Universidade Estadual Paulista "Júlio de Mesquita Filho"
Instituto de Biociências, Letras e Ciências Exatas
Departamento de Ciências da Computação e Estatística

Autor: Ricardo Luiz Cardim Di Chiacchio

Biblioteca orientada a objetos para simulação de redes de filas

Monografia apresentada 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 aprovação na disciplina Projeto Final.

São José do Rio Preto
2005

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Instituto de Biociências, Letras e Ciências Exatas
Departamento de Ciências da Computação e Estatística

Autor: Ricardo Luiz Cardim Di Chiacchio

Biblioteca orientada a objetos para simulação de redes de filas

Orientadora: Profa. Dra. Renata Spolon Lobato

Monografia apresentada 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 aprovação na disciplina Projeto Final.

São José do Rio Preto
2005

Universidade Estadual Paulista "Júlio de Mesquita Filho"
Instituto de Biociências, Letras e Ciências Exatas
Departamento de Ciências da Computação e Estatística

Autor: Ricardo Luiz Cardim Di Chiacchio

Biblioteca orientada a objetos para simulação de redes de filas

Monografia apresentada 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 aprovação na disciplina Projeto Final.

Profa. Dra. Renata Spolon Lobato

Ricardo Luiz Cardim Di Chiacchio

Banca Examinadora:
Prof. Dr. Aleardo Manacero Junior
Profa. Dra. Rogéria Cristiane Gratão de Souza

São José do Rio Preto
2005

Dedicatória

Aos meus pais José Edson (*In memorian*) e Edina

Às minhas irmãs Viviane e Alexandra

À Roslilane

Agradecimentos

A Deus, pela oportunidade de mais uma jornada na terra em busca de crescimento espiritual e busca por conhecimento e sabedoria.

Aos meus pais, José Edson (*In memorian*) e Edina, pelos anos de dedicação, pelas palavras certas nas horas certas, por compreender meus erros e indicar os caminhos a serem seguidos, pelo cuidado, pelo carinho, pelo amor e incentivo sem os quais não seria possível a realização de todos esses anos de estudo. Muito obrigado.

Às minhas irmãs Viviane e Alexandra que foram importante durante toda a minha vida e auxiliaram seus pais na criação do seu irmão caçula.

À Roslilane, pelo amor e carinho em todos esses anos de convívio e mesmo com toda a dificuldade que foi imposta pela distância, soube estar presente nas horas em que mais precisei de sua companhia.

À minha orientadora, Profa. Dra. Renata Spolon Lobato, pelo auxílio, paciência e dedicada orientação que possibilitaram o desenvolvimento deste trabalho.

Aos professores que me trouxeram conhecimento e possibilitaram a conclusão de mais essa etapa em meus estudos.

Aos companheiros de pensão, Carlos, Crispim, Lucas e Luiz, por estarem presentes no início destes quatro anos de estudos e pelo convívio em momentos alegres e tristes.

Ao Thiago, pelos conselhos, pelas conversas e pela paciência durante todos os três anos de convívio em prol da realização de nossos estudos.
Aos amigos que deixei para trás em busca de conhecimento e principalmente para aqueles que mesmo com a distância ainda encontram maneiras de fazer com que nossas amizades pareçam eternas.

Aos colegas que encontrei no decorrer desse processo de estudo e às amizades que surgiram no decorrer desses anos sem os quais todo o caminho seria completamente diferente.

À Bianca Dantas por auxiliar no uso da biblioteca SMPL.

Resumo

Este trabalho consiste em uma implementação de uma biblioteca orientada a objetos que auxilia na construção de programas de simulação de modelos de redes de filas. A biblioteca foi construída em linguagem C++ por ser uma linguagem de uso comum no meio científico. Com essa biblioteca o usuário não necessitará de construir todas as estruturas necessárias para a realização da simulação, pois ela possui classes que fazem a construção de centros de serviço, geradores de números pseudo-aleatórios, controle do relógio da simulação, lista de eventos futuros e alguns valores estatísticos. Foi implementado um método de análise de saída, *batch means*, e seu resultado pode ser visualizado através de gráficos gerados com o software GNUPlot. Alguns exemplos foram realizados com o intuito de apresentar o funcionamento da biblioteca e as possibilidades de utilização como uma ferramenta de auxílio a tomadas de decisão. O ambiente de trabalho foi o sistema operacional Linux com a distribuição Debian Sarge e compilador gcc.

Abstract

This work aims at the implementation of an object-oriented library that helps the construction of queue networks simulations programs. The use of C++ can be justified by its adoption in scientific environment. By using this library the user will not need to build all necessary data structures for the accomplishment of the simulation given that the library offers classes that reduce the development effort for building service centers, pseudo-random numbers generators, control of the clock of the simulation, future events list and some statistical values. A method of output analysis was implemented, batch means, and its result can be visualized through graphs generated with GNUPlot software. Some examples had been carried as a proof of concept and the possibilities of use as an aid tool for decision-making. The library was built using the Linux operating system (Debian Sarge distribution) and GCC compiler.

Índice

Lista de Figuras

Lista de tabelas

Lista de Abreviaturas e Siglas

SMPL: SiMulation Programming Language
UCP: Unidade Central de Processamento
FCFS: *First Come First Served*
LCFS: *Last Come First Served*
RR: *Round Robin*
PRTY: *nonpreemptive priority*
PRTYPR: *preemptive-resume priority*
u.t.: unidades de tempo
RFOO: Redes de Filas Orientada a Objetos
LEF: lista de eventos futuros
EOG: *Eye of Gnome*
cs1: centro de serviço 1
cs2: centro de serviço 2
cs3: centro de serviço 3
cs4: centro de serviço 4

Capítulo 1 - Introdução

A avaliação de desempenho é uma área da computação que tem se tornado essencial devido à evolução dos sistemas computacionais. A avaliação de desempenho de sistemas computacionais pode ser efetuada através das técnicas de aferição ou através das técnicas de solução de modelos (ou, simplesmente, técnicas de modelagem) (Spolon, 2001).

As técnicas de aferição são aplicadas principalmente na avaliação de sistemas já existentes ou em fase final de desenvolvimento. Como exemplo têm-se prototipação, *benchmarks* e coleta de dados (Balieiro, 2005).

A modelagem é geralmente utilizada quando se deseja avaliar um sistema ainda inexistente, trazendo como vantagem, a possibilidade de antever o desempenho de um sistema computacional. Para a avaliação dos modelos pode-se utilizar solução analítica ou solução através de simulação. O uso da simulação para efetuar a análise de sistemas computacionais vem crescendo e isso ocorre principalmente devido ao aumento da complexidade dos problemas (Sacchi, 2005).

Pode-se definir simulação como o processo de projetar um modelo computacional de um sistema real e conduzir experimentos com este modelo com o propósito de entender seu comportamento e/ou avaliar estratégias para sua operação (Pegden, 1990 *apud* Freitas, 2001).

O uso das técnicas de solução de modelos pode ser justificado por três razões fundamentais (Freitas, 2001): quando o sistema real ainda não existe e deseja-se conhecer o seu comportamento futuro; quando a experimentação com o sistema real é dispendiosa e a simulação pode ser uma alternativa mais realista do comportamento do sistema; quando efetuar experimentos com o sistema real não é apropriado ou mesmo quando isto pode requerer a destruição do próprio sistema.

O desenvolvimento de um programa de simulação envolve a construção de uma lista de eventos, de um relógio global que controla o processo de simulação e de variáveis que descrevem o estado do sistema em estudo, além da geração de números aleatórios que determinam, por exemplo, o tempo que um cliente leva para ser atendido. Esse projeto disponibiliza ao usuário da simulação esse conjunto de

estruturas e todas as rotinas necessárias para sua manipulação. Assim, o usuário não precisa implementar todas as construções necessárias para desenvolver um programa de simulação, mas sim utilizar as funções e estruturas definidas na biblioteca, em conjunto com uma linguagem de programação de uso comum. Com a biblioteca em linguagem C++, que é uma linguagem conhecida em meio acadêmico, o usuário pode utilizar as classes e seus métodos para o desenvolvimento de modelos de forma simplificada, não sendo necessário o aprendizado de uma nova linguagem nem o desenvolvimento de classes para fornecer suporte à simulação. Além disso, essa biblioteca poderá ser utilizada pelos alunos do departamento como ferramenta de aprendizado sobre simulação de modelos de redes de filas.

A análise dos resultados obtidos com a simulação também foi considerada, através da implementação de um método de análise de saída. A visualização desses resultados pode ser feita através de gráficos ou dos valores numéricos.

Também foi possível adquirir conhecimento sobre as técnicas de avaliação de desempenho, em particular simulação de sistemas discretos, orientada a evento, através da solução de modelos de filas. A análise de saída, efetuada através do uso de análises estatísticas, também foi motivo de aquisição de conhecimento, assim como as ferramentas utilizadas.

O capítulo 2 apresenta uma revisão da bibliografia utilizada no decorrer do projeto. O capítulo 3 apresenta os algoritmos e as estruturas criadas para o funcionamento da biblioteca. O capítulo 4 apresenta os e testes e a análise dos resultados obtidos, mostrando como pode ser utilizada a biblioteca. O capítulo 5 apresenta as conclusões, as contribuições e sugestões para trabalhos futuros a partir desse projeto.

Capítulo 2 – Revisão Bibliográfica

2.1 Considerações iniciais

As técnicas de avaliação de desempenho podem ser divididas em técnicas de aferição ou através das técnicas de solução de modelos (ou, simplesmente, técnicas de modelagem) (Spolon, 2001). A modelagem é geralmente utilizada quando se deseja avaliar um sistema ainda inexistente, uma das vantagens dessa técnica é a possibilidade de antever o resultado da execução do sistema (Sacchi, 2005).

Neste capítulo são discutidos os assuntos de interesse em todo o desenvolvimento do projeto. Na seção 2 será feita uma revisão das técnicas de avaliação de desempenho situando-se assim o tema. A seção 3 discute técnicas de aferição com ênfase na técnica de coleta de dados que pode ser utilizada em processos de simulação. Na seção 4 são discutidas as técnicas de modelagem, sendo a simulação uma delas. A seção 5 apresenta as fases da simulação. A seção 6 apresenta um estudo sobre Redes de Filas que é uma das formas utilizadas para representação de modelos. Na seção 7 encontra-se um exemplo da uma extensão funcional da linguagem C chamada SMPL (*SiMulation Programming Language*) (Macdougall, 1987) que servirá de base para esse projeto. Na seção 8 é apresentado um exemplo de simulação de um sistema simples de filas, que permite mostrar o funcionamento da simulação.

2.2 Técnicas de avaliação de desempenho

O processo de avaliação de desempenho de um sistema pode fornecer diversos resultados, como, por exemplo, a taxa de utilização de uma UCP (Unidade Central de Processamento) ou o tempo gasto para o consumo de determinado recurso (Silva, 2000).

Para a avaliação de desempenho de um sistema pode-se utilizar uma das técnicas que podem ser distribuídas em dois grupos: técnicas de aferição e técnicas de modelagem (Spolon, 2001).

As técnicas de aferição englobam prototipação, *benchmarks* e coleta de dados (Balieiro, 2005). Para a avaliação dos modelos pode-se utilizar solução analítica ou solução através de simulação (Santana *et al.*, 1994).

2.3 Técnicas de aferição

Técnicas de aferição são ferramentas utilizadas para analisar o desempenho de um sistema através da experimentação do mesmo (Santana *et al.*, 1994). Os resultados com essa abordagem fornecem resultados mais precisos do que as técnicas de modelagem, mas as técnicas de aferição apresentam a desvantagem de requererem o sistema finalizado ou em fase final de desenvolvimento. Pode-se encontrar outras dificuldades na aplicação desta técnica. Não permitir que a experimentação influencie no comportamento do sistema é um dos problemas encontrados. O melhor uso de cada uma das técnicas de aferição está representado na figura 2.1.

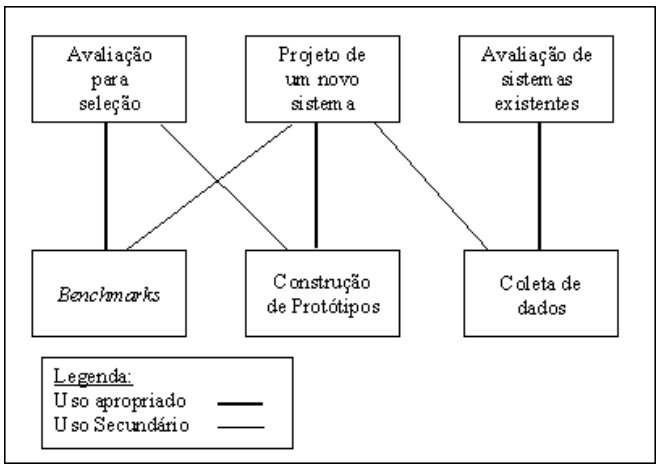


Figura 2.1 – Utilização das técnicas de aferição (Santana *et al.*, 1994).

2.3.1 Prototipação

A construção de protótipos corresponde a uma simplificação de um sistema, na qual são representadas somente as características essenciais do sistema. A determinação dessas características pode se tornar uma dificuldade e deve ser efetuada com cautela para que não ocorra perda na formulação do modelo (Orlandi, 1995).

Uma das motivações do uso desta técnica é que na construção de um protótipo o tempo e o custo podem ser inferiores em relação ao que é utilizado desses fatores no sistema real. Dessa forma, pode-se obter informações importantes sobre o comportamento sem a necessidade da construção do sistema real, economizando-se em tempo e custo para a avaliação do desempenho.

Protótipos apresentam flexibilidade nas alterações que eventualmente devem ser feitas sobre o sistema. A realização destas alterações sobre o sistema final pode-se tornar impraticável.

Apesar dessas vantagens em relação ao sistema real, a técnica de prototipação possui alto custo e pode demandar muito tempo se comparada com outras técnicas de avaliação de desempenho. Recomenda-se o desenvolvimento de protótipos como ferramenta de verificação do projeto final ou então para verificação de um sistema existente, situação em que poucas modificações no modelo são necessárias (Santana *et al.*, 1994).

2.3.2 *Benchmarks*

Benchmarks são programas utilizados para testar o desempenho de um software, hardware ou sistema computacional. A mesma tarefa ou programa é submetida a diferentes sistemas, sendo possível, dessa forma, a comparação dos resultados (Santana *et al.*, 1994).

Os resultados das realizações dos testes deste tipo de programa são dependentes da natureza do *benchmark*, por exemplo: o resultado de um *benchmark*

que envolve um grande número de cálculos em números de ponto flutuante provavelmente será diferente de um que faz vários acessos a arquivos.

Os *benchmark*s são indicados para a comparação de diferentes sistemas antes da aquisição de um novo. Uma das vantagens em utilizar essa técnica é que o software que faz a comparação pode ser encontrado no mercado e muitas vezes o seu uso é livre.

A execução de *benchmark*s deve ser cuidadosa, de modo que se evite que a execução do programa interfira nos resultados obtidos. Como exemplos de *benchmark* pode-se citar Linpack, usado para teste com algoritmos de álgebra linear, e SPEC, utilizado para testes em vários tipos de servidores (Weicker, 2002).

2.3.3 Coleta de dados

A Coleta de dados, entre todas as técnicas de avaliação de desempenho, é a que fornece o valor mais preciso das medidas. A aplicação dessa só é viável em sistemas que estão em operação. Recomenda-se seu uso quando o objetivo for unicamente a análise de desempenho de um sistema computacional (Santana *et al., 1994*).

Deve-se ter cuidado para que a coleta de dados não interfira no funcionamento do sistema. Se isso ocorrer, os resultados interpretados poderão sofrer influência do esforço gasto com a coleta dos dados. Outro uso típico desta técnica é na estimação dos parâmetros de entrada requisitados por modelos analíticos e/ou simulação (Santana *et al.*, 1994).

2.4 Técnicas de Modelagem

As técnicas de modelagem envolvem a solução de um modelo (uma abstração do sistema) através de solução analítica ou simulação. A especificação do modelo pode ser feita, por exemplo, através de Redes de Filas, Redes de Petri ou *Statecharts*. Nesse trabalho será abordado o estudo de Redes de Filas, apresentado na seção 6.

As técnicas de modelagem podem ser utilizadas tanto para sistemas que ainda estão em construção como para sistemas existentes.

O uso das técnicas de solução de modelos pode ser justificado por (Freitas, 2001):

- Quando o sistema real ainda não existe e deseja-se conhecer o seu comportamento futuro. Como exemplo, tem-se a modelagem e simulação do tráfego em uma rede de computadores (Wang & Wolff, 2003; Gu *et al.*, 2004);
- Quando a experimentação com o sistema real é dispendiosa e a simulação pode ser uma alternativa mais realista do comportamento do sistema, como por exemplo, na obtenção de novos equipamentos. Muitos estudos são feitos na área de automação com projetos em que a construção do *hardware* é irrealista (Kim & Kim, 2005);
- Quando efetuar experimentos com o sistema real não é apropriado ou mesmo quando isto pode requerer a destruição do próprio sistema. Como exemplo, pode-se citar o ataque de um *worm* a uma rede de computadores (Mirlovic *et al.*, 2005).

