CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
Faculdade de Tecnologia Ciências Sociais Aplicadas - FATECS
Curso de Engenharia da Computação

Hugo Fernandes Vilar de Almeida

PROPOSTA DE AUTOMAÇÃO RESIDENCIAL UTILIZANDO CELULAR COM TECNOLOGIA BLUETOOTH

Brasília
2008

Hugo Fernandes Vilar de Almeida

PROPOSTA DE AUTOMAÇÃO RESIDENCIAL UTILIZANDO CELULAR COM TECNOLOGIA BLUETOOTH

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Graduação em Engenharia da Computação.

Orientadora: Maria Marony Sousa Farias Nascimento

Brasília
2008

Hugo Fernandes Vilar de Almeida

CELULAR COMO CONTROLE UNIVERSAL

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB/ICPD) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Graduação em Engenharia da Computação.

Orientador: Maria Marony Sousa Farias Nascimento

Brasília, ___ de _____ de ____.

Banca Examinadora

_____

Prof. Dr. Cláudio Penedo de Albuquerque

_____

Prof. Dr. Francisco Javier De Obaldía Díaz

Agradecimentos

Agradeço aos meus pais, Erique Vilar de Almeida e Maria de Lourdes Fernandes Tavares de Almeida, pela paciência, carinho, compreensão e força que me deram não só durante a confecção do projeto final, mas também durante toda minha vida.

Agradeço também à minha namorada Érika de Figueiredo Morelo, que esteve do meu lado o tempo todo, me dando confiança quando era necessário e me ajudando no que lhe era possível.

À minha professora orientadora Maria Marony que esteve disponível sempre que precisei, fornecendo conselhos e críticas construtivas.

Resumo

Neste projeto, é demonstrada uma aplicação de automação residencial utilizando Bluetooth. Uma interface com o usuário é estabelecida fazendo com que seja selecionada a opção desejada por este. A opção é enviada a um computador que recebe o dado, e então o envia pela porta serial. Um microcontrolador recebe o dado enviado, e dependendo do dado recebido, executará a ação desejada pelo usuário. Esta ação pode ser ligar ou desligar um aparelho, ligar ou desligar uma lâmpada, etc. Neste projeto, o dispositivo a ser acionado é representado através de um conjunto de LEDs.

Palavras chave: Bluetooth, Java, Microcontrolador, J2ME, Automação residencial.

Abstract

Inside this work, is demonstrated a residential automation application using Bluetooth. An interface with the user is established making him to select the option desired. The option is sent to a computer that receives the data, and then sends it through the serial port. A microcontroller receives the data, and depending on the data received, it will execute the user desired action. This action can be turn on or turn off a device, turn on or turn off a lamp, etc. Inside this work, the device to be turned on or off is represented by a set of LEDs.

Keywords: Bluetooth, Java, Microcontroller, J2ME, Residential automation.

Sumário

Lista de Figuras

Lista de símbolos

API – Application Program Interface (Interface de Programação de Aplicativos)

CI – Circuito integrado

CPU – Central Processing Unit (Unidade Central de Processamento)

DCE – Data Communication Equipament (Equipamento de Comunicação de Dados)

DPTR – Data Pointer (Ponteiro de Dados)

DTE – Data Terminal Equipament (Equipamento Terminal de Dados)

GND – Ground (terra elétrico)

J2EE – Java 2 Enterprise Edition (Edição para Empresas)

J2ME – Java 2 Micro Edition (Edição Micro)

J2SE – Java 2 Standard Edition (Edição Padrão)

JVM – Java Virtual Machine (Máquina Virtual Java)

KVM – KyloByte Virtual Machine (Máquina Virtual KyloByte)

LED – Light Emitting Diode (Diodo Emissor de Luz)

PAN – Personal Area Network (Rede de Área Pessoal)

PDA – Personal Digital Assintant (Assistente Pessoal Digital)

RAM – Random Access Memory (Memória de Acesso Aleatório)

Capítulo 1 - Introdução

A qualidade de vida é um dos requisitos mais procurados pela população. Muitas vezes, essa qualidade de vida está intimamente ligada a algum desenvolvimento eletrônico. A proposta do projeto é aumentar a comodidade das pessoas, com o desenvolvimento de uma aplicação para simbolizar o ligar e desligar de lâmpadas a partir de um simples comando do celular.

Este projeto envolve o desenvolvimento de 3 softwares: um para o celular, um para o computador e outro para o microcontrolador. O primeiro estabelece uma interface entre o usuário e os outros dois softwares. O segundo recebe o comando enviado pelo celular, por meio da tecnologia Bluetooth e transmitirá estes dados pela porta serial. O Terceiro recebe os dados da porta serial e executa a ação de acordo com o dado recebido.

Um conjunto de LEDs ligados ao microcontrolador representa os dispositivos a serem ligados e desligados por meio de comandos enviados pelo celular.

1.1 Objetivo

O objetivo do projeto é aproveitar este novo ramo da tecnologia que envolve desenvolvimento para dispositivos móveis, utilizando a linguagem Java que é acessível, pois existem várias ferramentas que são software livre, ou seja, que pode ser usado, copiado, estudado ou modificado sem restrições. Outra vantagem grande da linguagem Java é a sua grande literatura disponível na internet e por meio

dos livros. Por meio do site de procura Google.com, é possível achar muitos artigos, comunidades e informações em geral sobre a linguagem.

Com ferramentas disponíveis para desenvolvimento, é possível então a criação de software, considerando a topologia ideal, para que as residências sejam cada vez mais inteligentes e que os equipamentos eletrônicos que nela se encontram sejam manipulados com maior conforto. A disposição do trabalho em termos de organização de capítulos será a seguinte:

- Capítulo 1 – Uma introdução contendo uma abordagem sobre o trabalho, motivação e objetivo.
- Capítulo 2 – Uma descrição do trabalho com a identificação de fatores e parâmetros associados e proposta de solução
- Capítulo 3 – Apresentação das bases teóricas para que seja possível o andamento do projeto.
- Capítulo 4 – Proposta de solução e topologia.
- Capítulo 5 – Aplicação da solução e apresentação de resultados. Aqui é mostrado o que foi feito de acordo com a proposta para a conclusão do projeto.
- Capítulo 6 – Conclusão sobre pontos fortes e pontos fracos, assim como limitações do projeto e sugestões para projetos futuros.

1.2 Apresentação do Problema

A tecnologia, hoje em dia, vem sendo desdobrada para o aumento de qualidade de vida, redução dos mais variados riscos e conforto. As pessoas que moram em grandes centros urbanos sentem, no seu dia-a-dia, diversas formas de tecnologia que vieram para melhorar suas vidas.

Um dos ramos de desenvolvimento de tecnologia voltado para atender necessidades mais básicas das pessoas no seu lar é chamado automação residencial. Com ela podemos preparar cafés apenas com o apertar de um botão, podemos lavar louças com muito mais facilidade, controlar temperatura de geladeiras, ligar ou desligar equipamentos elétricos ou eletrônicos de onde estivermos e etc. (LESSA, 2002)

Partindo da suposição de que o celular, pelo que o mesmo representa nos dias de hoje, e levando em consideração o grande número de celulares por pessoas (WWW.ANATEL.GOV.BR, 2008), e também o grande e crescente número de computadores existentes nas casas hoje em dia (FOLHA ON LINE, 2008), pode-se configurar um sistema que possa acionar os dispositivos de casa (por exemplo, lâmpadas) com o envio do comando pelo celular utilizando a tecnologia sem fio Bluetooth para o computador.

O computador por sua vez envia o dado recebido para um microcontrolador que faz um tratamento lógico da situação e executa as ações demandadas.

Segundo dados publicados na Folha On Line, se compararmos o primeiro trimestre do ano (2008) e o primeiro trimestre de 2007, houve um acréscimo de

18,7% na compra de computadores. Fato esse, que encoraja a automação residencial, visto que os computadores são os componentes mais caros da implementação. Levando em consideração que as pessoas têm computadores e celulares, falta a compra do microcontrolador, cabo serial e dispositivos Bluetooth caso necessário no último componente.

O objetivo é fazer com que o usuário tenha controle sobre equipamentos eletrônicos por meio do celular. Não seria mais necessário se deslocar ao cômodo da casa em que o equipamento eletrônico se encontra para poder controlá-lo. No caso, o projeto apenas controla LEDs presentes no kit microcontrolador. Os LEDs representam os equipamentos. Sendo assim, o microcontrolador liga ou desliga os supostos aparelhos.

Capítulo 2 – Referencial Teórico

2.1 Kit microcontrolador

Um kit de microcontrolador é uma peça onde se encontra o microcontrolador já com aparelhos conectados às suas portas, como na Figura 2-1 podemos perceber que no kit estão presentes 8 LEDs, uma porta serial pronta para receber o cabo, teclado e um display.

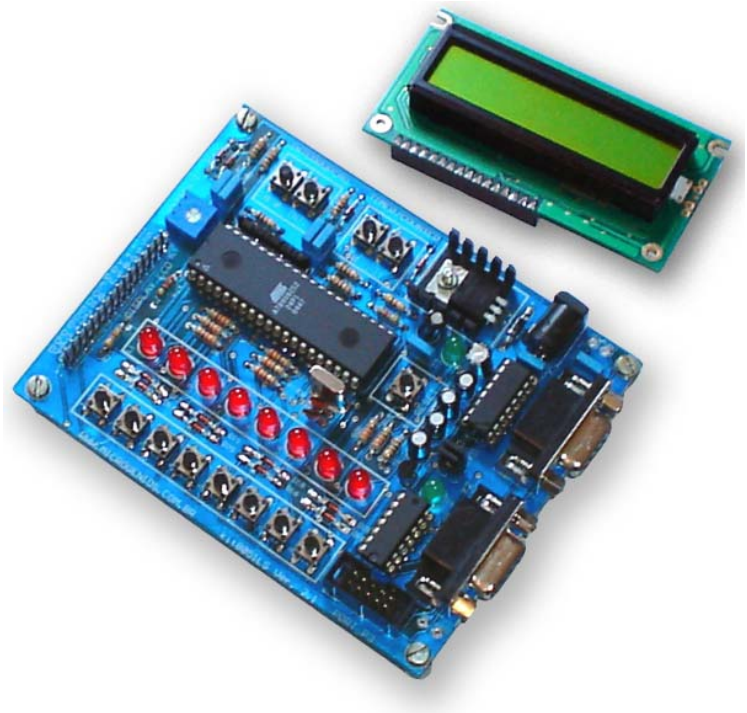

Figura 2-1 - Exemplo de kit de microcontrolador

2.1.1 Microcontrolador

Microcontroladores são dispositivos semicondutores em forma de CI (Circuito Integrado). São limitados nos quesitos memória e processamento. Não são aplicados em situações em que se necessite de grandes quantidades de memória. Usa-se microcontroladores para casos específicos principalmente, e são exemplos: automação residencial (por exemplo, luzes, microondas), automação industrial (por exemplo, robótica), automação embarcada (por exemplo, computadores de bordo) e automação predial (por exemplo, elevadores). (GIMENEZ, 2002)

Algumas definições são importantes para a utilização dos microcontroladores. São informações básicas que devem ser conhecidas para a compreensão. Elas são as seguintes:

Bits: é um valor lógico, um digito binário. Pode valer 0 ou 1.

Byte: é um agrupamento de 8 bits. Pode variar de 00h a FFh, ou também de 0_{10} a 255_{10}.

