ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

UMA APLICAÇÃO TELNET PARA DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso

Engenharia da Computação

Renato Augusto Gomes Pina França
Orientador: Prof. Sérgio Castelo Branco Soares

Recife, 26 de dezembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

UMA APLICAÇÃO TELNET PARA DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso

Engenharia da Computação

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

Renato Augusto Gomes Pina França
Orientador: Prof. Sérgio Castelo Branco Soares

Recife, 26 de dezembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

Renato Augusto Gomes Pina França

UMA APLICAÇÃO TELNET PARA DISPOSITIVOS MÓVEIS

Resumo

Este trabalho apresenta uma abordagem para a configuração de dispositivos de rede via celular através de uma aplicação desenvolvida utilizando Java ME, a linguagem da plataforma Java específica para o desenvolvimento de aplicações para dispositivos de recursos limitados, tais como celulares e PDAs.

Esta nova abordagem visa unir as vantagens e comodidades advindas dos dispositivos móveis às necessidades dos gerentes e administradores de rede no trabalho de manter e administrar corretamente a rede de sua empresa. Desta forma, em casos de emergência, falhas ou uma desconfiguração de um dos dispositivos da rede, uma nova configuração poderá ser feita sem que haja a necessidade do deslocamento até a empresa.

A aplicação consiste de um cliente Telnet simples, que atende a todas as restrições impostas pelos dispositivos móveis, mas que é capaz de realizar as funcionalidades básicas pertinentes a uma aplicação de cliente Telnet. Estas funcionalidades incluem estabelecimento de conexão remota, armazenamento de hosts, negociação de opções do Telnet, entre outras. A aplicação na sua versão atual lida mais especificamente com roteadores.

Abstract

This work presents an approach to configure network devices through mobile phones by an application developed using Java ME, the programming language of the Java platform that is used to develop application to devices with limited resources, such as mobile phones and PDAs.

The presented approach aims in joining benefits and comfort of mobile devices to the needs of network managers and administrators. Therefore, in situations of emergencies, flaws, or misconfiguration of a network device, a new configuration can be done without the need to be physically present in the company.

The application consists of a Telnet client that complies with all mobile devices constraints, however being capable to execute the basic functionalities of a Telnet application. These functionalities include remote connections, hosts registering, Telnet options negotiation, and others. The current application version deals specifically with routers configuration.

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

CLI – Command Line Interface
SNMP – Simple Network Management Protocol
HTTP – Hyper Text Transfer Protocol
PDA – Personal Digital Assistant
Kbps – Kilobit per second
TDMA – Time Division Multiple Access
CDMA – Code Division Multiple Access
GSM – Global System for Mobile Communications
GPRS – General Packet Radio Service
EDGE – Enhanced Data rates for Global Evolution
Mbps – Megabit per second
GPS – Global Positioning System
API – Application Programming Interface
JSR – Java Specification Request
RMI – Remote Method Invocation
JNI – Java Native Interface
FTP – File Transfer Protocol
RFC – Request for Comments
ARPA – Advanced Research Project Agency
OSI – Open Systems Interconnection
MAC – Media Access Control
RPC – Remote Procedure Calling
GPL – General Public License

Agradecimentos

Gostaria de agradecer primeiramente a **Deus** por me dar serenidade, coragem, discernimento e sabedoria para escolher o tema utilizado, me ensinando a expressar e colocar no papel todas as idéias que gostaria de por em prática.

A meus pais e minha irmã pela paciência que tiveram comigo diante meu estresse, nervosismo e até às vezes minhas ignorâncias por conta de utilizar dia e noite o computador.

Aos meus colegas de turma, Carlos Adriano, Ronaldo, Paulo André (o chefe), Valmir e a todos os outros que direta ou indiretamente contribuíram para que eu chegasse ao término deste trabalho, que serviu para meu crescimento como pessoa humana e profissional, dando-me a oportunidade de conviver com controvérsias de opiniões em relação ao mesmo assunto e aprendendo com situações sempre inesperadas.

Ao meu professor e orientador Sérgio Soares, pela dedicação e responsabilidade com que assumiu mais esta tarefa, não permitindo que, em momento algum eu me sentisse desamparado ou sem respostas para minhas dúvidas e agradeço principalmente por não permitir em momento algum que eu desistisse e nem se quer desanimasse da minha idéia.

Aos professores Carlos Alexandre, Ricardo Massa e Adriano Lorena por me suprir de conhecimento durante os 5 anos que estive na Poli. A todos os demais professores por tudo que fizeram por mim e por meu futuro como um Engenheiro da Computação.

Capítulo 1

Introdução

A velocidade no tráfego de dados e informações é um fator chave no processo de comunicação. Se tomarmos os meios de comunicação de massa, como rádio, televisão e a Internet [1], por exemplo, temos uma situação análoga à da informação impressa: grande massa de dados fluindo direto do produtor ao receptor humano, o qual deve, de alguma forma, integrá-los como conhecimento.

Por conta das inovações tecnológicas, a comunicação de dados atualmente atinge uma velocidade compatível com o volume de dados que se produz e se consome. Novamente aqui é necessário que a informação transite automaticamente do produtor ao receptor, uma vez que a presença do ser humano no meio do processo implica numa perda radical de eficiência. Um exemplo de como a dependência humana interfere no processo da comunicação pode ser observada na necessidade da intervenção manual do administrador de redes para a configuração de um roteador [2,3], responsável por toda comunicação da rede interna de uma empresa com a Internet.

Entende-se por intervenção manual aquela que obriga um administrador de rede a entrar em contato direto com um dispositivo de interesse e proceder com a configuração, sem o auxílio de um sistema ou software que automatize tal processo. Tal intervenção é realizada utilizando-se alguns protocolos de comunicação para contatar o dispositivo de interesse, tais como Telnet/CLI (*Command Line Interface*) [4,5], SNMP (*Simple Network Management Protocol*) [6] ou HTTP (*Hyper Text Transfer Protocol*) [7] (para dispositivos que possuam uma interface *web* de configuração). Entretanto, em todas estas possibilidades, há a necessidade de um computador conectado diretamente ao dispositivo a ser configurado, ou ligado através de uma rede, o que nem sempre está disponível.

Uma solução para resolver este problema é a utilização dos dispositivos móveis e das redes sem fio [8], particularmente pela possibilidade de integração com as redes tradicionais, agregando uma maior flexibilidade quanto à localização do administrador de rede da empresa, podendo este, em momentos de emergência, solucionar um eventual problema de onde quer que esteja.

Esta nova abordagem abriria uma gama de novas oportunidades para o gerenciamento e administração das redes de computadores, além de propiciar uma alternativa aos métodos tradicionais para a configuração dos dispositivos de rede, gerando benefícios tanto para administradores quanto para as empresas.

A utilização das redes sem fio para este proveito seria possível mediante a utilização de aplicações específicas para dispositivos móveis como telefones celulares, PDAs (*Personal Digital Assistants*) e pagers. Estas aplicações devem seguir certas regras e atender a várias limitações para serem corretamente implementadas. Limitações como baixo poder de processamento, memória disponível e interfaces com o usuário limitadas podem ser determinantes durante a construção destas aplicações.

Uma das maneiras mais disseminadas de obedecer estas regras e limitações na construção de aplicações para dispositivos móveis é a utilização da linguagem de programação J2ME [9], ou como é conhecida atualmente, Java ME (*Java Micro Edition*). A plataforma Java ME foi especificamente projetada pela Sun para atender às limitações de recursos dos dispositivos móveis, e através de suas Configurações e Perfis [10] determina o escopo de funcionalidades de uma aplicação de acordo com o dispositivo móvel alvo.

A Configuração CLDC (*Connected Limited Device Configuration*) [11] e o Perfil MIDP (*Mobile Information Device Profile*) [12] determinam uma plataforma mínima suportada por dispositivos móveis, como aparelhos celulares, agregando certas funcionalidades, como armazenamento persistente, conectividade e manipulação de eventos, a estes dispositivos.

Através de uma aplicação utilizando a Configuração CLDC e o Perfil MIDP da plataforma Java ME, este trabalho provê um meio de unir as vantagens e os benefícios advindos dos dispositivos móveis às necessidades dos administradores e gerentes de redes, permitindo a configuração e a simplificação do gerenciamento de dispositivos de rede. Ou seja, dar uma solução alternativa à configuração destes dispositivos sem que haja a necessidade da intervenção manual local do administrador ou gerente de rede.

Este trabalho está dividido em 4 capítulos. Neste primeiro, são apresentados as motivações, os objetivos e o ambiente ao qual este trabalho está inserido.

O Capítulo 2 especifica alguns conceitos básicos sobre as tecnologias envolvidas na aplicação desenvolvida por este trabalho. São apresentados os conceitos, fundamentos e a plataforma da linguagem de programação Java ME, bem como, os conceitos e o funcionamento do protocolo Telnet e dos roteadores.

No Capítulo 3 é apresentada a aplicação desenvolvida, a metodologia utilizada, alguns casos de uso importantes e uma explicação sobre as funcionalidades dos componentes utilizados no desenvolvimento desta aplicação. O capítulo mostra também quais as dificuldades encontradas durante o desenvolvimento desta aplicação e diretrizes a serem consideradas no desenvolvimento de aplicações para dispositivos móveis.

Finalmente, no Capítulo 4, são discutidas as conclusões, limitações, trabalhos relacionados e propostas para trabalhos futuros.

Capítulo 2

Tecnologias Utilizadas

Neste capítulo será explanado todos os conceitos básicos necessários para um bom entendimento das tecnologias e metodologias envolvidas no desenvolvimento da aplicação proposta por este trabalho.

2.1 Comunicação móvel

As tecnologias da telefonia móvel evoluíram em diversas fases principais denominadas "Gerações" ou simplesmente "G". Basicamente existem três destas gerações, cada uma provendo mais flexibilidade e melhores serviços que as anteriores [13].

2.1.1 Primeira Geração

Esta geração é simplesmente referenciada por "primeira geração" ou basicamente por 1G. Esta geração foi desenvolvida durante os anos 80 e início dos anos 90. Ela provia apenas serviços analógicos de voz e nenhum tipo de serviço de dados.

2.1.2 Segunda Geração

A segunda geração (2G) de tecnologias móveis utilizavam redes digitais, baseadas em circuitos. Como redes 2G eram digitais, elas eram capazes de suportar transmissões de dados a uma taxa de velocidade de aproximadamente 9.6 Kbps. Os três padrões envolvidos nas redes 2G são: o acesso múltiplo por divisão do tempo (TDMA - *Time Division Multiple Access*), O acesso múltiplo por divisão de código (CDMA - *Code Division Multiple Access*) e o sistema móvel global (GSM - *Global System for Mobile Communications*) [14].

Como redes 2G suportam transmissão de dados, elas são aptas a suprir telefones que suportam Java ME. Alguns fabricantes desenvolveram telefones com Java ME para redes 2G, porém a grande maioria está projetando seus aparelhos com suporte a Java para as redes 2.5G e 3G, onde maiores velocidades de transmissão de dados e de largura de banda tornam estas aplicações mais usáveis.

2.1.3 2.5G

2.5G é um acrônimo que representa os vários melhoramentos da tecnologia às redes móveis de segunda geração que buscam aumentar o numero de usuários que a rede pode atender impulsionando as transmissões de dados a taxas em torno de 56 Kbps. Os aperfeiçoamentos advindos das redes 2.5G foram projetados para sobrepor as redes de segunda geração com um mínimo adicional de infra-estrutura. Exemplos destas tecnologias incluem: GPRS (*General Packet Radio Service*) e EDGE (*Enhanced Data rates for Global Evolution*) [15]. Elas se caracterizam por serem tecnologias baseadas em pacotes e por manterem um alto índice de conectividade.

A introdução das redes 2.5G atualmente torna possível às operadoras móveis de desenvolverem novas experiências de mercado provendo serviços de alta velocidade de dados, demonstrando as possibilidades que serão oferecidas pela terceira geração (3G).

2.1.4 3G

A terceira geração de sistemas móveis de comunicação fornece uma gama de serviços de transferência de voz, vídeo, textos, dados e outras formas de informação, em alta velocidade. 3G surgiu do conhecimento e experiência gerada das gerações precedentes de comunicação móvel, chamadas 2G e 2.5G.

Entretanto, redes 3G utilizam freqüências de transmissão diferentes das anteriores e necessitam de uma nova infra-estrutura. Por causa disto as operadoras e portadoras estão investindo na construção e estruturação de suas redes de terceira geração. Com a adição destas redes, as taxas de transmissão de dados deverá subir para 144 Kbps em ambientes de alta movimentação, 384 Kbps em ambientes de baixa movimentação e até 2 Mbps em ambientes estacionários.

Os serviços 3G são advindos da convergência lógica de duas das maiores tecnologias atuais, a Internet e a telefonia móvel. Alguns dos serviços atualmente disponíveis para usuários das redes de terceira geração são:

- Voz sobre IP (VoIP);
- O envio/recebimento de imagens em tempo real;
- Determinação da posição geográfica (GPS - *Global Positioning System*);
- *Streaming* de áudio e vídeo.

Os diversos serviços que a terceira geração tornou disponível não resultaram apenas numa revolução corporativa, mas também numa revolução para os consumidores. Eles exigirão, cada vez mais, serviços e aplicações personalizados, buscando tornar suas vidas mais fáceis e melhorarem o meio como conduzem os seus negócios. Java ME aparece como sendo a melhor solução disponível atualmente para vencer este desafio.

2.2 Java Micro Edition

Em 1991 a Sun Microsystems, diante da idéia de que a próxima área em que os microprocessadores teriam um grande impacto seria a de dispositivos eletrônicos inteligentes

destinados ao consumidor final, financiou uma pesquisa para a criação de uma linguagem capaz de atender os requisitos destes dispositivos.

Esta pesquisa tinha o codinome *Green* e o seu resultado foi o desenvolvimento de uma linguagem, criada por James Gosling, baseada em C e C++ [16] que posteriormente foi denominada de *Oak* (carvalho) em homenagem a uma árvore que dava para a janela de seu escritório na Sun.

Certo tempo depois foi descoberto que havia uma linguagem de computador que utilizava este mesmo nome. Foi quando uma equipe da Sun, que estava numa cafeteria local, sugeriu o nome Java [17], elucidando o nome da cidade de onde um café específico servido era importado.

Entretanto, o mercado naquela época não se desenvolveu tão rápido como era esperado, e como neste mesmo período a *World Wide Web* estava tendo um grande crescimento, a Sun vislumbrou o potencial de utilizar Java para a criação de páginas *web* dinâmicas. Foi desta forma que Java ganhou força e se disseminou, consolidando-se como uma das linguagens mais conhecidas e utilizadas mundialmente.

Reconhecendo que não havia como propor uma plataforma genérica e completa, a Sun agrupou as tecnologias Java em três edições, cada uma visando uma área específica da indústria de computação, são elas:

1. *Java Enterprise Edition* (Java EE) [18], para empresas que necessitam atender seus clientes, empregados e fornecedores com soluções servidoras de grande escala.
2. *Java Standard Edition* (Java SE) [19], para o familiar e bem estabelecido mercado de computadores pessoais.
3. *Java Micro Edition* (Java ME), para as necessidades combinadas de:

    - Consumidores e fabricantes de dispositivos embarcados que constroem uma diversidade de equipamentos de informação,
    - Provedores de serviço que desejam disponibilizar conteúdo aos seus usuários através destes dispositivos.
    - Desenvolvedores que precisem criar conteúdos para *desktops*.

Cada edição da plataforma Java define um conjunto de tecnologias que podem ser usadas com um produto particular:

- Máquinas virtuais Java, que se ajuste dentro de um grande arranjo de dispositivos computacionais,
- Bibliotecas e APIs (*Application Programming Interfaces*) especializadas para cada tipo de dispositivo e,
- Ferramentas para a depuração e configuração do dispositivo.

A Figura 1 ilustra as edições da plataforma Java e seus mercados alvo, iniciando das plataformas de alto nível à esquerda e movendo-se através das plataformas de baixo nível na direita. Basicamente cinco mercados alvo ou gama de categorias de dispositivos são identificados. Servidores e computadores empresariais são suportados pela Java EE, os *desktops* e computadores pessoais pela Java SE. A plataforma Java ME é de um modo geral dividida em duas categorias que focam em dispositivos consumidores de “alto nível” e de “baixo nível”. Finalmente, o padrão Java Card [20] tem o seu foco principal no mercado de cartões inteligentes.

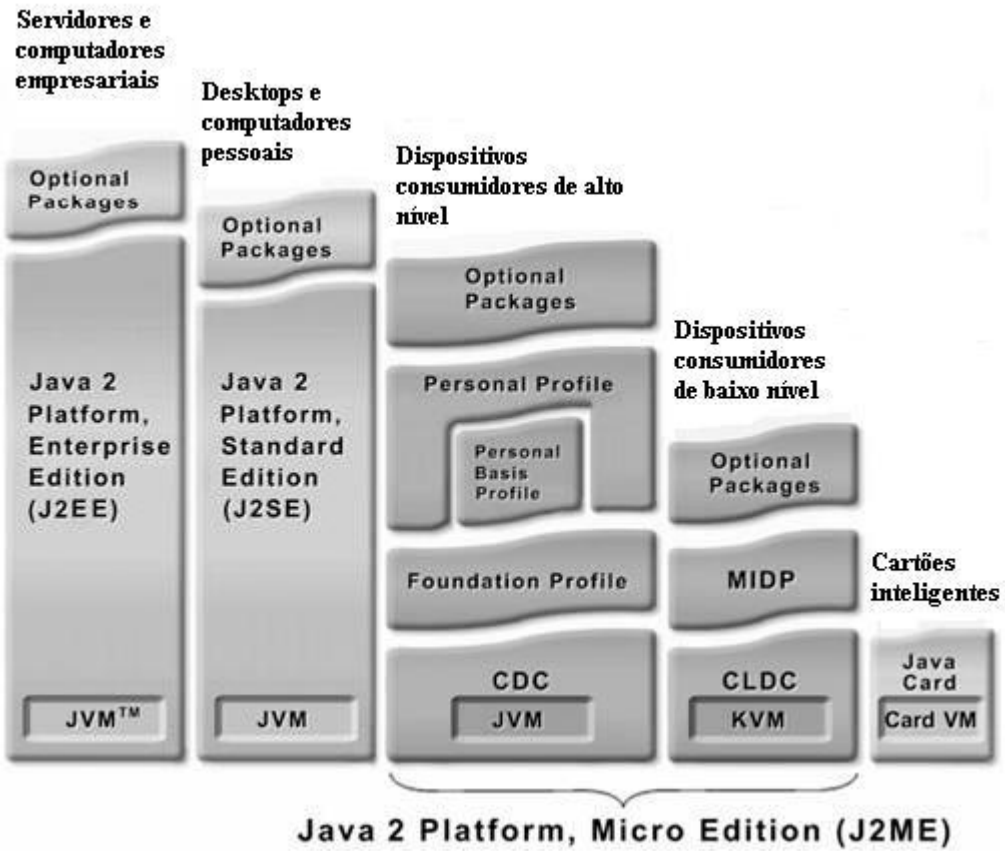

Figura 1: Plataformas Java e seus respectivos mercados consumidores [9]

2.2.1 Plataforma Java Micro Edition

A criação da plataforma Java ME nasceu da necessidade de se definir uma plataforma computacional que pudesse atender equipamentos eletrônicos específicos e/ou dispositivos embarcados.

Os criadores da plataforma Java ME dividiram tais dispositivos em duas categorias básicas, diferenciadas por níveis ou capacidades:

- Dispositivos de alto nível. Na Figura 1, esta categoria é representada pelo grupo denominado CDC (*Connected Device Configuration*) [21]. Exemplos típicos de dispositivos desta categoria incluem alguns PDAs, comunicadores sem fio, Internet TVs, sistemas de navegação automotivo, etc. Estes dispositivos tem várias formas de interagir com o usuário, possuem grandes quantidades de memória, normalmente iniciando entre 2 e 4 Megabytes e persistência, conexões com alta largura de banda, frequentemente usando TCP/IP (*Transport Control Protocol / Internet Protocol*) [22].
- Dispositivos de baixo nível. Na Figura 1, esta categoria é representada pelo grupo intitulado CLDC (*Connected Limited Device Configuration*). Telefones celulares, pagers e organizadores pessoais são exemplos dos dispositivos desta categoria. Estes dispositivos têm interface simples com os usuários, quantidade limitada de memória, normalmente 128-256 Kbytes, e conexões intermitentes e de baixa largura de banda. Nesta categoria as conexões de rede geralmente não são baseadas no TCP/IP e geralmente são alimentados por baterias.

A linha entre estas duas categorias vem se tornando muito confusa a cada dia. Como resultado do andamento e desenvolvimento tecnológico os dispositivos futuros farão uso de conexões sem fio preferencialmente às redes fixas que utilizam cabos. Na prática, a linhas entre estas duas categorias é definida mais pela disponibilidade de memória, poder de bateria, considerações de largura de banda e tamanho físico da tela do dispositivo que por suas funcionalidades ou tipo de conectividade.

A arquitetura Java ME foi projetada para ser flexível, modular e escalar a fim de se adaptar com a demanda do mercado. Deve abranger a crescente quantidade de tipos, configurações de hardware, aplicações e propriedades dos dispositivos existentes. Essa escalabilidade e modularidade são definidos pela tecnologia Java ME por um modelo de software de algumas camadas construída sobre o "Sistema Operacional Local" do aparelho. A Figura 2 ilustra estas camadas.

Figura 2: Camadas da plataforma Java ME [17]

Camada de Configuração: esta camada é pouco visível aos usuários, porém é muito importante para desenvolvedores de Perfis. A Configuração, em Java ME, define uma plataforma Java mínima para uma família de dispositivos. Todos os membros de uma dada família têm exigências similares de memória e poder de processamento. Uma Configuração é realmente uma especificação que identifica os recursos disponíveis no nível de sistema, como as potencialidades da linguagem Java, as características e atributos da máquina virtual presente, e o mínimo de bibliotecas Java que são suportadas.

Uma Configuração também especifica um conjunto mínimo de atributos para cada categoria de dispositivos. Fabricantes de aparelhos implementam Perfís para prover uma plataforma real para uma família de dispositivos que tem as potencialidades que uma dada Configuração especifíca.

Camada de Perfil: esta camada especifica o nível da interface de aplicação para uma classe de dispositivos em particular. A implementação de um Perfil consiste de um conjunto de bibliotecas de classe Java que provêem este nível de interface de aplicação. Assim um Perfil teoricamente pode especificar todos os tipos de funcionalidades e serviços.

Um Perfil é implementado em cima de uma Configuração, um passo mais perto da implementação de aplicações reais. Tipicamente um Perfil inclui bibliotecas que são mais específicas as características de uma categoria de dispositivos que as bibliotecas que compreendem as Configurações, ou seja, bibliotecas mais específicas que as disponibilizadas pelas Configurações.

Aplicações são então construídas sobre Configurações e Perfis e elas podem apenas usar bibliotecas de classes providas por estas duas especificações de baixo nível. Perfis podem ser montados em cima de outro Perfil. Uma implementação de uma plataforma de Java ME, no entanto, pode conter apenas uma Configuração. A Figura 3 demonstra a construção de uma plataforma de Java ME, onde pode haver vários Perfis para uma mesma Configuração.

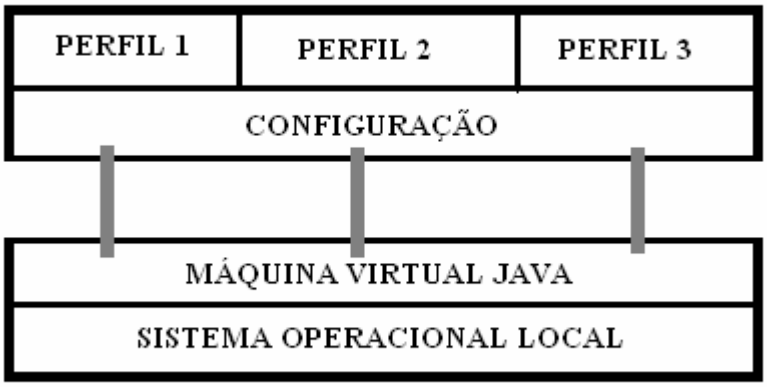

Figura 3: Plataforma Java ME

Uma outra camada que pode ser mencionada é a de *Pacotes Opcionais*. Algumas APIs são aplicáveis a um grande número de dispositivos e família de dispositivos. Um Pacote Opcional de Java ME tipicamente contém funcionalidades que são independentes de qualquer família de dispositivos. O principal objetivo de um pacote opcional deve carregar as definições de uma API que possa ser adicionada flexivelmente no topo de inúmeros Perfis diferentes. Um dispositivo pode suportar inúmeros Pacotes Opcionais.