Um modelo pode ser resolvido por solução analítica ou por solução através de simulação. Existem vantagens e desvantagens no uso de cada uma delas. Não existe uma regra para escolha de qual técnica utilizar, mas sistemas complexos requerem o uso da simulação (Santana *et al.*, 1994).

A técnica de solução analítica consiste em definir o problema através de definições lógicas e matemáticas as quais podem ser resolvidas. Apesar dessa técnica fornecer resultados precisos, conforme a complexidade aumenta também aumenta a dificuldade da solução analítica. Recomenda-se essa técnica para situações nas quais o modelo matemático é conhecido. Uma desvantagem em relação a essa técnica é a dificuldade em fazer modificações no modelo, podendo ter alto custo e despender muito tempo (Santana *et al.*, 1994).

A técnica de solução através de simulação consiste em modelar o sistema para transformá-lo em um programa que represente com fidelidade o sistema real. É uma técnica prognóstica, que examina o comportamento do sistema através de várias execuções do modelo. Recomenda-se o uso desta técnica para sistemas que ainda não

foram implementados e apresentam grande complexidade. A desvantagem no uso desta técnica consiste no cuidado em se fazer a verificação e validação para que se possa ter confiança nos resultados obtidos (Santana *et al.*, 1994).

2.5 Simulação

Simulação implica na modelagem de um processo ou sistema, de tal forma que o modelo imite as respostas do sistema real através da execução do comportamento estudado (Freitas, 2001).

Um modelo computacional é um programa de computador cujas variáveis aproximam ao máximo do comportamento dinâmico e estocástico do sistema real (Freitas, 2001).

2.5.1 Fases da simulação

O estudo de desempenho através da simulação pode ser dividido em três fases: fase de desenvolvimento na qual são feitas a descrição do modelo, coleta de dados, seleção do método de análise e desenvolvimento do programa de simulação. Fase de testes, na qual são feitas a depuração, verificação e validação do sistema, e fase de análise, na qual é efetuada a análise de saída do programa. Estas fases estão mostradas na figura 2.2. Uma quarta fase pode ser inserida neste contexto (Banks *et al.*, 2004) , a fase de documentação, na qual são feitas as conclusões com os dados obtidos nas fases anteriores. Nesta fase pode ser constatada a necessidade de modificação nas fases anteriores o que pode provocar uma iteração no processo de desenvolvimento.

2.5.1.1 Fase de desenvolvimento

O primeiro passo é a descrição do modelo que consiste no entendimento do sistema. Esse sistema deve ser definido com precisão e as metas pretendidas devem ser claras e objetivas. O tempo gasto em organizar e escrever essa descrição é compensado devido aos possíveis erros eliminados.

O próximo passo é a abstração do sistema e descrição do modelo, que envolve a decisão sobre quais elementos do sistema devem ser incluídos no modelo. Nesse passo descreve-se o que deve ser representado no modelo e o que deve ser excluído.

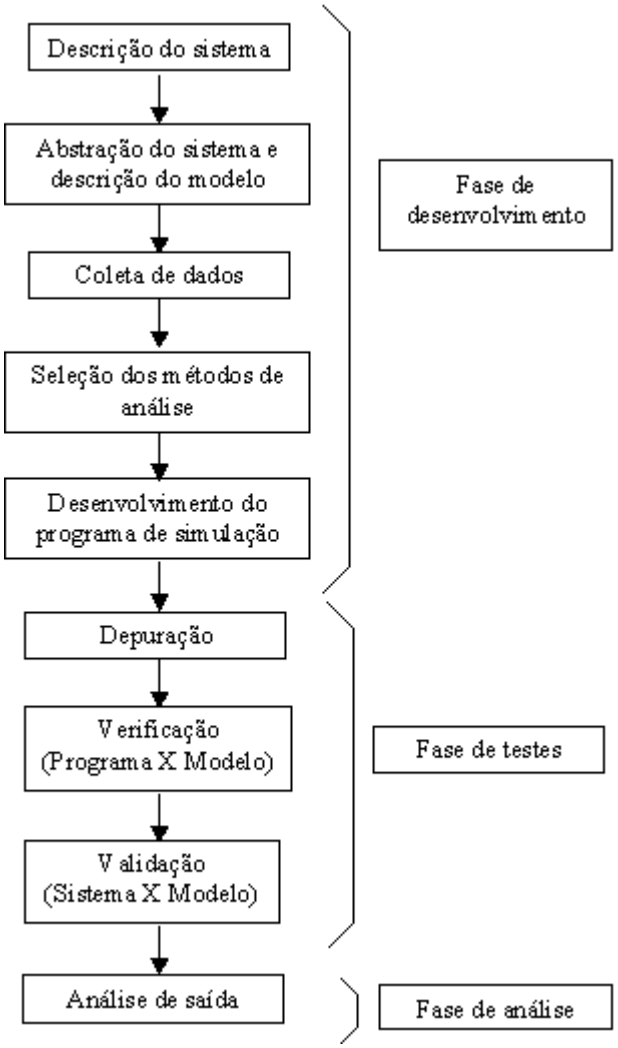


Figura 2.2 – Fases de desenvolvimento (Macdougall, 1987).

A coleta de dados é feita através dos requisitos obtidos nas fases anteriores. Se o sistema já existir, a coleta pode ser feita medindo diretamente do sistema. Na seleção do método de análise é escolhido o método analítico ou simulação.

O último passo desta fase é o desenvolvimento do programa de simulação. A seleção da linguagem e ferramentas depende do sistema sendo simulado e dos recursos disponíveis (Macdougall, 1987).

2.5.1.2 Fase de testes

Essa fase começa com a depuração do programa. Neste passo é verificado se o programa não apresenta erros de programação, lógica ou outros erros comuns de ocorrerem durante a programação.

O próximo passo é a verificação, que consiste em certificar de que o programa de simulação é uma implementação válida do modelo. Para modelos pequenos pode ser feita uma inspeção, para modelos maiores uma análise mais detalhada se torna necessária. A verificação pode ser feita com comparação a modelos analíticos.

A validação deve certificar que o comportamento do sistema está sendo realizado com fidelidade. Quando o sistema já existe, medidas do mesmo são comparadas com resultados de uma simulação do sistema e se os resultados coincidem, o programa de simulação é considerado válido. Quando o sistema não existe pode-se utilizar outro semelhante ou estimar os valores (Macdougall, 1987).

2.5.1.3 Fase de análise

Uma questão importante é quantas vezes executar um programa. Pode-se usar:

- Técnica de replicação, na qual um intervalo de confiança é conseguido através das várias execuções independentes do mesmo programa, conseguindo-se assim uma média das variáveis de saída. As execuções só param em um intervalo de tempo pré-determinado;

- Técnica de regeneração, na qual o programa volta a um ponto pré-determinado, diminuindo o processamento necessário;
- Técnica de média por divisões ou *batch means*. Ao contrário da técnica de replicação, este método é baseado em uma única execução do modelo na qual estima-se o intervalo de confiança. A seqüência original de observações é dividida em lotes que são intervalos de tempo menores que ocorrem durante a execução. As médias desses lotes (*batch means*) podem ser utilizadas para o cálculo do intervalo de confiança. Caso não se consiga chegar a um resultado preciso durante a primeira execução o segundo lote será executado e assim por diante até a sua finalização ou o alcance da precisão desejada (Macdougall, 1987).

2.5.2 Classificação da abordagem da simulação

Sistemas podem ser compostos por entidades. Essas entidades podem ser passivas ou ativas. Entidades passivas podem ter dois estados (ocupado ou desocupado) que são resultado de ações das atividades (Macdougall, 1987). A composição dinâmica de um sistema pode ser descrita por atividades, processos e eventos (Macdougall, 1987).

Uma atividade é a menor unidade de trabalho em um sistema. Em um programa simples pode-se definir como atividade cada função do programa, ou cada execução de uma linha de código. O nível de abstração que se deseja do sistema é o que define até que nível deve-se chegar na definição de atividade do sistema (Macdougall, 1987).

O conjunto de pequenas unidades que se relacionam, ou seja, de atividades relacionadas é chamado de processo. Num exemplo de um simples programa de computador, pode-se dizer que cada função do programa é uma atividade. A reunião do conjunto de funções que realizam entrada e saída tem como resultado o processo de entrada e saída deste programa (Macdougall, 1987).

A iniciação de atividades é disparada por eventos. Um evento é a mudança do estado de uma entidade de ativo para passivo, ou vice-versa. A execução do evento

ocorre no fim de uma atividade (Macdougall, 1987). O relacionamento entre eventos processo e atividades está apresentado na figura 2.3.

A classificação da abordagem da simulação pode ser dividida em modelos orientados a eventos e modelos orientados a processo (Soares, 1990).

Modelos orientados a eventos são preferíveis para serem usados com problemas de pequeno e médio porte. Um problema que pode ocorrer com modelos orientados a eventos é que há uma tendência a colocar ações que não estão relacionadas juntas em uma mesma rotina. Isso pode causar sérios prejuízos ao modelo. Um modo de evitar isso é estruturar cuidadosamente o modelo. Isso pode levar a uma visão de modelos orientados a processos (Soares, 1990).

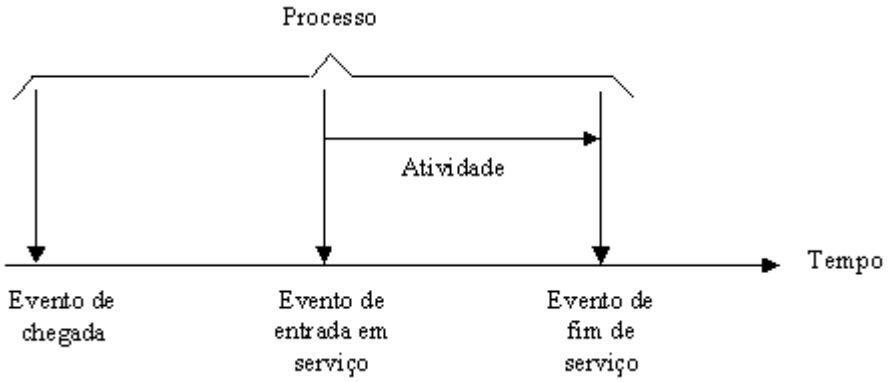


Figura 2.3 – Relacionamento entre evento, atividade e processo (Soares, 1990).

Modelos orientados a processos representam a maioria dos problemas de grande porte. Esses modelos são construídos com descrições simples das operações do sistema. Esse tipo de paradigma facilita a compreensão do modelo e a sua atualização. Isso faz com que o sistema seja construído com pequenas operações que se comunicam (Soares, 1990).

2.6 Redes de Filas

Fila é um conceito que está presente no cotidiano. Este conceito é o que está implícito de uma fila de banco, na qual existem alguns caixas bancários que atendem

as pessoas que chegam para realizar um pagamento, descontar um cheque ou outras operações bancárias. Quando os clientes chegam com uma taxa maior do que a taxa com que os caixas conseguem atender, forma-se uma fila de espera.

Uma rede de filas básica consiste de entidades chamadas centros de serviço e um conjunto de entidades chamadas usuários, que consomem recurso no centro de serviço. O centro de serviço é constituído por servidores e filas (áreas de espera) (Freitas, 2001).

Um exemplo básico de um centro de serviço é representado na figura 2.4.

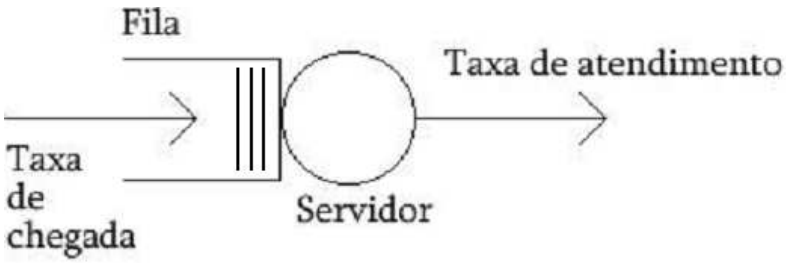


Figura 2.4 – Um centro de serviço simples (Soares, 1990).

Os usuários são as entidades que interagem com os centros de serviço. São essas entidades que serão atendidas nos servidores e também que preencherão as filas de espera. Os usuários podem ser representações de diversas entidades do mundo real, como por exemplo, pessoas, programas de computadores, mensagens de comunicação ou tarefas de uma manufatura (Freitas, 2001).

Outro sistema de filas pode conter várias filas de espera. Cada fila pode representar um tipo de usuário ao qual uma prioridade pode ser associada. Um exemplo é o algoritmo *round-robin* com múltiplas filas (Tanenbaum, 2003).

Um exemplo que apresenta estas variações é o de um roteador de rede de computadores (ou um comutador) em que vários datagramas chegam ao aparelho e são roteados para o endereço correspondente a seu destino (Kurose & Ross, 2003). O que acontece normalmente é que nem sempre a taxa de atendimento que se tem consegue ser superior a taxa de chegada o tempo todo o que provoca a formação de uma fila de datagramas. Se a comutação for feita com uma única fila e o roteamento for feito através de um barramento tem-se o exemplo mais simples de rede de fila em

que se tem apenas uma classe de pacotes e um único servidor. Essas variações são as modificações básicas em uma rede de filas e estão representadas na figura 2.5.

Usuários podem ter diferentes atributos que podem ser definidos de acordo com a necessidade de cada sistema. Esses atributos servem para tomadas de decisões durante o percurso do usuário através dos vários centros de serviço. Eles podem ser utilizados para uma decisão de roteamento ou de demanda de um serviço específico, por exemplo.

Isso leva a outra definição, que é a de classe de usuários. Como no caso do processador, pode-se ter várias classes de usuários (processos) sendo atendidas com uma taxa de serviço diferente. As classes de usuários também podem servir para alocar os usuários em diferentes filas e/ou diferentes servidores.

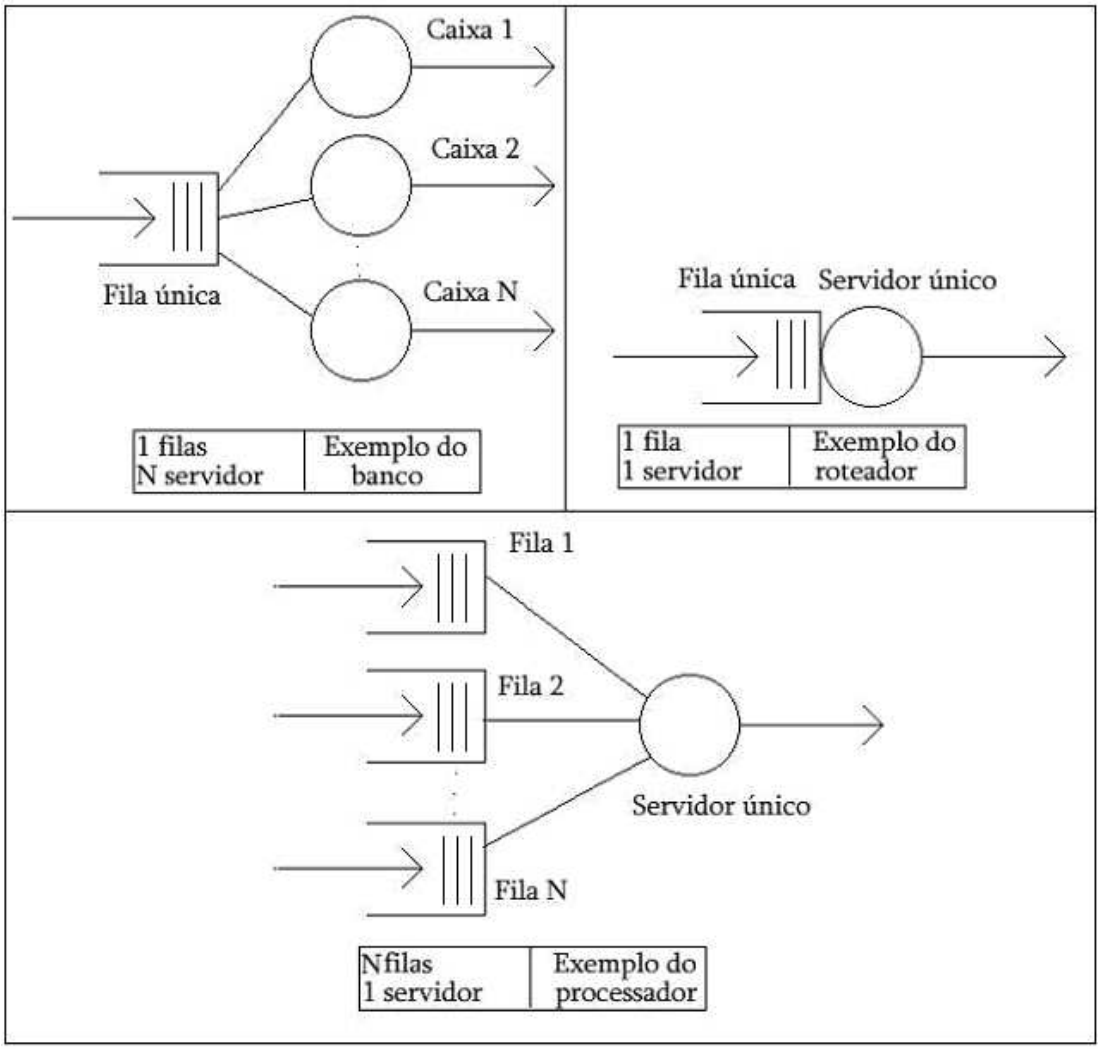


Figura 2.5 – Variações do modelo de redes de filas (Soares, 1990).

