DANILO ALEIXO GOMES DE SOUZA

CRIAÇÃO DE TESTES PARA UM SOFTWARE EMBARCADO CRÍTICO ESPACIAL

SÃO PAULO
2014

DANILO ALEIXO GOMES DE SOUZA

CRIAÇÃO DE TESTES PARA UM SOFTWARE EMBARCADO CRÍTICO ESPACIAL

RELATÓRIO FINAL DO PROJETO DE INICIAÇÃO CIENTÍFICA EM CIÊNCIA DA COMPUTAÇÃO PELA UNIVERSIDADE DE SÃO PAULO.

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
UNIVERSIDADE DE SÃO PAULO

SÃO PAULO
2014

AGRADECIMENTOS

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), à Pró-Reitoria de Pesquisa da Universidade de São Paulo (PRP-USP) e ao Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP) pela Bolsa PIBIC Institucional concedida para a realização deste trabalho.

À Profa. Dra. Ana Cristina Vieira de Melo, por acreditar na minha capacidade e dar as ferramentas necessárias, assim como valiosos conselhos para a realização do projeto.

Aos professores do IME-USP, cujos ensinamentos dados durante a graduação foram de grande importância para a realização deste trabalho.

Aos meus amigos, aos meus pais, pelo apoio oferecido.

RESUMO

O trabalho desenvolvido é uma Iniciação Científica na área de Engenharia de Software, que consiste no desenvolvimento de um software a fim de criar um ambiente de geração de eventos espaciais em linguagem Java que tem por finalidade estimular a verificação e testes dos estudos de caso, a fim de produzir dados suficientemente próximos à realidade e assim poder atestar a qualidade dos sistemas espaciais.

Palavras-chave: Testes, Validação, Sistemas Críticos, Perturbação de Dados.

SUMÁRIO

1 INTRODUÇÃO **7**
1.1 Objetivo do trabalho 7
1.2 Organização dos capítulos 8

2 CONCEITOS PRELIMINARES **9**
2.1 JAVA 9
2.2 Máquina de estados finita 9

3 SWPDC **11**
3.1 Descrição 11
3.2 Especificações técnicas 11

4 SIMULADOR **14**
4.1 Descrição 14
4.2 Especificações técnicas 14
4.2.1 Linguagem e ambiente de desenvolvimento. 14
4.3 Implementação 15
4.3.1 Simulador como máquina de estados finitos. 15
4.3.2 Geração de temperaturas 16
4.3.3 Perturbação de dados 17
4.4 Classes implementadas. 18
4.4.1 Estado 18
4.4.2 Iniciação. 18
4.4.3 Nominal 19
4.4.4 Starting Class 23

5 CONCLUSÃO **24**
5.1 Resultados finais 24
5.2 Usabilidade 24

LISTA DE FIGURAS

Figura 1: Máquina de estados do semáforo 10

Figura 2: Relação entre os modos de operação do PDC e os componentes do SWPDC. 12

Figura 3: Fluxo de dados entre o simulador e o SWPDC. 15

Figura 4: Algoritmo para gerar temperaturas. 16

Figura 5: Perturbação em degraus. 17

CAPÍTULO 1

INTRODUÇÃO

Dado o mundo em que nós vivemos, onde temos uma maciça utilização de softwares, desde coisas ínfimas como o sistema de uma calculadora, até softwares grandes e robustos que controlam um robô na superfície de Marte e permite que o mesmo faça escolhas, a confiança no software em que nos voltamos é indispensável. Para isso acontecer, diversos tipos de verificação e testes precisam ser usados no desenvolvimento do sistema, visto que muitas vezes a falha pode gerar diversos tipos danos irreversíveis. No caso do software embarcado espacial, o alto valor de investimento, além do fato de que após o lançamento é impossível a modificação do software, sendo esta realizada apenas com a reprogramação do sistema e novo lançamento, torna esse procedimento inviável devido ao custo.

Esse projeto visa criar um software onde ambientes reais poderão ser gerados para colocá-los em teste sobre um software espacial embarcado que analisa as condições atmosféricas, para esse fim as especificações do sistema serão inicialmente estudada, assim como um aprofundamento do conhecimento na linguagem Java, estudos em cima dos dados já gerados pelo sistema embarcado, a fim de reunir estes conhecimentos para desenvolver o sistema gerador de testes.

