UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
ESPECIALIZAÇÃO EM SISTEMAS DE COMPUTAÇÃO PARA WEB

UMA APLICAÇÃO DA COMPUTAÇÃO MÓVEL PARA A EMPRESA CHIPCARD

Monografia de Especialização

Miriam Thiel

Santa Maria, RS, Brasil
2007

UMA APLICAÇÃO DA COMPUTAÇÃO MÓVEL PARA A EMPRESA CHIPCARD

por

Miriam Thiel

Monografia apresentada ao curso de Especialização em Sistemas de Computação para Web, da Universidade Federal de Santa Maria (UFSM, RS) como requisito parcial para obtenção do grau de **Especialista em Sistemas de Computação para Web.**

Orientadora: Profª. Iara Augustin

Santa Maria, RS, Brasil

2007

Universidade Federal de Santa Maria
Centro de Tecnologia
Especialização em Sistemas de Computação para Web

A Comissão Examinadora, abaixo assinada, aprova a Monografia

UMA APLICAÇÃO DA COMPUTAÇÃO MÓVEL PARA A EMPRESA CHIPCARD

elaborada por

Miriam Thiel

como requisito parcial para obtenção do grau de

Especialista em Sistemas de Computação para Web

COMISSÃO EXAMINADORA:

Iara Augustin
(Presidente/Orientadora)

Márcia Pasin (UFSM)

Oni R. Sichonany (UFSM)

Santa Maria, 29 de janeiro de 2007.

RESUMO

Monografia de Especialização
Curso de Especialização em Sistemas de Computação para Web
Universidade Federal de Santa Maria

"**UMA APLICAÇÃO DA COMPUTAÇÃO MÓVEL PARA A EMPRESA CHIPCARD"**
AUTOR: MIRIAM THIEL
ORIENTADOR: IARA AUGUSTIN
Data e Local de Defesa: Santa Maria, Janeiro 30, 2007.

Aplicações que usam dispositivos móveis são extremamente interessantes, pois permitem que um usuário acesse informações sem estar fisicamente conectado a Internet.

Este trabalho foi desenvolvido com o objetivo de aplicar os conhecimentos adquiridos durante o curso, e para disponibilizar aos clientes da CHIPCARD uma aplicação para dispositivos móveis que permite efetuar consultas de crédito em locais onde a utilização de microcomputadores com acesso a Internet não está disponível.

Para a realização desse trabalho foi utilizado o `J2ME` para desenvolver a aplicação para dispositivos móveis e para a conexão do dispositivo com o servidor foi utilizado o protocolo TCP/IP através do endereço IP do `Socket`. No servidor existe uma aplicação em Clarion (linguagem de programação) com Visual Basic que possui suporte a `socket`. O sistema coleta a informação do telefone celular e processa a informação recebida retornando a resposta da consulta ao aparelho através do `socket`. Para a modelagem são utilizados os diagramas de casos de uso e diagramas de classe.

Palavras-Chave: Aplicação móvel, J2ME, dispositivos móveis, chipcard.

ABSTRACT

Monografia de Especialização
Curso de Especialização em Sistemas de Computação para Web
Universidade Federal de Santa Maria

"AN APPLICATION OF THE MOBILE COMPUTING FOR COMPANY CHIPCARD"

AUTHOR: MIRIAM THIEL
ADVISOR: IARA AUGUSTIN
Defense Date and Local: Santa Maria, January 30,2007.

Applications that use mobile devices are extremely interesting therefore allow that the user has access information without being physically connected the Internet. This work was developed aims to apply the knowledge acquired during the course, and to offer for the customers of the CHIPCARD an application for mobile devices that allows to effect consultations of credit in places where the use of microcomputers with access the Internet is not available.

For the accomplishment of this work the J2ME was used to develop the application for mobile devices and for the connection of the device with the server protocol TCP/IP through address IP of the Socket was used. In the server it exist an application in clarion (programming language) with Visual Basic that it possess support to socket. The system collects the information of the cellular telephone and processes the received information returning the reply from the consultation to the device through socket. For the modeling the diagrams of use cases and diagrams of classroom are used.

Keywords: Mobile application, J2ME, mobile devices, CHIPCARD.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

- **J2ME** **Java 2 Micro Edition**
- **GPRS** **General Packet Radio Service**
- **TCP** **Transmission Control Protocol**
- **IP** **Internet Protocol**
- **J2SE** **Java 2 Standard Edition**
- **UML** **Unified Modeling Language**
- **HTTP** **HyperText Transfer Protocol**
- **MIDP** **Mobile Information Device Profile**
- **OO** **Orientação a Objetos**
- **J2EE** **Java 2 Enterprise Edition**
- **API** **Application Programming Interface**
- **CPF** **Cadastro de pessoa física**
- **CNPJ** **Cadastro Nacional de Pessoa Jurídica**
- **AL** **Alerta Nacional**
- **RE** **Restrição**
- **RL** **Referência Nacional**
- **CCF** **Cadastro de Emitentes de Cheques sem Fundos**
- **XML** **eXtensible Markup Language**

SUMÁRIO

INTRODUÇÃO

Aplicações para dispositivos móveis é um dos assuntos mais atuais no mundo que une telecomunicações e tecnologia da informação. Nessa nova área de negócios residem aplicações corporativas que começam a abandonar os *desktops* e caminhar em direção a dispositivos portáteis do tipo *palmtops*, *handhelds* e celulares, cuja praticidade, baixo custo e capacidade tem motivado muitas empresas a planejarem a sua utilização no seu processo de automação. Recentes avanços na tecnologia empregada em tais dispositivos contribuíram para diminuir o custo dos mesmos e torná-los acessíveis a um grupo maior de pessoas.

Dispositivos móveis têm se tornado objetos atraentes na medida em que cada vez mais pessoas utilizam telefones celulares, em qualquer lugar e a qualquer hora. Uma característica muito importante desses dispositivos móveis é sua habilidade para se conectar à Internet e executar aplicações Web. Aplicações móveis podem ser desenvolvidas para enviar qualquer tipo de dados a qualquer usuário, em qualquer parte do mundo.

Essas aplicações podem ser criadas a partir de linguagens de programação que suportam os parâmetros inerentes ao projeto de software com essa finalidade e devem considerar tamanho do aplicativo, alocação de memória, demanda de área de armazenamento, comportamento da rede, e assim por diante. Aplicações móveis típicas incluem acesso remoto a dados, sendo muitas delas dirigidas à criação de facilidades para aumento da produtividade de escritórios e empresas. Essas aplicações são projetadas com base em modelos de componentes distribuídos e devem ser capazes de retomar seu curso, no caso de a rede sofrer uma desconexão momentânea.

