



AVALIAÇÃO DE DESEMPENHO POR MEIO DA FERRAMENTA MULTIMODELOS SPNP

Trabalho de Conclusão de Curso

Engenharia da Computação

Iago Alves da Silva
Orientador: Sérgio Murilo Maciel Fernandes



**UNIVERSIDADE
DE PERNAMBUCO**

**Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação**

IAGO ALVES DA SILVA

**AVALIAÇÃO DE DESEMPENHO POR
MEIO DA FERRAMENTA
MULTIMODELOS SPNP**

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, outubro, 2011.

MONOGRAFIA DE FINAL DE CURSO

Avaliação Final (para o presidente da banca)*

No dia 21 de Dezembro de 2011, às 11:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente IAGO ALVES DA SILVA, orientado pelo professor Sérgio Murilo Maciel Fernandes, sob título Avaliação de desempenho por meio da ferramenta multimodelos spnp, a banca composta pelos professores:

Maria Lencastre Pinheiro de Menezes Cruz

Sérgio Murilo Maciel Fernandes

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

Aprovada Aprovada com Restrições* Reprovada

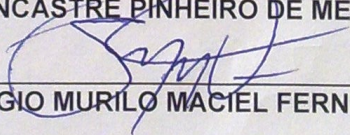
e foi-lhe atribuída nota: 8,0 (0,00)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 10 dias para entrega da versão final da monografia a contar da data deste documento.



MARIA LENCASTRE PINHEIRO DE MENEZES CRUZ



SÉRGIO MURILO MACIEL FERNANDES

De acordo

Recife

____/____/____

Sérgio Murilo Maciel Fernandes

*Dedico essa monografia
aos meus pais e ao meu irmão.*

Agradecimentos

Agradeço a toda minha família pelo apoio dado em todos os momentos. Agradeço ao meu orientador, e amigo, que foi bastante atencioso e prestativo desde antes do início oficializado da escrita desta monografia. É recompensante trabalhar sob a orientação de quem gosta do que faz, e que tem amplo conhecimento em diversas áreas além da área de atuação. Agradeço também aos amigos mais próximos por compreenderem minha ausência nos meses de dedicação à escrita e pesquisa, e também aos amigos que, apesar da distância, me deram grande apoio ajudando como puderam. Estendo os agradecimentos a todos os professores, funcionários e companheiros de turma da UPE.

*People used to believe that “software never breaks”
Jiantao Pan*

Resumo

Este trabalho apresenta um estudo sobre a ferramenta SPNP-Gui, abordando a configuração do seu ambiente de modelagem, a criação de redes de Petri, e uma análise sobre os resultados que podem ser alcançados através do uso da mesma. O objetivo foi fazer uma descrição em forma de tutorial visando facilitar o uso dessa ferramenta. No decorrer do documento é mostrado como a SPNP-GUI é capaz de abstrair, para o usuário, a parte matemática do modelo e como ela facilita, por exemplo, a geração de códigos-fonte baseados no modelo e integração desses modelos (com funções personalizadas em C). Também é mostrado como a ferramenta pode ser utilizada para estudos de avaliação de desempenho, para comparação com dados gerados matematicamente, como cálculo de marcações tangíveis usando cadeias de Markov. É feita uma comparação entre o SPNP-GUI e uma ferramenta similar, o *TimeNET*, e é demonstrado o porquê da escolha do SPNP-Gui. Por fim, são apresentados exemplos de geração de gráficos, geração de dados de análises feitas nos modelos de exemplo, avaliação de desempenho de *software* e de confiabilidade de um modelo *hardware-software* demonstrando a viabilidade do uso científico da ferramenta.

Abstract

This work presents a study about the SPNP-Gui modeling tool, about the configuration of its modeling environment, Petri nets creation, and analysis of the results that can be created using this tool. The objective was to create a description in tutorial look like, trying to make easier the learning of this tool. Throughout the document is shown as the SPNP-Gui can abstract, for the user, the math of the model and how it facilitates, for example, the generation of source-code based on the model and the integration of these models (with custom functions in C). It also shows how the tool can be used for performance evaluation studies, for comparison of with data generated mathematically, as the calculation of tangible markings using Markov chains. It's made a comparison between the SPNP-Gui and a similar tool, the TimeNET, and it's shown why the choice of SPNP-Gui. Finally it's show examples of graphic generation, data generation of analysis made in example models, software performance evaluation and reliability of a hardware-software model demonstrating the feasibility of using the tool.

