Acta Scientiarum 21(4):855-860, 1999.
ISSN 1415-6814.

SDD: simulação discreta em Delphi

Ademir Aparecido Constantino* e Ademir Franchini Junior

Departamento de Informática, Universidade Estadual de Maringá, Av. Colombo, 5790, 87020-900, Maringá-Paraná, Brazil.
**Author for correspondence. e-mail: ademir@din.uem.br*

RESUMO. O objetivo deste trabalho é apresentar um simulador desenvolvido em Delphi 4.0 para o ensino e aplicação de simulação discreta. Ele foi desenvolvido em princípio, para atender o curso de Ciência da Computação da UEM. O simulador, denominado SDD (Simulação Discreta em Delphi), é constituído de uma biblioteca de objetos e de funções, fornecendo suporte para representação e manipulação de entidades, filas, recursos e eventos. A biblioteca foi implementada em *Object Pascal* e deve ser usada em conjunto com o Delphi 4.0. O SDD possui uma interface gráfica que permite ao usuário atribuir valores aos parâmetros da simulação, executar a simulação e visualizar os resultados através de gráficos e de dados estatísticos, além de gerar relatórios. O SDD permite que os modelos sejam implementados através de dois métodos: duas-fases e três-fases. Dessa forma, o usuário poderá implementar os modelos de simulação, experimentando diferentes paradigmas de modelagem.

Palavras-chave: simulação discreta, modelagem, linguagem de simulação.

ABSTRACT. SDD: Discrete simulation using Delphi. The aim of this paper is to show a simulation system developed using Delphi 4.0 software for teaching and application of discrete simulation. It was developed to attend the Computer Science course in the State University of Maringá. The simulator, named SDD, is constituted of a library of objects and functions supporting the representation and manipulation of entities, queues, resources and events. The library was developed in Object Pascal and must be used with Delphi 4.0. The SDD has a graphic interface which enables the user to configure the simulation parameters, run the simulation, view the results by graphics and statistical data, and generate reports. Furthermore, the user may use the SDD to build event orientation based on models according to two approaches: two-phase and three-phase. Therefore, the user will be able to build simulation models by different approaches.

Key words: discrete simulation, modelling, language.

O termo simulação tem um significado bastante amplo dentro da comunidade científica. Dependendo da área na qual está sendo utilizada, os conceitos, as metodologias e as ferramentas computacionais podem ser completamente diferentes e incompatíveis. Este trabalho trata de simulação como ferramenta de Pesquisa Operacional, ou seja, de simulação discreta estocástica dinâmica, ou resumidamente, simulação discreta.

Existem diversas linguagens específicas de simulação que podem ser utilizadas para implementação de modelos de simulação discreta, tais como, *Siman* (Pegden *et al.*, 1990) e GPSS/H. As dificuldades na utilização dessas linguagens no ensino de simulação residem no fato de serem linguagens de custo elevado e de requererem que o usuário aprenda uma linguagem de programação específica só para simulação. Hoje, algumas dessas linguagens foram absorvidas por programas de simulação visual, tais como, Arena (Kelton *et al.*, 1998), Promodel, Taylor II, etc. Esses são ambientes gráficos que permitem ao usuário construir seu modelo através de ícones e de suas interações (*links*).

Uma outra alternativa é utilizar uma linguagem de propósito geral, tais como, Pascal, C, Fortan, etc. As vantagens de utilizar essas linguagens são o baixo custo (em relação às linguagens específicas de simulação) e o fato de serem bastante flexíveis e conhecidas nos meios acadêmicos. Por outro lado, essas linguagens não fornecem nenhum suporte específico para facilitar a implementação de modelos de simulação, tornando a tarefa de implementação complexa e altamente custosa. Na literatura, são

encontradas diversas ferramentas para simulação desenvolvidas em Pascal e C. Um trabalho mais recente e que mais se aproxima deste é o *Simul* (Saliby e Pimentel, 1992). O *Simul* é uma ferramenta para simulação discreta desenvolvida em Turbo Pascal. Embora seja uma ferramenta eficiente e robusta, ela foi desenvolvida em Pascal estruturado e a codificação de modelos é uma tarefa bastante custosa.