Hoje com a atual tecnologia disponível é comum encontrar aparelhos celulares com poder de processamento igual ou até superior a computadores do início da década passada. Esses celulares podem rodar aplicativos e jogos. A miniaturização dos dispositivos eletrônicos e o aumento do poder de processamento desses dispositivos aliado à necessidade de o ser humano permanecer informado a qualquer hora e em qualquer lugar foi um dos motivos da grande popularização dos

telefones celulares. Os telefones celulares mais modernos podem rodar aplicativos até então existentes apenas para computadores.

O desenvolvimento de aplicações para dispositivos móveis tem crescido de maneira vertiginosa. Poder construir soluções disponíveis para o usuário onde quer que ele esteja representa uma considerável possibilidade de expansão para o mercado de software.

Este trabalho realizará a implementação de um sistema de consulta de cheques por telefone celular. O objetivo geral deste trabalho é aplicar na prática os conhecimentos adquiridos no curso de Especialização em Sistemas de Computação para web estudando as tecnologias que melhor se aplicam ao desenvolvimento de uma aplicação para a empresa CHIPCARD Sistemas Inteligentes. O objetivo específico do trabalho é projetar, modelar e implementar uma aplicação de consulta de crédito baseado em dispositivos móveis (telefones celulares) para atender a demanda da empresa CHIPCARD.

Uma das áreas de atuação da empresa CHIPCARD é a de Consulta de Crédito. A possibilidade de disponibilizar para os clientes um tipo de consulta on-line em dispositivos móveis é de grande interesse para a empresa. Com a implantação de um sistema de consulta em telefones celulares, por exemplo, oferecer-se-á aos clientes uma capacidade de mobilidade que não existe hoje, permitindo que transações de vendas seguras sejam feitas em feiras, em sistemas de *delivery* ou em qualquer outro lugar onde a presença de um microcomputador não seja possível. Esse tipo de operação também acaba se beneficiando pela grande redução do custo de comunicação e a alta taxa de disponibilidade oferecida por serviços de comunicação como o GPRS.

O continuação deste texto detalha o trabalho realizado segundo a estrutura: No capítulo 2 serão apresentadas as tecnologias utilizadas no desenvolvimento da aplicação, no capítulo 3 a modelagem e detalhes de implementação da aplicação, no capítulo 4 a fase de teste. Para finalizar, as conclusões e trabalhos futuros são apresentados.

1 TECNOLOGIAS PARA DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS

Nesse capítulo serão descritas as tecnologias utilizadas para o desenvolvimento da aplicação, entre elas o J2ME, J2SE, protocolo HTTP e MIDP e também a modelagem UML.

1.1 O uso do J2ME na construção da aplicação proposta

A grande variedade de processadores e protocolos existentes no mundo da computação sem fio, segundo Fernando Pereira, Marco Valente, Roberto E Mariza Bigonha (2005), compromete a portabilidade de programas entre dispositivos móveis e constitui um obstáculo aos desenvolvedores de aplicações, pois, sem um padrão universalmente adotado, dificilmente um aplicativo projetado para determinado tipo de processador poderá ser utilizado em um aparelho diferente sem necessitar de modificações em seu código executável. A fim de amenizar os problemas causados pela grande variedade de protocolos existentes e prover aos projetistas de aplicações um modelo de desenvolvimento comum, foi desenvolvida a plataforma J2ME, um ambiente de execução Java que requer menos de um décimo dos recursos necessários ao sistema Java tradicional, conhecido como J2SE.

A tecnologia Java, segundo Brito e Robinson Cris (2006), disponibiliza recursos para desenvolver programas para estes pequenos dispositivos através de uma versão especial chamada Java 2 Micro Edition (J2ME). Essa versão apresenta vários recursos e bibliotecas para o desenvolvimento de aplicações para dispositivos móveis. O Java se destaca de outras linguagens de programação, pois dispõe de recursos de Orientação a Objetos, permitindo desenvolver códigos portáveis, reusáveis (que permitem a reutilização do código entre diferentes processadores e facilita a comunicação entre sistemas distribuídos) e robustos. A plataforma de desenvolvimento Java possibilita desenvolver aplicativos para qualquer dispositivo, desde *SmartCard's*, passando por celulares e computadores até grandes servidores. Desde que disponha uma máquina virtual Java (JVM). Esse recurso possibilita que o mesmo programa rode em várias plataformas.

O J2ME dispõe de recursos básicos e padrões, disponíveis na grande maioria dos aparelhos para celulares que possuem recursos a mais como teclas adicionais, e alertas vibratórios. Conectividades diferentes são disponibilizadas pelo próprio fabricante do aparelho, como API's adicionais, garantindo que o programa utilize o máximo dos recursos disponíveis.

Hoje é possível desenvolver aplicativos para celulares sem investir em recursos para adquirir ferramentas para o desenvolvimento,ou seja, é possível trabalhar somente com softwares livres.

Conforme Gomes (2006), a plataforma Java 2 Micro Edition tem por objetivo definir um conjunto de tecnologias centradas no desenvolvimento de soluções para dispositivos cujo propósito original não seja o processamento de dados, mas pelo fato de carregarem algum tipo de microprocessador, são potencialmente capazes de realizar operações computacionais. Através do J2ME torna-se possível desenvolver, atualizar e instalar novas aplicações segundo as necessidades particulares de cada usuário.

Algumas das vantagens em se utilizar a tecnologia Java são: dinamismo (novas aplicações podem ser baixadas da rede e instaladas no dispositivo a qualquer tempo), segurança (garantem a proteção das informações carregadas pelo dispositivo, onde dados de uma aplicação não são acessíveis por outras aplicações), portabilidade (aplicações podem ser portadas entre dispositivos de diferentes fabricantes e de diferentes tipos) e OO (alto nível de abstração do código, modularização e reusabilidade).

Java apresenta diversas configurações, cada uma delas específica para um determinado grupo de processadores. A configuração destinada a aparelhos de telefonia celular, denominada CLDC, é muito simples. Embora a CLDC disponibilize aos desenvolvedores de aplicações uma ampla variedade de funcionalidades, não contém muitos dos serviços tradicionalmente encontrados em outras versões da linguagem Java, como o mecanismo de Invocação Remota de Métodos.

