ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS MÓVEIS NA PLATAFORMA J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Bruna Georgina Bunzen de Albuquerque Romeiro
Orientador: Sérgio Castelo Branco Soares

Recife, Maio de 2005

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS MÓVEIS NA PLATAFORMA J2ME

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Bruna Georgina Bunzen de Albuquerque Romeiro
Orientador: Sérgio Castelo Branco Soares

Recife, Maio de 2005

UNIVERSIDADE
DE PERNAMBUCO

Bruna Georgina Bunzen de Albuquerque Romeiro

DESENVOLVIMENTO DE APLICATIVOS PARA DISPOSITIVOS MÓVEIS NA PLATAFORMA J2ME

Resumo

Este trabalho apresenta um estudo sobre dispositivos móveis e as plataformas J2ME e J2EE, explicando a implementação de um protótipo de *software* que auxiliará a obtenção de informações sobre docentes e discentes do NUPEC (Núcleo de Pesquisa em Engenharia da Computação) da Escola Politécnica da Univesidade de Pernambuco. O aplicativo cliente foi desenvolvido em J2ME e será responsável por fazer requisições para um aplicativo servidor. Tais requisições são consultas relativas a informações disponibilizadas em uma base de dados com informações sobre os projetos e pessoas do NUPEC. A aplicação servidora foi implementada utilizando a plataforma J2EE e será responsável por responder às requisições das aplicações cliente. Além de aplicar tecnologia no sistema implementado, este trabalho descreve tais tecnologias que utilizam a comunicação sem fio, permitindo assim avaliar o potencial das mesmas nesta área tão promissora.

Palavras chaves: Dispositivos móveis, J2ME, J2EE, Gerenciamento de informações.

Abstract

This work presents a study of mobile devices, J2ME and J2EE platforms, explaining the implementation of a software prototype that will allow information retrieval about professors and students of NUPEC (Computing Engineering Research Center) of the Polythechnic School of Engineering of the Pernambuco State University. The client application was developed using the J2ME platform and will send requests to a server application. Such requests are related to reports of information from a database that stores data related to NUPEC' projects and people. The server application was implemented using the J2EE platform and must response to the client applications requests. Besides applying technologies in the implemented software, this work presents such technologies that allow wireless communication, allowing to evaluate their potential in such promising area.

Keywords: Mobile devices, J2ME, J2EE, information management

Sumário

Índice de Figuras

Índice de Quadros

Tabela de Símbolos e Siglas

J2ME – *Java 2 Micro Edition*
NUPEC – Núcleo de Pesquisa de Engenharia da Computação
DSC – Departamento de Sistemas Computacionais
API – *Interface Programming Application*
J2SE – *Java 2 Standard Edition*
J2EE – *Java 2 Enterprise Edition*
PDA – *Personal Digital Assistant*
JVM – *Java Virtual Machine*
CDC – *Connected Device Configuration*
CLDC – *Connected Limited Device Configuration*
MIDP – *Mobile Information Device Profile*
KVM – *Kilo Virtual Machine*
EIS – *Enterprise Information System*
HTML – *Hiper Text Markup Language*
XML – *Extensible Markup Language*
JSP – *Java Server Pages*
HTTP – *Hypertext Transfer Protocol*
EJB – *Enterprise Java Beans*
MVC – *Model – View – Controller*
TCP – *Transmission Control Protocol*
UDP – *User Datagram Protocol*
GCF – *Generic Connection Framework*
FDMA – *Frequency Division Multiple Access*
TDMA – *Time Division Multiple Access*
CDMA – *Code Division Multiple Access*
GSM – *Global Standard Mobile*
SIM – *Subscriber Indentification Module*
PC – *Personal Computer*

Agradecimentos

A Deus, pela força concedida e pela oportunidade de estar concluindo mais uma etapa da minha vida.

À minha mãe, Ana Georgina Valença Bunzen, minhas tias Iracema Ferreira Bunzen (In memorian) e Georgiana Bunzen Gianelli e meus irmãos Fábio Bunzen Romão e Rômulo Costa Romão Júnior, pelo amor, força e apoio em todos os momentos.

Ao meu pai, Murilo de Albuquerque Romeiro por todo carinho e segurança que me foi passada.

Ao meu orientador, Prof. Sérgio Soares pela ajuda e dedicação para que esse trabalho fosse concluído com sucesso.

Aos professores de Engenharia da Computação da Universidade de Pernambuco, que sempre que solicitados se mostraram disponíveis a esclarecer dúvidas e questionamentos.

Aos Professores Ricardo Massa e Marcio Cornélio por terem aceitado o convite de compor a banca.

Ao colega Bruno Jamir, pela paciência e pelos conhecimentos compartilhados que foram de grande importância para o término deste trabalho.

Aos colegas, Lívia Brito e Leandro Marques pelo empenho em ajudar sempre que foram requisitados.

Às amigas do Club da Lulu, por cinco anos de amizade e companheirismo.

Aos demais colegas do curso de Engenharia da Computação, com destaque especial para Rodrigo Cursino, Fernando Antonio e Cláudio Cavalcante pelos anos de amizade que tornaram-se um grande aprendizado.

Capítulo 1

Introdução

Nos últimos anos vem ocorrendo um notável aumento na utilização de dispositivos móveis, que fazem uso da tecnologia de comunicação sem fio, em especial dos telefones celulares. Tais aparelhos apesar do poder computacional limitado, a cada momento possuem uma nova forma, tamanho, aumento na capacidade de processamento, além de novos aplicativos agregados.

À medida que a demanda por funcionalidades aumenta, as empresas de telefonia celular vêm acrescentando novas tecnologias a tais aparelhos, estimulando nos consumidores o desejo de possuir o mais recente modelo de uma determinada marca.

Devido a esse crescente mercado, cresce também a motivação no sentido de desenvolver novas aplicações para esses dispositivos. A presença da máquina virtual Java nesses equipamentos torna possível o desenvolvimento de aplicações utilizando a tecnologia *Java 2 Micro Edition* (J2ME), uma versão reduzida da linguagem Java utilizada para dispositivos móveis.

O desenvolvimento de aplicativos para celulares vem tornando-se cada vez mais freqüente nas empresas especializadas em produção de *software*. Contudo, esses sistemas não devem funcionar de maneira isolada. Entretanto, ignorar o fato de tais aparelhos serem inerentemente objetos de comunicação é subestimar o potencial deste artefato. Para tanto é necessário definir um protocolo de comunicação com sistemas corporativos, de forma que, possa ocorrer envio de informações entre as partes. Os celulares começaram a adquirir novas características, deixando de servir simplesmente para telefonia, ou seja, a mera transmissão de voz está perdendo o espaço para transmissão de dados.

O objetivo desse trabalho é relatar como funcionou o desenvolvimento de um aplicativo para dispositivos móveis, com o propósito de ser utilizado por discentes e docentes do NUPEC (Núcleo de Pesquisa em Engenharia da Computação) da Escola Politécnica de Pernambuco, unidade da Universidade de Pernambuco. O NUPEC é um laboratório onde os professores do Departamento de Sistemas Computacionais (DSC) orientam seus alunos de iniciação científica.

A intenção de desenvolver um sistema para celular surgiu quando alguns usuários do NUPEC se depararam com situações em que era necessário obter informações tanto de professores, quanto de alunos ou das atividades relacionadas com as pesquisas realizadas no núcleo, mesmo quando não estavam presentes fisicamente no mesmo.

A aplicação proverá aos usuários o serviço de consulta das informações disponibilizadas para os freqüentadores do núcleo. Um usuário que possuir o aplicativo instalado em seu celular pode, por exemplo, saber as pessoas que estão em determinado momento no núcleo ou as que passaram por lá em um determinado dia, além de outras consultas.

O sistema está dividido em um módulo cliente e um módulo servidor. O cliente é o aplicativo que estará rodando no celular e terá como uma de suas responsabilidades, receber a entrada de dados informados pelo usuário. Uma comunicação com o servidor deve ser estabelecida para que os dados sejam transmitidos. Ao receber os dados, o servidor deve processá-los e então encaminhá-los de volta como resposta à requisição enviada pelo cliente.

O Capítulo 2 explica as técnicas que foram utilizadas no desenvolvimento tanto do lado cliente quando do lado servidor.

Um histórico da comunicação móvel, bem como as vantagens e desvantagens dos dispositivos móveis é apresentado no Capítulo 3.

O Capítulo 4 mostra como foi implementado tanto o módulo cliente quando o módulo servidor com seus respectivos diagramas de classes. Por fim, o Capítulo 5 apresenta conclusões que podem ser tiradas com o presente trabalho, bem como sugestões de trabalhos futuros.

Capítulo 2

Tecnologias

2.1 Java

Java é uma linguagem de programação de alto nível, que segue o paradigma de programação orientada a objeto. Devido sua portabilidade, Java pode ser executada em diferentes ambientes operacionais dentro do conceito "escreva uma vez, execute em qualquer lugar" (*Write Once, Run Anywhere - WORA*). A linguagem também possui a presença de mecanismos de tratamento de exceções que tornam as aplicações mais robustas, não permitindo que elas falhem mesmo quando estão rodando sob condições anormais. [4]. As bibliotecas que fazem parte de Java definem interfaces portáveis.

Java também possui um coletor de lixo (*Garbage Collector – CG*). A coleta de lixo é uma eficiente técnica de liberação de memória utilizada pela linguagem Java [28]. Muitas linguagens permitem que o programador aloque um espaço de memória em tempo de execução utilizando ponteiros e pertencendo ao programador o dever de liberar a memória quando esta não estiver mais sendo utilizada. Esta, geralmente, é uma tarefa complexa e propensa a erros, uma vez que deixa a cargo do programador o gerenciamento de memória da aplicação [28]. O sistema de coleta de lixo de Java tira essa responsabilidade do programador, passando essa responsabilidade para o coletor de lixo que verifica quais os ponteiros de memória que não têm mais referências apontando para eles e, então, libera a memória.

A linguagem Java tem a intenção de ser usada em ambientes de rede/distribuídos. Nessa direção, grande ênfase tem sido colocada na segurança. O Java permite a construção de sistemas livres de vírus e adulterações [13].

Com essas qualidades, Java pode ser utilizada para a criação de vários tipos de aplicativos, desde aplicações *standalone* (local) até aplicações designadas para serem controladas pelo software que as executa, tais como APPLETS[1] [28], SERVLETS[2] [30] ou MIDLETS[3] [18] .

---

[1] Programas escritos em Java que podem ser inseridos em documentos de hipertextos – HTML, carregados na web e executados em um brouwser.

[2] Aplicações executadas em um servidor web

[3] Aplicações executadas em dispositivos móveis

A API (*Applications Programming Interface*) de Java consiste em um conjunto de bibliotecas de tempo de execução que fornecem ao desenvolvedor de software uma forma padrão de acessar os recursos do sistema [31]. A especificação da API Java assim como da máquina virtual, devem ser implementados para cada plataforma que é para garantir a independência das plataformas.

A máquina virtual Java é um computador hipotético, implementado como uma aplicação de software em uma máquina real [28]. A portabilidade e a segurança são características da plataforma Java que são possibilitadas pela existência da máquina virtual.

Devido ao desenvolvimento da linguagem Java e aos diferentes contextos nos quais ela está sendo aplicada, encontrou-se a necessidade de dividi-las em 3 edições, são elas:

- *Java 2 Standard Edition* (Java 2 Edição Padrão, J2SE) : possui um conjunto de ferramentas para o desenvolvimento de aplicações desktop;
- *Java 2 Enterprise Edition* (Java 2 Edição Corporativa, J2EE): é um super conjunto da J2SE, voltado para aplicações corporativas e distribuídas, voltada para servidor e utilizando EJB (*Entreprise JavaBeans*);
- *Java 2 Micro Edition* (Java 2 Edição Micro, J2ME): voltada para o desenvolvimento de aplicativos para dispositivos móveis. J2ME é um subconjunto da J2SE.
- *Java Card:* é uma solução da união *smart card* com algumas funcionalidades de segurança de redes. É uma plataforma que garante a privacidade e segurança de informações.

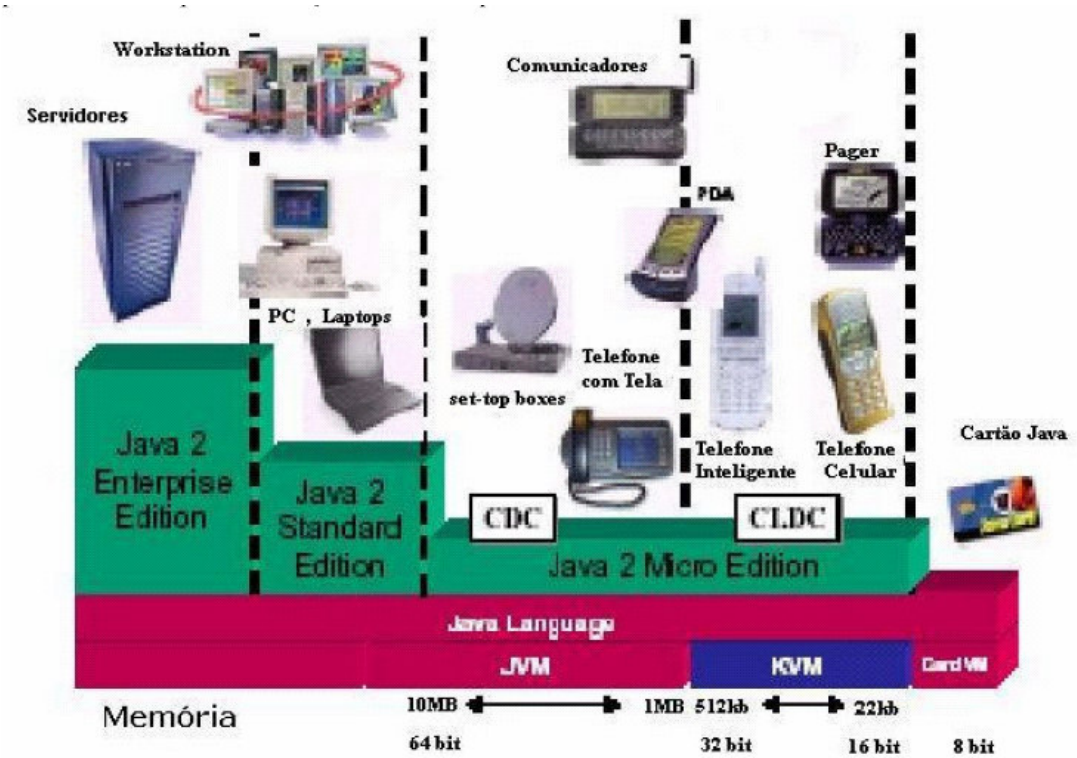

Figura 1. Plataforma Java 2 – Fonte [29]

Seguindo o foco deste trabalho, ocorrerá uma breve explicação das características e funcionalidades da plataforma J2ME.

