UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES

JOSÉ SILVESTRE
ROBERSON CAPILLE
RODRIGO JOSLIM

SISTEMA PARA DETECÇÃO DE AVARIA DE TEMPERATURA EM CENTRAIS DE PROCESSAMENTO DE DADOS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2014

JOSÉ SILVESTRE
ROBERSON CAPILLE
RODRIGO JOSLIM

SISTEMA PARA DETECÇÃO DE AVARIA DE TEMPERATURA EM CENTRAIS DE PROCESSAMENTO DE DADOS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Telecomunicações, do Curso Superior de Tecnologia em Sistemas de Telecomunicações, do Departamento Acadêmico de Eletrônica (DAELN), da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador: Prof. Dr. Luís Alberto Lucas

CURITIBA
2014

TERMO DE APROVAÇÃO

JOSÉ SILVESTRE
RODRIGO JOSLIM
ROBERSON CAPILLE

SISTEMA PARA DETECÇÃO DE AVARIA DE TEMPERATURA EM CPD

Este trabalho de conclusão de curso foi apresentado no dia 27 de Agosto de 2014, como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Telecomunicações, outorgado pela Universidade Tecnológica Federal do Paraná. Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Luis Carlos Vieira
Coordenador de Curso
Departamento Acadêmico de Eletrônica

_____
Prof. Esp. Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

_____
Prof. Dra. Simone Crocetti
UTFPR

_____
Prof. Msc. César Janeczko
UTFPR

_____
Prof. Dr. Luis Alberto Lucas
Orientador - UTFPR

“A Folha de Aprovação assinada encontra-se na Coordenação do Curso”

RESUMO

CAPILLE, R.; JOSLIM, R.; SILVESTRE, J.. **Sistema para detecção de avaria de temperatura em Centrais de Processamento de Dados**. 2014. 129 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações) – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

O presente trabalho apresenta um protótipo de sistema microcontrolado para monitoramento de temperatura em ambiente de Centrais de Processamento de Dados. O sistema é capaz de monitorar a temperatura em tempo real por via de um sensor, identificar se a temperatura está acima do limite desejável e enviar um alerta para um ou mais dispositivos móveis com sistema operacional Android instalado, informando o local da possível queda de refrigeração.

Palavras-chave: Microcontrolador. Sensores. Monitoração Remota. Dispositivos Móveis.

ABSTRACT

CAPILLE, R.; JOSLIM, R.; SILVESTRE, J.. **Sistema para detecção de avaria de temperatura em Centrais de Processamento de Dados**. 2014. 129 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Sistemas de Telecomunicações) – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2014.

This monograph introduces a prototype of microcontroller system for temperature monitoring in Data Centers environments. The system can monitor the temperature in real time via a sensor, identify whether the temperature is above the desirable limit and send an alert to one or more mobile devices with Android operating system, stating the location of the possible fall of refrigeration.

Keywords: Microcontroller. Sensors of Temperature. Remote Monitor. Mobile Devices.

LISTA DE FIGURAS

LISTA DE SIGLAS E ACRÔNIMOS

ADT – *Android Development Tools*
API - *Application Programming Interface*
AVD – *Android Virtual Device*
CPU – *Central Processing Unit*
EEPROM - *Electrically-Erasable Programmable Read-Only Memory*
IDE – *Interface Development Environment*
ISP – *In-System Programming*
LAN – *Local Area Network*
MAN – *Metropolitan Area Network*
NAT – *Network Address Translate*
ROM – *Read Only Memory*
RAM – *Random Access Memory*
TCP/IP – *Transport Control Protocol/ Internet Protocol*
USB – *Universal Serial Bus*
WAN – *Wide Area Network*
3G – Terceira Geração

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos a tecnologia tem se tornado cada vez mais importante, não somente para grandes corporações e órgãos científicos, mas também para o homem comum. Por meios eletrônicos é possível efetuar com facilidade: operações bancárias, acesso a sistemas educacionais, negociações de ativos na bolsa de valores, consulta a previsão do tempo, consulta a mapas, compra de passagens aéreas, empresas podem realizar o processamento de suas finanças, balanços, análises de vendas e demais estratégias por meio de complexos sistemas computacionais.

Devido a essa necessidade, é fundamental manter a disponibilidade da informação e do serviço, o que torna o controle da temperatura onde os servidores ficam alocados um dos fatores a serem observados, medidos e controlados. Com base nisso, a proposta deste trabalho é apresentar um sistema de alerta para eventuais falhas em sistemas de refrigeração que enviará dados sobre uma Central de Processamento de Dados (CPD) afetada para um aplicativo de celular.

Atualmente existem vários meios e soluções para alcançar esse controle. No entanto, o presente trabalho se propõe a usar uma tecnologia nova e em expansão no momento: o uso do sistema operacional *Android*.

1.1 PROBLEMA

Em corporações de médio e grande porte, como empresas prestadoras de serviços de telecomunicações, grandes varejistas e indústrias, há quase sempre grande demanda de processamento de dados, que geralmente é feita por servidores alocados em um ou mais espaços físicos conhecidos por Centrais de Processamentos de Dados ou ainda *Data Centers*. Nesses espaços físicos, onde se executam processos vitais para a empresa, como servidores de aplicação e de banco de dados, processos vinculados a finanças e vendas, sistemas antifraudes e de *business intelligence*, há grande preocupação com a manutenção da sala para que gere condições para o correto funcionamento das máquinas.

Uma falha em um sistema de refrigeração pode significar prejuízo para a empresa, visto que muitos colaboradores serão acionados após o incidente para corrigir os processos e analisar os dados, o que gera custo e retrabalho para a companhia. Além de possível perda de equipamentos.

1.2 JUSTIFICATIVA

Em cada *Data Center*, há um ou mais operadores trabalhando em turnos diferentes, auxiliando no monitoramento dos sistemas e máquinas. Entretanto, existem casos em que não há nenhum operador presente em determinado momento, como, por exemplo: janelas entre um turno e outro, confraternizações de final de ano, onde os operadores deixam seus postos de trabalho de forma temporária e também os horários da madrugada. Ainda existem os casos em que os operadores estão alocados em outros serviços, o que pode provocar a demora na detecção de falha no sistema de refrigeração da sala.

1.3 OBJETIVOS

A seguir estão apresentados o objetivo geral e os específicos deste estudo.

1.3.1 Objetivo Geral

Desenvolver um sistema de alerta para eventuais falhas em sistemas de refrigeração que atua de maneira proativa, avisando os responsáveis de maneira quase que imediata sobre uma possível falha no sistema de refrigeração.

1.3.2 Objetivos Específicos

- Reduzir o número de incidentes causados por falhas no sistema de refrigeração;

- Facilitar o ambiente de trabalho dos operadores de *Data Center*, bem como aumentar sua produtividade, fazendo com que os mesmos trabalhem com tranquilidade em outros serviços sem se preocupar em fazer inspeções presenciais ao ambiente onde estão as máquinas.

2 MÉTODO DE PESQUISA

Esta pesquisa será realizada em três frentes. Primeiro a partir de um estudo da tecnologia para desenvolvimento de aplicativos Android por meio de materiais disponíveis na internet, tais como: sites especializados, documentação oficial da Google, *blogs* de terceiros, fóruns de desenvolvimento, entre outros arquivos que compõe a pesquisa bibliográfica e documental.

O segundo passo será o aprofundamento do estudo da linguagem de programação Java por intermédio dos mesmos meios eletrônicos descritos no item acima, bem como livros disponíveis em bibliotecas de acervos públicos. Por último, um estudo de microcontroladores e protocolos de rede, por meio de livros técnicos e meios de comunicação eletrônicos (internet). Após adquirir tais conhecimentos, será possível começar o desenvolvimento da placa protótipo que conterá o sistema embarcado, bem como o desenvolvimento do aplicativo que executará nos dispositivos móveis (*smartphones*).

3 FUNDAMENTAÇÃO TEÓRICA

Este trabalho assemelha-se a um estudo apresentado como avaliação final do curso de Tecnologia em Eletrônica com ênfase em Telecomunicações, da UTFPR no ano de 2008, intitulado “Interface de Aquisição e Controle de Dados Aplicada a Medição de Temperatura (IACD)” (BUGALHO, 2008). A pesquisa de Bugalho (2008) possui como característica principal o monitoramento de temperatura de ambientes por meio de um dispositivo sensor acoplado a um microcontrolador, sendo possível o monitoramento da temperatura de forma simples e amigável via interface Web.

Entretanto, o presente trabalho diferencia-se do anterior por fazer uso de dispositivos móveis conectados a um servidor físico. Isso é possível em virtude do amplo desenvolvimento e da grande difusão dos sistemas operacionais para dispositivos móveis alcançados nos últimos anos.

Portanto, foi possível realizar melhorias na gestão dos ambientes monitorados, com o objetivo de facilitar a visão dos ambientes e setores que sofreram o incidente, pois, além de mostrar o endereço de qual ambiente ou setor sofreu a queda de refrigeração, o sistema não precisa de nenhum tipo de monitoramento presente, pois qualquer alteração envia um alerta para o dispositivo móvel do técnico responsável, independentemente de onde ele se encontre. Esse é um diferencial em relação ao trabalho anterior que necessita de alguém monitorando a tela da interface gráfica onde está alocado o servidor que mede a temperatura.

3.1 SISTEMAS EMBARCADOS

O objetivo desta seção é definir o conceito de sistemas embarcados e microcontroladores, além de abordar algumas características das principais arquiteturas de microprocessadores.

Sabe-se que sistemas embarcados são equipamentos de uso específico, que não possuem interface de usuário e geralmente são projetados para não serem programados pelo usuário final, como visto no caso dos computadores de uso genérico. Em um sistema embarcado, de forma geral, faz-se uso de um microcontrolador e de um programa escrito em uma Memória Apenas de Leitura

(*Read Only Memory* – ROM), que é chamado de *firmware*, alguns exemplos são: celulares, televisores, aparelhos de som, máquinas de lavar roupa, brinquedos, câmeras digitais, mp3 players, fornos de micro-ondas, controles remotos e outros aparelhos.

Para Yaghmour (2003), os sistemas embarcados podem ser classificados segundo quatro critérios: (1) conectabilidade, (2) tamanho, (3) requisitos de tempo e (4) interação com o usuário. No primeiro é definido se um sistema embarcado está apto para trabalhar em rede, no segundo, o sistema é classificado como: pequeno, médio e grande, sendo os pequenos caracterizados por Unidades Centrais de Processamento (*Central Processing Unit* – CPU) de baixa potência, um mínimo de 4MB de ROM, os médios por uma CPU de média potência com 32MB ou mais de ROM e os grandes por uma potente CPU e grandes quantidades de Memória de Acesso Aleatório (*Random Access Memory* – RAM), por exemplo, interruptores de telecomunicações e simuladores de voo.

No terceiro ponto, classifica-se o sistema quanto à restrição de tempo. Sistemas rigorosos requerem que o sistema reaja em um prazo pré-definido a fim de evitar catástrofes, como, por exemplo, o travamento de uma máquina de corte em uma fábrica ao detectar um braço humano. Já os sistemas suaves são aqueles em que a resposta em tempo útil não é fundamental. Por último, classifica-se o sistema quanto ao grau de interação com o usuário. Há sistemas como Assistentes Pessoais Digitais (*Personal Digital Assistants* – PDA) e *tablets* que interagem com o usuário, enquanto outros sistemas, como, por exemplo, os sistemas de controle industrial, possuem interação limitada, e há ainda os sistemas como componentes de controle de voo em que não há nenhuma interação direta com os pilotos.

3.2 MICROCONTROLADOR

O microcontrolador difere-se do microprocessador em diversos aspectos, dentre eles, o fato de possuir todos os periféricos como memória, *chipsets*, conversores AD/DA, CPU, dispositivos de entrada e saída integrados em um único componente. Além disso, possui como vantagem o fato de ser pequeno e mais barato que os microprocessadores sendo geralmente dispositivos de baixa potência.

Tanenbaum (2007) refere-se aos microcontroladores como “computadores embutidos em dispositivos que não são vendidos como computadores”.

3.3 PRINCIPAIS ARQUITETURAS DE PROCESSADORES

A seguir estão apresentados três tipos de arquiteturas de processadores: *Complex Instruction Set Computing*[1] (CISC), *Reduced Instruction Set Computing*[2] (RISC) e *Advanced RISC Machine* (ARM).

3.3.1 Arquitetura CISC

A arquitetura CISC é usada em processadores Intel e AMD e possui um grande número de instruções, o que facilita sua programação, mas, por outro lado, torna a execução mais lenta, porque as instruções complexas precisam ser traduzidas, internamente, em instruções mais simples. (GRUPONETCAMPOS, 2011).

3.3.2 Arquitetura RISC

A arquitetura RISC é usada em processadores PowerPC e SPARC, possui menos instruções do que a arquitetura CISC e com isso executa com mais rapidez. (GRUPONETCAMPOS, 2011).

3.3.3 Arquitetura ARM

A arquitetura ARM (*Advanced RISC Machine*) refere-se a um processador de 32 bits muito usado em sistemas embarcados, que possui ainda o modo *Thumb,* que é um segundo *set* de instruções com 16 bits, que ajuda a economizar a memória de programa. Recentemente foi lançado o ARM 8 que possui arquitetura de 64 bits. Amplamente usada na indústria e na informática, seu desenvolvimento se deu com o

[1] Tradução: Computador com Conjunto Complexo de Instruções
[2] Tradução: Computador com Conjunto Reduzido de Instruções

objetivo de obter o melhor desempenho possível, com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia.

Os processadores ARM possuem poucas instruções para programação, o que os tornam versáteis e de manuseio simples. Eles são encontrados em telefones celulares, calculadoras, periféricos de computador, equipamentos encontrados em lojas e supermercados, como leitores de código de barras e também em aplicações industriais. De acordo com Landim (2013), correspondem a grande maioria dos processadores embarcados RISC de 32 bits.

3.3.4 Microcontrolador P89LPC932A1

O P89LPC932A1 é um microcontrolador da fabricante Philips com propriedades compatíveis com o 80C51, e que possui as seguintes características:

- 256 bytes de memória RAM, capacidade para 512 bytes de memória auxiliar;
- 512 bytes na EEPROM em um chip habilitado para serialização de dispositivos, armazenamento de parâmetros de configuração;
- Dois comparadores analógicos com entradas selecionáveis e código de referência;
- 8 KB de memória flash apagável organizada dentro de setores de 1 KB e páginas de 64 bytes. Um único byte habilita qualquer byte(s) para ser usado como armazenamento de dados não volátil;
- Redefinição de baixa tensão (detecção de blecaute) permite um sistema de desligamento não abrupto quando há falha de energia;
- Baixo custo.

A seguir foram detalhadas algumas características do microcontrolador da Philips, o P89LPC932A1 (DATASHEET CATALOG, 2014):

- Sinal de relógio: Diferente do 80C51, que se baseava em um cristal de quartzo e em um ressonador cerâmico externo ligado ao microcontrolador, é possivel escolher a fonte do sinal de relógio e a forma de ser aplicado no microcontrolador e nos periféricos. O próprio microcontrolador possui internamente um oscilador RC com precisão de 2,5%;

- Memória LPC900: Não é possível endereçar memória externa a menos que seja feita uma interface em série ou barramento. A memória interna pode ser repetidamente programada, mesmo com o microcontrolador montado no circuito de aplicação e tambem é possivel escrever o próprio *bootloader,* que é o programa de inicialização do dispositivo;
- Saídas e Entradas: O número de pinos disponíveis para entradas e saídas é igual ao número total de pinos menos os dois usados para a tensão de alimentação. Por exemplo, para o microcontrolador P89LPC932 o número de entradas e saídas disponíveis é de 26 pinos;
- Sistema Supervisor: Não necessita adicionar componentes externos para se obter determinadas funções de supervisão, pois já possui algumas dessas funções como: detector/reset de alimentação ligada, detector de baixa tensão e circuito de guarda (*watchdog*);
- Gestão de energia: Possibilidade de colocar em estado de espera e baixa potência (*Power down*).