Quando existem usuários esperando para serem atendidos em uma ou mais filas e um servidor fica livre para atender outro usuário, é necessário que o centro de serviço escolha qual usuário será atendido. Neste caso é utilizado um algoritmo de escalonamento para escolher a ordem com que os usuários serão atendidos. Os algoritmos de escalonamento mais comuns são (Soares, 1990):

- FCFS (*First Come First Served*) – O primeiro a chegar será o primeiro a ser atendido;
- LCFS (*Last Come First Served*) – O último a chegar será o primeiro a ser atendido;
- RR (*Round Robin*) – Uma parcela de tempo no servidor é dada para cada usuário;
- PRTY (*nonpreemptive priority*) – Os usuários são atendidos segundo uma prioridade;
- PRTYPR (*preemptive-resume priority*) – Os usuários são atendidos segundo uma prioridade, mas podem ser retirados do servidor se um processo com prioridade maior chegar para ser atendido.

2.6.1 Notação Kendall

Para denotar as redes de filas utiliza-se a notação Kendall que define características do centro de serviço. Esta notação é definida pela quíntupla:

A/S/c/k/m

onde

- A representa a distribuição estatística da taxa de chegada;
- S representa a distribuição estatística da taxa de serviço;
- c representa o número de servidores no centro de serviço;
- k é o número máximo de usuários que pode estar na fila ao mesmo tempo e, ou seja a capacidade da fila;

- m é o número máximo de usuários na população.

Existe a possibilidade de se omitir dois componentes dessa quíntupla. Isso acontece quando se assume que a população do sistema é infinita e o espaço de alocação da fila também. Desta maneira tem-se a notação de modo simplificado,

A/S/c

Como exemplo pode-se considerar uma rede M/M/1. Esta notação indica que usuários chegam a um centro de serviço com uma taxa exponencial e são atendidos na mesma taxa por um único servidor.

As distribuições de probabilidades que são encontradas com mais freqüência (MacDougall, 1987) em estudos sobre o assunto são:

- D - que representa uma taxa de chegada ou taxa de serviço constante;
- M - distribuição exponencial;
- Ek - Distribuição k-Erlang;
- Hk – Distribuição hiperexponencial de k-estados.

2.7 Implementação dos Modelos de Simulação

A implementação de um modelo de simulação pode ser feita através do uso de linguagens de simulação, pacotes de uso específico, linguagens de programação convencionais ou extensões funcionais (Santana *et al.,* 1994).

As linguagens de simulação trazem a vantagem de que o programador não necessita criar todo o ambiente da simulação. Um dos problemas no uso desse enfoque é que o programador necessitará aprender uma linguagem nova. Como exemplos tem-se SIMSCRIPT (KO78, MA75 *apud* Santana *et al.*, 1994).

Pacotes de uso específico, como o próprio nome indica, são restritos para a avaliação de sistemas particulares. Isso cria um problema que é a falta de flexibilidade no uso deste tipo de ferramenta. Se o uso for restrito ao problema que a

ferramenta trata, esse enfoque é apropriado (Santana *et al.*, 1994). Como exemplos têm-se CMF, PET e VMPredictor (Santana *et al.*, 1994).

Para criar um programa em uma linguagem convencional como, por exemplo, Pascal, C, ou C++, o programador deve criar todo um ambiente necessário para a simulação, isto inclui estruturas de dados para servidores, usuários e filas, por exemplo. Uma das vantagens em se utilizar esse enfoque é que o usuário não precisa aprender uma nova linguagem, mas isto poderá requerer muito tempo, pois necessitará a criação de todo o ambiente.

Para a resolução deste problema têm-se extensões funcionais que são bibliotecas que quando inseridas em linguagens hospedeiras, como por exemplo, Pascal, C ou C++, compõem um ambiente completo de simulação. Como exemplo pode-se citar CSIM e SMPL (Santana *et al.*, 1994).

2.7.1 SMPL

SMPL é uma extensão funcional da linguagem que segue a orientação a eventos. Na extensão existem três tipos de entidades: centro de serviços, usuários e eventos (Macdougall, 1987).

Centro de serviço representa entidade que disponibiliza atendimento aos usuários do sistema. Usuários são entidades ativas do sistema. O comportamento do sistema é modelado pelo movimento dos usuários através dos vários centro de serviços que o modelo pode apresentar. Evento representa uma mudança de estado de qualquer entidade do sistema. Como exemplo, tem-se a mudança de estado do servidor de ocupado para desocupado.

O SMPL apresenta várias primitivas funcionais para o desenvolvimento de programas de simulação, das quais as principais são: *smpl*, *facility*, *request*, *release*, *schedule* e *cause* (Macdougall, 1994):

- *smpl*: é responsável por iniciar o sistema para a execução da simulação. Inicia as estruturas de dados e o relógio da simulação;
- *facility*: É responsável pela criação e nomeação dos centro de serviços do sistema. Para cada centro de serviço que é criado é retornado um

identificador, o qual é utilizado para qualquer tarefa a ser realizada pelo centro de serviço. Como exemplo de tarefas a serem realizadas pelo descritor pode-se citar a requisição e a liberação em servidores;

- *request*: requisita a utilização de um centro de serviço por um determinado usuário. Se todos os servidores do centro de serviço estão ocupados, o usuário solicitante é inserido na fila, caso contrário, um servidor livre é reservado para o usuário;
- *release*: libera um usuário de um servidor de um determinado centro de serviço. Se existir um usuário na fila ele é colocado no servidor, caso contrário é colocado como disponível;
- *schedule*: essa primitiva é responsável pelo escalonamento de eventos na lista de eventos futuros, a qual está em uma ordem crescente dos tempos de ocorrência dos eventos;

2.8 Exemplo de simulação de redes de filas

Como um exemplo do conceito de simulação de redes de filas, pode-se considerar a simulação manual do processo de atendimento a alunos por um professor na véspera de uma prova (Soares, 1990), no qual o professor atendendo o aluno é o servidor, os usuários são os alunos e a fila é a própria fila de alunos que se forma pela espera de atendimento. Ao chegar à sala do professor o aluno deve esperar em uma fila até que chegue a sua vez de ser atendido. Quando chega a sua vez, o aluno entra na sala, tira suas dúvidas e, após o tempo de atendimento, retira-se da sala do professor, podendo outro aluno ser atendido.

Com isso pretende-se obter o valor médio de espera na fila por um aluno e o tempo médio que o professor leva para tirar as dúvidas de um aluno. A tabela 2.1 representa os tempos que nos interessam para a modelagem deste sistema, na qual a coluna aluno representa um número que é a identificação do aluno. Tempo de chegada à sala, tempo de entrada em atendimento e tempo de saída da sala referem-se aos tempos, medidos em uma unidade de tempo (u.t.) qualquer, das ocorrências de eventos. Tempo de espera na fila e tempo total de atendimento (Tempo de

atendimento somado ao tempo de espera na fila) são medidas que serão utilizadas na análise dos dados de saída, essas colunas também estão em u.t.

Essa tabela é preenchida com valores gerados aleatoriamente. Como exemplo tem-se o aluno 2, o qual chega no tempo 5 entra em atendimento no tempo 8 e fica quatro unidades de tempo na fila.

Com os valores da tabela pode-se encontrar o tempo médio de espera na fila e o tempo médio do total de atendimento que são 2,9 u.t. em ambos os casos.

Nesse exemplo, os eventos que acontecem são chegada de um aluno, entrada em atendimento (o aluno entra na sala para falar com o professor) e saída da sala. Esses eventos podem ser colocados em uma lista na qual são ordenados em ordem cronológica. Essa lista é conhecida como lista de eventos futuros e contém em cada entrada, o usuário relacionado ao evento, o tempo em que ele ocorrerá e o tipo de evento que significa (chegada, atendimento ou saída).

Tabela 2.1 – Simulação de alunos sendo atendidos.

Aluno	Tempo de chegada à sala (u.t.)	Tempo de entrada em atendimento (u.t.)	Tempo de saída da sala (u.t.)	Tempo de espera na fila (u.t.)	Tempo total no sistema(u.t.)
1	3	3	8	0	5
2	5	8	12	3	7
3	6	12	15	6	9
4	8	15	17	7	9
5	12	17	20	5	8
6	18	20	23	2	5
7	20	23	25	3	5
8	21	25	28	4	7
9	24	28	30	4	6
10	25	30	32	5	7

Um exemplo do funcionamento da lista de eventos futuros será dado tendo como base a tabela 2.1, iniciando-se na chegada do evento 1 e com término na saída do mesmo aluno. No início da simulação a lista está vazia e ela começa a ser preenchida com a chegada do primeiro aluno. No exemplo dado, isto ocorre no instante de três u.t. (figura 2.6 (a)) ficando a lista com apenas um evento

inicialmente. Quando esse evento ocorre como o professor não está atendendo a nenhum aluno, o aluno referenciado como 1 é colocado imediatamente para consumir recursos do servidor, uma nova chegada é gerada (figura 2.6 (b)) e o relógio do sistema é atualizado para o valor três, sendo o evento de atendimento consumido um novo é colocado na lista (figura 2.6 (c)), o evento de saída do aluno 1. O próximo evento a ocorrer, de acordo com a lista, é o de chegada de um outro aluno (2) que deve ser colocado na fila, pois o servidor está ocupado, novamente o relógio é atualizado, uma nova chegada é gerada e um evento de atendimento também (figura 2.6 (d)). Sendo o evento de atendimento consumido (figura 2.6 (e)) deve-se consumir o próximo evento a chegada do aluno 3, sendo gerada uma nova chegada e o atendimento além da atualização do relógio (figura 2.6 (f)). Sendo assim, pode-se fazer execução do evento de saída do aluno 1, sempre atualizando o relógio (figura 2.6 (g)).

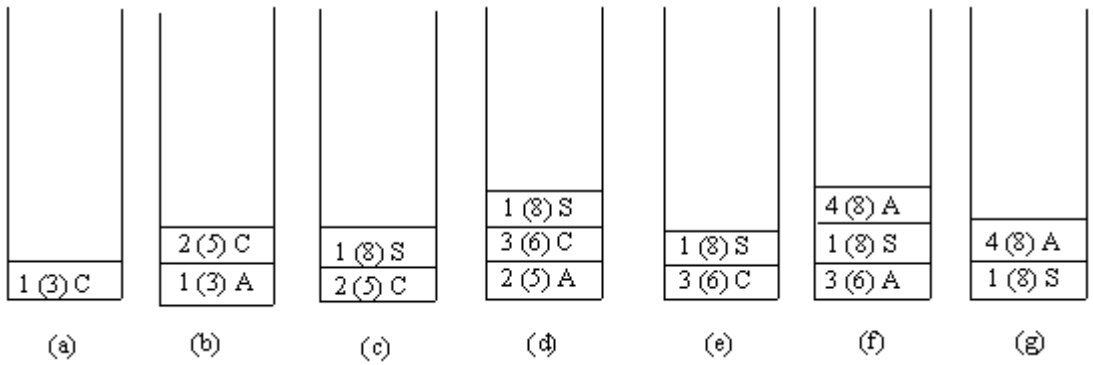


Figura 2.6 – Construção da lista de eventos futuros.

2.9 Considerações finais

O processo de avaliação de desempenho possui diversas técnicas que podem ser utilizadas. A técnica de resolução de modelos através de simulação é bem flexível e pode ser utilizada em sistemas existentes e inexistentes.

Este capítulo discutiu a avaliação de desempenho e em específico a técnica de modelagem com solução através de simulação, com o uso de redes de filas como técnica para especificar o modelo do sistema em estudo. No próximo capítulo será descrita a implementação da biblioteca.

Cap. 3 – Desenvolvimento da biblioteca RFOO

3.1 Considerações Iniciais

Este capítulo apresenta o desenvolvimento da biblioteca RFOO (Redes de Filas Orientada a Objetos). Essa biblioteca foi desenvolvida em C++ (Stroustrup, 2000) e com utilização de recursos do programa GNUPlot (Gnuplot, 2005). Essa extensão permitirá que o usuário utilize a simulação de redes de filas sem a necessidade de construir todas as estruturas e métodos necessários.

Estão descritos ao longo desse capítulo as classes criadas e os seus respectivos métodos. As classes implementadas dividem-se em pacotes que são chamados de Centro de Serviço, Estatística, Lista de Eventos Futuros e Relógio. Esses pacotes estão ilustrados na figura 3.1.

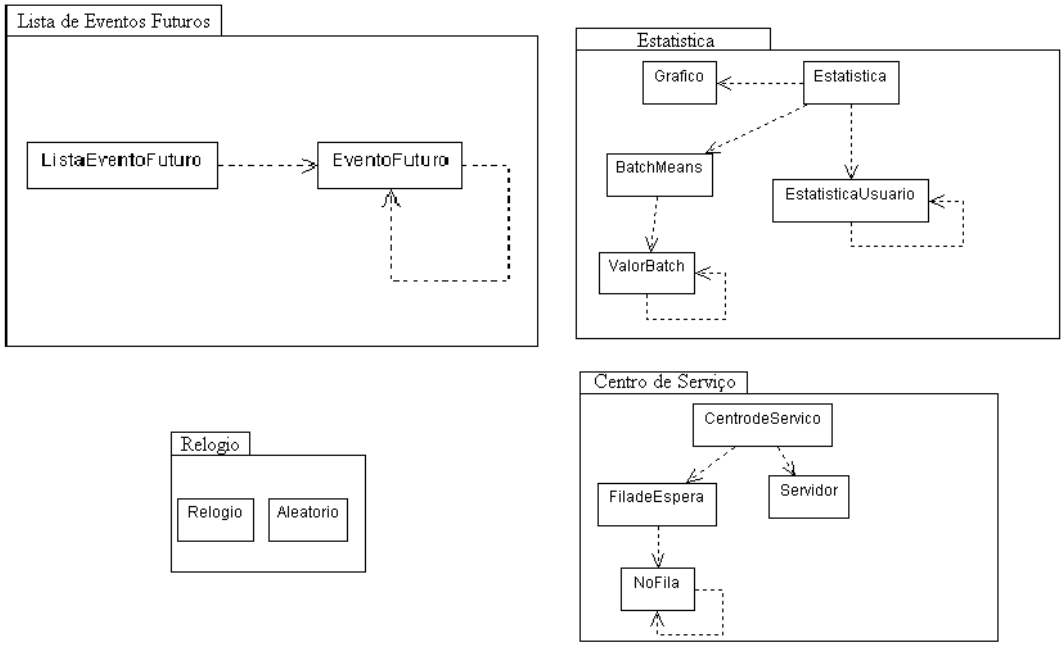


Figura 3.1 – Pacotes da biblioteca.

O pacote Centro de Serviço possui as classes CentrodeServico, Servidor, FiladeEspera e NoFila sendo que essas duas últimas classes citadas estão associadas. Ao usuário final está visível apenas a classe CentrodeServico.

O pacote Estatistica tem as classes principais Estatistica e EstatisticaUsuario. A classe EstatisticaUsuario é parte da classe Estatística e por esse motivo não é visível ao usuário final. Esse pacote também contém a classe Gráfico, que realiza o desenho de gráficos, e as classes BatchMeans e ValorBatch, que executam os cálculos necessários para a realização do *batch means*.

O pacote Lista de Eventos Futuros é formado pelas classes EventoFuturo e ListadeEventosFuturos, sendo que somente esta última é visível ao usuário.

O pacote Relogio possui a classe Relógio, que é responsável por controlar a passagem de tempo no sistema, e a classe Aleatório, que é responsável por gerar valores pseudoaleatórios , segundo distribuições de probabilidade. O diagrama de classes da biblioteca está representado na figura 3.2.

A seção 3.2 descreve o pacote Centro de Serviço, a seção 3.3 descreve o pacote Estatística, a seção 3.4 descreve o pacote Lista de Eventos Futuros, a seção 3.5 descreve o pacote Relógio, a seção 3.6 descreve a validação do sistema e a seção 3.7 apresenta as considerações finais.

