

# Modelagem e Simulação - INE 5425

## Programa de Simulação em Linguagem de Propósito Geral

Evandro Chagas Ribeiro da Rosa

25 de outubro de 2017

## 1 Manual

A interface de usuário é dividida em 3 partes.

### 1.1 Configuração

The screenshot shows the 'Configuração' (Configuration) window of the simulation software. It is divided into two main sections: 'Configuração' on the left and 'Relatório' (Report) on the right. The 'Configuração' section contains several input fields for parameters, each with a label and a value. A red rectangle highlights the configuration section. The 'Relatório' section displays various statistics and performance metrics.

Parameter	Value
TEC1	cons 2
TS1	expo 3
TFE1	0
TEF1	norm 10 2
TF1	tria 2 5 9
TEC2	unif 1 5
TS2	norm 3 0.5
TFE2	512
TEF2	cons 25
TF2	tria 5 8 12
Tempo de simulação	5000

Buttons: Executar, Avançar, Terminar simulação, Parar, Cancelar simulação.

Aqui são configurados todos os parâmetros da simulação. Os parâmetros que podem receber uma função de probabilidade devem corresponder com os seguintes padrões:

- expo media: Exponencial.
- norm media dp: Normal.
- tria min moda max: Triangular
- unif min max : Uniforme;
- cons valor: retorna sempre valor.

Usar “ . ” ao invés de “ , ” para separar a parte inteira da fracionária.

Siglas:

- TEC1: Tempo entre chegadas do Servidor 1.
- TS1: Tempo de serviço do Servidor 1.
- TFE1: Tamanho da fila de entrada do servidor 1, 0 significa ilimitado.
- TEF1: Tempo entre falhas do Servidor 1.
- TF1: Tempo de falha do Servidor 1
- TEC2: Tempo entre chegadas do Servidor 2.
- TS2: Tempo de serviço do Servidor 2.
- TFE2: Tamanho da fila de entrada do servidor 2, 0 significa ilimitado.
- TEF2: Tempo entre falhas do Servidor 2.
- TF2: Tempo de falha do Servidor 2

Apos tudo configurado clique em “Executar” para começar a simulação.

## 1.2 Execução

The screenshot shows the simulation software interface. The 'Execução' section is highlighted with a red box. It contains a text input field with the value '1', an 'Avançar' button, a 'Terminar simulação' button, and a 'Parar Cancelar simulação' button. The background shows the 'Configuração' and 'Relatório' sections.

Essa parte será usado durante a simulação.

- Botão “Avançar”: Avança a simulação no tempo conforme especificado na caixa de texto a esquerda (usar apenas numero separado por “ . ” ao invés de “ , ”).
- Botão “Terminar Simulação”: Executa a simulação até o final.
- Botão “Parar / Cancelar Simulação”: Para o avanço da simulação, caso já esteja parada cancela a simulação liberando a parte de configuração.

## 1.3 Relatório

The screenshot shows the simulation software interface. The 'Relatório' section is highlighted with a red box. It displays various statistics including 'Número Médio de Entidades nas Filas', '%Taxa Média de Ocupação dos Servidores', 'Tempo Médio de uma Entidade na Fila', 'Tempo Médio no Sistema', 'Contadores', 'Falhas', and 'Contagem de trocas e perdas'. The background shows the 'Configuração' and 'Execução' sections.

Durante a execução as estatísticas serão mostradas aqui, esse tela será atualizada a cada avanço no tempo.

## 2 Implementação

A implementação foi feita em C++11 utilizando Qt para interface gráfica.

### 2.1 Eventos

A gerencias dos eventos é feita pela classe `mod::Oraculo`, com os seguintes atributos:

- `tempo_`: guarda o tempo atual da simulação
- `tempo_total`: tempo que a simulação deve terminar.

- `events`, do tipo `std::multiset<Event>`, guarda os eventos ordenados pelo tempo em uma árvore *RB*.

Os eventos são da classe `mod::Event` que contem:

- `time_`: tempo em que o evento deve ser executado, utilizado para atribuir uma relação de ordem entre os eventos.
- `text`: uma descrição do evento, utilizado para *debugging*.
- `call`: do tipo `std::function<void()>`, todo evento é um método que recebe e retorna `void`;

### 2.1.1 Adicionar Evento

Para adicionar um evento basta chamar a função `mod::Oraculo::add_event` passando o método que sera chamado no evento (`F call`), tempo no qual o evento será chamado (`double time`) e a descrição do evento (`std::string text`). O Evento será criado com esse parâmetros e adiciono em `events`.

### 2.1.2 Executar Eventos

A função `mod::Oraculo::run` recebe como argumento o tempo limite de execução (`double limit`), os eventos serão chamados em ordem em quanto não estourar o tempo limite ou o tempo total de execução.