A Figura 1 apresenta a pinagem do microcontrolador utilizado:

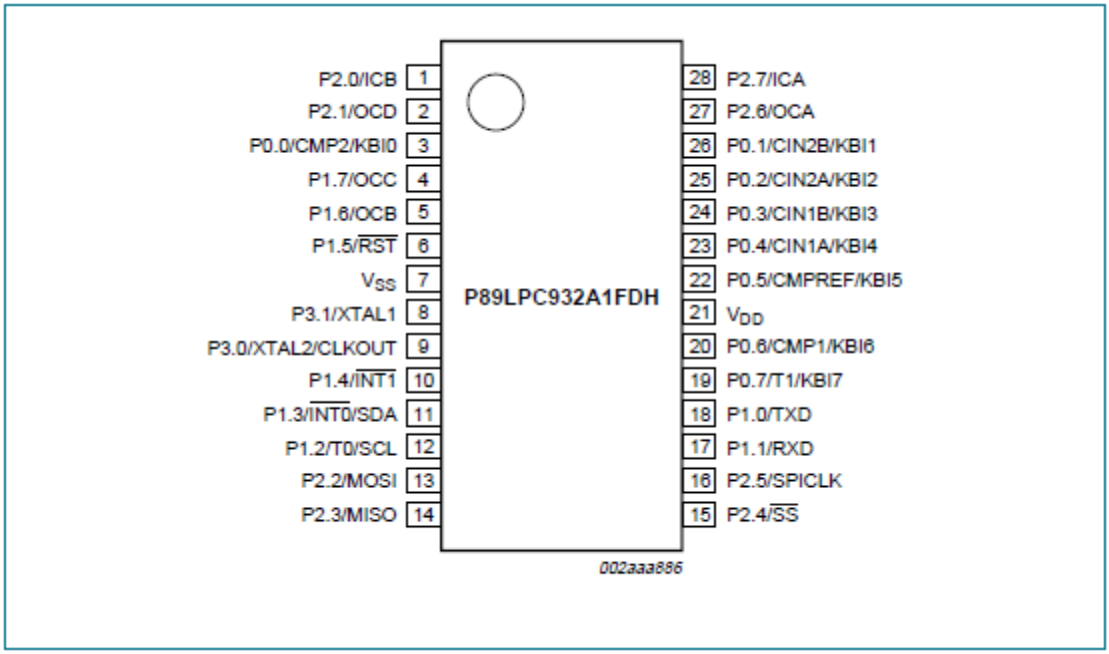

Figura 1 – P89LPC932A1 TSSOP28 pin configuration
Fonte: DATASHEET CATALOG (2014)

3.4 SENSORES

O sensor é um dispositivo eletroeletrônico que responde a estímulos físico-químicos externos transformando-os em sinais elétricos de maneira específica e mensurável analogicamente. Atualmente, os sensores são amplamente utilizados em diversas aplicações desde a medicina até a indústria e os transportes. (STEFFENS, 2006).

3.4.1 Sensor DS18B20

Utilizou-se para a medição de temperatura o sensor DS18B20, que fornece de 9 a 12 bits de precisão. Além disso, possui internamente um conversor A/D, que fornece a temperatura em formato digital por meio do barramento *one-wire* direto para o barramento do microcontrolador. O o*ne-wire* é um sistema de barramento que tem como característica fornecer dados de baixa velocidade, sinalização e sinal único de energia. Tem apenas dois cabos que são dados e GND e dispõe de um capacitor de 800 pF para armazenar carga e alimentar o dispositivo durante os períodos onde o cabo de dados estiver sendo usado para o tráfego de dados (WINGWIT, 2014).

A Figura 2 apresenta o diagrama em blocos do sensor DS18B20.

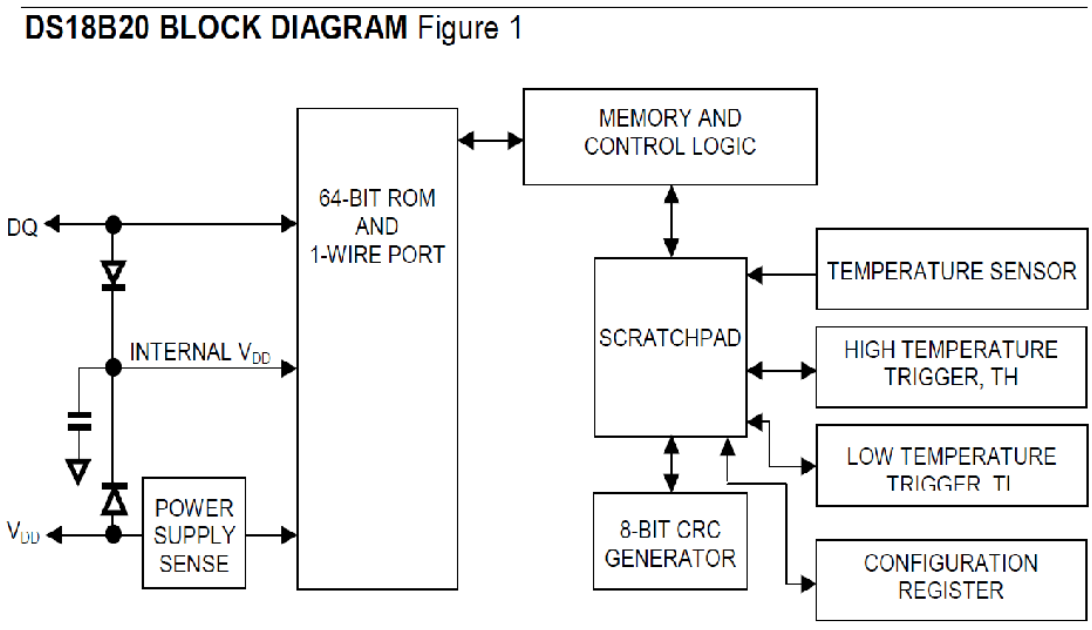

Figura 2 – Diagrama em blocos DS18B20
Fonte: ALLDATASHEET (2014)

Uma vantagem do sensor utilizado é a de não requerer componentes externos. Cada dispositivo tem um único código serial de 64 bits armazenado em uma ROM *onboard*. O código serial de 64 bits é o ID do sensor, cada sensor possui um código único e serve para o endereçamento dos sensores no barramento *one-wire*. Mede temperaturas de -55 °C a +125 °C, possuindo uma precisão de 0,5 °C para temperaturas compreendidas em -10 °C a +85 °C.

3.5 TCP/IP E CONCEITOS DE REDES

Uma rede de computadores é formada por dois ou mais dispositivos capazes de se comunicar entre si, tem como propósito o compartilhamento de informações entre os dispositivos. As redes podem ser classificadas como:

- LAN: é uma rede local situada em um espaço físico não muito grande, geralmente um domicílio, escritório, universidade;
- MAN: utilizada entre duas cidades;
- WAN: utilizada entre pontos intercontinentais.

Para a comunicação entre os dispositivos de uma rede, é necessário um protocolo de comunicação. Esse protocolo é um conjunto de regras e padrões estabelecidos para que possa ocorrer a comunicação eficiente entre duas máquinas. As redes podem ser programadas por *hardware*, *software* ou por uma combinação dos dois. Alguns dos objetivos dos protocolos de comunicação são: detectar erros e falhas ao longo da transmissão, o endereçamento do ponto de origem e de destino, estabelecer conexões e controle de fluxo. (KIOSKEA, 2014).

3.5.1 Protocolo IP

O IP é um protocolo padronizado em setembro de 1981 e associado a um endereço de 32 bits escrito com quatro octetos no formato decimal (exemplo: 192.168.1.7), que permite enviar dados para um determinado dispositivo em uma rede. (DIGITAL4, 2012). Todos os bits definem o endereço de uma máquina, no entanto, é variável o número de bits que irão definir a porção de *hosts* do endereço, visto que uma parte desse endereço (bits mais significativos) indicam a rede e a

outra parte (bits menos significativos) indicam qual é a máquina dentro da rede. (PINTO, 2011). O número de sub-redes e *hosts* é configurado pela máscara de sub-rede e se apresenta em três principais classes (Figura 3):

Classe	Primeiro Octeto	Parte da rede (N) e parte para hosts (H)	Máscara	Nº Redes	Endereços por rede
A	1-127	N.H.H.H	255.0.0.0	**126** (2^7 -2)	**16,777,214** (2^{24} -2)
B	128-191	N.N.H.H	255.255.0.0	**16,382** (2^{14}-2)	**65,534** (2^{16}-2)
C	192-223	N.N.N.H	255.255.255.0	**2,097,150** (2^{21}-2)	**254** (2^8-2)
D	224-239	Multicast	NA	NA	NA
E	240-255	experimental	NA	NA	NA

Figura 3 – Classes de endereço IP
Fonte: PINTO (2011)

A partir da Figura 3 nota-se que a coluna Máscara define a qual classe pertence o endereço IP. Endereços da classe A permitem mais *hosts* na rede, enquanto endereços da classe C permitem mais redes. Cabe ressaltar que na classe A o primeiro bit é 0 (zero), na classe B os primeiros dois bits são 10 (um e zero) e na classe C os primeiros três bits são 110 (um, um e zero), como demonstrado na Figura 4 :

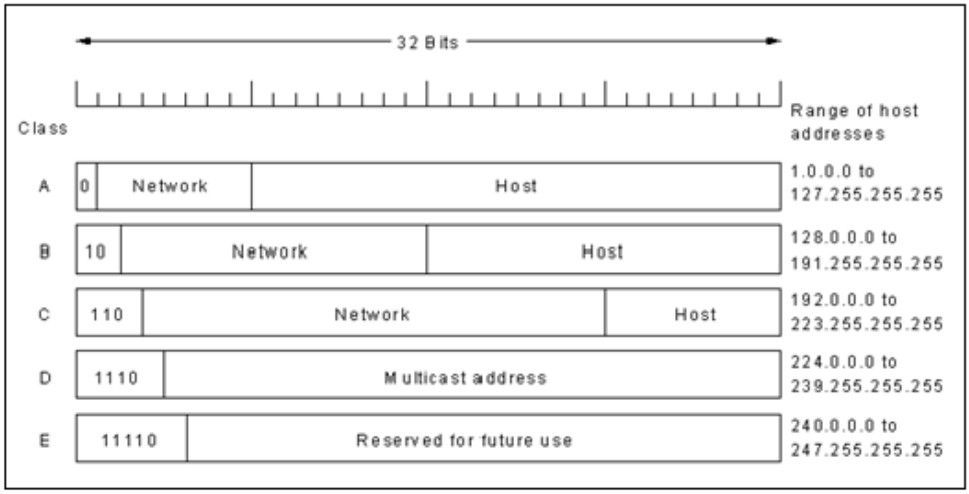

Figura 4 – Bits das classes IP
Fonte: UFCG (1998)

O número IP pode ser do tipo público ou privado. Endereços públicos fazem referência a uma máquina (roteador, computador, etc.) na internet, possuem recursos limitados e atualmente estão escassos na versão IPv4 (o protocolo IPv6 aumentará o intervalo de endereços). Endereços privados são utilizados no mesmo

domínio local e não são encaminháveis diretamente a internet, logo, uma máquina utilizando um IP privado terá que passar por um processo de conversão para um IP público. O processo de conversão de um IP privado para um IP público e vice-versa é conhecido como NAT (*Network Adress Translate*). Por meio do NAT é possível associar vários endereços privados a um único endereço público de saída para a Internet (*NAT Overload ou PAT*). (PINTO, 2011).

No entanto, mesmo mapeando o dispositivo na rede dentro do cabeçalho do protocolo IP, faz-se necessário entregar o pacote de dados para o aplicativo correto. Nesse caso há conexões chamadas de portas lógicas. Dentro do protocolo IP há o número da porta lógica especificando para qual aplicativo o pacote deverá ser entregue. (SIERRA, 2010). Algumas portas lógicas bastantes conhecidas são:

- 22: SSH – *Secure Shell* (protocolo de conexão segura);
- 20: FTP – *File Transfer Protocol* (Protocolo de Transferência de Arquivo);
- 80: *Hipertext Transfer Protocol* (Protocolo de Transferência de Hipertexto);
- 25: SMTP – Porta responsável pelo envio de e-mail;
- 110: POP3 – Porta usada para o recebimento de e-mails;

No desenvolvimento deste projeto foi utilizada a porta 1965, em virtude de essa estar acima da porta 1023, visto que as portas até o número 1023 são utilizadas pelo sistema operacional. As portas permitidas para uso são as que pertencem ao intervalo de 1024 a 65535.

3.5.2 Socket

O *socket* é um objeto que representa uma conexão de rede entre duas máquinas, logo, um elemento de *software* que facilita a vida do programador. Por intermédio dessa estrutura é possível ter uma interface de rede para aplicação, sendo possível ler e gravar bytes como um fluxo de dados, além de esconder os detalhes de baixo nível como: transmissão, tamanho de pacote, retransmissão e etc.

Segue um exemplo de como utilizar o objeto *Socket* em Java.

Socket chatSocket = new Socket (“196.164.1.103”, 4000);

O primeiro argumento dentro dos parênteses indica o endereço IP do servidor e o segundo argumento indica o número da porta lógica que receberá os dados. (SIERRA, 2010).

3.5.3 Protocolo de Transporte (TCP)

O Protocolo de Controle de Transporte (TCP – *Transport Control Protocol)* é responsável por assegurar que os dados chegarão sequencialmente e livres de erros. O TCP “cuidará das portas”, garantindo que os dados de uma aplicação que roda em um *host* A chegue a um *host* B.

Quando um pacote de dados é transmitido pela internet, ele poderá passar por diversos computadores, o que pode acarretar perda ou duplicação dos arquivos. Dessa forma o TCP foi concebido para verificar se o destinatário está pronto para receber os dados, cortar os grandes pacotes de dados e diminuí-los para que o IP os aceite, enumerando os pacotes e, ao recebê-los, verificando se eles chegaram.

Caso os pacotes não cheguem, é enviada uma mensagem reclamando o não recebimento do pacote, que novamente será enviado e, caso o pacote chegue, é enviada uma confirmação para o *host* que está enviando de que os pacotes foram recebidos com sucesso.

O TCP é um protocolo orientado à conexão logo, estabelece-se uma conexão antes do envio dos dados e também no encerramento da transmissão. (KIOSKEA, 2014). Na figura 5 podemos ver o estabelecimento e término de uma sessão TCP:

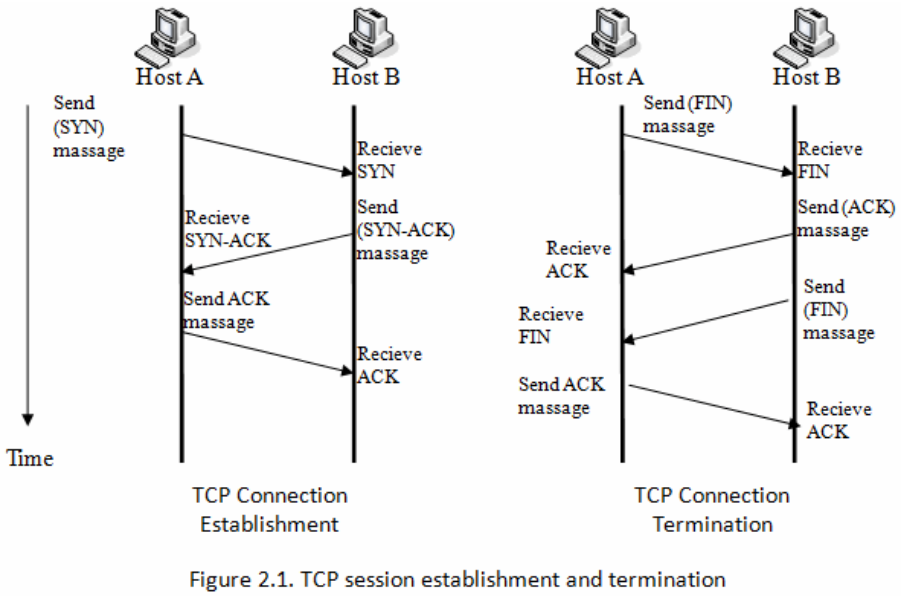

Figura 5 – Estabelecimento e término de sessão TCP
Fonte: MIKROTIK (2013)

3.5.4 Tecnologia 3G

A tecnologia 3G é a terceira geração de tecnologia de telefonia móvel e permite que as operadoras ofereçam um portfólio maior de serviços, pois há uma melhora significativa na eficiência espectral, o que aumenta a capacidade da rede. Atualmente a tecnologia é amplamente utilizada por celulares e outros dispositivos móveis para acesso a informações como notícias, músicas, vídeos, serviços de GPS entre outros. (CÂMARA, 2012).

3.6 LINGUAGEM DE PROGRAMAÇÃO JAVA

A linguagem de programação Java, criada em meados da década de 90, pela empresa Sun Microsystems, possui como características principais:

- A gratuidade com código fonte aberto, sendo uma das linguagens mais utilizadas no mundo;
- O vasto conjunto de bibliotecas, o que possibilita o uso constante e reuso de código, o que facilita e agiliza o desenvolvimento de aplicativos;
- O desenvolvimento e execução em qualquer plataforma, graças aos conceitos de máquina virtual e *bytecode*, onde, após a compilação, é gerado um arquivo intermediário em um formato conhecido como *bytecode*, que não é nem executável e nem arquivo texto, e que será interpretado e executado em qualquer plataforma que tiver uma máquina virtual Java instalada. (MENDES, 2009).

Segundo Mendes (2009, p. 17), a linguagem Java pode ser considerada simples, porque “permite o desenvolvimento em diferentes sistemas operacionais e arquiteturas de hardware, sem que o programador tenha que se preocupar com detalhes de infraestrutura”.

3.7 ANDROID

O Android é um sistema operacional para dispositivos móveis (*tablets*, celulares entre outros), desenvolvido pela aliança *Open Handset Alliance* – que é

uma sociedade entre empresas de telecomunicações e de informática, que tem como principal líder a empresa Google, - e recentemente vem sendo utilizado por vários fabricantes de celulares. O sistema operacional Android possui como características:

- Ter código fonte aberto, o que torna o sistema robusto, uma vez que diversas melhorias são feitas com frequência;
- Não estar preso a nenhum *hardware* específico;
- Utilizar uma versão personalizada do núcleo Linux, voltada para dispositivos;
- Fornecer bibliotecas e Interfaces de Programação de Aplicativos (API – *Application Programming Interface*) completa para a linguagem Java, além de possuir um emulador para simular o celular, permitindo testar as aplicações ainda em ambiente de desenvolvimento. Também é possível conectar um dispositivo móvel real na porta USB do computador e executar os aplicativos diretamente no dispositivo móvel. (CIDRAL, 2011).

