

**Universidade Federal de Santa Catarina
Departamento de Informática e Estatística**

**ATUALIZAÇÃO DO GRIDSIMULATOR
Uma ferramenta integrada a um ambiente grid
para desenvolvimento de modelos de simulação
discreta.**

**João Gabriel Crema
Leonardo Maruyama de Carvalho**

**Florianópolis
Fevereiro de 2007**

João Gabriel Crema
Leonardo Maruyama de Carvalho

Projeto de pesquisa do trabalho de conclusão de curso apresentado à Universidade Federal de Santa Catarina, como parte dos requisitos para a obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Prof. Paulo José de Freitas Filho

Florianópolis
Fevereiro de 2007

João Gabriel Crema
Leonardo Maruyama de Carvalho

Banca Examinadora

Paulo José de Freitas Filho, Dr.
Orientador

Membros:
Gian Ricardo Berkenbrock
Mário Antonio Ribeiro Dantas, Dr.

Sumário

Lista de figuras	6
Lista de quadros	7
Lista de tabelas	8
Resumo	9
1 Introdução	10
2 Projeto	12
2.1 Tema.....	12
2.2 Objetivos.....	12
2.2.1 Objetivos Gerais	12
2.2.2 Objetivos Específicos	12
3 Simulação	13
3.1 Definição	13
3.2 Modelos	15
3.2.1 Classificação de modelos de simulação	15
3.3 Tempo	17
4 GridSimulator	18
4.1 Componentes de simulação	21
4.2 Editor visual	25
4.3 Processador de modelos	26
4.3.1 Expressões	27
4.3.1.1 Expressões booleanas	29
4.3.2 Componentes de execução	30
5 Atualização do GridSimulator	32
5.1 Implementação de suporte multi-idioma.....	32
5.1.1 Desenvolvimento.....	33
5.1.2 Exemplos.....	38
5.2 Geração de relatórios	41
5.2.1 Introdução	41
5.2.2 Relatório em formato texto	43
5.2.3 Relatório em formato HTML.....	45
5.3 GridSimulator Launcher.....	48
6 Conclusão	51
6.1 Considerações Finais.....	51

6.2 Sugestões para trabalhos futuros.....	51
Referências	52
Referências consultadas	53

Lista de Figuras

4.1	Esquema do GridSimulator	19
4.2	Interação geral [BER 05]	20
4.3	Diagrama de atividades do processo de atribuição [BER 05].....	21
4.4	Diagrama de atividades do processo de decisão [BER 05].....	22
4.5	Diagrama de atividades do processo de geração [BER 05].....	22
4.6	Diagrama de atividades do processo de liberação de recurso(s) [BER 05]..	23
4.7	Diagrama de estado do recurso [BER 05].....	23
4.8	Diagrama de atividade do processo de requisição do(s) recurso(s) [BER 05]..	24
4.9	Diagrama de atividades do processo de retardo [BER 05].....	24
4.10	Diagrama de atividades do processo de saída [BER 05].....	25
4.11	Área de uso do GridSimulator Editor [BER 05].....	26
5.1	Modelo de arquivos com as strings traduzidas.....	35
5.2	Caixa de seleção de idioma (sistema em português).....	37
5.3	Caixa de seleção de idioma (sistema em inglês).....	37
5.4	Componente na versão original.....	38
5.5	Componente com suas propriedades traduzidas.....	39
5.6	Hints traduzido e na versão original.....	39
5.7	Caixa de alerta traduzida.....	39
5.8	Menu e itens traduzidos.....	40
5.9	Relatório original em linha de comando.....	41
5.10	Relatório em modo texto.....	45
5.11	Relatório em página HTML.....	47
5.12	Janela para busca de arquivo.....	49
5.13	Caminho do arquivo selecionado exibido.....	49
5.14	Interface inicial do GridSimulator Launcher.....	50

Lista de Quadros

4.1 Distribuições estatísticas [BER 05]	29
4.2 Estatísticas coletadas pelos componentes [BER 05]	30

Lista de Tabelas

5.1 Código de idiomas.....	34
5.2 Código de países.....	34

Resumo

Este trabalho propõe o desenvolvimento e a implementação de novas funções e componentes para uma ferramenta que possibilita a criação de modelos de simulação discreta e a execução destes em um grid computacional. Essa ferramenta foi desenvolvida por um aluno do PPGCC-UFSC, Gian Ricardo Berkenbrock, em seu trabalho de mestrado. O trabalho ora proposto vem complementar a pesquisa e o desenvolvimento já iniciado. Este ambiente, preliminarmente denominado GridSimulator, é composto de duas partes: editor e processador de modelos.

Entre as novidades desenvolvidas para o GridSimulator estão: suporte multi-idioma, onde agora também é possível trabalhar com o ambiente de criação de modelos em português; novos formatos de saída de relatório, tais como: relatório em página HTML e em arquivo texto (.txt) e o GridSimulator Launcher, que torna mais prático a execução do processador de modelos, uma vez que agora o usuário trabalha em modo gráfico, tanto para executar o processamento como para escolher uma das opções de formato do relatório de saída.

As novas funcionalidades foram desenvolvidas utilizando a tecnologia Java Standard Edition, C++ e Delphi, ao final deste trabalho, foi obtido um resultado que torna o GridSimulator uma ferramenta palusível de ser utilizada em ambiente acadêmico.

Palavras-Chaves: *Simulação Discreta, GridSimulator, Internacionalização, Localização.*

1 Introdução

Cada vez mais a relação custo-benefício é priorizada em qualquer ramo de atividade e conhecer previamente os caminhos que serão seguidos torna-se quase uma obrigação ao realizar-se um projeto.

Uma maneira de prever esses caminhos é a utilização da simulação discreta, através da qual é possível implementar praticamente qualquer situação da vida real na tela de um computador. Sendo assim, com a realização do projeto através da simulação, seria possível antever a sua viabilidade, com um menor custo.

Nos últimos anos a globalização tem exigido que softwares estejam adaptados aos vários locais ao redor do globo onde são executados. Com isso dá-se uma grande importância às atividades de internacionalização (I18N) e localização (L10N) de software.

As atividades de I18N e de L10N empregam tecnologias de diversas naturezas para se tornarem cada vez mais efetivas.

Outro ponto sempre priorizado é a produtividade. Facilitar a realização de tarefas é cada vez mais necessário. No caso específico dos softwares, ter interfaces intuitivas, que facilitem o trabalho do dia-a-dia, agilizam a conclusão das mesmas.

Em cima dessas tendências foi feito um estudo e uma análise sobre o GridSimulator e simulação discreta. Onde a partir disto novas funcionalidades foram idealizadas com o intuito de serem agregadas à ferramenta.

E com isso, este ambiente deverá constituir um núcleo de software bastante completo para ser, inicialmente, utilizado em nosso ambiente acadêmico.

2 Projeto

2.1 Tema

Atualização do GridSimulator - uma ferramenta integrada a um ambiente grid para desenvolvimento de modelos de simulação discreta.

2.2 Objetivos

2.2.1 Objetivo Geral

Aprofundar nossos conhecimentos na área de simulação discreta, suas formas de implementação e utilidades em contextos reais. Para isso faremos o estudo e melhoramento de uma ferramenta gráfica desenvolvida, com o intuito de criar modelos de simulação discreta. Em consequência disto, temos como objetivo secundário aprimorar nossas técnicas de programação e desenvolvimento.

2.2.2 Objetivos Específicos

Com este trabalho pretende-se:

- a) melhorar o processador para a simulação para que possa suportar características que dêem mais opções ao usuário.
- b) implementar suporte a multi-idioma.
- c) otimizar a execução dos componentes e diminuir a quantidade de eventos gerados.
- d) desenvolver outros tipos de relatórios de saída.

3 Simulação

3.1 Definição

Simulação ou ato de simular consiste na imitação de algum processo ou circunstância do mundo real. Reunindo características e comportamentos pertinentes de um sistema físico ou abstrato geralmente torna-se possível a simulação.

A alta capacidade de se adaptar e ser capaz de se adequar a qualquer situação do mundo real são fatores importantes para a larga utilização da simulação. Esta tem coberto várias áreas de aplicações, podemos citar como exemplos projetos de engenharia, sistemas econômicos, controle de fluxo, sistemas de transportes, projetos aeroespaciais, etc. Ou seja, a simulação pode abranger desde pequenos projetos locais, até um projeto altamente custoso onde seria inviável a sua realização.

Em computação, simulação consiste em empregar determinadas técnicas matemáticas em computadores digitais, as quais permitem imitar o funcionamento de, praticamente, qualquer tipo de operação ou processo (sistemas) do mundo real. [dFF 01]

A utilização da simulação pode prever através de modelos, acontecimentos que poderiam ser produzidos por alterações ou utilização de outros métodos na operação do sistema.

