# SIPPO 0.1 - SIMULADOR PARA PROBLEMAS DE PESQUISA OPERACIONAL

**Túlio Almeida Peixoto**
Universidade Candido Mendes - UCAM
Rua Anita Pessanha, 100, Campos dos Goytacazes, RJ, Brazil
tulioap@gmail.com

**João José de Assis Rangel**
Universidade Candido Mendes - UCAM
Rua Anita Pessanha, 100, Campos dos Goytacazes, RJ, Brazil
joao@ucam-campos.br

**Ítalo de Oliveira Matias**
Universidade Candido Mendes - UCAM
Rua Anita Pessanha, 100, Campos dos Goytacazes, RJ, Brazil
italo@ucam-campos.br

## RESUMO

Este artigo apresenta o simulador SIPPO (Simulador para Problemas de Pesquisa Operacional) que está sendo desenvolvido para simular modelos aplicados ao campo da pesquisa operacional. O objetivo deste simulador é o de contribuir para facilitar a difusão, o uso e a compreensão no Brasil da simulação desde a sua aplicação prática até a concepção interna de sua estrutura computacional e código fonte. Este simulador utiliza como base a biblioteca para simulação discreta JSL e uma modelagem baseada em fluxo de entidades. Os primeiros resultados obtidos para uma simulação do serviço de caixa de banco foram próximos aos resultados obtidos em simuladores comerciais como o Arena® e ProModel®.

**PALAVRAS-CHAVE. Simuladores. Simulação. IDEF-SIM.**

**Área principal: SIM - Simulação**

## ABSTRACT

This article presents the SIPPO simulator (Simulator for Operational Research Problems) that is being developed to simulate models applied to the field of operational research. The objective of this simulator is to help facilitate the dissemination, use and understanding of the simulation in Brazil since its practical application to the design of its internal structure and source code. This simulator uses the JSL library for discrete simulation modeling and a flow-based entities modelling. The first results for a simulation of bank teller's service were similar to results from commercial simulators such as Arena® and ProModel®.

**KEYWORDS. Simulators. Simulation. IDEF-SIM.**

**Main Area: SIM - Simulation**

# 1. Introdução

Este artigo apresenta o simulador SIPPO (Simulador para Problemas de Pesquisa Operacional) que está sendo desenvolvido para simular modelos aplicados ao campo da pesquisa operacional, especificamente para simulação de sistemas a eventos discretos. O objetivo deste simulador é o de contribuir para facilitar a difusão, o uso e a compreensão no Brasil da simulação desde a sua aplicação prática até a concepção interna de sua estrutura computacional e código fonte. Segundo Chwif e Medina (2006), o Brasil, em termos de disseminação e utilização da simulação computacional, ainda está com mais de uma década de atraso quando comparado com empresas norte-americanas e européias. Também, o número de técnicos habilitados em modelar e operar ambientes de simulação ainda é muito precário. Pode-se atribuir como uma das causas desta defasagem ao preço dos ambientes de simulação, já que normalmente tais ambientes são relativamente caros em questão de licença de uso e treinamentos. Em outros países a difusão da simulação se deu em consonância com o surgimento das primeiras linguagens computacionais como Simula e SIMSCRIPT que originalmente foi uma biblioteca de simulação para FORTRAN.

Foi o Simula a primeira linguagem de programação utilizada especialmente para simulação de sistemas a eventos discretos (DAHL et al., 1970), desenvolvida no Centro de Computação Norueguês em Oslo nos anos 60. Simula introduziu o uso da programação orientada a objetos e difundiu este paradigma de programação como Smaltalk (KAY, 1993) e Java (GOSLING, 2005).

Surgiram paralelamente a Simula outras linguagens específicas para simulação discreta como SIMSCRIPT em que a modelagem é orientada a eventos e GPSS (HENRIKSEN, 2000) que introduziu o paradigma de modelagem de fluxo de transações. Mais tarde, o GPSS influenciou a linguagem SIMAN (NANCE, 1995), que por sua vez derivou no ambiente de simulação Arena®.

Outros ambientes como ProModel®, Simul8® e o FlexSim® todos de origem norte americana são ambientes de simulação utilizados no Brasil. O ProModel® e o Simul8® usam o mesmo paradigma de modelagem de fluxo de transações utilizados no Arena com algumas diferenças como a forma de fazer animações. Já FlexSim® utiliza orientação a objeto e faz animações em 3D. Nos EUA existem, além desses ambientes mencionados, dezenas de outros simuladores de eventos discretos.