Segundo Smith (1989), uma linguagem para ensino de simulação deve dar suporte para trabalhar com pelo menos três áreas:

- modelagem de sistemas;
- projeto e análise de experimentos de simulação;
- desenvolvimento de programas (estruturas e algoritmos) de simulação.

No meio universitário, existem estudantes com perfis diferentes; para cada tipo, deve ser adotada uma ou mais dessas áreas. Por exemplo, os estudantes de estatística normalmente se interessam por projeto e análise de experimentos de simulação; já os estudantes de engenharia de produção freqüentemente se interessam pela modelagem de sistemas; os estudantes de ciência da computação preocupam-se com o desenvolvimento de programas de simulação; e finalmente, os estudantes de pesquisa operacional devem trabalhar com todas essas áreas. Este artigo apresenta uma ferramenta que dá suporte para trabalhar as três áreas.

Além das dificuldades supracitadas, as linguagens existentes não proporcionam trabalhar os diferentes paradigmas dentro do mesmo ambiente de programação. O objetivo deste trabalho, portanto, é apresentar uma ferramenta que pode auxiliar no ensino e na aplicação de simulação discreta, através de dois métodos: duas-fases e três-fases. Esta ferramenta, denominada SDD (Simulação Discreta em Delphi), apresenta caraterísticas inovadoras, tais como: implementação orientada a objetos e uma interface gráfica que facilita a experimentação de modelos. Os aspectos importantes do simulador são: simplicidade de implementação de modelos, eficiência e robustez.

Paradigmas de simulação

Este trabalho aborda somente a simulação discreta, isto é, a simulação de sistemas que se alteram em pontos discretos no tempo. Dentro deste contexto, são encontradas terminologias como:

1. **Entidades:** denotam pessoas, objetos ou coisas que se movem pelo sistema, causando mudanças. As entidades são de dois tipos: as permanentes e as temporárias. As entidades permanentes são criadas no início da simulação e permanecem até o final do experimento, enquanto as entidades temporárias entram e saem do sistema a qualquer momento.
2. **Recursos:** é uma maneira de representar objetos que serão usados por entidades, mas que não necessitam ser identificados no mesmo nível de detalhes que as entidades. Por exemplo, as ferramentas em uma oficina mecânica.
3. **Fila:** representa o estado ocioso das entidades.
4. **Evento:** uma ação momentânea praticada ou sofrida por uma entidade, causando alterações no sistema.
5. **Atividade:** uma operação temporária praticada ou sofrida por uma entidade, sendo iniciada e finalizada por um evento.
6. **Processo:** uma sucessão de atividades, em ordem cronológica, que descreve a vida de uma entidade.

A modelagem de sistemas discretos é caracterizada pela existência de dois tipos de eventos: **condicional** e **incondicional** (ou **limite**) (Pidd, 1992). Os eventos condicionais são aqueles que dependem de certas condições do estado do sistema, além do tempo, para ocorrer. Por outro lado, os eventos limites dependem somente do tempo para a sua ocorrência. Os eventos limites e condicionais são, às vezes, identificados por eventos B (*Bound*) e eventos C (*Conditional*), respectivamente.

Os paradigmas de simulação determinam a forma e como o modelo de simulação será implementado no computador (Davies and O'Keefe, 1988). Freqüentemente, as linguagens de simulação baseiam-se em pelo menos um dos três paradigmas existentes: orientado ao evento, orientado à atividade e orientado ao processo. Os paradigmas orientados ao evento e ao processo são os mais conhecidos e utilizados pela maioria das linguagens de simulação. O paradigma orientado ao evento ainda pode ser dividido em dois tipos: duas-fases e três-fases (Pidd, 1992).