Configurações, Perfis e Pacotes Opcionais usam as potencialidades da máquina virtual Java (JVM – *Java Virtual Machine*), que é considerada como parte da Configuração. A JVM geralmente executa em cima do Sistema Operacional Local que faz parte do aparelho.

2.2.2 O Processo da Comunidade Java (JCP)

O Processo da Comunidade Java ou *Java Community Process* (JCP) [9, 23] é um fórum criado pela Sun e por vários parceiros da indústria de tecnologia, como forma de evoluir e manter a tecnologia aberta e disponível para todos, permitindo que haja competição e inovação dentro do mundo Java, mas de forma a sempre garantir a compatibilidade e independência de plataforma. Basicamente ela é designada para garantir que a tecnologia Java está sendo desenvolvida de acordo com o consenso da comunidade. Este processo é descrito em [24].

Configurações e Perfis surgiram primeiramente na forma de Requisições de especificação Java (JSRs - *Java Specification Requests*) [23]. Uma JSR nada mais é que um documento, que diz basicamente o que um produto deve fazer, mas não diz como ele deve fazer. É simplesmente uma

especificação que detalha toda a interface, e o que deve acontecer do ponto de vista do desenvolvedor que utiliza a especificação. A partir dessas informações, qualquer fornecedor pode definir a sua implementação específica.

Todas as tecnologias, desde servidores, como Java EE até os pequenos aparelhos que usam Java ME, passando pelo mundo dos *desktops*, todos têm em comum o fato de serem mantidos, modificados e evoluídos através do *Java Community Process*.

Para se ter uma idéia do que está acontecendo no mundo de Java ME, a Tabela 1 mostra algumas das Configurações, Perfis e Pacotes Opcionais que estão disponíveis ou que estão sendo desenvolvidas.

Tabela 1: Algumas JSRs disponíveis para Java ME.

Configurações		
JSR	Nome	URL
30	Connected, Limited Device Configuration (CLDC) 1.0	http://jcp.org/jsr/detail/30.jsp
139	Connected, Limited Device Configuration (CLDC) 1.1	http://jcp.org/jsr/detail/139.jsp
36	Connected Device Configuration (CDC)	http://jcp.org/jsr/detail/36.jsp
Perfis		
JSR	Nome	URL
37	Mobile Information Device Profile 1.0	http://jcp.org/jsr/detail/37.jsp
118	Mobile Information Device Profile 2.0	http://jcp.org/jsr/detail/118.jsp
75	PDA Profile 1.0	http://jcp.org/jsr/detail/75.jsp
46	Foundation Profile	http://jcp.org/jsr/detail/46.jsp
129	Personal Basis Profile	http://jcp.org/jsr/detail/129.jsp
62	Personal Profile	http://jcp.org/jsr/detail/62.jsp
Pacotes Opcionais		
JSR	Nome	URL
66	RMI Optional Package	http://jcp.org/jsr/detail/66.jsp
82	Java APIs for Bluetooth	http://jcp.org/jsr/detail/82.jsp
120	Wireless Messaging API	http://jcp.org/jsr/detail/120.jsp
135	Mobile Media API	http://jcp.org/jsr/detail/135.jsp
179	Location API for Java ME	http://jcp.org/jsr/detail/179.jsp

Esta não é uma lista completa das JSRs disponíveis, para uma lista mais detalhada e completa consultar o site oficial da JCP [23].

2.2.3 Configurações e Perfis

As Configurações e os Perfis fornecem uma separação de interesses na arquitetura de Java ME entre a necessidade de portabilidade e a necessidade do suporte de uma grande quantidade de dispositivos e potencialidades. Configurações servem para aumentar a portabilidade para muitos dispositivos diferentes enquanto que os Perfis estão voltados para um dispositivo específico ou um grupo semelhante deles.

Por exemplo, Configurações incluem atributos do núcleo de Java como `String`, `Thread`, `System` e `Object`, assim como os meios para tratar dos fluxos de entrada e saída e conectividades de rede. Perfis, por outro lado, cuidam das características dos dispositivos como interfaces gráficas, manipulação de eventos e armazenamento de dados. Perfis também empacotam certos conjuntos de funcionalidades como potencialidades de multimídia e configuração de jogos.

Uma importante característica das Configurações é que elas compartilham um relacionamento aninhado. Isto significa que as Configurações podem ser pequenas ou grandes, mas todas elas devem pertencer a maior Configuração de Java ME. Esta relação deve ser sempre uma relação de conjunto e subconjunto. Sempre que há uma movimentação de uma Configuração mais específica para uma mais geral há também um ganho de portabilidade.

Este conceito seria bem explicitado ao mover-se da Configuração 1 para a Configuração 2 na Figura 4**.** Observa-se que os atributos presentes na primeira também estão contidas na segunda, assim pode-se substituir a Configuração 1 pela Configuração 2 com o intuito de aumentar a portabilidade.

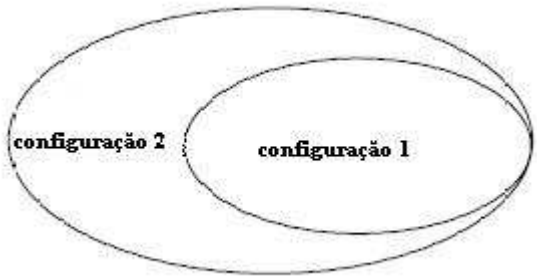

Figura 4: Organização de Configurações.

A portabilidade é maximizada quando se é movido do ambiente mais restrito para o mais completo. Por exemplo, uma aplicação desenvolvida para a Configuração 1 do exemplo anterior, pode ser corretamente portada para a Configuração 2.

Fabricantes de dispositivos devem aderir às especificações das configurações quando estiverem implementando ou portando máquinas virtuais ou Configurações para suas plataformas que compilem Java ME. Este ato permite a portabilidade entre dispositivos de fabricantes diferentes assim com diferentes tipos de aparelhos.

Configurações

Configurações são especificações da arquitetura de Java ME que são definidas por um grupo de especialista usando o JCP. Especificações de Configurações são criadas em cooperação com muitas indústrias participantes.

Atualmente, Java ME define duas Configurações:

- *Connected, Limited Device Configuration* (CLDC)
- *Connected Device Configuration* (CDC)

O CLDC é destinado às necessidades de dispositivos com limitações estritas tanto de memória, como de poder de processamento, consumo de energia e conectividade de rede. O CDC é voltado às necessidades de dispositivos mais poderosos. A Figura 5 ilustra a relação entre CLDC e CDC.

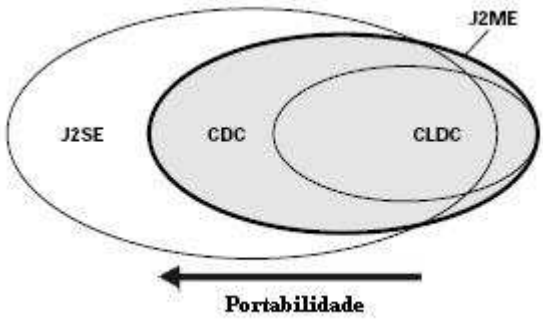

Figura 5: Relação entre CDC e CLDC

Configurações definem o contrato ou a forma de comunicação entre o Perfil e a máquina virtual Java. Lembrando que Perfis provêem APIs de uso específico para dispositivos específicos. Como foi mencionado, ambos, CLDC e CDC tem suas próprias máquinas virtuais. A Configuração CDC usa a *C-Virtual Machine* (CVM), enquanto a CLDC usa a chamada KVM (*Kilobyte Virtual Machine*). A implementação de uma máquina virtual de Java ME deve obedecer as especificações definidas pela Configuração. No caso da KVM, certas funcionalidades são removidas com a intenção de acomodar e obedecer às restrições de memória de CLDC. A Tabela 2 demonstra a relação entre Configurações e máquinas virtuais e mostra exemplos de dispositivos candidatos.

Tabela 2: Configurações, máquinas virtuais e exemplos de dispositivos.

Configuração	Máquina Virtual	Exemplos
CDC	CVM	Computadores pessoais de bolso, TVs via internet Comunicadores, Codificadores de TVs digitais
CLDC	KVM	Telefones celulares, PDAs, pagers

Connected Limited Device Configuration

A CLDC costitui o bloco básico de construção no qual Perfis Java ME para pequenos dispositivos, como telefones celulares, pagers e PDAs simples, são construídos. Estes dispositivos são caracterizados por recursos limitados de memória e capacidade de processamento, o que torna impossível à eles hospedarem uma versão completa da plataforma Java.

CLDC especifica um conjunto mínimo de pacotes Java e classes e uma máquina virtual com funcionalidades reduzidas que pode ser implementada com as restrições de recursos imposta por estes aparelhos. A especificação completa da CLDC pode ser encontrada em [25].

Um dispositivo de recursos limitados nesta especificação segue as seguintes características:

- 128 a 512 Kbytes de memória disponível para o ambiente Java;
- Um processador de 16 ou 32 bits;
- Baixo consumo de energia, pois a maioria é alimentada por bateria;

- Suporta algum tipo de conexão com a rede. Muito provavelmente por uma conexão intermitente, de largura de banda baixa de aproximadamente 9600 bps (bits por segundo) e ,freqüentemente, sem fio.

O CLDC é baseada na Java SE, mas omite algumas funcionalidades. CLDC foi criada a partir do zero e foi adicionado apenas o que fosse necessário, seguindo os seguintes critérios:

- Esta funcionalidade é apropriada para estes tipos de dispositivos?
- Os desenvolvedores podem facilmente recriar a funcionalidade se necessário? Isto se aplica ao se alternar assinaturas de métodos assim como classes inteiras.
- Há riscos de segurança relativos à funcionalidade nestes dispositivos?
- Estes dispositivos geralmente suportam esta funcionalidade?
- A funcionalidade necessitará de uma grande quantidade de espaço de código ou consumirá muitos recursos de memória ou ciclos de CPU?

Para atender a estes critérios, a CLDC removeu muitas características que estão disponíveis no ambiente Java SE. A lista a seguir mostra algumas das características que foram removidas ou modificadas para o ambiente CLDC (especificamente para a versão 1.0 desta):

Interface Nativa de Java (JNI - *Java Native Interface*) [26]: CLDC não suporta as características de JNI de Java SE, o qual permite que um código nativo seja invocado por classes de Java. JNI é omitido parcialmente porque requer muita memória para ser implementado e por outra parte para proteger dispositivos CLDC contra problemas de segurança que podem ser causado por códigos de aplicação maliciosos.

Invocadores de classe definidas pelo usuário: A especificação da CLDC requer implementações para prover seu próprio mecanismo invocador de classes que não pode ser sobrescrito ou estendido pelo código da aplicação.

Reflexão: O pacote `java.lang.reflect` e todas as configurações de `java.lang.Class` que estão conectadas com reflexão não estão disponíveis. Esta restrição é aplicada em parte para economizar memória.

Finalização de objetos: Finalização de objetos causa uma grande complexidade na máquina virtual trazendo muito pouco beneficio. Conseqüentemente, finalização não é implementada, e o pacote `java.lang.Object` da CLDC não contém um método `finalize`.

Referências Fracas: Referência fraca e o pacote `java.lang.ref` não são suportados por causa da memória necessária para a implementação deles.

Tipo ponto flutuante de dados: Muitos dos processadores utilizados nas plataformas alvo da CLDC não apresentam o hardware de ponto flutuante, a máquina virtual não é requerida para suportar operações de ponto flutuante.

Limitações nos tratamentos de exceções: Java SE tem um grande numero de classes que representam condições de erro e exceções. Como aplicações Java não são, em geral, esperadas de recuperação de erros (significando exceções lançadas derivadas da classe `java.lang.Error`), a maioria das classes que as representam não estão incluídas na CLDC. Quando tal erro ocorrer, o

dispositivo é responsável por tomar uma ação apropriada assim como reportá-la ao código da aplicação.

Configurações de Threads: CLDC provê a utilização de threads, porém não habilita a criação de threads monitoras, nem grupos de threads.

Para a versão 1.1 da CLDC, algumas destas características foram novamente introduzidas, pois eram de grande importância e utilidade durante o processo de desenvolvimento e de depuração de uma aplicação. Características como a notação de ponto flutuante e suporte a referencias fracas foram adicionadas.

Kilobyte Virtual Machine (KVM)

A KVM [27] adere à especificação da máquina virtual Java tanto quanto é possível. Entretanto as potencialidades da KVM são definidas em grande parte pelas especificações da CLDC. A KVM difere da especificação da máquina virtual Java apenas quando a CLDC requer ou permita que isto aconteça por razões de otimização ou suporte a APIs.

Por exemplo, frequentemente float e double não são suportados por dispositivos com CLDC. Como resultado, os criadores da CLDC decidiram que estes tipos de dados eram muito custosos para serem implementados em dispositivos que, em grande parte, não os suporta. Como resultado, os tipos de dados `float` e `double` não são suportados pela CLDC e não são reconhecidos pela KVM.

A KVM requer uma pequena quantidade de recursos do dispositivo, entre 40Kb e 80Kb dependendo das opções de compilação e da plataforma alvo. Isto habilita a KVM a funcionar em aparelhos com no mínimo 128 Kbytes de memória total. KVM foi desenvolvida desde o inicio em C e é designada para ser tão completa e rápida quanto for possível, rodando normalmente a 30% a 80% da velocidade da JVM padrão.

É importante salientar que os fabricantes não precisam utilizar a KVM para a CLDC, eles têm a opção de portar a KVM para seus dispositivos, ou simplesmente construir suas próprias máquinas virtuais que suportem a especificação da CLDC.

Connected Device Configuration

A CDC é a segunda das duas Configurações atualmente definidas para Java ME, e está voltada para dispositivos e ferramentas de rede com mais recursos que dispositivos CLDC. A CDC executa sobre a *C-Virtual Machine* (CVM) [28] que é totalmente complacente com a especificação da máquina virtual Java. O CDC é destinado aos dispositivos com no mínimo 512 Kbytes de memória, entretanto é designado para plataformas com aproximadamente 2 Megabytes de memória disponível. Tipicamente os dispositivos desta categoria apresentam poder de processamento considerável e suportam conexões com grande largura de banda e conteúdo *web* de alta fidelidade.

C-Virtual Machine

Embora a CVM esteja de acordo com as especificações da máquina virtual Java, sua implementação difere das máquinas virtuais de Java SE, pois são otimizadas para dispositivos e ferramentas de rede. Os algoritmos de coleta de lixo (*garbage collection*) são totalmente separados da máquina virtual possibilitando a adição de outros algoritmos de coleta de lixo à CVM.

A implementação de referência emprega um coletor de lixo generalizado, que usa períodos curtos de coleta de lixo que não sobrecarreguem a máquina virtual por longos períodos de tempo. A coleta de lixo é executada freqüentemente por curtos períodos de tempo. O coletor é mais exato, tendo conhecimento sobre todos os ponteiros no tempo da coleta de lixo fazendo com que não seja necessário consumir ciclos extras de processamento com varreduras na pilha de instruções.

Para acrescentar portabilidade entre plataformas, a implementação de referência define concorrência (multithreading) completamente dentro da máquina virtual Java. O uso destas threads permite uma maior portabilidade da máquina virtual desde que não haja nenhuma dependência com a concorrência do sistema operacional. Entretanto, a opção de empregar threads nativas é possível se um fabricante decidir implementá-la em sua plataforma específica.

Perfis

Perfis provêem APIs que focam em um único dispositivo, tal como um PDA, ou em grupo de dispositivos relacionados como telefones celulares e pagers. Os dispositivos suportados por um Perfil normalmente são muito parecidos em termos de como esses aparelhos são usados, quais as potencialidades de interface como usuário, como ou se ele se conecta a uma rede, como ele armazena dados e assim por diante. Perfis se dirigem às necessidades particulares de um segmento da indústria ou de mercado. Os Perfis equiparam-se ao comportamento mais específico disponível na arquitetura de Java ME.

Perfis são criados, por muitos participantes, através do Processo da Comunidade Java (JCP) da Sun. Embora um grupo de especialistas da JCP forneçam frequentemente implementações referenciais das especificações, os fabricantes de dispositivos é que decidem se optam por portar estas implementações referenciais fornecidas pela Sun ou por criar suas próprias implementações que se integre com a especificação.

Assim como uma Configuração faz a intermediação entre a máquina virtual e o Perfil, um Perfil é o responsável por fazer a comunicação entre o dispositivo e a aplicação real. Para que o dispositivo de um fabricante suporte um Perfil é necessário que todas as APIs e características especificadas no Perfil sejam suportadas.

Tipicamente, Perfis fornecem interface com usuário, entrada de métodos e mecanismos de persistência para um dado grupo de dispositivos. Estes tipos de Perfis são elaborados para definir um ambiente completo de desenvolvimento para um conjunto específico de dispositivos e, conseqüentemente, podem ser considerados como “Perfis de dispositivo”.

Por outro lado, Perfis podem ser criados para cumprir funcionalidades e serviços específicos. Exemplos destes tipos de Perfis são os seguintes: o Perfil de Invocação de Métodos Remotos (RMI) [29] ou o Perfil de Multimídia (*Multimedia Profile*) [30]. Este tipo de Perfil pode encapsular serviços para um determinado segmento de mercado, como, por exemplo, transações bancárias.

A vantagem de se encapsular serviços e funcionalidades específicas na forma de Perfis é permitir o reuso destas características mais facilmente por outros dispositivos. Isso prove modularidade e flexibilidade permitindo aos fabricantes de dispositivos escolher quais características são necessárias e quais são as mais importantes.

Um aparelho pode suportar mais de um Perfil dependendo das necessidades e potencialidades do dispositivo e quais Perfis o fabricante escolhe para suportar. Perfis adicionam modularidade a arquitetura Java ME, direcionando necessidades específicas e funcionalidade. Por exemplo, considerando três dispositivos: um PDA, um telefone celular e uma televisão digital interativa. Assumindo que todos possam realizar compras com cartão de crédito na Internet. Muito provavelmente as APIs específicas de dispositivo teriam que ser direcionadas para três

diferentes Perfis devido às necessidades de interface de usuário entre outras configurações específicas de dispositivo. Entretanto todos três aparelhos poderiam suportar o mesmo Perfil de transação segura de compra com cartão de crédito.

Devido às naturezas específicas dos Perfis e de sua modularidade, podemos esperar, num futuro, a criação de diversos Perfis para necessidades específicas. Porém é importante salientar que um simples Perfil, ou até mesmo um conjunto deles, não fazem um ambiente completo de Java ME, eles apenas adicionam atributos específicos.

Escolhendo um Perfil

Escolher o Perfil correto, ou o conjunto de Perfis, é uma das mais importantes decisões a ser tomada quando se está criando aplicações usando Java ME. Isto porque o Perfil é a parte da arquitetura de Java ME que está mais próxima dos dispositivos em si. A partir do momento em que se conhece quais dispositivos irão receber a aplicação, pode-se definir qual Perfil mais adequado para ser utilizado.

Há um número crescente de Perfis sendo desenvolvidos. O MIDP é um dos melhores conhecidos, pois foi o primeiro Perfil de Java ME lançado pelo JCP. Em adição aos Perfis oficiais de Java ME, um merece certa atenção. A API conhecida como KJava [31] e foi desenvolvida pela Sun Microsystems para testar e demonstrar a CLDC.

A Tabela 3 mostra uma lista de Perfis existentes e alguns que ainda estão em desenvolvimento.

Tabela 3: Perfis definidos para Java ME

Perfil	Configuração	Máquina Virtual	Exemplos de Dispositivos
MIDP	CLDC	KVM	Telefones celulares e pagers
RMI	CDC	CVM	Qualquer
Foundation	CDC	CVM	Roteadores e Impressoras de rede
PDAP	CLDC	KVM	PDAs
Multimedia	CLDC/CDC	KVM/CVM	Qualquer
Personal	CDC	CVM	Computadores de bolso
Personal Basis	CDC	CVM	Qualquer
Gaming	CLDC/CDC	KVM/CVM	Qualquer
WTCA	CLDC/CDC	KVM/CVM	Telefones celulares

Perfil de Dispositivo de Informação Móvel (MIDP)

MIDP é uma versão da plataforma Java baseada na CLDC e KVM que é apontado para pequenos dispositivos, principalmente telefones celulares e pagers. É também apropriada para executar em PDAs, e uma implementação é disponível para PalmOS V.3.5 ou maior. Entretanto, normalmente estes dispositivos usam o PDAP (*PDA Profile*) [10, 31] que também é hospedado pela CLDC. A especificação do MIDP foi desenvolvida sob o Processo da Comunidade Java (JCP) e está disponível em [23].

A posição lógica do MIDP na arquitetura de software de um dispositivo que o implementa é mostrado na Figura 6**.** O software que implementa o MIDP executa na KVM fornecido pela CLDC e provê serviços adicionais em benefício do código da aplicação que APIs do MIDP.

Aplicações do MIDP são chamadas de MIDlets [9, 10, 11]. Como a Figura 6 mostra, MIDlets podem usar diretamente tanto as facilidades do MIDP quanto as facilidades providas por outras APIs de Java ME.

A Sun provê uma referência de implementação do MIDP que pode ser usada pelo Windows. O Wireless Toolkit [25, 32], que contém versões do MIDP para Windows, Solaris e Linux, e um produto separado do MIDP para uso em PDAs baseados em PalmOS.

Fabricantes de dispositivos normalmente usam a referência de implementação da Sun como base para seus próprios produtos. Eles geralmente possuem códigos adicionais como parte da sua implementação do MIDP para prover características de gerencia tais como instalação, remoção e gerenciamento de MIDlets que não são portáveis entre dispositivos.

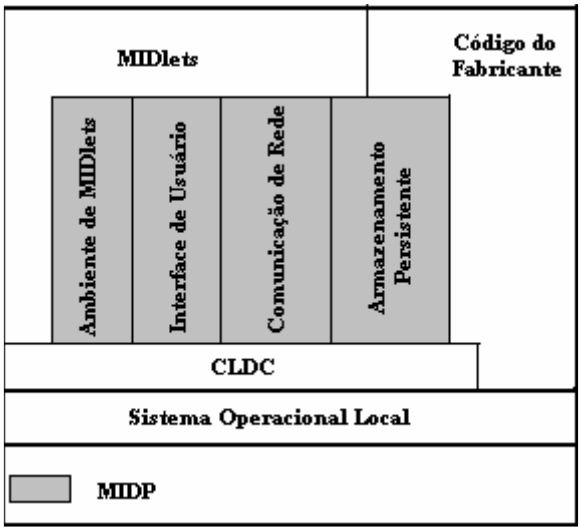

Figura 6: Plataforma do MIDP

Requisitos de hardware do MIDP

Como mencionado anteriormente, MIDP é direcionado para pequenos dispositivos com memória, processamento e potencialidades de tela limitados. Os requisitos mínimos de hardware estão descritos a seguir.

Tela

Dispositivos MIDP são caracterizados por pequenas telas. A especificação indica que a tela deve ter até 96 *pixels* de largura e 54 *pixels* de altura, e que cada *pixel* seja aproximadamente quadrado. A tela deve suportar no mínimo duas cores, e a maioria dos telefones não suportam mais que isso. No topo da escala, PDAs tipicamente tem 160 *pixels* em ambas as direções e suportam 65,536 cores diferentes. Esta grande disparidade de potencialidades acarreta ao desenvolvedor, que deseja escrever uma MIDlet inteiramente portável, grandes desafios.

Conectividade

Os dispositivos móveis de informação têm algum tipo de acesso de rede, seja uma conexão sem fio (wireless) em telefones e pagers, ou um modem ligado a um PDA. MIDP não assume que os dispositivos estejam permanentemente conectados a uma rede ou que a rede suporte diretamente TCP/IP. Isto, entretanto, requer que o fabricante, ou a operadora de serviços, do dispositivo disponibilize a impressão de que o aparelho suporte HTTP 1.1, seja diretamente

através de uma pilha de protocolos de Internet, como no caso de um Palm portátil conectado por um modem, ou montando uma conexão sem fio com a Internet via um gateway Wap (*Wireless Application Protocol*) [33].

Esta característica habilita os desenvolvedores a escrever MIDlets com funcionalidades de rede que trabalhem igualmente bem, mas com largura de banda menor, em todas as plataformas suportadas.

Memória

MIDP inclui muitas funcionalidades que não fazem parte do núcleo da plataforma Java e, conseqüentemente, requer mais memória que os ambientes CLDC mínimos são obrigados a fornecer. A especificação MIDP requer ao menos 128 Kbytes de RAM disponível para armazenar a implementação do próprio MIDP, mais que a necessária pela CLDC. Em adição a isto, deve haver ao menos 32 Kbytes disponíveis para recursos importantes (pilha) de Java.

