9.3 (more x thes)

JAIME SHINSUKE IDE

IMPLEMENTAÇÃO DE ALGORITMOS DE INTEGRAÇÃO MONTE CARLO PARA PROCESSAMENTO ESTATÍSTICO EM SISTEMAS EMBARCADOS

Trabalho de conclusão de curso apresentada à
Escola Politécnica da Universidade de São Paulo.

Área de Concentração:
Engenharia Mecânica - Automação e Sistemas

Orientador: Prof. Dr.
Fábio Gagliardi Cozman

São Paulo
2000

Aos meus pais e meu irmão

Agradecimentos

Agradeço ao Prof. Dr. Fábio Gagliardi Cozman pelo apoio que me deu desde o início do ano. Pelos seus conselhos e a orientação acadêmica.

Agradeço ao meu amigo Marko Ackermann pela amizade concretizada no bom humor e em inúmeros detalhes de serviços.

Agradeço aos meus amigos Júlio Rodrigo Caldo, Frederico Vines, Mathias Juan Perazzo e Alex de Oliveira pelas inúmeras ajudas.

Agradeço aos meus pais e meu irmão a minha própria vida e a educação dentro de uma família sadia.

Agradeço a tantas pessoas que me proporcionaram as condições para estudar e realizar este trabalho, especialmente ao professor Gilberto Martha de Souza pelas inúmeras orientações em Análise de Confiabilidade.

Agradeço ao Celso Barros pela sua amizade e inúmeras ajudas no laboratório, que me deram condições de avançar no projeto.

Agradeço a Deus por me ter criado e capacitado para estudar e trabalhar.

ÍNDICE

LISTA DE FIGURAS 3

RESUMO 5

1. INTRODUÇÃO 5

2. ESTUDO DE VIABILIDADE 7

2.1. Estabelecimento da Necessidade 7

2.2. Formulação do Projeto 8

3. UM POUCO SOBRE TEORIA DE PROBABILIDADE 9

3.1. Probabilidade Condicional 9

3.2. Redes Bayesianas 9

3.3. Método de Simulação Monte Carlo 11

3.3.1. Origem do método de Monte Carlo 11

3.3.2. Exemplo do método "Hit- or-Miss" 12

3.3.3. Dois fatores importantes do método 13

3.3.4. Problemas que podem ser resolvidos pelo método 13

3.4. Monte Carlo em Confiabilidade 14

3.4.1. Introdução 14

3.4.2. Etapas necessárias para a Simulação de Monte Carlo 14

3.5. Monte Carlo em Redes Bayesianas 15

4. DESCRIÇÃO DO PROJETO 17

4.1. A Solução Escolhida 17

4.2. Vantagens da Solução Escolhida 18

4.3. A linguagem Java 19

4.4. Sistema Embarcado aJile Systems aJ-PC104 20

4.4.1. Introdução 20

4.4.2. Java em Sistemas Embarcados 21

4.4.3. Descrição do aJ-PC104 23

4.4.3.1. Introdução 23

4.4.3.2. Ferramentas de Suporte 24

4.4.3.3. Características hardware do aJ-PC104 27

4.5. Programa JavaBayes 27

4.6. Implementação do Algoritmo de Confiabilidade 30

4.6.1 Definição do Sistema 30

4.6.2 Descrição da Metodologia 31

4.6.3 Arquitetura do programa McsEstrutural 32

4.6.4 Método de Integração Simpson 33

4.7. Método Gibbs Sampling 34

4.7.1. Aspectos Teóricos 34

4.7.2. Resolvendo o exemplo DogProblem 36

4.7.3. Implementação do Algoritmo 37

5. APRESENTAÇÃO DOS RESULTADOS 39

5.1. Cálculo da Confiabilidade 39

5.2. Cálculo da Probabilidade Condicional 40

6. DISCUSSÃO DOS RESULTADOS 40

7. CONCLUSÃO 41

8. REFERÊNCIAS BIBLIOGRÁFICAS 42

9. ANEXO 43

LISTA DE FIGURAS

Figura 1:Arquitetura do projeto. 8
Figura 2: Representação de um modelo estatístico. 10
Figura 3: Quadrado Unitário. 12
Figura 4: Arquitetura do Projeto: Cliente-Servidor. 18
Figura 5: Esquema de funcionamento da linguagem Java. 19
Figura 6: Ambiente NetWorking. 20
Figura 7: Modelo Embarcado Cliente-Servidor. 21
Figura 8: Alternativas de execução do código Java. 22
Figura 9: Sistema Embarcado aJ-PC104. 24
Figura 10: Apresentação do programa JEMBuilder. 25
Figura 11: Apresentação do debugador Charade. 26
Figura 12: Interface gráfica do programa JavaBayes. 27
Figura 13: Classes Estruturais da rede Bayesiana. 28
Figura 14: Classes de Inferências. 29
Figura 15: Histograma- Resistência x Solicitação. 31
Figura 16: Representação das Simulações de Monte Carlo. 32
Figura 17: Arquitetura do programa McsEstrutural. 32
Figura 18: Processo de obtenção da variável aleatória Y_x. 34
Figura 19: DogProblem. 36
Figura 20: Apresentação do programa GibbsSampling. 38
Figura 21: Apresentação do programa McsEstrutural. 38

RESUMO

Este trabalho de formatura consistiu na implementação de algoritmos de integração Monte Carlo em sistemas embarcados. O método Monte Carlo vem sendo estudado com grande interesse por pesquisadores para o cálculo de integrais mais complexas. O sistema embarcado é uma tendência das aplicações atuais, onde o processamento se dá localmente, comunicando-se com outros dispositivos conectados na rede.

O algoritmo foi implementado em duas aplicações. Um programa para o cálculo de confiabilidade de um sistema estrutural. Outro para realização de inferências numa Rede Bayesiana, que é um modelo estatístico.

Os dois programas foram escritos na linguagem Java e são executados diretamente, com processamento em tempo real, no sistema embarcado denominado aJ-PC104. Este sistema se comunica com qualquer computador PC que esteja localizado na rede local, através do protocolo da Internet, enviando-lhe os resultados obtidos.

Estes programas desenvolvidos são ideais para aplicações embarcadas, devido à sua eficiência e ao pequeno espaço de memória ocupado no sistema embarcado.

1. INTRODUÇÃO

Atualmente, o emprego da teoria de probabilidade na Inteligência Artificial e Robótica é cada vez maior. Vale ressaltar que este emprego não se limita apenas ao campo da engenharia, mas ao campo da economia, da genética e de outros. Dentre estas aplicações estão: a simulação de sistemas econômicos, diagnósticos médicos, auto diagnóstico de máquinas e de robôs, análise de imagens (Filtros de *Kalman*), análise de confiabilidade, tomada de decisões, etc. Sendo assim, este assunto é uma área de interesse para o desenvolvimento de máquinas de produção na engenharia mecânica.

A Teoria *Bayesiana* é uma destas teorias. Esta teoria forma a base para se construírem Redes *Bayesianas*, que são utilizadas para se modelar sistemas e obter estimativas desejadas.

O projeto consistiu na pesquisa, desenvolvimento e implementação de dois programas de resolução estatística. O primeiro programa calcula a confiabilidade de um sistema estrutural e o segundo realiza inferências numa *Rede Bayesiana*. Os dois programas se utilizam de métodos de *Simulação Monte Carlo*, o que possibilita realizar cálculos aproximados, reduzindo o espaço de memória necessário. Estes programas rodam num sistema embarcado, *aJ-PC104*, que se comunica com qualquer computador PC que esteja na rede local, através do protocolo da Internet. Este sistema embarcado consiste numa placa que pode ser conectado à uma máquina de produção, um robô, remoto. Esta placa rodará o programa em *Java* em tempo real, ou seja executará os *bytecodes* diretamente, assim aumentando a eficiência na execução do código e se comunicará com um PC da rede local.

O segundo algoritmo consiste numa resolução numérica (utilizando *Gibbs sampling*) do Método de Integração *Monte Carlo Markov Chain* (MCMC), que realiza

cálculos estatísticos aproximados; o que aumenta a variância dos valores calculados, mas aumenta bastante a velocidade de processamento, assim compensando o aumento da incerteza com o a velocidade das respostas.

O cap.2 apresenta uma breve introdução à utilidade do projeto desenvolvido. O cap.3 apresenta um pouco sobre Teoria de Probabilidade necessária para melhor compreensão dos programas desenvolvidos. Quem já estiver familiarizado com Teoria de Probabilidade, Redes Bayesianas e Análise de confiabilidade poderá se dirigir diretamente à leitura do cap.4.

2. ESTUDO DE VIABILIDADE

2.1. Estabelecimento da Necessidade

A automação industrial é um processo cada vez mais presente em industrias que queiram ser competitivas no mercado. Pois, em muitos casos, esta automação implica em diminuição dos custos e aumento da produtividade.

Neste contesto, as máquinas modernas que possibilitam esta automação não são tão autônomas como se pensam, pois necessitam de mão de obra qualificada e de assistência técnica quando ocorre algum defeito.

Quando se trata de máquinas e de robôs que trabalham em ambientes de difícil acesso, como por exemplo em missões espaciais e explorações petrolíferas é bastante interessante se ter máquinas capazes de detectar e de diagnosticar falhas. Pois, uma vez detectado a falha pode se tomar medidas corretivas correspondente a cada tipo de falha.

A primeira vista, pode parecer que detectar falhas é uma tarefa simples. Mas, em equipamentos mais complexos a detecção de falhas não é algo tão simples. Cada sensor, utilizado para monitorar um certo parâmetro, envolve incertezas, e analisar um conjunto de parâmetros com suas respectivas incertezas não é nada que se resolva tão facilmente. Exige um método estatístico para a sua resolução. Ou melhor, o método estatístico é um poderoso instrumento para tal tarefa, ainda mais quando os recursos de memória e velocidade de processamento são limitados. Assim é necessário um método que detecte falhas com uma boa precisão, com uma boa velocidade e que ocupe pouco espaço da memória.

2.2. *Formulação do Projeto*

O projeto consiste em :

- Elaborar um programa que realize estimativas (cálculos estatísticos), dado um sistema. Este programa deve realizar cálculos com uma boa velocidade e ser compatível com a memória disponível no sistema embarcado;
- Escolher um sistema embarcado (uma placa) que armazene e processe o programa; e que tenha uma interface com o computador da máquina;

O projeto deverá possuir a seguinte arquitetura:

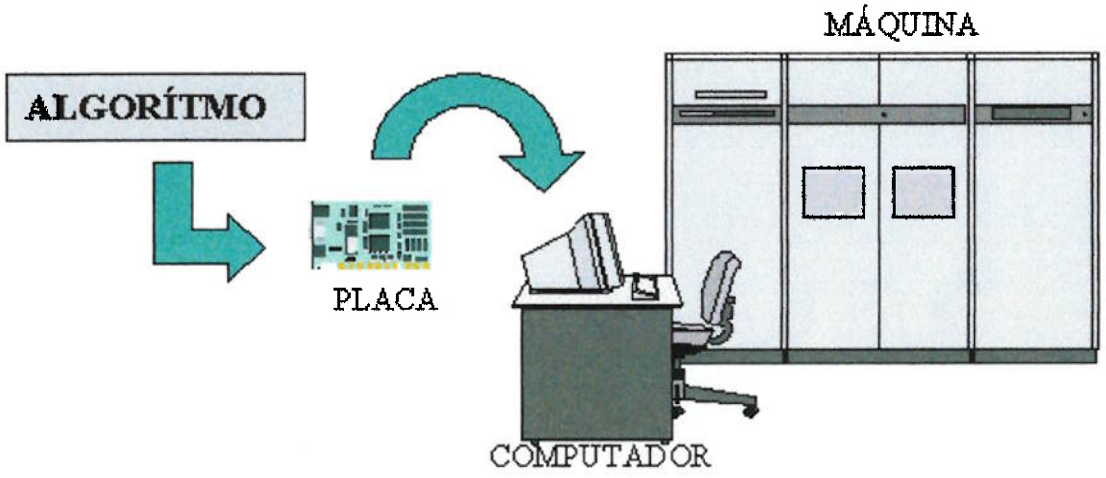

Figura 1:Arquitetura do projeto.

3. UM POUCO SOBRE TEORIA DE PROBABILIDADE

3.1. Probabilidade Condicional

Seja um conjunto de variáveis $X=\{X_i\}$, a distribuição conjunta p(X) contém um valor de probabilidade para cada combinação de variáveis. Como o número de combinações é normalmente muito elevado, fica inviável especificar a distribuição conjunta explicitamente. Tornando-se bastante útil o emprego de probabilidades condicionais.

Considerando duas variáveis X e Y, a probabilidade condicional de X dado Y é definida como:

$$p(X/Y) = p(X,Y) / p(Y)$$

Onde:

- p(X,Y) é a probabilidade conjunta de X e Y;
- p(Y) é a probabilidade de Y.

Essa definição pode ser estendida para qualquer número de variáveis, ou seja Y , por exemplo, poderia estar representando um conjunto de variáveis.

Este conceito é algo natural para nós e é uma maneira eficiente de representar distribuições conjuntas. Sendo desenvolvido na Teoria das Redes *Bayesianas*.

3.2. Redes Bayesianas

Redes *Bayesianas* são representações compactas e eficientes para um conjunto de distribuições de probabilidade. Representa um modelo estatístico complexo num pequeno número de valores probabilísticos.