Casos onde a simulação é amplamente utilizada:

- a) **Quando o sistema real ainda não existe** - nesse caso, o futuro sistema poderia ser totalmente planejado através da simulação. Podemos dar como exemplo, a criação de uma nova fábrica, um novo hospital ou até mesmo um sistema de vendas pela Internet.

- b) **Quando executar o sistema real é custoso** - projetos de grande porte, implantação de novos equipamentos podem se tornar dispendiosos, por isso há necessidade de um modelo de simulação, onde seja possível analisar o seu comportamento com muitos menos custos e assim prever os do sistema real. Dessa forma há a possibilidade de antever se será apropriado ou não a realização do mesmo.

- c) **Quando não é apropriado experimentar com o sistema real** - utilizado principalmente em situações onde não seria possível o teste real. Desastres ecológicos, aéreos, imprevisibilidade de uma usina nuclear são alguns exemplos. Toda a logística para acionamento e atuação dos serviços envolvidos seriam modelados e tratados no computador sem nenhum risco.

3.2 Modelos

Num processo de simulação, uma das etapas mais importantes é a modelagem do sistema em estudo. Nessa etapa procura-se copiar e criar dados e possíveis formas de atuação do sistema real. Assim possibilitando uma abrangência maior de possíveis casos do sistema. Sendo possível a criação de condições para uma análise mais profunda.

Segundo Freitas [dFF 01], a modelagem pressupõe um processo de criação e descrição, envolvendo um determinado grau de abstração que, na maioria das vezes, acarreta numa série de simplificações sobre a organização e o funcionamento do sistema real. As relações lógicas ou matemáticas, que no seu conjunto, constituem o que se denomina de modelos.

3.2.1 Classificação de modelos de simulação

Freitas [dFF 01] apresenta uma classificação de modelos segundo seu propósito:

a) Modelos voltados à previsão - utilizados para prever o estado de um sistema em um tempo futuro. É baseado em variáveis de seu comportamento e como se comportará no decorrer do tempo.

b) Modelos voltados à investigação - utilizados para busca de informações e desenvolvimento de hipóteses sobre o

comportamento de sistemas. Os experimentos recaem sobre as reações dos modelos a estímulos normais e anormais.

c) Modelos voltados à comparação - usados para avaliar os efeitos ao efetuar trocas nas variáveis de controle. É feita comparação entre as simulações.

Kelton [KEL 02] apresenta a seguinte classificação:

a) Estático versus Dinâmico - no modelo estático o sistema é avaliado num determinado ponto do tempo, e este não avança de forma natural. Ao contrário do dinâmico, que é analisado de acordo com as mudanças que são ocasionadas no decorrer do tempo.

b) Discreto versus Contínuo - no modelo discreto as mudanças ocorrem em determinados pontos do tempo simulado. No contínuo muda de forma constante.

c) Determinístico versus Estocástico - sistemas determinísticos não possuem entradas aleatórias, assim sendo, não possuem saídas aleatórias. Sistemas estocásticos possuem entradas com valores aleatórios, conseqüentemente, possuem saídas com valores aleatórios.

Optar pelo modelo vai depender do modo de estudo do analista sobre o sistema de interesse [BAN 99].

3.3 Tempo

Há dois tipos de tempos quando falamos de simulação: o tempo de simulação e o tempo real simulado. Por exemplo, foi realizada uma simulação com tempo interno de duas horas, de um modelo que avaliava o funcionamento de um lava-rápido, essa simulação durou aproximadamente 20 segundos. Neste exemplo o tempo de simulação é de duas horas, enquanto o tempo real simulado é de 20 segundos.

O tempo em uma simulação pode ser expressa por uma variável inteira ou de ponto flutuante.

O tempo pode evoluir de acordo com o tipo de simulação que foi implantada para o modelo. Na simulação discreta evolui em intervalos aleatórios podendo ser curtos ou longos. Na simulação contínua o tempo evolui em intervalos constantes e prefixados, podendo ser regulados de acordo com a necessidade do modelo.

4 GridSimulator

O GridSimulator é uma ferramenta para o desenvolvimento e execução de modelos discretos. Ele é dividido em dois projetos principais: o editor visual de modelos e o programa que executa os modelos.

Em uma visão geral da Ferramenta, temos o editor que trata de prover um ambiente para a criação de modelos, e o processador que trata de executar o modelo criado e gerar os resultados de saída. A comunicação entre editor e processador é feita através da linguagem XML (*Extensible Markup Language*).

O editor gera um arquivo XML que é interpretado pelo processador para sua execução. Assim o usuário tem a comodidade de desenvolver seu modelo onde seja melhor para ele trabalhar, depois com o modelo pronto ele pode enviá-lo ao processador pela Internet ou por uma rede local. A seguir temos um esquema da criação e execução de modelos usando o GridSimulator.

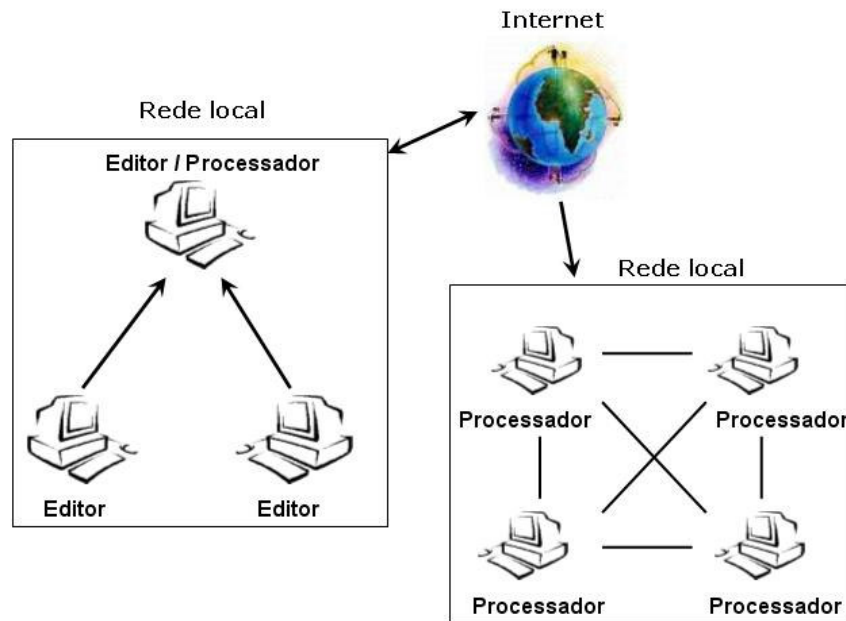


Figura 4.1: Esquema do GridSimulator

Uma ferramenta de modelagem deve possuir alguns módulos para o desenvolvimento. O diagrama a seguir mostra esses módulos de uma maneira mais ampla.

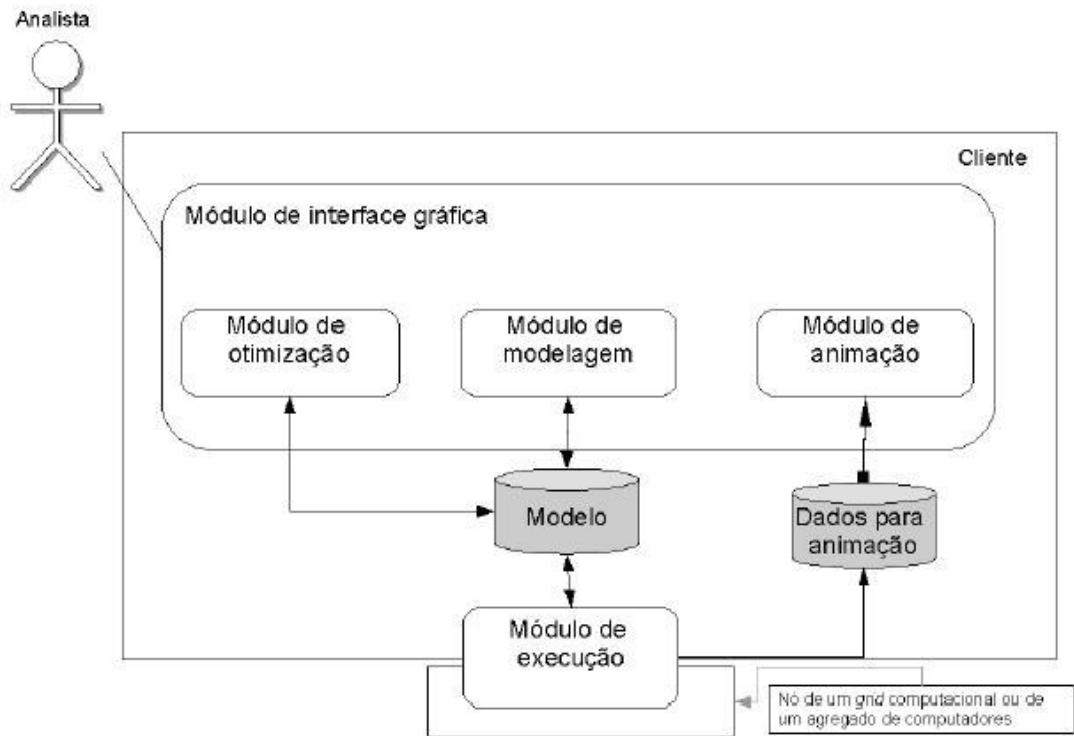


Figura 4.2: Interação geral [BER 05]