# 2. Ambientes e Bibliotecas de Simulação

Primeiramente é preciso distinguir ambientes de simulação e bibliotecas de simulação. Os ambientes gráficos e integrados geralmente não necessitam que o usuário entre com linhas de código para construir seus modelos de simulação, além de permitir animação da simulação do modelo. Já as bibliotecas de simulação geralmente são constituídas de um grupo de rotinas codificadas em uma linguagem de propósito geral de forma que esta esteja adequada para simular modelos (BANKS, 2010).

Atualmente estão disponíveis diversos ambientes para simulação de eventos discretos. O Arena® e o ProModel®, por exemplo, são relativamente fáceis de operar, mas tem a desvantagem de poucos especialistas treinados nestes softwares no Brasil, do preço da licença de uso ser ainda relativamente cara e a impossibilidade de distribuir o modelo de simulação integrado ao simulador sem o pagamento de tal licença. Por outro lado, existem simuladores em formato de biblioteca de software com licença de uso livre como o DSOL em Jacobs e Verbraek (2002) e o JSL (Java Simulation Library) em Rosset (2008), ambas sob a licença GNU General Public Licence (www.gnu.org).

Em seguida serão descritos alguns ambientes e bibliotecas de simulação. O foco da descrição será os ambientes de simulação mais utilizados no Brasil e algumas bibliotecas de simulação utilizadas mundialmente.

## Arena

O Arena® da Rockwell Corporation é um ambiente gráfico integrado de simulação discreta que contém os chamados *templates* que são blocos gráficos usados para construir um fluxograma que representa o modelo de simulação. Os *templates* são constituídos de blocos de código da linguagem SIMAN permitindo que o usuário execute a simulação sem precisar ver ou entrar com linha de código, salvo quando ele quiser construir seu próprio *template* para exprimir um comportamento do modelo que não existe no conjunto de *templates* do Arena®. A animação é opcional e feita separada do modelo de simulação devido à herança do software CINEMA que foi complementar a linguagem SIMAN antes do surgimento do Arena®.

A Figura 1 apresenta a tela de inicial do Arena®, onde podem ser visualizados os ambientes relativos à Área de Trabalho e aos *Templates*.

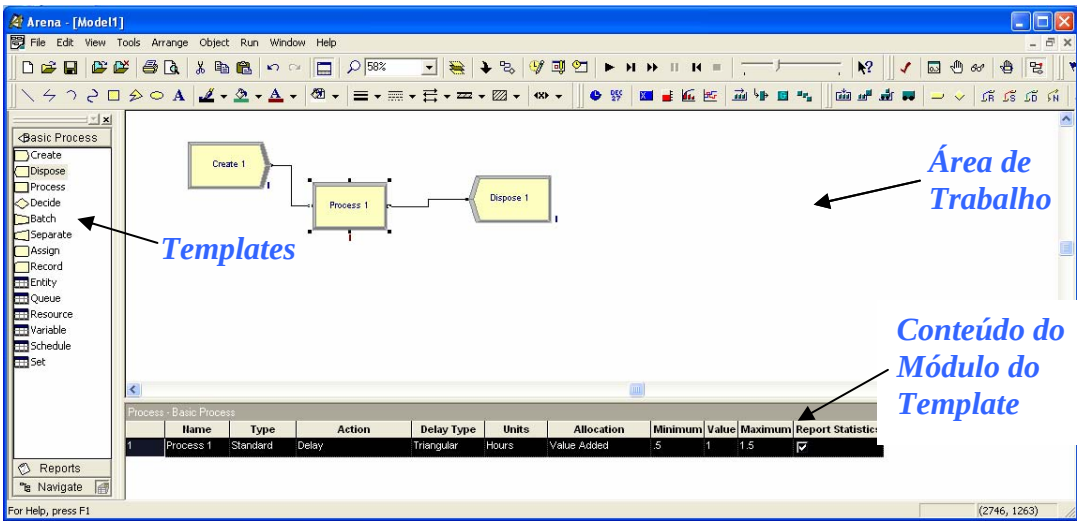


Figura 1: Ambiente de Desenvolvimento do Arena®.

## ProModel

Já no ProModel®, desenvolvido pela ProModel Corporation, o modelo de simulação e animação são integrados desde o início da modelagem. Se o usuário muda o modelo, automaticamente muda a animação. Isto pode ser útil, já que uma forma de validação do modelo é ver a animação que mostra as entidades fluindo pelo modelo.