2.2 Plataforma Java 2 Micro Edition (J2ME)

Java 2 *Micro Edition* (J2ME) [17] é a edição da linguagem Java para ser usada em dispositivos de computação portáteis e móveis, que possuem as seguintes características: mobilidade, baixa capacidade de processamento e pouca memória disponível, alimentação elétrica por baterias, pequenas áreas de *display*, e limitados e variados métodos de entrada e saída. Alguns exemplos destes dispositivos seriam os telefones celulares, *pagers*, PDAs (Assistentes Digitais Pessoais), *Palms*, entre outros [11].

J2ME não é um novo tipo de linguagem Java, ela foi apenas adaptada para poder executar os programas nos dispositivos acima citados. Sendo assim um programa na plataforma J2ME funciona corretamente nas outras duas plataformas, visto que a API utilizada na Micro Edição é um subconjunto das Edições Padrão e Corporativa.

A subdivisão da plataforma Java em especial destacando a plataforma J2ME pode ser observada na Figura 2.

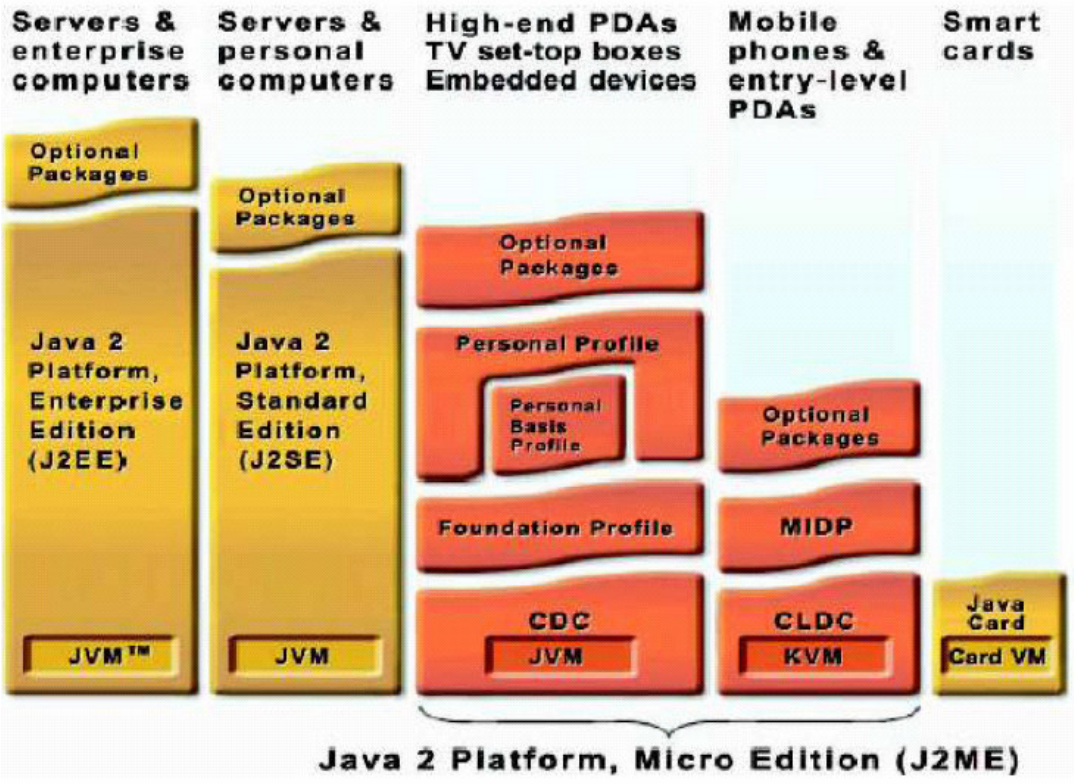

Figura 2. Divisão da Plataforma Java – Fonte [23]

Através de J2ME, torna-se possível desenvolver, atualizar e instalar novas aplicações segundo as necessidades particulares de cada usuário [17]. As aplicações que utilizam J2ME vão desde jogos até aplicações que acessam banco de dados, dentre outras opções.

Existe uma diversidade de dispositivos, no qual estes apresentam muitos pontos em comum, mas diferenciam-se em suas formas, funcionalidade e características. Existem também dispositivos com uma grande variedade de capacidade de processamento, memória e interação com o mundo exterior (interface com o usuário, métodos de entrada e saída de informações e dados). Afirmar que uma tecnologia serve para todos os dispositivos da mesma "família", tal como nas outras edições de Java, não funciona nos dispositivos de recursos limitados devido a todas essas diferenças [11].

Devido à diferença de tamanho e funcionalidade nesses dispositivos foram estabelecidos conceitos essenciais e de grande importância na plataforma J2ME que foi dividida em camadas para facilitar seu entendimento. São elas: Perfis, Configurações, JVM e o Sistema Operacional, como pode ser observado na Figura 3.

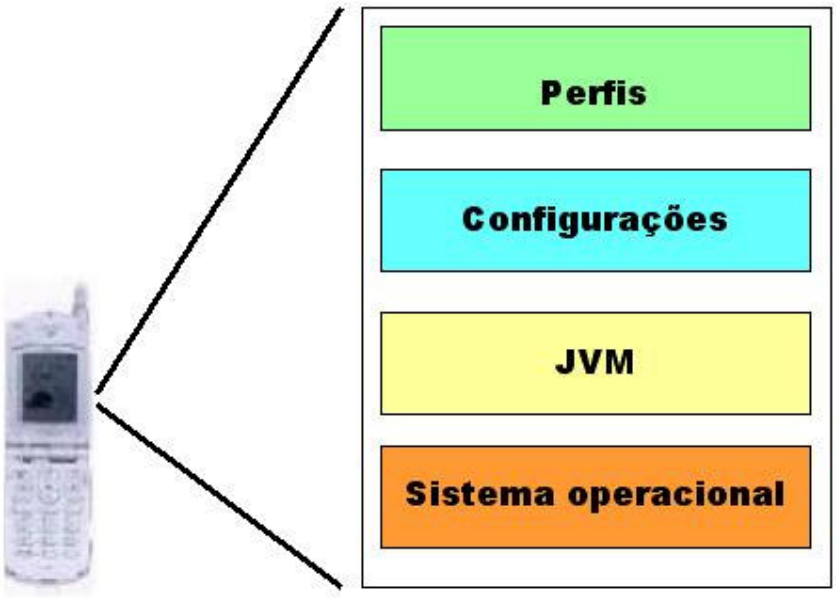

Figura 3. Arquitetura J2ME

2.2.1 Perfis

O perfil é a camada mais visível para usuários e desenvolvedores das aplicações, definindo um conjunto mínimo da API disponível em uma classe particular dos dispositivos, representando um segmento de mercado particular. Os perfis são implementados para um tipo de configuração particular variando de acordo com os dispositivos para o qual ele será utilizado.

As aplicações são feitas de acordo com um perfil específico, podendo ser instalada em qualquer dispositivo que possua suporte ao perfil determinado. O perfil que este trabalho terá foco é o MIDP [23] – *Mobile Information Device Profile* (Perfil do Dispositivo de Informação Móvel). Este perfil é para aplicações móveis e ajuda a complementar a configuração CLDC e será detalhado adiante.

O Perfil do Dispositivo de Informações Móveis (MIDP) é o primeiro perfil disponível para a plataforma J2ME. A combinação do CLDC e do MIDP fornece um ambiente completo de desenvolvimento para a criação de aplicações em celulares e *pagers* [25]. As aplicações que conseguem executar em dispositivos que possuem suporte a MIDP são chamados de MIDlets.

Para um dispositivo ter suporte ao MIDP, ele deve possuir um display de pelo menos 96 x 54 *pixels* e uma memória de pelo menos 128 Kbytes.

2.2.2 Configurações

Uma configuração define uma plataforma mínima para um grupo de dispositivos com características similares, tanto na memória quanto no poder de processamento. Sendo assim, uma configuração define as características suportadas tanto pela própria linguagem de programação Java quanto pela máquina virtual e também suas bibliotecas de classes e APIs, as quais um determinado fabricante pode esperar que estarão disponíveis em todos os dispositivos de uma mesma categoria [17].

Apenas duas configurações são aceitas pela *Sun* e serão descritas a seguir:

- CDC – *Connected Device Configuration* (Configuração para dispositivos conectados) – rege as configurações para aparelhos móveis um pouco maiores, com um mínimo de 2 *megabytes* de memória disponíveis. Alguns exemplos desses dispositivos são: televisão com internet, sistema de navegação de carros, entre outros;
- CLCD - *Connected, Limited Device Configuration* (Configuração para dispositivos com limite de conexão) – é responsável pela configuração de aparelhos pequenos e com algumas restrição de recursos como celulares e *pagers.* Utiliza uma máquina virtual chamada de KVM (*Kilo Virtual Machine*). Esta é bem reduzida em relação a máquina virtual Java tradicional e faz uso de um conjunto de bibliotecas de classes para ser utilizada dentro de um Perfil MIDP;

2.2.3 Máquina Virtual J2ME

A máquina virtual é um dos principais fundamentos da tecnologia Java, permitindo que as aplicações escritas possam ser portáveis para diversos tipos de hardware e diferentes sistemas operacionais.

Em J2ME deve-se utilizar uma máquina virtual Java que seja apropriada para os dispositivos como telefones celulares, *pagers* e PDAs que possuem tamanho de memória reduzido [17].

A máquina virtual que o aplicativo do trabalho em questão faz referência é a KVM (*Kilo Virtual Machine*), utilizada pela configuração CLDC, possuindo cerca de 80K de memória. A KVM pode executar em qualquer sistema que possuía um processador de 16 a 32 bits e um total de memória de 160 a 512K, devido a essas restrições não possui suporte a tipos de dados longos e pontos flutuantes. Seu projeto foi baseado em algumas importantes considerações, incluindo o tamanho reduzido para conservar um melhor espaço em memória quanto possível (tanto em termos de armazenamento quanto execução) e a capacidade de rodar em processadores de pequeno poder computacional [17].

2.3 Java 2 Enterprise Edition (J2EE)

A arquitetura J2EE foi desenvolvida sobre a plataforma J2SE (*Java 2, Standard Edition*) utilizando assim as APIs básicas para o desenvolvimento de programas e aplicações. Logo, é baseada na linguagem Java e segue a mesma lógica de escrever apenas uma vez e conseguir rodar em qualquer plataforma.

J2EE faz uso do modelo de aplicação multicamadas onde cada aplicação lógica é dividida em componentes de acordo com sua funcionalidade e vários componentes das aplicações podem ser instalados em diferentes máquinas dependendo da camada em que ele se encontra.

As camadas podem ser divididas da seguinte forma:

- A camada cliente que roda em uma máquina cliente;
- A camada de componentes WEB, aplicações J2EE que rodam no servidor.;
- A camada de informações (EIS – *Enterprise Information System*) que também roda no servidor;

Essas camadas podem ser melhor observadas na Figura 4.

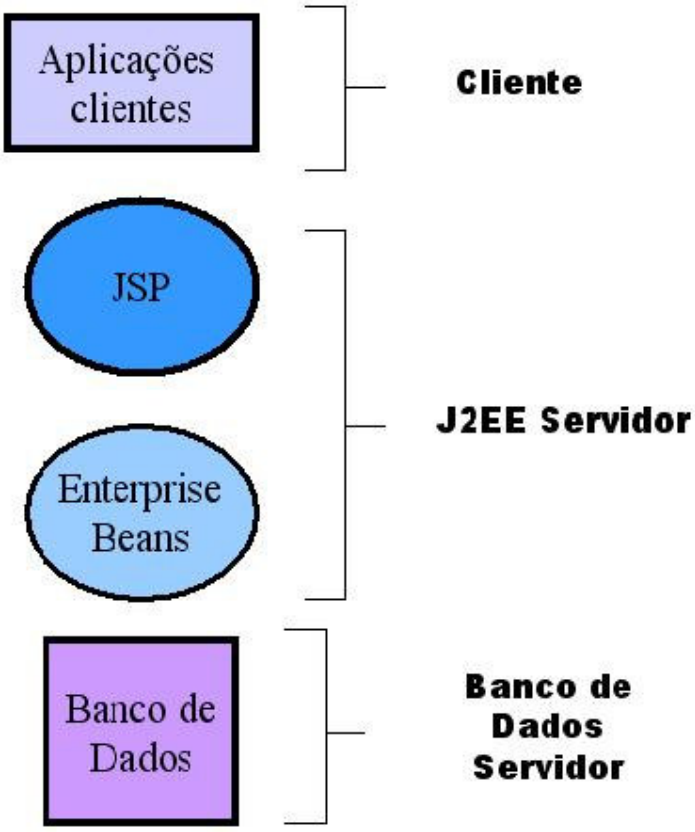

Figura 4. Camadas da Arquitetura J2EE

O lado cliente pode ser dividido em duas partes:

- Páginas web dinâmicas como, por exemplo, HTML (*Hiper Text Markup Language*) ou XML (*Extensible Markup Language*) [33] que são geradas pelos componentes web presentes na camada web.
- *Web browser* que são os responsáveis por redirecionar as páginas a medida que são recebidas pelo servidor.