Toda rede *Bayesiana* representa uma única distribuição conjunta. A regra de formação dessa distribuição é simples. Denotando por $pa\{X_i\}$ o conjunto de variáveis que são pais diretos da variável X_i na rede. Cada variável é associada a uma distribuição $p(Xi/pa\{X_i\})$, a não ser que não possua pais.

A propriedade básica em redes *Bayesianas* é que toda variável X_i é independente de todos os antecessores de x_i, dado os pais de X_i. Isto garante que a distribuição conjunta seja definida por[9]:

$$p(X) = \Pi_i p(X_i/pa\{X_i\})$$

Essa expressão é a chave para todas as propriedades em redes *Bayesianas*. Note que a expressão garante que uma distribuição conjunta, potencialmente complexa, possa ser representada através de blocos menores(as distribuições condicionais associadas aos nós da rede)[6].

Normalmente, as redes *Bayesianas* são utilizadas para realizar cálculos de probabilidade, dado um conjunto de variáveis observadas. Por exemplo, as observações $E=\{X_4=a,X_6=b\}$ indicam que as variáveis X_4 e X_6 estão fixas. Para calcular a distribuição condicional de uma variável qualquer X_q, dadas as observações em E, é preciso computar:

$$p(X_q/E) = p(X_q,X_4, X_6) / p(X_4, X_6)$$

A expressão geral para cálculos de distribuições, dado um conjunto de evidências E é dado por[6]:

$$p(X_q/E) = \Sigma_{\{Xi/Xq,E\}} [\ \Pi_i p(X_i/pa\{X_i\}\ ] / \Sigma_{\{Xi/E\}} [\ \Pi_i p(X_i/pa\{X_i\}\ ]$$

Ou seja, cada distribuição conjunta da expressão acima é dada pelo produto das probabilidades de cada elemento do conjunto, dado seus pais.

O seguinte grafo representa uma situação do cotidiano: um ladrão está querendo saber quais são as suas chances de assaltar uma casa. Se a família estiver em casa há uma certa possibilidade de que a luz esteja acesa e de que o cachorro esteja vigiando a casa. Mas, se o cachorro estiver com uma dor de barriga, isto pode influenciar o fato do cachorro estar em vigia. E, se o cachorro está vigiando, existe uma possibilidade de que ele esteja latindo.

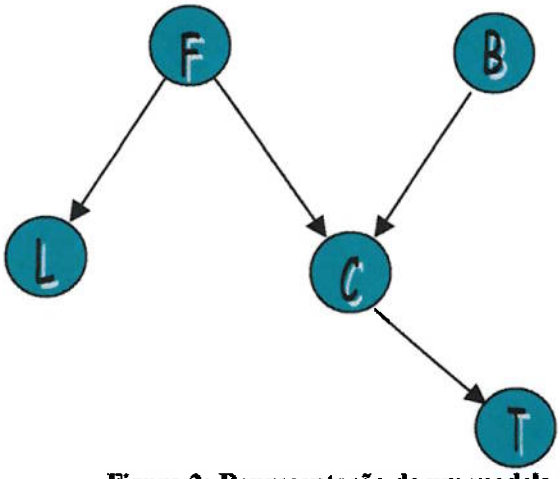

Figura 2: Representação de um modelo estatístico.

Onde:

- L : Luz acesa
- F : Família em casa
- C : Cachorro vigiando

- B : Dor de barriga
- T : Cachorro latindo

Cada um destes itens são chamados de "nós" da rede *Bayesiana.*

O ladrão pode querer saber qual é a probabilidade de que a família não esteja em casa (para que ele possa assaltar a casa) se a luz está acesa e o cachorro está latindo!

Esta probabilidade é dada por:

$$p (F= \text{família em casa} / L \text{ e } T)$$

Este é um exemplo de inferências em redes *Bayesianas*. Uma maneira de resolvê-lo é apresentado a seguir:

- Assume-se que toda variável é verdadeira(1) ou falsa(0).
- Probabilidades condicionais envolvidas:

p(L/F)	p(F)	p(C/F,B)	p(B)	p(T/C)

- O ladrão quer saber qual a probabilidade de que a família não esteja em casa, dado que a luz está acesa e o cachorro está latindo:

$$p(F=0/L=1,T=1)=p(F=0,L=1,T=1)/p(L=1,T=1)$$

$$=\Sigma_{\{X/F,L,T\}} [\Pi_i p(X_i/pa\{X_i\}] / \Sigma_{\{X/L,T\}} [\Pi_i p(X_i/pa\{X_i\}]$$

- Onde: $\Pi_i p(X_i/pa\{X_i\}= p(L/F)\ p(F)\ p(C/F,B)\ p(B)\ p(T/C)$

Note que este é o processo básico para o cálculo da probabilidade desejada. Mesmo que se faça reduções do tipo, colocar em evidência alguns termos constantes na somatória:

$$p(F,L,T) = p(F).p(L/F)\ \Sigma_{\{C,T,B\}}[\ p(C/F,B)\ p(B)\ p(T/C)]$$

O número de contas continua sendo razoavelmente alto, para uma simples rede *Bayesiana* com cinco variáveis.

3.3. Método de Simulação Monte Carlo

A idéia geral do método de Monte Carlo é resolver numericamente problemas matemáticos, através de simulações de variáveis randômicas (aleatórias).

3.3.1. Origem do método de Monte Carlo

O ano de "nascimento" do método Monte Carlo é geralmente aceito como sendo 1949, quando foi publicado um artigo entitulado "O método Monte Carlo" por Metropolis e Ulam [10]. O método em si já era utilizado na Estatística, muito antes de

1949. Mas, como era um processo muito trabalhoso realizar a simulação manualmente, o método de Monte Carlo só veio a se tornar uma técnica numérica universal com o adento dos computadores.

O nome "Monte Carlo" tem sua origem na cidade localizada em Mônaco, pelos seus famosos cassinos. O ponto é que, um dos mecanismos mais simples na geração de números randômicos é a roleta utilizada nos cassinos.

3.3.2. Exemplo do método "Hit- or-Miss"

Este exemplo simples ilustra a idéia básica do método de Monte Carlo. Supondo uma figura S qualquer, contido dentro de um quadrado unitário (fig.3), pode-se calcular a sua área da seguinte forma:

1°) N pontos são escolhidos randomicamente sobre o quadrado unitário;

2°) Chamando-se de N', o número de pontos localizados dentro de S, pode-se aproximar o valor da área de S pela proporção N'/N.

O número de pontos gerados na figura 1 é N=40. E, N'=12 pontos estão localizados dentro de S. Logo, a área de S é dado aproximadamente pela razão N'/N=0,30. Sendo que a área real é 0,35.

Na prática, o método de Monte Carlo não é utilizado para cálculos de área de figuras planas, como feito acima. Há outros métodos (como fórmulas de quadratura) para este cálculo, que, sendo mais complexas, fornecem um valor muito mais exato.

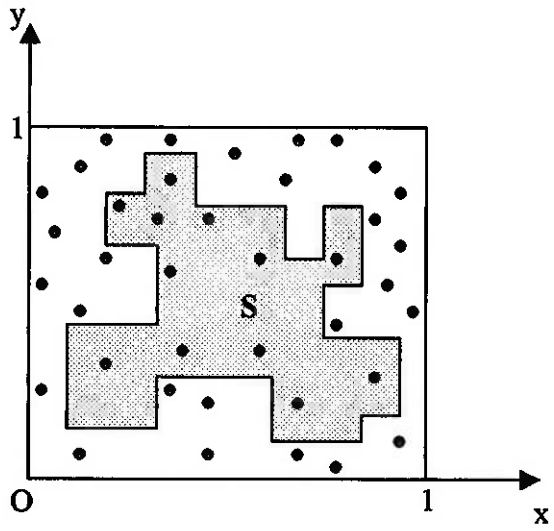

Figura 3: Quadrado Unitário.

Entretanto, o método "Hit-or-Miss", mostrado no exemplo anterior, permite estimar de forma simples o "volume multi-dimensional" de um corpo num espaço multi-

dimensional. Em muitos dos casos, o método de Monte Carlo é o mais comum para resolver este problema.

3.3.3. Dois fatores importantes do método

Uma das vantagens do método de Monte Carlo é a simplicidade estrutural de seu algoritmo computacional [11].

O segundo fator é que, como regra geral, o erro de cálculo é proporcional à $(D/N)^{1/2}$, onde D é uma constante e N é o número de tentativas feitas. No exemplo do "Hit-or-Miss", tentativa é entendida como sendo cada ponto gerado aleatoriamente sobre o quadrado unitário. Donde se conclui que, para reduzir o erro num fator de 10, é necessário aumentar N num fator de 100.

Fica claro que, é quase que impossível obter uma alta exatidão. Por isso, usualmente se diz que o método de Monte Carlo é fundamentalmente utilizado na resolução de problemas que exigem uma exatidão moderada (erro na faixa de 5 a 10%). Entretanto, o erro pode ser reduzido significativamente, pela engenhosa escolha de um método computacional que garantirá um valor pequeno da constante D [11].

O termo em plural "métodos de Monte Carlo" é freqüentemente utilizado, enfatizando que o mesmo problema pode ser solucionado, simulando-se de diversas formas as variáveis randômicas, ou seja, pela utilização de inúmeros métodos.

3.3.4. Problemas que podem ser resolvidos pelo método

Para entender que tipos de problemas podem ser resolvidos pelo método de Monte Carlo, é importante notar que o método permite a simulação de qualquer processo, que se desenvolve influenciado por fatores randômicos. Permite construir artificialmente modelos probabilísticos que solucionam muitos problemas matemáticos complexos.

Logo, o método de Monte Carlo é um método numérico universal para solução de problemas matemáticos; e o seu campo de aplicação se expande conjuntamente com a geração de novos computadores. São exemplos de campos de aplicação: o cálculo da confiabilidade de equipamentos, problemas astrofísicos, estimativa de integrais definidas, etc.

3.4. Monte Carlo em Confiabilidade

3.4.1. Introdução

O interesse em simulações na engenharia começou na década de 50, com advento dos métodos de MC (Monte Carlo). Técnicas baratas para realização de testes analíticos em sistemas de engenharia. O princípio destes métodos é o desenvolvimento de modelos analíticos, baseados computacionalmente, que predizem o comportamento de um sistema. Se um ou mais parâmetros do sistema são variáveis randômicas, o modelo é avaliado diversas vezes. Cada avaliação (chamado ciclo de simulação ou tentativas) é baseado num conjunto de parâmetros de entrada do sistema, selecionados aleatoriamente. Ferramentas analíticas são utilizadas para apurar a seleção randômica destes parâmetros, de acordo com a distribuição de probabilidades, para cada avaliação. Como resultado, várias previsões comportamentais do sistema são obtidas e para isto, métodos estatísticos são empregados para a interpretação dos tipos de distribuição das variáveis de saída [12].

3.4.2. Etapas necessárias para a Simulação de Monte Carlo

As etapas computacionais e analíticas necessárias para a simulação são:

(1) Definição do sistema;

(2) Geração das variáveis randômicas de entrada;

(3) Avaliação (simulação) do modelo;

(4) Análise estatística do comportamento resultante;

(5) Estudo de eficiência e convergência.

A definição do sistema deve incluir os seus limites, parâmetros de entrada, parâmetros de saída(comportamental) e modelos que relacionam os parâmetros de entrada com os de saída. A exatidão dos resultados da simulação é altamente dependente da exatidão da definição do sistema. É comum. na simulação de Monte Carlo, assumir modelos determinísticos. Entretanto, modelamentos não determinísticos(estatísticos) podem ser incorporados na análise com o uso de fatores de correção e adicionais variáveis, por exemplo, o coeficiente de variação. Todo parâmetro crítico deve ser incluído no modelo.

A definição de parâmetros de entrada deve incluir suas características estatísticas e probabilísticas, ou seja, o conhecimento dos tipos de distribuição. Os parâmetros de entrada devem ser geradas e substituídas no modelo para obter os

parâmetros de saída. Repetindo o procedimento N vezes (simulações), N conjunto de parâmetros de saída são obtidos. Assim, métodos estatísticos podem ser utilizados para obter, por exemplo, o valor médio, a variância, ou o tipo de distribuição dos parâmetros de saída. A exatidão dos resultados aumenta com o aumento do número de simulações. A convergência do método pode ser investigada através do estudo de seus comportamentos limites. Assim, a eficiência e a exatidão do método de simulação podem ser aumentados utilizando-se de técnicas para redução da variância. Tais técnicas se encontram na referência [12].

3.5. Monte Carlo em Redes Bayesianas

Diante de uma rede *Bayesiana* bastante complexa, muitas vezes é inviável calcular o valor desejado $p(X_Q/E)$, através dos métodos computacionais tradicionais. Pode ser inviável pelo espaço de memória que se ocupa para realizar os cálculos ou pelo tempo necessário para processar todas as contas. Uma maneira de contornar o problema é obter estimativas aproximadas, a partir de uma amostra de $p(X_Q/E)$.

Agora, o foco do problema passa a ser como gerar a amostra de $p(X_Q/E)$, e não mais calcular $p(X_Q/E)$. Uma idéia é: ordenar os nós da rede, de maneira que os nós sempre estejam precedidos de seus pais, e amostrá-los em seqüência. Assim, começando com X_1, um nó sem pais, acha-se uma amostra x_1 a partir de $p(X_1)$. Depois, seguindo para X_2 e achando x_2, a partir de $p(X_2)= p(X_1).\ p(X_2/X_1)$. E, assim sucessivamente, obtêm-se uma amostra $(x_1,x_2,...,x_n)$. O passo a seguir é verificar se a amostra é consistente com as evidências. Se a evidência não é representado na amostra, descarta-se esta amostra. E parte-se para uma nova amostragem, até que se obtenha uma que seja consistente. Este processo de amostragem é denominado "*logic samplig*"[7].