Os principais elementos do ProModel® são: *locations* (locais), *entities* (entidades), *arrivals* (chegadas), *processing* (processos). Os *locations* são elementos fixos onde as operações, neles são definidos: capacidade, unidades, regras de atendimento (FIFO, LIFO, etc.). As *entities* são elementos móveis que sofrem alguma transformação nos *locations*. Podem representar matéria-prima, produtos, pessoas, etc. Nos *arrivals* são definidos parâmetros como local da chegada da entidade no sistema, quantidade freqüência e distribuições de probabilidade. Em *processing* define-se as interligações entre os locais, os tempos de operação, os recursos necessários, a lógica de movimentação, os roteamentos de entidade, etc. É possível no ProModel® inserir comandos específicos de simulação e também criar rotinas específicas através do *Logic builder*. Este recurso do software confere a flexibilidade de programação de diferentes lógicas, de acordo com cada sistema analisado.

A Figura 2 apresenta a tela inicial para desenvolvimento de modelos com o ProModel, onde podem ser visualizados as partes relativas aos locais, entidades, etc.

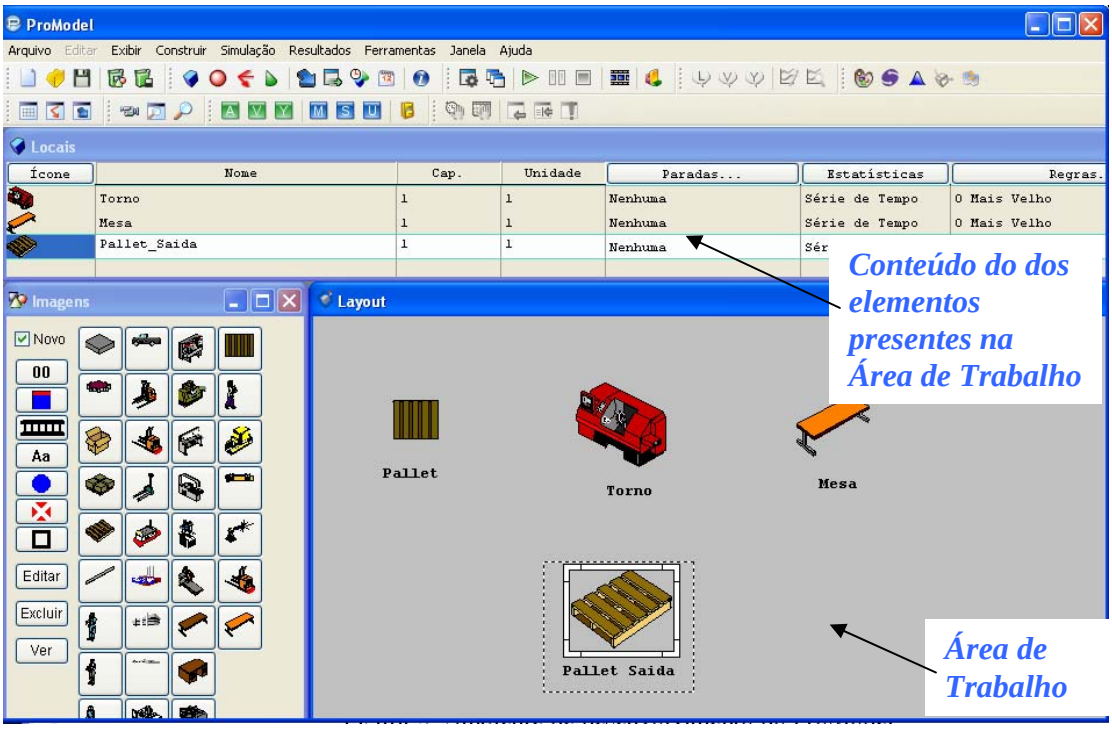


Figura 2: Ambiente de desenvolvimento do ProModel®.

## JSL

A biblioteca de simulação discreta JSL (*Java Simulation Library*) permite a modelagem do sistema orientada a eventos (*Event View*) ou orientada a processos (*Process View*). Na primeira abordagem, modela os pontos no tempo (eventos) quando o estado do sistema pode mudar e o tempo avança de um evento a outro como na linguagem SIMSCRIPT. Já a abordagem orientada a processos modela o fluxo de entidades (transações) através de uma série de processos.

Ambientes de simulação como o Arena® descrevem o processo (ou vida de uma entidade) através de uma serie de comandos. Isto é feito via uma representação em rede de fluxo de entidades como descrito em Joines e Robert (1996). O JSL implementa o processo deixando o usuário descrever o fluxo de uma entidade através de uma série de comandos. Essencialmente, os comandos são mantidos em uma lista e o comando atual que a entidade está em execução é sempre apontado para que o próximo comando possa ser executado. Isto é similar a noção de controle de estado de Jacobs e Verbraek (2002).