## 2.2 Funções de probabilidade

As funções de probabilidade são geradas pelo método `mod::parser` que recebe um `std::string` e retorna um `func::func` (Aka `std::function<double()>`). A `std::string text` deve seguir os seguintes padrões:

- `expo media`: Exponencial.
- `norm media dp`: Normal.
- `tria min moda max`: Triangular
- `unif min max`: Uniforme;
- `cons valor`: retorna sempre valor.

## 2.3 Elementos da modelagem

Todos elementos da modelagem estão conectados na classe `Estado`, a baixo segue os atributos da classe.

- **Entidade**: pode ser do Tipo `um` ou `dois`. Não é um atributo da classe `Estado`, mas é um elemento dinâmico que transita pelo sistema. Armazena o tempo em fila e tempo no sistema.

Atributos da classe:

- `double begin`: tempo em que a entidade entrou no sistema.
- `double fila_begin, fila_end`: tempo que a entidade entrou e saio da fila.
- `Tipo tipo_`: tipo da entidade.

- **Saida saida**: Remove as Entidades do sistema e responsável pelas estatísticas de: *Sairam do Sistema*, *Tempo Médio no Sistema* e *Tempo Médio de uma Entidade na Fila*.

Atributos da classe:

- `unsigned um, dois`: conta quantas entidades sairão do sistema.
- `double tempo_um, tempo_dois`: somatório do tempo de cada entidade que saiu do sistema.

- `double tempo_em_fila_um, tempo_em_fila_dois`: somatório do tempo em fila de cada entidade que saiu do sistema.
- **Servidor servidor1, servidor2**: Move as entidades para `saida` e responsável pelas estatísticas de *Número Médio de Entidades nas Filas*, *%Taxa Média de Ocupação dos Servidores*, *Numero de falhas*, *Tempo em falha* e *%Tempo em falha*.

Atributos da classe:

- `func::func ts, tef, tf`: tempo de serviço, entre falhas e em falha.
- `Saida &saida`: Referência para `saida`.
- `std::queue<Entidade> fila`: fila de entidades.
- `unsigned tfe`: tamanho da fila de entrada.
- `bool em_falha`: guarda se o servidor esta em falha.
- `unsigned n_falhas`: contador de falhas do servidor.
- `bool ocupado`: guarda se o servidor esta ocupado.
- `double t_servico, t_falha`: tempo total ocupado e em falha.
- `double begin_ocupado, begin_falha`: guarda o tempo da ultima vez que, mudou de não ocupado para ocupado e entrou em falha, usado para controle interno.
- `double mfila`: media de Entidades na fila.
- `double ponderacao`: utilizado para calcular a media ponderada de Entidades na fila, controle interno.
- `double last_time`: ultima vez que a media de Entidades na fila foi calculada, controle interno.
- `bool init`: utilizado para `ponderacao` seja atribuída apenas uma vez.

Eventos:

- *Sair do servidor* que passa um elemento para `Saida` e chama `mod::Servidor::executar_proximo` que pode gerar o próximo evento *Sair do servidor*.
- *Servidor em falha* coloca o `Servidor` em modo de falha e gera o evento a seguir.
- *Servidor voltou* tira o `Servidor` de modo de falha, chama `mod::Servidor::executar_proximo` e, `mod::Servidor::programar_falha` para gerar o próximo evento *Servidor em falha*.
- **Chegada chegada1, chegada2**: O Estado possui duas entradas, um para cada tipo de entidade. Responsável pela contagem de *Entraram no Sistema*, *Entidades perdidas* *Trocas efetuadas*.

Atributos da classe:

- `func::func tec`: tempo entre chegadas
- `Entidade::Tipo tipo`: tipo de entidade que é gerado
- `Servidor &primario, &secundario`: servidor principal e secundário, para caso o primeiro não esteja disponível.
- `unsigned entradas, trocas`: contagem de quantas Entidades entraram e quantas foram para o servidor secundário.
- `unsigned perdas`: contagem de quantas Entidades não conseguiram nem entrar na fila do servidor secundário.

Evento:

- **Evento Chegada**: Tenta adicionar uma entidade no `Servidor` e gera o próximo **Evento Chegada**.

### 2.3.1 Construtor

O construtor esclarece como as classes são relacionadas.

```
Estado(std::string tec1, std::string ts1,
std::string tef1, std::string tf1,
unsigned tfe1, std::string tec2,
std::string ts2, std::string tef2,
std::string tf2, unsigned tfe2,
double tempo_total)
:oraculo{tempo_total}, saida{oraculo},
servidor1{oraculo, func::parse(ts1),
func::parse(tef1),
func::parse(tf1),
saida, tfe1},
servidor2{oraculo, func::parse(ts2),
func::parse(tef2),
func::parse(tf2),
saida, tfe2},
chegada1{oraculo, func::parse(tec1),
Entidade::um, servidor1, servidor2},
chegada2{oraculo, func::parse(tec2),
Entidade::dois, servidor2, servidor1}
```