

INTERFACE GRÁFICA, PARTICIONAMENTO DA APLICAÇÃO E DISPOSITIVO GERADOR DE ARCABOUÇOS PARA A CMB-*Simulation*

GERALDO FRANCISCO DONEGÁ ZAFALON

PROFA. DRA. RENATA SPOLON LOBATO

ORIENTADORA

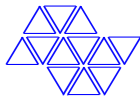
APRESENTADO POR:

GERALDO FRANCISCO DONEGÁ ZAFALON

ORIENTADO POR:

PROFA. DRA. RENATA SPOLON LOBATO

SÃO JOSÉ DO RIO PRETO
2006



INTERFACE GRÁFICA, PARTICIONAMENTO DA APLICAÇÃO E DISPOSITIVO GERADOR DE ARCABOUÇOS PARA A CMB-*Simulation*

GERALDO FRANCISCO DONEGÁ ZAFALON

PROFA. DRA. RENATA SPOLON LOBATO

ORIENTADORA

Projeto Final de Curso submetido ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

APROVADO POR:

PROFA. DRA. RENATA SPOLON LOBATO(PRESIDENTE)

PROF. DR. ALEARDO MANACERO JÚNIOR

PROF. DR. ALEDIR SILVEIRA PEREIRA

GERALDO FRANCISCO DONEGÁ ZAFALON PROFA. DRA. RENATA SPOLON LOBATO

Um problema bem entendido, é um problema meio resolvido.

Charles F. Kettering

*Ofereço à minha Noiva Letícia, à minha Mãe Maria Aparecida e ao meu
Pai Geraldo*

AGRADECIMENTOS

Agradeço primeiramente ao meu querido e amado Deus, que me deu forças em todos os momentos e me permitiu chegar até aqui. Agradeço a minha noiva Letícia Penna (Amor), por fazer parte da minha vida e me trazer amor, carinho, força, companheirismo, compreensão em todos os momentos. Ao meu pai Geraldão e à minha mãe Ninha, por terem me dado a oportunidade de chegar até aqui e terem me compreendido em todos os momentos. Aos meus amigos, André Cid (Stephão), Caio Carelo (Caion), Carlos Bonallumi (Bona), Eli Melo (Alias), Evandro Marucci (Marução), Jorge Luís (Maderão), Kelly Tafari (Cozinha), Leandro Rincon (Leandrão), Luigi Jacometti (Bérgão), Marcos Pulsoni (Marcão da Cantina), Raphael Fagliari (Oreia) e Willian Lima (Presto). Ao Prof. Dr. Aleardo Manacero Júnior (meu primeiro mestre) que me ensinou a dar os primeiros passos na Universidade, à Profa. Dra. Renata Spolon Lobato (minha mestra atual) que me ensinou a crescer tanto profissionalmente, quanto pessoalmente, ao Prof. Dr. Carlos Roberto Valêncio por todas as instruções, orientações, conselhos e pela amizade, e ao Prof. Dr. Elói Feitosa, grande amigo e guia. À Rosangela Bitonti (minha segunda mãe), mesmo estando longe nesse momento (Inglaterra), está sempre me apoiando. E, por fim a todos os meus familiares, em especial a minha madrinha Madalena Donegá, ao meu padrinho Adalberto Cunha, a minha prima Vivian Cunha, aos meus primos Adalberto Cunha Júnior e Silvio Donegá. Ao CNPq pela concessão da bolsa de iniciação científica.

SUMÁRIO

AGRADECIMENTOS	I
SUMÁRIO	III
LISTA DE FIGURAS	V
RESUMO	VII
ABSTRACT	IX
1 INTRODUÇÃO	1
1.1 Considerações Iniciais	1
1.2 Organização dos Capítulos	2
2 FUNDAMENTAÇÃO TEÓRICA	3
2.1 Considerações iniciais	3
2.2 Análise de Desempenho	3
2.3 Redes de Fila	5
2.4 Simulação Sequencial	7
2.5 Simulação Distribuída	11
2.6 Estudo da Ferramenta CMB- <i>Simulation</i>	14
2.7 Interação Humano-Computador	15
2.8 UML	17
2.9 Linguagem JAVA	18
2.10 Geradores de Aplicação	19
2.11 Técnicas de Particionamento	21
2.12 Considerações Finais	23
3 DESENVOLVIMENTO DO PROJETO	25
3.1 Considerações Iniciais	25
3.2 Diagramas de Casos de Uso	25
3.3 Implementação da interface	25
3.4 Modelos de filas	29

3.4.1	Modelos Prontos	29
3.4.2	Modelos do usuário	30
3.5	Considerações Finais	33
4	RESULTADOS	35
4.1	Considerações Iniciais	35
4.2	Interface gráfica	35
4.2.1	Modelos de filas	35
4.2.2	Arcabouço	41
4.2.3	Modelos desenvolvidos pelo usuário	41
4.3	Testes	45
4.4	Considerações Finais	49
5	CONCLUSÕES	51
5.1	Conclusão	51
5.2	Trabalhos Futuros	52
	REFERÊNCIAS BIBLIOGRÁFICAS	53

LISTA DE FIGURAS

2.1	Representação de um modelo de fila que exemplifica a LEF.	9
2.2	Esquematização de uma Lista de Eventos Futuros - Primeiro Momento.	10
2.3	Esquematização de uma Lista de Eventos Futuros - Segundo Momento.	11
2.4	Esquematização de uma Lista de Eventos Futuros - Terceiro Momento.	11
2.5	Uma situação de <i>deadlock</i> em uma simulação conservativa [1].	13
2.6	Estrutura do processo na <i>CMB-Simulation</i> [1].	14
2.7	Modelo geral de interface para interação humano-computador [2].	16
2.8	Módulos do gerador de aplicação [3].	20
2.9	Desenvolvimento em um Gerador de aplicação [3].	21
3.1	Modelagem da visão superficial do usuário para com a interface gráfica.	26
3.2	Primeira interação do usuário com a interface, em que ele se depara com as possíveis escolhas.	26
3.3	Usuário escolhe entre modelos de filas pré-definidos.	27
3.4	Usuário pode criar o seu próprio modelo de filas.	27
3.5	Um modelo de filas para analisar pontos de decisão [1].	33
3.6	Modelo a ser particionado.	34
4.1	Tela principal da interface gráfica.	36
4.2	Texto para ajuda na operação da interface gráfica.	36
4.3	Exemplo da tela de um modelo serial existente na <i>CMB-Simulation</i> antes do código.	37
4.4	Exemplo da tela de um modelo serial existente na <i>CMB-Simulation</i> depois do código.	38
4.5	Exemplo da tela de um modelo de servidor central existente na <i>CMB-Simulation</i> depois do código.	38
4.6	Exemplo da tela de um modelo de um sistema computacional, configuração tipo 1, existente na <i>CMB-Simulation</i> depois do código.	39
4.7	Exemplo da tela de um modelo de um sistema computacional, configuração tipo 2, existente na <i>CMB-Simulation</i> depois do código.	40
4.8	Exemplo da tela de um modelo hipotético, configuração tipo 1, existente na <i>CMB-Simulation</i> depois do código.	40

4.9	Exemplo da tela de um modelo hipotético, configuração tipo 2, existente na CMB- <i>Simulation</i> depois do código.	41
4.10	Tela mostrando o arcabouço generalizado.	42
4.11	Primeira tela para o usuário criar o seu modelo.	42
4.12	Listas que selecionam as taxas, probabilidades e ligações.	43
4.13	Tela para escolha do particionamento lógico.	44
4.14	Mostrando o uso de NONE.	44
4.15	Modelo criado pelo usuário.	45
4.16	Modelo criado pelo usuário.	46
4.17	Sugestão de Particionamento.	47
4.18	Código do modelo do usuário.	47
4.19	Código do modelo do usuário.	48
4.20	Uso de NONE.	49
4.21	Sugestão de particionamento.	50

RESUMO

A simulação de sistemas aparece como uma ferramenta bastante útil para realizar predição de desempenho de sistemas existentes e não existentes. Para isso, deve-se ter uma ferramenta de simulação confiável. Com o aumento da complexidade dos sistemas, a ferramenta deve conseguir gerar uma resposta em tempo hábil, com o uso de ferramentas paralelas ou distribuídas, e ainda assim, manter uma fácil operabilidade, para que pessoas com um conhecimento não muito elevado em simulação possam utilizá-la. Neste trabalho, desenvolveu-se uma interface gráfica para a ferramenta de simulação *CMB-Simulation*, uma ferramenta de simulação distribuída, permitindo que o usuário utilize os modelos de simulação prontos na ferramenta, ou crie os seus próprios modelos, sem possuir um conhecimento profundo em simulação distribuída.

ABSTRACT

The system's simulation appears as a very usefull tool to make performance analysis of existing and non-existing systems. For this, it must have itself a reliable simulation tool. With increasing of system's complexity, the tool must have generating good answer in a enough time trough the use of parallel and distributed tools, and besides that keeps an easy operability to people with a few knowledge in simulation and that they can use it. This work develops a GUI to CMB-simulation, a distributed simulation tool which allows the user to use the ready simulation models of tool, or to create his/her own models without a deep knowledge in distributed simulation.

CAPÍTULO 1

INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

O amplo desenvolvimento da computação tem-se mostrado possível a realização de tarefas antes inimagináveis. Uma das áreas que mais cresce dentro da computação é a do processamento paralelo e distribuído, que se mostra bastante útil e com grandes vantagens em relação a computação tradicional, dita seqüencial, visto o ganho de desempenho que se pode obter com o paralelismo.

A demanda por tarefas ou aplicações, cada vez mais complexas, provoca uma busca por um sistema computacional com um desempenho ainda melhor. A análise de desempenho objetiva medir o grau de eficiência de um sistema computacional, para que se saiba se esse sistema será eficiente para determinada aplicação.

Como formas de medir o desempenho de um determinado sistema, as técnicas de aferição (*benchmarks*, coleta de dados e protótipos) e as técnicas de modelagem (solução analítica do modelo ou através de simulação) são utilizadas. Devido a complexidade dos modelos propostos para os sistemas computacionais mais complexos, a análise por simulação seqüencial se torna inviável, dando lugar a simulação paralela.

A concepção de grandes sistemas, em geral, requer grandes montantes de dinheiro e necessita-se, a partir disso, possuir a garantia de que o sistema irá funcionar de forma ideal. Com isso, aparece a simulação de sistemas, como uma alternativa confiável e barata em relação a construção de um sistema real. A facilidade de se realizar uma simulação é representada pelo fato de não se necessitar do sistema pronto, nem mesmo um protótipo. Porém, muitas vezes é bastante complexo, principalmente para quem não possui conhecimentos em simulação, conceber modelos que garantam fidelidade em relação ao caso real. No entanto, se esse modelo é concebido de forma correta, a simulação torna-se uma ferramenta bastante confiável. Porém, a complexidade pode aumentar ainda mais, quando se fala do paradigma de simulação distribuída.

Na simulação seqüencial [4], quem assume um papel bastante importante é a lista de eventos futuros [5], que funciona como coordenadora da execução dos eventos. Para a simulação distribuída, essa lista de eventos necessitou sofrer mudanças, para que se pudesse garantir a consistência dos resultados em relação ao tempo. Para isso, houve o desenvolvimento de protocolos de sincronização entre os processos participantes da simulação, que são os protocolos conservativo e otimista.

Nos protocolos conservativos, os eventos são executados segundo a ordem que eles ocorrem na simulação [6], com isso o sistema não permite correr riscos. Contrariamente, os protocolos otimistas [7][8][9] permitem a execução até que um erro de causa e evento ocorra. Mas para isso os protocolos otimistas devem implementar um mecanismo que faça o sistema regredir até um estado seguro (*rollback*).

Com o fim de facilitar o trabalho do usuário, objetivou-se a construção de uma interface gráfica. Como a interface gráfica do projeto foi definida sobre a concepção de orientação a objetos, utilizando linguagem JAVA [10, 11, 12], utilizou-se dos diagramas de casos de uso da UML (*Unified Modeling Language*) para ilustrar graficamente o funcionamento dessa interface [13].

1.2 ORGANIZAÇÃO DOS CAPÍTULOS

No capítulo 2 tem-se toda a revisão bibliográfica realizada, bem como os estudos sobre o funcionamento da ferramenta CMB-*Simulation*. No capítulo 3, é apresentada toda implementação realizada e trechos relevantes de códigos. No capítulo 4, são apresentados todos os resultados obtidos e testes realizados. Finalmente, no capítulo 5, são feitas as conclusões e sugestões para trabalhos futuros.

CAPÍTULO 2

FUNDAMENTAÇÃO TEÓRICA

2.1 CONSIDERAÇÕES INICIAIS

Este capítulo engloba todo o embasamento teórico realizado para o desenvolvimento desse projeto. Nele estão desde os estudos de conceitos básicos em simulação, passando pelo entendimento de simulação distribuída, terminando na concepção da interface gráfica.

2.2 ANÁLISE DE DESEMPENHO

A análise de desempenho de sistemas computacionais possui um papel extremamente relevante, pois auxilia na concepção de um sistema que traga ganhos no tempo de obtenção de resultados de certos processamentos e ajudando a não ocorrência de subutilização de recursos.

Na implementação de muitos sistemas, apesar da importância, a análise de desempenho muitas vezes é uma passo ignorado, geralmente devido ao desconhecimento de como fazer essa análise. Isso, conseqüentemente, pode acarretar um baixo rendimento do sistema em questão.

Para realizar a análise de desempenho de um sistema computacional aplicam-se algumas técnicas. Essas técnicas mais importantes se dividem em dois grupos:

- Técnicas de Aferição:
 - *Bechmarking*;
 - Prototipação;
 - Coleta de dados.
- Técnicas de Modelagem:

- Solução de modelos através de técnicas analíticas;
- Solução de Modelos através de simulação.

A) TÉCNICAS DE AFERIÇÃO:

São técnicas de análise de desempenho obtidas a partir do estudo do próprio sistema em questão.

- **Benchmarking:** A técnica de *benchmarking* tem por objetivo estabelecer uma forma padronizada de análise para, por exemplo, efetuar a comparação entre diferentes tipos de equipamentos. Ela se resume na execução de uma determinada tarefa, com a mesma distribuição de carga para cada equipamento, com o objetivo de analisar qual deles realizar a tarefa com mais eficiência[14]. A seguir, alguns exemplos de *benchmarks*, tanto de domínio público, quanto de domínio privado. Uma descrição mais criteriosa pode ser encontrada em Weicker [15]:

- Domínio público:
 - * Whetstone - *Benchmark* sintético (análise em ponto-flutuante);
 - * Linpack - Álgebra linear (análise em ponto flutuante);
 - * Dhrystone - *Benchmark* sintético (análise em número inteiros).
- Domínio privado:
 - * GPC - Estações de trabalho;
 - * TPC - Servidores de bases de dados;
 - * EEMBC - Processadores embarcados.

- **Prototipação:** Inúmeras vezes o sistema a ser testado ainda não está totalmente disponível, com a isso a construção de um protótipo do sistema auxilia na análise do seu comportamento. Mesmo com o protótipo mostrando-se inflexível em relação a alterações na estrutura do sistema, o seu uso possui vantagens em relação a concepção direta do sistema em questão, por exemplo para medir o desempenho desse sistema;
- **Coleta de dados:** Técnica ideal para quando o sistema está fisicamente disponível e, através de algum mecanismo de *hardware* ou *software*, consegue-se medir, com a máxima precisão, o desempenho do sistema em questão. Vale a ressalva para o cuidado em relação a maneira como a coleta é feita, para que isso não interfira no resultado final;

B) TÉCNICAS DE SOLUÇÃO DE MODELOS:

Técnicas empregadas em predição de desempenho de sistemas não disponíveis, ou seja, que estão em fase de estudo para a construção e, também, em sistemas já existentes.

- **Solução de Modelos através de Métodos Analíticos:** Fornece o modelo do sistema a partir de relações

funcionais estabelecidas entre os parâmetros que são passados para o sistema [14]. Esse tipo de modelamento fornece uma representação fiel do sistema, porém esbarra no problema do aumento da complexidade do sistema, que se for ampliada pode trazer muitas complicações para a resolução analítica. Esse método é considerado de baixo custo para ser utilizado, desde que a solução analítica do sistema exista;

- **Solução de Modelos através de Simulação:** Técnicas de simulação são bastante flexíveis quando aplicadas à representação, tanto de sistemas existentes quanto de sistemas inexistentes [14]. Possui a característica de realizar prognósticos do sistema, por exemplo, experimentando como o sistema se comportaria em uma determinada situação a ele empregada. A dificuldade desse método encontra-se no momento de se fazer a verificação do programa de simulação e a validação dos dados resultantes.

2.3 REDES DE FILA

A simulação de sistemas possui um campo bastante variado, que não diz respeito apenas a sistemas de computação, mas também a outros campos que não estão ligados a área de computação, como por exemplo, simular os fenômenos naturais.

Em simulação dois tipos de eventos podem ser tratados, os discretos e os contínuos. Os eventos discretos são aqueles que realizam ações em saltos e, os contínuos, são aqueles que ocorrem em todos os instantes de tempo nos quais a ação se realiza. Uma transição de estados é realizada no decorrer da simulação e promove o início de uma nova atividade.

Simular, muitas vezes, pode não ser uma tarefa fácil. Inicialmente, torna-se importante a caracterização do tipo de sistema a ser simulado, tentando descrever com um bom grau de precisão, através de um estudo realizado previamente, como o sistema poderia se comportar. Isso serve para modelar o problema a ser tratado segundo eventos probabilísticos. Uma grande dificuldade, em alguns casos, consiste em encontrar a distribuição que se encaixa ao problema. Para isso, os métodos estatísticos existentes são de grande valia para o modelador. Nota-se que inúmeras vezes não se pode coletar dados reais para a construção do modelo.

A teoria de filas pode ser empregada, por exemplo, na parte de gerenciamento de processos candidatos em uma fila de processamento de um computador, requisições de conexões em uma rede de computadores, acesso a discos rígidos, entre outros, exemplos na área de sistemas de computação. Em outra escala, pode-se empregar esse tipo de simulação, por exemplo, em um sistema bancário para prever acúmulo de clientes e a necessidade ou não do aumento do número de atendentes disponíveis.

É importante desenvolver alguns conceitos sobre teoria de filas, como o tipo de notação utilizada para identificar uma fila, qual o seu significado e, também, medidas que analisem o desempenho de um sistema modelado pela teoria de filas.

A notação padrão de um sistema de filas é chamada de notação de *Kendall* e é utilizada para modelar um sistema com uma fila e um, ou múltiplos, servidores. Essa notação tem a forma: $A/S/c/k/m$, na qual A representa a distribuição do tempo entre as chegadas de clientes na fila, S diz respeito a distribuição do tempo de serviço, c representa o número de servidores, k representa o número máximo permitido de clientes no sistema e m corresponde ao número de clientes na fonte geradora.