4 METODOLOGIA

O protótipo do sistema para detecção de avaria de temperatura em CPD foi desenvolvido com a intenção de enviar alarmes para dispositivos móveis com Android instalado quando a temperatura do ambiente estiver acima da temperatura programada. Para a elaboração do modelo, o diagrama em blocos a seguir (Figura 6) foi idealizado contendo os principais componentes do sistema.

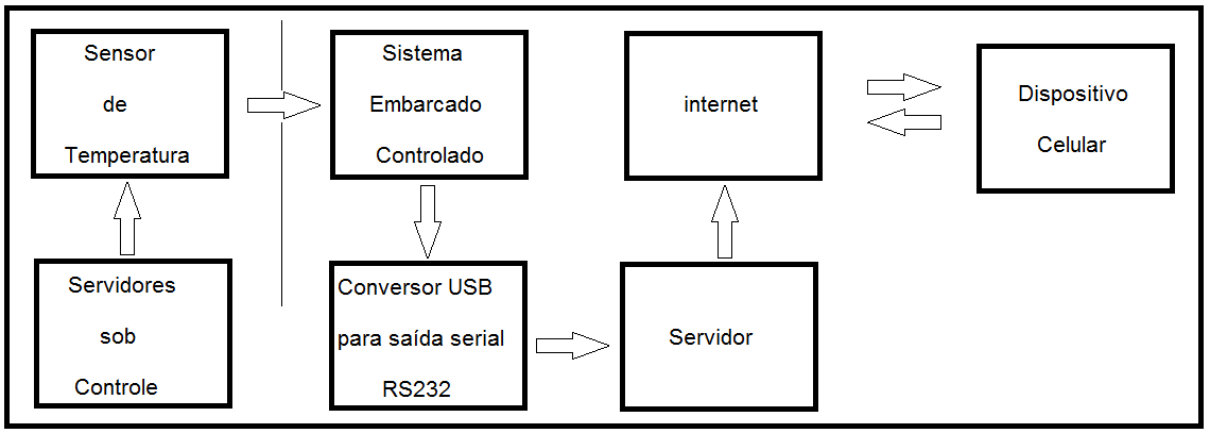

Figura 6 – Diagrama em blocos
Fonte: autoria própria (2014)

Os blocos que constituem o sistema são:

- Servidores sob controle: formado por equipamentos destinados a sofrer a monitoração de temperatura. Vale destacar que foi utilizada a terminologia de “Servidores” para tornar o diagrama mais didático e fácil de entender, uma vez que o sensor pode capturar a temperatura de um sistema de refrigeração (como um ar-condicionado) ou a temperatura ambiente de uma sala;
- Sensor de temperatura: trata-se do sensor DS18B20 descrito anteriormente, que utiliza o protocolo *one-wire* e mede temperaturas de -55 °C à +125 °C;
- Sistema embarcado controlado: formado pela placa de desenvolvimento genérica com o processador P89LPC932A1.

- Conversor de USB para saída Serial RS232: é um cabo que permite instalar uma saída serial RS232 (Porta COM) em um computador ou notebook por meio de uma porta USB, sem a necessidade de abertura do equipamento;
- Servidor: é o computador que receberá as informações de temperatura do sistema embarcado controlado e as enviará para o dispositivo com sistema operacional Android. O servidor irá executar um código Java que abrirá uma porta lógica via *socket* para transmissão dos dados;
- *Internet:* As informações provenientes do Servidor trafegarão via protocolo TCP/IP pela rede mundial de computadores (Internet) até o seu destino (celular cadastrado);
- Dispositivo celular: Dispositivo móvel que estará conectado ao Servidor e receberá os alertas.

4.1 DESCRIÇÃO DO SISTEMA

A seguir estão descritas algumas informações a respeito do sistema desenvolvido para este trabalho.

4.1.1 Aplicação embarcada

Para desenvolver a aplicação embarcada foram empregadas as seguintes ferramentas:

- Placa de Desenvolvimento Genérica com o processador P89LPC932A1 (Figura 7).

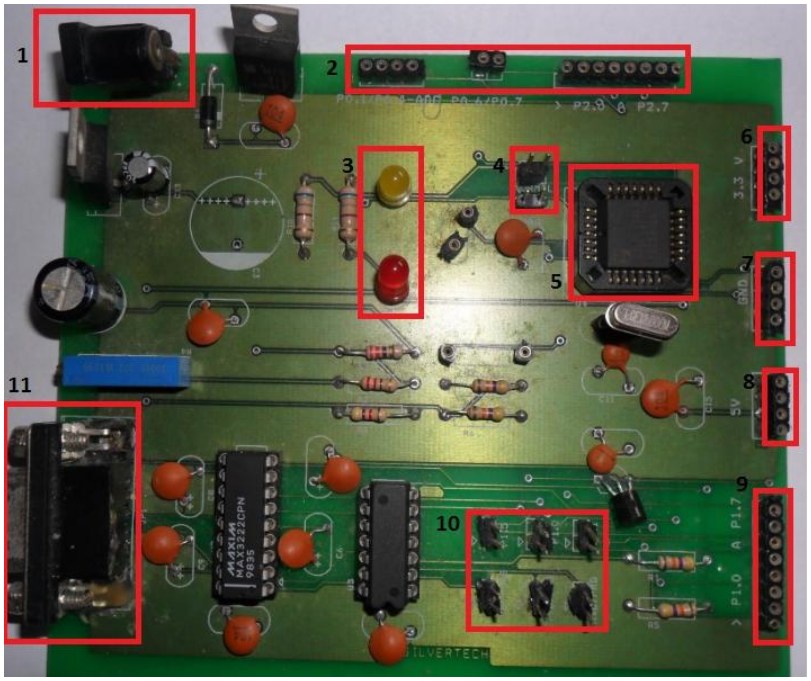

Figura 7 – Placa Microcontrolada desenvolvida pela equipe
Fonte: autoria própria (2014)

Essa placa possui as seguintes propriedades, as quais estão indicadas por números na Figura 7. O diagrama esquemático é mostrado na Figura 8:

1 - Entrada Alimentação 12VDC;

2 - *Ports* do microcontrolador;

3 - *Leds* indicadores de tensão de 5V e 3.3V;

4 - *Jumper* de programação;

5 - Microcontrolador P89LPC932A1;

6 - Pontos para prototipagem 3.3V;

7 - Pontos para prototipagem GND;

8 - Pontos para prototipagem 5V;

9 - *Ports* do microcontrolador;

10 - *Jumpers* para programação ISP e também para acesso à saída serial RS232, e;

11 - Conector DB9.

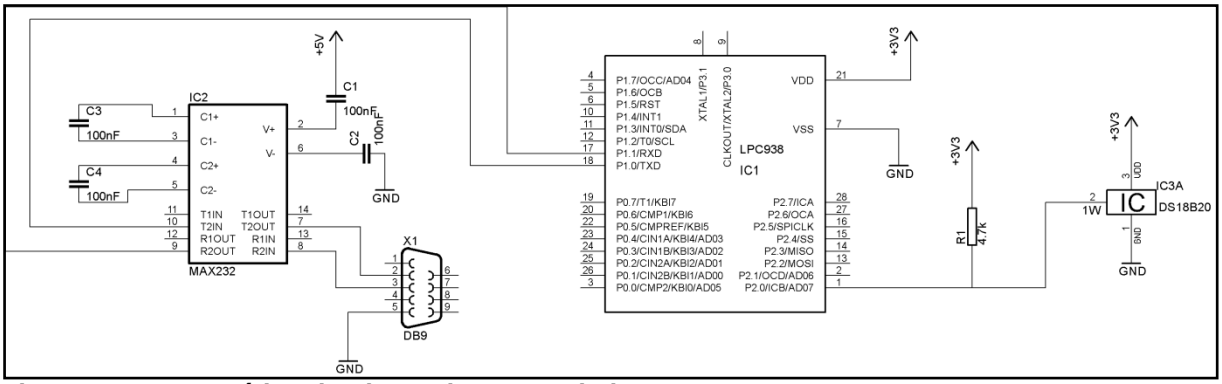

Figura 8 – Esquemático da Placa Microcontrolada
Fonte: autoria própria (2014)

O *software* embarcado tem como principal objetivo obter os dados do sensor DS18B20, por meio do protocolo *one-wire*, converter os dados recebidos e enviá-los via UART RS232 para o computador que estará executando o aplicativo Java. Uma das vantagens de utilizar microcontroladores NXP (fabricante Philips) deve-se ao fato de ser possível configurar os periféricos do microcontrolador via interface web, já gerando a configuração dos periféricos em linguagem C, o que diminui o tempo de desenvolvimento, bastando adicionar o código de configuração, ao programa embarcado (*firmware*). Na imagem a seguir (Figura 9) é possível visualizar a página da internet onde são geradas as rotinas de configuração dos periféricos do microcontrolador P89LPC932.

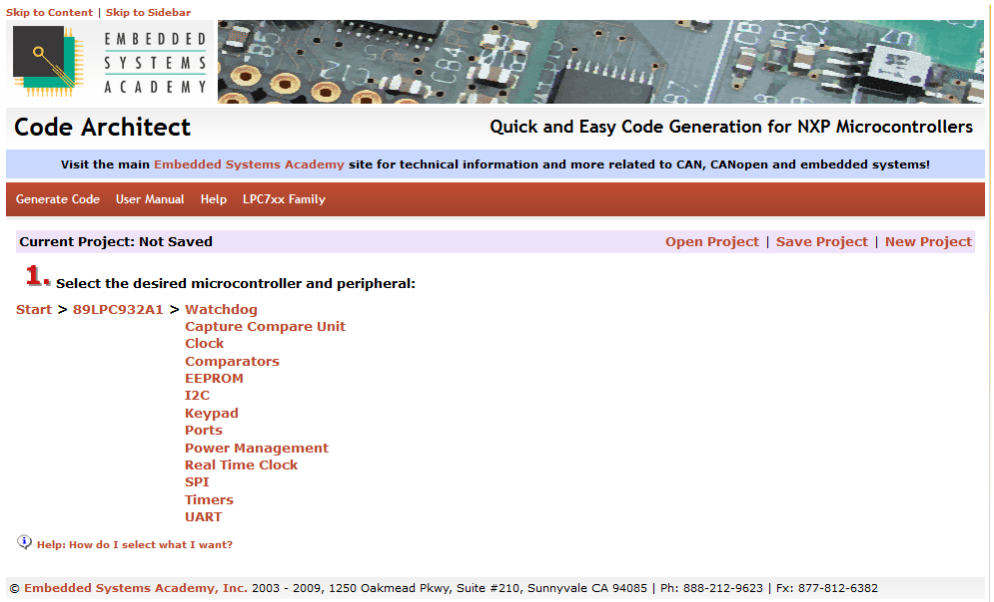

Figura 9 – Code Architect Interface Web
Fonte: CODE ARCHITECH (2014)

Um exemplo de código de configuração gerado pela página Code Architect pode ser visto na Figura 10, onde o Timer 1 está configurado para gerar uma interrupção a cada 10 milissegundos (ms).

```
/*************************************************************************
DESC:    Initializes the Real Time Clock and starts it running
         Generates an interrupt every 10.005ms
         Clock source: CPU Clock at 12.0 MHz
RETURNS: Nothing
CAUTION: Set EA to 1 to enable interrupts
*************************************************************************/
void rtc_init
  (
  void
  )
{
  // set reload value
  // count frequency = clock source / 128 / RTCH,RTCL
  RTCH = 0x03;
  RTCL = 0xAA;

  // select CPU Clock, enable interrupt source
  RTCCON = 0x62;

  // initialize watchdog/real time clock interrupt
  watchdogrtc_isrinit();

  // start real time clock
  RTCCON |= 0x01;
}
```

Figura 10 – Código Fonte em C pré-configurado
Fonte: autoria própria (2014)

Para a gravação do *software* embarcado dentro da memória do microcontrolador foi utilizado o programa *Flash Magic* (Figura 11) (FLASH MAGIC TOOLS, 2014), fornecido gratuitamente pela NXP para programação da maioria de seus microcontroladores via ISP. Essas memórias Flash são utilizadas para gravar a programação do microcontrolador por serem rápidas e de baixo custo.

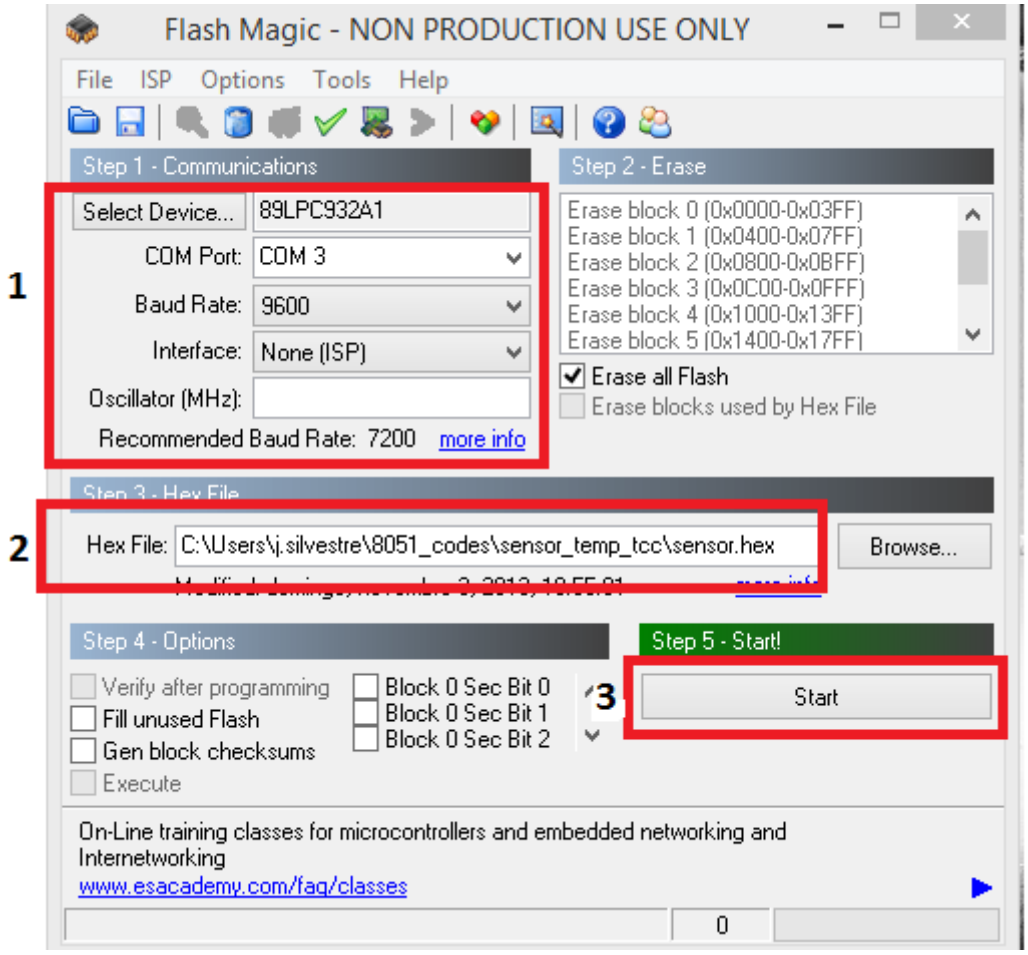

Figura 11 – Flash Magic Interface
Fonte: autoria própria (2014)

Os principais campos dessa interface são (Figura 11):

1) Seleção do dispositivo, velocidade da porta serial e interface ISP;
2) Seleção do arquivo .hex a ser gravado na flash;
3) *Start* para iniciar a gravação do *firmware* na memória flash.

Para gravar o P89LPC932A1 com o Flash Magic é necessário um circuito lógico (Figura 12) para fazer o microcontrolador entrar em modo de configuração ISP e então programar o chip. Tal circuito já está disponível na placa de desenvolvimento utilizada.

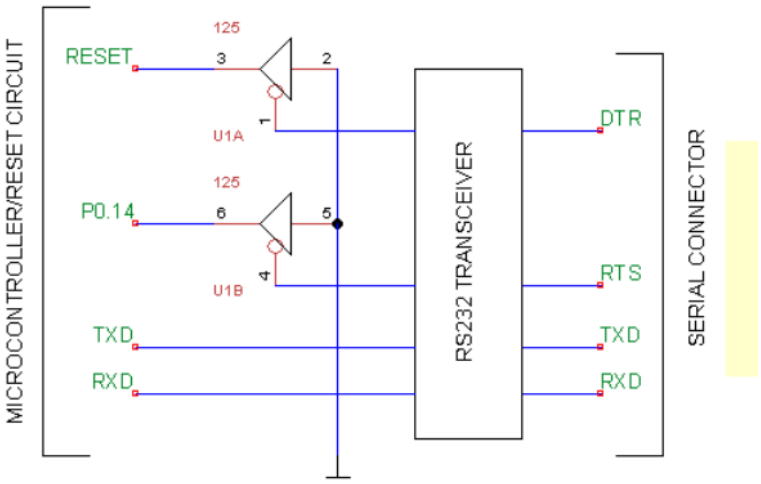

Figura 12 – Circuito lógico
Fonte: autoria própria (2014)

Para efetuar a gravação do *firmware* são necessários: a placa de desenvolvimento genérica com os "*jumpers*" ajustados corretamente, um cabo padrão serial RS232 e o *software* Flash Magic.

4.1.2 Firmware

O desenvolvimento do *firmware* do microcontrolador foi produzido em linguagem C, utilizando a IDE KEIL Uvision 4 para escrever e compilar o código fonte. O código fonte em C está disponível nos apêndices deste trabalho. (APÊNDICE A e B).

