Recebido: 27/11/2010
Aprovado: 17/08/2012

Usando o JSL para simulação de Monte Carlo

Túlio Almeida Peixoto (UCAM - RJ/Brasil) - tulioap@gmail.com
• Av. Anita Peçanha, 100, Parque São Caetano, Campos dos Goytacazes-RJ, 28030-335
João José de Assis Rangel (UCAM - RJ/Brasil) - joao@ucam-campos.br
Ítalo de Oliveira Matias (UCAM - RJ/Brasil) - italo@ucam-campos.br

RESUMO O uso da Simulação de Monte Carlo para planejamento e análise de risco tem uso frequente em planilhas eletrônicas. Porém, há limitações no processamento de dados em uma planilha sem suplementos, como o número limitado de linhas e colunas e a impossibilidade de integração com sistemas de banco de dados de grande porte como usados em ERP (*Enterprise Resource Planning*). Neste trabalho, será apresentada a aplicação da biblioteca de código aberto JSL (*Java Simulation Library*) para Simulação de Monte Carlo, como alternativa às planilhas eletrônicas. Os testes para validação foram realizados com exemplos da literatura e apresentaram resultados compatíveis.

Palavras-chave Simulação; Simulação de Monte Carlo; JSL; Java.

ABSTRACT *The use of Monte Carlo Simulation for planning and risk analysis is frequently used in spreadsheets. However, there are limitations on data processing in a spreadsheet without add-ons, such as limited number of rows and columns as well as the impossibility of integration with large database systems as used in ERP (Enterprise Resource Planning). In this work, the application of JSL (Java Simulation Library) open-source library for the Monte Carlo Simulation will be presented as an alternative to spreadsheets. The tests for validation were performed with examples from the literature and they presented compatible results.*

Keywords *Simulation; Monte Carlo Simulation; JSL; Java.*

PEIXOTO, T. A.; RANGEL, J. J. A.; MATIAS, I. O. Usando o JSL para simulação de Monte Carlo. **GEPROS. Gestão da Produção, Operações e Sistemas**, Ano 7, nº 4, out-dez/2012, p. 135-152.

1. INTRODUÇÃO

De acordo com Banks *et al.* (2010): "simulação é a imitação de uma operação de um processo ou sistema real". A simulação é usada para descrever e analisar o comportamento de um sistema, podendo responder a questões do tipo "o que acontece se" sobre um determinado sistema e ainda auxiliar no projeto e implantação de sistemas. Dessa forma, a simulação, hoje, possibilita a realização de experimentos sem alterar a configuração real de uma planta de manufatura ou permite ainda avaliar a contratação prévia de funcionários para verificar se algum tipo de serviço deve atender a uma demanda futura, dentre outras aplicações que envolvam análise prévia.

O surgimento da simulação utilizando computadores se deu logo depois da Segunda Guerra Mundial (1945). A simulação foi usada no trabalho de Stanislaw Ulam, John Von Neumann e outros ao usar o método Monte Carlo para resolver certos problemas na difusão de nêutrons, no projeto da bomba de hidrogênio nos EUA (GOLDSMAN *et al.*, 2009). Posteriormente, este método teve seu uso estendido para análise de risco em empreendimentos e outros problemas com muitos graus de liberdade (BARRAQUAND; LATOMBE, 1990).

A história da simulação se confunde com a própria história da computação. Foi com a simulação que surgiram as primeiras linguagens orientadas a objeto, como Simula (Dahl *et al.*,1970), nos anos 60, e de linguagens com uso mais específico para simulação discreta, como GPSS (HENRIKSEN; CRAIN, 2000) e SIMAN (Nance, 1995). O ambiente de simulação *Rockwell Arena*®, por exemplo, originou-se da integração da linguagem SIMAN com o *software* CINEMA, que fazia as animações do modelo de simulação. Existem diversos outros ambientes para a construção de modelos de simulação, como *ProModel*® e *Simul8*® , que também são bastante comuns no Brasil em empresas que utilizam simulação.

Existem também bibliotecas de simulação que podem ser usadas para integrar simuladores de uso específico ou em construções de ambientes de simulação como o DSOL (JACOBS *et al.*, 2002) e o JSL (ROSSETTI, 2008). Ambas as bibliotecas são de código aberto, com licença GPL (*General Public Library*), possuem modificação de código livre e uso permitido para fins acadêmicos e comerciais, além de serem implementadas na linguagem Java.

Com o avanço da tecnologia em sistemas de computação nas últimas décadas, o custo e o tempo de processamento dos computadores diminuíram drasticamente (MOLLICK, 2006). Desse modo, o uso da simulação, que antes era restrito a grandes projetos e à aplicação como ferramenta de apoio à decisão no nível estratégico, passou a ser utilizado também no nível tático-operacional (BANKS, 2008). Dessa forma, a simulação tornou-se uma técnica mais popular, fazendo com que, muitas vezes, a palavra simulação se tornasse sinônimo de 'teste'. E essa associação à palavra 'teste' mostrou-se ser um bom sinal para representar o seu uso.

Diante do contexto apresentado, o objetivo deste trabalho é mostrar a biblioteca de simulação JSL (*Java Simulation Library*) em Java (GOSLING *et al.*, 2005) para a construção de modelos de Simulação de Monte Carlo proposta por Rossetti (2008). Também será realizada uma avaliação em relação ao tempo de desenvolvimento, resultados e velocidade de execução dos modelos desenvolvidos com o JSL e planilha eletrônica. Para isto, serão descritos os resultados comparativos com o problema do jornaleiro e análise quantitativa de riscos em um projeto de TI.

Os resultados do trabalho demonstraram que o desenvolvimento dos modelos utilizando a biblioteca de simulação a eventos discretos JSL atingiram resultados semelhantes a aqueles obtidos normalmente com planilhas. No entanto, o modelo em JSL apresentou vantagens na etapa de desenvolvimento e execução.