Medidas de desempenho de um sistema de filas possuem fundamental importância para o modelador, visto que a partir dessas medidas conclui-se com precisão as características daquele sistema. Algumas medidas importantes:

- Taxa de Chegada na fila $\rightarrow \lambda = A/T$

Em que A é o número de chegadas na fila e T é o período de observação do sistema.

- Vazão (*Throughput*) $\rightarrow X = C/T$

Em que C é o número de atendimentos completados.

- Utilização do servidor $\rightarrow U = B/T$

Onde B é a média de tempo do servidor ocupado.

- Tempo médio de serviço por cliente $\rightarrow T_s = B/C$

LEI DE LITTLE (*Little's Law*)

Uma das leis mais importantes na teoria de filas. Pode ser aplicada independentemente do tipo das distribuições de tempo entre as chegadas e de serviço, e também, independente da disciplina da fila. Essa lei fornece média do número de clientes no sistema durante um período de observação. A lei é regida pela equação seguinte:

$$L = \lambda/W$$

Onde $\lambda = X$ é o *throughput* (taxa de chegada=*throughput*) do sistema (em um sistema de fluxo balanceado) e $W = \sum w_i / C$ representa o tempo médio que os clientes permanecem no sistema. w_i identifica o tempo gasto no sistema pelo i -ésimo cliente e C é o número de atendimentos completados.

LEI DO TEMPO DE RESPOSTA (*Response Time Law*)

Esta é também uma lei bastante importante no que diz respeito a um sistema de filas, principalmente no contexto de um sistema de tempo compartilhado. Matematicamente, traduz-se essa lei por [5]:

$$R = (N/X) - Z$$

R representa o tempo médio teórico esperado de um terminal, N diz respeito ao número de terminais no sistema, X representa o *throughput* e Z corresponde ao tempo de resposta de um terminal.

2.4 SIMULAÇÃO SEQUENCIAL

Uma simulação pode ser modelada a partir de dois tipos de modelos de eventos, os contínuos e os discretos. Os modelos de eventos contínuos são apropriados para sistemas que variam continuamente no tempo, ou seja, os eventos acontecem em todos os instantes de tempo. Os modelos de eventos discretos se encaixam em sistemas que variam em pontos discretos do tempo, admitindo assim saltos pelos instantes de tempo.

Esse projeto tratou de conceitos fundamentais de simulação por modelos de eventos discretos, levando-se em conta sistemas dinâmicos e estocásticos (possuem componentes e tempos aleatórios). Não será enfocada aqui nenhuma linguagem de simulação em particular, visto que cada linguagem possui suas características particulares.

A) CONCEITOS FUNDAMENTAIS

Os quatro conceitos que serão descritos a seguir são a base para a construção de um modelo de simulação por eventos discretos.

1. **Entidade:** Qualquer objeto ou componente do sistema que exige uma representação explícita no modelo [16].

Ex: Clientes, servidores, etc.

2. **Atributo:** Propriedades dadas a cada uma das entidades do modelo [16].

Ex: Prioridades de uso, trajetos definidos, etc.

3. **Evento:** Uma ocorrência instantânea que muda o estado do sistema [16].

Ex: Chegada de um novo cliente.

B) CONCEITOS IMPORTANTES

1. **Sistema:** Coleção de entidades que interagem em conjunto para realização de uma ou mais ações[16].

Ex: Pessoas e máquinas.

2. **Modelo:** Uma abstração de um sistema, contendo usualmente, relações estruturais, lógicas e matemáticas, as quais descrevem o sistema em termos do estado, entidades e atributos, conjuntos, processos, eventos, atividades e atrasos [16].

3. **Estado do Sistema:** Conjunto de variáveis que guardam as informações sobre o sistema e possibilitam descrevê-lo a qualquer instante de tempo [16].

4. **Lista:** Coleção de entidades associadas permanente, ou temporariamente e ordenadas segundo uma lógica[16].

Exemplos de lógicas de ordenação:

- FIFO (*First In-First Out*): Primeiro a chegar é o primeiro a ser atendido.
- Fila de prioridades (funciona segundo alguma prioridade definida anteriormente).

5. **Notificação de Evento:** Registro de um evento que ocorre no tempo corrente ou em algum ponto futuro, o qual não é necessário associar-se nenhum dado para a executar o evento. O registro inclui o tipo de evento e o tempo do evento[16].

6. **Lista de Eventos:** Lista de eventos que notifica os eventos futuros, ordenada pelo tempo de ocorrência do evento. Usualmente conhecida como Lista de Eventos Futuros (LEF) [16].

7. **Atraso:** Duração de tempo que possui tamanho inespecificado [16].

Ex: Atraso de um cliente em uma fila, depende de como ocorreram os eventos anteriores a ele.

8. **Relógio:** Variável que representa o tempo da simulação[16]. Esse tempo não se relaciona com o tempo real e é conhecido também como relógio da simulação.

Em simulação, a cada evento associa-se uma lógica. Assim, os eventos atribuem aos modelos uma característica dinâmica.

Com a posse de um estado inicial e uma lógica de processamento para cada evento, simular consiste em gerenciar a seqüenciação dos eventos. Para isso, cria-se uma lista de eventos futuros (LEF). Nessa lista, gerencia-se em que momento da simulação cada evento irá ocorrer, ou seja, ganha-se acesso ao centro de serviço. Isso ilustra um quadro típico de uma simulação seqüencial.

A lista deve ser robusta a ponto de manter o sistema simulado consistente. A LEF pode ser encarada como uma estrutura dinâmica que organiza, cronologicamente, os eventos de uma simulação. É a LEF que controla o *relógio* de uma simulação, que nada tem a ver com o relógio convencional, mas sim associa-se o seu deslocamento no tempo à ocorrência de um evento.

São comuns os erros no momento de se estruturar uma LEF, segundo a manutenção da consistência da ordem de chegada e execução de cada evento, por exemplo, executar eventos fora de uma ordem aceitável para a simulação. Na execução de uma simulação distribuída esse cuidado deverá ser redobrado e modificações serão efetuadas na LEF para que haja consistência entre os eventos, como será visto mais adiante.

Na figura 2.1, é mostrada uma representação do modelo de fila gerenciada pelo exemplo de LEF descrita nas figuras 2.2, 2.3 e 2.4.

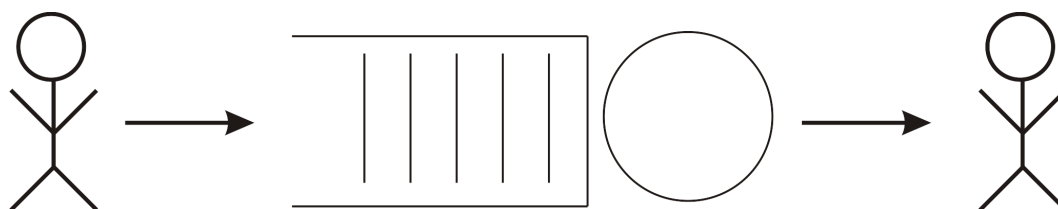


Figura 2.1: Representação de um modelo de fila que exemplifica a LEF.

No esquema da figura 2.2 verifica-se o serviço prestado por um único servidor e uma única fila, como mostrado na figura 2.1. Isso pode ser transferido e modelado para múltiplos servidores e uma fila, ou até múltiplas filas e múltiplos servidores e percebe-se como funciona a LEF, ou seja, como ela gerencia a organização de cada evento. Ainda na figura 2.2, nota-se a requisição ao servidor no tempo 3, porém o servidor encontra-se ocupado e só vai desocupar no tempo 5. Com isso, a requisição é alocada na LEF que gerenciará a sua entrada no servidor, após a sua desocupação em 5. No tempo 5, a requisição que chegou no tempo 3 toma o servidor e vai desocupá-lo no tempo 11. No tempo 7 chega uma outra requisição ao servidor, que já está ocupado, logo a requisição chegada em 7 fica na espera e é alocada na LEF, como pode ser visto na figura 2.3.

A requisição chegada em 7 assume o servidor no tempo 11 e vai deixá-lo no tempo 16.

Em termos de estrutura de dados, a LEF pode ser encarada como uma lista encadeada, em que se deve realizar operações de inserção, remoção, para que os eventos ocorram. Essa lista encadeada é atualizada constantemente a cada saída (evento em execução), ou chegada (novo evento) de um evento.

Modelo de uma Lista de Eventos Futuros

Números à esquerda: seqüência do relógio de simulação

Números entre parêntesis: eventos associados ao servidor

Eventos

0 = requisição ao servidor (chamada cliente)
1 = cliente toma servidor (ocupado)
2 = libera servidor

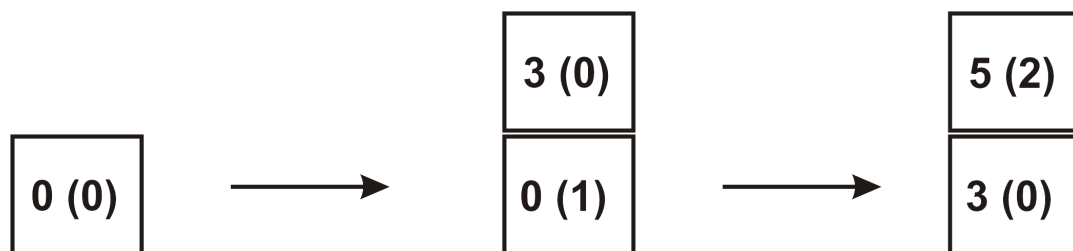


Figura 2.2: Esquematização de uma Lista de Eventos Futuros - Primeiro Momento.

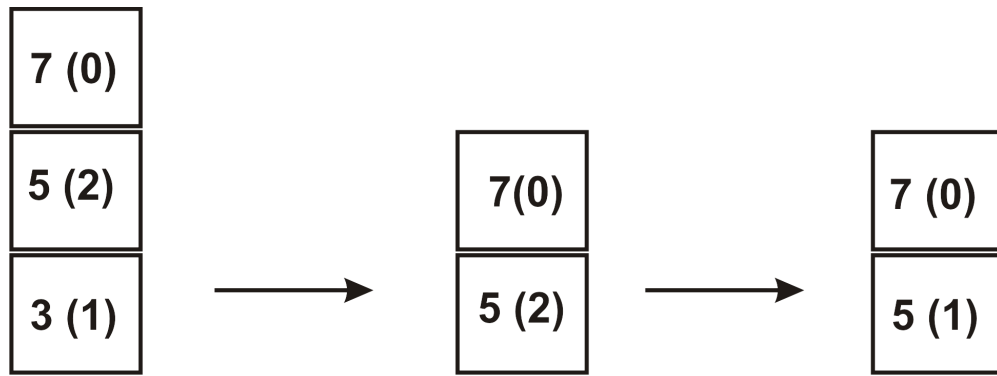


Figura 2.3: Esquematização de uma Lista de Eventos Futuros - Segundo Momento.

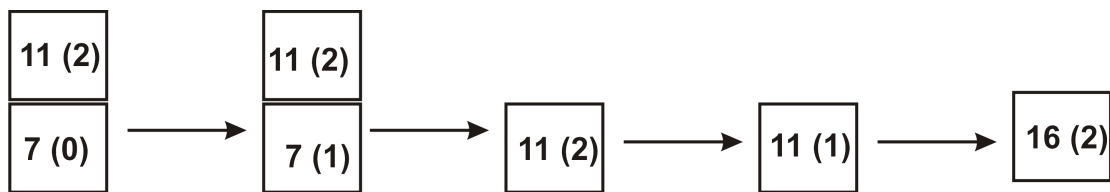


Figura 2.4: Esquematização de uma Lista de Eventos Futuros - Terceiro Momento.

2.5 SIMULAÇÃO DISTRIBUÍDA

Devido ao aumento na complexidade das tarefas executadas nas mais diversas áreas, como por exemplo engenharia, militar, ciência da computação, economia, ocorreu um aumento no tempo de processamento para se simular um determinado evento. Assim, houve a necessidade de desenvolver a simulação distribuída ou paralela, para que esse tempo fosse reduzido significativamente. Porém, paralelizar não é uma tarefa trivial, visto que os mecanismos de gerenciamento de tempo dos eventos são de características inerentemente sequenciais [17, 18] e, conseqüentemente, torna-se necessário o ajuste de particionamento de tarefas entre os processos e problemas de coerência temporal.

No que se relaciona ao particionamento entre os processos lógicos, que, nesse caso, são os processos disparados em cada unidade de processamento, deve-se levar em conta qual é o particionamento que resulta no melhor desempenho entre os centros de serviços dos modelos a serem simulados [1]. Segundo Ulson [3], algumas formas de particionamento são:

- Número de processos lógicos deve ser determinado em função das características da plataforma;
- Partição agrupa os recursos em um ordem sequencial para que não ocorra comunicação desnecessária entre os processos;
- Em ambientes de comunicação alto desempenho, a partição deve ser realizada maxi-

mizando o balanceamento na plataforma. Em ambientes de comunicação de baixo desempenho, deve-se reduzir a comunicação e a partição deve ser feita segundo a plataforma de *software* e *hardware* utilizada;

- Em modelos que se estruturam em série a partição deve adotar granulosidade grossa, reduzindo o número de processos e a comunicação entre eles;
- Modelos com estrutura de *feedback* (realimentação) alcançam melhores desempenhos, pois os resultados da realimentação são agrupados em um mesmo processo lógico, devido a redução na comunicação.

Na simulação distribuída ocorre a criação de processos lógicos que são representações de cada um dos processos do sistema real [1]. Cada processo lógico possui o seu relógio local (LVT - *Local Virtual Time*), que ficam submetidos ao relógio global da simulação (GVT - *Global Virtual Time*), para que não haja incoerência temporal.

Para que se consiga gerenciar de forma eficiente a comunicação entre os processos em um ambiente de passagem de mensagem, desenvolveram-se protocolos de comunicação para isso. Os protocolos criados para o gerenciamento foram os de controle síncrono e assíncrono.

Os protocolos de controle síncrono possuem um relógio global de controle de tempo, em que todos os processos compartilham esse relógio, processando todos os eventos que ocorrem em um determinado tempo na simulação. Protocolo ideal para ambientes de memória compartilhada.

Contrariamente, os protocolos de controle assíncronos possuem um relógio de controle local, o qual sincronizam com os processos com os quais se comunicam. Os protocolos assíncronos se dividem em conservativos [6], otimistas [9] e mistos [19]. As mensagens enviadas entre os processos que utilizam esses protocolos, além de carregarem o seu conteúdo, também levam a marca de tempo de cada um dos processos.

Os protocolos otimistas permitem a execução da simulação até que um erro de causa e efeito aconteça. A partir de então, o protocolo entra com o mecanismos de *rollback* [17] e recupera a simulação até um estado seguro. O principal representante dessa classe é o protocolo *Time Warp* [8].

Os protocolos mistos são assim chamados pois podem ser executados tanto de forma otimista quanto conservativa.

Os protocolos conservativos adotam uma política mais rígida de controle de erros de causa e efeitos, estabelecendo um intervalo de tempo (*lookahead*), em que a simulação pode transcorrer sem problemas de incoerência temporal. Pode-se citar como exemplo de protocolo conservativo o protocolo CMB [6].

O protocolo conservativo não permite que erros temporais ocorram, visto que nenhum processo pode ter o seu LVT com valor menor que o limite inferior do LVT de todos os canais de entrada. Com isso, exige-se que a sequência de marcas de tempo enviadas em um canal seja não decrescente [1].

Um dos grandes problemas enfrentados pelos protocolos conservativos é a ocorrência de *deadlocks*, devido a não recepção de uma mensagem do canal de entrada que possui o menor valor de relógio [1]. Os *deadlocks* ocorrem quando existe um ciclo de processos lógicos bloqueados e cada um está bloqueado devido a outro processo no ciclo[1]. Como medida para conter a ocorrência de *deadlocks* desenvolveram-se protocolos que adotam a passagem de mensagem nulas para os processos lógicos participantes, informando apenas o valor do menor relógio para que eles possam se atualizar. Existe também a abordagem de passagem de mensagem nula sob demanda, em que as mensagens nulas não são transmitidas após cada processamento, mas somente quando necessárias.

Como exemplo de ocorrência de *deadlock*, pode-se utilizar a figura 2.5, em que cada processo lógico está esperando, com o canal de entrada contendo o menor valor de relógio porque a fila correspondente está vazia. Todos os três estão em estado de bloqueio, pois o menor valor de relógio de canal refere-se às filas vazias (cada processo deve retirar sempre o evento do canal com menor valor de relógio).

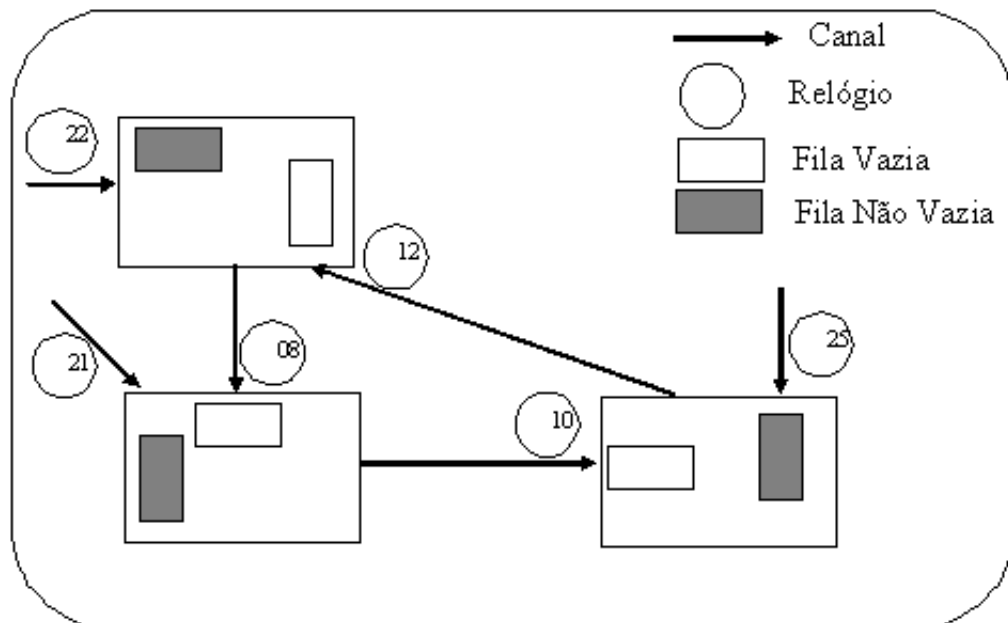


Figura 2.5: Uma situação de *deadlock* em uma simulação conservativa [1].

