UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE SISTEMAS DE INFORMAÇÃO – BACHARELADO

SOFTWARE PARA COMPARTILHAMENTO DE ARQUIVOS ENTRE CELULARES USANDO A TECNOLOGIA PEER-TO-PEER ALIADA À PLATAFORMA JXME

ROGER ROBERT KOCK

BLUMENAU
2006

2006/2-[10]

ROGER ROBERT KOCK

SOFTWARE PARA COMPARTILHAMENTO DE ARQUIVOS ENTRE CELULARES USANDO A TECNOLOGIA PEER-TO-PEER ALIADA À PLATAFORMA JXME

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Sistemas de Informação — Bacharelado.

Prof. Francisco Adell Péricas, Msc - Orientador

BLUMENAU
2006

2006/2-[10]

SOFTWARE PARA COMPARTILHAMENTO DE ARQUIVOS ENTRE CELULARES USANDO A TECNOLOGIA PEER-TO-PEER ALIADA À PLATAFORMA JXME

Por

ROGER ROBERT KOCK

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Francisco Adell Péricas, Msc – Orientador, FURB
Membro:	Prof. Sérgio Stringari, Msc – FURB
Membro:	Prof. Paulo Fernando da Silva, Msc – FURB

Blumenau, 08 de dezembro de 2006

Dedico este trabalho aos familiares e amigos que me apoiaram durante todo o processo de elaboração. Dedico também àqueles que com seu conhecimento e experiência contribuíram para que os objetivos estabelecidos fossem alcançados.

AGRADECIMENTOS

Gostaria de agradecer primeiramente a Deus, por ter me dado a oportunidade de chegar à universidade e concluir esta etapa em minha vida.

Agradeço também aos meus pais, Marlene e Rubens que me apoiaram não só na execução deste trabalho, mas também em todas as etapas necessárias para que eu chegasse até aqui. E aos meus avós Elsbeth e Sido, que foram e são mais do que simples avós e sim meus segundos pais.

Ao meu orientador, Francisco Adell Péricas que se dispôs a orientar este trabalho e me auxiliou sempre com extrema paciência, atenção e compreensão nos momentos de tempestade e de bonança.

Aos amigos Jefferson, Marlei, Rodrigo e Gilbran não somente por serem ótimos companheiros de curso, mas amigos para a vida.

E a Alexandre Vilcek, Tomas Piedrahita, Mohammed Abdelaziz e Edward Ribeiro, pessoas as quais nunca tive, por impossibilidades geográficas, a oportunidade de conhecer pessoalmente, mas que me deram força e conhecimento para tornar realidade a aplicação proposta neste trabalho.

Por fim, agradeço a todas aqueles que torceram e torcem por mim.

De que adianta viver, senão para lutar por causas nobres e fazer deste mundo confuso um lugar melhor para aqueles que nele viverão depois de nós?

Winston Churchill

RESUMO

Com o advento da terceira geração de sistemas de comunicação móvel, será oferecida uma nova gama de serviços de transferência em alta velocidade de texto, áudio, vídeo e dados em geral. Nesse contexto, o presente trabalho desenvolve uma aplicação que torna possível a conexão de dois ou mais telefones celulares em uma rede *peer-to-peer* (P2P), a qual é baseada no conceito de arquitetura centralizada, onde um servidor estará no centro da rede montada e todos os celulares participantes dela se conectarão ao mesmo. Este software é construído utilizando a plataforma JXME, que provê o suporte e os protocolos necessários para que uma rede deste tipo seja formada. Aliada à mesma, é utilizada a linguagem J2ME, especialmente projetada para execução em celulares e outros dispositivos móveis.

Palavras-chave: JXTA. *Peer-to-peer*. JXME. J2ME. Celular.

ABSTRACT

With the advent of the third generation of mobile communication systems, it will be offered several new services of high speed transference of text, audio, video and data in general. In this context, the present paper develops an application that makes possible a connection between two or among more cellphones in a peer-to-peer (P2P) network, which is based in the centralized architecture concept, in which a server is in the center of the organized network and all the participating cellphones will be connected to it. This software is built using the JXME platform, which provides the necessary support and protocols for a network of this kind work. Allied to it, it is used the J2ME language, specially designed to be executed in cellphones and other mobile devices.

Key-words: JXTA. Peer-to-peer. JXME. J2ME. Cellphone.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

P2P – *Peer-to-Peer*

URN – *Uniform Resource Names*

XML – *eXtensible Markup Language*

UDP – *User Datagram Protocol*

TCP – *Transmission Control Protocol*

HTTP – *Hypertext Transfer Protocol*

SMTP – *Simple Mail Transfer Protocol*

CLDC – *Connected Limited Device Configuration*

CDC – *Connected Device Configuration*

MIDP – *Mobile Information Device Profile*

JSR – *Java Specification Requests*

UML – *Unified Modeling Language*

IP – Internet *Protocol*

3G – *Third Generation*

GPRS – *General Packet Radio Service*

ID – Identificador

LBS – *Location Based Services*

SUMÁRIO

1 INTRODUÇÃO

Cada vez mais, os celulares transformam-se em um fenômeno de vendas ao redor do planeta. Somente no Brasil eles já somam mais de 75 milhões de unidades (MINISTÉRIO DA CIÊNCIA E TECNOLOGIA, 2005). Seu sucesso abre portas para que estes aparelhos deixem de ser simples telefones e ganhem atribuições antes pensadas apenas para computadores, com a vantagem de serem extremamente portáteis.

Em proporções similares, visualiza-se uma utilização cada vez mais intensa da tecnologia *Peer-to-Peer* (P2P) no mundo da computação doméstica e corporativa. Isto é comprovado por vários indicadores, como o fato de que ela ocupa atualmente de 40% a 60% de todo o tráfego da Internet (DURANTE, 2004).

Portanto, individualmente, ambas têm tido grande sucesso. Mas, e se elas fossem combinadas? Esta simples questão possibilita a abertura de uma impressionante gama de possibilidades e de novas funcionalidades para os aparelhos celulares, dentre elas compartilhar arquivos mundialmente entre usuários deste tipo de dispositivo.

De acordo com Handford (2005), o mercado de celulares está abrindo-se para a transmissão de mídias e a computação P2P, a qual, caracterizada por sua habilidade de criar, aproximar, interagir com grupos e disseminar informações, poderá proporcionar ambientes mais dinâmicos às redes móveis atuais.

Sendo assim, vislumbrando a aurora de um novo “estilo de vida” em que a necessidade do ser humano de permanecer informado a qualquer hora e em qualquer lugar passa a ser fundamental, este trabalho desenvolveu um aplicativo P2P de compartilhamento de arquivos entre celulares com o qual será possível, de maneira inédita, trocar arquivos e conseqüentemente, informações, entre estes dispositivos como se eles fossem computadores convencionais, mas sem a existência de entraves quanto a mobilidade dos dados carregados.

1.1 OBJETIVOS

O objetivo deste trabalho foi construir uma aplicação P2P para celulares, que viabilizará a conexão e a comunicação direta entre estes dispositivos.

Os objetivos específicos do trabalho são:

a) conectar dois aparelhos celulares através do simulador Sun Java Wireless Toolkit;
b) permitir trocar arquivos entre dois celulares conectados;
c) utilizar as bibliotecas da plataforma JXME e os recursos da linguagem J2ME.

1.2 ESTRUTURA DO TRABALHO

No capítulo um é apresentada uma introdução ao tema abordado, bem como os objetivos que foram traçados para o trabalho.

No capítulo dois são dissertadas as tecnologias necessárias para a construção de um aplicativo que compartilhe arquivos em uma rede P2P entre telefones celulares.

Já no capítulo três são mostrados os aspectos técnicos referentes ao desenvolvimento do trabalho, descrevendo a especificação e a implementação do aplicativo. Também são abordadas algumas características do software desenvolvido, baseadas nos conceitos já apresentados no capítulo dois.

No capítulo quatro são descritos os testes de *performance* realizados com a aplicação.

Por fim, no capítulo cinco são descritas as conclusões e dificuldades encontradas na confecção do software, bem como sugestões para futuras pesquisas nesta área.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas as tecnologias necessárias para que um software P2P de compartilhamento de arquivos para celulares possa ser construído. Em cada uma das seções são mostradas as alternativas que as tecnologias utilizadas apresentam para o desenvolvimento de aplicações móveis.

2.1 REDES P2P

O conceito de P2P está cada vez mais sendo explorado por indústrias e pela mídia. Muito do interesse atual por este tipo de rede é ocasionado devido ao apelo social, que popularizou produtos como o Napster, Gnutella, KaZaA, Skype, entre outros. Estas aplicações colocaram a tecnologia P2P ao alcance de milhares de usuários, mostrando o poder de seu paradigma (DURANTE, 2004).

Além dos softwares de compartilhamento de arquivos previamente citados, a rede P2P também é utilizada nas aplicações de mensagens instantâneas como o *I Seek You* (ICQ) e *Microsoft Network* (MSN) *Messenger*. Mas é possível definir P2P como algo muito mais simples que isso: uma rede P2P pode ser composta apenas por dois computadores que estão conectados e compartilham recursos entre si sem a interferência direta de um servidor (DOCHSTADER, 2001, tradução nossa).

Este conceito deixa bastante claro qual o objetivo da tecnologia P2P, mas ao observar com mais profundidade, percebe-se que muitas questões sobre o desenvolvimento de aplicativos dessa natureza surgem (WILSON, 2002):

a) como um dos *peers* da rede “vê“ a presença de outro?
b) como os *peers* podem ser organizados em grupos de interesse comum?
c) como um *peer* publica seus recursos (arquivos a compartilhar ou serviços oferecidos)?
d) como é possível identificar de maneira única os *peers* presentes na rede?
e) como trocar dados entre dois ou mais *peers*?

Muitas soluções foram criadas para responder estas questões. Mas a maior deficiência delas é que cada uma aborda o problema de uma forma muito específica, dificultando a compreensão e a implementação da mesma.

Uma proposta para solucionar isto, foi a criação da plataforma JXTA que é detalhada neste trabalho em uma seção posterior.

2.1.1 Arquiteturas P2P

Segundo Lv (2002, tradução nossa) existem 3 tipos principais de arquitetura P2P:

a) arquitetura descentralizada;
b) arquitetura híbrida;
c) arquitetura centralizada.

Na arquitetura descentralizada não existe um ponto central de controle (servidor) como na arquitetura centralizada que é descrita a seguir. Os *peers* são autônomos e completamente interligados, podendo assim se comunicar diretamente uns com os outros. Um exemplo desta arquitetura é o aplicativo KaZaA.

Já na arquitetura híbrida, que é a abordagem intermediária entre a arquitetura centralizada e descentralizada, o usuário tem a impressão de que existem dois níveis de *peers*. Isto se deve ao fato de que nesta arquitetura existe um *peer* global que mantém um certo controle sobre os *peers* que estão ligados a ele. O conceito geral se resume a várias sub-redes P2P e um *peer* controlador de cada sub-rede. O exemplo deste tipo de arquitetura é o software Skype.

Por fim, a arquitetura centralizada utiliza, como o próprio nome sugere, um servidor central o qual mantém diretórios dos arquivos de cada *peer*. Quando um cliente conecta-se nesse tipo de rede, seu diretório é atualizado e qualquer mensagem de controle ou de busca é enviada ao servidor central, que cruza as referências de uma busca com os dados de seu banco de dados e envia a resposta ao cliente, conforme pode ser visto na Figura 1. A partir daí o cliente pode contatar os *peers* diretamente.

A grande vantagem dessa arquitetura é a *performance* nas buscas, já que elas são repassadas apenas ao servidor central de toda a rede. Como exemplo deste tipo de arquitetura pode ser citado o Napster.

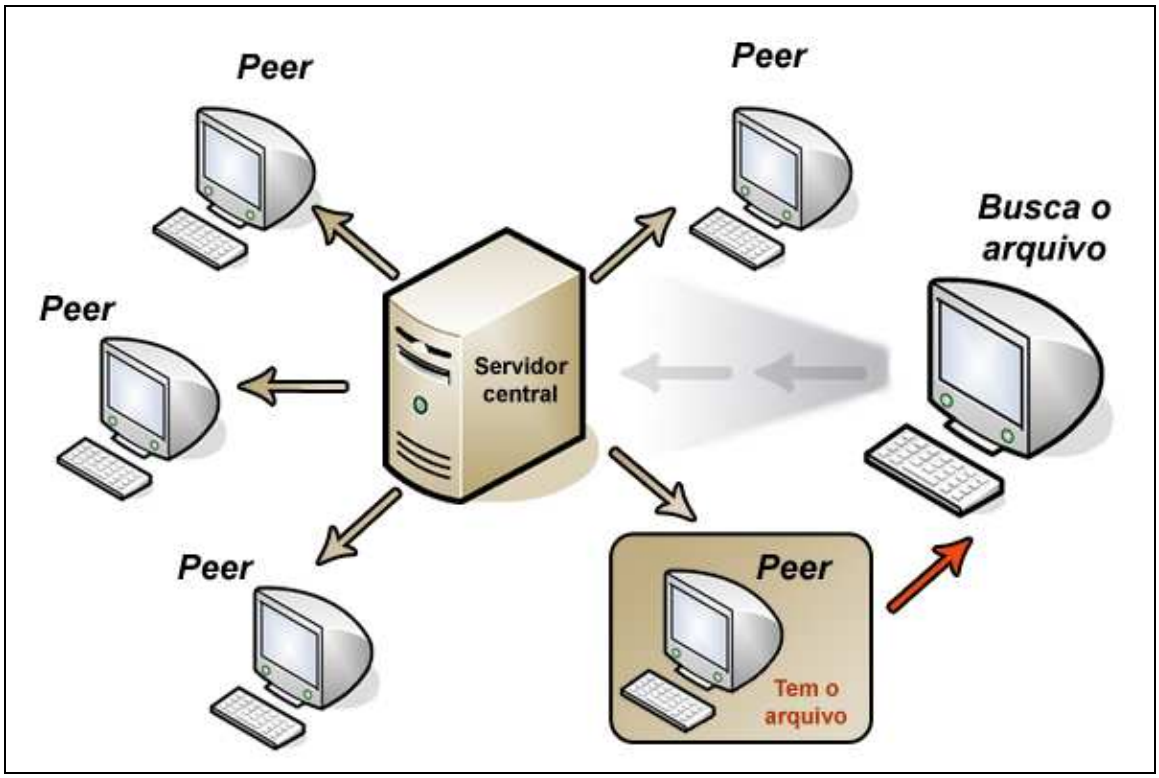


Figura 1 – Arquitetura centralizada

2.2 PLATAFORMA JXTA

O nome JXTA vem da palavra *juxtaposition*, ou justaposição em português. Segundo Vilcek (2006, p. 63) “isto se deve ao fato de esta tecnologia criar um modelo de arquitetura onde todas as entidades participantes são “clientes” e ao mesmo tempo “servidores”, numa configuração muito mais orientada à comunicação.”

JXTA é uma arquitetura *open source* que define um conjunto de protocolos para as funcionalidades requeridas em uma rede P2P, independentemente do sistema operacional, linguagem de programação e tipo de transporte empregado na mesma (JXTA PROGRAMMER GUIDE, 2005, tradução nossa).

Os objetivos que direcionam este conceito e o desenvolvimento da tecnologia JXTA são (JXTA, 2006, tradução nossa):

a) interoperabilidade: a tecnologia é desenhada para permitir que os *peers* comuniquem-se, independentemente do sistema P2P utilizado;

b) independência de plataforma: o JXTA não depende de plataforma de hardware ou sistema operacional;

c) ubiqüidade: o projeto JXTA é estruturado de forma que possa ser executado em qualquer dispositivo que possua alguma capacidade de processamento e comunicação.

Sendo assim, utilizando os protocolos desta plataforma é possível criar uma rede virtual P2P sobre a rede física da Internet. Os *peers* podem trocar mensagens independentemente da infra-estrutura da rede e do protocolo de transporte. A Figura 2 mostra o mapeamento de uma rede virtual de JXTA comparado com a rede que compõe a Internet:

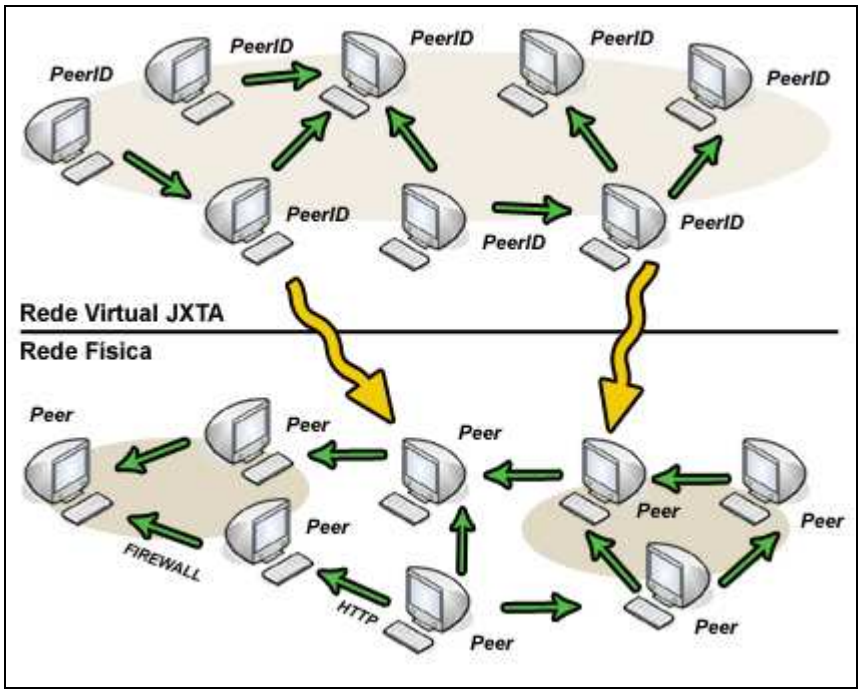


Figura 2 – Rede virtual de JXTA

Atualmente existem três implementações da plataforma de JXTA:

a) JXTA-Java SE;

b) JXTA-C/C++;

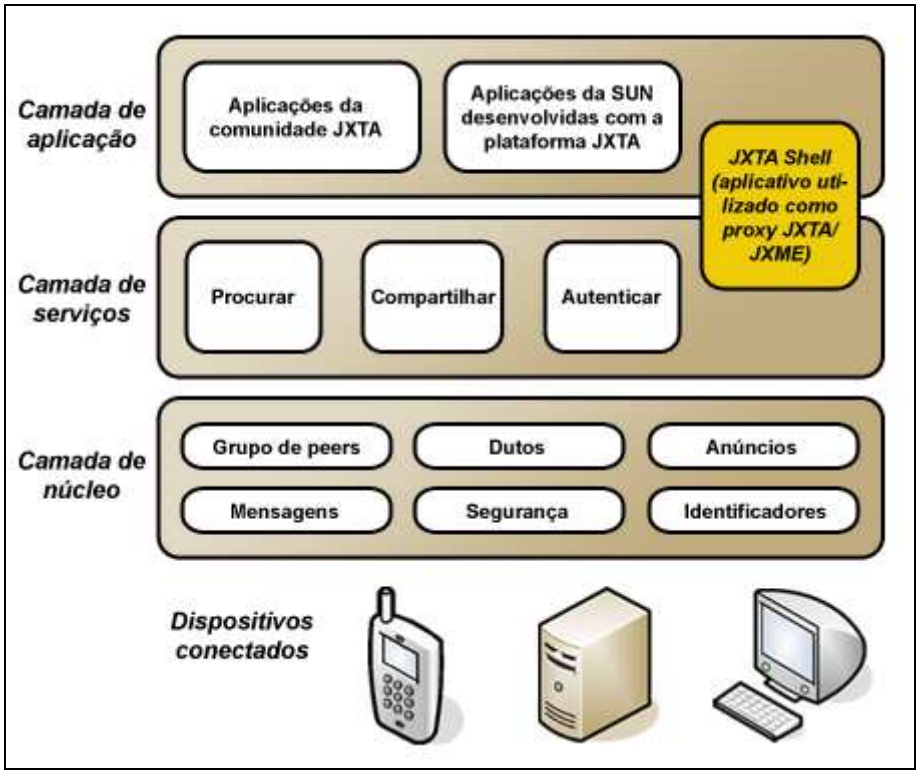
c) JXTA-JAVA ME (JXME).