1.1 OBJETIVOS DO TRABALHO

Busca-se com esse projeto um aprofundamento no estudo da linguagem Java, visto que esta é uma das linguagens mais utilizadas no mercado hoje em dia e que é pouco vista em classe, com o estudo

do Java podemos preparar o aluno para situações comumente vivenciadas no mercado de trabalho e no desenvolvimento de software que temos na atual fase da ciência da computação. Assim como o estudo do software já implementado, conhecido como SWPDC e buscar informações a respeito das condições atmosféricas com o intuito de posteriormente desenvolver um software que simule condições reais, para a verificação do software embarcado espacial, tentando aproximar ao máximo os resultados ao mundo real.

1.2 ORGANIZAÇÃO DOS CAPÍTULOS

Os capítulos estão organizados da seguinte maneira:

Parte Técnica

Conceitos Preliminares: contem uma breve explicação da teoria necessária para o entendimento do trabalho
Software:
Conclusão: apresentação dos resultados obtidos

CAPÍTULO 2

CONCEITOS PRELIMINARES

2.1 JAVA

A linguagem de programação Java foi originalmente produzida pela Sun Microsystems que foi iniciada por James Gosling e lançada em 1995.

Os principais diferenciais da linguagem Java são:

– Orientação a Objetos: Em Java tudo é um objeto e pode ser facilmente estendido ou reutilizado.
– Concisa e Simples: Java é fácil de ser entendido, implementado e utilizado.
– Robusta: Os programas em Java são confiáveis, pois reduz os imprevistos em tempo de execução, como por exemplo, variáveis que são automaticamente inicializadas e uso disciplinado de ponteiros.
– Portável: Os programas funcionam do mesmo jeito em qualquer ambiente, já que tudo roda na máquina virtual.
– Segura: A linguagem possui restrição de arquivos e manipulação segura dos ponteiros.
– Desalocação automática de objetos: libera o espaço de memória por meio de um coletor de lixo.

2.2 MÁQUINA DE ESTADOS FINITA

Uma máquina de estados finita é um modelo matemático, composto de estados, transições e ações e é usado para representar programas que podem exercer mais de um comportamento, cada

comportamento é um estado e o programa só pode estar em um estado de cada vez, o chamado estado atual.

Um estado armazena informações sobre o passado, isto é, ele reflete as mudanças desde a entrada no estado até o momento presente. Uma transição representa uma mudança de estado e precisa de uma condição para ocorrer. Uma ação é a descrição de uma atividade que deve ser realizada num determinado momento.

Se pegarmos um semáforo como exemplo, temos três estados: Verde, Amarelo e Vermelho, as transições ocorrem em decorrer do tempo de cada estado, então podemos supor que o semáforo fique 30s na cor verde, 30s na cor vermelha e 2s na cor amarela. Então temos o seguinte diagrama de estados:

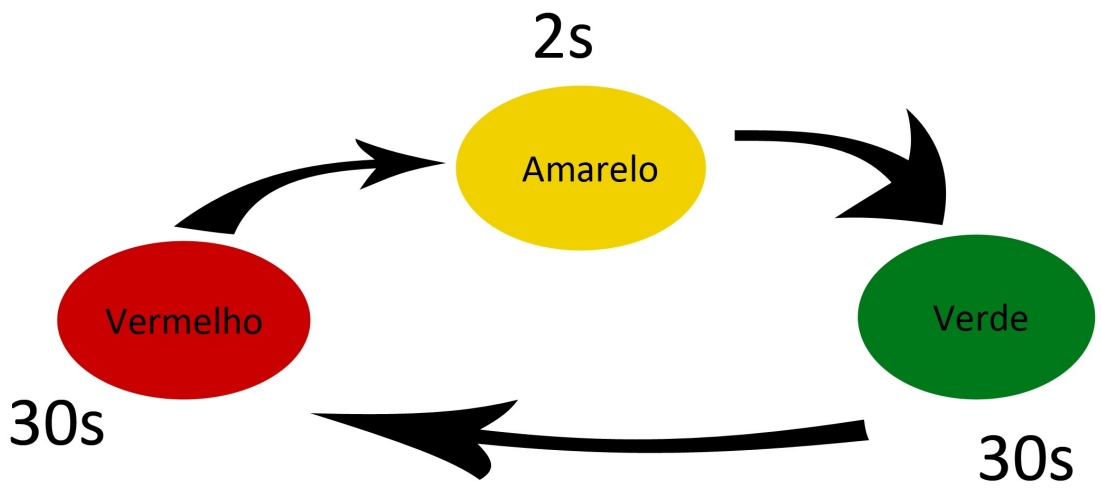


Figura 1: Máquina de estados do semáforo

CAPÍTULO 3

SWPDC

Neste capítulo, será apresentado o software SWPDC na qual o simulador foi desenvolvido para sua verificação. Os itens a serem abordados são:

- Descrição do SWPDC
- Especificações técnicas e Implementações das funcionalidades da ferramentas

3.1 DESCRIÇÃO

O SWPDC é uma ferramenta desenvolvida pela Camila Achutti que age como um software real embarcado para gerenciar dados espaciais. Ele foi desenvolvido no contexto de Qualidade de Software Embarcado em Aplicações Espaciais (QSEE) do Instituto Nacional de Pesquisas Espaciais (INPE).