2.6 ESTUDO DA FERRAMENTA CMB-*Simulation*

CMB-*Simulation* é uma ferramenta de simulação distribuída conservativa, que implementa soluções com mensagens nulas, e nulas sob demanda, para o estudo de modelos de redes de filas.

Um estudo detalhado do funcionamento da ferramenta CMB-*Simulation* foi realizado, através da análise do seu código fonte e também da sua instalação e execução. Houve a realização de testes com diferentes números de processos para compreender o real funcionamento da ferramenta.

O modelo executado para compreender o funcionamento da CMB-*Simulation* foi o serial (M/M/1). Porém, vale notar que a ferramenta também disponibiliza outros modelos de filas pré-definidos, que são: sistema computacional, sistema computacional 1, sistema computacional 2, hipotético 1, hipotético 2.

Na figura 2.6 [1] é apresentada a estrutura de um processo lógico da CMB-*Simulation*. Todos os processos da simulação possuem uma estrutura semelhante a essa.

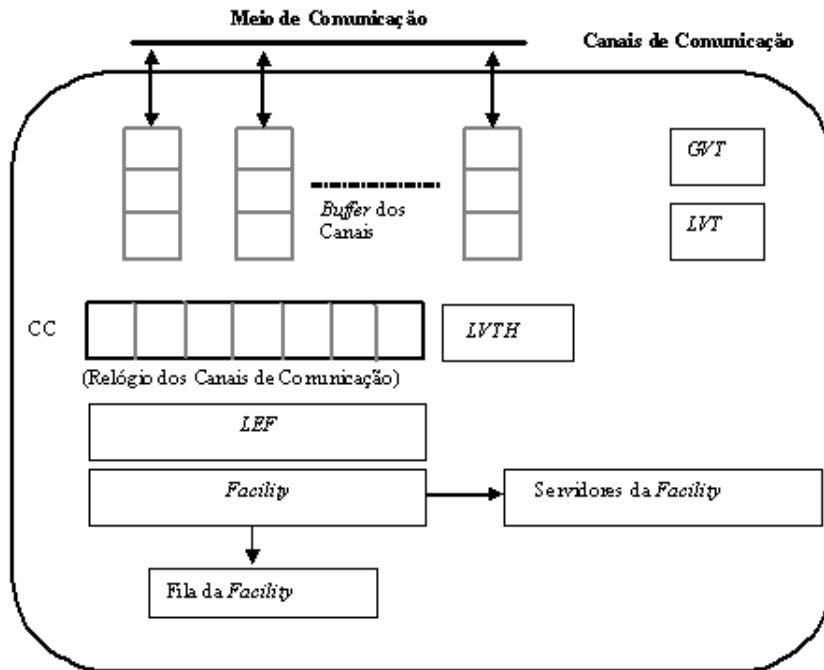


Figura 2.6: Estrutura do processo na CMB-*Simulation* [1].

Todo processo possui um número pré-definido de canais bidirecionais que servem para a comunicação entre os processos e também possuem um vetor CC de tamanho n , onde n é o número de processos que constituem a simulação. Os processos possuem o seu LVT e o LVTH, que é o valor mínimo entre os relógios dos canais.

Cada processo possui um número mínimo de recursos (*Facility*) e um número pré-definido de servidores (*Servidores da Facility*).

Inicialmente, o processo verifica se existe alguma mensagem nos canais de comunicação para serem armazenadas nos *buffers* dos canais. Em seguida, verifica se existe algum evento na LEF a ser executado, caso afirmativo o evento é retirado da fila. Caso contrário, a mensagem do canal com menor valor de relógio é retirada. Em qualquer um dos casos, o LVT é atualizado e propagado para os demais processos através de trocas de mensagens.

2.7 INTERAÇÃO HUMANO-COMPUTADOR

O relacionamento entre os seres humanos e os computadores pode ser focado em três principais ângulos: o do ser humano, o do computador, ambos tratados isoladamente, e, por fim, essas duas peças vistas como uma só. Quando se fala em interação, tem-se a idéia de ações que se realizam por dois agentes e em que ambos vão trocando informações de controle até que o término da ação seja atingido. Na interação humano-computador ocorre o mesmo. O ser humano transmite as entradas ao computador, que vai processando e repassando as saídas. Porém, vale ressaltar que existem outras formas de interações que não sejam retroativas, como por exemplo, o caso em que se colocam os dados de entrada ao computador e ele executa a tarefa sem interagir com o seu operador [2].

Um dos modelos de interação humano-computador mais utilizados é o modelo de *Norman*. Esse modelo é bastante simples, pois reúne características comuns de ações do usuário que muitas vezes não se leva em consideração. Esse modelo divide-se em 7 fases [2]:

- Estabelecimento da meta;
- Formação da intenção;
- Especificação da seqüência da ação;
- Execução da ação;
- Percepção do estado do sistema;
- Interpretação do estado do sistema;
- Avaliação do sistema final quanto as metas e as intenções.

A interação humano-computador não seria viável se não existisse um agente intermediário que amenizasse as diferenças em termos estruturais de um humano e de um computador,

como por exemplo, linguagem falada, ações realizadas, entre outras ações executadas entre eles. Esse agente intermediário recebe o nome de interface, que tenta trazer ao ser humano facilidades para operar de forma correta determinado dispositivo. A interface realiza a tradução do que se deseja, tanto passar para o sistema, quanto do que se recebe do sistema. Um interface é considerada de boa qualidade quando ela exige o mínimo do usuário para a compreensão das ações realizadas. A figura 2.7 caracteriza um modelo básico de uma interface.

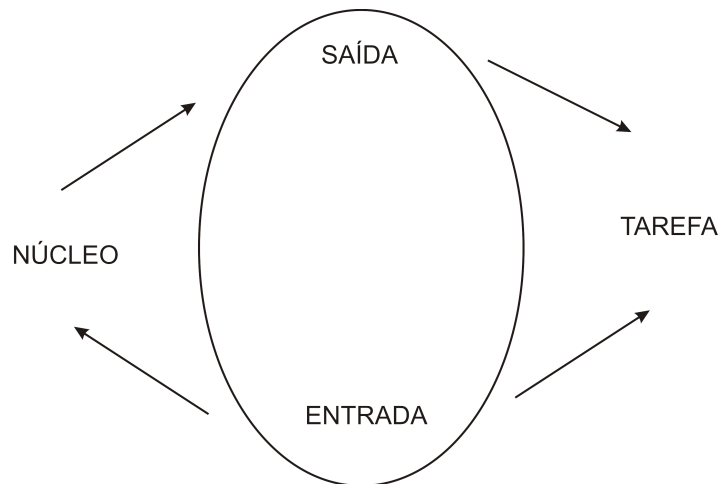


Figura 2.7: Modelo geral de interface para interação humano-computador [2].

O aspecto visual na composição da interface com o usuário possui um papel preponderante. O importante é que o usuário sinta-se confortável ao se deparar com a dada interface. Surgem as dúvidas quanto a composição da interface: escolha, por exemplo, por um botão mais arredondado ou não, colorações, disposições dos atalhos na tela, etc. Tudo isso deve ser muito bem analisado antes de se construir a interface de fato. Existem pesquisas no âmbito da construção de interfaces, e o campo da ergonomia é muito explorado, visto que ele tenta trazer conforto ao usuário. Vale lembrar que o campo da ergonomia é totalmente independente do campo de interface humano-computador, porém os dois podem caminhar juntos.

Agregado à ergonomia está a criação dos estilos de interação entre usuário e máquina. As formas ergonomicas só podem ser definidas depois da escolha da forma de interação. Existem diversas formas de interação, dentre elas, interface por linha de comando, menus, linguagem natural, formulário, entre outros [2].

A elaboração de uma interface de qualidade entre usuário e computador leva a uma operação do sistema com menor chance de erros. Existem dois tipo de erros comuns em sistemas interativos: erros de deslize e erros de fato. Os erros de deslize são geralmente

ocasionados não pela falta de informação do usuário, que de fato conhece como operar o sistema, mas pode ter ocorrido um pequeno deslize, como por exemplo apertar um botão sem desejar. Os erros de fato são erros ocasionados muitas vezes porque o usuário está se deparando pela primeira vez com o sistema, ou pelo mau desenvolvimento da interface humano-computador, induzindo o usuário ao erro. Assim, o cuidado na confecção de uma interface deve ser redobrado e um estudo de possíveis técnicas de contenção de erros deve ser realizado.

Dentro da concepção de um sistema interativo, existem sempre dúvidas se aquele sistema de fato está cumprindo com as exigências necessárias, no que diz respeito ao lado do usuário. Nesse momento, surgem questões como: Como um sistema pode ser desenvolvido para assegurar a sua usabilidade? Como demonstrar e mensurar essa usabilidade? [2]. A busca pelas respostas dessas questões pode tomar uma abordagem segundo a qual tenta-se conceber paradigmas para a confecção desses modelos interativos. Vários tipos de paradigmas de interação foram criados, dentre alguns pode-se citar: paradigma de tempo compartilhado, manipulação direta, hipertexto, computação ubíqua.

A concepção da interface gráfica para o projeto adotou o paradigma da manipulação direta, em que não se utiliza nenhuma linha de comando para executar as ações, mas sim objetos visuais como ícones, por exemplo [2]. Cita-se a seguir algumas características relevantes do paradigma de manipulação direta [2]:

- Visibilidade dos objetos de interesse;
- Ações incrementadas da interface que resultem em respostas rápidas para todas as ações;
- Reversão de todas as ações, em que o usuário é encorajado a explorar sem penalidades severas;
- Corretude sintática de todas as ações, em que toda a ação do usuário seja uma operação legal;
- Substituição de comandos complexos de linguagem por ações de manipulação direta de objetos visuais.

2.8 UML

A UML, surgida na década de 90, objetivou tornar-se um padrão de linguagem para modelagem de sistemas orientados a objetos [13]. É uma linguagem apoiada pela OMG (*Object*

Managment Group), como linguagem oficial para modelagem de sistemas orientados a objetos.

A modelagem de um sistema através da UML, resume-se a simplificar a compreensão do funcionamento desses sistemas, através de uma coleção de componentes gráficos e textuais, para que o programador, ou mesmo o usuário, quando se deparar com esse sistema, não tenha dificuldades em compreender o seu funcionamento.

Existem três elementos principais que compõe o padrão UML, que são: blocos de construção básicos, regras de como esses blocos podem ser interligados e mecanismos básicos aplicados à UML [20].

Os blocos de construção básicos são:

- Itens: blocos básicos orientados a objetos;
- Relacionamentos: blocos relacionais básicos da UML;
- Diagramas: são a representação gráfica da união entre os itens e os relacionamentos;

A concepção de sistemas orientados a objetos necessita de um estudo antecipado para que se possa criar um projeto de sistema que possua boa usabilidade, fácil manutenção e legibilidade. Para que isso ocorra, necessita-se de conhecimento prévio por parte do desenvolvedor da análise e do projeto de orientação a objetos e, além disso, conhecimento de programação orientada a objetos [13].

Um diagrama de caso de uso trata-se de um artefato criado de forma rápida e que ilustra facilmente os eventos relacionados ao sistema em discussão [13].

Torna-se importante, antes de começar um projeto lógico da funcionamento de uma aplicação, definir o seu comportamento de forma superficial, como se fosse uma "caixa-preta", descrevendo o que o sistema faz, e não como ele faz [13]. O diagrama de caso de uso facilita a compreensão do sistema no que diz respeito a interação de atores externos com o sistema [13].

2.9 LINGUAGEM JAVA

A concepção da interface gráfica para a CMB-Simulation foi toda realizada em cima da linguagem JAVA [10, 11, 12] devido a sua portabilidade e ao seu embasamento nos conceitos de orientação a objetos.

Linguagem JAVA [10, 11, 12], atualmente, é uma das linguagens mais utilizadas para a criação de diversos tipos de sistemas, entre eles a concepção de interfaces gráficas. A

boa aceitação dessa linguagem também é um fator de importância para a sua utilização no projeto.

Para o bom entendimento da linguagem e para usufruir de todos os seus benefícios e facilidades, necessita-se ter bom conhecimento dos conceitos de orientação a objetos, bem como da programação orientada a objetos.

Para o projeto, utilizou-se para a compilação a *jdk-1.5.0* e a IDE (*Integrate Development Environment*) *Net-Beans-5.0*.

2.10 GERADORES DE APLICAÇÃO

No surgimento dos computadores, a grande preocupação era com o *hardware*. Porém, com o avanço da microeletrônica essa preocupação foi deixada de lado, pois o poder computacional aumentou consideravelmente e a um custo cada vez menor.

O aumento da demanda por *softwares* de complexidade cada vez maior e falta de uma forma organizada para a produção desses *softwares*, resultaram, muitas vezes, na obtenção de *softwares* de baixa qualidade.

Os geradores de aplicação surgem para aumentar a produtividade, mantendo um bom nível do *software* produzido. Basicamente, como dito anteriormente, um gerador de aplicação é um utilitário, que a partir de uma especificação em alto nível de um problema implementável, transforma automaticamente essa especificação na implementação do problema.

Uma analogia que pode ser feita é entre um gerador e um compilador. O compilador produz código objeto em linguagem de baixo nível [3], podendo exigir, ou não, algum tipo de análise e técnicas de otimização. O gerador de aplicação cria programas em linguagem de alto nível, com a necessidade de pouca, ou nenhuma otimização de código.

Utilizando-se um gerador de aplicação, as especificações são transformadas em produtos automaticamente, traduzindo as informações de alto nível para um nível mais baixo. Para alterar o produto final, basta modificar a especificação e executar novamente o gerador. As especificações servem para descrever a tarefa ou o problema a ser feito, podendo se encontrar em uma das seguintes formas:

- um diálogo interativo, em que o usuário seleciona as opções através de um menu;
- formas gráficas;
- escritas em alguma linguagem textual.

Independente da forma empregada, as especificações são utilizadas pelo gerador de aplicação para criar um ou mais produtos, constituídos normalmente por segmentos de código, estrutura de dados ou subrotinas.

Na realidade, os geradores de aplicação criam apenas parte de um sistema. Nem sempre a automação completa do sistema é o método mais econômico, pois precisa-se avaliar a porcentagem do sistema que necessita ser gerada. Em alguns casos, é mais interessante se ter uma representação simples, porém restrita, do que uma representação abrangente, mas complexa [21].

Em geral, os geradores de aplicação são compostos por três módulos principais:

- Módulo de interface;
- Módulo analisador de especificações;
- Módulo gerador de produtos.

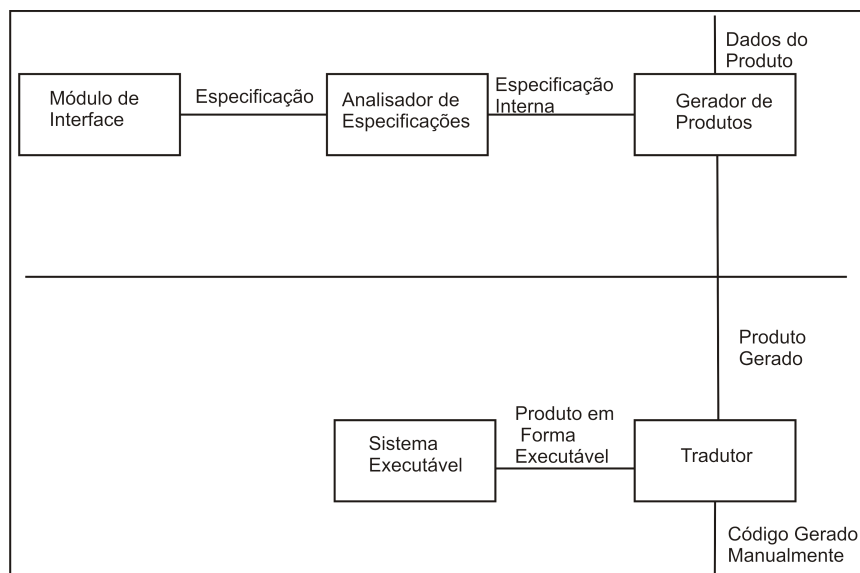


Figura 2.8: Módulos do gerador de aplicação [3].

MÓDULOS DO GERADOR DE APLICAÇÃO

Módulo de interface: estabelece o contato com o usuário, com o objetivo de obter uma especificação consistente e completa do problema. Quando esse módulo não aparece explicitamente nos geradores, a entrada dos dados é feita de forma textual, podendo ser fornecida em um arquivo separado. A idéia da interface é trazer uma característica amigável ao gerador de aplicação, porém, é de responsabilidade do usuário agrupar todas as informações necessárias à especificação do problema.

Módulo analisador de especificações: é de responsabilidade desse módulo as análises sintática e semântica dos dados de entrada, com a função de produzir as estruturas de dados intermediárias utilizadas pelo módulo Gerador de Produtos. As estruturas são dependentes do domínio e podem ser combinações de tabelas, árvores sintáticas e outras estruturas [22].

Módulo gerador de produtos: possui a função de gerar o produto desejado pelo usuário, além de uma documentação para auxiliar o usuário na compreensão e execução do programa. Possui um conjunto de rotinas padrão para acesso às estruturas de dados criadas pelo Módulo analisador, reconhecendo descrições de produtos em uma linguagem apropriada. A linguagem indica as manipulações necessárias na estrutura de dados intermediária e o formato final do produto gerado.

O desenvolvimento um sistema baseado no uso de um gerador de aplicação tem início com a especificação. Essa especificação é fornecida para o gerador de aplicação, que é responsável por criar o produto da aplicação na linguagem de programação alvo. O produto final e o código produzido manualmente, se existir, são compilados, gerando o sistema executável [23]. Na figura 2.9 são observados esses passos anteriores.

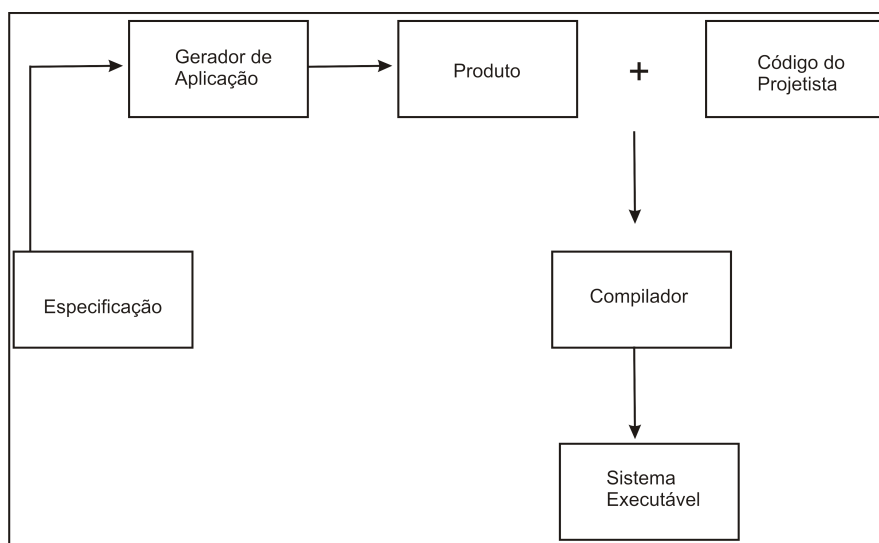


Figura 2.9: Desenvolvimento em um Gerador de aplicação [3].