Na prática, 32 Kbytes de pilha é muito limitante e obriga que o desenvolvedor tenha muito cuidado na alocação de objetos e evite manter referência a objetos mais tempo que necessário, pois permite que o coletor de lixo (*garbage collector*) libere espaço o mais rápido possível.

Assim como requisições de RAM (*Random Access Memory*), dispositivos com MIDP devem também dispor de ao menos 8 Kbytes de memória não volátil para ser usado na persistência de dados no qual as MIDlets podem salvar informações de maneira que não sejam perdidas quando o aparelho for desligado.

Dispositivos de entrada

Assim como a tela (*display*), existem muitas formas diferentes de dispositivos de entrada que podem ser encontradas na plataforma MIDP. O mínimo requisito feito pela especificação MIDP é que o dispositivo tenha o equivalente a um teclado que permita ao usuário teclar de 0 até 9, juntamente com um conjunto de setas e um botão de seleção como mostrado na parte superior da Figura 7, onde o botão de seleção encontra-se no meio das setas.

Estas exigências são normalmente encontradas em telefones celulares e podem ser satisfeitas de diversas maneiras em outros dispositivos. Num Palm, por exemplo, existem botões que podem ser programados para agir como setas direcionais, enquanto a operação de seleção pode ser executada com o toque de uma caneta especial (graffiti) na tela.

Figura 7: Teclado típico de um celular

Existem vários outros tipos de teclados mais sofisticados suportados pela plataforma MIDP. Um exemplo a ser considerado é o teclado de um dispositivo chamado RIM, da fabricante Blackberry [34], que contém um teclado alfanumérico completo, como mostrado na Figura 8. Este tipo de teclado facilita a digitação de caracteres numéricos e alfabéticos.

Figura 8: Outro exemplo de teclado

Requisitos de software do MIDP

A versão de referência da Sun do MIDP não é um produto comercial. Fabricantes de dispositivos devem portar a referência de implementação para seus próprios hardware e software implementando códigos que façam a ligação entre as expectativas da referência da Sun e o hardware e sistema operacional utilizado pelo fabricante.

Assim como requisitos de hardware, mostrado anteriormente, a implementação de referência faz as seguintes suposições sobre as potencialidades oferecidas pela base de software no qual será hospedada (visto como "Sistema Operacional Local" na Figura 6):

- O sistema operacional deve prover um ambiente seguro de execução, no qual a JVM possa rodar. Muitos destes problemas de execução são observados quando do uso de *multithreads*. Neste caso, se o sistema operacional não suportar *multithreading*, o MIDP deve criar a ilusão de que este recurso está disponível.
- Suporte a conexões de rede é necessária de qualquer forma. Em algumas plataformas, como o PalmOS, a *socket-level API* está disponível no qual o suporte do MIDP ao HTTP pode ser viabilizado. No caso de dispositivos que não oferecem este tipo de interface, incluindo aquelas que possuem acesso a uma conexão de rede baseada em IP [35], o fabricante deve fornecer um meio de conectar este dispositivo com a Internet.
- O software deve fornecer acesso ao sistema do teclado e dos comandos direcionais do dispositivo. O software deve estar apto a manipular eventos quando teclas são pressionadas e soltas e quando as setas direcionais são selecionadas.
- Deve ser possível acessar a tela do dispositivo. MIDP habilita as MIDlets a tratar a tela do dispositivo como um *array* retangular de *pixels*, no qual cada um destes *pixels* pode conter uma das cores suportadas por este dispositivo.

- A plataforma deve prover algum tipo de armazenamento persistente que não perca seu estado quando o aparelho for desligado. O software deve fornecer algum tipo de interface programática para este mecanismo de armazenamento persistente.

Outros Perfis

Foundation Profile

Usado com a Configuração CDC e é o núcleo para quase todos os outros Perfis que são usados com a CDC, pois o *Foundation Profile* contem classes do núcleo de Java.

Game Profile

É normalmente usado com a Configuração CDC e contem as classes necessárias para o desenvolvimento de jogos para quaisquer pequenos dispositivos computacionais que usem a CDC.

PDA Profile

É utilizado com a CLDC e contem classes que utilizam recursos sofisticados encontrados em PDAs. Estes equipamentos normalmente apresentam telas melhores e mais memória do que normalmente é encontrado em dispositivos que utilizam o MIDP.

RMI Profile

Usado com a CDC e com o Foundation Profile para prover Invocação Remota de Métodos.

Alguns outros Perfis estão disponíveis atualmente, tais como: *Personal Profile*, *Personal Basis Profile* e WTCA (*Wireless Telephony Communication APIs*) [21], entre muitos outros desenvolvidos para suprir algumas necessidades existentes para dispositivos com funcionalidades restritas.

2.2.4 MIDlets e MIDlets Suítes

Aplicações Java que executam no MIDP são conhecidas como MIDlets. Uma MIDlet consiste de pelo menos uma classe que deve ser derivada da classe abstrata definida do MIDP `javax.microedition.midlet.MIDlet`.

MIDlets rodam em um ambiente de execução dentro da JVM que provê um ciclo de vida bem definido, controlado via métodos da classe `MIDlet`, ao qual cada MIDlet deve implementar. Uma MIDlet pode também implementar métodos na classe MIDlet para obter serviços deste ambiente, e pode apenas usar as APIs definidas na especificação do MIDP ou se for portável pelo dispositivo.

Um grupo de MIDlets relacionadas podem ser reunidas numa *MIDlet Suite*. Todas as MIDlets de uma *Suite* são empacotadas e instaladas em um dispositivo como uma simples entidade, e podem ser desinstaladas e removidas apenas em grupo.

Empacotamento da MIDlet

MIDlets precisam ser devidamente empacotadas antes que possam ser transferidas e instaladas em um dispositivo. A subclasse de *MIDlet,* que atua como o principal ponto de entrada para a MIDlet, juntamente com quaisquer outras classes necessárias e quaisquer imagens ou outros arquivos necessários em tempo de execução, devem ser colocados em um único arquivo JAR.

Informações de empacotamento que dizem ao dispositivo o que há dentro do arquivo JAR devem estar descritos no arquivo chamado "manifesto". Estas mesmas informações contidas no arquivo de manifesto devem estar em um outro arquivo chamado de "*Java application descriptor*", ou simplesmente arquivo JAD, que é guardado separadamente do arquivo JAR.

Um JAR pode conter mais de uma MIDlet, neste caso todas as MIDlets devem estar numa mesma "*MIDlet Suite*.

Tanto o arquivo de manifesto com o arquivo JAD são arquivos de texto e cada linha tem a seguinte forma:

atributo-nome: atributo-valor

O nome e o valor são separados por dois pontos e por um opcional espaço em branco. Todos os atributos relevantes para a instalação de uma MIDlet têm o prefixo "MIDlet-". Uma lista completa destes atributos pode ser observado na Tabela 4. Os valores das colunas JAR e JAD indicam quando um atributo é Necessário (N), Opcional (O) ou Ignorado (I) no arquivo correspondente àquela coluna.

Tabela 4: Atributos de empacotamento das MIDlets

Nome do atributo	JAR	JAD
MIDlet-Name	N	M
MIDlet-Version	N	M
MIDlet-Vendor	N	M
MIDlet-n	N	I
MicroEdition-Profile	N	I
MicroEdition-Configuration	N	I
MIDlet-Descriptor	O	O
MIDlet-Icon	O	O
MIDlet-Info-URL	O	O
MIDlet-Data-Size	O	O
MIDlet-Jar-URL	I	M
MIDlet-Jar-Size	I	M
MIDlet-Install-Notify	I	O
MIDlet-Delete-Confirm	I	O
MIDlet-specific attributes	O	O

O trabalho do arquivo de manifesto é indicar ao dispositivo o nome e a versão da *MIDlet Suite* no JAR e especificar quais dos arquivos .class contidos correspondem as MIDlets individuais. Para que se faça uso destas informações é necessário que o dispositivo carregue o JAR e extraia o manifesto. Feito isto, ele pode então mostrar os valores associados dos atributos MIDlet-Name, MIDlet-Vendor e MIDlet-Version e os atributos opcionais MIDlet-Description e

MIDlet-Icon. Estes atributos ajudam aos usuários a decidir se uma determinada MIDlet deve ou não ser instalada.

Entretanto, o arquivo JAR de uma MIDlet pode ser um tanto grande e pode se tornar um pouco complicado ter que carregar todo o arquivo JAR, principalmente com as conexões lentas atualmente disponíveis, para só depois determinar se deve ou não instalá-la.

Para resolver este problema, alguns dos atributos do arquivo de manifesto, juntamente com algumas informações adicionais, são duplicados no arquivo JAD. Então, em vez de baixar todo o arquivo JAR, o dispositivo MIDP primeiramente carrega este arquivo JAD, que normalmente é muito menor que o arquivo JAR e pode ser transferido mais rapidamente.

O dispositivo então exibe o conteúdo do arquivo JAD ao usuário que pode então decidir sobre a instalação da aplicação sem mesmo necessitar requisitar o arquivo JAR. O arquivo JAD contém alguns atributos vindos do arquivo de manifesto e outros que não aparecem neste. Os atributos mais comuns são os seguintes:

MIDlet-Name: Indica o nome da *MIDlet Suite*
MIDlet-Version: Indica a versão da MIDlet
MIDlet-Vendor: Indica o desenvolvedor da MIDlet
MIDlet-Icon: Especifica o ícone, se houver, da tela inicial da aplicação
MIDlet-Description: Breve descrição da aplicação
MIDlet-info-URL: Indica um endereço para um arquivo de informações
MIDlet-DATA-Size: Indica o tamanho dos dados

Estes atributos, com exceção do último, podem ser todos apresentados ao usuário no intuito de ajudar na decisão de instalar ou não esta aplicação. Os três primeiros destes atributos são obrigatórios, tanto no arquivo JAR como no arquivo JAD, e a especificação do MIDP mostra que os valores em ambos os arquivos sejam idênticos. Os outros atributos são opcionais.

O arquivo JAD também especifica dois atributos que não estão presentes no arquivo de manifesto:

MIDlet-Jar Size: Indica o tamanho real do arquivo JAR
MIDlet-Jar-URL: Indica a URL na qual o arquivo JAR pode ser encontrado

O primeiro dos atributos especifica o tamanho em bytes do arquivo JAR. Esta informação pode ser útil para se determinar se há memória suficiente no dispositivo para suportar a aplicação. Assumindo que o usuário tenha decidido instalar a *MIDlet Suite*, o próximo passo é carregar o arquivo JAR, que pode ser encontrado usando o valor do atributo MIDlet-Jar-URL.

2.3 TELNET

No início dos trabalhos das redes de computadores, um dos mais importantes problemas que cientistas da computação precisaram resolver foi como possibilitar alguém que estivesse operando um computador a acessar e usar outro, como se ele ou ela estivesse conectada localmente a esta outra máquina. O protocolo criado para suprir esta necessidade foi chamado de Telnet, e esforço para desenvolvê-lo foi considerado próximo ao do desenvolvimento da Internet e do protocolo TCP/IP como um todo.

Mesmo que a maioria dos usuários da Internet de hoje não utilizem o protocolo Telnet diretamente, eles usam alguns dos seus princípios subjacentes indiretamente o tempo todo. Toda vez que se enviamos uma parte de um e-mail, usa transferência de arquivos por FTP (*File Transfer Protocol*) [36], ou carrega uma página da *web*, estamos usando tecnologias baseadas no Telnet. Por esta razão, o protocolo Telnet pode ser considerado historicamente, como o mais importante protocolo do TCP/IP.

2.3.1 Visão Geral

Telnet, é que um protocolo de comunicação de dados, na verdade, um conjunto de regras para a comunicação entre computadores. O protocolo Telnet, realmente, é uma abreviação para Protocolo de Redes de Telecomunicações (*Telecommunications Network Protocol*). A comunicação via Telnet funciona entre diferentes tipos de computadores e sistemas operacionais porque o protocolo é o mesmo para todos os sistemas. Telnet é freqüentemente usado para login remoto, onde se acessa um computador a partir de outro. Uma aplicação Telnet típica usaria o protocolo Telnet para mover dados entre um terminal emulado e um servidor Unix.

Muitos computadores suportam o protocolo Telnet. O principal uso do Telnet é a comunicação entre terminais conectados, pode-se usar o Telnet para se conectar a um computador remoto e inserir comandos como se você realmente estivesse neste computador remoto. Pode-se também usar Telnet para acessar bancos de dados remotos, distribuir tarefas entre múltiplos computadores, controlar outros computadores, remotamente executar aplicações e até mesmo configurar roteadores.

Telnet é similar ao TCP [1, 35]. De fato, foi construído em cima dele. Telnet provê características adicionais o que o faz ser considerado como um protocolo de alto nível. Utilizando Telnet, ambas as extremidades de uma conexão pode transmitir dados. Normalmente, um terminal envia comandos para um host, e o host responde com mensagens ou dados. Por exemplo, o usuário de um terminal pode digitar “ajuda” e o host pode responder com “Digite um comando para obter ajuda”.

Telnet provê negociação de opções como um método para estabelecer convenções entre dois computadores. Durante uma sessão, um dos dois computadores pode enviar ao outro um comando, requisitando ou disponibilizando uma opção específica. Uma opção típica é o “eco”. O processo de negociação de determina se um ou outro computador irá suportar “eco”.

2.3.2 Conexões do Telnet

Conexões são feitas em uma porta, que é um método de distinguir múltiplas comunicações em um mesmo computador. A maioria das conexões do Telnet é feita pela porta 23, a porta padrão deste protocolo.

Para realizar uma conexão é preciso especificar uma porta e um nome de host ou endereço para se conectar. Todos os computadores numa rede têm um endereço IP, um número que os distingue de outros computadores da rede. Todos os endereços IP têm o seguinte formato: xxx.xxx.xxx.xxx, no qual cada xxx é um numero decimal de 0 a 255, exceto para o primeiro conjunto que deve ser de 0 a 247.

Terminal Virtual de Rede (NVT)

O Terminal Virtual de Rede (NVT – *Network Virtual Terminal*) [4] é um conceito usado pelo Telnet para representar ambos os computadores em uma conexão. O NVT é dispositivo imaginário baseado em caracteres com um teclado e uma impressora (entrada e saída). Dados que

chegam vão para a impressora, e entradas do teclado vão para a outra extremidade da conexão Telnet. Ambas as extremidades de uma conexão Telnet devem suportar o conceito de um NVT.

Ambas as extremidades da conexão Telnet mapeia suas próprias características de dispositivo terminal para a do NVT. Por exemplo, um programa precisa alterar todos os caracteres ASCII de 8 bits para o conjunto de caracteres do NVT. O NVT deve julgar um contrapeso entre tornar-se muito restritivo (não fornecendo a alguns computadores um vocabulário suficientemente rico para mapearem em seus conjuntos de caracteres) ou tornar-se muito inclusivo (penalizando usuários com terminais mais modestos).

O NVT suporta um conjunto de caracteres de 7 bits conhecidos como NVT ASCII. Esses caracteres são idênticos aos do texto ASCII normal, exceto que eles são representados, cada um, por apenas 7 bits. O oitavo bit, o mais significativo, do byte deve ser colocado para 0. NVT ASCII é utilizado por muitos protocolos da Internet. Um final de linha, por exemplo, é transmitido como o caractere de seqüência CR (*carrier return*) seguido por LF (*line feed*). Se for necessário transmitir o atual CR, ele é transmitido seguido de um caractere NUL (todos os bits são 0).

A Tabela 5 mostra os códigos de controle que precisam ser entendidos pelos Terminais Virtuais de Rede (NVTs).

Tabela 5: Códigos dos tipos de controle do Telnet

Nome	Código	Valor Decimal	Função
NULL	NUL	0	Nenhuma operação.
Line Feed	LF	10	Move a impressora para a próxima linha, mantendo a mesma posição horizontal.
Carriage Return	CR	13	Move a impressora para a margem esquerda da linha em uso.

Os seguintes códigos de controle, exibidos na Tabela 6, são opcionais, mas devem ter o indicado efeito definido na tela.

Tabela 6: Códigos de controle opcionais do Telnet

Nome	Código	Valor Decimal	Função
Bell	BEL	7	Produz um sinal audível ou visível.
Back Space	BS	8	Move a impressora um caractere na direção da margem esquerda.
Horizontal Tab	HT	9	Move a impressora horizontalmente a distancia de uma tabulação.
Vertical Tab	HV	11	Move a impressora verticalmente a distancia de uma tabulação.
Form Feed	FF	12	Move a impressora ao topo da próxima pagina, mantendo a mesma posição horizontal.

O teclado do NVT é especificado como sendo capaz de gerar todos os 128 códigos da tabela ASCII usando teclas, combinações de tecla ou seqüência de teclas.

2.3.3 Comandos

O protocolo Telnet também especifica vários comandos que especificam o método e os vários detalhes de uma interação entre cliente e servidor. Esses comandos são incorporados juntamente com uma rajada de dados comuns. Eles são representados usando bytes especiais com valor variando entre 240 a 254. Para saber a diferença entre bytes de dados destes valores e comandos

Telnet, cada comando é precedido por caractere especial de escape chamado "Interprete como Comando" (IAC – *Interpret as Command*), que tem como valor 255.

Quando a aplicação que está recebendo os dados vê esse caractere especial, ele sabe que o próximo byte é um comando e não dados. Assim, dado que o comando Telnet "Interromper Processo", tem o valor 244, para enviar este comando o cliente Telnet precisa transmitir o byte 255 e depois o 244. O conjunto completo de caracteres especiais pode ser observado na Tabela 7.

Tabela 7: Caracteres especiais do Telnet

Nome	Código Decimal	Significado
SE	240	Fim dos parâmetros de sub-negociação.
NOP	241	Nenhuma operação
DM	242	Marca de dados.
BRK	243	Pare. Indica que uma tecla de Pare ou atenção foi acionada.
IP	244	Suspender, interromper ou abortar o processo ao qual o NVT esteja conectado.
AO	245	Abortar saída para o usuário.
AYT	246	Você está ai. Resposta do cliente para determinar se a conexão está ativa.
EC	247	Apagar ultimo caractere recebido.
EL	248	Apagar a ultima linha recebida.
GA	249	Vá em frente. Permite que a outra extremidade envie dados.
SB	250	Sub-negociação das opções indicadas a seguir.
WILL	251	Enviado para indica o desejo de usar uma opção particular. Também enviado para informar que precisa iniciar usando uma opção.
WONT	252	Enviado para indica uma recusa a um pedido de utilização de uma opção
DO	253	Enviado para indicar que a outra parte pode utilizar a opção. Também enviado para requisitar ao outro dispositivo que inicia usando uma opção.
DONT	254	Enviado para indicar que a outra parte não pode utilizar a opção.
IAC	255	Interprete como comando.

2.3.4 Negociação de Opções

Há uma variedade de opções que podem ser negociadas entre um cliente Telnet e um servidor usando comandos em qualquer estágio durante a conexão. Elas são descritas em detalhes em RFCs específicas nas quais as mais importantes podem ser observadas na Tabela 8.

Tabela 8: Tabela de opções do Telnet

Código Decimal	Nome	RFC
1	Eco	857
3	*Half-Duplex*	858
5	*Status*	859
6	Marca de tempo	860
24	Tipo de terminal	1091
31	Tamanho da janela	1073
32	Velocidade do terminal	1079
33	Controle de fluxo remoto	1372
34	Modo de linha	1184
36	Variáveis de ambiente	1408

A maioria das opções usadas pelo Telnet tem a finalidade de melhorar a eficiência de como os dados são transmitidos entre dispositivos. Por exemplo, por padrão o NVT assume comunicação do tipo *Half-Duplex*, no qual antes de qualquer transmissão os dispositivos necessitam enviar um comando "Vá em frente" (*Go Ahead*) após cada transmissão. Entretanto, praticamente todos os hardwares disponíveis atualmente suportam comunicações *Full-Duplex*, então normalmente os dispositivos concordam em usar a opção "Anula Vá em frente" (*Suppress Go Ahead*) para eliminar a necessidade de enviar este caractere depois de cada transmissão de dados. Similarmente, é possível aos dispositivos negociarem o envio de dados de 8 bits ao invés do padrão de 7 bits do NVT ASCII.

Habilitar opções utilizando negociações

O primeiro estágio na negociação de opções do Telnet é a decisão do cliente e do servidor de permitirem o uso de uma opção ou não. Um dos aspectos da simetria das operações de sincronização do Telnet é que ambas as extremidades da conexão podem escolher iniciar o uso de uma opção.

O dispositivo que inicia pode especificar que quer iniciar usando uma opção, ou se quer que o outro dispositivo inicie usando uma opção. O segundo dispositivo deve responder aceitando ou não o pedido. Uma opção só pode ser utilizada se ambos os lados estiverem de acordo quanto ao uso desta opção.

Esta negociação é realizada utilizando quatro comandos do protocolo Telnet, são eles:

1. **WILL**: Enviado pelo dispositivo que inicia para indicar que ele quer iniciar utilizando uma determinada opção. Há duas possíveis respostas:

    a. **DO**: Enviado para indicar que se está de acordo, e que o dispositivo iniciante pode começar utilizando esta opção. A opção está habilitada.
    b. **DONT**: Enviado para especificar que o iniciante não deve utilizar esta opção.

2. **DO**: Enviado pelo iniciante para requisitar que o outro dispositivo inicie utilizando uma opção. Este dispositivo pode responder:

    a. **WILL**: Enviado para especificar que o dispositivo concorda em iniciar utilizando uma opção. A opção está habilitada.
    b. **WONT**: Enviado para dizer ao iniciante que o dispositivo não pode utilizar a opção requisitada.

Como uma opção apenas pode ser utilizada se ambas as extremidades concordarem com o uso dela, também é possível desabilitar uma opção a qualquer tempo apenas enviando um comando DONT ou WONT para o outro dispositivo.

Independente da função, habilitar ou desabilitar, a estrutura de um comando segue o mesmo padrão de três bytes dado por:

{ [IAC], [Tipo da operação], [Opção] }

O primeiro byte (255) indica para o destinatário que o próximo byte representa um comando. O segundo, representa qual o comando será utilizado (WILL, WONT, DO, DONT). E o terceiro e ultimo representa a opção negociada (Eco, Tipo de Terminal,...).

2.4 Roteadores

A comunicação de dados tornou-se parte fundamental da computação. As redes de abrangência mundial reúnem dados sobre assuntos diversificados como condições atmosféricas, cotações de moedas e tráfego aéreo. Os grupos estabelecem listas de correio eletrônico para que possam compartilhar informações de interesse comum.

Infelizmente, a maioria das redes constitui entidades independentes estabelecidas para atender às necessidades de grupos isolados. Os usuários selecionam uma tecnologia de hardware que seja adequada aos seus problemas de comunicação. É impossível a estruturação de uma rede universal com base em uma única tecnologia de hardware, já que nenhuma rede única atende a todas as aplicações. Alguns usuários precisam de uma rede de alta velocidade para conectar-se a máquinas, mas essas redes não podem ser expandidas para alcançar grandes distâncias. Outros preferem uma rede de velocidade inferior que faça conexão com máquinas a centenas de milhas de distância.

Nos últimos 30 anos, foi desenvolvida uma nova tecnologia para possibilitar a interconexão de muitas redes físicas diferentes e fazê-las operar como uma unidade coordenada. Essa tecnologia, denominada "interligação de redes", acomoda distintas tecnologias básicas e hardware, proporcionando uma forma de interconectar redes heterogêneas e um conjunto de convenções que possibilitam as comunicações. A tecnologia de interligação de redes esconde os detalhes de hardware de rede, e permite que os computadores se comuniquem independentemente de suas conexões físicas.

Para avaliar o impacto causado pela tecnologia de interligação de redes, pense no quanto ela influenciou as atividades profissionais. Os centros nacionais podem coletar dados de um fenômeno natural e torná-los disponíveis para todos os cientistas. Os programas e serviços de computadores disponíveis em um determinado local podem ser utilizados por cientistas de outras localidades. Em conseqüência, houve um aumento na velocidade com que se desenvolveram as investigações científicas. E as mudanças são drásticas.

Ao longo dos anos, as agências governamentais norte-americanas perceberam a importância e o potencial da tecnologia de interligação de redes e financiaram as pesquisas que possibilitaram a interconexão global de redes. A tecnologia da ARPA (Advanced Research Project Agency), como era chamada, continha um conjunto de padrões de rede que especificavam os detalhes de sistema nas quais os computadores se comunicavam, bem como um conjunto de convenções para a interconexão de redes e para roteamento. Denominada oficialmente por Pilha de Protocolos de Interligação de Redes TCP/IP, e normalmente citado como TCP/IP, essa pilha pode ser utilizada para comunicação em qualquer conjunto de redes interconectadas. Algumas empresas, por exemplo, começaram a utilizar o TCP/IP para interconectar todas as redes de sua organização, ainda que a empresa não se comunique com redes externas. Outros grupos utilizam o TCP/IP para estabelecer comunicações entre sites geograficamente distantes [14].