Desenvolvido na plataforma Java por interesse do INPE para que acompanhe os novos rumos da pesquisa espacial e desenvolvimento de softwares de missão crítica.

3.2 ESPECIFICAÇÕES TÉCNICAS

O SWPDC é um software embarcado que já foi implementado em C e Assembly. O software possui quatro modos de operação do *Payload Data Handling Computer* (PDC), sendo eles: Iniciação, Segurança, Nominal e Diagnóstico. Tem, ainda, 5 processos componentes, sendo eles: Comunicação, Controle de Serviços de Aplicação, Gerenciamento de Dados, Gerenciamento de Estados e

Estado de Suporte. A relação entre esses modos de operação do PDC e os componentes do SWPDC estão ilustrados na figura abaixo:

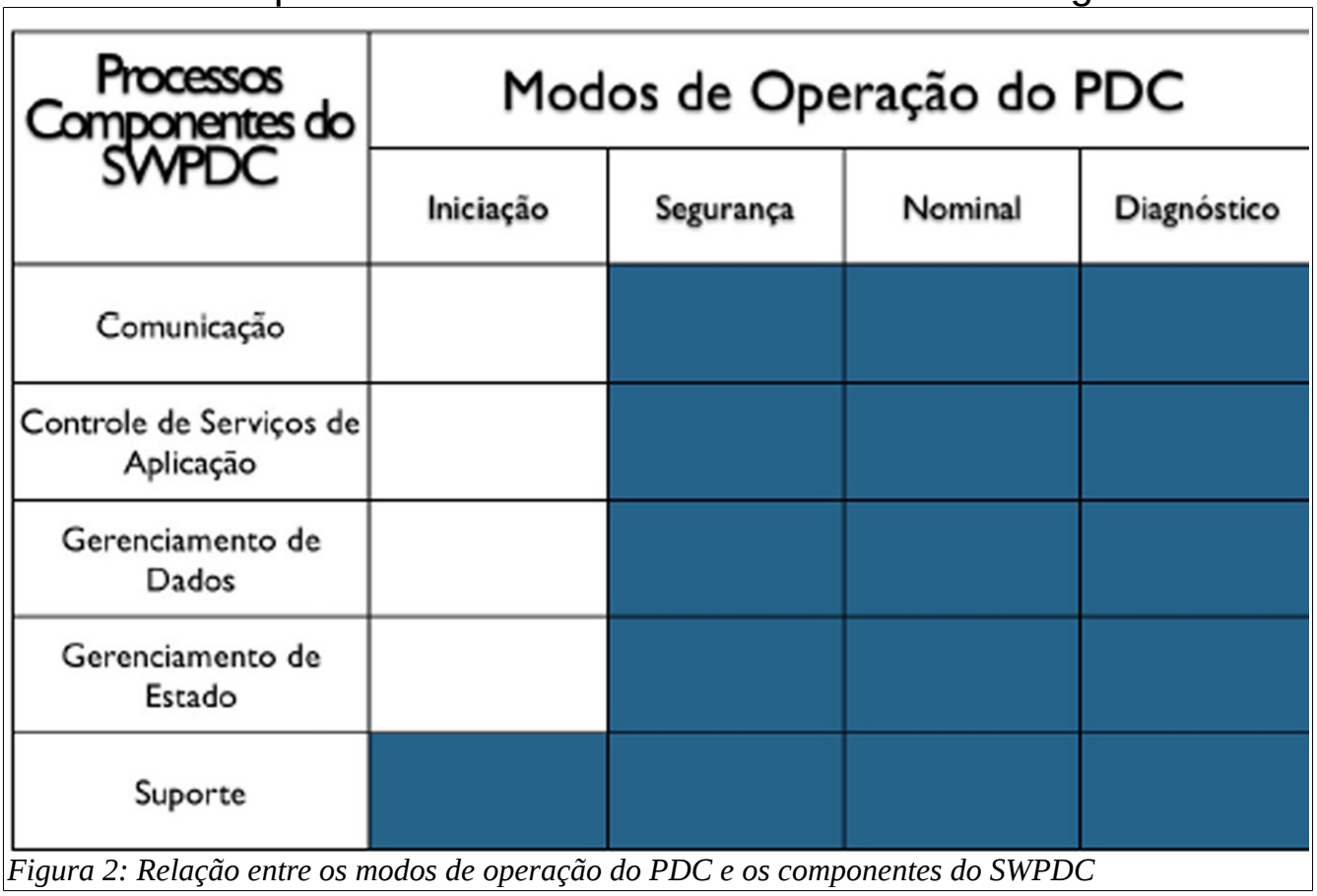


Figura 2: Relação entre os modos de operação do PDC e os componentes do SWPDC

O processo de comunicação citado acima é responsável por realizar a comunicação entre *On-Board Data Handling Computer* (OBDH) e os *Event Pre-Processors* (EPPs). Já o Controle de Serviços de Aplicações distribuiu os comandos recebidos do OBDH para os processos destinatários. O Gerenciamento de Dados é responsável pelo gerenciamento de memória com 5 tipos de dados existentes (científicos, diagnóstico, teste, descarga de memória e *housekeeping*), pela formatação dos dados para telemetria, pela carga e execução de programas na memória do PDC e pela descarga de memória do PDC. O processo de Suporte realiza funções de inicialização do SWPDC/PDC (são nessas funções que o bom funcionamento do hardware é testado) e aquisição e atuação em

canais discretos e no hardware do PDC. Por fim, o Gerenciamento de Estado realiza o gerenciamento do modo de operação do PDC, inclusive o processo de troca de modo de operação, assim como implementa mecanismos de tolerância a falhas no sistema, que deverão detectar, confinar e recuperar a ocorrência de erros, além disso deve gerar relatórios de eventos que os causaram.