2.11 TÉCNICAS DE PARTICIONAMENTO

Para se obter um bom desempenho em um ambiente de simulação distribuída, deve-se, primordialmente, tentar realizar o melhor balanceamento de carga nos processos paralelos.

Assim, deve-se tomar cuidada com alguns pontos [3]:

- Conhecer a fundo o sistema, e seu modelo, a ser simulado, explorando todo o seu paralelismo;
- Analisar a melhor forma de particionar o modelo;
- Conhecer as características de arquitetura da plataforma, para que se possa avaliar a relação entre o balanceamento de carga e a comunicação.

Os itens anteriormente citados trazem questões que devem ser levadas em conta no momento de se analisar um sistema:

- Como a estrutura do modelo influencia na partição dos processos lógicos;
- Como a partição do modelo influencia no desempenho da simulação;
- O relacionamento entre o particionamento do modelo em diferentes plataformas e o desempenho da simulação.

PARTIÇÃO DO MODELO EM PROCESSOS LÓGICOS

Analisa-se nessa seção o particionamento dos processos lógicos em um ambiente de simulação que utiliza o protocolo de sincronização CMB, que foi o protocolo utilizado para o projeto da *CMB-Simulation*.

A partir de estudos conduzidos por Smaragdakis [22] e Ulson [3], em que foram avaliadas diversas formas de particionamento para diferentes modelos de redes de filas, algumas sugestões são feitas para obter um melhor desempenho.

O que se pode concluir durante os estudos, foi que o CMB é, no tipo de plataforma estudada, um protocolo que trabalha melhor em simulações que executem em granulosidade grossa, pois a comunicação entre os processos é um fator que pode degradar o desempenho.

Outro fator que deve ser levado em conta nos modelos, são os pontos em que existem realimentações (*feedback*). É muito interessante que nos pontos que haja recursos envolvidos em *feedback* não ocorram divisão em processos lógicos distintos, pois isso acaba gerando um aumento no tráfego de mensagens.

O agrupamento dos processos em uma seqüência lógica também é um fator a se considerar, pois se os processos não necessitarem estabelecer comunicação para saber o passo seguinte, pode-se melhorar o desempenho da simulação.

Em cada ponto de decisão, que são pontos em que deve-se optar por um tipo de recurso, a utilização de probabilidades pode afetar o desempenho do sistema. A inserção dessas

probabilidades pode levar a um aumento de tarefas nos sistema e, por consequência, um aumento do número de mensagens trocadas entre os processos.

2.12 CONSIDERAÇÕES FINAIS

Depois de toda a fundamentação teórica realizada e o entendimento prático da CMB-*Simulation*, inicia-se o processo de implementação da interface gráfica, que pode ser acompanhado no capítulo 3.

CAPÍTULO 3

DESENVOLVIMENTO DO PROJETO

3.1 CONSIDERAÇÕES INICIAIS

Este capítulo apresenta todo o processo de desenvolvimento da interface gráfica para a *CMB-Simulation*, bem como os principais trechos de códigos de que realizam as funcionalidades. No capítulo 4, de resultados, são apresentadas todas as figuras relativas a essas implementações.

3.2 DIAGRAMAS DE CASOS DE USO

Como toda a interface é concebida em JAVA, devido a sua portabilidade, e como JAVA é uma linguagem orientada a objetos, decidiu-se realizar a modelagem da interface utilizando os diagramas da UML [13, 24]. Os diagramas escolhidos foram os de casos de uso.

A figura 3.1 mostra o usuário executando a abertura da interface. Dessa forma o usuário não pratica nenhuma ação além da execução da interface.

Na figura 3.2, o usuário começa a adaptar-se com a ergonomia [2] da interface. Verifica-se que, nesse momento, o usuário pode começar a escolher o que deseja fazer e a interface pode transmitir a resposta. A figura 3.2 refere-se ao momento de escolha, em que o usuário deve optar por um modelo pronto, ou por um modelo que ele mesmo pode criar.

Nas figuras 3.3 e 3.4 encontram-se os casos de uso em que o usuário realiza a ação da escolha na interface.

3.3 IMPLEMENTAÇÃO DA INTERFACE

Com a modelagem da interface terminada, consegue-se uma maior compreensão de como a interface irá operar. Isso torna muito fácil e eficiente o trabalho de implementação.

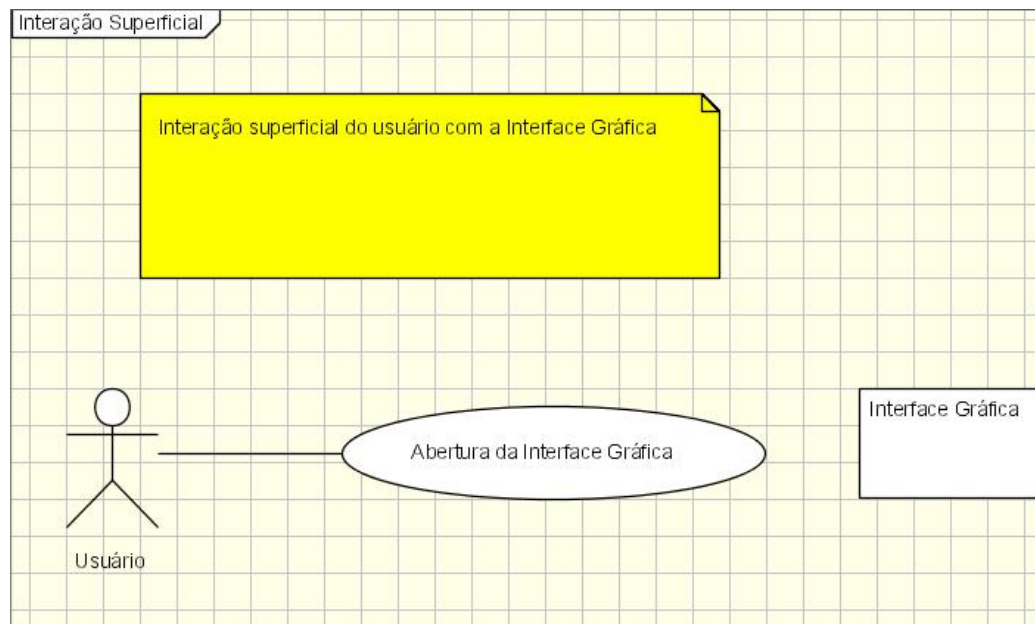


Figura 3.1: Modelagem da visão superficial do usuário para com a interface gráfica.

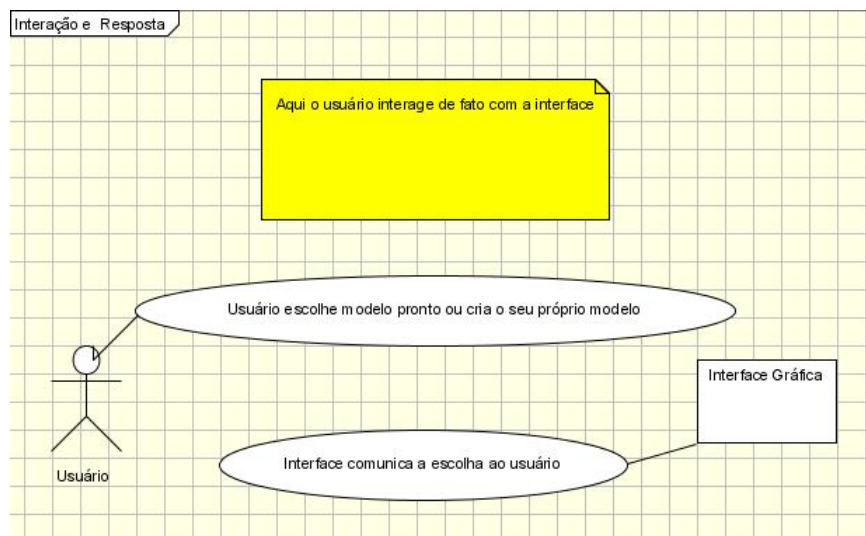


Figura 3.2: Primeira interação do usuário com a interface, em que ele se depara com as possíveis escolhas.

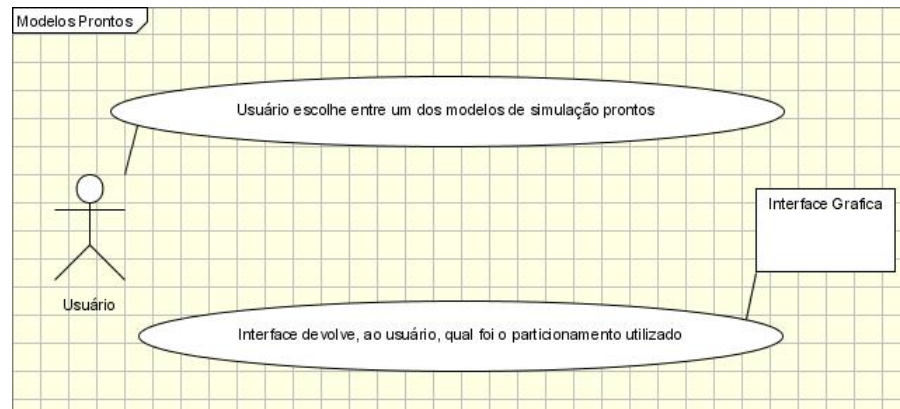


Figura 3.3: Usuário escolhe entre modelos de filas pré-definidos.

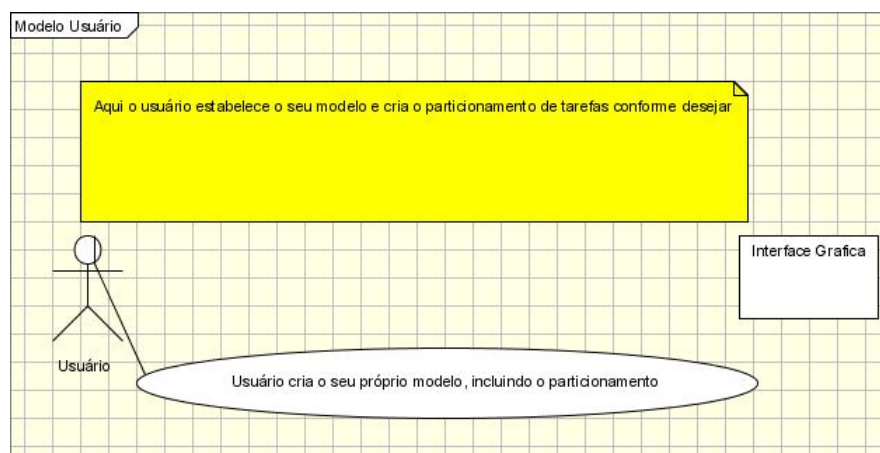


Figura 3.4: Usuário pode criar o seu próprio modelo de filas.

Ao se implementar a interface pensou-se em uma forma simples e direta, para que o usuário pudesse compreender facilmente a sua operabilidade [2].

A idéia principal é colocar os objetos componentes da interface gráfica dispostos nas bordas da interface, pois segundo [2] é algo que torna mais agradável ao usuário, devido a adaptação do campo visual do ser humano. A escolha dos botões com formas mais arredondas e textura em alto-relevo também traz uma boa aceitação ao olho humano, pois apresentando essa forma mais ergonômica, a interface se torna mais atrativa para o uso.

Dessa forma, implementou-se primeiramente a tela principal da interface gráfica, em que estão dispostos os botões para realizar as chamadas para as outras funcionalidades.

A seguir é mostrado um trecho de código da tela principal da interface gráfica, mais detalhes podem ser vistos no apêndice. Nesse são feitas as chamadas para as funcionalidades da interface:

```
private void jButton10ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
// Aciona o mecanismo de leia-me!!
    new estoumexendo.GeraArquivo(8);

}

private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//mecanismo de geração de arcahouços
    new estoumexendo.GeraArquivo(7);

}
```

3.4 MODELOS DE FILAS

A interface comporta dois tipos de ferramentas para se usar na criação de um modelo de um sistema de filas. Uma delas, localizada no lado direito da tela principal da interface, que engloba os modelos de filas prontos da *CMB-Simulation* e a outra se localiza do lado direito, e disponibiliza métodos para que o usuário crie os seus próprios modelos.

3.4.1 MODELOS PRONTOS

Utilizou-se uma forma de captura de um arquivo texto, disponibilizando esse arquivo texto em uma área e deixá-lo disponível para alteração. O conjunto principal da rotina que realiza essa ação, pode ser visto no apêndice.

Caso o usuário queira gravar as alterações processadas no código que, como exemplo pode ser visualizado em 4.4, ele pede para que as alterações sejam gravadas. Dessa forma, uma rotina varre a área em que se localiza o código, verifica o que é novo e insere no código antigo, ou seja, realiza um atualização. Um trecho da rotina que faz essa tarefa pode ser visto com mais detalhes no apêndice.

3.4.2 MODELOS DO USUÁRIO

Na interface gráfica, é permitido que o usuário crie os seus próprios modelos de simulação, com algumas restrições. Caso os modelos que fossem desenvolvidos tivessem características inerentemente sequenciais, a criação de um mecanismo que suportasse qualquer modelo não seria tão complexa de ser implementada. Porém, aqui os tratamentos são dados no âmbito da simulação distribuída, em que há a necessidade de, depois da criação do modelo, realizar o particionamento lógico da aplicação.

O particionamento lógico da aplicação consiste em indicar dentro do modelo quais conjuntos de filas e centros de serviços serão executados em quais processos lógicos da simulação distribuída. Através das primitivas do MPI, deve-se definir como será a comunicação entre os processos. Cada um desses processos lógicos pode ser executado em uma unidade física de processamento.

Através de estudos conduzidos por Ulson [3], verifica-se que determinadas formas de particionamento pioram o desempenho de uma simulação, devido a vários fatores, porém o mais crítico deles é a interdependência entre processos nos modelos em que existe realimentação de uma determinada fila por um determinado centro de serviço.

MECANISMO GERADOR DE ARCABOUÇOS

Depois dos estudos conduzidos com a *CMB-Simulation*, entendeu-se que todos os programas de simulação criados na ferramenta possuíam uma forma padrão, com estruturas que estão presentes em todos os programas. O que difere cada um dos programas de simulação são as estruturas de desvio condicional, marcadas pelo uso das estruturas *case*, que se colocam no meio da rotina, que tentam representar o modelo pré-definido.

Na tela principal da interface gráfica, foi implementado um botão que deixa a possibilidade para o usuário ter acesso ao arcabouço mais generalizado de um programa de simulação para a *CMB-Simulation*. No apêndice mostra-se como é feita a leitura da parte essencial para a codificação de um programa de simulação.

O usuário pode realizar as alterações que quiser no arcabouço de código, inserindo as estruturas condicionais, se assim desejar e construir o seu próprio modelo de simulação textualmente.

Na tela principal da interface gráfica, para facilitar, o usuário tem a opção para acionar uma ferramenta para a criação do modelo do usuário e, na sequência, foi implementada uma forma de capturar as entradas do usuário.

Criou-se uma rotina que executa no momento em que o usuário pressiona um dos botões contendo o conjunto fila/centro de serviço, e é mostrado a ele, uma tela com três opções.

Essas opções foram implementadas a partir do conceito de *Jlist* em Java, nas quais pode ser feita apenas uma escolha, em cada uma delas. Um trecho de código das *JLists* pode ser visto no apêndice.

A captura das entradas nessa *JList* foi feita a partir de *Listeners*, que são métodos que ficam executando o tempo todo a espera de uma ação, como pode ser visto na seqüência.

```
colorList.addListSelectionListener(

// Classe interna para seleção de eventos
    new ListSelectionListener() {

        public void valueChanged( ListSelectionEvent event ) {

            container.setBackground( String[] strings.getSelectedIndex() );

        }

    }
); //fim da chamada para addListSelectionListener
```

Quando o usuário cria o seu próprio modelo, o resultado dessa criação também é uma arcabouço de código, porém um pouco mais refinado que o anteriormente citado. Esse arcabouço produzido necessita de mais algumas alterações por parte de usuário, para que possa funcionar na *CMB-Simulation*. Como o próprio nome relata, é apenas um arcabouço, ou seja, não é o programa completo de simulação. Ele cria as principais estruturas para o uso.

O arcabouço obtido com o gerador de aplicação, fornece uma estrutura do tipo:

```
////Aqui entrar as chamadas de bibliotecas
.
.

//Começo da execução da simulação
While()
{
```

Case 1:

Case 2:

Case 3:

```
.
.
.
.
.
} //fim da execução
```

//reportagem do que foi obtido

A quantidade de estruturas *case* depende da quantidade de pontos de decisão existentes no modelo do usuário. Por exemplo, como pode ser visto na figura 3.5, um sistema com um servidor central (UCP), em que o usuário (tarefa) deve decidir em um primeiro ponto, qual disco (disco 1, disco 2, disco 3 ou disco 4) vai acessar. Isso caracteriza um ponto de decisão.

Depois de implementado o mecanismo gerador de arcabouços, desenvolveu-se a técnica para particionar a aplicação, a partir de estudos conduzidos por [22, 3].

O particionamento da aplicação permite que o usuário agrupe nos mesmos processos lógicos as estruturas filas/centros de serviço que desejar. Mesmo que o usuário não faça a melhor escolha de particionamento, o sistema permite executá-lo.

Na figura 3.6 tem-se um modelo de um sistema de filas. Esse modelo serve para ilustrar como o mecanismo trata no momento de particionar tal modelo.

Como pode ser verificado, no conjunto fila/centro de serviço número 3, existe uma realimentação nesse conjunto. Pelos estudos mostrados na seção 2.10 sobre técnicas de particionamento, verifica-se que pontos em que existam realimentações devem ser mantidos isolados, em um único processo lógico, para que o desempenho da simulação melhore.