O servidor Web é quem executa as tecnologias *servlets* e JSP (*Java Server Pages*) vindas de clientes externos. Os *servlets* são programas Java desenvolvidos no lado servidor e que trata das requisições HTTP feitas pelos clientes, podendo responder essas requisições dinamicamente ou simplesmente repassar essas requisições para outros *servlets* ou outros servidores.

Um arquivo JSP possui tanto código Java como HTML e logo após a primeira execução é então transformado em um *servlet,* esse servlet é compilado para que possa ser executado. Quando um protocolo HTTP recebe uma requisição de um JSP, ele repassa essa requisição para o *container* [2], uma interface entre o servidor e um componente, neste caso o JSP, e este então chama o respectivo *servlet* compilado e devolve o HTML para o servidor HTTP.

O *container* EJB é uma interface entre o servidor EJB e os componentes de negócio ou componentes EJB, ou ainda *beans*. Os *beans* são os objetos da especificação EJB e executam o serviço funcional, como transferência bancária, compra de produtos pela Web, inscrição para um congresso, etc., e o servidor EJB executa o serviço não–funcional que é configurado pelo desenvolvedor da aplicação EJB [24].

2.4 Struts

O *Struts* [14] é um *framework open source* da *Jakarta* [16]. Esse *framework* utiliza as mesmas tecnologias de aplicações Java, porém organiza-se de maneira diferente. A idéia principal do Struts é separar os códigos responsáveis pelo processamento, daqueles códigos voltados para a apresentação de dados. Essa separação pode ser feita sem a utilização desse *framework*, porém fica sob responsabilidade do desenvolvedor. O *Struts* implementa a arquitetura *Model – View – Controller* (MVC – Modelo – Visualização - Controle) [10].

A arquitetura MVC [10] define a total separação do Modelo (objetos pertencentes à camada de negócios), da Visualização (interface com o usuário ou com outro sistema qualquer) e do Controlador, que controla o fluxo do sistema. É de responsabilidade do controlador interceptar as requisições HTTP vindas do cliente.

O *Struts* possui a classe Java denominada `Action`, onde está a lógica para obter cada requisição. Um objeto `Action` é o responsável pelo processamento dos dados que são enviados através de uma determinada requisição. Para escrever um `Action` é necessário herdar da classe `org.apache.struts.action.Action` e deve-se sobrescrever o método `execute`. A Figura 5 mostra um trecho de código do sistema em questão onde uma classe `Action` é criada e o método `execute` é sobre-escrito.

```
import org.apache.struts.action.Action;

public class ConsultaAction extends Action {


    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {

        //objeto que contém a chave para o redirecinamento
        ActionForward result = null;
```

Figura 5. ConsultaAction extendendo a classe Action

A classe `ActionServlet` é o controle do *Struts* e deve ser configurada para interceptar todas as requisições e então direcioná-las para os `Actions` correspondentes. Um trecho de código referente a configuração do `ActionServlet` no `web.xml` pode ser visto na Figura 6.

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
        <param-name>config</param-name>
        <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
```

Figura 6. ActionSrvlet no web.xml

Para que o `ActionServlet` consiga fazer o direcionamento de requisições corretamente, é necessário configurar os `Actions` existentes em um arquivo de configuração chamado `struts-config.xml` e é este arquivo que deve conter todas as informações necessárias para o funcionamento desse *framework.*

2.5 Wireless

Redes sem fio [8] são uma nova alternativa às redes convencionais com fio, fornecendo as mesmas funcionalidades, porém de forma flexível pois permite uma comunicação entre diversos pontos sem a necessidade de utilizar cabos (wire = fio, less = sem). Essa tecnologia faz uso de um sistema de antenas interligado entre si que transmite as informações via onda de rádio ou infravermelhas, disponibilizando a portabilidade e a praticidade da informação independente do lugar, perdendo assim a dependência de objetos fixos.

As redes sem fio vêm sendo amplamente utilizadas, pois permitem uma comunicação de alta velocidade a baixo custo. O custo semelhante ao de uma conexão discada, porém enquanto a velocidade via *modens* chega a no máximo de 56Kbits, o acesso via rádio possui uma velocidade mínima de 64Kbits.

2.6 Arquitetura Cliente/Servidor

O trabalho tema desta monografia foi desenvolvido em um ambiente distribuído, onde um cliente se comunica com um servidor depois de estabelecida uma conexão. Ou seja, uma típica relação cliente/servidor onde a conexão é feita entre a plataforma *Java 2 Plataform, Enterprise Edition* (módulo servidor) e a *Java 2 Plataform, Micro Edition* (módulo cliente).

No servidor, aplicações J2EE podem ser desenvolvidas e implantadas utilizando qualquer uma das várias aplicações servidoras disponíveis nesta plataforma. Este trabalho faz uso do *framework* Struts, desenvolvido pela *Jakarta*, como explicado na Seção 2.5.

No cliente, aplicações são desenvolvidas e essas aplicações podem executar em qualquer dispositivo com suporte a MIDP como, por exemplo, aparelhos celulares, *pagers* ou assistentes pessoais (PDA). Dessa forma, é possível abranger diversos tipos de clientes tornando as aplicações mais acessíveis.

Muitas aplicações só conseguem o resultado desejado após estabelecer uma conexão com o ambiente servidor. Esta conexão pode permanecer ativa durante todo ciclo de vida da aplicação ou ser ativada apenas quando for necessária uma comunicação com o servidor.

Para obter uma solução na tecnologia J2ME, é importante ser observado que tanto a conexão de rede quanto os recursos são bastante limitados, o que não reflete um desenvolvimento típico em um computador que possui um ambiente fixo de rede. Em um serviço móvel nem sempre é possível ter uma conexão ativa de rede, variando de acordo com a localização geográfica.

O HTTP (*Hypertext Transfer Protocol*) é o único protocolo de transporte que é suportado pela plataforma J2ME, conseqüentemente o protocolo HTTP provê a comunicação entre a aplicação MIDP e o servidor J2EE.

2.7 HTTP (Hypertext Transfer Protocol)

Um dos mais críticos aspectos de J2ME é a questão de obter conexão com uma rede de computadores. Embora alguns serviços J2ME possam ser utilizados sem ser necessário estar conectado a uma rede, a habilidade de fazer conexão com uma rede de computadores provê à plataforma J2ME acesso aos recursos disponíveis nesta rede.

Muitos serviços J2ME suportam essa capacidade, como, por exemplo, abrir uma porta para mandar e receber *emails,* exatamente como ocorre em um computador padrão, só que dentro do espaço móvel. Essa plataforma tornou-se um cliente capaz de interagir com sistemas padrões, bancos de dados, Intranet corporativas e a Internet [34].

O HTTP [36] é um protocolo de requisição e respostas, ou seja, o MIDP cliente manda um HTTP *request* para o servidor J2EE que analisa a requisição e retorna um HTTP *response*, como pode ser visto na Figura 7. Esse protocolo não mantém ligação permanente entre o cliente e o servidor, ou seja, depois de enviar a resposta para o cliente, o servidor toma a iniciativa de fechar a conexão.

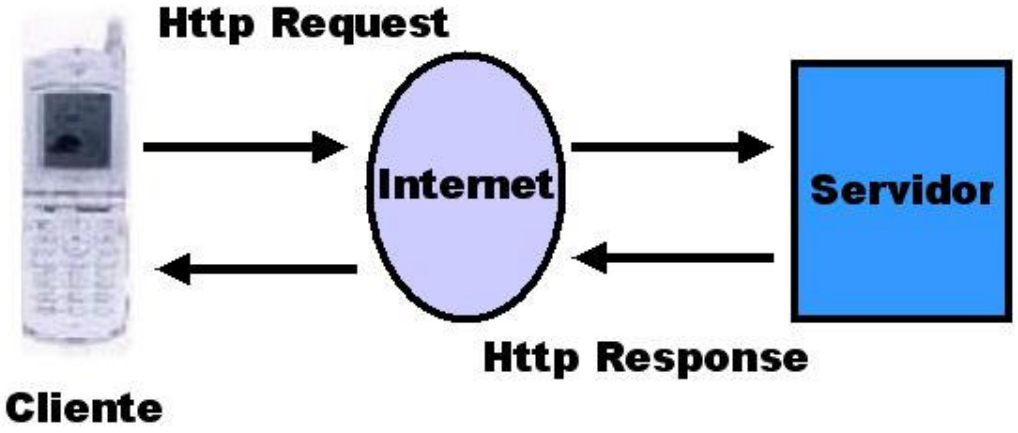

Figura 7. Protocolo HTTP

O HTTP é um protocolo de comunicação ideal para aplicações móveis utilizando Java, visto que o MIDP 1.0 suporta o HTTP, já outros protocolos como o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*) são opcionais no MIDP 1.0, uma vez que não são todos os serviços do MIDP que suportam *socket* ou o UDP.

Os principais métodos que existe no protocolo HTTP são:

- GET: recupera as informações identificadas no recurso da rede. Se o recurso for um processo executável, ele retornará a resposta do processo e não seu texto, Também existe o GET Condicional que só trará a informação se ela foi modificada depois da data da última transferência;
- HEAD: variante do método GET, porém não possui a transferência de dados da entidade para o cliente. Solicita ao servidor apenas o cabeçalho com informações sobre o recurso, normalmente utilizadas para depuração;
- POST: permite que o cliente envie mensagens e conteúdos de formulário para o servidor que vai manipular os dados da maneira desejada. Através desse método, pode-se, por exemplo, postar uma mensagem em uma lista de discussão;
- PUT: esse método permite que um cliente, que possua autoridade, atualize ou armazene dados em um recurso, se o recurso já existir. Caso não exista, permite que o recurso seja criado;
- DELETE: um cliente autorizado solicita ao servidor a remoção dos dados identificado na URI;
- LINK: estabelece uma ligação entre páginas.

Um exemplo do formato de mensagem do protocolo HTTP pode ser visto na Figura 8.

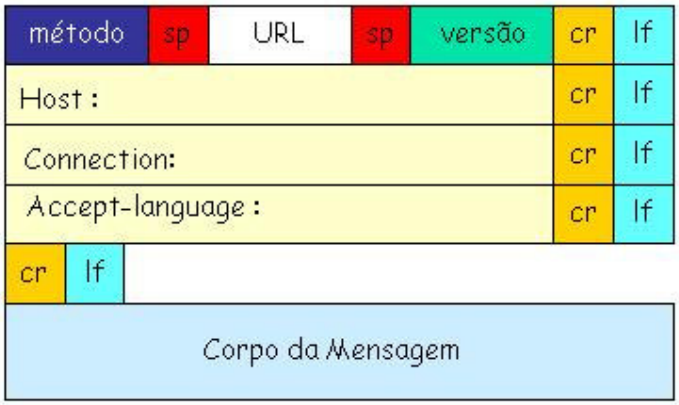

Figura 8. Formato de Requisição de Mensagem

Na primeira linha da solicitação contém o método, seguido do local de onde o recurso foi solicitado indicado pela URL e a versão do HTTP que está sendo utilizado. Dos métodos disponíveis, o GET e o POST são os mais utilizados.

O MIDP possui um suporte padrão para HTTP 1.1, APIs para geração de requisições de HTTP GET, POST e HEAD, manipulação básica de cabeçalho e geração de mensagens.

O subconjunto da arquitetura J2ME responsável por conseguir estabelecer uma conexão com uma rede desejada é chamado *Generic Connection Framework* (GCF).

2.7.1 Generic Connection Framework (GCF)

O *Generic Connection Framework* fica localizado no pacote javax.microedition.io e possui uma classe (`Connector`), um exceção (`ConnectionNotFoundException`) e oito interfaces, como pode ser vista na Figura 9.

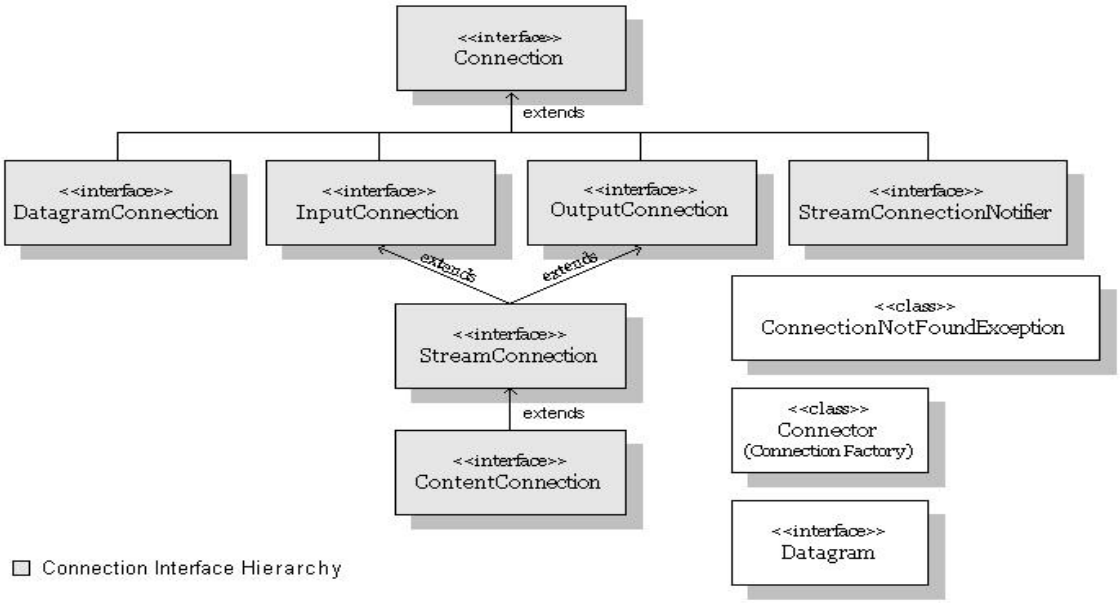

Figura 9. Hierarquia das interfaces no GCF e Classes relacionadas – Fonte [35]

No início da hierarquia está a interface *Connection,* é o tipo mais básico de conexão, e todas as outras interfaces herdam dela. À medida que se percorre para baixo nos níveis da hierarquia, as conexões vão tornando-se mais complexas.

A classe `Connector` é utilizada para gerar instâncias de conexão utilizando os métodos que possui. A instância gerada é suportada pela interface *Connection* ou uma de suas descendentes no nível da hierarquia. Essa classe possui três métodos `open`, com as seguintes assinaturas:

- `open(String url)`
- `open(String url, int mode)`
- `open(String url, int mode, boolean timeouts)`

Os parâmetros passados por estes métodos significam:

- A `url` é a URL da conexão;
- O `mode` é a método de acesso;
- O `timeout` é uma forma de indicar que uma exceção será levantada ou não, caso ocorra um estouro no tempo

Esse trabalho fez uso da interface `HttpConnection` que herda da `ContentConnection`, como pode ser visto na Figura 10 .

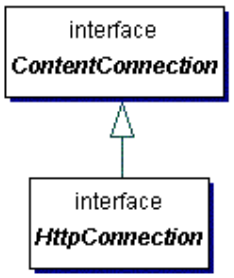

Figura 10. Relação entre HttpConnection e ContentConnection

Após obtida uma instância de `HttpConnection`, para abrir a conexão com entre o cliente e o servidor foi utilizado o método `open`, da classe `Connector`, sendo passada apenas a URL como parâmetro, podendo ser observado no Quadro 1.