3.2 O Pacote Centro de Serviço

Um centro de serviço é formado por filas e servidores. Essa é a base da classe CentrodeServico, que é formada pela classe Servidor e a classe FiladeEspera. FiladeEspera é formada por várias instancias da classe NoFila. A classe NoFila corresponde a instâncias de um vetor dinâmico cujos campos são um valor inteiro e um ponteiro para a próxima instância. Esse vetor obedece à estrutura de dados convencional de uma fila na qual o usuário adicionado é inserido no final da estrutura e o que é requisitado é retirado do início da estrutura. Essa estrutura de dados pode ser visualizada na figura 3.3.

A classe FiladeEspera contém um ponteiro para a primeira instancia de NoFila e este para o segundo a assim por diante, até o último que irá conter um ponteiro para NULL indicando que ele é o final da fila. A classe Servidor possui valores inteiros para indicar quantos servidores estarão livres. O detalhamento do pacote Centro de Serviço é mostrado na figura 3.4.

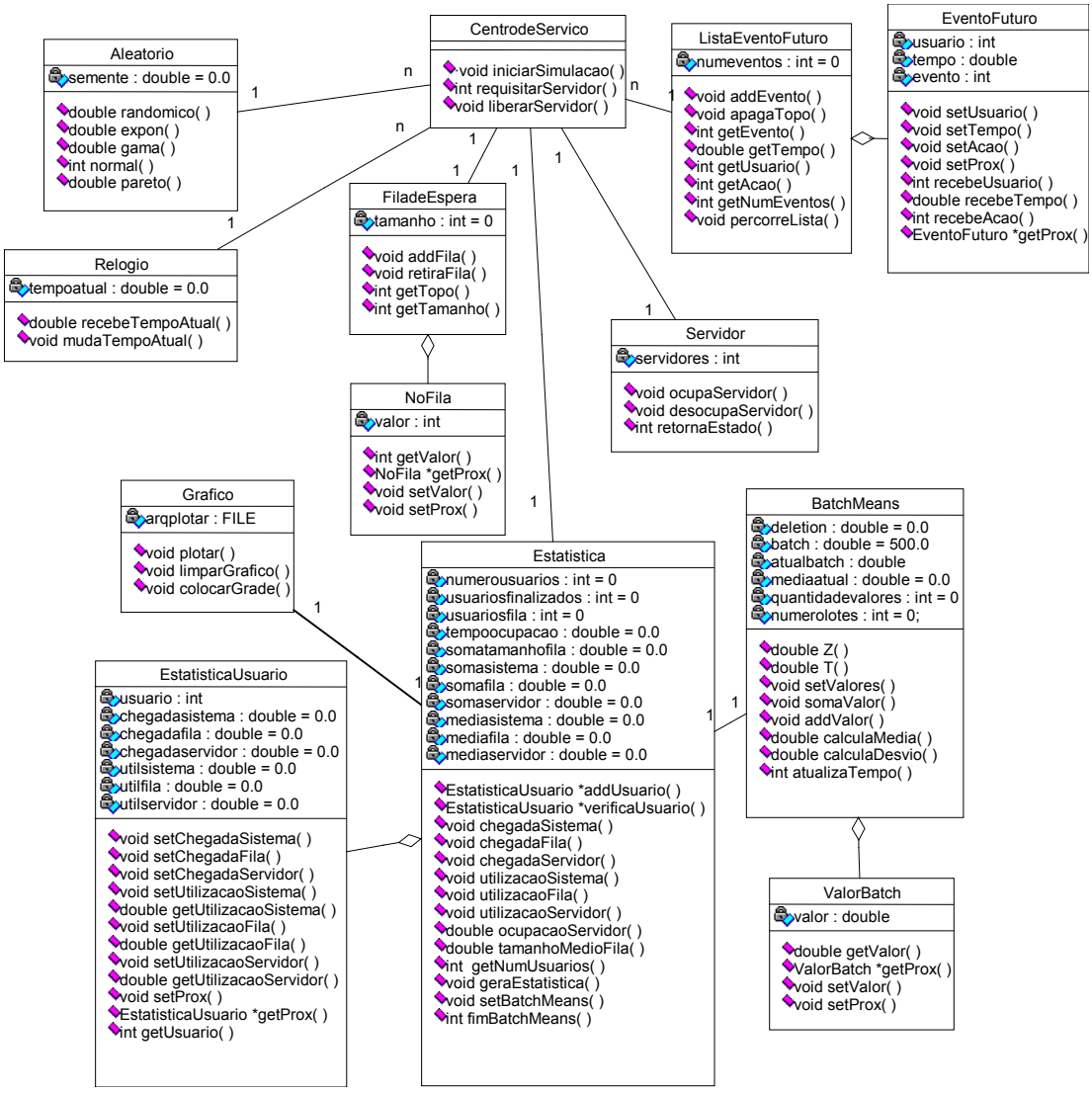


Figura 3.2 – Diagrama de classes da biblioteca RFOO.

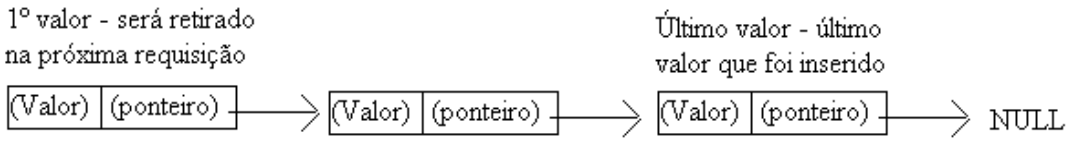


Figura 3.3 – Estrutura da fila na classe NoFila.

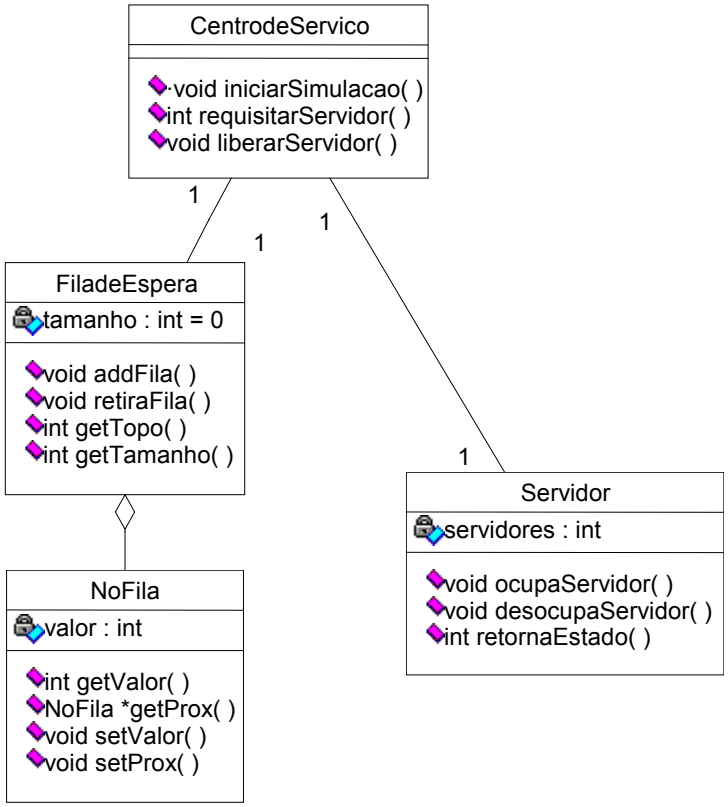


Figura 3.4 – O pacote CentrodeServico.

No construtor dessa classe o usuário deve indicar quantos servidores estarão disponíveis no Centro de Serviço criado. O construtor também instancia a classe FiladeEspera, que será utilizada durante o ciclo de vida de CentrodeServico. A classe CentrodeServico possui três métodos:

- void iniciarSimulacao(ListaEventoFuturo *lef, int usuario) - Esse método inicia a simulação gerando um primeiro evento que será colocado na LEF (lista de eventos futuros). Esse evento é gerado no instante zero;
- int requisitarServidor(Estatistica *estatistica, ListaEventoFuturo *lef) - Esse método faz a requisição de um servidor do centro de serviço em questão. Caso o usuário consiga acesso ao servidor é retornado o valor 1 (*True*). Caso contrário é retornado o valor 0 (*False*);
- void liberarServidor(ListaEventoFuturo *lef,Estatistica *estatistica) - Esse método faz a liberação de um servidor. Ele retira o usuário do servidor e se existir alguém na fila, gera uma nova entrada na lista de eventos

futuros, na qual é adicionada uma requisição ao servidor. O usuário que estava em espera vai entrar em atendimento.

As outras classes desse pacote não são visíveis ao usuário. Na classe Servidor encontram-se todos os atributos e métodos necessários para o funcionamento dos servidores no sistema simulado.

A classe Servidor apresenta como dado o número de servidores que estão disponíveis. Ao ser chamado, o construtor da classe recebe o número de servidores disponíveis. A classe apresenta os seguintes métodos:

- void ocupaServidor() – Esse método ocorre sempre que um usuário consegue acesso a um servidor;
- void desocupaServidor() – Esse método ocorre sempre que um usuário desocupa o servidor;
- int retornaEstado() – Esse método retorna o número de servidores disponíveis no centro de serviço.

A classe FiladeEspera representa a fila de usuários que estão aguardando recurso, ou seja, um servidor livre. Ela é formada por um valor inteiro que indica o tamanho da fila e um vetor dinâmico de tipos NoFila.

O construtor dessa classe recebe o construtor da fila que está sendo criada e o vetor NoFila que contém os usuários. Os métodos dessa classe são:

- void addFila(int usuario) – Esse método adiciona um usuário à fila de espera. Isso acontece quando não existe nenhum servidor disponível ao usuário que faz a requisição;
- int getTopo() – Esse método recebe o usuário que se encontra no início da fila, ou seja, o primeiro usuário a ser retirado da fila;
- int getTamanho() – Esse método recebe o tamanho da fila. Ele é útil para se obter conhecimento se um usuário está à espera de serviço. Caso seja retornado o valor zero, o centro de serviço pode atualizar o estado de mais um servidor, marcando-o como disponível.

NoFila é uma classe que é utilizada como um vetor dinâmico e possui os identificadores dos usuário que se encontram na fila e um ponteiro para o próximo usuário. Caso não existam mais usuários na fila, o valor NULL é passado para este ponteiro.

O construtor dessa classe recebe o valor do usuário e o ponteiro para o próximo usuário (que é nulo quando não existe um próximo usuário). Os métodos dessa classe são:

- int getValor() – Esse método recebe o valor do usuário que está vinculado a este nó da fila;
- NoFila *getProx() – Esse método recebe o ponteiro do usuário que se encontra no próximo nó da fila;
- void setValor(int v) – Este método altera o identificador do usuário que está neste nó;
- void setProx(NoFila *p) – Esse método altera o ponteiro para o próximo usuário da fila.

3.3 O Pacote Estatística

Para a construção deste pacote foi necessária a criação de cinco classes. A classe EstatisticaUsuario guarda os valores das estatísticas de cada usuário. A classe Estatistica calcula os valores das estatísticas do sistema considerando todos os usuários. A classe Grafico faz o desenho de gráficos utilizando o software GNUPlot e as classes BatchMeans e ValorBatch fazem os cálculos referentes ao *batch means*.

A classe Estatística faz os cálculos relativos à média do tempo no servidor e na fila, além do tamanho médio da fila e da taxa de ocupação do servidor. Essa classe também faz a contagem de quantos usuários passaram pelo centro de serviço e quantos passaram pela fila e possui métodos que apresentam esses dados ao usuário final. A classe guarda o valor de quantos usuários passaram pelo sistema, quantos saíram efetivamente do servidor e quantos usuários tiveram acesso à fila. Além disso, ela faz cálculos da porcentagem de ocupação do servidor, do tamanho médio da fila e média do tempo de permanência dos usuários no centro de serviço, no servidor e na fila. O diagrama que representa esse pacote está ilustrado na figura 3.5.

O construtor da classe Estatística inicia as variáveis que serão utilizadas nos cálculos das probabilidades. O construtor pode receber um tipo EstatisticaUsuario ou um tipo nulo. A classe apresenta os seguintes métodos:

- EstatisticaUsuario *addUsuario(int usuário) - Método de inserção na lista de estatísticas dos usuários. Sempre que um usuário chega ao sistema ele é realizado. Esse método retorna um EstatisticaUsuario porque quando ocorre uma chegada de um usuário ao sistema ele é adicionado e o sistema precisa de um ponteiro para esse usuário para fazer as alterações corretas;

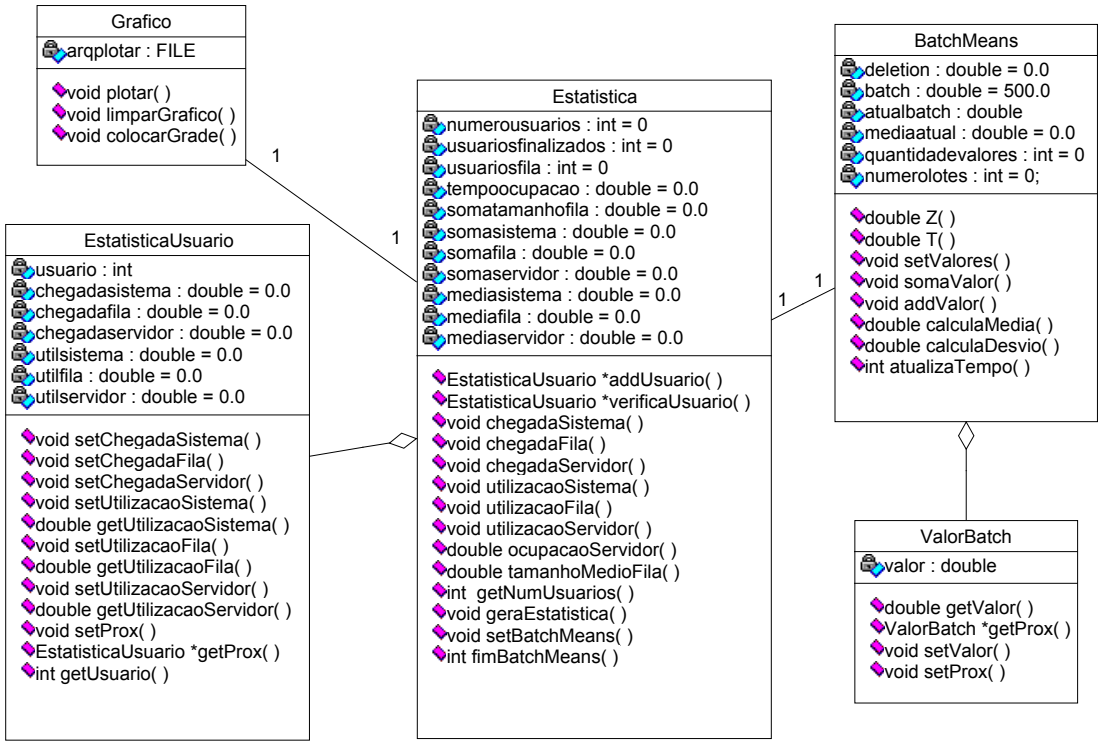


Figura 3.5 – O pacote Estatística.

- EstatisticaUsuario *verificaUsuario(int usuario) – Esse método verifica a existência de um usuário. Retorna um ponteiro para a estatística desse usuário se ele existir e NULL se não existir;
- void chegadaSistema(double chegada,int usuario) – Esse método é realizado quando um usuário chega ao sistema. Nesse método é criada uma nova instância de EstatisticaUsuario, que estará ligada ao usuário que acabou de chegar. É também marcado nessa nova instância o instante de tempo em que o usuário chega ao sistema;
- void chegadaFila(double chegada, int usuario,FiladeEspera *Fila) – Nesse método é marcado o instante de tempo em que o usuário entra na fila.

Nesse método também é feito o cálculo do tamanho médio da fila e incrementado o contador de números de usuários que entram na fila;

- void chegadaServidor(double chegada, int usuario) – Esse método é responsável pela atualização do tempo de chegada relativo a instancia da variável usuário;
- void utilizacaoSistema(double saida, int usuario) – Nesse método é efetuado o cálculo sobre o tempo que o usuário utilizou o sistema. Esse cálculo é feito quando o usuário sai do centro de serviço. As variáveis que guardam a soma do tempo de permanência no centro de serviço, no servidor e na fila são atualizadas. Também é feita a atualização da variável que conta quantos usuário saíram do sistema;
- void utilizacaoFila(double saida, int usuario, FiladeEspera *Fila) – Neste método é feito o cálculo do tempo que o usuário ficou na fila. Este método ocorre quando o usuário sai da fila;
- void utilizacaoServidor(double saida, int usuario) – Este método é responsável pelo cálculo de quanto tempo o usuário ficou no servidor. Este método ocorre quando o usuário sai do servidor. Neste método também é feita a chamada do método que adiciona o valor de ocupação do servidor ao lote atual da classe BatchMeans;
- double ocupacaoServidor() – Este método ocorre quando o usuário pede para as informações estatísticas serem exibidas. Ele calcula a taxa de ocupação do servidor;
- double tamanhoMedioFila() - Este método ocorre quando o usuário pede para as informações estatísticas serem exibidas, pois calcula o tamanho médio da fila de espera;
- int getNumUsuarios() – Este método retorna o número de instâncias de EstatisticaUsuario existentes;
- void geraEstatistica() – Esse método apresenta na tela, na forma de texto, as informações estatísticas a respeito do sistema: número de usuários no sistema, números de usuários na fila, média de permanência no centro de serviço, média do tempo de espera na fila, média de permanência no servidor, ocupação do servidor e tamanho

médio da fila. O resultado apresentado na tela está ilustrado na figura 3.6.