Imaginando-se que o modelo da figura 3.6 seja inserido na interface, particionado, e considerando-se que o usuário tenha escolhido particionar colocando os conjuntos 1 e 3 no mesmo processo lógico, o mecanismo de particionamento vai sugerir que aquele não é o particionamento indicado, e sim, por exemplo, colocar os dois processos que não possuem realimentação no mesmo processo lógico, nesse caso, 1 e 2, e permitir que 3 fique isolado em

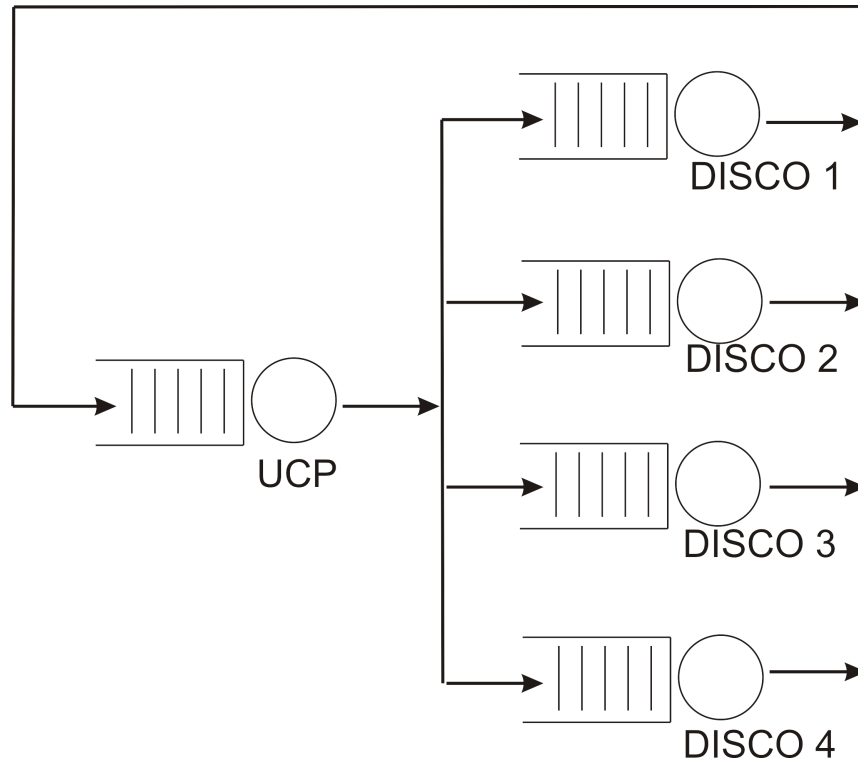


Figura 3.5: Um modelo de filas para analisar pontos de decisão [1].

um processo lógico. Porém, vale ressaltar que caso o usuário deseje manter o seu próprio particionamento, é possível. Essa liberdade foi permitida pois o usuário pode querer estudar o comportamento da simulação distribuída em condições desfavoráveis de particionamento.

Com o arcabouço pronto, a diretiva que é inserida para indicar qual canal o processo lógico vai ocupar é o *CMB_CHANNEL_PROCESS(Num_processo, Num_Canal)*. São passados como parâmetros o número do processo e o número do canal de comunicação.

3.5 CONSIDERAÇÕES FINAIS

Neste capítulo foi visto como foram implementadas todas as partes da interface gráfica, suas ferramentas e seus mecanismos. Os trechos de código ajudaram a elucidar como foi realizada essa implementação. No capítulo 4 serão colocados de uma forma mais explícita, através de telas explicativas, todos os resultados obtidos com a implementação descrita no capítulo 3 e, ainda, alguns testes propostos para o projeto.

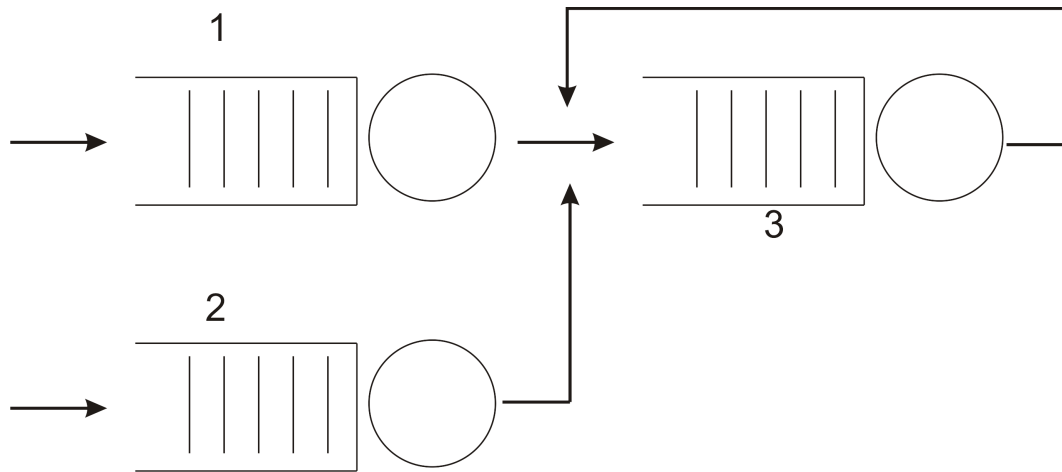


Figura 3.6: Modelo a ser particionado.

CAPÍTULO 4

RESULTADOS

4.1 CONSIDERAÇÕES INICIAIS

Esse capítulo engloba tudo o que foi obtido com as construções das estruturas mostradas no capítulo 3. Nesse capítulo, são apresentadas as telas e alguns testes obtidos com o sistema em questão.

4.2 INTERFACE GRÁFICA

A figura 4.1 mostra a tela principal da interface gráfica para a CMB-*Simulation*. A interface gráfica também fez parte de um trabalho que foi publicado em congresso, como visto em [25], em que foi abordado o funcionamento da CMB-*Simulation* [1] como ferramenta de análise de desempenho e a sua interface gráfica.

Visualizando-se a figura 4.1, estão à direita os botões relativos à utilização de modelos prontos existentes na CMB-*Simulation*. As funcionalidades desses botões serão explicadas na seção 4.2.1.

Na parte central, abaixo do símbolo do Grupo de Sistemas Paralelos e Distribuídos, encontra-se uma opção de *leia-me*. Nessa opção, os usuários que não estiverem habituados com as funcionalidades da interface, e/ou tenham alguma dúvida sobre a ferramenta, podem acionar essa função de ajuda para tentar solucionar a dúvida. A figura 4.2 mostra uma parte do texto para ajuda.

4.2.1 MODELOS DE FILAS

Como citado anteriormente, existem modelos de filas prontos dentro da CMB-*Simulation* que podem ser utilizados pelo usuário, caso algum desses seja interessante a ele. Dependendo do tipo de sistema que o usuário deseja simular para avaliação de desempenho, esses modelos são bastante úteis.



Figura 4.1: Tela principal da interface gráfica.

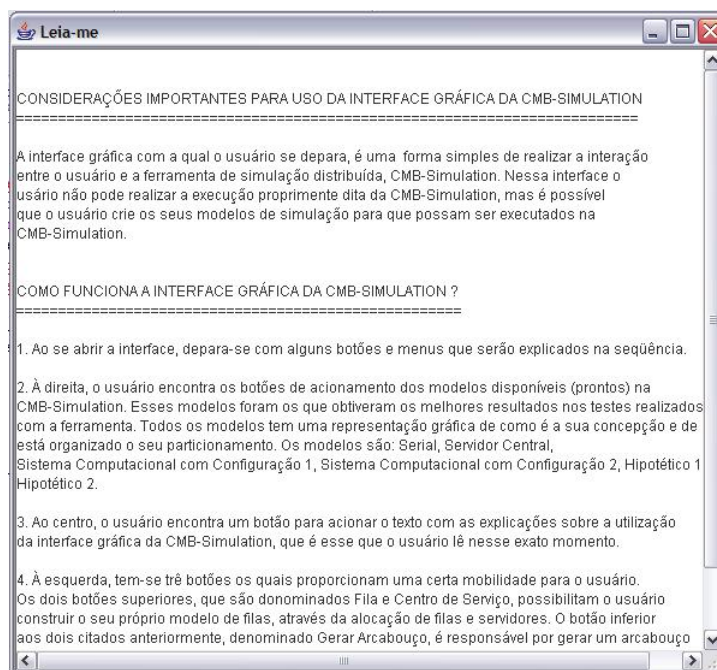


Figura 4.2: Texto para ajuda na operação da interface gráfica.

Os botões que identificam cada um dos modelos existentes na *CMB-Simulation* que são modelo Serial, Sistema Computacional Configuração 1, Sistema Computacional Configuração 2, Sistema Servidor Central, Sistema Hipotético 1 e Sistema Hipotético 2, ao serem pressionados carregam um novo quadro com uma tela em que se pode visualizar o modelo pronto, como pode ser visto na figura 4.4.

A visualização do modelo pronto acontece da seguinte forma: acompanhando pela figura 4.3, nota-se a área onde aparecerá o código e, ao lado, o modelo que aquele código vai representar. No exemplo que está mostrado, trata-se de um modelo serial com o particionamento que apresentou melhores resultados na execução da *CMB-Simulation* no trabalho desenvolvido por [1].

Para visualizar o código do modelo, basta o usuário posicionar o cursor no campo onde está o nome do modelo, que nesse caso, está colocado `serial.c`, e pressionar a tecla *enter*. Assim, aparecerá o código, como pode ser visto na figura 4.4.

Existe também a possibilidade de alteração desse código. Qualquer parâmetro ou linha, pode ser alterado. Depois, se pressionar o botão *gravar as alterações*, tudo que foi mudado naquele código, será gravado em um arquivo chamado `mudanca.c`, que será criado na própria pasta local, onde a *CMBSimulation* executa. Esse modelo pode ser executado normalmente na *CMB-Simulation*.

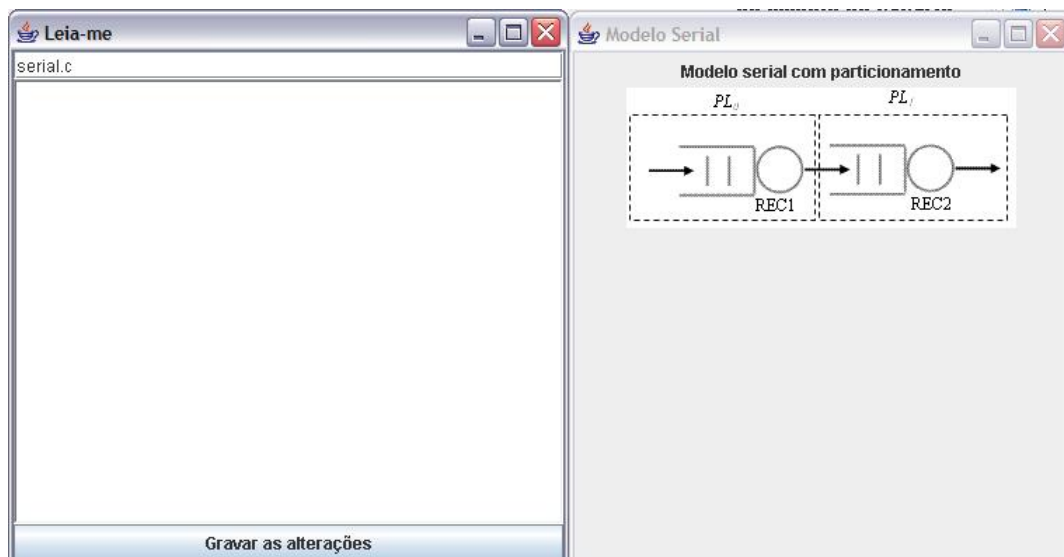


Figura 4.3: Exemplo da tela de um modelo serial existente na *CMB-Simulation* antes do código.

A figura 4.5 mostra o modelo e o código relativo ao modelo para o esquema de um servidor central.

Nas figuras 4.6 e 4.7, tem-se os modelos de um sistema computacional com diferentes

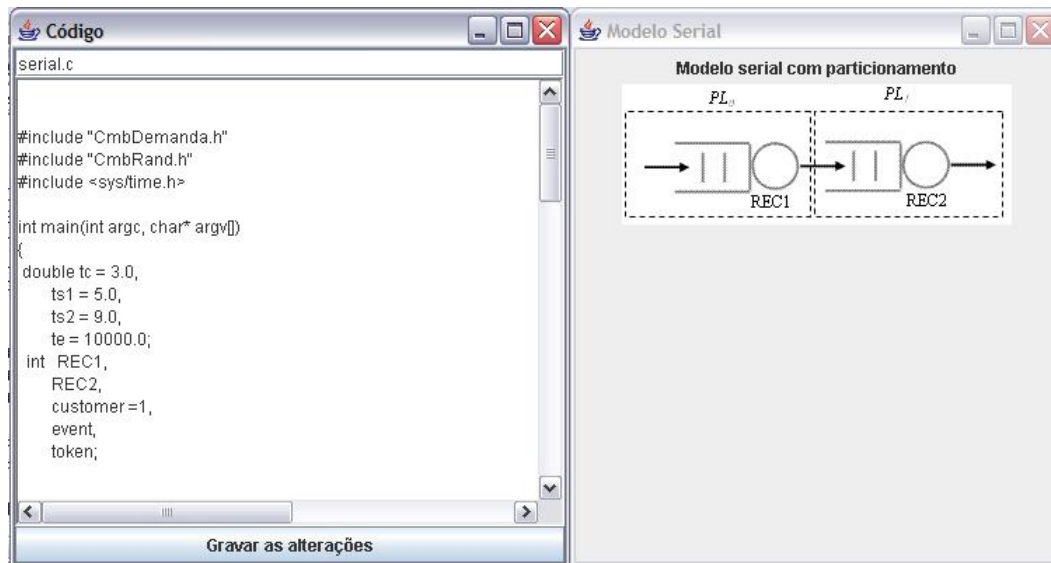


Figura 4.4: Exemplo da tela de um modelo serial existente na *CMB-Simulation* depois do código.

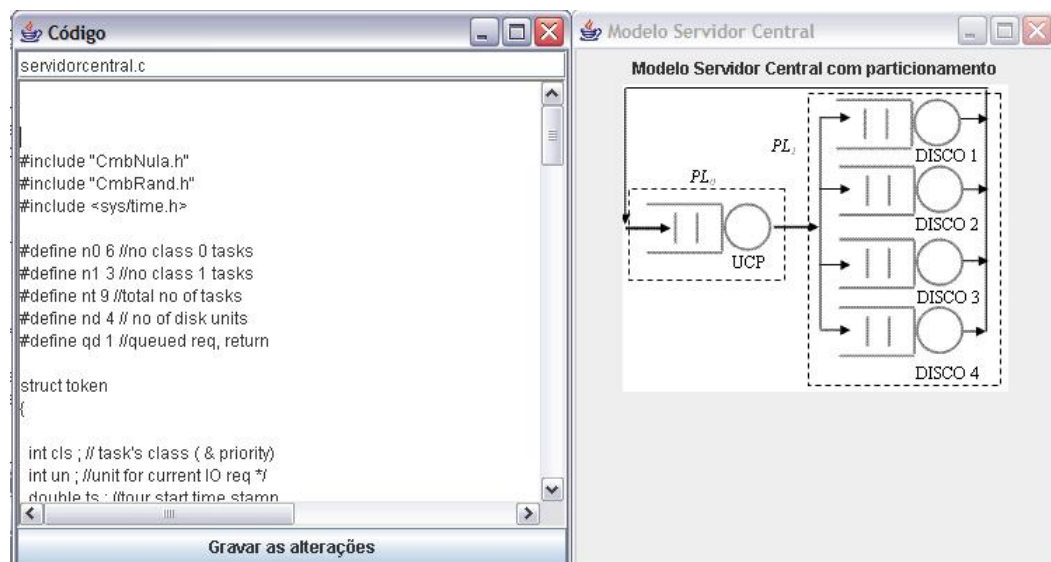


Figura 4.5: Exemplo da tela de um modelo de servidor central existente na *CMB-Simulation* depois do código.

tipos de particionamento.

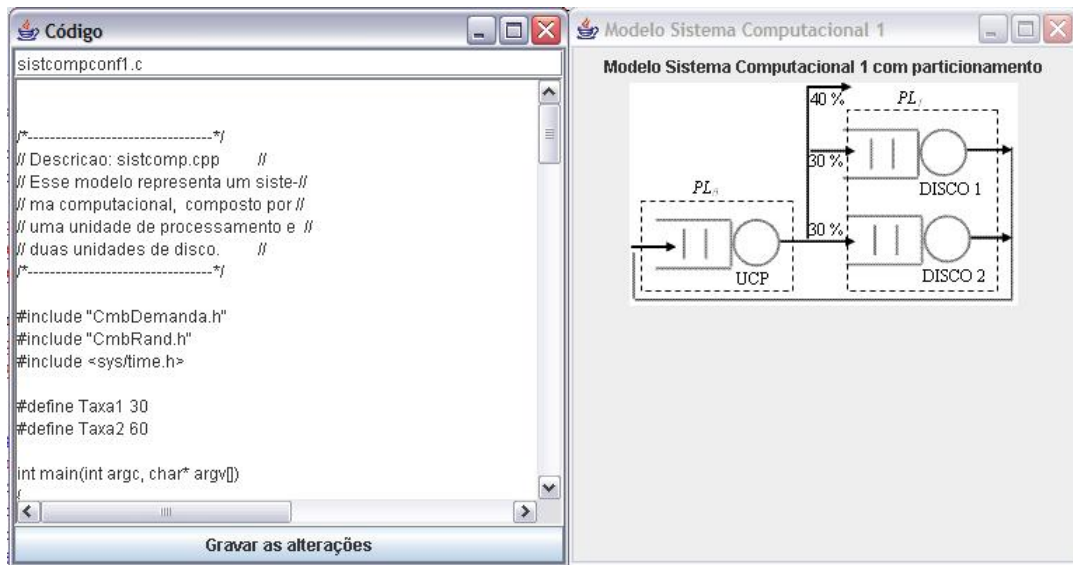


Figura 4.6: Exemplo da tela de um modelo de um sistema computacional, configuração tipo 1, existente na *CMB-Simulation* depois do código.

As figuras 4.8 e 4.9 mostram os modelos de dois sistemas hipotéticos com diferentes tipos de particionamento.

Os tipos de configurações citados nas figuras 4.6, 4.7, 4.8 e 4.9 relacionam-se aos tipos de particionamento que resultaram no melhor desempenho durante as simulações testadas por Balieiro [1].