O problema do "*logic samplig*" é que: quanto maior for o número de evidências, mais tempo levará para que a amostra satisfaça as evidências. Se uma das evidências tem pouca probabilidade para ocorrer, imagine quanta amostras terão que ser geradas até que consiga uma que seja consistente!

Suponha agora uma rede com três variáveis X_1, X_2, X_3; onde X_2 está fixo. Como X_2 está fixo, o valor de X_3 pode ser gerado imediatamente de acordo com $p(X_3/X_2)$. Mas, o valor de X_1 não pode ser gerado tão facilmente, de acordo com $p(X_1/X_2)$. Seria bom poder gerar o valor de X_1, de alguma forma, a partir de X_2 fixo e de

$p(X_2/X_1)$. Esta é a idéia explorada em *Gibbs sampling*, uma das técnicas conhecidas pelo nome geral de "*Markov Chain Monte Carlo*"(MCMC). Esta técnica não gera inicialmente a amostra correta, mas faz com que a amostragem convirja para uma correta distribuição, na medida em que mais amostras são geradas[7].

4. DESCRIÇÃO DO PROJETO

4.1. A Solução Escolhida

O projeto desenvolvido neste trabalho, que vem a ser a solução proposta para a necessidade inicialmente descrita, é constituído pelas seguintes partes:

- Desenvolvimento, implementação e testes de dois programas. O primeiro programa, *McsEstrutural*, calcula a confiabilidade de um determinado sistema estrutural. E o segundo, *GibbsSampling*, realiza inferências para uma Rede Bayesiana com 5 nós.
- Utilização de cálculos aproximados, baseados em métodos de simulação Monte Carlo, para o desenvolvimento dos programas. O que reduz o espaço de memória e o tempo necessários para o processamento.
- Utilização de um programa, o *JavaBayes*[1], disponível na Internet, para comparação dos resultados. Este programa realiza cálculos exatos.
- A escolha da linguagem Java para a implementação dos programas. Esta linguagem possui inúmeras vantagens de se trabalhar num ambiente de Internet.
- Seleção, escolha e utilização da placa *aJ-PC104* [2], um sistema embarcado que roda o código *Java* (bytecode) em tempo real. A placa executa os programas desenvolvidos e se comunica com qualquer computador, conectado na rede local, através do protocolo da Internet. Trabalhando, assim, numa arquitetura cliente&servidor. Ou seja, o sistema embarcado (cliente) realiza cálculos remotamente e se comunica com um computador da rede local (servidor).

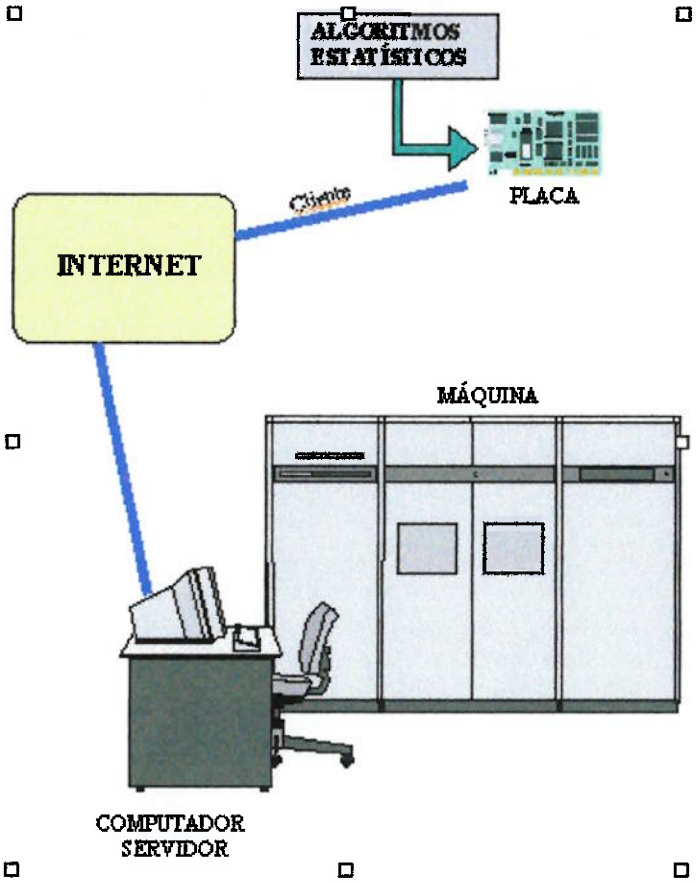

Figura 4: Arquitetura do Projeto: Cliente-Servidor.

4.2. *Vantagens da Solução Escolhida*

Os cálculos estatísticos exatos são normalmente bastante demorados. O método *Monte Carlo Markov Chain*(MCMC) realiza cálculos aproximados, o que aumenta consideravelmente a velocidade de processamento. Ganhando em tempo o que se perde em certeza. Em muitos casos, a velocidade de processamento é mais importante que um aumento desconsiderável da variância.

A linguagem *Java* permite que o código do programa seja rodado em qualquer plataforma. Ou seja, pode ser rodado independentemente do sistema operacional do computador onde o sistema está embarcado, desde que o computador possua um interpretador *Java*.

A escolha de uma placa que roda o *bytecode* diretamente elimina qualquer necessidade de se converter o algoritmo compilado (escrito em *Java*) numa outra forma

de código. E ainda, aumenta a eficiência na sua execução, aumentando a velocidade de processamento.

4.3. *A linguagem Java*

O objetivo deste item não é explicar detalhadamente sobre a linguagem *Java*, mas de observar aspectos mais gerais que auxiliarão na compreensão do projeto como um todo. Principalmente, quando se trata da implementação do algoritmo no sistema embarcado. O estudo da linguagem pode ser feita pela referência [3] dada.

O fundamental é compreender que a linguagem Java traz todas as características e ferramentas que permitem trabalhar num ambiente de *networking* (Internet).

O manuseio desta linguagem pode ser representado pela figura abaixo:

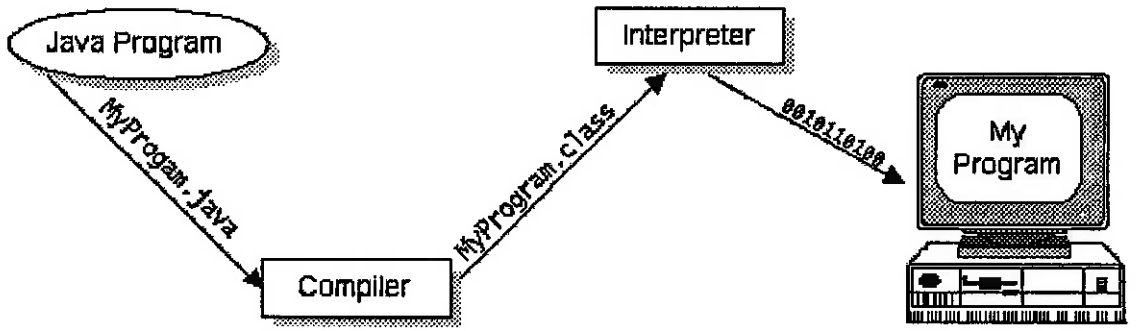

Figura 5: Esquema de funcionamento da linguagem *Java*.

O programa escrito em *Java* é compilado, sendo convertido num código intermediário, *bytecode* (*.class), que irá ser executado em qualquer plataforma que possua um interpretador *Java*. O código que será carregado no sistema embarcado (a placa) é exatamente o *bytecode*.

4.4. Sistema Embarcado aJile Systems aJ-PC104

4.4.1. Introdução [1]

A tecnologia Sun'sJava™ veio revolucionar o mundo da computação. A essência desta revolução consiste no modelo de computação "*network-centric*". Este novo paradigma causou um profundo impacto em sistemas embarcados, em que modelos centralizados estão sendo substituídos por modelos envolvendo trabalho em rede (*Networking*) com certo nível de inteligência e de autonomia embarcada. Ou seja, existe uma tendência atual de descentralização do processamento, potencializado pelos padrões de comunicação da Internet, possibilitando que, dezenas de milhões de computadores, baseados em diferentes plataformas de hardware e utilizando diferentes sistemas operacionais e programas aplicativos, comuniquem-se entre si (Wang[2]). Esta mudança vem ocorrendo pelos seguintes motivos:

- Os benefícios do emprego do Java como meio de programação estão agora largamente aceitos, devido aos modelos de objetos simplificados, portabilidade, segurança, compactibilidadde e extenssibilidade.
- O Java oferece uma oportunidade de padronizar o desenvolvimento e manutenção de uma série de aplicações. A noção de "*single point of maintenance*" é revolucionária em softwares embarcados.
- Disponibilidade da Internet como meio de comunicação omnipresente.

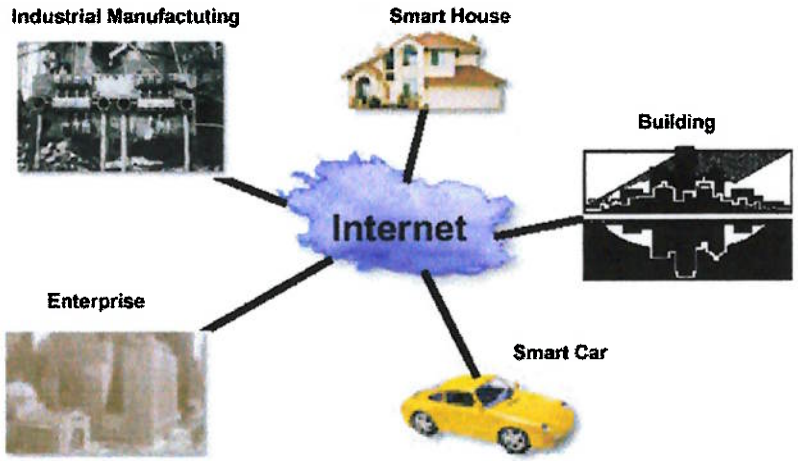

Figura 6: Ambiente *NetWorking*.

---

[1] Danh Le Ngoc e Dr. David Hardin, aJile Systems Inc., *Low-power, Direct Java Execution for Real-Tme Networked Embedded Aplications*, 1999.

[2] Charles B. Wang: Fundador da CA, *Techno Vision II*, MAKRON Books, 1998.

O casamento do par, Tecnologia Java e Padrões Internet, está rapidamente estabelecendo por si mesmo um novo paradigma de aparelhos embarcados trabalhando em rede. Estes aparelhos logo estarão presentes em residências, escritórios, automóveis, indústrias, como ilustrado na fig. 6.

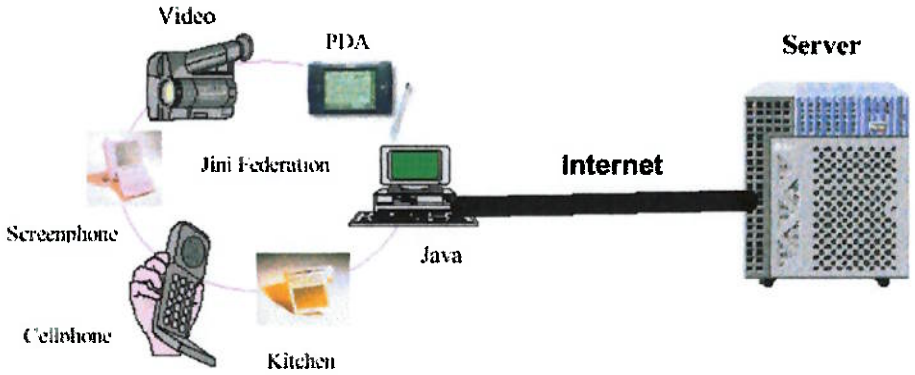

Figura 7: Modelo Embarcado Cliente-Servidor.

A linguagem orientada a objetos, Java, permitirá a reutilização de programas, enquanto que os aparelhos embarcados trabalharão distribuidamente em conjunto, conectados por um ambiente de rede. Como ilustrado na fig. 7.

4.4.2. Java em Sistemas Embarcados

A maioria dos aparelhos embarcados distribuídos atualmente empregam micro controladores de 8-bit, 16-bit, ou ocasionalmente 32-bit. Entretanto, o nível de performance destes micro controladores tem se tornado inadequados para o controle de operações cada vez mais complexas. Como os programas em Java são traduzidas para bytecode JVM (conjunto de instruções para o JVM), para que o CPU possa entender e executar o programa, é necessário que o código bytecode seja interpretado por compilador Just-In-Time (JIT) ou um interpretador, que o converterá em instruções próprias às da máquina. Este processo de conversão resulta num atraso na execução do programa, tomando um tempo extra e consumindo mais recursos, tais como memória e potência. A Sun Microsystem desenvolveu um chip, o picoJavaTM, que atende à execução direta do Java, mas com um consumo muito alto de potência (1,5W para 100MHz) para muitos sistemas embarcados. Além disso, como a maioria dos sistemas de tempo real necessitam de um rápido tempo de resposta aos eventos (na ordem de 10s a microsegundos, ou até menos), é inviável utilizar programas em Java,

tanto para um interpretador JVM rodando sobre um microprocessador embarcado, ou mesmo para processadores baseados na tecnologia picoJava. O interpretador JVM em conjunto com o software de suporte (Java runtime class libraries) tipicamente necessitam de mais de 1 Mbyte de memória, além do que é necessário para o aplicativo e o sistema de operação.