Sumário

1Capítulo	1
Introdução.....	1
1.1Contextualização.....	2
1.2Proposta.....	3
2Capítulo	2
Conceitos e terminologia.....	4
2.1Sistemas.....	4
2.2Modelos.....	5
2.3Processos.....	5
2.4Modelos gráficos.....	6
3Capítulo	3
Redes de Petri.....	9
3.1Rede de Petri básica.....	9
3.2Ferramentas.....	10
4Capítulo	4
Ferramenta SPNP-Gui.....	13
4.1Método.....	13
4.2Adquirindo.....	13
4.3Utilizando a ferramenta.....	16
4.4Analisando a rede.....	21
4.5Validação.....	27

5Capítulo	5
Conclusão e Trabalhos Futuros.....	35
5.1Conclusão.....	35
5.2Trabalhos futuros.....	36
6Bibliografia.....	39
7Apêndice	A
Códigos C de redes de Petri.....	41

Índice de Figuras

Ilustração 1: Exemplo de diagrama de Venn.....	6
Ilustração 2: Exemplo de diagrama de blocos.....	7
Ilustração 3: Exemplo de cadeia de Markov.....	7
Ilustração 4: Exemplo de rede de Petri.....	8
Ilustração 5: Rede de Petri com arco de valor 2.....	11
Ilustração 6: Novo projeto.....	17
Ilustração 7: Botões da interface.....	18
Ilustração 8: Exemplo de modelo.....	18
Ilustração 9: Propriedade de um lugar.....	19
Ilustração 10: Propriedades de uma transição.....	20
Ilustração 11: Outro exemplo de rede de Petri.....	21
Ilustração 12: Janela de análise de rede.....	22
Ilustração 13: Exemplo de análise.....	23
Ilustração 14: Exemplo de erro comum.....	24
Ilustração 15: Janela de geração de gráfico.....	26
Ilustração 16: Exemplo de gráfico gerado.....	27
Ilustração 17: EDSPN sem cobertura de falhas para o bloco básico X.....	28
Ilustração 18: Molloy.....	30
Ilustração 19: Performance de software.....	31
Ilustração 20: Hardware-Software.....	34

Índice de Tabelas

Expressões analíticas e métricas.....	28
Comparação de resultados.....	29
Alcançabilidade.....	30
Probabilidade de finalização.....	32
Hardware-Software.....	33

Tabela de Símbolos e Siglas

EDSPN - *extended deterministic stochastic petri net*

SCPNS - *stochastic petri nets*

SPNP - *stochastic petri net package*

SPNs - *stochastic petri nets*

SRN - *stochastic reward net*

TimeNET - *TIMEd Net Evaluation Tool*

Capítulo 1

Introdução

O grande desenvolvimento de novas tecnologias e a rápida evolução da ciência nos dias atuais tem propiciado um significativo aumento da informação gerada e manipulada pelo homem. Este, por sua vez, tem alavancado a própria geração de mais conhecimentos e a necessidade de manipulação desses dados. O ser humano almeja confiar poder nos sistemas críticos como, por exemplo, sistemas de aviação, sistemas médicos, de transportes e outros sistemas altamente complexos transferindo para os computadores a função de controle e supervisão. Os sistemas, portanto, têm-se tornado cada vez mais complexos e essa complexidade implica na criação de novos sistemas ainda mais complexos que os anteriores. Anos atrás o processamento era realizado por computadores com grande capacidade de processamento, em empresas de grande porte. Hoje a computação não depende somente desses computadores, podendo utilizar centenas de computadores de menor capacidade de processamento ligados em rede e trabalhando em paralelo. Cada um desses computadores além de representar um percentual de falha do sistema como um todo também contribui para aumentar a complexidade do sistema computacional. Apesar do volume de conhecimento gerado computacionalmente, a criação e modelagem dos sistemas ainda requer conhecimento especializado na área e modelos otimizados, gerados em um menor tempo e com uma menor taxa de falha.

A necessidade de uma maior qualidade nos sistemas desenvolvidos, deve garantir uma maior correteude tanto do *hardware*, quanto do *software*. O crescimento das responsabilidades, embutido em tais sistemas, requer cada vez mais a previsão por meio de modelos de medidas de dependabilidade (como, por exemplo, confiabilidade, disponibilidade e segurança). Isso se deve às consequências que possam advir do mal funcionamento de tais sistemas, através de enormes perdas financeiras ou de vidas humanas.

1.1 Contextualização

Modelar um sistema não é algo trivial, dado que é preciso transformar dados reais em modelos matemáticos que devem refletir a realidade de maneira bem próxima à realidade. Com a globalização, pequenas empresas passam a ter contato com todo o globo e seus produtos devem ser gerados de forma que durem e sejam confiáveis. O objetivo é que eles possam competir no mercado internacional da área de atuação dessa empresa concorrendo com as grandes corporações já existentes. Uma das qualidades necessárias para que um produto ganhe o mercado é a faixa de tempo em que pode ser usado antes de precisar ser substituído; para tal qualidade torna-se preciso que antes da criação do mesmo sejam feitos modelos que auxiliem no estudo das características do produto. Esses modelos devem ser analisados e facilmente alterados durante o momento de criação para que seja rápido e, se possível, intuitivo de se alterar; no mínimo de tempo possível esse modelo deve possuir alta taxa de confiabilidade, para que ele possa ser transformado em um produto de fato.

Alguns sistemas precisam receber mais atenção no seu desenvolvimento porque são sistemas críticos. Sistemas críticos são aqueles que envolvem riscos, sejam ecológicos, humanos ou financeiros. Alguns exemplos de sistemas críticos são sistemas de transporte, centrais nucleares e aparelhagens médicas. Uma falha não prevista nem detectada e, portanto, não tratada em uma central nuclear, por exemplo, põe em risco os funcionários, a população próxima e toda a natureza ao redor dessa usina. Em casos mais graves, o risco se estende também para áreas mais distantes, causando maiores danos a populações e meio ambiente.