Registradores: são responsáveis pelo armazenamento de informações. Apresentam a característica de serem voláteis, ou seja, eles perdem as informações quando são desligados.

Instruções: são comandos guardados em memória que indicam o que deve ser feito.

Unidade Central de Processamento (CPU): é o cérebro do microcontrolador. Ele realiza a busca e execução de um programa. Fisicamente, ele é um semicondutor constituído por milhões de transistores.

Interrupções: é a ação do microcontrolador de parar a execução normal das instruções correntes em virtude de uma troca de dados com o sistema. (GIMENEZ, 2002)

2.1.2 Microcontrolador 8051

São características dos microcontroladores da família 8051:

- CPU (Unidade central de processamento) de 8 bits otimizada para aplicações de controle;
- Capacidade de processamento de operações lógicas;
- 64 Kbytes de endereçamento de memória de programa;
- 64 Kbytes de endereçamento de dados;
- 8Kbytes de memória de programa interna;
- 256 Bytes de memória RAM de dados interna;
- 32 linhas de entrada e saída bidirecionais e individualmente endereçáveis;
- Um canal serial full duplex. (ATMEL, 2005)

2.1.3 Organização da memória

No microcontrolador abordado as memórias de programa e de dados são logicamente separadas como mostrado na Figura 2-2. Essa separação permite que a memória de dados sejam acessados por meio de endereços de 8 bits e por endereçamento indireto de 16 bits, gerados pelo registrador DPTR (*Data Pointer*). Isso torna mais rápida a manipulação e armazenamento pela *Central Unit Processing* (CPU) de 8 bits. (CORRADI JUNIOR, 2006)

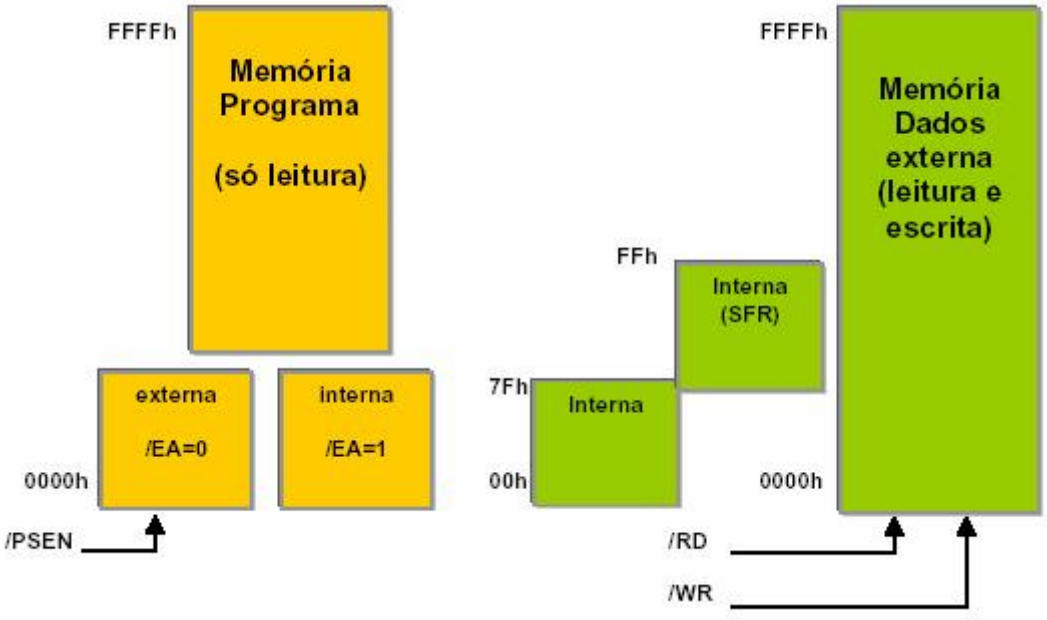

Figura 2-2 - Separação entre memórias do microcontrolador

2.1.4 Central Processing Unit (CPU)

É na CPU onde ficam os registradores, contadores e circuitos lógicos necessários. Ela é responsável pela procura de instruções para serem processadas, decodificação e execução das instruções. Nela, podemos basicamente verificar um ciclo de trabalho bem característico: primeiro a CPU busca por alguma instrução a ser executada na memória, depois decodifica a instrução que deverá ser executada e em seguida executa a tal instrução. (CORRADI JUNIOR, 2006)

2.1.5 LEDs do kit microcontrolador

Para a utilização dos LEDs é necessário saber como eles funcionam e como manipulá-los. Os LEDs do kit de microcontrolador são listados em uma das portas do kit. A porta utilizada é a P2. O kit utilizado possui 8 LEDs, estando listados da seguinte forma: P2.0 para o primeiro LED, até a P2.7 para o oitavo LED. Para

ascendê-los, basta igualar o respectivo LED a 0 (zero) e para desligá-los, basta igualá-lo a 1 (um).

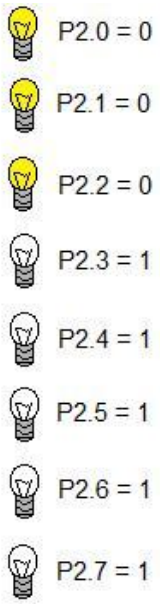

Figura 2-3 - Exemplo de configuração da porta P2

Na Figura 2-3, está contido um exemplo de configuração da porta em que os diodos LEDs estão ligados. No exemplo, os 3 primeiros LEDs foram igualados a 0 para que fossem acendidos e o restante foi igualado a 1 para que permanecesse desligado. (CORRADI JUNIOR, 2006)

2.1.6 Porta serial do kit microcontrolador

A porta serial que existe no microcontrolador 8051 é full duplex, ou seja, ela permite o envio e recepção simultâneos. Essa porta serial também possui um buffer onde os dados são armazenados para envio. Com a presença do buffer no microcontrolador é possível receber bits antes mesmo que os bits anteriores já tenham sido lidos. Com isso não há interrupção tanto na leitura quanto na recepção de dados. (CORRADI JUNIOR, 2006)

2.2 Comunicação serial

Comunicação em serie é o tipo de conexão em que o transmissor envia um bit de cada vez, e o receptor recebe um bit de cada vez. Contrariando o tipo de conexão paralela que envia e recebe mais de um bit de cada vez. Separando os bits de um em um, é percebida uma melhora com relação à interferência eletromagnética; problema muito freqüente nas conexões paralelas. O estabelecimento de uma conexão serial pode ser efetuado de duas formas: síncrono e assíncrono. O modo síncrono é executado com a ajuda de um sinal de relógio que sincroniza o transmissor e o receptor. Nas transmissões e recepções assíncronas, os dados trafegam numa velocidade constante sem sinal de relógio e obedecem a um formato específico. Para o estabelecimento de uma conexão serial assíncrona é preciso que alguns parâmetros sejam definidos. Os parâmetros serão explanados na próxima seção. (CORRADI JUNIOR, 2006)

2.2.1 Parâmetros de uma conexão serial

Velocidade de transmissão: é uma representação que define a velocidade que será utilizada no trafego de dados. É um número expresso em bits por segundo. O número pode ser representado por uma medida inglesa chamada *baudrate*. No caso da transferência de dados para o microcontrolador, não há diferença.

Start bit: é um único bit que serve para indicar que o emissor deseja abrir uma conexão.

Número de bits da palavra: nada mais é que o número de bits que constituirão uma palavra.

Bit de paridade: bit de paridade serve para identificação de erro na transmissão. É feita uma soma nos bits iguais a 1 no transmissor e depois verificada se a quantidade de bits 1 corresponde ao que foi enviado no receptor.

Stop bit: 1 ou 2 bits são reservados para que haja uma separação entre duas palavras enviadas consecutivamente.

2.2.2 Normas para conexões seriais

Foram criadas normas para as conexões seriais assim como para qualquer outro tipo de conexão. Seja qual for o tipo de conexão. Podemos citar algumas normas criadas como: RS232, RS 422 e V24. Mas, as mais aceitas universalmente e mais utilizadas normas são a RS232 e a V24. (CORRADI JUNIOR, 2006)

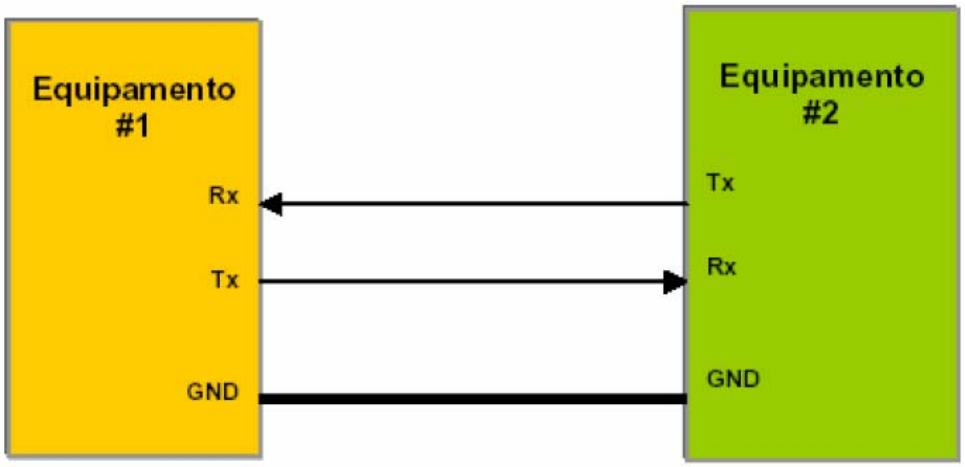

Figura 2-4 - Comunicação serial usando padrão RS232 (apenas 3 ligações)

2.2.3 Padrão RS 232

O padrão RS232 é reconhecido oficialmente como a interface entre um *Data Terminal Equipament* (DTE) e um *Data Communication Equipament* (DCE). O padrão tem um sinal Tx que é o de transmissão e um Rx que é o de recepção. O nível de diferencial binário é feito comparando com a tensão do sinal GND. É permitido um máximo de 2 aparelhos em uma conexão ponto a ponto com uma distância máxima de 50 metros.(LABRAMO, 2006) O cabeamento pode ser feito com uma grande quantidade de pinos assim como pode ser com poucos. Os cabos mais comuns são o de 25 pinos e o de 9 pinos. O de 25 pinos liga todos os pinos e possui todos os sinais possíveis. No padrão DB9 que utiliza 9 pinos, o cabo não faz algumas conexões incomuns. É uma versão mais enxuta da versão com 25 pinos. (OMEGA.COM, 2008)

2.3 Bluetooth

Bluetooth é um padrão de comunicação de redes PAN (*Personal Area Network*). É usado para conectar redes de curta distância e também para a substituição dos cabos. Uma de suas principais utilizações é para a interligação entre palmtops, celulares e periféricos de computador. (MILLER, 2001)

A tecnologia Bluetooth foi construída com o objetivo de consumir pouca energia. Ela é uma tecnologia que utiliza sinais de rádio freqüência para estabelecer conexões ponto-a-ponto e ponto-a-multiponto para transferência de dados e voz. Para que haja uma conexão entre os dispositivos é necessário que eles contenham rádios Bluetooth, cada um deles. Cada conexão pode ter até 8 (oito) dispositivos

diferentes, sendo que 1 deles será o aparelho central (*master*) e os outros serão subordinados (*slaves*). Ao se conectarem, os dispositivos estabelecem o que é chamado *piconet* conforme a figura 2-5; e, quando as *piconets* se conectam, formam por sua vez as redes de dispersão ou *scatternets* ilustradas pela figura 2-6. (MILLER, 2001)