Embora a tecnologia TCP/IP seja por si só notável, ela é especialmente interessante porque sua viabilidade foi demonstrada em larga escala. Ela constitui a tecnologia de base para uma interligação de redes global que conecta domicílios, universidades e escolas, organismos e laboratórios do governo em praticamente todos os países do mundo. Nos Estados Unidos, a National Science Foundation (NSF), o Department of Energy (DOE), o Department of Defense (DOD), a Health and Human Service Agency (HHS) e a National Aeronautics and Space Administration (NASA) participaram do financiamento da Internet e utilizaram o TCP/IP para conectar muitas de suas instalações de pesquisa. Atualmente, a tecnologia de interligação de redes não é mais conhecida como internet ARPA/NSF, internet TCP/IP, e sim, simplesmente por internet global, ou apenas Internet.

2.4.1 Introdução

A Internet nada mais é que um conjunto de redes interconectadas, formadas por diversos dispositivos tais como *switches*, *bridges* e roteadores [2, 37]. Embora os limites entre estes dispositivos fossem historicamente muito distintos, os avanços tecnológicos diminuíram estas distinções.

Os roteadores ofereceram uma potencialidade única de descobrir trajetos, ou rotas, através de grandes e complexas redes de computadores. Mais importante, os roteadores podiam comparar diferentes rotas através da rede podendo determinar a rota mais eficiente, entre todos os possíveis, entre dados pontos da rede.

O roteamento continua sendo crítico para as redes de computadores, porém deixou de ser função apenas de roteadores. O roteamento pode também ser realizado por computadores conectados a LANs (*Local Área Networks*) [3] e também por certos switches.

Visto que s redes de computadores são melhores entendidas a partir do uso de um modelo de camadas, vamos fazer uma explanação no modelo de referencia OSI [38]. Que forma um contexto para a análise dos mecanismos de passagem de dados através de computadores conectados, assim como entre redes, usando o Protocolo da Internet (IP).

2.4.2 Modelo de Referência OSI

A ISO (International Organization for Standardization) desenvolveu o modelo de referência OSI para facilitar a interconexão aberta entre sistemas de computadores[3]. Diz-se interconexão aberta, pois pode ser suportada em um ambiente com diversas plataformas diferentes.

O modelo de referência identifica e estratifica em camadas lógicas ordenadas todas as funções necessárias para estabelecer, usar, definir e desmanchar uma sessão de comunicação entre dois computadores, independente dos fabricantes e arquiteturas destes computadores.

Implícito nesta definição do modelo de referência OSI está a compreensão de que há uma distancia desconhecida e uma incerta quantidade de aparelhos que separam a comunicação entre estes dois dispositivos. Consequentemente, o modelo define mecanismos para a passagem de dados entre duas máquinas que compartilham a mesma LAN ou WAN (*Wide Area Network*) [1, 2]. Mais importante, o modelo identifica funções que habilitam máquinas que estejam a meio mundo de distancia uma da outra sem conexões diretas a passar dados entre si.

Atualmente, o modelo de referência OSI é considerado como lógico, mas velho e particularmente não utilizável. Quando foi desenvolvido, a aproximadamente 20 anos atrás, foi considerado revolucionário, mas naquela época os fabricantes de computadores podiam prender seus clientes em apenas uma arquitetura. Atualmente, isto não é mais possível, consumidores necessitam misturar e juntar componentes para construir sua própria infra-estrutura de rede de computadores.

As Sete Camadas

O modelo OSI categoriza os vários processos necessários numa sessão de comunicação em sete distintas camadas funcionais [38, 39]. As camadas são organizadas baseadas na seqüência de eventos natural que ocorre durante uma sessão de comunicação. A Figura 9 ilustra o modelo de referência OSI.

As camadas de 1 a 3 provêem acesso de rede e as camadas restantes são dedicadas à logística para o suporte de comunicações fim-a-fim.

Figura 9: Camadas do modelo de referência OSI

Camada 1: A Camada Física

A camada mais baixa do modelo de referência OSI é denominada Camada Física. Ela é responsável pela transmissão e recepção do fluxo de bits através do meio físico. Esta camada é responsável pela parte elétrica (ou ótica), mecânica e de interfaces funcionais ligadas ao meio físico. Ela é responsável por receber os quadros vindos da camada de enlace transmití-los na forma de bits através de um canal de transmissão. Algumas questões abordadas por esta camada, são:

1. bit 1/0 transmitido, bit 1/0 recebido;
2. Quantos volts representam os números 1 e 0;
3. Quantos microssegundos duram um bit;
4. Transmissão simultânea em ambos os sentidos;
5. Como é estabelecida e desfeita a conexão;
6. Quantos e para que serve cada pino do conector;

A camada física opera, literalmente, apenas com 1's e 0's e não tem nenhum mecanismo para determinar a significância dos bits transmitidos ou recebidos. Esta camada apenas preocupa-se com as características físicas das técnicas de sinalização seja ela mecânica ou ótica. Isto inclui a voltagem da corrente elétrica usada para transportar o sinal, o tipo de meio de comunicação e as características de impedância e até mesmo a forma física dos conectores usados para a conexão do meio físico.

Os meios de transmissão incluem quaisquer formas atuais de transporte de sinais gerados pelos mecanismos da primeira camada do modelo OSI. Alguns exemplos de meios de transmissão são: cabo coaxial, fibra ótica e cabo de par-trançado.

Camada 2: Camada de Enlace

A segunda camada do modelo de referência OSI é chamada de Camada de Enlace. Esta camada é responsável pela validação fim-a-fim dos dados que estão sendo transmitidos.

Os protocolos da camada de enlace são responsáveis por definir o formato das unidades de dados, denominada quadro, trocadas entre os dispositivos da rede ao enviar e receber essas unidades.

Um quadro é uma estrutura inerente à camada de enlace que contém informação suficiente para garantir que os dados possam trafegar toda a rede até o seu destino. Cada quadro da camada de enlace encapsula um datagrama da camada de rede.

Resumidamente, os principais serviços providos pelos protocolos da camada de enlace estão:

Delimitação de dados: Encapsulamento dos datagramas da camada de rede em quadros da camada de enlace antes de transmitir.

Detecção e correção de erros: Detecções de erro são realizadas em quadros de chegada, a partir de bits de detecção de erros que são enviados juntamente com os dados do quadro. Após identificado o erro, o protocolo da camada de enlace pode tentar identificar qual lugar do quadro contém o erro, tentando assim corrigí-lo.

Entrega confiável: Garantia de que o protocolo da camada de enlace vai transportar cada datagrama da camada de rede pelo enlace sem erro.

Controle de fluxo: Utilizado para evitar que um dos lados do enlace congestione o outro lado enviando quadros rápidos demais.

A camada física e de enlace juntas provêem todos os mecanismos necessários às aplicações para contatarem e se comunicarem com outros dispositivos conectados numa mesma LAN através de um endereçamento de máquina denominado endereço MAC (*Media Access Control*) [8], que é único para cada máquina. Por exemplo, na Figura 10 todas as máquinas podem acessar diretamente o servidor local. Entretanto isto não é possível quando se trata redes interligadas como a Internet.

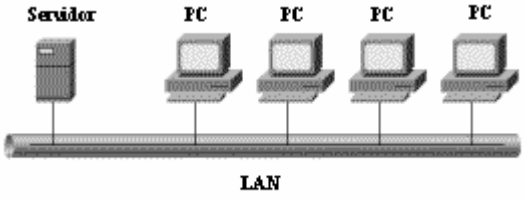

Figura 10: Entrega de diagramas localmente

Camada 3: Camada de Rede

Os protocolos desta camada são responsáveis pelo estabelecimento da rota a ser usada entre a fonte e o computador de destino. Esta camada é desprovida de quaisquer mecanismos de detecção e correção de erros e, consequentemente, é forçada a confiar no serviço de transmissão confiável fim-a-fim das camadas de enlace e transporte.

Embora algumas tecnologias da camada de enlace suportem entrega confiável muitos outros não suportam. Por causa disto, os protocolos da terceira camada, como o IP, assumem que

protocolos da quarta camada, como o TCP, irão prover esta funcionalidade em vez de deixar isto a cargo dos protocolos da segunda camada.

A Figura 11 ilustra a mesma rede da Figura 10. A única diferença é que há uma segunda rede conectada através de um roteador. O roteador eficientemente isola os dois domínios da camada de enlace. A única maneira de haver comunicação entre estas duas redes é através do endereçamento da camada de rede.

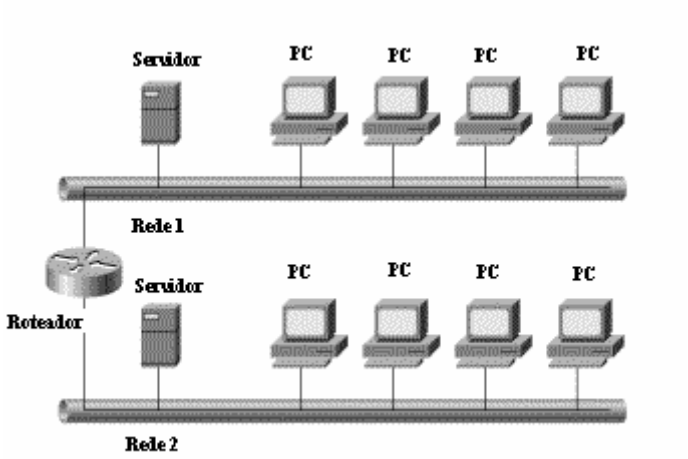

Figura 11: Entrega de pacotes entre redes

Nesta situação, se um usuário da rede 1 precisar acessar informações armazenadas na rede 2, então endereçamento da camada de rede será necessário. A camada de rede pode executar esta função intermediária porque ela tem sua própria arquitetura de endereçamento, que é separada e distinta do endereçamento de máquina da camada de enlace.

Os mecanismos da camada de rede foram implementados numa série de protocolos que podem transportar dados de aplicação através de segmentos de LAN ou até mesmo WANs, que são redes de longa distancia. Estes protocolos são chamados de "protocolos roteáveis" [36], pois seus datagramas podem ser enviados por roteadores para fora da rede local. Protocolos roteáveis incluem IP, IPX (*Internetwork Packet Exchange*) e AppleTalk [40].

Cada um destes protocolos, assim como outros protocolos roteáveis, tem sua própria arquitetura de endereçamento da terceira camada. Esta arquitetura de endereçamento é usada para identificar máquinas que estão conectadas por diferentes redes. Roteadores são necessários para calcular as rotas e transferir os dados contidos dentro dos pacotes dos protocolos roteáveis às máquinas que se encontram além da rede local da máquina transmissora.

Ao contrário das duas primeiras camadas, a camada de rede é opcional na comunicação de dados. A camada de rede é apenas necessária se os dois computadores estiverem em redes separadas, ou se as aplicações comunicantes precisarem dos seus serviços.

No primeiro caso, os diferentes domínios de rede precisam estar interconectados de alguma forma, como mostrado na Figura 11, senão a comunicação não irá ocorrer. Alternativamente, aplicações de software podem requerer o uso de mecanismos da camada de rede ou da camada de transporte, independente de como os dispositivos comunicantes estão interconectados.

Camada 4: Camada de Transporte

A camada de transporte provê serviços similares aos da camada de enlace, que é a integridade fim-a-fim das transmissões. Ao contrário da camada de enlace, a camada de

transporte pode prover esta função além do segmento local da LAN. Pode detectar pacotes que foram danificados ou perdidos durante a transmissão, e pode automaticamente gerar um pedido de retransmissão.

Outra função significativa da camada de transporte é a capacidade de reordenar os pacotes que, por muitas razões, podem ter chegado fora de ordem. Os pacotes de dados podem tomar vários caminhos através da rede, por exemplo, ou alguns podem ter se perdido no tráfego. Neste caso, a camada de transporte pode identificar a seqüência original de pacotes e por eles de volta em seqüência antes de passar seus conteúdos para a camada de sessão.

Bem como o relacionamento entre a primeira e segunda camada, a terceira camada do modelo OSI é geralmente relacionada com a quarta camada. Dois exemplos específicos conjuntos de protocolos roteáveis que integram estas duas camadas são: o padrão aberto TCP/IP e o IPX/SPX (*Internetwork Packet Exchange / Sequenced Packet Exchange*) [39, 40] da Novell.

Este relacionamento é ilustrado na Figura 12, usando o modelo de referência do TCP/IP. Juntas estas camadas provêem os mecanismos que possibilitam a transferência de informações entre máquinas fonte e destino através de redes de comunicações que atravessam o domínio da camada 2. Estas camadas também desempenham outras funções como a reordenação de pacotes que chegaram fora de ordem e retransmissão de pacotes que foram perdidos ou danificados.

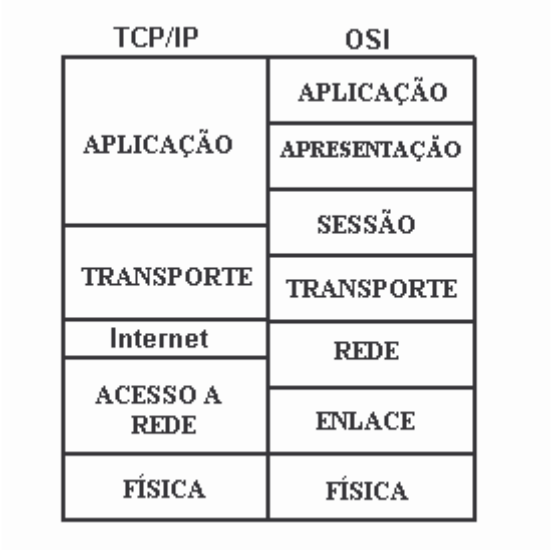

Figura 12: Comparação entre o modelo OSI e modelo TCP/IP

Camada 5: Camada de Sessão

Muitos protocolos empacotam suas funcionalidades em protocolos em suas camadas de transporte. Alguns exemplos específicos de serviços da camada de sessão são as chamadas remotas de procedimentos (RPCs - *Remote Procedure Calling*) e os protocolos de qualidade de serviços como o RSVP (*Resource Reservation Protocol*), que é um protocolo de reserva de largura de banda.

Camada 6: Camada de Apresentação

A camada de apresentação é responsável por gerenciar a forma como os dados são codificados. Nem todos os sistemas de computador utilizam o mesmo esquema de codificação de dados, e a camada de apresentação é responsável por prover a tradução entre esquemas de

codificação incompatíveis, como o ASCII (*Standard Code for Information Interchange*) e o EBCDIC (*Extended Binary Coded Decimal Interchange Code*) [41].

A camada de apresentação pode ser utilizada para sanar diferenças entre notações de ponto flutuante, assim como fornecer serviços de codificação e decodificação.

Camada 7: Camada de Aplicação

A camada mais alta do modelo de referência OSI é a camada de aplicação. Apesar deste nome, ela não inclui aplicações de usuário. Em vez disso, ela provê a interface entre estas aplicações e os serviços de rede.

Esta camada foi criada pensando-se na inicialização das sessões de comunicações. Por exemplo, um cliente de e-mail pode gerar uma requisição para recuperar novas mensagens do servidor de e-mail. Esta aplicação cliente gera automaticamente um pedido ao protocolo apropriado da camada 7 e abre uma sessão de comunicação para pegar os arquivos necessários.

2.4.3 Lógica adjacente

A identificação e estratificação da seqüência de eventos que suportam uma sessão de comunicação de rede é um conceito extremamente poderoso. Um dos benefícios chave desta abordagem é que habilita um conceito conhecido como "lógica adjacente", que se refere a aparente capacidade de protocolos de mesma camada, em máquinas de fonte e destino, de se comunicar diretamente um com o outro. Por exemplo, o protocolo IP da máquina fonte é logicamente adjacente ao protocolo IP da máquina de destino quando estiverem se comunicando.

Isto não é, naturalmente, como uma comunicação realmente ocorre. Na realidade, a orientação vertical da pilha de protocolos é um reconhecimento do fluxo funcional de processos e de dados dentro de cada máquina. Cada camada tem interfaces com suas camadas fisicamente adjacentes. Por exemplo, o protocolos IP, que é da camada de rede, na máquina fonte está fisicamente adjacentes aos protocolos TCP e UDP (*User Datagram Protocol*) da camada de transporte e aos protocolos da camada de enlace que também estejam presentes.

As diferenças entre o fluxo lógico das comunicações e o fluxo atual de uma sessão são ilustradas na Figura 13 usando o modelo de referência OSI. Como mostrado na figura, embora as comunicações fluam verticalmente através de cada pilha de protocolos, cada camada parece poder comunicar-se diretamente com a semelhante da outra máquina.

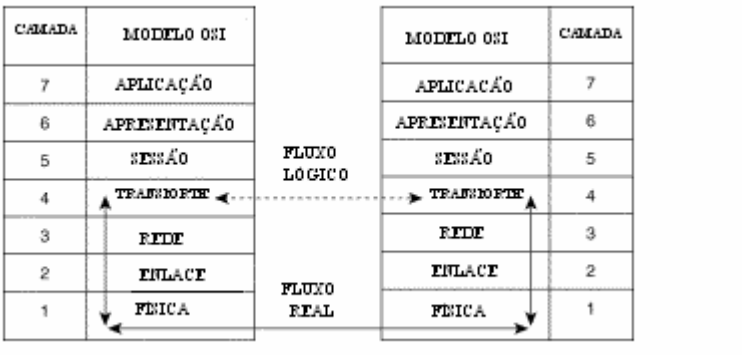

Figura 13: Diferença entre o fluxo real e o lógico de comunicação

A Necessidade de Rotear

O conceito de lógica adjacente trabalha justamente bem entre duas máquinas conectadas numa mesma LAN, e também o faz entre máquinas que estão a milhares de quilômetros de distancia conectadas pela Internet. Obviamente, há diferenças significativas entre estes dois exemplos, mas o conceito de adjacência lógica continua verdadeiro. A grande diferença está no estabelecimento da conexão entre a fonte e a máquina de destino.

Numa LAN, as duas máquinas podem se comunicar apenas enviando seus dados empacotados pelo meio de transmissão. No caso da Internet, as duas máquinas estão separadas por uma quantidade desconhecida de dispositivos de rede e meios de transmissão. Inundar os meios de comunicação na esperança de entregar os pacotes não é uma escolha viável.

A única solução lógica é identificar um caminho através da Internet até o destino. Encontrar um caminho, principalmente o melhor caminho, através do labirinto da rede é uma tarefa difícil. Os roteadores são os responsáveis pelo encaminhamento de pacotes utilizando o endereço de camada de rede.

Roteador

Ao contrário da maioria dos componentes de uma LAN, os roteadores são inteligentes. Mais importante, ele pode operar em todas as camadas do modelo OSI, que é melhor que apenas as duas primeiras. Isto os habilita a interconectar múltiplas LANs utilizando o endereço de terceira camada.

Um roteador deve ter duas ou mais interfaces físicas para interconectar transmissões de LANs e/ou WANs. O roteador aprende sobre os endereços das máquinas ou redes que estão conectados de algum modo através de suas interfaces. Uma lista destes endereços é mantida em tabelas que correlacionam os endereços da terceira camada com os números das portas ao qual estão direta ou indiretamente conectados.

Os roteadores utilizam dois tipos de protocolos de rede, e ambos operam na terceira camada. Eles são os protocolos roteáveis e os protocolos de roteamento. *Os protocolos roteáveis*, também conhecidos como protocolos roteados, são assim conhecidos por serem aqueles que encapsulam informações de usuário e dados em pacotes. Um exemplo de um protocolo roteado é o IP. O protocolo IP é responsável pelo encapsulamento de dados da aplicação para o transporte através de rede até os destinos apropriados.

Protocolos de roteamento [40] são usados entre roteadores para determinar rotas disponíveis, comunicar o que é conhecido sobre as rotas disponíveis e encaminhar pacotes de protocolos roteados ao longo destas rotas. O propósito de um protocolo de roteamento é fornecer ao roteador todas as informações necessárias sobre a rede para o envio dos pacotes de dados. Estes protocolos são responsáveis pela construção das tabelas de roteamento.

Roteamento

Roteadores são utilizados para a transferência de pacotes de dados ente dois dispositivos de rede que não necessariamente estejam numa mesma rede local. Roteamento é um processo cumulativo no qual se descobrem os caminhos através da rede até destinos específicos, comparam matematicamente rotas redundantes e constroem tabelas compostas por informações de roteamento.

Os roteadores podem verificar dentre as várias rotas diferentes disponíveis e escolher a melhor. Este processo é conhecido como “cálculo de rotas”. Esse calculo de rotas é realizado pelos roteadores com o auxílio dos protocolos de roteamento. Atualmente há uma grande

variedade destes protocolos para serem escolhidos. Muitos destes protocolos são amplamente suportados permitindo a utilização de diferentes roteadores, construídos por diversos fabricantes.

Alguns exemplos de protocolos de roteamento são:

- *Routing Information Protocol* (RIP)
- *Open Shortest Path First* (OSPF)
- *Interior Gateway Routing Protocol* (IGRP)

Cada um destes protocolos de roteamento possui suas próprias características, benefícios e limitações. Em grandes redes, como a Internet, é muito comum se encontrar alguns destes protocolos trabalhando em conjunto.

O roteamento é a principal forma utilizada na Internet para a entrega de pacotes de dados entre dispositivos finais (equipamentos de rede de uma forma geral, incluindo computadores, roteadores e etc.). O modelo de roteamento utilizado é o do salto-a-salto (*hop-by-hop*), onde cada roteador que recebe um pacote de dados, abre-o, verifica o endereço de destino no cabeçalho IP, calcula o próximo salto que vai deixar o pacote um passo mais próximo de seu destino e entrega o pacote neste próximo salto.

Este processo se repete sucessivamente até a entrega do pacote ao seu destinatário. No entanto, para que isto funcione, é necessário que os roteadores responsáveis pelo processamento de cada pacote de dados estejam corretamente configurados**.** Existem muitas maneiras de configurar um roteador, tais como: via console diretamente conectado, via Telnet, SNMP , HTTP, entre outros [42, 43].

Capítulo 3

Aplicação

Neste capítulo é apresentado o TelnetME, o qual foi desenvolvido, basicamente, com o objetivo de prover um meio de unir as vantagens e benefícios advindos dos dispositivos móveis para permitir a correta configuração e a simplificação do gerenciamento de dispositivos de rede, ou seja, dar uma solução alternativa à configuração destes dispositivos sem que haja necessidade da presença de um computador em mãos ou na rede local específica, proporcionando, então, uma maior facilidade em sua configuração.

Este novo meio de configuração de dispositivos de rede acarreta num significativo avanço para administradores de redes por propiciar um método mais contemporâneo e dinâmico quando comparado aos métodos atualmente disseminados.

Um exemplo indagável para o uso desta aplicação proposta seria advinda da necessidade de um gerente ou administrador de redes de configurar um dispositivo de rede de sua empresa mesmo quando estiver no meio do trânsito ou em algum lugar onde não haja a disponibilidade de um computador ou de algum tipo de conexão com a Internet.

Esta abordagem poderia impedir, por exemplo, que uma empresa de comércio eletrônico deixasse de realizar transações comerciais, devido unicamente a um dispositivo de rede mal configurado que estivesse impedindo o acesso dos usuários ao servidor da empresa. Mais especificamente, a versão atual da aplicação desenvolvida se preocupa com a configuração de roteadores, e que após alguns testes, além de se mostrar capaz de realizar a função pela qual foi implementada, a configuração de roteadores, mostrou-se também capaz de desempenhar funções pertinentes a aplicações de clientes Telnet, tal como o acesso a servidores de e-mail.

3.1 A Aplicação

O TelnetME é um cliente Telnet para dispositivos móveis, celulares ou PDAs, desenvolvido com Java ME e com plataforma CLDC/MIDP, que é provido de uma interface adaptada às potencialidades limitadas dos dispositivos móveis. É capaz também de emular um terminal compatível com o ANSI, com barra de rolagem de texto e conexão direta via TCP com *sites* ou dispositivos remotos.

Algumas funcionalidades disponíveis são de grande ajuda durante à configuração de roteadores, tais como:

- suporte a alguns caracteres especiais de controle e de escape;
- suporte a alguns símbolos utilizados na escrita;
- gerenciamento e armazenamento de hosts;
- suporte a histórico de comandos.