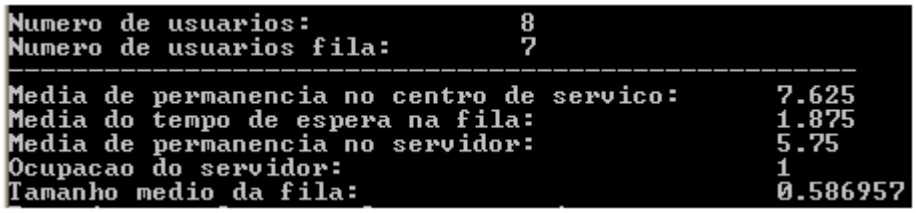


Figura 3.6 – Apresentação do método Estatistica::geraEstatistica().

- void setBatchMeans(double del, double bat) – Esse método altera os valores estatísticos que serão desconsiderados no início da simulação e o valor do lote do *batch means*. Como o usuário da biblioteca não tem acesso à classe BatchMeans esse é o modo dele definir esses valores;
- int fimBatchMeans() – Esse método é responsável pelo critério de parada do *batch means*. Nesse método é feita uma chamada à função que atualiza o tempo do *batch means* e retorna o valor lógico que indica se a simulação deve continuar (verdadeiro) ou parar (falso).

A classe EstatisticaUsuario guarda o tempo de chegada de cada usuário no sistema, na fila e no servidor, assim como o tempo que cada usuário ficou em cada uma das entidades. Além disso, também possui um ponteiro para o próximo usuário. O funcionamento dessa classe é como uma instância de um vetor cujo primeiro elemento está presente na classe Estatistica. Quando um novo usuário é adicionado ele é inserido ao final e a leitura é realizada com todos os usuários do vetor para os cálculos estatísticos.

O construtor da classe recebe o identificador do usuário e inicia todas as variáveis. Os métodos dessa classe são:

- void setChegadaSistema(double chegada) – Atualiza o instante de tempo de chegada ao sistema deste usuário. Esse método é chamado dentro de Estatística:: void chegadaSistema(double chegada, int usuario);

- void setChegadaFila(double chegada) – Atualiza o instante de tempo de chegada à fila deste usuário. Esse método é chamado dentro de Estatística:: void chegadaFila(double chegada, int usuario);
- void setChegadaServidor(double chegada) - – Atualiza o instante de tempo de chegada ao servidor deste usuário. Esse método é chamado dentro de Estatística:: void chegadaServidor(double chegada, int usuario);
- void setUtilizacaoSistema(double saida) – Calcula e atualiza o tempo de permanência deste usuário no centro de serviço. Esse método é chamado dentro de Estatística:: void utilizacaoSistema(double saida, int usuario);
- double getUtilizacaoSistema() – Recebe o tempo de permanência no centro de serviço deste usuário. Também é chamado dentro de Estatística:: void utilizacaoSistema(double saida, int usuario);
- void setUtilizacaoFila(double saida) – Calcula e atualiza o tempo de permanência deste usuário na fila. Esse método é chamado dentro de Estatística::void utilizacaoFila(double saida, int usuario, FiladeEspera *Fila);
- double getUtilizacaoFila() – Recebe o tempo de permanência na fila deste usuário. Também é chamado dentro de Estatística::void utilizacaoFila(double saida, int usuario, FiladeEspera *Fila);
- void setUtilizacaoServidor(double saida) – Calcula e atualiza o tempo de permanência deste usuário no servidor. Esse método é chamado dentro de Estatística:: void utilizacaoServidor(double saida, int usuario);
- double getUtilizacaoServidor() – Recebe o tempo de permanência no servidor deste usuário. Esse método também é chamado dentro de Estatística:: void utilizacaoServidor(double saida, int usuario);
- void setProx(EstatisticaUsuario *p) – Atualiza o ponteiro para a próxima instância de EstatisticaUsuario;
- EstatisticaUsuario *getProx() – Recebe o ponteiro para a próxima instância de EstatisticaUsuario;
- int getUsuario() – Recebe o identificador do usuário desta instancia.

A classe Grafico realiza o desenho de gráficos na tela do usuário da biblioteca. Esse procedimento é realizado com o auxílio do programa Gnuplot e de um software

para visualização de imagens. Nesse trabalho foi utilizado o EOG (*Eye of Gnome*) (Gnome, 2005). A classe Grafico possui como atributo um tipo FILE que é o arquivo no qual serão escritos os comandos para o programa Gnuplot. Os métodos da classe são:

- void plotar() – Desenha o gráfico na tela, o usuário deve realizar a chamada a este procedimento para obter o gráfico dos valores relativos a média dos lotes do *batch means*. A visualização de um exemplo de um desses gráficos é dada na figura 3.7;

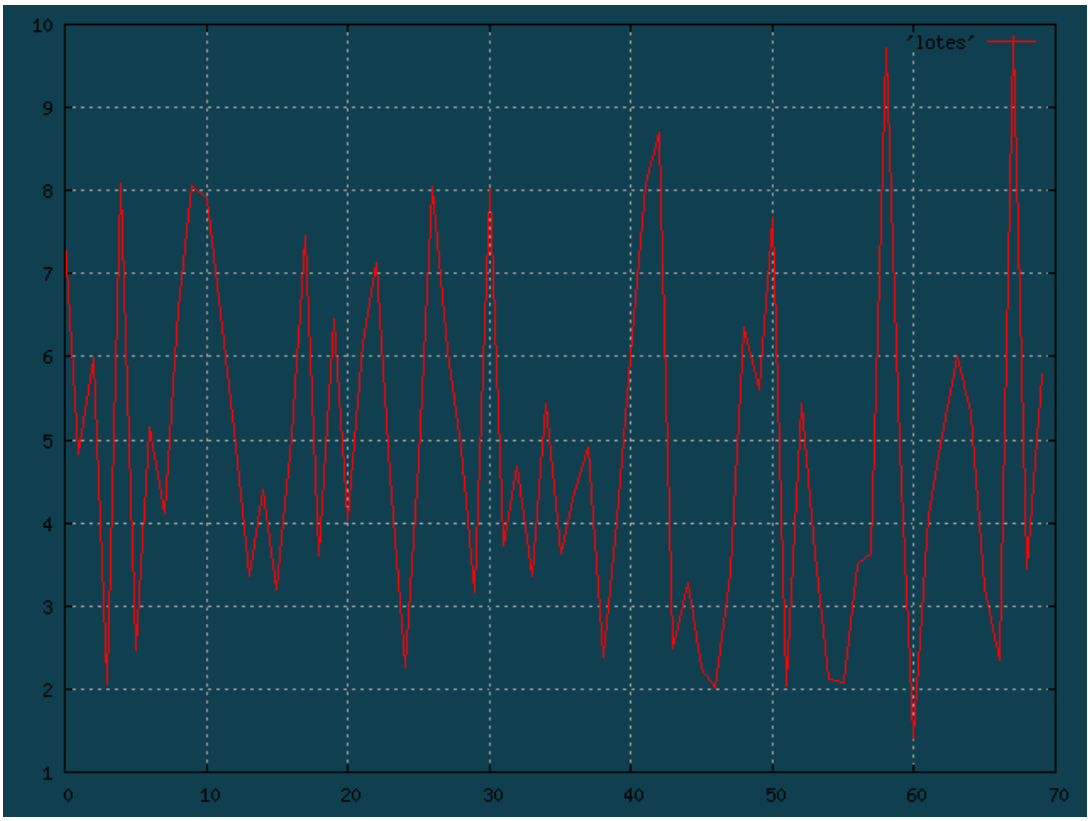


Figura 3.7 – Apresentação do gráfico com os valores dos lotes do *batch means*.

- void limparGrafico() – Limpa a figura que conterá o gráfico;
- void colocarGrade() – Coloca uma grade no gráfico para facilitar a visualização.

A classe BatchMeans é responsável pelos cálculos necessários para se fazer a execução do método de análise de saída *batch means*. Esse método consiste na divisão do programa em lotes e na execução desses lotes até que se consiga um intervalo de confiança com a precisão desejada. Nessa classe existe a variável

deletion do tipo *double*, que guarda informação sobre quanto tempo será esperado para que o sistema entre em equilíbrio e o *batch means* possa ser iniciado.

As variáveis batch, atualbatch, guardam, respectivamente, informação sobre o tamanho do lote e quanto falta para o lote atual finalizar. Quando atualbatch chega ao fim, são feitos testes para verificar se a simulação alcançou a precisão desejada. Caso a resposta for positiva, o sistema pára, caso contrário outro lote é executado.

Mediaatual e quantidadevalores são variáveis que guardam informações sobre os valores da variável e fazem o cálculo da média do lote atual quando esse finaliza. Numerolotes é um contador do número de lotes executados. Essa variável é útil porque o sistema só começa a fazer a verificação do intervalo de confiança após a execução de dez lotes. Além disso, ela é utilizada em alguns cálculos do intervalo de confiança.

O método *batch means* (divisões por lotes) consiste em dividir uma execução que se apresenta muito longa em conjuntos de k execuções de tamanho m, que são chamadas de *batchs* ou lotes. Calcula-se a média desses lotes separadamente e utiliza-se a média desses lotes para se calcular a média geral e o intervalo de confiança do sistema. Nesse trabalho utilizou-se a média de ocupação do servidor para o cálculo do *batch means*.

A classe BatchMeans apresenta os seguintes métodos:

- double Z (double p) & double T (double p, int ndf) – Esses métodos foram retirados da biblioteca SMPL (SiMulation Programming Language) (Fishman 1978 *apud* Macdougall, 1987) e são utilizados na obtenção do intervalo de confiança que deve ser determinado segundo certos valores estatísticos pré-estabelecidos que estão contidos nesses métodos.
- void setValores(double del, double bat) – Esse método é utilizado para a mudança do intervalo de tempo que será esperado até que o sistema entre em equilíbrio (del) e o tamanho do lote (bat). Os valores padrões para essas variáveis são 0.0 para deletion e 500.0 para batch.
- void somaValor (double valor) – Esse método faz a soma dos valores da variável que está em estudo no *batch means* para que seja possível o cálculo da média do lote.

- double calculaMedia ( ) & double calculaDesvio ( ) – Esses métodos calculam respectivamente a média e o desvio-padrão das médias dos lotes. Esses valores são utilizados nos cálculos dos intervalos de confiança.
- int atualizaTempo (double recebido) – Essa função é chamada toda vez que um teste para o certificar se o *batch means* atingiu a precisão é realizado. Ela recebe o tempo atual e atualiza a variável deletion até que ela chegue em zero. Quando isso ocorrer ela começa a contar o tempo do lote que ao chegar ao fim realiza o cálculo da média desse lote e guarda-o. Quando já foram executados dez lotes ou mais o sistema realiza um teste para certificar-se que a média geral atingiu a precisão. Se isto é verdadeiro é retornado o valor zero para que o sistema pare, em qualquer outro caso é retornado o valor um.

A classe ValorBatch é uma instância de um vetor dinâmico que guarda as informações sobre as médias dos lotes que foram executados. Cada instância dessa classe possui o valor da média do lote e um ponteiro para a próxima instância, assim como métodos para a modificação desses valores e a sua obtenção. O modo de funcionamento desse vetor é semelhante ao da classe EstatisticaUsuario na qual um usuário é inserido ao final do vetor. Sua leitura, quando realizada, ocorre em todas as instâncias do vetor para cálculos do *batch means*. O detalhamento dessa classe está representado na figura 3.14.

3.4 O Pacote Lista de Eventos Futuros

A lista de eventos futuros (LEF) é uma estrutura em que cada um de seus nós contém um identificador para um usuário, a ação que esse usuário irá executar e o instante em que essa ação ocorre. Essa lista é ordenada de forma crescente, durante a entrada dos eventos na lista e em relação ao tempo de ocorrência dos eventos. Essa lista indica qual evento será executado e na ocorrência desse evento, ele é retirado da lista e o relógio é atualizado.

A LEF foi implementada com a classe EventoFuturo como uma lista ordenada cujo primeiro elemento está presente na classe ListaEventoFuturo. As instâncias de

EventoFuturo são ordenadas durante a inserção na lista e é retirado sempre o primeiro elemento da lista, possibilitando assim que os eventos ocorram de forma crescente ao instante de ocorrência em relação ao relógio do sistema.

Esse pacote contém duas classes: ListadeEventoFuturo, que possui os métodos para controle da lista e são acessíveis para o usuário final, e EventoFuturo, classe que contém as informações de um nó dessa lista. Seus métodos não são acessíveis para o usuário da biblioteca.

A classe ListadeEventoFuturo possui uma variável que indica quantos nós estão presentes na lista e um vetor dinâmico de eventos que formará a lista. O detalhamento desse pacote está representado na figura 3.8.

Os métodos da classe ListadeEventoFuturo são:

- void addEvento(int u, double t, int a) - Este método fará a inserção de eventos na lista de eventos futuros. Ele recebe como parâmetros o usuário a ser inserido(u), o instante do tempo de ocorrência do evento (t) e o evento que irá ocorrer(a). Essa inserção é feita de forma ordenada de forma que no momento em que os eventos forem retirados da lista eles sejam retirados de forma cronológica. Nesse método é feita a atualização do número de eventos na lista;

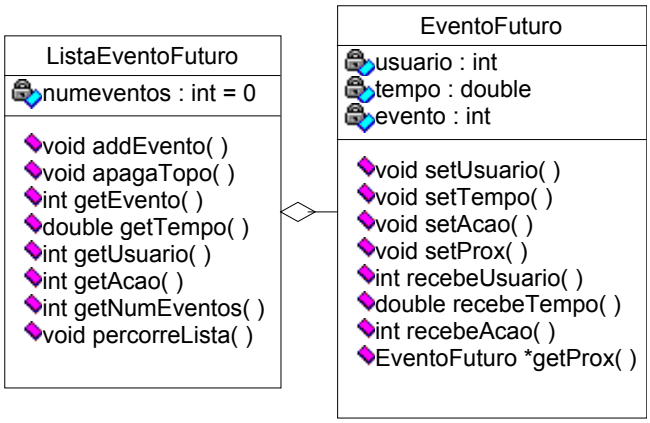


Figura 3.8 – O pacote ListadeEventoFuturo.

- void apagaTopo(relogio Relogio) - Este método, além de apagar o topo da lista também atualiza o relógio fazendo o acerto no tempo da simulação e diminui o contador de nós da lista. O usuário da biblioteca não precisa realizar a chamada a esse método, pois ele é chamado dentro do método getEvento();

- int getEvento(relogio *Relogio) - Este método recebe o evento do topo da lista e realiza a sua retirada, atualizando o relógio de acordo com o tempo recebido pelo evento. Ele retorna ao usuário o evento que estava no tipo EventoFuturo que foi retirado da lista;
- double getTempo() – Retorna o instante de tempo do evento que está no topo da lista;
- int getUsuario() – Retorna o identificador do usuário que está o topo da lista;
- int getAcao() – Retorna o evento que está no topo da lista. Diferentemente de getEvento(), este método não apaga o topo da lista;
- int getNumEventos() – Retorna o número de eventos contidos na lista;
- void percorreLista() - Esse método existe apenas para apresentar a LEF ao usuário. Ele não tem utilidade durante a simulação, mas pode ser utilizado para depuração do programa sem provocar alterações na LEF. A apresentação desse método está na figura 3.9.

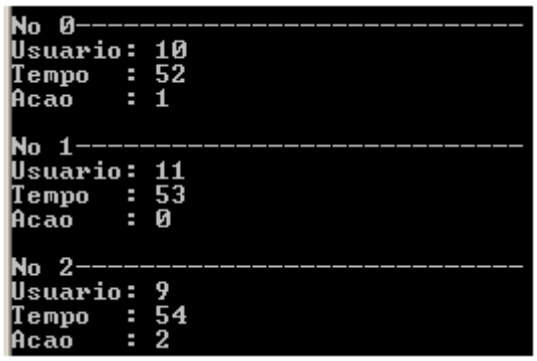


Figura 3.9 – Apresentação de ListadeEventosFuturos::percorreLista().