Sendo este último utilizado para a construção do software proposto neste trabalho.

2.2.1 Camadas JXTA

De acordo com Wilson (2002, tradução nossa), a plataforma JXTA é dividida em três camadas, conforme pode ser visto na Figura 3:

a) camada núcleo: provê os elementos essenciais para uma solução P2P, sendo que é nessa camada que são implementados os protocolos da plataforma JXTA;
b) camada de serviços: consiste nos serviços de rede que são desejáveis, mas não são uma parte obrigatória de uma solução P2P, dentre eles:
   - procurar por recursos em um *peer*,
   - compartilhar arquivos de um *peer*,
   - executar autenticação de *peers*;
c) camada de aplicação: nesta camada são usados os recursos da camada de serviço para prover uma interface onde um usuário pode invocar funcionalidades desta camada.



Fonte: adaptado de P2P Architectures (2003, tradução nossa)

Figura 3 – Camadas lógicas de JXTA

O aplicativo JXTA Shell, identificado na Figura 3 como integrante das camadas de aplicação e de serviços, é utilizado como *proxy* pelas aplicações JXTA, sendo detalhado no capítulo referente ao desenvolvimento do trabalho.

2.2.2 Terminologia

Esta é uma introdução à terminologia e aos conceitos da plataforma JXTA e sua relação com a estrutura comum a todas as redes P2P. A plataforma JXTA usa um conjunto de termos que necessitam ser entendidos para que a estrutura das aplicações P2P possa ser também compreendida.

2.2.2.1 Identificadores (JXTA ID)

De acordo com JXTA PROGRAMMER GUIDE (2005 apud VOSS, 2004), todos os recursos de uma rede JXTA (*peers*, grupos de *peers*, anúncios e dutos) são assinados por um identificador (ID) JXTA único. Para expressar os identificadores JXTA, são usados os *Uniform Resource Names* (URN), os quais são representados em um arquivo texto no formato *eXtensible Markup Language* (XML), sendo este reconhecido por toda a plataforma JXTA. O Quadro 1 mostra dois exemplos de JXTA IDs:

Identificador do grupo de *peers* JXTA
`urn:jxta:uuid-8E0CAE9528CE4A7D8778F0762F712D6002`
 	
Identificador de um anúncio de um grupo de *peers* JXTA
`urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306`

Quadro 1 – Identificadores de elementos da plataforma JXTA

Todos os IDs são gerados aleatoriamente pela plataforma JXTA quando cada elemento da rede é criado.

2.2.2.2 *Peers*

Wilson (2002, tradução nossa) define *peers* como sendo qualquer entidade capaz de fazer algum trabalho útil e comunicar os resultados a outra entidade através da rede. Ou seja, eles são os nós numa rede P2P.

Na plataforma JXTA não é necessário que todos os nós de uma rede sejam computadores, visto que a definição da plataforma sugere que a aplicação P2P esteja espalhada por diversos tipos de dispositivos, como por exemplo, aparelhos celulares. Isto é possível pois cada nó opera de forma independente sendo identificado unicamente através de um JXTA ID.

Em uma rede P2P, os *peers*, dependendo de suas funcionalidades e capacidades, podem ser classificados em três tipos:

a) *peers* simples: tem pouca responsabilidade na rede P2P, podem apenas enviar e receber mensagens, servindo a um usuário final. Dispositivos como celulares estão inclusos nesta categoria;
b) *peers* de encontro: também denominados *rendezvous peers*, eles fornecem informações sobre a localização de outros *peers* e seus respectivos serviços na rede. São usados para propagar mensagens e anúncios dentro de um grupo de *peers* e permitem que *peers* simples de uma rede privada comuniquem-se com outros membros do grupo fora desta. Apenas não permitem a comunicação através de *firewalls*, ficando esta tarefa a cargo dos *peers relay*;
c) *peers relay*: destinam-se a prover mecanismos de comunicação com *peers* separados por um *firewall* ou por máscaras de rede. Ou seja, são capazes de definir rotas para mensagens destinadas a outros *peers* da rede. Também são usados como *proxy* pelos *peers* simples (com menos capacidade de processamento), realizando todo o processamento intensivo de descoberta de *peers*, grupos, criação de grupos e dutos de comunicação (VILCEK, 2006).

2.2.2.3 Grupos de *peers*

O principal objetivo dos grupos de *peers*, de acordo com Wilson (2002, tradução nossa), é separar o ambiente JXTA em partes menores e mais privadas, o que se torna necessário considerando que todas as aplicações P2P vão compartilhar os mesmos protocolos na rede. Por exemplo, um aplicativo de mensagens instantâneas poderá usar um grupo e um programa de compartilhamento de arquivos usará outro. Traversat et al. (2003, tradução nossa) identifica três razões para que os grupos de *peers* sejam utilizados:

a) interesse mútuo: *peers* que conectam na rede com um mesmo objetivo e com uma mesma aplicação ou serviço, provavelmente desejarão formar um grupo para manter o serviço privado entre os membros, evitando tráfego de rede desnecessário;
b) segurança: um grupo de *peers* pode criar restrições de uso, implementando serviços de autenticação. Isso limita o acesso ao grupo e seus serviços;
c) monitoramento: grupos de *peers* permitem que seus membros sejam monitorados quanto ao seu *status* (conectado ou desconectado) e quantidade de informações que os mesmos estão trafegando na rede.

2.2.2.4 Anúncios

Todos os recursos de uma rede JXTA como *peers*, grupos de *peers*, dutos e serviços são representados por anúncios. De acordo com Traversat et al. (2003, p. 3, tradução nossa) “anúncios são descritores estruturados de recursos, independentes de linguagem, representados como documentos XML”. O Quadro 2 ilustra este conceito:

```
<!DOCTYPE jxta:PGA>
<jxta:PGA xmlns:jxta="http://jxta.org">
 <GID>
   urn:jxta:uuid-8E0CAE9528CE4A7D8778F0762F712D6002
 </GID>
 <MSID>
   urn:jxta:uuid-DEADBEEFDEAFBABAFEEDBABE000000010306
 </MSID>
 <Name>
   SIMCAGROUP
 </Name>
 <Desc>
   Anúncio do grupo de peers SIMCA.
 </Desc>
</jxta:PGA>
```

Quadro 2 – Estrutura de um anúncio de um grupo de *peers*

No exemplo citado, é possível a visualização dos identificadores:

a) <GID>: é o identificador unívoco do grupo de *peers*;
b) <MSID>: é o identificador do anúncio que descreve todos os serviços disponíveis no grupo de *peers*;
c) <Name>: nome do grupo de *peers*;
d) <Desc>: Descrição opcional do grupo de *peers*.

2.2.2.5 Transporte de dados

Tendo como principal objetivo a troca de dados entre os *peers*, um sistema P2P utiliza-se de uma camada denominada rede de transporte, a qual é responsável por todos os aspectos de transmissão de dados, incluindo a segmentação de pacotes e a adição de cabeçalhos apropriados para controlar o destino dos mesmos. O protocolo de transporte não é fixo, podendo ser de baixo nível como o *User Datagram Protocol* (UDP) ou o *Transmission Control Protocol* (TCP), ou alto nível como o *Hypertext Transfer Protocol* (HTTP) (utilizado pelo protocolo de ligação de dutos, que é detalhado na próxima seção) ou o *Simple Mail Transfer Protocol* (SMTP) (WILSON, 2002, tradução nossa).

Para lidar com os dados que irá transportar, a camada de transporte utiliza-se dos seguintes recursos:

a) dutos (*pipes*): os dutos são canais de comunicação virtuais usados para mandar e receber mensagens. Eles ligam um ou mais pontos de fim (*endpoints*) e podem ser classificados em dutos de saída de dados (*output pipes*) e dutos de entrada de dados (*input pipes*);
b) pontos de fim (*endpoints*): são, de maneira simplificada, os *peers* da rede. Como citado anteriormente, eles são ligados através de dutos que transportam mensagens de um *peer* a outro;
c) mensagens: contém os dados a serem transmitidos de um *peer* (*endpoint*) a outro.

Sendo assim, conforme a Figura 4, os *peers* são conectados por dutos de entrada e saída de dados, os quais por sua vez são identificados por um JXTA ID único, para que sejam facilmente encontrados na rede.

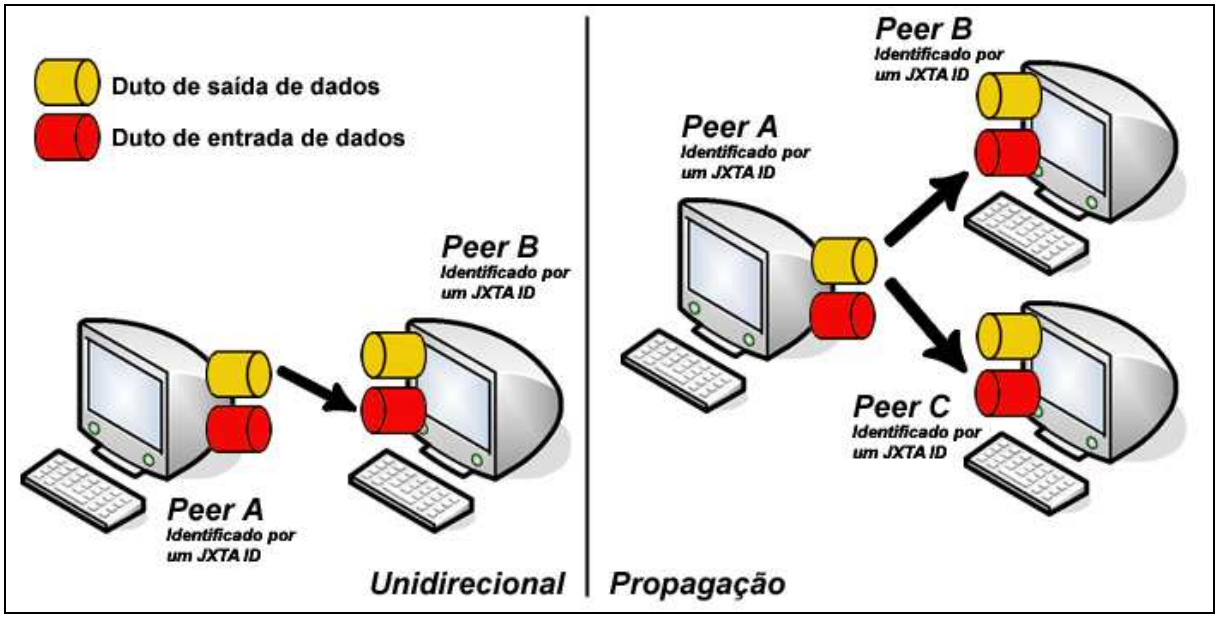


Figura 4 – Funcionamento dos dutos

Mas além de serem classificados em *output* e *input pipes*, os dutos também são classificados em dois tipos de implementações diferentes (WILSON, 2002, tradução nossa):

a) Unidirecionais (*unicast pipe*): conecta dois dutos (um duto de entrada e o outro de saída de dados) através de um canal unidirecional e assíncrono;
b) Propagação (*propagating pipe*): conecta um duto de saída à múltiplos dutos de entrada.

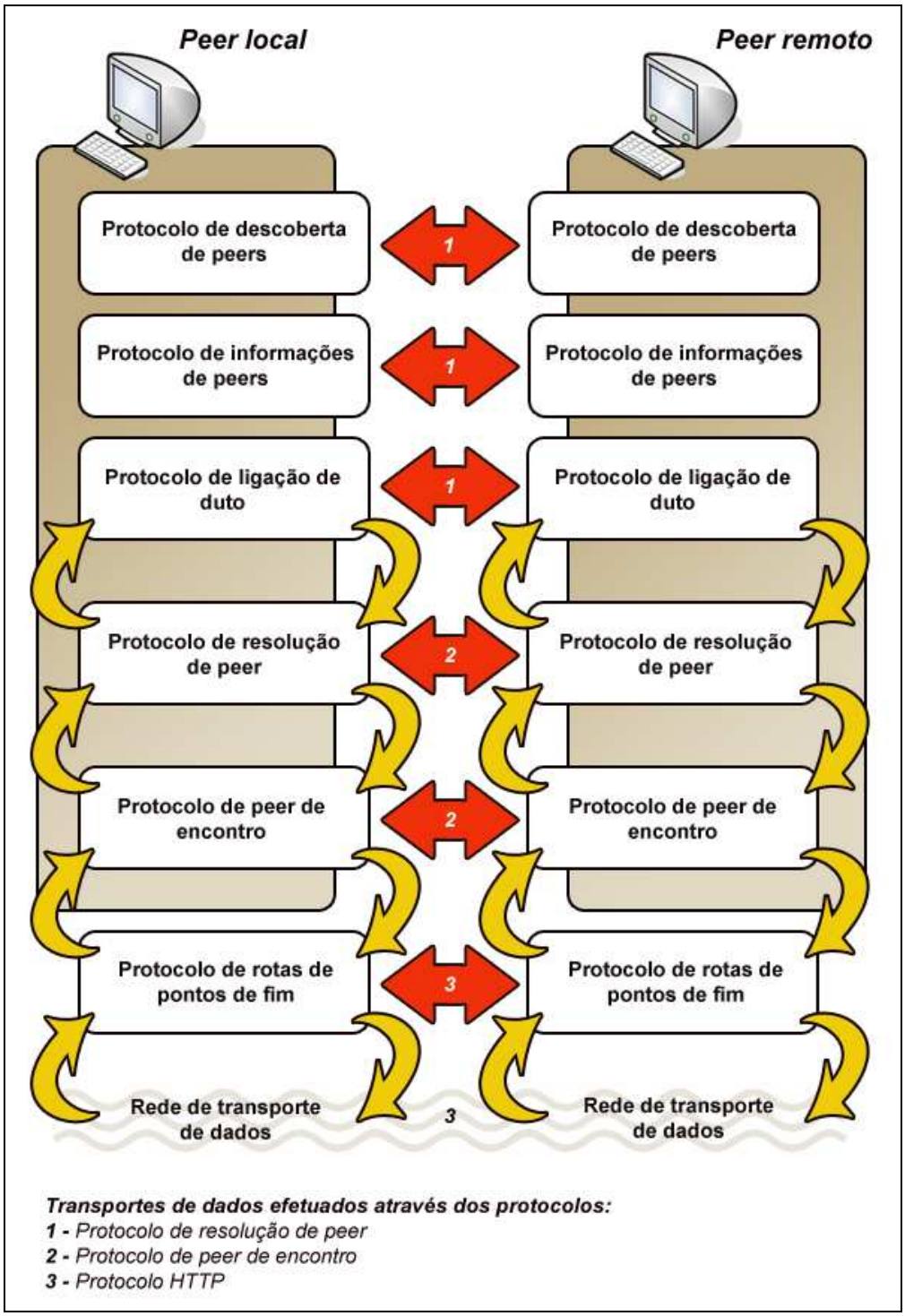
2.2.2.6 Segurança

A rede JXTA, por ser uma rede P2P dinâmica, necessita de diferentes níveis de segurança de acesso aos recursos que disponibiliza. Sendo assim, os *peers* JXTA operam em modelos baseados em papéis, onde cada *peer* possui níveis de privilégio diferentes na rede, ou seja, cada um pode executar tarefas diferentes. Abaixo seguem os cinco princípios básicos de segurança da plataforma JXTA (JXTA PROGRAMMER GUIDE, 2005 apud VOSS):

a) sigilo: garante que conteúdos de mensagens não são abertos sem autorização;
b) autenticação: garante que o remetente que solicita alguma informação realmente é quem diz ser;
c) autorização: certifica que o remetente possui autorização para enviar mensagens;
d) integridade dos dados: garante a integridade da mensagem durante o caminho ao destino;
e) refutabilidade: certifica que a mensagem foi transmitida por um remetente corretamente identificado e que a mesma não é o reenvio de uma mensagem previamente transmitida.

2.2.2.7 Protocolos JXTA

De acordo com Traversat et al. (2003, tradução nossa), a plataforma JXTA é composta por seis protocolos que são baseados em mensagens XML. Cada protocolo cobre um aspecto fundamental de uma rede P2P, e é dividido em uma parte responsável pelo *peer* local, e outra pelo *peer* remoto, conforme a Figura 5.



Fonte: adaptado de JXTA Specification (2006, tradução nossa)

Figura 5 – Pilha de protocolos JXTA

De acordo com Wilson (2002, tradução nossa), a parte responsável pelo *peer* local gera mensagens e as envia ao *peer* remoto. Já no *peer* remoto a mensagem é processada para uma determinada tarefa. Desta maneira, cada protocolo é semi-independente um do outro.

Os seis protocolos da plataforma JXTA podem ser divididos em duas categorias:

a) protocolos de especificação do núcleo;

b) protocolos padrão.

A categoria de protocolos de especificação do núcleo é composta por protocolos que são essenciais a qualquer solução P2P, sendo eles (WILSON, 2002, tradução nossa):

a) protocolo de resolução de *peers*: este protocolo permite o envio de uma mensagem genérica para outro *peer* e processa uma resposta genérica para uma requisição;
b) protocolo de rotas de pontos de fim: provê um mecanismo para determinar a rota de uma mensagem entre dois *peers* na rede. Para descobrir esta rota, o protocolo envia uma mensagem ao *peer relay* disponível, o qual a recebe e, se souber a rota, envia uma resposta, caso contrário pergunta a um outro *peer relay*, até que um deles responda positivamente.