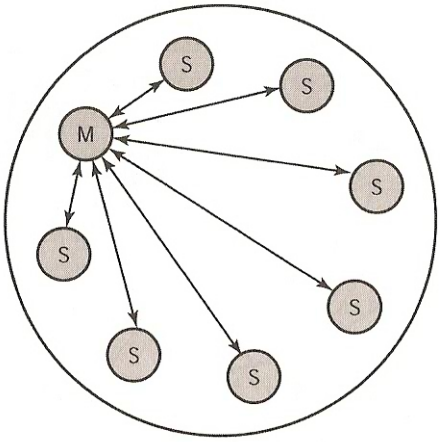

Figura 2-5 - Uma piconet

Uma vantagem que a tecnologia Bluetooth trouxe foi a redução na complexidade da interligação de dispositivos. Por exemplo: em uma ligação com fios, os conectores e cabos têm que ser os mesmos, senão a conexão não será possível no ambiente. Com o advento dessa nova tecnologia, as pessoas não devem ter mais preocupação sobre as especificações de fios existentes. (MILLER, 2001)

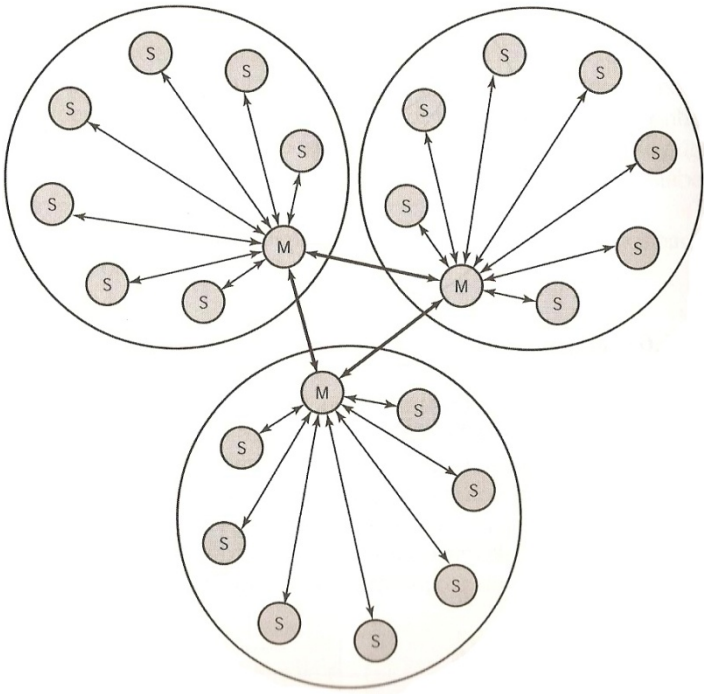

Figura 2-6 - Uma scatternet

2.3.1 Bluetooth: conexão por ondas de rádio

O Bluetooth utiliza ondas de rádio que nada mais são que pulsos de energia eletromagnética. As ondas de rádio são geradas por meio de oscilações provocadas pelo transmissor a uma freqüência específica. Para receber dados, é necessário um receptor que esteja ajustado para receber sinais de rádio a certa freqüência; se não houver essa combinação de freqüência entre transmissor e receptor, os sinais passarão pelo receptor sem que sejam percebidos. (MILLER, 2001)

A comunicação Bluetooth utiliza uma freqüência de rádio específica, assim como qualquer outro equipamento de rádio freqüência. A freqüência que o rádio Bluetooth usa é de 2,40 a 2,48 GHz. Esta faixa de freqüência, por não ser licenciada, é também utilizada por outros equipamentos de rádio. Para que o

Bluetooth não sofra muita interferência, ele usufrui de um mecanismo para evitar interferência e redução de taxa de transferência de dados. Este mecanismo faz com que a conexão seja dinâmica, ou seja, ela vai saltando aleatoriamente e constantemente de freqüência em freqüência, dentro do escopo determinado, para que a conexão não permaneça tempo bastante transmitindo na mesma freqüência para ser interferida. (MILLER, 2001)

A tecnologia permite conexões com até 100 metros de distância, porém, os dispositivos menores, como o celular, possuem normalmente uma antena com alcance de 10 metros. O padrão atual do Bluetooth pode alcançar taxas de conexão de 3Mbps, isso sem contar com o overhead que o protocolo necessita. (MORIMOTO, 2008)

2.4 Tecnologia Java

2.4.1 Visão geral

Eis uma breve descrição sobre a linguagem Java: ela é simples, robusta, orientada a objetos, distribuída, segura, neutra em relação à arquitetura, portável interpretada, possui alto desempenho, múltiplas linhas de execução e é dinâmica. Com todas essas características fica claro que a linguagem é uma excelente escolha para qualquer tipo de aplicação. A primeira versão foi lançada em 1995. O J2SE surgiu da idéia “write once, run anywhere™”, que significa “escreva uma vez, execute onde quiser” que permite a programação de aplicações usando o mesmo código não interessando o sistema operacional que esteja rodando o programa. Tal

proeza é conseguida como advento da Java Virtual Machine (JVM) que ajuda a traduzir a linguagem para o sistema operacional vigente. (MUCHOW, 2004)

A linguagem Java é dividida principalmente de acordo com suas aplicações fim. Elas são: J2ME, J2SE e J2EE. O J2ME que significa Java 2 *Micro Edition* é voltada para desenvolvimento de aplicações com recursos reduzidos. Esse tipo de programa será executado em dispositivos com processamento e memórias reduzidas. O J2SE que significa Java 2 *Standard Edition* é a base da plataforma e inclui o ambiente de execução e as bibliotecas comuns. Já o J2EE que significa Java 2 *Enterprise Edition* é voltado para o desenvolvimento de aplicações corporativas e para internet. (JOHNSON, 2008)

2.4.2 J2ME

A edição *Micro Edition* foi desenvolvida para equipamentos de performances reduzidas como PDAs (*Personal Digital Assistants*, Assistentes Digitais Pessoais), celulares e *pagers*. Das limitações envolvidas podemos citar: processamento e memórias reduzidas, menores telas com suas menores resoluções. Seria muito bom se pudéssemos utilizar a API (*Application Program Interface*, Interface de Programação de Aplicativos) J2SE para o desenvolvimento das aplicações desejadas, porém nem todas as funcionalidades podem ser executadas nesse tipo de dispositivo. (JOHNSON, 2008)

Para abstrair o modo como o J2ME funciona em seus diversos aparelhos finais, essa tecnologia foi separada em camadas. As camadas foram divididas como na Figura 2-7:

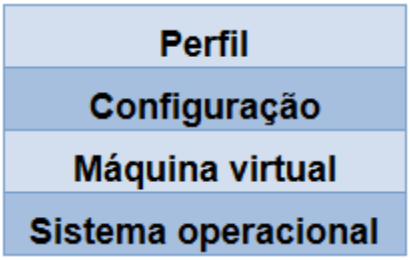

Figura 2-0-7 - Camadas da arquitetura J2ME

A primeira camada que o Java trata é a *KyloByte Virtual Machine* (KVM). Ela faz a tradução entre a arquitetura Java para a língua do sistema operacional residente. Visto que ele foi projetado para máquinas com pouca capacidade, ele deveria ser pequeno. Ele possui aproximadamente apenas 80KB. (JOHNSON, 2008)

A segunda camada diz respeito à adequação das APIs ao aparelho utilizado. De acordo com a categoria de aparelho que a pessoa possui, a camada de configuração disponibiliza certo tipo de APIs. Assim, uma categoria de dispositivos terá aplicações compatíveis uns com os outros. Temos atualmente 2 tipos principais de configurações: o *Connected Limited Device Configuration* (CLDC) e o *Connected Device Configuration* (CDC). O tipo que é usado para celulares é o CLDC. Os equipamentos compatíveis com o CLDC possuem as maiores restrições quanto a recursos de memória, processamento e capacidade gráfica. Uma importante limitação aos dispositivos compatíveis com o CLDC é a não execução de operações de ponto flutuante. O CDC é utilizado para aparelhos com uma disponibilidade de recursos um pouco maior. Esse tipo é aplicado aos PDAs por exemplo. (JOHNSON, 2008)

Na última camada da arquitetura J2ME, as APIs mais específicas são abordadas. Ela é chamada de camada dos perfis e funciona como uma extensão da

camada anterior, a de configuração. Os perfis especificam um pouco mais as aplicações e possuem funcionalidades de mais alto nível. O perfil mais utilizado para desenvolvimento para celulares é o *Mobile Information Device Profile* (MIDP). Esse é o único perfil indicado para dispositivos que utilizam o CLDC como configuração. (JOHNSON, 2008)

Capítulo 3 – Proposta de Solução e Modelo

A visão geral do projeto é mostrada na Figura 3-1. Nesta proposta, cada dispositivo (celular, computador e microcontrolador) tem um software associado.

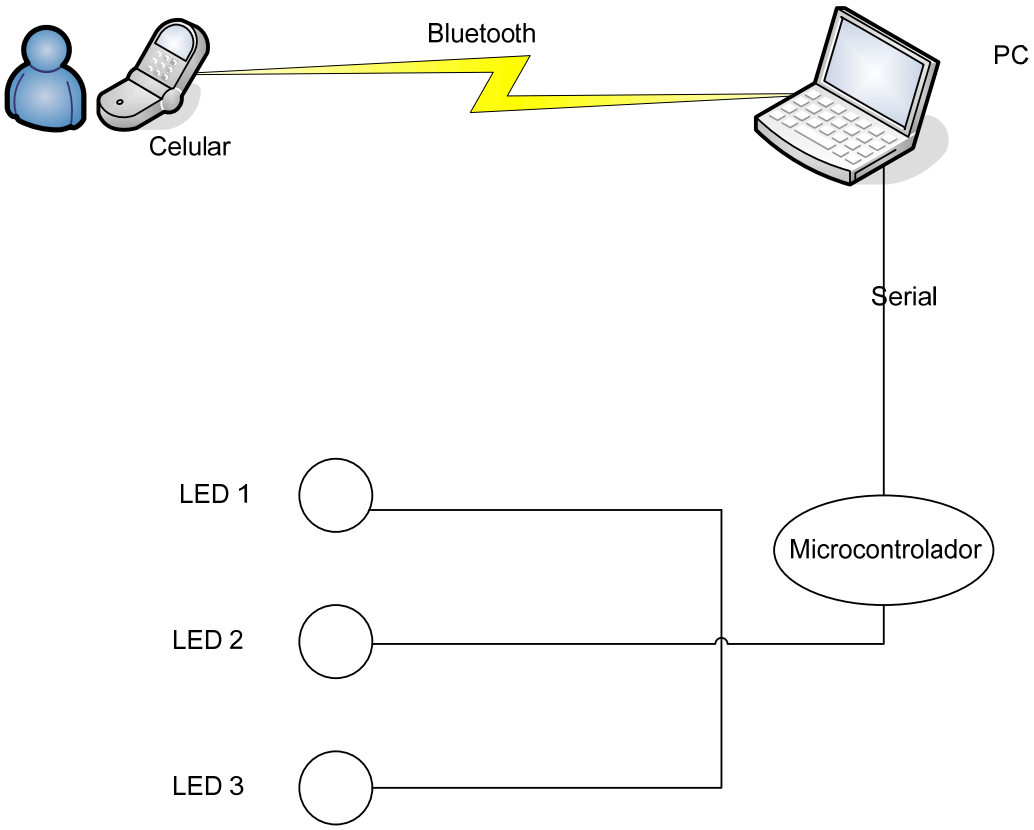

Figura 3-1 - Visão geral do projeto

3.1 Projeto para a aplicação para o celular

No celular, a MIDlet, que nada mais é que o próprio software desenvolvido sob plataforma J2ME, faz uma interface com o usuário, e possui um

comando para estabelecer uma conexão Bluetooth com o computador. Nesse software, o usuário pode digitar o caractere que corresponde ao que ele deseja e em seguida enviá-lo para o servidor escolhido previamente. Essa interface deve também informar quando a conexão não obtém êxito. Tal funcionalidade é abrangida pelo software. Quando a conexão apresentar algum problema, esse é então mostrado ao usuário.

