FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL SERAPILHA DURELLI

ADAPTABILIDADE DE APLICAÇÕES WEB PARA DISPOSITIVOS MÓVEIS

MARÍLIA
2008

RAFAEL SERAPILHA DURELLI

ADAPTABILIDADE DE APLICAÇÕES WEB PARA DISPOSITIVOS MÓVEIS

Trabalho de Curso apresentado ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marilia – UNIVEM, como requisitos parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientadora:
Profa. Dra. Maria Istela Cagnin Machado

MARÍLIA
2008

DURELLI, Rafael Serapilha
Adaptabilidade de Aplicações Web para Dispositivos Móveis / Rafael Serapilha Durelli; orientadora: Maria Istela Cagnin Machado. Marília,SP:[s.n.], 2008.
82 f.

Trabalho de Curso (Graduação em Ciência da Computação) – Curso de Ciência da Computação, Fundação de Ensino "Eurípides de Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, Marília, 2008.

1. Computação Móvel 2. Aplicação Móvel 3. Adaptabilidade de Software ...4 J2ME (*Java 2 Micro Edition*) 5. WAP (*Wireless Application Protocol*)

CDD: 006

RAFAEL SERAPILHA DURELLI

ADAPTABILIDADE DE APLICAÇÕES WEB PARA DISPOSITIVOS MÓVEIS

Resultado: 8,5

ORIENTADORA: Profª. Drª. Maria Istela Cagnin Machado

1º EXAMINADOR: Jose Eduardo Santarem Segundo

2º EXAMINADOR: Fábio Lucio Meira

Marília, 19 de novembro de 2008.

Aos meus pais.
Ao meu irmão.
A minha namorada.

AGRADECIMENTOS

Primeiramente eu agradeço a Deus por ter me ajudado nas horas que eu mais precisava e ter me dado força para ir à frente. Agradeço também a minha família, pela educação, apoio, motivação, confiança e ensinamentos que me ajudaram no crescimento pessoal e moral.

À minha namorada por compreender e ter muita paciência e por sempre estar presente quando eu precisava.

À minha orientadora Profa. Dra. Maria Istela Cagnin Machado, pela confiança depositada em mim.

RESUMO

A grande oferta de serviços de telecomunicação e de tecnologias computacionais, capazes de prover mobilidade aos diferentes participantes em diferentes áreas e projetos, apresenta a oportunidade para o desenvolvimento de pesquisas no campo da computação móvel. Neste trabalho apresentam-se algumas das tecnologias que podem ser utilizadas no desenvolvimento de aplicações móveis, focando a adaptabilidade de aplicações web para dispositivos móveis. A adaptabilidade no contexto desse trabalho consiste em disponibilizar aos usuários as funcionalidades que são por eles utilizadas com frequência para qualquer tipo de dispositivo móvel. Para implementar a adaptabilidade de aplicações Web para dispositivos móveis é feito um estudo de caso nesta monografia utilizando a arquitetura M-AVA, que visa a adaptabilidade de AVAs (Ambiente Virtuais de Aprendizagem) para dispositivos móveis utilizando a adaptação de conteúdo e de interface do usuário. No estudo de caso, um protótipo de um Ambiente Virtual de Aprendizagem é adaptado para ser utilizado também em celulares, por meio da implementação da arquitetura M-AVA utilizando as tecnologias Java (especificamente, J2ME - *Java 2 Micro Edition*) e WAP (*Wireless Application Protocol*). O estudo de caso conduzido colaborou para o refinamento e aperfeiçoamento da arquitetura utilizada.

Palavras-Chave: Computação Móvel, Aplicação Móvel, Adaptabilidade de Software, J2ME (*Java 2 Micro Edition*), WAP (*Wireless Application Protocol*).

ABSTRACT

The major provision of telecommunications and computing technologies, able to provide mobility to the different participants in different areas and projects, presents an opportunity for the development of research in the field of mobile computing. This work sets out a number of technologies that can be used in the development of mobile applications, focusing on the adaptability of web applications for mobile devices. The adaptability in the context of this work is to provide users with the features that are frequently used by them to any mobile device. To implement the adaptability of Web applications for mobile devices is done by a case study in this monograph using the M-AVA architecture, which aims at the adaptability of AVAS (Virtual Learning Environment) for mobile devices using the adaptation of content and user interface . In the case study, a prototype of a Virtual Learning Environment is also adapted for use on mobile phones, through the implementation of the architecture M-AVA using the Java technology (specifically, J2ME - Java 2 Micro Edition) and WAP (Wireless Application Protocol). The case study conducted cooperating for the refinement and improvement of the architecture used.

Keywords: Mobile Computing, Mobile Application, J2ME (*Java 2 Micro Edition*), Adaptability, WAP (*Wireless Application Protocol*)

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AVA	Ambiente Virtual de Aprendizagem
CDC	Configuração de Dispositivo Conectado
CLDC	Configuração de Dispositivo Conectado Limitado
CSS	*Cascading Style Sheels*
DAO	*Data Acess Object*
GCF	*Generic Connection FrameWork*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IEEE	*Institute of Eletrical and Eletronics Engineers*
IP	*Internet Protocol*
J2EE	*Java Enterpresi Edition*
J2ME	*Java Micro Edition*
J2SE	*Java Stantard Edition*
JSP	*JavaServer Pages*
JVM	*Java Virtual Machine*
MIDP	*Móbile Information Device Profile*
MIME	*Multipurpose Internet Mail Extensions*
PDA	*Personal Digital Assistant*
PHP	*Hypertext Preprocessor*
POA	Programação Orientada a Aspectos
POO	Programação Orientada a Objetos
QoS	*Quality of Service*
SIG	*Special Interest Group*
TCP	*Transmission Control Protocol*
URL	*Uniform Resource Locator*
WAE	*Wireless Application Environment*
WAP	*Wireless Application Protocol*
WDP	*Wireless Datagram Protocol*
WML	*Wireless Markup Language*
WSP	*Wireless Session Protocol*
WTLS	*Wireless Transport Layer Security Specification*

WTP	*Wireless Transaction Protocol*
XML	*Extensible Marckup Language*

SUMÁRIO

INTRODUÇÃO

Contexto

A computação móvel emerge como uma tecnologia inovadora para a área educacional, pelo simples motivo de que cada vez mais celulares e PDA's (*Personal Digital Assistant*) tornam mais populares na sociedade nas últimas décadas, sendo assim, muitas pessoas podem dispor do custo de um aparelho móvel. Tais aparelhos necessitam de aplicações específicas para fins específicos. As tecnologias móveis se encontram atualmente na franca evolução e parecem destinadas a transformar-se em um novo paradigma dominante da computação.

Sob essa perspectiva, existem diversas aplicações web, em diferentes domínios de aplicação (por exemplo, comércio eletrônico, educacional), que necessitam estar disponíveis também em dispositivos móveis para atender a demanda e a necessidade de seus usuários. Para permitir isso é necessário que as aplicações web sejam adaptadas para o contexto de dispositivos móveis.

Sistema adaptativo é um sistema capaz de mudar seu comportamento automaticamente, durante sua execução, de acordo com mudanças no contexto do ambiente em que se encontra, e sistema adaptável é um sistema que permite adaptar facilmente uma estrutura completa ou partes específicas devido a mudança no requisito (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003).

Adaptabilidade no contexto desse trabalho consiste em disponibilizar aos usuários as funcionalidades que são por eles utilizadas com freqüência, sendo que as demais também poderão ser acessadas, só que não tão diretamente.

Para implementar adaptabilidade nesta monografia é utilizada a arquitetura M-AVA, definida por Bartholo (2008), que propõe a adaptabilidade de AVAs (Ambientes Virtuais de Aprendizagem) para dispositivos móveis utilizando uma proposta híbrida que visa a adaptação de conteúdo e de interface do usuário.

Motivação

Com a disseminação da computação móvel, diversas aplicações *Web* devem também ser disponibilizadas para dispositivos móveis, então a tendência é que tais aplicações sejam

adaptadas para tais tipos de recursos. Existem na literatura diversas técnicas de apoio a adaptabilidade de software, como por exemplo, a adaptabilidade baseada em componentes (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003), reflexão computacional (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003) , adaptabilidade de conteúdo (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003), adaptabilidade de interface do usuário (REFERENCIA), adaptabilidade dinâmica (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003), entre outras. No entanto, existem poucas iniciativas de utilização de uma abordagem híbrida para atender os diferentes aspectos de adaptabilidade ou a combinação de várias técnicas, conforme propõe a arquitetura M-AVA (BARTHOLO, 2008).

Objetivos

Esta monografia tem como objetivo adaptar uma aplicação web, mais especificamente o protótipo de um AVA, para dispositivos móveis (especialmente celulares), utilizando a arquitetura M-AVA (BARTHOLO, 2008).

Como objetivos específicos do trabalho podem ser elencados os seguintes: investigar na literatura as principais técnicas relacionadas a adaptabilidade de software; estudar os fundamentos da plataforma J2ME (Java *Micro Edition*); desenvolver um protótipo de uma aplicação *Web* para dispositivos móveis com técnicas de adaptabilidade, seguindo a arquitetura M-AVA de Bartholo (2008); e sugerir melhorias na arquitetura utilizada para refinamento e aprimoramento da mesma.

Organização da Monografia

Este capítulo apresentou uma visão geral da proposta desse trabalho, destacando o contexto no qual a pesquisa está inserida, a motivação e justificativa, e também os objetivos a serem alcançados. Os outros capítulos estão estruturados da seguinte maneira:

No Capítulo 1 apresenta-se uma revisão bibliográfica sobre computação móvel, com o intuito de mostrar os conceitos básicos de J2ME e do protocolo *WAP* (*Wireless Application Protocol*).

No Capítulo 2 apresenta-se uma revisão bibliográfica sobre adaptabilidade de software, destacando a arquitetura M-AVA, que será utilizada neste trabalho

No Capítulo 3 apresentam-se detalhes sobre a implementação deste trabalho, tais como, diagrama de caso de uso, diagrama de classe, imagens e códigos ilustrando o funcionamento da aplicação móvel.

CAPÍTULO 1. COMPUTAÇÃO MÓVEL

1.1. Considerações Iniciais

Neste capítulo são apresentadas uma introdução a computação móvel e duas das tecnologias utilizadas para o desenvolvimento de aplicações móveis: WAP e J2ME.

1.2. Conceitos

A Computação Móvel está relacionada à mobilidade de software, hardware e dados (BARTHOLO, 2008 ; GIALDI, 2004), tendo por objetivo prover ao usuário acesso permanente a uma rede fixa ou móvel, independente de sua localização física, ampliando desta maneira o conceito de computação distribuída. Os dispositivos móveis são classificados em Assistente Digital Pessoal (PDA), *smart phones* (telefones inteligentes), telefones celulares, *laptops* e *notebooks*.

Salienta-se que nos últimos anos, as empresas vêm investindo maciçamente nas tecnologias para dispositivo sem fio. Como resultado, há várias tecnologias disponíveis para o desenvolvimento de aplicações móveis, como SMS (*Simple Message Service*), MMS (*Multimedia Message Service*), BREW (*Binary Runtime Enviroment for Wireless*) e J2ME (*Java* 2 *Micro Edition*) (Seção 1.6). Ainda há outras tecnologias como *WAP* (*Wireless Application Protocol*) (Seção 1.3) que em conjunto com o *WML (Wireless Markup Language)* (Seção 1.4), permite que o usuário visualize determinado conteúdo da web, como se estivesse em uma página web, só que seu conteúdo é formatado e otimizado especificamente para dispositivos móveis.

A infra-estrutura para computação móvel está relacionada a protocolos de comunicação e a plataformas de desenvolvimento (BARTHOLO, 2008), comentados a seguir.

- Protocolos de comunicação

- **Bluetooth**: é um padrão de comunicação proposto pelo Bluetooth SIG (*Special Interest Group*) que é um consórcio das maiores empresas de telecomunicação e computação do mundo.
- **IEEE 802.11:** é uma especificação para suportar a comunicação de redes locais sem fio, que padroniza as camadas físicas e de enlance para redes sem fio.

- Plataforma de desenvolvimento

- **WAP (***Wireless Application Protocol*): de acordo com MESQUISTA et al. (2007) é um protocolo para desenvolvimento de aplicações sem fio, cujo objetivo é a criação e manutenção de um padrão capaz de ser identificado e entendido por qualquer telefone celular, e/ou aplicação residente nesse dispositivo, independentemente do meio de transmissão e da codificação de sinal que seja utilizada.
- **J2ME** (Java 2 *Micro Edition*)**:** é uma plataforma Java que possui uma coleção de tecnologias e especificações desenvolvidas para utilização de dispositivos de computação móvel. Possui as mesmas características de robustez e portabilidade da linguagem de programação Java (DEITEL e DEITEL, 2005).

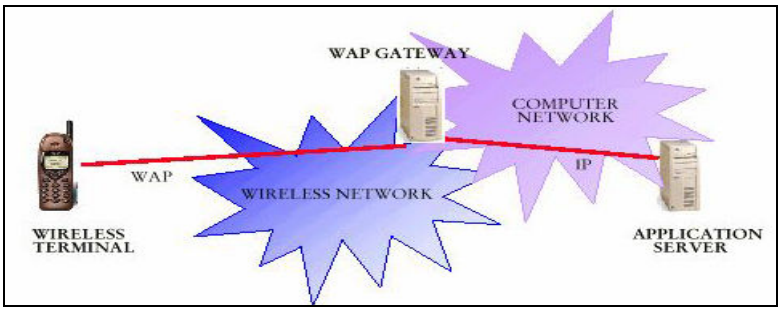
Neste trabalho são utilizadas as tecnologias WAP e J2ME, que estão detalhadas nas próximas seções para facilitar o entendimento das mesmas.

1.3. Protocolo WAP

WAP (*Wireless Application Protocol* ou Protocolo de Aplicação sem fio) é um protocolo desenvolvido para o mercado da Internet, utilizada em sistemas de comércio eletrônico. O WAP foi iniciado pela *Ericson, Nokia, Motorola* e Phone.com em 1997.

WAP foi construído com muita semelhança a Web, visando justamente se agregar a ela com o menor esforço possível. Em virtude dessas características semelhantes entre WAP e Web nota-se que as duas tecnologia tem muito em comum. WAP, igualmente a Web, é basicamente uma pilha de protocolos de comunicação, numa filosofia cliente/servidor, ou seja, o cliente faz uma requisição (request) de alguma informação para o servidor e este lhe responde (*response*) os dados requeridos. Na Figura 1 é ilustrado o protocolo WAP e o protocolo Web.

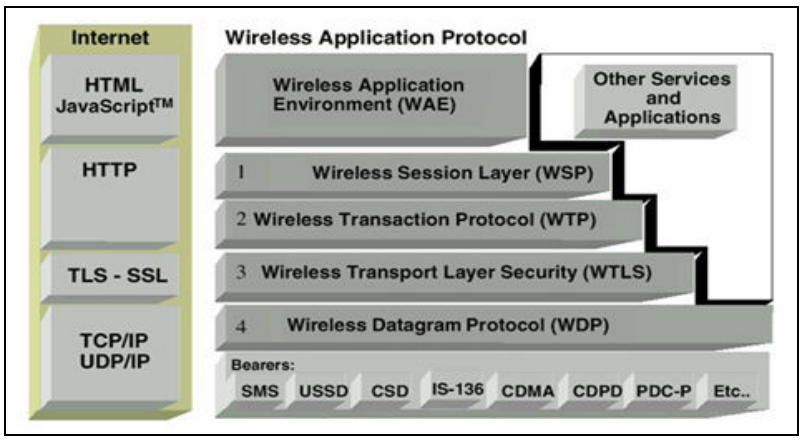
Figura 1 : WAP e Web (Junior, 2008)



1.3.1. Pilha de Protocolo WAP

A fim de minimizar a largura de banda requerida e garantir que a Internet sem fio pudesse rodar aplicação WAP normalmente, um novo protocolo foi desenvolvido. No entanto, muitas comparações entre a arquitetura TCP (*Transmission Control Protocol*)/IP (*Internet Protocol*) e WAP podem ser feitas. Na Figura 2 são ilustrados o protocolo WAP e uma comparação entre a arquitetura TCP/IP.

Figura 2 : Protocolo WAP (Junior, 2008)



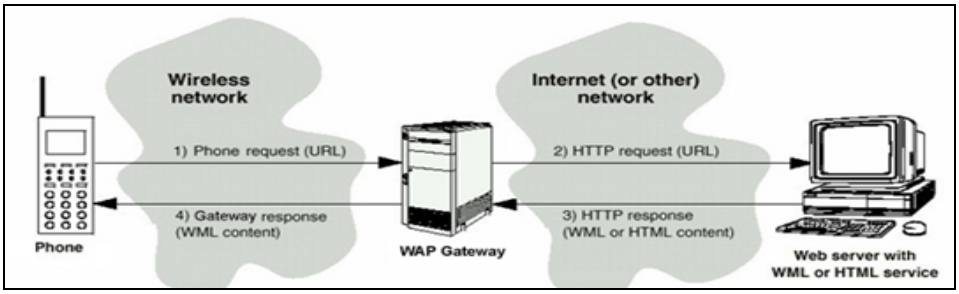
Analisando a Figura 2 observam-se os seguintes elementos:

- *Wireless Application Environment* (WAE): consiste na camada de aplicação do protocolo, estabelecendo padrões para os navegadores de páginas Web e para linguagens de renderização (WML – *Wireless Markup Language*). Na Internet isso é equivalente a especificação do HTML (*HyperText Markup Language*).
- *Wireless Application Protocol* (WAP): essa camada é dividida em quatro sub-camadas, citadas a seguir, e é a camada responsável pela requisição e transporte dos dados, nível requerido de segurança e controle transacional.

- WSP (*Wireless Session Protocol*): provê uma camada de sessão de peso leve, para permitir uma troca eficiente de dados entre as aplicações.
- WTP (*Wireless Transaction Protocol*): provê apoio de transação, acrescentando confiança ao serviço de datagrama provido pelo WDP.
- WTLS (*Wireless Transport Layer Security Specification*): camada de segurança opcional, em geral, utilizada em aplicação do tipo comércio eletrônico.
- WDP (*Wireless Datagram Protocol*): camada responsável por enviar e receber mensagens por das redes disponíveis.

A arquitetura do sistema que possibilita que o dispositivo sem fio (comumente um celular) se conecte com a Internet em si é trivial. Basicamente o dispositivo sem fio estabelece um circuito de troca de dados junto ao hardware da operadora do dispositivo sem fio. Feito isso, o dispositivo está pronto para fazer requisições à rede, no entanto, o dispositivo móvel não faz isso diretamente, mas sim através de um elemento que desempenhará o papel do *proxy* da conexão. Este elemento geralmente é um simples computador, conhecido como WAP *Gateway* e tem a função de fazer as requisições na Internet pelo dispositivo móvel. A Figura 3 ilustra-se o funcionamento de WAP *Gateway*.