Já a categoria de protocolos padrão é composta por protocolos que são opcionais para aplicativos P2P, porém seu uso é altamente recomendável para uma maior interoperabilidade com outros aplicativos semelhantes (TRAVERSAT et al., 2003, tradução nossa):

a) protocolo de descoberta de *peers*: através dele, os *peers* descobrem os recursos disponíveis na rede enviando requisições a outros membros da mesma, que geralmente são *peers* de encontro (*rendezvous peers*), os quais fornecem informações sobre a localização de outros *peers* e seus respectivos serviços na rede. O protocolo de descoberta de *peers* vai definir como os *peers* requisitam notificações e respondem requisições;
b) protocolo de encontro: descreve como as mensagens serão propagadas para outros *peers* através de um *peer* de encontro. Para que isso aconteça, cada *peer* conecta-se a um *peer* de encontro e cria um "contrato", o qual especifica o tempo que o *peer* poderá usá-lo para propagar suas mensagens, antes que este precise renová-lo. Para que estas transações funcionem corretamente, o protocolo de encontro dispõe de três tipos de mensagens:
    - mensagem de requisição de contrato: é enviada quando um *peer* requisita um contrato ao *peer* de encontro,
    - mensagem de contrato aprovado: é enviada pelo *peer* de encontro para aprovar o contrato, e determinar o tempo de validade do mesmo,
    - mensagem de cancelamento de contrato: é enviada por um *peer* para se desconectar do *peer* de encontro;
c) protocolo de informações de dutos: permite que um *peer* obtenha informações sobre o *status* de outros *peers* da rede previamente descobertos. Essas informações podem ser, por exemplo, o tempo que um *peer* está conectado, ou então a quantidade de

tráfego por ele processada. O protocolo de informações de dutos não é um dos protocolos obrigatórios da plataforma JXTA, sendo que ele só precisa ser implementado quando um *peer* necessita monitorar o *status* de um *peer* remoto para tomar decisões de gerenciamento do mesmo;

d) protocolo de ligação de dutos: estabelece como devem ser criadas as conexões entre *peers* conectados a um determinado duto e como serão mandadas as informações. Esse protocolo define o processo de ligação de um duto a um ponto de fim, utilizando-se das seguintes mensagens (LIMA, 2005):
   - mensagem de pergunta de ligação de dutos: é enviada para perguntar a um *peer* se ele está ligado a um duto com o mesmo ID,
   - mensagem de resposta de ligação de dutos: resposta da requisição;

   Observação: para o transporte dos dados pela rede P2P, é utilizado o protocolo HTTP, sendo o fato motivador da utilização do mesmo, a sua capacidade de transpor barreiras como *firewalls* e permitir transferência de informações a dispositivos móveis como celulares. Possui apenas a desvantagem de não permitir a transmissão de dados a vários *peers* simultaneamente (*broadcast*).

2.3 APARELHOS CELULARES

Os aparelhos celulares tornam-se cada vez mais necessários em nossas vidas. Eles estão sempre conosco e sempre ligados, muitas vezes servindo até para propósitos muito além de um simples telefone e sim quase uma extensão da personalidade e do modo de vida de cada pessoa.

Com o avanço no desenvolvimento de celulares mais modernos, percebe-se a fusão de diversas tecnologias do nosso cotidiano em um único dispositivo. Porém, mesmo com um avanço incrivelmente rápido, os aparelhos celulares ainda possuem limitações bastante acentuadas.

2.3.1 Limitações

Se comparado com um computador convencional, um aparelho celular ainda é um dispositivo restrito em desempenho. Para uma melhor compreensão, no Quadro 3 são listados alguns modelos de telefones disponíveis atualmente no mercado junto de suas respectivas fotos e limitações nos seguintes aspectos:

a) tamanho do *display*;
b) número de cores reproduzidas pelo *display*;
c) tamanho da memória *Random Access Memory* (RAM);
d) tamanho da memória persistente (destinada ao armazenamento de arquivos);
e) tamanho máximo suportado para uma aplicação em Java;
f) processador;
g) duração da bateria.

Nokia 6600				
	**Tamanho *display***	176x208 px.	**Tamanho aplicação**	Dinâmico
	Cores *display*	65536	**Processador**	104 MHz
	Tamanho RAM	Dinâmico	**Duração bateria**	240h (repouso)
	Tamanho memória	38 MB		4h (conversação)
Nokia 6680				
	**Tamanho *display***	176x208 px.	**Tamanho aplicação**	Dinâmico
	Cores *display*	262144	**Processador**	120 MHz
	Tamanho RAM	Dinâmico	**Duração bateria**	240h (repouso)
	Tamanho memória	10 MB		6h (conversação)
Sony Ericsson K700				
	**Tamanho *display***	176x220 px.	**Tamanho aplicação**	300 KB
	Cores *display*	65536	**Processador**	160 MHz
	Tamanho RAM	1,5 MB	**Duração bateria**	360h (repouso)
	Tamanho memória	32 MB		7h (conversação)
Motorola V220				
	**Tamanho *display***	128x128 px.	**Tamanho aplicação**	100 KB
	Cores *display*	65536	**Processador**	90 MHz
	Tamanho RAM	800 KB	**Duração bateria**	226h (repouso)
	Tamanho memória	2 MB		5h (conversação)

Fonte: adaptado de GSM Arena (2006, tradução nossa)

Quadro 3 – Comparação de aparelhos celulares disponíveis no mercado

2.4 A LINGUAGEM J2ME

Com a grande demanda para desenvolvimento de novas aplicações voltadas aos celulares, as empresas que fabricam estes aparelhos deixaram de ser as únicas responsáveis por produzir os softwares contidos nos mesmos. Estas então, precisaram encontrar um meio de permitir que outras organizações pudessem desenvolver também aplicativos para celulares (PINHEIRO, 2003).

A linguagem Java 2 Micro Edition (J2ME) foi a primeira iniciativa neste sentido, sendo apresentada pela primeira vez na Conferência de Desenvolvedores JavaOne, em meados de 1999. O objetivo desta nova tecnologia era justamente ser utilizada nos novos aparelhos celulares que estavam invadindo o mercado na época (GOMES, 2006).

Com esta solução adotada, permitiu-se em menos tempo o desenvolvimento de um número muito maior de aplicações, devido à grande disponibilidade de programadores que conheciam a linguagem Java e que poderiam migrar facilmente, para J2ME (PINHEIRO, 2003).

2.4.1 Configurações J2ME

Atualmente existem duas configurações na linguagem J2ME. A primeira e mais restrita é o *Connected Limited Device Configuration* (CLDC), que define um ambiente de execução e um conjunto de rotinas específicas para ambientes mais restritos que possuam capacidade mínima de processamento, fonte de energia limitada, *display* reduzido e conectividade intermitente, como é o caso dos aparelhos celulares e a maioria dos dispositivos móveis existentes no Brasil (GOMES, 2006).

Mais especificamente, o que a configuração CLDC necessita para funcionar corretamente é um dispositivo com as seguintes características (JSR-139, 2006):

a) ao menos 160 *kilobytes* (KB) de memória disponível para a aplicação Java;
b) processador com velocidade entre 8 e 32 MHz;
c) processador de 16 ou 32 bits;
d) energia limitada, usualmente movido à bateria;

e) conectado a uma rede, com banda de transmissão de dados limitada (9600 bps ou menos);

f) suportar, para acesso a arquivos, a *Java Specification Requests* (JSR) 75.

A segunda configuração da linguagem J2ME é a *Connected Device Configuration* (CDC), que especifica um conjunto de tecnologias destinadas à utilização em dispositivos mais potentes que aqueles contemplados na configuração CLDC. Estes dispositivos, por sua vez, devem contemplar os seguintes requisitos (JSR-36, 2006):

a) ao menos 2 *megabytes* de memória disponível para a aplicação Java;

b) processador de 32 bits;

c) estar conectado a uma rede;

d) suportar, para acesso a arquivos, a JSR 75.

A JSR 75, citada como requisito das configurações CLDC e CDC, torna-se essencial apenas quando a aplicação a ser executada necessita acessar/criar arquivos na memória, sendo que ela define de que maneira arquivos e diretórios devem ser acessados/criados em um dispositivo através da linguagem J2ME (JSR-75, 2006).

2.4.2 MIDP

Como já descrito, a linguagem J2ME pode ser dividida em duas configurações (CLDC e CDC). Estas configurações, por sua vez, também podem ser divididas em perfis (*profiles*).

Os *profiles* nada mais são do que definições da interface de um dispositivo, incluindo especificações de bibliotecas Java disponíveis para os desenvolvedores de aplicativos do mesmo (JSR-37, 2006, tradução nossa).

Diferentemente das configurações, os *profiles* não são aceitos como padrões mundiais. Isto ocorre pelo fato de existirem muitos *profiles* diferentes, algumas vezes até o mesmo fabricante de aparelhos adota perfis distintos em seus modelos.

Porém, apesar de não existir um consenso sobre os mesmos, entre os celulares um perfil amplamente utilizado é o *Mobile Information Device Profile* (MIDP).

Assim, um aparelho celular define suas capacidades Java através de uma configuração e um perfil, e estes quando são iguais entre dispositivos de fabricantes como Nokia, Siemens e LG indicam que programar para um celular fabricado pelas mesmas é exatamente igual, pois a

aplicação construída terá de funcionar de maneira idêntica nos aparelhos das três empresas citadas.

2.5 JXTA PARA J2ME (JXME)

Desde a criação do JXTA, vários projetos derivaram de seu conceito, sendo um dos principais exemplos a plataforma JXME.

> O projeto de desenvolvimento da tecnologia JXTA para dispositivos móveis foi iniciada pela Sun Microsystems, e rapidamente transferido para a comunidade de desenvolvimento JXTA, dada a percepção geral da comunidade de que esta tecnologia poderia ter conseqüências importantes no desenvolvimento da computação para dispositivos móveis como telefones celulares e PDAs (VILCEK, 2006, p. 63).

A linguagem adotada para a plataforma JXME foi o J2ME, que é uma versão da linguagem Java voltada aos dispositivos móveis. De acordo com Vilcek (2006), ela foi utilizada devido a sua já consagrada aceitação como padrão entre os diversos fabricantes de aparelhos e também entre a comunidade de desenvolvedores.

Da mesma maneira que o JXTA, a plataforma JXME também possui objetivos, os quais são (VILCEK, 2006):

a) criar uma infra-estrutura para computação P2P adequada aos dispositivos móveis;
b) proporcionar interoperabilidade com JXTA em outras plataformas como estações de trabalho e servidores;
c) ser pequena o suficiente para ser utilizada nos dispositivos que implementam J2ME (a grande maioria sendo celulares, que possuem fortes restrições de memória e processamento);
d) ser compatível com MIDP, o perfil da plataforma J2ME mais usado em telefones celulares atualmente.

2.5.1 A plataforma JXME

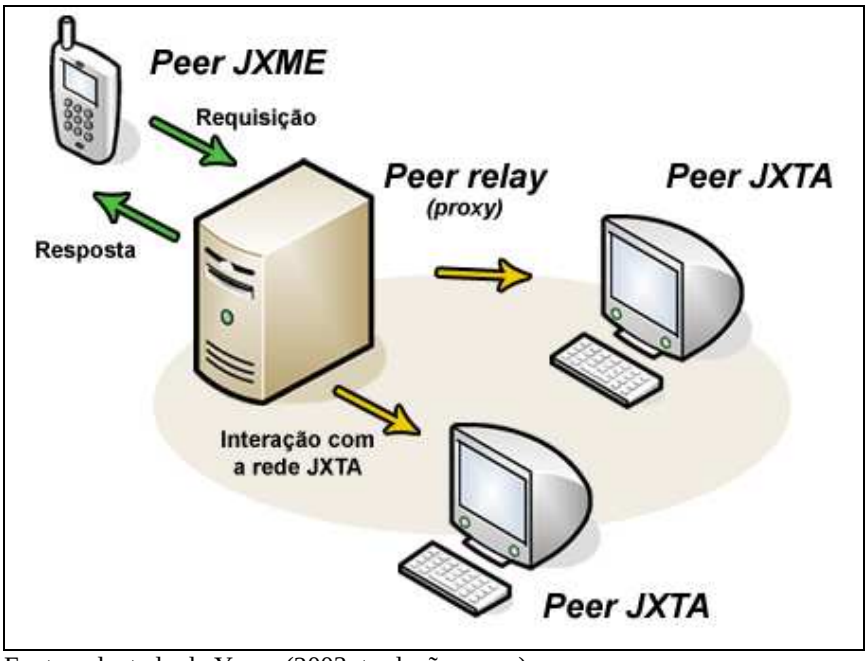
Devido a todas as limitações previamente descritas que os aparelhos celulares possuem, um *peer* JXME não pode executar as mesmas tarefas que os *peers* JXTA (mais sofisticados e

robustos) executam. Dentre estas tarefas estão, por exemplo, oferecer serviços para outros membros de um grupo de *peers* ou até buscar recursos na rede P2P (VILCEK, 2006).

Para suprir estas deficiências, os *peers* JXME utilizam os serviços disponibilizados pelos *peers relay*. Esta comunicação é feita através de conexões HTTP, pelas quais os *peers* JXME e *relay* trocam mensagens objetivando estarem sincronizados com relação às tarefas requisitadas, executadas e seus respectivos resultados.

2.5.2 Cenário de execução

Quando um *peer* JXME faz uma requisição de busca por algum elemento na rede JXTA, como um duto por exemplo, a requisição é enviada a algum *peer relay* presente na rede, conforme Figura 6.



Fonte: adaptado de Yuan (2003, tradução nossa)

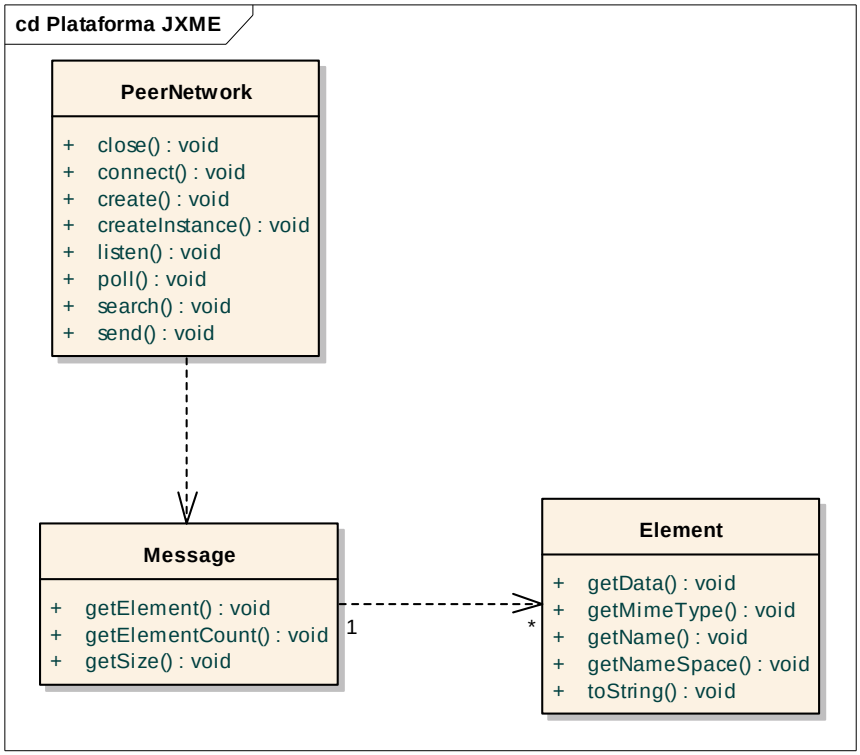
Figura 6 – Cenário de conexão e interação de um *peer* JXME com o resto da rede P2P

O *peer relay* então, propaga a requisição pela rede, coletando as respostas e armazenando-as temporariamente até que o *peer* JXME entre em contato (isto é definido na implementação do aplicativo, pois nela é possível programar para que o *peer* JXME entre em contato a cada dez segundos, por exemplo, ou somente quando o usuário ordenar) para buscar as mensagens a ele endereçadas.

Mas apesar desta dependência, um *peer* JXME não necessita estar sempre atrelado a um único *relay*, e tão pouco um conjunto de *peers* JXME precisa estar conectado ao mesmo *relay* para que se comuniquem. “Portanto, este arranjo não compromete a visão da arquitetura P2P, oposta ao conceito cliente-servidor” (VILCEK, 2006, p. 64).

2.5.3 Interface da plataforma JXME

A interface da plataforma JXME é formada por três classes, conforme a Figura 7, que fornecem a funcionalidade necessária para a integração dos *peers* JXME numa rede P2P (YUAN, 2003, tradução nossa).



Fonte: adaptado de Yuan (2003)

Figura 7 – Diagrama de classes da interface JXME

Abaixo segue uma descrição detalhada de cada classe com seus respectivos métodos (VILCEK, 2006):

a) `Element()`: representa um elemento (arquivo compartilhado) pertencente a uma mensagem JXME. Esta classe fornece os métodos para identificar os diversos campos que compõem um elemento:

- `getData()`: retorna a informação que um elemento transporta,
- `getName()`: retorna o nome do elemento,
- `getMimeType()`: retorna o tipo do arquivo associado à informação que o elemento transporta,
- `getNameSpace()`: retorna o código que identifica o elemento,
- `toString()`: retorna uma representação textual do elemento (nome, tipo e tamanho);

b) `Message()`: representa uma mensagem JXME composta por um ou mais elementos. Esta classe implementa três métodos para manipular um objeto mensagem:
   - `getElement()`: retorna um elemento pertencente à mensagem,
   - `getElementCount()`: retorna a quantidade de elementos pertencentes à mensagem,
   - `getSize()`: retorna o tamanho, em *bytes*, da mensagem;

c) `PeerNetwork()`: é a classe utilizada para especificar um ambiente JXME e executar suas tarefas, como:
   - `createInstance()`: instanciar um *peer* JXME,
   - `connect()`: conectar um *peer* JXME a um *relay* via http,
   - `create()`: criar grupos de *peers* na rede e dutos para comunicação entre eles,
   - `poll()`: buscar mensagens no servidor,
   - `listen()`: abrir um duto para o recebimento de mensagens,
   - `search()`: procurar por *peers*, grupos de *peers* e dutos,
   - `send()`: abrir um duto para o envio de mensagens,
   - `close()`: encerrar a comunicação;

2.6 TRABALHOS CORRELATOS

A tecnologia P2P para dispositivos móveis é bastante recente e devido a isso poucos trabalhos tratam dela. Porém, conforme é descrito a seguir, existem algumas pesquisas que exploram os princípios básicos do assunto.

Arora, Haywod e Pabla (2002) elaboraram um artigo que explica os fundamentos da plataforma JXME, abordando desde funcionalidades técnicas até aspectos mais humanos como suas perspectivas para o futuro da tecnologia, porém não demonstram um software que use os conceitos por eles detalhados.

Já Lima (2005) propõe uma aplicação prática para esta plataforma: o desenvolvimento de um aplicativo de mensagens instantâneas que é executado em telefones celulares de uma maneira muito similar aos *chats* que são conhecidos hoje. Neste trabalho, Lima propõe uma conexão direta entre celulares, ou seja, sem a necessidade de um *peer relay* para tratar as mensagens enviadas e recebidas por cada celular.

Da mesma maneira em seu artigo na revista Web Mobile, Vilcek (2006) discorre sobre a criação de um *chat* para aparelhos celulares, sendo esta e a aplicação de Lima (2005) muito similares na interface e no funcionamento.

Portanto, todos os trabalhos citados têm relação com o aqui proposto, pois utilizam as mesmas tecnologias (JXME e J2ME) e alguns dos conceitos que serão aplicados na construção de um software de compartilhamento de arquivos para celulares. Os mesmos apenas diferem deste por objetivarem a transferência de mensagens de texto e não de arquivos através de uma rede móvel.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são apresentados os aspectos técnicos referentes ao desenvolvimento do trabalho. O mesmo teve por objetivo desenvolver uma aplicação capaz de compartilhar arquivos entre diversos telefones celulares ligados a um servidor central através da rede de transmissão de dados utilizada pelos mesmos.

Para o detalhamento da implementação e das rotinas do software proposto são demonstrados os seus aspectos principais através dos diagramas de classes, casos de uso e atividades.

3.1 CENÁRIO

Documentos, relatórios e apresentações são arquivos que podem ser necessários em qualquer lugar, porém muitas vezes não estão disponíveis por uma dificuldade geográfica ou por inacessibilidade a um computador conectado à Internet.

Com este cenário, o presente estudo visa utilizar a tecnologia JXME para prover aos celulares a capacidade de compartilhar e transmitir arquivos estabelecendo uma alternativa a usuários da Internet que necessitam diariamente acessar documentos e outros arquivos, porém nem sempre tem consigo um computador.