Sistemas críticos tendem a possuir modelos mais complexos, pois buscam cobrir máximo de casos de falha, diminuindo assim a possibilidade de que algum defeito deixe ser calculado.

Visto que sistemas matemáticos tendem a se tornar mais difíceis de se analisar, quando tornam-se mais e mais complexos, foi necessário usar de abstrações para que o sistema, como um todo ou parte dele, seja mais facilmente verificado. Algumas dessas abstrações são as cadeias de Markov [3], redes de Petri [5] e modelos de blocos [8]. Isso facilita a criação de modelos para análise de

métricas (como confiabilidade, disponibilidade e segurança) por usuários que não precisam ser extremamente especializados em matemática, e ainda assim, conseguem manter o rigor matemático.

1.2 Proposta

A proposta deste trabalho inclui o desenvolvimento de um tutorial para a ferramenta de desenvolvimento de modelos, utilizando redes de Petri, SPNP (*Stochastic Petri Net Package*) [7].

Alguns modelos são reproduzidos, analisados pela ferramenta SPNP-Gui e comparadas com os dados originais já validados matematicamente ou através de outra ferramenta de modelagem. Havendo similaridade ou, até mesmo, igualdade dos dados tem-se que a ferramenta pode ser usada como substituta do método que foi usado originalmente (nos artigos e teses de onde foram retirados os modelos).

Inicialmente são mostrados conceitos relevantes para melhor entendimento do trabalho. Em seguida é visto como instalar e utilizar a ferramenta SPNP-Gui na forma de um tutorial. Posteriormente são demonstrados reproduções e suas respectivas comparações com os dados originais de forma que fique claro a possibilidade de utilizar a ferramenta SPNP-Gui como gerador e validador de modelos usando redes de Petri.

Capítulo 2

Conceitos e terminologia

Este trabalho envolve uma série de conceitos e terminologias que são essenciais para a compreensão do tema abordado. Neste capítulo serão descritos os termos a serem utilizados tanto no desenvolvimento do tutorial da ferramenta SPNP-Gui e nas validações mostradas posteriormente.

2.1 Sistemas

Sistemas são objetos físicos ou abstratos que podem ser decompostos em outros objetos inter-relacionados que têm um objetivo em comum. Deve ser possível arbitrar quais são esses objetos que fazem parte do sistema [2].

2.1.1 Sistemas estáticos

São ditos sistemas estáticos aqueles que geram resultados independentes de eventos gerados antes de um tempo t anterior à entrada de dados. Também podem ser chamados de sistemas sem memória.

2.1.2 Sistemas dinâmicos

Sistemas dinâmicos são os que dependem de eventos em um tempo t anterior à entrada de dados, por isso podem ser chamados de sistemas com memória.

2.1.3 Sistemas estacionários

São sistemas que independem do deslocamento do tempo t , ou seja dada uma entrada $u(t)$, será retornada uma saída $y(t)$ que é idêntica à saída $y(t-a)$ que seria o resultado da aplicação da entrada $u(t-a)$ onde a é o deslocamento do tempo t .

2.1.4 Sistemas variantes no tempo

São assim chamados por gerarem saídas diferentes para mesma entrada por levarem o tempo t em consideração, ou seja, t pode ser considerado uma das variáveis de entrada.

2.1.5 Sistemas orientados a tempo

Os sistemas orientados a tempo são aqueles que possuem mudanças de estado de acordo com a passagem do tempo, o que pode ser traduzido como: as mudanças de estado do sistema só ocorrerão em conjunto com os disparos do *clock*.

2.1.6 Sistemas orientados a eventos

Diferentemente dos sistemas orientados a tempo, o *clock* não é levado em consideração para a mudança de estado do sistema em questão, assim pode-se dizer que o sistema é assíncrono já que o acionamento é dado através de interrupções do sistema.

2.2 Modelos

Modelos são simplificações de objetos de estudos como fenômenos ou sistemas reais ou teóricos. Modelo é uma representação simplificada de algo de maneira que retire grande parte da complexidade, tornando-o passível de ser estudado. “[...] modelos podem substituir complexidade por simplicidade” [6].

2.3 Processos

Diz-se que processo é uma sequência de mudanças de um sistema, em seus atributos ou em suas propriedades por uma relação causa-consequência [2].

2.3.1 Determinístico

O processo é dito determinístico quando o tempo máximo de sua execução é conhecido antecipadamente, como o que ocorre nos sistemas embarcados de tempo real crítico.

2.3.2 Processo estocástico

O termo estocástico, do grego $\sigma\acute{o}\chi\omicron\varsigma$, pode ser traduzido como objetivo, alvo ou adivinhação. Diz-se que os processos são estocásticos quando eles possuem sequências aleatórias de mudanças em um sistema.

2.4 Modelos gráficos

Criar modelos pode facilitar o estudo, mas ainda assim os modelos podem continuar complexos quando não há muito que possa ser desconsiderado. Outro problema dos modelos é que eles não são facilmente observáveis, por serem essencialmente textuais ou matemáticos.