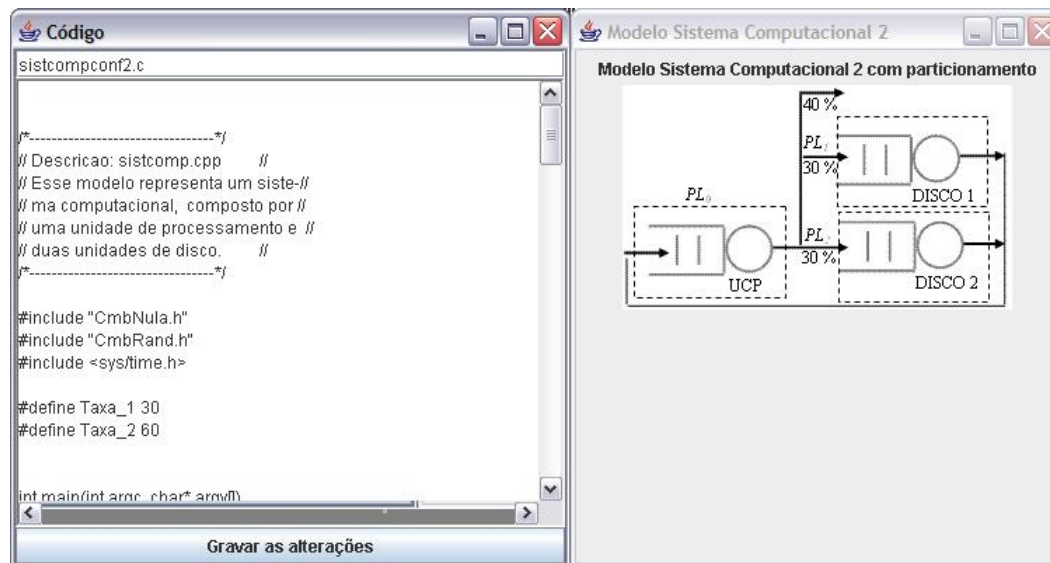


Figura 4.7: Exemplo da tela de um modelo de um sistema computacional, configuração tipo 2, existente na *CMB-Simulation* depois do código.

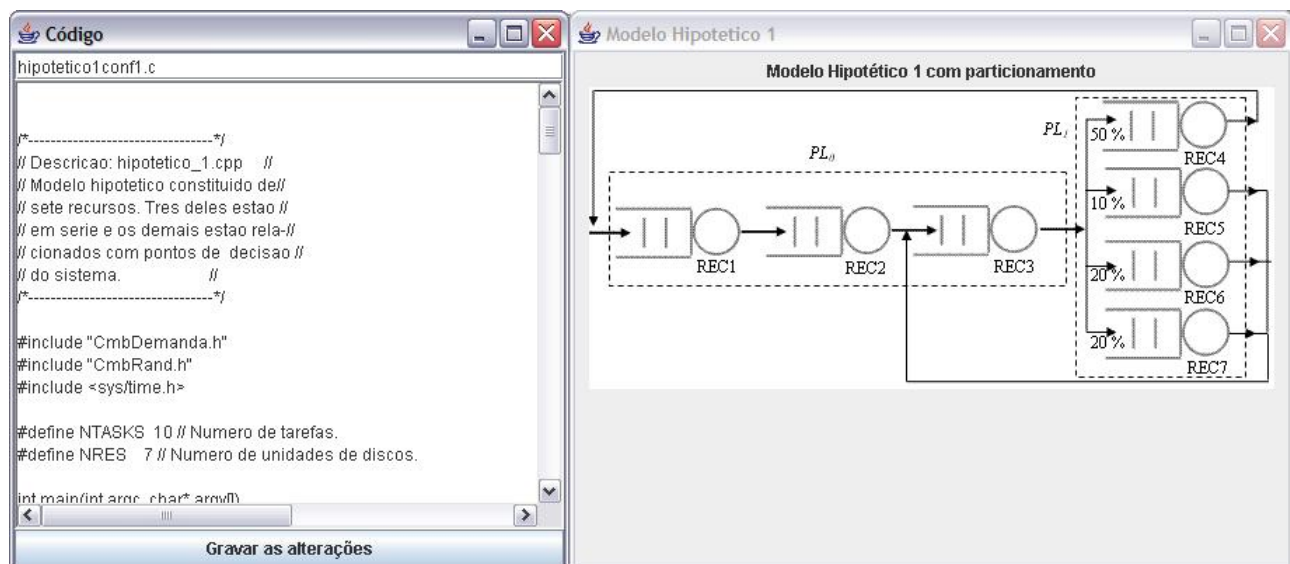


Figura 4.8: Exemplo da tela de um modelo hipotético, configuração tipo 1, existente na *CMB-Simulation* depois do código.

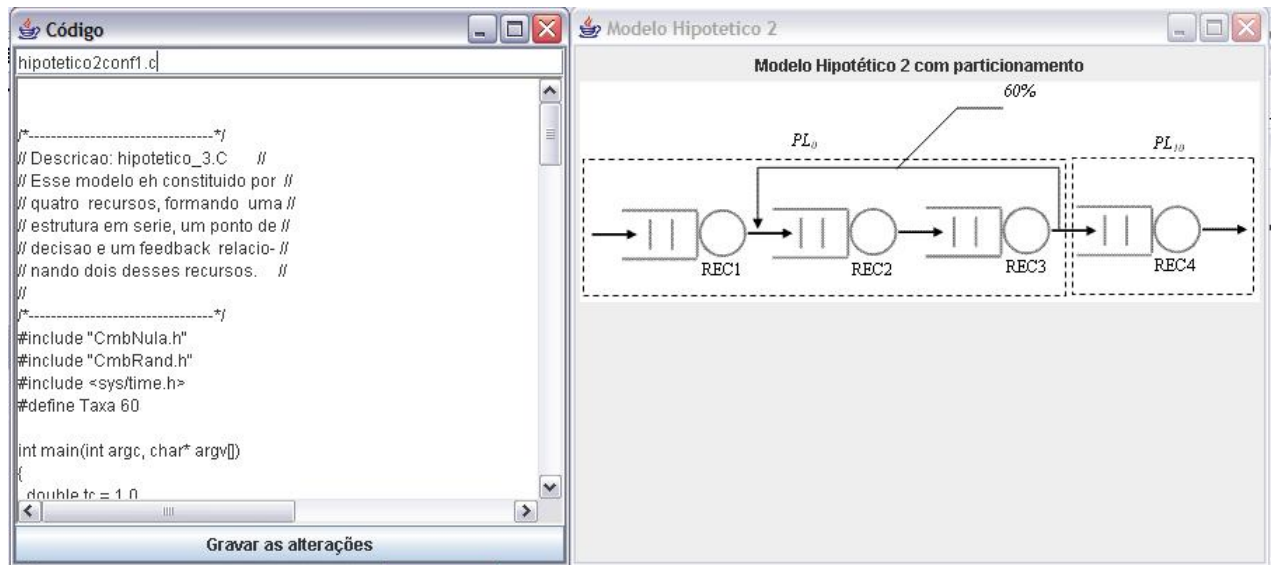


Figura 4.9: Exemplo da tela de um modelo hipotético, configuração tipo 2, existente na CMB-Simulation depois do código.

4.2.2 ARCABOUÇO

Existe a possibilidade, na interface, do usuário desejar um arcabouço bastante genérico de um programa de simulação feito na CMB-Simulation. Na figura 4.10, é mostrado como é o arcabouço de código para qualquer programa de simulação que execute na CMB-Simulation. Esse programa aparece em uma área de texto, que pode ser alterada conforme o usuário desejar. Trata-se de algo bastante simples mas que é muito útil para usuários que possuem uma experiência um pouco maior em simulação e preferem a criação de seus programas de simulação desenvolvendo o código diretamente.

4.2.3 MODELOS DESENVOLVIDOS PELO USUÁRIO

No momento que o usuário vai desenvolver o seu próprio modelo, ele deve seguir alguns passos para que o processo de criação seja efetivado. Esses passos são através de telas disponibilizadas pela interface. Na sequência, são mostrados os passos que o usuário deve realizar para criar o seu modelo.

A figura 4.11 mostra a primeira tela de interação que o usuário se depara quando começa a criar o seu modelo. Nessa tela, estão dispostos os botões que usuário pode utilizar para criar os seus modelos. Os botões possuem o desenho de um conjunto fila/centro de serviço, que o usuário escolhe segundo as suas necessidades.

Todas as vezes que o usuário escolhe um desses botões para selecionar um conjunto fila/centro de serviço, aparece uma tela secundária, que pode ser vista na figura 4.12, em

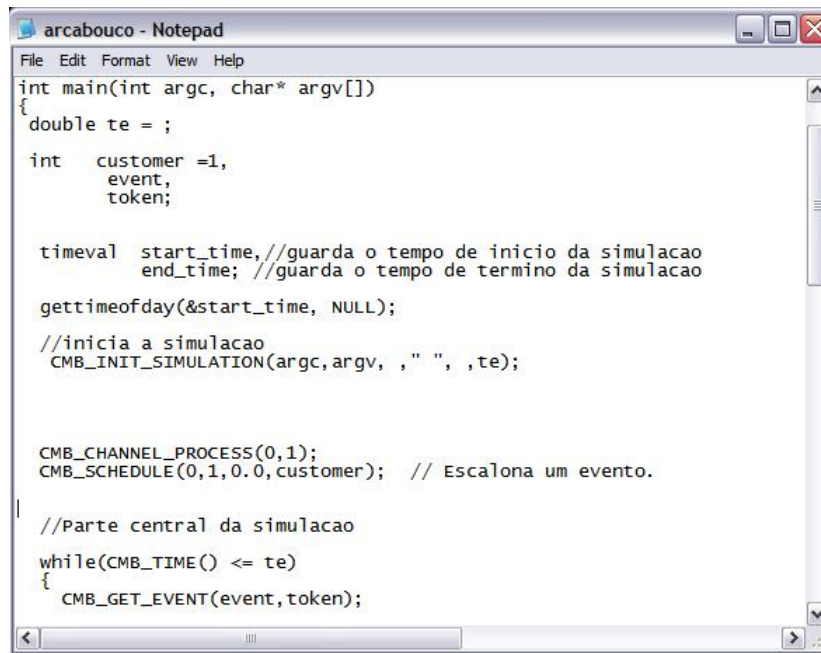


Figura 4.10: Tela mostrando o arcabouço generalizado.

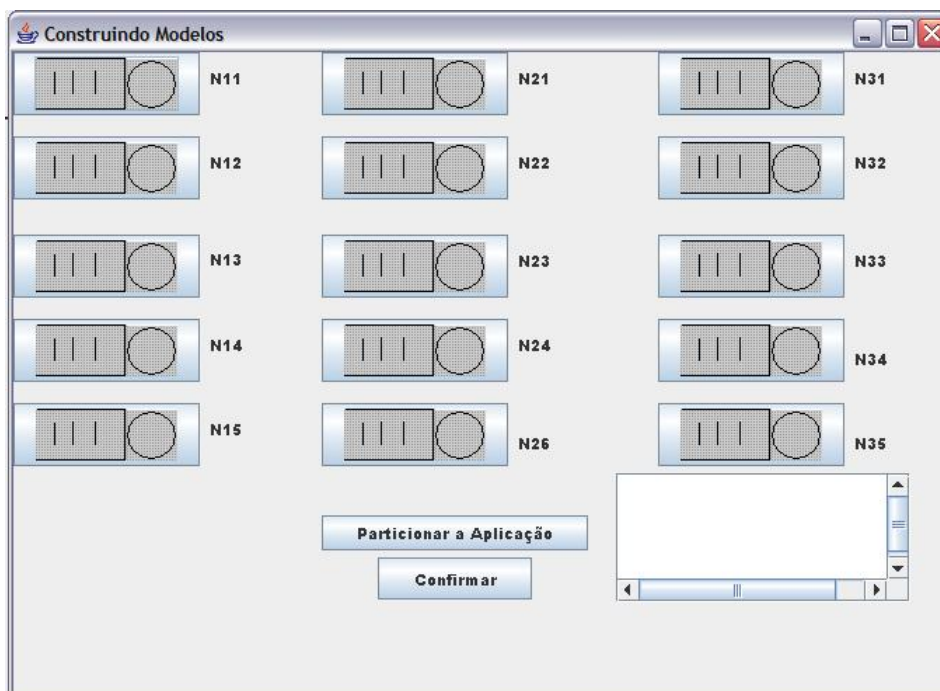


Figura 4.11: Primeira tela para o usuário criar o seu modelo.

que ele escolhe as taxas, probabilidades e ligação entre os conjuntos.

As taxas são as distribuições de probabilidades existentes na *CMB-Simulation* para gerenciar a forma de como acontece a chegada e o atendimento de clientes. A probabilidade é utilizada, quando necessária, em pontos de decisão, para regular a chance de se optar por um caminho ou por outro. Por fim, a ligação entre os conjuntos serve para indicar para onde vai o cliente depois de executar em um centro de serviço.

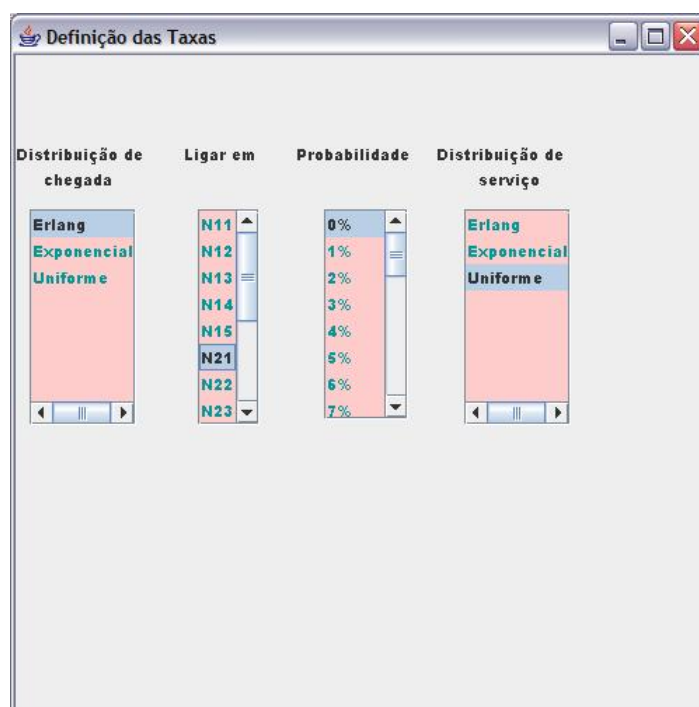


Figura 4.12: Listas que selecionam as taxas, probabilidades e ligações.

Acompanhando pela figura 4.11, tem-se que depois que o usuário cria o seu modelo, através do pressionamento dos botões e da escolhas das distribuições feitas na lista mostrada na figura 4.12, ele pode particionar a aplicação. O particionamento da aplicação acontecerá com a escolha de quais conjuntos filas e centros de serviços ficarão agregados nos mesmos processos lógicos.

A escolha dos conjuntos que constituirão o mesmo processo lógico acontece na tela apresentada na figura 4.13.

Caso o usuário deseje deixar um conjunto fila/centro de serviço sozinho em um processo lógico, ele pode usar a opção NONE, como pode ser visto na figura 4.14.

Depois que o usuário realiza o particionamento desejado, volta-se a tela da figura 4.11, em que o usuário pressiona o botão *particionar a aplicação*. Ao lado desse botão, existe uma área de texto em que se mostra ao usuário se aquele é um particionamento indicado ou não. O análise do sistema para uma resposta sobre o particionamento baseia-se nos estudos

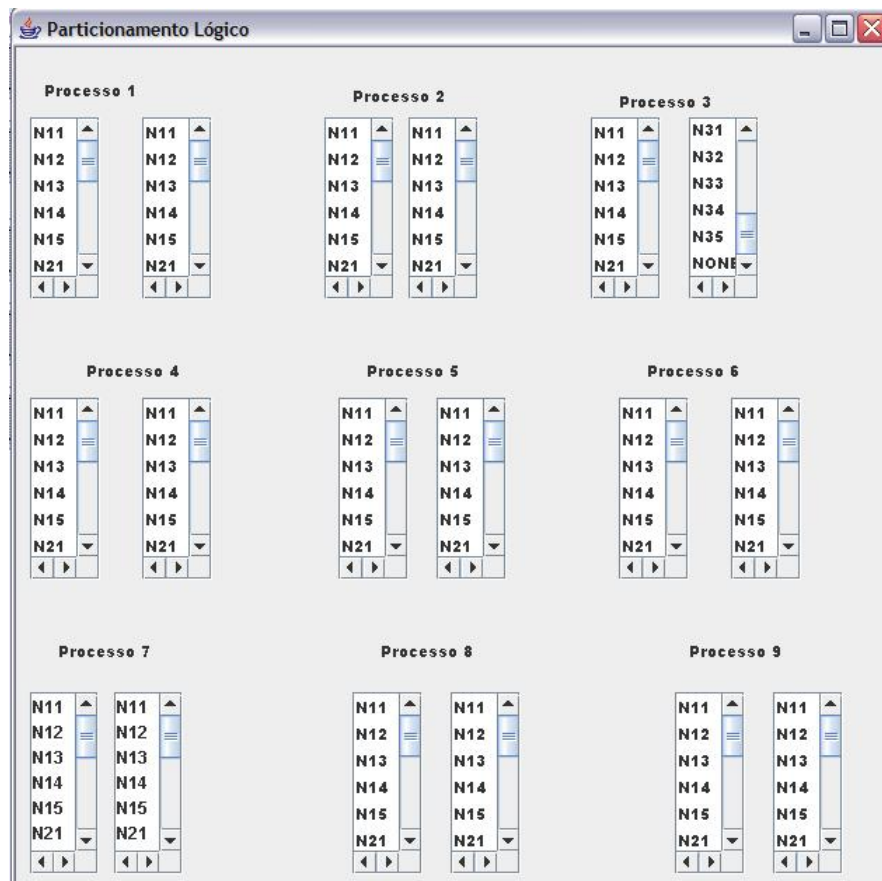


Figura 4.13: Tela para escolha do particionamento lógico.