Desta maneira, é apresentado neste capítulo a aplicação SIMCA (Sistema Móvel de Compartilhamento de Arquivos) a qual torna possível a conexão de celulares utilizando tecnologias já acessíveis atualmente a qualquer usuário.

3.2 REQUISITOS

De acordo com Leite (1999), requisitos são objetivos ou restrições estabelecidas por clientes e usuários do sistema, os quais definem as diversas características do sistema. Os requisitos de software são, portanto, condições ou capacidades necessárias que o software

deve possuir para que o usuário possa resolver um problema ou atingir um objetivo. Tradicionalmente, os mesmos são separados em requisitos funcionais e não-funcionais.

3.2.1 Requisitos funcionais

Os requisitos funcionais são a descrição das diversas funções que clientes e usuários querem ou precisam que o software faça. Os mesmos definem a funcionalidade desejada do aplicativo.

O termo função é usado no sentido de operação, podendo esta ser realizada através comandos dos usuários ou através da ocorrência de eventos externos ao sistema (LEITE, 1999).

O Quadro 4 lista os requisitos funcionais atendidos pelo sistema:

Código	**Descrição do Requisito**	**Caso de Uso**
RF01	O sistema deverá permitir ao usuário conectar-se a um servidor que possibilitará sua participação na rede P2P.	UC01
RF02	O sistema deverá permitir ao usuário desconectar-se do servidor que provê acesso à rede P2P.	UC02
RF03	O sistema deverá ser capaz de anunciar sua presença na rede P2P.	UC03
RF04	O sistema deverá ser capaz de reconhecer a presença de outros *peers* conectados à rede P2P.	UC04
RF05	O sistema deverá permitir ao usuário compartilhar arquivos que estejam em seu celular com outros usuários.	UC05
RF06	O sistema deverá permitir ao usuário fazer o *download* de um arquivo que esteja compartilhado no celular de um outro usuário.	UC06
RF07	O sistema deverá permitir ao usuário pesquisar por arquivos disponíveis nos celulares conectados à rede P2P visualizando o diretório compartilhado destes.	UC07
RF08	O sistema deverá permitir ao usuário pesquisar por arquivos disponíveis em celulares conectados à rede P2P informando o	UC08

	nome do documento que deseja pesquisar.	

Quadro 4 – Requisitos funcionais

3.2.2 Requisitos não-funcionais

Requisitos não-funcionais são, de acordo com Leite (1999), “as qualidades globais de um software, como manutenibilidade, usabilidade, desempenho, custos e várias outras”.

O Quadro 5 lista os requisitos não-funcionais atendidos pelo sistema:

Código	**Descrição do Requisito**
RNF01	O sistema deverá utilizar a plataforma JXME para executar as tarefas necessárias a um software P2P, como conectar-se com outros usuários e transferir arquivos.
RNF02	O sistema deverá ser construído na linguagem J2ME.
RNF03	O sistema deverá utilizar a plataforma CLDC do J2ME.
RNF04	O sistema deverá ser portável para todos os tipos de celulares capazes de executar aplicativos Java e que suportem a JSR 75.
RNF05	O sistema deverá possuir uma interface com poucos componentes visuais, para que seja de fácil operação e execução no celular, já que este possui limitações de processamento.

Quadro 5 – Requisitos não-funcionais

3.3 ESPECIFICAÇÃO

A especificação do aplicativo SIMCA se dará através dos seguintes diagramas da *Unified Modeling Language* (UML):

a) diagrama de casos de uso;
b) diagrama de classes;
c) diagrama de atividades.

Todos os diagramas foram construídos utilizando a ferramenta Enterprise Architect.

3.3.1 Diagramas de casos de uso

Diagramas de casos de uso documentam os requisitos funcionais de um sistema, fornecendo assim uma visão do papel que o sistema deve ter, sempre com foco nas necessidades do usuário.

Desta maneira, os casos de uso representam as funcionalidades de um sistema que são ativadas por pessoas ou coisas que invocam as mesmas (atores).

3.3.1.1 Conexão com o servidor

A Figura 8 representa os casos de uso relacionados às ações de conexão/desconexão na rede P2P por parte do usuário e do *peer* que está conectando/desconectando na mesma.

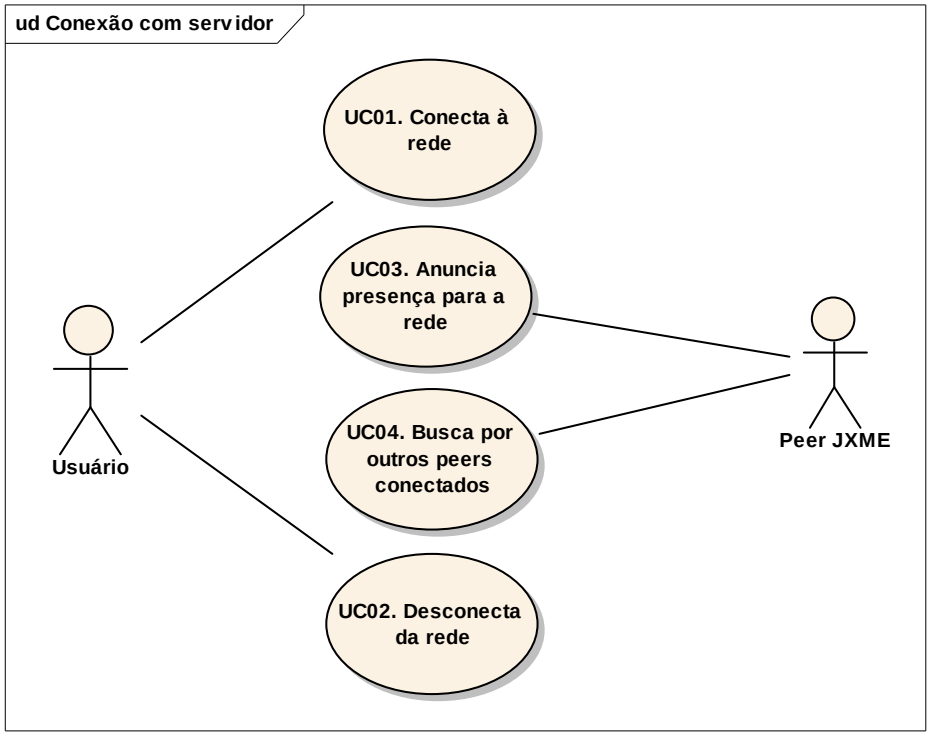


Figura 8 – Diagrama do caso de uso conexão com servidor

O Quadro 6 apresenta um detalhamento do caso de uso 01, apresentado na Figura 8:

UC01: Conecta à rede	
Resumo	Usuário tenta conectar seu aparelho celular à rede P2P informando um nome de usuário, IP de um servidor JXME e o número de uma porta de conexão.
Seqüência de ações	1. Usuário informa à aplicação um nome de usuário, IP de um servidor e a porta para conexão. 2. Usuário seleciona a opção “Conectar”. 3. A aplicação conecta-se ao servidor JXME e junta-se a um grupo de *peers*. 4. A aplicação cria os dutos de comunicação. 5. A aplicação anuncia os arquivos que o *peer* possui em seu diretório compartilhado.
Exceções	No passo 3, o grupo de *peers* não existe. O aplicativo então, cria o grupo. Ainda no passo 3, se a aplicação não conseguir conectar-se no servidor, a tela de conexão é mostrada novamente ao usuário para que o mesmo possa informar o IP de um outro servidor ou tentar conectar-se no mesmo outra vez.

Quadro 6 – Detalhamento do caso de uso 01

O Quadro 7 apresenta um detalhamento do caso de uso 02, apresentado na Figura 8:

UC02: Desconecta da rede	
Resumo	Usuário fecha a aplicação e desconecta-se do servidor através do qual estava compartilhando/copiando arquivos para/de outros *peers*.
Seqüência de ações	1. Usuário seleciona a opção “Sair”. 2. A aplicação manda uma mensagem ao servidor avisando que o *peer* não está mais conectado à rede. 3. A aplicação é finalizada.

Quadro 7 – Detalhamento do caso de uso 02

O Quadro 8 apresenta um detalhamento do caso de uso 03, apresentado na Figura 8:

UC03: Anuncia presença para a rede	
Resumo	*Peer* local anuncia sua presença a outros *peers* que estão conectados no mesmo servidor e fazem parte do mesmo grupo de *peers*.

Seqüência de ações	1. *Peer* cria os dutos de comunicação e com isso avisa o servidor de sua conexão. 2. Servidor avisa os outros *peers* conectados que um novo membro juntou-se ao grupo.

Quadro 8 – Detalhamento do caso de uso 03

O Quadro 9 apresenta um detalhamento do caso de uso 04, apresentado na Figura 8:

UC04: Buscar por outros *peers* conectados	
Resumo	*Peer* local busca por outros *peers* que estão conectados no mesmo servidor e fazem parte do mesmo grupo de *peers*.
Seqüência de ações	1. *Peer* local busca por mensagens de conexão enviadas ao servidor por outros *peers*. 2. *Peer* local adiciona outros *peers* na sua lista de *peers* conectados.

Quadro 9 – Detalhamento do caso de uso 04

3.3.1.2 Transmissão e compartilhamento de arquivos

A Figura 9 representa os casos de uso relacionados às ações de compartilhar e copiar arquivos em uma rede P2P.

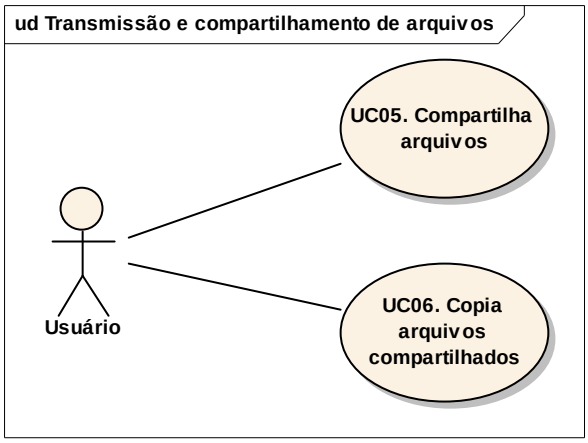


Figura 9 – Diagrama do caso de uso transmissão e compartilhamento de arquivos

O Quadro 10 apresenta um detalhamento do caso de uso 05, apresentado na Figura 9:

UC05: Compartilha arquivos	
Resumo	Usuário compartilha um diretório de seu celular, o qual contém arquivos de diversos tipos e tamanhos, com outros *peers* da rede P2P.
Seqüência de ações	1. Usuário disponibiliza arquivos em um diretório compartilhado da memória de seu celular.

Quadro 10 – Detalhamento do caso de uso 05

O Quadro 11 apresenta um detalhamento do caso de uso 06, apresentado na Figura 9:

UC06: Copia arquivos compartilhados	
Resumo	Usuário faz o *download* de arquivos compartilhados por outros *peers* conectados à rede P2P.
Seqüência de ações	1. Usuário seleciona um dos *peers* conectados à rede P2P. 2. Usuário seleciona um dos arquivos que este *peer* está compartilhando. 3. Arquivo selecionado é copiado para o diretório compartilhado do *peer* local.
Exceções	No passo 1, se o usuário utilizar a pesquisa por nome de arquivo, a aplicação mostrará uma tela com um campo no qual o usuário deve informar o nome do arquivo a ser pesquisado, e logo após, nos resultados da pesquisa, poderá selecioná-lo para fazer o *download*.

Quadro 11 – Detalhamento do caso de uso 06

3.3.1.3 Pesquisa de arquivos

A Figura 10 representa os casos de uso relacionados às ações de pesquisar por arquivos em uma rede P2P.

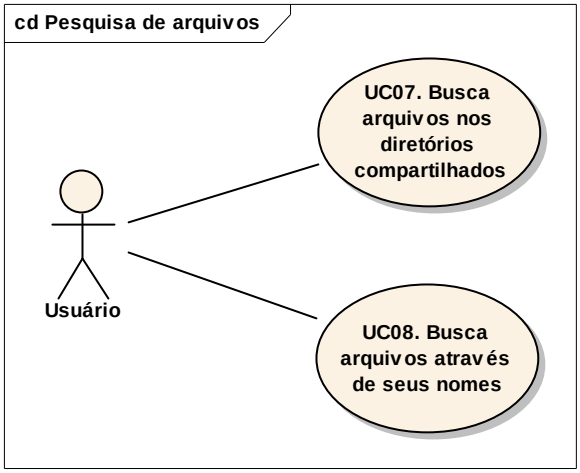


Figura 10 – Diagrama do caso de uso pesquisa de arquivos

O Quadro 12 apresenta um detalhamento do caso de uso 07, apresentado na Figura 10:

UC07: Busca arquivos nos diretórios compartilhados	
Resumo	Usuário acessa o diretório compartilhado de um determinado celular, o qual contém os arquivos que aquele celular compartilha com a rede P2P.
Seqüência de ações	1. Usuário seleciona um dos *peers* conectados à rede P2P.

Quadro 12 – Detalhamento do caso de uso 07

O Quadro 13 apresenta um detalhamento do caso de uso 08, apresentado na Figura 10:

UC08: Busca arquivos através de seus nomes	
Resumo	Usuário informa à aplicação SIMCA o nome do arquivo que deseja copiar e o software procura se algum dos *peers* conectados possui um arquivo com este nome.
Seqüência de ações	1. Usuário digita nome do arquivo que deseja copiar. 2. Aplicação mostra o resultado da busca.

Quadro 13 – Detalhamento do caso de uso 08

3.3.2 Diagrama de classes

De acordo com Pilone e Pitman (2006, p.5), “os diagramas de classes utilizam classes e interfaces para documentar detalhes sobre as entidades que formam seu sistema e as relações estáticas entre elas”.

A Figura 11 apresenta as classes componentes da aplicação SIMCA, proposta neste estudo:

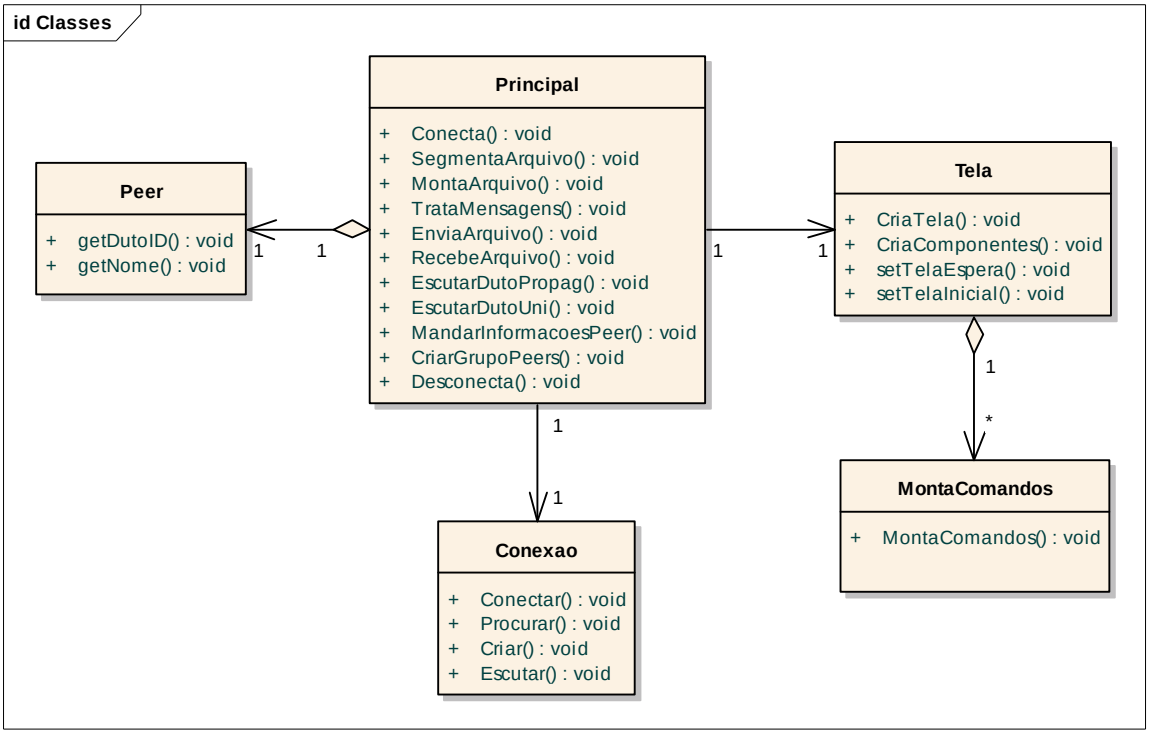


Figura 11 – Diagrama de classes

O Quadro 14 apresenta uma descrição das funcionalidades de cada uma destas classes:

Classe	Descrição
Peer	Nesta classe são guardadas todas as informações referentes a um *peer* na rede P2P, como seu nome e o ID de seus dutos.
Tela	Responsável pelas interações gráficas da aplicação com o usuário. Nela são tratadas as rotinas que exibem botões, listas, textos e imagens na tela do celular.
Conexao	Lida com todas as tarefas relacionadas à comunicação do *peer* com a rede P2P, sendo que todas as mensagens vindas de outros *peers* são encaminhadas para tratamento na classe Principal.
MontaComandos	Na classe MontaComandos são montados os comandos de cada tela,

	que serão posteriormente apresentados no celular pela classe Tela. Além disso, esta classe monitora qualquer comando dado pelo usuário ao celular, sendo os mesmos encaminhados para a classe Principal para que sejam processados.
Principal	A classe Principal, como o próprio nome diz, exerce o papel de classe "motriz" da aplicação SIMCA. Nela são feitas as chamadas para os métodos das classes Tela e Conexao. Além disso, na classe Principal são processados todos os comandos dados pelo usuário ao celular e também todas as mensagens vindas de outros *peers*.

Quadro 14 – Descrição das classes

3.3.3 Diagramas de atividades

Os diagramas de atividades são utilizados para documentar o fluxo de uma atividade ou comportamento para o próximo. Sendo assim, conceitualmente muito similares a um fluxograma (PILONE; PITMAN, 2006).

3.3.3.1 Conectar

Na Figura 12 são demonstrados através de um diagrama de atividades todos os passos necessários por parte de dois *peers* e de um *peer* servidor (*peer relay*) para que uma rede P2P seja estabelecida.