Esse problema gerou a criação de abstrações mais facilmente apresentáveis como a seguir:

- Diagramas de Venn, pelo inglês John Venn 1834-1923, muito utilizado em teoria dos conjuntos. Exemplo na Ilustração 1 onde se observa três conjuntos que possuem conteúdo em comum;

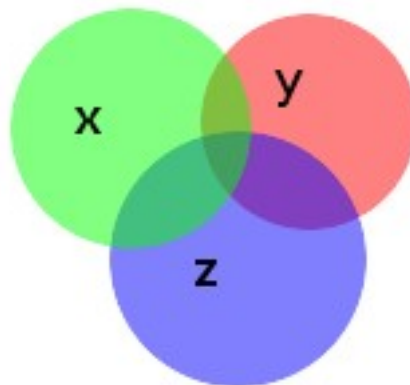


Ilustração 1: Exemplo de diagrama de Venn

- Diagramas de blocos representam através de figuras, geralmente blocos para objetos do sistema representado, relações entre objetos do sistema, podendo representar também o fluxo entre os objetos. Exemplo na Ilustração 2 que demonstra um sistema que pode ser dividido em três objetos distintos e como eles estão interligados entre si para formar o sistema;

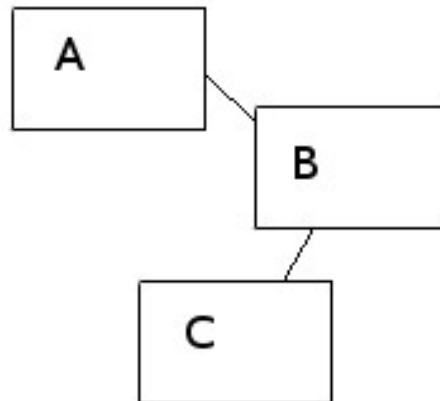


Ilustração 2: Exemplo de diagrama de blocos

- Cadeias de Markov, pelo russo Andrei Markov 1856-1922, ilustram os possíveis estados de um sistema, cada mudança de estado independe do estado anterior. Exemplo na Ilustração 3. Observe que com poucos elementos se pode mostrar um sistema de três estados possíveis e quais as

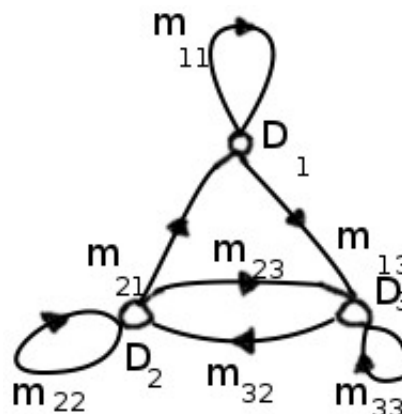


Ilustração 3: Exemplo de cadeia de Markov relações entre eles (como o estado prévio de um estado qualquer);

- Redes de Petri [5], pelo alemão Carl Petri 1926-2010, ilustra o estado atual de um sistema e permite uma fácil visualização de possíveis estados posteriores. Exemplo na Ilustração 4 em que reproduz uma rede para análise de confiabilidade sem cobertura de falhas como descrito em [1]. Nota-se que poucos elementos visuais podem servir para abstrair cálculos matemáticos (como será demonstrado no capítulo de validação da ferramenta).

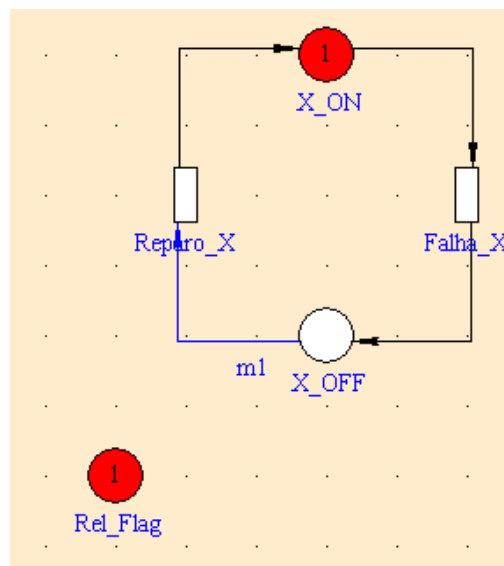


Ilustração 4: Exemplo de rede de Petri

Para esta monografia, será dado ênfase no estudo das redes de Petri por ser esse tipo de modelo representado pela ferramenta *SPNP-Gui*.

Capítulo 3

Redes de Petri

Também são conhecidas como redes lugar/transição ou redes P/T. É uma abstração matemática para modelagem e análise de sistemas.

3.1 Rede de Petri básica

Uma rede de Petri comum é composta por lugares, transições e arcos (*places, transitions* e *arcs* respectivamente em inglês). As notações usadas serão:

- $P \rightarrow$ lugar;
- $T \rightarrow$ transição;
- $F(P \times T) \rightarrow$ um arco que ligue um lugar P para uma transição T ;
- $F(T \times P) \rightarrow$ um arco que ligue uma transição T para um lugar P ;

Um arco interliga um lugar a uma transição ou uma transição a um lugar, mas nunca transições entre si, nem lugares entre si.