Os módulos acima têm as seguintes funções:

- Interface gráfica: meio de interação com o usuário;
- Otimização: este módulo foca a otimização do modelo de acordo com as necessidades do usuário;
- Modelagem: fornece maneiras de melhor criar modelos, utilizando o ponto de vista do processo;
- Animação: esse módulo depende de onde ocorre a simulação, caso não seja na máquina local, a animação pode ser feita depois da simulação, baseada nos resultados obtidos;
- Execução: este módulo pode estar tanto no computador local

quanto remoto, sendo responsável pelo recebimento e execução de modelos, gerando os resultados de saída.

4.1 Componentes da simulação

Alguns componentes são necessários para o desenvolvimento de um modelo de simulação. A seguir temos uma breve descrição destes componentes.

- Atribuição: este componente realiza alteração de determinados atributos das entidades por ele manipuladas. Após a chegada da entidade, ele faz os cálculos para a atribuição e faz a alteração dos valores e então encaminha a entidade alterada para seu sucessor.



Figura 4.3: Diagrama de atividades do processo de atribuição [BER 05]

- Decisão: tem como função determinar o fluxo lógico das entidades dentro do modelo. O encaminhamento da entidade recebida por ele vai depender do resultado da condição determinada a ele.



Figura 4.4: Diagrama de atividades do processo de decisão [BER 05]

- Gerador: responsável pela geração das entidades do modelo, podendo gerar quantas entidades o usuário determinar a cada evento e então as encaminha ao sucessor.



Figura 4.5: Diagrama de atividades do processo de geração [BER 05]

- Libera: este componente libera os recursos solicitados pelas entidades. Ao receber uma entidade, os recursos associados a ela são liberados e os componentes que estão aguardando por estes recursos são notificados de sua liberação.

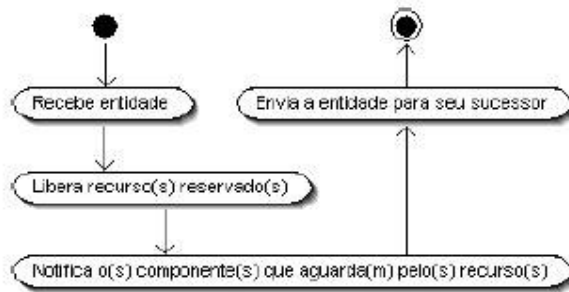


Figura 4.6: Diagrama de atividades do processo de liberação de recurso(s) [BER 05]

- Recurso: O recurso não tem uma seqüência de atividades, mas sim diferentes estados possíveis. Ele pode estar Livre, quando está disponível para ser reservado a qualquer entidade; Ocupado, quando está reservado a alguma entidade; e Falhado, quando não pode ser reservado a nenhuma entidade.

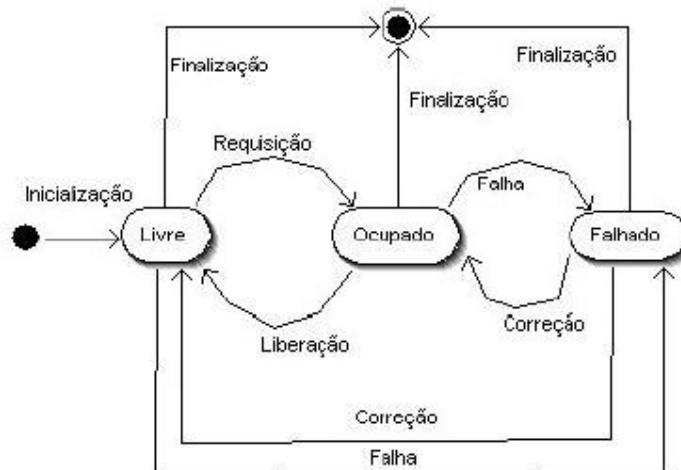


Figura 4.7: Diagrama de estado do recurso [BER 05]

- Requisita: faz a requisição dos recursos necessários à entidade. Ao

receber a entidade, verifica a disponibilidade dos recursos requeridos; caso estejam livres, são reservados e a entidade é encaminhada ao sucessor, senão a entidade é colocada na fila para aguardar a liberação dos recursos.

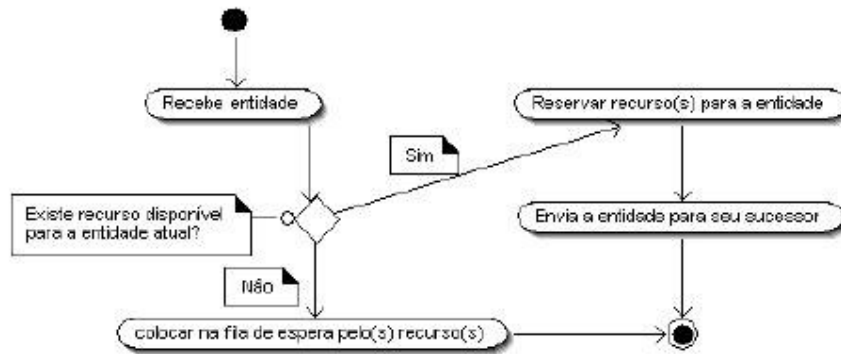


Figura 4.8: Diagrama de atividade do processo de requisição do(s) recurso(s) [BER 05]

- Retardo: este componente gera um retardo no tempo da entidade. Ao receber a entidade, ela fica aguardando o tempo definido pelo retardo para seguir ao sucessor.



Figura 4.9: Diagrama de atividades do processo de retardo [BER 05]

- Saída: é onde a entidade é retirada da execução do modelo. Os

dados de interesse são colhidos da entidade recebida e então é removida do modelo.

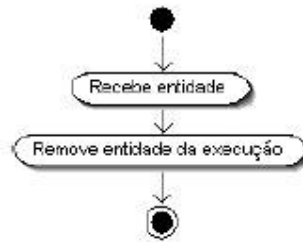


Figura 4.10: Diagrama de atividades do processo de saída [BER 05]

4.2 Editor Visual

O GridSimulator Editor tem o objetivo de proporcionar ao analista uma ferramenta de criação de modelos para simulação discreta e sua execução. O editor foi desenvolvido utilizando a plataforma Java. Para salvar um modelo em formato XML, a biblioteca JDOM para Java foi utilizada. Já a interface gráfica usa o *framework* JhotDraw.

O editor tem três áreas de interesse para o usuário: os itens de menu, a barra com os componentes e a área de modelagem.

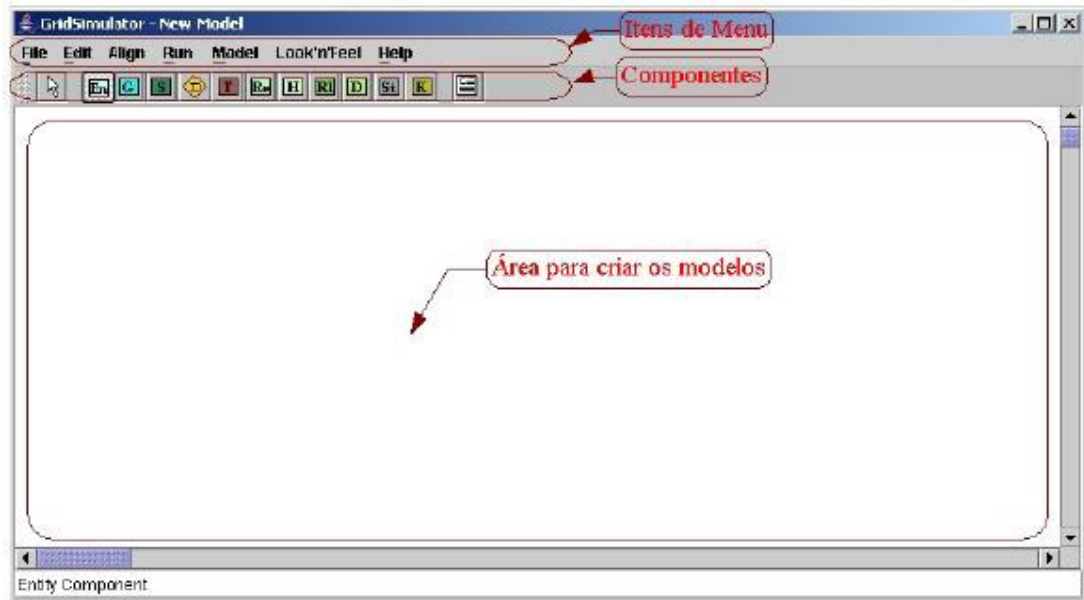


Figura 4.11: Área de uso do GridSimulator Editor [BER 05]

4.3 Processador de modelos

O processador de modelos, denominado de Ares, é responsável por interpretar o modelo criado, executá-lo, colher as estatísticas e gerar o relatório final. O processador interpreta o arquivo XML gerado pelo Editor e fornece classes para a manipulação dos dados do arquivo. Foi desenvolvido utilizando-se a linguagem C++.