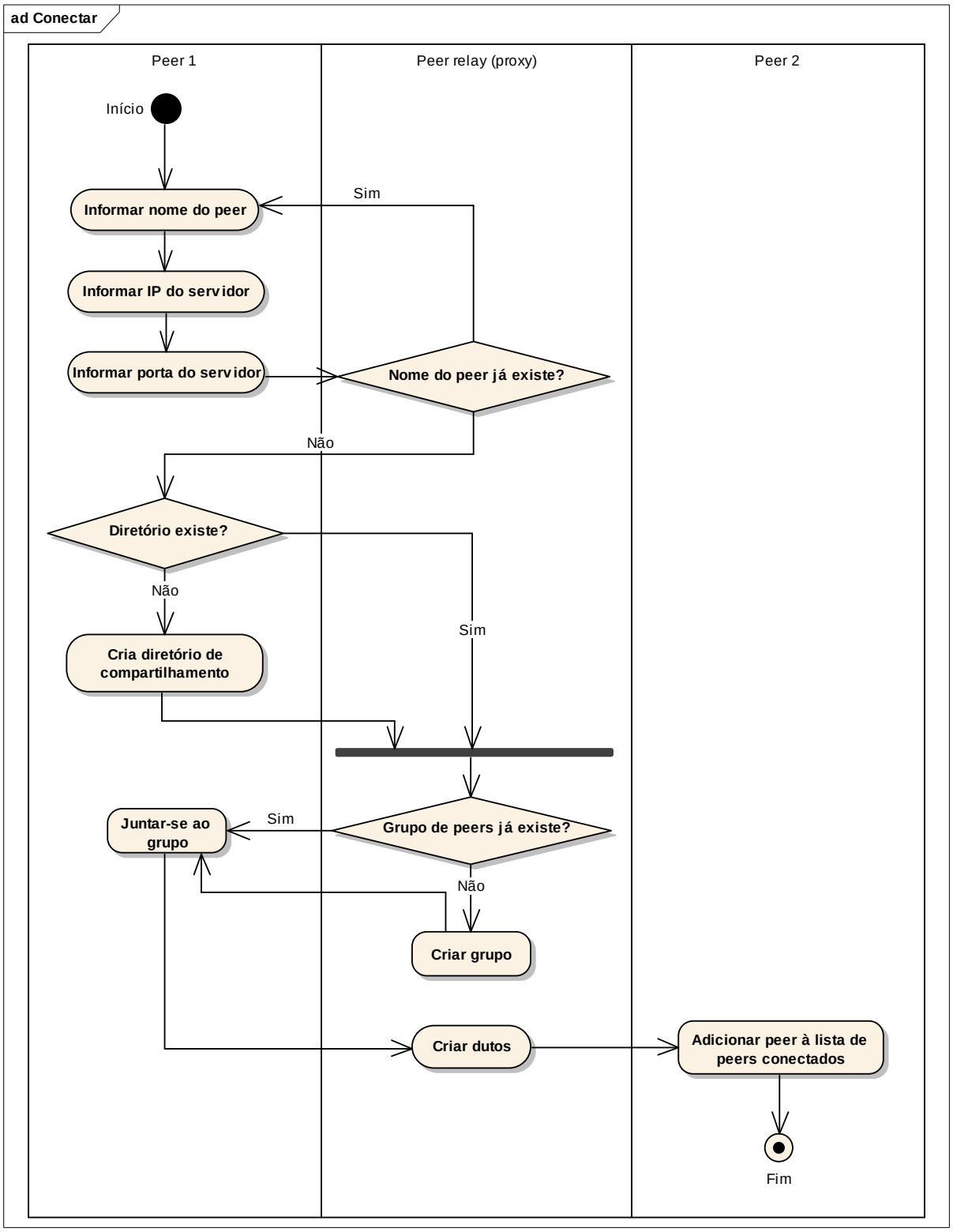


Figura 12 – Diagrama das atividades de conexão

Sendo assim, observa-se que logo após inserir os dados para conexão (nome do *peer*, IP do servidor e porta do servidor), o *peer relay* verifica se já existe um *peer* com este nome conectado na rede. Se for afirmativo, a tela com os dados da conexão será mostrada

novamente ao usuário para que ele mude o nome do *peer* ou então tente se conectar a um outro *peer relay*.

Caso o nome do *peer* ainda não exista, a aplicação verifica se um diretório de compartilhamento de arquivos já existe no celular. Se não existir, ele é criado localmente no aparelho pelo software P2P aqui proposto.

Logo após, é executada a busca pelo o grupo de *peers* criado pela aplicação SIMCA (SIMCAGROUP). Se este não for encontrado, o *peer relay* cria um grupo de *peers* com o nome de SIMCAGROUP e o celular que está executando a aplicação se junta a ele. Caso o grupo já exista apenas é ignorada a etapa de criação do mesmo.

Por fim, são criados os dutos de transmissão de dados do *peer* que está conectando na rede. Com esta ação uma mensagem de conexão é transmitida para todos os *peers* conectados no mesmo *peer relay* e membros do mesmo grupo avisando que a rede P2P que eles participam conta com um novo membro. Estes, por sua vez, adicionam o *peer* recém conectado à sua lista de *peers* disponíveis.

3.3.3.2 Pesquisar e carregar arquivos com a busca em diretórios

Na Figura 13 são demonstrados através de um diagrama de atividades todos os passos necessários por parte de dois *peers* e de um *peer* servidor (*peer relay*) para que um arquivo possa ser carregado de um *peer* a outro, utilizando o método de busca em diretórios compartilhados, conforme descrito no caso de uso 07.

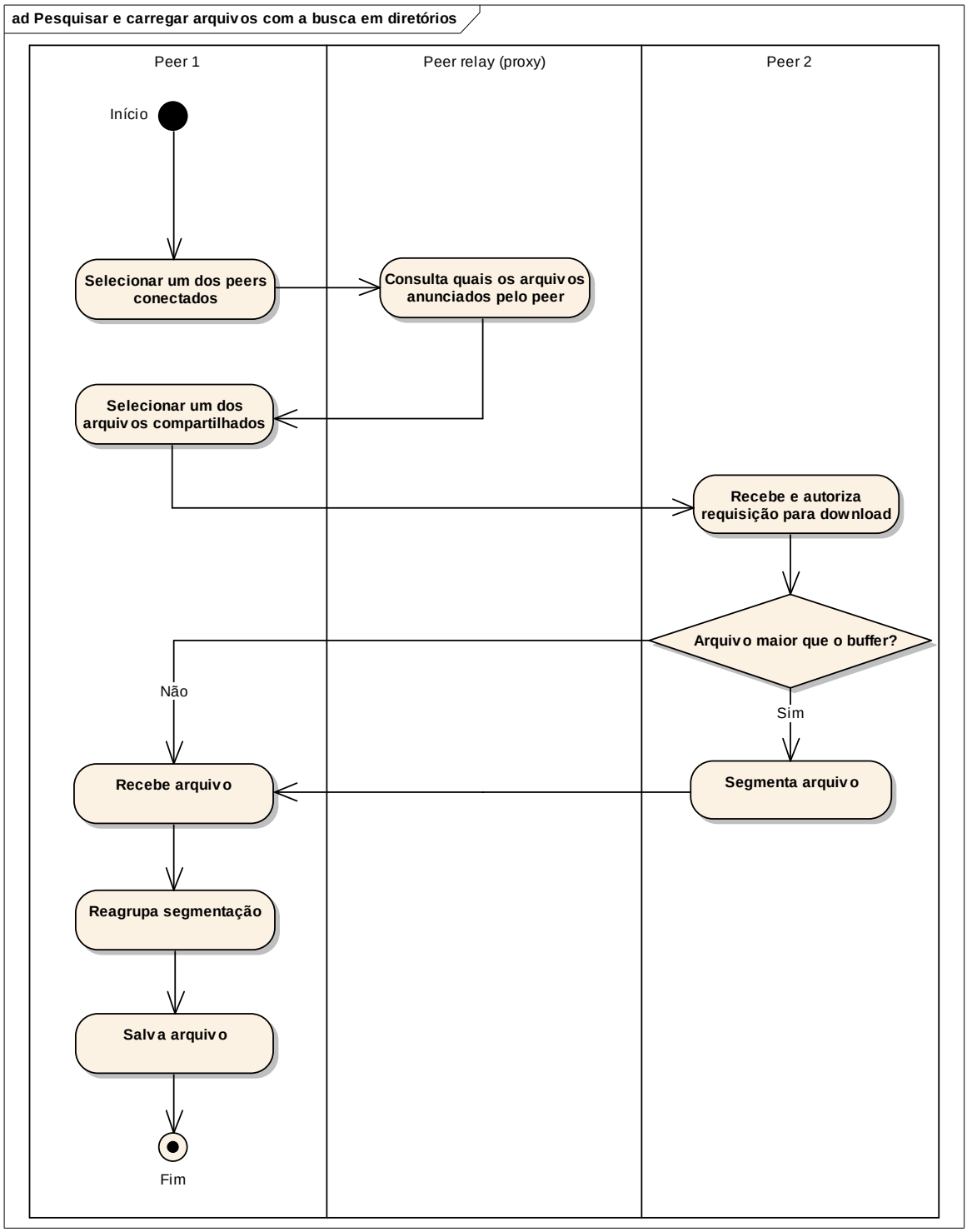


Figura 13 – Diagrama das atividades de pesquisa (diretório) e carregamento de arquivos

Observa-se portanto, que estando o usuário conectado com o seu *peer* à rede P2P, ele pode selecionar um dos outros *peers* conectados e visualizar quais arquivos o mesmo compartilha.

Feito isso, o usuário escolhe qual arquivo deseja carregar para seu celular e o *peer* que contém o mesmo recebe uma requisição avisando que o arquivo escolhido será copiado por outro *peer*. Um importante detalhe é que o usuário do outro *peer* (portador do arquivo a ser

copiado) não toma conhecimento desta requisição, ela é tratada somente pela aplicação, que a recebe e autoriza o *download*.

Neste momento, o *peer* portador do arquivo analisa o tamanho do mesmo e verifica se ele é maior que o tamanho do *buffer* definido para a aplicação (o tamanho do *buffer* da aplicação é detalhado e justificado no capítulo 4). Se for, ele segmenta o arquivo e envia os pedaços ao *peer* que requisitou a cópia dele. Caso contrário, a atividade de segmentação do arquivo é ignorada e o mesmo é enviado em apenas um pedaço.

Finalizando a operação, o *peer* requisitante recebe o arquivo, salvando-o temporariamente na memória do celular. Caso o mesmo tenha vindo em forma de segmentos, estes são reagrupados e salvos em memória (desta vez de maneira definitiva) como um arquivo único. Se o arquivo vier em somente um segmento, a rotina responsável pelo reagrupe destes será executada da mesma maneira, porém ao reconhecer a quantidade de segmentos recebidos, irá apenas encaminhar o arquivo para o salvamento em memória.

3.3.3.3 Pesquisar e carregar arquivos com a busca por nome

Na Figura 14 são demonstrados através de um diagrama de atividades todos os passos necessários por parte de dois *peers* e de um *peer* servidor (*peer relay*) para que um arquivo possa ser carregado de um *peer* a outro, utilizando o método de busca através de nomes de arquivos, conforme descrito no caso de uso 08.

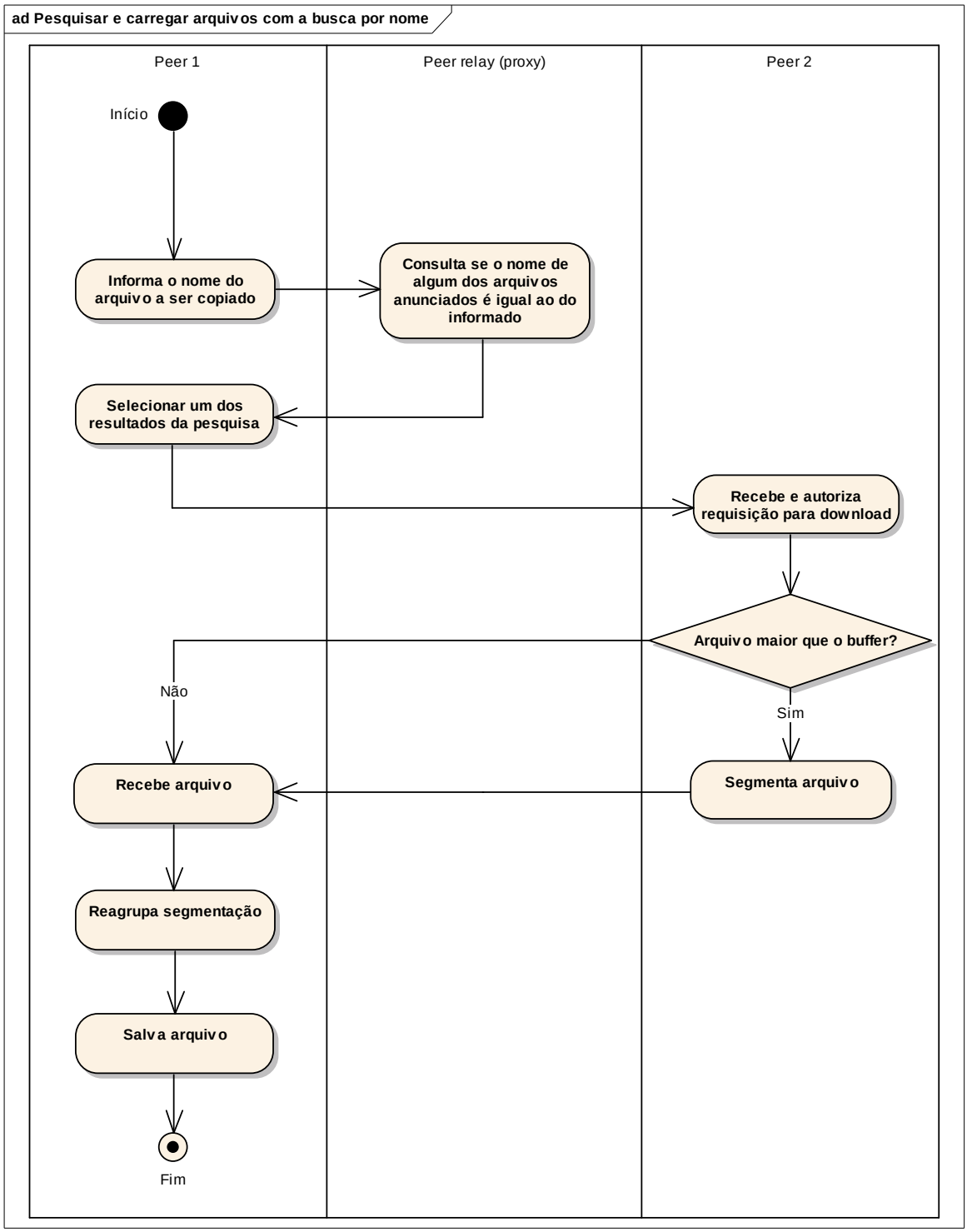


Figura 14 – Diagrama das atividades de pesquisa (nome) e carregamento de arquivos

Observa-se portanto, que estando o usuário conectado com o seu *peer* à rede P2P, ele pode informar à aplicação SIMCA o nome do arquivo que deseja copiar, visualizando logo após o resultado da busca efetuada.

A partir deste ponto, o comportamento da aplicação é idêntico ao do mostrado na Figura 13, ou seja, o usuário escolhe qual arquivo deseja carregar para seu celular e o *peer* que contém o mesmo recebe uma requisição avisando que o arquivo escolhido será copiado

por outro *peer*. Um importante detalhe é que o usuário do outro *peer* (portador do arquivo a ser copiado) não toma conhecimento desta requisição, ela é tratada somente pela aplicação, que a recebe e autoriza o *download*.

Neste momento, o *peer* portador do arquivo analisa o tamanho do mesmo e verifica se ele é maior que o tamanho do *buffer* definido para a aplicação (o tamanho do *buffer* da aplicação é detalhado e justificado no capítulo 4). Se for, ele segmenta o arquivo e envia os pedaços ao *peer* que requisitou a cópia dele. Caso contrário, a atividade de segmentação do arquivo é ignorada e o mesmo é enviado em apenas um pedaço.

Finalizando a operação, o *peer* requisitante recebe o arquivo, salvando-o temporariamente na memória do celular. Caso o mesmo tenha vindo em forma de segmentos, estes são reagrupados e salvos em memória (desta vez de maneira definitiva) como um arquivo único. Se o arquivo vier em somente um segmento, a rotina responsável pelo reagrupe destes será executada da mesma maneira, porém ao reconhecer a quantidade de segmentos recebidos, irá apenas encaminhar o arquivo para o salvamento em memória.

3.3.3.4 Desconectar

Na Figura 15 são demonstrados através de um diagrama de atividades todos os passos necessários por parte de dois *peers* e de um *peer* servidor (*peer relay*) para que um *peer* possa desconectar-se da rede P2P e os outros fiquem sabendo desta operação.

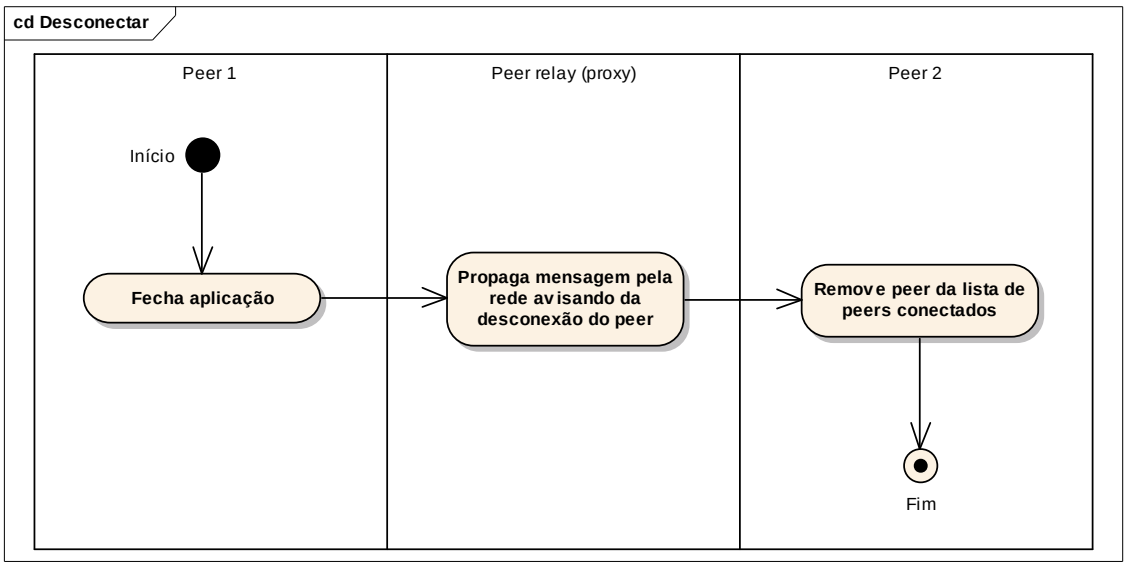


Figura 15 – Diagrama das atividades de desconexão

Para que um *peer*, executando a aplicação SIMCA, possa desconectar-se da rede P2P é necessário que o usuário acione o comando “Desconectar”, presente na tela da mesma.

Com isso a aplicação será fechada e uma mensagem de desconexão será enviada ao *peer relay*. Este por sua vez, propagará a mensagem a todos os *peers* da rede, que eliminarão o *peer* desconectado de sua lista de *peers* disponíveis.

3.4 IMPLEMENTAÇÃO

Neste capítulo são apresentadas algumas características e limitações técnicas do sistema P2P proposto, sendo elas comparadas às possibilidades dos telefones celulares atuais, da tecnologia JXME e das redes sem fio, objetivando uma transposição dos conceitos teóricos para o funcionamento prático.

Além disso, também são descritas as ferramentas utilizadas para a implementação do aplicativo SIMCA, tais como:

a) ambiente de desenvolvimento Java NetBeans;
b) simulador Sun Java Wireless Toolkit.

3.4.1 Características e limitações

Como discutido na revisão bibliográfica, um telefone celular necessita do auxílio de um *peer relay* (*proxy*) para operar normalmente em uma rede JXTA, pois este deve efetuar as operações que exigem maior poder computacional do que o provido pelos celulares. Sendo assim, uma rede P2P entre estes dispositivos não pode ser composta apenas pelos mesmos. A Figura 16 demonstra como é montada a rede física da aplicação desenvolvida e como ela é vista na rede virtual que a plataforma JXME cria.