1.2 J2SE

O J2SE (Java 2 Standard Edition) é o ambiente de desenvolvimento mais utilizado para aplicações móveis. Isso porque seu uso é voltado a computadores

pessoais e servidores, onde há bem mais necessidade de aplicações. Além disso, pode-se dizer que essa é a plataforma principal, já que, de uma forma ou de outra, o J2EE(Java 2 Enterprise Edition) e o J2ME têm sua base aqui. Pode-se dizer que esses ambientes de desenvolvimento são versões aprimoradas do J2SE para as aplicações a que se propõem.

1.3 Protocolo HTTP e MIDP

Com a popularização da Internet há alguns anos, o protocolo de comunicação HTTP se tornou onipresente em servidores . Mais do que simplesmente uma forma de ligação entre navegadores e Web Servers, o HTTP passou a ser um denominador comum na conexão entre computadores, desde servidores até pequenos dispositivos como celulares e PDAs. Isso se comprova na própria API (Application Programming Interface) de conectividade disponível no J2ME (Java 2 Plataform, Micro Edition), que oferece uma maneira muito simples de manipular este protocolo.

Todo o processo de conexão e transferência de dados será feito utilizando-se o protocolo HTTP. O Midlet desenvolvido vai respeitar totalmente a implementação da API do MIDP 1.0, garantindo com isso que a aplicação desenvolvida seja compatível com qualquer dispositivo móvel que suporte JAVA.

As consultas de crédito *on-line* são processadas em um servidor Web, rodando as ferramentas apache, PHP e mysql. A aplicação executando no dispositivo móvel envia a solicitação de consulta ao servidor usando o protocolo HTTP e a função GET e recebe um *array* de bytes com o resultado da consulta efetuada, a ser apresentada ao usuário.

1.4 Modelagem UML

A UML (Unified Modeling Language) apresenta uma série de diagramas para a modelagem de sistemas orientados a objetos. Os diagramas mais comuns são o diagrama de casos de uso (representa as funcionalidades de um sistema) e o diagrama de classes descreve as classes do modelo numa visão estática.

Segundo Ivan Jacobson (2005) , um caso de uso é um documento narrativo que descreve a seqüência de eventos de um ator que usa um sistema para completar um processo. Um caso de uso é uma técnica de modelagem usada para descrever o que um novo sistema deve fazer. Ele é construído através de um processo interativo no qual as discussões entre o cliente e os desenvolvedores do sistema conduzem a uma especificação do sistema da qual todos estão de acordo.

A classe é uma descrição de um tipo de objeto. Todos os objetos são exemplares das classes, onde a classe descreve as propriedades e comportamentos dos mesmos. As classes são utilizadas para classificar os objetos que identificamos no mundo real, sendo assim, elas devem ser retiradas do domínio do problema e serem nomeadas pelo que elas representam no sistema. Essa atividade deve ser feita por alguém com bastante conhecimento do domínio do problema.

Este diagrama lista todos os conceitos do domínio que serão implementados no sistema. O diagrama de classe é muito importante porque descreve a estrutura do sistema a desenvolver.

2 UMA APLICAÇÃO DA COMPUTAÇÃO MÓVEL PARA A EMPRESA CHIPCARD – ESTUDO DE CASO

Neste capítulo será descrito o estudo de caso da aplicação desenvolvida para a empresa CHIPCARD.

2.1 Uma aplicação da computação móvel para a empresa CHIPCARD

Em alguns locais, onde a utilização de um microcomputador com acesso a Internet não está disponível, o telefone celular pode ser a melhor opção para consulta a banco de dados remotos. Na solução criada para a CHIPCARD, é possível a consulta a um banco de dados com informações de crédito, sendo utilizada como chave de consulta o CPF ou o CNPJ a ser consultado.

O usuário seleciona a aplicação de consulta de crédito no aparelho celular; a aplicação abre um campo para a digitação do CPF/CNPJ; o usuário digita o CPF/CNPJ; e a aplicação acessa os dados no servidor da CHIPCARD através do protocolo TCP/IP pelo endereço IP definido no socket, utilizando a porta 9311. O servidor busca em sua base de dados, informações referentes ao cadastro de emissores de cheques sem fundos do banco central, cheques sustados do Banco do Brasil (alerta nacional), restrições comerciais inseridas no sistema por outros usuários ou referências locais. Efetuada a consulta, retorna a resposta para o usuário, que pode ser:

- “consulte[xx]” , no caso de cheques sem fundos emitidos sendo que o XX representa o número de cheques;
- “consulte[AL]”, no caso de ocorrência de alerta nacional;
- “consulte[RE]”, quando encontrada alguma restrição no sistema;
- “consulte[RL]”, no caso de existir alguma referência local para o CPF/CNPJ consultado;
- “inválido”, quando o CPF/CNPJ informado for inválido e

- “nada consta”, quando o CPF/CNPJ não tiver nenhuma restrição.

2.2 Modelagem UML

Será utilizado o diagrama de caso de uso para descrever a funcionalidade do sistema.

2.2.1 Cliente consulta crédito

No diagrama de caso de uso, o cliente acessa a aplicação, digita o CPF/CNPJ a ser consultado conforme figura 2.1.

2.2.1.1 Diagrama de caso de uso

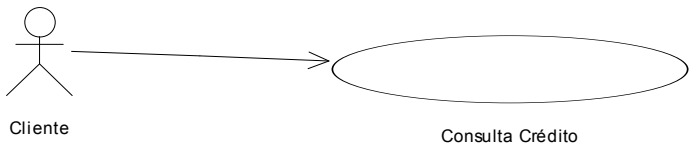

Figura 2.1 Diagrama de caso de uso

2.2.1.2 Descrição do caso de uso

1 – Cliente escolhe opção consulta crédito

2 - O sistema mostra campo para digitação de CPF/CNPJ

3 - Cliente digita CPF/CNPJ

4 - O sistema consulta a base de dados

5 - O sistema retorna “Nada Consta”.

5.1 – Cliente tem cheque sem fundos: “Consulte[XX].

5.2 - Cliente tem alerta nacional: “Consulte[AL].

5.3 - Cliente tem restrição no sistema: “Consulte[RE].

5.4 - Cliente tem referência local: “Consulte[RL].

5.5 - CPF/CNPJ digitado tem número de dígitos incorretos: “Inválido”.