O projeto de fluxograma das operações do celular esta representado na Figura 3-2 abaixo:

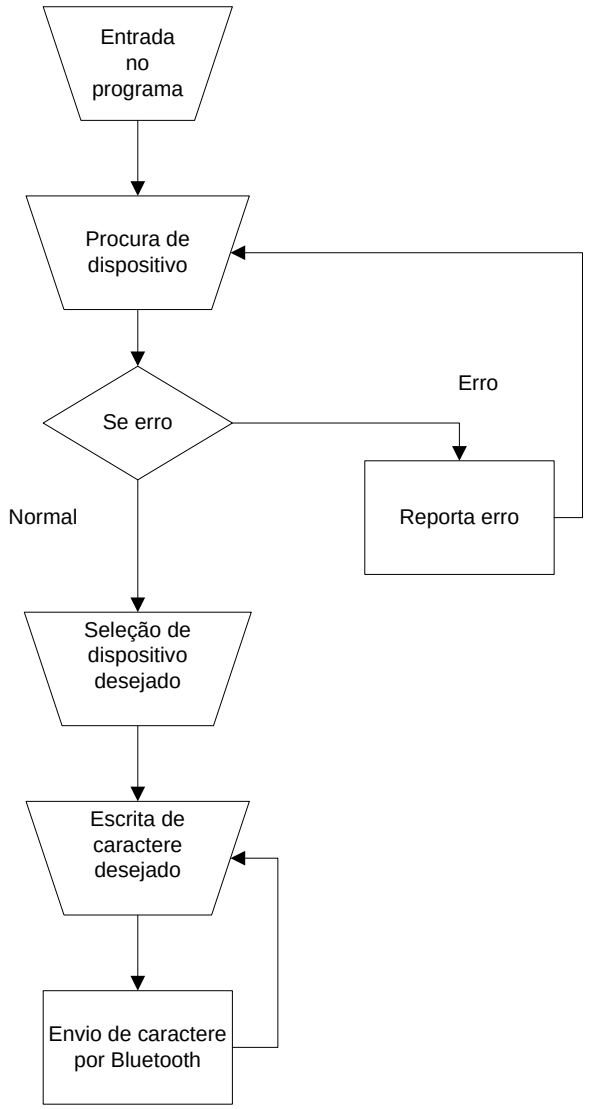

Figura 3-2 - Fluxograma do celular

3.2 Projeto para aplicação do computador

O computador apresenta um software desenvolvido usando a tecnologia J2SE que, por sua vez, utiliza uma API específica para a comunicação Bluetooth e outra para comunicação serial. É necessário utilizar essas APIs, pois o computador faz o papel de intermediador na solução. Ele fica aguardando uma conexão cliente ser estabelecida. Quando a conexão é estabelecida, o computador recebe o dado enviado pelo celular e então envia o comando para a porta serial conectada a um microcontrolador. E o fluxograma que demonstra as operações realizadas pelo servidor é o da Figura 3-3.

3.3 Projeto para aplicação do microcontrolador

O microcontrolador é programado em linguagem assembly. O programa nele armazenado, faz com que o mesmo receba os dados da porta serial, armazene em seu buffer, compare o dado com as opções permitidas e execute os comandos relativos a estas opções. O diagrama com as execuções e comparações feitas pelo software do microcontrolador é mostrado na Figura 3-4.

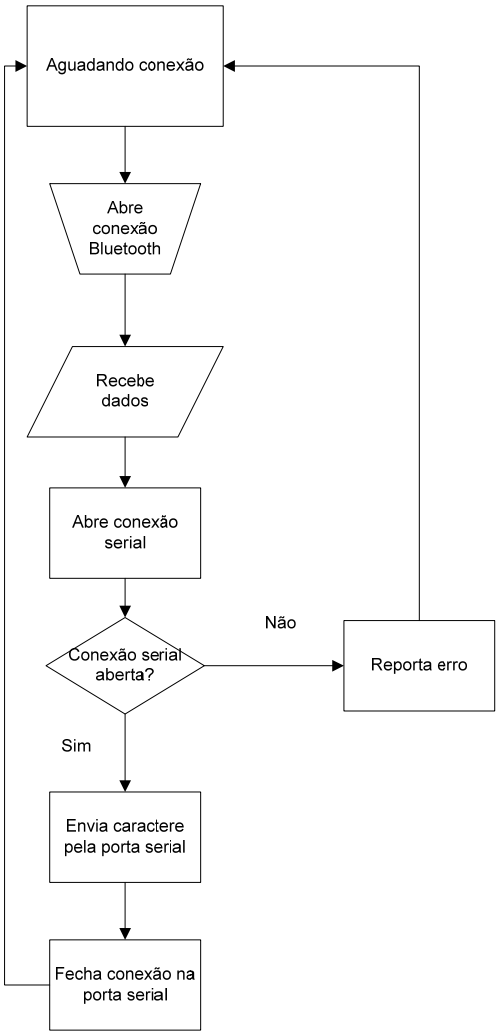

Figura 3-3 – Fluxograma do computador

3.1 Projeto de envio e recebimento de dados entre os dispositivos

Os programas devem fazer as comunicações uns com os outros. A Figura 3-5 demonstra o estabelecimento de conexão entre os dispositivos e também ilustra o envio de dados entre eles.