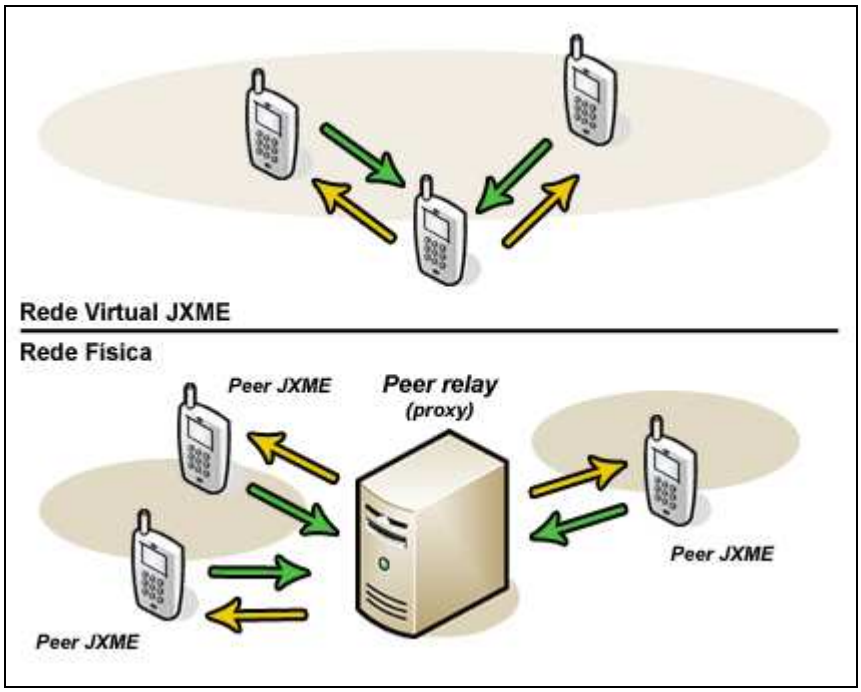


Figura 16 – Arquitetura de uma rede P2P móvel

Desta maneira, o aplicativo SIMCA necessita de um servidor *proxy* para realizar operações mais complexas como a descoberta de recursos (arquivos) disponíveis nos *peers* conectados na rede.

3.4.1.1 *Proxy* JXME

O *proxy* é um serviço localizado em um *peer relay* e destina-se a prover mecanismos de comunicação com *peers* separados por um *firewall*, definindo rotas para mensagens destinadas a outros *peers* da rede.

Porém o *peer relay* não consegue efetuar todas estas operações sozinho, ele necessita de um software que o guie e utilize sua capacidade de processamento para resolver as requisições dos *peers* JXME. Para o funcionamento do aplicativo SIMCA é utilizado o software JXTA Shell, que executa estas operações. A Figura 17 demonstra a interface deste programa.

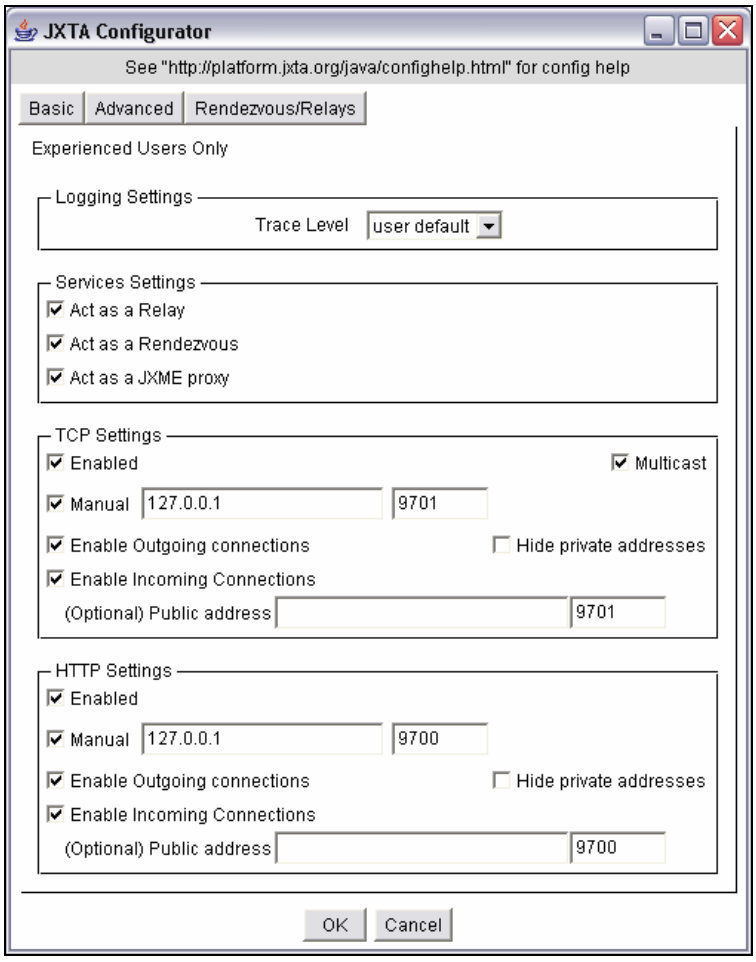


Figura 17 – Configurações do JXTA Shell

Analisando a figura, percebe-se que as três opções do grupo *Service Settings* estão marcadas. Isto significa que o *peer relay* que está executando o software atua ao mesmo tempo como um serviço de *proxy* JXME, *peer* de encontro e *peer relay*.

O endereço de IP definido na área de HTTP *Settings* indica que a conexão dos *peers* se dará na máquina local, através da porta 9700.

3.4.1.2 Operações dos *peers* JXME

Durante a revisão bibliográfica algumas funções essenciais aos *peers* de um sistema P2P foram estabelecidas:

a) conexão;
b) descoberta de *peers*;
c) anúncio de recursos (arquivos);

d) compartilhamento de arquivos.

Nesta seção, as mesmas serão descritas com um foco na sua utilização pelo aplicativo SIMCA.

3.4.1.2.1 Conexão

Para que uma conexão possa ser iniciada, é necessário que a aplicação crie uma nova instância da classe `PeerNetwork` (abordada no capítulo referente à plataforma JXME) e através desta nova instância envie uma requisição de conexão ao servidor *proxy* (*peer relay*).

O Quadro 15 mostra o código fonte responsável pelas chamadas das ações de conexão do *peer* à rede P2P.

```
/* 1: Conectar */
  if (Conexao.conectar(SIMCA + "_" + NomePeer)) {
    Tela.waitingbar.setLabel("Conectado.");
    Tela.waitingbar.setLabel("Juntando-se ao grupo de peers...");
    Tela.setTelaEspera();

    /* 2: Criar ou juntar-se a um grupo de peers */
    if (CriarGrupoPeers()) {
      Tela.waitingbar.setLabel("Associação completada. Abrindo dutos");
      Tela.setTelaEspera();

      /* 3: Criar (caso ainda não existam) e
            "escutar" dutos de propagação */
      if (EscutarDutoPropag()) {

        /* 4: Criar (caso ainda não existam) e
            "escutar" dutos unidirecionais */
        if (EscutarDutoUni()) {
          Tela.waitingbar.setLabel("Dutos abertos. Requisitando dados
                                    do servidor");
          Tela.setTelaEspera();
          Tela.setTelaInicial();

          /* 5: Manda as informações do peer que está conectando para o
                peer relay e procura por outros peers conectados */
          if (MandarInformacoesPeer()) {
            Tela.waitingbar.setLabel("Informações do peer enviadas");
            Conexao.procurar();
          }
        }
      }
    }
  }
```

Quadro 15 – Trecho do código fonte de conexão de *peers*

No código é possível observar a chamada do método **`Conexao`**`.conectar`, o qual cria uma instância da classe `PeerNetwork`, verifica se o nome do *peer* já existe na rede e cria o diretório de compartilhamento caso o mesmo ainda não exista.

Logo após, para separar um grupo de usuários com interesses em comum do resto da rede, um grupo de *peers* é criado através do método `CriarGrupoPeers()` (no caso da aplicação proposta este grupo de peers será denominado e tratado na rede P2P como SIMCAGROUP), sendo que o primeiro *peer* da rede cria o grupo, caso ele ainda não exista e junta-se a ele.

Os dutos pelos quais trafegarão os dados transmitidos também são criados (caso ainda não existam) pelos métodos `EscutarDutoPropag() e EscutarDutoUni()`. O primeiro a ser estabelecido é o duto de propagação, sendo criado pelo primeiro *peer* que se conecta na rede. Os outros *peers* buscam este duto e ficam “escutando-o”.

Além dele, também é criado o canal de comunicação individual, que nada mais é do que um duto unidirecional criado por todos os *peers* ao se conectarem na rede, servindo como duto único para cada *peer*.

Quando a conexão já estiver estabelecida, o *peer* que está conectando manda suas informações ao *proxy* através do método `MandarInformacoesPeer()` e busca se outros *peers* estão conectados a ele através do método **`Conexao`**`.procurar()`.

A representação gráfica das mensagens trocadas pelo *peer* JXME e pelo *proxy* durante as ações de conexão do Quadro 15 são vistas na Figura 18.

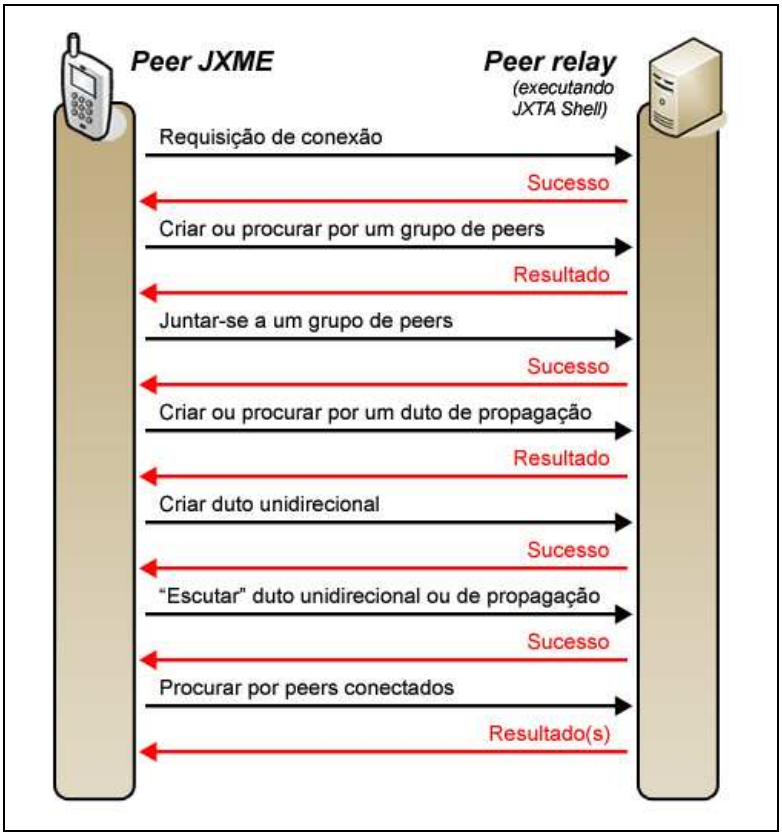


Figura 18 – *Peer* JXME conectando à rede

3.4.1.2.2 Descoberta de *peers* na rede

A plataforma JXME permite uma busca de *peers* através de anúncios (identificados por um JXTA ID) propagados pelos mesmos na rede.

Um peer pode procurar por estes anúncios a cada dez segundos, por exemplo, ou então a cada minuto. Porém, para minimizar o tráfego na rede e torná-la mais estável, o aplicativo SIMCA fará estas buscas apenas quando um novo *peer* conectar-se na rede e criar um novo duto. A forma de buscar por novos *peers* conectados é representada na Figura 19.

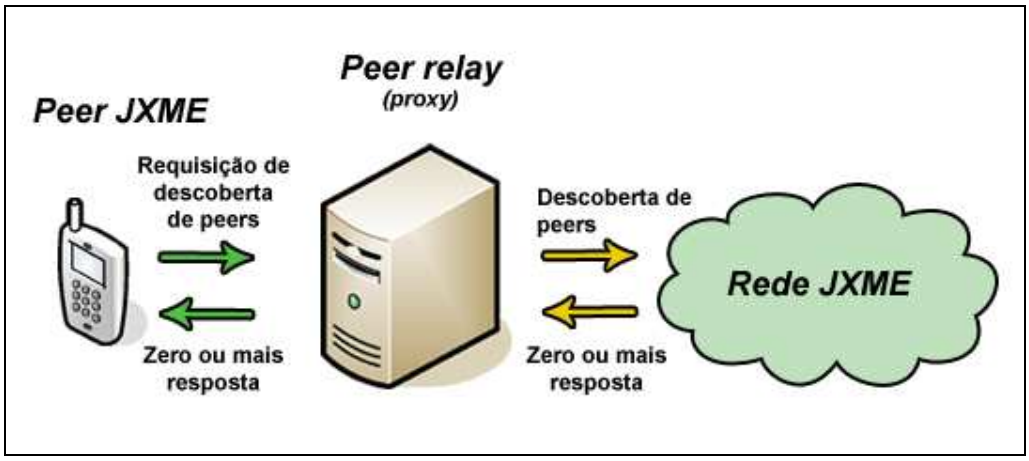


Figura 19 – Descoberta de novos *peers*

A mensagem de requisição de descoberta de *peers* é uma mensagem vazia com um cabeçalho de descoberta de *peers*. A mensagem de resposta pode conter várias informações como o nome do *peer* descoberto e o identificador de seu duto, porém deve ser bastante pequena para evitar um tráfego desnecessário na rede.

No Quadro 16 pode ser visto o trecho do código fonte da aplicação que faz a descoberta de *peers* na rede.

```
try{

  /* Consulta quais os peers conectados */
  ConsultaId = PeerNetwork.search("PEER", 9999);
  }

  /* Captura possíveis exceções ocorridas durante o processo */
  catch(IOException ioex){
  ioex.printStackTrace();
  }
```

Quadro 16 – Trecho de código fonte de descoberta de *peers*

Esta descoberta é feita através do método `PeerNetwork.search` que tem como parâmetros:

a) tipo de recurso que se deseja procurar;
b) número máximo de resultados.

3.4.1.2.3 Anunciar arquivos

Apesar de a plataforma JXME permitir a criação de anúncios de *peers*, grupos de *peers* e dutos, a mesma não permite criar anúncios de arquivos compartilhados. Sendo assim, para contornar esta limitação, o aplicativo SIMCA utiliza a JSR 75 para pesquisar no diretório

compartilhado de cada *peer* e a partir dos resultados, criar um arquivo de anúncio do conteúdo que este *peer* compartilha. Sempre que um anúncio é criado, ele é enviado ao *proxy* JXME, e armazenado na memória do mesmo para eventuais buscas que outros *peers* conectados a ele possam fazer.

No Quadro 17 é exibido o trecho do código fonte da aplicação que faz a criação de anúncios de arquivos na rede.

```
try {

/* Conecta no diretório compartilhado (usando a JSR 75) */
Diretorio = (FileConnection) Connector.open("file:///" + NomePeer +
                                            "_Compartilhados/");

/* Guarda o nome dos arquivos encontrados */
NomArqs = Diretorio.list();

/* Cria um array de elementos */
Element[] elementos = new Element[3];

  /* Nome dos arquivos compartilhados */
  elementos[0] = new Element("NomArqs", itemname.getBytes(), null, null);

  /* ID do peer que compartilha dos arquivos */
  elementos[1] = new Element("PeerID", itemdata, null, null);

  /* Informações sobre tamanho, data de criação e permissões dos
     arquivos*/
  elementos[2] = new Element("InfArqs", itemdata, null, null);

  /* Cria mensagem com os elementos */
  Message message = new Message(elementos);
  return message;
}
```

Quadro 17 – Trecho do código fonte de criação de anúncios de arquivos

A criação de anúncios é feita através da conexão da JSR 75 no diretório compartilhado do *peer*, utilizando o método `Connector.open`, logo após isso, os resultados da busca dentro do diretório (`Diretorio.list()`) são guardados na variável `NomArqs`.

O aplicativo SIMCA então, cria um *array* de três elementos (`Element[] elementos = new Element[3]`) que serão transmitidos em uma mensagem ao *peer relay*. Neste *array* constam informações sobre o nome dos arquivos disponíveis, suas propriedades (tamanho, data de criação e permissões) e o ID do *peer* que os compartilha.

3.4.1.2.4 Compartilhamento de arquivos

Compartilhar arquivos, principalmente os com tamanhos grandes, tornou-se uma das atividades mais populares nas redes P2P convencionais. Uma razão para isso poderia ser a disseminação da Internet de banda larga por vários países do mundo.

Porém a rede de transmissão de dados entre celulares e os próprios aparelhos celulares não foram projetados para isso e portanto apresentam algumas limitações como:

a) conectividade entre dispositivos altamente instável se comparada às redes de transmissão convencionais;
b) capacidade da memória dos aparelhos celulares bastante limitada;
c) baixa taxa de transmissão.

O problema relacionado à taxa de transmissão está ligado intrinsecamente ao tipo de rede disponível. Por exemplo, com a implementação das redes *Third Generetion* (3G) a transmissão de dados poderá ser de até 384 Kbps (WI-FI PLANET, 2006, tradução nossa). Porém utilizando redes *General Packet Radio Service* (GPRS) a taxa de transmissão pode chegar somente a 48 Kbps (MOBILE PHONES UK, 2006, tradução nossa).

Já a limitação de memória dos dispositivos celulares é um problema cada vez menos importante, pois modelos mais recentes já habilitam o uso de cartões de memória que lhes permite armazenar até 1 Gb de arquivos (NOKIA 3250, 2006, tradução nossa).

Por fim, o problema da instabilidade da conexão entre os aparelhos. Esta pode ser uma questão gravíssima do ponto de vista de redes P2P, sendo que existem poucas alternativas para que ela seja minimizada. A única ação que pode ser tomada é garantir através do software desenvolvido a integridade dos dados transmitidos na rede, mesmo que um dos *peers* envolvidos em uma transação desconecte-se por motivos que fogem ao seu controle.

Em suma, as dificuldades encontradas na transmissão e compartilhamento de arquivos de qualquer tamanho e tipo entre telefones celulares são bastante significativas e necessitam de atenção especial.

Porém, ao analisar de maneira cuidadosa os mesmos problemas focando o compartilhamento de arquivos de grande porte (leia-se mais de 70 KB), novas barreiras são impostas:

a) A plataforma JXTA possui um *buffer* limite para transmissão de arquivos pela rede, que é de 64 KB (JXTA ZONE, 2006, tradução nossa);

b) A plataforma JXME, por conseqüência, possui o mesmo limite de *buffer*.

JXTA, por sua vez, segmenta seus arquivos transmitidos para que não ultrapassem o limite do *buffer* estabelecido. A plataforma JXME, ao contrário, não possui recursos para segmentação de suas mensagens, o que torna um aplicativo P2P utilizando-se dos recursos de JXME impraticável.

Para resolver este problema e tornar a aplicação SIMCA viável, é implementada no software proposto uma funcionalidade que segmenta qualquer arquivo enviado em pedaços de 16 KB (o motivo para a escolha deste tamanho é descrito no capítulo 4), antes que os mesmos sejam transmitidos pela rede. Esta rotina é disponibilizada no apêndice A.

Posteriormente estes segmentos são reagrupados e salvos quando forem recebidos por outro *peer* na rede.

3.4.2 Ferramentas utilizadas

Para o desenvolvimento do aplicativo SIMCA, foi utilizada a ferramenta NetBeans. Esta, é uma aplicação de desenvolvimento grátis, feita 100% em Java, tendo se tornado nos últimos anos uma das mais tradicionais plataformas para o desenvolvimento de softwares nesta linguagem (OLIVEIRA, 2005).