4.1.3 Interface gráfica e servidor

O código do servidor foi implementado em linguagem de programação Java, utilizando a IDE Netbeans. De forma básica, o que o servidor faz é executar um *loop* onde um *socket* é aberto e fica à espera de uma conexão externa (endereço IP do cliente). No dispositivo móvel o cliente deverá entrar com o IP do servidor para efetuar a conexão.

Na Figura 13 está apresentada a interface que rodará no Servidor Java.

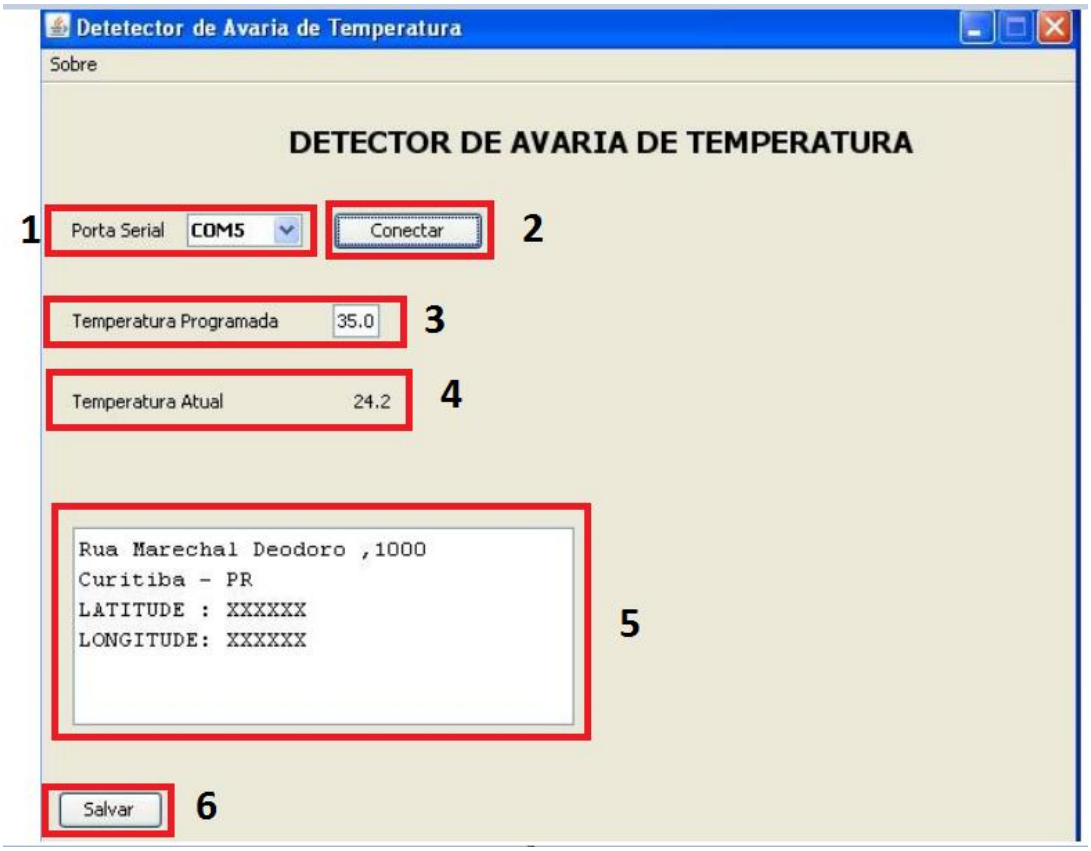

Figura 13 – Interface de controle de temperatura
Fonte: autoria própria (2014)

Descrição da interface:

1. Campo para selecionar a porta serial que está conectada ao sistema embarcado que lê a temperatura;

2. Botão para conectar a porta serial ao sistema embarcado;

3. Temperatura programada;

4. Temperatura atual do sistema embarcado (temperatura real);

5. Texto que será enviado quando a temperatura real for maior que a temperatura programada;

6. Botão para salvar as configurações (temperatura programada e mensagem).

4.1.4 Aplicativo cliente no dispositivo Android

Para a criação do aplicativo cliente que executa nos dispositivos móveis, foi utilizada a IDE Eclipse (*Interface Development Environment*) juntamente com o

plugin ADT (*Android Developer Tools*), que são atualmente as ferramentas mais populares para criar e testar aplicativos Android.

Para testar o aplicativo, foi criada uma AVD (*Android Virtual Device*) que emula o dispositivo móvel. Já para fazer a conexão serial com o sistema embarcado, foi utilizada a API rxtx, que é uma biblioteca de terceiros, disponibilizada gratuitamente. No arquivo de download, há um tutorial explicando como configurá-la para o Java.

Na figura 14, temos a interface gráfica que executará nos dispositivos móveis, no caso celulares.

Figura 14 – Android virtual device
Fonte: autoria própria (2014)

Descrição da interface:

1. Campo para digitar o IP do servidor;
2. Botão para se conectar ao servidor;
3. Mensagem informando que o dispositivo está conectado ao Servidor;
4. Mensagens de alarme.

5 RESULTADOS E DISCUSSÕES

A fim de avaliar o sistema desenvolvido, os testes foram divididos em duas fases. A primeira com teste do sensor de temperatura, pois é um item crucial para a credibilidade do sistema e a segunda com testes de conectividade. Cada teste foi apresentado em seção específica a seguir.

5.1 TESTE DO SENSOR DE TEMPERATURA

No protótipo implementado não foi preciso realizar ajustes de *offset* do sensor, pois o modelo está lendo apenas a temperatura ambiente. Caso este projeto fosse para controlar a temperatura de um forno, por exemplo, então seria necessário implementar o ajuste de *offset*.

No teste do sensor DS18b20 a temperatura ambiente, lida pelo sensor, foi comparada a de um termômetro de mercúrio. Os resultados estão demonstrados na Quadro 1.

HORA	DS18B20	TERMOMETRO DE MERCURIO	DIFERENÇA
09:40	7.1 ºC	7.0 ºC	0.1 ºC
09:54	10.0 ºC	10.0 ºC	0.0 ºC
10:15	10.2 ºC	10.0 ºC	0.2 ºC
11:00	13.5 ºC	13.2 ºC	0.3 ºC
12:00	16.8 ºC	17.0 ºC	0.2 ºC
14:00	18.5 º C	18.5 ºC	0.0 ºC
15:00	20.2 ºC	20.2 ºC	0.0 ºC
16:00	19.8 ºC	19.8 ºC	0.0 ºC
17:00	18.3 ºC	18.3 ºC	0.0 ºC
18:00	17.0 ºC	17.0 ºC	0.0 ºC

Quadro 1 – Monitoramento de temperatura
Fonte: autoria própria (2014)

Os elementos do Quadro 1 estão demonstrados graficamente a seguir (Gráfico 1).

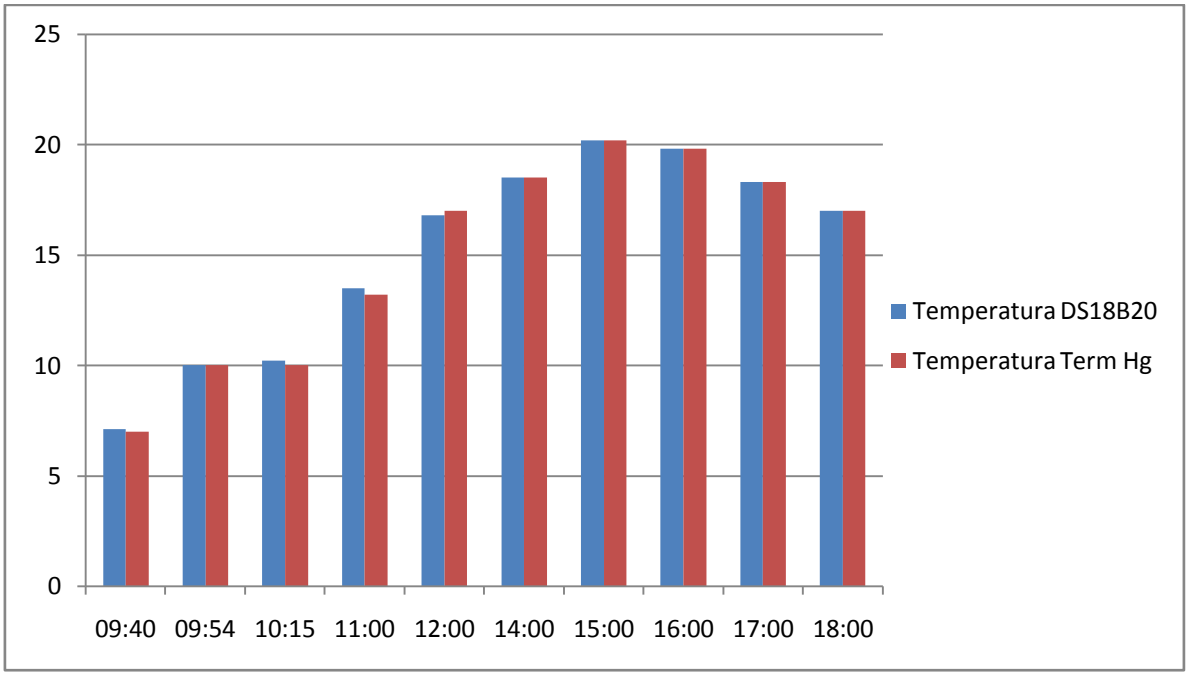

Gráfico 1 – Comparativo de Temperatura
Fonte: autoria própria (2014)

Por meio da análise da Tabela 1, é possível observar que não houve diferença significativa de temperatura entre o termômetro de mercúrio e o sensor DS18b20. Portanto, pode concluir-se que, desse modo, em todas as aplicações práticas de aumento de temperatura em um ambiente de *Data Center,* o sensor desempenhará a contento o seu papel na medição dos valores para alarme.

5.2 TESTES DE CONECTIVIDADE

Todos os testes efetuados tiveram como conexão padrão uma rede Wi-Fi, os quais tiveram os testes de comunicação realizados via interface Unix e uma porta, onde foi emulado com sucesso o sistema embarcado/servidor. Não foram realizados testes empregando a rede 3G, pois seria necessário configurar um servidor DNS (*Domain Name System*), o que iria encarecer o projeto, visto que o dispositivo celular e o servidor estariam fora da mesma rede. Sugere-se que tal funcionalidade possa ser explorada em trabalhos futuros.

Como mencionado anteriormente, o DS18b20 possui um ID de 64 bits que permite com um único barramento *one-wire* ler a temperatura de determinado sensor, permitindo com um único sistema embarcado monitorar vários ambientes

simultaneamente. Neste projeto essa funcionalidade não foi explorada, o que permite a outros alunos oportunidade de estudá-los mais profundamente.

A fim de ilustrar um barramento *one-wire* com vários sensores, segue a Figura 15.

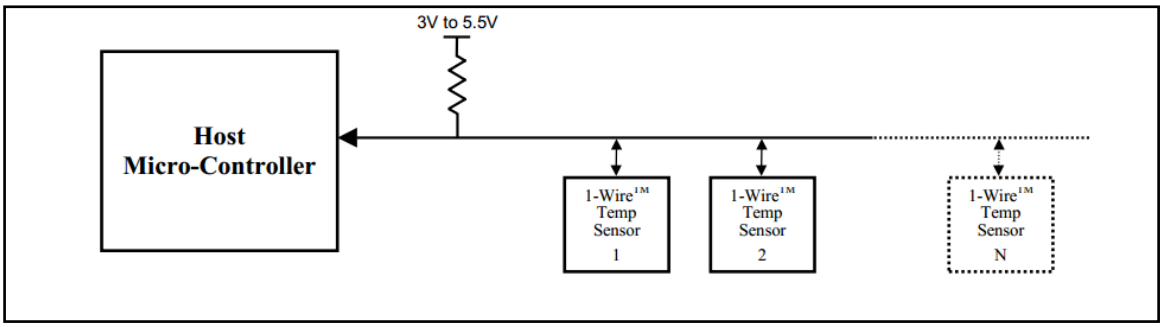

Figura 15 – Barramento one-wire com múltiplos sensores
Fonte: autoria própria (2014)

6 CONSIDERAÇÕES FINAIS

Por meio do desenvolvimento deste projeto pode-se concluir que a utilização de microcontroladores do tipo NXP da família Philips e de tecnologias com códigos abertos (*open-source),* como a utilizada para o desenvolvimento do aplicativo para celular Android deste estudo, são formas versáteis, práticas e baratas para implantação de projetos que necessitem de automação, monitoramento remoto e medição de grandezas físicas. Por intermédio delas é possível implementar várias soluções tanto de caráter comercial como de problemas do cotidiano.

Quanto ao uso do microcontrolador P89LPC932A1, obteve-se a vantagem de não precisar configurar manualmente os periféricos do microcontrolador devido ao fato de ser possível configurar parte dos periféricos via interface Web. Além disso, não foi necessária a conversão da programação em C para a programação em nível de máquina Assembly.

Na utilização da tecnologia Google Android para o desenvolvimento de aplicativos para dispositivos móveis, pode-se aprender muito sobre um mercado novo e ainda em expansão, o qual permite inúmeras possibilidades de programação. Além disso, o estudo permitiu consolidar os conhecimentos de eletrônica e programação de computadores, integrando-os de forma prática na resolução de um problema real, de forma que gerou grande desenvolvimento pessoal e profissional.

REFERÊNCIAS

ALLDATASHEET. **DS18B20-PAR Datasheet (PDF) - Maxim Integrated Products.** Disponível em: <http://www.alldatasheet.com/datasheet-pdf/pdf/108957/MAXIM/DS18B20-PAR.html>. Acesso em: 29 jul. 2013.

BUGALHO, A.; SANTOS, D.J.; **Interface de Aquisição e Controle de Dados Aplicada a Medição de Temperatura** (IACD). UTFPR. Curitiba, Paraná, Brasil, 2008.

CÂMARA, Marlon. **O que é internet 3G?** [S.l], 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2012/01/o-que-e-internet-3g.html>. Acesso em: 18 set. 2013.

CIDRAL, Beline. **Afinal, o que é Android?** Disponível em: <http://www.techtudo.com.br/artigos/noticia/2011/01/afinal-o-que-e-android.html> Acesso em: 03 ago. 2014.

CODE ARCHITECT. **Quick and Easy Code Generation for NXP Microcontrollers, 2014.** Disponível em: <http://www.codearchitect.org/nxp/v2/>. Acesso em: 17 jul. 2014.

DATASHEET CATALOG. **P89LPC932A1 Datasheet**. Disponível em: <http://pdf.datasheetcatalog.net/datasheet/philips/P89LPC932A1_2.pdf>. Acesso em: 29 jul. 2013.

DEITEL, Paul J.. **Android para programadores** (1ª Ed.). Bookman Editora Ltda., 2012.

DIGITAL4. **O que é endereço IP ou internet protocol.html**. Disponível em: <http://www.digital4.net.br/2012/12/o-que-e-endereco-ip-ou-internet-protocol.html>. Acesso em: 06 jan. 2014.

FLASH MAGIC TOOL. **Embedded Systems Academy**. Disponível em: <http://www.flashmagictool.com/>. Acesso em: 22 jul. 2014.

GRUPONETCAMPOS. **Arquitetura RISC e CISC:** qual a diferença. Disponível em: <http://www.gruponetcampos.com.br/2011/03/arquitetura-cisc-e-risc-qual-diferenca/>. Acesso em: 17 jul. 2014.

KIOSKEA. **O protocolo TCP**. Disponível em: <http://pt.kioskea.net/contents/284-o-protocolo-tcp>. Acesso em: 17 jul. 2014.

LANDIM, Wikerson. **Por que os dispositivos ARM podem mudar o rumo dos dispositivos eletrônicos?** Disponível em: <http://www.tecmundo.com.br/qualcomm/7708-por-que-os-processadores-arm-podem-mudar-o-rumo-dos-dispositivos-eletronicos-.htm>. Acesso em: 17 jul. 2014

MENDES, Douglas Rocha. **Programação Java com ênfase em Orientação a Objetos** (1ª Ed.). Novatec Editora Ltda, 2009.

MIKROTIK. **Manual:** Connection oriented communication (TCP/IP). Disponível em: <http://wiki.mikrotik.com/wiki/Manual:Connection_oriented_communication_%28TCP/IP%29>. Acesso em: 13 mai. 2014.

ORACLE. **Java SE Development Kit 7 Downloads.** Disponível em: <http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html?ssSourceSiteId=otnpt>. Acesso em: 29 jul. 2013.

PINTO, Pedro. **Redes – Classes de endereços IP, sabe quais são**. Disponível em: <http://pplware.sapo.pt/tutoriais/networking/classes-de-endereos-ip-sabe-quais-so/>. Acesso em: 17 abr. 2014.

SIERRA, Kathy; BATES, Bert. **Use a Cabeça Java**. Editora Alta Books, Rio de Janeiro, 2010.

STEFFENS, César Augusto. **O que são sensores**. Disponível em: <http://www.if.ufrgs.br/mpef/mef004/20061/Cesar/SENSORES-Definicao.html>. Acesso em: 03 ago. 2014.

TANENBAUM, A. S.. **Organização Estruturada de Computadores** (5ª Ed.). Pearson Prentice-Hall, 2007.