O trabalho foi organizado da seguinte forma; na segunda seção, a seguir, é apresentada a revisão da literatura. A terceira seção apresenta os componentes do JSL para a realização de uma simulação Monte Carlo. A quarta seção apresenta o desenvolvimento dos modelos de simulação. A quinta seção apresenta a discussão dos resultados. A sexta e última seção apresenta, então, as conclusões do trabalho.

2. REVISÃO BIBLIOGRÁFICA

2.1. Ambientes e Bibliotecas de Simulação

Pode-se caracterizar a simulação como sendo discreta, contínua, baseada em agentes e Monte Carlo. Um sistema pode ser modelado, também, fazendo uma combinação destas abordagens. Por exemplo, o processo de transferência de produtos para um tanque de processo em uma indústria química, utilizando uma frota de caminhões, pode ser modelado com simulação de eventos discretos. Já o controle das válvulas e fluxos de produto de um tanque de processo para outro pode ser modelado com simulação contínua (CHWIF; MEDINA, 2006).

Em uma simulação de eventos discretos, o sistema é representado como uma sequência cronológica e aleatória de eventos. Cada evento ocorre em determinado instante no tempo e marca a mudança de estado no sistema. A simulação de eventos discretos pode ser usada para simular desde um sistema de manufatura até um sistema de atendimento como um caixa de banco ou um *call center*. Exemplos de simuladores de eventos discretos são o *Rockwell Arena*® (KELTON *et al*., 2007) e *Promodel*® (HARREL *et al*., 2000). Estes últimos, devido à combinação de simulação e animação de modelos, são considerados também como ambientes de simulação.

A simulação contínua é usada quando o sistema muda seu estado em função do tempo, de maneira contínua. Tipicamente, o sistema é descrito sob forma de equações diferenciais. Este tipo de simulação é utilizado para simular circuitos elétricos, um sistema de reservatórios conectados por tubos e controlados por válvulas, como também simular problemas de difusão de calor e massa, dentre outros. Uma simulação de eventos discretos pode ser convertida de maneira aproximada em uma simulação contínua quando o número de entidades no sistema for relativamente grande para um curto intervalo de tempo (BANKS *et al*., 2010).

Já a Simulação de Monte Carlo usa geradores de números aleatórios para simular sistemas, onde não se considera o tempo explicitamente como variável. A Simulação de Monte Carlo pode ser usada desde a integração numérica de funções matemáticas como a simulação de riscos em um empreendimento. A Simulação de Monte Carlo normalmente pode ser realizada por meio de planilhas eletrônicas, por bibliotecas de *software* com pelo menos algum gerador de número aleatório. Um exemplo de simulador Monte Carlo é o suplemento *@Risk da Palisade*® para a planilha eletrônica *Microsoft Excel*® (BARRAQUAND; LATOMBE, 1990).

Uma forma de implementar modelos de simulação, além do uso de ambientes, é fazer uso de bibliotecas, seja para fazer um simulador específico ou adicionar a simulação como ferramenta de análise em um sistema de informação existente. Como exemplo, pode-se citar a biblioteca em Java DSOL (*Distributed Simulation Object Library*), que 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 que informações que estão distribuídas em vários sistemas de informação sejam utilizadas para alimentar o modelo de simulação. Para a simulação, por exemplo, de 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 tal qual o DSOL.

Outra biblioteca de simulação é a JSL (*Java Simulation Library*), que trabalha com simulação de eventos discretos e os modelos podem ser desenvolvidos como em um sistema de filas ou em uma abordagem mais detalhada por modelagem orientada a objetos. Um uso do JSL na construção de um ambiente de simulação pode ser visto em Peixoto *et al*. (2010) e este ambiente pode ser obtido pelo endereço http://bitbucket.org/tulioap/sippo. O JSL também tem componentes que são úteis para fazer simplesmente uma Simulação de Monte Carlo.

A facilidade por usar uma biblioteca de *software* em vez de planilhas é a possibilidade de fazer a Simulação de Monte Carlo em nível tático e operacional, ou seja, realizar a simulação através de uma grande massa de dados através de banco de dados em uso de uma empresa. Isto porque a linguagem Java suporta conexões com diversos tipos de banco de dados de uso comercial como o *Oracle*® ou *SQL Server*®, como também em bancos de dados de código aberto como o *PostgreSQL*, além de ser multiplataforma. Segundo L'Ecuyer (2001) e Apigian e Gambill (2004), o gerador de números aleatórios embutido em algumas planilhas eletrônicas não é tido como confiável.

2.2. Problema do Jornaleiro

O "problema do jornaleiro" é um problema clássico de gestão de estoques (CHWIF; MEDINA, 2006). Neste problema, um jornaleiro deve decidir quantos jornais deve comprar do seu fornecedor para atender seus clientes durante um domingo. Caso o jornaleiro compre mais jornais que o número verificado de clientes, a sobra de jornal será vendida para um serviço de reciclagem de papel por R$ 1,00 (por jornal). Caso contrário, se tiver mais clientes que jornal comprado, o jornaleiro deixa de ganhar o valor da venda (R$ 3,00) por cliente não atendido. Cada lote de 100 jornais é vendido por R$ 250,00 para o jornaleiro. Então, o lucro para este problema pode ser expresso pela expressão 1.

$$\text{Lucro} = 3 \min \times (\text{demanda,pedido}) \times 2{,}5 \text{ pedido} + 1 \times \max(0, \text{pedido} \times \text{demada}) \qquad (1)$$

Pela sua experiência vendendo jornais, o jornaleiro sabe que a distribuição do número de clientes que compram no domingo é discreta e pode ser representada pela Tabela 1.

Tabela 1 - Função densidade de probabilidade para o número de jornais vendidos no domingo.

Número de Clientes (demanda)	Probabilidade
100	0,10
150	0,25
200	0,35
250	0,20
300	0,10

Fonte: Chwif e Medina, 2006.

Para simular um domingo, sorteia-se uma demanda mostrada na Tabela 1. Por exemplo, considere que o jornaleiro realizou um pedido de 200 jornais para atender à demanda de domingo. Para descobrir o lucro médio no domingo, é feita a Tabela 2, com as demandas ocorridas e o lucro obtido para 10 "domingos", ou 10 "replicações".