Todos os eventos B's são programados (ou agendados) para ocorrerem em um tempo futuro, com relação ao tempo da simulação. O tempo de ocorrência de um evento B normalmente é gerado aleatoriamente, segundo uma distribuição de probabilidade. Os eventos programados são armazenados e ordenados (cronologicamente) em uma lista denominada *calendário da simulação*. A parte do programa de simulação responsável por avançar o tempo e executar os eventos do *calendário* (em ordem cronológica) é denominada *executivo*. A cada iteração,

o *executivo* realiza três tarefas: 1) o relógio da simulação é atualizado para o tempo do evento que se encontra no topo do *calendário*; 2) todos os eventos B's, que estão no topo do calendário e que estão com o seu tempo de ocorrência igual ao do relógio da simulação, são retirados do *calendário* e executados; 3) todos os eventos C's são testados e, caso suas condições sejam satisfeitas, são executados.

O método das três-fases foi descrito pela primeira vez por Tocher (1963) e mais tarde detalhado por Pidd (1992). A diferença entre os dois paradigmas (duas-fases e três-fases) está na forma como os eventos condicionais são tratados. Pelo método das três fases, os eventos condicionais são ordenados de tal forma que, a cada ocorrência de um evento limite, são testados todos os eventos condicionais e, caso suas condições sejam satisfeitas, são executados. Já pelo método das duas-fases, os eventos condicionais são incorporados pelos eventos limites convenientes, ou seja, cada evento condicional é codificado junto com aqueles eventos limites cujas ocorrências satisfazem, pelo menos, uma das condições exigidas para que o evento condicional ocorra. Pelo método das duas-fases, a terceira tarefa realizada pelo executivo, apresentada anteriormente, é eliminada.

O método das três-fases é um método de programação que facilita o desenvolvimento evolucionário. Essa é sua principal vantagem, pois o usuário inicia com um simples modelo, podendo, posteriormente, acrescentar novas características do sistema real com muita facilidade, sem grandes alterações no modelo original.

Culturalmente, o método das duas-fases é uma abordagem mais conhecida e utilizada nos EUA, enquanto o das três-fases é mais popular na Inglaterra (Pidd, 1992).

Tanto pelo método das duas-fases quanto pelo método das três-fases o modelo de simulação é implementado de maneira que os eventos fiquem em blocos separados e independentes. Por outro lado, através do paradigma orientado ao processo, o modelo é implementado de maneira que os eventos fiquem numa seqüência cronológica, formando os ciclos de vida de cada tipo de entidade do sistema.

Atualmente, o SDD permite que o usuário implemente seu modelo utilizando o método das duas-fases ou das três-fases. A abordagem orientada ao processo ainda está em fase de testes.

Estrutura do SDD

A ferramenta denominada SDD (Simulação Discreta em Delphi) foi implementada em Delphi 4.0. O compilador Delphi foi escolhido por ser um ambiente de desenvolvimento de sistemas bastante utilizado pelos acadêmicos e por profissionais da área de computação. Dessa forma, os acadêmicos estarão trabalhando com um ambiente conhecido, podendo ser reutilizado na sua profissão futuramente.

O simulador SDD é constituído por um conjunto de bibliotecas de objetos e de funções. As bibliotecas fornecem suporte para representação e manipulação de entidades, filas, recursos e eventos, além de suporte para coleta de estatísticas e geração de variáveis aleatórias. As bibliotecas foram implementadas em *Object Pascal* e devem ser usadas em conjunto com o Delphi 4.0. Isso significa que o usuário implementa seu modelo de simulação no Delphi, utilizando o suporte oferecido pelas bibliotecas do SDD.

A programação orientada a objetos é um termo amplamente utilizado nos últimos tempos, como sendo um estilo de programação. Uma das principais alegações feitas sobre a programação orientada a objetos é a maneira natural de pensar sobre as coisas. Nos últimos anos, tem havido muito interesse em juntar a programação orientada a objetos com a simulação computacional. Exemplos de tentativas de implementação de sistemas em C++ incluem o trabalho de Joines *et al.* (1992) e Fishwick (1992), ambos adotaram o paradigma orientado a processos. Pidd (1995) mostra que o método das três-fases pode ser muito útil para a implementação de sistemas de simulação orientado a objetos.