Outras funcionalidades suportadas são específicas do protocolo Telnet, como negociação de opções, como:

- eco;
- *window Size;*
- autenticação automática em hosts.

Estas funcionalidades unidas com outras anteriormente descritas provêem todas as necessidades específicas para o estabelecimento de sessões e, posteriormente, uma correta configuração de dispositivos de rede, especialmente roteadores.

3.2 Requisitos

TelnetME necessita de um dispositivo compatível com Java ME e com a plataforma MIDP 1.0, que suporte conexões via *sockets* e que contenha uma memória disponível de aproximadamente 1 Megabyte. Atualmente, o número de dispositivos, de diversos fabricantes, que disponibilizam estas funcionalidades é muito grande e podem ser citados:

1. Nokia: 3220, 6600, 7610, 2310;
2. Motorola: V300, V60, C650;
3. Siemens: CX65, S65, CF62;
4. Samsung: D500, X480, E320;
5. Dispositivos com PalmOS;

É importante salientar que apesar de funcional, o protocolo Telnet não é seguro e antes de utilizar o TelnetME, o usuário deve ter consciência que durante uma comunicação utilizando este protocolo as informações são enviadas puramente como texto, sem nenhuma proteção para senhas ou quaisquer outros dados. Uma forma de proteger os dados durante a comunicação é utilizar o protocolo SSH (*Secure Shell Remote Login Protocol*)[44] juntamente com o Telnet, o que além de acarretar um esforço muito maior, é de difícil implementação e será deixado para trabalhos futuros.

3.3 Casos de Uso

A Figura 14 ilustra através de um diagrama as funcionalidades do sistema, ou seja, as possíveis ações que podem ser tomadas por um usuário ao utilizar o TelnetME.

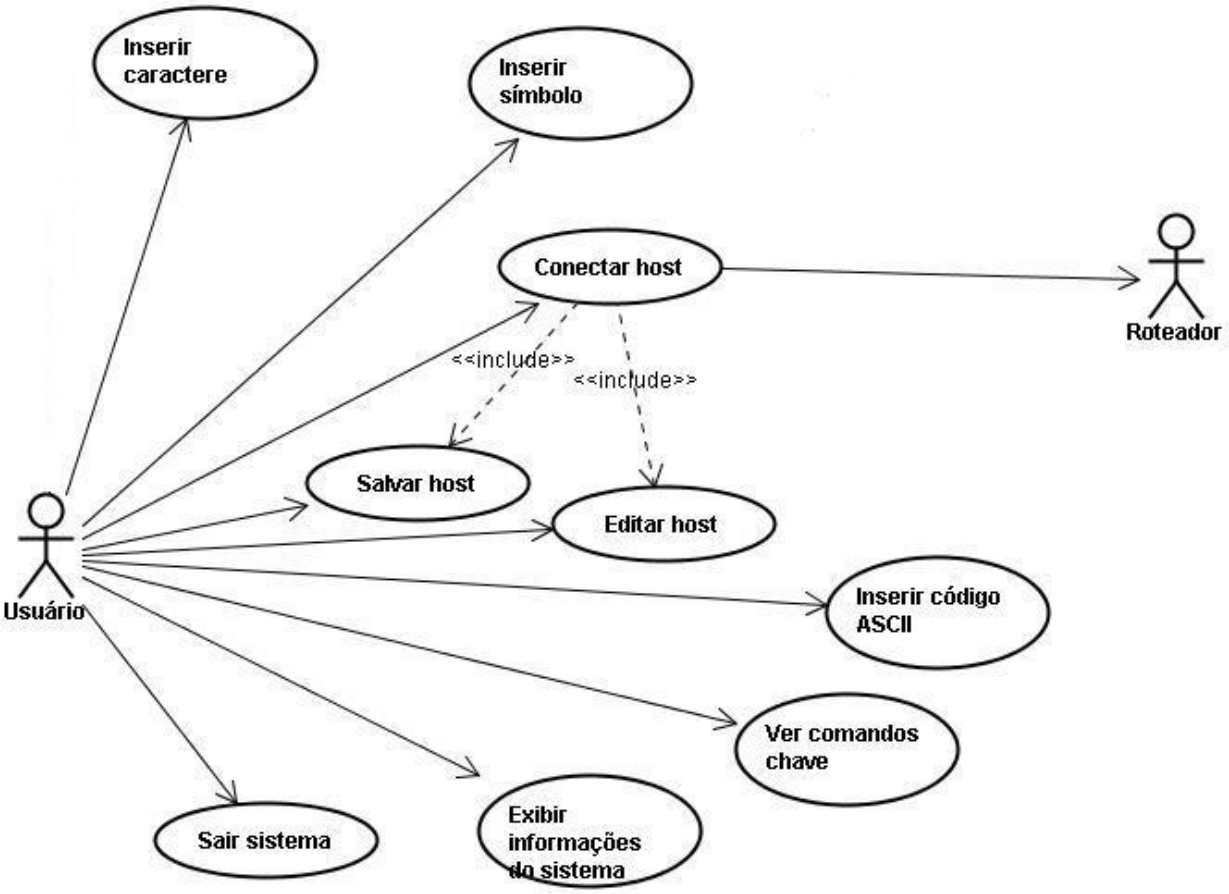

Figura 14: Casos de uso do sistema

Dentre os casos de uso da aplicação, dois podem ser considerados cruciais quando da utilização do sistema, “Salvar host” e “Conectar host”, portanto os mesmos foram escolhidos para serem apresentados em mais detalhes a seguir. No Apêndice A todo o código fonte do sistema é apresentado, enquanto que no Apêndice B é ilustrado a especificação de todos os casos de uso .

Salvar host

A principal funcionalidade do primeiro caso de uso e sua fundamental importância vem da necessidade de um usuário salvar os parâmetros de conexão para um determinado host, seja ele um servidor ou um roteador. Esta funcionalidade garante que estes parâmetros possam ser salvos através do armazenamento persistente do dispositivo, podendo ser resgatada e utilizada posteriormente.

Com isto, o usuário terá a oportunidade de armazenar seus hosts mais importantes, conectando-se a eles mais facilmente sempre que preciso. A Figura 15 ilustra os passos necessários para o armazenamento dos parâmetros de configuração de um determinado host na memória do dispositivo.

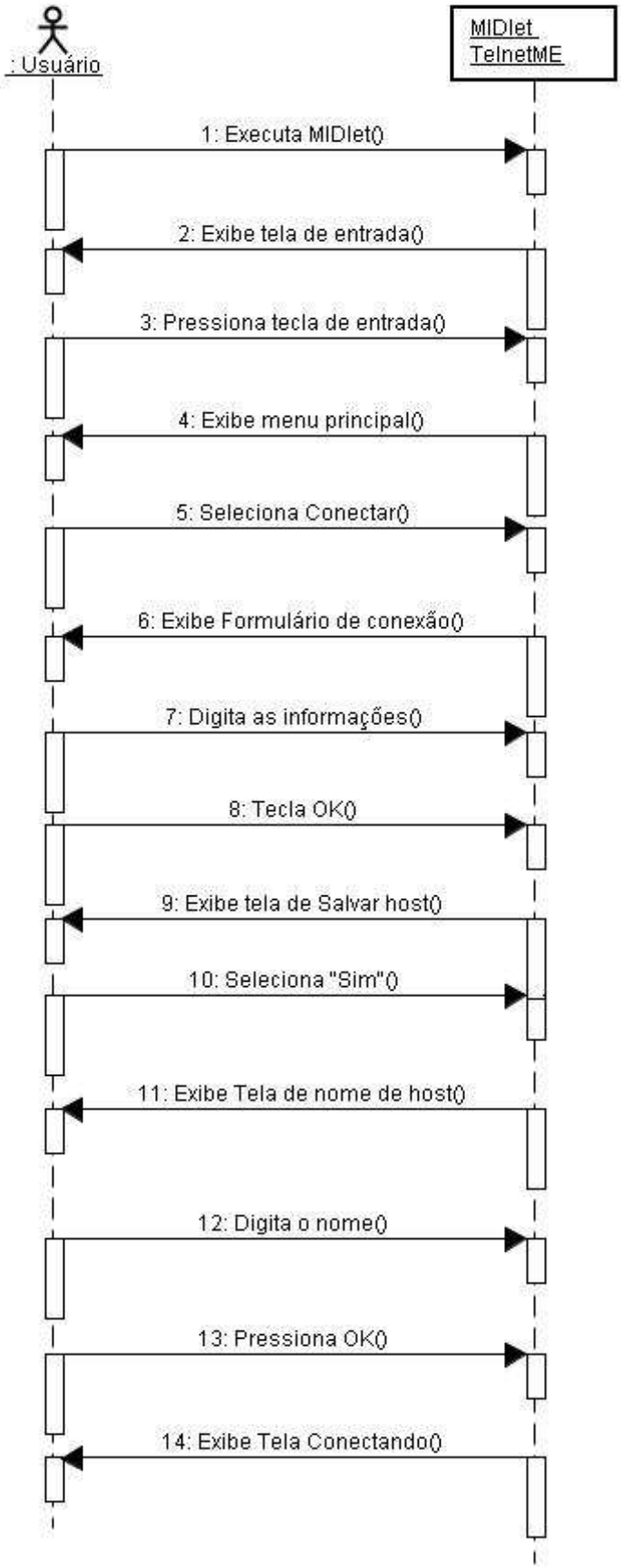

Figura 15: Diagrama de seqüência do caso de uso Salvar host

A partir da execução da MIDlet pelo usuário, a aplicação irá automaticamente exibir uma tela de entrada ao qual solicita ao usuário a selecionar uma determinada tecla para entrar no sistema, como pode ser visto na Figura 16. Esta tecla, em especial, pode variar de dispositivo para dispositivo, visto que o controle e o gerenciamento dos códigos de tecla podem variar de acordo com a plataforma e fabricante escolhido.

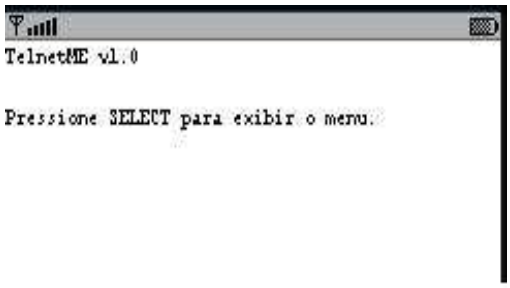

Figura 16: Tela inicial da aplicação

Após pressionada a tecla, o menu principal será exibido contendo as seguintes opções: Entrada de texto, Conectar, Comandos chave, Informações e Sair. Ao se escolher a opção Conectar será exibido um formulário no qual o usuário deve informar o endereço do host, a porta, e, opcionalmente, o nome de usuário e senha. Estas duas telas podem ser observadas na Figura 17.

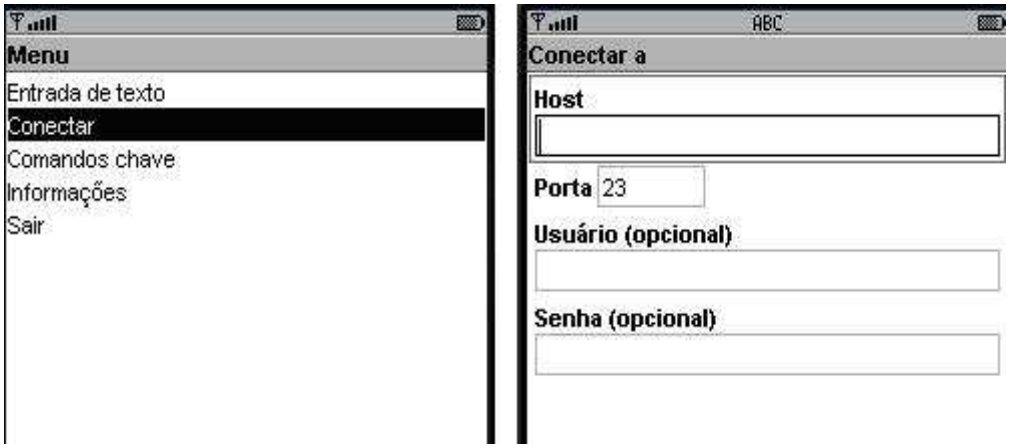

Figura 17: Telas de Menu principal (esquerda) e Formulário conectar (direita)

A porta 23 foi colocada como constante por ser esta a porta padrão utilizada pelo protocolo Telnet, mas pode ser alterada de acordo com as necessidades do dispositivo ao qual se quer conectar. Outra informação importante a ser explicitada é o fato de que senhas são enviadas em forma de texto, podendo assim ser interceptada.

Depois de digitadas as informações, e pressionada a tecla OK e será dada a opção de salvar as configurações previamente definidas na memória do dispositivo. Esta opção é dada ao usuário, porém ainda que este decida por não salvar as informações, a conexão será iniciada. Entretanto se o usuário decidir por se conectar novamente àquele host, ele terá que fornecer novamente as informações necessárias.

Optando-se por salvar as informações, uma nova tela será exibida, na qual um nome que identifique o host ao qual pertencem estas informações deverá ser digitado e, posteriormente, pressiona-se OK, salvando o host na memória. As telas de salvar host e de inserção de nome do host são ilustradas na Figura 18.

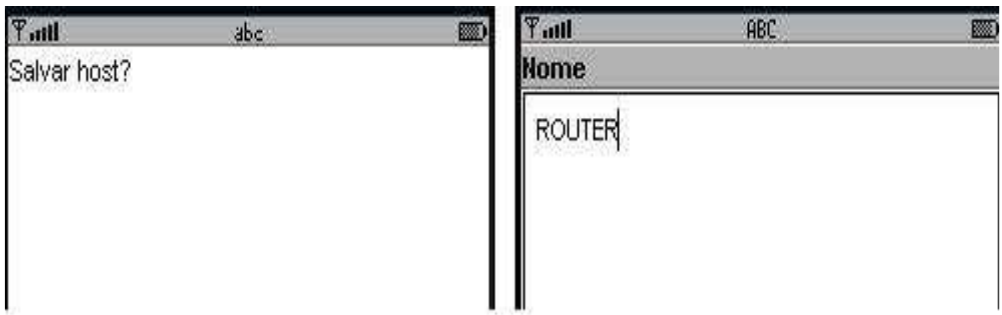

Figura 18: Telas de Salvar host (esquerda) e Nome do host (direita)

Conectar host

O segundo caso de uso, "Conectar host", é o mais importante de todos. Ele mostra os passos necessários para o estabelecimento de uma sessão Telnet com um roteador, ou também um servidor Telnet. Neste caso, parte-se do pressuposto de que já existe uma configuração de host salva na memória do dispositivo. A Figura 19 mostra os passos necessários para o estabelecimento da sessão, visto não haver a necessidade de inserir os parâmetros de conexão do host, sendo necessário apenas escolher um dos disponíveis.

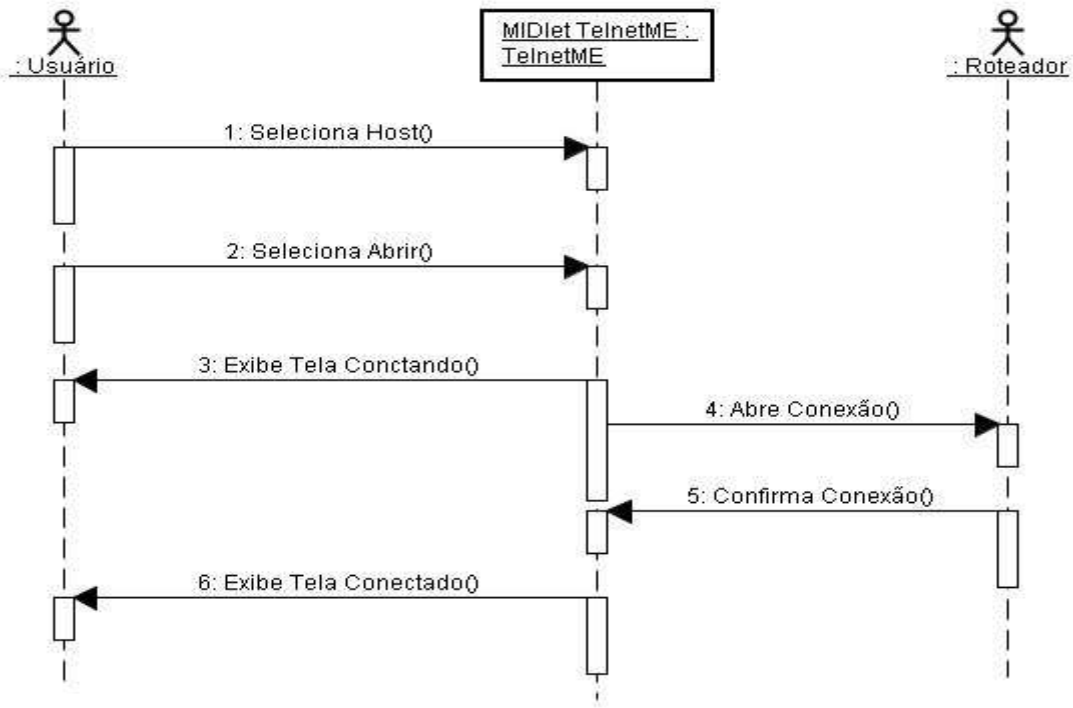

Figura 19: Diagrama de seqüência do caso de uso Conectar host

Inicialmente, deve-se escolher o host ao qual se quer conectar, como mostrado na Figura 20. Selecionada a opção "abrir", a MIDlet irá tentar estabelecer uma sessão com este host, que responderá com um IAC após o contato ser confirmado. Além do IAC, o host poderá e deverá enviar informações adicionais que serão exibidas ao usuário, que poderá ter a certeza que a sessão

foi estabelecida. No caso de ocorrência de falhas durante a comunicação, um alerta será exibido informando ao usuário o erro.

Figura 20: Tela de Escolha de host

Após estabelecida a conexão, dados podem ser trocados por ambas as extremidades da conexão e a aplicação fornecerá todo o suporte para o ajuste da tela de acordo com o envio e a chegada de dados, ativando, quando necessário, a barra de rolagem de texto.

3.4 Arquitetura do Sistema

Visando obedecer aos critérios de baixo poder de processamento e memória limitada, o sistema foi desenvolvido utilizando-se apenas três classes. É importante salientar que, mesmo sendo utilizada uma linguagem orientada a objetos como Java ME, certas características deste tipo de linguagem não foram empregadas, tais como: a utilização de classes e métodos abstratos, alto nível de modularidade e alguns padrões de projeto.

A decisão de não seguir estritamente os padrões do Java deu-se da necessidade de implementar uma aplicação pequena, simples e funcional, que atendesse a diversos tipos e arquiteturas de dispositivos móveis. Com resultado final, levando-se em consideração portabilidade e desempenho, foi obtida uma aplicação constituída por três classes básicas Java e com tamanho aproximado de 10 Kbytes.

As classes do projeto foram definidas e desenvolvidas com atribuições, responsabilidades e funcionalidades específicas. Estas classes são:

1. **`TelnetME`**
2. **`ControlaCanvas`**
3. **`ControlaFluxo`**

A classe `TelnetME`

Esta é a classe principal da aplicação, e conseqüentemente, estende de MIDlet em `javax.microedition.midlet.MIDlet`, e, assim sendo, possui os principais métodos responsáveis pelo ciclo de vida da aplicação como um todo.

Na classe MIDLET encontra-se um método para cada mudança de estado da aplicação. Assim que o usuário fizer uma chamada a um MIDLET, este aplicativo usará o método `startApp` para mudar o estado deste MIDLET, neste caso para “ativo”.

Supondo que com o aplicativo no estado “ativo”, o usuário receba uma ligação. Neste caso, o método `pauseApp` é invocado para que o estado do aplicativo mude para “interrompido”, e, assim que esta ação for realizada, o método `notifyPaused` será chamado.

Após finalizar a ligação o usuário poderá voltar ao aplicativo que estava “interrompido”. Neste momento o método `startApp` modifica o estado do aplicativo que estava “interrompido”, voltando novamente para ativo.

Após a mudança do estado, o método `resumeRequest` será chamado. E, finalmente, quando o usuário deseja finalizar o aplicativo, o método `destroyApp` é responsável por passar a MIDlet para o estado “destruído”. Esta mudança acarreta na invocação do método `notifyDestroyed`. No caso de ocorrer algum problema na mudança entre estados, a exceção `MIDletStateChangeException` é lançada. Quando isso ocorre o estado da aplicação é imediatamente alterado para destruído.

A Figura 21 ilustra a seqüência dos possíveis estados de uma MIDlet e os respectivos métodos de mudança de estado.

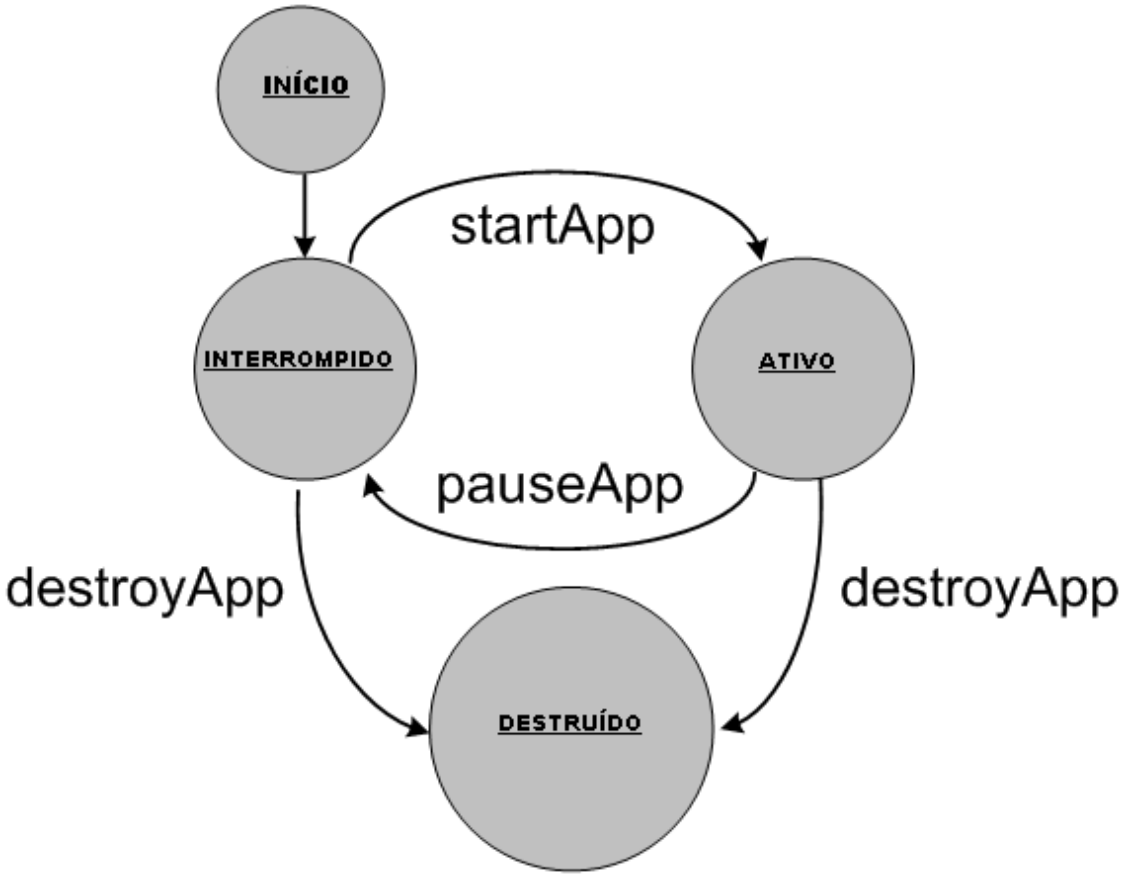

Figura 21: Ciclo de vida de uma MIDlet

As declarações destes métodos contidos no `TelnetME` são mostradas abaixo:

```
public void startApp(){ }
public void pauseApp(){ }
public void destroyApp(boolean flag){
     ControlaFluxo fluxo = ControlaCanvas.contFluxo;
         if(fluxo != null)fluxo.fechaConexao();
      notifyDestroyed();
    }
```

Além destes métodos obrigatórios, a classe `TelnetME` implementa outros métodos como:

1. `TelnetME`: método construtor responsável por inicializar o *display* e por invocar uma nova instância da classe `ControlaCanvas`.

2. `armazena`: este método é responsável pela inicialização das informações de hosts contidas na memória persistente [27] do dispositivo e adicioná-las num vetor de Strings que, posteriormente, será utilizado pela a aplicação.

3. `removeRecordStore`: método responsável pela remoção de informações de host da memória persistente.

4. `adicionaRecordStore`: método responsável pela adição de informações de host na memória persistente.