2.3 Adaptação para a Mobilidade

A aplicação existente foi remodelada para permitir o acesso através de dispositivos móveis conforme a figura 2.2. A nova aplicação consiste em três classes: SocketMidlet, Client, Sender.

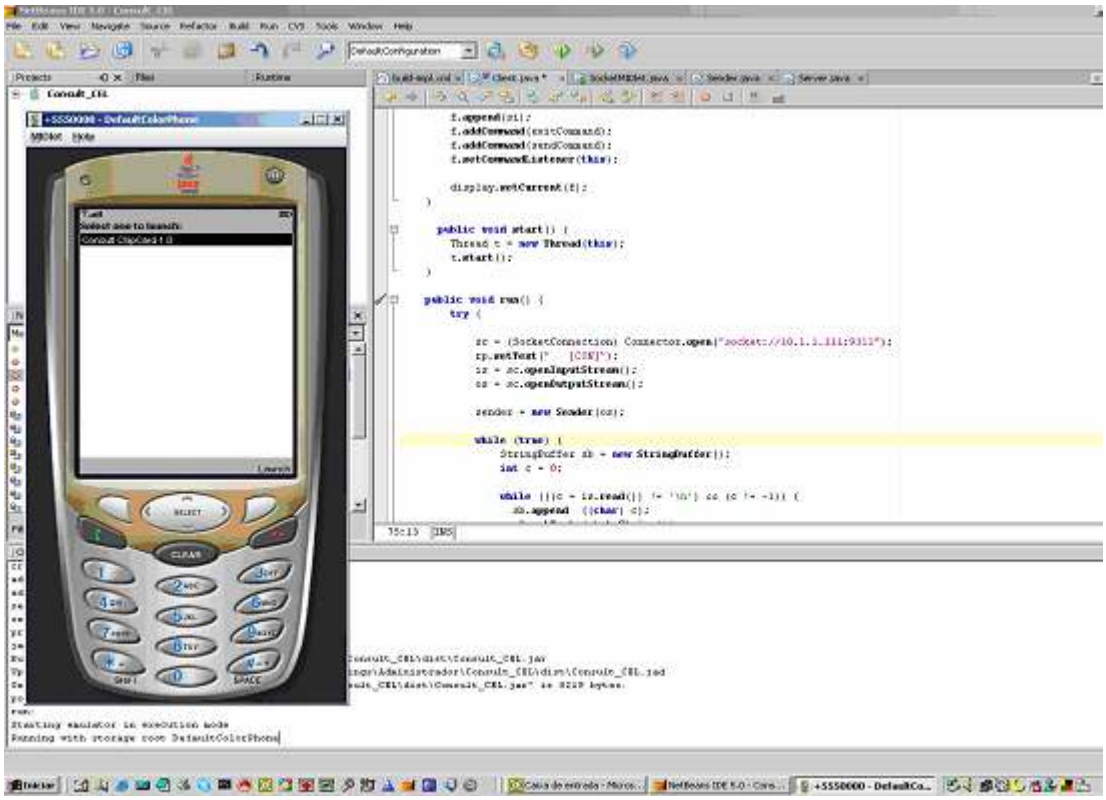

Figura 2.2 Execução da simulação do dispositivo móvel

2.3.1 Classe SocketMidlet

A primeira classe, `SocketMidlet (veja as figuras 2.3 e 2.4)`, é a classe da aplicação responsável pela gerência do `midlet` e a chamada da tela principal da aplicação, que é a classe `client`.

2.3.1.1 Descrição da classe `SocketMidlet`

A classe `SocketMidlet` estende a classe `Midlet`, e faz a chamada e instancia a classe `client`, que é a classe principal da aplicação. A função da classe `Midlet` é criar o ambiente necessário dentro do dispositivo móvel para que as outras classes da aplicação possam ser executadas.

Algumas variáveis foram declaradas como referência nessa classe, são elas:

- `Private static Display display`; que cria uma nova tela;
- `Private Form f`; // que é a variável que trabalha o formulário.
- `Private Boolean isPaused`;// que é a variável que vai indicar se o aplicativo está ativo.
- `Private Client client`; // que herda a classe `Client`
- `Private Command exitCommand = new Command "Saída", Command.EXIT, 1)` // que cria o botão de saída da aplicação.
- `Private Command exitCommand = new Command "Iniciar", Command.ITEM, 1)//` que cria o botão de iniciar da aplicação.

Na definição pública da classe `SocketMidlet`, instancia uma nova classe `client` e inicia a aplicação através da função `client.start`().

Além disso, são definidas algumas propriedades como:

- `isPaused` - que retorna se a aplicação está ativa ou não

-`startApp` - que seta a variável `isPaused` com `false`, indicando que a aplicação está ativa

- `pauseApp` - que seta a variável `isPaused` com `True` indicando que a aplicação está pausada

- `destroyApp` - que verifica se uma nova classe `client` foi criada e para sua execução através do comando `Client.stop`.

Todo o gerenciamento de execução é feito na rotina `commandAction` que é disparada quando uma tecla é pressionada; no caso de pressionar o botão de saída `(exitCommand)`, a rotina chama o método `destroyApp` e notifica o dispositivo do final de execução chamando a função `notifyDestroyed`. Caso a tecla pressionada seja o botão da aplicação (`startCommand`), uma nova classe `client` é instanciada e inicializada através da função `clientStart`.

2.3.1.2 Diagrama da classe `SocketMidlet`

A figura 2.3 mostra o diagrama da classe descrita anteriormente.