O SDD é composto por várias classes de objetos que podem ser utilizadas na implementação dos modelos de simulação. A Figura 1 ilustra um conjunto de classes existentes e suas relações hierárquicas. Esse conjunto de classes de objetos e suas relações foram desenvolvidos especificamente para esta ferramenta.

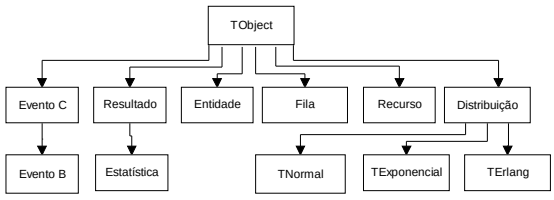

Figura 1. As principais classes de objetos do SDD

Como exemplo, a Figura 2 ilustra a classe EntidadeOBJ para a representação de entidades. A definição de EntidadeOBJ inclui todas as características que são comuns para todos os tipos de entidades. A classe EntidadeOBJ herda as características da classe TObject que é a base para todas as outras classes em Delphi.

```
EntidadeObj = Class (TObject)
  Privated
    Disponivel   : Boolean;          // informação sobre sua utilização
    Atributo     : Byte;             // atributo para identificação da entidade
    EventoB      : Procedure;        // um ponteira para um evento limite
    Tempo        : Real;             // o tempo da ocorrência do evento limite
    Cooperante   : Entidade;         // ponteiro para um entidade cooperante
  Public
    CONSTRUCTOR Create(A : byte);
   // Outros métodos são incorporados aqui
End;
```

Figura 2. Os elementos da classe EntidadeOBJ

A programação orientada ao objeto possibilita o encapsulamento das características de cada objeto, não permitindo que outros objetos interfiram nos seus dados diretamente. Essa caraterística facilita o controle do ciclo de números aleatórios independentes. Aproveitando esse recurso da linguagem, foi implementada uma classe de objetos para cada tipo de distribuição de probabilidades, ou seja, para cada variável aleatória do modelo, o usuário deverá implementar um objeto da classe correspondente (exemplo: TNormal, TUniforme, etc.). O SDD possui uma classe geral (freqüentemente denominada de classe abstrata) que é utilizada como base para a criação de outras classes para representar cada tipo de distribuição de probabilidade. A Figura 3 mostra a definição de algumas dessas classes. Esta habilidade de definir uma nova classe em termos de uma outra classe anterior é conhecida como *herança* e evita a necessidade de redefinir certas propriedades na nova classe. Por exemplo, a classe TDistribution possui um método chamado GERAR que é herdado por todos os tipos (uma classe para cada tipo) de distribuição de probabilidade, porém, para cada distribuição, seu código será diferente, por isso as palavras *VIRTUAL* e *ABSTRACT* ocorrem após o método.

```
TDistribution = class (TObject)  //Classe abstrata para as distribuições de probabilidade
  Private
    S: ^LongInt;                          //ponteiro para a semente de números aleatórios
    Function RAND: Extended;              //gerador de números aleatórios
  Public
    Function Gerar: Extended; Virtual; Abstract;// função que retorna o valor da variável aleatória
end;

TUniforme = class(TDistribution)          // classe da distribuição uniforme
TNormal   = class(TDistribution)          // classe da distribuição normal
TErlang   = class((TDistribution)         // classe da distribuição Erlang
```

Figura 3. Definição da classe abstrata para as variáveis aleatórias

Observe que para cada distribuição de probabilidade a sua classe correspondente possuirá um gerador de números aleatórios (com o mesmo código para todas as classes). Como cada classe possui sua própria semente para geração de números aleatórios, isso significa que cada variável aleatória terá seu próprio ciclo de números aleatórios independentes. Dessa forma, o controle dos ciclos fica mais fácil e transparente para o usuário, dando maior segurança e confiabilidade no modelo implementado. Uma idéia simples, porém, muito eficiente.