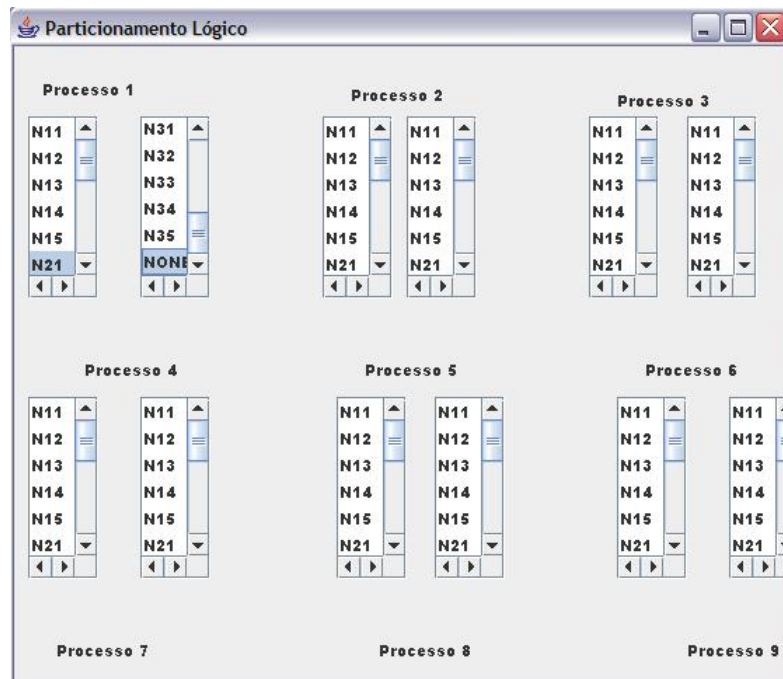


Figura 4.14: Mostrando o uso de NONE.

sobre particionamento realizados por Smaragdakis [22] e que são mostrados na seção 2.11.

Caso o particionamento não seja o indicado, o mecanismo particionador sugere algum outro particionamento. O usuário pode alterar o particionamento ou não. Isso depende da sua necessidade.

Após a escolha do particionamento, basta o usuário pressionar o botão *confirmar*, em que uma rotina executa para a geração do arcabouço de código com o particionamento desejado.

4.3 TESTES

Nesta seção são realizados os testes com a ferramenta. Utiliza-se dos testes para comprovar e elucidar ainda mais o funcionamento dos mecanismos explicados nas seções anteriores.

A seguir serão apresentados, nas figuras 4.15 e 4.16, dois exemplos de modelos que o usuário pode criar para a geração de arcabouço e particionamento na *CMB-Simulation* e que foram utilizados para os testes.

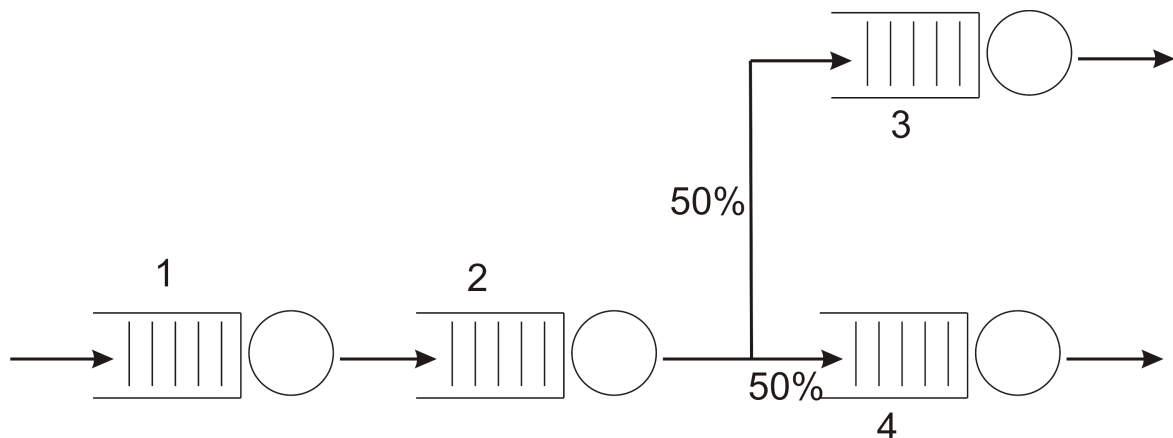


Figura 4.15: Modelo criado pelo usuário.

Na figura 4.15, tem-se um modelo, o qual tem a parte inicial serial, e em um determinado ponto, existe um ponto de decisão com taxas probabilísticas diferentes para cada caminho a ser tomado.

Como mostrado anteriormente na figura 4.11, o usuário começa a construir o seu modelo, pressionando os botões disponíveis. A cada botão pressionado é aberta uma tela (figura 4.12) questionando sobre as taxas que o operador deseja inserir em seu modelo, que são as disponíveis para a *CMB-Simulation* e sobre a probabilidade em cada ponto.

Realizado o procedimento de construção do modelo, o usuário deve pressionar o botão de particionamento (*Particionar a Aplicação*). Pressionando o botão de particionamento,

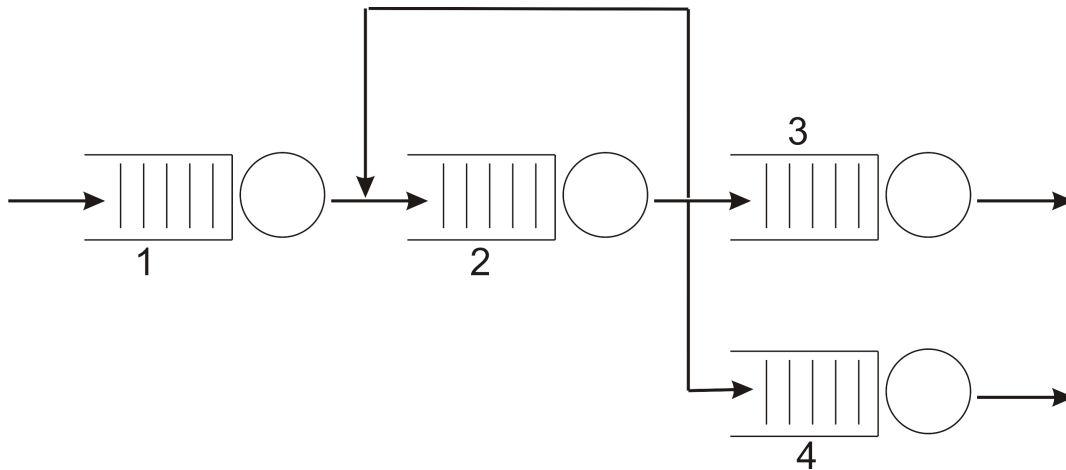


Figura 4.16: Modelo criado pelo usuário.

uma tela é disponibilizada (figura 4.13), em que o usuário escolhe dois a dois, os processos dos conjuntos filas/centros de serviços que vão ocupar os mesmos processos lógicos, também mostrada anteriormente.

Considerando o modelo proposto na figura 4.15, suponha-se que o usuário deseja realizar o seguinte particionamento lógico dos processos e que N12 seja o conjunto 1, N22 seja o conjunto 2, N31 o conjunto 3 e N32 o conjunto 4: colocar no mesmo processo lógico os conjuntos fila/centro de serviço 1 e 3, e o conjunto 2 e 4. Pelos estudos conduzidos por Smaragdakis [22], nota-se que o conjunto 3 possui dependência em relação a tarefa realizada pelo conjunto 2. Assim, o agrupamento sugerido pelo usuário não é o melhor. Nesse sentido, a interface gráfica sugere um particionamento, como pode ser visto na caixa de texto, na figura 4.17, localizada ao lado do botão de particionamento.

Caso o usuário deseje trocar o particionamento, basta ele selecionar novamente o botão de particionamento. Senão, ele necessita apenas de pressionar *confirmar*, e o particionamento que ele escolheu será o utilizado, mesmo que não seja o indicado.

Quando o usuário pressiona o botão *confirmar*, é gerado o arcabouço de código do modelo que ele mesmo criou e gerado um arquivo `arcabouco.c`, como pode ser visto na figura 4.18. Mostra parcialmente o código gerado, ressaltando abertura do canal de comunicação e os eventos relativos ao processo lógico 1.

A figura 4.19 mostra o núcleo do arcabouço gerado, com todas as estruturas que irão controlar a execução da simulação.

Para a criação do modelo visto na figura 4.16, o mesmo procedimento ocorre como anteriormente mostrado para o modelo da figura 4.15, considerando agora, que o elemento 1 da figura 4.16 seja o elemento N11 da figura 4.11, o elemento 2 seja o N21, o elemento 3 seja N31 e que o elemento 4 seja N32. Nesse caso, a diferença é que o modelo proposto

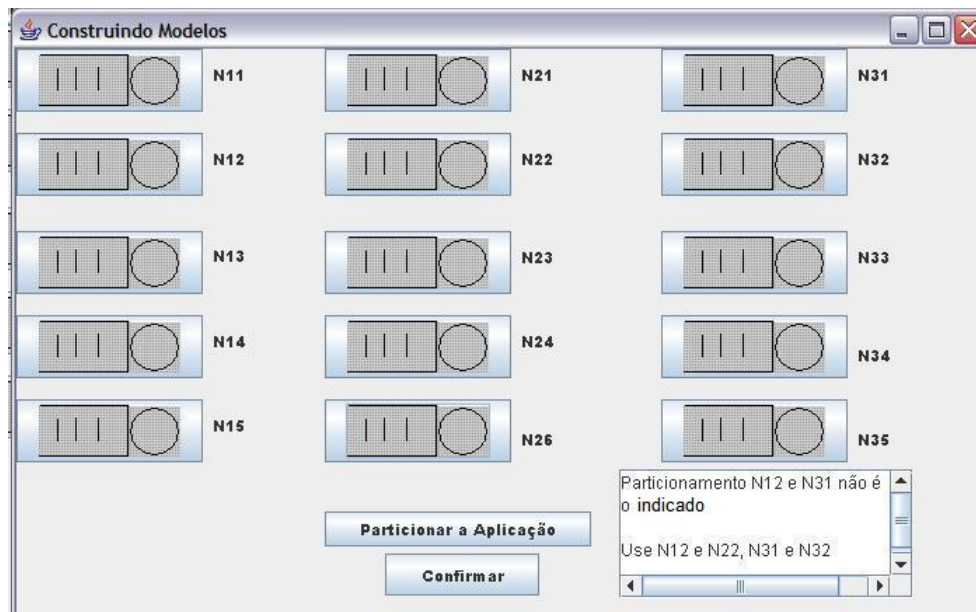


Figura 4.17: Sugestão de Particionamento.

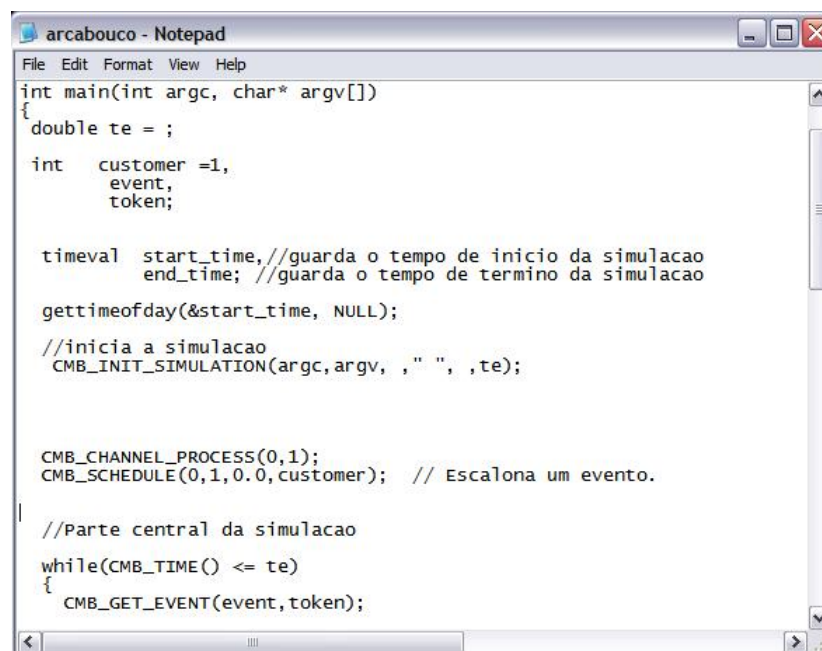
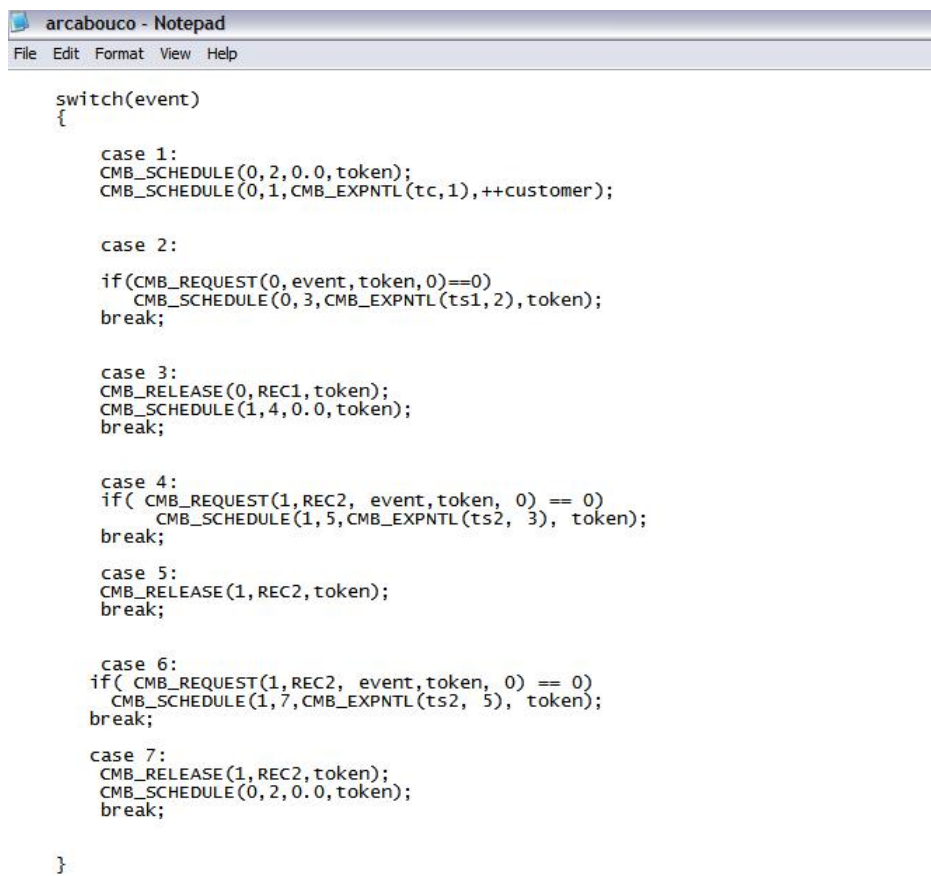


Figura 4.18: Código do modelo do usuário.



```
switch(event)
{
    case 1:
        CMB_SCHEDULE(0,2,0.0,token);
        CMB_SCHEDULE(0,1,CMB_EXPNTL(tc,1),++customer);

    case 2:
        if(CMB_REQUEST(0,event,token,0)==0)
            CMB_SCHEDULE(0,3,CMB_EXPNTL(ts1,2),token);
        break;

    case 3:
        CMB_RELEASE(0,REC1,token);
        CMB_SCHEDULE(1,4,0.0,token);
        break;

    case 4:
        if( CMB_REQUEST(1,REC2, event,token, 0) == 0)
            CMB_SCHEDULE(1,5,CMB_EXPNTL(ts2, 3), token);
        break;

    case 5:
        CMB_RELEASE(1,REC2,token);
        break;

    case 6:
        if( CMB_REQUEST(1,REC2, event,token, 0) == 0)
            CMB_SCHEDULE(1,7,CMB_EXPNTL(ts2, 5), token);
        break;

    case 7:
        CMB_RELEASE(1,REC2,token);
        CMB_SCHEDULE(0,2,0.0,token);
        break;
}
```

Figura 4.19: Código do modelo do usuário.

na figura 4.16 possui um ponto de realimentação, na saída do conjunto 1. Segundo [22], processos que realizam realimentação devem ser colocados sozinhos em um único processo lógico, ou seja, o conjunto 2 deveria estar sozinho em um processo lógico.

O usuário pode, se desejar, não isolar o conjunto 2, porém não é garantido que ele consiga um desempenho considerável em sua simulação.

Para permitir que o usuário deixe um conjunto fila/centro de serviço em um único processo lógico existe a opção NONE, como poder vista selecionada na figura 4.14, anteriormente citada, e em detalhes na figura 4.20.

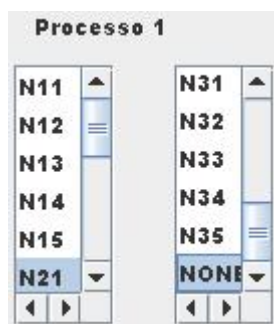


Figura 4.20: Uso de NONE.

Tomando como exemplo a escolha de N21 como o conjunto que sofre realimentação, como está na figura 4.20 e a outra opção como NONE, mostra que N21 ficará sozinho no processo lógico.

Em relação a sugestão do particionamento, como o caso de realimentação é um dos abordados pela interface, o sistema sugere, considerando que seja o elemento N21 que sofre realimentação, que esse elemento seja colocado sozinho em um processo lógico. Supondo que o usuário tenha selecionado colocar os conjuntos 1 e 2 no mesmo processo lógico, o sistema sugere que ele não coloque, como pode ser visto na figura 4.21.

O usuário pode desejar conservar o particionamento criado por ele, porém não é garantido um bom desempenho da simulação.

4.4 CONSIDERAÇÕES FINAIS

Os resultados e testes apresentados nas seções anteriores, através das telas de execução, demonstraram que se cumpre o que foi proposto no projeto, englobando a concepção de uma interface gráfica e dos mecanismos de particionamento lógico e geração de arcações. Com essa demonstração, elucidou-se como é o funcionamento da interface gráfica e como procede-se para utilizar o mecanismo de construção dos modelos, com o intuito de gerar o

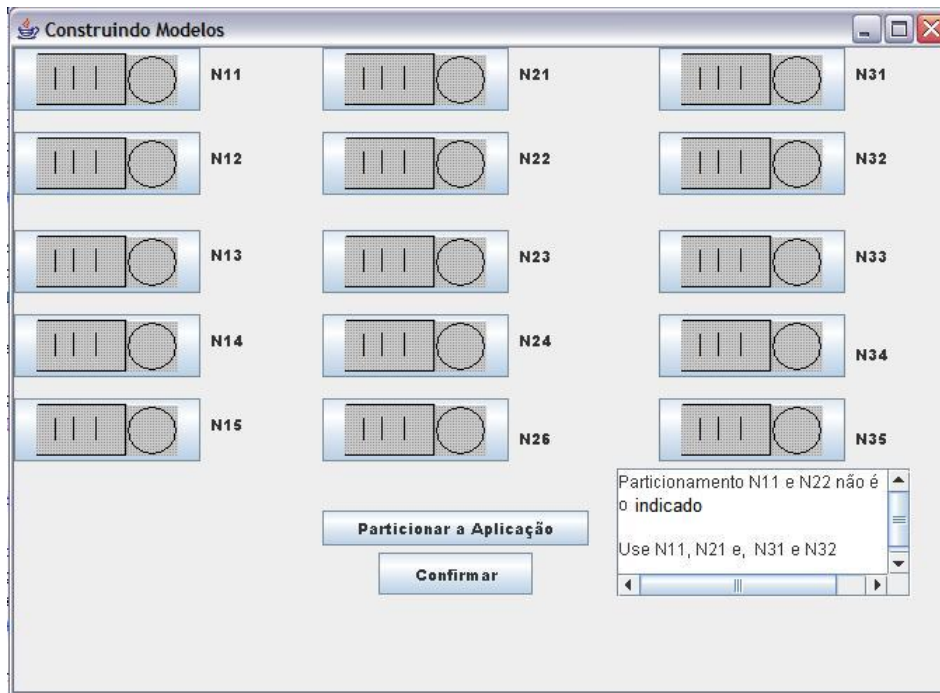


Figura 4.21: Sugestão de particionamento.

arcabouço e o seu particionamento.