Um lugar pode conter zero ou N marcações chamadas de *tokens*, que geralmente são representados por fichas ou números. Um lugar é chamado de entrada quando há uma transição dele para um arco, e é chamado de saída quando o arco vem de uma transição até o lugar.

Transições podem ser ditas habilitadas quando é possível remover N ($N > 0$) marcações de um lugar de entrada ou quando não há lugar de entrada (nesse caso a transição poderá disparar indefinidamente). A transição é dita desabilitada quando o lugar de entrada não possui marcações suficientes para que a transição dispare. Quando um lugar é entrada para duas ou mais transições, os disparos serão indeterminísticos, ou seja, não é possível determinar qual deles disparará primeiro.

3.1.1 Extensões das redes de Petri

Existem casos em que são definidas restrições ou novas maneiras de se trabalhar com uma rede de Petri. Essas definições servem para que as redes sejam adaptadas para casos específicos de modelagem, às vezes servindo para que seu comportamento seja mais próximo de uma dada realidade a ser modelada ou para melhor visualização ou até para otimizar o processo de modelagem.

Nesta monografia são citadas propriedades das seguintes extensões:

- Redes de Petri priorizadas: as transições de maior prioridade são disparadas antes de qualquer outra transição de menor prioridade. Transições de prioridades iguais agem como em uma rede de Petri comum;
- Redes de Petri temporizadas: o tempo também pode ser modelado no sistema através de transições temporizadas que possuem menor prioridade que transições não temporizadas;
- Redes de Petri estocásticas ou SPNs (do inglês *stochastic Petri nets*): as transições possuem aleatoriedade ajustável e utiliza distribuição exponencial para cronometrar a rede.
- Redes de Petri estocásticas coloridas ou SCPNs (do inglês *stochastic Petri nets*): em uma rede de Petri comum, as marcações não podem ser diferenciadas entre si, para tal deve-se fazer uma replicação da rede para que se possa acompanhar o trajeto de uma marcação específica. Já em uma rede de Petri colorida as marcações possuem tipos que as distinguem umas das outras evitando replicações.

3.2 Ferramentas

3.2.1 TimeNET

O TimeNET [9] (*timed net evaluation tool*) é uma ferramenta de modelagem, avaliação de redes temporizadas e análise de SPNs e SCPNs, desenvolvida na *Fakultät Für Informatik und Automatisierung* (Faculdade de Informática e

Automação) na Alemanha. Versões anteriores já foram objeto de publicação [9] de artigos demonstrando a validade e como pode ser utilizada para facilitar o desenvolvimento e validação de modelos utilizando redes de Petri.

Inicialmente foi desenvolvida para sistemas *UNIX*, mas foi portada para o *Windows* facilitando a distribuição e utilização da ferramenta por usuários com menos disponibilidade de aprender a usar outros sistemas operacionais.

O *TimeNET* foi desenvolvido em *Java* e permite, inclusive, a modelagem e análise de redes de Petri coloridas. É capaz de fazer diversos tipos de análises sobre redes de Petri como por exemplo a detecção de estados absorventes.

A ferramenta permite a criação de funções personalizadas para exibição de parâmetros na interface e para modificação no funcionamento da rede. Para isso é preciso aprender uma linguagem desenvolvida pela equipe da ferramenta.

O problema da ferramenta surgiu desde o lançamento da versão para *Windows* na qual o *TimeNET* deixou de ser capaz de continuar processando quando um arco possui um valor maior que o lugar de entrada como mostrado na Ilustração 5. Essa situação é descrita como um estado absorvente.

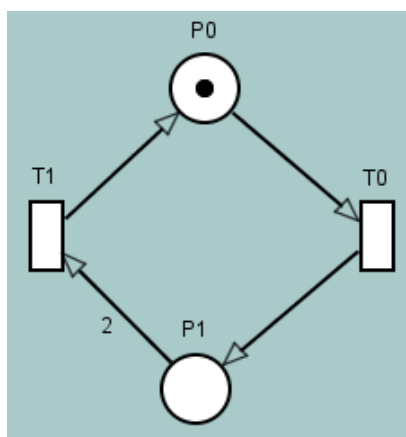


Ilustração 5: Rede de Petri com arco de valor 2

Neste caso o processamento é interrompido e é retornado um erro. A ferramenta foi projetada para agir assim nesta situação, que interromperia a ativação da transição T1. Por ser um requisito do projeto da ferramenta, não há possibilidade de continuar analisando a rede até que seja removido o estado absorvente.

3.2.2 **SPNP**

É uma ferramenta, sem interface visual, para a criação de redes de Petri. Por ser baseada em redes de Petri abstrai a necessidade de conhecimento matemático por parte do usuário. Assim como o *TimeNET* pode ser usado nos mais diversos tipos de modelos e análises, mas por não possuir interface visual e de depender puramente de conhecimento em programação na linguagem C acaba por não ser intuitiva.