Figura 3 : Funcionamento do WAP Gateway (Junior, 2008)



Como pode ser observado o WAP Gateway está entre o dispositivo móvel e o protocolo Web, enviando páginas de um para outro como se fosse um *proxy*. O WAP *Gateway*, quando recebe uma URL (*Uniform Resource Locator*), ou seja, o conteúdo pedido pelo celular, faz uma verificação de sintaxe no código WML recebido, verificando se não está mal formatado e se seu tamanho não ultrapassa a capacidade suportada pelo celular. Feito isso, o WAP *Gateway* compila tal código substituindo as tag's do WML por códigos binários que ocupam menos espaços para serem transportados na rede, otimizando assim a banda de passagem.

1.4. WML (*Wireless Markup Language*)

Web sites robustos e de boa visualização são criados utilizando HTML (*HyperText Markup Language*), CSS(*Cascading Style Sheels*) e *JavaScript.* No entanto, tais tecnologias não funcionam muito bem para dispositivos móveis, porque foram desenvolvidas para computadores com alto poder de processamento, que não é o caso de celulares, PDA(*Personal Digital Assistant*), dentre outros dispositivos móveis. Portanto, uma nova linguagem de marcação, denominada WML (*Wireless Markup Language*), foi criada respeitando todas essas restrições. WML possibilita um modelo de navegação para dispositivos com telas pequenas e sem facilidade para entrada de dados, como o é caso da ausência de um teclado e mouse em um celular. Assim como o HTML, o WML trabalha com *tags*, sendo algumas compatíveis com os tipos de linguagens de desenvolvimento, como é o caso da linguagem de *scripting* WMLScript

1.4.1 Exemplo básico de WML

A unidade básica do WML é o *card* que especifica uma simples interação entre o usuário e o *user agent*. Um *user agent* é o nome do browser que o dispositivo móvel suporta. Vários *cards* são agrupados juntos em decks, que representam a unidade importante de um arquivo WML. Na Figura 4 é ilustrado um exemplo básico de WML.

Figura 4 : Exemplo de codificação em WML

```
1.  <?xml version="1.0"?>
2.  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD
3.
4.  <wml>
5.      <card id="MainCard" title="Esse é o primeiro card">
6.          <p >
7.              Hello Word WML
8.          </p>
9.      </card>
10. </wml>
```

Quando visualizado em um emulador WAP, o mesmo interpreta as *tags* WML e transforma o código da figura 4 como ilustra na Figura 5.

Figura 5 : Código WML interpretado

1.4.2 Exemplo Wap e Servlets

Construir documentos WML dinamicamente utilizando *Servlets* pode ser uma tarefa fácil, uma vez que se conhece a sintaxe de WML. O código da Figura 6 tem o intuito de mostrar que é possível trabalhar com *servlet* e WAP e tem como objetivo mostrar a data corrente utilizando a classe `Date` do pacote `java.util`, que gera a data correspondente a do sistema operacional.

Figura 6 : Servlet e WAP

```
1.  import java.io.*;
2.  import java.text.DateFormat;
3.  import java.util.Calendar;
4.  import java.util.Locale;
5.  import javax.servlet.*;
6.  import javax.servlet.http.*;
7.
8.  public class MobileDate extends HttpServlet {
9.  public void service (HttpServletRequest request, HttpServletResponse
10. response) throws ServletException, IOException {
11.       // seta content type para dispositivo sem fio
12.       response.setContentType("text/vnd.wap.wml");
13.
14.       PrintWriter out = response.getWriter();
15.       // escreve dados.
16.       Locale local = new Locale("pt", "br");
17.       Calendar calendario = Calendar.getInstance(local);
18.       DateFormat dF = DateFormat.getDateInstance(DateFormat.FULL);
19.       String date = dF.format(calendario.getTime());
20.       out.println("<?xml version=\"1.0\"?>");
21.       out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
22.       out.println(" \"http://www.wapforum.org/DTD/wml_1.1.xml\">");
23.       out.println("<wml>");
24.       out.println("<card title=\"MobileDate\">");
25.       out.println(" <p >");
26.       out.println("Date and Time Service<br/>");
27.       out.println("Date is: "+ date);
28.       out.println("</p>");
29.       out.println("</card>");
30.       out.println("</wml>");
31.   }
32. }
```

É possível notar que o código da Figura 6 usa somente *tags* WML, com exceção das classes que são responsáveis por formatar a data corrente. No entanto, a parte mais importante a ser observada nesse código está apresentada na linha 12, ele configura o MIME TYPES (*Multipurpose Internet Mail Extensions*) para informar para o servidor, que o conteúdo é do tipo WAP, fazendo que o mesmo chame o interpretador de WAP. Quando esse *servlet* é invocado por um dispositivo móvel que tenha suporte para WAP, o mesmo mostrará a data corrente como mostrada na Figura 7. Para maiores informações sobre servlet consultar seção 2.5.1.

Figura 7 : Data Corrente utilizando Servlet e WAP

1.4.3 WAP E JSP (JavaServer Pages)

Java Server Pages (JSP) é uma tecnologia que permite aos desenvolvedores de programas Web "misturar" códigos HTML estáticos com Java. Tais códigos são executados no servidor, fazendo com que o servidor construa uma página dinâmica e retorne como resultado uma página criada dinamicamente para o cliente. Portanto, desenvolver aplicações utilizando WAP e JSP pode ser feita de maneira mais fácil do que *Servlet* e WAP. Na Figura 8 é ilustrado um simples exemplo de WAP e JSP que tem a mesma funcionalidade do código listado na Figura 6.

Figura 8 : JSP e WAP

```
1.  <?xml version="1.0"?>
2.  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
3.  "http://www.wapforum.org/DTD/wml_1.1.xml">
4.  <%
5.  response.setContentType("text/vnd.wap.wml");
6.  out.println("<wml>");
7.  out.println("<card title=\"MobileDate\">");
8.  out.println(" <p align=\"center\">");
9.  out.println("Date and Time Service<br/>");
10. out.println("Date is: "+ new java.util.Date());
11. out.println("</p>");
12. out.println("</card>");
13. out.println("</wml>");
14. %>
```

Observa-se que na linha de código 5, também é necessário configurar o MIME corretamente para fazer com que o dispositivo móvel posso interpretar o WML corretamente. Na Figura 10 é mostrada a execução do código listado acima.

Figura 9 : Data Corrente utilizando JSP e WAP

1.5. Linguagem Java

A linguagem de programação Java é uma linguagem orientada a objeto, desenvolvida na década de 70 pelo programador James Gosling, na empresa *Sun Microsystem*. Diferentemente das linguagens de programação convencionais, que são compiladas para um código nativo, a linguagem Java gera um tipo de arquivo chamado *bytecode*, que é interpretado por uma *Java Virtual Machine* (JVM). Para cada tipo de sistema operacional existe uma JVM específica que faz a ligação de um programa Java ao próprio sistema operacional. Java tem uma vantagem que sua sintaxe é muito parecida com C e C++, o que facilita o aprendizado para programadores dessas linguagens, adicionalmente possui mecanismo de segurança de execução e acesso de dados, pode ser usada em ambientes

distribuídos e em rede, é portável, a programação de conexão de rede é mais simples do que em outras linguagens de programação como C, e também permite a programação concorrente (MULTITHREADING) (HORSTMANN e CORNEL, 2000).

Em relação às outras linguagens de programação, Java eliminou a manipulação e alocação direta de memória, sendo que o coletor de lixo do Java se encarrega de gerenciar e coletar os dados que não são mais necessários no programa, também não requer índices de inserção e remoção de *array*, evitando acesso, manipulação ou escrita de dados sobre outros *arrays* (HORSTMANN e CORNEL, 2000).

Uma classe em Java é como se fosse um molde para se criar objetos, contendo métodos e atributos que servem para definir o comportamento do objeto. Cada objeto pode ter um comportamento diferente de um outro objeto, até mesmo se eles forem definidos na mesma classe, pois no mesmo momento eles podem ter diferentes valores nos seus atributos. Java também provê encapsulamento de dados, que é uma das características de uma linguagem orientada a objetos. Encapsulamento de dados consiste em esconder os dados do usuário de um objeto, fazendo com que o usuário conheça apenas os métodos necessários para mudar o comportamento do objeto. Esta é uma propriedade que também garante segurança na modificação do objeto e é uma boa prática de Engenharia de Software, pois os métodos só podem alterar os atributos do mesmo objeto em que estão inseridos (DEITEL E DEITEL, 2003).

Herança e polimorfismo são outras características importantes do paradigma orientado a objetos e presentes na linguagem Java. Herança é a capacidade de estender uma outra classe, para prover o reuso de métodos e atributos. A classe que herda de outra classe pode ser chamada de sub-classe, classe filha, ou classe derivada, enquanto que a classe que provê a herança é chamada de super-classe, classe pai, ou ainda classe base. Se nenhuma super-classe for explicitamente definida, a classe *Object* é assumida como tal, por padrão, sendo a única classe em Java que não herda de nenhuma outra classe (DEITEL e DEITEL, 2003). Em Java cada classe não pode ter mais que uma super-classe pois a linguagem Java não permite herança múltipla, mas é possível implementá-la por meio de uma interface.

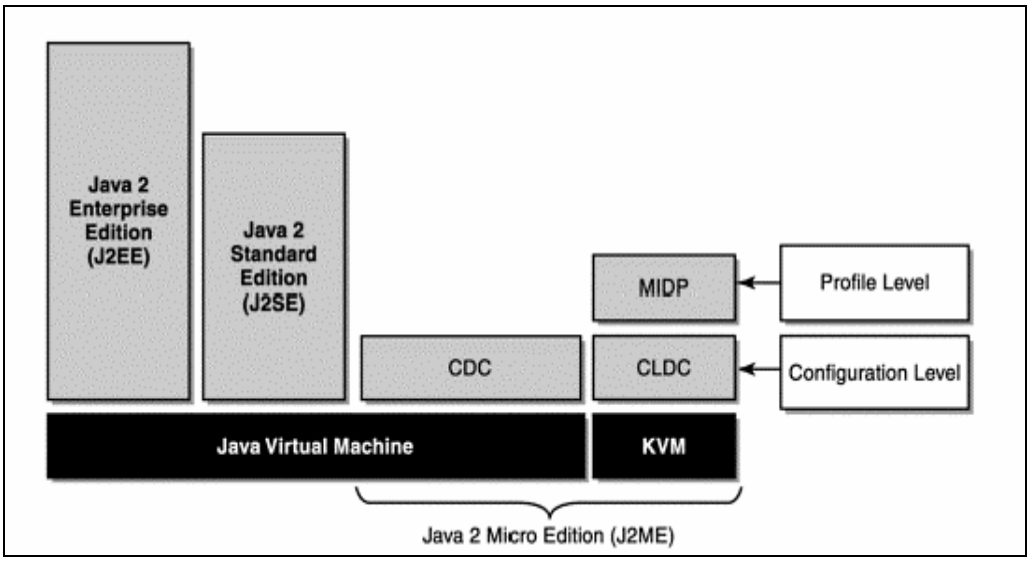
Interface é como se fosse uma classe abstrata Java com todos os métodos abstratos, ou seja, sem implementação. Uma interface apenas diz o que faz, mas não como faz (HORSTMANN e CORNEL, 2000). Uma classe pode estender somente uma classe através da palavra-chave *extends*, mas pode implementar nenhuma, uma ou várias interface através da palavra-chave *implements*, no entanto deve fornecer implementação para todos os métodos definidos nas interfaces. Assim, uma classe pode estender uma única classe e implementar

várias interface, já uma interface não pode estender nenhuma classe, mas pode estender quantas interfaces quiser.

Polimorfismo é a capacidade de um objeto ser representado por diversas formas, Poli (muitas) Morfismo (formas). Segundo Deitel (2003, p.374), "herança e polimorfismo são duas tecnologias fundamentais que permitem a verdadeira programação orientada a objetos."

A linguagem Java possui diferentes plataformas para propósitos específicos que estão ilustradas na Figura 11 e comentadas a seguir.

Figura 10 : Edições da plataforma Java 2 (MUCHOW, 2001)



• STANDARD EDITION (J2SE): foi a primeira de todas as plataformas desenvolvidas em Java, projetada para execução em máquinas simples de computadores pessoais e estações de trabalho, é a versão básica do Java.

• ENTERPRISE EDITION (J2EE): destinada para aplicações de grande porte, a qual faz extenso uso de EJB (*Enterprise Java Beans*). Tem suporte interno para *Servlets*, JSP, XML e é destinada a aplicativos baseados em servidor.

• MICRO EDITION (J2ME): projetada para dispositivos com memória, vídeo e poder de processamento limitado. Essa plataforma é de interesse deste trabalho e está detalhada na Seção 2.6.

Máquinas Virtuais Java ou JVM (*Java Virtual Machine*) como é mais conhecida é um mecanismo que permite executar códigos Java em qualquer plataforma de sistema operacional. A JVM transforma os arquivos de classes em um código de máquina para a plataforma que está executando a JVM, fazendo com que os programas escritos em Java

executem no sistema operacional nativo. Ela também é responsável por fornecer segurança, alocar ou não alocar memória e gerenciar *thread*.

Nas Seções 2.5.1 e 2.5.2 apresentam tecnologias da linguagem Java voltadas para o desenvolvimento de aplicações Web e que podem ser utilizadas em conjunto com WAP.

1.5.1 *Servlets*

Servlets são códigos Java que rodam em uma aplicação servidor, mas não possuem o método `main` que aplicações Java costumam ter, eles estão sob o controle de outro aplicação Java chamada de contêiner (DEITEL e DEITEL, 2005), por exemplo, o Tomcat (DEITEL e DEITEL, 2005). Com o contêiner os desenvolvedores de aplicações Web não tem que se preocupar com tarefas do tipo: (BASHAM, 2008)

- Suporte para comunicação: o contêiner faz como que os *servlets* se comuniquem de maneira fácil com o servidor Web, fazendo com que o desenvolvedor não precise se preocupar em escutar eventos em uma respectiva porta. O próprio contêiner sabe o protocolo usado entre o servidor Web e o contêiner.
- Gerenciamento ciclo de vida: o contêiner controla a vida e a morte de todos os *servlets* hospedados no mesmo. Ele também é responsável por carregar as classes, instanciar e inicializar os *servlets*, invocar métodos do *servlets* e fazer os *servlets* serem eleitos para o coletor de lixo (*garbage coletor*).
- Suporte a *multithreading*: o contêiner automaticamente cria uma nova *thread* para cada solicitação que ele recebe.

Embora os *servlets* possam responder a qualquer tipo de requisição, eles são comumente usados para ampliar as aplicações hospedadas em servidores Web (DEITEL e DEITEL, 2005). Para tais aplicações a tecnologia Java *servlets* define classes *servlets* específicas para o protocolo HTTP.

Segundo Deitel e Deitel (2005), um *servlet* estende a funcionalidade de um servidor, como um servidor Web que serve páginas da Web para um navegador do usuário usando o protocolo HTTP. Os pacotes `javax.servlet` e `javax.servlet.Http` fornecem classes e interfaces para definir os *servlets* . Todos os *servlets* devem implementar a interface `Servlets` do pacote `javax.servlet`. Os métodos da interface `Servlets` são invocados pelo contêiner (DEITEL e DEITEL, 2005).

De acordo com Deitel e Deitel (2005), os pacotes de *servlets* definem duas classes abstratas que implementam a interface `Servlets`, são a classe `GenericServlets` (do pacote `javax.servlet`) e a classe `Http*Servlets*` (do pacote `javax.servlet.Http`). Pode se observar que a classe `GenericServlets` é um *servlet* independente de protocolos, já o `HttpServlets` como o próprio nome diz utiliza o protocolo HTTP para fazer a troca de informações entre o servidor e o cliente.

Na maioria das vezes os *servlets* estendem a classe `Http*Servlets*` , que sobrescreve o método `service` para fazer uma distinção entre as solicitações típicas recebidas de um navegador WEB do cliente. A classe `HttpServlets` define métodos como `doPost`, `doGet` para responder a solicitação típicas post e get de um cliente, respectivamente. Segundo Deitel e Deitel (2005) esses métodos são chamados pelo método `service`, que por sua vez é chamado pelo contêiner quando a solicitação chega ao servidor.

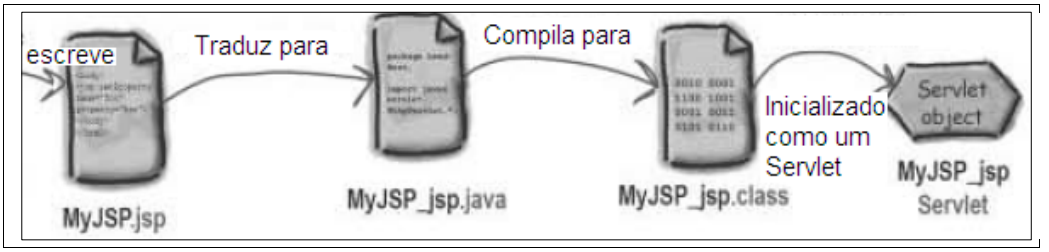
Para fazer uma solicitação (request) ou responder (response) de volta para o cliente há duas interfaces para fazer o respectivo serviço que são `HttpServletsRequest` e `HttpServletsResponse`, esses objetos são criados quando o contêiner vê que a solicitação é para uma *servlet* criando assim os dois objetos.

1.5.2 JSP (*JavaServer Pages*)

JavaServer Pages é o significado da sigla JSP, e é uma tecnologia orientada a criar páginas Web com o auxílio da linguagem de programação Java (DEITEL e DEITEL, 2005).

Segundo BASHAM *et al.* (2008), um JSP frequentemente tornará um *servlet* rodando em uma aplicação Web. JSP é muito parecido com qualquer outro *servlet*, exceto que o *servlet* é criado pelo desenvolvedor, já no caso de JSP o próprio contêiner gera um respectivo *servlet* para o JSP. O contêiner obtém o que está escrito em JSP, traduz em arquivo Java (.java) que é um *servlet*, em seguida compila o arquivo gerando o *bytecode* (.class). Na Figura 14 é ilustrado o que o contêiner faz com um arquivo JSP (BASHAM, 2008).

Figura 11 : Tradução JSP para Servlet (BASHAM, 2008)



JSP foi desenvolvido para ajudar os projetistas que criam páginas em HTML, já que nem todos conhecem a tecnologia Java, fazendo com que em uma empresa seja possível separar as atividades, ou seja, os desenvolvedores Java escrevem o código Java, enquanto que os desenvolvedores HTML fazem as páginas Web.

1.6. JAVA MICRO EDITION – J2ME

A plataforma *Java Micro Edition* ou J2ME, como é mais conhecida, é destinada diretamente a dispositivos de pequeno porte, tais como telefones celulares, PDAs, comutadores, roteadores de redes, componentes para automação residencial, televisores digital de nova geração, etc. Foi apresentada em meados de 1999 numa conferência JavaOne promovida pela *Sun MicroSystems* (MUCHOW, 2001).