UFCG, Jacques. **ENDEREÇAMENTO IP**. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/1998.1/ir/itcpip/itcpip9.htm>. Acesso em: 03 ago. 2014

WINGWIT. **1-Wire Bus Protocol**. Disponível em: <http://pt.wingwit.com/Ferragens/networkequipment/46083.html#.U967TPldU_U>. Acesso em: 03 ago. 2014.

YAGHMOUR, Karim. **Building Embedded Linux Systems**. Califórnia: O'Reilly, 2003.

APÊNDICE A – SOFTWARE EMBARCADO DESENVOLVIDO

Código Fonte ds18b20.c

```
#include <REG938.H>
#include "ds1820.h"
#include "uart.h"
sbit DQ = P2^0; // pino do DS1820 //
unsigned char bdata temp_lsb;
unsigned char bdata temp_msb;
float temp_tot;
sbit lsb0 = temp_lsb^0;
sbit lsb1 = temp_lsb^1;
sbit lsb2 = temp_lsb^2;
sbit lsb3 = temp_lsb^3;
sbit lsb4 = temp_lsb^4;
sbit lsb5 = temp_lsb^5;
sbit lsb6 = temp_lsb^6;
sbit lsb7 = temp_lsb^7;
sbit msb0 = temp_msb^0;
sbit msb1 = temp_msb^1;
sbit msb2 = temp_msb^2;

/*********************************************************************************************/
/*                DELAY 480 us  for DS1820                                                   */
/*                    Cristal de 10M                                                         */
/*********************************************************************************************/
void delay_480us(void) // timer zero
{
  TF0 =0;
  ET0 = 0; //disable interrupt timer 0
          // timer values
  TH0 = 0xF6;
  TL0 = 0xA0;
```

```
  TR0 =1; // liga timer 0
 while(!TF0){} // espera overflow//
 TR0 = 0; //desliga timer 0
 TF0 =0;

 }

/*****************************************************************************************/
/*                DELAY 15 us  for DS1820                                                */
/*                    Cristal de 10M                                                     */
/*****************************************************************************************/

void delay_15us(void)
{
 TF0 =0;
  ET0 = 0; //disable interrupt timer 0
  // timer values
TH0 = 0xFF;
  TL0 = 0xB5;

  TR0 =1; // liga timer 0
 while(!TF0){} // espera overflow//
 TR0 = 0; //desliga timer 0
 TF0 =0;
 }

/*****************************************************************************************/
/*                DELAY 70 us  for DS1820                                                */
/*                    Cristal de 10M                                                     */
/*****************************************************************************************/
void delay_70us(void)
{
  TF0 =0;
```

```
 ET0 = 0; //disable interrupt timer 0
          // timer values
 TH0 = 0xFE; // valores para 70us
 TL0 = 0xA2; // cristal 10M
 TR0 =1;
 while(!TF0){} // espera overflow//
 TR0 = 0;
 TF0 =0;
}

/*****************************************************************************************/
/*      OW_RESET - performs a reset on the one-wire bus and                              */
/*         returns the presence detect. Reset is 480us, so delay                         */
//                      and 70us -    for
DS1820                                  */                                        */
/*****************************************************************************************/

unsigned char ow_reset(void)
{
unsigned char presence;
DQ = 0; //pull DQ line low
delay_480us(); // leave it low for 480us
DQ = 1; // allow line to return high
delay_70us();
presence = DQ; // get presence signal
delay_410us(); // wait for end of timeslot
return(presence); // presence signal returned
            } // 0=presence, 1 = no part


/*****************************************************************************************/
/*         READ_BIT - reads a bit from the one-wire bus. The delay                       */
/*     required for a read is 15us, so the DELAY routine won't work.                     */
```

```
/*     We put our own delay function in this routine in the form of a                    */
/*                    FOR DS1820                                                          */
/*************************************************************************************/
unsigned char read_bit(void)
{

DQ = 0; // pull DQ low to start timeslot
DQ = 1; // then return high
delay_15us();
return(DQ); // return value of DQ line
  }


/*************************************************************************************/
/*          WRITE_BIT - writes a bit to the one-wire bus, passed in bitval.          */
/*************************************************************************************/
void write_bit(char bitval)
{
DQ = 0; // pull DQ low to start timeslot
if(bitval==1) DQ =1; // return DQ high if write 1
delay_104us();
DQ = 1;
}// Delay provides 16us per loop, plus 24us. Therefore delay(5) = 104us


/*************************************************************************************/
/*              READ_BYTE - reads a byte from the one-wire bus.                      */
/*                          for DS1820                                               */
/*************************************************************************************/
unsigned char read_byte(void)
{
unsigned char i;
unsigned char value = 0;
```

```
for (i=0;i<8;i++)
{
if(read_bit()) value|=0x01<<i; // reads byte in, one byte at a time and then
                        // shifts it left
delay_104us();
delay_15us(); // wait for rest of timeslot
}
return(value);
          }

/*********************************************************************************************/
/*        WRITE_BYTE - writes a byte to the one-wire bus.                                    */
/*                    for ds1820                                                             */
/*********************************************************************************************/
void write_byte(char val)
{
unsigned char i;
unsigned char temp;
for (i=0; i<8; i++) // writes byte, one bit at a time
{
temp = val>>i; // shifts val right 'i' spaces
temp &= 0x01; // copy that bit to temp
write_bit(temp); // write bit in temp into
}
delay_104us();
 }
/*********************************************************************************************/
/*                Read Temperature do DS1820                                                 */
/*********************************************************************************************/

float Read_Temperature(void)
{
char valor1[] ="{      }\r\n";
```

```
char get[10];
unsigned char presence;
int k;

presence = ow_reset();
if(presence) // se nao encontrar sensor executa o if...
{

return 1000;  //erro de sensor
            //vai ser interpretado na funcao main
}
write_byte(0xCC); //Skip ROM
write_byte(0x44); // Start Conversion
delay_104us();
ow_reset();
write_byte(0xCC); // Skip ROM
write_byte(0xBE); // Read Scratch Pad
for (k=0;k<9;k++){get[k]=read_byte();}
temp_msb = get[1]; // Sign byte + lsbit
temp_lsb = get[0]; // Temp data plus lsb]
if(temp_msb <=0x80)
{

  calcula_temp();
                }
else
  if(temp_msb >= 0x80)
  {
 calcula_temp();
temp_lsb = (~ temp_lsb) + 1;
   temp_msb = ~temp_msb;
    calcula_temp();
    temp_tot = (temp_tot * -1);
```

```
                        }

return temp_tot;
}
void calcula_temp(void)

{
unsigned char b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10;

b0 = lsb0;
b1 = lsb1;
b2 = lsb2;
b3 = lsb3;
b4 = lsb4;
b5 = lsb5;
b6 = lsb6;
b7 = lsb7;
b8 = msb0;
b9 = msb1;
b10= msb2;

temp_tot=
(b0*0.0625)+(b1*0.125)+(b2*0.25)+(b3*0.5)+(b4*1)+(b5*2)+(b6*4)+(b7*8)+
(b8*16)+(b9*32)+(b10*64);

}

void delay_410us(void)
{
  TF0 =0;
  ET0 = 0; //disable interrupt timer 0
          // timer values
```

```
// timer values
 TH0 = 0xF7;
 TL0 = 0xFE;
  TR0 =1; // liga timer 0
 while(!TF0){} // espera overflow//
 TR0 = 0; //desliga timer 0
 TF0 =0;
// ET0 = 1;  // enable interrupt timer 0//

 }

void delay_104us(void)
{
  TF0 =0;
  ET0 = 0; //disable interrupt timer 0
           // timer values
  // timer values
 TH0 = 0xFD;
 TL0 = 0xF8;
TR0 =1; // liga timer 0
 while(!TF0){} // espera overflow//
 TR0 = 0; //desliga timer 0
 TF0 =0;
// ET0 = 1;  // enable interrupt timer 0//

 }
```

```
#include <REG938.H>
#include "uart.h"

//*****************************************************************************//
//            ROTINA DE INTERRUPCAO                                            //
//   9600 baud, 8 data bits, 1 stop bit and no flow control                    //
//*****************************************************************************//
void UART_ISR(void) interrupt 4
{
RI = 0; // clear receive interrupt flag

//apenas reseta o flag de recepcao da serial

            }


/*****************************************************************************/
/*              CONFIGURA PINOS  e registradores DA SERIAL
*/
/*****************************************************************************/
void init_uart(void) //
{
SCON=0x50; // select BRG as UART Baud Rate Gen
SSTAT=0x60; // separate Rx / Tx interrupts
BRGR0=0xF0; // setup BRG for 9600 baud @ 7.373MHz internal RC oscillator
BRGR1=0x02;
BRGCON = 0x03;     // enable BRG
```

```
ES = 1; // enable UART interrupt
EA = 1;         // enable ALL interrupts

}



/*****************************************************************************/
/*                      PrinString                                           */
/*    Envia caracteres de uma string para a funcao Out_Char                  */
/*****************************************************************************/

void PrintString( char *s)
{
while (*s) //enquanto caracter permanece no lado //
{
if (*s == '\n')
ua_outchar('\r');
ua_outchar(*s);
s++;
}
}
/*****************************************************************************/
/*              ua_outchar                                                   */
/* envia dados para dispositivo escolhido -> neste caso serial               */
/*****************************************************************************/
void ua_outchar(char c)
{

SBUF = c;
TI = 0;
while (!TI);
}
```

```
#include <REG938.H>
#include<absacc.h>
#include<stdio.h>
#include "ds1820.h"
#include "uart.h"
void ports_init(void);
void delay1s(void);
char valor[] ="           ";
// o"{" e "}" fazem parte do protocolo de comunicacao
// tudo que estiver entre esses dois tokens serao  decodificados
// pela a aplicacao do PC //

void main(void){
float temperatura;

ports_init(); // inicializa ports
TMOD &= 0xF0;  // inicializa timer zero
TMOD |= 0x01;  // 16 bits periodico overflow sem interrupcao
TAMOD &= 0xFE;
init_uart(); // inicializa uart
PrintString("DS18B20");

while(1){
temperatura =  Read_Temperature();
if(temperatura == 1000)
```

```
PrintString("{ Erro Sensor } \n");
else{
sprintf (valor, "%5.1f",(float) temperatura);
PrintString("{");
PrintString(valor); // imprime o valor da temperatura na serial
PrintString("}");
delay1s();
}
}
}


void ports_init(void) // inicializa pinos da porta serial
{                     // inicializa pino que vai receber os dados do sensor

 P1M1 &= 0xFC;
 P1M1 |= 0x20;
 P1M2 &= 0xDF;
 P1M2 |= 0x03;
 P2M1 &= 0xFE;
 P2M2 &= 0x01;

} // ports_in

void delay1s(void)
{

 RTCH = 0xFF;
 RTCL = 0xFF;
 RTCCON = 0x60; // select CPU Clock
 RTCCON |= 0x01; // start rtc

while(1){
```

```
  if (RTCCON & 0x80) //wait overflow rtc
     {
      RTCCON &= ~0x80; // clear flag
       RTCCON &= ~0x01; // stop the real time clock
        break;
              }
               }
}
```

