivos com recursos limitados e à idéia de que as redes de interconexão ficariam ainda mais instáveis, tornando a dependabilidade e a tolerância a falhas um desafio ainda maior [Hellbrück e Fischer 2004].

Houve um esforço para portar o JGroups para ambiente móvel através do JGroups-ME [Shinde, 2004], projeto que possibilita a criação de um grupo composto por nodos móveis e um roteador de mensagens rodando o JGroups tradicional. O presente trabalho dá continuidade à idéia do JGroups-ME implementando partes relevantes do JGroups que não haviam sido portadas para computação móvel resultando em *framework* para auxiliar na construção de sistemas de comunicação de grupo em ambiente totalmente móvel.

2. JGroups-ME e Definição do Protótipo

Como o JGroups não podia ser utilizado em computação móvel com a configuração MIDP2.0/CLDC1.1 pelo fato da máquina virtual Java não oferecer todas as classes necessárias à aplicação, fez-se necessário adaptar o JGroups e implementar as classes

que faltavam [Shinde, 2004]. Até então, a versão móvel do JGroups mantinha o seu foco nos nodos cliente e era necessário que um roteador de mensagens estivesse rodando em uma máquina *desktop* para que o sistema de comunicação em grupo funcionasse. Desta forma, para possibilitar a criação de um sistema de comunicação em grupo móvel, fez-se necessário portar o roteador de mensagens para a plataforma Java ME, a criação de um *midlet* que disparasse o novo roteador de mensagens e a disponibilização de uma aplicação para demonstrar o funcionamento do sistema.

2.1. Funcionamento do Protótipo

Foi utilizada uma aplicação de *chat* que permite a conversa entre todo o grupo ou a conversa reservada entre dois nodos participantes deste grupo. O protótipo foi testado no simulador Sun WTK 2.2, em ambiente Linux configurado com um perfil compatível com MIDP2.0/CLDC1.1. Em uma instância foi executado o roteador e logo após foram executados três *midlets* com o *chat* propriamente dito. O comportamento do sistema foi comparado ao comportamento do JGroups-ME com o *router* em um computador *desktop*. A figura 1 representa a estrutura do sistema de comunicação de grupo desenvolvido no protótipo.

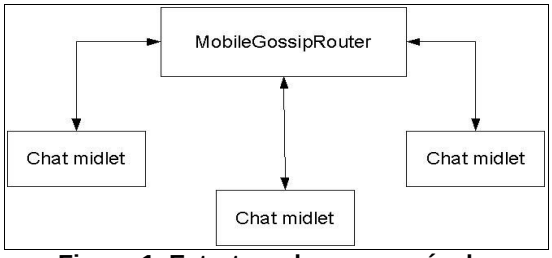

Figura 1. Estrutura do grupo móvel

2.2. MobileGossipRouter

A primeira parte da aplicação é a implementação do *router* em versão móvel. A figura 2 exibe como foi instanciado o roteador dentro do *midlet* bem como os parâmetros adotados para o mesmo.

```
37    protected void startApp() throws MIDletStateChangeException {
38        int port=7500;
39        long expiry = GossipRouter.EXPIRY_TIME;
40        long timeout = GossipRouter.GOSSIP_REQUEST_TIMEOUT;
41        long routingTimeout = GossipRouter.ROUTING_CLIENT_REPLY_TIMEOUT;
42        GossipRouter router=null;
43        String address="127.0.0.1";
44
45        System.out.println("GossipRouter is starting...");
46
47        try {
48            router= new GossipRouter(port, address, expiry, timeout, routingTimeout);
49            router.start();
50        }
51        catch(Exception e) {
52            System.out.println("Couldn't initiate the router.!");
53        }
54
55    }
```

Figura 2. Inicialização do roteador

Conforme ilustrado na figura 2, para o protótipo estão sendo utilizadas as configurações padrão do GossipRouter para o tempo de expiração das mensagens, tempo de resposta para estabelecer a conexão e o tempo limite de inatividade para os membros do grupo. É importante que a inicialização do roteador esteja dentro de um bloco *try/catch* uma vez que a porta que está tentando ser aberta pela aplicação pode estar sendo utilizada para outro recurso ou ser inacessível para o *midlet* do roteador.