Com a possibilidade de utilizar Java em tais dispositivos, tem-se acesso aos recursos inerentes da linguagem e, consequentemente, várias vantagens em relação às demais tecnologias *wireless*, como portabilidade, ou seja, a capacidade de desenvolver aplicações que executem em qualquer dispositivos, não se preocupando em saber quem é o fabricante do dispositivo "*Write Once, Run AnyWhere*™" (Escreva uma vez, execute em qualquer lugar).

Como a diversidade de modelos de dispositivos móveis é muito grande, eles foram divididos em configurações diferentes, tais como CDC (Configuração de Dispositivo Conectado) e CLDC (Configuração de Dispositivo Conectado Limitado), descrita na Seção 2.6.1 pois é de interesse deste trabalho. A CDC tem um conjunto de APIs para suportar dispositivos "fixos", como um computador ligado à televisão, já a CLDC tem um conjunto de APIs para dar suporte para dispositivos com poder de processamento, vídeo e memória limitados.

A configuração CDC foi projetada para dispositivos que possuem mais memória, cerca de 2 MB de RAM e 2.5 MB de ROM, a JVM para a CDC tem a mesma especificação da J2SE, conforme ilustrado na Figura 10. Por outro lado, a configuração CLDC foi projetada

para dispositivos que tem carência de memória, tipicamente cerca de 128 KB a 512 KB. Para solucionar o problema de carência de memória, a Sun desenvolveu a *K Virtual Machine* (máquina virtual K).

É importante salientar que quando se trata de CLDC, a máquina virtual não é a mesma que a CDC, pois CLDC tem o poder de processamento muito limitado como já foi comentado.

Cada configuração de dispositivo móvel consiste em uma maneira diferente de definição de característica da linguagem em relação ao hardware (quanto ao uso de memória, recurso da tela, conectividade com redes, poder de processamento e suas limitações) e de bibliotecas para o Java (MUCHOW, 2001).

1.6.1. *Connected Limited Device Configuration* (CLDC)

De acordo com (MUCHOW, 2001), a CLDC tem dois objetivos: o primeiro é definir uma especificação para uma JVM que é a KVM, um tipo de *Java Virtual Machine* para J2ME, e o segundo é definir um conjunto de classes (bibliotecas) Java para possibilitar ao programador fazer a iteração com o dispositivo móvel . Esses dois objetivos têm uma coisa em comum que é suportar uma ampla variedade de dispositivos com memória, capacidade de vídeo e recursos limitados. A CLDC define em sua API os pacotes `java.lang`, `java.util` e `javax.microedition.io` (conexão e interfaces), e não tem suporte a ponto flutuante como as outras plataformas Java. MIDP é um perfil, descrito na Seção 1.6.1.1, responsável por definir todos os pacotes da CLDC, incluindo `javax.microedition.lcdui` (interface com o usuário), `javax.microedition.rms` (sistema de gerência de registros para persistência das informações), `javax.microedition.midlet` (suporte para aplicações MIDP, conhecidos como MIDLETS, apresentados na Seção 1.6.1.2.).

Os requisitos mínimos de memória que devem ter os dispositivos móveis para utilizar a CLDC são os seguintes:

- 128 Kilobytes de memória para executar a JVM e as bibliotecas CLDC: essa memória deve preservar seu conteúdo, mesmo quando o dispositivo for desligado, ou seja, uma memória não volátil.
- 32 Kilobytes de memória disponível durante o tempo de execução do aplicativo para alocação de objetos: essa memória é frequentemente referida como memória volátil ou "*heap*".

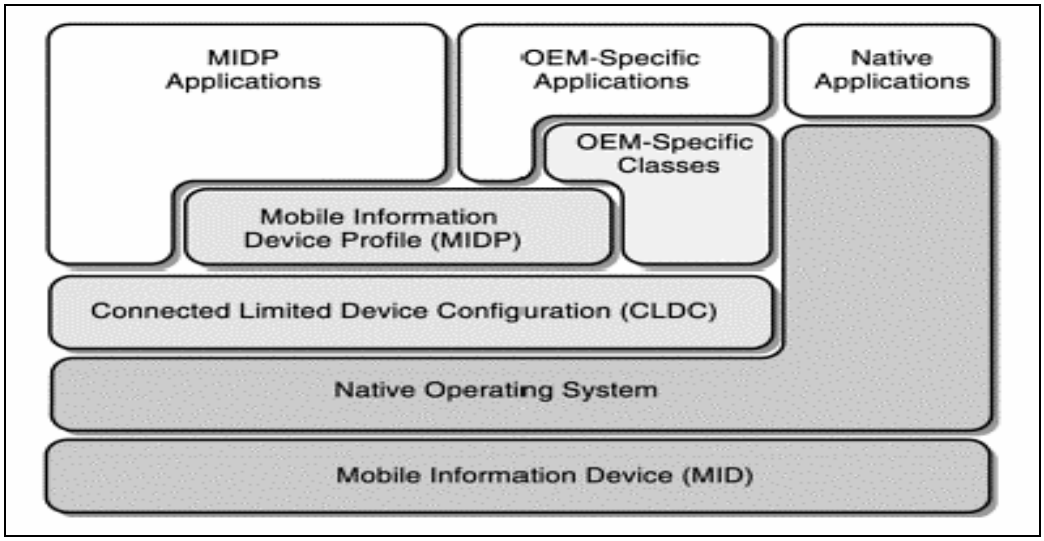
Em relação aos requisitos mínimos de software para utilizar a CLDC, o sistema operacional do aplicativo móvel deve ser capaz de executar a JVM e gerenciar aplicativos Java de tal maneira que seja capaz de remover aplicativos Java dos dispositivos, bem como selecionar a ativar aplicativos.

Uma coisa muito interessante sobre os requisitos de software é que a CLDC não tem que se preocupar com a implementação desses recursos, deixando toda a implementação para o fabricante do dispositivo.

1.6.1.1. MIDP

Mobile Information Device Profile (Perfil de Dispositivo de Informação Móvel) é um perfil definido pela Sun para a configuração CLDC, destinado para dispositivos móveis com restrição de memória, tela pequena, sem fio, como é o caso do tradicional telefone celular. Com esse perfil qualquer desenvolvedor de aplicações pode modelar um sistema e ter a certeza (garantia) de que a sua aplicação móvel será executada pelos dispositivos que implementam esse perfil, independente do fabricante. A idéia é a mesma quando aplicações são desenvolvidas para um determinado sistema operacional, sendo assim não tendo a preocupação sobre o fabricante do computador que irá executá-lo. Na Figura 12 é apresentada a arquitetura do MIDP.

Figura 12 : Arquitetura de dispositivos de informação móvel. (MUCHOW, 2001)



Observa-se que no nível inferior da Figura 12 está o hardware, logo acima do hardware está o sistema operacional nativo do dispositivo móvel, no canto superior direito da

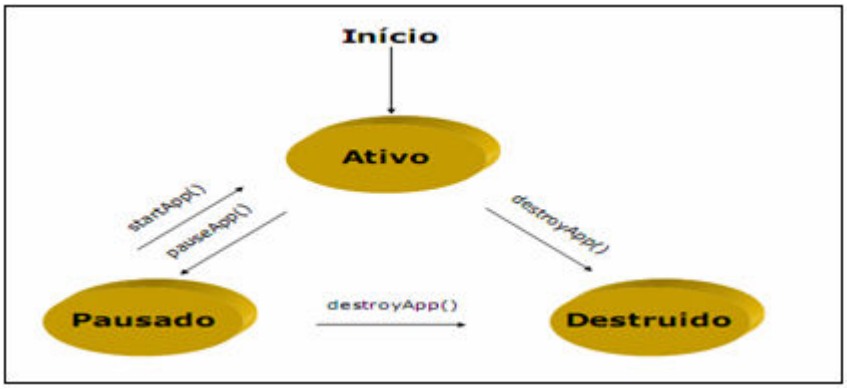
figura tem-se uma aplicação nativa de um dispositivo móvel, por exemplo, o programa de configuração que vem instalado no dispositivo, permitindo que o usuário configure o tipo de toque, o volume, a data, a hora, etc. As classes específicas de OEM (*Original Equipment Manufacturer*) são fornecidas pelos fabricantes de dispositivos móveis. O inconveniente é que essas classes são específicas a todos os dispositivos, fazendo com que se perca a portabilidade. Observa-se que a CLDC é instalada sob o sistema operacional nativo e é a base da MIDP, adicionalmente, os aplicativos MIDP têm acessos às bibliotecas tanto da CLDC quanto da MIDP.

1.6.1.2. Midlets

Uma MIDlet é uma aplicação MIDP, quem tem acesso tanto as bibliotecas da CLDC quanto as bibliotecas do MIDP. Um conjunto de MIDlets consiste em uma ou mais MIDlets empacotadas, usando um arquivo JAR. Em Java cria-se um MIDlet usando uma classe abstrata chamada `MIDlet`. Esta classe oferece três métodos abstratos que são usados pela aplicação para gerenciar a MIDlet: `startApp`, `pauseApp` e `destroyApp`.

Segundo Martins (2003), o método `startApp` é chamado imediatamente depois do construtor e cada vez que um aplicativo é ativado. Isto quer dizer que este método não é chamado somente quando o aplicativo é inicializado, como acontece no método `main` em uma aplicação J2SE. O método `pauseApp` é chamado para fazer uma notificação de que a MIDlet está para ser colocada no estado de pausa e o método `destroyApp` é chamado para sinalizar que a MIDlet está para ser desligada, todos os recursos que estão em uso pelo MIDlet precisam ser liberados nessa hora. Esse método também pode ser chamado pela própria MIDLet para iniciar o desligamento. Na Figura 13 é ilustrado o ciclo de vida de um MIDlet.

Figura 13 : Ciclo de vida MIDlet



Note que o um MIDlet pode ir do estado ativo e pausado inúmeras vezes em uma aplicação, deste modo não é aconselhável colocar código de inicialização no método `startApp`, no entanto, uma vez que o MIDlet foi para o estado destruído o mesmo não tem como retornar para o estado ativo ou pausado (MARTINS, 2003).

1.7. Considerações Finais

Neste capítulo foram apresentadas as tecnologias possíveis de serem utilizadas no desenvolvimento de aplicações para dispositivos móveis, entre elas a tecnologia WAP e a J2ME, ambas trabalhando com o Java. WAP trabalhando principalmente em conjunto com JSP, visto que seu foco é programação Web. Portanto foi dado um enfoque maior à tecnologia Java e todos os recursos necessários para a criação de aplicação móveis por meio desta tecnologia.

No próximo capítulo são apresentados os conceitos de adaptabilidade de software e possíveis técnicas que podem ser empregadas para isso. Será apresentado também a arquitetura M-AVA que foi selecionada para ser utilizada e refinada nesta monografia.

CAPÍTULO 2. ADAPTABILIDADE DE SOFTWARE

2.1. Considerações Iniciais

Neste capítulo são apresentados conceitos sobre adaptabilidade de software, as possíveis técnicas a serem utilizadas para o desenvolvimento de um software adaptativo para dispositivos móveis, além da arquitetura M-AVA que é a arquitetura escolhida utilizada neste trabalho para adaptar aplicações Web para o contexto de dispositivos móveis.

2.2. Software Adaptativo

O interesse por sistemas com requisitos de adaptabilidade tem crescido muito nos últimos anos e atualmente uma variedade de técnicas permitem que o software adapte-se dinamicamente de acordo com seu ambiente de execução. Segundo Mckinley (2004) duas revoluções na área de computação são consideradas principais responsáveis pelo crescimento dessa abordagem de desenvolvimento. A primeira é o surgimento da computação ubíqua que promete mudar a forma tradicional de como, quando e onde o ser humano interage com os computadores. A segunda é a crescente demanda pela computação autônoma que se constitui no desenvolvimento de sistemas que são auto-configuráveis, auto-otimizados, auto-monitorados e auto-protegidos, precisando de instruções mínimas e de alto nível (KEPHART, 2003).

Segundo Dantas e Borba (2003), o termo adaptabilidade tem sido usado para referenciar sistemas de software adaptável e adaptativo. Freqüentemente as palavras adaptável e adaptativo têm sido usadas de maneira intercambiável, no entanto, quando são usadas juntas elas podem apresentar diferentes significados.

Sistema adaptativo é um sistema capaz de mudar seu comportamento automaticamente, durante sua execução, de acordo com mudanças no contexto do ambiente em que se encontra, e sistema adaptável é um sistema que permite adaptar facilmente uma estrutura completa ou partes específicas devido a mudança no requisito (SOUZA, 2004; KULESZA, 2006; DANTAS e BORBA, 2003).

De acordo com Loureiro (2006), adaptação significa dizer que uma aplicação ou algoritmo não tem agora uma única especificação de saída, mas possivelmente um conjunto válido de saídas ou resultados que são aceitáveis em função das condições existentes em um determinado momento do tempo. Por exemplo, se um dado momento só existe energia

disponível na unidade móvel para mais ***t*** segundos, então não adianta executar um processamento que gaste mais que esse tempo, a menos que essa computação possa chegar a um resultado com uma "precisão menor" em menos de ***t*** segundos (LOUREIRO, 2006).

Outro exemplo de adaptação que pode ser ilustrado é quando em um dado instante um usuário móvel está acessando as suas mensagens eletrônicas em um ambiente externo de baixa velocidade com uma "alta" taxa de erro. Neste cenário pode-se optar por mostrar somente o remetente e o assunto de cada mensagem. Suponha ainda que o usuário continue a se mover e passa a acessar uma infra-estrutura interna de comunicação sem fio que possui uma taxa de comunicação acima de um Mbps e uma "baixa" taxa de erro, agora o usuário poderia passar a ver as mensagens integralmente. Segundo Loureiro (2006), adaptação pode ser feita tanto no dado que está sendo transmitido para a unidade móvel quanto no próprio processamento solicitado pelo usuário, o tipo de adaptação que deve ver feita depende das condições e tipo do ambiente móvel.

Conforme Loureiro (2006), um conceito ligado com adaptação em computação móvel é o de QoS *(Quality of Service)*. QoS define características não funcionais de um sistema que afetam a qualidade percebida dos resultados. Por exemplo, numa aplicação multimídia é importante para o usuário a resolução de uma imagem, taxa de quadros por segundos e a qualidade do áudio. No ambiente móvel há uma grande variação de QoS que deve ser tratada pelo desenvolvedor da aplicação.(LOUREIRO, 2006).

Outro conceito importante relacionado com adaptação em computação móvel é o da usabilidade, que está relacionada com a facilidade do usuário se interagir com a interface de uma aplicação móvel. Usabilidade está tradicionalmente associada a quatro atributos (NIELSEN, 1993): facilidade de aprendizagem, eficiência de uso, minimização de erros e satisfação.

2.3. Técnicas de Adaptação

Aplicações móveis possuem uma baixa largura de banda e instabilidade (muita variação), redes heterogêneas, riscos de segurança, baixa autonomia de energia e pouca capacidade de armazenamento dos dispositivos móveis, além de interfaces pequenas devido ao tamanho dos dispositivos móveis, causando assim dificuldade na interação do usuário com a aplicação móvel e, portanto, obrigando a utilização de técnicas de adaptação de interface e de conteúdo para melhorar tais tipos de aplicações e, conseqüentemente, a interação do

usuário. De acordo com Loureiro (2006), se uma interface é ineficaz, as funcionalidades e a utilidade do sistema ficam limitadas, os usuários tornam-se confusos, frustrados e irritados.

De acordo com Loureiro (2006), a adaptação do ponto de vista do sistema pode ser feita por meio da criação e utilização de novos protocolos específicos para a comunicação móvel. Sendo que quando as condições da rede mudam, o sistema pode trocar dinamicamente para um novo protocolo. Outras técnicas de adaptabilidade são: aumento de compressão, que é aplicada aos dados antes da transmissão e a utilização de pré-busca e cache durante os períodos de alta conectividade como preparação para futuras reduções na largura de banda.

De acordo com Bartholo (2008 apud CARVALHO, 2005) a adaptação pode ocorrer em tempo de construção ou em tempo de execução. Na **adaptação em tempo de execução**, a aplicação adaptável modifica seu comportamento no tempo de execução de acordo com o contexto, buscando atender as necessidades do ambiente, sem a intervenção do engenheiro de software. A **adaptação em tempo de construção** ocorre quando uma aplicação é transformada, ou desenvolvida, para se adequar à mudança de um ambiente antes de ser executa. Esse tipo de adaptação exige a intervenção do engenheiro de software e é caracterizada pela construção de novas versões da aplicação.

A adaptação pode ser relacionada ao conteúdo e a interface do usuário (BARTHOLO, 2008), conforme descritas a seguir:

i) adaptação de conteúdo: a necessidade da adaptação de conteúdo está relacionada a de conteúdos ser destinada principalmente para PCs, dificultando a visualização destes em dispositivos móveis. Devido a isso, é necessário adaptar os conteúdos de acordo com os recursos disponíveis em cada dispositivo móvel (BARTHOLO, 2008 apud SANTANA *et. al.*, 2007). Segundo consta no trabalho de Bartholo (2008) uma adaptação de conteúdo pode ser realizada:

- no servidor de origem: permite o fornecimento de conteúdos já adaptados aos dispositivos dos usuários, mas obriga o servidor a conter várias versões de um mesmo conteúdo;
- no dispositivo de usuário: permite o desenvolvimento de processos específicos de adaptação em função dos recursos disponíveis no dispositivo, mas sobrecarregando-o com tais processos;
- em um servidor de adaptação dedicado: alivia o servidor de origem e o dispositivo de usuário dos processos de adaptação, mas acrescenta um *overhead* principalmente em relação à comunicação.

ii) adaptação da interface: de acordo com Bartholo (2008) pode ser empregada de duas formas: estática e dinâmica:

- adaptação estática da interface do usuário**:** é realizada durante o desenvolvimento do software e neste tipo de adaptação o conceito de adaptabilidade está relacionada à manutenção, modificação e expansão do sistema (BARTHOLO, 2008 apud ITO,2007).
- adaptação dinâmica da interface do usuário: ocorre durante o tempo de execução do software. É considerada mais flexível, possibilitando ao desenvolvedor a modificação de código em tempo de execução. Segundo Menkhaus (2002), esse tipo de adaptabilidade pode ser originada de três maneiras diferentes:
    - mudanças no perfil do usuário: requer uma adaptação da interface do usuário de acordo com seus objetivos, partindo do pressuposto da forma de utilização da aplicação;
    - mudanças de funcionalidade: requer mudanças da interface, ou seja, nos formulários e nos componentes de interação para um conjunto de funcionalidades específicas;
    - mudanças de contexto do sistema: refere-se a mudanças relacionadas a aspectos tecnológicos do ambiente onde será instalado o software;

Conforme consta no trabalho de Bartholo (2008) adaptação da interface do usuário durante a execução do software pode ter diferentes dimensões. O primeiro fator se refere as funcionalidades de cada interface como, por exemplo, layout ou modelo de comunicação e o segundo fator está relacionado ao o tempo no qual ela é adaptada.