APÊNDICE B – PROGRAMA ANDROID DESENVOLVIDO

```
package projeto.tcc;
import java.io.*;
import java.net.*;
import java.util.*;


public class Servidor
{
   ArrayList clientOutputStreams;
   NewJframe frame = NewJframe.getInstance();
   public class ThreadRecebeDados implements Runnable {
      BufferedReader reader;
      Socket sock;

      public ThreadRecebeDados(Socket clientSOcket) {
         try {
            sock = clientSOcket;
            InputStreamReader isReader = new
InputStreamReader(sock.getInputStream());
            reader = new BufferedReader(isReader);

         } catch (Exception ex) { ex.printStackTrace(); }
      }

      public void run() { //  se receber mensagem "teste"
         String message;
         try {
            while ((message = reader.readLine()) != null) {
               System.out.println("read " + message + " " + frame.getDataHora());
               if(message.contains("teste"))
               ResponderClientes(frame.getResposta());
            }
```

```
        } catch (Exception ex) {

            fecharSocket();
            ex.printStackTrace(); }
    }

   private void  fecharSocket(){
     try{
     sock.close();
     }catch(Exception e){}

   }
}



public void iniciarServidor() {
    clientOutputStreams = new ArrayList();
    try {
       ServerSocket serverSock = new ServerSocket(1965);
       while(true) {
          Socket clientSocket = serverSock.accept();
          PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
          clientOutputStreams.add(writer);

          Thread t = new Thread(new ThreadRecebeDados(clientSocket));
          t.start();
          ResponderClientes("Conectado : " +  frame.getDataHora());
          System.out.println("Conectado : " + frame.getDataHora());
       }
    } catch (Exception ex) { ex.printStackTrace(); }
}
```

```
    public void ResponderClientes(String message) {
        Iterator it = clientOutputStreams.iterator();
        while (it.hasNext()) {
            try {
                PrintWriter writer = (PrintWriter) it.next();
                writer.println(message);
                writer.flush();
            } catch (Exception ex) {}// ex.printStackTrace(); }
        }
    }
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * NewJframe.java
 *
 * Created on 27/06/2011, 10:50:32
 */

/**
 *
 * @author jose.araujo
 *
 */
package projeto.tcc;

import gnu.io.CommPortIdentifier;
import java.awt.CardLayout;
import java.text.SimpleDateFormat;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.*;
import serialjava.tcc.SerialConnection;
import serialjava.tcc.SerialConnectionException;
import serialjava.tcc.SerialParameters;

public class NewJframe extends javax.swing.JFrame {

    private final SerialConnection connection;
    private static NewJframe frame; // singleton pattern
```

```
    private float tempProgramada;
    private float tempAtual;
    //private Servidor server;
    private  Servidor server;
    private String texto = " ";
    public static NewJframe getInstance(){ // singleton pattern
    if(frame == null){
     frame = new NewJframe();
     }
     return frame;
     }

    private NewJframe() {


     initComponents();
    // instancio a classe SerialConnection e SerialParameters padrão singleton//
    connection =  SerialConnection.getInstance(SerialParameters.getInstance2());

    }



    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {
```

```
jPanelPrincipal = new javax.swing.JPanel();
jPanelConfigRS232 = new javax.swing.JPanel();
jLabelPortaSerial = new javax.swing.JLabel();
jComboBoxSerial = new javax.swing.JComboBox();
jButtonConectar = new javax.swing.JButton();
jLabel44 = new javax.swing.JLabel();
jLabelTempProgramada = new javax.swing.JLabel();
jTextFieldTempProgramada = new javax.swing.JTextField();
jLabelTempAtual = new javax.swing.JLabel();
jLabelMostraTemp = new javax.swing.JLabel();
jButton1 = new javax.swing.JButton();
jScrollPane1 = new javax.swing.JScrollPane();
jTextArea1 = new javax.swing.JTextArea();
jMenuBar1 = new javax.swing.JMenuBar();
jMenu1 = new javax.swing.JMenu();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Detetector de Avaria de Temperatura");

jPanelPrincipal.setLayout(new java.awt.CardLayout());

jLabelPortaSerial.setText("Porta Serial ");

jComboBoxSerial.setEditable(true);
jComboBoxSerial.setFont(new java.awt.Font("Tahoma", 1, 11)); // NOI18N
jComboBoxSerial.setForeground(new java.awt.Color(51, 0, 255));
jComboBoxSerial.setToolTipText("Configure a Serial Disponivel");
jComboBoxSerial.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jComboBoxSerialActionPerformed(evt);
    }
});
```

```
        jButtonConectar.setText("Conectar");
        jButtonConectar.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButtonConectarActionPerformed(evt);
            }
        });

        jLabel44.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
        jLabel44.setText("DETECTOR DE AVARIA DE TEMPERATURA");

        jLabelTempProgramada.setText("Temperatura Programada");

        jTextFieldTempProgramada.setText("35.0");
        jTextFieldTempProgramada.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jTextFieldTempProgramadaActionPerformed(evt);
            }
        });
        jTextFieldTempProgramada.addFocusListener(new
java.awt.event.FocusAdapter() {
            public void focusLost(java.awt.event.FocusEvent evt) {
                jTextFieldTempProgramadaFocusLost(evt);
            }
        });

        jLabelTempAtual.setText("Temperatura Atual");

        jButton1.setText("Salvar");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
```

```
        });

        jTextArea1.setColumns(20);
        jTextArea1.setRows(5);
        jTextArea1.setText("Rua Marechal Deodoro ,1000\nCuritiba - PR\nLATITUDE :
XXXXXX\nLONGITUDE: XXXXXX");
        jScrollPane1.setViewportView(jTextArea1);

        javax.swing.GroupLayout jPanelConfigRS232Layout = new
javax.swing.GroupLayout(jPanelConfigRS232);
        jPanelConfigRS232.setLayout(jPanelConfigRS232Layout);
        jPanelConfigRS232Layout.setHorizontalGroup(

jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanelConfigRS232Layout.createSequentialGroup()

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                    .addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                        .addGap(148, 148, 148)
                        .addComponent(jLabel44))
                    .addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                        .addContainerGap()
                        .addComponent(jButton1))
                    .addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                        .addGap(19, 19, 19)

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                            .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 299,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addGroup(jPanelConfigRS232Layout.createSequentialGroup()

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                        .addComponent(jLabelTempProgramada)

.addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                            .addComponent(jLabelPortaSerial)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                            .addComponent(jComboBoxSerial,
javax.swing.GroupLayout.PREFERRED_SIZE, 69,
javax.swing.GroupLayout.PREFERRED_SIZE))
                        .addComponent(jLabelTempAtual))

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)

.addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                            .addGap(18, 18, 18)

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.LEADING)
                                .addComponent(jButtonConectar,
javax.swing.GroupLayout.PREFERRED_SIZE, 90,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jTextFieldTempProgramada,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                            .addGap(31, 31, 31)
```

```
                                .addComponent(jLabelMostraTemp)))))))
                .addContainerGap(95, Short.MAX_VALUE))
        );
        jPanelConfigRS232Layout.setVerticalGroup(

jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(jPanelConfigRS232Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel44, javax.swing.GroupLayout.PREFERRED_SIZE,
47, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(jLabelPortaSerial)
                    .addComponent(jComboBoxSerial,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jButtonConectar))
                .addGap(32, 32, 32)

.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                    .addComponent(jLabelTempProgramada)
                    .addComponent(jTextFieldTempProgramada,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(31, 31, 31)
```

```
.addGroup(jPanelConfigRS232Layout.createParallelGroup(javax.swing.GroupLayout.
Alignment.BASELINE)
                .addComponent(jLabelTempAtual)
                .addComponent(jLabelMostraTemp))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 66,
Short.MAX_VALUE)
            .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 118,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(39, 39, 39)
            .addComponent(jButton1)
            .addGap(193, 193, 193))
        );

        jPanelPrincipal.add(jPanelConfigRS232, "tela2");

        jMenu1.setText("Sobre");
        jMenuBar1.add(jMenu1);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 615, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
```

```
                .addComponent(jPanelPrincipal,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 635, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADIN
G)
                .addComponent(jPanelPrincipal,
javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>//GEN-END:initComponents

    private void jButtonConectarActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButtonConectarActionPerformed
       abrirPortaSerial();
        TemperaturaTask taskTemp = new TemperaturaTask();
        taskTemp.execute();
        server =  new Servidor();
        Runnable ThreadupServidor = new upServidor();
        Thread upServer = new Thread(ThreadupServidor);
        upServer.start();
//        ServidorTask taskServer = new ServidorTask();
//        taskServer.execute();
```

```
    }//GEN-LAST:event_jButtonConectarActionPerformed

    private void jComboBoxSerialActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jComboBoxSerialActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_jComboBoxSerialActionPerformed

    private void
jTextFieldTempProgramadaActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jTextFieldTempProgramadaActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_jTextFieldTempProgramadaActionPerformed

    private void jTextFieldTempProgramadaFocusLost(java.awt.event.FocusEvent evt)
{//GEN-FIRST:event_jTextFieldTempProgramadaFocusLost
        setTempProgramada(jTextFieldTempProgramada.getText());
    }//GEN-LAST:event_jTextFieldTempProgramadaFocusLost

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton1ActionPerformed
      //  server = new Servidor();
      //  server.iniciarServidor();
    }//GEN-LAST:event_jButton1ActionPerformed




/**************************************************************************/
/*lista as portas seriais encontradas no computador                       */
/**************************************************************************/
     private void listPorts() {
          CommPortIdentifier portId;
```

```
        Enumeration en = CommPortIdentifier.getPortIdentifiers();
          while (en.hasMoreElements()) {
            portId = (CommPortIdentifier)en.nextElement();
            if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
              jComboBoxSerial.addItem(portId.getName().toString());

            }
          }

      }

  /************************************************************************/
  /* Configura os parametros da serial                                    */
  /************************************************************************/

   private void configurarPortaSerial(){

SerialParameters.getInstance2().setPortName(jComboBoxSerial.getSelectedItem().t
oString());
  SerialParameters.getInstance2().setBaudRate("9600");
  SerialParameters.getInstance2().setFlowControlIn("RTS/CTS In");
  SerialParameters.getInstance2().setFlowControlOut("RTS/CTS Out");
  SerialParameters.getInstance2().setDatabits("8");
  SerialParameters.getInstance2().setStopbits("1");
  SerialParameters.getInstance2().setParity("None");
    }


/************************************************************************/
/*  abrir porta serial                                                  */
/************************************************************************/
    private void abrirPortaSerial(){
    int a;
```

```
    if(connection.isOpen()) return; // porta ja esta aberta
    configurarPortaSerial();
    try {

            a =  connection.openConnection();
            } catch (SerialConnectionException e2) {

            JOptionPane.showMessageDialog(null, "Erro ao Abrir Porta, Tente de
Novo",null,JOptionPane.ERROR_MESSAGE);

              return;
           }
          if(a!= -1){
            JOptionPane.showMessageDialog(null, "CONECTADO !!!");

            return ;// porta aberta
           }
          else{
           JOptionPane.showMessageDialog(null, "Verifique se Algum Aplicativo esta
Usando A serial, e Tente de Novo",null,JOptionPane.ERROR_MESSAGE);
          connection.closeConnection();
            System.exit(0);

            }
            }


     public void MostrarTela(String nome){

   CardLayout card = (CardLayout)jPanelPrincipal.getLayout();
   card.show(jPanelPrincipal,nome);
   }
```

```
/*****************************************************************************/
/*                          METODO MAIN                                      */
/*****************************************************************************/

public static void main(String args[]) {

try {

  UIManager.setLookAndFeel( UIManager.getSystemLookAndFeelClassName()) ;
}
catch (Exception e) {}


        frame =  NewJframe.getInstance(); // singleton pattern//
        frame.pack(); // aumenta a janela só o tamanho de ficar visivel na tela
        frame.MostrarTela("tela2");
        frame.setVisible(true);
        frame.setResizable(false);
        frame.setTempProgramada("35.0");
        frame.listPorts();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   }

public float getTempAtual(String temp){
 String temp2 ="" ;
 String temp1 ="";
 //System.out.println("temp : " + temp);
 if(temp.contains("{")){
  temp1 = temp.replace("{","");
```

```
 }
 if(temp1.contains("}")){
 temp2 = temp1.replace("}", "");
  }
 String temp3 = temp2.trim();
 jLabelMostraTemp.setText(temp3);
 try{
 Float varTemp = Float.parseFloat(temp3);
 tempAtual =   varTemp;
  }catch(Exception e){
 System.out.println("Erro conversão de Temperatura");

 }

return tempAtual;
}


public float getTempprogramada(){

 return tempProgramada;
 }

public void setTempProgramada(String temp){
 String tempPrg = jTextFieldTempProgramada.getText();
 float varTemp = (float)0.0;//.parseFloat(tempPrg);
 try{
  varTemp = Float.parseFloat(tempPrg);
// System.out.println("TempFloat : " + (float)varTemp);
 }catch(Exception e){
 JOptionPane.showMessageDialog(this,"Digite Apenas Numeros
!!!!","Erro",JOptionPane.ERROR_MESSAGE);
 jTextFieldTempProgramada.setText("0");
```

```
  }

 tempProgramada= varTemp;
 }
public String getDataHora(){
  // String data = "yyyy/MM/dd";
    String data = "dd/MM/yyyy";
   String hora = "HH:mm:ss";
   String data1, hora1;
   java.util.Date agora = new java.util.Date();
   SimpleDateFormat formatar = new SimpleDateFormat(data);
   data1 = formatar.format(agora);
   formatar = new SimpleDateFormat(hora);
   hora1 = formatar.format(agora);
   String result = data1 + " " + hora1;
   //System.out.println(data1 + " " + hora1);
   return result;
}



public void verficarTempAlarme(){

 if(tempProgramada <= tempAtual){
  texto = jTextArea1.getText();
  texto = texto.replaceAll("\n",">");

  texto = getDataHora() +">" +texto + ">" + "Temperatura Atual "  + tempAtual
+">"+"Temperatura Programada "  + tempProgramada +"> "+"ALARME!!!>"+"\n";
 // System.out.println(texto);

 }
 else{
```

```
    texto = "....";
 /// texto = jTextArea1.getText();
 //texto = texto + " SEM ALARME !!" + "Temperatura Atual : " + tempAtual + "\n";
 }

}

public String getResposta(){

return texto;
}


class TemperaturaTask extends SwingWorker<Integer, Integer> { // loop

  @Override
  protected Integer doInBackground() throws Exception {
   lerTemperatura();
    return 0;
  }

  @Override
  protected void done() {
    if (isCancelled())
      System.out.println("Cancelled !");
    else
      System.out.println("Done !");
  }
}

public void lerTemperatura() { // fica no loop lendo temperatura
```

```
    while(true){
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            Logger.getLogger(NewJframe.class.getName()).log(Level.SEVERE, null, ex);
        }
   long  tempo = 1000;
   long t0 = System.currentTimeMillis();
     while(connection.buffer_cheio == 0)
     {
      if (System.currentTimeMillis() - t0 > tempo)
          break;
     }
     getTempAtual(connection.inputBuffer.toString());
     verficarTempAlarme();
     connection.inputBuffer.delete(0,connection.inputBuffer.length());
     connection.buffer_cheio =0;

      }
}


class upServidor implements Runnable{

    public void run(){

     server.iniciarServidor();// metodo de loga duração que ira ser executado
    }
}

//  Runnable ThreadupServidor = new upServidor();
//  Thread upServer = new Thread(ThreadupServidor);
//  upServer.start();
```

```
//class ServidorTask extends SwingWorker<Integer, Integer> {
//
//  @Override
//  protected Integer doInBackground() throws Exception {
//    server.iniciarServidor();// metodo de loga duração que ira ser executado
//    return 0;
//  }
//
//  @Override
//  protected void done() {
//    if (isCancelled())
//      System.out.println("Cancelled !");
//    else
//      System.out.println("Done !");
//  }
//    }
```

```
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButtonConectar;
    private javax.swing.JComboBox jComboBoxSerial;
    private javax.swing.JLabel jLabel44;
    private javax.swing.JLabel jLabelMostraTemp;
    private javax.swing.JLabel jLabelPortaSerial;
    private javax.swing.JLabel jLabelTempAtual;
    private javax.swing.JLabel jLabelTempProgramada;
    private javax.swing.JMenu jMenu1;
```

```
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JPanel jPanelConfigRS232;
    private javax.swing.JPanel jPanelPrincipal;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTextArea1;
    private javax.swing.JTextField jTextFieldTempProgramada;
    // End of variables declaration//GEN-END:variables

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package serialjava.tcc;

/* @(#)SerialConnection.java       1.6 98/07/17 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
```

```
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR
A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED.
SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY,
ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */


import gnu.io.*; // serial API
import java.io.*;
//import java.awt.TextArea;
//import java.awt.event.*;
```

```
import java.util.TooManyListenersException;
import javax.swing.*;

/**
A class that handles the details of a serial connection. Reads from one
TextArea and writes to a second TextArea.
Holds the state of the connection.
 */
public class SerialConnection  implements SerialPortEventListener,
        CommPortOwnershipListener {


    private SerialParameters parameters;
    private OutputStream outputSerial;
    private InputStream inputSerial;
    private static  SerialConnection serialconnection;
    private CommPortIdentifier portId;
    private SerialPort sPort; // sport é o objeto criado -> PortaSerial<-
    private boolean open;
    public StringBuffer inputBuffer = new StringBuffer();
    public StringBuffer outputBuffer = new StringBuffer();
    public byte[] buffer_in = new byte[3000];// buffer de entrada
    public int indice;
    public char buffer_cheio = 0;
    public byte byte_ant;
     private static final int AGUARDANDO_PROTOCOLO = 14;
    //private static final int WAITING = 0;
    ////private static final byte ACK = 6;
   // private static final byte NACK = 15;
    public  static int protocolo;
    public  static int acknowledge = 0;

  //  public static boolean tipo_protocolo; // antigo =false novo = 1;
```

```
  // public static int varControleJprogress; //para controlar o progresso da  barra
Jprogress.
  // SerialDemo(args) serial = new SerialDemo(args);
   /**
   Creates a SerialConnection object and initilizes variables passed in
   as params.

   @param parent A SerialDemo object.
   @param parameters A SerialParameters object.
   @param messageAreaOut The TextArea that messages that are to be sent out
   of the serial port are entered into.
   @param messageAreaIn The TextArea that messages comming into the serial
   port are displayed on.
    */
   private SerialConnection(SerialParameters parameters){

     this.parameters = parameters;
     protocolo = AGUARDANDO_PROTOCOLO;

     }

      //método padrao SINGLETON para instanciar SerialConnection //
      public synchronized static SerialConnection getInstance(SerialParameters
parameters){

           if(serialconnection == null) {
               serialconnection = new SerialConnection(parameters);
            }
           return serialconnection;
      }
```

```
/**
Attempts to open a serial connection and streams using the parameters
in the SerialParameters object. If it is unsuccesfull at any step it
returns the port to a closed state, throws a
<code>SerialConnectionException</code>, and returns.