CAPÍTULO 4

SIMULADOR

Neste capítulo, será apresentado o simulador o qual foi desenvolvido no trabalho. Os itens a serem abordados são:

- Descrição do Simulador
- Especificações técnicas e Implementações das funcionalidades da ferramentas
- Exemplos de uso

4.1 DESCRIÇÃO

O simulador desenvolvido neste projeto tem como objetivo verificar o SWPDC, rodando o simulador em paralelo com o software embarcado podemos gerar dados coerentes ao mesmo, ou usar técnicas de perturbação de dados com a finalidade testar o SWPDC.

4.2 ESPECIFICAÇÕES TÉCNICAS

Nesta seção, relatarei detalhes sobre a implementação do simulador e suas funcionalidades.

4.2.1 LINGUAGEM E AMBIENTE DE DESENVOLVIMENTO

A linguagem utilizada para a implementação do simulador foi o Java 6.0, e o ambiente de desenvolvimento escolhido foi o Eclipse 3.7.2.

4.3 IMPLEMENTAÇÃO

4.3.1 SIMULADOR COMO UMA MÁQUINA DE ESTADOS FINITA

O programa é uma máquina de estados finita, possuindo três estados: Desligado, Iniciação e Nominal. O estado Desligado é utilizado para manter o programa sem execução, ligando apenas quando o SWPDC fizer a requisição, o estado Iniciação é utilizado para inicializar todos as saídas e entradas do simulador, o estado Nominal é o estado em que instruções serão interpretadas e executadas.

O simulador foi pensado para ser implementado junto com o SWPDC, por isso a entrada do simulador é um arquivo de instruções do SWPDC em formato .txt. Já suas saídas podem ser enviadas ao SWPDC como o registroTemp.txt, que armazena as temperaturas geradas pelo simulador e o sysInfo.txt, que guarda as informações do sistema como o estado da máquina. Temos ainda o log.txt que registra os eventos relevantes durante a execução do sistema.