A Figura 3 mostra o ambiente de desenvolvimento NetBeans com a biblioteca JSL.

## DSOL

A biblioteca em Java DSOL (*Distributed Simulation Object Library*) permite uma modelagem usando vários formalismos como guiado a eventos, a processo e a atividade. Também permite simulação de sistemas de eventos discretos combinada com simulação de sistemas de tempo contínuo. No DSOL há a possibilidade de informações que estão distribuídas em vários sistemas de informação sejam utilizadas para alimentar o modelo de simulação. A simulação, por exemplo, uma cadeia de suprimentos em que sistemas de informação estão distribuídos em toda cadeia (VERBRAECK, 2004) torna-se útil um simulador que também trabalhe com dados distribuídos como DSOL faz.

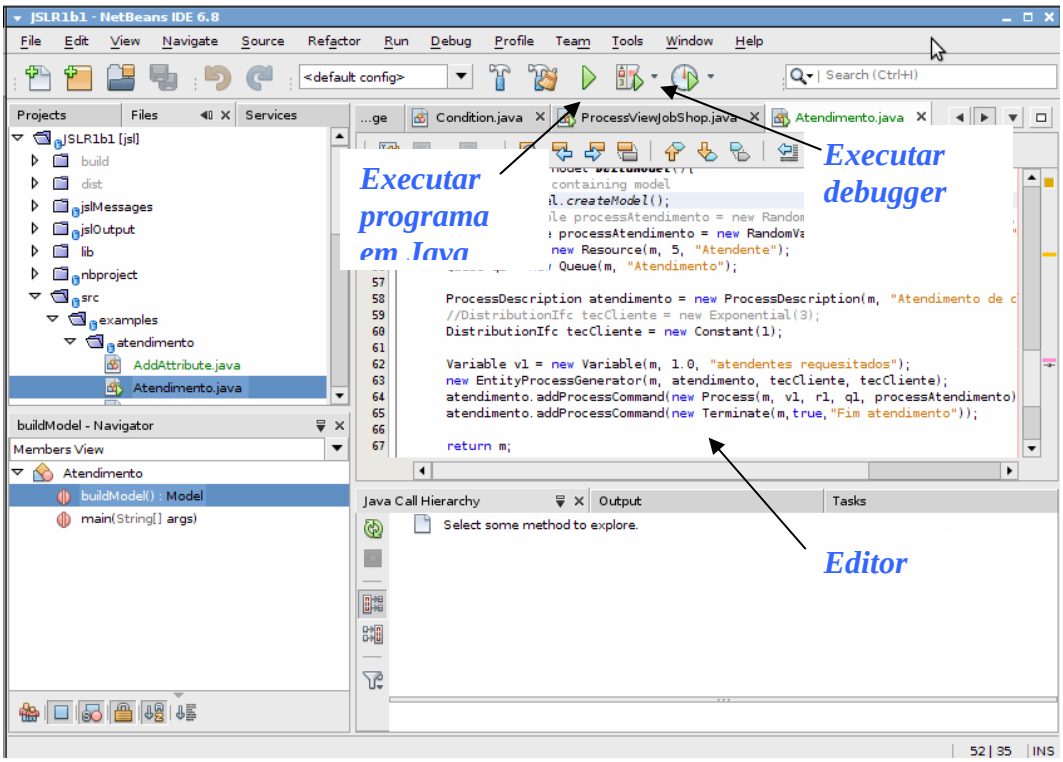


Figura 3: Ambiente de desenvolvimento NetBeans com o uso da biblioteca JSL.

A simulação de um modelo no DSOL é também definida e compilada em Java como no JSL, mas o DSOL interpreta o código compilado do modelo e executa o experimento que pode ser definido em um arquivo XML. A Figura 4 mostra a interface gráfica que vem com a biblioteca que auxilia o analista a executar e ver a saída de seus experimentos.