A classe `ControlaCanvas`

A principal funcionalidade da classe `ControlaCanvas` é o gerenciamento da exibição de mensagens, formulários, alertas e menus na tela do dispositivo. Ela também é responsável pela forma à qual a tela é dimensionada e adaptada para a impressão dos caracteres, assim como o tratamento de caracteres especiais. Resumindo, todas as funcionalidades da aplicação que de certa forma necessite ou utilize a tela do dispositivo é provido por esta classe.

Por haver restrição quanto ao tamanho da tela dos dispositivos, alguns cálculos de largura, altura, tamanho da fonte, entre outros, são realizados para garantir que o texto de entrada ou de saída se ajuste à tela. Estes cálculos são também importantes para a utilização da barra de rolagem, que é uma outra funcionalidade provida por esta classe.

Dentre os métodos implementados por esta classe podemos destacar:

1. `imprimeString`: este método é responsável por imprimir na tela do dispositivo uma determinada string que é passada como parâmetro.

2. `imprimeChar`: este método auxilia o método `imprimeString` na exibição de um texto na tela do dispositivo.

3. `fazerDados`: este método é responsável por formar um tipo de dado que contém todas as informações de um host.

4. `atualizaHistorico`: método que atualiza um vetor que contém os últimos *strings* de texto transmitidos. Este histórico é bastante útil, pois minimiza o tempo de digitação dos comandos a serem enviados ao dispositivo de rede.

A classe `ControlaFluxo`

Esta última classe é responsável pelo estabelecimento de conexões e pelo fluxo e exibição em tela dos dados de entrada e saída da aplicação. Outra funcionalidade extremamente importante disponibilizada por esta classe são as negociações de opções do Telnet. A citada exibição de entrada e saída de dados corresponde aos dados recebidos através da conexão e aos dados inseridos pelo usuário via teclado respectivamente.

O estabelecimento de conexões é garantido através do uso de conexões via sockets que é o mecanismo de mais baixo nível de comunicação básica entre um dispositivo móvel e um servidor remoto ou entre dispositivos móveis.

Conexões via sockets são disponibilizadas através da GCF (*Generic Connection Framework*) [11, 12]. A GCF provê um conjunto único de abstrações que podem ser usados em nível de programação para manipular diversas formas de comunicação, ao invés de utilizar diferentes abstrações para diferentes protocolos.

Neste *framework*, todas as conexões são criadas usando o método estático `open` da classe `Connector`. Se bem sucedido, este método retorna um objeto que implementa uma das interfaces de conexão genérica. A Figura 22 ilustra o relacionamento destas interfaces em nível hierárquico, na qual a interface `Connection` é a interface base.

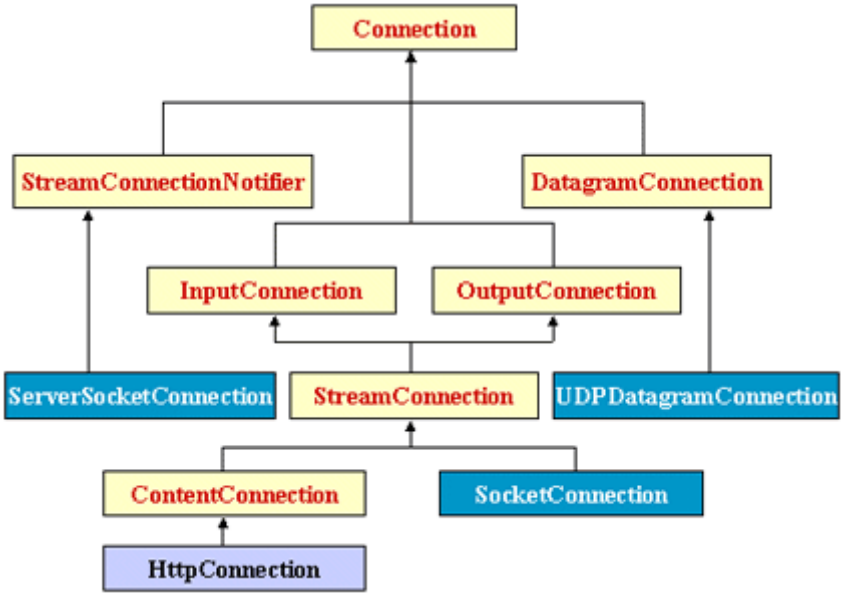

Figura 22: Hierarquia de interfaces da GCF [12]

Na classe `ControlaFluxo` a conexão é aberta através de *sockets* através das linhas de código mostradas a seguir:

```
connection = (StreamConnection)Connector.open("socket://" +
dados[1] + ":" + dados[2], 3, false);
iStrm = connection.openInputStream();
oStrm = connection.openOutputStream();
```

As duas linhas de código abaixo da declaração de conexão são responsáveis pela abertura de fluxo de entrada e envio de dados da aplicação através da conexão.

Outros métodos importantes presentes na classe `ControlaFluxo` são:

1. `imprimeFluxo`: método responsável por enviar um texto do fluxo para o método apropriado da classe `ControlaCanvas` para ser impresso na tela do dispositivo.
2. `fechaConexao`: método responsável por fechar a conexão, assim como os fluxos de entrada e saída de dados.
3. `run`: este é o método da interface `Runnable`. É o método responsável por selecionar os dados de entrada, identificando e diferenciando os bytes de dados dos comandos Telnet.

Mostradas algumas das funcionalidades de cada classe presente na aplicação, é importante apresentar o relacionamento entre estas classes. Este relacionamento é ilustrado através do diagrama de classe mostrado na Figura 23. É importante salientar que este diagrama mostra apenas uma parte dos métodos e atributos presentes nestas classes.

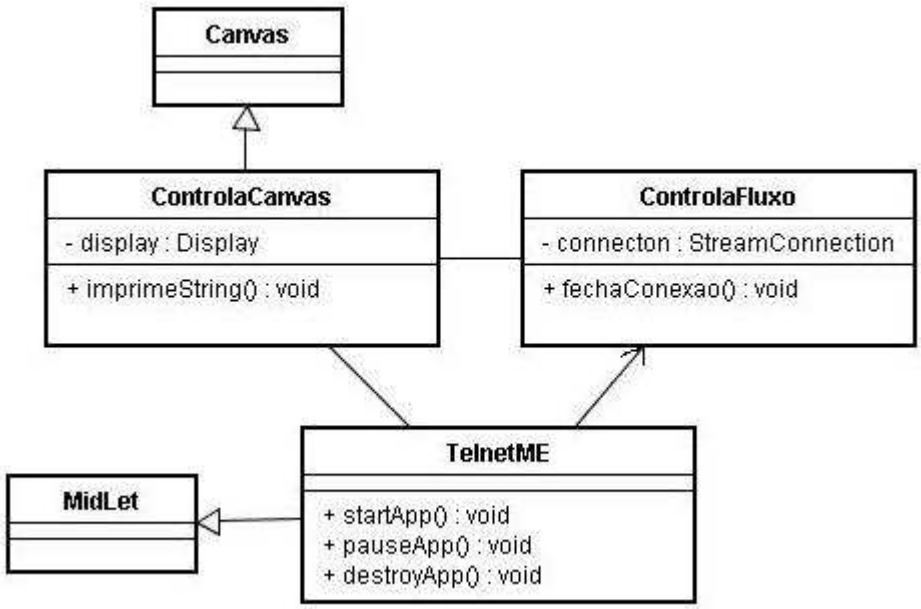

Figura 23: Diagrama de classes do sistema

3.5 Estudo de Caso

O estudo de caso realizado com o fim de provar a veracidade da aplicação desenvolvida, TelnetME, e do seu funcionamento na configuração de roteadores, deu-se com a utilização de um aparelho celular Nokia 3220 e um roteador Cisco 7200 VXR NPE-300.

Este modelo específico de aparelho celular foi utilizado pois era um dos poucos disponíveis que atendia aos requisitos mínimos impostos pela aplicação TelnetME, entre eles a capacidade de estabelecer conexões via sockets e suporte a MIDP 1.0. Um outro fator decisivo na escolha deste dispositivo foi a disponibilidade de um cabo de dados compatível com o aperelho, necessário para uma instalação mais rápida e sem custos adicionais da aplicação. Uma outra forma testada de instalação foi via WAP, porém a instabilidade das conexões atuais via GPRS fizeram com que a tentativa falhasse.

A grande dificuldade da realização dos testes do TelnetME na configuração de um roteador foi a difícil disponibilização deste dispositivo, pois este, normalmente, está em constante operação, além de ser necessário ter acesso como administrador, ou seja, login e senha, para a autenticação e posterior configuração do aparelho. Entretanto com o auxílio e permissão do administrador de rede do Ministério da Saúde (DATASUS/PE), os testes poderam ser realizados.

A primeira etapa realizada antes do início da configuração do roteador foi a reinicialização do mesmo e o salvamento dos parâmetros de configuração, para que no caso da ocorrência de erros, estes parâmetros não fossem perdidos.

A segunda etapa foi a liberação da aceitação de conexões Telnet externas, ou seja, de dispositivos que se encontram fora da rede interna do Ministério da Saúde. Após a liberação, o roteador foi novamente reiniciado.

Como os parâmetros de configuração do roteador já tinham sido salvas e as conexões externas via Telnet liberadas, o próximo passo foi a utilização do TelnetME na tentativa do estabelecimento da conexão. Devido a alguns problemas de sinal, muitas das primeiras tentativas falharam fazendo com que fosse necessário se deslocar para uma outra área onde o sinal pudesse se manter constante e com um bom nível.

Após solucionados estes problemas iniciais, a aplicação mostrou-se capaz de estabelecer comunicação e após a inserção, pelo administrador, do número IP do roteador e do login e senha a conexão foi estabelecida e só então o processo de configuração do roteador pôde ser iniciado.

Inicialmente o administrador propôs alterar o modo de execução do dispositivo para "modo administrador", buscando assim ter acesso a parâmetros de configuração mais simples e específicos. Dentre os parâmetros configurados durante o teste podemos citar:

- Configuração do OSPF [5];
- Configuração do IGRP [4];
- Alteração da senha de administrador;
- Configuração de uma das portas de comunicação;

Finalmente é importante salientar que a aplicação TelnetME mostrou-se capaz de desempenhar as funções pela qual foi desenvolvida, ou seja, ela foi capaz de estabelecer conexão e configurar determinados parâmetros de configuração de um roteador.

3.6 Diretrizes para o Desenvolvimento de MIDlets

Algumas das diretrizes úteis durante o processo de desenvolvimento de aplicações para dispositivos móveis são:

- Manter a interface simples: manter a interface do aplicativo simples e intuitiva, de forma que o usuário não tenha duvidas do que deve ser feito. Deve-se reduzir a quantidade de informações exibidas no dispositivo e oferecer sempre que possível listas de seleção ou menus para o usuário ao invés de forçá-lo a digitar uma seqüência de teclas.

- Definir os dispositivos-alvo: deve-se ter uma noção dos dispositivos no qual a aplicação será utilizada antes mesmo do começo do desenvolvimento da aplicação. Deve-se primeiramente saber as necessidades dos usuários em potenciais, os requisitos impostos pelas redes e em que plataformas o aplicativo funcionará (telefones celulares, PDAs, etc).

- Gerenciamento de mensagens: em certas aplicações pode ser possível fazer diversas tarefas ao mesmo tempo em que uma mensagem está sendo recebida ou transmitida. Como conseqüência das baixas taxas de transmissão de dados em redes sem fio, a transmissão ou recepção da mensagem pode ser demorada, sendo importante manter o usuário informado do progresso da transmissão.

- Definir as funções da aplicação: quando a aplicação necessita desempenhar funções de comunicação cliente-servidor, as tarefas das duas partes devem estar muito bem definidas, ou seja, quais as funções que devem estar implementadas no dispositivo móvel e quais devem estar no servidor. MIDlets permitem residir muitas da funcionalidades da aplicação no dispositivo, podem recuperar dados do servidor, processá-los e exibi-los na tela do dispositivo localmente. Esta abordagem reduz a interação com a rede sem fio, conseqüentemente diminuindo o tráfego da rede.

- Definição dos tipos de dados: os dados podem ser representados de muitas formas, uns mais compactos do que outros. Deve-se buscar utilizar o tipo de dado mais compacto, visando diminuir a quantidade de bits armazenados e transmitidos, tendo-se sempre em mente que a utilização de certos tipos mais compactos podem acarretar em perda de desempenho. Por exemplo, textos normalmente são muito mais compactos se representados através do tipo `StringBuffer` do que se representados na forma de uma string.

Capítulo 4

Conclusão

4.1 Conclusões

A principal contribuição deste Trabalho de Conclusão de Curso é apresentar uma nova e versátil abordagem para a configuração de roteadores através do uso de dispositivos móveis, como o telefone celular. Para isto foi utilizada a linguagem Java ME e o protocolo Telnet, como explicado no Capítulo 3, possibilitando o desenvolvimento de uma aplicação, denominada TelnetME, que atendesse aos requisitos dos dispositivos móveis e que implementasse as funções e opções básicas de um cliente Telnet.

Esta nova abordagem de configurar roteadores através de um cliente Telnet que reside em um dispositivo móvel visa principalmente a assistir os administradores e gerentes de redes na reparação de erros de configuração nos roteadores da empresa, principalmente quando estes não dispõem de um computador interligado ao roteador ou que possa se conectar a ele através do protocolo HTTP.

É importante salientar que as conexões remotas através do protocolo Telnet predominantemente não são seguras, sendo passíveis de interceptação, devendo-se então ser utilizada apenas em casos de extrema importância e urgência e, de preferência, em conjunto com algum outro protocolo ou aplicação que forneça uma maior segurança aos dados transmitidos. Uma forma de tornar conexões Telnet mais seguras seria utilizar o protocolo SSH em conjunto com o protocolo Telnet, porém esta nova implementação ficará para as próximas versões do sistema.

Um outro erro que não pode ser cometido é a comparação, seja de desempenho, interface gráfica, funcionalidades ou outras, entre o TelnetME e os clientes Telnet voltados para *desktops*. Este tipo de comparação não é válida, pois as arquiteturas e recursos envolvidos são muito diferentes. A quantidade de memória, poder de processamento, taxa de transmissão de dados, latência e potencialidades de interface de usuário dos *desktops* são inúmeras vezes superiores aos disponíveis nos dispositivos móveis.

Por implementar um cliente Telnet simples, porém robusto, o TelnetME mostrou-se capaz de realizar outras operações além da configuração de roteadores, tais como o acesso a servidores de e-mail, servidores de dados e comunicação com diversos sistemas que disponibilizam serviços através do protocolo Telnet na Internet.

4.2 Dificuldades Encontradas

Durante todas as etapas de desenvolvimento e testes da aplicação várias dificuldades surgiram. Algumas simplesmente por falta de conhecimento das tecnologias envolvidas, outras pela falta de experiência com os dispositivos de teste, entre vários outros. Dentre os problemas mais importantes podem ser citados:

- entendimento das restrições da linguagem Java ME;
- entendimento do funcionamento do protocolo Telnet;
- instalação da aplicação em celulares;
- disponibilidade de roteador para testes.

Entendimento das restrições da linguagem Java ME: por haver diferenças entre as bibliotecas e funcionalidades disponíveis nas versões Java ME e Java SE, um prévio estudo foi necessário para minimizar futuros erros durante o processo de desenvolvimento e depuração da aplicação.

Entendimento do funcionamento do protocolo Telnet: o conhecimento de como implementar o controle de negociação de opções foi um fator de grande complicação durante o desenvolvimento do projeto. Decidir quais opções deveriam ou não ser suportadas, assim como prover um meio para suportar as negociações foi um fator bastante complicado.

Instalação da aplicação em celulares: mesmo suportando MIDP 1.0 e até MIDP 2.0, a instalação de aplicações em celulares e em outros dispositivos móveis ainda é bastante complicada. Certos aparelhos rejeitam instalações feitas via Wap [45] e HTTP, aceitando apenas instalações no qual se utiliza um cabo de dados específico para o aparelho.

Disponibilidade de conexões via sockets nos aparelhos: certos aparelhos, mesmo sendo capazes de estabelecer conexões através de GPRS, EDGE ou por outra forma, não provêem suporte a sockets. Esta restrição é encontrada frequentemente em dispositivos móveis da fabricante Nokia.

Disponibilidade de roteador para testes de configuração: a indisponibilidade de um roteador complicou bastante os testes da aplicação, pois foi necessário a obtenção da senha de administrador para que se proceda a configuração deste dispositivo. Além disto, era necessário interromper a operação do roteador durante o teste de configuração, sendo assim necessário prévia autorização.

4.3 Trabalhos Relacionados

Os trabalhos relacionados abaixo citados fazem referência a aplicações ou sistemas que também implementam clientes Telnet, mas que não apresentam funcionalidades específicas no auxílio da configuração de dispositivos de rede. Um outro sistema citado é exclusivamente usado para a configuração de roteadores, porém este não utiliza o protocolo Telnet e é destinado a computadores pessoais.

4.3.1 MidpSSH

MidpSSH [46] é um cliente SSH e Telnet para dispositivos MIDP 1.0/2.0 como telefones celulares com suporte a Java e outros dispositivos móveis. O MidpSSH foi desenvolvido por Karl von Randow e possui licença do tipo GPL (*General Public License*).

O MidpSSH foi baseado nas aplicações Floyd SSH e Telnet Floyd, que foram desenvolvidas por Radek Polak, que portou o Applet Java Telnet/SSH para a plataforma MIDP. MidpSSH apenas adicionou uma interface gráfica e algumas novas funcionalidades.

Entre as principais funcionalidades implementadas pelo MidpSSH, podemos citar:

1. *Download* de uma lista de sessões para o telefone via HTTP.
2. Cores padrões suportadas corretamente.
3. Oferecer a opção de utilizar SSH 1 ou SSH 2.
4. Suporte a MIDP 1.0 ou 2.0, dependendo do dispositivo.
5. Versão disponível especialmente para Blackberry.
6. Suporte ao uso da opção "*Terminal Type*".
7. Suporte ao uso da opção "Eco".
8. Suporte ao uso da opção "*Window Size*".

Apesar do TelnetME não dar suporte a SSH ou ao MIDP 2.0, muitas das outras funcionalidades implementadas pelo MidpSSH também estão disponíveis no TelnetME. Entretanto, o TelnetME possui várias vantagens. Por dar suporte a todas estas funcionalidades o tamanho do arquivo "Jar" do MidpSSH é bem maior que o do TelnetME, que, dependendo do tipo do arquivo escolhido , pode ser até 12 vezes maior (10,1 Kb do TelnetME e 130Kb do MidpSSH).

O tamanho da aplicação pode ser fundamental na determinação dos dispositivos que suportarão a aplicação, e, com isto, pode-se facilmente afirmar que o TelnetME poderá ser suportado por mais aparelhos quando for levado em conta espaço disponível para a alocação de aplicações Java ME.

O suporte do TelnetME ao "gerenciamento de hosts", que são informações dos servidores armazenados, e histórico de comandos são algumas das funcionalidades que adicionam muitas facilidades durante o processo de estabelecimento de comunicação e corrente configuração de um roteador ou simplesmente durante uma sessão com um servidor. Estas funcionalidades não estão disponíveis no MidpSSH, fazendo com que o estabelecimento de comunicação seja mais demorado, mesmo para servidores já conhecidos, e haja a necessidade de re-digitar os comandos durante a troca de informações na comunicação.

4.3.2 Mocha Telnet

O Mocha Telnet [47] é um cliente Telnet, desenvolvido pela empresa MochaSoft, e que está disponível para diversos dispositivos como alguns celulares Nokia e Sony Ericsson, PDAs, estações de trabalho com Windows, Linux e Mac OS, entre outros.

O Mocha Telnet está disponível para celulares 3650/6600/9210/9500 da Nokia. Independente do modelo, a aplicação disponibiliza as mesmas potencialidades, que são:

1. Suporte a emulação de terminal do tipo vt220;
2. Suporte a SSH1 e SSH2;
3. Tela de exibição de 24*80 *pixels*;
4. Teclas de funções definidas pelo usuário.

Apesar de ambas as aplicações, Mocha Telnet e TelnetME, poderem ser utilizadas para a configuração de roteadores, a mais importante diferença ao se comparar o Mocha Telnet com o TelnetME é que o primeiro foi implementado utilizando a linguagem C ao invés da linguagem Java ME. Por ser implementada nesta linguagem, o Mocha Telnet não possui um arquivo Jar e sim um arquivo no formato "SIS", necessário para a transferência e instalação da aplicação no dispositivo. Este arquivo SIS normalmente possui tamanho muito superior ao arquivo Jar, chegando a uma diferença de aproximadamente 600%, comparando-se o TelnetME (10,1 Kb) e o Mocha Telnet (68,9 Kb).

Esta diferença é importante quando se leva em consideração a memória disponível em alguns dispositivos móveis. Uma outra diferença entre as duas aplicações é que o Mocha Telnet é voltado para dispositivos coloridos, com no mínimo 256 cores disponíveis. Este tipo de abordagem restringe a variedade de dispositivos que podem utilizar a aplicação, visto que a maioria dos dispositivos mais antigos não disponibilizava tantas cores.

Já o TelnetME foi projetado para utilizar apenas duas cores, preto e branco, por serem obrigatoriamente suportadas por dispositivos com MIDP, podendo assim atingir uma maior gama de aparelhos e plataformas.

Diferentemente do TelnetME, o Mocha Telnet não provê suporte a opção "*Window Size*", fornecendo apenas uma tela de 24*80 *pixels*, desperdiçando recursos de dispositivos mais avançados e com telas maiores. Por prover suporte a negociação desta opção, o TelnetME pode oferecer "tela cheia" (*full screen*) para todos os dispositivos.

4.3.3 Cisco Router Web Setup (CRWS)

O CRWS [42] é executado em computadores pessoais com sistema operacional Windows e é o meio mais recomendado para a configuração de roteadores Cisco. Uma Interface de Linha de Comando (CLI) é disponibilizada para que haja a configuração dos roteadores. Para se comunicar com o CRWS, o computador precisa estar na mesma sub-rede que o roteador e deve ser configurado através do protocolo DHCP (*Dynamic Host Configuration Protocol*). Este aplicativo pode ser utilizado para configurar muitas das famílias mais utilizadas de roteadores Cisco disponíveis, tais como:

1. Cisco 806 *router*;
2. Cisco 82x *routers*;
3. Cisco 83x *routers*;
4. Cisco Soho 7x *routers*;
5. Cisco Soho 9x *routers*.

Mesmo apresentando os mesmos objetivos, a configuração de roteadores, o CRWS e o TelnetME são muito diferentes na forma na qual realizam esta operação, aonde o CRWS já está instalado no roteador acessando-o diretamente, enquanto o TelnetME utiliza o protocolo Telnet para a configuração.

4.4 Trabalhos futuros

O TelnetME atualmente conta com todas as funcionalidades básicas que um cliente Telnet necessita para estabelecer sessões com hosts, sejam eles servidores ou roteadores, entretanto

ainda peca em aspectos como segurança e na utilização em abordagens mais complexas e específicas.

Com o intuito de prover uma maior generalização quanto ao uso desta aplicação, as possíveis futuras versões do sistema devem suportar novas funcionalidades, tais como:

Suporte a cores. Esta funcionalidade garantiria uma melhor visualização do texto na tela do dispositivo. O uso desta nova funcionalidade seria restrito a dispositivos móveis com suporte a pelo menos 256 cores ou até 65 mil cores dependendo do aparelho.

Emulação de terminais do tipo vt100 e vt320. Esta funcionalidade garantiria uma maior diversificação quanto ao uso do TelnetME, visto que outras abordagens como conexões com servidores de e-mail, conexões seriais diretas e uma grande variedade de hosts da plataforma UNIX.

Autenticação segura via SSH1 ou SSH2. Visa garantir segurança na utilização de sessões Telnet em casos onde um nível mínimo de segurança nos dados transferidos é requerido. O SSH é um protocolo para autenticação remota segura e outros serviços contextualizados em redes abertas.

Utilização da plataforma MIDP 2.0. A plataforma MIDP 2.0 disponibiliza muitas outras funcionalidades que as disponíveis na primeira versão. Entretanto, atualmente, é muito restrito o número de aparelhos com suporte a esta plataforma.

Acesso a lista de servidores. Esta funcionalidade acarretaria numa facilidade para os usuários do TelnetME que poderiam ter sempre a disposição uma lista com seus principais hosts continuamente disponível, podendo ser baixado para o dispositivo em caso de necessidade.

Armazenamento de comandos padrões de configuração de roteadores. Visa garantir economia de tempo e maior versatilidade na comunicação entre o dispositivo e o roteador. Como as principais funções de configuração já estariam disponíveis, muito tempo seria poupado, evitando perdas de conexão devido a longos tempos de espera de dados.