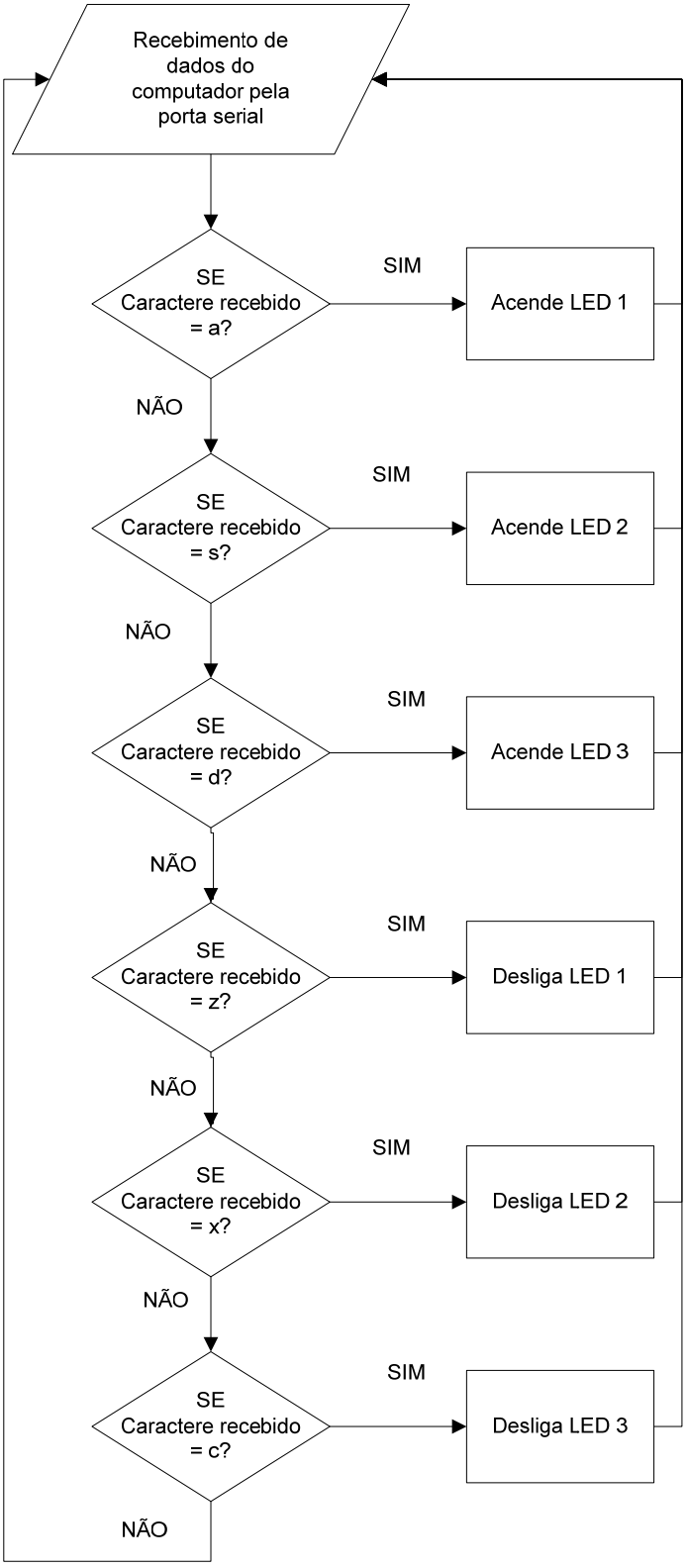

Figura 3-4 - Fluxograma do microcontrolador

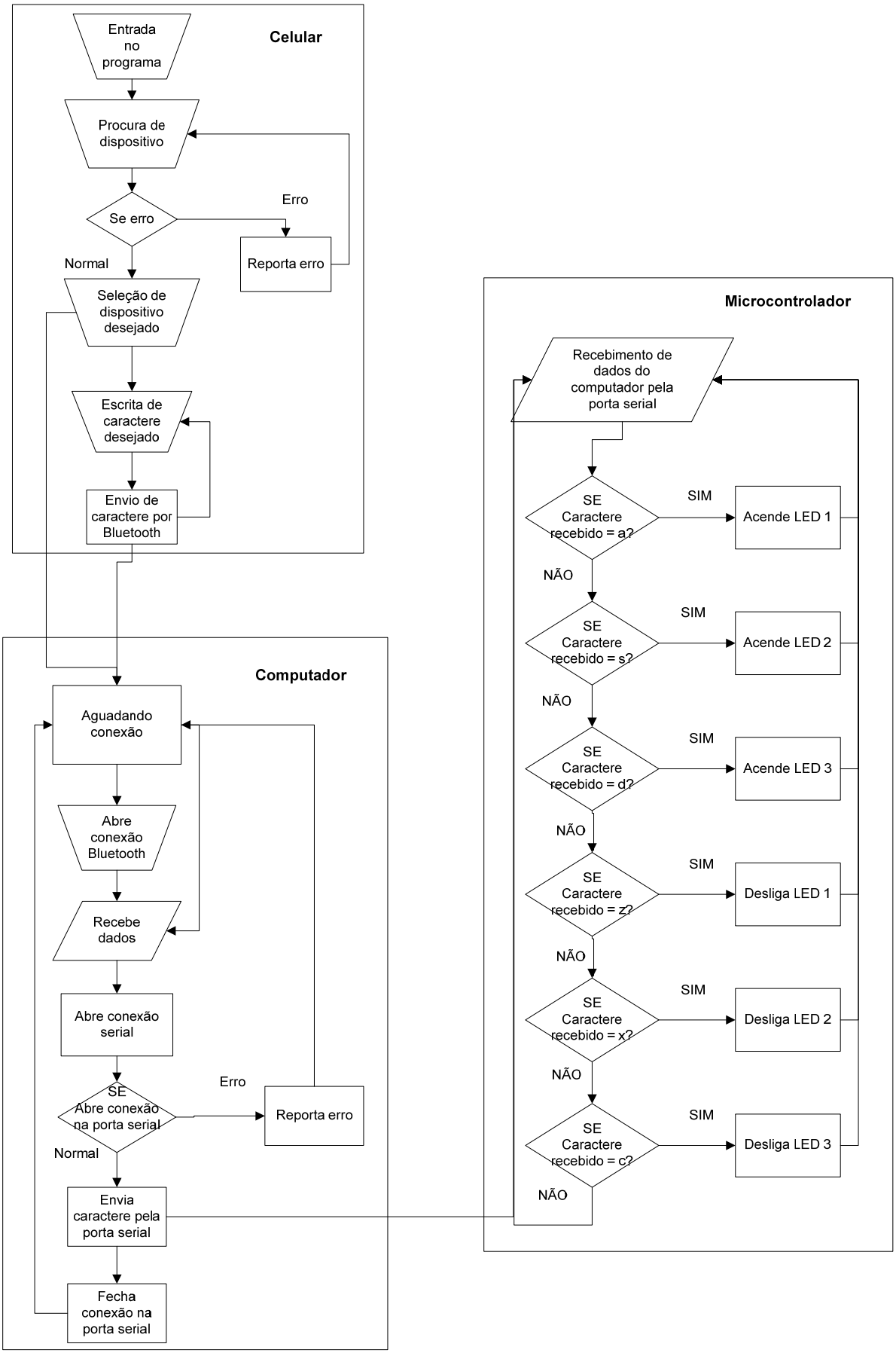

Figura 3-5 - Fluxograma dos equipamentos se comunicando

Capítulo 4 - Aplicação da Solução com Resultados

Considerando o problema apresentado neste projeto, analisando a topologia apresentada no capítulo anterior, é possível então a codificação do software desejado.

4.1 Hardware

- 1 computador com Bluetooth
- 1 celular com Bluetooth
- 1 kit de microcontrolador
- 1 cabo serial para conectar o computador com o microcontrolador

Os equipamentos utilizados no projeto estão ilustrados na Figura 4-1 seguinte:

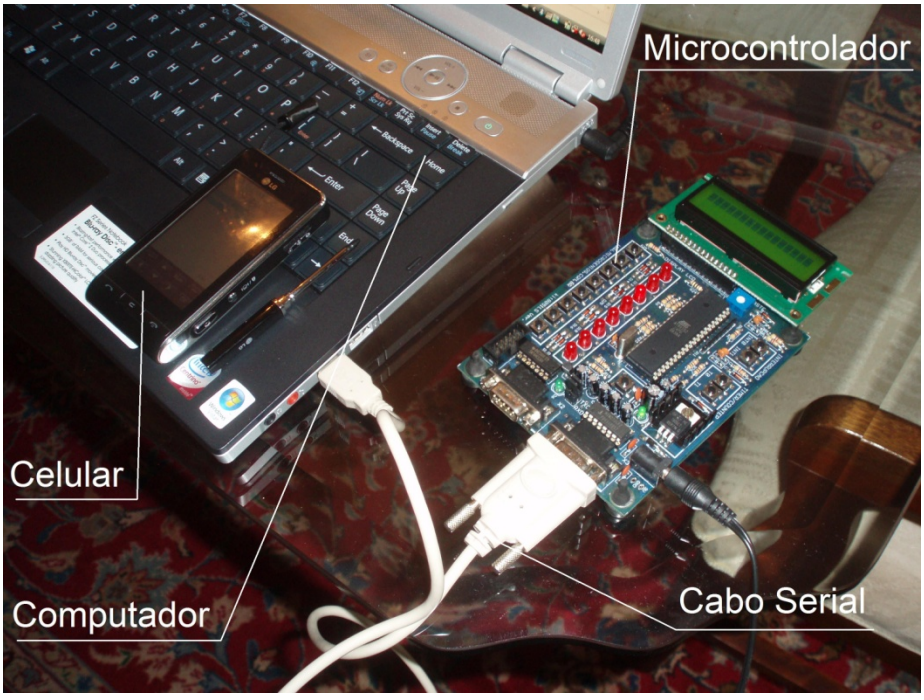

Figura 4-1 - Foto das ligações entre dispositivos

4.2 Software

4.2.1 Aplicação do celular

A aplicação desenvolvida para o celular utiliza a tecnologia J2ME que é voltada para programação de aplicações para dispositivos de pequeno porte. Antes de começar é importante ter em mente alguns conceitos:

Lista – tela que é mostrada no display do aparelho móvel que lista opções. No caso, uma lista será usada para mostrar ao usuário os aparelhos disponíveis em sua volta.

Textbox – tela que é mostrada no display onde o usuário deve escrever o que é pertinente à aplicação. No caso, uma textbox é mostrada ao usuário para que ele digite o caractere correspondente à opção que ele deseja.

O código será comentado a seguir:

```
public class ProjetoFinal extends MIDlet

implements CommandListener,DiscoveryListener  {
```

Aqui tudo começa. Dentro dessa classe está todo o código que é preciso. Nas primeiras linhas desse código pode-se perceber a presença das declarações de variáveis e objetos que serão utilizados posteriormente.

```
startApp()
```

Essa função é chamada quando o programa é iniciado. No primeiro agrupamento de linhas o programa inicia alguns objetos com seus parâmetros

iniciais como 2 listas e uma textbox. No segundo conjunto de linhas alguns comandos já declarados são incluídos tanto nas listas quanto na textbox.

commandAction(Command com, Displayable dis)

O programa, com o auxílio da interface CommandListener, pode ficar esperando a execução de eventos. Quando um evento é identificado, o CommandListener detecta o comando e inicia o commandAction(Command com, Displayable dis). Dentro dessa função está o que o software irá fazer, de acordo com o comando pressionado.

No projeto, há 4 tipos de comandos dentro do commandAction(Command com, Displayable dis): o comando saída, o comando ok e dois comandos seleção. Um dos comandos é acionado quando o usuário pressiona o botão “procurar servidor” na primeira lista. O outro comando é acionado na lista onde no display aparece o resultado da procura de dispositivos.

acharDispositivos()

Quando programa executa o método acharDispositivos(), uma rotina que procura por dispositivos é acionada.

acharServicos()

Nessa função o programa executa uma procura por serviços na sua área de cobertura.

deviceDiscovered(RemoteDevice remoteDevice,DeviceClass deviceClass)

servicesDiscovered(int transID,ServiceRecord[] serviceRecord)

inquiryCompleted(int param)

serviceSearchCompleted(int transID, int respCode)

Esses três métodos devem ser executados juntamente com a interface DiscoveryListener. Eles são chamados quando uma procura por dispositivos for iniciada.

deviceDiscovered(RemoteDevice remoteDevice,DeviceClass deviceClass)

Esse método é chamado pela aplicação quando um dispositivo foi encontrado.

servicesDiscovered(int transID,ServiceRecord[] serviceRecord)

Esse método é chamado quando um serviço é encontrado

inquiryCompleted(int param)

Esse método é chamado quando a inquiry foi completada

serviceSearchCompleted(int transID, int respCode)

Esse método é chamado quando uma procura por serviços foi completada.

do_alert(String msg,int time_out)

A função do do_alert() é mostrar no display alguns avisos que se fazem necessários para que o usuário se mantenha informado.

pauseApp()

Quando o programa precisa ser pausado, essa função executa isso.

destroyApp(boolean unconditional)

Quando o programa precisa ser finalizado, essa função executa isso.

4.2.2 Aplicação do computador

A aplicação do computador foi dividida em duas classes. A primeira classe possui a execução da conexão Bluetooth e chama as funções presentes na segunda classe. A segunda possui as funções para a comunicação serial.

A execução do programa começa na função main(). Dentro da função main() há apenas uma linha. Essa linha chama a função Server().

Server()

O Server() começa com a execução da função executaConfiguracoesIniciais().

executaConfiguracoesIniciais()

Aqui o software configura o dispositivo Bluetooth com parâmetros iniciais. Uma dos parâmetros é fazer com que o dispositivo possa ser descoberto.

aguardaConexao()

Quando o servidor executa essa função, o servidor fica disponível e aguardando conexões clientes.

Para dar prosseguimento à operação, um celular cliente conecta com o servidor e, em seguida, executa uma rotina para recebimento do dado. Com o dado já no computador, resta agora enviá-lo pela porta serial. O envio do dado recebido pela porta serial é tratado na função processaDados().

processaDados()

Essa chama funções da classe Serial.java. Na classe Serial.java há apenas funções que trabalham com a conexão serial.

ObterIdDaPorta()

Essa função verifica se a porta designada está operante.

AbrirPorta()

Aqui a conexão serial é aberta.

EnviarUmaString(conteúdo)

Com essa função o computador envia o caractere recebido pelo Bluetooth.

FecharCom()

Depois de enviar o dado desejado é necessário que o programa feche a conexão serial. Isso é feito nessa função.

4.2.3 Aplicação do microcontrolador

A aplicação desenvolvida para o microcontrolador tem o objetivo de receber dados da porta serial e comparar com caracteres pré-definidos. Se o dado recebido for igual aos caracteres que ele espera receber, então, o microcontrolador liga ou desliga os LEDs desejados de acordo com o caractere recebido.

```
mov ie,#10010000b  ;Habilita a interrupção serial
mov scon,#50h      ;Modo 1
```

```
mov pcon,a          ;Coloca bit SMOD ativo
mov tmod,#20h       ;Timer 1 em modo 2
mov th1,#0F4h       ;4800bps para cristal de 11,0592mhz
setb tr1            ;Iniciar timer 1
sjmp $              ;Loop infinito
```

O trecho do código acima define parâmetros para a comunicação serial. Sem essas definições, o programa pode não estabelecer a conexão serial com sucesso.

As rotinas definidas no programa como: seri, limpa, limpa1, limpa2, limpa3 e limpa4; comparam o caractere recebido com ‘a’, ‘s’, ‘d’, ‘z’, ‘x’ e ‘c’ respectivamente. Por exemplo: se o microcontrolador receber um ‘z’, então ele primeiramente executa a rotina seri, que o compara com o caractere ‘a’. Como o caractere não é ‘a’, então ele entra na próxima rotina até que haja igualdade entre o caractere recebido e o caractere da rotina.

Cada rotina executa uma ação diferente. A relação de ações que cada rotina faz, está listada a seguir:

- Seri – liga o primeiro LED
- Limpa – liga o segundo LED
- Limpa1 – liga o terceiro LED
- Limpa2 – desliga o primeiro LED
- Limpa3 – desliga o segundo LED
- Limpa4 – desliga o terceiro LED

4.3 Execução no celular

Uma vez conectados todos os aparelhos e cabos, pode-se iniciar a execução do projeto. Primeiramente é ligado o dispositivo Bluetooth do celular e do computador. É importante frisar que o display do aparelho celular utilizado possui a tecnologia touch screen, o que significa que algumas opções deverão ser selecionadas com o toque no próprio display. Após garantir que a conexão Bluetooth está funcionando devemos iniciar o aplicativo presente no celular chamado Controle universal como apresenta a Figura 4-2.