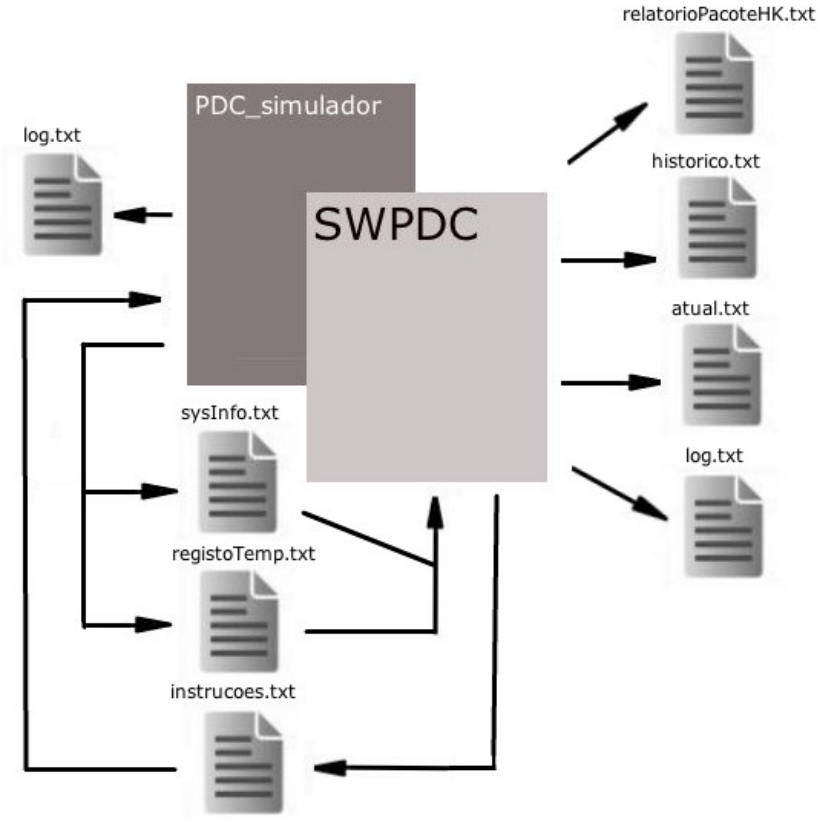


Figura 3: Fluxo de dados entre o simulador e o SWPDC

O fluxo de dados entre os programas pode ser vista pela imagem ao lado.

4.3.2 GERAÇÃO DE TEMPERATURAS

As temperaturas são geradas tentando chegar próximo aos eventos reais, por isso, o algoritmo usado para gerá-las consiste primeiramente de uma temperatura de início, a qual podemos nos basear para criar os outros registros.

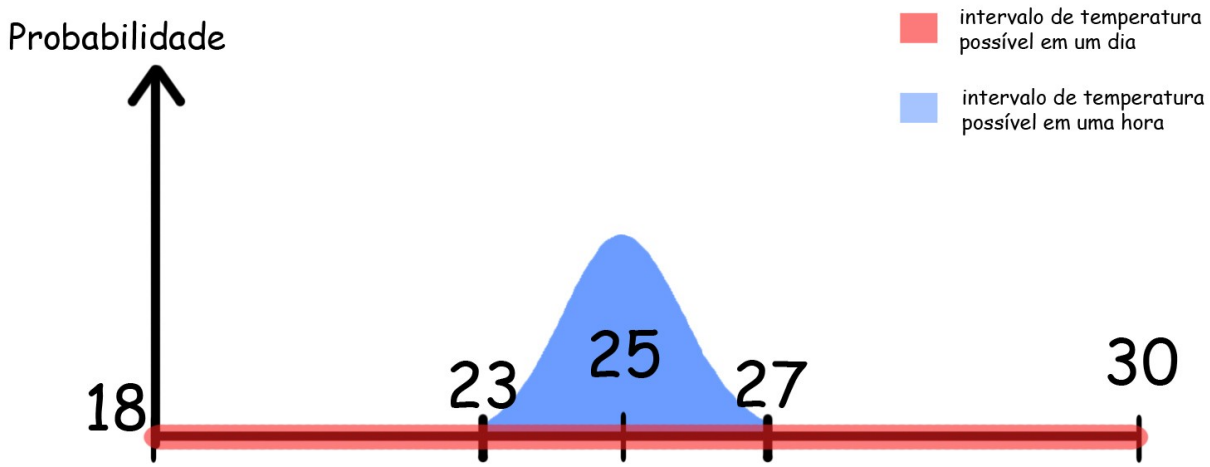
Supomos que cada solicitação registra doze temperaturas dentro de um dia, tendo como intervalo entre elas o período de uma hora.

Primeiramente precisamos colocar limites diários, que seriam a máxima e a mínima, assim nenhum valor pode ultrapassar esses limites. Verificando os bancos de dados do Climatempo podemos supor que a diferença entre máxima e mínima está de oito a doze graus Celsius, porém esse valor é ajustável.

A cada hora analisamos o valor do último registro de temperatura e criamos um espectro de valores possíveis que a mesma pode variar dentro dessa hora, o tamanho desse espectro como padrão é cinco, porém como as outras variáveis, ela é ajustável.

O espectro segue uma distribuição Gaussiana, portanto assim conseguimos fazer com que a probabilidade da temperatura se manter é maior do que a probabilidade da temperatura variar muitos graus.

A figura abaixo conseguem exemplificar o algoritmo:



16 *Figura 4: Algoritmo para gerar temperaturas*

4.3.3 PERTURBAÇÃO DE DADOS

Usamos alguns métodos de perturbação de dados, justamente para verificar o SWPDC, são eles:

– Perturbação em degraus: o algoritmo gera uma maior amplitude nas amostras.