Tabela 2 – Demanda e lucro obtidos a cada domingo para um pedido de 200 jornais.

Domingo	Demanda ocorrida	Lucro
1	200	R$ 100,00
2	150	R$ 0,00
3	200	R$ 100,00
4	100	R$ -100,00
5	300	R$ 100,00
6	150	R$ 0,00
7	200	R$ 100,00
8	250	R$ 100,00
9	150	R$ 0,00
10	100	R$ -100,00
	Lucro médio:	R$ 30,00
	Desvio padrão:	R$ 82,33

Fonte: Adaptado de Chwif e Medina (2006).

Pelo resultado obtido na Tabela 2, ainda não se pode tirar conclusões sobre o resultado da simulação, pois em 10 replicações ela se encontra em um regime transitório (o desvio padrão está relativamente alto). O resultado final para uma simulação com 10.000 replicações para cada tipo de pedido (100, 150, 200, 250 e 300) pode ser visto na Tabela 3. O resultado é que o lucro máximo é alcançado com a compra do lote de 150 jornais (R$ 66,81) no dia de domingo.

Tabela 3 – Lucro médio obtido em função do número de jornais adquiridos.

Lote	Lucro Médio Obtido
100	R$ 50,00
150	R$ 66,81
200	R$ 56,44
250	R$ 7,62
300	- R$ 51,30

Fonte: Adaptado de Chwif e Medina (2006).

2.3. Análise Quantitativa de Risco

Outra aplicação onde se pode usar Simulação de Monte Carlo é a análise quantitativa de risco. Conforme o PMBOK (PMI, 2004), um risco de projeto é definido como uma incerteza em função de um evento ou condição que, se ocorrer, terá um impacto positivo ou negativo sobre os objetivos do projeto. Existem dois tipos gerais de riscos: conhecidos e não conhecidos. Os riscos conhecidos são aqueles identificados e analisados. Já os riscos desconhecidos não podem ser gerenciados de forma pró-ativa, o que normalmente obriga gerentes de projeto a alocar contingências para estes casos.

Em seguida, será apresentado um exemplo de aplicação da Análise Quantitativa de Risco (AQR) contextualizado em um projeto na área de TI. Mais detalhes sobre este exemplo podem ser encontrados em Matias Jr. (2006).

O problema é saber quantos dias são necessários para que um servidor fique disponível para realizar um pacote de trabalho do projeto. Inicialmente, foi realizada uma entrevista com o departamento de importação do fornecedor, a fim de entender o seu processo de importação de equipamentos similares ao recurso do projeto. O processo foi dividido em quatro etapas: (1) gerar ordem de compra; (2) despacho na fábrica; (3) trâmites alfandegários; e (4) transporte local. Neste caso, optou-se por inserir no modelo quatro variáveis aleatórias, cada qual representando uma destas etapas. Portanto, o tempo total do processo de entrega do equipamento foi representado pela soma das quatro variáveis. A partir das entrevistas, a estratégia de modelagem adotada considerou que todas as variáveis seguem um comportamento compatível com a distribuição de probabilidade triangular. Esta distribuição é uma das mais usadas para modelar opinião de especialistas (VOSE, 2002). Os parâmetros de cada função de distribuição estão apresentados na Tabela 4.

Tabela 4 – Tempo de cada processo do fornecedor do recurso.

Processos do Fornecedor	Tempo (dias)
Gerar ordem de compra	Triangular (1; 2; 4)
Despacho na fábrica	Triangular (1; 2; 3)
Trâmites alfandegários	Triangular (7; 20; 40)
Transporte local	Triangular (3; 5; 12)

Fonte: dados da pesquisa.

Ficou acordado que a consultoria responsável pela implementação do projeto tem remuneração de aproximadamente R$ 800,00/dia. O contrato não isenta períodos de inatividade em função de atrasos no provimento de recursos por parte do contratante. Com a simulação depois de executada, atingiu-se uma média de 33,33 dias de tempo total para a entrega do recurso solicitado. A Tabela 5 apresenta seis cenários com seus respectivos impactos sobre o orçamento do projeto.

Tabela 5 – Cenários de efeitos do risco sobre o custo do projeto.

Cenário	Tempo de entrega	Probabilidade de ocorrência	Atraso em dias	Impacto no orçamento (R$)
1º	até 20 dias	1,8%	0	0
2º	até 25 dias	12,9%	5	4.000
3º	até 30 dias	34,4%	10	8.000
4º	até 35 dias	59,1%	15	12.000
5º	até 40 dias	81,1%	20	16.000
6º	até 45 dias	94,2%	25	20.000

Fonte: Adaptado de Matias Jr. (2006).