Configuração de outros dispositivos. Esta seria o maior e mais significativo trabalho futuro, pois permitiria a extensão da aplicação e das atuais diretrizes visando atender outros dispositivos de rede.

Bibliografia

[1] A FARREL, A. *The Internet and Its Principles: A Comparative Approach*. Morgan Kaufmann, 2003.
[2] TANENBAUM, A. S. *Redes de Computadores.* Campus, 2003.
[3] PETERSON, L. e DAVIE, B. *Computer Networks: A System Approach*. Morgan Kaufmann, 1999.
[4] McQUERRY, S. *Interconectando Cisco Network Devices*. Alta Books, 2001.
[5] COMER, D. *Internetworking with TCP/IP: Principles, Protocols, and Arquitecture*. Prentice Hall, 2002.
[6] FILHO, A. *Domínio Linux: do Básico à Servidores*. Visual Books, 2002.
[7] FLICKENGER, R. *Building Wireless Community Networks, 2nd Edition*. O'Reilly, 2003.
[8] STALLINGS, W. *Data and Computer Communications, Seventh Edition*. Prentice Hall, 2003.
[9] KEOGH, J. *J2ME: The Complete Reference*. McGraw-Hill/Osborne, 2003.
[10] *Java 2 Micro Edition (J2ME) Technology for Creating Mobile Devices*. White Paper, Sun, 2000. Disponível em http://www.java.sun.com/.
[11] KNUDSEN, J. *Wireless Java Developing with J2ME, Second Edition*. Apress, 2003.
[12] ORTIZ, C. e GIGUERE, E. *Mobile Information Device Profile for Java 2 Micro Edition (J2ME): Professional Developer's Guide*. John Wiley & Sons, 2001.
[13] HAYKIN, S. *Modern Wireless Communication*. Prentice Hall, 2004.
[14] STALLINGS, W. *Data and Computer Communications*. Prentice Hall, 2004.
[15] HALONEN, T., ROMERO, J. e MELERO, J. *GSM, GPRS and EDGE Performance*. John Wiley & Sons, 2003.
[16] TENENBAUM, A. e AUGENSTEIN, M. e LANGSAM, Y. *Data Structures Using C and C++*. Prentice Hall, 1996.
[17] FLANAGAN, D. *Java in a Nutshell*. O'Reilly, 2005.
[18] SCHILDT, H. *Java: The Complete Reference*, J2SE. Osborn, 2005.
[19] CRUME, J. e WEAVER, J. *Beginning J2EE*. Apress, 2005.
[20] CHEN, Z. *Java Card Technology for Smart Cards*. Addison Wesley, 2000.
[21] HEMPHILL, D. e WHITE, J. *J2ME Java in Small Things*. Manning, 2002.
[22] CRIMGER, R., LASSALE, P., PARIHAR, M., *et al. TCP/IP: a Bíblia*. Campus, 2002.
[23] http://jcp.org/en/home/index. Visitado pela ultima vez em 12/11/2005.
[24] http://jcp.org/introduction/overview/. Visitado pela ultima vez em 12/11/2005.
[25] MUCHOW, J. W. *Core J2ME Tecnologia e MIDP*. Pearson, 2003.
[26] GORDON, R. *Essencial Java: Java Native Interface*. Prentice Hall, 1998.
[27] HEMPHILL, D. e WHITE, J. *J2ME Java in Small Things*. Manning, 2002.
[28] TOPLEY, K. *J2ME in a Nutshell*. O'Reilly, 2002.
[29] PITT, E. *JAVA.RMI*. Addison Wesley, 2001.
[30] TREMBLETT, P. *Mobile Instant Wireless Java with J2ME*. Osborne/McGraw-Hill, 2002.

[31] YUAN, M. *Enterprise J2ME: Developing Mobile Java Applications*, Prentice Hall, 2003.
[32] http://java.sun.com/products/sjwtoolkit/download-2_2.html. Visitado pela ultima vez em 12/11/2005.
[33] AREHART, C. *Professional WAP*. Makron, 2000.
[34] MABE, D. *Blackberry Hacks*. O'Reilly, 2005.
[35] CRIMGER, R., LASSALE, P., PARIHAR, M., *et al. TCP/IP: a Bíblia*. Campus, 2002.
[36] HALL, E. *Internet Application Protocols*. O'Reilly, 2001.
[37] AKIN, T. *Hardening Cisco Routers*. O'Reilly, 2003.
[38] LARMOUTH, J. *Understanding OSI*. Thomson Learning, 1996.
[39] HUNT, C. *TCP/IP Network Administration*. O'Reilly, 2002.
[40] STALLINGS, W. *Computer Networking with Internet Protocols*. Pearson, 2003.
[41] BAMFORD, C. e CURRAN, P. J. *Data Structures, Files and Databases*. MacMillan, 1993.
[42] TRIPOD, M. *Cisco Router Configuration & Troubleshooting*. Cisco Press, 2000.
[43] LEINWAND, A. *Cisco Router Configuration*. Cisco Press, 2000.
[44] DWIVED, H. *Implementing SSH*. John Wiley Consumer, 2003.
[45] MANN, S. e SBIHLI, S. The *Wireless Application protocol (WAP)*. John Wiley Consumer, 2000.
[46] http://www.xk72.com/midpssh/. Visitado pela ultima vez em 12/11/2005.
[47] http://www.mochasoft.dk/ . Visitado pela ultima vez em 12/11/2005.

Apêndice A

Códigos-Fonte

Classe principal da aplicação – TelnetME

```
package com.br.telnetME;

import java.util.Vector;
import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordStore;

public final class TelnetME extends MIDlet
{
    public static ControlaCanvas canvas;
    public static Display display;
    protected static Vector vetorIndex;
    protected static Vector vetorString;

    public TelnetME()
    {
        display = Display.getDisplay(this);
        canvas = new ControlaCanvas(this, display, armazena(), false);
    }

    public void startApp()  {  }
    public void pauseApp()  {  }
    public void destroyApp(boolean flag)
    {
```

```
      ControlaFluxo fluxo = ControlaCanvas.contFluxo;
        if(fluxo != null)
            fluxo.fechaConexao();
        notifyDestroyed();
    }

    public void adicionaRecordStore(String dados[])
    {
        try
        {
         RecordStore recordstore = RecordStore.openRecordStore("hosts", true);
         vetorIndex.addElement(new Integer(recordstore.getNextRecordID()));
         vetorString.addElement(dados);
         String s = "";
         for(int i = 0; i < 5; i++)
             s = s + dados[i] + '\n';

         byte abyte0[] = s.getBytes();
         recordstore.addRecord(abyte0, 0, abyte0.length);
         recordstore.closeRecordStore();
        }
        catch(Exception exception) { }
    }

    public void removeRecordStore(int index)//retira um host do recordstore
pelo index
    {
      vetorString.removeElementAt(index);
        int j = ((Integer)vetorIndex.elementAt(index)).intValue();
        try
        {
            RecordStore recordstore = RecordStore.openRecordStore("hosts",
false);
            recordstore.deleteRecord(j);
            recordstore.closeRecordStore();
        }
        catch(Exception exception) { }
        vetorIndex.removeElementAt(index);
    }
```

```
    private Vector armazena()
    {
      vetorString = new Vector();
      vetorIndex = new Vector();
        try
        {
            RecordStore recordstore = RecordStore.openRecordStore("hosts",
false);
            RecordEnumeration recordenumeration =
recordstore.enumerateRecords(null, null, false);
            RecordEnumeration recordenumeration1 =
recordstore.enumerateRecords(null, null, false);
            String elementos[];
            for(; recordenumeration.hasNextElement();
 vetorString.addElement(elementos))
            {
                  vetorIndex.addElement(new
Integer(recordenumeration1.nextRecordId()));
                  elementos = new String[5];
                String s = new String(recordenumeration.nextRecord());
                int i = 0;
                for(int j = 0; j < 5; j++)
                {
                  elementos[j] = s.substring(i, i = s.indexOf('\n', i));
                     i++;
                }
            }

            recordenumeration.destroy();
            recordenumeration1.destroy();
            recordstore.closeRecordStore();

        }
        catch(Exception exception) { }
        return vetorString;
    }

}
```

Classe de controle de tela – ControlaCanvas

```
package com.br.telnetME;

import java.util.Vector;
import javax.microedition.lcdui.*;

//responsável pelo controle de tela
public class ControlaCanvas extends Canvas
    implements CommandListener, Runnable
{
    public static TelnetME telnet;
    public static ControlaFluxo contFluxo;


    public static Display display;
    protected static List lsMenu;
    protected static TextBox tbTexto;
    protected static List lsEntrada;
    protected static List lsCaractere;
    protected static List lsHistorico;
    protected static List lsSimbolo;

    protected static TextBox tbCodAscii;
    protected static List lsControleCaract;

    protected static Form fmVazio;  //form com os dados vazios (null)
    protected static Form fmCompleto;     //form com os dados definidos

    protected static TextField tfNome;
    protected static TextField tfHost;
    protected static TextField tfPorta;
    protected static TextField tfUsuario;
    protected static TextField tfSenha;

    protected static Form fmSalvarHost;   //form de salvar host
    protected static Form fmSobrescHost;  //form de sobrescrever host
    protected static Form fmRemoveHost;   //form de remover host da lista de
hosts lsHosts
```

```
    protected static int indexRemovRecStore;

    protected static TextBox tbNome;
    protected static List lsHosts;
    protected static List lsOpcoes;


    protected static Font fonte;
    protected static boolean houveMudancaTexto;

    protected static int larguraTotal;
    protected static int alturaTotal;
    protected static Command cmOk;
    protected static Command cmCancel;
    protected static Command cmSim;
    protected static Command cmNao;

    protected static int indexLinha;      //index da linha
    protected static String linhaDoTexto[];    //conjunto de linhas escritas
    protected static int linhaAtual;      //linha atual (do texto), onde está o
marcador
    protected static int numeroLinhasTotal;    //numero de linhas das 2 telas
    protected static int larguraTexto;
    protected static int alturaFonte;

    protected static boolean flag1; //
    protected static boolean flag2; //

    protected static int proximaAlturaTexto;   //altura onde vai ser impressa
o proximo caractere na linha ou na outra linha
    protected static int proximaLarguraTexto;  //largura onde vai ser impresso
o proximo caractere na linha

    protected static int alturaRealFonte;
    protected static int larguraFonte;

    protected int key;
    protected long tempoPressInic;  //tempo q a tecla fica pressionada (faz
rolar a barra)
    protected static boolean teclaPressionada;
    protected static int numeroLinhasTela;     //numero de linhas de 1 tela
    protected static int numeroColunas;
```

```
    public static Vector vtHostsRecStore; //vetor q contem os hosts já
armazenados no RecordStore
    public static String tipoDados[];

    protected static final String caracControle1[] = {
        "Space", "TAB", "DEL", "ESC", "CTRL-C", "CTRL-Z", "Backspace",
"Return"
    };
    protected static final char caracControle2[] = {
        ' ', '\t', '\177', '\033', '\003', '\032', '\b', '\r'
    };
    protected static final String caracSimbolos1[] = {
        "/ slash", "\\ backslash", "& amperstand", "| pipe", "< lower", ">
greater", "* asterisk", "_ underscore", "~ tilde", "$ dollar",
        "' quote", "\" double quote", "# pound"
    };
    protected static final char caracSimbolos2[] = {
        '/', '\\', '&', '|', '<', '>', '*', '_', '~', '$',
        '\'', '"', '#'
    };

    protected static boolean existeHostVetor;  //existe host armazenado no
vetor de hosts

    protected static Vector vtHistorico;


    protected static boolean conexaoON;
    protected static boolean estaConectado;
    protected static boolean tentandoConectar;

    public ControlaCanvas (TelnetME client, Display disp, Vector vector,
boolean bool)
    {
        telnet = client;
        display = disp;
        vtHostsRecStore = vector;

        cmOk = new Command("OK", 4, 1);
        cmCancel = new Command("Cancelar", 3, 1);
        cmSim = new Command("Sim", 4, 1);
        cmNao = new Command("Não", 3, 1);
```

```
lsCaractere = new List("Caractere", 3);
lsCaractere.append("Caracteres de Controle", null);
lsCaractere.append("Símbolos", null);
lsCaractere.append("Código ASCII", null);
lsCaractere.addCommand(cmCancel);
lsCaractere.setCommandListener(this);

lsEntrada = new List("Entrada", 3);
lsEntrada.append("Texto", null);
lsEntrada.append("Caracteres", null);
lsEntrada.append("Histórico", null);
lsEntrada.addCommand(cmCancel);
lsEntrada.setCommandListener(this);

lsMenu = new List("Menu", 3);
lsMenu.append("Entrada de texto", null);
lsMenu.append("Conectar", null);
lsMenu.append("Comandos chave", null);
lsMenu.append("Informações", null);
lsMenu.append("Sair", null);
lsMenu.addCommand(cmCancel);
lsMenu.setCommandListener(this);

lsControleCaract = new List("Caracteres de Controle", 3);

for(int aux1 = 0; aux1 < caracControle1.length; aux1++)
    lsControleCaract.append(caracControle1[aux1], null);

lsControleCaract.addCommand(cmCancel);
lsControleCaract.setCommandListener(this);

lsSimbolo = new List("Símbolo", 3);
for(int aux2 = 0; aux2 < caracSimbolos1.length; aux2++)
    lsSimbolo.append(caracSimbolos1[aux2], null);

lsSimbolo.addCommand(cmCancel);
lsSimbolo.setCommandListener(this);
lsHistorico = new List("Histórico", 3);

lsHistorico.addCommand(cmCancel);
lsHistorico.setCommandListener(this);
```

```
        tbTexto = new TextBox("Texto", null, 128, 0);
        tbTexto.addCommand(cmOk);
        tbTexto.addCommand(cmCancel);
        tbTexto.setCommandListener(this);

        tbCodAscii = new TextBox("Código ASCII", null, 3, 2);
        tbCodAscii.addCommand(cmOk);
        tbCodAscii.addCommand(cmCancel);
        tbCodAscii.setCommandListener(this);

        vtHistorico = new Vector(10);

        alturaTotal = getHeight();
        larguraTotal = getWidth();
        fonte = Font.getFont(32, 0, 8);
        alturaFonte = fonte.getHeight(); //distancia entre as linhas da base
        numeroLinhasTela = alturaTotal / alturaFonte;//numero de linhas na
tela
        numeroLinhasTotal = 2 * numeroLinhasTela;    //talvez seja o tamanho
da tela virtual (2 telas reais cheias)
        linhaDoTexto = new String[numeroLinhasTotal];//lista de string
        larguraFonte = fonte.charWidth('W');
        numeroColunas = larguraTotal / larguraFonte; //numero de caracteres na
linha
        alturaRealFonte = fonte.getBaselinePosition();     //distancia entre a
linha de base e o topo do caractere

        zeraParamTexto();

        imprimeString("TelnetME v1.0");
        imprimeString("\n\nPressione " + getKeyName(getKeyCode(8)) +
(hasPointerEvents() ? " or tap screen" : "") + " para exibir o menu.");
        display.setCurrent(this);
    }

    public void imprimeString(String str)
    {
        for(int aux = 0; aux < str.length(); aux++)
            imprimeChar(str.charAt(aux));

    }
```

```
    public void verificaConexao()
    {
        if(conexaoON)
        {
            return;
        } else
        {
            estaConectado = false;
            contFluxo = null;
            imprimeString("\nConexão perdida.");
            repaint();
            lsMenu.set(1, "Conectar", null);
            exibeAlerta(null, "Conexão perdida ", ((Displayable) (this)));
            return;
        }
    }

    public void imprimeChar(char caract)  //imprime um caractere
    {
        if(caract == '\b') //espaço
        {
            if(linhaDoTexto[indexLinha].equals(""))  //se a string for vazia
                  indexLinha--;

            String stringInteira = linhaDoTexto[indexLinha];
            linhaDoTexto[indexLinha] = stringInteira.substring(0,
stringInteira.length() - 1);  //tira as duas ultimas silabas

        } else
        if(larguraTexto + fonte.charWidth(caract) > larguraTotal || caract ==
'\n' || caract == '\r') //nova linha ou enter
            if(indexLinha == numeroLinhasTotal - 1)        //penultima linha
da tela
            {
                for(int i1 = 0; i1 < indexLinha; i1++)
                  linhaDoTexto[i1] = linhaDoTexto[i1 + 1];

                linhaDoTexto[indexLinha] = "";
            } else
```

```
            {
                  indexLinha++;
            }
        if(caract != '\n' && caract != '\b')    //diferente de nova linha e
espaço
            linhaDoTexto[indexLinha] += caract;
        atualizaParamTexto();
        houveMudancaTexto = true;
    }



    private void zeraParamTexto()   //zera os parametros do texto
    {
      linhaAtual = indexLinha = numeroLinhasTela;
        for(int index = 0; index < numeroLinhasTotal; index++)
            linhaDoTexto[index] = "";

        proximaLarguraTexto = 0;
        proximaAlturaTexto = 0;
        larguraTexto = 0;
    }

    private void atualizaParamTexto()
    {
      larguraTexto = fonte.stringWidth(linhaDoTexto[indexLinha]);
        if(indexLinha >= linhaAtual && indexLinha < linhaAtual +
numeroLinhasTela)
        {
            flag2 = true;
            proximaLarguraTexto = larguraTexto;
            proximaAlturaTexto = (indexLinha - linhaAtual) * alturaFonte;
      //muda de linha

            if(larguraTexto + larguraFonte > larguraTotal) //fim da linha
            {
                  proximaLarguraTexto = 0;      //
                  proximaAlturaTexto += alturaFonte;
            }
        } else
        {
```

```
            flag1 = false;
            flag2 = false;
        }
    }

    public void paint(Graphics graphic)
    {
      graphic.setColor(255, 255, 255);    //seta p preto
        graphic.fillRect(0, 0, larguraTotal, alturaTotal); //pinta a tela de
branco
        graphic.setFont(fonte);     //seta a fonte
        graphic.setColor(0, 0, 0);  //seta a cor p preto

        int altura = 0; // altura da fonte no texto

        for(int j = linhaAtual; j < linhaAtual + numeroLinhasTela; j++)
        {
            graphic.drawString(linhaDoTexto[j], 0, altura, 20);
            altura += alturaFonte;
        }

        if(flag1 && estaConectado)  //quadradinho piscando
            graphic.fillRect(proximaLarguraTexto, proximaAlturaTexto,
larguraFonte, alturaRealFonte);
    }

    public void run()
    {
      tentandoConectar = true;
      zeraParamTexto();
        imprimeString("Conectando a " + tipoDados[1] + "...");
        repaint();
        try
        {
            contFluxo = new ControlaFluxo(this, tipoDados);
        }
        catch(Exception exception)
        {
            imprimeString("\nConexão falhou.");
            repaint();
```

```
            exibeAlerta(null, "Impossível de conectar ao host " + tipoDados[1]
+ " na porta " + tipoDados[2] + ".", ((Displayable) (existeHostVetor ?
((Displayable) (lsHosts)) : ((Displayable) fmVazio))));
            tentandoConectar = false;
            return;
        }
        tentandoConectar = false;
        fmVazio = null;
        tfHost = null;
        tfPorta = null;
        tfUsuario = null;
        tfSenha = null;
        tipoDados = null;
        estaConectado = true;
        zeraParamTexto();
        repaint();
        lsMenu.set(1, "Close", null);

        int count = 0;
        while(estaConectado)
        {
            if(count++ % 4 == 0)    //testa mudança de estado (evento)a cada 4
rodadas
            {
                if(!flag1 && flag2)
                  flag1 = true;
                else
                  flag1 = false;
                repaint(proximaLarguraTexto, proximaAlturaTexto, larguraFonte,
alturaRealFonte);
            }
            if(houveMudancaTexto)
            {
                repaint();
                houveMudancaTexto = false;
            }
            if(tempoPressInic != 0L)
            {
                int valor = getGameAction(key);
                if(valor == 1 || valor == 6)
                {
                    long tempoPressFim = System.currentTimeMillis();
```

```
                    if(tempoPressFim - tempoPressInic > 1000L ||
teclaPressionada)
                    {
                        keyPressed(key);  //pressionar botão
                        teclaPressionada = true;
                    }
                }
            }
            try
            {
                Thread.sleep(250L);
            }
            catch(InterruptedException interruptedexception) { }
        }
    }



    private Form exibeForm(String s1)
    {
        Form fmExibeString = new Form(null);
        fmExibeString.append(s1);
        fmExibeString.addCommand(cmSim);
        fmExibeString.addCommand(cmNao);
        fmExibeString.setCommandListener(this);
        return fmExibeString;
    }

    private Form criaFormDados(String dadosForm[])//cria formulário com os
dados
    {
        Form fmDados = new Form(dadosForm != null ? "Editar" : "Conectar a");
        if(dadosForm != null)
        {
            tfNome = new TextField("Nome", dadosForm[0], 32, 0);
            fmDados.append(tfNome);
        }
        tfHost = new TextField("Host", dadosForm == null ? null :
dadosForm[1], 32, 0);
        fmDados.append(tfHost);
        tfPorta = new TextField("Porta", dadosForm == null ? "23" :
dadosForm[2], 5, 4);
```

```
        fmDados.append(tfPorta);
        tfUsuario = new TextField("Usuário (opcional)", dadosForm == null ?
null : dadosForm[3], 32, 0);
        fmDados.append(tfUsuario);
        tfSenha = new TextField("Senha (opcional)", dadosForm == null ? null :
dadosForm[4], 32, 0x10000);
        fmDados.append(tfSenha);
        fmDados.addCommand(cmOk);
        fmDados.addCommand(cmCancel);
        fmDados.setCommandListener(this);
        return fmDados;
    }


    public void exibeAlerta(String s1, String s2, Displayable displayable)
    {
        Alert alert = new Alert(s1, s2, null, AlertType.INFO);
        alert.setTimeout(-2);
        display.setCurrent(alert, displayable);
    }

    private void atualizaHistorico(String s1, String s2)
    {
        int indexVetorHistorico;
        if((indexVetorHistorico = vtHistorico.indexOf(s1)) != -1) //se houver
s1 no historico
        {
            vtHistorico.removeElementAt(indexVetorHistorico);
            lsHistorico.delete(indexVetorHistorico);
        } else
        if(vtHistorico.size() == 10)
        {
            vtHistorico.removeElementAt(9);
            lsHistorico.delete(9);
        }
        vtHistorico.insertElementAt(s1, 0);
        lsHistorico.insert(0, s2, null);
    }


    private void iniciaThread()
```

```
{
    display.setCurrent(this);
    (new Thread(this)).start();
}


private String[] fazerDados(String nomeHost)
{
    String str1[] = new String[5];
    str1[0] = nomeHost;
    str1[1] = tfHost.getString();
    str1[2] = tfPorta.getString();
    str1[3] = tfUsuario.getString();
    str1[4] = tfSenha.getString();
    return str1;
}


public void commandAction(Command command, Displayable displayable)
{
    Object obj = null;
    if(command == cmCancel)
        obj = this;
    else
    if(displayable == lsMenu)
    {
        int indexMenu = lsMenu.getSelectedIndex();
        switch(indexMenu)
        {
        case 0: // '\0'
            if(estaConectado)
                obj = lsEntrada;
            else
              exibeAlerta(null, "Não conectado", this);
            break;

        case 1: // '\001'
            if(tentandoConectar)
                obj = this;
            else
            if(estaConectado)
```

```
                {
                  estaConectado = false;
                    contFluxo.fechaConexao();
                } else
                if(vtHostsRecStore.size() > 0)
                {
                  lsHosts = new List("Hosts", 3);
                  lsHosts.append("<novo>", null);
                    for(int l2 = 0; l2 < vtHostsRecStore.size(); l2++)

      lsHosts.append(((String[])vtHostsRecStore.elementAt(l2))[0], null);

                    lsHosts.addCommand(cmCancel);
                    lsHosts.setCommandListener(this);
                    obj = lsHosts;
                } else
                {
                    obj = fmVazio = criaFormDados(((String []) (null)));
      //cria form vazio
                }
                break;

            case 2: // '\002'
                  //exibeAlerta("Comandos chave ", "[" +
getKeyName(getKeyCode(8)) + "] Menu\n[" + getKeyName(getKeyCode(1)) + "/" +
getKeyName(getKeyCode(6)) + "] Movimentar Barra de Rolagem\n[" +
getKeyName(getKeyCode(5)) + "] Retornar\n[" + getKeyName(getKeyCode(2)) + "]
Espaço em Branco\n[" + getKeyName(49) + "] Inserir Texto\n[" + getKeyName(50)
+ "] Inserir Caractere\n[" + getKeyName(51) + "] Histórico", this);
                  exibeAlerta("Comandos chave ", "[" +
getKeyName(getKeyCode(8)) + "] Menu\n[" + getKeyName(getKeyCode(1)) + "/" +
getKeyName(getKeyCode(6)) + "] Movimentar Barra de Rolagem\n[" +
getKeyName(getKeyCode(2)) + "] Retornar\n[" + getKeyName(getKeyCode(5)) + "]
Espaço em Branco\n[" + getKeyName(49) + "] Inserir Texto\n[" + getKeyName(50)
+ "] Inserir Caractere\n[" + getKeyName(51) + "] Histórico", this);
                break;

            case 3: // '\003'
                  zeraParamTexto();
                imprimeString("TelnetME \n" +
                        "Aplicação desenvolvida como Trabalho de Conclusão de
Curso da Escola Politécnica de Pernambuco em 2005.2 \n \n" +
                        "Renato Augusto G. P. França");
```

```
                obj = this;
                repaint();
                break;

            case 4: // '\004'
                  conexaoON = false;
                telnet.destroyApp(true);
                break;
            }
        } else
        if(displayable == lsHosts)
        {
            int indexHosts2 = lsHosts.getSelectedIndex();
            if(indexHosts2 == 0)
            {
                  fmVazio = criaFormDados(((String []) (null))); //cria form
vazio
                obj = fmVazio;
            } else
            {
                  lsOpcoes = new
List(((String[])vtHostsRecStore.elementAt(indexHosts2 - 1))[0], 3);
                  lsOpcoes.append("Abrir", null);
                lsOpcoes.append("Editar", null);
                lsOpcoes.append("Remover", null);
                lsOpcoes.addCommand(cmCancel);
                lsOpcoes.setCommandListener(this);
                obj = lsOpcoes;
            }
        } else
        if(displayable == lsOpcoes)
        {
            int indexOpcoes = lsOpcoes.getSelectedIndex();
            if(indexOpcoes == 0)
            {
                  existeHostVetor = true;
                tipoDados =
(String[])vtHostsRecStore.elementAt(lsHosts.getSelectedIndex() - 1);
                iniciaThread();
            } else
            if(indexOpcoes == 1)
```

```
            {
                  fmCompleto =
criaFormDados((String[])vtHostsRecStore.elementAt(lsHosts.getSelectedIndex() -
1));
                obj = fmCompleto;
            } else
            if(indexOpcoes == 2)
            {
                  fmRemoveHost = exibeForm("Remover " +
((String[])vtHostsRecStore.elementAt(lsHosts.getSelectedIndex() - 1))[0] +
"?");
                obj = fmRemoveHost;
            }
            lsOpcoes = null;
        } else
        if(displayable == fmCompleto)
        {
            int indexHosts = lsHosts.getSelectedIndex();
            String dados[] = fazerDados(tfNome.getString());
            telnet.removeRecordStore(indexHosts - 1);
            telnet.adicionaRecordStore(dados);
            lsHosts.delete(indexHosts);
            lsHosts.append(dados[0], null);
            obj = lsHosts;
        } else
        if(displayable == fmVazio)
        {
            existeHostVetor = false;
            fmSalvarHost = exibeForm("Salvar host?");
            obj = fmSalvarHost;
        } else
        if(displayable == fmSalvarHost)
        {
            if(command == cmSim)
            {
                  tbNome = new TextBox("Nome", tfHost.getString(), 32, 0);
                  tbNome.addCommand(cmOk);
                tbNome.setCommandListener(this);
                obj = tbNome;
            } else
            {
                  tipoDados = fazerDados(null);
```

```
                iniciaThread();
            }
            fmSalvarHost = null;
        } else
        if(displayable == fmRemoveHost)
        {
            if(command == cmSim)
            {
                int indexHosts3 = lsHosts.getSelectedIndex();
                telnet.removeRecordStore(indexHosts3 - 1);
                lsHosts.delete(indexHosts3);
            }
            fmRemoveHost = null;
            obj = lsHosts;
        } else
        if(displayable == fmSobrescHost)
        {
            if(command == cmSim)
            {
                  tipoDados = fazerDados(tbNome.getString());
                telnet.removeRecordStore(indexRemovRecStore);
                telnet.adicionaRecordStore(tipoDados);
                iniciaThread();
            } else
            {
                obj = tbNome;
            }
            fmSobrescHost = null;
        } else
        if(displayable == tbNome)
        {
            lsHosts = null;
            tipoDados = fazerDados(tbNome.getString());
            boolean flag = false;

            for(int indexRS = 0; indexRS < vtHostsRecStore.size(); indexRS++)
            {

if(!((String[])vtHostsRecStore.elementAt(indexRS))[0].equals(tbNome.getString(
)))
                    continue;
```

```
                flag = true;
                indexRemovRecStore = indexRS;
                break;
            }
            if(flag)
            {
                obj = fmSobrescHost = exibeForm("Sobrescrever '" +
tbNome.getString() + "' ?");
            } else
            {
                telnet.adicionaRecordStore(tipoDados);
                iniciaThread(); }
        } else
        if(displayable == lsEntrada)
        {
            int indexEntrada = lsEntrada.getSelectedIndex();
            switch(indexEntrada)
            {
            case 0: // '\0'
                obj = tbTexto;
                break;

            case 1: // '\001'
                obj = lsCaractere;
                break;

            case 2: // '\002'
                obj = lsHistorico;
                break;
            }} else
        if(displayable == lsCaractere)
        {
            int indexCaractere = lsCaractere.getSelectedIndex();
            switch(indexCaractere)
            {
            case 0: // '\0'
                obj = lsControleCaract;
                break;

            case 1: // '\001'
                obj = lsSimbolo;
```

```
                break;

            case 2: // '\002'
                obj = tbCodAscii;
                break;
            }
        } else
        if(displayable == tbTexto || displayable == lsControleCaract ||
displayable == lsSimbolo || displayable == lsHistorico || displayable ==
tbCodAscii)
        {
            String s1 = null;
            String s2 = null;
            if(displayable == tbTexto)
            {
                s1 = s2 = tbTexto.getString();
                tbTexto.setString(null);
            } else
            if(displayable == lsControleCaract)
            {
                int indexControlChar = lsControleCaract.getSelectedIndex();
                s1 = "" + caracControle2[indexControlChar];
                s2 = caracControle1[indexControlChar];
            } else
            if(displayable == lsSimbolo)
            {
                int indexSimbolo = lsSimbolo.getSelectedIndex();
                s1 = "" + caracSimbolos2[indexSimbolo];
                s2 = caracSimbolos1[indexSimbolo];
            } else
            if(displayable == lsHistorico)
            {
                int indexHistorico = lsHistorico.getSelectedIndex();
                s1 = (String)vtHistorico.elementAt(indexHistorico);
                s2 = lsHistorico.getString(indexHistorico);
            } else
            {
                try
                {
                    int i4 = Integer.parseInt(tbCodAscii.getString());
                    if(i4 > 0 && i4 < 256)
```

```
                {
                    s1 = "" + (char)i4;
                    s2 = "char #" + i4;
                }
            }
            catch(NumberFormatException numberformatexception) { }
        }
        if(s1 != null)
        {
              contFluxo.imprimeFluxo(s1);
            atualizaHistorico(s1, s2);
        }
        obj = this;
    }
    if(obj != null)
        display.setCurrent(((Displayable) (obj)));
}

public void keyPressed(int valorKey)
{
  key = valorKey;
    if(tempoPressInic == 0L)
        tempoPressInic = System.currentTimeMillis();
    int tecla = getGameAction(valorKey);
    Object obj = null;
    if(tecla == 8)  //select
        obj = lsMenu;
    else
    if(tecla == 1 && linhaAtual > 0)
    {
        linhaAtual--;
        atualizaParamTexto();
        repaint();
    } else
    if(tecla == 6 && linhaAtual < numeroLinhasTotal - numeroLinhasTela)
    {
        linhaAtual++;
        atualizaParamTexto();
        repaint();
    } else
    if(estaConectado)
```

```
            if(tecla == 5)
                  contFluxo.imprimeFluxo("\n");
            else
            if(tecla == 2)
                  contFluxo.imprimeFluxo("\b");
            else
                switch(valorKey)
                {
                case 49: // '1'
                    obj = tbTexto;
                    break;

                case 50: // '2'
                    obj = lsCaractere;
                    break;

                case 51: // '3'
                    obj = lsHistorico;
                    break;
                }
        if(obj != null)
            display.setCurrent(((Displayable) (obj)));
    }

    public void pointerPressed(int i1, int j1)
    {
        display.setCurrent(lsMenu);
    }

    public void keyReleased(int i1) //tecla é solta
    {
      tempoPressInic = 0L;
        teclaPressionada = false;
    }

}
```