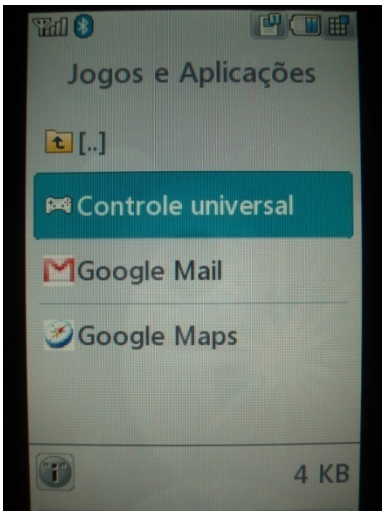

Figura 4-2 - Lista de aplicativos disponíveis no celular

Quando o aplicativo é iniciado, é apresentada uma opção em que o celular começa a procurar por dispositivos que possuam Bluetooth e forneçam o serviço desenvolvido no projeto. A opção é ilustrada na Figura 4-3.

Figura 4-3 - Primeira imagem ao entrar no aplicativo desenvolvido

Para que o usuário seja informado que o aparelho celular está procurando dispositivos, é exibido um alerta no display dizendo “Procurando dispositivos...”, como na Figura 4-4.

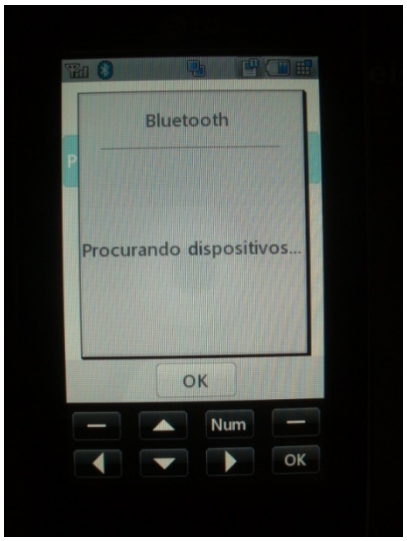

Figura 4-4 - Procurando dispositivos

Concluída a pesquisa por dispositivos que se enquadram nos requisitos, ou seja, equipamentos que forneçam o serviço requerido e que possuam Bluetooth, uma lista é mostrada ao usuário. O usuário deve, então, selecionar o dispositivo

desejado. No exemplo da Figura 4-5, só existe uma opção: HUGO-PC. A opção é a que deverá ser escolhida no caso, pois esse é o nome atribuído ao computador servidor da aplicação.

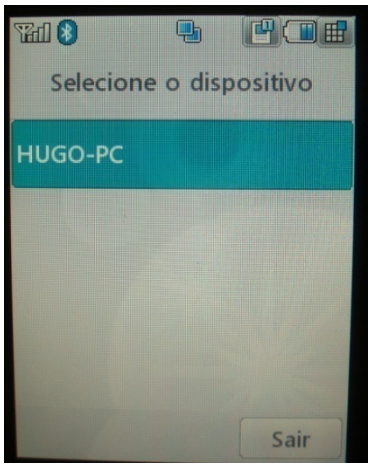

Figura 4-5 - Lista de dispositivos que oferecem o serviço e possuem Bluetooth

Figura 4-6 - Outro exemplo de procura de dispositivos

Depois de selecionado o servidor ao qual se deve conectar, o celular mostra em seguida, uma *TextBox* onde deverá ser escolhido o que será transmitido para o computador servidor da aplicação. No exemplo, os caracteres “a”, “s”, “d” ligam os 1º, 2º e 3º LEDs respectivamente, e os caracteres “z”, ”x”, ”c” desligam os mesmos LEDs. Na Figura 4-7, o usuário está prestes a enviar o caractere “z”, que no caso significa desligar o primeiro LED.

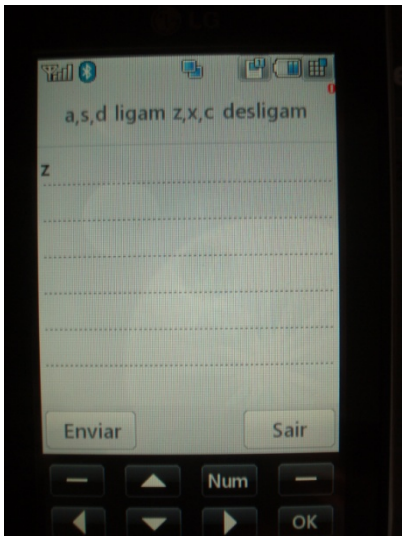

Figura 4-7 - Tela onde deve ser inserido o caractere desejado

4.4 Execução no computador

Na Figura 4-8 é mostrada uma tela com o servidor executando suas funções. A ferramenta para desenvolvimento em Java Eclipse é usada para que o código seja rodado e mostre no console seus resultados.

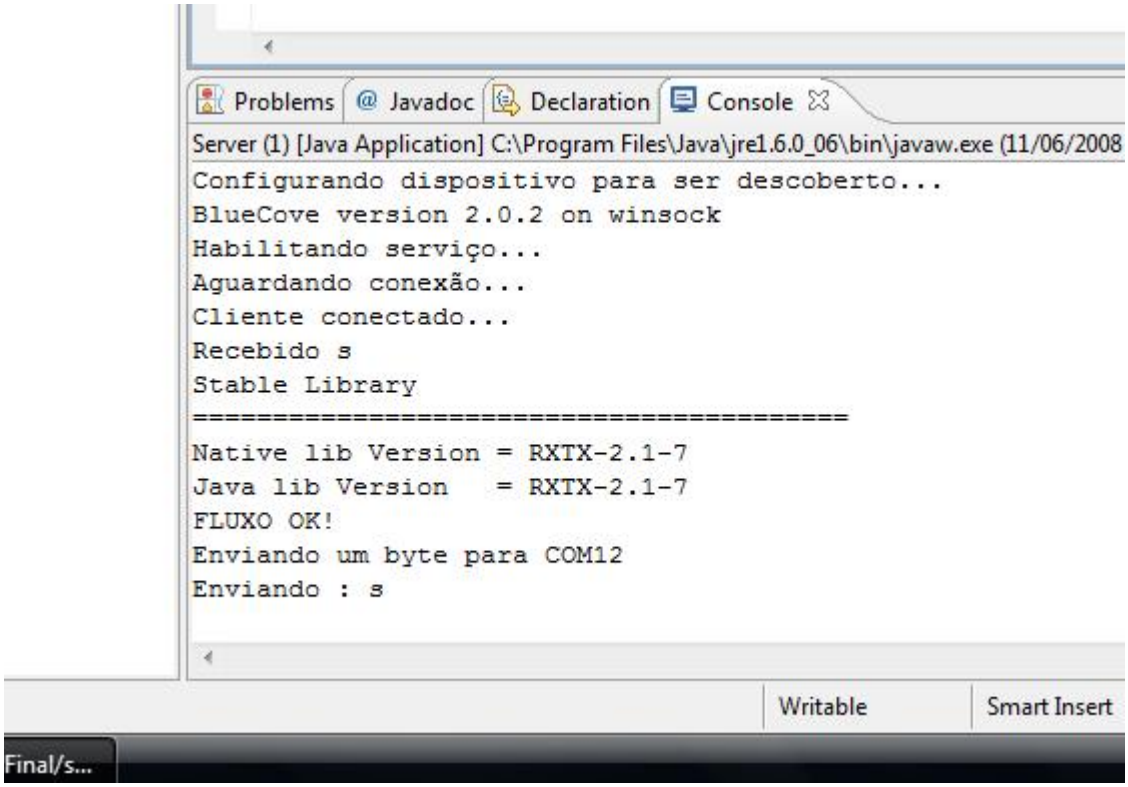

Figura 4-8 - Print Screen do servidor executando o serviço

As quatro primeiras linhas representam a iniciação do serviço. Primeiramente, o código configura o dispositivo para ser descoberto por outros aparelhos. A segunda linha mostra a API usada para a conexão Bluetooth; e na terceira e quarta, o programa exibe mensagens informando que o servidor está começando a executar o serviço e está pronto esperando alguma conexão cliente.

Quando o aparelho celular entra em contato, o servidor exibe outra mensagem relatando que há um cliente conectado. Com a conexão concluída, o cliente está pronto para o envio do caractere, situação que também está ilustrada no exemplo. O cliente, na próxima linha, já enviou um caractere; e o caractere envolvido é o “s”.

Seguindo a ordem, as próximas quatro linhas dizem respeito à outra API utilizada na codificação do projeto. Ela é a API de comunicação serial, indispensável

para a conexão com o microcontrolador. O restante das linhas diz respeito ao envio do caractere pela porta serial que tem do outro lado o microcontrolador esperando o caractere em questão.

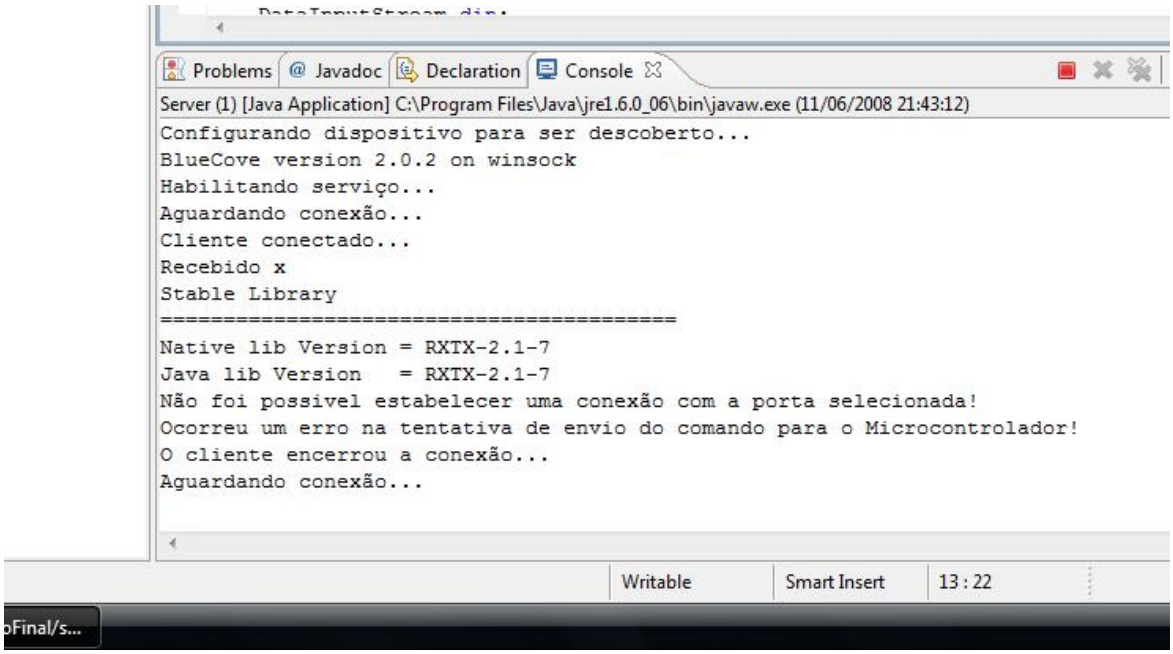

Figura 4-9 Servidor detecta que a porta serial não está conectada

Caso o cabo serial não esteja conectado ao computador, as mensagens: “Não foi possível estabelecer uma conexão com a porta selecionada!” e “Ocorreu um erro na tentativa de envio do comando para o Microcontrolador!”, como na figura 4-9, aparecem.