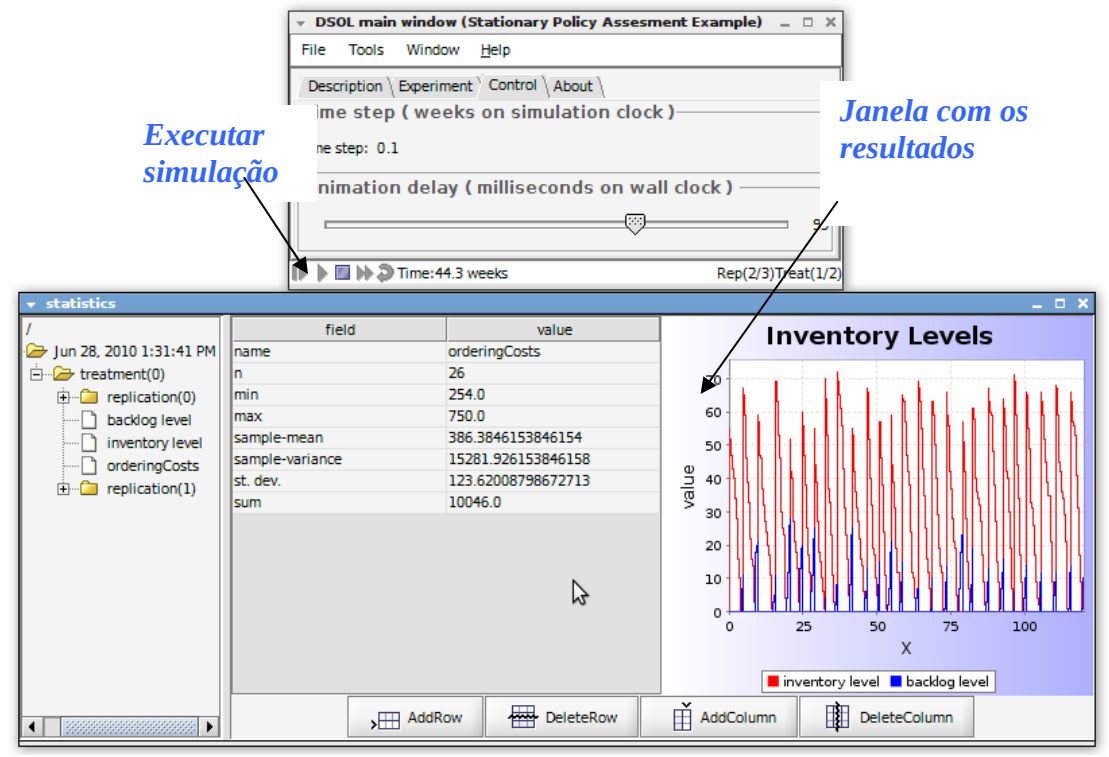


Figura 4: Interface gráfica para o DSOL.

# 3. O Simulador SIPPO

O simulador SIPPO (que está em sua versão 0.1) usa como base de funcionamento a biblioteca JSL por ter todos componentes essenciais, como: gerador de números aleatórios, distribuições de probabilidade, tratamento dos dados de saída, extensibilidade e multiplataforma; por ser escrita em Java. Internamente, os modelos são construídos orientados a processos e novos comandos podem ser implementados na medida em que surgem modelos mais complexos. Estes comandos se assemelham em funcionalidade como os *templates* do Arena®.

O código do modelo da fila M/M/1 com tempo entre chegadas médio de 1 minuto e tempo de serviço médio de 2 minutos é mostrado na Figura 5. Na linha 45 é criado o *ProcessDescription* que internamente cria uma lista de comandos necessários para descrever o modelo que podem ser *Seize, Delay, Release, Process* dentre outros. As entidades são criadas com *EntityProcessGenerator* (linha 49) com o tempo entre chegadas de 1 unidade de tempo médio com distribuição exponencial. Em seguida, são definidos os comandos onde passaram as entidades (clientes) que são *Process* (linha 50) e *Terminate* (linha 51). O comando *Process* requisita um recurso (variável *v1* na linha 48) que existe no conjunto de recursos que no total é apenas de um servidor (linha 42). No mesmo comando, *Process*, tem o tempo médio de serviço de 2 unidades de tempo e segue uma distribuição exponencial (linhas 40 e 41). Finalmente, o comando *Terminate* é onde as entidades saem do sistema.

```
14  public class MM1 {
15
16      public static void main(String[] args) {
17
18          Model m = buildModel();
19          // criar o experimento que executa o modelo
20          Experiment e = new Experiment(m, "Modelo MM1");
21
22          // definir os parâmetros do experimento
23          e.setNumberOfReplications(100);
24          e.setLengthOfReplication(100.0);
25          e.setLengthOfWarmUp(0);
26
27          // mostrar relatórios
28          e.turnOnExperimentReport();
29          e.turnOnExperimentConsoleOutput();
30
31          // executar experimento
32          e.runAll();
33          System.out.println("Feito!");
34      }
35
36      public static Model buildModel(){
37
38      // criar modelo
39      Model m = Model.createModel();
40      RandomVariable tcServidor = new RandomVariable(m,
41                                      new Exponential(2), "processAtendimento");
42      Resource r1 = new Resource(m, 1, "Servidor");
43      Queue q1 = new Queue(m, "Fila");
44
45      ProcessDescription mm1 = new ProcessDescription(m, "Modelo M/M/1");
46      DistributionIfc tecCliente = new Exponential(1);
47
48      Variable v1 = new Variable(m, 1.0, "servidores requesitados");
49      new EntityProcessGenerator(m, mm1, tecCliente, tecCliente);
50      mm1.addProcessCommand(new Process(m, v1, r1, q1, tcServidor));
51      mm1.addProcessCommand(new Terminate(m,true,"Fim do servico"));
52
53      return m;
54
55      }
56  }
```