2.4. Arquitetura de Adaptação proposta por Bartholo (2008)

Bartholo (2008) propõe a arquitetura M-AVA (AVA Móvel) que busca estabelecer adaptabilidade de AVAs para dispositivos móveis, partindo da criação de uma interface genérica para ser utilizada por dispositivos móveis e atendendo desta forma diversos usuários de uma mesma aplicação, considerando o tipo de dispositivo móvel em tempo de execução.

Segundo afirma Bartholo (2008), a adaptação proposta pela Arquitetura M-AVA ocorrerá em tempo de construção e é caracterizada pela construção de novas versões da aplicação para diferentes tipos de dispositivos. A arquitetura M-AVA proverá adaptação de conteúdo e adaptação de interface do usuário, comentadas a seguir.

De acordo com a proposta de Bartholo (2008), a adaptação de conteúdo da arquitetura M-AVA é realizada no servidor web, permitindo o fornecimento de conteúdos (dados) já adaptados aos dispositivos dos usuários, obrigando que o servidor contenha informações de conteúdo disponível para PCs e para dispositivos móveis, a nível de banco de dados. Assim, é necessário haver no banco de dados campos com informações resumidas para serem apresentadas nos dispositivos móveis (BARTHOLO, 2008). Exemplificando, tendo a tabela TAREFAS (Quadro 1), que armazena informações das tarefas que são indicadas pelos professores e postadas em um AVA e que devem ser realizadas pelos alunos, com os seguintes campos: `nome`, `descrição`, `dataentrada`, `dataentrega`, `curso`. Observa-se que foi adicionado um campo na tabela com tamanho reduzido, denominado `descrição_resumida`, cujo conteúdo será utilizado para a exibição de informações no dispositivo móvel sobre o curso relacionado à tarefa postada (BARTHOLO, 2008).

Quadro 1: Tabela TAREFAS (extraído de Bartholo, 2008)

TAREFAS
nome: texto(20)
descrição: texto(100)
descrição_resumida: texto(25)
dataentrada: data
dataentrega: data
curso: numérico

A arquitetura M-AVA utiliza a adaptação de interface do usuário estática e dinâmica. Conforme comentado anteriormente, a primeira é alcançada durante o desenvolvimento do *software* pela metodologia de desenvolvimento de uma interface diferente de AVAs para dispositivos móveis e AVAs para desktop, sendo estática a partir do momento que são estabelecidas e criadas interfaces diferentes devido ao tamanho da tela de cada tipo de dispositivo. O segundo tipo de adaptação é alcançado em tempo de execução do *software* (AVA), classificado de adaptação por seleção, que consiste na seleção de um modelo da interface dentro de um conjunto de implementações existentes (MENKHAUS, 2002), proporcionada pelo serviço de identificação do dispositivo.

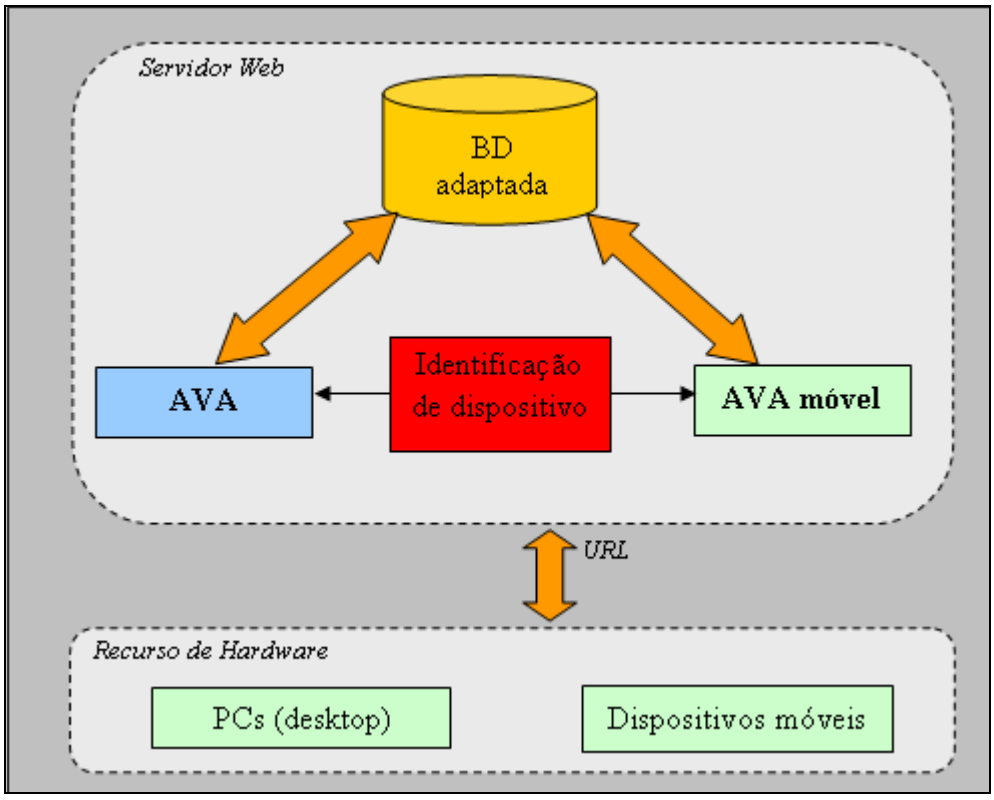
Assim, uma vez que a arquitetura M-AVA propõe adaptabilidade no contexto de conteúdo e de interface do usuário pode-se dizer que é uma arquitetura híbrida (BARTHOLO,

2008), sendo uma vantagem em relação a outras técnicas, abordagens e arquiteturas de adaptabilidade propostas na literatura.

Na arquitetura M-AVA, apresentada na Figura 14, tanto o AVA móvel quanto o AVA para *desktop* devem estar disponíveis no servidor web, em conjunto com a respectiva base de dados (uma única para ambas aplicações, ou seja, web e móvel). Os usuários devem acessar o AVA (independente de recurso de hardware, isto é, independente do tipo de dispositivo) por meio de uma URL única (BARTHOLO, 2008).

Durante o acesso, identifica-se o dispositivo, sendo que se o acesso estiver sendo efetuado por meio de um dispositivo móvel então o AVA móvel disponível no servidor web será executado, caso contrário será executado o AVA para *desktop*. Cabe ressaltar que o acesso ao conteúdo dinâmico de cada AVA é o mesmo, uma vez que eles possuem acesso a mesma base de dados que deve estar adaptada em relação ao conteúdo (BARTHOLO, 2008).

Figura 14 : Arquitetura M-AVA (extraída de Bartholo, 2008)



2.5. Considerações Finais

Neste capítulo foi apresentado o conceito de adaptabilidade e as possíveis técnicas utilizadas para prover adaptabilidade, objetivando um embasamento teórico para o desenvolvimento de aplicações com adaptabilidade.

Sistema adaptável é um sistema que permite adaptar facilmente uma estrutura completa ou partes específicas devido a mudanças nos requisitos e sistema adaptativo é o sistema capaz de mudar seu comportamento automaticamente, durante sua execução, de acordo com mudanças no contexto do ambiente em que está inserido (SOUZA, 2004; KULESKA, 2006; DANTAS e BORBA, 2003).

No contexto deste trabalho são considerados os sistemas adaptativos, no qual modifica seu próprio comportamento com respeito a mudanças em seu ambiente, uma vez que AVAs são adaptados de acordo com a mudança do recurso, no caso de computadores pessoais para dispositivos móveis, conforme proposto na arquitetura M-AVA de Bartholo (2008), que tem como objetivo adaptar AVAs para dispositivos móveis considerando tanto a adaptação de conteúdo quanto a de interface, além de considerar adaptação dinâmica e estática. No próximo capítulo será apresentado um estudo de caso conduzido que visa implementar a arquitetura M-AVA.

CAPÍTULO 3. ESTUDO DE CASO: UMA IMPLEMENTAÇÃO DA ARQUITETURA M-AVA

3.1. Considerações Iniciais

Este capítulo tem como objetivo apresentar o que foi desenvolvido neste trabalho, ou seja, uma implementação da arquitetura M-AVA. Na seção 3.2 descreve-se detalhadamente o estudo de caso conduzido. Na Seção 3.3 apresenta a modelagem conceitual do AVA. Na seção 3.4 descreve-se detalhadamente como a aplicação foi implementada, tanto o AVA-WEB, e na seção 3.5 e 3.6 será explicado a aplicação móvel off-line e a aplicação móvel on-line.

3.2. Descrição do Estudo de Caso

Segundo Meirelles et al, (2004), serviços de telecomunicação e de artefatos computacionais, capazes de prover mobilidade aos diferentes participantes de projeto educacionais, apresentam a oportunidade para o desenvolvimento de pesquisas no campo da computação móvel aplicada à educação.

Este estudo de caso tem por objetivo construir uma aplicação Web, mais precisamente um ambiente virtual de aprendizagem, buscando prover a adaptabilidade para dispositivos móveis. Para isso, a arquitetura proposta por Bartholo (2008), apresentada na Seção 2.4., foi analisada. Em seguida, desenvolveu-se um protótipo de um Ambiente Virtual de Aprendizagem (AVA) para WEB. O desenvolvimento do AVA Web foi baseado nas funcionalidades estabelecidas por Bartholo (2008) e contidas no Apêndice A.

Sucintamente o AVA Web deve cadastrar aluno, cadastrar professor, cadastrar curso, enviar matérias para um respectivo curso, adicionar tarefas, sendo que quando um professor adicionar uma tarefa para um determinado curso, o AVA enviará um e-mail para todos os alunos matriculados no curso informando sobre a nova tarefa cadastrada no sistema. O aluno poderá enviar tarefas resolvidas para os professores, fazer *downloads* de matérias.

Após o desenvolvimento do AVA-Web foram desenvolvidos dois tipos de aplicações móveis (AVA móvel), ou seja, *off-line* e *on-line*, a fim de prover adequadamente a adaptabilidade especificada na arquitetura M-AVA. Na aplicação móvel *off-line*, não há necessidade da existência de uma conexão constante com o servidor para o seu funcionamento. Por outro lado, na aplicação *on-line* é necessário fazer o uso de interfaces Web utilizando o protocolo *WAP*.

Resumidamente, o AVA móvel deve conter as seguintes funcionalidades:

- O administrador poderá cadastrar um usuário remotamente
- O administrador poderá enviar e-mail para um ou todos níveis de usuário do AVA-WEB
- O professor poderá visualizar todos os cursos que ministra
- O professor poderá cadastrar uma nova tarefa remotamente.
- O aluno poderá visualizar todos os cursos em que está matriculado
- O aluno poderá visualizar todas as tarefas de um determinado curso remotamente

Para a especificação dos modelos utilizados na implementação da arquitetura M-AVA são utilizados os diagramas de caso de uso e de classes da UML (*Unified Modeling Language*) (CONALLEN, 1999).

O diagrama de casos de uso (*use case*) é utilizado normalmente nas fases de levantamento e análise de requisitos do sistema, embora é consultado durante todo o processo de modelagem e do desenvolvimento do software. Apresenta uma linguagem simples e fácil compreensão para que os usuários possam ter uma idéia geral de como o sistema irá se comportar (CONALLEN, 1999). O diagrama de classes é o diagrama mais utilizado e mais importante da UML, servindo de apoio para a maioria dos outros diagramas(CONALLEN, 1999) . Como o próprio nome diz, ele define a estrutura das classes utilizadas pelo sistema, determinando seus atributos e métodos, além de estabelecer como as classes se relacionam e como trocam informações entre si.

3.3. Modelagem Conceitual do AVA

Na seção a seguir é apresentado o Diagrama de Classe do AVA.

3.3.1. Diagrama de Classes

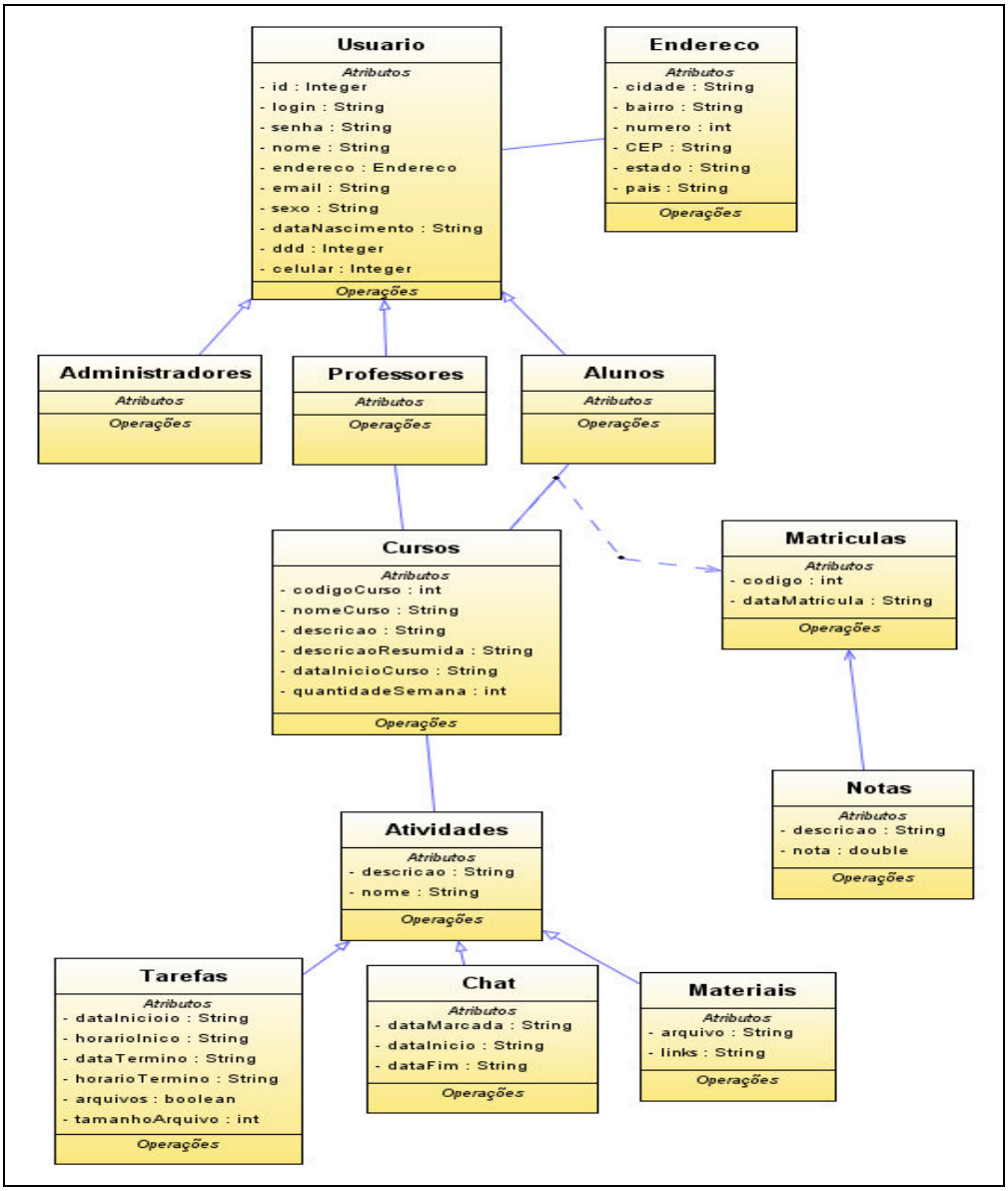
Na Figura 15 é ilustrado o diagrama de classe do AVA. A classe Usuário é uma classe abstrata, na qual é a super classe das classes Administrador, Professor e Aluno. Além de possuir todos os atributos listados no diagrama de classe da figura abaixo ela possui um atributo “endereço” no qual é uma instancia da classe *Endereço*. Como todos os atributos da classe “Usuário” são declarados como *private* para garantir o encapsulamentos dos dados, foi necessário criar métodos *getter’s* e *setter’s* específicos para cada atributos. A classe Cursos

contém informações relevante sobre os cursos tais como: código do curso, nome do curso, descrição do curso, descrição resumida do curso, data de inicio do curso e quantidade de semana que cada curso possui. Além dos atributos listados no diagrama de classe, essa classe possui métodos *getter's* e *setter's* responsáveis por obter e modificador os atributos, uma vez que os atributos são declarados com *private*. A classe Atividade é também uma classe abstrata, sendo que a mesma é super-classe das classes "Tarefas" e "Materiais", a mesma contem dois atributos que são: descrição, nome da atividade. A sub-classe "Material" é responsável por guardar o nome do arquivo totalmente qualificado, no entanto a sub-classe "Tarefa" contém outros atributos que são: dataInicio, horárioInicio, dataTermino, horárioTermino, arquivos, tamanhoArquivo. As principais funcionalidades do sistema AVA-WEB são:

- Usuário que possuir nível de administrador poderá cadastrar qualquer tipo de nível de usuário, cadastrar cursos e escolher um determinado professor para ministrar o curso.
- Usuário com nível de professor poderá enviar tarefas, matérias, adicionar notas e enviar tarefas por e-mail através do sistema AVA-WEB, para um respectivo curso que o mesmo ministrar.

Usuários que possuem o nível de aluno poderia visualizar todos os cursos que o mesmo participa, visualizar tarefas, fazer *downloads* de materiais, visualizar notas, e ainda enviar tarefas resolvidas para os professores.

Figura 15 : Diagrama de Classe do AVA



O diagrama de classes foi mapeado para uma base de dados do SGBD (Sistema de Gerenciamento de Banco de Dados) MySQL[1]. Tal base de dados é utilizada tanto pela aplicação *Web* quanto pelas aplicações móveis. Para cada classe (Administrar, Professor, Aluno) foi criada uma tabela, foi escolhido esse mapeamento, pois um usuário com o nível de Administrador possui mais atributos que um usuário de nível de professor e ainda um usuário com nível professor geralmente possui mais atributos que um usuário com o nível de aluno. A seguir é apresentada uma breve descrição das tabelas que o AVA possui:

- Administrador: tabela responsável pelo armazenamento de usuários que possuem nível de acesso “Administrador”.

- Aluno: tabela responsável pelo armazenamento de usuários que possuem nível de acesso “Aluno”.
- Professor: tabela responsável pelo armazenamento de usuários que possuem nível de acesso “Professor”.
- Cursos: tabela responsável pelo armazenamento de informações referentes aos cursos oferecidos.
- Materiais: tabela responsável pelo armazenamento do caminho onde se encontra uma determinada tarefa no servidor Web, ou seja, está tabela não possui um campo do tipo *blob* (tipo responsável por armazenar conteúdos binários) no qual o arquivo em si é persistido.
- Tarefas: tabela responsável pelo armazenamento de uma tarefa de um determinado curso.
- Matrículas: tabela responsável por armazenar o ID de um determinado aluno e o ID de um determinado curso, por se tratar de um relacionamento vários para vários, e adicionalmente também armazena a data correspondente à realização da matrícula.
- Notas: tabela que armazena as notas e a média que um aluno obteve em um determinado curso.
- TarefasResolvidas: tabela que armazena o caminho de um determinado arquivo enviado pelo aluno, cujo conteúdo é a resolução de uma determinada tarefa postada pelo professor. O nome e a data de envio do arquivo também são armazenados.