```
String url = "http://localhost:8080/tcc/consulta.do";
HttpConnection conn = null;
conn = (HttpConnection) Connector.open(this.url);
conn.setRequestMethod(HttpConnection.POST);
```

Quadro 1. Obtendo a conexão com o HttpConnection

Capítulo 3

Comunicação Móvel

3.1 Histórico da Comunicação Móvel

Um sistema de comunicação móvel tem como característica a possibilidade de movimento relativo entre as partes como, por exemplo, a comunicação entre o telefone celular e a estação base na telefonia celular. Sistemas móveis usam a tecnologia sem fio para possibilitar uma comunicação transparente enquanto o usuário se desloca [7].

O desejo da humanidade em comunicar-se livre de fios ocorre desde os primórdios da civilização. Na Grécia antiga o uso de sinais de fumaça é mencionado como forma de comunicação. No final do século XVIII, Claude Chape inventa a telegrafia óptica (1794), possibilitando a comunicação sem fio para longas distâncias. Em 1820, Hans Christian Oersted descobre experimentalmente que a corrente elétrica produz um campo magnético [7]. Em 1864, James C. Maxwell lança os fundamentos teóricos sobre campos magnéticos com suas famosas equações. Em 1876, Alexander Graham Bell inventa o telefone [7].

A comunicação móvel teve alguns marcos importantes dentre eles podem ser destacados:

- O aparecimento dos primeiros sistemas públicos de telefonia móvel dos Estados Unidos, até então com poucos usuários, sendo utilizado apenas por alguns negociantes e por policiais que possuíam rádios receptores e transmissores nos veículos. [6]
- Surgimento de pesquisas realizadas por algumas telefônicas do mundo todo, com o intuito de resolver as limitações existentes na telefonia móvel. [6]
- Na década de 80, unidades móveis começaram a ser instaladas em veículos, dessa forma a comunicação era veicular e não pessoal. Nesta mesma década foi projetada e construída pela Motorola a rede ARDIS (*Advanced Radio Data Information Service*), primeira grande rede de dados sem fio dos Estados Unidos, especializada para clientes da IBM. [6]
- A era da telefonia celular teve seu início efetivo no início dos anos 90, quando o usuário podia portar o aparelho embora suas dimensões iniciais fossem grandes [7].

Foram adotados vários padrões em diferentes países para a telefonia celular e ficaram conhecidos com a Primeira Geração (1G). Os sistemas de 1G utilizam a transmissão de dados de modo analógicos e a técnica de utilizada de acesso ao meio é a FDMA (*Frequency Division*

Multiple Access – Acesso Múltiplo por Divisão de Frequência) [6], que possuía uma baixa qualidade e uma incompatibilidade com os diversos sistemas existentes.

Depois da 1G ocorreu o surgimento da Segunda Geração (2G), com o objetivo de aumentar a capacidade da primeira geração. A 2G utilizava o TDMA (*Time Division Multiple Access* – Acesso múltiplo por divisão de tempo) [6] e o CDMA (*Code Division Multiple Access* – Acesso múltiplo por divisão de código). [6]

Atualmente as redes celulares digitais possuem o serviço GSM (*Global Standard Mobile*) [6], que foi criado para prover serviços celulares modernos. Uma inovação encontrada no sistema GSM é a utilização do SIM (*Subscriber Indentification Module*) que contém algumas identificações do usuário assim como chave de código de privacidade. O SIM é conectado a um terminal GSM, pode ser removido de um aparelho e conectado em outro; sem o SIM o terminal fica impossibilitado de operar.

3.2 Dispositivos Móveis

Muito mais do que assistentes pessoais ou agendas eletrônicas, os dispositivos móveis passaram a ser computadores que podem ser facilmente levados a qualquer lugar, criados para atender profissionais e pessoas em movimento que necessitam de rapidez, facilidade e segurança no acesso a informações corporativas e pessoais. Além disso, as grandes inovações trazidas pela tecnologia *wireless* fizeram com que a indústria deste setor tenha tido um crescimento explosivo nos últimos anos, tornando-se uma das mais eficientes e rápidas áreas tecnológicas do mundo, permitindo que as pessoas comuniquem-se de forma barata e fácil sem ficarem presas aos seus telefones ou computadores de mesa [19].

Quando são mencionados dispositivos móveis (celulares e PDA), é feito uma referência a aparelhos que são encontrados facilmente no cotidiano das pessoas e que estão se tornando cada vez mais eficazes quando se fala de comunicação e, de preferência, *on-line*. Esses equipamentos permitem que os usuários se desloquem junto com seu ambiente computacional e tenham um acesso constante às fontes de informações.

Figura 11. Dispositivos Móveis – Fonte: [29]

Em se tratando especificamente do aparelho celular, há proximadamente uma década foi um artigo de luxo, hoje em dia tornou-se um bem de consumo bastante acessível. À medida que a demanda por funcionalidades aumenta, as empresas de telefonia celular vêm acrescentando novas tecnologias a tais aparelhos, variando seus tamanhos, fazendo modificações desde *hardware* até *software*, estimulando nos consumidores o desejo de possuir o mais recente modelo de um determinado celular.

Os celulares ainda possuem como principal função o serviço de voz, porém a transmissão de dados vem tornando cada vez mais freqüente. Como exemplo pode-se citar a possibilidade de ter acesso a informação e serviços a qualquer hora acessando a Internet. Pode-se dizer que os celulares estão tornando-se pequenos computadores, possuindo a vantagem do tamanho reduzido.

O número total de telefones celulares excedeu 600 milhões no ano de 2001 e a estimativa era de haver 1 bilhão em 2003 [27]. Em contraste, os PC ficaram em torno de 311 milhões no início do ano 2000 [27].

3.2.1 Vantagens dos Dispositivos Móveis

Do ponto de vista empresarial, os dispositivos móveis são ótimos geradores de informação, podendo ser utilizado na automatização do processo, até nas coletas de informações estratégicas, pois com suas reduzidas dimensões podem ser transportados e estar presentes em todas as situações em que um profissional pode atuar [29].

Algumas vantagens dos dispositivos móveis em relação aos micro-computadores são listadas a seguir:

- Tamanho: bastante reduzidos e muito mais leves do que os PCs, podendo ser transportados de forma muito mais prática;
- Fácil manuseio: os dispositivos móveis possuem uma interface gráfica simples de manusear se comparado aos computadores;
- Consumo de energia: por serem menores e mais econômicos gastam menos energia que os computadores visto que o tempo de recarga é menor;
- Custos operacionais: como os dispositivos móveis são mais compactos e possuem atividades específicas, estes aparelhos não possuem alguns periféricos internos, como discos rígidos e discos flexíveis, diminuindo consideravelmente os custos com a manutenção;

Outra característica que ajuda no desenvolvimento da comunicação sem fio é o fato de que as pessoas estão cada vez mais dependentes das informações disponibilizadas na Internet, o que antes poderia ser feito apenas via terminal remoto, agora pode ser acessado via dispositivo móvel. A tecnologia sem fio disponibiliza ao usuário a possibilidade de obter informações que lhes sejam úteis, a qualquer momento ou qualquer lugar.

A mobilidade é outra característica que deve ser levada em consideração. A capacidade de poder continuar uma comunicação e manter o envio de dados constante mesmo quando em movimento pode ser considerada uma das melhores vantagens de um dispositivo móvel.

3.2.2 Desafio dos Dispositivos Móveis

Toda mudança requer adaptação, com as redes sem fio não foi diferente. Pelo crescente mercado de dispositivos móveis foi necessária uma grande adaptação das tecnologias já desenvolvidas para os computadores remotos, para que estas também estejam disponíveis para os dispositivos móveis. Essa adaptação não é trivial.

Um fato de grande importância a ser observado é relacionado à forma como os celulares e dispositivos portáteis são utilizados. Além de algumas limitações como tela e bateria, esses dispositivos não são usados da mesma forma que um computador, mesmo que consiga obter o mesmo desempenho tecnológico. Um aparelho celular é utilizado em situações específicas como, por exemplo, no trânsito. Esse fato deve ser levado em consideração para não ficar limitado no momento dos produtos serem construídos.

A tecnologia sem fio precisa trabalhar dentro de restrições dos dispositivos, alguns problemas podem ser destacados:

- Recurso de memória limitado: dispositivos como celulares e PDAs (*Personal Digital Assistant* – Assistente Pessoais Digitais) possuem pouca memória, exigindo assim que o gerenciamento de memória seja fundamental;
- Baixa capacidade: o processamento de um dispositivo móvel pode variar de 32Kbytes a 64Mbytes, o que faz com que este seja um dos principais desafios enfrentados pelas operadoras de telefonia celular;
- Entrada de dados: a capacidade de entrada de dados nesses dispositivos ocorre de forma limitada, quando relacionado a um telefone celular, possuem doze teclas , sendo dez delas números e as outras duas caracteres especiais, já nos *palmtops* a entrada de dados é realizada através de uma caneta ou de teclas alfanuméricas;
- Largura de banda: mesmo obstáculo possuído pela computação convencional, a segurança, já que os dados são transmitidos pelo ar através de ondas eletromagnéticas, onde a confiabilidade é bastante questionada. As redes *wireless* estão sujeitas a mais erros do que as redes com fio;
- Interface Reduzida: normalmente esses dispositivos possuem uma tela de pequena dimensão, tornando limitada a quantidade de informações que pode ser visualizada, sem disponibilidade de janelas;
- Pela característica de possuir grande mobilidade, pode acarretar perda de conexão;

Mas atualmente grandes fabricantes fazem investimentos acentuados nesta área para fazer com que tenhamos mais recurso tecnológico nos dispositivos móveis, embora ainda não se faça uso de todos esses recursos. O mercado está esperando um crescimento grande no número de usuários conectados a rede sem fio.

.

Capítulo 4

Uma ferramente para gerenciamento de informações e recursos humanos

Foi desenvolvido um protótipo de software para simulação de transmissão de dados através de uma comunicação entre o emulador de um dispositivo MIDP e um servidor que disponibiliza informações dos projetos, professores e alunos do Núcleo de Pesquisa do Departamento de Sistemas Computacionais da Escola Politécnica da Universidade de Pernambuco.

O aplicativo está dividido em duas partes: a primeira delas, um protótipo do *software* do celular, será executado utilizando um emulador, que simule o comportamento de um dispositivo móvel, tendo suporte a tecnologia J2ME com o perfil MIDP. Esse dispositivo deve interagir com o servidor depois de estabelecida uma conexão HTTP, onde são passados alguns parâmetros. Os parâmetros informados variam de acordo com a consulta a ser realizada, e é através deles que o servidor saberá qual é a base de dados referenciada na consulta, dentre outras informações.

A segunda parte é responsável pelo servidor da aplicação e foi desenvolvida utilizando os recursos disponíveis na plataforma J2EE. Após o aplicativo servidor receber a requisição vinda do módulo cliente, este executa os dados recebidos e encaminha de volta o resultado do processamento. Os dados, então, podem ser visualizados na tela do dispositivo móvel, que neste caso estará sendo executa no emulador. A forma como ocorre a comunicação entre o cliente e o servidor pode melhor ser observada através do diagrama de seqüência[5] que pode ser visualizado na Figura 12.

Entretanto, para tornar possível estabelecer uma conexão entre essas duas partes, o módulo que trata as requisições deverá estar associado a um servidor de aplicação que ofereça suporte a J2EE. No caso em questão é aplicado o *Tomcat* [1] que é mantido em execução em um PC (*Personal Computer*) que funciona como um servidor. Neste são instalados *servlets* que serão responsáveis por tratar as requisições recebidas do cliente. O *Tomcat* é um subprojeto do projeto *Jakarta Apache Software Foundation* [16], que tem como objetivo o desenvolvimento de aplicativos com código aberto baseado na plataforma Java. Essas tecnologias foram apresentadas no Capítulo 2.

Dessa forma, é necessário que o servidor da aplicação seja iniciado como um processo ativo para tornar possível o recebimento das requisições vindas do cliente e assim poder estabelecer a conexão entre o dispositivo móvel e o sistema corporativo desenvolvido na arquitetura J2EE.

O NUPEC possui um sistema cujo nome é CIAP – Controle Inteligente de Acesso de Pessoas, que tem o objetivo de gerenciar informações de pessoas que freqüentam o núcleo, bem

como monitoramento de horários de acesso. Baseado neste sistema e após algumas pesquisas e conversas com os professores do NUPEC, foram levantados alguns requisitos a serem implementados. Esses requisitos serão listados a seguir e mais bem explicados e exemplificados no decorrer do Capítulo.

- Informar as pessoas que estão no NUPEC no momento da consulta;
- Informar os horários em que os professores e alunos possivelmente estarão no NUPEC;
- Informar as pessoas que passaram pelo NUPEC em um determinado dia
- Informar dados acadêmicos dos alunos;
- Informar dados dos professores
- Informar a quantidade de horas que uma pessoa permaneceu no NUPEC durante a semana
- Informar dados sobre projetos em desenvolvimento no NUPEC

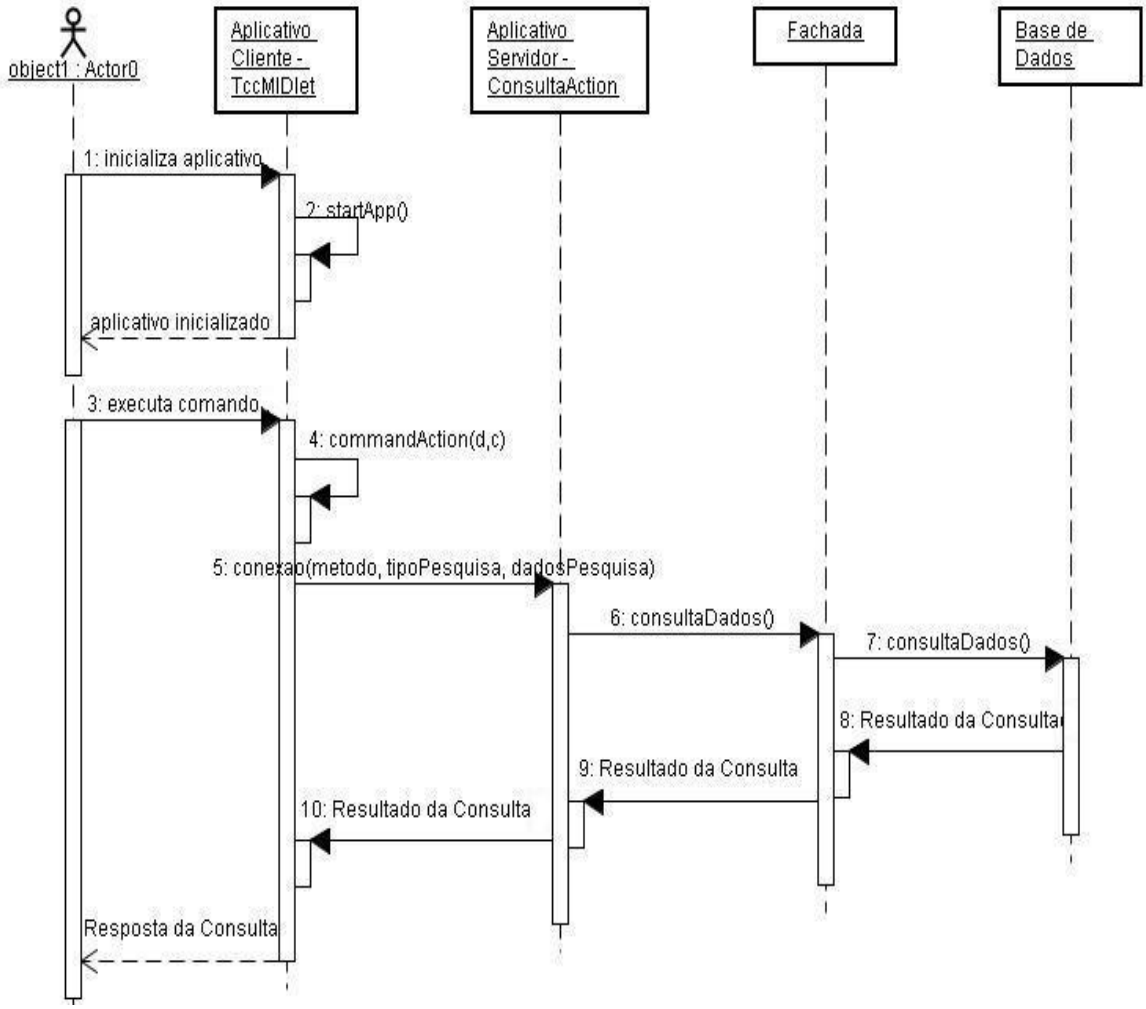

Figura 12. Diagrama de Seqüência da Comunicação entre o cliente e o Servidor

4.1 Ferramentas Utilizadas

A implementação do módulo servidor foi feita utilizando a linguagem Java coorporativa (J2EE) através da IDE Eclipse [9] onde o código foi escrito, compilado e depurado. A plataforma Eclipse foi desenvolvida por empresas que apóiam o uso de uma arquitetura aberta para criação de ambientes integrados de desenvolvimento nas iniciativas de código livre. A estrutura da IDE pode ser observada na Figura 13.

Para o desenvolvimento do módulo cliente foi utilizada a linguagem Java para dispositivos móveis (J2ME) com perfil MIDP 1.0. Encontra-se disponível no *site* da *Sun Microsystems* [9] a ferramenta utilizada que é denominada Wireless Toolkit que provê a implementação do MIDP junto com um emulador genérico de terminais para poder ser visualizado o comportamento do Midlet. Como o Toolkit não possui uma IDE, para completar o desenvolvimento do código foi acrescentado o *plugin* Eclipse J2ME [26] na IDE Eclipse.

A versão utilizada do Wireless Toolkit foi a 2.1_01 e o emulador escolhido foi o *MediaControlSkin* [15], como demonstrado na Figura 14.

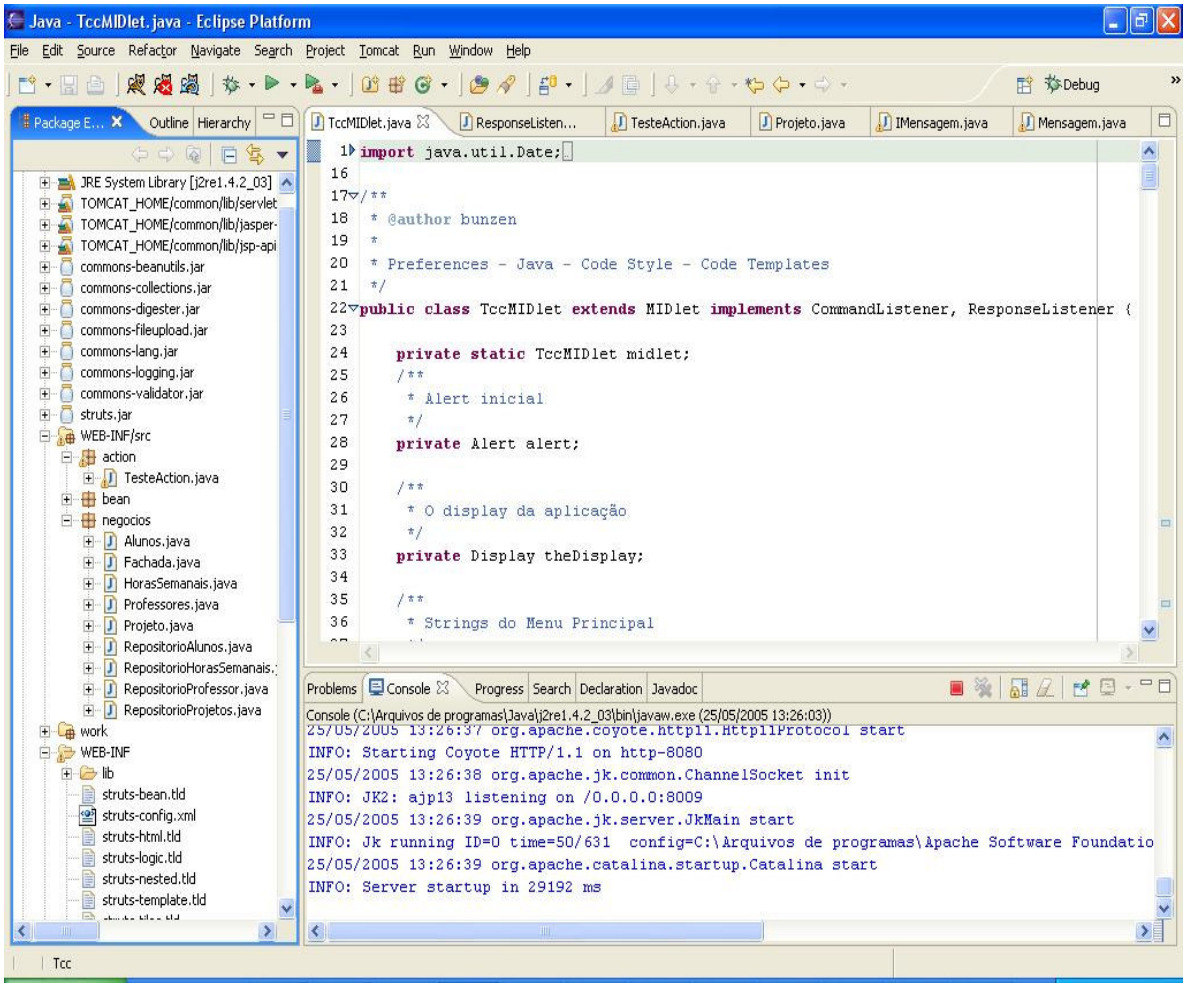

Figura 13. IDE Eclipse

Figura 14. Emulador MediaControlSkin

4.2 Arquitetura da Aplicação

O aplicativo foi desenvolvido baseado na arquitetura cliente/servidor, onde o cliente estabelece uma conexão através do protocolo HTTP e então as requisições são enviadas para o servidor *Tomcat* através do método POST. O Servidor recebe as requisições, verifica os parâmetros recebidos para saber qual base de dados está sendo referenciada e, só então, os dados são processados. Depois de obtido o resultado, uma mensagem com resposta da requisição é devolvida para o módulo cliente da aplicação e então exibida na tela do emulador.

No escopo da aplicação, o cliente é um aparelho celular com a característica MIDP, rodando o aplicativo desenvolvido na plataforma J2ME. Já o servidor é um PC com o Tomcat instalado rodando a aplicação J2EE.

4.2.1 Arquitetura do Cliente da Aplicação

A implementação cliente foi totalmente desenvolvida utilizando a linguagem Java para dispositivos móveis, tendo como ponto de partida a classe `TccMIDLet, MIDLet` responsável pela aplicação. Uma visão geral da arquitetura cliente pode ser visualizada na figura seguinte que apresenta o diagrama de classe para este módulo.

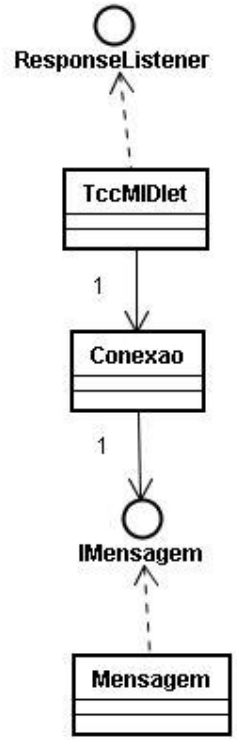

Figura 15. Diagrama de Classe do Módulo Cliente

Quando o usuário deseja inicializar a aplicação, uma mensagem é enviada para a KVM do dispositivo MIDP e esta então chama o método `startApp`. É de responsabilidade deste método carregar o aplicativo, criar a imagem que é passada como parâmetro do *alert* que será mostrado por um tempo pré-determinado no momento da inicialização, só então, a tela que possui o menu principal do aplicativo é exibida. O código referente ao método `startApp` pode ser visto no Quadro 2.