Os microcontroladores da aJile System de 32-bits vêm solucionar este problema. Tem um baixo consumo e são projetados para trabalharem em rede em sistemas embarcados. Eles rodam os programas com mais rapidez, num fator de 10 vezes, que os dos processadores tradicionais. A fig. 8 ilustra as diferentes alternativas para execução de aplicações em Java.

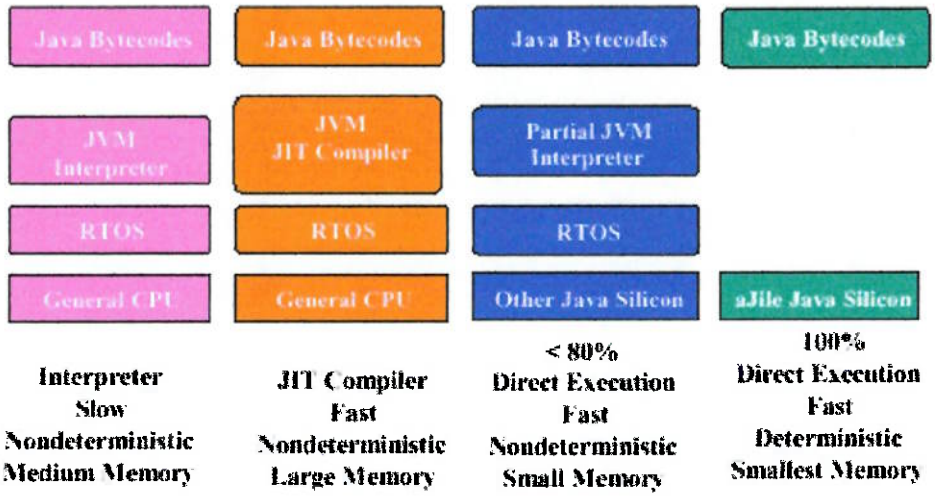

Figura 8: Alternativas de execução do código Java.

4.4.3. Descrição do aJ-PC104

4.4.3.1. Introdução

A baixa performance e o espaço largo ocupado pelos programas implementados em linguagem Java têm impedido a aceitação destes programas no mundo dos sistemas embarcados com processamento em tempo real. O sitema embarcado produzido pela empresa aJile, o aJ-PC104 Ethernet-enabled, torna viável e real o emprego do Java nestes tipos de aplicações. Trazendo ao sistema embarcado todas as vantagens apresentadas pelo Java. Baseado no microprocessador Java, JEM2 direct-execution, o aJ-PC104 proporciona uma eficiente plataforma para desenvolvimento de aplicações em tempo real inteiramente em Java. Este sistema embarcado suporta o popular protocolo de via de dados PC/104, permitindo ao projetista controlar uma larga variedade de periféricos com comunicação PC/104.

Os fatores do sistema embarcado aJ-PC104 incluem:

- Microprocessador com execução direta em Java, JEM2;
- Suporte para múltiplas JVM's (Java Virtual Machine);
- Interface PC/104;
- Ethernet (10 Base-T);
- 1 MB SRAM;
- 1 MB memória Flash;
- Controlador de interrupção programável;
- Porta serial;
- 8-bit para uso genérico de E/S;
- Optimizing Linker/ Application Builder;
- Debugador PC-based; com interface do aJ-PC104 ao PC através da porta paralela.

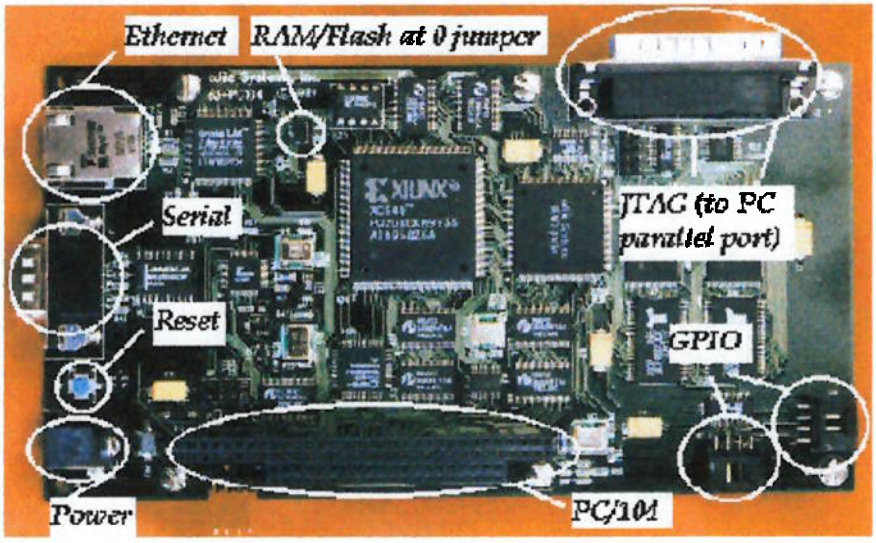

Figura 9: Sistema Embarcado aJ-PC104.

4.4.3.2. Ferramentas de Suporte

O ambiente de desenvolvimento aJile permite a utilização de qualquer tipo de compilador que produza arquivos padrões de classes. Assim, os desenvolvedores tem a liberdade de escolher o Ambiente de Desenvolvimento de Java Integrado (Java Integrated Development Environment- IDE) que desejarem. Os elementos necessários no manuseio das ferramentas aJile são o "linkador" estático (static linker), que produz imagens de formato binário JEM, e as ferramentas de "debugging" de baixo nível. O processo de "linkagem" estática é controlado por uma aplicação com interface gráfica amigável, o JEMBuilder. E, o "debugging" de baixo nível é feito pelo Charade.

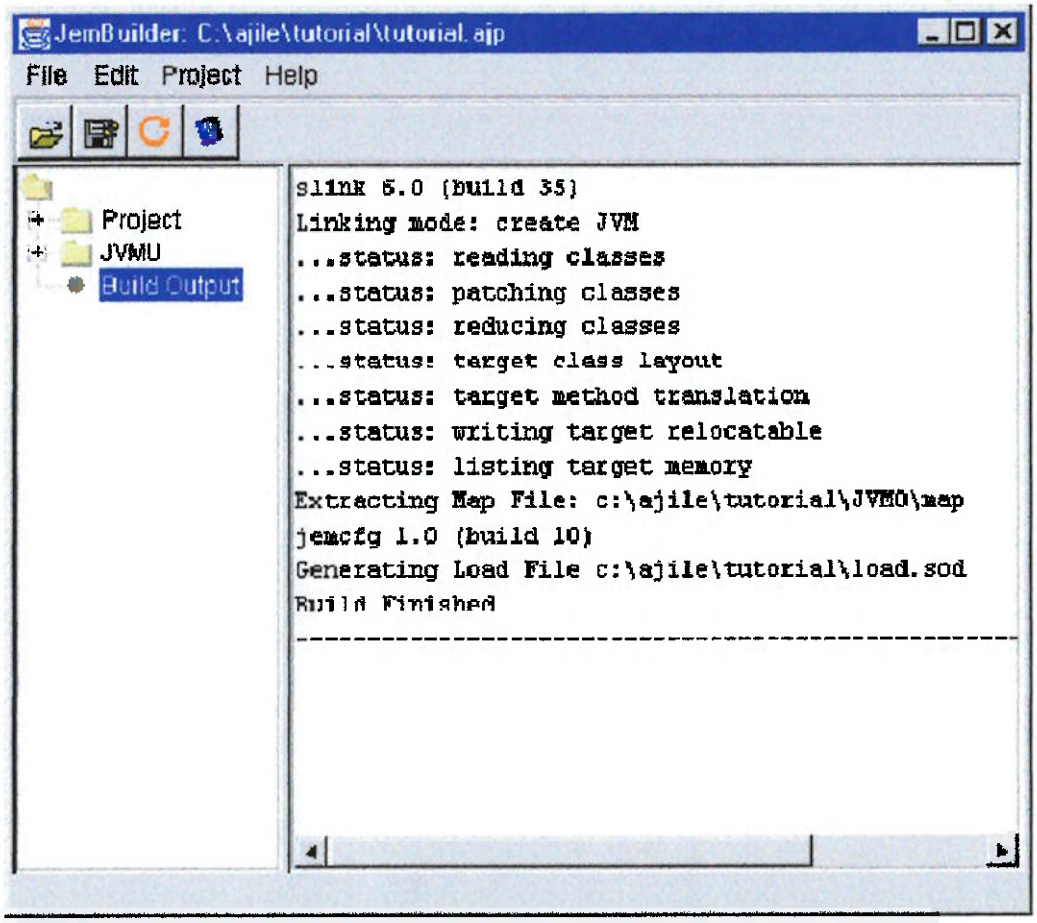

Figura 10: Apresentação do programa JEMBuilder.

A maioria das implementações Java dependem das resoluções das classes runtime, mas não necessitam ter todas as classes presentes na hora da construção. O JEMBuilder, escrito inteiramente em Java, compara as dependências de classes e cria imagens de memórias JEM2 RAM e ROM para aquelas classes que são realmente necessárias. O JEMBuilder pode eliminar os métodos, os campos e as constantes não utilizadas, resultando na redução de memória (freqüentemente, num fator de 10) para aplicações dedicadas em Java. Esta análise é possível devido às características do Java, tais como a ausência de ponteiros de manipulação.

O Charade é um "debugador" de baixo nível utilizado na família de microprocessadores JEM. Ele integra uma interface gráfica, que traz facilidades de manuseio ao usuário, com a poderosa ferramenta da linha de comando para proporcionar os seguintes fatores:

- Controle total da execução do programa;
- Acesso aos registros internos JEM e à memória requerida;
- Controle do breakpoint;
- Macros e variáveis definidas pelo usuário;

- Display de memória solicitada e janela de procura;
- Comando de execução de arquivo;
- Ambiente configurável pelo usuário.

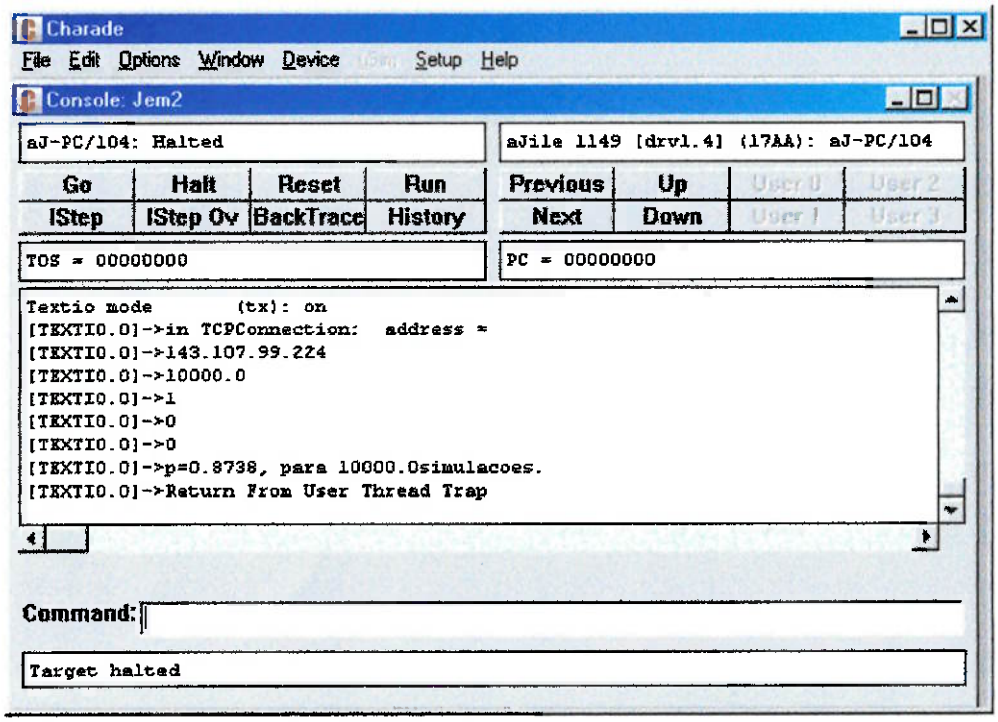

Figura 11: Apresentação do debugador Charade.

4.4.3.3. Características hardware do aJ-PC104

Alimentação: 5 volts.

JTAG para a Porta Paralela do PC: Esta interface paralela transporta os sinais IEEE 1149.1 (JTAG) entre o servidor PC e o aJ-PC104.

Porta serial: É possível fazer a interface do aJ-PC104 com o PC utilizando-se de um cabo serial null-modem 9-condutor DB-9 fêmea para DB-9 fêmea.

Ethernet: É possível comunicar a placa com uma rede local, utilizando-se de cabos de rede (conectores RJ-45). Também conhecidos como cabos de protocolo de comunicação TCP/IP.