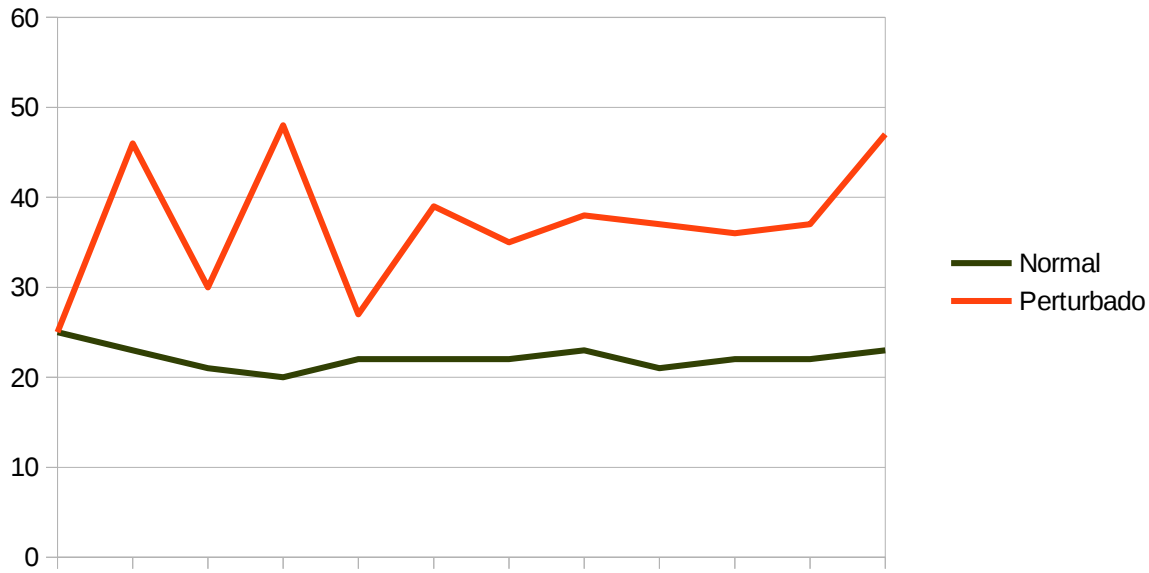


Figura 5: Perturbação em degraus

– Testar limites: As amostras são os limites de temperaturas possíveis.
– Divisão: Seja n uma amostra, após a perturbação a amostra será 1 ÷ n.
– Multiplicação: Seja n uma amostra, após a perturbação a amostra será n x n.
– Negação: Seja n uma amostra, após a perturbação a amostra será -n.
– Valor absoluto: Seja n uma amostra, após a perturbação a amostra será |n|.

4.4 CLASSES IMPLEMENTADAS

4.4.1 ESTADO

Utilizamos a interface Estado que implementamos nas classes Nominal e Iniciação.

4.4.2 INICIAÇÃO

Essa classe inicializa o sistema, deixando-o pronto para a execução nominal.

```
package Simulador;

import static Library.Escrever.escrever;

public class Iniciacao implements Estado
{

    /*
     * O metodo inicializa o sistema, pegando o diretorio utilizado pelo projeto
     * e usamos ele como diretorio para nosso sistema. Retornamos true caso a
     * inicializacao foi um sucesso
     */
    boolean inicializaDiretorio()
    {
        // Pegamos o diretorio onde esta localizado o projeto em java
        try {
            StartingClass.dir = new File(System.getProperty("user.dir"));
        } catch (Exception e) {
            System.out.println("Erro ao localizar diretório");
            e.printStackTrace();
            return false;
        }

        return true;
    }
```

```
boolean inicializaLog()
{
    try {

        // Criamos o arquivo log.txt
        StartingClass.log = new File(StartingClass.dir, "log.txt");
        if (!StartingClass.log.exists())
            StartingClass.log.createNewFile();

        // Inicializamos o FileWriter e o PrintWriter do log
        StartingClass.logFileWriter = new FileWriter(StartingClass.log,
                true);
        StartingClass.logPrintWriter = new PrintWriter(
                StartingClass.logFileWriter);

    } catch (Exception e) {
        System.out.println("Erro ao inicializar log");
        e.printStackTrace();
        return false;
    }

    return true;
}
```

No resto da classe, seguimos a mesma lógica do código apresentado acima, continuamos inicializando as variáveis, criando os *FileWriters* das saídas e criando os *buffers* de leitura para as entradas.