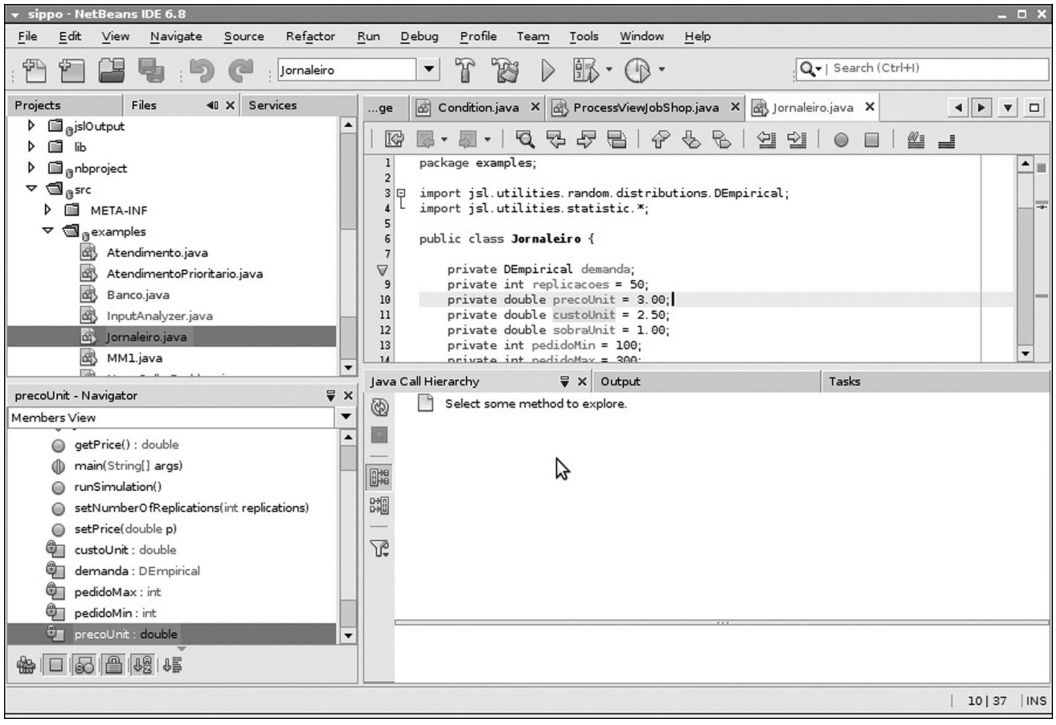
Conforme as estimativas da simulação realizada, o gerente de projeto possui subsídios quantitativos para fundamentar suas decisões acerca do risco analisado. Um gerente mais conservador possivelmente estaria levando em conta um cenário menos otimista (5º ou 6º). Já gerentes mais arrojados poderiam adotar cenários intermediários (3º ou 4º) por estarem próximos ao valor médio da simulação realizada.

3. COMPONENTES DO JSL PARA SIMULAÇÃO DE MONTE CARLO

Neste trabalho, a biblioteca JSL foi escolhida por ter todos os componentes essenciais para um simulador, tais como: gerador de números aleatórios, distribuições de probabilidade e tratamento dos dados de saída. A biblioteca também é extensível e multiplataforma, por ser escrita em Java, que é orientada a objetos e executa em várias plataformas.

Para obter o JSL, basta ir ao site http://bitbucket.org/rossetti/jsl e baixar o código fonte. Já para o desenvolvimento dos projetos em JSL, pode ser utilizado o ambiente de desenvolvimento integrado *NetBeans* 6.8, como mostrado na Figura 1. Ele facilita o desenvolvimento do código em Java através de um editor com coloração de sintaxe, integração com compilador e depurador de código. Desenvolvedores em Java geralmente são habituados a utilizar ambientes integrados como este. O *NetBeans* e o JSL são projetos de código aberto com licença GPL (*General Public Licence*). Esta licença pode ser encontrada em www.gnu.org/licenses/gpl.html e uma cópia do *NetBeans* pode ser obtida em www.netbeans.org.

Figura 1 - Ambiente de desenvolvimento *NetBeans*.



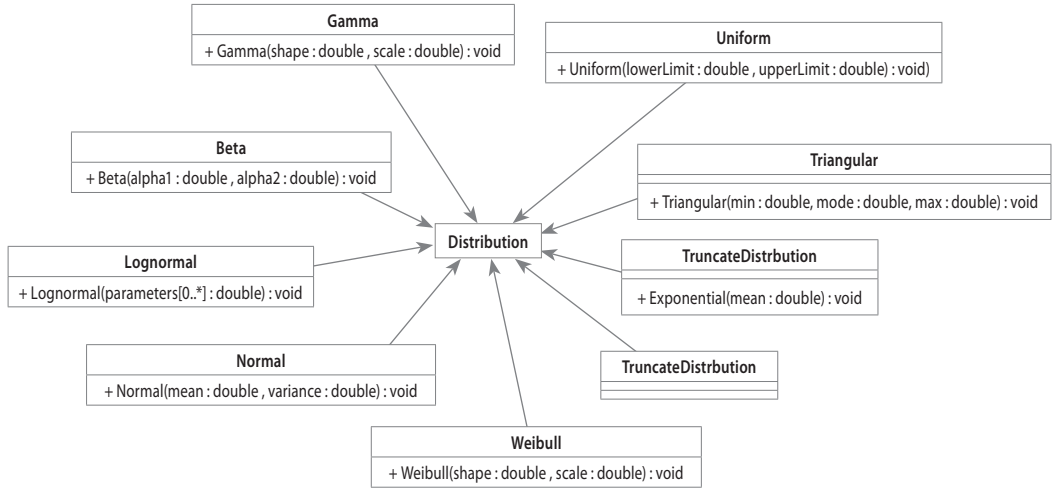
Fonte: Elaboração dos autores.

Para fazer a simulação de Monte Carlo, de uma forma geral, é preciso seguir os seguintes passos:

- Definir as variáveis de entrada;
- Gerar observações ou replicações de alguma distribuição de probabilidades para cada entrada;
- Fazer um cálculo em função das observações obtidas;
- Agregar os resultados de cada cálculo em um resultado final.

No caso da Simulação de Monte Carlo utilizando o JSL, basicamente são usadas as classes que representam as distribuições de probabilidade do pacote *Random* e as classes do pacote *Statistic* para coletar estatísticas sobre as replicações. Distribuições de probabilidade como normal e exponencial estão representadas pelo diagrama de classes UML - *Unified Modeling Language*, mostrado na Figura 2. Para exemplificar, a distribuição Normal é definida pela classe Normal (Figura 2), que tem o método com o mesmo nome e este recebe como parâmetros a média (*mean*), que é do tipo *double* e variância (*variance*), que também é do tipo *double*. As demais classes que representam as outras funções podem ser utilizadas da mesma forma.