4.5. Programa JavaBayes

O sistema *JavaBayes* [1] é um conjunto de ferramentas para inferências com modelos gráficos, contendo um editor gráfico. *JavaBayes* manuseia Redes *Bayesianas* e calcula valores como: *posterior marginals* e *posterior espectations.*

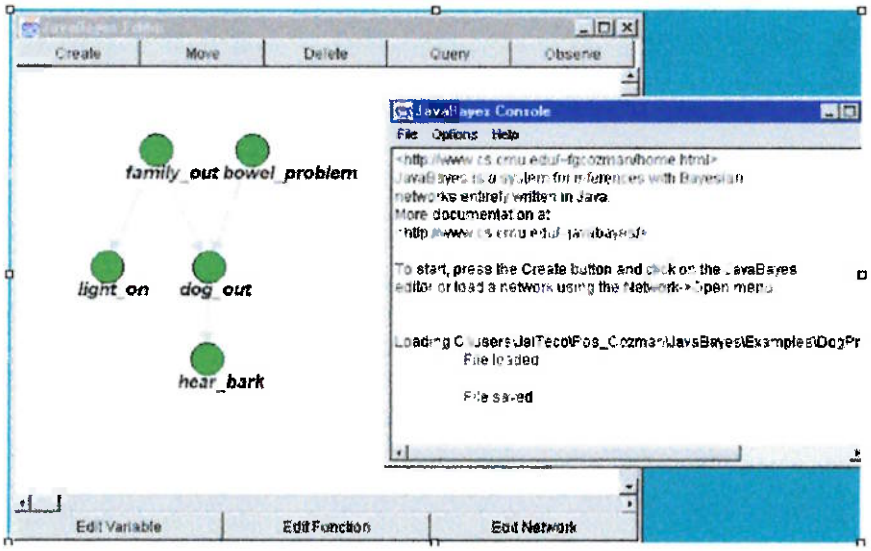

Figura 12: Interface gráfica do programa *JavaBayes*.

As classes apresentadas na fig.13, que fazem parte do pacote *BayesianNetwork*, são classes que constituem a estrutura de dados da rede *Bayesiana*:

- *BayesNet*: Contém construtores básicos para construção da rede, e. dados como o nome da rede, o número de variáveis e de probabilidades. Além de conter métodos relacionados com a rede.
- *DiscreteFunction*: Basicamente, esta é a classe que define objetos que representam as probabilidades condicionais. Guarda as relações de probabilidade entre os nós da rede. Contém inúmeros métodos auxiliares para realização de inferências.
- *DiscreteVariable*: Representa as variáveis, ou seja, os nós da rede *Bayesiana*. Armazena o nome da variável e os valores que pode assumir.

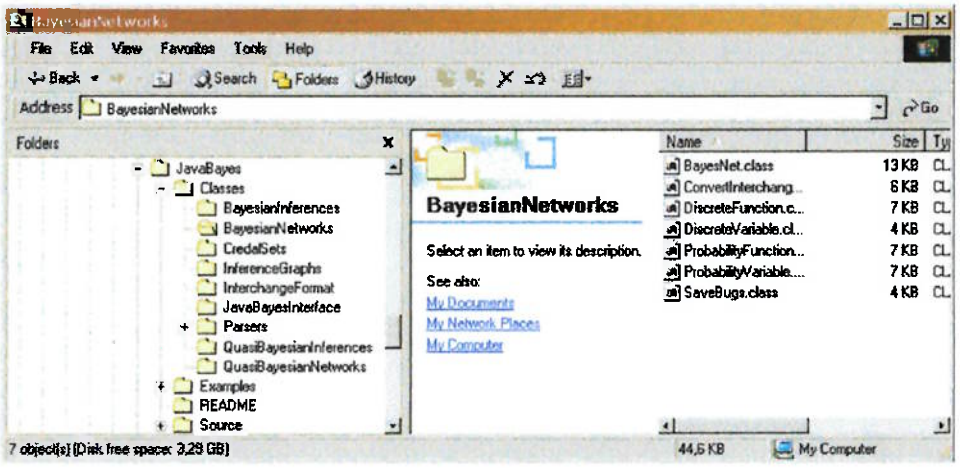

Figura 13: Classes Estruturais da rede Bayesiana.

Estas três classes formam a estrutura básica da rede. Mas, para poder trabalhar com uma rede concreta (como se fosse para abrir um arquivo) é preciso "extender" uma outra classe do tipo *BayesNet*.

Para trabalhar com uma rede concreta, por exemplo o *DogProblem*, o objeto do tipo BayesNet, que em si é abstrato, realiza uma "extensão" ao *DogProblem.class;* passando a constituir uma rede concreta, com nome e probabilidades concretas.

Existe uma classe *JavaBayes*. Esta classe é a classe principal do programa. Em *Java*, como também em outras linguagens de programação, existe sempre uma classe que contem o método principal. Este método é quem irá inicializar todas as demais classes. Particularmente, a classe *JavaBayes*, quando rodada, inicializa um menu de opções. Dependendo da opção selecionada, o método irá chamar uma ou outra classe.

A fig.14 mostra as classes empregadas para realização das inferências. As classses *Bucket's, DSeparation* e *Ordering* contém métodos auxiliares de otimização

para realização dos cálculos. A classe *Gibbs sampling* foi implementada neste projeto, servindo como mais uma opção para realização de inferências.

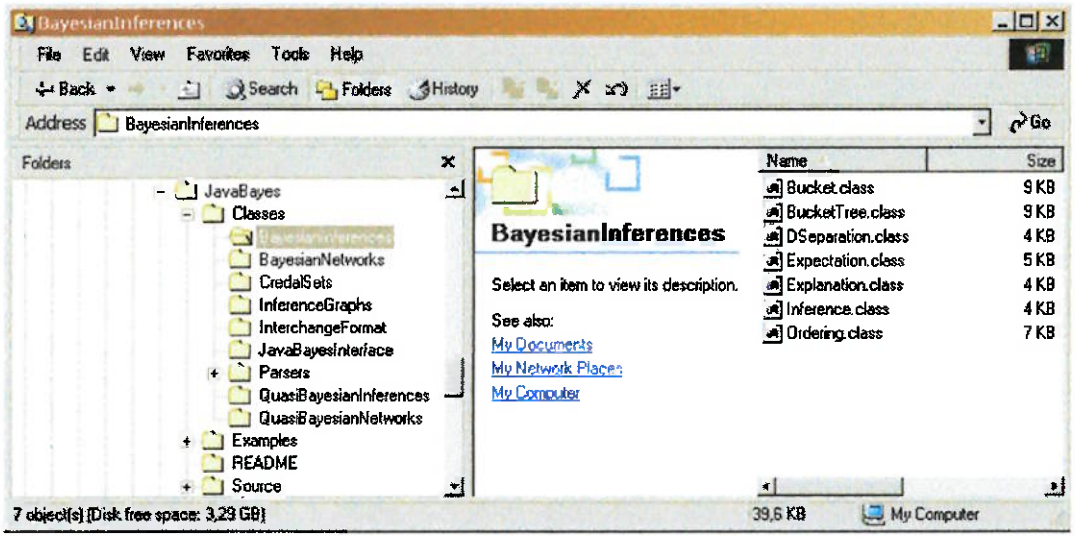

Figura 14: Classes de Inferências.

4.6. Implementação do Algoritmo de Confiabilidade

O objetivo é desenvolver um método computacional (simulação de Monte Carlo) para o cálculo de confiabilidade de um sistema estrutural. Tomou se o artigo de Hasan e Ayyub [13] como referência para o desenvolvimento e para a comparação de resultados. Pois, o foco deste trabalho não é o desenvolvimento do modelo estrutural em si, mas o desenvolvimento da ferramenta de análise de quaisquer sistema estrutural.

A confiabilidade de um componente estrutural pode ser definida como sendo a probabilidade do componente resistir à uma dada demanda, num certo intervalo de tempo. Confiabilidade é uma medida probabilística da garantia de desempenho do sistema.

4.6.1 Definição do Sistema

Uma das formas de falha de um componente estrutural é por fratura. Sua função de desempenho pode ser descrita pela expressão:

$$g = (YS/12) - (wL^2/8)$$

Onde: • **g**: é a função desempenho
- **Y**: é a tensão de escoamento
- **S**: é a área média da seção
- **w**: é a força média aplicada
- **L**: é o comprimento da estrutura

A primeira parcela da expressão representa a "resistência", enquanto que a segunda representa a "solicitação" que atua sobre o sistema. Portanto, quando $g < 0$, significa que a estrutura falha.

Cada um dos parâmetros não são determinísticos. Possuem uma média e uma variância. Assim, uma estrutura com "determinadas" características (resistência) nem sempre resistirá à uma "determinada" solicitação. Existe uma certa probabilidade de falhar , ou não.

4.6.2 Descrição da Metodologia

A função de desempenho pode ser escrita de outra forma:

$$g = R - S$$

Onde: • **R**: Representa a resistência da estrutura.

• **S**: Representa a solicitação a que a estrutura é submetida.

Tendo-se uma amostra de 100 estruturas e realizando uma bateria de testes, obter-se-ia um histograma como representado na fig.15.

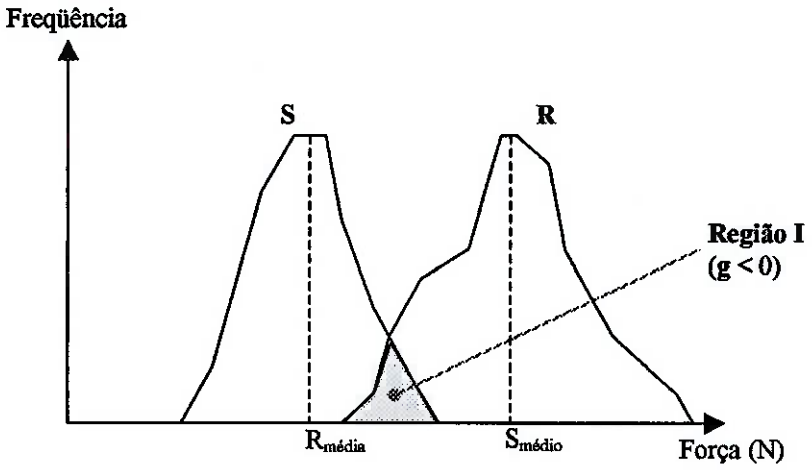

Figura 15: Histograma- Resistência x Solicitação.

A resistência, R, varia em torno de uma média $R_{média}$ e a solicitação, S, em torno de $S_{média}$. A interseção entre as duas regiões R e S fornece a região **I**. Nesta região, R é menor que S, e portanto, ocorre falha (g<0). A questão agora é calcular a área desta região, pois ela fornece dados sobre a ocorrência de falhas. Para calcular a área desta região, emprega-se o Método de Simulação Monte Carlo. Sem possuir uma fórmula analítica exata para o cálculo desta área, é possível chegar num valor bastante aproximado, através das simulações!

A fig.16 representa os valores das resistências e solicitações geradas aleatoriamente, dada as distribuições do parâmetros de entrada (Y, S, w e L). A taxa de falha é dado pela razão do número de pontos gerados internamente à região I (ou seja, $g_i < 0$) sobre o número total de pontos gerados (n_{total}):

$$\text{Taxa de falha} = n^{o}_{internos} / n^{o}_{total}$$

Assim, a confiabilidade do sistema estrutural será dado por:

$$\text{Confiabilidade} = 1 - \text{Taxa de falha} = 1 - n^{o}_{internos} / n^{o}_{total}$$

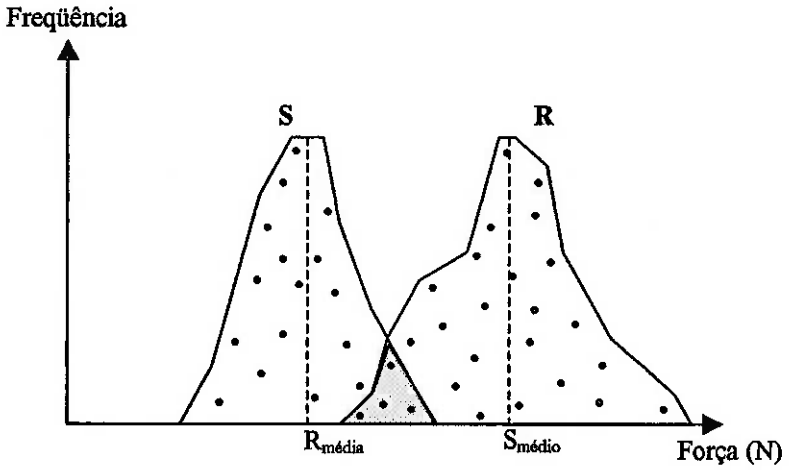

Figura 16: Representação das Simulações de Monte Carlo.

4.6.3 Arquitetura do programa McsEstrutural

A arquitetura do programa é representado pela fig.17 e o seu código fonte encontra-se em Anexo.