A classe EventoFuturo representa um nó da LEF e possui como atributos da classe o identificador do usuário, o tempo em que o evento ocorrerá e a ação (o próprio evento) que ocorrerá. Possui também um ponteiro para o próximo evento da lista que será nulo caso não exista um outro evento. O construtor dessa classe recebe esses quatro valores mencionados durante a instanciação da classe. Essa classe possui apenas métodos de entrada e de saída de dados que serão utilizados pela classe ListadeEventosFuturos para ordenar, incluir, apagar e alterar os eventos da lista.

3.5 O Pacote Relógio

O pacote relógio, além de possuir a estrutura Relógio, que é responsável por controlar a passagem de tempo no relógio virtual do sistema, possui também a classe Aleatorio que faz a geração de números aleatórios os quais serão os acréscimos nos tempo de simulação que serão utilizados para calcular o tempo de chegada e de atendimento.

A classe Relogio possui uma variável do tipo *double* que é o tempo atual do relógio do sistema. Além disso, possui métodos para que esse tempo seja alterado e consultado, que são mudaTempoAtual() e recebeTempoAtual(), respectivamente. O detalhamento dessa classe está representado na figura 3.10.

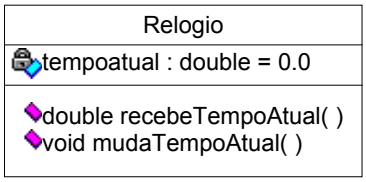


Figura 3.10 – A classe Relógio.

A classe Aleatorio é responsável por gerar números aleatórios seguindo distribuições de probabilidades conhecidas. Essa classe possui uma variável, a semente da geração de números, a qual é passada para a classe através do construtor.

Uma das maneiras de gerar números pseudoaleatórios (Lewis *et al.*, 1969 *apud* Macdougall, 1987) é através da fórmula,

$$I_{n+1}=\alpha I_n \bmod M$$

na qual M, sendo primo, é o período do gerador, ou seja, o número de valores gerados antes da seqüência começar a repetir, que será igual a M-1 e o valor para I_0 será a semente do gerador (método conhecido como *multiplicative congruential method*) (Banks et al., 2004) . A figura 3.11 apresenta o detalhamento da classe.

Essa classe possui alguns métodos adaptados do trabalho de geração de distribuições de probabilidades desenvolvido pelo discente Geraldo F. D. Zafalon (Zafalon & Manacero, 2004) e do trabalho de dissertação de mestrado de Rodrigo P. S. Sacchi (Sacchi, 2005). A classe Aleatorio possui os seguintes métodos:

- double randomico() – Esse método faz a geração de números aleatórios entre zero e um, conforme foi descrito anteriormente. Desse modo pode-se obter as probabilidades que serão utilizadas na geração de distribuições. Ele é adaptado do trabalho de Rodrigo P. S. Sacchi (Sacchi, 2005 *apud* MacDougall, 1987);

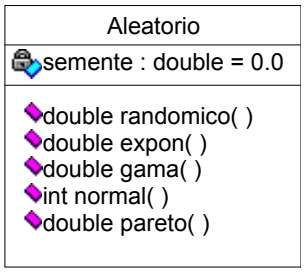


Figura 3.11 – A classe Aleatorio.

- double Expntl(float media) – Esse método gera um número aleatoriamente segundo uma distribuição exponencial. Ao método é passada a média da distribuição exponencial (Macdougall, 1987);
- double gama(int app) – Esse método gera um número segundo uma distribuição gama. É passado ao método o parâmetro app para que seja encontrado o valor esperado (Zafalon & Manacero, 2004);
- int Normal(int medianormal,int desvio) – Esse método faz a geração de um número segundo uma distribuição de probabilidade do tipo normal com media medianormal e desvio-padrão desvio (Macdougall, 1987);
- double pareto(double alfa, double beta) – Esse método retorna um número segundo a distribuição pareto com parâmetros alfa e beta (Zafalon & Manacero, 2004).
- Doublé Uniform(double a, double b) – Esse método um número segundo a distribuição uniforme com parâmetros a e b.
- double Hyperx(double x, double s) – Esse método gera um número segundo a distribuição hiperexponencial com parâmetros x e s.

3.6 Validação do sistema

A validação da biblioteca foi efetuada através da comparação do resultado de sua execução com o resultado da execução do SMPL. O modelo escolhido foi o M/M/1 (Macdougall, 1987) (figura 3.12).

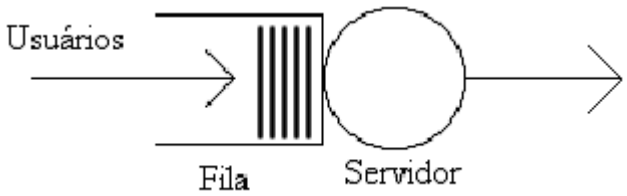


Figura 3.12 – O modelo M/M/1.

Nesse sistema a taxa de chegada de usuários e de atendimento do servidor foi gerada por uma distribuição exponencial com média de 10 u.t. e 20 u.t respectivamente. Como as execuções dos programas desenvolvidos com essas bibliotecas possuem valores do tempo de atendimento do servidor e de chegada de usuários aleatórios, foi utilizado o mesmo algoritmo para geração de números aleatórios nas duas implementações e com a mesma semente de geração de números aleatórios. Os resultados obtidos com as execuções dos programas podem ser visualizados na figura 3.13.

Percebe-se que os resultados obtidos com a execução do programa desenvolvido com a biblioteca RFOO (figura 3.21 (a)) são idênticos aos resultados obtidos com a execução do programa desenvolvido com a biblioteca SMPL (figura 3.21 (b)). Como exemplo pode-se citar o tempo médio de permanência no servidor que é de 9,91835 u.t no programa desenvolvido com a biblioteca RFOO e no programa desenvolvido com a biblioteca SMPL.

Observa-se que, além das informações do relatório gerado pelo SMPL, a biblioteca RFOO também permite ao usuário da simulação visualizar a média de tempo de espera na fila.

```
Numero de usuarios:             10031
Numero de usuarios fila:        4963
-----------------------------------------------------
Media de permanencia no centro de servico:      19.6542
Media do tempo de espera na fila:               9.73584
Media de permanencia no servidor:               9.91835
Ocupacao do servidor:                           0.497409
Tamanho medio da fila:                          0.488372
r3d12:/windows/faculdade/pf/exemplos# ./mm1q2
```

(a)

```
                          smpl SIMULATION REPORT


MODEL: M/M/1 Queue                                        TIME: 200018.40163
                                                       INTERVAL: 200018.40163

                            MEAN BUSY      MEAN QUEUE        OPERATION COUNTS
FACILITY          UTIL.      PERIOD         LENGTH      RELEASE   PREEMPT   QUEUE
server          0.497409     9.91835       0.488372       10031         0    4963
-------------------------------------------------------------------------------
REAL TIME: 21765 u-second(s)
```

(b)

Figura 3.13 – Resultados das execuções da validação

Outro modelo utilizado para a validação do sistema é o que está representado na figura 3.14. Esse sistema foi adaptado do modelo de uma UCP e quatro discos que está descrito no livro *Simulation Computer Systems: Techniques & Tools* (MacDougall, 1987). O número de discos foi alterado para facilitar a apresentação da saída do programa. Um estudo mais detalhado desse modelo foi realizado e está apresentado no capítulo 4.

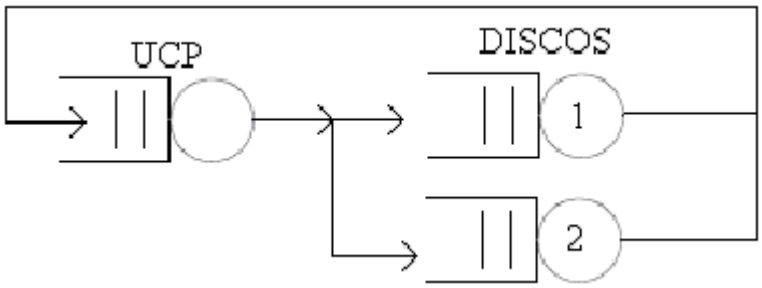
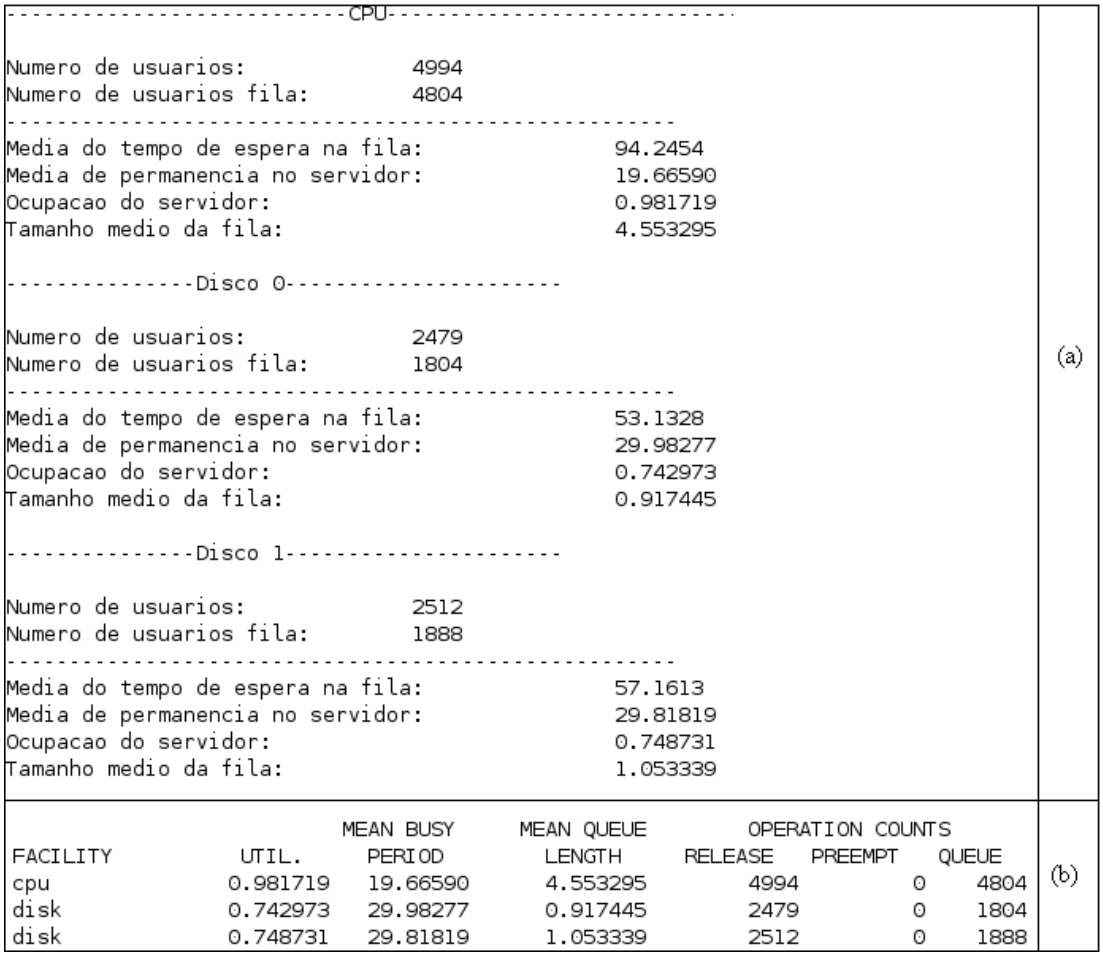


Figura 3.14 – Segundo modelo validado.

A figura 3.15 apresenta a saída do programa desenvolvido com a biblioteca RFOO (figura 3.15 (a)) e com a biblioteca SMPL (figura 3.15 (b)). Como se pode perceber, os resultados dos cálculos estatísticos das execuções com e o mesmo gerador de números pseudo-aleatórios e semente de geração é idêntico. Como exemplo, cita-se a taxa de ocupação da UCP que é de 0.981719 para as duas execuções.



```
----------------------------CPU-----------------------------

Numero de usuarios:             4994
Numero de usuarios fila:        4804
------------------------------------------------------
Media do tempo de espera na fila:               94.2454
Media de permanencia no servidor:               19.66590
Ocupacao do servidor:                           0.981719
Tamanho medio da fila:                          4.553295

---------------Disco 0----------------------

Numero de usuarios:             2479
Numero de usuarios fila:        1804
------------------------------------------------------
Media do tempo de espera na fila:               53.1328
Media de permanencia no servidor:               29.98277
Ocupacao do servidor:                           0.742973
Tamanho medio da fila:                          0.917445

---------------Disco 1----------------------

Numero de usuarios:             2512
Numero de usuarios fila:        1888
------------------------------------------------------
Media do tempo de espera na fila:               57.1613
Media de permanencia no servidor:               29.81819
Ocupacao do servidor:                           0.748731
Tamanho medio da fila:                          1.053339
```

(a)

		MEAN BUSY	MEAN QUEUE	OPERATION COUNTS		
FACILITY	UTIL.	PERIOD	LENGTH	RELEASE	PREEMPT	QUEUE
cpu	0.981719	19.66590	4.553295	4994	0	4804
disk	0.742973	29.98277	0.917445	2479	0	1804
disk	0.748731	29.81819	1.053339	2512	0	1888

(b)

Figura 3.15 – Resultados do segundo modelo utilizado para validação.

3.7 Considerações Finais

Este capítulo mostrou o desenvolvimento da biblioteca na linguagem C++ que auxilia na simulação de Redes de Filas. O usuário tem a disposição classes que contém as principais estruturas necessárias para a simulação, como centros de serviço, relógio do sistema, lista de eventos futuros e cálculos estatísticos.

A validação do sistema ocorreu com sucesso, efetuando-se a comparação dos resultados de um programa na biblioteca RFOO com o mesmo programa utilizando-se a biblioteca SMPL. Para a validação foi desenvolvido o modelo M/M/1. O capítulo quatro apresenta os testes de desenvolvimento com outros modelos de redes de filas.

Capítulo 4 - Testes

4.1 Considerações Iniciais

Neste capítulo os exemplos apresentados visam o uso da biblioteca RFOO como essa extensão funcional a qual poderá ser utilizada por alunos do curso de ciências da computação. São apresentados alguns exemplos através dos quais pode-se alterar os seus parâmetros e assim obter resultados diferentes que auxiliam na tomada de decisões por parte do usuário da biblioteca.

Para obter a certeza de que o modelo apresentava-se em equilíbrio durante a execução foi utilizado o *batch means* com 95% de confiança. Os parâmetros utilizados foram retirados dos autores consultados e não são representativos de sistemas reais.

4.2 Modelos Estudados

Esse modelo representa um sistema que contém uma UCP (Unidade Central de Processamento) com quatro discos (Macdougall, 1987), apresentado na figura 4.1, no qual processos chegam a uma UCP são executados e ao serem finalizados, são passados para o disco. Após o ciclo de leitura ou escrita no disco os processos retornam para a UCP.

Como esse é um sistema fechado, o número de usuários deve ser adicionado manualmente e não através de eventos do modelo. Nesse problema foram adicionados 100 usuários que iniciam o ciclo fazendo uma requisição à UCP, após sair da UCP o usuário escolhe aleatoriamente qualquer um dos quatro discos e faz uma requisição ao que foi escolhido. Como parâmetros para esse exemplo foram utilizados: atendimento pela UCP com tempo médio de 10 unidades de tempo segundo distribuição exponencial e aos discos uma média de 30 unidades de tempo com desvio-padrão de 7,5 segundo distribuição Erlang (MacDougall, 1987).

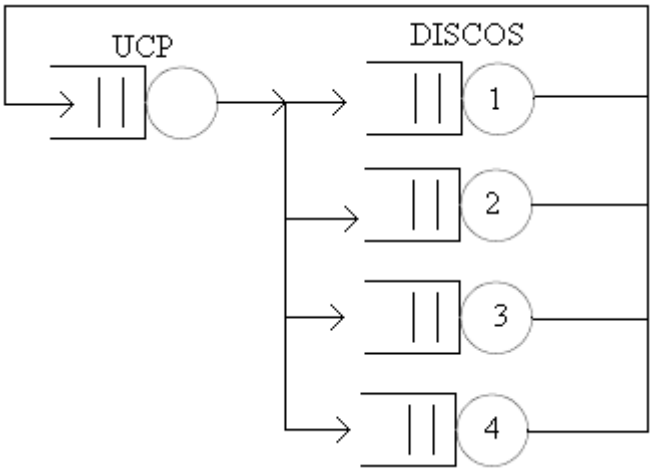


Figura 4.1 – Exemplo de uma UCP e 4 discos.