```
protected void startApp() throws MIDletStateChangeException  {
      midlet = this;
      initialImage = Image.createImage("/tcc.png");
      alert = new Alert ("Monografia", "Tcc", initialImage,
      AlertType.INFO);
      this.theDisplay.setCurrent(alert, menuPrincipalList);
      }
```

Quadro 2. Método startApp

Devido a classe MIDlet ser abstrata, tem alguns métodos que precisam ser implementados para que ocorra uma mudança de estados. Dentre eles temos o `startApp`, que já foi discutido no parágrafo anterior, como também o `pauseApp` e o `destroyApp`.

O método `pauseApp` faz parte do ciclo de vida de um MIDlet e quando chamado, sinaliza para o MIDlet suspender temporariamente alguns processos. O método deve fazer parte do código mesmo que não seja utilizado no aplicativo.

Já o `destroyApp` geralmente é utilizado para liberar recursos e salvar algum dado na persistência pois , após sua execução, o ciclo de vida do MIDlet é encerrado. Nesse trabalho, não foi utilizada persistência de dados, visto que, o resultado de uma pesquisa varia a cada nova consulta. Esse método será chamado sempre que o usuário apertar o botão "SAIR", interrompendo o aplicativo. O Quadro 3 mostra onde o `destroyApp` é chamado.