Exemplo de um problema

A experimentação do modelo equivale a executar um programa no ambiente Delphi. A implementação de um modelo consiste em escrever um arquivo-texto com os elementos do modelo. Para ilustrar o uso do SDD, será considerada uma versão simplificada do caso *Great Western Steel,* apresentado por Vatter *et al.* (1978, *apud* Saliby, 1993). Trata-se de um simples sistema de fila do tipo M/G/3, onde os navios chegam em um porto com três cais. Ao tempo entre-chegadas, segue uma distribuição exponencial negativa com média de 0,73 dias, enquanto ao tempo de serviço segue uma distribuição normal com média de 1,22 dias e o desvio-padrão de 0,15 dias. A fila de chegada não possui restrição de capacidade e sua disciplina segue o sistema FIFO (First-In First-Out). A Figura 4 apresenta o DCA (Diagrama Ciclo da Atividade), que é uma forma gráfica para modelar a estrutura estática de um sistema, facilitando seu entendimento e a implementação do modelo de simulação (Pidd, 1992).

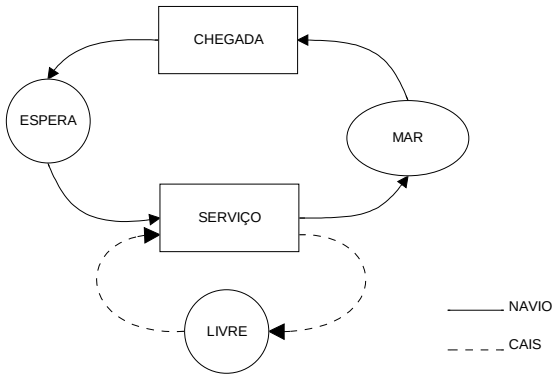

Figura 4. Diagrama Ciclo de Atividades para o problema

Neste exemplo, são identificados um tipo de entidade (NAVIO), um tipo de recurso (CAIS), duas filas (ESPERA e LIVRE) e duas atividades (CHEGADA e SERVIÇO). Observe que o recurso CAIS pode, também, ser entendido como entidade permanente; neste caso, quando o CAIS estiver ocioso, ele é colocado na fila LIVRE. Pelo método das três-fases, as atividades CHEGADA e SERVIÇO dão origem a três eventos: Chegada, Início do Serviço e Fim do Serviço. A partir dessa descrição, o modelo é implementado no SDD, cuja a implementação e a experimentação são mostradas na próxima seção.

Descrição funcional do SDD

A implementação de um modelo no SDD consiste em escrever o modelo em um arquivo-texto que será compilado pelo Delphi. A Figura 5 apresenta a implementação do problema citado na seção 858. O arquivo com o modelo é incluído no programa principal (veja a Figura 6) e, então, compilado.

```
//Arquivo PORTO.TXT
Var
  NAVIO                : Entidade;
  CAIS                 : Recurso;
  ESPERA               : Fila;
  Chegada, Fim_Servico : EventoB;
  Inicio_Servico       : EventoC;
  Tempo_Chegada        : TExponencial;
  Tempo_Servico        : TNormalTrunc;

Procedure ChegaPeca; { Evento B1 }
begin
  ESPERA.PoeNaCauda(EntidadeCorrente);
  NAVIO:= NovaEntidade(1);
  Chegada.Programar(NAVIO,Tempo_Chegada.Gerar);
end;

Procedure InicioServico; { Evento C1 }
begin
  While (ESPERA.Tamanho>0) and (CAIS.Disponivel>0) do
    begin
      NAVIO := ESPERA.Topo;
      CAIS.Adquire(1);
      Fim_Servico.Programar(NAVIO,Tempo_Servico.Gerar);
    end
end;

Procedure FimServico; { Evento B2 }
begin
  EliminaEntidade(EntidadeCorrente);
  CAIS.Libera(1);
end;

Procedure DefineModelo;
begin
  DfNomeDoModelo('PORTO'); {Máximo 7 caracteres}
  DfEventoB(Chegada,ChegaPeca,'Chegada dO NAVIO');
  DfEventoB(Fim_Servico,FimServico,'Fim do serviço');
  DfEventoC(Inicio_Servico,InicioServico,'Início do serviço');
  DfFila(ESPERA,'ESPERA',TempoNumero);
  DfRecurso(CAIS,'CAIS');
  Stream.DfExponencial(Tempo_Chegada,0.73);
  Stream.DfNormalTrunc(Tempo_Servico,1.22,0.15);
end;

Procedure Inicializa;
begin
  NAVIO:= NovaEntidade(1);
  Chegada.Programar(NAVIO,TEMPO_Chegada.Gerar);
end;
```