O processamento é realizado da seguinte forma:

- primeiramente são gerados os eventos iniciais de cada componente;
- os eventos são ordenados pelo tempo de execução, com o menor tempo é o primeiro da fila;

- depois de formada a fila, começa-se a iteração;
- retira-se o primeiro evento e verifica-se se o tempo de evento é maior que o de parada;
- em caso positivo encerra-se a iteração e gera o relatório final.

Senão a iteração continua.

- novo teste é realizado;
- verifica-se a consistência do tempo, pois não pode ser menor que o tempo atual de execução, pois significa que é um tempo no passado e já deveria ter sido processado. Encerra-se a iteração, gera mensagem de erro e gera relatório final;

- casos os testes sejam mal sucedidos, o evento é processado e depois inicia-se a iteração.

Os eventos são executados, através da consulta de suas propriedades. As informações são armazenadas cada uma para um tipo de dado. Que pode ser: inteiro, real, booleano, string, unidade de tempo, expressão e expressão booleana.

4.3.1 Expressões

As expressões são strings que o processador interpreta e gera resultados a partir delas. O analista pode criar expressões para cálculos usando: as variáveis do modelo; os atributos das entidades; símbolos suportados; valores constantes; distribuições estatísticas; e as quatro

operações básicas (+, -, /, *). A ordem de avaliação das operações é multiplicação, divisão, soma e subtração.

O resultado da avaliação gera um valor real, mas se na expressão estiver presente o prefixo "(int)" gera um valor inteiro.

A seguir temos uma descrição das possíveis partes de uma expressão:

- Símbolos suportados: o processador suporta atualmente só o símbolo CTime (Current Time), que retorna o valor corrente do tempo;

- Variáveis: as variáveis são também expressões, que permitem uma maior flexibilização na criação dos modelos;

- Atributos: são valores reais pertinentes às entidades;

- Distribuições estatísticas: são funções geradoras de números aleatórios de acordo com determinada distribuição probabilística. As distribuições suportadas são apresentadas abaixo:

Nome	Abreviatura	Número de parâmetros	Exemplo
Uniforme	UNIF	2	UNIF(12;34)
Normal	NORM	2	NORM(10;2)
Erlang	ERLA	2	ERLA(4;15.0)
Beta	BETA	2	BETA(3;40)
Gamma	GAMM	2	GAMM(15;50)
LogNormal	LOGN	2	LOGN(15;3)
Weibull	WEIB	2	WEIB(15;3)
Exponencial	EXPO	1	EXPO(12)
Poisson	POIS	1	POIS(15)
Triangular	TRIA	3	TRIA(50;110;200)
Constante	-	1	145

Quadro 4.1: Distribuições estatísticas [BER 05]

4.3.1.1 Expressões booleanas

As expressões booleanas são comparações entre as expressões apresentadas anteriormente e podem ser dos tipos: igualdade (==), diferença (!=), maior que (>), menor que (<), maior ou igual que (>=) e menor ou igual que (<=). Este tipo de expressão é normalmente usado pelo componente de decisão.

4.3.2 Componentes de execução

Os componentes são capazes de gerar estatísticas e as estatísticas podem ser geradas padrão ou condicionais, pois sua saída no relatório final pode ser opcional.

O quadro abaixo mostra os componentes e as estatísticas geradas por cada um.

Componente	Estatísticas	Padrão	Condicional
Entity	-	-	-
Resource	Utilização	Sim	Sim
	Quantidade de solicitações (hold)	Sim	Sim
Generator	Contador de saídas	Sim	Sim
	Estatística do tempo entre chegadas	Sim	Sim
Hold	Contador de entradas e saídas	Sim	Sim
	Contador de entidades descartadas por falta de espaço na fila	Sim	Sim
	Tamanho médio da fila	Sim	Sim
	Tempo médio de espera na fila	Sim	Sim
Delay	Contador de entradas e saídas	Sim	Sim
	Estatística do tempo de espera	Sim	Sim
Release	Contador de entradas e saídas	Sim	Sim
Decision	Contador de entradas	Sim	Sim
	Contador de saídas verdadeiras	Sim	Sim
	Contador de saídas falsas	Sim	Sim
Set	-	-	-
Keep	Estatísticas das variáveis escolhidas	Não	Não
Trash	Contador de entradas	Sim	Sim

Quadro 4.2: Estatísticas coletadas pelos componentes [BER 05]

Componente	Estatísticas	Padrão	Condicional
	Tempo da entidade no sistema	Não	Sim
	Tempo da entidade em espera	Não	Sim
	Tempo da entidade em trabalho com recurso	Não	Sim
Server	Contador de entradas e saídas	Sim	Sim
	Contador de entidades descartadas por falta de espaço na fila	Sim	Sim
	Tamanho médio da fila	Sim	Sim
	Tempo médio de espera na fila	Sim	Sim
	Estatística do tempo de espera	Sim	Sim

Quadro 4.2: Estatísticas coletadas pelos componentes [BER 05]

5 Atualização do GridSimulator

5.1 Implementação de suporte multi-idioma

O termo globalização tem sido muito comumente utilizado mais num sentido de integração econômica, onde há uma reorganização geopolítica do mundo em blocos comerciais e não mais ideológicos.

Nos tempos atuais não há atividade que escape dos efeitos da globalização, em qualquer momento do dia-a-dia somos atingidos pelos seus efeitos.

A globalização tem sido caracterizada principalmente pelo predomínio de interesses econômicos. Em face disso, a busca por novos mercados ao redor do mundo torna-se cada vez mais necessário.

No caso específico do mercado de softwares essa necessidade é ainda maior devido ao amplo mercado que possui. Como nem todos os países falam o mesmo idioma ou têm os mesmos costumes, pois são diferentes na forma de escrever, de formatar um número, formatar a sua moeda etc. Torna-se quase que obrigatório que o software seja "adaptável" ao local onde ele se encontra.

Devido aos fatos que foram citados acima, uma das propostas desse Trabalho de Conclusão de Curso (TCC) era dar suporte multi-idiomas ao GridSimulator, nossa base para esse trabalho. Apesar de termo globalização estar ligado diretamente a fins lucrativos, o GridSimulator foi inteiramente desenvolvido sem nenhuma intenção de ser comercializado. O nosso objetivo ao realizar essa proposta foi poder fazer com que o GridSimulator seja uma ferramenta de uso acadêmico, onde seja facilitado a vida do usuário.

Foi então realizada, através do método de Internacionalização (I18N) e Localização (L10N) de software, a possibilidade de escolha em qual idioma gostaria de se trabalhar.

Esse trabalho foi direcionado a realizar a L10N do GridSimulator, ou seja, torná-lo adaptável para atender às exigências do idioma, culturais e outros requisitos de um mercado alvo específico ("local").

O GridSimulator foi originalmente desenvolvido em língua inglesa, pois tinha como objetivo disponibilizá-lo na comunidade de Software Livre.

Inicialmente, havia apenas a opção de se trabalhar em uma interface em inglês, hoje também é possível trabalhar com a interface em português.

5.1.1 Desenvolvimento

A proposta da I18N e L10N é desenvolver softwares que possam ser "traduzidos" para qualquer idioma que for necessário. Mas sem que haja a necessidade de recompilar a cada inserção desse novo idioma. Dessa forma a tradução via `hardcode`, fica totalmente descartada.

Como JAVA foi a linguagem na qual foi desenvolvido o editor gráfico do GridSimulator, isso facilitou o nosso trabalho. Pois Java possui classes que auxiliam nessa tarefa.