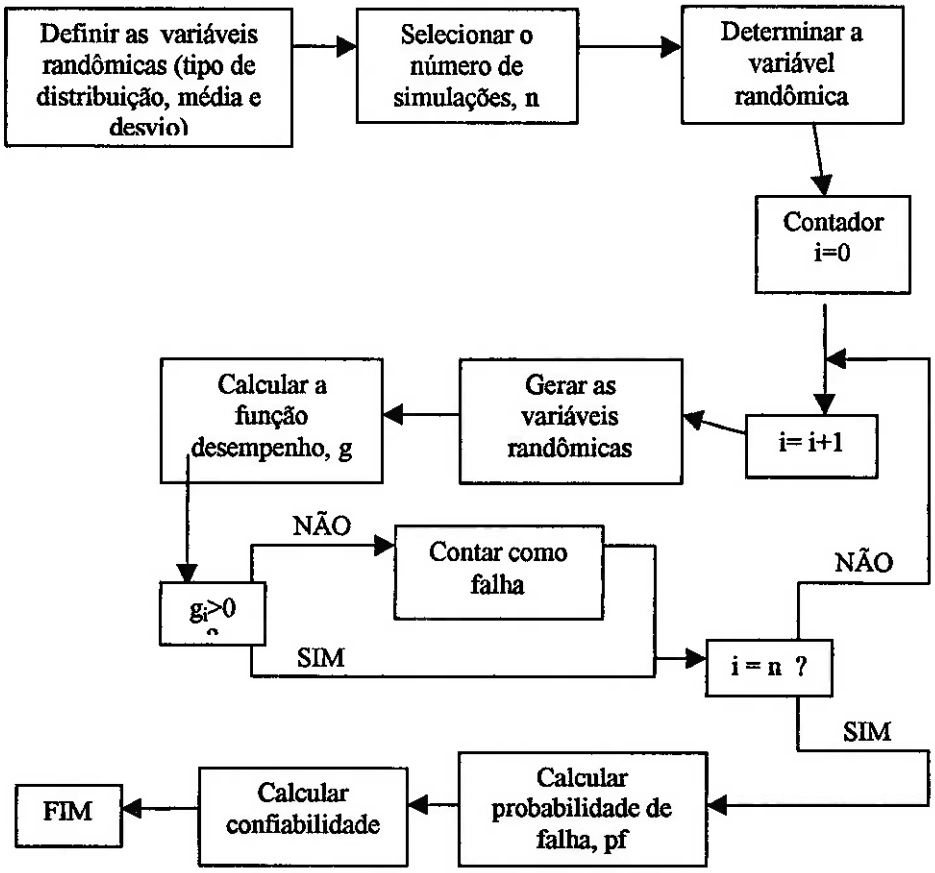

Figura 17: Arquitetura do programa McsEstrutural.

As variáveis aleatórias são geradas segundo métodos encontrados na biblioteca da CERN [15]. Estes métodos são baseados no gerador de números

aleatórios *MersenneTwister* [14], um eficiente algoritmo gerador de números disponível na Internet (a seqüência de números gerados só se volta a se repetir depois de 2^{64}.números!). Vale ressaltar que a eficiência da simulação Monte Carlo depende fundamentalmente da qualidade destes números. Os métodos fornecidos pela CERN oferecem grande facilidade para manuseio com diversos tipos de distribuições.

Também se desenvolveu um método substituto para a geração de distribuições *Gaussianas* (parâmetros de entrada), empregando-se o método de integração *Simpson* para calcular a *densidade probabilidade acumulada* da distribuição.

4.6.4 Método de Integração Simpson

Os números aleatórios na Simulação Monte Carlo precisam ser gerados segundo uma dada distribuição *Gaussiana*, pois assim ocorre na realidade para maioria das medidas mecânicas. Pode-se empregar métodos disponíveis na Internet, como os da CERN [15] ou é necessário desenvolver outros métodos.

Uma maneira de gerar estas variáveis aleatórias é gerar números aleatórios entre 0~1 e interpolar o valor correspondente na tabela da *Gaussiana* normalizada. E fazer as devidas correções para obter as variáveis nas dimensões adequadas. Uma outra maneira, a qual é utilizada no programa desenvolvido *McsEstruturalSimpson* (em Anexo), é a seguinte:

- Gerar um número aleatoriamente entre 0 e 1, **x**.
- Encontrar , por exemplo, o valor Y_x que possui a densidade probabilidade acumulada igual ao **x**. Para isto, utiliza-se o método de integração Simpson para integrar a função densidade probabilidade da variável Y.

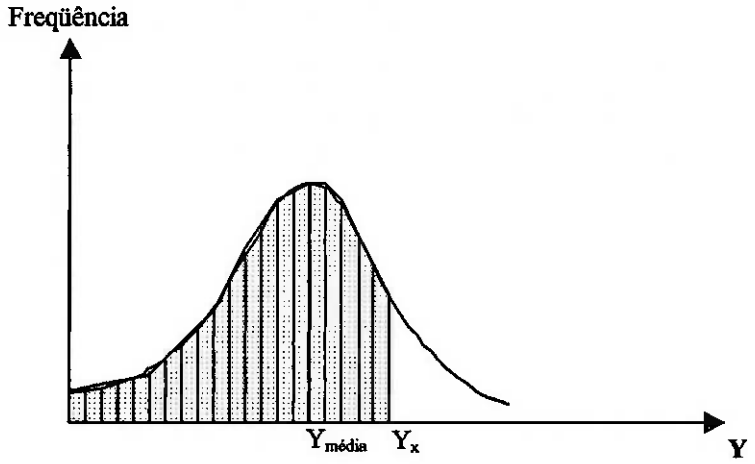

Figura 18: Processo de obtenção da variável aleatória Y_x.

Este processo é descrita na fig.18. A idéia do método de integração Simpson é utilizar, para um conjunto de 3 pontos, um polinômio interpolador de 2ª ordem, calculando-se a partir dela o valor aproximado da integral. A região a ser integrada é repartida em *n* partes. O valor da integral vai sendo obtida pelo acúmulo de áreas, de 3 em 3 pontos, progressivamente, até atingir o valor da probabilidade **x**. Quando atinge **x**, é porque se atingiu o valor Y_x procurado.

4.7. Método Gibbs Sampling

4.7.1. Aspectos Teóricos

Supondo um complexo modelo estatístico composto por um número de variáveis $X_1,...,X_n$. A rede *Bayesiana* é um exemplo deste modelo. Denotando por θ_t, o estado deste modelo num dado momento t, com todas as suas variáveis assumindo uma combinação de valores. Supondo agora que exista um método estocástico capaz de gerar novos valores, partindo do estado θ_t para θ_{t+1}; ter-se-á construído uma cadeia de Markov. Esta é a idéia principal do método *Gibbs sampling*.

Seja uma distribuição $p(X_i/\{ X_j\}_{j=1..n,j\neq i})$, a esta distribuição se dá o nome de probabilidade condicional total(*full conditional probability*)[7]. Para cada variável X_i, tem-se uma probabilidade condicional total, que contém todas as probabilidades condicionais contendo a variável X_i.

O método começa com um estado inicial qualquer, θ_0. Dado um estado θ_t, gera-se o estado θ_{t+1}, como mostrado a seguir:

1) X_1 gerado a partir de $p(X_1/X_2,...,X_n)$.

2) X_2 gerado a partir de $p(X_2/X_1,...,X_n)$.

i) X_i gerado a partir de $p(X_i/\{ X_j\}_{j=1..n,j\neq i})$.

n) X_n gerado a partir de $p(X_n/X_1,...,X_{n-1})$.

Não há necessidade de atualizar as variáveis numa ordem particular, pois, de qualquer forma, irão convergir para um estado final.

Assim, a partir de θ_0.se gera inúmeras amostras, até atingir uma amostra que não mudará mais. Armazena-se esta amostra final, e obtém-se as estimativas de interesse.

O método *Gibbs sampling* pode ser facilmente implementado no contexto de redes *Bayesianas*, pois a probabilidade condicional total é dado como sendo proporcional à produtória das probabilidades condicionais desta rede[7]:

$$Q(X_i) \propto p(X_i/pa\{X_i\}).\Pi_j p(Y/pa\{Y\})$$

Onde j percorre todas as variáveis que são filhos de X_i. Ou seja, para cada X_i, a sua probabilidade condicional total envolve probabilidade dos pais, dos filhos e dos pais destes filhos. É o produto de todas as probabilidades que se refira a X_i.

Com uma simples idéia, os métodos de MCMC(*Gibbs sampling* é um destes) possuem um enorme potencial para resolução de cálculos estatísticos. Sendo necessário apenas que sejam desenvolvidos e implementados. Maiores detalhes são encontrados na referência[8].

4.7.2. Resolvendo o exemplo DogProblem

O *Gibbs sampling* ficará mais claro com a resolução do exemplo DogProblem:

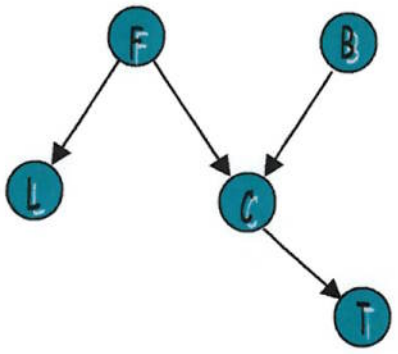

Figura 19: DogProblem.

A probabilidade condicional total para cada variável é dada por:

- p_F=p(L/F)
- p_L=p(L/F).p(F).p(C/F,B)
- p_C=p(C/F,B).p(T/C)
- p_B=p(B).p(C/F,B)
- p_T=p(T/C)

Note que, cada probabilidade condicional total é o produto de todas as probabilidades que contem a variável em questão. Estas probabilidades condicionais totais fornecerão uma amostra para cada variável.

Para cada iteração, as variáveis são percorridas; sendo que para cada uma, calcula-se a probabilidade condicional total, deixando todas as demais variáveis fixas (como se fossem evidências). Desta probabilidade total faz se a amostragem do valor assumido pela variável (verdadeiro ou falso).

Supondo-se que n iterações são realizadas, obtêm-se a seguinte tabela:

Iteração	L	F	C	B	T
θ_1	L=1	F=0	C=0	B=0	T=1
θ_2	L=1	F=0	C=1	B=0	T=1
θ_3	L=1	F=1	C=1	B=1	T=1
θ_4	L=1	F=0	C=1	B=0	T=1
θ_5	L=1	F=1	C=0	B=1	T=1
...	...	...	...	...	...
θ_n	L=1	F=1	C=1	B=1	T=1

Obs.: Note que, o valor das evidências(L=1 e T=1) sempre são mantidas fixas.

Esta tabela fornece n amostras, sendo que para cada amostra teremos uma distribuição de valores, para cada variável.

Logo, para achar a probabilidade de F, basta pegar as n amostras de F e calcular:

$$P(F=0)= (n^{os} \text{ de "0"}) / n$$

Ou seja, se são realizadas 100 iterações, e são computados 15 zeros; a probabilidade de que a família não esteja em casa é de apenas 15%. E, portanto, o ladrão deverá deixar a tentativa de furto para uma outra ocasião!

4.7.3. Implementação do Algoritmo

O algoritmo de *Gibbs sampling* foi implementado em *Java*. E se encontra em ANEXO como *GibbsSamplig*.

O programa possui a seguinte estrutura:

- Para cada variável (Xi), seleciona as probabilidades condicionais que contenham Xi. Ou seja, cada variável F, L, T, C e B possui sua expressão para probabilidade condicional total.
- Para cada variável (Xi), multiplica todas as probabilidades, obtendo a probabilidade condicional total (pct) para cada uma delas. Ao final, normaliza esta probabilidade.
- Para cada iteração:
  - Para cada variável Xi:
    - Seleciona a probabilidade condicional total de Xi;
    - Faz a amostragem aleatória de Xi;
  - Contabiliza o número de vezes que a variável requerida assume um valor (0 ou 1);
- Calcula a probabilidade para a variável requerida, a partir das amostragens obtidas.

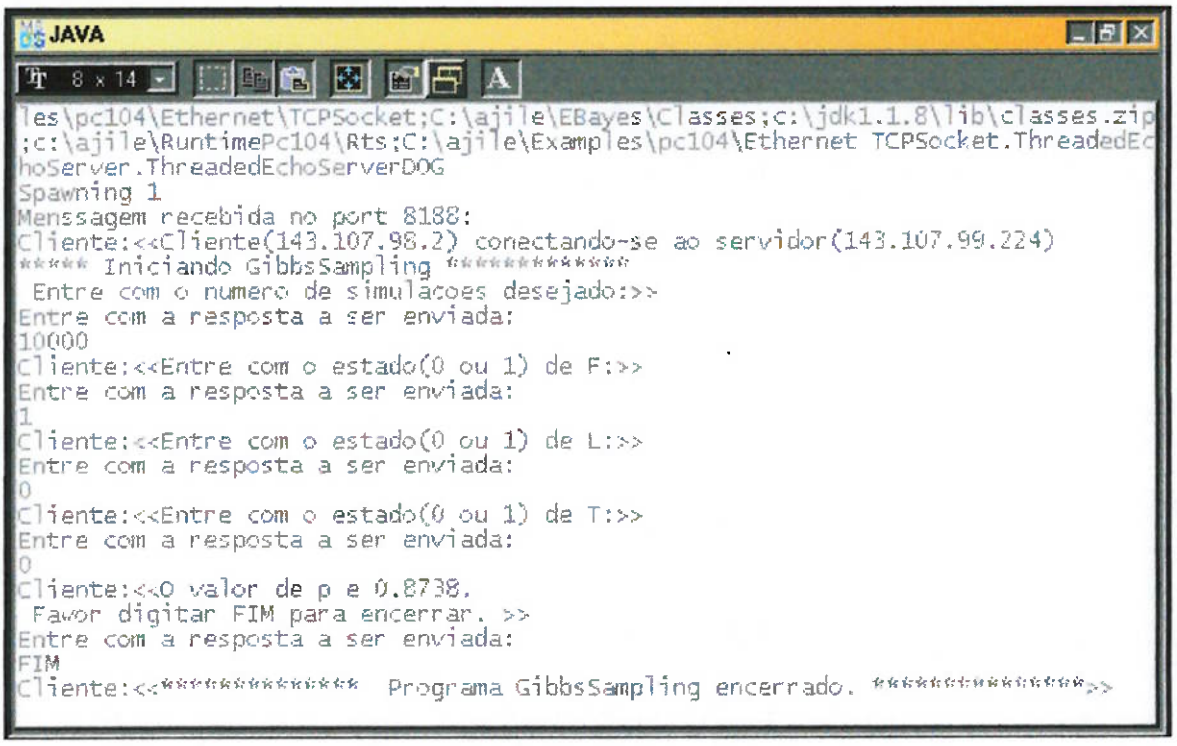

Figura 20: Apresentação do programa GibbsSampling.