Figura 5. Exemplo de um modelo implementado com o SDD

```
program PORTO;          // Pograma Princial
uses
 Forms,  UMEnu, UPersonal, Rand, SDD;

 {$I  PORTO.txt }       // inclusão do arquivo com o modelo

begin
  Application.Initialize;
  DefineModelo;
  Simular(Inicializa);
  MostraFormulario;
  Application.Run;
end.
```

Figura 6. Programa principal que inclui o arquivo com o modelo

O SDD possui uma interface pronta que é apresentada no momento em que o programa (incluindo o modelo) é executado. A interface (Figura 7) possibilita ao usuário modificar alguns parâmetros do modelo (opção MODELO), estabelecer os parâmetros da simulação (opção SIMULAR), tais como, tempo de simulação, tempo de aquecimento e número de replicações (Figura 8) e, finalmente, visualizar os resultados coletados (opção RESULTADOS).

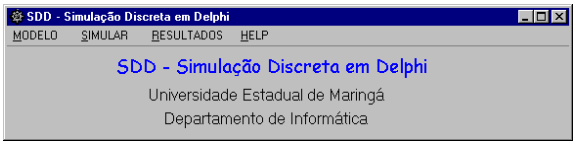

Figura 7. Tela de inicialização do SDD

Para o modelo do exemplo, foi considerada a duração da corrida igual a 365 dias e 50 replicações. A Figura 9 apresenta a tela dos resultados estatísticos da simulação, onde se observa que o CAIS foi utilizado em, aproximadamente, 56% (1,667 de 3) do tempo. Nessa interface, o usuário também tem a opção de gerar relatórios em arquivos no formato ASCII.

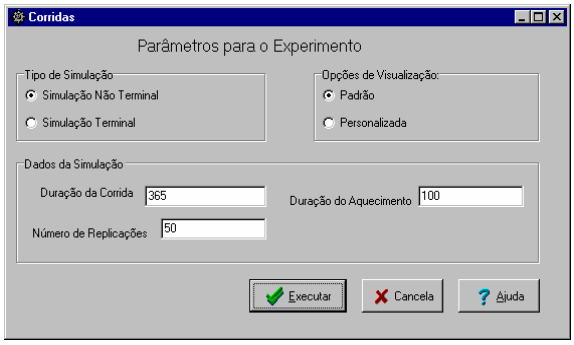

Figura 8. Tela com os parâmetros da simulação

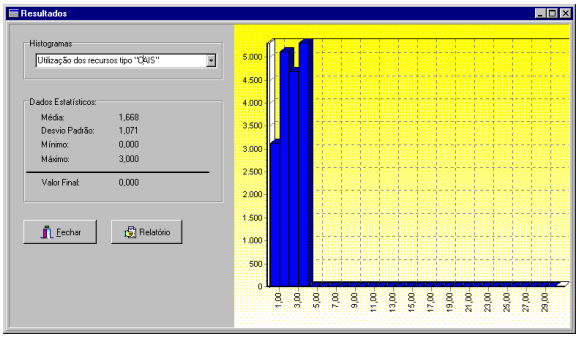

Figura 9. Tela de apresentação dos resultados da simulação

O SDD é capaz de coletar automaticamente os dados estatísticos referentes às filas e aos recursos implementados no modelo. É muito interessante que as estatísticas sejam produzidas automaticamente para aqueles estudantes que não tenham especificado, principalmente em suas primeiras simulações, quais dados tem de ser coletados e o que deve ser feito com eles. Isso significa que, pelo menos em suas primeiras simulações, os estudantes necessitam implementar somente o modelo do sistema. Essa facilidade reduz o tempo de implementação do modelo e aumenta a confiabilidade dos resultados coletados.