Primeiramente foi feito um estudo sobre I18N, L10N e sobre as classes `Locale` e `ResourceBundle` ambas do pacote `java.util`, para um melhor desenvolvimento.

Um objeto `Locale` representa uma região geográfica, política ou cultural específica. Uma operação que requer `Locale` para executar sua tarefa é chamada de `locale-sensitive` e usa o `Locale` para fornecer a informação sob medida para o usuário [JAV 06].

Para a criação de um objeto `Locale`, especifica-se o código do idioma e código do país.

Por exemplo, se quisermos especificar o idioma inglês da Inglaterra, invocamos o construtor da seguinte forma:

```
local = new Locale ("en", "GB");
```

O primeiro argumento do construtor é o código do idioma, um par de letras minúsculas, conforme o ISO-639. Abaixo uma tabela com alguns códigos.

Código do Idioma	Descrição
pt	Português
en	Inglês
fr	Francês
ja	Japonês
ko	Coreano
Zh	Chinês

Tabela 5.1: Código de idiomas

O segundo argumento é o código do país, um par de letras maiúsculas, conforme o ISO-3166. Abaixo uma tabela com alguns códigos.

Código do Idioma	Descrição
BR	Brasil
US	Estados Unidos
FR	França
JP	Japão
KR	República da Coreia
CN	China

Tabela 5.2: Código de países

Ainda é possível indicar um terceiro parâmetro, este indicaria qual sistema estaria sendo executado o software.

Exemplo:

```
local = new Locale ("pt", "BR", "UNIX")
```

O segundo e o terceiro argumentos são opcionais. E na falta de algum argumento, será pego locale default onde o programa está sendo executado.

É necessária a criação de arquivos para armazenar as strings traduzidas. Os nomes dos arquivos devem seguir um padrão: nome do arquivo seguido do locale a que se destina e possuir a extensão *.properties*. Os arquivos com as traduções possuem o seguinte formato:

- palavra-chave = string traduzida

Abaixo exemplos desses arquivos:

- Mensagens_pt_BR.properties e;
- Mensagens_en_US.properties.

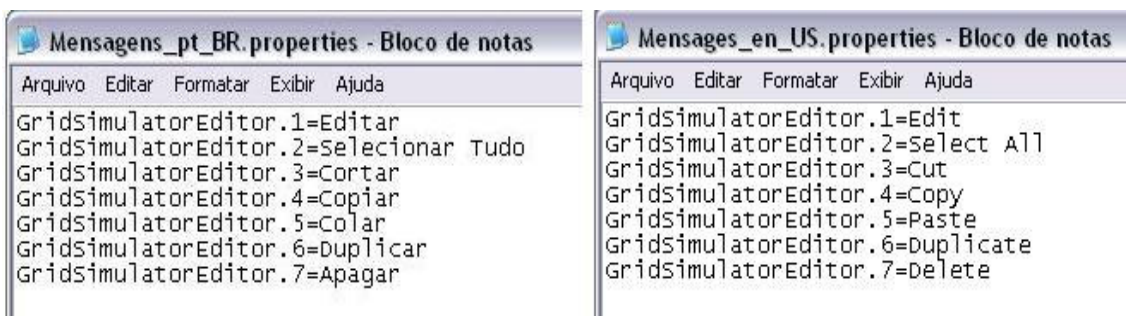


Figura 5.1: Modelo de arquivos com as strings traduzidas

A classe (abstrata) `ResourceBundle` contém os objetos específicos do locale. Este é usado para carregar dinamicamente as strings localizadas.

O `ResourceBundle` é criado como demonstrado logo abaixo:

```
rb = ResourceBundle.getBundle ("Mensagens", local);
```

As palavras-chaves são especificadas quando as mensagens traduzidas são buscadas pelo método `getString` do `ResourceBundle`.

Por exemplo para recuperar a string identificada pela palavra chave 'GridSimulatorEditor.1', invocaríamos o método da seguinte maneira:

```
String msg1 = rb.getString("GridSimulatorEditor.1");
```

Caso o locale definido fosse `pt_BR`, o valor de `msg1` seria igual a "Editar", caso fosse `en_US`, `msg1` seria igual a "Edit".

Nós desenvolvemos duas classes para efetivar o processo de internacionalização do `GridSimulator`. As classes: `I18NAction` e `SelectWindow`.

A classe `I18N` é onde utilizamos as duas classes comentadas logo acima, a `Locale` e `ResourceBundle`.

É nessa classe que utilizamos o método `getBundle` da classe `ResourceBundle` para podermos acessar nossa base de strings localizadas e o locale no qual gostaríamos de trabalhar.

```
I18NAction.resource = ResourceBundle.getBundle("Mensagens",  
local);
```

É através da classe `SelectWindow`, que é definido o `Locale`, pois ali o usuário tem a opção de escolher em qual idioma gostaria de trabalhar: Português (“pt_BR”) ou Inglês (“en_US”). Nessa classe é utilizado o método `getDefault`, da classe `Locale`, para que a tela de seleção também esteja internacionalizada.

```
Locale defaultIdioma = Locale.getDefault();
```

Através desse método, guardamos na variável `defaultIdioma` o `Locale` do sistema onde está sendo executado o `GridSimulator`, e assim também internacionalizamos nossa tela de seleção.

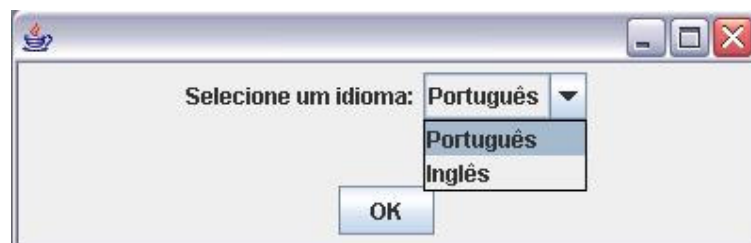


Figura 5.2: Caixa de seleção de idioma (sistema em português)

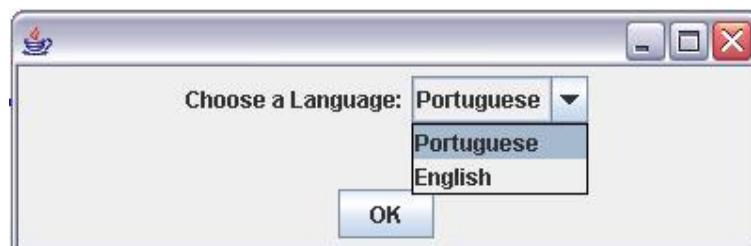


Figura 5.3: Caixa de seleção de idioma (sistema em inglês)

Uma vez definido o idioma, temos o GridSimulator adaptado para um melhor aproveitamento.

5.1.2 Exemplos

Abaixo estão alguns exemplos do resultado da localização do GridSimulator.

Primeiramente mostraremos um componente, junto com suas propriedades. Primeiramente na versão original e logo após em português.