CAPÍTULO 5

CONCLUSÕES

5.1 CONCLUSÃO

Os estudos conduzidos durante a revisão bibliográfica foram de profunda importância para embasar teoricamente a execução do projeto e também adquirir conhecimentos importantes sobre simulação distribuída. Também com o mesmo grau de importância mostrou-se o contato com a ferramenta *CMB-Simulation* e a possibilidade de instalação e execução da ferramenta em um ambiente de *cluster*. Foi muito importante conhecer bem a ferramenta, para que se pudesse construir uma interface gráfica adequada as funções que ela gerencia. Todos os resultados foram obtidos dentro do prazo previsto pelo cronograma de trabalho.

O estudo da linguagem JAVA continuou por toda a execução do projeto, mas o período de adaptação a linguagem também teve seu peso de forma relevante, visto que mesmo com o conhecimento prévio de programação orientada a objetos, existem diferenças em relação a outras linguagens de programação orientada a objetos.

Os estudos conduzidos na modelagem da interface gráfica a partir dos diagramas de seqüência da UML [13] mostraram-se como uma ótima forma de se compreender como a interface iria trabalhar mesmo antes de ela estar pronta e isso trouxe um melhor embasamento para a construção do projeto lógico da interface.

A concepção da interface gráfica prezou pela simplicidade, de modo que mesmo uma pessoa leiga em conceitos de simulação distribuída, possa realizar uma simulação com os modelos pré-definidos ou até mesmo se arriscar a construir o seu próprio modelo. Essa interface, agregada a *CMB-Simulation*, pode se tornar futuramente uma ferramenta, por exemplo, para ensino de simulação distribuída, pois o usuário pode ter contato com a simulação sem conhecer os detalhes de seu funcionamento e, também, de programação paralela.

A metodologia empregada permitiu que se fizesse um estudo sobre geradores de aplicação, entendendo qual é o resultado da aplicação desses geradores no processo de produção

de um *software*.

Mostrou-se como foram construídos os módulos do gerador de aplicação, representando-se alguns trechos de códigos dos módulos.

Apresentou-se ainda como seria um exemplo da forma de particionamento de uma aplicação gerada pela interface para um determinado modelo de rede de fila construído pelo usuário.

Finalizando-se, demonstrou-se um exemplo que permite ao usuário da *CMB-Simulation* construir os seus próprios modelos de redes de filas.

5.2 TRABALHOS FUTUROS

Como proposta para trabalhos futuros, cita-se a implementação de um mecanismo gerador de arcabouços para a *CMB-Simulation* para programas de simulação particionados para executar em MRIP (*Multiple Replication In Parallel*), que se condiciona a criar diversas instâncias do mesmo programa de simulação para a execução nos nós do *cluster*. Essas instâncias são cópias idênticas do programa de simulação.

A idéia de utilizar-se MRIP no lugar de SRIP (*Single Replication In Parallel*), que é como a *CMB-Simulation* funciona, deve-se ao fato de que em MRIP, caso ocorra uma falha em algum dos nós de processamento, os outros permitirão a análise do resultado final, pois estavam executando uma cópia do programa de simulação. Em SRIP, como existe a interdependência entre os processos lógicos, a falha em algum nó de processamento pode prejudicar a simulação por completa.

Também, agregado à execução em MRIP, pode-se implementar os mecanismos de análise de saída das simulações em MRIP. Esses mecanismos utilizam de métodos estatísticos que poderão ser utilizados para verificar e validar os resultados.

Este trabalho foi contemplado com uma bolsa de iniciação científica do CNPq, que foi de grande auxílio durante o seu desenvolvimento. O trabalho foi publicado, como parte complementar ao trabalho desenvolvido por Balieiro [1], na XXXII Conferência Latinoamericana de Informática, em 2006, em Santiago, no Chile, com o título "*Abordagem Conservativa para Simulação Distribuída de Modelos de Redes de Fila*".

REFERÊNCIAS BIBLIOGRÁFICAS

- [1] BALIEIRO, M. O. S. *Protocolo Conservativo CMB para Simulação Distribuída*. Dissertação (Mestrado) — Departamento de Departamento de Computação e Estatística, Universidade Federal de Mato Grosso do Sul, 2005.
- [2] FINLAY, A. D. J.; ABOWD, G. D.; BEALE, R. *Human-Computer Interaction*. Third: Pearson Education Limited, 2004.
- [3] ULSON, R. S. *Simulação Distribuída em Plataformas de Portabilidade: Viabilidade de Uso e Comportamento do Protocolo CMB*. Tese (Doutorado) — Instituto de Física de São Carlos, Universidade de São Paulo, 1999.
- [4] SOARES, L. F. G. *Modelagem e Simulação Discreta de Sistemas: VII Escola de Computação - São Paulo*, 1990.
- [5] MACDOUGALL, M. H. *Simulating Computer Systems*: MIT Press, 1987.
- [6] MISRA, J. Distributed discrete-event simulation. In: *ACM Computing Surveys*, 1896. v. 18, n. 1, p. 39–65.
- [7] CAROTHERS, C. D.; BAUER, D.; PEARCE, S. Ross: A high-performance, low memory, modular time warp system. In: *Proceedings of The 14th Workshop on Parallel and Distributed Simulation (PADS'2000)*, 2000. p. 53–60.
- [8] FUJIMOTO, R. M. *Parallel and Distributed Simulation Systems*: John Wiley & Sons, Inc., 2000.
- [9] JEFFERSON, D. R. Virtual time. In: *IEEE Transactions on Programming Languages and Systems*, 1985. v. 7, n. 3, p. 404–425.
- [10] HORSTMANN, C. S.; CORNELL, G. *Core Java 2 - Volume I (Fundamentals 6th)*: Prentice-Hall, 2002.
- [11] HORSTMANN, C. S.; CORNELL, G. *Core Java 2 - Volume II (Advanced Features)*: Prentice-Hall, 2001.
- [12] DEITEL, H. M.; DEITEL, P. J. *Java How to Program*: Prentice-Hall, Inc, 2002.

- [13] LARMAN, C. *Utilizando Uml e Padrões*: Bookman, 2004.
- [14] SANTANA, R. H. C. et al. *Técnicas para avaliação de desempenho de sistemas computacionais*, Setembro 1994.
- [15] WEICKER, R. Benchmarking. In: *Lecture Notes in Computer Science*, 1994.
- [16] BANKS, J. et al. *Discrete-Event System Simulation*: Prentice-Hall International Series, 2001.
- [17] SPOLON, R. *Um Método para Avaliação de Desempenho de Protocolos de Sincronização Otimistas para Simulação Distribuída*. Tese (Doutorado) — Instituto de Física de São Carlos - USP, 2001.
- [18] PORRAS, J.; IKONEN, J.; JARNO, H. Applying a modified chandy-misra algorithm to the distributed simulation of a celular network. In: *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS'98)*, 1998. p. 188–195.
- [19] JHA, V.; BAGRODIA, R. A performance evaluation methodology for parallel simulation protocols. In: *Proceedings of the 10th Workshop on Parallel and Distributed Simulation*, 1996. p. 180–185.
- [20] BRUSCHI, S. M. *ASDA - Um Ambiente de Simulação Distribuída Automático*. Tese (Doutorado) — ICMC - USP, 2002.
- [21] JUNIOR, E. K. S. *Desenvolvimento de geradores de aplicação configuráveis por linguagens de padrões*. Dissertação (Mestrado) — Instituto de Ciências Matemáticas e de Computação (ICMC) - USP/São Carlos, 2005.
- [22] SMARAGDAKIS, Y.; BATORY, D. Application generators. *The University of Texas at Austin*, 2004.
- [23] AHO, A. V.; SETHI, R.; ULLMAN, J. D. *Compilers: Principles, Techniques, and Tools*: Addison-Wesley, 1986.
- [24] PRESSMAN, R. S. *Software Engeneering: a practitioner's approach*: McGraw-Hill, 1987.
- [25] LOBATO, R. S. et al. Abordagem conservativa para simulação distribuída de modelos de redes de fila. In: *Anais da XXXII Conferência Latinoamericana de Informática - Santiago-Chile*, 2006. p. 190–202.

Apêndice

TRECHO DE CÓDIGO DA TELA PRINCIPAL DA INTERFACE

```
private void jButton11ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
}

private void jButton10ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
// Aciona o mecanismo de leia-me!!
    new estoumexendo.GeraArquivo(8);

}

    private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//mecanismo de geração de arcabouços
    new estoumexendo.GeraArquivo(7);

}

    private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
//Sistema hipotetico 2
    new estoumexendo.hipotetico2();

}
```

```
private void jButton7ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//chamada para a criacao dos modelos do usuário
    new estoumexendo.structuremodels();
//utilizar para desenhar as filas
}
```

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//Sistema de servidor central
    new estoumexendo.ServidorCentral();
}
```

```
private void jButton5ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//Sistema Hipotetico 1
    new estoumexendo.hipotetico1();
}
```

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//Sistema computacional configuracao 2
    new estoumexendo.SistemaComputacional2();
}
```

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//Sistema computacional configuracao 1
    new estoumexendo.SistemaComputacional1();
}
```

```

private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
// TODO add your handling code here:
//Modelo Serial
//quando o usuario clicar no botao, deve aparecer na
//tela a estruturacao do modelo de filas
    new estoumexendo.serial();

```

MODELOS PRONTOS

```

public GeraArquivo( int QualArquivo ) {
    super("Arquivos");

    if ( QualArquivo >= 1 && QualArquivo <= 6){
        //estrutura que seleciona qual dos arquivos prontos eu uso
        switch(QualArquivo){
            case 1: TelaTexto = new JTextField("serial.c" );
                    break;

            case 2: TelaTexto = new JTextField("servidorcentral.c" );
                    break;

            case 3: TelaTexto = new JTextField("sistcompconf1.c" );
                    break;

            case 4: TelaTexto = new JTextField("sistcompconf2.c" );
                    break;

            case 5: TelaTexto = new JTextField("hipotetico1conf1.c" );
                    break;

            case 6: TelaTexto = new JTextField("hipotetico2conf1.c" );
                    break;

            //case 7: TelaTexto = new JTextField("Arcabouco.c");

```

```

        // break;

    }//fim da estrutura switch

    TelaTexto.addActionListener( this );

    //passar para esse construtor: o Arquivo como uma string
    TelaArea = new JTextArea();
    Aceitar = new JButton();
    Aceitar.addActionListener(this);
    Aceitar.getMaximumSize();
    Aceitar.setText("Gravar as alterações");

    ScrollPane scrollPane = new ScrollPane();
    scrollPane.add( TelaArea );

    Container ConteudoArquivo = getContentPane();
    ConteudoArquivo.add(TelaTexto, BorderLayout.NORTH );
    ConteudoArquivo.add( scrollPane, BorderLayout.CENTER );
    ConteudoArquivo.add(Aceitar,BorderLayout.SOUTH);

    }//fim do if antes do switch

```

MECANISMO GERADOR DE ARCABOUÇOS

```

if( QualArquivo == 7)
    { //segundo if
        TelaArea = new JTextArea();
        Aceitar = new JButton();
        Aceitar.addActionListener(this);
        Aceitar.getMaximumSize();
        Aceitar.setText("Gravar as alterações");
    }

```

```

ScrollPane scrollPane = new ScrollPane();
scrollPane.add( TelaArea );
Container ConteudoArquivo = getContentPane();
ConteudoArquivo.add( scrollPane, BorderLayout.CENTER );
ConteudoArquivo.add(Aceitar,BorderLayout.SOUTH);
File Arcabouco = new File("Arcabouco.c");

try {
    BufferedReader EntradaArcabouco = new BufferedReader(
        new FileReader( Arcabouco ) );
    //objeto que vai bufferizando o texto do arquivo
    StringBuffer BufferArcabouco = new StringBuffer();
    String TextoArcabouco;
    TelaArea.append( "\n\n" );

    //aqui eu vou lendo o arquivo.
    while ( ( TextoArcabouco = EntradaArcabouco.readLine() ) != null )
        BufferArcabouco.append( TextoArcabouco + "\n" );

    TelaArea.append( BufferArcabouco.toString() );
}

    // process file processing problems
catch( IOException ioException ) {
    JOptionPane.showMessageDialog( this,
        "FILE ERROR",
        "FILE ERROR", JOptionPane.ERROR_MESSAGE );
} //fim do catch

} //fim do segundo if

```

ROTINA PARA GRAVAÇÃO DAS ALTERAÇÕES

```

public void actionPerformed(ActionEvent e) {

    if( e.getSource() == Aceitar)
    {
        try{
            //System.out.println(Geraldo);

            //nesse ponto eu tenho que passar a leitura da string para um arquivo
            String temporaria = TelaArea.getText();
            //System.out.println(temporaria);
            File ArquivoArea = new File("mudanca.c");
            PrintWriter printer1 = new PrintWriter(ArquivoArea);
            printer1.print(temporaria);
            printer1.close();

        }

        catch( IOException ioException ) {
            JOptionPane.showMessageDialog( this,
            "FILE ERROR",
            "FILE ERROR", JOptionPane.ERROR_MESSAGE );
        }

        //System.out.print(ArquivoArea.canWrite());

        return;
    }

    File nome = new File( e.getActionCommand() );

    // if name exists, output information about it
    if ( nome.exists() ) {
        // output information if name is a file
    }

```

```

        if ( nome.isFile() ) {

// append contents of file to outputArea
        try {
            BufferedReader input = new BufferedReader(
                new FileReader( nome ) );

            //objeto que vai bufferizando o texto do arquivo
            StringBuffer buffer = new StringBuffer();
            String texto;
            TelaArea.append( "\n\n" );

            //aqui eu vou lendo o arquivo.
            while ( ( texto = input.readLine() ) != null )
                buffer.append( texto + "\n" );

            TelaArea.append( buffer.toString() );
        }

// process file processing problems
        catch( IOException ioException ) {
            JOptionPane.showMessageDialog( this,
                "FILE ERROR",
                "FILE ERROR", JOptionPane.ERROR_MESSAGE );
        }
    }

// output directory listing
    else if ( nome.isDirectory() ) {
        String directory[] = nome.list();

        TelaArea.append( "\n\nDirectory contents:\n");

        for ( int i = 0; i < directory.length; i++ )
            TelaArea.append( directory[ i ] + "\n" );
    }

```

```

}

// not file or directory, output error message
else {
    JOptionPane.showMessageDialog( this,
    e.getActionCommand() + " Does Not Exist",
    "ERROR", JOptionPane.ERROR_MESSAGE );
}

} // fim do metodo actionPerformed()

```

JList

```

private void initComponents() {
    jScrollPane1 = new javax.swing.JScrollPane();
    jList1 = new javax.swing.JList();
    jScrollPane2 = new javax.swing.JScrollPane();
    jList2 = new javax.swing.JList();
    jScrollPane3 = new javax.swing.JScrollPane();
    jList3 = new javax.swing.JList();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();

    getContentPane().setLayout(null);

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    jList1.setBackground(new java.awt.Color(255, 204, 204));
    jList1.setFont(new java.awt.Font("Arial Black", 0, 11));
    jList1.setForeground(new java.awt.Color(0, 153, 153));
    jList1.setModel(new javax.swing.AbstractListModel() {
        String[] strings = { "Erlang", "Exponencial", "Uniforme" };
    });
}

```



```

        public int getSize() { return strings.length; }
        public Object getElementAt(int i) { return strings[i]; }
    });
jScrollPane1.setViewportViewView(jList1);

getContentPane().add(jScrollPane1);
jScrollPane1.setBounds(10, 110, 76, 154);

jList2.setBackground(new java.awt.Color(255, 204, 204));
jList2.setFont(new java.awt.Font("Arial Black", 0, 11));
jList2.setForeground(new java.awt.Color(0, 153, 153));
jList2.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "N11", "N12", "N13", "N14", "N15", "N21",
        "N22", "N23", "N24", "N25", "N31", "N32", "N33", "N34", "N35" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jScrollPane2.setViewportViewView(jList2);

getContentPane().add(jScrollPane2);
jScrollPane2.setBounds(180, 110, 44, 154);

jList3.setBackground(new java.awt.Color(255, 204, 204));
jList3.setFont(new java.awt.Font("Arial Black", 0, 11));
jList3.setForeground(new java.awt.Color(0, 153, 153));
jList3.setModel(new javax.swing.AbstractListModel() {
    String[] strings = { "Erlang", "Exponencial", "Uniforme" };
    public int getSize() { return strings.length; }
    public Object getElementAt(int i) { return strings[i]; }
});
jScrollPane3.setViewportViewView(jList3);

getContentPane().add(jScrollPane3);
jScrollPane3.setBounds(320, 110, 76, 154);

jLabel1.setFont(new java.awt.Font("Arial Black", 0, 11));

```

```

jLabel1.setText("Distribui\u00e7\u00e3o de");
getContentPane().add(jLabel1);
jLabel1.setBounds(0, 60, 100, 20);

jLabel2.setFont(new java.awt.Font("Arial Black", 0, 11));
jLabel2.setText("chegada");
getContentPane().add(jLabel2);
jLabel2.setBounds(20, 80, 50, 17);

jLabel3.setFont(new java.awt.Font("Arial Black", 0, 11));
jLabel3.setText("Ligar em");
getContentPane().add(jLabel3);
jLabel3.setBounds(170, 60, 60, 20);

jLabel4.setFont(new java.awt.Font("Arial Black", 0, 11));
jLabel4.setText("Distribui\u00e7\u00e3o de");
getContentPane().add(jLabel4);
jLabel4.setBounds(300, 60, 100, 20);

jLabel5.setFont(new java.awt.Font("Arial Black", 0, 11));
jLabel5.setText("servi\u00e7o");
getContentPane().add(jLabel5);
jLabel5.setBounds(330, 80, 50, 17);

pack();
} //final do metodo para implementacao das listas

```