3.2.3 **SPNP-Gui**

O *SPNP-Gui* é uma interface visual para se utilizar o *SPNP* sem a necessidade de escrever em código a criação do modelo a ser analisado. Com a interface se torna uma ferramenta semelhante ao *TimeNET* por possuir um vasto repertório de funções para modelar e analisar redes de Petri. Apesar de ser uma interface visual para o *SPNP*, ainda é possível utilizar de código C para modificações no funcionamento da rede.

O diferencial do *SPNP-Gui* é a possibilidade de optar por continuar processando em casos que podem ser considerados como indesejados como o estado absorvente citado anteriormente. Por tal motivo o *SPNP-Gui* foi a ferramenta escolhida objeto de estudo desta monografia.

Capítulo 4

Ferramenta *SPNP-Gui*

Este capítulo contempla uma espécie de tutorial escrito no intuito de facilitar e agilizar a criação de modelos em redes de Petri utilizando uma ferramenta com interface, o *SPNP-Gui*. O guia apresentado em conjunto com a ferramenta é específico para o *SPNP*, que é a parte não visual da ferramenta, por este motivo foi criado um tutorial que guia o aprendizado da ferramenta *SPNP-Gui*, pois um guia para a interface gráfica pode ser necessário para aqueles interessados em utilizar a interface gráfica. No *site* da ferramenta há um breve manual para o *SPNP-Gui*, mas está incompleto até a presente data (8 de novembro de 2011).

A interface *SPNP-Gui* facilita, não somente, a criação, edição e visualização de redes de Petri o que reduz o intervalo entre criação e testes. Ainda assim é preciso um certo conhecimento na linguagem de programação C para fazer algumas modificações no código gerado pela ferramenta.

4.1 Método

O tutorial foi escrito através da experimentação, criação de modelos e erros gerados durante todo o processo, desde instalação até à análise de dados dos modelos verificados. Então, o que é explicado é mostrado pelo caminho que normalmente funciona e costuma não gerar erros na ferramenta.

4.2 Adquirindo

4.2.1 Como Baixar e Instalar

Vá no *site*: <http://people.ee.duke.edu/~kst/>

Clique em *software packages* e localize *SPNP*.

Última visita feita no dia 8 de novembro de 2011.

4.2.2 Contato com Kishor Trivedi

Para ter o direito de uso da ferramenta, deve-se entrar em contato com o autor e assinar um termo de concordância de que a ferramenta terá uso puramente acadêmico além de outros termos. Aviso aos estudantes: deve-se ter a assinatura também de um professor da instituição para que o termo seja válido.

http://shannon.ee.duke.edu/tools/agreement_snpn.htm

4.2.3 Baixando

No *site* pode-se baixar alguns arquivos relativos à ferramenta em questão, mas o link “*download SPNP!*” é onde deve ser clicado. Coloque o nome e senha recebidos no passo anterior, e baixe o arquivo para algum lugar de preferência.

4.2.4 Instalação no *Win 7 Professional*

Descompacte o arquivo na pasta C:/Snpn-Gui.

A ferramenta não possui instalador, então é preciso fazer mais alguns passos antes de iniciar o uso.

4.2.5 Variáveis de usuário e ambiente

Para que o SPNP-Gui funcione, algumas variáveis do *Windows* devem ser configuradas:

- Pressione as teclas *windows+pause* para exibir as configurações básicas sobre o computador;
- Clique em “configurações avançadas do sistema”;
- Na guia “avançado” clique em “variáveis de ambiente”;
- Em “variáveis de usuário” clique em “novo”, o nome da variável é “*SPNP-DIRECTORY*” (sem aspas) e o valor é “C:\Snpn-Gui\snpn” (também sem aspas);

- Em “variáveis do sistema” procure a variável “*Path*”, edite e adicione no início “C:\Spnp-Gui\spnp\bin;” (sem aspas);
- Feche essa janela apertando em “*ok*” e feche a outra apertando “*ok*”;

4.2.6 Compatibilidade

O *SPNP-Gui* foi desenvolvido no ano 2000 e, portanto, não é muito compatível com versões posteriores do Windows . Para diminuir a quantidade de incompatibilidades vá em “C:\Spnp-Gui\” e, com o botão direito, clique em “*Interface SPNP*”, depois em “propriedades”. Na guia de compatibilidade marque a opção “executar este programa em modo de compatibilidade” e escolha “*Windows 2000*” . Feche em “*ok*”.

Problemas Comuns

Apesar de estar em modo de compatibilidade, o programa ainda sofre os seguinte erros:

- Travamentos;
- Fechamentos inesperados do programa;
- Área de modelagem congelada ou não atualizada corretamente;
- Apertar no botão fechar, indicado por um “X”, nem sempre fecha, procure usar sempre o botão “*dismiss*” ou “*cancel*” quando pretender fechar alguma janela interna ao programa;
- Nem sempre há o botão “*ok*”. Para salvar e fechar, procure o botão “*validate*” e depois “*dismiss*”.

Por isso deve-se adquirir o hábito de salvar com frequência para evitar perder horas de trabalho caso a ferramenta feche inesperadamente.