Classe de controle de fluxo de dados – ControlaFluxo

```
package com.br.telnetME;
```

```
import java.io.*;
import javax.microedition.io.*;

// implementa funcionalidades como: conexoes e fluxo
public class ControlaFluxo
    implements Runnable
{

    public static ControlaCanvas canvas;
    public static StreamConnection connection;

    public static OutputStream oStrm;
    public static InputStream iStrm;

    protected static boolean flag;
    protected boolean entra;
    protected static boolean existeFluxo; // flag p mostrar qd exste fluxo
    protected static boolean mark;

    protected static String usuario;
    protected static String senha;

    public ControlaFluxo(ControlaCanvas eCanvas, String dados[])
        throws IOException
    {
        entra = true;
        canvas = eCanvas;
        connection = (StreamConnection)Connector.open("socket://" + dados[1] +
":" + dados[2], 3, false);
        iStrm = connection.openInputStream();
        oStrm = connection.openOutputStream();
        usuario = dados[3];
        senha = dados[4];
        if(usuario.equals(""))
        {
            usuario = null;
            senha = null;
        } else
        if(senha.equals(""))
            senha = null;
        existeFluxo = true;
```

```
        (new Thread(this)).start();
    }


    //envia para imprimir na tela os dados de chegada e enviado ao servidor ou
digitado
    // ou por enviar os dados p o servidor e imprimir na tela ...

    public void run()  //inicia troca de dados e analisa se os dados podem ser
impressos
    {
        try
        {
            oStrm.write(255);
            oStrm.write(253);
            oStrm.write(1);               //ECHO
            while(existeFluxo)
            {

                int entradaInicial = iStrm.read();   //ler proximo byte
(valor) da entrada
                if(entradaInicial == -1)  // fim da entrada
                {
                  fechaConexao();
                    return;
                }

                //===========

                if(entradaInicial == 255)
                {
                    int segundaEntrada = iStrm.read();
                    if(segundaEntrada >= 250 && segundaEntrada < 255)
                    {
                        int terceiraEntrada = iStrm.read();
                        if(segundaEntrada == 253)    //comando "DO"
                        {
                              oStrm.write(255);

                            if(terceiraEntrada == 24)//"WILL 24"
                            {
```

```
                    oStrm.write(251);
                    oStrm.write(terceiraEntrada);
                } else
                if(terceiraEntrada == 31)//WINDOW_SIZE
                {
                    oStrm.write(251);
                    oStrm.write(terceiraEntrada);
                     oStrm.write(255);
                     oStrm.write(250);
                     oStrm.write(terceiraEntrada);
                     oStrm.write(0);
//TRANSMIT_BINARY
                     oStrm.write(ControlaCanvas.numeroColunas);
                     oStrm.write(0);
                     oStrm.write(ControlaCanvas.numeroLinhasTotal);
                     oStrm.write(255);
                     oStrm.write(240);
                } else
                {
                    oStrm.write(252);       //comando "WONT"
                    oStrm.write(terceiraEntrada);
                }
            } else
            if(segundaEntrada == 250)           //comando "SB"
            {
                if(terceiraEntrada == 24)//TERMINAL-TYPE
                {
                    iStrm.read();
                    iStrm.read();
                    iStrm.read();
                       oStrm.write(255);
                       oStrm.write(250);
                       oStrm.write(terceiraEntrada);
                       oStrm.write(0);
                       oStrm.write("ansi".getBytes());
                       oStrm.write(255);
                       oStrm.write(240);          //comando "SE"
                }
            } else
            if(segundaEntrada == 251)           //comando "WILL"
            {
```

```
                        oStrm.write(255);
                    if(terceiraEntrada == 1)
                    {
                     oStrm.write(253);
                        entra = false;
                    } else
                    {
                     oStrm.write(254);       //comando "DONT"
                    }
                    oStrm.write(terceiraEntrada);
                }
            }
            continue;
        }

        //==========

        if(entradaInicial == 27)  //Output Marking
            flag = true;
        else
        if(flag)
        {
            flag = false;
            if(entradaInicial == 91)
                mark = true;
        } else
        if(mark && (entradaInicial >= 65 && entradaInicial <= 90 ||
entradaInicial >= 97 && entradaInicial <= 122))
        {
            mark = false;
            continue;
        }

        if(entradaInicial != 0 && entradaInicial != 7 &&
entradaInicial != 13 && entradaInicial != 27 && !mark)
        {
            if(usuario != null && entradaInicial == 58)
            {
                oStrm.write((usuario + '\n').getBytes());
                usuario = null;
            } else
```

```
                if(senha != null && entradaInicial == 58)
                {
                    oStrm.write((senha + '\n').getBytes());
                    senha = null;
                } else
                if(entradaInicial == 9)     //Output Page Size
                    entradaInicial = 32;    //Terminal Speed
                canvas.imprimeChar((char)entradaInicial);
            }
        }
    }
    catch(IOException ioexception)
    {
        fechaConexao();
    }
}

public void fechaConexao()
{
    if(!existeFluxo)
        return;
    try
    {
        existeFluxo = false;
        iStrm.close();
        oStrm.close();
        connection.close();
    }
    catch(IOException ioexception) { }

    canvas.verificaConexao();
}

public void imprimeFluxo(String str)
{
    try
    {
        oStrm.write(str.getBytes());
        if(entra)
            canvas.imprimeString(str);
    }
```

```
        catch(IOException ioexception)
        {
            fechaConexao();
        }
    }
}
```

Apêndice B

Especificações dos Casos de Uso

Caso de Uso – Sair Sistema

1. Breve descrição

Caso de uso referente à saída do sistema em um momento determinado ou qualquer momento da execução da aplicação.

2. Fluxo de eventos

Fluxo básico

- O usuário inicia a execução da MIDlet TelnetME.
- Sistema exibe a tela de entrada do sistema.
- O usuário pressiona uma tecla específica para entrar no sistema.
- Sistema exibe o Menu principal da aplicação.
- O usuário seleciona a opção “Sair” do menu.
- O sistema finaliza.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos descritos no fluxo básico, o usuário pode decidir por finalizar a aplicação pressionado a tecla de desligar do dispositivo.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

5. Pós-condições

Aplicação finalizada

O sistema será finalizado.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Exibir Informações do Sistema

1. Breve descrição

Caso de uso referente à exibição de informações sobre o sistema, tais como nome e motivação da aplicação e nome do desenvolvedor.

2. Fluxo de eventos

Fluxo básico

- O usuário inicia a execução da MIDlet TelnetME.
- Sistema exibe a tela de entrada do sistema.
- O usuário pressiona uma tecla específica para entrar no sistema.
- Sistema exibe o Menu principal da aplicação.
- O usuário seleciona a opção “Informações” do menu.
- Sistema exibe a Tela de Informações.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada
O usuário deve ter iniciado a execução da aplicação.

5. Pós-condições

Tela exibida ou cancelamento
A Tela de informações do sistema é exibida ou a operação foi cancelada.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Ver Comandos chave

1. Breve descrição

Caso de uso referente à exibição de comandos chave a serem usados durante a utilização da aplicação.

2. Fluxo de eventos

Fluxo básico

- O usuário inicia a execução da MIDlet TelnetME.
- Sistema exibe a tela de entrada do sistema.
- O usuário pressiona uma tecla específica para entrar no sistema.
- Sistema exibe o Menu principal da aplicação.
- O usuário seleciona a opção “Comandos chave” do menu.
- Sistema exibe a Tela de Comandos chave.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

5. Pós-condições

Tela exibido ou cancelamento

O Tela de comandos chave é exibida ou a operação foi cancelada.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Salvar host

1. Breve descrição

Caso de uso referente aos passos envolvidos para salvar um host na memória persistente do dispositivo.

2. Fluxo de eventos

Fluxo básico

- O usuário inicia a execução da MIDlet TelnetME.
- Sistema exibe a tela de entrada do sistema.
- O usuário pressiona uma tecla específica para entrar no sistema.
- Sistema exibe o Menu principal da aplicação.
- O usuário seleciona a opção “Conectar host” do menu.
- Sistema exibe o Formulário de conexão.
- O usuário digita as informações de configuração requeridas, deixando a opção “Porta” como padrão igual a 23.
- O usuário seleciona OK.
- Sistema exibe Tela de Salvar host.
- O usuário seleciona a opção “Sim”.
- Sistema exibe Tela de nome de host.
- O usuário digita o nome do host a ser salvo.
- O usuário pressiona “OK”
- Sistema exibe Tela Conectando.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Não Salvar host

No passo 10 (dez), do fluxo básico, o usuário pode escolher a opção “Não” e decidir por não salvar as configurações do host em questão.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

Configuração de host

O usuário deve ter os parâmetros de configuração necessários para o estabelecimento da conexão com o host, tais como numero IP do mesmo e os opcionais, usuário e senha.

5. Pós-condições

Host Salvo

Pelo menos um host foi salvo na memória do dispositivo.

Operação cancelada

A operação foi cancelada durante o processo de salvamento do host.

Substituir host

Ao se tentar salvar um host com o mesmo nome de um já existente, será dada a opção de sobrescrever host.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Editar host

1. Breve descrição

Caso de uso referente aos passos envolvidos para editar os parâmetros de configuração de um host que já esteja salvo na memória persistente do dispositivo.

2. Fluxo de eventos

Fluxo básico

- O usuário inicia a execução da MIDlet TelnetME.
- Sistema exibe a tela de entrada do sistema.
- O usuário pressiona uma tecla específica para entrar no sistema.
- Sistema exibe o Menu principal da aplicação.
- O usuário seleciona a opção “Conectar host” do menu.
- Sistema exibe uma Tela com a lista dos hosts já salvos na memória.
- O usuário seleciona o host a ser editado.
- Sistema exibe Tela de escolha de operações.
- O usuário seleciona a opção “Editar”.
- Sistema exibe o Formulário com os parâmetros do host.
- O usuário faz as alterações.
- O usuário pressiona “OK”
- Sistema exibe a Tela com a lista dos hosts já atualizada.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada
O usuário deve ter iniciado a execução da aplicação.

Host Salvo
Deve haver ao menos um host salvo na memória do dispositivo.

5. Pós-condições

Host Atualizado
Os parâmetros atualizados do host editado já estão salvos na memória do dispositivo.

Operação cancelada
A operação foi cancelada durante o processo de edição do host.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Conectar host

1. Breve descrição

Caso de uso que demonstra os passos envolvidos no estabelecimento de uma conexão com o host.

2. Fluxo de eventos

Fluxo básico

- Include Salvar host.
- O usuário seleciona um host.
- Sistema exibe Tela de escolha de operações.
- O usuário seleciona “Abrir”.
- Sistema exibe Tela conectando.
- Sistema abre uma conexão com o roteador.
- Roteador confirma o estabelecimento da conexão.
- Sistema exibe a Tela de Conectado.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Erro de conexão

No passo 6, por algum motivo não esperado, a conexão pode não ser estabelecida. Da mesma forma, ela pode ser perdida em algum momento após ser confirmada.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

Host Salvo

Deve haver ao menos um host salvo na memória do dispositivo.

5. Pós-condições

Conexão estabelecida
A conexão com o host de destino está estabelecida.

Operação cancelada
A operação foi cancelada durante a conexão com o host.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Exibir histórico

1. Breve descrição

Caso de uso que ilustra os passos necessários para a utilização do histórico de comandos.

2. Fluxo de eventos

Fluxo básico

- Include Conectar host.
- O usuário pressiona a tecla “Select” ou “Menu” dependendo do dispositivo.
- Sistema exibe Tela de Menu.
- O usuário seleciona “Entrada de texto”.
- Sistema exibe Tela de Entrada de texto.
- O usuário seleciona “Histórico”.
- Sistema abre uma tela com a lista de Histórico de comandos.
- O usuário escolhe o comando.
- Sistema exibe a Tela de Texto com o conteúdo do item do histórico escolhido.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Erro de conexão

No passo 2, por algum motivo não esperado, a conexão pode ser perdida em algum momento após ser confirmada.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

Host conectado

A aplicação deve ter estabelecido conexão com o host.

5. Pós-condições

Histórico exibido

O conteúdo do item do histórico escolhido é exibido na Tela de texto, podendo ser então utilizada para um novo comando.

Operação cancelada

A operação foi cancelada durante o processo de exibição de histórico.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Inserir caractere

1. Breve descrição

Caso de uso que ilustra os passos necessários para a inserção de caracteres especiais no texto.

2. Fluxo de eventos

Fluxo básico

- Include Conectar host.
- O usuário pressiona a tecla “Select” ou “Menu” dependendo do dispositivo.
- Sistema exibe Tela de Menu.
- O usuário seleciona “Entrada de texto”.
- Sistema exibe Tela de Entrada de texto.
- O usuário seleciona “Caracteres”.
- Sistema abre uma tela com a lista de tipos de caracteres.
- O usuário escolhe o tipo “Caracteres de controle”.
- O usuário escolhe o caractere de controle que quer adicionar.
- Sistema exibe a Tela de Texto com o caractere escolhido já inserido.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Erro de conexão

No passo 2, por algum motivo não esperado, a conexão pode ser perdida em algum momento após ser confirmada.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada
O usuário deve ter iniciado a execução da aplicação.

Host conectado
A aplicação deve ter estabelecido conexão com o host.

5. Pós-condições

Caractere inserido
O caractere escolhido é exibido na Tela de texto, podendo ser então utilizada por um novo comando.

Operação cancelada
A operação foi cancelada durante a inserção do caractere.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Inserir símbolo

1. Breve descrição

Caso de uso que ilustra os passos necessários para a inserção de um símbolo no texto.

2. Fluxo de eventos

Fluxo básico

- Include Conectar host.
- O usuário pressiona a tecla “Select” ou “Menu” dependendo do dispositivo.
- Sistema exibe Tela de Menu.
- O usuário seleciona “Entrada de texto”.
- Sistema exibe Tela de Entrada de texto.

- O usuário seleciona “Caracteres”.
- Sistema abre uma tela com a lista de tipos de caracteres.
- O usuário escolhe o tipo “Símbolos”.
- O usuário escolhe o símbolo.
- Sistema exibe a Tela de Texto com o símbolo escolhido já inserido.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Erro de conexão

No passo 2, por algum motivo não esperado, a conexão pode ser perdida em algum momento após ser confirmada.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

Host conectado

A aplicação deve ter estabelecido conexão com o host.

5. Pós-condições

Símbolo inserido

O símbolo escolhido é inserido na Tela de texto, podendo ser então utilizada por um novo comando.

Operação cancelada

A operação foi cancelada durante a inserção do símbolo.

6. Pontos de Extensão

Não se aplica.

Caso de Uso – Inserir código ASCII

1. Breve descrição

Caso de uso que ilustra os passos necessários para a inserção de um código ASCII no texto.

2. Fluxo de eventos

Fluxo básico

- Include Conectar host.
- O usuário pressiona a tecla “Select” ou “Menu” dependendo do dispositivo.
- Sistema exibe Tela de Menu.
- O usuário seleciona “Entrada de texto”.
- Sistema exibe Tela de Entrada de texto.
- O usuário seleciona “Caracteres”.
- Sistema abre uma tela com a lista de tipos de caracteres.
- O usuário escolhe o tipo “Código ASCII”.
- O usuário digita o caractere do qual deseja obter o código ASCII.
- Sistema exibe a Tela de Texto com o código ASCII caractere escolhido já inserido.

Fluxos alternativos

Cancelamento

Em qualquer um dos passos citados no fluxo básico, o usuário pode decidir por finalizar a aplicação.

Erro de conexão

No passo 2, por algum motivo não esperado, a conexão pode ser perdida em algum momento após ser confirmada.

3. Requisitos Especiais

Não se aplica.

4. Pré-condições

Aplicação iniciada

O usuário deve ter iniciado a execução da aplicação.

Host conectado

A aplicação deve ter estabelecido conexão com o host.

5. Pós-condições

Código ASCII inserido

O código ASCII do caractere escolhido é exibido na Tela de texto, podendo ser então utilizada por um novo comando.

Operação cancelada

A operação foi cancelada durante a inserção do código ASCII.

6. Pontos de Extensão

Não se aplica.