Outro exemplo utilizado para verificar o funcionamento da biblioteca foi o de um servidor de arquivos (Santana 1989b apud Spolon, 2001), apresentado na figura 4.2, no qual os clientes que chegam requisitam serviço à UCP, e após deixarem a UCP, podem abandonar o sistema (com probabilidade p) ou requisitar acesso ao disco (com probabilidade 1-p). Ao saírem do disco fazem uma nova requisição à UCP. Como parâmetros para esse modelo foram utilizados, tempo entre chegadas dos usuários ao sistema o qual é obtido por uma distribuição exponencial com média 5, o tempo médio de atendimento da UCP e do disco, respectivamente de 10 u.t. e 20 u.t segundo uma distribuição exponencial e a probabilidade p de um usuário abandonar o sistema ao sair da UCP que é igual a 0,7 (Spolon, 2001).

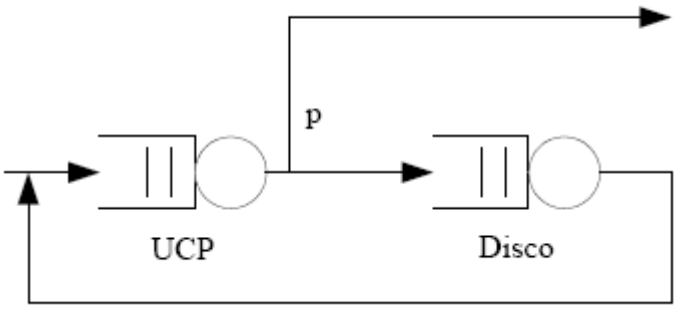


Figura 4.2 – Exemplo do servidor de arquivos.

O terceiro exemplo é um modelo hipotético, o qual é constituído por quatro centros de serviço em série, como apresentado na figura 4.3 (Balieiro, 2005). Nesse

modelo, quando um usuário chega ao sistema, faz uma requisição ao primeiro centro de serviço. Após receber serviço, o usuário requisita o terceiro centro de serviço. Após receber serviço o usuário retorna ao segundo centro com probabilidade p ou vai para quarto centro com uma probabilidade de 1-p, e após ser atendido, abandona o sistema.

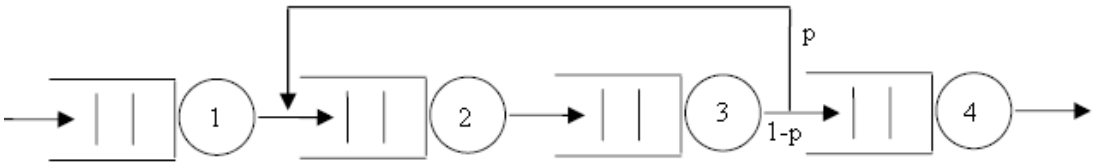


Figura 4.3 – Exemplo do modelo hipotético 1.

Os parâmetros definidos para esse sistema são a probabilidade de um usuário retornar ao segundo centro de serviço como p = 0,6, o tempo entre chegadas de usuários ao sistema de 50 unidades de tempo segundo uma distribuição exponencial e o tempo de atendimento igual a 30, 10, 10 e 60 unidades de tempo segundo uma distribuição exponencial para os centros de serviço 1, 2, 3 e 4 respectivamente.

O quarto exemplo é um modelo hipotético apresentado na figura 4.4 (Uls99 *apud* Balieiro, 2005). Esse é um modelo fechado, constituído por sete recursos, três dos quais estão em série e quatro em paralelo e dependentes de pontos de decisão.

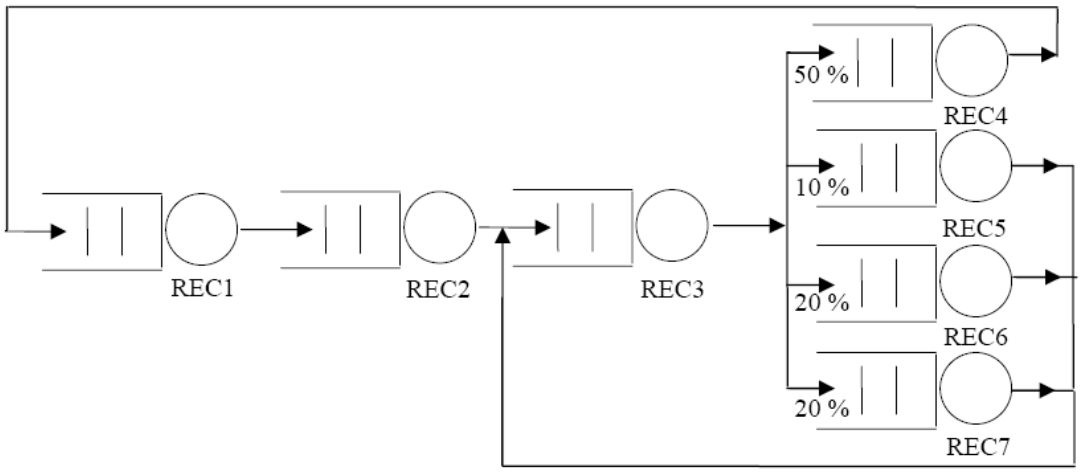


Figura 4.4 – Exemplo hipotético 2.

O trajeto dos usuários nesse sistema consiste em solicitar e receber atendimento do recurso 1, recurso 2 e recurso 3, ao sair desse recurso o usuário tem um ponto de decisão no qual pode seguir para qualquer um dos quatro recursos seguintes segundo uma probabilidade definida no modelo. Se o usuário escolher o recurso 4 após o atendimento volta ao recurso 1, caso contrário volta ao recurso 3.

Os parâmetros para esse sistema também são hipotéticos e não representativos de um sistema real. Os parâmetros são: tempo de atendimento médio de 50 u.t. segundo uma distribuição exponencial para os sete centros de serviço. Probabilidade de escolher o centro de serviço 4 ao sair do centro de serviço 3 de 0,5, probabilidade de 0,1 para o centro de serviço 5 e probabilidade de 0,2 para os centros 6 e 7. Como esse é um sistema fechado, foram inseridos um total de 100 usuários no sistema.

4.3 Soluções para os modelos

Os modelos propostos foram desenvolvidos com a biblioteca RFOO e os resultados de suas execuções estão descritos ao longo dessa seção. Foram realizadas execuções com os parâmetros descritos na seção anterior e novos valores de parâmetros foram utilizados para que resultados diferentes fossem alcançados com o mesmo modelo.

No modelo de uma UCP com quatro discos em um sistema fechado foram realizadas execuções variando-se o tempo médio de atendimento da UCP, o número de discos e o número de usuários e verificando-se o que ocorria com a taxa de ocupação da UCP e o tempo médio de fila dos discos.

A aplicação método *batch means* foi utilizada em todos os modelos para se obter o tempo necessário para a execução de cada modelo com a confiança desejada. No modelo da UCP e quatro discos os valores das médias dos lotes de uma execução está representado no gráfico apresentado na figura 4.5.

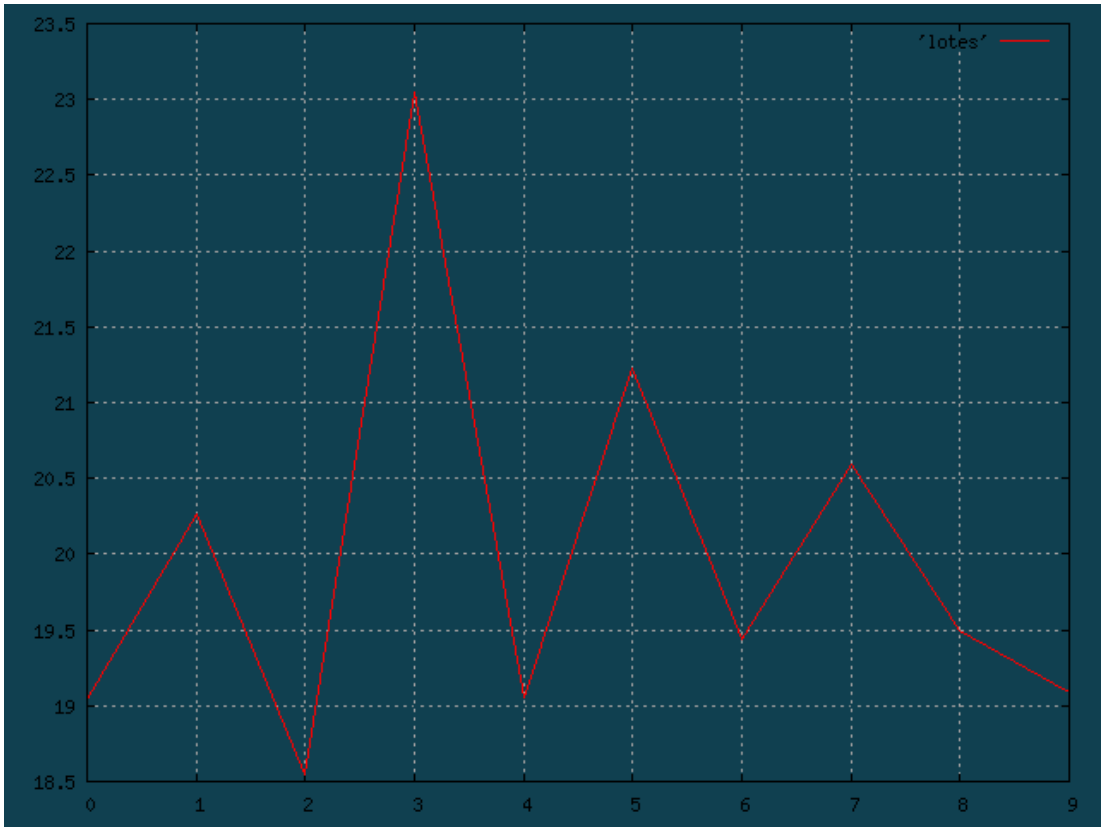


Figura 4.5 – Gráfico relativo ao modelo da UCP e quatro discos.

A tabela 4.1 apresenta os valores que foram atribuídos aos parâmetros de entrada e que foi obtido em relação às taxas de ocupação. Com a configuração que foi apresentada na seção anterior, a taxa de ocupação da UCP é de 1,0, o que indica que ela é utilizada o tempo todo, enquanto o tamanho médio da fila formada nos discos é de apenas 0,13 aproximadamente, o que indica que quase não se forma fila com essa configuração.

Pode-se perceber pela tabela que a diminuição do tempo médio de atendimento da UCP pela metade implica em alteração no tempo médio da fila nos discos, devido ao fato dos discos trabalharem com um tempo de atendimento mais lento em relação à UCP. Diminuindo-se cada vez mais essa taxa obtém-se uma menor taxa de ocupação da UCP e um aumento no tamanho médio das filas nos discos. Uma outra alteração realizada foi a diminuição do número de usuários no sistema para 50 e depois para 10. O que se pode observar dessa alteração é que o tempo médio de fila dos discos diminui de acordo com a subtração do número de usuários do sistema. Como último teste foi alterado o número de discos no sistema e

percebe-se que com a subtração, o tempo médio de fila nos discos aumenta, pois há um menor número de discos para atender o mesmo número de usuários. Com essa última alteração, a taxa de ocupação da UCP diminui, pois os usuários passam tempo maior na fila dos discos e um tempo menor na UCP.

Tabela 4.1 – Variações do modelo de UCP e 4 discos.

UCP		Discos					Número de usuários no sistema
Tempo Médio de atendimento	Taxa de ocupação	Número Discos	Tamanho médio das filas dos discos				
			Disco 1	Disco 2	Disco 3	Disco 4	
20	1,0	4	0,117	0,125	0,111	0,115	100
10	1,0	4	1,182	1,189	1,151	1,137	100
8	0,997	4	6,512	6,73	6,125	6,766	100
5	0,601	4	22,799	23,155	23,514	22,269	100
5	0,65	4	11,087	11,029	10,997	11,038	50
5	0,551	4	1,448	1,441	1,471	1,416	10
5	0,443	3	2,223	2,28	2,216	-	10
5	0,315	2	3,893	3,848	-	-	10
5	0,329	2	25,39	22,237	-	-	50
5	0,167	1	48,831	-	-	-	50

Os resultados das execuções do modelo do servidor de arquivos estão apresentados na tabela 4.2. Para os parâmetros apresentados na seção anterior, o tamanho da fila que se forma na UCP é de 1291,89, enquanto o disco apresenta um tamanho médio de fila de 0,887. Isso se deve ao fato de que o tempo entre chegadas de usuários ao sistema é menor que o tempo médio de atendimento da UCP e, além disso, 30% dos usuários que abandonam a UCP retornam a essa após serem atendidos pelo disco.

Uma forma de amenizar essa situação seria aumentar o tempo entre chegadas de usuários ao sistema. Isso foi realizado aumentando esse tempo para 10 e posteriormente para 15. Com essas alterações percebe-se que o tamanho médio da fila da UCP decresce para 292,584 e 17,547 respectivamente.

Para representar a situação de maior ocorrência de requisições ao disco, a probabilidade p de um usuário deixar o sistema ao sair da UCP foi alterada. Pôde-se perceber que o tempo médio de fila aumentou, tanto no disco quanto na UCP, devido ao fato de mais usuários continuarem no sistema. Quando a probabilidade diminui

para 0,5 o tamanho médio da fila na UCP é de 139,718 e de 22,972 no disco. Quando a probabilidade muda para 0,3, o tamanho médio da fila passa a ser de 166,565 na UCP e de 199,283 no disco. Se ao invés de diminuir a probabilidade de um usuário abandonar o sistema faz-se um acréscimo dessa probabilidade o tamanho médio das filas diminui tanto para a UCP quanto para o disco.

Retornando o tempo entre chegadas de usuários, primeiro para 10 e depois para 5, o tamanho médio da fila da UCP chega a 1091,9 (mantendo a probabilidade de um usuário abandonar o sistema em 0,9).

Diminuindo o tempo de atendimento da UCP para 5, o que equivale a trocar a UCP por uma mais rápida, percebe-se que o tamanho médio da fila na UCP diminui para 210,629. Diminuindo-se o tempo de atendimento médio da UCP para 4 o tamanho médio da fila da UCP chega a 7,06. A diferença entre o tempo médio de atendimento da UCP igual a 4 para quando o tempo médio era 5 ocorre porque com o tempo de atendimento da UCP é igual ao tempo entre chegadas de usuários ao sistema. Como existe uma realimentação decorrente da saída de usuários do disco, a fila da UCP cresce.

Tabela 4.2 - Variações do modelo do servidor de arquivos.

Tempo Entre chegadas	UCP		Disco		Probabilidade de abandonar o sistema
	Tempo Atendimento	Tamanho Médio da Fila	Tempo Atendimento	Tamanho Médio da Fila	
5	10	1291,89	20	0,887	0,7
10	10	292,584	20	0,857	0,7
15	10	17,547	20	0,737	0,7
15	10	139,718	20	22,972	0,5
15	10	166,565	20	199,283	0,3
15	10	2,209	20	0,0306	0,9
10	10	110,512	20	0,0509	0,9
5	10	1091,9	20	0,049	0,9
5	5	210,629	20	0,272	0,9
5	4	7,06	20	0,351	0,9
5	4	60,334	20	711,238	0,5
5	4	53,09	20	235,685	0,7
5	4	253,099	10	2,331	0,7

Pode-se perceber ainda que a alteração na probabilidade de usuários que deixam o sistema influi no tamanho médio da fila do disco e da UCP. Isso ocorre porque se essa probabilidade diminuir, um maior número de usuários irá continuar a realimentar o sistema e irão ser adicionados aos que já estão no sistema. Se o tempo de atendimento do disco for diminuído, o que equivale a trocar o disco por um mais rápido, o tamanho médio da fila no disco tende a diminuir.

Para o modelo hipotético 1 foi feita a suposição de o tempo entre chegadas de usuários ao sistema diminuir para 40 u.t., com isso o tamanho médio da fila do centro de serviço 4 (cs4) chega a 83,299 de acordo com a tabela 4.3

Diminuindo-se cada vez mais o tempo entre chegadas de usuários ao sistema percebe-se que o tamanho médio das filas dos quatro centros de serviço tende a aumentar conforme os usuários chegam ao sistema de forma mais rápida. Se for diminuído o tempo de atendimento do centro de serviço 1 (cs1), percebe-se que o tamanho médio da fila deste centro de serviço diminui, pois pode atender de forma mais rápida aos usuários que chegam em cs1.

Tabela 4.3 - Variações do modelo hipotético 1.