Figura 5: Modelo MM1 em JSL.

Depois de definido o modelo é necessário definir os parâmetros do experimento (Figura 5) que são: número de replicações (linha 23), tamanho das replicações (linha 24), período de aquecimento ou *Warm-up* (linha 25) e, se desejar, mostrar relatórios (linhas 28 e 29).

Para tratar os dados de entrada será usada a biblioteca JSC (*Java Statistical Classes*) proposto por Bertie (2002). O tratamento da entrada consiste em encontrar uma distribuição que tenha melhor aderência e JSC implementa o método KS para atingir este objetivo.

A licença de uso do SIPPO assim como as bibliotecas que ele usa é GNU GPL (*GNU General Public Licence*) o que permite seu uso sem pagamento de licença. Mais informações sobre esta licença podem ser obtidas em www.gnu.org.

# 4. Aplicação do SIPPO em um Problema de Atendimento

No primeiro momento a aplicação do SIPPO será em um problema dito de atendimento, problema este que é relativamente simples de funcionar, pois apresentam poucas regras operacionais. Problemas como este são normalmente classificados como: dinâmicos, estocásticos e discretos. Como problemas de atendimento podem-se incluir: agências bancárias, postos de atendimentos e serviços, *call centers*, sistemas computacionais cliente-servidor, servidores *web*, dentro outros de natureza semelhantes.

Considere por exemplo uma fila em uma agência bancária. O cliente entra na fila, depois é atendido pelo caixa e posteriormente sai da respectiva agência. O que se deseja descobrir neste problema é o número de funcionários necessários para atender os clientes sem ultrapassar o tempo máximo que um deles deve permanecer na fila.

Às vezes ultrapassar esse tempo implica em pagar uma multa, porém ao contratar mais atendentes aumenta-se o custo. Desse jeito, o problema de atendimento se desdobra em um problema de pesquisa operacional (CHWIF e MEDINA, 2006) que é descobrir o número de atendentes adequado para atendimento da lei ao menor custo total.

Outra aplicação seria de centro de atendimento ou *call centers*. O modelo de simulação é basicamente o mesmo, mas o objetivo seria aumentar a produtividade do atendente (taxa de utilização do servidor) sem aumentar o número de ligações perdidas (causadas normalmente pelo excesso no tempo de espera) (FREITAS FILHO et al, 2007).

O simulador SIPPO (em sua primeira versão) trata de forma integrada tanto os dados de entrada como o modelo de simulação e permite algumas modificações em um modelo mais básico como número de recursos ou forma de chegada das entidades. Dessa forma, pretende-se que um usuário iniciante em simulação possa modelar problemas de atendimento de forma mais detalhada que em um ambiente de simulação. Esta questão é levantada, pois na formação de um aluno, o entendimento de conceitos relativos à estrutura do software e compreensão detalhada do problema promove de certa forma, uma formação mais ampla e sólida.

Como exemplo de aplicação um modelo de atendimento é apresentado na Tabela 1 e Figura 6. Foi utilizada a linguagem IDEF-SIM (LEAL, ALMEIDA e MONTEVECHI, 2008) para representar o modelo conceitual do sistema. Isto porque o problema é facilmente modelado por diagrama de fluxos e permite um mapeamento do modelo conceitual e tradução para o de simulação de maneira mais prática e ágil.