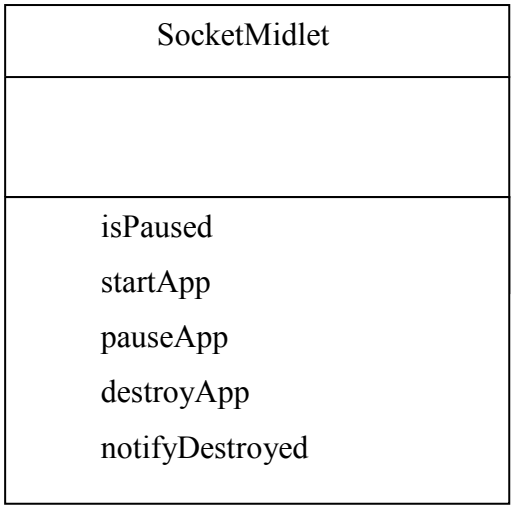

Figura 2.3 Classe `SocketMidlet`

A figura 2.4 mostra o código da implementação da classe `SocketMidlet`

```
public class SocketMIDlet extends MIDlet implements CommandListener {
    private static Display display;
    private Form f;
    private boolean isPaused;
    private Client client;
    private Command exitCommand = new Command("Saida", Command.EXIT,
1);
    private Command startCommand = new Command("Iniciar",
Command.ITEM, 1);
    public SocketMIDlet() {
        client = new Client(this);
        client.start();     }
    public boolean isPaused() {
        return isPaused; }
    public void startApp() {
        isPaused = false;    }
    public void pauseApp() {
        isPaused = true;    }
    public void destroyApp(boolean unconditional) {
        if (client != null) {
            client.stop();       }
    }
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(true);
            notifyDestroyed();
        } else if (c == startCommand) {
            String name = cg.getString(cg.getSelectedIndex());
            client = new Client(this);
            client.start();
        }
    }
}
```

Figura 2.4 Código da classe `SocketMidlet`

2.3.2 Classe `Client`

A segunda classe é a `Client(Veja as figuras 2.5,2.6,2.7 e 2.8)`. É a classe com a tela principal da aplicação e responsável pela apresentação da resposta fornecida pelo servidor na tela do dispositivo móvel.

2.3.2.1 Descrição da Classe `Client`

Nessa classe tem-se a tela de operação do sistema e a chamada à classe `Sender`, que é a classe responsável pelo envio da `string` de consulta ao servidor. Vale ressaltar que a classe `client` tem o atributo `runnable`, que força a existência do método `run` nessa aplicação, propriedade essa que vai conter a maior parte do código responsável pela execução do aplicativo.

Algumas variáveis foram declaradas como referência nessa classe, são elas:

- `Private static Display display`; que cria uma nova tela;
- `Private Form f`; // que é a variável que trabalha o formulário.
- `Private StringItem s1`;// variável `string` utilizada para apresentação de mensagens na tela do dispositivo.
- `Private StringItem s2`;// variável `string` utilizada para apresentação de mensagens na tela do dispositivo.
- `Private StringItem si`;// variável `string` utilizado para criar um campo em branco na tela.
- `Private StringItem rp`;// variável `string` que apresenta na tela o resultado da conexão.
- `Private TextField tf`;// campo de entrada de dados posicionado na tela onde será digitado o CPF/CNPJ para consulta.
- `Private boolean stop`; // variável utilizada para indicar se a aplicação está ativa ou não.

- `Private Command sendCommand = new Command` “consultar”, `Command.ITEM`,1)// que cria o botão que faz a consulta no servidor.
- `Private Command exitCommand = new Command` “Saida”, `Command.EXIT`,1)// que cria o botão para fechar a tela de consulta.
- `ImputStream is`; // declaração do `buffer` de entrada do `Socket`(classe `ImputStream` contida na biblioteca `javax.microedition.io`).
- `OutputStream os`; // declaração do `buffer` de saída do `Socket`(classe `OutputStream` contida na biblioteca `javax.microedition.io`).
- `SocketConnection sc`; // declaração da classe `Socket` a ser utilizada.
- `Sender sender`; // declaração da classe `Sender` descrita no próximo capítulo.

A declaração da classe pública `Client` usando como parâmetro a dependência do `SocketMidlet` inicializa o formulário de operação criando um novo ambiente de tela através da variável `display` e aplicando nessa nova tela um novo formulário (`Form f`), ao mesmo tempo em que inicializa e mostra na tela as variáveis da mesma. A propriedade `Start` dispara uma nova `thread` da classe `Client`. Todo o funcionamento dessa `thread` é gerenciado pela propriedade `run`. É nessa propriedade que o `Socket` é inicializado (`sc = (SocketConnection) Connector.open` (“`socket://10.1.1.111:9311`)) , sendo informado o endereço IP para conexão ao servidor bem como a porta a ser utilizada nessa conexão. Aqui também são criados os dois `buffers` de entrada e saída do `Socket`.

Após as inicializações também é criada uma nova `thread` para classe `sender` que vai ser responsável pelo envio de informações ao servidor. Depois disso a classe `Client` fica gerenciando um laço que lê o `buffer` de entrada (is), e coloca o `byte` lido na variável `sb`(que é uma variável local que foi declarada na inicialização do laço). Também a cada `byte` lido do `buffer` a variável local `sb` é copiada para a propriedade `text` da variável `s2` que é a variável de tela

responsável por apresentar o resultado da consulta ao usuário. Vale notar no código mostrado que a variável local C retorna o `byte` lido do `buffer` de entrada, sendo ela responsável pelo fim do `looping` de leitura, quando recebido um caracter de nova linha `(\n)`, que vai indicar que a resposta foi totalmente recebida ou um valor igual a -1 que indica que a conexão do `Socket` por algum motivo não existe mais. Caso o conteúdo de C seja -1 o `Midlet` é informado que a aplicação será destruída e recebe novamente o controle do processo. Caso nenhuma dessas situações seja encontrada a tela do dispositivo móvel é atualizada.

Uma vez fora do laço de recebimento, a propriedade `stop` da classe `client` é setada e nela todos os `streams` abertos são fechados liberando a aplicação para fim de execução. Também a variável que indica o status da conexão `(rp)` é setada com o texto que indica desconexão ([DES]). Como segurança, dentro do método `run` é feito o gerenciamento de erro utilizando o comando `catch`, que gera um alerta caso o servidor não responda a tentativa de conexão. Também os erros E/S e de execução são tratados dentro dessa classe gerando uma exceção de erro do tipo `printStackTrace`.

O método `commandAction` gerencia o usuário quanto a pressionar o comando `send`, enviando então para o servidor o conteúdo do `tf`, concatenado com as palavras de segurança utilizadas pelo servidor. Nessa especificação ficou definido que toda a consulta feita pelo dispositivo móvel deve ser formada pela palavra `CHI`, mais o CPF/CNPJ consultado sem pontos e barras e a palavra `CARD`.

2.3.2.2 Classe `Client`

A seguir será mostrado o diagrama de classe `Client`.