Tempo entre Chegadas	Tempo Atendimento				Probabilidade	Tamanho médio da fila			
	cs1	cs2	cs3	cs4		cs1	cs2	cs3	cs4
50	30	10	10	60	0,6	0,959	0,483	0,495	83,299
40	30	10	10	60	0,6	2,134	1,04	1,033	203,124
30	30	10	10	60	0,6	32,123	3,349	3,366	375,509
20	30	10	10	60	0,6	416,449	3,975	4,233	402,149
20	20	10	10	60	0,6	31,966	193,642	48,304	548,038
20	20	5	5	60	0,6	30,558	0,937	0,929	795,463
20	20	5	5	60	0,7	33,001	3,758	3,747	794,655
20	20	5	5	60	0,8	32,955	182,107	57,211	558,917
20	20	5	5	60	0,9	30,22	644,922	82,612	79,561

Deve-se dar uma atenção especial ao centro de serviço 2 (cs2) que recebe uma realimentação da saída do centro de serviço 3 (cs3) necessitando-se assim que esses dois centros de serviço fossem mais rápidos nos atendimento aos clientes. Aumentando-se a probabilidade p os tamanhos médios das filas, em cs2 e cs3, aumentam e em cs4 diminui, por causa dos usuários que retornam.

No modelo hipotético 2 percebe-se, pela tabela 4.4, que com os parâmetros iniciais, o centro de serviço 3 apresenta um tamanho médio de fila de 94,838 e, a alteração do tempo de atendimento deste centro de serviço, implica na diminuição do tamanho médio de sua fila e no aumento dos centros 1, 2 e 4. Os aumentos do tamanho médios das filas destes centros de serviço estão relacionados com um atendimento mais rápido por parte do recurso 3. O aumento ocorrido nos centros 1, 2 e 4 deve-se ao fato da probabilidade de um usuário requisitar recursos ao centro de serviço 4 é de 0,5, e após o atendimento os usuários retornam ao centro de serviço 1.

Se forem trocadas as probabilidades dos centros 4 e 5, os tempo de filas irão se alterar, pois, no exemplo, com a alteração realizada, o centro de serviço 5 apresenta um tamanho médio de fila de 96,141.

Outra configuração para o modelo estudado é utilizar uma probabilidade de 0,7 para centro de serviço 4 e 0,1 para os centros 5, 6 e 7. O que se pode perceber é que, com essa alteração as filas dos centros 1, 2 e 4 aumentam novamente e as filas dos centros 5, 6 e 7 tendem a se igualar quando o sistema tende para o equilíbrio.

Tabela 4.4 – Variações do modelo hipotético 2.

Tempo médio de Atendi-mento		Probabilidade de o cliente acessar o recurso				Tamanho médio da fila						
rec3	rec4	rec4	rec5	rec6	rec7	rec1	rec2	rec3	rec4	rec5	rec6	rec7
50	50	0,5	0,1	0,2	0,2	0,976	0,508	94,838	0,5	0,012	0,052	0,05
25	50	0,5	0,1	0,2	0,2	24,862	24,334	23,754	21,664	0,046	0,248	0,246
10	50	0,5	0,1	0,2	0,2	30,197	30,926	0,253	33,753	0,047	0,261	0,252
10	50	0,1	0,5	0,2	0,2	0,113	0,112	0,027	0,049	96,141	0,26	0,29
10	50	0,7	0,1	0,1	0,1	31,772	31,4	0,11	31,074	0,024	0,024	0,024
10	10	0,7	0,1	0,1	0,1	48,244	48,692	0,11	0,049	0,023	0,023	0,023

Quando o tempo de atendimento no centro de serviço 4 diminui, percebe-se que a fila em 1 e 2 aumenta, isso ocorre porque os usuários que estavam sendo enfileirados no centro de serviço 4 estão agora ocupando a fila no centro de serviço 1 e 2.

4.4 Considerações Finais

Através desse capítulo pôde-se mostrar o uso da biblioteca RFOO no desenvolvimento da simulação de modelos de redes de filas. Pôde-se utilizar a biblioteca para a simulação de servidores mais rápidos ou mais lentos e perceber as modificações dos resultados. Através dessas modificações pode-se tomar conclusões quanto ao benefício, por exemplo, de se adicionar uma UCP mais rápida em um servidor de arquivos. As conclusões e sugestões para trabalhos futuros serão apresentadas no próximo capítulo.

Cap 5 – Conclusão, contribuições e propostas para trabalhos futuros

Através desse trabalho foi possível aprender sobre avaliação de desempenho, especialmente simulação e técnicas de modelagem de redes de filas. Para isso foi necessário o estudo das técnicas de avaliação de desempenho e da modelagem por redes de filas e sua teoria. A teoria apresentada no segundo capítulo pode ser utilizada por outros alunos para base de trabalhos futuros. O trabalho desenvolvido consistiu na construção de uma biblioteca que auxilia na construção de um programa para a simulação de redes de filas. Essa biblioteca também pode ser utilizada por outros alunos como uma forma de estudo da simulação de redes de filas através de uma linguagem orientada a objetos ou ainda para o aprimoramento dessa biblioteca.

No capítulo 4 foram apresentados alguns exemplos de como os programas desenvolvidos com essa biblioteca podem ser utilizados e como a simulação pode facilitar a tomada de decisão de projeto como na compra de equipamentos, por exemplo.

Como sugestão para trabalhos futuros pode-se realizar melhorias na biblioteca como a construção de métodos que possibilitem a simulação com preempção, centros de serviço com múltiplas filas e usar a técnica de replicação como análise de saída. Uma outra sugestão é a construção de um gerador de código a partir dessa biblioteca a partir de parâmetros estabelecidos pelo usuário.

Referências Bibliográficas

(Balieiro, 2005) BALIEIRO, M. O. S., *Protocolo Conservativo CMB para computação distribuída,* Dissertação de mestrado, DCT – UFMS, 2005.

(Banks *et al.*, 2004) BANKS, J., CARSON, J. S. II, NELSON, B. L., *Discrete-event System Simulation***,** Quarta edição, Prentice Hall, 2004.

(Freitas ,2001) FREITAS, P. J.F., *Introdução à Modelagem e Simulação de Sistemas*, Visual Books Ltda, 2001.

(Gnome, 2005) GNOME OFFICE, *página de referência sobre os softwares do pacote gnome office,* 2005, Disponibiliza o programa e informações sobre o mesmo, Disponível em: http://www.gnome.org/gnome-office/eog.shtml acessado em 14 de outubro de 2005.

(Gnuplot, 2005) GNUPLOT HOMEPAGE, *página oficial do programa gnuplot,* 2005, Disponibiliza o programa e informações sobre o mesmo, Disponível em: http://www.gnuplot.info/ acessado em 10 de agosto de 2005.

(Gu *et al.*, 2004) GU, Y., LIU, Y., MISRA, V., PRESTI, F. L., TOWSLEY, D. F., *Scalable fluid models and simulations for large-scale IP networks,* ACM Press –TOMACS, volume 14, issue 3, p. 305-324, 2004.

(Kim & Kim, 2005) KIM, H. Y., KIM, T. G., *Performance simulation modeling for fast evaluation of pipelined scalar processor by evaluation reuse***,** proceedings of annual conference on Design automation, p. 341-439, 2005.

(Kurose & Ross, 2003) KUROSE, J. F., ROSS, K. W., *Redes de Computadores e a Internet: Uma Nova Abordagem*, Pearson – Addison-Wesley, 1ª edição, 2003.

(Macdougall, 1987) MACDOUGALL, M. H., *Simulating Computer Systems Techniques and Tools,*The MIT Press, 1987.

(Mirlovic *et al.*, 2005) MIRLOVIC, J., SWANY, M., WEI, S., *Distributed Worm Simulation with a Realistic Internet Model,* Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, p.71-79, 2005.

(Orlandi, 1995) ORLANDI, R. C. G. S., *Ferramentas para Análise de Desempenho de Sistemas Computacionais Distribuídos,* Dissertação (Mestrado), ICMC - USP, São Carlos, 1995.

(Sacchi 2005) SACCHI, R. P. S., *ETW: Um núcleo para simulação distribuída otimista,* dissertação de mestrado, DCT –UFMS, 2005.

(Santana *et al.*, 1994) SANTANA, R. H. C., SANTANA, M. J., ORLANDI, R. C. G. S., SPOLON, R., CALÔNEGO, N. Jr., *Técnicas para avaliação de desempenho em sistema computacionais,* Notas didáticas do ICMC, ICMC – USP, 1994.

(Silva, 2000) SILVA, A. R. F., *Modelos de Rede de Filas para Sistemas Computacionais Distribuídos: Simulação X Métodos Analíticos,* Dissertação de Mestrado, ICMC – USP, 2000.

(Soares, 1990) SOARES, L. F.G., *Modelagem e Simulação Discreta de Sistemas*, VII Escola de Computação, 1990.

(Spolon, 2001) SPOLON, R., *Um método para a avaliação de desempenho de protocolos de sincronização otimistas para simulação distribuída,* Tese de doutorado, FFI-IFSC-USP, 2001.

(Stroustrup, 2000) STROUSTRUP, B., *A linguagem de programação C++*, Bookman, 3ª edição, 2000.

(Tanenbaum, 2003) TANENBAUM, A. S., *Sistemas Operacionais Modernos*, Pearson – Prentice Hall, 2ª edição, 2003.

(Wang & Wolff, 2003) WANG, C., WOLFF, R. W., *Efficient simulation of queues in heavy traffic*, Volume 13, issue 1, p. 62-82 ACM Press - TOMACS, 2003.

(Weicker, 2002) WEICKER, R., *Performance evaluation of complex systems: techniques and tools*, Lecture Notes Computer Science, p. 181-182, 2002.

(Zafalon & Manacero, 2004) Zafalon, G. F. D., Manacero Júnior, A., *Desenvolvimento de geradores independentes de números aleatórios, segundo distribuições probabilísticas variadas*, in XVI Congresso de Iniciação Científica (CIC) da Unesp, Ilha Solteira, 2004.

Apêndice A – Estrutura e saída do programa

Este apêndice apresenta o programa referente ao modelo do servidor de arquivos como exemplo de uma estrutura de um programa desenvolvido com a biblioteca RFOO, além disso, são apresentadas também algumas execuções desse programa com diferentes sementes de geração de números pseudo-aleatórios.

Como código do programa apresenta-se:

```
//parametros do sistema
const double ENTRECHEGADA = 5.0; // média do tempo entre chegadas
const double SERVICOCPU   = 4.0; // média do tempo de serviço da UCP
const double SERVICODISCO = 10.0; // média do tempo de serviço do disco
const double P            = 0.7; // probabilidade de o cliente deixar do
                                    //sistema

//eventos
const int CHEGADA       = 1;
const int REQUISITACPU  = 2;
const int SAIDACPU      = 3;
const int REQUISITADISCO = 4;
const int SAIDADISCO    = 5;

//fim da simulação
const double FIM = 10000.0;

int main(){
  // iniciando os centros de serviços
  CentrodeServico cpu(1),disco(1);

  // iniciando a lef, o relógio e o gerador
  ListaEventoFuturo lef(0,NULL);
  relogio Relogio(0);
  tRand Gerador(1);

  //iniciando as classes Estatísticas
  Estatistica EstatCpu(NULL), EstatDisco(NULL);
```

```
//iniciando os usuários
int usuario = 0;

cpu.iniciaSimulacao(&lef,usuario,CHEGADA);

while(Relogio.recebeTempoAtual() < FIM){
  switch(lef.getEvento(&Relogio)){
        case CHEGADA :  //Chegada de um usuário ao sistema
                        // é colocado o usuário atual em atendimento
                        lef.addEvento(lef.getUsuario(),
                                        lef.getTempo(),REQUISITACPU);
                        // é gerada uma nova chegada
                        lef.addEvento(++usuario,lef.getTempo()+
                                Gerador.Expntl(ENTRECHEGADA),CHEGADA);
                        break;
        case REQUISITACPU : // usuário requisita a cpu
                           // se a CPU estiver vazia o usuário entra em
                           // atendimento
                           if(cpu.requisitaServidor(&EstatCpu,&lef)){
                                 lef.addEvento(lef.getUsuario(),
                                   lef.getTempo()+Gerador.Expntl(SERVICOCPU),
                                                SAIDACPU);
                           }
                           break;
        case SAIDACPU : // sai da cpu e pode sair do sistema ou requisitar o
                        // disco
                        double escolha;
                        escolha = Gerador.Ranf();
                        if(escolha >= P) {
                          // probabilidade de o usuário requisitar o disco
                          lef.addEvento(lef.getUsuario(),lef.getTempo(),
                                        REQUISITADISCO);
                        } //caso abandone o disco não há eventos a serem
                          //realizados
                        //liberando a cpu
                        cpu.liberarServidor(&lef,&EstatCpu,REQUISITACPU);
                        break;
        case REQUISITADISCO : // requisita leitura do disco
                           if(disco.requisitaServidor(&EstatDisco,&lef)){
```

```
                                    lef.addEvento(lef.getUsuario(),
                                        lef.getTempo()+Gerador.Expntl(SERVICODISCO),
                                                    SAIDADISCO);
                                    }
                                    break;
            case SAIDADISCO : // sai do disco e requisita a cpu
                                lef.addEvento(lef.getUsuario(),lef.getTempo(),
                                            REQUISITACPU);
                                disco.liberarServidor(&lef,&EstatDisco,
                                                        REQUISITADISCO);
                                break;
              }
            }

            cout << "\n----------------------------CPU-------------------------------";
            EstatCpu.geraEstatistica();
            cout << "\n---------------------------Disco-----------------------------";
            EstatDisco.geraEstatistica();
            cout << "\nTempo da simulacao: " << Relogio.recebeTempoAtual() << endl;
            return(0);

}
```

Abaixo estão apresentadas as saídas das execuções do programa com as sementes 1973272912, 747177549, 20464843, 640830765, 1098742207, respectivamente.

Execução do programa com a semente 1973272912:

```
----------------------------CPU-------------------------------
Numero de usuarios:            5061
Numero de usuarios fila:       5045
-----------------------------------------------------
Media de permanencia no centro de servico:     650.281
Media do tempo de espera na fila:              646.347
Media de permanencia no servidor:               3.93503
Ocupacao do servidor:                         0.995674
Tamanho medio da fila:                         180.304
```

----------Disco----------
Numero de usuarios: 1498
Numero de usuarios fila: 1136

Media de permanencia no centro de servico: 43.0662
Media do tempo de espera na fila: 34.7143
Media de permanencia no servidor: 9.94615
Ocupacao do servidor: 0.745253
Tamanho medio da fila: 2.48165

Tempo da simulacao: 20001.7

Execução do programa com a semente 747177549:

----------CPU----------
Numero de usuarios: 4939
Numero de usuarios fila: 4933

Media de permanencia no centro de servico: 1155.58
Media do tempo de espera na fila: 1151.54
Media de permanencia no servidor: 4.04413
Ocupacao do servidor: 0.99878
Tamanho medio da fila: 320.792

----------Disco----------
Numero de usuarios: 1554
Numero de usuarios fila: 1183

Media de permanencia no centro de servico: 40.6123
Media do tempo de espera na fila: 32.9117
Media de permanencia no servidor: 9.9432
Ocupacao do servidor: 0.773825
Tamanho medio da fila: 2.38681

Tempo da simulacao: 20001

Execução do programa com a semente 20464843:

----------CPU----------
Numero de usuarios: 4965
Numero de usuarios fila: 4960

Media de permanencia no centro de servico: 1036.19
Media do tempo de espera na fila: 1032.16
Media de permanencia no servidor: 4.02524
Ocupacao do servidor: 0.999237
Tamanho medio da fila: 288.664

----------Disco----------
Numero de usuarios: 1498
Numero de usuarios fila: 1090

Media de permanencia no centro de servico: 38.3323
Media do tempo de espera na fila: 31.0905
Media de permanencia no servidor: 9.56692
Ocupacao do servidor: 0.716977
Tamanho medio da fila: 2.15582

Tempo da simulacao: 20000.6

Execução do programa com a semente 640830765:

----------CPU----------
Numero de usuarios: 4913
Numero de usuarios fila: 4912

Media de permanencia no centro de servico: 979.418
Media do tempo de espera na fila: 975.348
Media de permanencia no servidor: 4.07056
Ocupacao do servidor: 0.999939
Tamanho medio da fila: 256.64

----------Disco----------
Numero de usuarios: 1427
Numero de usuarios fila: 1022

Media de permanencia no centro de servico: 34.6773
Media do tempo de espera na fila: 26.686
Media de permanencia no servidor: 10.1844
Ocupacao do servidor: 0.727847
Tamanho medio da fila: 1.78462

Tempo da simulacao: 20000.9

Execução do programa com a semente 1098742207:

----------CPU----------

Numero de usuarios: 4999

Numero de usuarios fila: 4947

Media de permanencia no centro de servico: 710.234

Media do tempo de espera na fila: 706.299

Media de permanencia no servidor: 3.96879

Ocupacao do servidor: 0.991664

Tamanho medio da fila: 194.576

----------Disco----------

Numero de usuarios: 1452

Numero de usuarios fila: 1045

Media de permanencia no centro de servico: 31.1025

Media do tempo de espera na fila: 22.3846

Media de permanencia no servidor: 10.1

Ocupacao do servidor: 0.734215

Tamanho medio da fila: 1.52678

Tempo da simulacao: 20006.7