```
if (c = = exitCommand) {
      destroyApp(false);
      notifyDestroyed();
}
```

Quadro 3. Método destroyApp

Na Figura 16 temos um exemplo do ciclo de vida de um MIDlet.

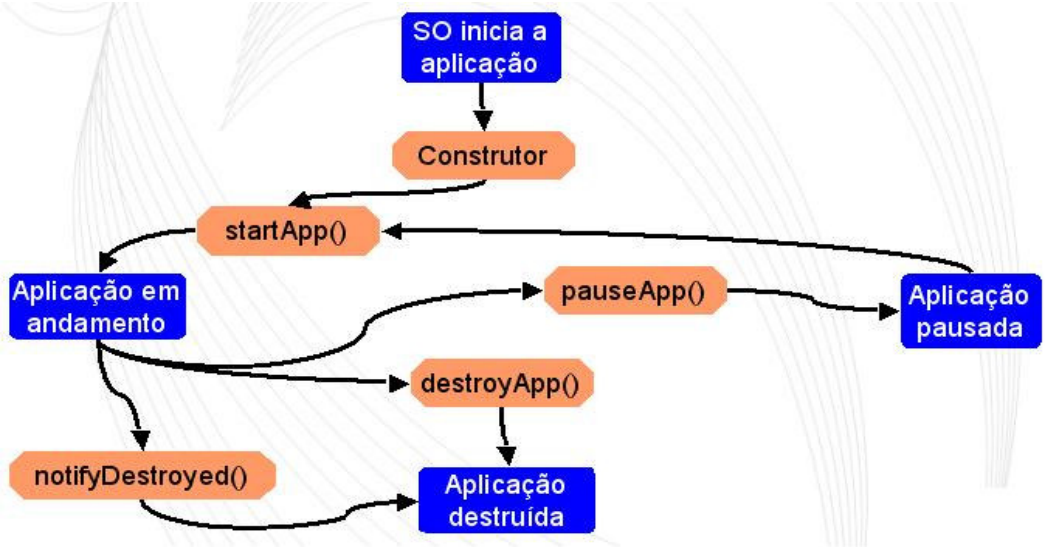

Figura 16. Ciclo de Vida de um Midlet

Quando o aplicativo é inicializado, uma tela com as opções do menu inicial é exibida. No momento em que o usuário escolhe um item para pesquisa, uma conexão com o servidor deve ser estabelecida, passando os parâmetros da consulta. Esses dados devem ser entregues ao servidor e o cliente fica à espera de uma resposta.

Ao receber uma resposta à requisição, a classe de comunicação (classe `Conexao`) encaminha a resposta para a classe responsável por tratá-la (classe `Mensagem`) e também deve notificar ao Midlet sobre a resposta. Para isso foi criada a interface `ResponseListener`, que possui o método `setResponse` e recebe como parâmetro a *String* de resposta. Também foi criado um método `setListener` de um objeto que implementa a interface `ResponseListener` na classe `Mensagem`.

No caso em questão, como o Midlet deve ser notificado, então a classe `TccMIDlet` implementa a interface `ResponseListener`, e no corpo do método `setResponse` faz o tratamento apropriado da resposta enviada pelo servidor. Dessa forma a classe `Mensagem` chama simplesmente o método de seu *listener* quando recebe a resposta, notificando assim ao Midlet que a mensagem chegou.

Usando um exemplo para o melhor entendimento do que foi descrito acima, no Quadro 4 está o código referente ao pedido de conexão quando um usuário deseja saber as informações de um determinado professor. Os parâmetros passados no construtor da classe `Conexao` significam:

- O método utilizado: no caso como foi 0 o método foi o POST;
- A base de dados da consulta: como foi 1 significa que a pesquisa é referente a professores;
- A String de consulta, que neste caso é o *login* do professor;

```
else if (c == pesquisarCommand) {
     String professor = professorField.getString();
     Conexao conexao;
     conexao = new Conexao("0", "1", professor);
}
```

Quadro 4. Envio de dados da Consulta

No Quadro 5 está descrita a forma como foi implementado o momento em que a classe `Conexao` recebe a resposta do servidor. A resposta chega em formato de *stream* e é passada junto com o tamanho como parâmetro do método `receber` para sofrer o tratamento adequado.

```
if (rc == HttpConnection.HTTP_OK) {
  dis = new DataInputStream(conn.openInputStream());
  int len = (int) conn.getLength();
  formMsg.receber(dis, len);
}
```

Quadro 5. Classe Conexão recebendo resposta do servidor

No Quadro 6 está a demonstração de como ocorre o tratamento da resposta, realizado pela classe `Mensagem`. A mensagem recebida é lida no formato UTF e transformada para String onde será passada como parâmetro do método `setResponse`.

```
public void receber(DataInputStream dis, int len){
      try{
          if(len > 0){
               String respostaServidor =
dis.readUTF();

     this.listener.setResponse(respostaServidor);
           }
     } catch(IOException e){
         e.printStackTrace();
     } catch(Exception e){
         e.printStackTrace();
     }
```

Quadro 6. Tratamento da resposta vinda do servidor

Ao final do ciclo é executado o tratamento através do método `setResponse` no Midlet da aplicação, como mostra o Quadro 7. Como pode ser visto, a variável `resultadoForm` é do tipo `Form`. Um `Form` é um componente gráfico que faz parte da interface de alto nível de J2ME e pode conter alguns itens. No exemplo abaixo está sendo utilizado uma `StringItem`, que como um próprio nome diz é um item em formato de texto. Também estão sendo adicionados ao `Form` dois `Commands`.

Um `Command` é responsável por encapsular as informações de uma determinada ação. No exemplo do Quadro 7 podemos ver o `voltarCommand` e o `sairCommand`. Se o usuário escolher a opção de `voltarCommand`, será exibida a tela anterior, caso contrário o método `destroyApp` será chamado interrompendo o aplicativo.

```
public void setResponse(String resp) {
          resultadoForm = new Form("Resultado da
Pesquisa");
          StringItem item = null;
          item = new StringItem("",resp);
          resultadoForm.append(item);
          resultadoForm.addCommand(voltarCommand);

          resultadoForm.addCommand(sairCommand);
          resultadoForm.setCommandListener(this);
          theDisplay.setCurrent(resultadoForm);
}
```

Quadro 7. Método setResponse no TccMIDlet

O procedimento descrito resulta no seguinte comportamento no emulador. (Ver Figura 17)

Figura 17. Resultado da Pesquisa por Professor

Na Figura 17 pode-se ver inicialmente a tela de professores, onde foi informado o login do professor e depois o usuário clicou no `command` "PESQ". No momento de estabelecer a conexão é informado ao cliente que vai ocorrer uma transferência de dados. Caso o usuário escolha a opção "YES" o resultado da pesquisa é informado na tela seguinte, caso contrário não ocorrerá o envio dos dados.

Como foi visto, a classe `Conexao` é responsável por estabelecer conexão com o servidor da aplicação, através do protocolo HTTP, sendo assim, fica sob a responsabilidade dessa classe preencher as propriedades do cabeçalho necessário ao protocolo HTTP como pode ser visto no Quadro 8.

```
conn.setRequestProperty("User-Agent","Profile/MIDP-1.0
Configuration/CLDC-1.0");
conn.setRequestProperty("Content-Type", contentType);
conn.setRequestProperty("Content-Language", "en-US");
conn.setRequestProperty("Accept", accept);
conn.setRequestProperty("Content-Length", tamanhoBytes
+ "");
```

Quadro 8. Propriedades do cabeçalho HTTP

É no construtor da classe `Conexao` onde são informados :

- `metodoConexao`: apenas o método POST foi utilizado para estabelecer a conexão;
- `tipoPesquisa`: informa a base de dados que deve ser utilizada;
- `dadoPesquisa`: texto que varia conforme a consulta a ser realizada;

```
public Conexao(String metodoConexao, String
tipoPesquisa, String dadoPesquisa) {
          this.metodoConexao = metodoConexao;
          this.tipoPesquisa = tipoPesquisa;
          this.dadoPesquisa = dadoPesquisa;
}
```

Quadro 9. Construtor da classe Conexao

Os valores do `tipoPesquisa` podem variar da seguinte forma:

- 0: Consulta referente a projetos;
- 1: Consulta referente a professores;
- 2: Consulta referente a alunos;
- 3: Consulta referente a horários;
- 4: Consulta referente às pessoas que estão no NUPEC;
- 5: Consulta referente às pessoas que passaram no NUPEC em um determinado dia;

4.2.2 Arquitetura Servidor da Aplicação

O aplicativo que é executado no módulo servidor foi desenvolvido utilizando a linguagem Java coorporativa, e possui o importante papel de capturar as solicitações vindas do cliente para fazer o processamento necessário.

Visando diminuir a complexidade na implementação, a arquitetura foi criada de forma que apenas a classe `ConsultaAction` receba as requisições.

O sistema servidor foi desenvolvido seguindo os padrões *Facade* [10], *Singleton* [10] e *Bridge* [10]. A estruturação do sistema em subsistemas é essencial para a redução de complexidade, facilitando a manutenção.

O padrão *Facade* propõe uma interface unificada para um conjunto de interfaces de um subsistema, definindo uma única interface de alto nível, que torna o subsistema mais fácil de ser utilizado. Sendo assim as solicitações ao sistema servidor serão feitas a classe `Fachada` e não a cada objeto particular. O objeto `Fachada` se encarrega de encaminhar a solicitação ao objeto correspondente.

O objetivo da utilização do padrão *Singleton* é garantir que a classe `Fachada` possua apenas uma instância, e possibilitar o acesso global a essa instância. A forma como foi utilizado esses dois padrões podem ser visto no Quadro 10.

O padrão Bridge tem como intenção separar a camada de dados da camada de negócios utilizando interfaces, de modo que elas possam variar fácil e independentemente.

A `Fachada` contém instâncias das interfaces `IRepAlunos`, `IRepProfessores`, `IRepHorasSemanais` e `IRepProjetos`.

```
public static Fachada getInstance(){
          if(Fachada.fachada == null){
               fachada = new Fachada();
          }
          return fachada;
}
```

Quadro 10. Método getInstance da classe Fachada

A Figura 18 mostra uma visão geral do comportamento do módulo servidor.

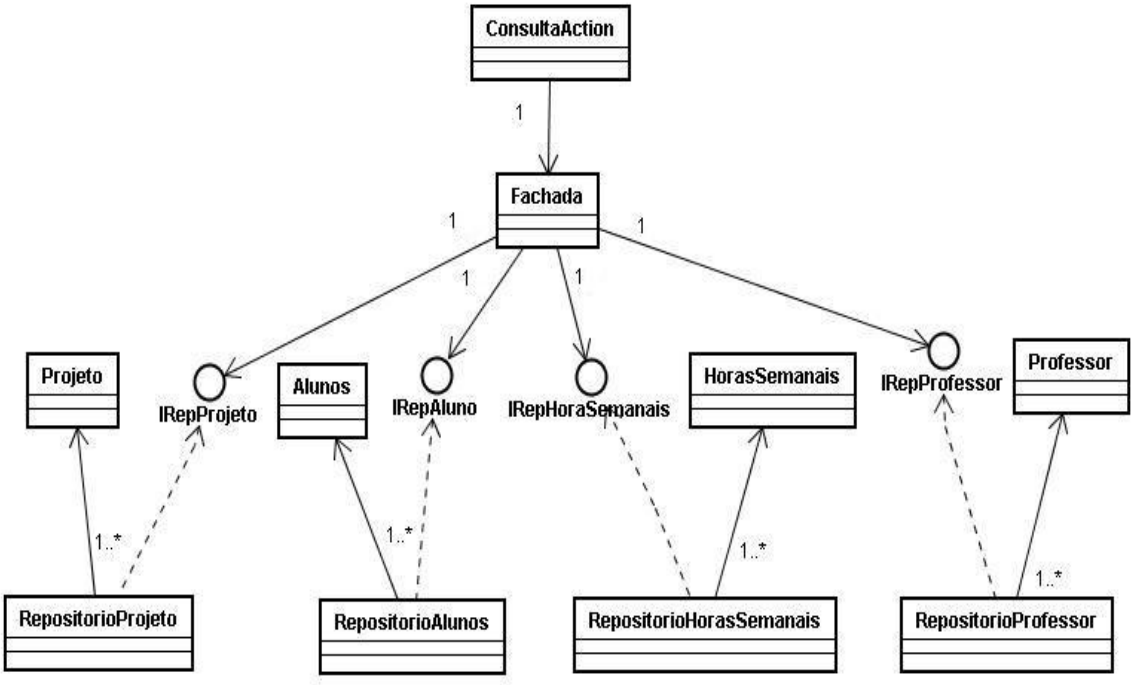

Figura 18. Diagrama de Classes do módulo Servidor

Seguindo o exemplo utilizado no módulo cliente, no momento em que o usuário informa o *login* do professor e aperta o botão "PESQ", uma conexão é estabelecida e o cliente envia uma requisição ao servidor.

A solicitação chega ao sistema coorporativo através da classe `ConsultaAction`, e este captura o parâmetro (`tipoPesquisa`) que informa a base de dados que será utilizado, como pode ser visto no Quadro 11.