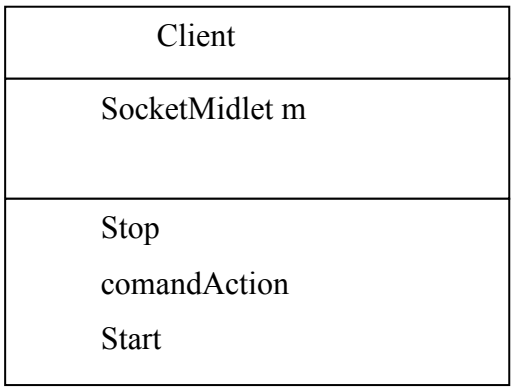

Figura 2.5 Classe `Client`

```
public class Client implements Runnable, CommandListener {
    private SocketMIDlet parent;
    private Display display;
    private Form f;
    private StringItem s1;
    private StringItem s2;
    private StringItem si;
    private StringItem rp;
    private TextField tf;
    private boolean stop;
    private Command sendCommand = new Command("Consultar",
Command.ITEM, 1);
    private Command exitCommand = new Command("Saída",
Command.EXIT, 1);
    InputStream is;
    OutputStream os;
    SocketConnection sc;
    Sender sender;
    public Client(SocketMIDlet m) {
            parent = m;
       display = Display.getDisplay(parent);
       f  = new Form       ("CPF/CNPJ :");
       tf = new TextField  ("", "", 14, TextField.ANY);
       rp = new StringItem ("[Retorno]"," ");
       s1 = new StringItem (" ", " ");
       s2 = new StringItem (" ", " ");
       si = new StringItem (" ", " ");
           f.append(tf);
       f.append(rp);
       f.append(s1);
       f.append(s2);
       f.append(si);
       f.addCommand(exitCommand);
       f.addCommand(sendCommand);
       f.setCommandListener(this);

       display.setCurrent(f);    }
```

Figura 2.6 Código da Classe `Client`

```
public void start() {
   Thread t = new Thread(this);
   t.start();    }    public void run() {       try {
      sc = (SocketConnection) Connector.open("socket://10.1.1.111:9311");
      rp.setText("   [CON]");
      is = sc.openInputStream();
      os = sc.openOutputStream();
      sender = new Sender(os);
      while (true) {
         StringBuffer sb = new StringBuffer();
         int c = 0;
           while (((c = is.read()) != '\n') && (c != -1)) {
           sb.append  ((char) c);
           s2.setText (sb.toString());                        }
          if (c == -1) {
           parent.notifyDestroyed();
           parent.destroyApp(true);
              break;  }
         rp.setLayout (1);
         s1.setLayout (3);
         s1.setText   (tf.getString());
         s2.setLayout (3);
         s2.setText   (sb.toString());
         si.setLayout (1);
         tf.setString (null);                                  }
      stop();
      rp.setText("   [DES]");
      f.removeCommand(sendCommand);
   } catch (ConnectionNotFoundException cnfe) {
      Alert a = new Alert("Cliente", "Servidor OUT",
            null, AlertType.ERROR);
      a.setTimeout(Alert.FOREVER);
      a.setCommandListener(this);
      display.setCurrent(a);
   } catch (IOException ioe) {
      if (!stop) {
         ioe.printStackTrace();          }
   } catch (Exception e) {
      e.printStackTrace();
```

Figura 2.7 código da Classe Client

```
            public void commandAction(Command c, Displayable s) {
                if (c == sendCommand && !parent.isPaused()) {

                  sender.send("CHI"+tf.getString()+"CARD");
                }
                if ((c == Alert.DISMISS_COMMAND) || (c == exitCommand)) {
                  parent.notifyDestroyed();
                  parent.destroyApp(true);
                }
            }
            public void stop() {
               try {
                  stop = true;
                  if (sender != null) {
                     sender.stop();          }
                  if (is != null) {
                     is.close();          }
                  if (os != null) {
                     os.close();          }
                  if (sc != null) {
                     sc.close();
                  }
               } catch (IOException ioe) {}
            }
}
```

Figura 2.8 Código da Classe Client

2.3.3 Classe Sender

A terceira classe é a Sender (Veja as figuras 2.9 e 2.10).É a classe responsável pelo envio de dados pelo *socket*.

2.3.3.1 Descrição da Classe Sender

A classe Sender estende a classe Padrão Thread, e sua função principal é o envio de dados pelo Socket conectado. Essa classe recebe como Parâmetro o buffer a ser enviado.

Algumas variáveis foram declaradas como referência nessa classe, são elas:

- Private OutputStream os; // declaração do buffer de saída do Socket(classe OutputStream contida na biblioteca javax.microedition.io).
- Private String Message; // string que contém a informação a ser enviada ao servidor.

A declaração da classe pública sender define o sincronismo do buffer de saída do Socket utilizada nessa thread com o da classe que a chamou (client). O parÂmetro recebido é atribuído a variável local message e toda a execução é controlada pelo método run.

Dentro do método run existe um laço que é responsável pelo envio dos bytes pelo Socket e pelo gerenciamento de erros de execução.

2.3.3.2 Classe `Sender`

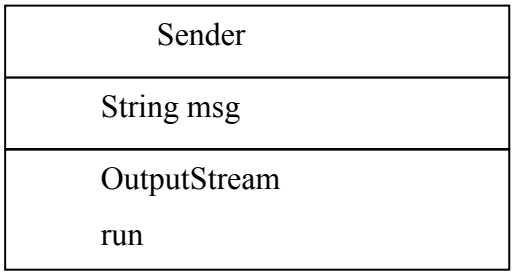

Figura 2.9 Classe `Sender`

```
public class Sender extends Thread {
   private OutputStream os;
   private String message;
   public Sender(OutputStream os) {
      this.os = os;
      start();    }
   public synchronized void send(String msg) {
      message = msg;
      notify();    }
   public synchronized void run() {
      while (true) {
         if (message == null) {
            try {
               wait();
            } catch (InterruptedException e) {
            }
         }
         if (message == null) {
            break;          }
         try {
            os.write(message.getBytes());
         } catch (IOException ioe) {
            ioe.printStackTrace();          }
        message = null;       }
   }
   public synchronized void stop() {
      message = null;
      notify();    }
}
```

Figura 2.10 código da Classe `Sender`

2.4 Descrição da aplicação existente no servidor

A aplicação que roda no servidor do `consult` é composta por duas ferramentas:

- um `bridge` desenvolvido em Visual Basic que gerencia a porta 9311 e permite múltiplas conexões nessa mesma porta. Esse software recebe um `array` de dados dos clientes e faz uma conexão segura com a segunda aplicação;
- uma aplicação, desenvolvida em Clarion (www.clarion.com.br) ,que busca as informações necessárias no banco de dados utilizado. Esse banco usa um padrão proprietário da topspeed (www.clarion.com.br), e tem capacidade para gerenciamento máximo de bancos com até 40GB. Uma vez recebida e processada as informações do `bridge` todas as verificações necessárias num banco de dados são feitas pela aplicação Clarion; entre elas a liberação do tipo de consulta efetuada para o cliente, a validade do código de segurança fornecida pelo cliente, verificando também se o cliente está em dia com seus pagamentos e também a integridade da informação fornecida, feitas todas as verificações iniciais, a aplicação faz a consulta do CPF ou CNPJ informado nos diversos bancos de dados (CCF, restrição comercial, cheques sustados, alerta nacional, histórico de consumo, cheques pré-datados e cadastro nacional de telefone), montando com isso o `array` de dados a ser enviado como resposta.