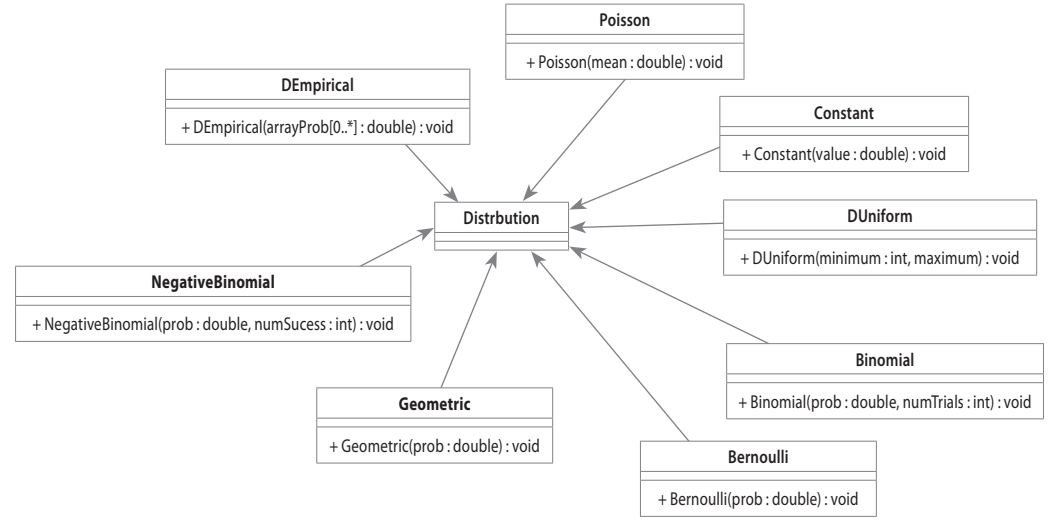
Figura 2 – Distribuições Contínuas em JSL.



Fonte: Elaboração dos autores.

Já na Figura 3 está o diagrama de classes das distribuições discretas. Como exemplo, de forma semelhante, a distribuição de *Poisson* é definida pela classe *Poisson* mostrada na Figura 3, que tem o método com o mesmo nome e este recebe como parâmetro a variável *mean*, que é do tipo *double*.

Figura 3 – Distribuições Discretas em JSL.

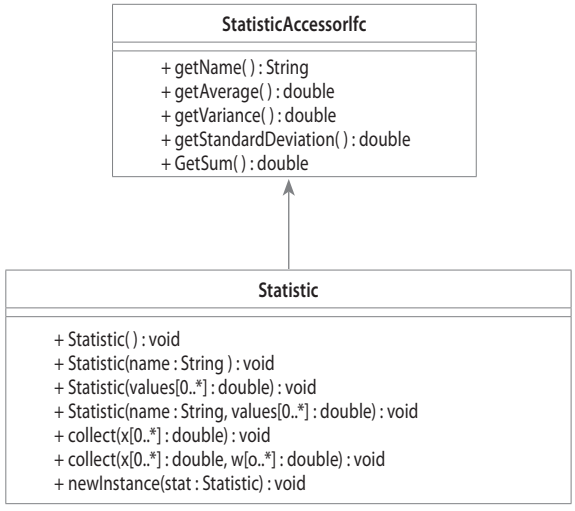


Fonte: Elaboração dos autores.

O método mais importante da classe *Statistic*, mostrado na Figura 4, é o *collect*, que é responsável por coletar o resultado das observações e posteriormente processar a estatística descritiva (média e desvio padrão) de todo conjunto de dados. Outro método também importante é o construtor *Statistic* que, além de simplesmente inicializar um objeto sem número nenhum, ainda coletado na sua versão sem parâmetros (1º e 2º métodos), pode também inicializar um objeto com valores do tipo double de uma só vez (3º e 4º métodos). A classe *newInstance* é usada para fazer reflexão de classes quando de alguma maneira não é possível usar o operador *new* para usar o método construtor *Statistic*.

Além disso, na mesma figura, a classe *Statistic* herda da interface *StatisticAccessorIfc* os métodos como média amostral (*getAverage*), variância (*getVariance*), desvio padrão (*getStandardDeviation*), dentre outros métodos usados na estatística descritiva.

Figura 4 – Classe *Statistic* e sua interface (*StatisticAccessorIfc*) em JSL.



Fonte: elaboração dos autores.

4. DESENVOLVIMENTO DOS MODELOS DE SIMULAÇÃO

Os modelos de simulação foram desenvolvidos em JSL, orientados pelo método proposto por Banks *et al.* (2010), seguindo os seguintes passos específicos para este trabalho. São eles: formulação e análise do problema; planejamento do projeto; modelo conceitual; tradução do modelo conceitual; verificação e validação; experimentação; interpretação e análise de resultados. As simulações somente foram iniciadas após a verificação e validação do modelo, garantindo que os pressupostos e as hipóteses estavam corretamente implementados nos modelos conceitual e computacional (SARGENT, 2007).

4.1. Modelo do Problema do Jornaleiro

Primeiro passo: definir as variáveis de entrada.

Conforme a sessão 4, o problema somente tem uma variável de entrada, que é a demanda de jornais em um dia de domingo. Além disso, a demanda é uma variável aleatória.

Segundo passo: definir as distribuições de probabilidade para cada variável.

A variável de demanda segue uma distribuição de probabilidade discreta empírica. A Figura 5 ilustra como definir a Tabela 1 em termos dos componentes do JSL. A linha 27 (Figura 5) adiciona o ponto (100, 10%) à função de probabilidade empírica (primeira linha da Tabela 1). Já a linha 28 adiciona o ponto (150, 25%) à função correspondendo à segunda linha da Tabela 1. Vão se adicionando os pontos até chegar à demanda de 300 clientes, que corresponde ao ponto (300, 10%) da última linha da Tabela 1. Como ele é o último ponto a ser definido, não é necessário definir a probabilidade que o valor ocorre, pois o somatório de todos os pontos obrigatoriamente é um.

Figura 5 - Distribuição empírica da demanda de clientes em JSL.

```
24      public Jornaleiro() {
25
26          demanda = new DEmpirical();
27          demanda.addProbabilityPoint(100.0, 0.10);
28          demanda.addProbabilityPoint(150.0, 0.25);
29          demanda.addProbabilityPoint(200.0, 0.35);
30          demanda.addProbabilityPoint(250.0, 0.20);
31          demanda.addLastProbabilityPoint(300.0);
32
33      }
```

Fonte: Dados da pesquisa.

Terceiro passo: fazer um cálculo em função das observações obtidas.