4.4.3 NOMINAL

A classe nominal é a classe onde calculamos as temperaturas, seguindo o algoritmo descrito acima.

```
package Simulador;

import java.util.Random;

public class Nominal implements Estado
{
    static Random rand = new Random();
```

```java
 private static int MENOR_TEMPERATURA_POSSIVEL = 0;
 private static int MAIOR_TEMPERATURA_POSSIVEL = 40;
 private static int TAMANHO_INTERVALO_HORARIO = 5;
 private static int TAMANHO_INTERVALO_DIARIO_MENOR = 8;
 private static int TAMANHO_INTERVALO_DIARIO_MAIOR = 12;

 private static float[] temperaturas = new float[12];
 private float[] intervaloCadaHora = new float[TAMANHO_INTERVALO_HORARIO];
 private static int minima, maxima, tamanhoIntervalo;
 private static float tempAtual = 25; // começa com uma temperatura ambiente

 boolean obterAmostras()
 {
     try {
         String temps = "";
         for (int i = 0; i < temperaturas.length; i++)
             temps += String.format("%.1f, ", temperaturas[i]);

         escrever(StartingClass.registroPrintWriter, "[numAm] "
                 + StartingClass.numAmostra++ + " " + "[dados] " + " " + temps);
     } catch (Exception e) {
         escrever(StartingClass.logPrintWriter, e.getMessage());
         return false;
     }

     return true;
 }


/*
 * O metodo geraTemperaturas usa uma função gaussiana para gerar as temperaturas
 *
 */
void geraTemperatura(int flag)
{
    int cont = 0;
    int minLocal, maxLocal;
    float valorPerturbado = tempAtual;

    tamanhoIntervalo = rand.nextInt(TAMANHO_INTERVALO_DIARIO_MAIOR
            - TAMANHO_INTERVALO_DIARIO_MENOR)
            + TAMANHO_INTERVALO_DIARIO_MENOR; // Intervalo entre 8 e 12
    minima = (int)tempAtual - tamanhoIntervalo / 2;
    maxima = minima + tamanhoIntervalo;

    if (minima < MENOR_TEMPERATURA_POSSIVEL) {
        minima = MENOR_TEMPERATURA_POSSIVEL;
        maxima = minima + tamanhoIntervalo;
    }
    if (maxima > MAIOR_TEMPERATURA_POSSIVEL) {
        maxima = MAIOR_TEMPERATURA_POSSIVEL;
        minima = maxima - tamanhoIntervalo;
    }
```

```
while (cont < temperaturas.length) {
    if(flag > 0)
    {
        temperaturas[cont++] = valorPerturbado;
    }else {
        temperaturas[cont++] = tempAtual;
    }


    /* aqui trabalhamos o caso do numero ser par */
    if(TAMANHO_INTERVALO_HORARIO % 2 == 1)
    {
        minLocal =  (int)tempAtual - TAMANHO_INTERVALO_HORARIO /2;
        maxLocal = (int)tempAtual + TAMANHO_INTERVALO_HORARIO /2;
    } else {
        minLocal = (int)tempAtual - TAMANHO_INTERVALO_HORARIO /2;
        maxLocal = (int)tempAtual + TAMANHO_INTERVALO_HORARIO /2 - 1;
    }


    if(minLocal < minima) {
        minLocal = minima;
        maxLocal = minLocal + TAMANHO_INTERVALO_HORARIO - 1;
    }
    if(maxLocal > maxima) {
        maxLocal = maxima;
        minLocal = maxLocal - TAMANHO_INTERVALO_HORARIO - 1;
    }

    for(int i = 0; i < TAMANHO_INTERVALO_HORARIO; i++) intervaloCadaHora[i] = minima + i;

    /*
     * Escolhemos a temperatura aleatoriamente, seguindo uma distribuiçao normal
     * que sorteia um indice para o vetor de temperaturas possiveis ára aquela hora
     */

    int x = (int) ((rand.nextGaussian() + 3 ) * TAMANHO_INTERVALO_HORARIO / 6);
    if(x >= TAMANHO_INTERVALO_HORARIO) x = TAMANHO_INTERVALO_HORARIO - 1;

    if(intervaloCadaHora[x] > MAIOR_TEMPERATURA_POSSIVEL) tempAtual = MAIOR_TEMPERATURA_POSSIVEL;
    else if(intervaloCadaHora[x] < MENOR_TEMPERATURA_POSSIVEL) tempAtual = MENOR_TEMPERATURA_POSSIVEL;
    else tempAtual = intervaloCadaHora[x];


    /* Perturbaçao em degrau
     * Geramos maior maior amplitude entre as amostras
     */
    if(flag == 1)
    {
        if(maxima - intervaloCadaHora[x] > intervaloCadaHora[x] - minima)
```

```
        {
            if(maxima - intervaloCadaHora[x] > intervaloCadaHora[x] - minima)
            {
                valorPerturbado = intervaloCadaHora[x] + rand.nextInt(TAMANHO_INTERVALO_HORARIO
                                                                      * TAMANHO_INTERVALO_HORARIO);
            } else {
                valorPerturbado = intervaloCadaHora[x] -  rand.nextInt(TAMANHO_INTERVALO_HORARIO *
                                                                      TAMANHO_INTERVALO_HORARIO);
            }
        }

        /*
         *  Testar os limites
         */
        if(flag == 2)
        {
            switch(rand.nextInt(2))
            {
            case 0:
                valorPerturbado = MENOR_TEMPERATURA_POSSIVEL; break;
            case 1:
                valorPerturbado = MAIOR_TEMPERATURA_POSSIVEL; break;
            default:
                valorPerturbado = 0;
            }
        }


        /* Divide
         */
        if(flag == 3)
            valorPerturbado = 1 / tempAtual;

        /* Multiply
         */
        if(flag == 4)
            valorPerturbado = tempAtual * tempAtual;

        /* Negative
         */
        if(flag == 5)
            valorPerturbado = -tempAtual;

        /*Absolute
         */
        if(flag == 6)
            valorPerturbado = Math.abs(tempAtual);


    }

}
```

```
    void geraTemperaturas(int menorTemperaturaPossivel, int maiorTemperaturaPossivel,int tamanhoIntervaloHorario,
            int tamanhoIntervaloDiarioMenor, int tamanhoIntervaloDiarioMaior, int flag)
    {
        MENOR_TEMPERATURA_POSSIVEL = menorTemperaturaPossivel;
        MAIOR_TEMPERATURA_POSSIVEL = maiorTemperaturaPossivel;
        TAMANHO_INTERVALO_HORARIO = tamanhoIntervaloHorario;
        TAMANHO_INTERVALO_DIARIO_MENOR = tamanhoIntervaloDiarioMenor;
        TAMANHO_INTERVALO_DIARIO_MAIOR = tamanhoIntervaloDiarioMaior;
        geraTemperatura(0);
    }

}
```