Gives a timeout of 30 seconds on the portOpen to allow other applications
to reliquish the port if have it open and no longer need it.
 */
public int  openConnection() throws SerialConnectionException {

    // Obtain a CommPortIdentifier object for the port you want to open.
     int a;
    try {

        portId =  CommPortIdentifier.getPortIdentifier(parameters.getPortName());
    } catch (NoSuchPortException e) {
       throw new SerialConnectionException(e.getMessage());

    }



     // Open the port represented by the CommPortIdentifier object. Give
     // the open call a relatively long timeout of 30 seconds to allow
     // a different application to reliquish the port if the user
     // wants to.
    try {

        sPort = (SerialPort) portId.open("SerialDemo", 30000);
```

```
        }
       //catch (PortInUseException e) {System.out.println("ERRO DE ABERTURA.");}
        catch (java.lang.UnsatisfiedLinkError e)
        {

            JOptionPane.showMessageDialog(null,"Porta Serial em
uso","erro",JOptionPane.ERROR_MESSAGE);



         return(-1);



        }
        catch (Exception e){
             JOptionPane.showMessageDialog(null,"Porta Serial em
uso","erro",JOptionPane.ERROR_MESSAGE);
             return(-1);

        }

        // Set the parameters of the connection. If they won't set, close the
        // port before throwing an exception.
        try {
            setConnectionParameters();
        } catch (SerialConnectionException e) {
            sPort.close();
            throw e;
        }

        // Open the input and output streams for the connection. If they won't
        // open, close the port before throwing an exception.
        try {
```

```
    outputSerial = sPort.getOutputStream();
    //retorna  a inputstream(dados) que é lido da conexão aberta
    inputSerial = sPort.getInputStream();
    ///***System.out.println(outputSerial);
} catch (IOException e) {
    sPort.close();
    throw new SerialConnectionException("Error opening i/o streams");
}



try {
    sPort.addEventListener(this);
} catch (TooManyListenersException e) {
    sPort.close();
    throw new SerialConnectionException("too many listeners added");
}

// Set notifyOnDataAvailable to true to allow event driven input.
sPort.notifyOnDataAvailable(true);

// Set notifyOnBreakInterrup to allow event driven break handling.
sPort.notifyOnBreakInterrupt(true);

// Set receive timeout to allow breaking out of polling loop during
// input handling.
try {
    sPort.enableReceiveTimeout(30);
} catch (UnsupportedCommOperationException e) {
}

// Add ownership listener to allow ownership event handling.
portId.addPortOwnershipListener(this);
```

```
    open = true;
    return(1);
}

/**
Sets the connection parameters to the setting in the parameters object.
If set fails return the parameters object to origional settings and
throw exception.
 */
public void setConnectionParameters() throws SerialConnectionException {

    // Save state of parameters before trying a set.
   int oldBaudRate ;
   int oldDatabits ;
   int oldStopbits ;
    int oldParity ;
    int oldFlowControl ;

    try
   {
    oldBaudRate = sPort.getBaudRate();
    oldDatabits = sPort.getDataBits();
    oldStopbits = sPort.getStopBits();
    oldParity = sPort.getParity();
    oldFlowControl = sPort.getFlowControlMode();
   }
   catch (java.lang.NullPointerException e)
   {return ;}
    // Set connection parameters, if set fails return parameters object
    // to original state.
    try {
        sPort.setSerialPortParams(parameters.getBaudRate(),
                parameters.getDatabits(),
```

```
                parameters.getStopbits(),
                parameters.getParity());
        } catch (UnsupportedCommOperationException e) {
            parameters.setBaudRate(oldBaudRate);
            parameters.setDatabits(oldDatabits);
            parameters.setStopbits(oldStopbits);
            parameters.setParity(oldParity);
            throw new SerialConnectionException("Unsupported parameter");
        }

        // Set flow control.
        try {
            sPort.setFlowControlMode(parameters.getFlowControlIn()
                    | parameters.getFlowControlOut());
        } catch (UnsupportedCommOperationException e) {
            throw new SerialConnectionException("Unsupported flow control");
        }
    }

    /**
    Close the port and clean up associated elements.
     */
    public void closeConnection() {
        // If port is alread closed just return.

        //System.out.println("estou no closeConecction"); //debug
        if (!open) {
            return;
        }

        // Remove the key listener.
      //  messageAreaOut.removeKeyListener(keyHandler);
```

```
    // Check to make sure sPort has reference to avoid a NPE.
    if (sPort != null) {
        try {
            // close the i/o streams.
            outputSerial.close();
            inputSerial.close();
        } catch (IOException e) {
            System.err.println(e);
        }

        // Close the port.
        sPort.close();

        // Remove the ownership listener.
        portId.removePortOwnershipListener(this);
    }

    open = false;
}

/**
Send a one second break signal.
 */
public void sendBreak() {
    sPort.sendBreak(1000);
}

/**
Reports the open status of the port.
@return true if port is open, false if port is closed.
 */
public boolean isOpen() {
    return open;
```

```
}

/**
Handles SerialPortEvents. The two types of SerialPortEvents that this
program is registered to listen for are DATA_AVAILABLE and BI. During
DATA_AVAILABLE the port buffer is read until it is drained, when no more
data is availble and 30ms has passed the method returns. When a BI
event occurs the words BREAK RECEIVED are written to the messageAreaIn.
 */
public synchronized void serialEvent(SerialPortEvent e) {
    // Create a StringBuffer and int to receive input data.
    //inputBuffer é o buffer de armazenamento dos dados recebidos
    // na porta serial

    int newData = 0;

    // Determine type of event.
    switch (e.getEventType()) {

        // Read data until -1 is returned. If \r is received substitute
        // \n for correct newline handling.
        case SerialPortEvent.DATA_AVAILABLE:
            while (newData != -1) {
                try {
                 //   System.out.println("recepção serial ");
                    //le os proximos bytes vindos de Input Stream
                    // os ranges dos bytes é de 0 a 255
                    // se ocorrer o fim do arquivo o valor -1 é retornado
                    newData = inputSerial.read();
                    if (newData == -1) {
                        break;
                    }
```

```
    if((byte)newData == '{')
    {
    inputBuffer.append((char)newData);

     return;
    }
    if((byte)newData =='}')
    {
     inputBuffer.append((char)newData);
      buffer_cheio = 1;
      return;
    }

     inputBuffer.append((char)newData);




  } catch (IOException ex) {
      System.err.println(ex);
      return;
  }

 }
```

```
//*******************************************************//
   // INPUT BUFFER  É O BUFFER DE ENTRADA DA SERIAL          //
```

```
                // INPUT BUFFER É IGUAL A UM CARACTER 0 a 255            //
                 break;

            // If break event append BREAK RECEIVED message.
            case SerialPortEvent.BI:
            //    messageAreaIn.append("\n--- BREAK RECEIVED ---\n");
        }

    }

    /**
    Handles ownership events. If a PORT_OWNERSHIP_REQUESTED event is
    received a dialog box is created asking the user if they are
    willing to give up the port. No action is taken on other types
    of ownership events.
     */
    public void ownershipChange(int type) {
        if (type == CommPortOwnershipListener.PORT_OWNERSHIP_REQUESTED) {
            // PortRequestedDialog prd = new PortRequestedDialog(parent);
        }
    }

/**
 O metodo enviar mensagem escreve no buffer serial de ESCRITA

*/

    public synchronized void enviarMensagem(String message){
          String msg ;
        msg =message;
          try{
        outputSerial.write(msg.getBytes());
        outputSerial.flush();
```

```
        }catch(IOException e){
        System.out.println("ERRRROOOOOOO");
        }
    }

    public void enviarArray(char[] ptr,int k) {

     int n;
    try{
    for( n= 0; n< k;n++){
    outputSerial.write(ptr[n]);
    outputSerial.flush();
    }
     }catch(IOException e){
        System.out.println("ERRRROOOOOOO");
        }

    }



}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package serialjava.tcc;


//import gnu.io.*; // serial API

/* @(#)SerialConnectionException.java 1.3 98/06/04 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR
A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED.
SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY,
ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN
```

```
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */


public class SerialConnectionException extends Exception {
// SerialDemo serial = new SerialDemo("");
    /**
     * Constructs a <code>SerialConnectionException</code>
     * with the specified detail message.
     *
     * @param   s   the detail message.
     */
    public SerialConnectionException(String str) {
        super(str);
    }

    /**
     * Constructs a <code>SerialConnectionException</code>
     * with no detail message.
     */
    public SerialConnectionException() {
        super();
    }
}
```

```
/*
 * @(#)SerialDemo.java     1.9 98/06/05 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR
A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED.
SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY,
ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
```

```
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */

package serialjava.tcc;
import gnu.io.*; // serial API
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Properties;
import java.util.Enumeration;

/**
Main file for SerialDemo program. This program illustrates many of the
abilities of the javax.comm api. This file contains the GUI framework that
the program runs in.
 */
public class SerialDemo extends Frame implements ActionListener {

    final int HEIGHT = 450;
    final int WIDTH = 410;
    private MenuBar mb;
    private Menu fileMenu;
    private MenuItem openItem;
    private MenuItem saveItem;
    private MenuItem exitItem;
    //private Button openButton;
```

```
public Button openButton;
private Button closeButton;
private Button breakButton;
private Panel buttonPanel;
private Panel messagePanel;
private TextArea messageAreaOut;
private TextArea messageAreaIn;
private ConfigurationPanel configurationPanel;
private SerialParameters parameters;
private SerialConnection connection;
private Properties props = null;

/**
Main method. Checks to see if the command line agrument is requesting
usage informaition (-h, -help), if it is, display a usage message and
exit, otherwise create a new <code>SerialDemo</code> and set it visible.
 */
public static void main(String[] args) {
    if ((args.length > 0)
            && (args[0].equals("-h")
            || args[0].equals("-help"))) {
        System.out.println("usage: java SerialDemo [configuration File]");
        System.exit(1);
    }

    SerialDemo serialDemo = new SerialDemo(args);
    serialDemo.setVisible(true);
    serialDemo.repaint();
      serialDemo.messageAreaOut.setEditable(false);

}

/**
```

```
Create new <code>SerialDemo</code> and initilizes it. Parses args to
find configuration file. If found, initial state it set to parameters
in configuration file.

@param args command line arguments used when program was invoked.
 */
public SerialDemo(String[] args) {
    super("Serial Demo");

  //  parameters = new SerialParameters();

    // Set up the GUI for the program
    addWindowListener(new CloseHandler(this));

    mb = new MenuBar();

    fileMenu = new Menu("File");

    openItem = new MenuItem("Load");
    openItem.addActionListener(this);
    fileMenu.add(openItem);

    saveItem = new MenuItem("Save");
    saveItem.addActionListener(this);
    fileMenu.add(saveItem);

    exitItem = new MenuItem("Exit");
    exitItem.addActionListener(this);
    fileMenu.add(exitItem);

    mb.add(fileMenu);

    setMenuBar(mb);
```

```
messagePanel = new Panel();
messagePanel.setLayout(new GridLayout(2, 1));

messageAreaOut = new TextArea();
messagePanel.add(messageAreaOut);

messageAreaIn = new TextArea();
messageAreaIn.setEditable(false);
messagePanel.add(messageAreaIn);

add(messagePanel, "Center");

configurationPanel = new ConfigurationPanel(this);

buttonPanel = new Panel();

openButton = new Button("Open Port");
openButton.addActionListener(this);
buttonPanel.add(openButton);

closeButton = new Button("Close Port");
closeButton.addActionListener(this);
closeButton.setEnabled(false);
buttonPanel.add(closeButton);

breakButton = new Button("Send Break");
breakButton.addActionListener(this);
breakButton.setEnabled(false);
buttonPanel.add(breakButton);

Panel southPanel = new Panel();
```

```
    GridBagLayout gridBag = new GridBagLayout();
    GridBagConstraints cons = new GridBagConstraints();

    southPanel.setLayout(gridBag);

    cons.gridwidth = GridBagConstraints.REMAINDER;
    gridBag.setConstraints(configurationPanel, cons);
    cons.weightx = 1.0;
    southPanel.add(configurationPanel);
    gridBag.setConstraints(buttonPanel, cons);
    southPanel.add(buttonPanel);

    add(southPanel, "South");

    parseArgs(args);

   // connection = new SerialConnection(this, parameters,
   //         messageAreaOut, messageAreaIn);
    setConfigurationPanel();

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

    setLocation(screenSize.width / 2 - WIDTH / 2,
            screenSize.height / 2 - HEIGHT / 2);

    setSize(WIDTH, HEIGHT);
}

/**
Sets the GUI elements on the configurationPanel.
 */
public void setConfigurationPanel() {
```

```
    configurationPanel.setConfigurationPanel();
}

/**
Responds to the menu items and buttons.
 */
public void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();
    int a;
    // Loads a configuration file.
    if (cmd.equals("Load")) {
        if (connection.isOpen()) {
            AlertDialog ad = new AlertDialog(this, "Port Open!",
                    "Configuration may not",
                    "be loaded",
                    "while a port is open.");
        } else {
            FileDialog fd = new FileDialog(this,
                    "Load Port Configuration",
                    FileDialog.LOAD);
            fd.setVisible(true);
            String file = fd.getFile();
            if (file != null) {
                String dir = fd.getDirectory();
                File f = new File(dir + file);
                try {
                    FileInputStream fis = new FileInputStream(f);
                    props = new Properties();
                    props.load(fis);
                    fis.close();
                } catch (FileNotFoundException e1) {
                    System.err.println(e1);
                } catch (IOException e2) {
```

```
                System.err.println(e2);
            }
            loadParams();
        }
    }
}

// Saves a configuration file.
if (cmd.equals("Save")) {
    configurationPanel.setParameters();
    FileDialog fd = new FileDialog(this, "Save Port Configuration",
            FileDialog.SAVE);
    fd.setFile("serialdemo.properties");
    fd.setVisible(true);
    String fileName = fd.getFile();
    String directory = fd.getDirectory();
    if ((fileName != null) && (directory != null)) {
        writeFile(directory + fileName);
    }
}

// Calls shutdown, which exits the program.
if (cmd.equals("Exit")) {
    shutdown();
}

// Opens a port.
if (cmd.equals("Open Port")) {
   // System.out.println("Open Port");
    openButton.setEnabled(false);
    Cursor previousCursor = getCursor();
    setNewCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
    configurationPanel.setParameters();
```

```
            try {
                System.out.println("vou chamar OPENCONECTION"); //debug
                                    // int a;
             a =  connection.openConnection();
               System.out.println("voltei da chamada de OPENCONECTION"); //debug
                        } catch (SerialConnectionException e2) {

               AlertDialog ad = new AlertDialog(this,
                        "Error Opening Port!",
                        "Error opening port,",
                        e2.getMessage() + ".",
                        "Select new settings, try again.");
               openButton.setEnabled(true);
               setNewCursor(previousCursor);
               return;
            }
             if(a!= -1){

            portOpened();
            setNewCursor(previousCursor);
            connection.enviarMensagem("0123456789rapadura is sweet but no
software");
             }
             else{
             portClosed();
             setNewCursor(previousCursor);
             }
        }
        // Closes a port.
        if (cmd.equals("Close Port")) {
            System.out.println("close port"); //debug
            portClosed();
```

```
    }

    // Sends a break signal to the port.
    if (cmd.equals("Send Break")) {
        connection.sendBreak();
    }
}
// }
/**
Toggles the buttons to an open port state.
 */
public void portOpened() {
    openButton.setEnabled(false);
    closeButton.setEnabled(true);
    breakButton.setEnabled(true);
    // messageAreaIn.setEditable(true);
    messageAreaOut.setEditable(true);
            //messageAreaOut.setText("Area Out");
}

/**
Calls closeConnection on the SerialConnection and toggles the buttons
to a closed port state.
 */
public void portClosed() {
    //System.out.println("close port"); //debug
    connection.closeConnection();
    openButton.setEnabled(true);
    closeButton.setEnabled(false);
    breakButton.setEnabled(false);
    messageAreaIn.setEditable(false);
    messageAreaOut.setEditable(false);
}
```

```
/**
Sets the <code>Cursor</code> for the application.
@param c New <code>Cursor</code>
 */
private void setNewCursor(Cursor c) {
    setCursor(c);
    messageAreaIn.setCursor(c);
    messageAreaOut.setCursor(c);
}

/**
Writes the current parameters to a configuration file of the
java.properties style.
 */
private void writeFile(String path) {

    Properties newProps;
    FileOutputStream fileOut = null;

    newProps = new Properties();

    newProps.put("portName", parameters.getPortName());
    newProps.put("baudRate", parameters.getBaudRateString());
    newProps.put("flowControlIn", parameters.getFlowControlInString());
    newProps.put("flowControlOut", parameters.getFlowControlOutString());
    newProps.put("parity", parameters.getParityString());
    newProps.put("databits", parameters.getDatabitsString());
    newProps.put("stopbits", parameters.getStopbitsString());

    try {
        fileOut = new FileOutputStream(path);
    } catch (IOException e) {
```

```
            System.out.println("Could not open file for writiing");
        }

        newProps.save(fileOut, "Serial Demo poperties");

        try {
            fileOut.close();
        } catch (IOException e) {
            System.out.println("Could not close file for writiing");
        }
    }

    /**
    Cleanly shuts down the applicaion. first closes any open ports and
    cleans up, then exits.
     */
    private void shutdown() {
        connection.closeConnection();
        System.exit(1);
    }

    /**
    Finds configuration file in arguments and creates a properties object from
    that file.
     */
    private void parseArgs(String[] args) {
        if (args.length < 1) {
            return;
        }

        File f = new File(args[0]);

        if (!f.exists()) {
```

```
        f = new File(System.getProperty("user.dir")
                + System.getProperty("path.separator")
                + args[0]);
    }

    if (f.exists()) {
        try {
            FileInputStream fis = new FileInputStream(f);
            props = new Properties();
            props.load(fis);
            fis.close();
            loadParams();
        } catch (IOException e) {
        }
    }
}

/**
Set the parameters object to the settings in the properties object.
 */
private void loadParams() {
    parameters.setPortName(props.getProperty("portName"));
    parameters.setBaudRate(props.getProperty("baudRate"));
    parameters.setFlowControlIn(props.getProperty("flowControlIn"));
    parameters.setFlowControlOut(props.getProperty("flowControlOut"));
    parameters.setParity(props.getProperty("parity"));
    parameters.setDatabits(props.getProperty("databits"));
    parameters.setStopbits(props.getProperty("stopbits"));

    setConfigurationPanel();
}

/**
```

```
GUI element that holds the user changable elements for connection
configuration.
 */
class ConfigurationPanel extends Panel implements ItemListener {

    private Frame parent;
    private Label portNameLabel;
    private Choice portChoice;
    private Label baudLabel;
    private Choice baudChoice;
    private Label flowControlInLabel;
    private Choice flowChoiceIn;
    private Label flowControlOutLabel;
    private Choice flowChoiceOut;
    private Label databitsLabel;
    private Choice databitsChoice;
    private Label stopbitsLabel;
    private Choice stopbitsChoice;
    private Label parityLabel;
    private Choice parityChoice;

    /**
    Creates and initilizes the configuration panel. The initial settings
    are from the parameters object.
     */
    public ConfigurationPanel(Frame parent) {
        this.parent = parent;

        setLayout(new GridLayout(4, 4));

        portNameLabel = new Label("Port Name:", Label.LEFT);
        add(portNameLabel);
```

```
portChoice = new Choice();
portChoice.addItemListener(this);
add(portChoice);
listPortChoices();
portChoice.select(parameters.getPortName());

baudLabel = new Label("Baud Rate:", Label.LEFT);
add(baudLabel);