Este passo é definir como acontecerá a simulação. A Figura 6 mostra o código em JSL. O laço mais interno é para fazer a simulação para um número arbitrário de replicações. A cada iteração deste laço é sorteada uma demanda (linha 53) e calculado o lucro em função desta demanda (linhas 54 e 55). Finalmente, o lucro é registrado (linha 56) para se obter a média no final (linha 59). Já o laço mais externo é para fazer a simulação para cada tipo de pedido em separado. O valor do pedido varia de 100 a 300 jornais, em 5 tipos de lote de jornais.

Figura 6 - Cálculo do lucro e simulação em JSL.

```
47      public void runSimulation() {
48
49          Statistic profitStat = new Statistic("Lucro");
50          double lucro = 0;
51          for (int p = pedidoMin; p <= pedidoMax; p = p + 50) {
52              for (int k = 1; k <= this.replicacoes; k++) {
53                  double d = demanda.getValue();
54                  lucro = precoUnit*Math.min(d, p) - custoUnit*p
55                          + sobraUnit*Math.max(0, p - d);
56                  profitStat.collect(lucro);
57              }
58              System.out.println("Estatisticas para o pedido = "+ p);
59              System.out.println(profitStat.getAverage());
60              profitStat.reset();
61          }
62      }
```

Fonte: Dados da pesquisa.

Quarto passo: agregar os resultados de cálculo em um resultado final.

No caso do problema do jornaleiro, o que deve se aferir é o maior lucro obtido em função da demanda de jornais vendidos. Agregar este resultado é fazer uma média das replicações para cada tipo de lote pedido pelo jornaleiro, que pode ser de 100 a 300 jornais, variando de 50 a 50. Esta média é feita acumulando o valor do lucro para cada replicação (linha 56). Exibe-se, então, a média (linhas 58 e 59), mostrada na Figura 7.

Definem-se também os parâmetros do experimento, que são o número de replicações (10.000 replicações) e o preço unitário do jornal. O número de replicações foi o mesmo de Chwif e Medina (2006), para fins de comparação.

Figura 7 - Definição dos parâmetros do experimento e simulação.

```
53      public static void main(String[] args) {
54          Jornaleiro j = new Jornaleiro();
55          System.out.println("Preco $3.00");
56          j.setNumberOfReplications(10000);
57          j.setPrice(3.00);
58          j.runSimulation();
59      }
```

Fonte: Dados da pesquisa.

4.2. Modelo para Análise Quantitativa de Risco

Primeiro passo: definir as variáveis de entrada.

Conforme a sessão 5, o problema tem quatro variáveis de entrada, cada uma correspondendo a um processo do fornecedor: Gerar ordem de compra, despacho na fábrica, trâmites alfandegários e transporte local. Cada variável de entrada é independente entre si. Isto é importante para a realização da Simulação de Monte Carlo.

Segundo passo: definir as distribuições de probabilidade para cada variável.

Todas as variáveis de entrada seguem distribuição triangular com os parâmetros definidos na Tabela 5. Em termos de JSL, usa-se a classe Triangular, que tem três parâmetros: valor mínimo, moda e valor máximo, respectivamente, conforme a Figura 8.

Figura 8 – Definição das variáveis de entrada em JSL.

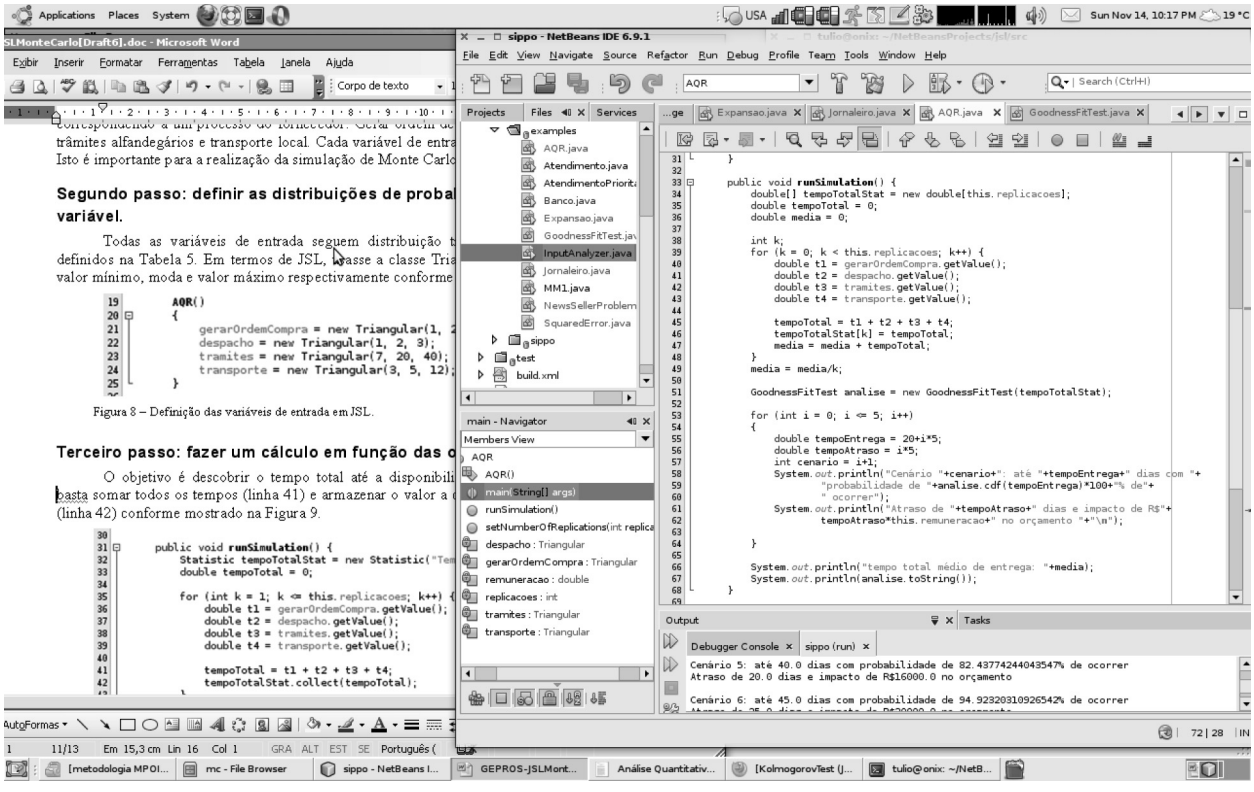
```
20      AQR()
21      {
22          gerarOrdemCompra = new Triangular(1, 2, 4);
23          despacho = new Triangular(1, 2, 3);
24          tramites = new Triangular(7, 20, 40);
25          transporte = new Triangular(3, 5, 12);
26          remuneracao = 800;
27      }
```

Fonte: Dados da pesquisa.