3.4. Implementação da Aplicação Web

Por se tratar de uma aplicação voltada para uso em conjunto com a Internet, o AVA Web contém em sua estrutura um módulo que deve prover o tratamento do servidor Web. Tal servidor é implementado neste trabalho com o uso de *Servlets*, JSP e API’s necessárias para a execução de funcionalidades como: enviar *email*, efetuar *upload*, realizar conexão com o SGBD.

Toda a implementação do AVA Web foi feita com o uso da ferramenta NetBeans 6.1.. Essa ferramenta e todos os outros recursos utilizados são *open-source* (software livre) e estão disponíveis nos sites de seus respectivos desenvolvedores. A escolha desta ferramenta foi devido ao fato da mesma dar suporte a *Servlets* e JSP. Outro fator importante é a facilidade

[1] http://www.mysql.com - Acesso em 10 de abril de 2008

da ferramenta lidar com servidor Web, uma vez que, instalando o *NetBeans* o usuário tem a opção de instalar o servidor *Web.*

A criação do banco de dados da aplicação foi feita no SGBD MySQL 5.0, que é gratuito. Sua utilização foi devido a facilidade de interação que possui com a linguagem Java, que é realizada com uso do *driver* de conexão *JDBC* (*Java Database Connection*), o qual torna possível a comunicação entre a linguagem Java e o SGBD. A seguir será apresentado os requisitos funcionais da aplicação AVA WEB que são:

1. O sistema deve permitir a inclusão, remoção, alteração de USUÁRIOS, contendo os seguintes atributos: nome, cidade, estado, pais, telefone, email, etc. Sendo que os Usuários podem ser classificados como:
    - Aluno: Pode acessar o ambiente através de um login, visitar disciplinas e ver os conteúdos delas.
    - Professor: Tem acesso aos cursos em que está designado como professor e pode incluir ou remover materiais.
    - Administrador: Tem acesso a todas as funcionalidades da aplicação *WEB* e pode modificá-las.
2. O sistema deve permitir a inclusão, alteração e remoção de CURSOS, podendo ser configurado pelo professor as seguintes características:
    - Nome do Curso
    - Uma breve descrição do curso
    - Definição da data de inicio do curso
    - Quantidade de semana que o curso consumirá
3. O sistema deve permitir a inclusão, alteração e a remoção de NOTAS por meio de um boletim em que o professor poderá lançar notas de acompanhamento dos alunos em um curso.
4. Cada curso pode trabalhar com diversas atividades como TAREFAS, Chat, FÓRUM, MATERIAIS, sendo que as atividades devem ser cadastradas e escolhidas pelo professor. Quando uma nova atividade for proposta, deve ser enviado um e-mail para todos os alunos que estão cadastrados naquele curso.
5. O sistema deve permitir a inclusão, alteração e a remoção de TAREFAS, podendo ser configuradas pelo professor. A opção TAREFAS permite aos professores a proposição de trabalhos e recebimento dos mesmos submetidos pelos estudantes, podendo ser estabelecido um prazo determinado para a submissão das

tarefas. Características necessárias das tarefas são: nome da tarefa, descrição, data e horário da disponibilização da tarefa, data e horário do término da tarefa, permitir ou não envio de arquivos, qual o tamanho do arquivo permitido e envio de e-mail para o professor informando a postagem da resolução da tarefa.

6. Todo curso tem a possibilidade de incluir, alterar ou remover MATERIAIS. Os materiais são todos os tipos de conteúdos que serão apresentados no curso. Podem ser documentos arquivados no servidor, páginas criadas com o uso dos editores de textos. Um aluno pode freqüentar vários cursos. Sendo possível que quando o aluno entre com o seu *login*, verifique em quais cursos ele está cadastrado.

7. O professor, ao cadastrar um curso, poderá informar quem serão os participantes (alunos) do curso, sendo que, deverá estar disponível para ele uma lista dos alunos já cadastrados no AVA, podendo definir se estes poderão fazer parte deste novo curso e ainda a possibilidade de cadastrar novos usuários.

Em seguida será apresentado alguns dos principais trechos de código da aplicação AVA-Web tais como fazer *uploads*, enviar e-mail.

Fazer *upload* é um processo muito comum realizado por qualquer um que acessa a internet, *upload* é o processo inverso do *download*, ou seja, você envia o arquivo para um respectivo servidor. Para tornar possível essa funcionalidade com *Servlet* e JSP existe uma API desenvolvida pela Apache Commons chamada Commons FileUpload. Tal API foi utilizada no AVA-Web. A seguir será apresentado um trecho de código que ilustra o funcionamento da mesma.

Figura 16 : utilizando FileUpload

```
01.  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
02.            throws ServletException, IOException {
03.
04.        boolean isMultiPart = ServletFileUpload.isMultipartContent(request);
05.        if (isMultiPart) {
06.
07.            FileItemFactory factory = new DiskFileItemFactory();
08.            ServletFileUpload servletFileUpload = new ServletFileUpload(factory);
09.
10.            try {
11.
12.                List<FileItem> itens = servletFileUpload.parseRequest(request);
13.                for (FileItem fileItem : itens) {
14.
15.                    if (!fileItem.isFormField()) {
16.                        /*false reprenseta um form de upload
17.                         */
18.                        if (fileItem.getName().length() > 0) {
19.
20.                            String caminho = this.inserirArquivoDiretorio(fileItem, nomeCurso);
21.                            String[] nomeMaterial = fileItem.getName().split("\\\\");
22.                            RequestDispatcher rd = request.getRequestDispatcher("principalProfessor.jsp");
23.                            rd.forward(request, response);
24.                        }
25.                    }
26.                }
27.
28.            } catch (FileUploadException e) {
29.
30.                throw new ServletException(e);
31.
32.            } catch (SQLException e) {
33.
34.                throw new ServletException(e);
35.
36.            }
37.
38.        }
39.
40.    }
```

A primeira coisa que deve ser feita para fazer o upload de um arquivo utilizando Servlet e JSP é verificar se a solicitação é do tipo multipart/form-data isso é possível por meio do método estático *isMultipartContext()*, esse método recebe como parâmetro um objeto do tipo *HttpServletRequest*, afim de analisar a solicitação e verificar é *multipart/form-data*. Após isso é criado um objeto do tipo *DiskFileUpload* no qual é responsável por analisar solicitação do arquivo que será feito o *upload* .Em seguida é criado um objeto do tipo *ServletFileUpload* no qual é responsável por "dividir" o arquivo para fazer o *upload*, isso é possível por meio do método *parseRequest(),* esses arquivos são colocadas em um lista do tipo *FileItem*. Com os arquivos já divididos é feita um iteração sobre a lista do tipo *FileItem* para armazenar o mesmo no servidor Web.

Outra funcionalidade relevante da aplicação AVA-Web é a capacidade de enviar e-mail, a mesma foi implementada utilizando a API Commons Email desenvolvida pela Apache

Commons juntamente com a API JavaMail da SUN. A seguir será apresentada a classe responsável por essa funcionalidade e também será comentada a respeito da mesma.

Figura 17 : Classe FabricaEmail

```
01. import org.apache.commons.mail.EmailException;
02. import org.apache.commons.mail.MultiPartEmail;
03. import org.apache.commons.mail.SimpleEmail;
04.
05. /**
06. *
07. * @author Rafael Durelli
08. */
09. public class FabricaEmail {
10.
11.     private static  SimpleEmail simpleEmail;
12.     private static MultiPartEmail multiPartEmail;
13.
14.
15.     public static SimpleEmail getEmail()throws EmailException{
16.
17.         simpleEmail = new SimpleEmail();
18.         simpleEmail.setHostName("smtp.gmail.com");
19.         simpleEmail.setSmtpPort(587);
20.         simpleEmail.setSSL(true);//requerido para o gmail...
21.         simpleEmail.setAuthentication("ava-web", "senha");
22.         return simpleEmail;
23.
24.     }
25.
26.     public static MultiPartEmail getEmailAttachment() throws EmailException{
27.
28.         multiPartEmail = new MultiPartEmail();
29.         multiPartEmail.setHostName("smtp.gmail.com");
30.         multiPartEmail.setSmtpPort(587);
31.         multiPartEmail.setSSL(true);
32.         multiPartEmail.setAuthentication("ava-web", "senha");
33.         return multiPartEmail;
34.
35.     }
36.
37.
38.
39. }
```

Essa classe como o próprio nome diz é responsável por criar os objetos necessários para enviar e-mail. A mesma possui dois métodos estáticos, onde o primeiro é responsável por criar um objeto *SimpleEmail*, essa classe é restrita a enviar somente e-mail simples, ou seja, somente mensagens de texto, já o objeto *MultiPartEmail* é possível enviar por e-mail um arquivo anexado.

As linhas 18 e 29 são responsável por definir o servidor que está utilizando para enviar e-mail, as linhas 19 e 30 são responsáveis por definir as portas que serão responsáveis por enviar o e-mail, nas linhas 20 e 31 é realizada uma configuração necessário para o servidor que está sendo utilizado, por ultimo, mas não menos importante ocorre nas linhas 21 e 32 que são responsáveis por fazer autenticação do servidor.

Ainda no projeto *Web* foram criados pacotes que contêm os *beans*, representando as classes de regras de negócios do sistema (Administrador, Aluno, Professor, Curso, etc),

objetos DAO (*Data Access Object*) que são responsáveis por acessar o banco de dados através da API JDBC e vários *Servlets* que são responsáveis por processar a solicitação de um respectivo dispositivo móvel, e posteriormente enviar um resposta para o dispositivo móvel.

Os *Servlets* responsáveis por fazer a interação com os dispositivos móveis utilizam dois objetos das classes `DataInputStream`, `DataOutputStream` do pacote `java.io`. O primeiro objeto "lê" os dados que contêm no protocolo HTTP e o segundo "escreve" dados em tal protocolo.

3.5. Implementação da Aplicação Móvel Off-Line

As pessoas sentem a necessidade de estarem informadas, quer seja por meio de jornais quer seja por uma consulta em uma página de notícias na internet. Hoje em dia, já é rotina poder checar a caixa de e-mails pelo celular ou atualizar a agenda de compromissos direto para o servidor da empresa. A mobilidade aliada à tecnologia sem fio trouxe para o mercado um novo paradigma de comunicação, a comunicação móvel sem fio.

Este módulo do trabalho tem como objetivo implementar a adaptabilidade de AVA Web para o AVA móvel utilizando a tecnologia J2ME (*Java Micro Edition*).

A tecnologia J2ME, além de fornecer uma rica API, para tratar de conectividade e persistência de dados, provê também portabilidade ao sistema desenvolvida. Essa última característica foi o motivo primordial na adoção desta tecnologia, uma vez que a aplicação móvel poderá funcionar em qualquer celular que tenha suporte para Java, bastando somente que o dispositivo (que pode ser um PDA ou um celular) suporte a configuração e o perfil desejado. Para maiores informações sobre J2ME consulte o Capítulo 1.

O AVA móvel tem como objetivo facilitar o usuário do AVA *Web*, permitindo que algumas das funcionalidades do sistema *Web*, sejam executadas de qualquer lugar e a qualquer momento. Uma das funcionalidades disponível para o sistema AVA móvel é possibilitar que o aluno visualize remotamente todas as tarefas adicionadas para um respectivo curso.

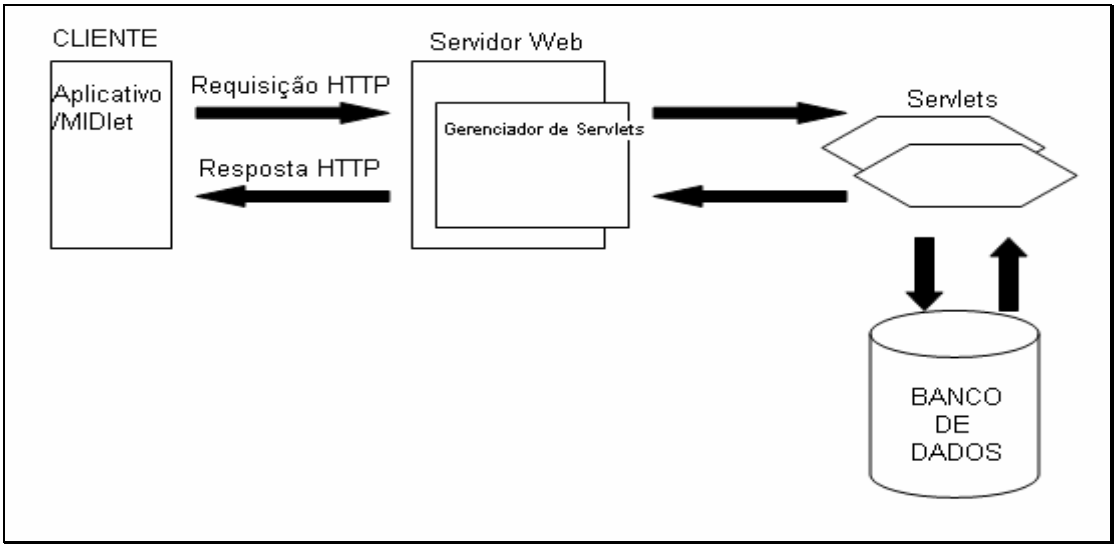
O mesmo servidor utilizado para implementar o módulo *Web*, foi utilizado para fazer a interface entre as duas aplicações (AVA *Web* e AVA móvel). Como comentado no capitulo 1, quando J2ME é utilizado para o desenvolvimento de aplicações móveis é necessário estender a classe *MIDlet*. Neste trabalho, a classe que estende a *MIDlet* interpreta as requisições e respostas vindas do *Servlet* remoto por meio da GCF (*Generic Connection FrameWork*). GCF é um framework desenvolvido especificamente para a especificação J2ME

que trata diversos tipos de conexões, uma vez que as API da especificação J2SE foi considerada muito grande para caber na memória de um dispositivo móvel.

A comunicação entre os o AVA *Web* e o AVA móvel *off-line* é realizado por meio do protocolo HTTP (*HyperText Transfer Protocol*), que é implementado na especificação J2ME pela *interface* `HttpConnection`. Como ocorre no AVA *Web*, foram também utilizados objetos das classes `DataInputStream`, `DataOutputStream`,.

Na Figura 17 apresenta-se, de forma genérica, como o *MIDlet* se comunica com os *Servlets* a fim de obter os dados do SGBD.

Figura 18 : Interação MIDlet com Servlets



A fim de melhorar o desempenho e permitir que o usuário continue realizando outras tarefas enquanto os dados são recebidos pelo dispositivo móvel, o envio e o recebimento de dados foram implementados com *Threads* por meio da classe `Thread` do pacote `java.lang`.

Salienta-se que o perfil MIDP, por mais que seja limitado em recursos aos componentes gráficos, disponibiliza uma amigável interface gráfica com o usuário. O AVA móvel *off-line* utiliza vários componentes deste perfil com a finalidade de realizar a comunicação visual com o usuário.

Para utilizar o sistema AVA móvel *off-line*, o usuário deve fazer o *login* para poder ter acesso as funcionalidades de um determinado nível usuário (Figuras 20 e 21), ou seja, Administrador, Professor e Aluno, os quais podem acessar funcionalidades específicas.

Figura 19 : Tela de login do AVA móvel off-line

A seguir será apresenta o método responsável por criar a figura ilustrada acima.

Figura 20 : método criaLogin

```
01.  public void criaLogin() {
02.          try {
03.              this.tickerPrincipal = new Ticker("LOGIN");
04.              this.telaPrincipal = new Form("ESCOLHA O NIVEL PARA EFETUAR O LOGIN");
05.              this.telaPrincipal.setTicker(tickerPrincipal);
06.              this.opcoesPrincipal = new ChoiceGroup("NIVEL DE USUÁRIO", ChoiceGroup.EXCLUSIVE);
07.              this.telaPrincipal.append(this.opcoesPrincipal);
08.              this.okPrincipal = new Command("OK", Command.OK, 0);
09.              this.voltarPrincipal = new Command("VOLTAR", Command.BACK, 1);
10.              this.helpPrincipal = new Command("AJUDA", Command.HELP, 0);
11.              this.telaPrincipal.addCommand(okPrincipal);
12.              this.telaPrincipal.addCommand(voltarPrincipal);
13.              this.telaPrincipal.addCommand(helpPrincipal);
14.              this.opcoesPrincipal.append("ADMINISTRADOR", Image.createImage("/user_green.png"));
15.              this.opcoesPrincipal.append("PROFESSOR", Image.createImage("/user_red.png"));
16.              this.opcoesPrincipal.append("ALUNO", Image.createImage("/user_suit.png"));
17.              this.telaPrincipal.setCommandListener(this);
18.          } catch (IOException e) {
19.
20.              e.printStackTrace();
21.
22.          }
23.      }
```

A linha 3 cria um objeto *Ticker* esse objeto é um componente de texto rolante, na linha 4 está criando um objeto do tipo *Form* o qual é responsável por anexar outros tipo de componentes gráficos. Nas linhas 8, 9, 10 está sendo criado objetos *Command*, o modo mais simples de pensar em um objeto *Command* é como um “botão”, algo que é pressionado ou selecionado. As linhas 11, 12, 13 são responsáveis por adicionar os objetos *Command* no objeto *Form*, as linhas 14, 15, 16 criam os respectivos níveis do sistema Ava móvel off-line.

Figura 21 : Tela de Login do Administrador do AVA móvel off-line

É importante ressaltar que o *login* e a senha informados na Figura 21 devem ser os mesmos que constam na base de dados que o sistema AVA WEB utiliza, uma vez que, quem é responsável por verificar se o usuário é válido é um *Servlet*, no lado do servidor. Isso implica que o mesmo *login* e senha utilizada para fazer o *login* no sistema AVA WEB deve ser utilizado no AVA móvel. As telas de *login* apresentadas são utilizadas para todos os níveis de usuário.

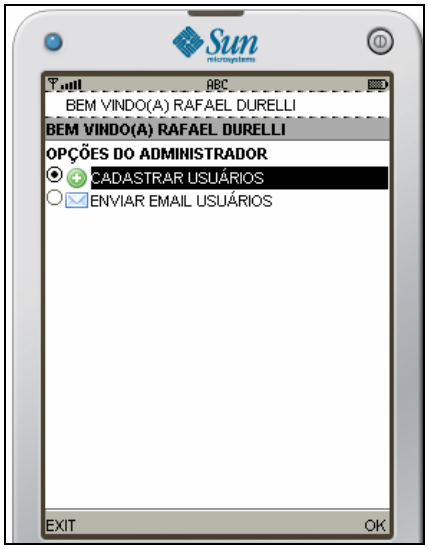
O código responsável por fazer a conexão com o servidor *Web* é apresentado na Figura 22.