Apesar da ferramenta não utilizar dos padrões de interface visual (como o botão de fechar já citado), ela segue um padrão de interface, então com o uso

frequente o usuário se habitua a esses pequenos desvios do padrão da ferramenta, mas que não interferem no poder da mesma.

4.3 Utilizando a ferramenta

4.3.1 Usando o SPNP-Gui

Será usada a notação “menu → submenu” para uma sequência de menus a serem clicados. Em alguns casos mais complicados, serão usadas imagens para facilitar.

4.3.2 Gerenciando a pasta de arquivos

O padrão da ferramenta é salvar tudo na pasta onde ela se encontra instalada. O ideal é separar os arquivos dos projetos e os arquivos do programa. No menu *File* → *Preferences* modifique para as pastas de preferência.

4.3.3 Criando um novo projeto

Para criar um novo projeto: *File* → *New*. A janela exibida será como na Ilustração 6.

Os campos que devem receber maior atenção são:

- “*Name of the project*”: nome do projeto que possuirá vários modelos;
- “*Name of the model*”: nome do primeiro modelo a ser feito (novos modelos podem ser adicionados depois).

4.3.4 Criando novas redes em um projeto

Para adicionar novos modelos no projeto vá em *Model Editor* → *New Stochastic Reward Net*.

4.3.5 Salvando

Salvando o projeto: *File* → *Save Project*.

Salvando o modelo: *File* → *Save Model*.

Salvando o projeto com novo nome: *File* → *Save Project as*.

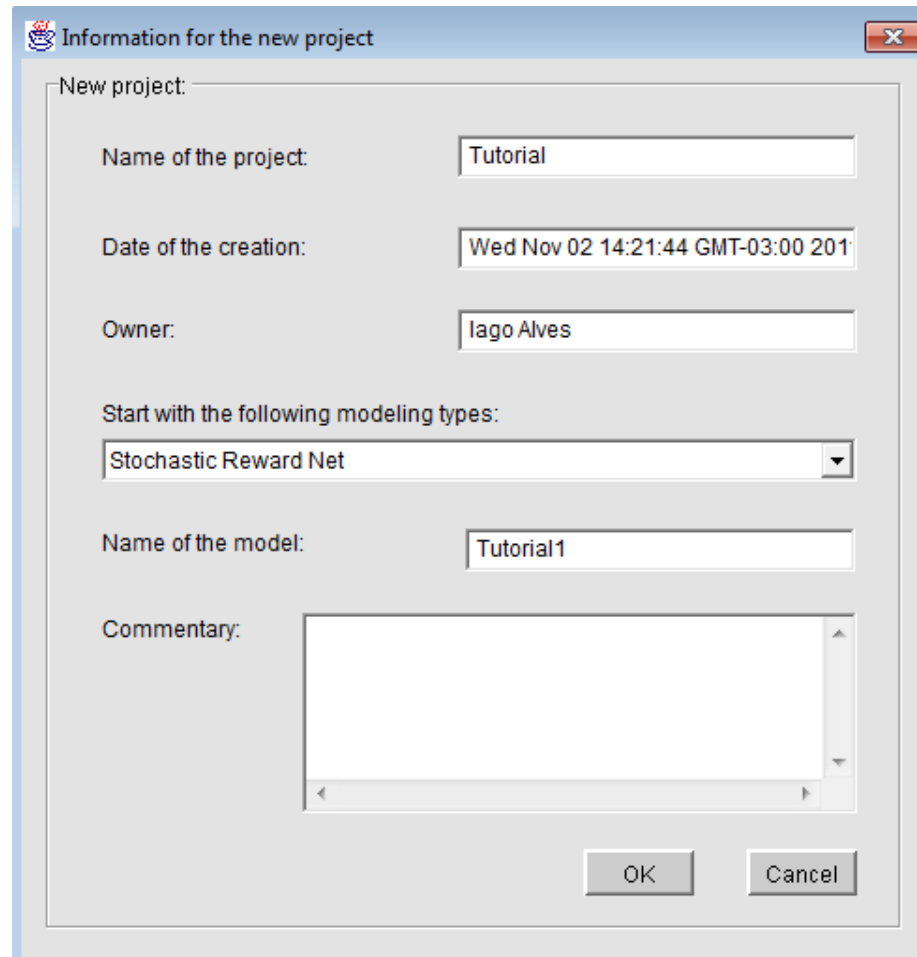


Ilustração 6: Novo projeto

4.3.6 Abrindo

Abrindo um projeto *File* → *Open Project*

Abrindo um modelo *File* → *Open Model*

4.3.7 Botões de interface

Os botões da interface possuem um *label* indicativo como mostrado na Ilustração 7. Cada botão exibe cada componente que será inserido como: lugar, transições, arcos etc.

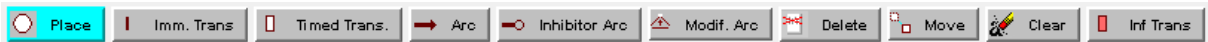


Ilustração 7: Botões da interface

Para colocar um arco é preciso, pelo menos, de um lugar e uma transição. O arco deve ser levado de um para outro. Obs.: a seta nem sempre acompanha bem a posição do mouse, se guie pela seta, não só pelo mouse.