Tabela 1: Tempo entre chegadas e tempo de serviço do modelo.

| **Tempo entre chegada de clientes (1 por vez)** | **Tempo de serviço do atendente** |
|---|---|
| Exponencial com média de 2 min. | Triangular com min. de 2 min., média de 4 min. e máx. de 10 min. |

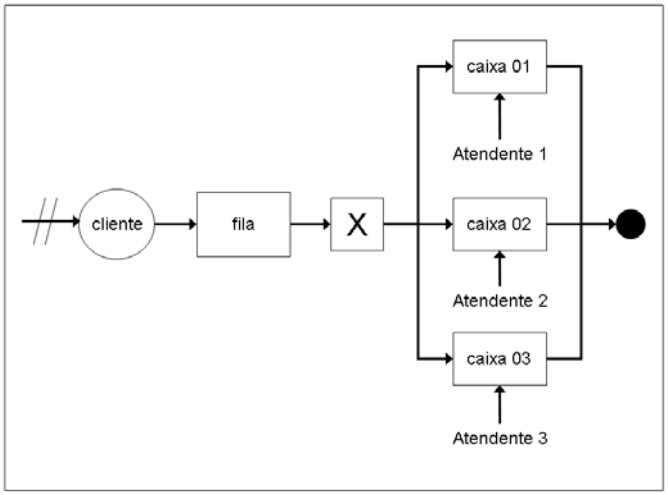


Figura 6: Modelo de simulação do Banco em IDEF-SIM. *Adaptado de Leal, Almeida e Montevechi (2008).*

A Figura 7 mostra o modelo de simulação em SIPPO 0.1 traduzido a partir do modelo conceitual mostrado na Figura 6. O mesmo modelo conceitual citado foi também traduzido para o Arena®12 (versão acadêmica) e Promodel®7.5 (versão acadêmica) para execução e comparação entre os resultados.

A Tabela 2 apresenta os resultados obtidos executando o modelo para um período de 6 horas (360 minutos) com 999 replicações em Arena®, ProModel® e SIPPO. Isto porque 999 é o número de replicações máximo suportado pelo ProModel®.

Tabela 2: Resultados do modelo de simulação.

| Simulador | Número de clientes na fila | Tempo na Fila (min) | Tempo de execução (s) | Tempo de desenvolvimento (min) |
|---|---|---|---|---|
| SIPPO | 4,18 | 8,07 | 6,2 | 30 |
| Arena® | 4,32 | 8,29 | 23,8 | 5 |
| ProModel® | 4,37 | 8,50 | 31,9 | 20 |

```
41    public static Model buildModel(){
42
43    // criar modelo
44    Model m = Model.createModel();
45    RandomVariable tcServidor = new RandomVariable(m,
46                                new Triangular(2,5,10), "processo Atendimento");
47    Resource r1 = new Resource(m, 3, "Atendente");
48    Queue q1 = new Queue(m, "Fila");
49
50    ProcessDescription mm1 = new ProcessDescription(m, "Modelo de Banco");
51    DistributionIfc tecCliente = new Exponential(2);
52
53    Variable v1 = new Variable(m, 1.0, "atendentes requesitados");
54    new EntityProcessGenerator(m, mm1, tecCliente, tecCliente);
55    mm1.addProcessCommand(new Process(m, v1, r1, q1,tcServidor));
56    mm1.addProcessCommand(new Terminate(m,true,"Fim do servico"));
57
58    return m;
59
60    }
```

Figura 7: Modelo do banco em SIPPO.

Observa-se que o SIPPO executou em um tempo consideravelmente menor (6,2s) do que os outros simuladores (23,8s no Arena® e 31,9s no ProModel®), já que o seu código está livre de vários recursos presentes nos ambientes de simulação. As simulações foram feitas em um computador com processador Core Duo 1.6Ghz com 3GB de memória RAM. Apesar do tempo de execução ser menor no SIPPO, o tempo de construção do modelo neste simulador foi maior, como é de se esperar.

De forma a se poder comparar melhor os resultados é mostrado na Tabela 3 o erro relativo entre os simuladores comerciais e o SIPPO. Observa-se um erro máximo de 5,33% (ProModel® em relação ao SIPPO).

Tabela 3: Erro relativo do SIPPO para os resultados.

| | **Erro Relativo (%)** | |
|---|---|---|
| **Simulador** | Número de clientes na fila | Tempo na Fila |
| Arena® | 3,35 | 2,73 |
| ProModel® | 4,55 | 5,33 |

A presença de um erro relativo em torno de 4% mostrado na Tabela 3 é aceitável em se tratando de sistemas estocásticos e com diferentes geradores de números aleatórios.

# 5. Discussão e Conclusões

Os primeiros resultados do SIPPO em comparação aos outros simuladores foram relativamente parecidos. Espera-se, assim, que este simulador possa contribuir para o panorama atual da simulação no Brasil.