O processo todo desde a conexão no `bridge` até o envio da resposta não leva mais do que um segundo. O fato da escolha de um `array` de dados para recebimento e envio de informações pelo servidor possibilita a implantação de um mecanismo de comunicação muito simples e suportado por qualquer linguagem ou terminal de consulta permitindo então a implementação de ferramentas de consulta de crédito nos mais diversos dispositivos.

3 TESTES

Para efetuar os testes foram usados CPF/CNPJ válidos, inválidos, com restrições e sem restrições nas seguintes etapas:

- Foram digitados vários CPF/CNPJ com número de dígitos menor do que o padrão, onde retornava a resposta “inválido’.
- Foram digitados vários CPF/CNPJ válidos e sem nenhuma restrição, onde retornava a resposta “nada consta’ (veja as figuras 4.1 e 4.2).
- Foram digitados vários CPF/CNPJ válidos que possuíam cheques sem fundos emitidos, onde retornava a resposta ”consulte[XX]”, sendo o “XX” representa a quantidade de cheques sem fundos foram emitidos.
- Foram digitados vários CPF/CNPJ válidos com cheques sustados, onde retornava a resposta “consulte[AL], sendo que o “AL” representa a quantidade de cheques do Banco do Brasil que foram sustados.
- Foram digitados vários CPF/CNPJ válidos com restrições no sistema, onde retornava a resposta “consulte[RG]” , sendo que o “RG” representa a quantidade de restrições para esse CPF/CNPJ.
- Foram digitados vários CPF/CNPJ válidos com referências locais, onde retornava a resposta “consulte[RL], sendo que o “RL” representa a quantidade de referências locais que foram encontradas nesse CPF/CNPJ.

As figuras 3.1 e 3.2 mostram uma consulta sendo efetuada em um dispositivo móvel.