Terceiro passo: fazer um cálculo em função das observações obtidas.

O objetivo é descobrir o tempo total até a disponibilidade do servidor. Neste caso, basta somar todos os tempos (linha 45) e armazenar o valor a cada replicação ou observação (linha 46), conforme mostrado na Figura 9.

Figura 9 – Cálculo do tempo total de entrega e simulação em JSL.



Fonte: Dados da pesquisa.

Quarto passo: agregar os resultados de cálculo em um resultado final.

Neste caso deseja-se saber, além da média do tempo total de entrega, o risco de atraso para 0, 5, 10, 15, 20 e 25 dias. A média, já calculada na linha 49, é exibida na linha 66 na Figura 9. O risco para cada cenário é obtido através da função de probabilidade cumulativa (linha 59). A distribuição que melhor adere à série de observações do tempo de entrega foi a Normal com o auxilio do comando *GoodnessFitTest* da linha 51. Este método foi feito com o auxílio da biblioteca JSC - *Java Statistical Classes*, proposto por Bertie (2002).

Definem-se também os parâmetros do experimento, que são os números de replicações (5.000 replicações). O número de replicações foi o mesmo de Matias Jr. (2006) para fins de comparação. Este código é apresentado na Figura 10.

Figura 10 - Definição dos parâmetros do experimento e simulação.

```
47
48      public static void main(String[] args) {
49          AQR a = new AQR();
50          a.setNumberOfReplications(5000);
51          a.runSimulation();
52      }
53
54  }
```

Fonte: Dados da pesquisa.

5. DISCUSSÃO DOS RESULTADOS

5.1. Resultados do Problema do Jornaleiro

Os resultados da simulação em JSL foram bem próximos aos resultados de Chwif e Medina (2006), como mostrado na Tabela 6. Pode-se verificar que a diferença do lucro máximo obtido no JSL (R$ 65,22) com a simulação anterior foi de 2,37%. Esta diferença pode ser considerada aceitável, em se tratando de sistemas estocásticos e com geradores de número aleatórios diferentes.

Tabela 6 – Lucro médio obtido em função do número de jornais adquiridos.

Lote	Lucro Médio Obtido	Lucro Médio Obtido (JSL)
100	R$ 50,00	R$ 50,00
150	R$ 66,81	R$ 65,22
200	R$ 56,44	R$ 55,85
250	R$ 7,62	R$ 9,85
300	-R$ 51,30	-R$ 56,61

Fonte: Dados da pesquisa.

Além disso, o uso do JSL demonstrou-se mais conveniente que o uso somente de planilhas neste caso. Note que, para codificar o problema, foram necessários em torno de 70 linhas de código Java e em planilha eletrônica sem suplementos seria necessário usar cerca de 50 mil linhas, já que são necessárias 10.000 replicações para cada um dos 5 tipos de demanda (Tabela 1). Dessa forma, o tempo de desenvolvimento ficou maior (uma hora a mais) quando o problema foi definido usando planilhas, além de ser mais inconveniente mudar o número de replicações, já que na planilha foi preciso adicionar ou remover várias linhas. O tempo de simulação foi de 1 segundo em ambas as ferramentas considerando que a máquina foi um Intel Core 2 Duo® 1,6 GHz com 3GB de memória RAM.

5.2. Resultados da Análise Quantitativa de Riscos

Os resultados da simulação em JSL foram bem próximos aos resultados de Matias Jr. (2006) e estão apresentados na Tabela 7. Pode-se verificar que a diferença do tempo de entrega obtido no JSL (33,39) com a simulação anterior foi de 0,18%. Esta diferença pode ser considerada aceitável em se tratando de sistemas estocásticos e com geradores de número aleatórios diferentes.

Tabela 7 – Comparação do tempo de entrega do recurso conforme Matias Jr. (2006) *versus* JSL.

Tempo de entrega	Tempo de entrega (JSL)
33,33 min.	33,39 min.

Fonte: Dados da pesquisa.

Para a análise dos riscos, a distribuição que mais aderiu foi a normal. Porém, isto não alterou de maneira geral os resultados alcançados por Matias Jr. (2006). Observa-se na Tabela 7 que continua uma distinção entre os cenários mais pessimistas (5º e 6º) e otimistas (3º e 4º), sendo que a média ainda está entre o 3º e 4º cenários.

Tabela 8 – Cenários de efeitos do risco sobre o custo do projeto pelo JSL.

Cenário	Tempo de entrega	Probabilidade de ocorrência	Atraso em dias	Impacto no orçamento (R$)
1.º	até 20 dias	2,9%	0	0
2.º	até 25 dias	11,8%	5	4.000
3.º	até 30 dias	31,6%	10	8.000
4.º	até 35 dias	58,9%	15	12.000
5.º	até 40 dias	82,4%	20	16.000
6.º	até 45 dias	94,9%	25	20.000

Fonte: Dados da pesquisa.

Foi feita também a mesma simulação utilizando planilhas eletrônicas. A primeira dificuldade encontrada é que não havia uma fórmula pré-definida para usar a distribuição triangular. Assim, foi definida uma fórmula para a geração de observações para tal distribuição. Foram usadas 5000 linhas da planilha (sem suplementos), uma para cada replicação. O tempo de simulação foi de 1 segundo em ambas as ferramentas (planilha e biblioteca), considerando que a máquina foi um Intel Core Duo® 1,6GHz com 3GB de memória RAM.