4.5 Execução no microcontrolador

Este programa faz comunicação serial entre PC e microcontrolador. Os parâmetros da conexão escolhidos são os seguintes:

- Envio de bits = 4800bps
- Bits de dados = 8bits de dados
- Paridade = sem paridade
- Bits de parada = 1 bit de parada
- Controle de fluxo = nenhum.

Esses parâmetros devem ser os mesmos nas duas pontas do cabo serial para que a comunicação seja efetuada com sucesso.

O programa no microcontrolador é responsável pela ativação e desativação destes LEDs.

O programa estabelece uma conexão serial full duplex com o computador, ou seja, que permite o recebimento e envio de informações simultaneamente. O microcontrolador espera o recebimento dos caracteres predefinidos. Para o recebimento dos caracteres “a”, “s” e “d”, o microcontrolador ascende os LEDs 1, 2 e 3 respectivamente. Já o recebimento dos caracteres “z”, ”x” e “c” pela porta serial, desliga os LEDs 1,2 e 3 respectivamente.

Na Figura 4-10 está representada a ligação de todas as lâmpadas.

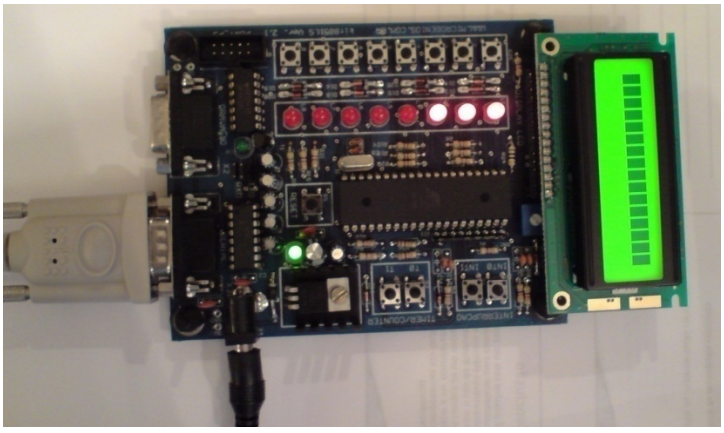

Figura 4-10 - Os 3 LEDs acesos

Capítulo 5 – Conclusão

Seguindo o mercado mundial de automação residencial, é percebido, portanto, que com a utilização de dispositivos que são considerados normais no ambiente residencial, no caso do celular e do computador, é possível um arranjo tal, de forma que o conforto e bem-estar do usuário sejam melhorados.

Foi visto também que o ramo de compra e venda de computadores vem aumentando com políticas do governo brasileiro, vindo a facilitar sua aquisição a uma grande parte da população. Além das aplicações típicas dos computadores domésticos, vimos uma possibilidade de sua aplicação na automação residencial.

Neste projeto, o único dispositivo a não fazer parte do dia-a-dia de pessoas comuns é o microcontrolador. Utilizado no projeto para controlar o acionamento ou não dos LEDs nele contidos. Considerando que os outros equipamentos são, e estão cada vez mais disponíveis nas casas dos brasileiros, o projeto aqui apresentado não possui um custo elevado; pelo menos para ligar LEDs.

Foi percebido também durante a fase de testes que o aparelho cliente, no caso o celular, no momento da procura por aparelhos que fornecem o serviço de Bluetooth, encontra dificuldade quando existe mais de um dispositivo ao alcance. Mostrando então no display do celular que registros não foram encontrados.

Podemos então a partir do estudo aqui proposto, concluir que a automação residencial está cada vez mais acessível. Com avanços tecnológicos, barateamento de equipamentos, políticas favoráveis e facilidade crescente no manuseio e codificação dos itens.

5.1 Problemas encontrados

Durante a confecção do projeto foram encontradas algumas dificuldades, elas estão listadas a seguir:

- Primeiramente foi o aprendizado sobre a linguagem Java. A linguagem Java não é uma linguagem fácil de aprender. Requer muitas horas de estudo por dia. Além disso, é necessário o conhecimento sobre Programação Orientada a Objetos (POO).
- Achar API para comunicação serial que funcionasse no MS Windows. No site da SUN não existe uma versão dessa API específica para o MS Windows. Foi preciso pesquisar e testar muito para descobrir a RXTXComm (API utilizada para realizar a comunicação serial).
- Procura de ajuda com APIs utilizadas. A programação utilizando as APIS de comunicação serial e Bluetooth não é um assunto conhecido por muitas pessoas. Portanto, o apoio que tive foi o da Internet por meio de comunidades, site do Java e sites com lições.
- Dificuldade na procura de dispositivos no cliente quando o ambiente possui mais de um aparelho equipado de Bluetooth.

5.2 Trabalhos Futuros

Alguns fatores presentes no projeto que podem ser alterados e melhorados. Eles estão listados a seguir:

1. O projeto poderia ser implementado com lâmpadas reais, aplicando uma automação residencial real e tornando o projeto mais comercializável.
2. Melhoramento na dificuldade que o presente trabalho encontra ao listar os dispositivos quando existe mais de um dispositivo com o serviço Bluetooth ativado no perímetro.
3. A interface com o usuário poderia ser mais intuitiva. Uma idéia seria tornar o envio de caracteres transparente para o usuário, mostrar as opções disponíveis em uma lista com escolha exclusiva, e daí enviar o caractere, ao invés de escrever o caractere na textbox e enviá-lo.
4. Os erros que são mostrados no computador poderiam ser visualizados no display do celular, assim como uma confirmação ou exibição de erro a respeito da conclusão da operação efetuada.
5. Uma tela com o status sobre cada lâmpada ou LED também poderia ser incluída, para que o usuário saiba quais lâmpadas ou LEDs estão ligadas ou desligadas.

Bibliografia

ATMEL Corporation [2005]. AT89S52 Datasheet.

CORRADI JUNIOR [2006]; Microcontrolador 8051, Colégio Técnico de Campinas

FOLHA ON LINE; Brasil vende mais de 21 computadores por minuto, diz consultoria, disponível em <http://www1.folha.uol.com.br/folha/informatica/ult124u409535.shtml> acessado em 06/06/2008

FOROUZAN, Behrouz A. [2006]; Comunicação de Dados e Redes de Computadores

GIMENEZ, Salvador P. [2002]; Microcontroladores 8051

HORSTMANN, Cay S. [2001]; Core JAVA 2 Volume I – Fundamentos. Editora Makron

JOHNSON, Thienne M. [2008]; Java para dispositivos móveis. Editora Novatec.

LABRAMO Centronics, [2006]; Seriais: RS 232/ 422/ 485/ 499/ 530 disponível em http://www.labramo.com.br/faq1.htm#RS232 acessado em 06/07/2008

LESSA, Renato e SILVA, Jaguaraci; Sistema integrado e multiplataforma para controle remoto de residências, disponível em <http://www.logicengenharia.com.br/mcamara/ALUNOS/Residencial.PDF> acessado em 17/06/2008

MILLER, Michael [2001]; Descobrindo Bluetooth. Editora Campus.

MORIMOTO, Carlos; Bluetooth, disponível em <http://www.guiadohardware.net/artigos/bluetooth/> acessado em 05/06/2008

MUCHOW, John W. [2004]; Core J2ME: Tecnologia e MIDP. 1ª ed. Makron Books.

OMEGA.COM; The RS-232 Standard, disponível em <http://www.omega.com/TechRef/pdf/RS-232.pdf acessado em 06/06/2008

SUN; Part II: The Java APIs for Bluetooth Wireless Technology, disponível em <http://developers.sun.com/mobility/midp/articles/bluetooth2/ acessado em 26/05/2008

SUN; User`s guide: Sun JavaTM Wireless Toolkit for CLDC, disponível em <http://java.sun.com/products/sjwtoolkit/wtk2.5.2/docs/UserGuide.pdf> Acessado em 12/03/2008

Apêndice I

Aqui será disposto o código utilizado para realização das atividades previstas no projeto de conclusão do curso. Para a execução do código foi necessária a utilização de APIs (*Application Program Interface*) específicas para o tratamento da conexão serial e para a conexão Bluetooth. A API utilizada para comunicação serial foi a *bluecove* versão 2.0.2; e a API para comunicação serial foi a RXTXcomm.

Código do servidor

Em seguida será apresentado o código que está residindo e executando no computador:

Classe Serial.java

```
import gnu.io.CommPortIdentifier;

import gnu.io.SerialPort;

import java.io.OutputStream;
```

```
public class Serial {


        public String Dadoslidos;

        public int nodeBytes;

        private int baudrate;

        private int timeout;

        private CommPortIdentifier cp;

        private SerialPort porta;

        private OutputStream saida;

        private String Porta;

        protected String peso;


        public Serial(String p, int b, int t) {          //parâmetros para conexão serial

                this.Porta = p;

                this.baudrate = b;

                this.timeout = t;

        }


        public boolean ObterIdDaPorta() {          // Verifica se a porta está pronta
```

```
            try {
                cp = CommPortIdentifier.getPortIdentifier(Porta);
                if (cp == null) {
                    System.out.println("Erro na abertura da porta!");
                    return false;
                }
            } catch (Exception e) {
                System.out
                        .println("Não foi possivel estabelecer uma conexão
com a porta selecionada!");
                return false;
            }
            return true;
        }

        public boolean AbrirPorta() {       //Abre conexão na porta serial
            try {
                porta = (SerialPort) cp.open("SerialComLeitura", timeout);
                porta.setSerialPortParams(baudrate, porta.DATABITS_8,
```

```
                                porta.STOPBITS_1, porta.PARITY_NONE);
                porta.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
        } catch (Exception e) {
                System.out.println("Erro abrindo comunicação: " + e);
                return false;
        }
        return true;
}

public boolean EnviarUmaString(String msg) {
                        //envia a string recebida pela conexão Bluetooth
        try {
                saida = porta.getOutputStream();
                System.out.println("FLUXO OK!");
        } catch (Exception e) {
                System.out.println("Erro.STATUS: " + e);
                return false;
        }
        try {
```

```
                System.out.println("Enviando um byte para " + Porta);
                System.out.println("Enviando : " + msg);
                saida.write(msg.getBytes());
                Thread.sleep(100);
                saida.flush();
        } catch (Exception e) {
                System.out.println("Houve um erro durante o envio. ");
                System.out.println("STATUS: " + e);
                return false;
        }
        return true;

}

public void run() {
        try {
                Thread.sleep(5);
        } catch (Exception e) {
                System.out.println("Erro de Thred: " + e);
```

```
        }
    }

    public boolean FecharCom() {              //Fecha comunicação na porta serial
        try {
            porta.close();
        } catch (Exception e) {
            System.out.println("Erro fechando porta: " + e);
            return false;
        }
        return true;
    }

    public String obterPorta() {          //retorna a porta

        return Porta;
    }

    public int obterBaudrate() {
```

```
                    //retorna Baudrate (um parâmetro para conexão serial)
            return baudrate;
        }
}
```

Classe Server.java

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



public class Server {
                    //Parâmetros da comunicação serial
        String PORTA = "COM12";
        int BAUDRATE = 4800;
        int TIMEOUT = 0;

        public final UUID uuid = new UUID( // UUID do serviço, tem que ser única
```

```
                "27012f0c68af4fbf8dbe6bbaf7aa432a", false);
        // Pode ser gerada aleatoriamente
public final String name = "Servidor";      // Nome do serviço
public final String url = "btspp://localhost:" + uuid
                + ";name=" + name + ";authenticate=false;encrypt=false;";
                // URL do serviço
LocalDevice local = null;
StreamConnectionNotifier server = null;
StreamConnection conn = null;
DataInputStream din;

public Server() {                   //a execução do programa começa aqui
        executaConfiguracoesIniciais();
        //configura os parametros iniciais para o funcionamento do serviço
        aguardaConexao();
        //fica esperando uma conexão cliente
        while (true)
                try {
                        String cmd = "";
```