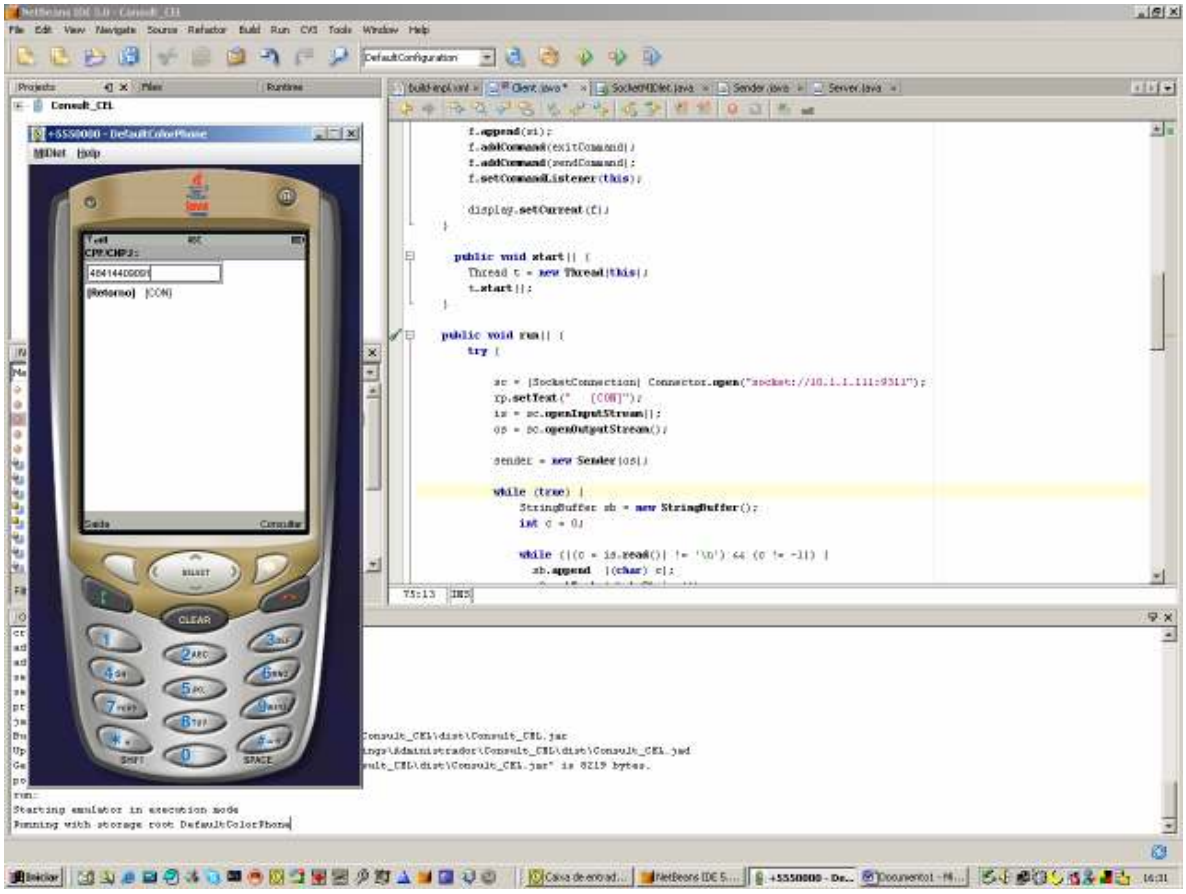

Figura 3.1 Tela de digitação do CPF

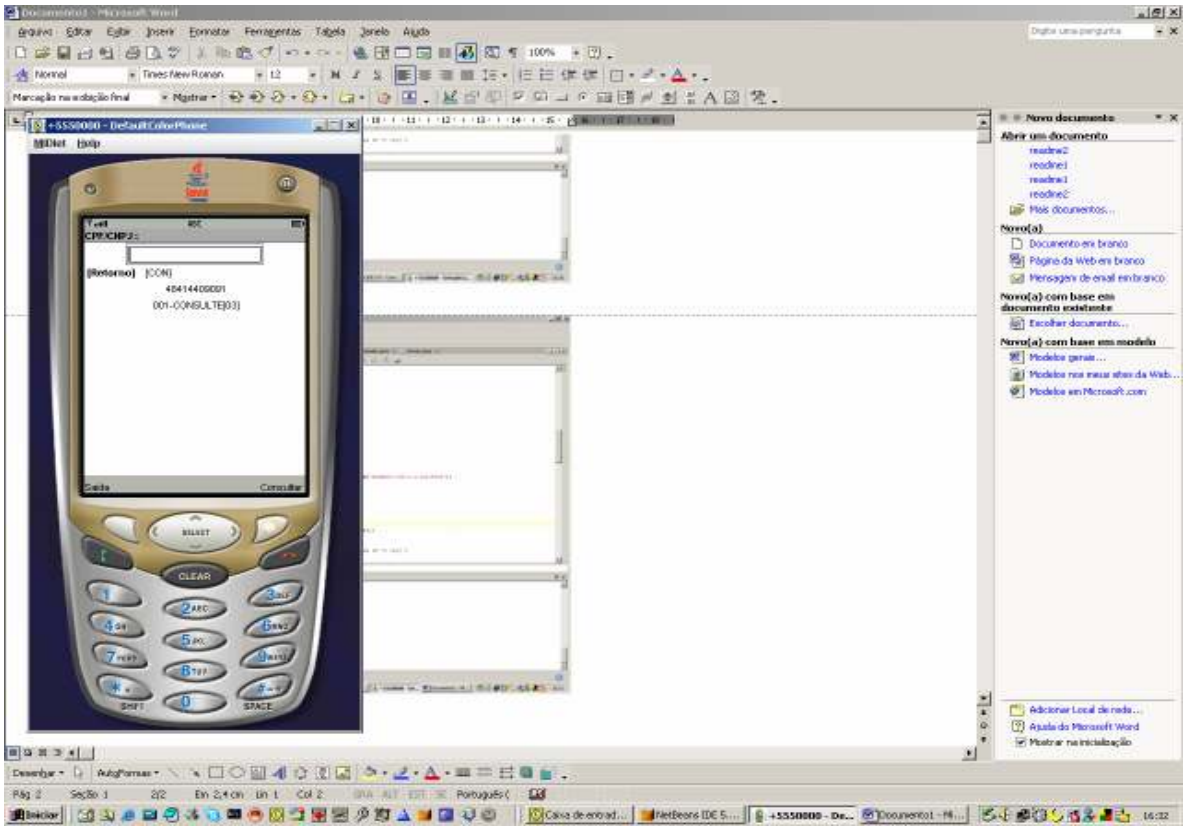

Figura 3.2 Tela de resultado da consulta

CONCLUSÃO

Neste trabalho foi apresentado o desenvolvimento de uma aplicação em JAVA que possibilita a comunicação de dispositivos móveis com outros equipamentos através do protocolo TCP/IP implementando assim uma nova funcionalidade para o Consult que é o sistema hoje existente. Uma solução como essa cria a possibilidade da execução de diversas tarefas, a qualquer hora e em qualquer lugar, sem termos a necessidade de acesso a um microcomputador com Internet.

Uma vez implementado o projeto, ficou clara a necessidade dos clientes da empresa CHIPCARD de ter acesso a mais esse recurso para diminuir a inadimplência em suas empresas e poder acompanhar o histórico de cheques emitidos, sem fundos e sustados de seus clientes.

A meta da empresa com esse projeto é disponibilizar essa aplicação a todos os seus clientes que hoje já utilizam o Consult (consulta de crédito) na versão desktop em suas empresas e também para novos clientes.

Muito ainda precisa ser feito em termos de melhorias no sistema, e deverão ser implementados nas próximas versões. Algumas melhorias são a da autenticação de segurança, criar rotinas de identificação do usuário e senha, aumentar os tipos de consulta disponíveis, implementar o resultado das consultas em XML, para trazer uma consulta gráfica com melhor aproveitamento de tela do dispositivo.

O objetivo original proposto - levar aos clientes a opção de efetuar consultas de crédito em qualquer lugar em que estiverem- foi alcançado com sucesso, pois hoje já se tem essa possibilidade e a aplicação já está sendo testada por alguns clientes. A facilidade de poder fazer suas consultas sem a necessidade de um microcomputador com Internet a sua disposição tem impressionado muito favoravelmente os clientes da CHIPCARD.

Hoje, os servidores de consulta de crédito da CHIPCARD atendem uma média de 25.000 consultas por mês, e com a implementação do sistema de consulta em dispositivos móveis, a expectativa é de um crescimento de mais de 50% nesse volume.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM. Emerson. **J2SE, J2EE e J2ME:** Uma breve explicação. Disponível em: <HTTP://www.infowester.com>. Acesso em: 10 jun 2006.

BRITO, Robison Cris. **Desenvolvimento de aplicativos para celular** – Java 2 micro edition. Centro Federal de Educação Tecnológica do Paraná, Unidade de Pato Branco.

DE PAULA, Gustavo Eliano. **O que podemos e o que não podemos fazer em Java para os celulares**, Revista Mundo Java, São Paulo,2006, número 16, ano III.

GAZINEU, Daniel. **O Java fala ao telefone**, Revista Mundo Java, São Paulo,2006, número 16 **–** ano III **.**

GOMES, Alexandre. **Tutorial J2ME Visão Geral.** Disponível em: <HTTP://www.mundooo.com.br.Acesso em 10 jun 2006.

OLIVEIRA, Elisamara de. **Byte-papo:** Mobile Applications. Disponível em:<HTTP://www.fiap.com.br>. Acesso em 10 jun.2006.

PEDROSO, Roger. **Estabelecendo conexões HTTP com J2ME**. Revista Mundo Java,São Paulo, 2006, Número 11, ano II.

MEDEIROS, Fábio. **Comunicação serial com J2ME Acessando a porta serial de PDAs usando J2ME**, Revista Web Móbile , Edição 10, 2006 – ano 2

SPÍNOLA, Rodrigo Oliveira. **UML na prática construindo Diagramas de Classes.** Revista SQL magazine, 2006, Edição 34, ano 3.