5.3. Análise dos Resultados

Neste trabalho, foi apresentado o uso de uma biblioteca de simulação em Java para uso específico, no caso, a Simulação de Monte Carlo para problemas conhecidos na literatura, como o problema do jornaleiro e de análise quantitativa de riscos. Foi demonstrada uma forma alternativa do método mais tradicional, com uso de planilhas sem o suporte de *softwares* adicionais.

Foi demonstrado que o desenvolvimento dos modelos utilizando o JSL atingiu resultados semelhantes a aqueles reproduzidos em planilhas. Por outro lado, o tempo de desenvolvimento do algoritmo foi comparável à elaboração de planilhas. Isto ocorreu, pois o número de linhas de código foi relativamente reduzido (algumas dezenas de linhas) comparado ao número de linhas em planilhas eletrônicas que fica na ordem de milhares devido ao número de replicações. Além disso, o tempo de simulação ficou na ordem de segundos apesar do número de replicações ser da ordem de milhares de replicações em um computador com um processador de 1,6GHz e 3GB de memória.

Outra vantagem de se usar uma linguagem de programação como Java, ao contrário de usar somente planilhas, é o fato que Java permite integração com banco de dados de diversos tipos, usando a interface JDBC. Assim, um mesmo sistema pode ler as observações reais e elaborar as distribuições teóricas ou empíricas que são utilizadas na simulação, fazer a simulação e, com o resultado desta simulação, atuar no sistema de forma a fechar o ciclo de obtenção e análise de dados de forma automática.

6. CONCLUSÕES

Foi demonstrada neste trabalho a aplicação da biblioteca de simulação a eventos discretos JSL para a construção de modelos de Simulação de Monte Carlo. Os resultados confirmaram a viabilidade para a aplicação proposta. Para este fim, foi realizada avaliação em relação ao tempo de desenvolvimento, resultados e velocidade de execução dos modelos desenvolvidos com o JSL e planilha eletrônica utilizando problemas conhecidos da literatura como o problema do jornaleiro e análise quantitativa de riscos em um projeto de TI.

Por fim, espera-se que a ferramenta apresentada neste trabalho possa contribuir para ampliar a utilização da simulação tanto no campo acadêmico quanto no profissional. No campo acadêmico, para difundir os conceitos de Simulação de Monte Carlo em cursos de graduação de engenharia e computação. Já no campo profissional, a biblioteca pode incluir aspectos de análise a sistemas que somente cuidariam de questões operacionais como um *software* de ERP – *Enterprise Resource Planning* que opera em servidores *web*.

REFERÊNCIAS BIBLIOGRÁFICAS

APIGIAN, C. H.; GAMBILL, S. E. Is Microsoft Excel 2003 Ready for the Statistics Classroom. **Journal Of Computer Information Systems**, n. 45, p.27-35, 2004.

BANKS, J.; CARSON, J. S.; NELSON, B. L.; NICOL, D. M. **Discrete-event system simulation**. 5nd ed. New Jersey: Prentice Hall, 2010.

BANKS, J. Some Burning Questions about Simulation. ICS Newsletter: **INFORMS Computing Society**, Spring, n. , p.11-14, 2008.

BARRAQUAND, J.; LATOMBE, J. A Monte-Carlo algorithm for path planning with many degrees of freedom. *In*: IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATIO, Cincinnati. **Proceedings…** p. 1712 - 1717, 1990.

BERTIE, A. J.; LATOMBE, J. Java and interactive simulations for learning statistics planning with many degrees of freedom. *In*: CALRG (COMPUTERS AND LEARNING RESEARCH GROUP) CONFERENCE, 2002. Open University. **Proceedings…** 2002.

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

DAHL, O.; MYHRHAUG B.; NYGAARD, K. **SIMULA 67:** Common Base Language. Norwegian Computing Center, Oslo, 1970.

GOLDSMAN, D.; NANCE, R. E.; WILSON, J. R. A brief history of simulation from 1777 to 1981. *In*: **Proceedings of the 2009 Winter Simulation Conference**, Austin, 2009, p. 310 - 322.

GOSLING, J.; *et. al*. **The Java language specification**, third edition. Addison-Wesley, 2005.

HARREL, C.; GHOSH, B.; BOWDEN, R. **Simulation using ProModel**. New York: McGraw-Hill, 2000.

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

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**, 2002, San Diego, p. 793 - 800.

KELTON, W. D.; SADOWSKI, R. P. E STURROCK, D.T. **Simulation with Arena**. Forth Edition, New York: McGraw-Hill, 2007.

L'ECUYER, P. **Software for Uniform Random Number Generation: Distinguishing the Good and the Bad**. IEEE Press, p. 95-105, Dec. 2001.

MATIAS JR., R. Análise Quantitativa de Risco Baseada no Método de Monte Carlo: Abordagem PMBOK. *In*: **I CONGRESSO BRASILEIRO DE GERENCIAMENTO DE PROJETOS**, Florianópolis, 2006.

MOLLICK, E. Establishing Moore's Law. **IEEE Annals of the History of Computing** v. 28, n. 3, p. 62-75, 2006.

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

PEIXOTO, T. A.; RANGEL, J. J. A.; MATIAS, I. O. SIPPO 0.1 – Simulador Para Problemas de Pesquisa Operacional. *In*: XLII SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, Bento Gonçalves, **Anais…** 2010.

PMI. **Guide to the project management body of knowledge**. Edition 3. USA: Project Management Institute, p. 388, 2004.

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

SARGENT, R. G. Verification and Validation of Simulation Models. *In*: **PROCEEDINGS OF THE WINTER SIMULATIONS CONFERENCE**, Baltimore, EUA, 2010, p. 1-10.

VERBRAECK, A. Real-time visualization and modeling of supply chains. *In*: **Real-time:** managing the new supply chain. Praeger Publishers. Westport, 2004.

VOSE, D. **Risk Analysis: a quantitative guide**. 2 Edition. UK: John Wiley & Sons, 2002. p. 418.