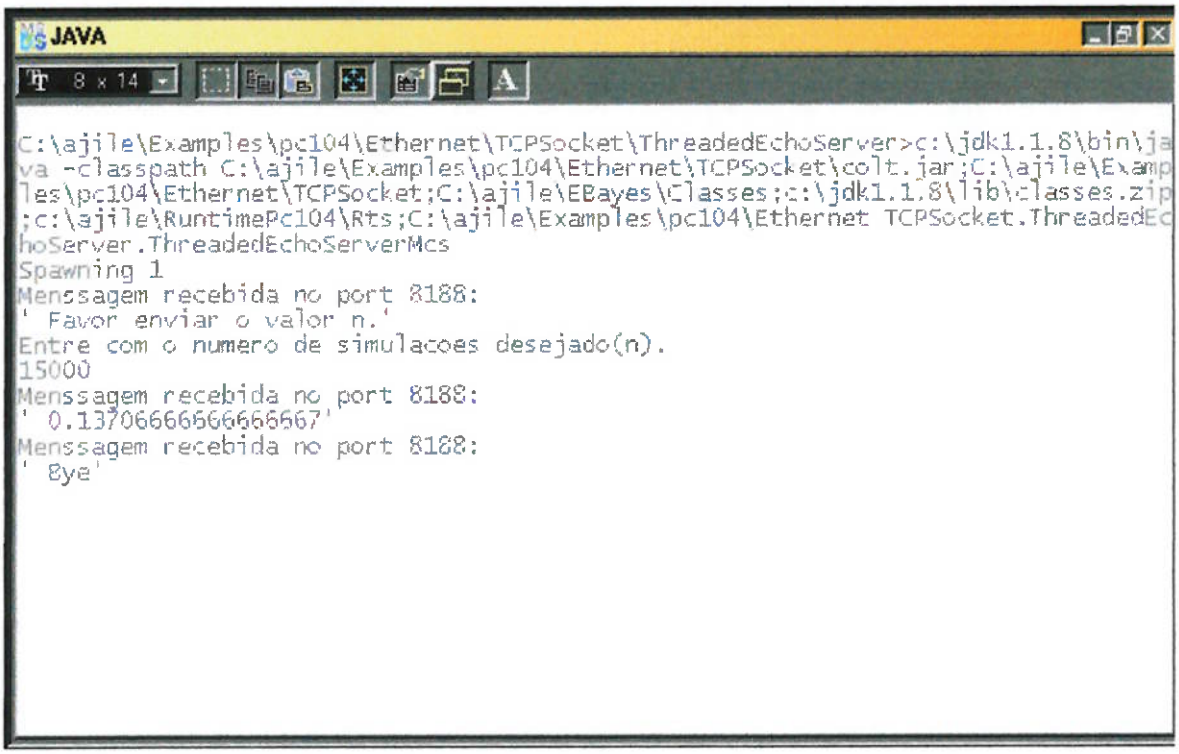

Figura 21: Apresentação do programa McsEstrutural.

5. APRESENTAÇÃO DOS RESULTADOS

5.1. Cálculo da Confiabilidade

	Variável	Média	Desvio	Tipo de Distribuição
Dados de	Y	44	4,4	Gaussiana
Entrada	S	128,2	6,4	Gaussiana
	W	7	1,75	Gaussiana
	L	20	1,0	Gaussiana

	Número de simulações	Confiabilidade
Dados	100	0,84
Obtidos	1000	0,847
(*McsEstrutural*)	10.000	0,8651
	10.000	0,8645

	Número de Divisões (Simpson)	Número de simulações	Confiabilidade
Dados	100	100	0,82
Obtidos	100	1000	0,86
(*McsEstrutural*	100	10.000	0,8585
Simpson)	1000	100	0,85
	1000	1000	0,869
	1000	10.000	0,866

Dado de Referência [13]: Confiabilidade=0,8681.

5.2. *Cálculo da Probabilidade Condicional*

Os resultados obtido são do programa *GibbsSampling*

F	L	T	Prob. Exata (JavaBayes)	Gibbs n=100	Gibbs n=1000	Gibbs n=10.000	Gobbs n=100.000
0	0	0	0,12456	0,15	0,12	0,1262	0,12563
0	0	1	0,04	0,02	0,037	0,0407	0,03987
0	1	0	0,6576	0,65	0,674	0,6609	0,65696
0	1	1	0,36	0,30	0,355	0,3733	0,35706

6. *DISCUSSÃO DOS RESULTADOS*

Obteve-se resultados muito satisfatórios em ambos os programas. No cálculo de confiabilidade, tanto o *McsEstrutural*, como o *McsEstruturalSimpson* fornecem resultados muito aproximados do resultado dado pela referência (0,8681) para 10 mil simulações. A diferença entre os dois é que o *McsEstruturalSimpson* é muito mais lento, pois utiliza-se do método de Simpson para calcular a função densidade probabilidade acumulada.

No cálculo da probabilidade condicional (inferência na rede Bayesiana) obteve-se resultados bastante razoáveis com apenas 1000 simulações. Sendo que para 10 mil simulações, o resultado já fica bastante refinado.

Com relação aos testes efetuados no sistema embarcado, obteve-se boa comunicação entre o aJ-PC104 (sistema remoto) e o computador PC (servidor), para redes locais. Enfrentando-se alguns problemas, quando a comunicação passa por um roteador.

7. CONCLUSÃO

Os resultados obtidos foram bastante satisfatórios. Desenvolveu-se um programa de análise de confiabilidade, o *McsEstrutural*, que poderá ser empregado na simulação de diversos sistemas, não apenas estruturais, como desenvolvido neste trabalho. Sendo necessário para isto apenas algumas adaptações.

Já o programa *GibbsSampling*, fornece resultados muito bons de inferências numa rede Bayesiana com 5 nós. O programa é específico para 5 nós, mas pode ser alterado facilmente, de acordo com o emprego específico que se fizer dela. O objetivo principal foi atingido: " desenvolver e implementar métodos numéricos (ferramentas) para solucionar problemas estatísticos num sistema embarcado" .

A idéia inicial era desenvolver um programa que fosse adicionado ao programa *JavaBayes*, mas isto se mostrou complexa. Optou-se por desenvolver as próprias ferramentas que seriam empregadas no sistema embarcado.

O emprego do sistema embarcado aJ-PC104 foi bastante interessante. Pois, foi possível manusear esta nova tecnologia que vem satisfazer as emergentes necessidades do mundo da Internet. Acredita-se que foram dados os primeiros passos para o emprego destas novas tecnologias no ambiente industrial de produção e de automação.

As aulas de pós-graduação, sobre "Teoria de Probabilidades em Inteligência Artificial e Robótica", foram fundamentais para a compreensão dos métodos estatísticos utilizados.

Como se pode observar, a teoria de Probabilidade é uma grande ferramenta para se estar resolvendo problemas envolvendo incertezas e tomada de decisões. E, com o advento de novas tecnologias (ambiente fornecido pelo *Java*), é possível estar realizando os cálculos estatísticos, através de algoritmos computacionais, em sistemas que poderão ser embarcados em diversos tipos de máquinas e robôs.

8. REFERÊNCIAS BIBLIOGRÁFICAS

[1]. Author: Fabio Cozman,< fgcozman@usp.br> <http://www.cs.cmu.edu/~fgcozman/home.html >.

[2]. < http://www.ajile.com/aj-pc104_press_release.html >.

[3]. http://*Java*.sun.com.

[4]. Disciplina de Pós-Graduação, 1°. período, PMR-5102:: *Teoria da Probabilidade em Inteligência Artificial e Robótica*. Prof. Fábio G. Cozman.

[5]. Krause, Paul J. (1998). *Learning Probabilistic Networks*. Philips Research Laboratories. Crossoak Lane. Redhill, Surrey RH1 5HA. United Kingdom.

[6]. Cozman, Fábio G. (1998). *Redes Bayesianas para tomada de decisões,* Anais CONAI'98, pp. 258-261.

[7]. Cozman, Fábio G. (1999). *Sensitivity and Robustness Analysis of Bayesian Network*, 4°.SBAI - Simpósio Brasileiro de Automação Inteligente, pp. 251-255.

[8]. W.R. Gilks, R. Sylvia and D.J. Spiegelhalter (1996). *Introducing Markov Chain Monte Carlo*, pp. 1-16.

[9]. J.Pearl(1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kauffman.

[10] Metropolis, N. and Ulam, S., *The Monte Carlo methods,* J. Am. Stat. Assoc., 44, n247, 335-341, 1949.

[11] Sobol, Ilya M.(1994), *A Primer for Monte Carlo methods,* CRC Press, Inc., Florida.

[12] Sundararajan, C., *Probabilistic Structural Mechanics Handbook,* Ed. Chapman&Hall, 53-59.

[13] Kamal, Hasan and Ayyub, Bilal, *Reliability Assessment of Structural Systems Using Discrete-Event Simulation,* University of Maryland, 1999.

[14] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 1, January 1998, pp 3-30. <http://www.math.keio.ac.jp/matumoto/emt.html >.

[15] Biblioteca da CERN (Centro Europeu de Pesquisas Nucleares) de classes Java disponíveis na Internet <http://www.cern.ch/RD11/rkb/AN16pp/node188.html>