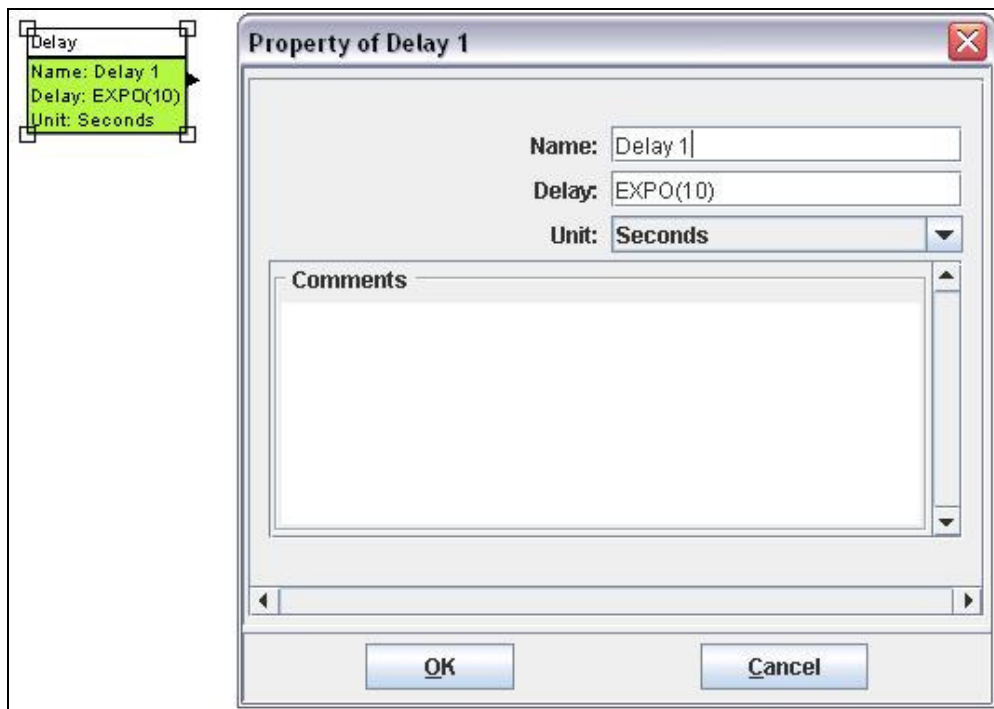


Figura 5.4: Componente na versão original

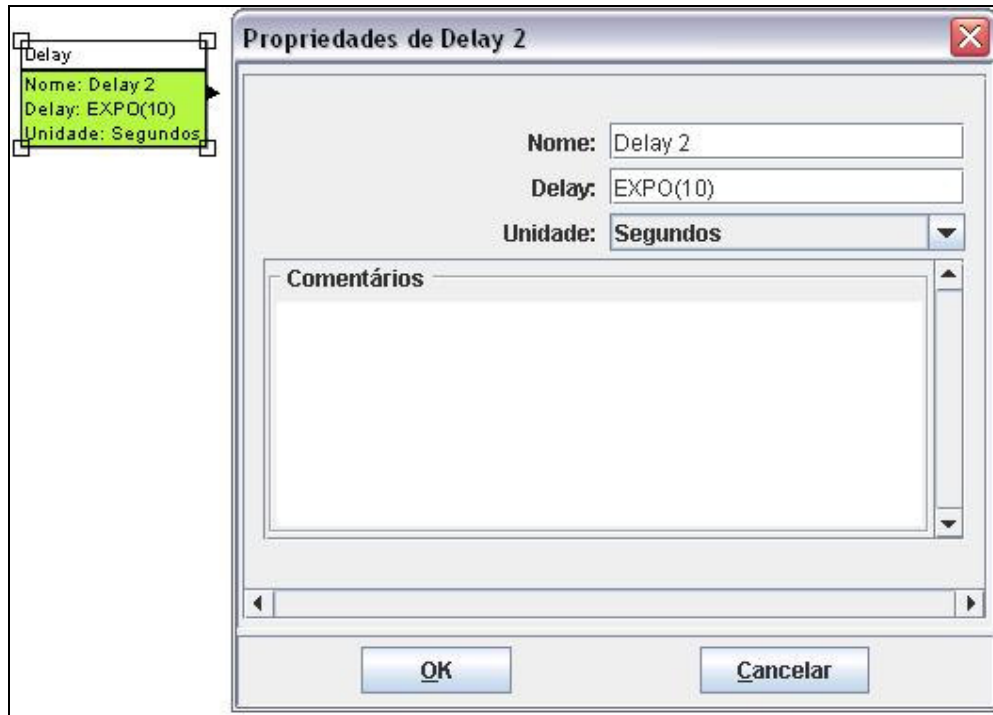


Figura 5.5: Componente com suas propriedades traduzidas

Abaixo vemos os *hints* dos componentes do GridSimulator.



Figura 5.6: *Hints* traduzido e na versão original

Menus e as caixas de mensagens, agora também apresentam-se na versão em português.

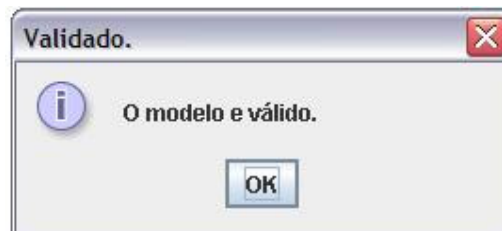


Figura 5.7: Caixa de alerta traduzida

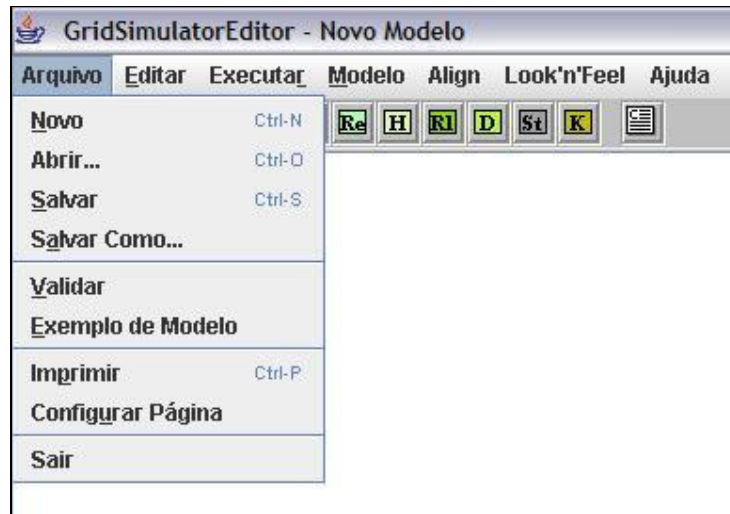


Figura 5.8: Menu e itens traduzidos

5.2 Geração de relatórios

5.2.1 Introdução

O GridSimulator originalmente tinha apenas uma forma de visualização de relatório, que era mostrado em tela, após sua execução em linha de comando. Abaixo temos uma imagem exemplificando:



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\gabriel\Desktop\pen\Pen Drive>sinlib.exe simples.gsml
REPORT 1 of 2
Simulated time: 500.0000      Seconds
Executed time: 0.0000      Seconds

TALLY Variables
Description                    Mean      Min      Max      SD      Obs
-----
Entity 10.SYSTIME                245.160  34.852  377.588  97.098  19      Seconds
Entity 10.WAITTIME               232.084  28.868  375.380  102.144 19      Seconds
Entity 10.WORKTIME               13.076   0.416   82.000   19.035  19      Seconds
Entity 2.SYSTIME                 225.517  28.868  378.169  112.229 19      Seconds
Entity 2.WAITTIME               213.158  0.000   375.023  115.218 19      Seconds
Entity 2.WORKTIME               12.359   0.024   37.798   12.727  19      Seconds
Generator 1.TimeBetweenArrival   5.984    1.330   9.950    2.506   84      Seconds
Generator 9.TimeBetweenArrival   5.686    1.088   9.835    2.730   89      Seconds
Server 3.DelayTime              13.953   0.024   82.000   17.553  39      Seconds
Server 3.Queue.WaitTime         226.475  0.000   375.380  109.084 39      Seconds
Server 7.DelayTime              0.000    inf     -inf     0.000   0       Seconds
Server 7.Queue.WaitTime         0.000    inf     -inf     0.000   0       Seconds
```

Figura 5.9: Relatório original em linha de comando

Essa forma de visualização não atende a todas as necessidades do usuário, como por exemplo, ter acesso ao relatório futuramente e facilidade de interpretação.

Sendo assim, duas novas formas de relatório foram agregadas, uma em formato de texto simples e outra em formato de página de Internet. Para este fim, no momento da execução da simulação, além de informar o programa simulador e o modelo a ser simulado, o

usuário deverá informar também um caminho com o nome de arquivo de relatório a ser criado e sua extensão, para que o simulador possa identificar qual tipo de arquivo será gerado.

No momento da execução, o simulador identificará quantos argumentos foram informados. Caso exista algum argumento além do arquivo do programa de simulação e do modelo simulado, o simulador analisará o seu conteúdo procurando pela sua extensão, que pode ser tanto “.txt” como “.html”, indicando respectivamente, um arquivo texto ou uma página de Internet.

O código usado para fazer essa diferenciação é o seguinte:

```
string nomeRelatorio = argv[2];  
int posHtml=nomeRelatorio.find(".html");  
int posTxt=nomeRelatorio.find(".txt");
```

A string nomeRelatorio recebe o conteúdo do argumento referente ao nome do arquivo de relatório, então as variáveis posHtml e posTxt armazenam um valor inteiro, que se for maior que -1, significa que o nome de arquivo informado é um tipo válido. Se após essa verificação as duas variáveis contiverem o valor -1, a simulação é executada gerando o relatório original na tela.

Se o tipo de arquivo for válido, o simulador criará um novo arquivo com o nome informado pelo usuário. Em C++ se utiliza streams para acessar (ler e escrever) arquivos no disco rígido. As operações de leitura e escrita são executadas através dos operadores de inserção e

extração (<< >>), da mesma forma utilizada com os streams de entrada e saída padrão.

Para utilizar as funcionalidades de acesso a arquivos é necessário incluir o cabeçalho <fstream>:

```
#include <fstream>
```

Para escrever em um arquivo se declara um objeto da classe **ofstream**, passando para o seu construtor o nome do arquivo que se deseja salvar:

```
ofstream arquivoRelatorio("relatorio.txt");
```

Então antes de escrever no arquivo, verificamos se ele abriu corretamente:

```
if (myfile.is_open())
```

Assim, ao invés da informação ser enviada para a tela do usuário, cada mensagem é adicionada no arquivo criado:

```
arquivoRelatorio << "\nSimulated time: " << mdl->getStopTime();
```

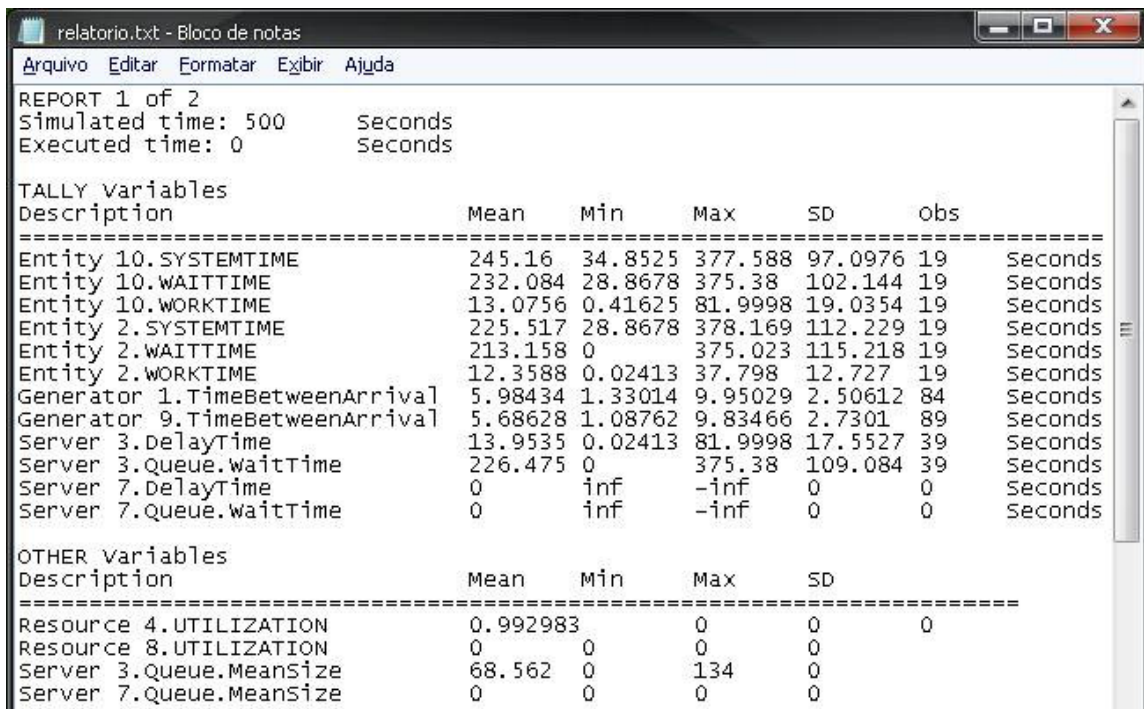
5.2.2 Relatório em formato texto

O relatório em formato de arquivo texto tem um aspecto muito semelhante ao relatório original mostrado em tela, mantendo a estrutura. Para criar esse relatório, basicamente trocamos o destino das mensagens

de simulação, que originalmente seriam usadas com o comando *cout* e as destinamos ao arquivo criado. Abaixo temos uma parte do código utilizado:

```
arquivoRelatorio << "Simulated time: " << mdl->getStopTime();  
arquivoRelatorio << "\t" << StringUtil::toString(mdl->getExecutionUnitTime()) <<endl;  
arquivoRelatorio << "Executed time: " << difftime(finish,initial) << "\tSeconds" <<endl  
if(tally.size(>0)){  
    arquivoRelatorio << "TALLY Variables" <<endl;  
    arquivoRelatorio << "Description";  
    arquivoRelatorio << "\tMean\tMin\tMax\tSD\tObs";
```

A seguir temos um exemplo de um relatório em formato texto:



```
relatorio.txt - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
REPORT 1 of 2
Simulated time: 500      Seconds
Executed time: 0       Seconds

TALLY Variables
Description              Mean      Min      Max      SD      Obs
-----
Entity 10.SYSTEMTIME     245.16   34.8525  377.588  97.0976  19   Seconds
Entity 10.WAITTIME       232.084  28.8678  375.38   102.144  19   Seconds
Entity 10.WORKTIME       13.0756  0.41625  81.9998  19.0354  19   Seconds
Entity 2.SYSTEMTIME     225.517  28.8678  378.169  112.229  19   Seconds
Entity 2.WAITTIME        213.158  0         375.023  115.218  19   Seconds
Entity 2.WORKTIME        12.3588  0.02413  37.798   12.727   19   Seconds
Generator 1.TimeBetweenArrival 5.98434  1.33014  9.95029  2.50612  84   Seconds
Generator 9.TimeBetweenArrival 5.68628  1.08762  9.83466  2.7301   89   Seconds
Server 3.DelayTime       13.9535  0.02413  81.9998  17.5527  39   Seconds
Server 3.Queue.waitTime  226.475  0         375.38   109.084  39   Seconds
Server 7.DelayTime       0         inf       -inf      0         0   Seconds
Server 7.Queue.waitTime  0         inf       -inf      0         0   Seconds

OTHER Variables
Description              Mean      Min      Max      SD
-----
Resource 4.UTILIZATION   0.992983  0         0         0         0
Resource 8.UTILIZATION   0         0         0         0
Server 3.Queue.MeanSize  68.562   0         134       0
Server 7.Queue.MeanSize  0         0         0         0
```

Figura 5.10: Relatório em modo texto

5.2.3 Relatório em formato HTML

Já o relatório em formato de página de Internet, traz ao usuário uma forma muito mais amigável para visualizar os resultados. Para tanto, é necessária uma construção mais elaborada do relatório, respeitando a sintaxe da linguagem HTML em sua estrutura e tags.

O cabeçalho de um arquivo HTML tem algumas tags fundamentais para sua descrição, como “<html>”, “<head>” e “body”. Essas tags são definidas no arquivo do relatório da seguinte maneira:

```
arquivoRelatorio << "<html>\n<head>\n<title>Relatorio</title>\n</head>\n<body>";
```

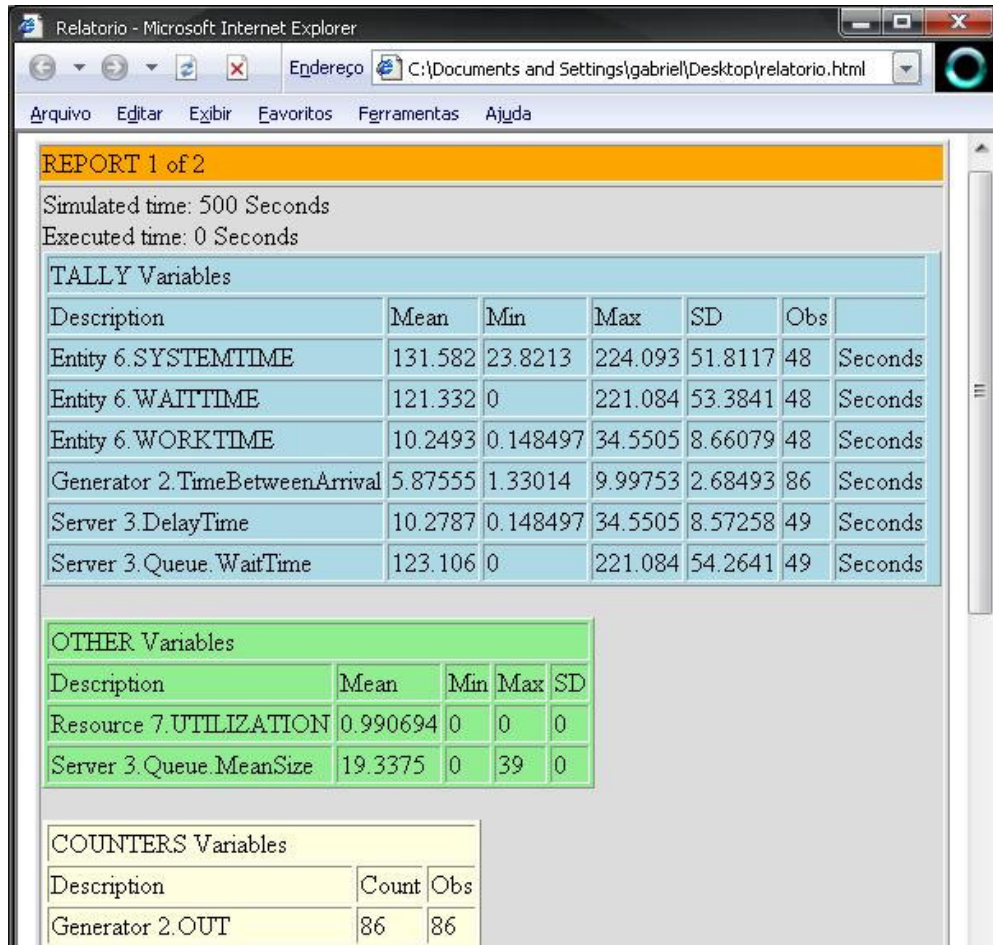



Figura 5.11: Relatório em página HTML

5.3 GridSimulator Launcher

O GridSimulator originalmente é executado em linha de comando, sendo que o usuário precisa informar no mínimo dois arquivos, o simulador e o modelo a ser simulado. O simulador é um arquivo executável, enquanto o modelo é um arquivo do formato “.gsml”, gerado pelo editor visual. A seguir um exemplo de como iniciar uma simulação.