Análise do SDD

O objetivo do SDD não é competir com os simuladores já existentes, mas servir como recurso didático para o ensino de simulação discreta, através dos diferentes paradigmas. As experiências na utilização do SDD em sala de aula têm sido bastante animadoras, por ser uma ferramenta simples, flexível e eficiente. O simulador foi aplicado nos anos de

1998 e 1999, no curso de Ciência da Computação da UEM. Através de uma avaliação verbal com os alunos, pôde ser constatada a sua simplicidade, eficiência e robustez. A ferramenta despertou em alguns alunos a curiosidade de conhecer detalhes de implementação do sistema.

Os estudantes de Ciência da Computação têm uma necessidade especial de trabalhar com estrutura de dados (por exemplos: listas e registros) e algoritmos. O SDD fornece uma excelente fonte de exemplos de aplicações de estruturas avançadas (estruturas dinâmicas), além de trabalhar com programação orientada a objetos.

Os alunos de Ciência da Computação estão constantemente envolvidos com problemas que necessitam de uma modelagem computacional. O SDD possibilita que os alunos trabalhem as diferentes maneiras de implementação de um mesmo sistema, exercitando sua capacidade de abstração e modelagem computacional de sistemas reais.

Segundo Smith (1989), uma linguagem de simulação para ensino deve ser barata, robusta, portátil e capaz de suportar modificações. O SDD possui todas essas características.

Deve ser ressaltado que o SDD não tem o propósito de ser somente uma ferramenta para auxiliar no ensino de simulação discreta, mas também oferecer suporte para a pesquisa sobre linguagens de simulação e para aplicações de simulação em geral.

Referências bibliográficas

Davies, R.M.; O'Keefe, R.O. *Simulation modelling with pascal.* New Jersey: Prentice-Hall, 1988.

Fishwick, P.A. SIMPACJ: getting started with simulation programming in C and C++. WINTER SIMULATION CONFERENCE, 1992, San Diego, CA. *Proceedings*... San Diego: Society for Computer Simulation, 1992. p. 145-153.

Joines, J. A.; Powell, K.A.; Roberts, S.D. Object-oriented modelling and simulation in C++. WINTER SIMULATION CONFERENCE, 1992, San Diego, CA. *Proceedings*... San Diego: Society for Computer Simulation, 1992. p. 145-153.

Kelton, W.D.; Sadowski, R.P.; Sadowski D.A. *Simulation with Arena*. Boston: MacGraw-Hill, 1998.

Pegden, C. D.; Shannon, R. E.; Sadowski, R. P. *Introduction to simulation using SIMAN*. New Jersey: McGraw-Hill,1990.

Pidd, M. *Computer simulation in management science*. Shichester: Wiley, 1992.

Pidd, M. Object-orientation, discre simulation and the three-phase approach. *J. Operat. Res. Soc.*, *46*:362-374, 1995.

Saliby, E.; Paul, R.J. Implementing descriptive sampling in three-phase discrete event simulation models. *J. Operat. Res. Soc.*, *44*(2):147-160, 1993.

Saliby, E.; Pimentel, M. *Simul:* Um Sistema Computacional para Simulação a Eventos Discretos em Turbo-Pascal. Rio de Janeiro, COPPAD/UFRJ, 1992.

Smith, G. A language for teaching discrete-event simulation. *J. Operat. Res. Soc.*, *40*(9):761-770, 1989.

Tocher, K. D. *The art of simulation*. Great Britain: English University Press, 1963.

Vatter, P. A.; Bradley, S. P.; Frey Jr, S. C.; Jackson, B. B. *Quantitative methods in management:* text em cases. Homewood: Irwin. 1978.

Received on October 15, 1999.
Accepted on November 24, 1999.