Mas para a construção de softwares em J2ME, o NetBeans necessita do auxílio de uma ferramenta extra, que é o NetBeans Mobility Pack. O mesmo quando instalado integra-se ao NetBeans e fornece ferramentas específicas para o desenvolvimento de aplicações para dispositivos móveis.

Para testes foi utilizado o simulador Sun Java Wireless Toolkit, que de acordo com Matsumoto (2005), é um conjunto de ferramentas para execução e simulação de aplicativos Java que rodam em dispositivos móveis.

3.4.3 Operacionalidade da implementação

Nesta seção é apresentada a seqüência de telas e operações que um usuário faz para que consiga conectar seu *peer* na rede P2P e copiar um arquivo de um outro *peer* conectado.

3.4.3.1 Conectar

A tela de configurações de conexão, mostrada como a primeira tela na Figura 20, é onde o usuário define qual o nome do *peer*, o endereço de IP do *peer relay* (servidor) e a porta utilizada para conexão no mesmo.

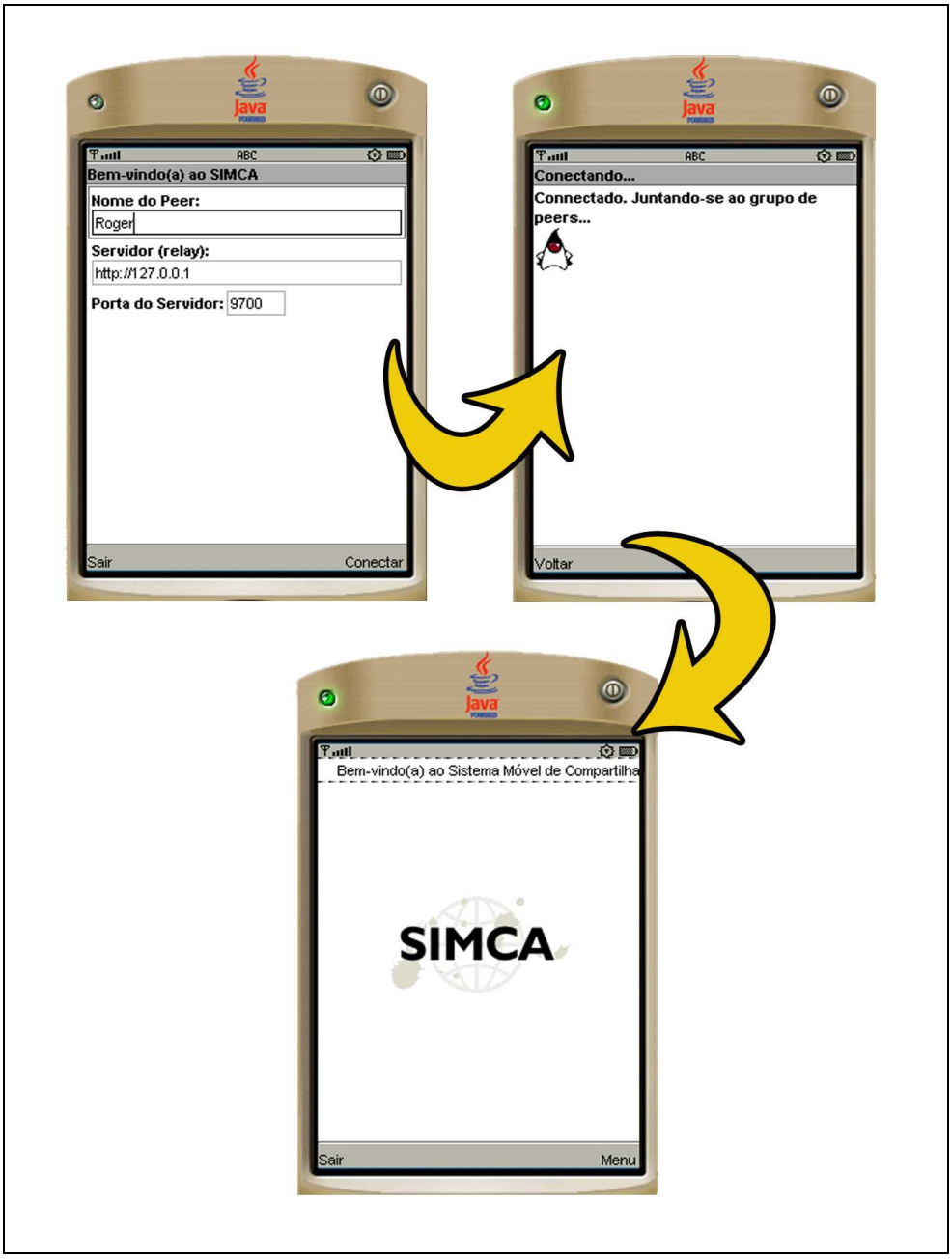


Figura 20 – Seqüência de ações para a conexão de um *peer*

Efetuadas as configurações, o usuário aciona o comando “Conectar” e a aplicação executa uma série de ações, conforme a segunda tela da Figura 20, para que a conexão com o *proxy* (*peer relay*) seja estabelecida e a tela inicial do SIMCA seja mostrada.

3.4.3.2 Copiar arquivo

O *peer* conectado então, habilita as opções de busca de arquivos, como pode ser visto na primeira tela da Figura 21. Na mesma pode-se também visualizar o botão sair, que se acionado, fecha a aplicação e manda uma mensagem ao *proxy* avisando que o *peer* desconectou-se da rede.

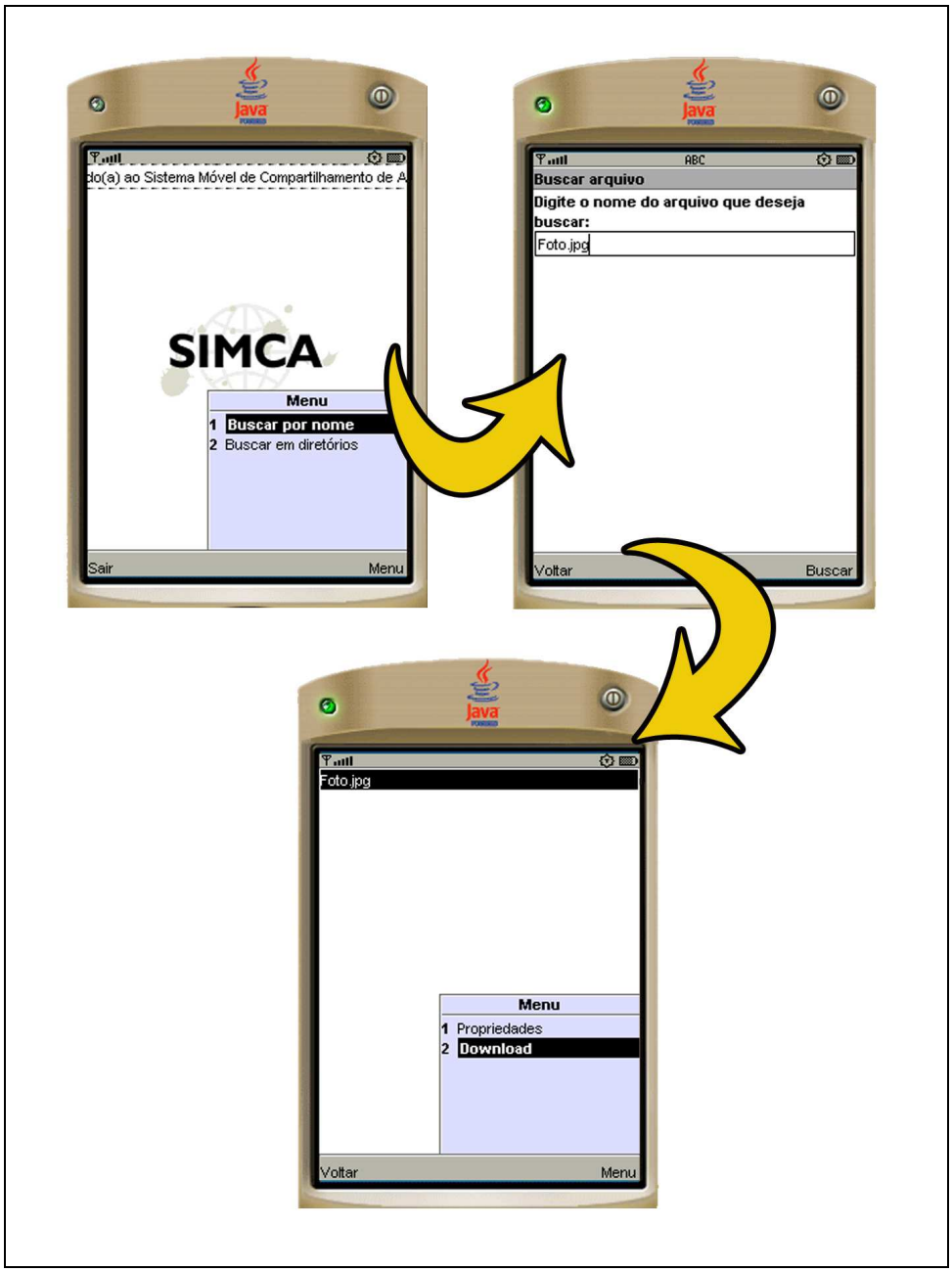


Figura 21 – Busca por nome

Selecionando a opção “Buscar por nome”, a aplicação mostra a tela de busca, conforme segunda tela da Figura 21.

Logo após, se acionando o comando “Buscar”, são mostrados os resultados da pesquisa e habilitados os comandos “Propriedades” e “Download”, que são detalhados em uma ação posterior.

Mas se o usuário selecionar a opção “Buscar em diretório”, uma lista com os *peers* conectados ao servidor no momento é mostrada ao usuário, conforme visto na primeira tela da Figura 22.

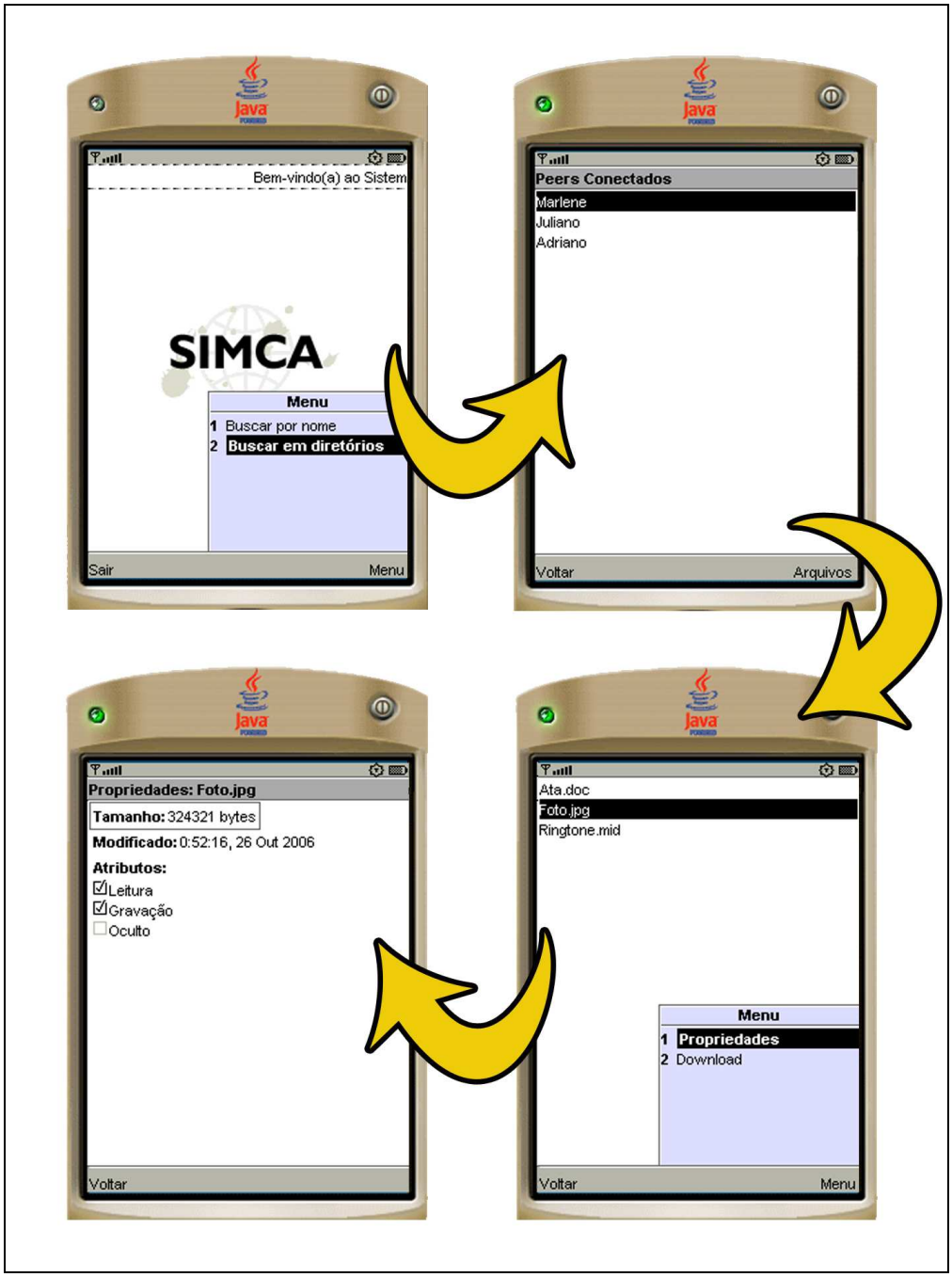


Figura 22 – Busca em diretórios

Após isso, é possível selecionar um *peer* e ativar o comando “Arquivos”, o qual mostrará quais arquivos o *peer* selecionado compartilha.

Novamente ao escolher um arquivo habilitam-se as opções “Propriedades” e “Download”, onde caso o usuário clique na opção “Propriedades”, serão mostradas informações sobre o arquivo a ser baixado.

Se a opção “Download” for selecionada, o *peer* fará o *download* do arquivo escolhido e salvará o mesmo na memória do celular, mostrando uma mensagem após a conclusão do processo, conforme ilustrado na Figura 23.

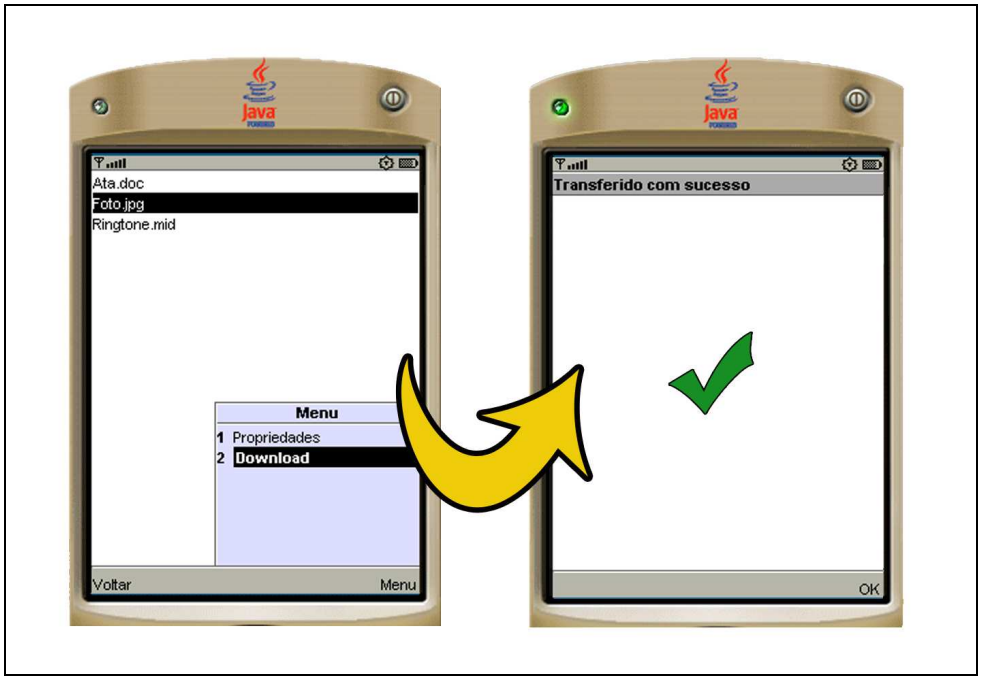


Figura 23 – *Download* de arquivos

4 TESTES DE *PERFORMANCE*

Neste capítulo é testada a *performance* do software desenvolvido fazendo transferências de arquivos entre dois *peers* executando o aplicativo SIMCA. Ambos estão conectados ao aplicativo JXTA Shell através do simulador Sun Java Wireless Toolkit, sendo executados em um computador local que imita uma rede GPRS.

4.1 VARIÁVEIS DE TESTE

Durante os testes, são analisadas três variáveis:

a) tamanho do *buffer* (em KB);
b) tamanho do arquivo transmitido (em KB);
c) tempo (em segundos) para a transmissão total do arquivo.

Destas, o tamanho do *buffer* é de extrema importância para a aplicação proposta, pois nela é especificada a quantidade de informações que será transmitida em cada segmento de mensagem. Como visto no capítulo referente às características do sistema, o tamanho máximo do *buffer* suportado pela plataforma JXME é de 64 KB, por isso o SIMCA divide o arquivo transmitido em segmentos de 60 KB. Porém este tamanho de segmento pode não ser o mais otimizado, sendo interessante mantê-lo o menor possível para que não provoque um processamento excessivo no *peer relay*, o qual deve estar disponível para atender várias requisições simultaneamente.

Portando, um dos objetivos dos testes foi identificar o menor tamanho de *buffer* junto do melhor tempo de transmissão de arquivos. Para isso foram testados arquivos com três tamanhos diferentes:

a) 80 KB;
b) 150 KB;
c) 250 KB.

Em cada teste o tamanho do *buffer* é aumentado gradualmente (de quatro em quatro KB) e o teste efetuado três vezes para garantir sua credibilidade.

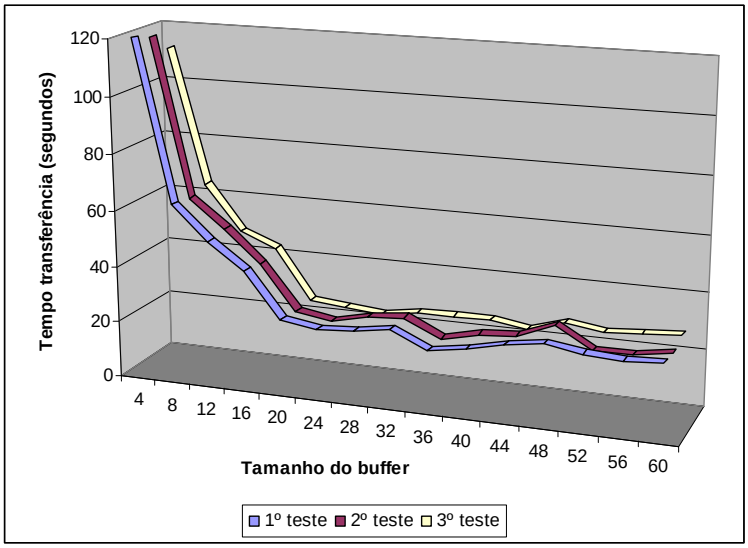
4.1.1 Resultados dos testes

Os resultados dos testes, junto de um gráfico representativo dos mesmos, são mostrados nas Tabelas 1, 2 e 3, onde o tamanho do *buffer* é exibido em *kilobytes* e o tempo de transferência em segundos.