9. ANEXO

```
1  //=============================================================
2  //     Copyright (c) 2000, Escola Politécnica-USP
3  //                    All Rights Reserved
4  //=============================================================
5  //
6  // PACKAGE NAME : appGibbsSampling
7  // TARGET       : AJ-PC104 board
8  // @DATE        : 04/12/2000
9  // @AUTHOR      : Jaime Shinsuke Ide
10 //                jside@bol.com.br
11 //=============================================================
12
13 // O programa roda na placa aJ-PC104(cliente:143.107.98.2) e se comunica com o servidor,
14 // com endereço IP "143.107.99.224".
15
16 // Este programa utiliza:
17 //   Para geração de números aleatórios-MersenneTwister.
18 //   Bibliotecas de manuseio com sistema embarcado aJ-PC104.
19 //   Rede Bayesiana com 5 nós.
20 //   Método Gibbs Sampling para realização de inferências.
21
22 package pc104.Ethernet.TCPSocket;
23
24 import java.io.*;
25 import java.net.*;
26 import com.ajile.jem.rawJEM;
27 import com.ajile.jem.SystemOut;
28 import java.lang.System;
29 import java.net.Socket;
30 import java.io.OutputStream;
31 import java.util.*;
32 import java.lang.*;
33 import edu.cornell.lassp.houle.RngPack.RandomElement;
34 import cern.jet.stat.Probability;
35 import cern.jet.random.*;
36 import MersenneTwister;
37
38 public class appGibbsSampling {
39
40   static int msgID = 1;
41   static int maxPktSize = 128;
42   // Métodos que retornam os valores das probabilidades condicionais.
43   public double pF (int f){
44     double f0=0.15;
45     double f1=0.85;
46     if (f == 0) return f0;
47     else return f1;
48     }
49   public double pLF (int l, int f){
50     double l0f0=0.4;
51     double l0f1=0.9;
```

```
52      double l1f0=0.6;
53      double l1f1=0.1;
54      if (l == 0) {
55          if (f==0) return l0f0;
56          else return l0f1;
57          }
58      else {
59          if (f==0) return l1f0;
60          else return l1f1;
61          }
62      } // final do pLF.
63    public double pTC (int t, int c){
64      double t0c0=0.7;
65      double t0c1=0.05;
66      double t1c0=0.3;
67      double t1c1=0.95;
68      if (t == 0) {
69          if (c==0) return t0c0;
70          else return t0c1;
71          }
72      else {
73          if (c==0) return t1c0;
74          else return t1c1;
75          }
76      } // final do pTC.
77    public double pCFS (int c, int f, int s){
78      double c0f0s0=0.8;
79      double c0f0s1=0.95;
80      double c0f1s0=0.2;
81      double c0f1s1=0.5;
82      double c1f0s0=0.2;
83      double c1f0s1=0.05;
84      double c1f1s0=0.8;
85      double c1f1s1=0.5;
86      if (c == 0) {
87          if (f==0) {
88              if (s==0) return c0f0s0;
89              else return c0f0s1;
90              }
91          else {
92              if (s==0) return c0f1s0;
93              else return c0f1s1;
94              }
95          }
96      else {
97          if (f==0) {
98              if (s==0) return c1f0s0;
99              else return c1f0s1;
100             }
101         else {
102             if (s==0) return c1f1s0;
```

```
103             else return c1f1s1;
104             }
105         }
106     } // final do pLF.
107   public double pS (int s){
108     double s0=0.05;
109     double s1=0.95;
110     if (s == 0) return s0;
111     else return s1;
112     }
113   // Método que dado a e b devolve o valor de a normalizado.
114   public double normaliza (double a, double b) {
115     double aNormalizado=a/(a+b);
116     return aNormalizado;
117     }
118
119   public static void main(String[] args) {
120     appGibbsSampling appObj = new appGibbsSampling();
121     appObj.run();
122   }
123   void run() {
124     MersenneTwister rn;
125     appGibbsSampling dog = new appGibbsSampling();
126     double n; //número de simulações.
127     double drn; // double random number.
128     rn = new MersenneTwister(1);// random number.
129     int eL=0,eF=0,eC=0,eS=0,eT=0; // Declaração das variáveis de estado(0 ou 1).
130     double pctL0,pctF0,pctC0,pctS0,pctT0=0; // Probabilidades Condicionais Totais(pct) para estado 0.
131     double pctL1,pctF1,pctC1,pctS1,pctT1=0; // pct para estado 1.
132     double p; //Probabilidade Inferida.
133     int eQ=0; //Estado da variável inferida(Query).
134     int eEvidenciaL=0,eEvidenciaT=0; // Estado das evidências.
135     String exit="blabla";
136     boolean parar=false;
137    try {// Tenta estabelecer a conexão.
138       Socket client = new Socket(InetAddress.getByName("143.107.99.224"), 8188);
139       InputStream  is  =  client.getInputStream();
140       OutputStream out =  client.getOutputStream();
141       String menss1 = "Cliente(143.107.98.2) conectando-se ao servidor(143.107.99.224)\n***** Iniciando GibbsSampling *************\n Ent
142       String menss3 = "Entre com o estado(0 ou 1) de F:";
143       String menss4 = "Entre com o estado(0 ou 1) de L:";
144       String menss5 = "Entre com o estado(0 ou 1) de T:";
145       String menss7 = "**************  Programa GibbsSampling encerrado. ***************";
146       String bye = "\n Favor digitar FIM para encerrar. ";
147       String byeS="...";
148       byte[] inBuf  = new byte[128];
149    while (!parar)
150     {
151       out.write(menss1.getBytes(), 0, menss1.length());
152       int numBytes = is.read(inBuf, 0, 128);
153       String outstr = new String(inBuf, 0, numBytes);
```

```
154         n=(new Integer(outstr)).doubleValue();
155
156         out.write(menss3.getBytes(), 0, menss3.length());
157         numBytes = is.read(inBuf, 0, 128);
158         outstr = new String(inBuf, 0, numBytes);
159         eQ=(new Integer(outstr)).intValue();
160
161         out.write(menss4.getBytes(), 0, menss4.length());
162         numBytes = is.read(inBuf, 0, 128);
163         outstr = new String(inBuf, 0, numBytes);
164         eEvidenciaL=(new Integer(outstr)).intValue();
165
166         out.write(menss5.getBytes(), 0, menss5.length());
167         numBytes = is.read(inBuf, 0, 128)
168         outstr = new String(inBuf, 0, numBytes);
169         eEvidenciaT=(new Integer(outstr)).intValue();
170
171         System.out.println(n);
172         System.out.println(eQ);
173         System.out.println(eEvidenciaL);
174         System.out.println(eEvidenciaT);
175         //Início das simulações.
176       double neF=0; //Número de vezes que o Estado de F vale 0.
177       double j=0;
178       while (j<n) {
179           // Atualização das pct, seguida das atualizações de estado.
180           // Evidências não variam!!
181           eL=eEvidenciaL;
182           eT=eEvidenciaT;
183           // Probabilidade Condicional Total de L.
184           pctL0=dog.pLF(0,eF);
185           pctL1=dog.pLF(1,eF);
186           pctL0=dog.normaliza(pctL0,pctL1);
187           // Probabilidade Condicional Total de F.
188           pctF0=dog.pLF(eL,0)*dog.pF(0)*dog.pCFS(eC,0,eS);
189           pctF1=dog.pLF(eL,1)*dog.pF(1)*dog.pCFS(eC,1,eS);
190           pctF0=dog.normaliza(pctF0,pctF1);
191           drn = rn.nextDouble();
192           if (drn>pctF0) eF=1;
193           else eF=0;
194           if (eF==eQ) neF++; // Incrementa se estado de F assume eQ.
195           // Probabilidade Condicional Total de C.
196           pctC0=dog.pCFS(0,eF,eS)*dog.pTC(eT,0);
197           pctC1=dog.pCFS(1,eF,eS)*dog.pTC(eT,1);
198           pctC0=dog.normaliza(pctC0,pctC1);
199           drn = rn.nextDouble();
200           if (pctC0 < drn) eC=1;
201           else eC=0;
202           // Probabilidade Condicional Total de S.
203           pctS0=dog.pS(0)*dog.pCFS(eC,eF,0);
204           pctS1=dog.pS(1)*dog.pCFS(eC,eF,1);
```

```
205           pctS0=dog.normaliza(pctS0,pctS1);
206           drn = rn.nextDouble();
207           if (pctS0 < drn) eS=1;
208           else eS=0;
209           // Probabilidade Condicional Total de T.
210           pctT0=dog.pTC(0,eC);
211           pctT1=dog.pTC(1,eC);
212           pctT0=dog.normaliza(pctT0,pctT1);
213           j++;
214           } // Final do while.
215
216           p=neF/n; //Cálculo da  probabilidade requerida.
217           System.out.println("p="+p+", para "+n+"simulacoes.");//(Opcional) Para enviar o resultado ao Charade.
218         // Procedimento para transformar pf num string
219         String resultado = String.valueOf(p);
220         String envioResultado = "O valor de p e " + resultado+"."+bye;
221         // Menssagem 3: Envia resultado C-S
222         out.write(envioResultado.getBytes(), 0, envioResultado.length());
223         // Mensagem 4 e 5: Envia menssagem de despedida C-S
224         // Mas, só para de enviar as menssagens 1~4, quando obtém uma resposta de despedida!
225         // out.write(bye.getBytes(), 0, bye.length());
226         numBytes = is.read(inBuf, 0, 128);
227         byeS = new String(inBuf, 0, numBytes);
228         if (byeS.trim().equals("FIM")) parar=true;
229        } //Fim do while.
230         out.write(menss7.getBytes(), 0, menss7.length());
231         client.close();
232
233       } // fim do try
234       catch (Exception e) {
235         System.out.println("Unexpected program exception.");
236         rawJEM.breakPoint(0xff);
237       }
238     }
239   }
240
241
```

```java
//=============================================================
//     Copyright (c) 2000, Escola Politécnica-USP
//                    All Rights Reserved
//=============================================================
//
// PACKAGE NAME : ThreadedEchoHandlerGibbs
// TARGET       : AJ-PC104 board
// @DATE        : 04/12/2000
// @AUTHOR      : Jaime Shinsuke Ide
//
//=============================================================

// Este programa roda num PC que está na mesma rede que a do sistema
// embarcado aJ-PC104, comunicando-se com ela como sendo o servidor.

// Recebe os pacotes TCP de aJ-PC104.
// Cria distintos "thread" para cada conexão Socket.
// A classe "ThreadedEchoHandlerGibbs" representa o "thread" que é criado.

package TCPSocket.ThreadedEchoServer;

import java.io.*;
import java.net.*;

public class ThreadedEchoHandlerGibbs extends Thread {
  private Socket incoming;
  private int counter;
  public ThreadedEchoHandlerGibbs(Socket i, int c) {
    incoming = i;
    counter = c;
  }
  public void run() {
    try {
      InputStream in = incoming.getInputStream();
      OutputStream out = incoming.getOutputStream();
      DataInputStream dis = null;
      DataInputStream dis1 = null;
      boolean done = false;
      byte[] inbuf = new byte[1024];

      System.out.println("Menssagem recebida no port 8188: ");
      while (!done) {
        String line,resp=null;
        int numBytes = in.read(inbuf);
        if (numBytes < 0) done = true;
        else
        { // Procedimento default que imprime na tela as menssagens recebidas.
          String outstr = new String(inbuf, 0, numBytes);
          System.out.println("Cliente:<<" + outstr + ">>");
          if (outstr.trim().equals("**************  Programa GibbsSampling encerrado. ***************")) done=true;
          else{
```

```
52              System.out.println("Entre com a resposta a ser enviada:");
53              dis = new DataInputStream(System.in);
54              line = dis.readLine();
55              out.write((line).getBytes());
56              } // fim do else.
57            }//fim do else
58            }// fim do while
59          incoming.close();
60        }//fim do try
61        catch (Exception e) { System.out.println(e); }
62    }
63  }
64
65
```

```java
//=============================================================
//     Copyright (c) 2000, Escola Politécnica-USP
//                    All Rights Reserved
//=============================================================
//
// PACKAGE NAME : ThreadedEchoServerGibbs
// TARGET       : AJ-PC104 board
// @DATE        : 04/12/2000
// @AUTHOR      : Jaime Shinsuke Ide
//
//=============================================================

// Este programa roda num PC que está na mesma rede que a do sistema
// embarcado aJ-PC104, comunicando-se com ela como sendo o servidor.

// Recebe pacotes TCP de aJ-PC104.
// A classe "ThreadedEchoServerGibbs" gera distintos "thread"
// (os ThreadedEchoHandlerGibbs) para cana conexão Socket.

package TCPSocket.ThreadedEchoServer;

import java.io.*;
import java.net.*;
import java.lang.Integer;///
import java.util.StringTokenizer;///

public class ThreadedEchoServerGibbs  {

  public static void main(String[] args) {
    DataInputStream dis = null;// Faz parte da leitura de dados.
    try {
      ServerSocket s = new ServerSocket(8188); // Cria um Socket na porta 8188.
      // Gera um novo "thread"para cada conexão.
      for (int i = 1;;) {
        Socket incoming = s.accept();
        System.out.println("Gerando: " + i);
        new ThreadedEchoHandlerGibbs(incoming, i).start();
        i++;
      }
    }
    catch (Exception e) { System.out.println(e);}
  }
  private Socket incoming;
  private int counter;
}
```

```java
//============================================================
//     Copyright (c) 2000, Escola Politécnica-USP
//                    All Rights Reserved
//============================================================
//
// NAME : McsEstrutural
// @DATE        : 04/12/2000
// @AUTHOR      : Jaime Shinsuke Ide
//                jside@bol.com.br
//============================================================

// O programa calcula a confiabilidade de um sistema estrutural.
// Este programa possui as seguintes ferramentas:
// -Geração de números aleatórios: MersenneTwister
// -Curvas de entrada: Gaussiana, utilizando a biblioteca da CERN para obter a FDP.

import java.io.*;
import java.util.StringTokenizer;
import java.util.*;
import java.lang.Integer;
import java.lang.*;
import edu.cornell.lassp.houle.RngPack.RandomElement;
import cern.jet.stat.Probability;
import cern.jet.random.*;

public class McsEstrutural {

public static void main(String args[])
{ Normal y,s,w,l;
  DataInputStream dis = null;//
  double dy,ds,dw,dl;
  double n,nf; //Número de ciclos de simulação, e de falhas.
  double gm;
  double pf; //Probabilidade de falha
  cern.jet.random.engine.RandomEngine engine_y = new cern.jet.random.engine.MersenneTwister();
  cern.jet.random.engine.RandomEngine engine_s = new cern.jet.random.engine.MersenneTwister(1);
  cern.jet.random.engine.RandomEngine engine_w = new cern.jet.random.engine.MersenneTwister(2);
  cern.jet.random.engine.RandomEngine engine_l = new cern.jet.random.engine.MersenneTwister(3);
  y= new Normal(44.0,4.4,engine_y);
  s= new Normal(128.2,6.4,engine_s);
  w= new Normal(7.0,1.75,engine_w);
  l= new Normal(20.0,1.0,engine_l);

  try {
        if (args.length > 0)
            dis = new DataInputStream(new FileInputStream(args[0]));
        else
            dis = new DataInputStream(System.in);
    } catch (IOException e) {
        System.exit(0);
    }
```

```
52    String line=null, command, name = null, value = null;
53    StringTokenizer st;
54    String exit="blabla";
55    // Inicia um looping até que o usuário digite "sair".
56    System.out.println();
57    System.out.println("***** Iniciando McsEstrutural************");
58    while (!exit.trim().equals("sair"))
59      {
60        try {
61          System.out.println("Entre com o numero de simulacoes desejado:");
62          line = dis.readLine();
63          } catch (Exception e) { System.out.println(e); }
64
65    n=(new Integer(line)).doubleValue();
66    System.out.println("Numero de simulacoes:" + line);
67    nf=0;
68    double coefVar=0;
69    for (int j=0;j<n;j=j+1)
70         {
71          dy = y.nextDouble();
72          ds = s.nextDouble();
73          dw = w.nextDouble();
74          dl = l.nextDouble();
75
76          gm = ((dy * ds)/12)-((dw * dl * dl)/8);
77          if (gm < 0.0) nf++;
78          }
79    pf=nf/n;
80    double confiabilidade=1-pf;
81    coefVar=Math.sqrt((1-pf)*pf/n)/pf;
82    System.out.println("Probabilidade de falha, pf:"+pf);
83    System.out.println("A confiabilidade vale "+ confiabilidade+" para "+(n)+" simulacoes.");
84    System.out.println("Coef. Variancia, COV:"+coefVar+"para"+(n)+"simulacoes.");
85    try {
86          System.out.println("Digite 'sair' para encerrar o programa.");
87          line = dis.readLine();
88          } catch (Exception e) { System.out.println(e); }
89    exit=line;
90
91    } // Final do while.
92    System.out.println("************Programa McsEstrutural encerrado.**************");
93
94    }
95    }
96
```