```
C:\>arquivos>simlib.exe model.gsml
```

Caso o usuário queira gerar um relatório, um terceiro argumento deve ser passado, logo após o modelo:

```
C:\>arquivos>simlib.exe model.gsml relatório.html
```

Pensando em automatizar e agilizar esse processo, foi idealizado o GridSimulator Launcher, que é um programa que roda em janela, utilizando recursos de carregamento de arquivos e execução de rotinas para facilitar a simulação.

A interface do Launcher é simples e objetiva. Para cada argumento que seria passado por linha de comando, existe uma área para que o usuário defina o caminho do arquivo. Para isso, é fornecida uma janela comum para busca de arquivos.

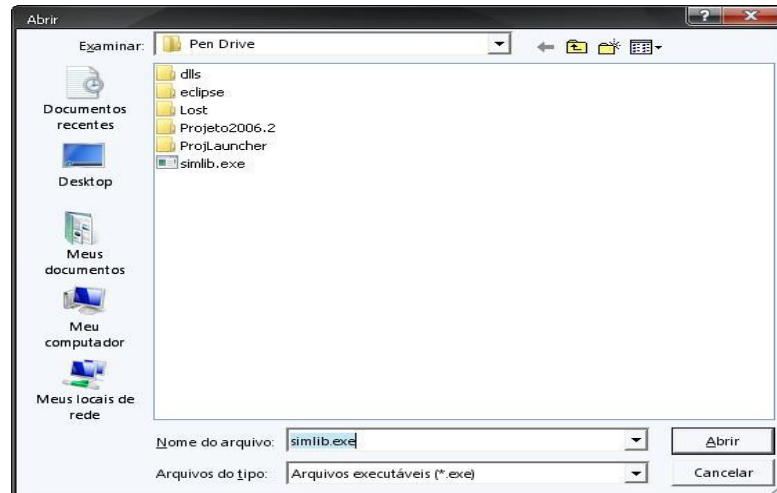


Figura 5.12: Janela para busca de arquivo

Quando o arquivo é escolhido, o seu caminho completo é mostrado em um label.



Figura 5.13: Caminho do arquivo selecionado exibido

O usuário ainda pode escolher o tipo, nome e destino do relatório. A simulação só será liberada quando as três partes do programa estiverem preenchidas.

Abaixo está a interface do Launcher:

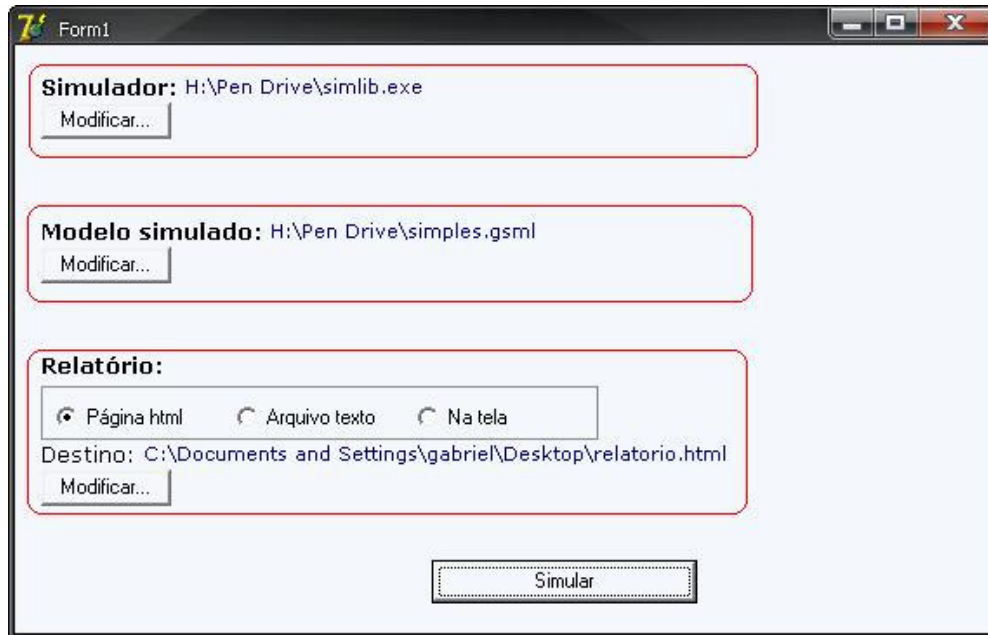


Figura 5.14: Interface inicial do GridSimulator Launcher

Ao clicar no botão “Simular”, o programa executa a seguinte linha de comando:

```
ShellExecute(Handle,'open', PChar(''+caminhoSimulador+''), PChar(''+caminhoModelo+'+'+''+caminhoRelatorio+''), nil, SW_SHOWNORMAL) ;
```

Essa linha de comando é usada para executar aplicações externas, então a variável “caminhoSimulador” contém o endereço do arquivo simulador, a variável “caminhoModelo” contém o endereço do modelo simulado e a variável “caminhoRelatorio” vai indicar onde será criado o arquivo de relatório.

6 Conclusão

6.1 Considerações finais

Simulação é uma maneira de reduzir custos e realizar um melhor planejamento nos mais variados tipos de projetos, e o GridSimulator foi desenvolvido para este fim.

Embora seja uma ferramenta completa para a criação e execução de modelos de simulação, o GridSimulator ainda podia ser aperfeiçoado, e foi em cima disso que as novas funcionalidades foram desenvolvidas.

O suporte multi-idioma, os novos tipos de relatório e o Launcher foram os aperfeiçoamentos agregados ao simulador.

Com isso, o GridSimulator tornou-se mais ágil e versátil, provendo ao seu usuário uma maior facilidade de uso, conseqüentemente aumentando sua produtividade.

6.1 Sugestões para trabalhos futuros

- a) Otimizar a execução dos componentes e diminuir a quantidade de eventos gerados;
- b) Oferecer outros formatos de relatórios;
- c) Oferecer ao usuário a possibilidade de trabalhar em outros idiomas.

Referências

- [BAN 99] BANKS, J.; CARSON, J. S.; NELSON, B. L. **Discrete-Event System Simulation**. Prentice-Hall, 1999.
- [BER05] BERKENBROCK, Gian Ricardo. **Uma ferramenta para o desenvolvimento de modelos de simulação integrada ao ambiente *grid***. Florianópolis, 2005. 100p. Dissertação (Mestrado em Ciências da Computação), Curso de Ciências da Computação, Universidade Federal de Santa Catarina.
- [dFF 01] DE FREITAS FILHO, Paulo José. **Introdução à Modelagem e Simulação de Sistemas**. Florianópolis: Visual Books, 2001.
- [JAV 06] Java Technology. **Internationalization and Localitazion**. Disponível em:
<<http://java.sun.com/docs/books/tutorial/i18n/intro/index.htm>>
Acesso em: março de 2006.
- [KEL 02] KELTON, W. D.; SADOWSKI, R. P.; SADOWSKI, D. A. **SimulationWith Arena**. McGraw-Hill, 2002.

Referências consultadas

- [ALV 01] ALVARENGA, Maria; ROSA, Maria. **Apontamentos de metodologia para a ciência e técnica de redação científica (monografias, dissertações e teses):** de acordo com a ABNT 2000. 2. ed.. Porto Alegre: Sérgio Antonio Fabris Editor, 2001. 181p.
- [LAK 91] LAKATOS, Eva Maria , MARCONI, Marina de Andrade. **Metodologia do trabalho científico.** 3. ed. rev. ampl. São Paulo: Atlas, 1991.
- [The 04] The Apache Software Foundation. **Xerces-C++ Parser.**
Disponível em:
<<http://xml.apache.org/xerces-c/>>. Acesso em: agosto de 2006.