Tabela 1 – Representação do teste 01

TESTE 01 – Tamanho do arquivo transmitido: 80 KB

Tamanho do *buffer* (em KB)	**Transferência em segundos**		
	1º teste	**2º teste**	**3º teste**
4	120	118	112
8	63	62	64
12	51	52	48
16	42	41	43
20	26	25	25
24	24	23	24
28	25	26	23
32	27	27	25
36	21	21	25
40	23	24	25
44	26	25	23
48	28	30	27
52	26	23	25
56	25	23	26
60	26	25	27

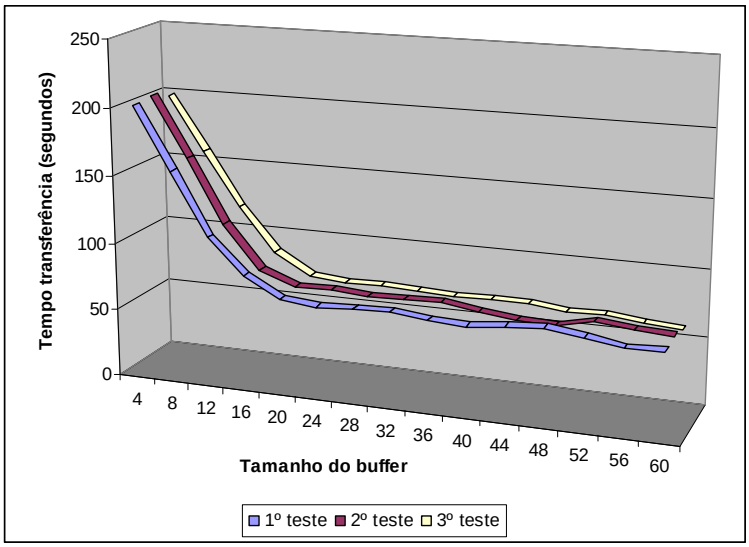


Analisando os resultados do teste 01, fica evidente que a transferência de vários segmentos menores é muito pior do que transmitir menos segmentos de tamanho maior. Percebe-se também que, transmitindo um arquivo com 80 KB de tamanho, o gráfico apresenta uma queda significativa entre um *buffer* de 4 e 16 KB.

Tabela 2 – Representação do teste 02

TESTE 02 – Tamanho do arquivo transmitido: 150 KB

Tamanho do *buffer* (em KB)	**Transferência em segundos**		
	1º teste	**2º teste**	**3º teste**
4	201	203	198
8	155	159	158
12	109	112	118
16	83	80	86
20	69	70	71
24	66	71	68
28	67	69	69
32	68	69	68
36	65	70	66
40	64	66	67
44	67	63	67
48	69	62	64
52	65	68	65
56	61	65	62
60	62	64	60

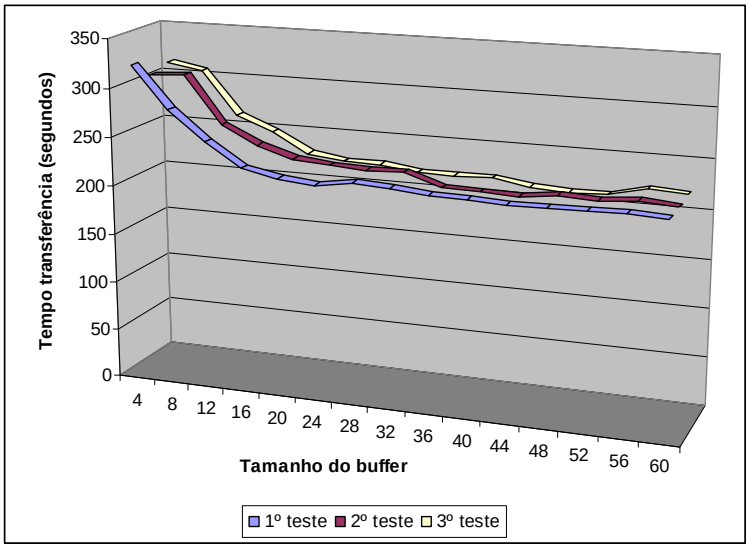


No teste 02, observa-se uma semelhança bastante grande com o teste 01 no que diz respeito à variação do tempo de transmissão, novamente pode ser constatado que o gráfico apresenta uma queda significativa entre um *buffer* de 4 e 16 KB. Já com um *buffer* com mais de 20 KB a variação do tempo para a transmissão do arquivo é menor do que a observada no teste 01, mostrando uma tendência da rede de estabilizar-se mais com um arquivo de maior porte.

Tabela 3 – Representação do teste 03

TESTE 03 – Tamanho do arquivo transmitido: 250 KB

Tamanho do buffer (em KB)	**Transferência em segundos**		
	1º teste	**2º teste**	**3º teste**
4	323	305	312
8	280	308	305
12	251	260	262
16	227	242	248
20	220	231	228
24	216	228	223
28	222	226	223
32	220	227	218
36	216	215	218
40	216	214	219
44	214	213	213
48	215	217	211
52	215	215	212
56	216	219	221
60	214	216	219



O teste 03, por fim, reforça a tendência já mostrada nos dois testes anteriores, de que a partir de um *buffer* de 16 KB o tempo de transmissão não varia mais tanto. Desta maneira, conclui-se que o tamanho ideal para um *buffer* é de 16 KB considerando os fatores tempo de transmissão e carga de processamento do *peer relay*.

5 CONCLUSÕES

Neste capítulo são analisados e discutidos quais os resultados, dificuldades e possibilidades da plataforma JXME que foram encontradas durante a pesquisa e desenvolvimento do aplicativo SIMCA.

5.1 RESULTADOS DA PESQUISA

Este trabalho atingiu na íntegra seu objetivo inicial, que é o de implementar um sistema para compartilhamento de arquivos entre celulares usando a tecnologia P2P aliada à plataforma JXME.

Isso foi possível devido ao fato de atualmente os telefones celulares possuírem menos limitações que há alguns anos atrás e continuam evoluindo de maneira muito rápida. Os dispositivos acessíveis ao usuário comum são agora capazes de participar de uma rede P2P, possuindo cartões de memória com capacidades nunca antes pensadas.

Sendo assim, um *peer* JXME pode enviar e receber arquivos de grande porte como vídeos e imagens, que com a introdução das redes 3G serão transmitidos em velocidades superiores até a da Internet de banda larga.

A plataforma JXME funciona independente de modelo de aparelho celular e plataforma do mesmo, tornando-se bastante flexível para o desenvolvimento de aplicações P2P móveis.

5.2 DIFICULDADES

Para que a criação de uma rede P2P entre celulares fosse possível, era necessária a utilização dos recursos e protocolos da plataforma JXME juntamente com a linguagem J2ME. Porém tendo em vista que a plataforma JXME é uma tecnologia ainda em desenvolvimento, a

documentação da mesma é muito escassa, sendo encontradas apenas referências básicas em alguns artigos e trabalhos de conclusão de curso.

Além deste fato, de acordo com a comunidade JXME, a plataforma nunca foi usada para desenvolvimento de aplicações de compartilhamento de arquivos como o SIMCA, e justo por isso apresentou inúmeras barreiras para que o aplicativo proposto nesta pesquisa funcionasse. Abaixo seguem três limitações que foram encontradas e que, se não contornadas, tornariam o SIMCA inoperante:

a) impossibilidade na transmissão de arquivos com mais de 64 KB;
b) impossibilidade no anúncio de diretórios compartilhados;
c) dificuldade no anúncio de *peers* conectados.

A impossibilidade na transmissão de arquivos com mais de 64 KB foi resolvida pelo software construído neste estudo através da segmentação dos arquivos enviados pela rede, o que permite a transmissão de documentos com qualquer tamanho. A rotina desenvolvida, que é disponibilizada no apêndice A, foi submetida aos coordenadores do projeto JXME e está sendo estudada pelos mesmos para ser liberada junto com a plataforma em versões futuras.

Quanto à impossibilidade no anúncio de diretórios compartilhados, foi necessária a utilização da JSR 75, que acessa o diretório do celular e, pesquisando no mesmo, cria uma mensagem de texto com todos os arquivos contidos neste diretório e envia ao *peer relay*, que o mantém e utiliza para efetuar pesquisas quando requisitado.

Mas a dificuldade no anúncio de *peers* conectados não dependia nem do SIMCA, nem da plataforma JXME e sim dependia do aplicativo JXTA Shell, que apresentava uma limitação desta funcionalidade para *peers* JXME que se conectassem a ele. Esta limitação gerou a abertura de um relatório de não-conformidade junto à comunidade responsável pelo JXTA Shell, sendo esta limitação posteriormente corrigida por uma atualização do programa.

5.3 CONSIDERAÇÕES FINAIS

Neste estudo foi construído um programa capaz de conectar telefones celulares e criar entre eles uma rede de compartilhamento de arquivos, baseando-se em uma arquitetura centralizada, na qual todos os *peers* (celulares) conectavam-se em um servidor central (*proxy* ou *peer relay*).

Através do desenvolvimento dessa aplicação, confirmou-se a viabilidade da transmissão de dados entre celulares, mas além disso, evidenciou-se que a disseminação desta atividade é apenas uma questão de tempo, visto que a tecnologia envolvida neste processo evolui constantemente.

A evolução tecnológica provou também ter um preço, pois tanto o JXME quanto os celulares são plataformas em estado de maturação visto que ainda não fornecem um suporte confiável para este tipo de softwares.

Sendo assim, apesar das dificuldades apresentadas, esta é uma área promissora e que tende a desenvolver-se de forma extremamente rápida, apresentando num futuro próximo a popularidade hoje vista com as aplicações P2P convencionais.

5.4 PESQUISAS FUTURAS

Com a evolução das redes de transmissão de dados e dos celulares em si, uma alternativa muito interessante ao software aqui desenvolvido seria a adoção de uma arquitetura descentralizada, na qual não existiria a figura do *proxy* e todos os *peers* teriam o papel de servidores na rede P2P.

Além desta, surge com grande força nos grupos de discussão da tecnologia Java a utilização dos *Location Based Services* (LBS), estes, nada mais são do que aplicações que usam a localização geográfica atual do usuário para que este interaja de forma mais expressiva num determinado contexto. Sendo assim, poderia ser desenvolvida uma aplicação P2P que se conectasse e formasse grupos apenas com celulares que estão próximos de si geograficamente.

REFERÊNCIAS BIBLIOGRÁFICAS

ARORA, Akhil; HAYWOOD, Carl; PABLA, S. Kuldip. **JXTA for J2ME:** extending the reach of wireless with JXTA technology. Palo Alto, 2002. Disponível em: <http://www.jxta.org/project/www/docs/JXTA4J2ME.pdf>. Acesso em: 19 mar. 2006.

DOCHSTADER, Mark. **Peer to peer networking:** next big boom or next big bust?, [S.l.]. 2001. Disponível em: <http://networking.ittoolbox.com/documents/peer-publishing/peer-to-peer-networking-next-big-boom-or-next-big-bust-1842>. Acesso em: 26 mar. 2006.

DURANTE, Gabriel Barros. **Redes peer-to-peer**. Rio de Janeiro, [2004]. Disponível em: <http://www.gta.ufrj.br/grad/04_1/p2p/>. Acesso em: 19 mar. 2006.

GOMES, Alexandre. Tutorial J2ME: visão geral, [S.l.], [2006?]. Disponível em: <http://www.mundooo.com.br/php/mooartigos.php?pa=showpage&pid=9>. Acesso em: 27 mar. 2006.

GSM ARENA. **GSM Arena**. Washington. USA, [2006?]. Disponível em: <http://www.gsmarena.com>. Acesso em: 03 ago. 2006.

HANDFORD, Richard. Celulares abrem mercado para mídia. **Revista Digital**, Porto Alegre, n. 288, 2005. Disponível em: <http://www.revistadigital.com.br/tendencias.asp?NumEdicao=288&CodMateria=2488>. Acesso em: 19 mar. 2006.

JSR-36. **J2ME Connected Device Configuration 1.1**. San Jose. USA, [2006?]. Disponível em: <http://jcp.org/en/jsr/detail?id=36>. Acesso em: 15 ago. 2006.

JSR-37. **J2ME Mobile Information Device Profile**. San Jose. USA, [2006?]. Disponível em: <http://jcp.org/en/jsr/detail?id=37>. Acesso em: 16 ago. 2006.

JSR-75. **Java Specification Requests**. San Jose. USA, [2006?]. Disponível em: <http://jcp.org/en/jsr/detail?id=75>. Acesso em: 16 ago. 2006.

JSR-139. **J2ME Connected Limited Device Configuration 1.1**. San Jose. USA, [2006?]. Disponível em: <http://jcp.org/en/jsr/detail?id=139>. Acesso em: 15 ago. 2006.

JXTA. **Project JXTA**. San Jose. USA, [2006?]. Disponível em: <http://www.jxta.org>. Acesso em: 02 abr. 2006.

JXTA PROGRAMMER GUIDE. **Project JXTA v2.3: Java Programmer´s Guide**, San Jose, USA, 2005. Disponível em: <http://www.jxta.org/docs/JxtaProgGuide_v2.3.pdf>. Acesso em: 15 jun. 2006.

JXTA SPECIFICATION. **JXTA v2.0 Protocols Specifcation**. San Jose. USA, [2006]. Disponível em: <http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.html>. Acesso em: 12 jul. 2006.

JXTA ZONE. **Demystifying Pipes, JxtaSockets, JxtaMulticastSocket, and JxtaBiDiPipes**. San Jose. USA, [2006?]. Disponível em: <http://blogs.sun.com/hamada/entry/pipes_jxtabidipipes_and_jxtasockets>. Acesso em: 21 ago. 2006.

LEITE, Jair C. **Análise e especificação de requisitos**. Rio Grande do Norte, [1999]. Disponível em: <http://www.dimap.ufrn.br/~jair/ES/es991c3.html>. Acesso em: 16 set. 2006.

LIMA, Alessandro V. F. **P2P, LBS e comunidades virtuais:** os ingredientes para aplicações inovadores em sistemas 3G. 2005. 51f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife.

LV, Qin; CAO, Pei; COHEN, Edith; LI, Kai; SHENKER, Scott. Search and Replication in Unstructured Peer-to-Peer Networks. In: ACM INTERNATIONAL CONFERENCE ON SUPERCOMPUTING, 16., 2002, Nova York. **Proceedings...** Nova York, 2002. p. 1-2.

MATSUMOTO, Patrícia Megumi. **Sun Java Wireless Toolkit (WTK)**. São Paulo, [2005]. Disponível em: <http://www.linux.ime.usp.br/~patty/mac499/tecnica/tecnologias/wtk.html>. Acesso em: 25 set. 2006

MINISTÉRIO DA CIÊNCIA E TECNOLOGIA. Brasília, [2005]. Disponível em: <http://www.mct.gov.br/SEPIN/Imprensa/Noticias_5/Telecom_5.htm>. Acesso em: 19 mar. 2006.

MOBILE PHONES UK. **What is GPRS?**. United Kingdom, [2006?]. Disponível em: <http://www.mobile-phones-uk.org.uk/gprs.htm>. Acesso em: 22 ago. 2006.

NOKIA 3250. **Nokia 3250**. [S.l.], [2006?]. Disponível em: <http://europe.nokia.com/3250>. Acesso em: 30 ago. 2006.

OLIVEIRA, Eric C. M. **Programação Java com IDE NetBeans**. [S.l.], [2005]. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=805>. Acesso em: 28 set. 2006.

P2P ARCHITECTURES. **P2P Architectures**. Aarhus. Dinamarca, [2003]. Disponível em: <http://www.daimi.au.dk/~marius/p2p-course/lectures/03/talk.html>. Acesso em: 18 ago. 2006.

PILONE, Dan; PITMAN, Neil. **UML 2:** rápido e prático. Castelo Rio de Janeiro: Alta Books, 2006.

PINHEIRO, Christiano. **J2ME:** java para os portáteis. [S.l.], [2003]. Disponível em: <http://www.imasters.com.br/artigo.php?cn=1539&cc=19>. Acesso em: 27 mar. 2006.

TRAVERSAT, Bernard et al. Project JXTA 2.0 Super-Peer Virtual Network. Palo Alto, 2003. Disponível em: <www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf>. Acesso em: 20 jul. 2006.

VILCEK, Alexandre. Mobile peer-to-peer com JXTA. **Web Mobile**, Rio de Janeiro, n. 7, p. 62-72, fev. 2006.

VOSS, José Junior. **Protótipo de software para compartilhar informações entre computadores através da tecnologia peer-to-peer (P2P), usando a plataforma JXTA**. 2004. 70f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

WI-FI PLANET. **3G Definition**. Darien. USA, [2006?]. Disponível em: <http://wi-fiplanet.webopedia.com/TERM/3/3G.html>. Acesso em: 20 ago. 2006.

WILSON, Brendon J. **Project JXTA book**, Indianapolis, USA, [2002]. Disponível em: <http://www.brendonwilson.com/projects/jxta/pdf/JXTA.pdf>. Acesso em: 30 maio 2006.

YUAN, Michael. **Móbile P2P Messaging**. Austin, [2003]. Disponível em: <http://www-128.ibm.com/developerworks/wireless/library/wi-p2pmsg2/>. Acesso em: 12 ago. 2006.

APÊNDICE A – Solução para limitação de tamanho de arquivo transferido

Conforme descrito na seção de dificuldades encontradas no desenvolvimento do aplicativo SIMCA, um dos maiores problemas da plataforma JXME é a impossibilidade de transmitir arquivos maiores de 64 KB. Esta limitação foi resolvida pela rotina exibida no Quadro 18, que segmenta os arquivos de tamanhos maiores e os envia em vários pedaços ao *peer* requisitante.

```
byte[] allbytes = BytesArquivo;
int TamanhoArquivo = allbytes.length;

byte[] inputbytes = new byte[BUFFERSIZE];

int FimSegmento = 0;
int outByteCounter = 0;
int IdSegmento = 0;

/*1: Enviar arquivo em segmentos */
while (outByteCounter <= TamanhoArquivo) {

  /*1.1: Determina qual segmento será enviado */
  if ((outByteCounter + BUFFERSIZE) >= TamanhoArquivo) {
    System.arraycopy(allbytes, outByteCounter, inputbytes, 0,
                     BUFFERSIZE);
    FimSegmento = BUFFERSIZE + outByteCounter;
    outByteCounter = outByteCounter + inputbytes;
  }

  /*1.2 Criar mensagem do segmento */
  SegmentaArquivo(Nome, inputbytes, String.valueOf(outByteCounter),
                  String.valueOf(FimSegmento),
                  String.valueOf(TamanhoArquivo));
}
```

Quadro 18 – Trecho do código fonte de segmentação de arquivos

No código é possível observar que a rotina enviará segmentos de *bytes* enquanto os *bytes* que já foram enviados (`outByteCounter`) forem menores que o tamanho do arquivo. Logo após, o SIMCA, através da função `System.arraycopy()` cria um segmento de arquivo, baseando-se em quantos *bytes* já foram enviados, no total a ser transmitido (`allbytes`) e no tamanho do *buffer* (`BUFFERSIZE`), gravando na variável `inputbytes` os *bytes* a serem transmitidos neste segmento.

Por fim, a rotina `SegmentaArquivo()` é chamada, passando-se para ela o nome do

arquivo que está sendo transmitido, junto de quantos *bytes* já foram enviados, até qual *byte* está sendo mandado neste segmento (`FimSegmento`) e qual o tamanho total do arquivo (`TamanhoArquivo`).