```
String tipoPesquisa = request.getHeader("tipoPesquisa");
```

Quadro 11. Obtendo o tipo da pesquisa

No exemplo que está sendo ilustrado o `tipoPesquisa` recebe o valor igual a "1", o que significa que a consulta é referente a professores e que a pesquisa deve ser realizada na base de dados dos professores.

```
1. else if (tipoPesquisa.equals("1") ) {
2.   String idProf = request.getParameter("id");
3.   Professores prof = fachada.selectProf(idProf);
   4.   ByteArrayOutputStream baos = new
   5.   ByteArrayOutputStream();
6. DataOutputStream dos = new DataOutputStream(baos);
7. dos.writeUTF("Nome do Professor: "
8.
9.        + prof.getNomeProf() + "\n"
10.       + "Responsável pelo Projeto: "
11.       + prof.getProjetoOrienta() + "\n"
12.       + "Título: "
     13.  + prof.getTitulo() + "\n"
     14.  + "Dias no Núcleo: "
     15.  + prof.getDias() + "\n"
     16.  + "Horário no Núcleo: "
     17.  + prof.getHorarios());
18.  data = baos.toByteArray();
```

Quadro 12. Exemplo da consulta dos Professores

O acesso aos dados dos professores é feito através da `Fachada`, chamando o método `selectProf` passando como parâmetro o `idProf` que chegou através da solicitação do cliente, como pode ser visto na linha 3 no Quadro 12.

A classe `RepositorioProfessores` é responsável pela manipulação do armazenamento de dados e isola os dados referente a professores do resto do aplicativo. Esta classe está simulando uma base de dados com informações dos professores, utilizando armazenamento em memória, de forma que se for necessária uma troca no meio de armazenamento (arquivos, banco de dados, etc.), apenas esta classe, referente a professores, deverá ser trocada ou modificada. O mesmo ocorrendo para os demais repositórios de dados do sistema.

Depois de processar os dados, o servidor escreve a mensagem utilizando o formato UTF-8 através do método `writeUTF` na linha 7 do Quadro 12, visto que, a leitura desse *array* de *byte* é realizado no lado cliente com o método `readUTF`. Finalizando o ciclo, a mensagem é enviada de volta para módulo cliente e assim apresentada na tela do emulador. O código fonte responsável por essa etapa pode ser visto no Quadro 13.

```
response.setStatus(HttpServletResponse.SC_OK);
response.setContentLength(data.length);
response.setContentType("application/octet-stream");
OutputStream os = response.getOutputStream();
os.write(data);
os.close();
```

Quadro 13. Dados encaminhados para o cliente

4.3 Limitações do Contexto

O NUPEC possui um relógio de ponto MADIS RODBEL [20] modelo 3707 TCP / IP, onde discentes e docentes devem passar um cartão de identificação tanto no momento de entrada quanto no momento de saída do núcleo. O relógio então, deveria comunicar-se com um sistema existente no NUPEC e armazenar essas informações em um banco de dados. A partir desse ponto, o protótipo de *software* referente a esta monografia, utilizaria as devidas informações enviadas pelo relógio.

Entretanto, o protocolo utilizado para realizar a transferência de dados entre o sistema existente e o relógio de ponto não funcionou corretamente. O relógio não consegue entender o formato da mensagem que chega até ele, o que indica que a especificação de formato da mensagem, enviada pela empresa do relógio, possui algum erro.

Com essa falha de comunicação o banco de dados é impedido de ser alimentado. Dessa forma, o sistema referente a esse trabalho não teria os dados disponíveis para realizar as consultas necessárias de acordo com a preferência do usuário.

De acordo com o que foi descrito acima, o aplicativo em questão foi desenvolvido usando uma simulação dos dados para consulta, utilizando armazenamento em memória. O fato é que, independente do mecanismo de persistência que foi utilizado, foram construídas interfaces com as assinaturas dos métodos que devem ser implementados em cada forma de persistência. Com isto, se o mecanismo de persistência for modificado, ou seja, o relógio consiga alimentar o banco de dados, então o método de persistência mude de memória para banco de dados, basta criar um novo repositório que acesse o banco de dados e implemente a mesma interface já utilizada. Ou seja, não será necessário modificar qualquer outra classe que não esteja na camada de dados.

Um outro problema encontrado no decorrer da implementação, foi à falta de familiaridade com o *Tomcat*, sua instalação e integração com o *Struts.* Foram encontrados vários erros decorrentes a falta de *libs* e arquivos de configuração necessários para o correto funcionamento do *Struts* junto com o servidor de aplicação.

4.4 Resultados

Foi desenvolvido um protótipo de software, onde um aplicativo cliente comunica-se com um sistema servidor através de uma conexão HTTP e então transmite os dados para uma determinada consulta. Os dados são processados pela aplicação servidora e depois uma mensagem enviada para o cliente como resposta à requisição.

A aplicação foi testada usando um PC (*Personal Computer*) com um processador de 1.4GHz e 256 de RAM com o servidor *Tomcat* hospedando a aplicação do módulo servidor. Já o

aplicativo cliente foi executado utilizando o emulador *Wireless Toolkit* para simular o dispositivo móvel.

Ao executar uma aplicação J2ME são gerados dois novos arquivos. Um arquivo no formato "*jar*" (*Java Application Resources*) e outro no formato "*jad*" (*Joint Application Development*) . O arquivo *jar* encapsula todas as classes de um aplicativo Java. Em especial para J2ME esse arquivo encapsula toda a aplicação, incluindo o Midlet, as outras classes e figuras se existirem. O arquivo *jad* contém todas as informações da aplicação, tais como nome do Midlet, tamanho do arquivo *jar,* a versão do aplicativo, o caminho do arquivo *jar,* dentre outras. O arquivo *jar* do aplicativo em questão tem o tamanho de 100K, como pode ser visto no Quadro 14. O aplicativo gerado pelo módulo cliente só pode ser instalado em dispositivos móveis que possuam as seguintes características:

- suporte a *MIDP 1.0;*
- transferência de dados ;
- memória que suporte um aplicativo de 100k.

```
MIDlet-1: Tcc, Tcc.png, TccMIDlet
MIDlet-Jar-Size: 100
MIDlet-Jar-URL: Tcc.jar
MIDlet-Name: Tcc
MIDlet-Vendor: Unknown
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

Quadro 14. Arquivo Tcc.jad

A união dos dois módulos, resultou no diagrama de classes que pode ser observado na na Figura 19.

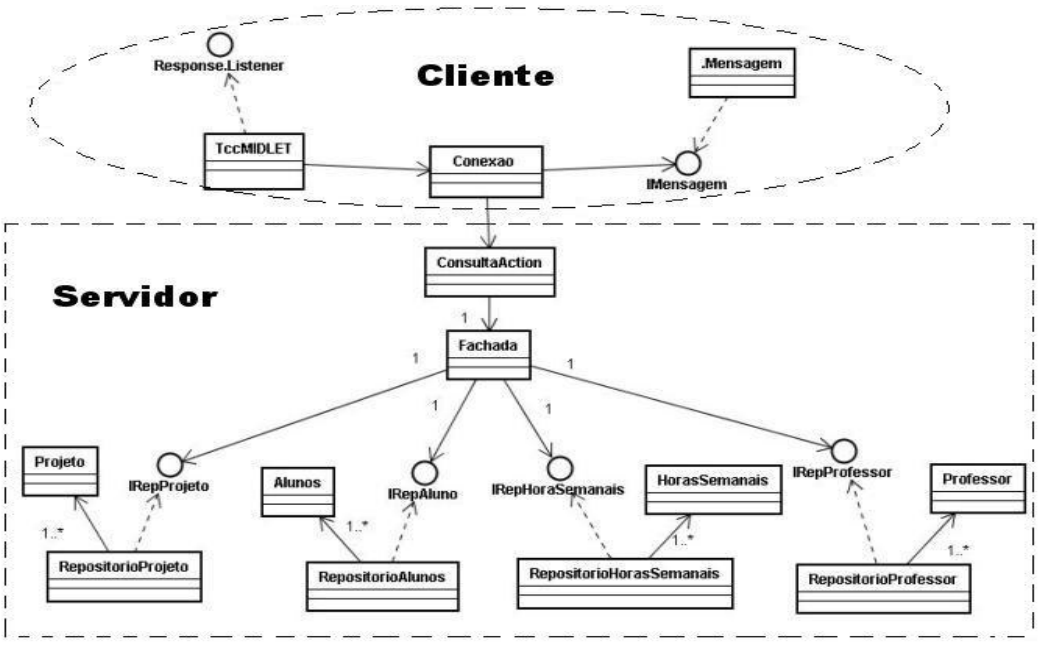

Figura 19. Diagrama de Classes do Sistema

O diagrama de casos de usos do sistema em questão pode ser observado na Figura 20.

Figura 20. Diagrama de casos de usos

Os arquivos do sistema em questão estão disponibiliados na web no endereço: http://www.cin.ufpe.br/~dpm/Arquivos/mono.zip .

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Conclusões

No presente trabalho foi constatada a evolução da comunicação móvel bem como o crescente aumento na utilização de dispositivos móveis, sendo destacados os telefones celulares. Foram realizados estudos detalhados sobre as tecnologias J2ME e J2EE, seus conceitos e características. Também foram utilizados os recursos necessários para estabelecer a comunicação entre os aplicativos que rodavam em suas respectivas plataformas.

O protótipo desenvolvido nesse trabalho comprovou, através de testes realizados, ter cumprido seus objetivos, ou seja, permite o usuário escolher qual consulta deseja realizar e informar os dados de entrada. Esses dados são enviados corretamente para a aplicação servidora que os recebe, processa de acordo com a base de dados referente à consulta realizada e manda de volta a resposta para o aplicativo cliente. O módulo cliente aceita a mensagem com o resultado da consulta e mostra na tela do emulador, encerrando o ciclo da pesquisa.

À medida que cresce o mercado de telefonia celular, aumenta também a diversidade de aplicativos que são desenvolvidos para esses aparelhos. Empresas de desenvolvimento criam cada vez mais *softwares* que trocam dados na rede e se comunicam com servidores. Contudo, apesar de todo avanço tecnológico, a transferência de grandes quantidades de dados via rede é sempre um fator problemático devido às limitações de envio de dados comuns à tecnologia de telefonia móvel. Por exemplo, ainda não existe um aplicativo que faça processamento gráfico em um servidor e envie imagens para o celular. Portanto, uma aplicação que faça consultas, como o aplicativo do trabalho em questão, é essencial que estas sejam simples e diretas, tanto a mensagem de solicitação quanto a resposta.

5.2 Trabalhos Futuros

Como trabalho futuro, poderia ser desenvolvido um sistema interno de envio de mensagens para celular, onde as mensagens poderiam ser enviadas em *broadcast* com um aviso ou algo referente a um determinado grupo de pesquisas. Um administrador poderia cadastrar no sistema informações como datas de reuniões ou seminários, e estas seriam enviadas para os usuários em uma data estipulada.

Relacionado ao trabalho em questão, seria interessante a adição de novas funcionalidades que agregassem mais valores a aplicação. Atualmente o sistema não faz nenhuma distinção entre professores e alunos, já que apenas consultas podem ser realizadas, e estas não exigem tal controle. Contudo, podem ser acrescentados novos requisitos que levem a necessidade de controlar algumas áreas de acesso. Para tanto, os professores deveriam receber tratamento diferenciado em relação aos alunos utilizando alguma forma de validação, podendo os docentes ter acesso às áreas restritas.

Dentre outros, podemos citar tais exemplos como possibilidade de novos requisitos que poderiam enriquecer o aplicativo:

- bloquear/desbloquear a entrada de alunos ao NUPEC;
- modificar informações relativas tanto a projetos quanto a discentes e docentes;
- cálculo mensal de horas trabalhadas por cada aluno, podendo ser dividido em projetos caso algum aluno esteja alocado em mais de um projeto.

O sistema pode ser adaptado de forma que possa funcionar como um gerenciador de recursos, dessa maneira pessoas podem ser cadastradas e descadastradas e também dados podem ser modificados.

Uma outra possibilidade de melhoria é tornar a interface gráfica mais rica de detalhes. Como já foi informado em Capítulos anteriores, o módulo cliente foi desenvolvido utilizando apenas a API nativa de J2ME, que é composta por alguns componentes pré-definidos, como por exemplo, *Forms*, *Alerts*, *DateFields, TextBoxs*, dentre outros. Essa interface de alto nível é fácil de ser implementada, porém, possui um *design* simples e não muito atraente.

Seria interessante migrar o código da interface gráfica para utilizar componentes otimizados pertencentes a API de baixo nível. Esses componentes herdam da classe *Canvas* e são responsáveis por pintar a tela *pixel* a *pixel.* Tornando assim, as telas do aplicativo cliente, personalizadas, possuindo um *design* muito mais bonito, com detalhes atrativos e possibilidade de animações gráficas.