A conexão é feita com o *Servlet* `LoginAdministrador`, informando o nome do usuário e a senha por meio da URL. Na linha 09 cria-se um objeto `HttpConnection`, que é responsável por enviar os dados para o *Servlet*, a fim de verificar se o administrador é um usuário válido ou não. Nas linhas 10 até 13 configuram a propriedade de solicitação necessária para fazer a conexão com o *Servlet*. Se o mesmo for um administrador válido a tela ilustrada na Figura 23 é apresentada ao usuário.

Figura 22 : Trecho de código de conexão com o servidor Web

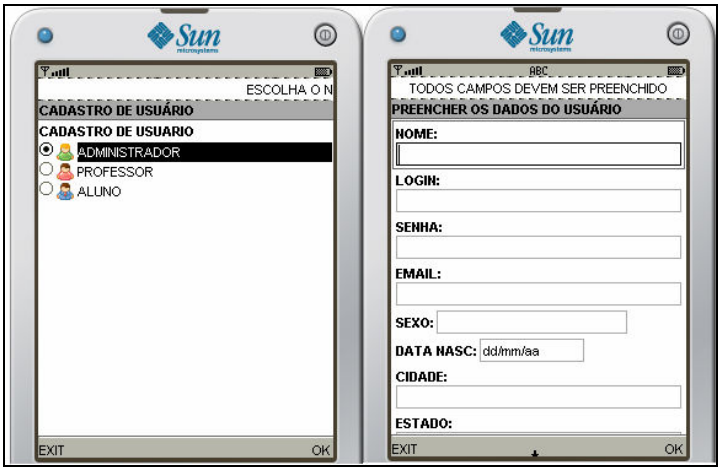
```
01. public void chamaServletLoginAdministrador() throws IOException {
02.
03.       String URL = "http://localhost:8084/TCC/LoginAdministrador" + "?nome=" + textFieldLoginAdministrador.getString() + "&senha=" +
    textFieldSenhaAdministrador.getString();
04.       HttpConnection conexao = null;
05.       DataInputStream IS = null;
06.
07.       try {
08.
09.           conexao = (HttpConnection) Connector.open(URL);
10.           conexao.setRequestMethod(HttpConnection.GET);
11.           conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
12.           conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
13.           conexao.setRequestProperty("Content Language", "en Ca");
```

Figura 23 : Funcionalidades do Administrador do AVA móvel



Um usuário do AVA móvel com o nível de administrador possui duas funcionalidades: cadastrar usuário e enviar e-mail para os usuários (Figura 23). A primeira funcionalidade cadastra usuários do sistema em qualquer lugar, a qualquer momento, tornando possível que o administrador possa cadastrar um usuário remotamente. A segunda funcionalidade envia remotamente e-mail para todos os usuários cadastrados no sistema, por grupo de usuário (ou seja, por Administrador ou por Aluno ou por Professor).

Figura 24 : Cadastro de Usuário



Na Figura 25 é apresentado o código responsável por por implementar o cadastro do usuário.

Figura 25 : Cadastrar remotamente um usuário no AVA móvel

```
01.  public void chamaServletCadastrarAdministrador() throws IOException {
02.
03.          String URL = "http://localhost:8084/TCC/CadastrarAdministrador";
04.          HttpConnection conexao = null;
05.          DataOutputStream OS = null;
06.          try {
07.
08.              conexao = (HttpConnection) Connector.open(URL);
09.              conexao.setRequestMethod(HttpConnection.POST);
10.              conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
11.              conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
12.              conexao.setRequestProperty("Content Language", "en Ca");
13.              OS = conexao.openDataOutputStream();
14.              OS.writeUTF(this.nomeUsuario.getString());
15.              OS.writeUTF(this.loginUsuario.getString());
16.              OS.writeUTF(this.senhaUsuario.getString());
17.              OS.writeUTF(this.emailUsuario.getString());
18.              OS.writeUTF(this.sexoUsuario.getString());
19.              OS.writeUTF(this.dataNascimentoUsuario.getString());
20.              OS.writeUTF(this.cidadeUsuario.getString());
21.              OS.writeUTF(this.estadoUsuario.getString());
22.              OS.writeInt(Integer.parseInt(this.numeroUsuario.getString()));
23.              OS.writeUTF(this.cepUsuario.getString());
24.              OS.writeUTF(this.ruaUsuario.getString());
25.              OS.writeUTF(this.bairroUsuario.getString());
26.              OS.writeUTF(this.paisUsuario.getString());
27.              OS.writeInt(Integer.parseInt(this.dddUsuario.getString()));
28.              OS.writeInt(Integer.parseInt(this.celularUsuario.getString()));
29.
30.          } finally {
31.
32.              if (OS != null) {
33.
34.                  OS.close();
35.              }
36.
37.              if (conexao != null) {
38.                  conexao.close();
39.
40.              }
41.
42.          }
43.          this.alertaCadastroSucesso = new Alert("USUÁRIO CADASTRADO COM SUCESSO ");
44.          this.display.setCurrent(alertaCadastroSucesso, this.telaAdministrador2);
45.      }
```

O código nas linhas 3 até 12 são praticamente idênticas as da Figura 22 , ou seja, responsável por fazer a configuração da solicitação. No entanto, as linhas 14 até 28 são responsáveis por obter os dados que se encontram nos *TextFields* e enviá-los para o *Servlet* `CadastrarAdministrador`, por meio do objeto `DataOutputStream`. Para isso foi utilizado o método `writeUFT()`, responsável por enviar cadeias de caracteres (`String`), e `writeInt()`, responsável por enviar um primitivo inteiro.

Na Figura 26 é apresentada a tela de envio de e-mail no AVA móvel.

Figura 26 : E-mail remoto no AVA móvel off-line

O código listado na Figura 27 faz a conexão com o *Servlet* `EnviarEmailAdministrador`, que utiliza a API javaMail para tornar possível o envio de e-mail. Para isso, antes de enviar os dados para o *Servlet* é necessário configurar a solicitação, como comentado anteriormente. Isso é feito por meio do método `setRequestProperty()`. Após a configuração da solicitação, os dados são enviados por meio do objeto `DataOutputStream`.

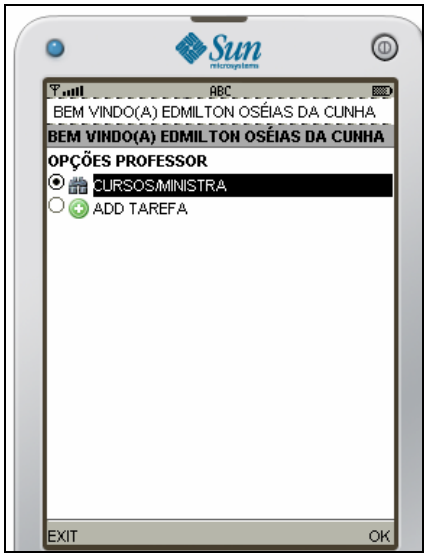
Figura 27 : Código que implementa o envio de email remotamente

```
01.  public void chamaServletEnviarEmailAdministrador() throws IOException {
02.
03.          String URL = "http://localhost:8084/TCC/EnviarEmailAdministrador";
04.          HttpConnection conexao = null;
05.
06.          DataOutputStream OS = null;
07.          try {
08.
09.              conexao = (HttpConnection) Connector.open(URL);
10.              conexao.setRequestMethod(HttpConnection.POST);
11.              conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
12.              conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
13.              conexao.setRequestProperty("Content Language", "en Ca");
14.              OS = conexao.openDataOutputStream();
15.              OS.writeUTF(this.administrador.getEmail());
16.              OS.writeBoolean(this.Administrador);
17.              OS.writeBoolean(this.Professor);
18.              OS.writeBoolean(this.Aluno);
19.              OS.writeUTF(this.assuntoEmail.getString());
20.              OS.writeUTF(this.textoEmail.getString());
21.              this.emailEnviarSucess = new Alert("EMAIL ENVIADO COM SUCESSO");
22.              this.display.setCurrent(emailEnviarSucess, telaAdministrador);
23.
24.
25.          } finally {
26.              if (OS != null) {
27.
28.                  OS.close();
29.              }
30.
31.              if (conexao != null) {
32.                  conexao.close();
33.
34.              }
35.
36.          }
37.      }
```

A seguir são comentadas sobre as funcionalidades específicas do nível de usuário Professor.

Um usuário com o nível de Professor, no AVA móvel, pode visualizar os cursos que ministra e enviar uma tarefa para um respectivo curso (Figura 28), fazendo com que os alunos que participam de tal curso possam visualizar esta tarefa tanto no AVA *WEB* quanto no AVA móvel.

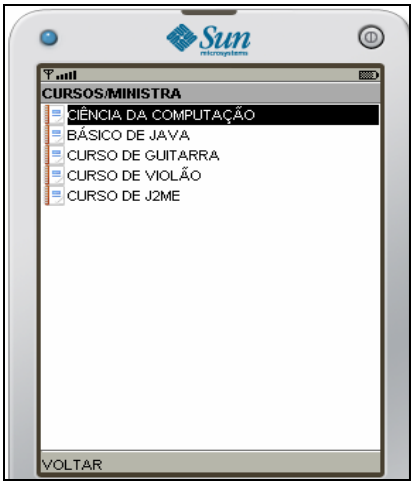
Figura 28 : Funcionalidades do nível professor do AVA móvel



Quando o usuário faz o *login* no nível de professor, uma tela como apresentada na Figura 28 é mostrada para o usuário, com as funcionalidades CURSOS/MINISTRAR e ADD TAREFA. A primeira funcionalidade mostra todos os cursos que o professor ministra.

Quando o administrador cadastra um curso e o atribui para um professor no AVA *WEB*, o usuário com nível de professor pode visualizar todos os cursos que são ministrados por ele. Na Figura 29 são listados, no AVA móvel, todos os cursos que um determinado professor ministra e na Figura 30 apresenta-se o respectivo trecho de código.

Figura 29 : Lista de curso



A figura acima mostrou que um determinado professor pode visualizar os cursos que o mesmo ministra o código responsável por fazer isso será apresentado na figura abaixo.

Figura 30 : listar cursos

```
01.  public void chamaServletVisualizarCursosProfessorMinistra() throws IOException {
02.
03.          String URL = "http://localhost:8084/TCC/VisualizarCursosProfessorMinistra";
04.          HttpConnection conexao = null;
05.          DataInputStream IS = null;
06.          DataOutputStream OS = null;
07.          try {
08.              conexao = (HttpConnection) Connector.open(URL);
09.              conexao.setRequestMethod(HttpConnection.POST);
10.              conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
11.              conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
12.              conexao.setRequestProperty("Content Language", "en Ca");
13.              OS = conexao.openDataOutputStream();
14.              OS.writeInt(this.professor.getId().intValue());
15.              IS = conexao.openDataInputStream();
16.              int numeroDeCurso = IS.readInt();
17.              if (numeroDeCurso != 0) {
18.                  this.listaCursosMinistra = new List("CURSOS/MINISTRA", List.IMPLICIT);
19.                  this.voltarProfessor = new Command("VOLTAR", Command.BACK, 0);
20.                  this.listaCursosMinistra.addCommand(this.voltarProfessor);
21.                  this.listaCursosMinistra.setCommandListener(this);
22.                  for (int i = 0; i < numeroDeCurso; i++) {
23.                      int idCurso = IS.readInt();
24.                      String nomeCurso = IS.readUTF();
25.                      try {
26.                          this.listaCursosMinistra.append(nomeCurso, Image.createImage("//report.png"));
27.                      } catch (IOException e) {
28.                          e.printStackTrace();
29.                      }
30.                  }
31.                  this.display.setCurrent(this.listaCursosMinistra);
32.              } else {
33.                  this.alertaCursosParticipa = new Alert("PROFESSOR NÃO MINISTRA NENHUM CURSO");
34.                  this.display.setCurrent(alertaFaileLogin, telaProfessor);
35.              }
36.          } finally {
37.              if (IS != null) {
38.                  IS.close();
39.              }
40.              if (OS != null) {
41.                  OS.close();
42.              }
43.              if (conexao != null) {
44.                  conexao.close();
45.              }
46.          }
```

O método listado na figura 28, como os demais métodos apresentados anteriormente, primeiramente configura a solicitação utilizando o método *setRequestProperty* do objeto *HttpConnection*. Após fazer a conexão com o respectivo *Servlet*, os dados são extraídos para o AVA móvel utilizando o objeto *DataInputStream*, por meio dos métodos *read's* específicos. Na linha 16 é possível notar que é extraído a quantidade de cursos que um determinado professor possui para fazer um *loop* e adicionar todos os seus respectivos cursos em um objeto do tipo *javax.microedition.lcdui.List*. Este objeto tem as características de uma lista gráfica, sendo que o mesmo será exibido no display do aplicativo móvel. Na Figura 31 é mostrado o *Servlet* responsável por enviar os cursos que um respectivo professor ministra para o dispositivo móvel.

Figura 31 : Servlet responsável por listar cursos

```
01. public class VisualizarCursosProfessorMinistra extends HttpServlet {
02. 
03.     private DataOutputStream out = null;
04.     private DataInputStream in = null;
05. 
06. 
07.     protected void doPost(HttpServletRequest request, HttpServletResponse response)
08.             throws ServletException, IOException {
09. 
10.         try {
11.             ProfessorDAO professorDAO = new ProfessorDAO();
12.             in = new DataInputStream((InputStream) request.getInputStream());
13.             int codigoProfessor = in.readInt();
14.             List<Cursos> listaDeCursos = professorDAO.cursosProfessorMinistra(codigoProfessor);
15.             out = new DataOutputStream((OutputStream) response.getOutputStream());
16.             if (listaDeCursos != null) {
17. 
18.                 out.writeInt(listaDeCursos.size());
19.                 for (Cursos cursos : listaDeCursos) {
20.                     out.writeInt(cursos.getCodigoCurso());
21.                     out.writeUTF(cursos.getNomeCurso());
22.                 }
23. 
24.             } else {
25.                 out.writeInt(0);
26.             }
27. 
28.         } catch (SQLException e) {
29. 
30.             throw new ServletException(e);
31. 
32.         } finally {
33. 
34.             if (in != null) {
35. 
36.                 in.close();
37.             }
38.             if (out != null) {
39. 
40.                 out.close();
41. 
42.             }
43. 
44.         }
45. 
46.     }
47. }
```

Quando uma solicitação do dispositivo móvel chega a esse *Servlet*, o mesmo executa o método *doPost(),* criando um objeto do tipo *ProfessorDAO*, esse objeto é responsável por fazer a persistência dos objetos do tipo *Professor*. Na linha 12 é utilizado o método *getInputStream()* do objeto do tipo *HttpServletRequest*, com esse método é possível obter um objeto do tipo *InputStream*, tornando assim possível criar um objeto *DataInputStream* para poder obter os dados enviados pelo dispositivo móvel. A linha 13 lê o código do professor enviado pelo dispositivo móvel a fim de fazer uma consulta a base de dados para listar todos os cursos que o mesmo ministra, e em seguida colocar os cursos em um objeto do tipo *java.util.List.* A linha 18 é responsável por enviar a quantidade de cursos que o respectivo professor ministra, a linha 19 até a 22 for um *loop* enviando todos os códigos e nomes dos cursos que o professor ministra por meio do objeto *DataOutputStream*.

Como comentado anteriormente, a outra funcionalidade que um usuário com o nível de Professor, no AVA móvel, pode efetuar é o envio (cadastro) de uma tarefa para um respectivo curso. Na Figura 32 apresenta-se a tela para adicionar uma nova tarefa a um determinado curso e, em seguida, na Figura 33 é apresentado o método responsável por enviar os dados para o devido *Servlet* e, na Figura 34 é apresentado o *Servlet* responsável por adicionar a tarefa na base de dados.

Figura 32 : Adicionar tarefa remotamente

A parte mais importante do método apresentado na Figura 33 encontra-se na linha 13, que é responsável por obter o id (identificador) de um respectivo curso que o usuário escolheu para adicionar uma tarefa remotamente. Após ter obtido o id do curso é necessário extrair dos *textfields* informações relevantes para cadastrar a respectiva tarefa no banco de dados. Os dados da tarefa são enviado para o *Servlet*, por meio do objeto da classe `DataOutputStream` (linhas 17, 18, 19 e 20). Após a execução do método chamaServletCadastrarTarefa, o *Servlet* `CadastrarTarefa` é chamado (Figura 34) para fazer a inserção de uma tarefa remotamente na base de dados. Em seguida será enviado e-mail para todos os alunos que participam do respectivo curso a fim de informar que foi adicionada uma nova tarefa.

Figura 33 : método chamaServletCadastrarTarefa

```
01.  public void chamaServletCadastrarTarefa() throws IOException {
02.
03.          String URL = "http://localhost:8084/TCC/CadastrarTarefa";
04.          HttpConnection conexao = null;
05.          DataOutputStream OS = null;
06.          try {
07.              conexao = (HttpConnection) Connector.open(URL);
08.              conexao.setRequestMethod(HttpConnection.POST);
09.              conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
10.              conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
11.              conexao.setRequestProperty("Content Language", "en Ca");
12.              OS = conexao.openDataOutputStream();
13.              int idCurso = cursosTarefas.getCodigoCurso().intValue();
14.              String assunto = assuntoTarefa.getString();
15.              String data = dataEntregaTarefa.getString();
16.              String descricao = descricaoTarefa.getString();
17.              OS.writeInt(idCurso);
18.              OS.writeUTF(assunto);
19.              OS.writeUTF(data);
20.              OS.writeUTF(descricao);
21.              Alert cadastradoTarefa = new Alert("CADASTRO", "TAREFA CADASTRADA COM SUCESSO", Image.createImage("//tarefa.png"),
     AlertType.CONFIRMATION);
22.              this.display.setCurrent(cadastradoTarefa, listaCursosTarefa);
23.          } catch (IOException e) {
24.              e.printStackTrace();
25.          } finally {
26.              if (OS != null) {
27.                  OS.close();
28.              }
29.              if (conexao != null) {
30.                  conexao.close();
31.              }
32.          }
33.      }
```

As linhas 8 até a 10 do código listado na Figura 34 são responsáveis por obter a data que a tarefa foi adicionada, as linhas 11, 12, 13 criam objetos responsáveis por fazer a iteração com o respectivo banco de dados, a linha 14 cria um objeto `Tarefa`, que é informado como parâmetro em um método pertencente ao objeto `TarefaDAO` para cadastrar uma nova tarefa para um respectivo curso. A linha 15 cria um objeto do tipo `DataInputStream` para obter os dados passados pelo dispositivo móvel, em seguida, os dados são lidos utilizando os respectivos métodos *read's*. Esse procedimento é feito nas linhas 16, 17, 18, 19. Em seguida, o objeto `Tarefa` é configurado pelos métodos *setters* com os seus respectivos dados, chamando o método `adicionarTarefa()` do objeto `TarefaDAO` para fazer a inserção no banco de dados. As linhas 26 até 31 correspondem ao envio de e-mail aos alunos do curso, notificando a respeito da nova tarefa adicionada.