Por possibilitar também a criação de novos comandos graças à biblioteca JSL, o SIPPO pode ser usado como simulador para outros modelos. Por outro lado, os desenvolvedores em Java que conheçam os conceitos de simulação discreta, além da possibilidade de desenvolver novos modelos, também podem contribuir para o desenvolvimento do SIPPO, pois se trata de um software de código aberto.

Futuramente, podem ser feitos testes com programadores com conhecimento básico em linguagem Java e programação orientada a objetos. Isto porque as bibliotecas que compõem o SIPPO têm todos os recursos básicos que um simulador de eventos discretos necessita ter bem como a modelagem da simulação ser orientada a fluxo de entidades.

Destaca-se também que a criação do SIPPO teve como objetivo principal o de contribuir para facilitar a difusão, o uso e a compreensão no Brasil da simulação desde a sua aplicação prática até a concepção interna de sua estrutura computacional e código fonte. Fica claro então que não se pretendeu aqui fazer comparações do software com os produtos comerciais do ponto de vista de capacidade em se resolver problemas. A questão está centrada na formação de pessoas aptas a poderem utilizar a simulação discreta e os softwares de simulação de forma mais sólida e ampla.

Vale ainda citar que as empresas que comercializam os softwares de simulação promovem diversos treinamentos para os seus produtos. No entanto, estes treinamentos muito diferem da formação que se espera de um profissional, que deve ser acompanhada de uma compreensão ampla e profunda do máximo de conceitos relativos à simulação discreta.

# 6. Obtenção do Software

O software pode ser obtido em http://bitbucket.org/tulioap/sippo. É recomendado utilizar o IDE NetBeans 6.8 para executar e desenvolver o SIPPO.

# Referências

**Banks, J.; Carson, J. S.; Nelson, B. L.; Nicol, D. M.** Discrete-event system simulation. 5nd ed. New Jersey: Prentice Hall, 2010.

**Bertie, A. J.** Java and interactive simulations for learning statistics. *In: CALRG (Computers and Learning Research Group) conference.* Institute of Educational Technology, Open University, 2002.

**Chwif, L.; Medina, A. C.** Uma análise crítica da Lei Municipal 13.948 ou Lei das Filas sob a ótica da Pesquisa Operacional: conclusões derivadas de modelos de simulação de eventos discretos. *In: XXVI Encontro Nacional de Engenharia de Produção*, 2006.

**Chwif, L.; Medina, A. C.** Modelagem e Simulação de Eventos Discretos: Teoria e Aplicações, São Paulo: Bravarte, 2006.

**Dahl, O.; Myhrhaug B.; Nygaard, K.** Common Base Language, Norwegian Computing Center, 1970.

**Freitas Filho, P. J.; Cruz, G. F.; Seara, R. e Steinmann G.** Using Simulation to Predict Market Behavior for Outbound Call Centers. *In: Winter Simulation Conference*, 2007.

**Gosling J.; Joy, B.; Steele G.; Bracha, G.** The Java language specification, third edition. Addison-Wesley, 2005

**Henriksen, J. O.; Crain, R. C.** GPSS/H: A 23-Year Retrospective View, *In Proceedings of the 2000 Winter Simulation Conference*, pp. 177-182, 2000.

**Jacobs, P.H.M.; Lang, N.A.; Verbraeck, A.** D-SOL: A distributed Java based discrete event simulation architecture. *In: Proceedings of the 2002 Winter Simulation Conference*, IEEE, San Diego, CA, 8-11 December, pp.793-800, 2002.

**Kay, A. C.** The early history of Smalltalk. *ACM SIGPLAN Notices*. Vol. 28, pp 69-95, 1993

**Leal, F.; Almeida, D. A. de; Montevechi, J. A. B.** Uma proposta de técnica de modelagem conceitual para a simulação através de elementos do IDEF. *In: Simpósio Brasileiro de Pesquisa Operacional*, João Pessoa-PB: Anais XL, 2008.

**Nance, R. E.** Simulation programming languages: an abridged history. *In: Proceedings of the 1995 Winter Simulation Conference*, pp. 1307-1313, 1995.

**Rosseti, M.D.** Java Simulation Library (JSL): an open-source object-oriented library for discrete-event simulation in Java. In: *Int. J. Simulation and Process Modelling*, Vol. 4, No. 1, pp.69-87, 2008.

**Verbraeck, A.** Real-time visualization and modeling of supply chains. *In: real-time: managing the new supply chain*. chapter 7. Praeger Publishers, Westport: CT, USA, 2004.