baudChoice = new Choice();
baudChoice.addItem("300");
baudChoice.addItem("2400");
baudChoice.addItem("9600");
baudChoice.addItem("14400");
baudChoice.addItem("28800");
baudChoice.addItem("38400");
baudChoice.addItem("57600");
baudChoice.addItem("152000");
baudChoice.select(Integer.toString(parameters.getBaudRate()));
baudChoice.addItemListener(this);
add(baudChoice);

flowControlInLabel = new Label("Flow Control In:", Label.LEFT);
add(flowControlInLabel);

flowChoiceIn = new Choice();
flowChoiceIn.addItem("None");
flowChoiceIn.addItem("Xon/Xoff In");
flowChoiceIn.addItem("RTS/CTS In");
flowChoiceIn.select(parameters.getFlowControlInString());
flowChoiceIn.addItemListener(this);
add(flowChoiceIn);
```

```
flowControlOutLabel = new Label("Flow Control Out:", Label.LEFT);
add(flowControlOutLabel);

flowChoiceOut = new Choice();
flowChoiceOut.addItem("None");
flowChoiceOut.addItem("Xon/Xoff Out");
flowChoiceOut.addItem("RTS/CTS Out");
flowChoiceOut.select(parameters.getFlowControlOutString());
flowChoiceOut.addItemListener(this);
add(flowChoiceOut);

databitsLabel = new Label("Data Bits:", Label.LEFT);
add(databitsLabel);

databitsChoice = new Choice();
databitsChoice.addItem("5");
databitsChoice.addItem("6");
databitsChoice.addItem("7");
databitsChoice.addItem("8");
databitsChoice.select(parameters.getDatabitsString());
databitsChoice.addItemListener(this);
add(databitsChoice);

stopbitsLabel = new Label("Stop Bits:", Label.LEFT);
add(stopbitsLabel);

stopbitsChoice = new Choice();
stopbitsChoice.addItem("1");
stopbitsChoice.addItem("1.5");
stopbitsChoice.addItem("2");
stopbitsChoice.select(parameters.getStopbitsString());
stopbitsChoice.addItemListener(this);
add(stopbitsChoice);
```

```
    parityLabel = new Label("Parity:", Label.LEFT);
    add(parityLabel);

    parityChoice = new Choice();
    parityChoice.addItem("None");
    parityChoice.addItem("Even");
    parityChoice.addItem("Odd");
    parityChoice.select("None");
    parityChoice.select(parameters.getParityString());
    parityChoice.addItemListener(this);
    add(parityChoice);
}

/**
Sets the configuration panel to the settings in the parameters object.
 */
public void setConfigurationPanel() {
    portChoice.select(parameters.getPortName());
    baudChoice.select(parameters.getBaudRateString());
    flowChoiceIn.select(parameters.getFlowControlInString());
    flowChoiceOut.select(parameters.getFlowControlOutString());
    databitsChoice.select(parameters.getDatabitsString());
    stopbitsChoice.select(parameters.getStopbitsString());
    parityChoice.select(parameters.getParityString());
}

/**
Sets the parameters object to the settings in the configuration panel.
 */
public void setParameters() {
    parameters.setPortName(portChoice.getSelectedItem());
    parameters.setBaudRate(baudChoice.getSelectedItem());
```

```
    parameters.setFlowControlIn(flowChoiceIn.getSelectedItem());
    parameters.setFlowControlOut(flowChoiceOut.getSelectedItem());
    parameters.setDatabits(databitsChoice.getSelectedItem());
    parameters.setStopbits(stopbitsChoice.getSelectedItem());
    parameters.setParity(parityChoice.getSelectedItem());
}

/**
Sets the elements for the portChoice from the ports available on the
system. Uses an emuneration of comm ports returned by
CommPortIdentifier.getPortIdentifiers(), then sets the current
choice to a mathing element in the parameters object.
 */
void listPortChoices() {
    CommPortIdentifier portId;

    Enumeration en = CommPortIdentifier.getPortIdentifiers();

    // iterate through the ports.
    while (en.hasMoreElements()) {
        portId = (CommPortIdentifier) en.nextElement();
        if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
            portChoice.addItem(portId.getName());
        }
    }
    portChoice.select(parameters.getPortName());
}

/**
Event handler for changes in the current selection of the Choices.
If a port is open the port can not be changed.
If the choice is unsupported on the platform then the user will
be notified and the settings will revert to their pre-selection
```

```
state.
 */
public void itemStateChanged(ItemEvent e) {
    // Check if port is open.
    if (connection.isOpen()) {
        // If port is open do not allow port to change.
        if (e.getItemSelectable() == portChoice) {
            // Alert user.
            AlertDialog ad = new AlertDialog(parent, "Port Open!",
                    "Port can not",
                    "be changed",
                    "while a port is open.");

            // Return configurationPanel to pre-choice settings.
            setConfigurationPanel();
            return;
        }
        // Set the parameters from the choice panel.
        setParameters();
        try {
            // Attempt to change the settings on an open port.
            connection.setConnectionParameters();
        } catch (SerialConnectionException ex) {
            // If setting can not be changed, alert user, return to
            // pre-choice settings.
            AlertDialog ad = new AlertDialog(parent,
                    "Unsupported Configuration!",
                    "Configuration Parameter unsupported,",
                    "select new value.",
                    "Returning to previous configuration.");
            setConfigurationPanel();
        }
    } else {
```

```
                // Since port is not open just set the parameter object.
                setParameters();
            }
        }
    }

    /**
    Handles closing down system. Allows application to be closed with window
    close box.
     */
    class CloseHandler extends WindowAdapter {

        SerialDemo sd;

        public CloseHandler(SerialDemo sd) {
            this.sd = sd;
        }

        public void windowClosing(WindowEvent e) {
            sd.shutdown();
        }
    }
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package serialjava.tcc;

/* @(#)SerialParameters.java       1.5 98/07/17 SMI
 *
```

```
 * purposes.
 */


import gnu.io.*; // serial API

/**
A class that stores parameters for serial ports.
*/
public class SerialParameters {

    private String portName;
    private int baudRate;
    private int flowControlIn;
    private int flowControlOut;
    private int databits;
    private int stopbits;
    private int parity;
    private static SerialParameters parameters;

    /**
    Default constructer. Sets parameters to no port, 9600 baud, no flow
    control, 8 data bits, 1 stop bit, no parity.
    */
    private SerialParameters () {
        this("",
            9600,
            SerialPort.FLOWCONTROL_NONE,
            SerialPort.FLOWCONTROL_NONE,
            SerialPort.DATABITS_8,
            SerialPort.STOPBITS_1,
            SerialPort.PARITY_NONE );
```

```
}

/**
Paramaterized constructer.

@param portName The name of the port.
@param baudRate The baud rate.
@param flowControlIn Type of flow control for receiving.
@param flowControlOut Type of flow control for sending.
@param databits The number of data bits.
@param stopbits The number of stop bits.
@param parity The type of parity.
*/
private SerialParameters(String portName,
                         int baudRate,
                         int flowControlIn,
                         int flowControlOut,
                         int databits,
                         int stopbits,
                         int parity) {

    this.portName = portName;
    this.baudRate = baudRate;
    this.flowControlIn = flowControlIn;
    this.flowControlOut = flowControlOut;
    this.databits = databits;
    this.stopbits = stopbits;
    this.parity = parity;
}

/**
Sets port name.
@param portName New port name.
```

```
 *

*/
     public synchronized static SerialParameters getInstance2(){
if(parameters == null) parameters = new SerialParameters();
return parameters;

}
public void setPortName(String portName) {
      this.portName = portName;
}

/**
Gets port name.
@return Current port name.
*/
public String getPortName() {
      return portName;
}

/**
Sets baud rate.
@param baudRate New baud rate.
*/
public void setBaudRate(int baudRate) {
      this.baudRate = baudRate;
}

/**
Sets baud rate.
@param baudRate New baud rate.
*/
public void setBaudRate(String baudRate) {
```

```
        this.baudRate = Integer.parseInt(baudRate);
    }

    /**
    Gets baud rate as an <code>int</code>.
    @return Current baud rate.
    */
    public int getBaudRate() {
        return baudRate;
    }

    /**
    Gets baud rate as a <code>String</code>.
    @return Current baud rate.
    */
    public String getBaudRateString() {
        return Integer.toString(baudRate);
    }

    /**
    Sets flow control for reading.
    @param flowControlIn New flow control for reading type.
    */
    public void setFlowControlIn(int flowControlIn) {
        this.flowControlIn = flowControlIn;
    }

    /**
    Sets flow control for reading.
    @param flowControlIn New flow control for reading type.
    */
    public void setFlowControlIn(String flowControlIn) {
        this.flowControlIn = stringToFlow(flowControlIn);
```

```
}

/**
Gets flow control for reading as an <code>int</code>.
@return Current flow control type.
*/
public int getFlowControlIn() {
	return flowControlIn;
}

/**
Gets flow control for reading as a <code>String</code>.
@return Current flow control type.
*/
public String getFlowControlInString() {
	return flowToString(flowControlIn);
}

/**
Sets flow control for writing.
@param flowControlIn New flow control for writing type.
*/
public void setFlowControlOut(int flowControlOut) {
	this.flowControlOut = flowControlOut;
}

/**
Sets flow control for writing.
@param flowControlIn New flow control for writing type.
*/
public void setFlowControlOut(String flowControlOut) {
	this.flowControlOut = stringToFlow(flowControlOut);
}
```

```
/**
Gets flow control for writing as an <code>int</code>.
@return Current flow control type.
*/
public int getFlowControlOut() {
      return flowControlOut;
}

/**
Gets flow control for writing as a <code>String</code>.
@return Current flow control type.
*/
public String getFlowControlOutString() {
      return flowToString(flowControlOut);
}

/**
Sets data bits.
@param databits New data bits setting.
*/
public void setDatabits(int databits) {
      this.databits = databits;
}

/**
Sets data bits.
@param databits New data bits setting.
*/
public void setDatabits(String databits) {
      if (databits.equals("5")) {
         this.databits = SerialPort.DATABITS_5;
      }
```

```
        if (databits.equals("6")) {
            this.databits = SerialPort.DATABITS_6;
        }
        if (databits.equals("7")) {
            this.databits = SerialPort.DATABITS_7;
        }
        if (databits.equals("8")) {
            this.databits = SerialPort.DATABITS_8;
        }
    }

    /**
    Gets data bits as an <code>int</code>.
    @return Current data bits setting.
    */
    public int getDatabits() {
        return databits;
    }

    /**
    Gets data bits as a <code>String</code>.
    @return Current data bits setting.
    */
    public String getDatabitsString() {
        switch(databits) {
            case SerialPort.DATABITS_5:
                return "5";
            case SerialPort.DATABITS_6:
                return "6";
            case SerialPort.DATABITS_7:
                return "7";
            case SerialPort.DATABITS_8:
                return "8";
```

```
            default:
                return "8";
        }
    }

    /**
    Sets stop bits.
    @param stopbits New stop bits setting.
    */
    public void setStopbits(int stopbits) {
        this.stopbits = stopbits;
    }

    /**
    Sets stop bits.
    @param stopbits New stop bits setting.
    */
    public void setStopbits(String stopbits) {
        if (stopbits.equals("1")) {
            this.stopbits = SerialPort.STOPBITS_1;
        }
        if (stopbits.equals("1.5")) {
            this.stopbits = SerialPort.STOPBITS_1_5;
        }
        if (stopbits.equals("2")) {
            this.stopbits = SerialPort.STOPBITS_2;
        }
    }

    /**
    Gets stop bits setting as an <code>int</code>.
    @return Current stop bits setting.
    */
```

```
public int getStopbits() {
	return stopbits;
}

/**
Gets stop bits setting as a <code>String</code>.
@return Current stop bits setting.
*/
public String getStopbitsString() {
	switch(stopbits) {
		case SerialPort.STOPBITS_1:
			return "1";
		case SerialPort.STOPBITS_1_5:
			return "1.5";
		case SerialPort.STOPBITS_2:
			return "2";
		default:
			return "1";
	}
}

/**
Sets parity setting.
@param parity New parity setting.
*/
public void setParity(int parity) {
	this.parity = parity;
}

/**
Sets parity setting.
@param parity New parity setting.
*/
```

```
public void setParity(String parity) {
      if (parity.equals("None")) {
         this.parity = SerialPort.PARITY_NONE;
      }
      if (parity.equals("Even")) {
         this.parity = SerialPort.PARITY_EVEN;
      }
      if (parity.equals("Odd")) {
         this.parity = SerialPort.PARITY_ODD;
      }
}

/**
Gets parity setting as an <code>int</code>.
@return Current parity setting.
*/
public int getParity() {
      return parity;
}

/**
Gets parity setting as a <code>String</code>.
@return Current parity setting.
*/
public String getParityString() {
      switch(parity) {
         case SerialPort.PARITY_NONE:
               return "None";
         case SerialPort.PARITY_EVEN:
               return "Even";
         case SerialPort.PARITY_ODD:
               return "Odd";
         default:
```

```
            return "None";
        }
    }

    /**
    Converts a <code>String</code> describing a flow control type to an
    <code>int</code> type defined in <code>SerialPort</code>.
    @param flowControl A <code>string</code> describing a flow control type.
    @return An <code>int</code> describing a flow control type.
    */
    private int stringToFlow(String flowControl) {
        if (flowControl.equals("None")) {
            return SerialPort.FLOWCONTROL_NONE;
        }
        if (flowControl.equals("Xon/Xoff Out")) {
            return SerialPort.FLOWCONTROL_XONXOFF_OUT;
        }
        if (flowControl.equals("Xon/Xoff In")) {
            return SerialPort.FLOWCONTROL_XONXOFF_IN;
        }
        if (flowControl.equals("RTS/CTS In")) {
            return SerialPort.FLOWCONTROL_RTSCTS_IN;
        }
        if (flowControl.equals("RTS/CTS Out")) {
            return SerialPort.FLOWCONTROL_RTSCTS_OUT;
        }
        return SerialPort.FLOWCONTROL_NONE;
    }

    /**
    Converts an <code>int</code> describing a flow control type to a
    <code>String</code> describing a flow control type.
    @param flowControl An <code>int</code> describing a flow control type.
```

```
    @return A <code>String</code> describing a flow control type.
    */
    String flowToString(int flowControl) {
        switch(flowControl) {
            case SerialPort.FLOWCONTROL_NONE:
                return "None";
            case SerialPort.FLOWCONTROL_XONXOFF_OUT:
                return "Xon/Xoff Out";
            case SerialPort.FLOWCONTROL_XONXOFF_IN:
                return "Xon/Xoff In";
            case SerialPort.FLOWCONTROL_RTSCTS_IN:
                return "RTS/CTS In";
            case SerialPort.FLOWCONTROL_RTSCTS_OUT:
                return "RTS/CTS Out";
            default:
                return "None";
        }
    }
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package serialjava.tcc;

/*
 * @(#)AlertDialog.java      1.3 98/06/04 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR
A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED.
SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY,
ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
```

```
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */

import java.awt.*;
import java.awt.event.*;

/**
A single response modal alert dialog. This class is configurable for message
and title. The width of the dialog will be longer than the longest message
line.  When the OK button is pressed the dialog returns.
*/
public class AlertDialog extends Dialog implements ActionListener {

    /**
    Creates a new <code>AlertDialog</code> with three lines of message and
    a title.

    @param parent Any Frame.
    @param title The title to appear in the border of the dialog.
    @param lineOne The first line of the message in the dialog.
    @param lineTwo The second line of the message in the dialog.
    @param lineThree The third line of the message in the dialog.
    */
    public AlertDialog(Frame parent,
                       String title,
                       String lineOne,
                       String lineTwo,
```

```
                    String lineThree) {
        super(parent, title, true);

        Panel labelPanel = new Panel();
        labelPanel.setLayout(new GridLayout(3, 1));
        labelPanel.add(new Label(lineOne, Label.CENTER));
        labelPanel.add(new Label(lineTwo, Label.CENTER));
        labelPanel.add(new Label(lineThree, Label.CENTER));
        add(labelPanel, "Center");

        Panel buttonPanel = new Panel();
        Button okButton = new Button("OK");
        okButton.addActionListener(this);
        buttonPanel.add(okButton);
        add(buttonPanel, "South");

        FontMetrics fm = getFontMetrics(getFont());
        int width = Math.max(fm.stringWidth(lineOne),
                Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

        setSize(width + 40, 150);
        setLocation(parent.getLocationOnScreen().x + 30,
                    parent.getLocationOnScreen().y + 30);
        setVisible(true);
    }

    /**
    Handles events from the OK button. When OK is pressed the dialog becomes
    invisible, disposes of its self, and retruns.
    */
    public void actionPerformed(ActionEvent e) {
        setVisible(false);
        dispose();
```

```
    }
}


/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package serialjava.tcc;

/**
 *
 *
 */
class NewJframe {

}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package serialjava.tcc;

/*
 * @(#)PortRequestedDialog.java          1.3 98/06/04 SMI
 *
 * Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.
 *
 * Sun grants you ("Licensee") a non-exclusive, royalty free, license
 * to use, modify and redistribute this software in source and binary
 * code form, provided that i) this copyright notice and license appear
```

```
 * on all copies of the software; and ii) Licensee does not utilize the
 * software in a manner which is disparaging to Sun.
 *
 * This software is provided "AS IS," without a warranty of any kind.
 * ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND
WARRANTIES,
 * INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR
A
 * PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED.
SUN AND
 * ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
 * LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE
 * SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS
LICENSORS
 * BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT,
 * INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE
DAMAGES,
 * HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY,
ARISING
 * OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS
BEEN
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * This software is not designed or intended for use in on-line control
 * of aircraft, air traffic, aircraft navigation or aircraft
 * communications; or in the design, construction, operation or
 * maintenance of any nuclear facility. Licensee represents and
 * warrants that it will not use or redistribute the Software for such
 * purposes.
 */

import java.awt.*;
import java.awt.event.*;
```

```
//import javax.comm.*;
//import gnu.io.*; // serial API
/**
Informs the user that an other application has requested the port they
are using, and then asks if they are willing to give it up. If the user
answers "Yes" the port is closed and the dialog is closed, if the user
answers "No" the dialog closes and no other action is taken.
 */
public class PortRequestedDialog extends Dialog implements ActionListener {

    private SerialDemo parent;

    /**
    Creates the a dialog with two buttons and a message asking the user if
    they are willing to give up the port they are using.

    @param parent The main SerialDemo object.
     */
    public PortRequestedDialog(SerialDemo parent) {
        super(parent, "Port Requested!", true);
        this.parent = parent;

        String lineOne = "Your port has been requested";
        String lineTwo = "by an other application.";
        String lineThree = "Do you want to give up your port?";
        Panel labelPanel = new Panel();
        labelPanel.setLayout(new GridLayout(3, 1));
        labelPanel.add(new Label(lineOne, Label.CENTER));
        labelPanel.add(new Label(lineTwo, Label.CENTER));
        labelPanel.add(new Label(lineThree, Label.CENTER));
        add(labelPanel, "Center");

        Panel buttonPanel = new Panel();
```

```
    Button yesButton = new Button("Yes");
    yesButton.addActionListener(this);
    buttonPanel.add(yesButton);
    Button noButton = new Button("No");
    noButton.addActionListener(this);
    buttonPanel.add(noButton);
    add(buttonPanel, "South");

    FontMetrics fm = getFontMetrics(getFont());
    int width = Math.max(fm.stringWidth(lineOne),
            Math.max(fm.stringWidth(lineTwo), fm.stringWidth(lineThree)));

    setSize(width + 40, 150);
    setLocation(parent.getLocationOnScreen().x + 30,
            parent.getLocationOnScreen().y + 30);
    setVisible(true);
}

/**
Handles events generated by the buttons. If the yes button in pushed the
port closing routine is called and the dialog is disposed of. If the "No"
button is pushed the dialog is disposed of.
 */
public void actionPerformed(ActionEvent e) {
    String cmd = e.getActionCommand();

    if (cmd.equals("Yes")) {
        parent.portClosed();
    }

    setVisible(false);
    dispose();
}}
```