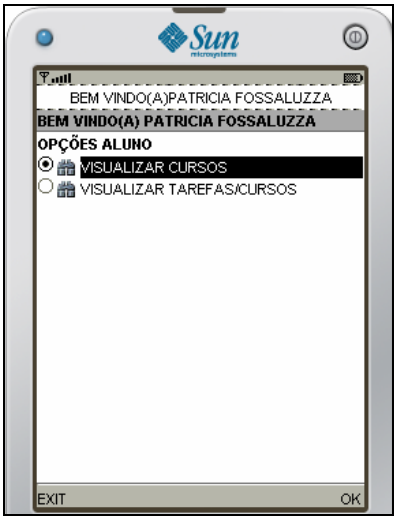
Figura 34 : Servlet CadastrarTarefa

```
01. public class CadastrarTarefa extends HttpServlet {
02.    private DataInputStream IS = null;
03.
04.
05.     protected void doPost(HttpServletRequest request, HttpServletResponse response)
06.     throws ServletException, IOException {
07.         try{
08.             Calendar calendar = Calendar.getInstance();
09.             DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT);
10.             String data = df.format(calendar.getTime());
11.             TarefaDAO tarefaDAO = new TarefaDAO();
12.             AlunoDAO alunoDAO = new AlunoDAO();
13.             CursoDAO cursoDAO = new CursoDAO();
14.             Tarefa tarefa = new Tarefa();
15.             IS = new DataInputStream((InputStream)request.getInputStream());
16.             int codigoCurso = IS.readInt();
17.             String assuntoTarefa = IS.readUTF();
18.             String dataEntrega = IS.readUTF();
19.             String descricao = IS.readUTF();
20.             tarefa.setIdCurso(codigoCurso);
21.             tarefa.setAssuntoTarefa(assuntoTarefa);
22.             tarefa.setDataEntregaTarefa(dataEntrega);
23.             tarefa.setDataLiberadaTarefa(data);
24.             tarefa.setDescricaoTarefa(descricao);
25.             tarefaDAO.adicionarTarefa(tarefa);
26.             List<Aluno> alunos = alunoDAO.listarTodosCurso(codigoCurso);
27.             Cursos cursos = cursoDAO.recuperarCurso(codigoCurso);
28.             Email email = new Email();
29.             for (Aluno aluno : alunos) {
30.                 email.sendEmailTarefa("sistemaAVA@gmail.com", aluno, cursos);
31.             }
32.         }catch(IOException e){
33.             throw new ServletException(e);
34.         }catch(SQLException e){
35.             throw new ServletException(e);
36.         }catch(EmailException e){
37.             throw new ServletException(e);
38.         }finally{
39.             if(IS != null){
40.                 IS.close();
41.             }
42.         }
43.     }
44.
45. }
```

A seguir são comentadas sobre as funcionalidades específicas do nível de usuário Aluno.

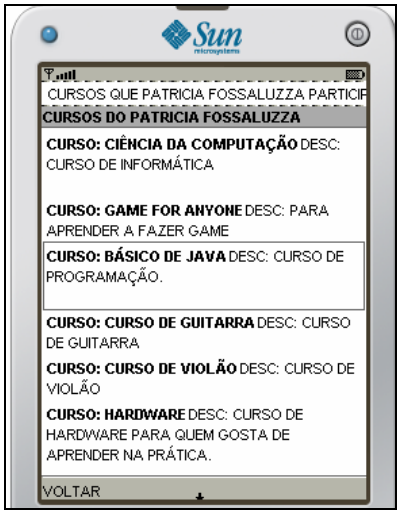
Um usuário com o nível de Aluno, no AVA móvel, pode visualizar remotamente os cursos que participa e visualizar as tarefas de um determinado curso que participa (Figura 35).

Figura 35 : Funcionalidades Aluno Móbile



Durante a visualização dos cursos que participa, além do nome do curso é possível visualizar também uma descrição resumida do curso (Figura 36).

Figura 36 : Curso que aluno participa



Na Figura 37 é apresentado o método que faz a conexão com um respectivo *Servlet,* o qual envia o nome e a descrição resumida do respectivo curso que o aluno participa, e após isso será apresentado o código do *Servlet* que tem como função enviar os nomes e as descrições dos cursos que um determinado aluno participa.

As linhas 7 até 11 da Figura 37 são responsáveis por criar um objeto `HttpConnection` e configurar a solicitação por meio do método `setRequestProperty()`. Na linha 12 obtém-se um objeto `DataOutputStream`, responsável por enviar os dados para o respectivo *Servlet*, sendo que a única informação enviada para o *Servlet* nesse método é o *id* (identificador) do aluno por meio do método

`writeInt()` na linha 13. Após a execução do método `writeInt()`, o *Servlet* `VisualizarCursosQueAlunoFaz`, apresentado na Figura 38, é executado.

Figura 37 : método chamaServletVisualizarCursosQueAlunoFaz

```
01.  public void chamaServletVisualizarCursosQueAlunoFaz() throws IOException {
02.          String URL = "http://localhost:8084/TCC/VisualizarCursosQueAlunoFaz";
03.          HttpConnection conexao = null;
04.          DataInputStream IS = null;
05.          DataOutputStream OS = null;
06.          try {
07.              conexao = (HttpConnection) Connector.open(URL);
08.              conexao.setRequestMethod(HttpConnection.POST);
09.              conexao.setRequestProperty("IF-Modified-Since", "20 Jan 2001 16:19:14 GMT");
10.              conexao.setRequestProperty("User-Agent", "Profile/MIDP 2.0 Configuration/CLDC-1.0");
11.              conexao.setRequestProperty("Content Language", "en Ca");
12.              OS = conexao.openDataOutputStream();
13.              OS.writeInt(this.aluno.getId().intValue());
14.              IS = conexao.openDataInputStream();
15.              int numeroDeCurso = IS.readInt();
16.              if (numeroDeCurso != 0) {
17.                  this.telaCursosParticipa = new Form("CURSOS DO " + aluno.getNome());
18.                  this.voltarAluno2 = new Command("VOLTAR", Command.BACK, 1);
19.                  this.tickerCursosParticipa = new Ticker("CURSOS QUE " + aluno.getNome() + " PARTICIPA");
20.                  this.telaCursosParticipa.setTicker(this.tickerCursosParticipa);
21.                  for (int i = 0; i < numeroDeCurso; i++) {
22.                      String nomeCurso = IS.readUTF();
23.                      String descricaoResumidaCurso = IS.readUTF();
24.                      this.cursosParticipa = new StringItem("CURSO: " + nomeCurso, "DESC: " + descricaoResumidaCurso);
25.                      this.telaCursosParticipa.append(this.cursosParticipa);
26.                  }
27.                  this.telaCursosParticipa.setCommandListener(this);
28.                  this.telaCursosParticipa.addCommand(this.voltarAluno2);
29.                  this.display.setCurrent(this.telaCursosParticipa);
30.              } else {
31.                  this.alertaCursosParticipa = new Alert("ALUNO NÃO MATRICULA NENHUM CURSO");
32.                  this.display.setCurrent(alertaFaileLogin, telaAluno);
33.              }
34.          } finally {
35.              if (IS != null) {
36.                  IS.close();
37.              }
38.              if (OS != null) {
39.                  OS.close();
40.              }
41.              if (conexao != null) {
42.                  conexao.close();
43.              }
44.          }
45.      }
```

De acordo com a Figura 38, a primeira tarefa que o *Servlet* faz é obter um objeto `DataInputStream` por meio do objeto `HttpServletRequest` utilizando para isso o método `getInputStream()` (linha 8). Na linha 9 é chamado o método `readInt()` para obter o *id* (identificador) do aluno que foi informado pelo aplicativo móvel, na linha 12 é retornado um objeto `java.util.List` ou `null`, caso o aluno não esteja matriculado em nenhum curso.Na linha 14 um teste *if* é feito afim de verificar se a lista foi criada ou não, se a mesma não foi criado a clausula *else* será executada, enviando assim para o dispositivo móvel o valor 0 (zero) que significa que o aluno não esta matriculado em nenhum curso. Caso a lista tenha sido criada a clausula *if* será executada fazendo com que os códigos listados nas linhas 16 até 19 sejam executadas. Dentro da clausula *if* a primeira informação a ser enviado para o dispositivo móvel é o número de curso que o respectivo aluno está matriculado isto é possível

por meio do método *size()* do objeto *java.util.List*, ainda dentro do *if* é feita uma iteração sobre a lista, enviando assim todos os nomes e descrições resumida dos cursos para o dispositivo móvel. Após o *Servlet* ter anexado todas as informações necessárias, o código listado na figura 37 retorna a ser executado a partir da linha 14, onde um objeto *DataInputStream* é criado para obter os valores que o *Servlet* gerou. A linha 15 da figura 37 obtém um inteiro que representa a quantidade de cursos que aluno está cadastrado, um teste condicional é executado na linha 16, se a quantidade de cursos que o aluno participa for igual a 0 (zero) somente uma mensagem será mostrada na tela do dispositivo móvel terminando assim o método. No entanto, se a quantidade de cursos for diferente de 0 (zero), a clausula *if* será executada fazendo uma iteração sobre o objeto *DataInputStream* para obter todos os nomes e descrições dos cursos e mostrá-los na tela do dispositivo móvel.

Figura 38 : Servlet VisualizarCursosQueAlunoFaz

```
01.  public class VisualizarCursosQueAlunoFaz extends HttpServlet {
02.
03.      private DataInputStream in = null;
04.      private DataOutputStream out = null;
05.
06.      protected void doPost(HttpServletRequest request, HttpServletResponse response)
07.              throws ServletException, IOException {
08.          in = new DataInputStream((InputStream) request.getInputStream());
09.          int codigoAluno = in.readInt();
10.          try {
11.              AlunoDAO alunoDAO = new AlunoDAO();
12.              List<Cursos> cursos = alunoDAO.listarTodosCursoQueParticipa(codigoAluno);
13.              out = new DataOutputStream((OutputStream) response.getOutputStream());
14.              if (cursos != null) {// ou seja, existe cursos.........
15.
16.                  out.writeInt(cursos.size());
17.                  for (Cursos cursosl : cursos) {
18.                      out.writeUTF(cursosl.getNomeCurso());
19.                      out.writeUTF(cursosl.getDescricaoResumidaCurso());
20.                  }
21.              } else {
22.                  out.writeInt(0);// não possui nenhum curso cadastrado para esse aluno.
23.
24.              }
25.          } catch (SQLException e) {
26.              throw new ServletException(e);
27.          } finally {
28.              if (in != null) {
29.                  in.close();
30.              }
31.              if (out != null) {
32.                  out.close();
33.              }
34.          }
35.      }
36.  }
```

Como comentado anteriormente, a outra funcionalidade que um usuário com o nível de Aluno, no AVA móvel, pode efetuar é a visualização das tarefas de um determinado curso que participa. Na Figura 39 apresentam-se as telas que representam tal funcionalidade, ou

seja, primeiramente o aluno deve selecionar um dos cursos que participa e em seguida são listadas as tarefas do curso escolhido.

Figura 39 : Tarefas de um respectivo curso

Na Figura 37 é apresentado o *Servlet* responsável por enviar todas as tarefas de um respectivo curso.

Figura 40 : Servlet VisualizarTarefas

```
01. public class VisualizarTarefas extends HttpServlet {
02.
03.     private DataInputStream in = null;
04.     private DataOutputStream out = null;
05.
06.     protected void doPost(HttpServletRequest request, HttpServletResponse response)
07.             throws ServletException, IOException {
08.
09.         in = new DataInputStream((InputStream) request.getInputStream());
10.         int idCurso = in.readInt();
11.         try {
12.
13.             TarefaDAO tarefaDAO = new TarefaDAO();
14.             List<Tarefa> tarefas = tarefaDAO.listarTarefasCurso(idCurso);
15.             out = new DataOutputStream((OutputStream)response.getOutputStream());
16.             if (tarefas != null) {
17.                 out.writeInt(tarefas.size());
18.                 for (Tarefa tarefa : tarefas) {
19.
20.                     out.writeUTF(tarefa.getAssuntoTarefa());
21.                     out.writeUTF(tarefa.getDescricaoTarefa());
22.                     out.writeUTF(tarefa.getDataLiberadaTarefa());
23.                     out.writeUTF(tarefa.getDataEntregaTarefa());
24.                 }
25.
26.             } else {
27.                 out.writeInt(0);//não possui tarefas para esse aluno
28.             }
29.         } catch (SQLException e) {
30.             e.printStackTrace();
31.         }
32.     }
33. }
```

Como ocorre com os demais Servlets, o método `doPost()` é executado. Nesse método cria-se um objeto `DataInpuStream` para obter os dados enviados pela aplicação móvel. Após ter obtido o *id* (identificador) do curso por meio do método `readInt()`, pode-se listar todas as tarefas cadastradas na base de dados para aquele respectivo curso, isso é possível por meio do método `listarTarefasCurso()` do objeto `TarefaDAO`. Na linha 15 cria-se um objeto `DataOutputStream`, que é responsável por enviar informações das tarefas para o dispositivo móvel. A linha 17 envia o número de tarefas que o respectivo curso possui e as linhas 18 até 24 são responsáveis por fazer a iteração na lista que contém as tarefas, sendo que as informações relevantes sobre as tarefas são passadas como parâmetro nos métodos `writeUTF()` do objeto `DataOutputStream`.

3.6. Implementação da aplicação móvel on-line

Este módulo do trabalho tem como objetivo implementar a adaptabilidade de AVA Web para o AVA móvel utilizando as tecnologias WAP (*Wireless Application Protocol*), *WML* (*Wireless Markup Language*) e JSP (*JavaServer Pages*) a fim de implementar uma aplicação móvel *on-line*. Neste módulo foi implementada apenas a visualização dos cursos que o aluno participa do nível de usuário Aluno. Deixando as demais funcionalidades para trabalhos futuros.

Segundo Teixeira *et al.* (2004), a tecnologia *WAP* (*Wireless Application Protocol*) está posicionada na convergência de duas tecnologias de rede que estão evoluindo muito rapidamente: a transmissão de dados sem fio e a *Internet.* As aplicações *WAP* são especificadas por meio de um conjunto de formatos bem conhecidos, baseados nos conteúdos utilizados na *Internet* tradicional, sendo assim os dados são transportados utilizando os protocolos de comunicação da *Internet*.

Os dispositivos móveis que utilizam a tecnologia *WAP* possuem um tipo de micro *browser* de uma forma análoga ao *browser* convencional. De acordo com Teixeira *et al.* (2004*)*, o micro *browser* interpreta e exibe o conteúdo desenvolvido para o ambiente *WAP*. Este conteúdo é criado com a ajuda de uma linguagem de *script* chamada *WML* (*Wireless Markup Language*)*,* que é semelhante ao *HTML* (*HyperText Markup Language*) que é uma linguagem utilizada para construir sites na Internet. No entanto, o *WML* foi construído para atender as necessidades dos dispositivos e redes sem fio com baixo poder de processamento.

Maiores informações sobre o protocolo *WAP* e *WML* são obtidas no Capítulo 1 desta monografia.

Tanto o HTML quando o WML geram conteúdo estático, ou seja, não possuem recursos para que os resultados obtidos através de consultas a bancos de dados, por exemplo, sejam formatados e exibidos no *browser*. Para gerar conteúdo dinâmico foi utilizado no desenvolvimento desse módulo a linguagem de programação JSP (*JavaServer Pages*) (Seção 1.4.3) em conjunto com o WML.

Utilizando a API JSTL (*JavaServer Pages Standard Tag Library*) (Java Sun, 2008) é possível fazer consulta ao banco de dados de maneira fácil em uma página JSP. Na Figura 41 é mostrado o código que tem como função listar todos os cursos que o aluno participa.

Figura 41 : Código JSP + WAP

```
1.  <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
2.  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
3.  <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
4.  "http://www.wapforum.org/DTD/wml_1.1.xml">
5.  <sql:setDataSource var="dataSource"
6.                     driver="com.mysql.jdbc.Driver"
7.                     user="root"
8.                     password="senha"
9.                     scope="session"
10.                    url="jdbc:mysql://localhost/tcc"></sql:setDataSource>
11. <%
12.             response.setContentType("text/vnd.wap.wml");
13. %>
14. <wml>
15.     <card id="MainCard" title="This is a first card">
16.         <sql:query dataSource="${dataSource}" var="alunos">
17.             select cursos.nome_curso, aluno.nome_usuario from aluno, cursos, matriculas where aluno.id_usuario = matriculas.id_usuario and
    cursos.id_curso = matriculas.id_curso and aluno.id_usuario = ?
18.             <sql:param value="2"/>
19.         </sql:query>
20.
21.         <table columns="5">
22.             <tr><td>CURSO</td></tr>
23.             <c:forEach var="aluno" items="${alunos.rows}">
24.                 <tr><td>${aluno.nome_curso}</td></tr>
25.
26.             </c:forEach>
27.         </table>
28.
29.     </card>
30. </wml>
```

As linhas 1 e 2 do código apresentado na Figura 41 faz com que seja possível utilizar a API JSTL. Ressalta-se que estão sendo utilizadas somente as APIS *core* e *sql*, representadas respectivamente pelos *prefixs* “c” e “sql”. O prefixo “c” significa *core*, esta API contém as funções genéricas, tais como, *if* (condição), *for* (iteração), etc. O prefixo representado por

"sql" como o próprio nome diz, possui *tags* específicas para manipular um respectivo banco de dados na aplicação.

Para fazer a conexão com o banco de dados da aplicação, a API JSTL disponibiliza um *tag* chamada *setDataSource*, a mesma é responsável por fazer a ligação entre a aplicação com o banco de dados. Na Figura 42 é apresentado a especificação dessa *tag* para a aplicação AVA móvel on-line.

Figura 42 : tag <set:DataSource...

```
<sql:setDataSource var="dataSource"
                   driver="com.mysql.jdbc.Driver"
                   user="root"
                   password="senha"
                   scope="session"
                   url="jdbc:mysql://localhost/tcc"></sql:setDataSource>
```

A *tag setDataSource* possui vários atributos que estão explicados a seguir:

- var: este atributo cria uma variável.
- driver: este atributo tem com função informar para a *tag setDataSource* qual *driver* está utilizando para fazer a conexão com o banco de dados.
- user: este atributo representa o usuário do respectivo banco de dados
- password: este atributo representa a senha do respectivo banco de dados
- scope: determine quanto tempo a conexão deve permanecer.
- url: informa o caminho do banco de dados e o nome do mesmo.

Para tornar possível que os emuladores de aplicações *WAP* possam interpretar o código listado na Figura 41, é necessário configurar o *MIME* (*Multipurpose Internet Mail Extensions*) *types*, por meio do objeto `response` do tipo `javax.servlet.http.HttpServletResponse`. Para isso é necessário utilizar o método `setContentType("text/vnd.wap.wml")`, que garante que o *MIME* correto seja configurado para o documento *WML*.