4.4.4 STARTING CLASS

Starting Class é a classe que usamos como main para sintetizar as funções das outras classes.

Nesta função apenas lemos as instruções do arquivo correspondente e colocamos as instruções em uma pilha, identificamos qual instrução está sendo solicitada e a executamos.

CAPÍTULO 5

CONCLUSÃO

Neste capítulo os resultados finais do trabalho serão apresentados

5.1 RESULTADOS FINAIS

O simulador, produto final deste trabalho, consegue realizar todos as funcionalidades descritas. Suas principais características são:

- Consegue gerar temperaturas condizentes com uma leitura real, seguindo um algoritmo
- Utiliza diversas técnicas de perturbação de dados para conseguir verificar possíveis erros no SWPDC
- Os modelos de geração de temperatura são flexíveis, podendo funcionar tanto para temperaturas terrestres tanto para temperaturas estratosféricas, por exemplo.

5.2 USABILIDADE

O software realiza seu objetivo principal, de verificar o SWPDC, mas também abre espaço para outros projetos, o IPEN (Instituto Nacional de Pesquisas Espaciais) se ofereceu no projeto pois viu que não possuía nenhum simulador suficientemente realista para utilizar em suas pesquisas, sendo essas uma das outras possíveis utilidades do software desenvolvido neste projeto.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Bruce Eckel. *Thinking in Java*. Prentice Hall, 2007.
[2] Alfredo Goldman, Fabio Kon, Paulo J. S. Silva. *Introdução a Ciência da Computação em Java e Orientação a Objetos*. São Paulo IME-USP, 2006.
[3] Deitel, Harvey M. e Deitel, Paul J. *Java: Como Programar*. Bookman, 2004.
[4] Eclipse. Disponível em: <http://www.eclipse.org/>. Acessado em: 07 de maio de 2013.
[5] Paulo Blauth Menezes. *Linguagens Formais e Autômatos*. Bookman, 2011.
[6] Camila Achutti. *Verificação e validação de softwares de missão crítica*. Monografia. Universidade de São Paulo, São Paulo, Brasil, 2014.
[7] Jeff Offutt & Wuzhi Xu. *Generating Test Cases for Web Services Using Data Perturbation.* Department of Information and Software Engineering, George Mason University, Fairfax, VA, USA.