Bibliografia

[1] APACHE TOMCAT. Disponível em <http://jakarta.apache.org/tomcat/>. Acesso em: mar - 2005.

[2] ARMSTRONG, Erik et al. *The J2EE™ 1.4*. Disponível em <http://www.java.sun.com>. Acesso em: fev - 2005

[3] CAVANESS, Chuck. *Programming Jakarta Struts*, O'Relly & Associates, Inc, 2002.

[4] CESTA, A. *A linguagem de programação Java™*. Instituto de Computação, São Paulo, 1996. Disponível em: <http://www.dcc.unicamp.br/~aacesta >. Acesso em: nov – 2004

[5] CHEESEMAN John, DANIELS, John . *UML Components: A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.

[6] DIAS, Clessis; FONTES, Wescley. *Desenvolvimento de Aplicações para Dispositivos Móveis utilizando a Plataforma J2ME*. 2003. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Federal do Pará, Belém.

[7] DIAS, K. L; Sadok, D. F.H. *Internet Móvel: Tecnologias, Aplicações e QoS*. XIX Simpósio Brasileiro de Redes de Computadores, Florianópolis, 2001.

[8] DORNAN, Andy. *Wireless communication: O guia essencial de comunicação sem fio*. Rio de Janeiro: Campus, 2001.

[9] FERRAMENTA ECLIPE. Disponível em: <http://www.eclipse.org/>. Acesso em: dez – 2004.

[10] GAMMA,E. HELM,R. JOHNSON,R VLISSIDES, J *Design Patterns: Elements of Reusable Object-Oriented Software (GOF)* Addison Wesley, 1995.

[11] GOMES, A. *J2ME – Visão Geral*. Disponível em: <http://www.mundooo.com.br/> . Acesso em: jan 2005.

[12] GUPTA, V.; DASS, A.; CHAUHAN, Y. *Cracking the Code – Wireless Programming with J2ME™*. Hungry Minds, Inc, 2002.

[13] HORSTMANN, Cay S., CORNELL, Gary. *Core Java: fundamentos*. v.1, São Paulo: Pearson MakronBooks, 2003. cap.1, p. 1-14.

[14] HUSTED, Ted et al. *Struts em ação*, São Paulo: Editora Ciência Moderna, 2004.

[15] J2ME Wireless Toolkit 2.2. Disponível em: <http://java.sun.com/products/j2mewtoolkit/download-2_2.html>. Acesso em fev - 2005.

[16] JAKARTA APACHE SOFTWARE FOUNDATION. Disponível em: <http://jakarta.apache.org>. Acesso em mar – 2005.

[17] JAVA 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices. White Paper. Sun, 2000. Disponível em <http://java.sun.com/developer/technicalArticles/ConsumerProducts/intro/#j2me>. Acesso em jan – 2005.

[18] KEOGH, James. *J2ME: The Complete Reference*. McGraw-Hill/Osborne, 2003.

[19] LAUDON, Kenneth C.; LAUDON, Jane Price. Sistemas de informação. Rio de Janeiro: LTC, 1999.

[20] MADIS RODBEL. Disponível em: <http://www.rodbel.com.br> . Acesso em fev 2005.

[21] MAGAN, M.; VARGAS, P.; AZZOLIN, D. *Técnicas para desenvolvimento de aplicações orientadas a objetos utilizando a linguagem Java*. III Simpósio Brasileiro de Linguagens de Programação, Porto Alegre, 1999.

[22] MAHMOUD, Q. H. *Wireless Software Design Techniques What every wireless software developer should know*. 2002. Disponível em <http://wireless.java.sun.com/midp/articles/uidesign/ > . Acesso em: nov – 2004.

[23] MUCHOW, John W. *Core J2ME - Technology & MIDP*. Pearson MakronBooks, 2004.

[24] PASIN, Márcia. *Réplicas para alta disponibilidade em arquiteturas orientadas a componentes com suporte de comunicação de grupo*. 2003. 127 p. Tese de Doutorado – Instituto de Informática, Universidade do Rio Grande do Sul, Rio Grande do Sul, 2003.

[25] BANSAL, V.; DALTON, *A. Performance analysis of web services on wireless PDAs*. Duke University Computer Science, Duke, Disponível em: <http://www.cs.duke.edu/~vkb/advnw/project/index.html>. Acesso em : abr – 2005.

[26] J2ME Wireless Toolkit 2.2. Disponível em: <http://java.sun.com/products/j2mewtoolkit/download-2_2.html>. Acesso em fev – 2005

[27] RIGGS, Roger, et al. *Programming wireless devices with the Java 2 platform, micro edition*. Boston: Addison Wesley, 2001.

[28] ROCHA, H. *Desenvolvimento de Applets & Aplicações em Java*. Belém, 1998.

[29] SCHAEFER, Carina. *Protótipo de aplicativo para transmissão de dados a partir de dispositivos móveis aplicados a uma empresa de transporte*. 2004. 53f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

[30] SERVLET ESSENTIALS. Disponível em: <http://www.novocode.com/doc/servlet-essentials/>. Acesso em maio – 2005

[31] SILVA, W. *Tecnologias Java para Sistemas Embarcados*. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Informática, Universidade Federal de Pernambuco, Recife, 2001.

[32] PLUGIN ECLIPSEME. Disponível em: <http://eclipseme.sourceforge.net>. Acesso em nov – 2004.

[33] TITTLEL. São Paulo: Ed. XML. Bookman, 2003.

[34] WHITE, James P.; HEMPHIL, David A. *Java 2 Micro Edition*, UK: Ebook Edition, 2003.

[35] SUN. Disponível em : <http://www.sun.com >. Acesso em: nov – 2004

[36] Making HTTP Connections with MIDP. Disponível em :http://developers.sun.com/techtopics/mobility/midp/ttips/httpcon/. Acesso em fev – 2005.

Apêndice A

Testes

Apresentaremos aqui as principais telas do sistema, com suas respectivas entradas de dados, para consultar informações sobre projetos, alunos e professores do NUPEC (Núcleo de Pesquisa em Engenharia da Computação) da Escola Politécnica de Pernambuco.

Horas Semanais no Núcleo

Essa pesquisa diz respeito a quantidade de horas passadas no NUPEC, por uma determinada pessoa, durante a semana correspondente.

Ao inicializar o aplicativo, o usuário pode escolher a primeira opção que indica: Horas semanais no núcleo e apertar no botão "SELEC". Em seguida, o aplicativo passa para a tela seguinte onde é pedido o *login* da pessoa a qual deve ser feita a consulta. Se o usuário escolher a possibilidade de voltar, o sistema retorna a tela anterior, caso contrário é mostrada a tela com o resultado da busca. O ciclo descrito pode ser observado na Figura 21.

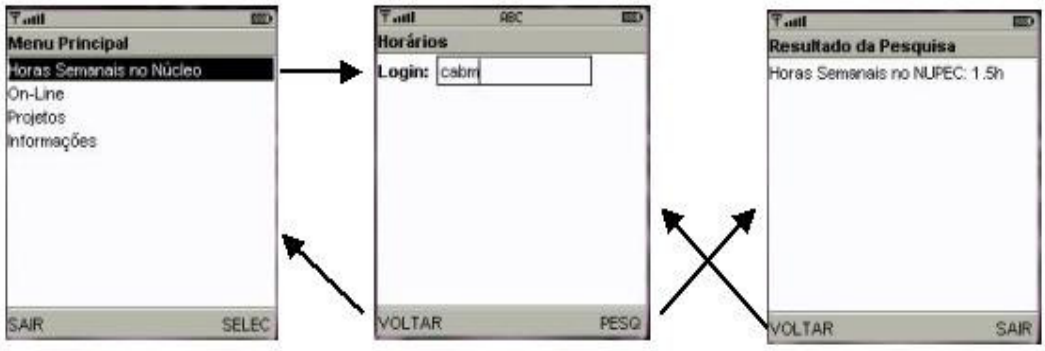

Figura 21. Horas Semanais no Núcleo

Busca On-Line

A consulta de busca On-line deve ser realizada caso ocorra a necessidade de saber quem está presente no NUPEC em um determinado momento.

Caso seja desejada realizar essa busca, ao inicializar o aplicativo, no Menu Principal deve ser escolhida da opção: On-Line. O aplicativo será direcionado para a tela de Resultado da Pesquisa onde estará sendo informado o *login* dos professores e alunos que estão no núcleo.

A lógica de implementação dessa consulta foi feita simulando a comunicação do relógio de ponto com a base de dados. Levando em consideração que sempre que uma pessoa for entrar ou sair do Núcleo, o cartão de identificação deve ser passado no relógio, então toda vez que um determinado login aparecer em uma quantidade ímpar de vezes, significa que a pessoa está presente no NUPEC. As telas referentes a essa consulta, pode ser vista na Figura 22.

Figura 22. Busca On-Line

Projetos

A consulta referente a Projetos informa os dados de um determinado projeto que esta sendo desenvolvido no Núcleo. Tais dados são o nome do projeto, o nome do professor responsável e a principal técnica utilizada. A navegação das telas pode ser vista na Figura 23. Na tela de Resultados, caso a opção "SAIR" for escolhida acarretará na saída do aplicativo.

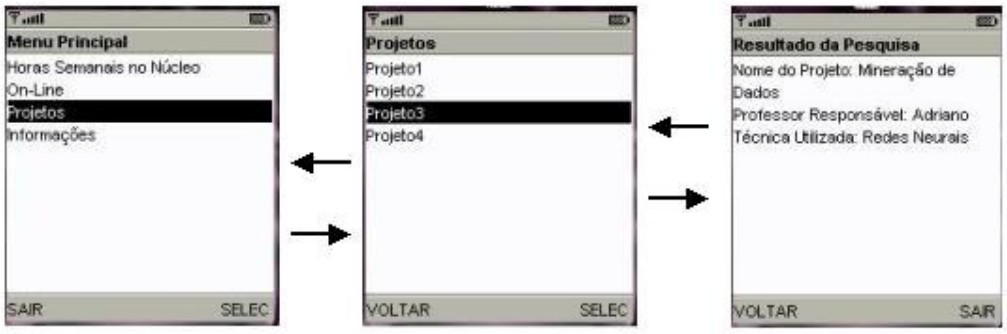

Figura 23. Projetos

Informações

Caso a opção Informações, disponível no Menu principal, for escolhida, o aplicativo será direcionado para uma nova tela com três outras opções. Como pode ser visto na Figura 24.

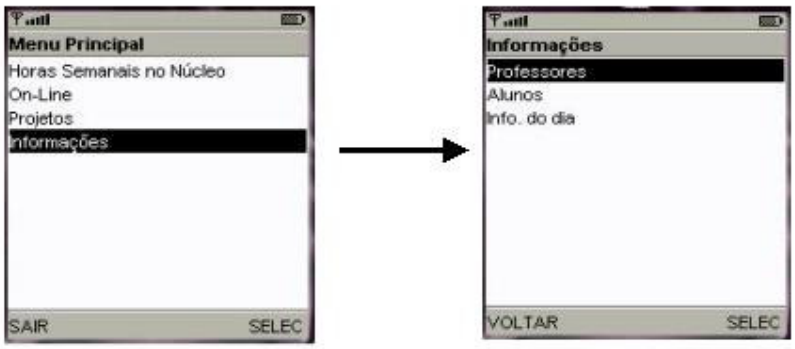

Figura 24. Informações

Se a opção escolhida for Professores, uma busca será realizada na base de dados de professores. O usuário informa o *login* do professor e em seguida o aplicativo mostra uma tela com as informações do mesmo. O comportamento dessa busca pode ser visto na Figura 25.

Figura 25. Consulta de Professores

Caso a opção escolhida for aluno, o usuário irá informar o *login* do aluno e o sistema mostrará na tela as informações referente ao aluno, como mostra a Figura 26.

Figura 26. Consulta Alunos

Se a escolha for Info. do dia, será mostrada na tela um calendário onde o usuário poderá escolher uma data para consulta. Depois de escolhida a data, deve ser clicado no botão “SELEC” e então os dados da consulta serão mostrados na tela do celular. O ciclo descrito, pode ser visto da Figura 27.

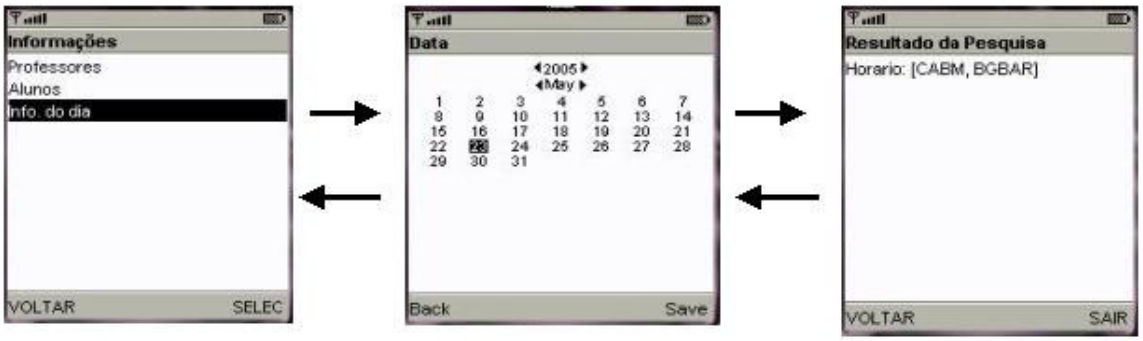

Figura 27. Consulta Info.Dia