```
                    char c;
                    while (((c = din.readChar()) > 0) && (c != '\n'))
                        cmd = cmd + c;
                    System.out.println("Recebido " + cmd);
                    if (!processaDados(cmd))
                    //executa a rotina para envio de dados para porta serial
                        System.out
                                .println("Ocorreu um erro na tentativa
de envio do comando para o Microcontrolador!");
                } catch (Exception e) {
                    if (e.toString().equals("java.io.EOFException")) {
                        System.out
                                .println("O cliente encerrou a
conexão...");
                        aguardaConexao();
                    } else {
                        System.out.println("Erro: " + e.toString()
                                + ".\nO servidor será encerrado!");
                    }
```

```
                }
        }

        public void executaConfiguracoesIniciais() {
                //Estabelece as configurações iniciais para a comunicação Bluetooth
                try {
                        System.out
                                        .println("Configurando dispositivo para ser
descoberto...");
                        local = LocalDevice.getLocalDevice();
                        local.setDiscoverable(DiscoveryAgent.GIAC);
                        System.out.println("Habilitando serviço...");
                        server = (StreamConnectionNotifier) Connector.open(url);
                } catch (Exception e) {
                        System.out.println("Erro: " + e.toString());
                }
        }

        public void aguardaConexao() {
```

```
                //Aguarda uma conexão cliente
        try {
                System.out.println("Aguardando conexão...");
                conn = server.acceptAndOpen();
                System.out.println("Cliente conectado...");
                din = new DataInputStream(conn.openInputStream());
        } catch (Exception e) {
                System.out.println("Erro: " + e.toString());
        }
}

public boolean processaDados(String conteudo) {
        Serial leitura = new Serial(PORTA, BAUDRATE, TIMEOUT);
        if (!leitura.ObterIdDaPorta())
                //Verifica se a porta pode ser utilizada
                return false;
        if (!leitura.AbrirPorta())
                //Abre comunicação serial
                return false;
```

```
            if (!leitura.EnviarUmaString(conteudo))
                //Envia o caractere pela porta já aberta
                return false;
            if (!leitura.FecharCom())
                //Fecha conexão serial
                return false;
            return true;
        }

        public static void main(String args[]) {
            new Server();
        }

}
```

Código do aparelho celular

```
package hello;
```

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import javax.bluetooth.*;
import java.io.*;


public class ProjetoFinal extends MIDlet
   implements CommandListener,DiscoveryListener  {
    List lista_princ,lista_disp;
    Command saida,ok;
    TextBox cmd;
    Display display;
    java.util.Vector dispositivos,servicos;
    LocalDevice local;
    DiscoveryAgent agente;
    DataOutputStream dout;
    int currentDevice = 0;
          //Usado como indicador para o dispositivo contactado pelo servidor
```

```
public void startApp() {
    lista_princ = new List("Automacao residencial",Choice.IMPLICIT);
                  //Menu principal
    lista_disp  = new List("Selecione o dispositivo",Choice.IMPLICIT);
                 //lista de dispositivos
    cmd       = new TextBox("a,s,d ligam z,x,c desligam","",1,TextField.ANY);
    saida      = new Command("Sair",Command.EXIT,1);
    ok        = new Command("Enviar",Command.OK,1);
    display   = Display.getDisplay(this);

    lista_princ.addCommand(saida);
    lista_princ.setCommandListener(this);
    lista_disp.addCommand(saida);
    lista_disp.setCommandListener(this);
    cmd.addCommand(ok);
    cmd.addCommand(saida);
    cmd.setCommandListener(this);

    lista_princ.append("Procurar servidor",null);
```

```
    display.setCurrent(lista_princ);



    }
public void commandAction(Command com, Displayable dis) {
    if (com == saida){                          //Botão saida apertado
        destroyApp(false);
        notifyDestroyed();
    }
    if (com == List.SELECT_COMMAND){
        if (dis == lista_princ){                        //Após selecionar o celular procura
            if (lista_princ.getSelectedIndex() >= 0){           //dispositivos
                acharDispositivos();
                do_alert("Procurando dispositivos...", Alert.FOREVER);
            }
        }
        if (dis == lista_disp){                         //Seleção de dispositivo
            StreamConnection con = null;
            ServiceRecord servico = (ServiceRecord)
                        servicos.elementAt(lista_disp.getSelectedIndex());
```

```
        String url = servico.getConnectionURL(
                        ServiceRecord.NOAUTHENTICATE_NOENCRYPT,
                        false);
        try {
            con = (StreamConnection) Connector.open(url);
                        //Estabelece a conexão
            dout = new DataOutputStream(con.openOutputStream());
                        //Pega o output stream para ser enviado
            display.setCurrent(cmd);
                        //Mostra a TextBox
        } catch (Exception e) {this.do_alert("Erro na conexao" , 4000);}


    }
 }
if(com == ok){                          //usuário mandando comando
    try{
        dout.writeChars(cmd.getString());
        dout.flush();
        cmd.setString("");
```

```
        } catch (Exception e) {this.do_alert("Erro no envio de dados" , 4000);}
    }
}

public void acharDispositivos(){
                        //procura dispositivos
    try{
        dispositivos             = new java.util.Vector();
        LocalDevice local    = LocalDevice.getLocalDevice();
        DiscoveryAgent agent = local.getDiscoveryAgent();
        agent.startInquiry(DiscoveryAgent.GIAC,this);
    }catch(Exception e){this.do_alert("Erro no inicio da procura" , 4000);}
 }

public void acharServicos(RemoteDevice device){
                        //procura serviços
    try{
        UUID[] uuids  = new UUID[1];
        uuids[0]      = new UUID("27012f0c68af4fbf8dbe6bbaf7aa432a",false);
```

```
                //UUID de cada serviço
        local        = LocalDevice.getLocalDevice();
        agente       = local.getDiscoveryAgent();
        agente.searchServices(null,uuids,device,this);
    }catch(Exception e){this.do_alert("Erro no inicio da procura" , 4000);}
}

public void deviceDiscovered(RemoteDevice remoteDevice,DeviceClass
deviceClass) {
                //Método chamado pela interface DiscoveryListener
                //Método chamado quando dispositivo é encontrado
    dispositivos.addElement(remoteDevice);
}

public void servicesDiscovered(int transID,ServiceRecord[] serviceRecord) {
                //Método chamado pela interface DiscoveryListener
                //Chamado quando é encontrado serviço durante a procura
    for (int x = 0; x < serviceRecord.length; x++ )
        servicos.addElement(serviceRecord[x]);
```

```
    try{
        lista_disp.append(((RemoteDevice)dispositivos.elementAt(currentDevice)).
                                    getFriendlyName(false),null);
    }catch(Exception e){this.do_alert("Erro no inicio da procura" , 4000);}
}
public void inquiryCompleted(int param){
                        //Método chamado pela interface DiscoveryListener
                        //Chamado quando a inquiry está completa
    switch (param) {
        case DiscoveryListener.INQUIRY_COMPLETED:
                        //Inquiry completada normalmente
            if (dispositivos.size() > 0){
                        //Pelo menos um dispositivo foi encontrado
                servicos = new java.util.Vector();
                this.acharServicos((RemoteDevice)
                        dispositivos.elementAt(0));
                        //verifica se o primeiro dispositivo oferece o serviço
            }else
                do_alert("Dispositivos nao encontrados",4000);
```

```
        break;
      }
}

public void serviceSearchCompleted(int transID, int respCode) {
                        //Método chamado pela interface DiscoveryListener
                        //Chamado quando a procura por serviços foi completada
   switch(respCode) {
      case DiscoveryListener.SERVICE_SEARCH_COMPLETED:
         if(currentDevice == dispositivos.size() -1){
                        //Todos os dispositivos foram procurados
            if(servicos.size() > 0){
               display.setCurrent(lista_disp);
            }else
               do_alert("O servico nao foi encontrado",4000);
         }else{                        //Procura o próximo dispositivo
            currentDevice++;
            this.acharServicos((RemoteDevice)dispositivos.elementAt(currentDevice));
         }
```

```
        break;
        case DiscoveryListener.SERVICE_SEARCH_DEVICE_NOT_REACHABLE:
            this.do_alert("Dispositivo nao alcancavel" , 4000);
        break;
        case DiscoveryListener.SERVICE_SEARCH_ERROR:
            this.do_alert("Erro na procura de servico" , 4000);
        break;
        case DiscoveryListener.SERVICE_SEARCH_NO_RECORDS:
           this.do_alert("Registros nao encontrados" , 4000);
        break;
        case DiscoveryListener.SERVICE_SEARCH_TERMINATED:
           this.do_alert("Inquiry Cancelada" , 4000);
        break;
    }
}

public void do_alert(String msg,int time_out){
                        //Emite alertas no display do celular
   if (display.getCurrent() instanceof Alert ){
```

```
        ((Alert)display.getCurrent()).setString(msg);
        ((Alert)display.getCurrent()).setTimeout(time_out);
    }else{
        Alert alert = new Alert("Bluetooth");
        alert.setString(msg);
        alert.setTimeout(time_out);
        display.setCurrent(alert);
    }
}

public void pauseApp() {}
                    //Aplicação pausada
public void destroyApp(boolean unconditional) {}
                    //Aplicação finalizada

}
```

Código do microcontrolador

```
        org 00h
        jmp 40h
        org  23h
        ljmp seri

         org 40h

        mov ie,#10010000b  ;Habilita a interrupção serial
        mov scon,#50h      ;Modo 1
        mov pcon,a         ;Coloca bit SMOD ativo
        mov tmod,#20h      ;Timer 1 em modo 2
        mov th1,#0F4h      ;4800bps para cristal de 11,0592mhz
        setb tr1           ;Iniciar timer 1
        sjmp $             ;Loop infinito
        org 0100

seri:   mov a,sbuf
        clr ri
        cjne a,#'a',limpa  ;compara com o caractere ‘a’
        clr p2.0
```

```
        mov a,#'a'
        mov sbuf,a
        jnb ti,$
        clr ti
        reti

limpa:  cjne a,#'s',limpa1    ;compara com o caractere ‘s’
        clr p2.1
        mov a,#'s'
        mov sbuf,a
        jnb ti,$
        clr ti
        reti

limpa1: cjne a,#'d',limpa2    ;compara com o caractere ‘d’
        clr p2.2
        mov a,#'d'
        mov sbuf,a
        jnb ti,$
        clr ti
        reti
```

```
limpa2:   cjne a,#'z',limpa3    ;compara com o caractere ‘z’
          setb p2.0
          mov a,#'z'
          mov sbuf,a
          jnb ti,$
          clr ti
          reti

limpa3:   cjne a,#'x',limpa4    ;compara com o caractere ‘x’
          setb p2.1
          mov a,#'x'
          mov sbuf,a
          jnb ti,$
          clr ti
          reti

limpa4:   cjne a,#'c',limpa5    ;compara com o caractere ‘c’
          setb p2.2
          mov a,#'c'
          mov sbuf,a
```

```
            jnb ti,$
            clr ti
            reti

espera2:    jnb ti,espera2
            clr ti
            reti
            end
```