A fim de gerar documentos dinâmicos na aplicação *WAP* foi utilizado a *tag* ilustrada na Figura 43. Essa *tag* tem dois atributos que são:

- *dataSource*: utilizado para definir o *data source* que foi definido em uma outra *tag*, apresentada na Figura 41. Como pode ser observado, o atributo *var* na Figura 41 tem o mesmo nome que o atributo *dataSource* da Figura 43 utiliza.

- *var*: cria uma variável, tornando possível para o desenvolvedor utilizá-la em outras *tags*.

Figura 43 : tag <set: query ...

```
<sql:query dataSource="${dataSource}" var="alunos">
          select cursos.nome curso, aluno.nome usuario from aluno, cursos, matriculas where aluno.id_usuario = matriculas.id_usuario and
cursos.id_curso = matriculas.id_curso and aluno.id_usuario = ?
          <sql:param value="2"/>
      </sql:query>
```

Como pode ser observado na Figura 43, dentro da tag <*sql: query*> tem-se um respectivo código *sql*, que é responsável por listar todos os cursos que um aluno participa. Salienta-se que para informar qual o id do aluno utiliza-se a *tag* <*sql:param value=”2”/*>. O conteúdo do atributo *value*, no caso *“2”* na Figura 43, é informado como parâmetro no `select`, representado pelo símbolo “?”. Após a execução do *sql* que retorna todos os cursos de um respectivo aluno, é utilizada a *tag* <*c:forEach*>, a fim de iterar sobre as linhas resultantes da consulta. Na Figura 44 é apresentada a parte do código responsável por fazer tal iteração.

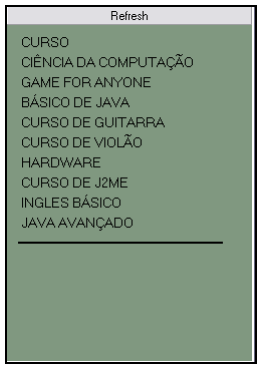
Figura 44 : tag <c:forEach>

```
<table columns="5">
            <tr><td>CURSO</td></tr>
            <c:forEach var="aluno" items="${alunos.rows}">
                <tr><td>${aluno.nome_curso}</td></tr>

            </c:forEach>
        </table>
```

O código listado na Figura 44 cria uma tabela por meio da *tag* <*table*>, sendo que a *tag* <*c:forEach*> dinamicamente criará várias linhas dependendo de quantas tarefas o atributo *items* da *tag* <*c:forEach*> possui. A ferramenta de desenvolvimento utilizada na implementação foi o NetBeans e o emulador utilizado foi o EasyPad WAPtor. Na Figura 45 é mostrada a lista de cursos de um determinado aluno.

Figura 45 : EasyPad WAPtor



3.7. Considerações Finais

Neste capítulo busca-se apresentar que a proposta da arquitetura M-AVA de Bartholo (2008) que visa estabelecer a adaptabilidade de ambientes virtuais de aprendizagem para dispositivos móveis é viável e pode ser implementada utilizando a tecnologia WAP.

Durante o desenvolvimento da aplicação móvel verificou-se que a arquitetura M-AVA permite também a extensão de um AVA para aplicações móveis utilizando J2ME , porém esse tipo de aplicação precisa ter no dispositivo móvel a aplicação, ou seja, o software precisa estar instalado no dispositivo móvel.

O estudo de caso apreesentado contribuiu para o refinamento da arquitetura M-AVA que a princípio não considerada a adaptação pelo perfil do usuário.

Segundo consta em Bartholo (2008) a adaptação pelo perfil do usuário é proporcionada pela mudança na interface de acordo com o tipo de usuário e pelas informações e funcionalidades que podem ser disponibilizadas para cada um. Como na implementação são utilizados três usuários diferentes (administrador, aluno e professor), e que possuem possibilidades de acessos diferentes.

CONCLUSÃO

Considerações Iniciais

Neste capítulo é apresentado um resumo do trabalho realizado, as indicações das contribuições obtidas e as principais limitações. Além disso, são citadas sugestões de trabalhos futuros decorrentes do trabalho realizado.

Resumo do Trabalho Realizado

Neste trabalho foi apresentada a computação móvel (Capítulo 1) e as possíveis tecnologias utilizadas no desenvolvimento de aplicações móveis. Este trabalho teve como objetivo utilizar técnicas de adaptabilidade de aplicações *Web* para dispositivos móveis. Foram pesquisadas na literatura técnicas de adaptabilidade para dispositivos móveis (Capítulo 2) e optou-se por utilizar a arquitetura M-AVA, proposta por Bartholo (2008), a qual foi descrita na Seção 2.4.

A arquitetura M-AVA trata a adaptabilidade de AVAs para dispositivos móveis. Neste trabalho, tal arquitetura foi implementada e validada. Para isso, foi criada uma aplicação *WEB* de um AVA utilizando a tecnologia Java (Seção 3.2.2) e a partir desta, aplicou-se a adaptabilidade para dispositivos móveis (Seções 3.2.3 e 3.2.4).

A aplicação *WEB* foi criada utilizando tecnologia Java, mais especificamente *Servlets*, *JSP (JavaServer Pages)* e outras API's e também foi utilizado o banco de dados relacional MySQL. Para criar a aplicação *WEB* foi necessário estudar *Servlets* e *JSP* e as API's utilizadas na aplicação, tais como *JavaMail, FileUpload,* etc. A aplicação móvel foi implementada de duas maneiras: uma utilizando J2ME e a outra utilizando *WAP*. A aplicação J2ME faz a interação com o servidor Apache por meio do protocolo *HTTP* utilizando o *GCF (Generic Connection Framework)*, que é um *framework* com um conjunto de interfaces que são definidas no pacote `javax.microedition.io`, utilizando a *interface* `HtttpConnection` que tem o propósito de enviar e receber dados no protocolo *HTTP*. Para enviar e receber dados, a *MIDlet* utiliza os objetos `DataOutputStrem` e `DataInputStream`, que podem ser encontrados no pacote `java.io`. A aplicação construída com *WAP* utiliza a linguagem *WML*, que é similar ao *HTML*. Tanto o *HTML* quanto o *WML* geram conteúdos estáticos e, para tornar a geração desses conteúdos dinâmicos com *WAP*, foi necessário estudar *WAP* em conjunto com *JSP*.

Contribuições do trabalho

Como contribuição deste trabalho de conclusão de curso tem-se a avaliação da arquitetura M-AVA (Bartholo, 2008), por meio de sua implementação. Como resultado dessa avaliação observou-se que a arquitetura M-AVA é viável e pode ser implementada utilizando a tecnologia *WAP*, de acordo com as indicações de Bartholo (2008).

Outra contribuição obtida foi a possibilidade de estender a Arquitetura M-AVA para o desenvolvimento da aplicação móvel utilizando J2ME, lembrando que nesse tipo de aplicação o software precisa estar instalado no dispositivo móvel, sendo necessário que o dispositivo móvel tenha disponível JVM.

Ainda no desenvolvimento da aplicação móvel, este trabalho contribuiu com o refinamento da arquitetura M-AVA que a princípio não possuía adaptação pelo perfil do usuário. Este refinamento ocorreu durante o desenvolvimento das aplicações móveis, em que foi observado que existiam usuários diferentes e que estes eram tratados de maneira diferente de acordo com o seu perfil.

Trabalhos Futuros

Como trabalhos futuros de pesquisa foram identificados para dar continuidade ao trabalho realizado neste trabalho de conclusão de curso:

- A implementação da arquitetura M-AVA na tecnologia Android ;
- O paradigma orientado a aspectos para a definição da arquitetura e a utilização do AspectJ para a definição a implementação da adaptabilidade de AVAs para dispositivos móveis.

REFERÊNCIAS

(BARTHOLO, 2008) BARTHOLO, V. F., ADAPTABILIDADE DE AMBIENTES VIRTUAIS DE APRENDIZAGEM PARA DISPOSITIVOS MÓVEIS. 2008. Projeto de Qualificação apresentado ao Programa de Mestrado Stricto Sensu em Ciência da Computação do Centro UNIVEM – Universitário Eurípides de Marília, Marília, SP.

(BASHAM, 2008) BASHAM, BRYAN.; SIERRA, KATHY.; BATES, BERT. HEAD FIRST – Servlets & JSP. ed 2. Sebastopol - CA: O'Reilly Media. 2008 p.863

(BOOCH, 1994) Booch, G. (1994). Object**–Oriented Analysis and Design with Applications**. Benjamin/Cummings, second edition.

(BROWN E WALLNAU, 1996) BROWN, A.W., K.C WALLNAU (1996). **Component-Based Software Engineering**. IEEE Computer Society Press.

(CAMARGO, 2006) CAMARGO, V.V., **FRAMEWORKS TRANSVERSAIS: DEFINIÇÕES, CLASSIFICAÇÕES, ARQUITETURA E UTILIZAÇÃO EM UM PROCESSO DE DESENVOLVIMENTO DE SOFTWARE.** 2006. Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação – ICMC – USP, São Carlos.

(CONALLEN, 1999) CONALLEN, Jim. **. Building web applications with UML**. (Boston): Addison-Wesley, 1999. 300p.(Addison- Wesley object thecnology series)

(DANTAS e BORDA, 2003) DANTAS, A. e BORBA, P. ,**DEVELOPING ADAPTIVE J2ME APPLICATIONS USING ASPECTJ.** In Proceedings of the 7th Brazilian Symposium on Programming Languages, pages 226–242, May 2003

(DEITEL e DEITEL, 2003) DEITEL, H. M.; DEITEL, P. J.. . Java: como programar. 4ª ed. Porto Alegre: Bookman, 2003. 1386p.

(DEITEL e DEITEL, 2005) DEITEL, H.M., DEITEL, P.J., **Java: Como Programar**. Editora Prentice-Hall. ISBN: 8576050196. 2005. 6ª.ed., p. 1152.

(DIJKSTRA, 1976) DIJKSTRA, E. W. A Discipline of Programming. Prentice-Hall, 1976.

(GIALDI, 2004) GIALDI, M.V.. **UM MODELO PARA PORTAIS MÓVEIS BASEADO EM MIDDLEWARE REFLEXIVO E AGENTES MÓVIES**. 2004. Dissertação

(Mestrado) – Universidade Estadual de Campinas, Instituto de Computação. Campinas – São Paulo.

(GOMES, 2005) GOMES, W. E.C.P., **UMA PLATAFORMA DE DESENVOLVIMENTO DE SOFTWARE BASEADO EM COMPONENTES PARA DISPOSITIVOS MÓVEIS**, 2005. Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação. Campinas – SP

(HORSTMANN e CORNEL, 2000) HORSTMANN, Cay S; CORNELL, Gary. **Core Java 2: Volume I – Fundamentals**. 5ª ed.Palo Alto: Prentice Hall, 2000.

(Java Sun, 2008) Java Sun Developer Network. Disponivel em <http://java.sun.com/products/jsp/jstl/>. Acesso em: 15 de março de 2008.

(Junior, 2008) Junior, Givanildo Francisco da Silva .;WAP – WIRELESS APPLICATION PROTOCOL.Disponível em <http://www.wirelessbrasil.org/wirelessbr/colaboradores/givanildo_wap/wap_01.html>; Acesso em *: 31 de outubro de 2008).*

(KEPHART, 2003) KEPHART, J. O. e CHESS, D. M. **The Vision of Autonomic Computing**. IEEE Computer Magazine, jan., 2003.

(KULESZA, 2006) KULESZA, R., **UMA CAMADA DE ADAPTAÇÃO PARA TRANSMISSÃO DE MÍDIAS DIGITAIS.** 2006. Dissertação de Mestrado em Engenharia Elétrica da Escola Politécnica da Universidade de São Paulo, São Paulo.

(LOUREIRO, 2006) LOUREIRO, A.A.F., SADOK, D.F.H., MATEUS, G.R.,NOGUEIRA, J.M.S., KELNER,J..**Computação Sem Fio e Computação Móvel: Tecnologias, Desafios e Oportunidades.** Minicurso apresentado no Congresso da Sociedade Brasileira de Computação, Campinas, São Paulo, agosto de 2003.

(MARTINS, 2003) Renato Passarinho, Diga "Alô" ao J2ME: Um tutorial de programação.

(MENKHAUS, 2002) MENKHAUS, Guido**. Adaptive User Interface Generation in a Mobile Computing Environment.** PhD thesis, Salzburg University, 2002.

(MEYER, 1997) Meyer, B. (1997). **Object–Oriented Software Construction**. Prentice–Hall, second edition.

(MUCHOW, 2001) MUCHOW, John W.. **Core J2ME Technology & MIDP**. Prentice Hall, 2001.

(PIVETA, 2001) PIVETA, Eduardo. **Um modelo de suporte a programação orientada a aspectos**. Dissertação de Mestrado. Universidade Federal de Santa Catarina, 2001.

(SOUZA, 2004) SOUZA, A.D.D. de, **STRUCTURING ADAPTIVE APPLICATIONS USING ASPECTJ.** 2004. Dissertação de Mestrado. Universidade Federal de Pernambuco, Pernambuco.

(TEIXEIRA, 2004) TEIXEIRA, Pereira; BARBOSA, Felipe; DIAS, Clenerson; TAVARES, Emerson. I WorkComp Sul, Florianópolis – SC – 2004.

APÊNDICE A

O objetivo deste apêndice é apresentar a descrição dos requisitos funcionais e não funcionais, a partir de um estudo exploratório do Moodle (utilização do ambiente e análise de sua documentação), do protótipo de software a ser desenvolvido neste trabalho.

Documento de Requisitos – Protótipo AVA

A – VISÃO GERAL DO SISTEMA

O sistema tem como objetivo ser uma ferramenta de gerenciamento de cursos a distância, possibilitando que educadores criem com facilidade cursos on-line de qualidade. Consiste basicamente do gerenciamento de alunos, professores, cursos, material, agenda. O sistema deve ainda emitir diversas consultas, possibilitando um melhor gerenciamento educacional.

B – REQUISITOS FUNCIONAIS

1. O sistema deve permitir a inclusão, alteração e remoção de USUÁRIOS, contendo os seguintes atributos: nome, endereço, cidade onde mora, estado, país, telefone, e-mail. Sendo que os usuários podem ser classificados como:

- **Visitante:** Pode acessar o ambiente e as informações constantes da tela de abertura do ambiente. Pode visitar disciplinas que permitam o acesso de visitantes (sem código de inscrição) e ver o conteúdo delas.
- **Aluno:** Usuário matriculado em um ou mais cursos. Tem acesso a todas as atividades e materiais dos cursos.
- **Professor:** Tem acesso aos cursos em que está designado como professor e pode incluir ou remover atividades e materiais.
- **Administrador:** Tem acesso a todas as instâncias da instalação e pode modificá-las. O administrador possui controle total sobre o ambiente desde o design até registros de operações muito bem detalhados.

2. O sistema deve permitir a inclusão, a alteração e a remoção de CURSOS, podendo ser configurado pelo professor as seguintes características:

- Nome do curso.
- Criação de um código para o curso.
- Uma apresentação breve.
- Formatos do curso (semanal, por tópicos, social e simples).
- Definição da data de início do curso.
- Quantidade de semanas que será usada no curso ou tópicos.
- Acesso reservado (permite bloquear qual usuário terá acesso ao curso).
- Mostrar nota (habilitar ou desabilitar a opção de visualização das notas atribuídas ao aluno (boletim));
- Tamanho máximo para upload (refere-se ao tamanho limite do arquivo a ser enviado);

3. O sistema deve permitir a inclusão, a alteração e a remoção NOTAS, através de um boletim em que o professor poderá lançar notas de acompanhamento dos alunos em um curso. Sendo necessário para isso uma descrição da avaliação, a nota da avaliação feita e ainda o peso da avaliação, em que o peso servirá para que o professor determine a média do aluno de acordo com as notas que ele obteve no curso.

4. Cada curso pode trabalhar com diversas atividades, como TAREFAS, CHAT, FÓRUM, MATERIAIS, sendo que as atividades (serão descritas com mais detalhes abaixo)

devem ser cadastradas e escolhidas pelo professor. Quando uma nova atividade for proposta deve ser notificado nas Notícias do Curso(na WEB) e enviado um e-mail para os alunos cadastrados no curso.

5. O sistema deve permitir a inclusão, a alteração e a remoção de TAREFAS, podendo ser configuradas pelo professor e relacionadas a um curso. Tarefas: este módulo permite aos professores a proposição de trabalhos e recebimento dos trabalhos submetidos pelos estudantes, podendo ser estabelecido um prazo determinado para a submissão das tarefas. Características necessárias das tarefas são: nome da tarefa, descrição, data e horário da disponibilização da tarefa, data e horário do término da tarefa, permitir ou não envio de arquivos, qual tamanho do arquivo permitido, envio de e-mail para o professor informando a postagem da resolução da tarefa.

6. Ao ser criado um curso automaticamente é liberado para ele um CHAT. Este módulo promove a comunicação entre os estudantes e os professores que se encontram conectados ao ambiente naquele instante. Para o CHAT são necessárias as características: nome da sala, assunto, data horário do próximo *chat.*

7. O professor poderá liberar, de acordo com cada curso um FÓRUM. Este módulo é ferramenta de discussão que possibilita debates entre alunos e professores através do ambiente. Características necessárias são: nome, assunto, data postagem.

8. Todo curso tem a possibilidade de incluir, alterar ou remover MATERIAIS. Os materiais são todos os tipos de conteúdos que serão apresentados no curso. Podem ser documentos arquivados no servidor, páginas criadas com o uso dos editores de textos ou arquivos de outros sítios visualizados no ambiente do curso.

9. Toda tela de curso deve possuir NOTICIAS, que serão incluídas, excluídas ou removidas pelo professor. Sendo que estas notícias deverão conter informações a respeito do que será feito no curso, quais materiais foram disponibilizados, qual fórum está agendado, etc., funcionando muitas vezes como uma agenda, fornecendo o roteiro dos acontecimentos do curso.

10. Um aluno pode freqüentar vários cursos. Sendo possível que quando o aluno entre com o seu *login*, verifique em quais cursos ele está cadastrado.

11. O professor ao cadastrar um curso poderá informar quem serão os participantes (alunos) do curso, sendo que deverá estar disponível para ele uma lista dos alunos já cadastrados no AVA, podendo definir se estes poderão fazer parte deste novo curso e ainda a possibilidade de cadastrar novos usuários.

C – REQUISITOS NÃO FUNCIONAIS

C1. Confiabilidade

12. O sistema deve ter capacidade para recuperar os dados perdidos da última operação que realizou em caso de falha.

C2. Eficiência

13. O sistema deve responder a consultas em menos de cinco segundos.

C3. Portabilidade

14. O sistema deve ser executado em computadores com sistema operacional Linux ou Windows.

C4. Usabilidade

15. O sistema deve ser fácil de manter e permitir evoluções na interface com o usuário. Para tanto, ele deve utilizar arquitetura em 3 camadas (*Model/Viewer/Control*).

C5. Manutenibilidade

16. O sistema deve permitir evoluções nas suas funcionalidades.