A classe *GossipRouter* permanece praticamente inalterada em relação à versão original, com exceção das classes que são importadas por esta e a correção de um erro no GossipRouter original, que não tratava as mensagens *heartbeat* como atividade e acabava por desconectar os integrantes do grupo mesmo que estes estivessem operantes. Foram feitas as implementações das classes para ambiente móvel, as quais foram colocadas dentro do *package* **bridge** . A figura 3 ilustra algumas das classes que foram portadas ou estendidas para o que a aplicação funcionasse como esperado.

```
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.jgroups.Address;
import org.jgroups.conf.ClassConfigurator;
import org.jgroups.util.Util;

import EDU.oswego.cs.dl.util.concurrent.ConcurrentHashMap;
import bridge.java.io.DataInputStream;
import bridge.java.io.DataOutputStream;
import bridge.java.lang.Thread;
import bridge.java.net.InetAddress;
import bridge.java.net.ServerSocket;
import bridge.java.net.Socket;
import bridge.java.util.Iterator;
import bridge.java.util.LinkedList;
import bridge.java.util.List;
import bridge.java.util.Map;
import bridge.java.util.Timer;
```

Figura 3. Alterações no GossipRouter original

Com estas alterações, pôde-se rodar o GossipRouter em ambiente móvel para rotear as mensagens e manter um cache com a lista de integrantes do grupo. Vale ressaltar que há dezenas de classes implementadas dentro do *package* bridge que são utilizadas tanto para o lado servidor quanto para as aplicações cliente do JGroups-ME e que não estão apresentadas na figura 3.

2.3. Chat Midlet (aMidlet)

Foi utilizada a classe aMidlet que já esta já faz parte do pacote do JGroups-ME para demonstrar o funcionamento do sistema. Essa classe estende a classe JGChat que é a responsável pela criação do canal de comunicação no JGroups-ME. A pilha de protocolos é passada à classe JGChat (juntamente com os parâmetros de cada protocolo) como uma string e o canal será criado baseado nestas configurações.

3. Resultados e conclusões

O sistema comportou-se de maneira esperada e o comportamento do grupo foi o mesmo obtido quando utilizada a versão antiga do JGroups-ME ou mesmo a versão original do JGroups. Não foram feitos testes com o roteador gerenciando mais de um grupo ou com um grupo muito grande. Presume-se que o sucesso desses testes dependerá das restrições de memória dos dispositivos nos quais eles forem executados. Não foi realizado nenhum tipo de teste de *benchmark* mas acredita-se que o desempenho não

será equivalente uma vez que dispositivos móveis ainda apresentam restrições na velocidade de processamento, conforme descrito anteriormente neste artigo. Com base nas limitações dos dispositivos móveis, deduz-se que a confiabilidade de um grupo rodando totalmente em ambiente móvel é muito menor do que se o mesmo estivesse rodando em computação tradicional, especialmente no que tange ao roteador, a parte crítica do sistema. Porém, como a aplicação se comportou dentro do esperado no ambiente de testes, conclui-se que não há limites teóricos para que tal sistema possa ser desenvolvido em ambiente real e a história já comprovou que limitações de hardware tendem a ser suprimidas ao longo do tempo.

Concluiu-se, portanto, que é perfeitamente possível a criação de um sistema de comunicação em grupo em ambiente de computação móvel para que possam ser desenvolvidas aplicações com tolerância a falhas e confiabilidade na entrega de mensagens. Acredita-se que, com a chegada da tecnologia 3G e com a diminuição das restrições dos dispositivos móveis, mais investigações serão realizadas na área de sistemas distribuídos em computação móvel e em pouco tempo já haverá aplicações que façam uso deste paradigma de programação na vida quotidiana dos usuários da computação móvel.

Referências

Ban, B. (2006) Reliable Multicasting with the JGroups Toolkit , Red Hat Inc., http://www.jgroups.org/javagroupsnew/docs/manual/html/index.html .

Coulouris, G., Dollimore J., Kindberg, T. (2004) Distributed Systems: Concepts and Design , Addison Wesley.

Hellbrück, H. e Fischer, S. (2004) MINE and MILE: Improving Connectivity in Mobile Ad-Hoc Network , In: Mobile Computing and Communications Review MCCR, Vol. 8, No. 4.

Shinde, M. (2004), P2P using Group Communication over J2ME , JGroups, http://javagroups.cvs.sourceforge.net/javagroups/JGroups-ME/docs/P2P-JGME.pdf?revision=1.1&view=markup .