4.3.8 Menus da área de criação

Primeiro cria-se o modelo (um lugar, uma transição e um arco) como na Ilustração 8:

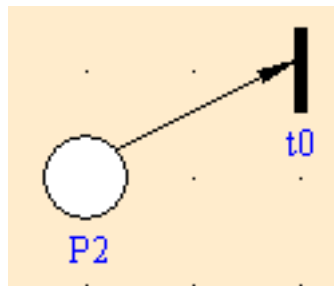


Ilustração 8:

Exemplo de modelo

Clicar com o botão direito em qualquer um dos três elementos faz surgir um menu com a opção “*Property*” e é nela que se configura os dados do elemento em questão como: nome do lugar/transição, marcações, peso etc.

Esses valores podem ser uma palavra, assim o programa gera uma variável com esse nome. Em alguns casos existe a possibilidade de escolher criar uma função e não uma variável (como explicado no próximo tópico).

Clicar com o botão direito em uma área vazia aparece um menu com opções de visualização.

4.3.9 Variáveis

No modelo anterior, modifique o lugar P2 (abra as propriedades clicando com o botão direito nele) para que fique como na Ilustração 9.

Ao clicar em “*validate*” e então em “*dismiss*” o lugar P2 passa a ter 'x' tokens.

Para verificar o valor de x , vá em “*Model editor* → *Variable / Constant*”. Abrirá uma janela com todas as variáveis e/ou constantes declaradas. Nessa janela pode-se criar novas variáveis e constantes que poderão ser usadas nas funções personalizadas.

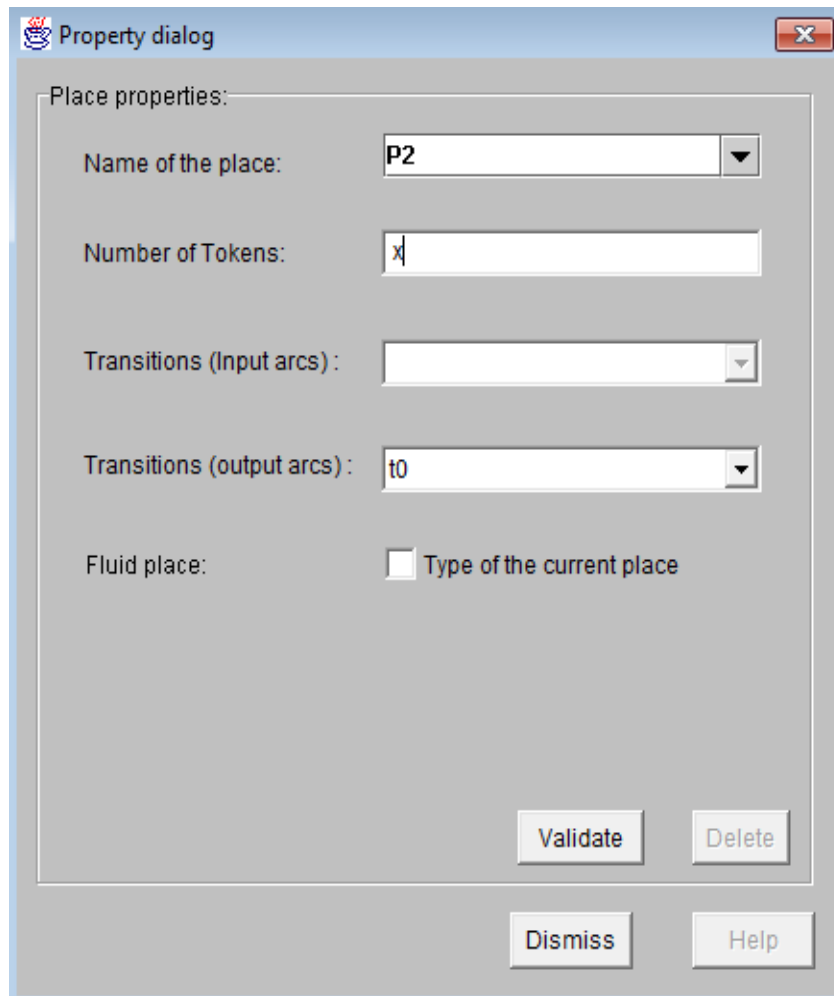


Ilustração 9: Propriedade de um lugar

4.3.10 Funções

No modelo anterior, modifique a transição t_0 (abra as propriedades clicando com o botão direito nela) para que fique como na Ilustração 10.

Ao clicar em “*validate*” e então em “*dismiss*” o lugar P_2 passa a ter ' x ' tokens.

Para verificar o valor de x , vá em “*Model editor* → *Function definition*”. Abrirá uma janela com todas as funções declaradas. Nessa janela pode-se criar novas funções para serem usadas pela simulação. Para cada função deve-se selecionar o

tipo de função (função de guarda, função de probabilidade etc) e o SPNP-Gui já configurará o tipo de retorno esperado pela função C que você irá criar.

Sempre que quiser trocar a função por outra já listada é preciso clicar antes em “*validate*” para que a mesma seja salva. Obs.: O programa não faz verificação em tempo real do código escrito.

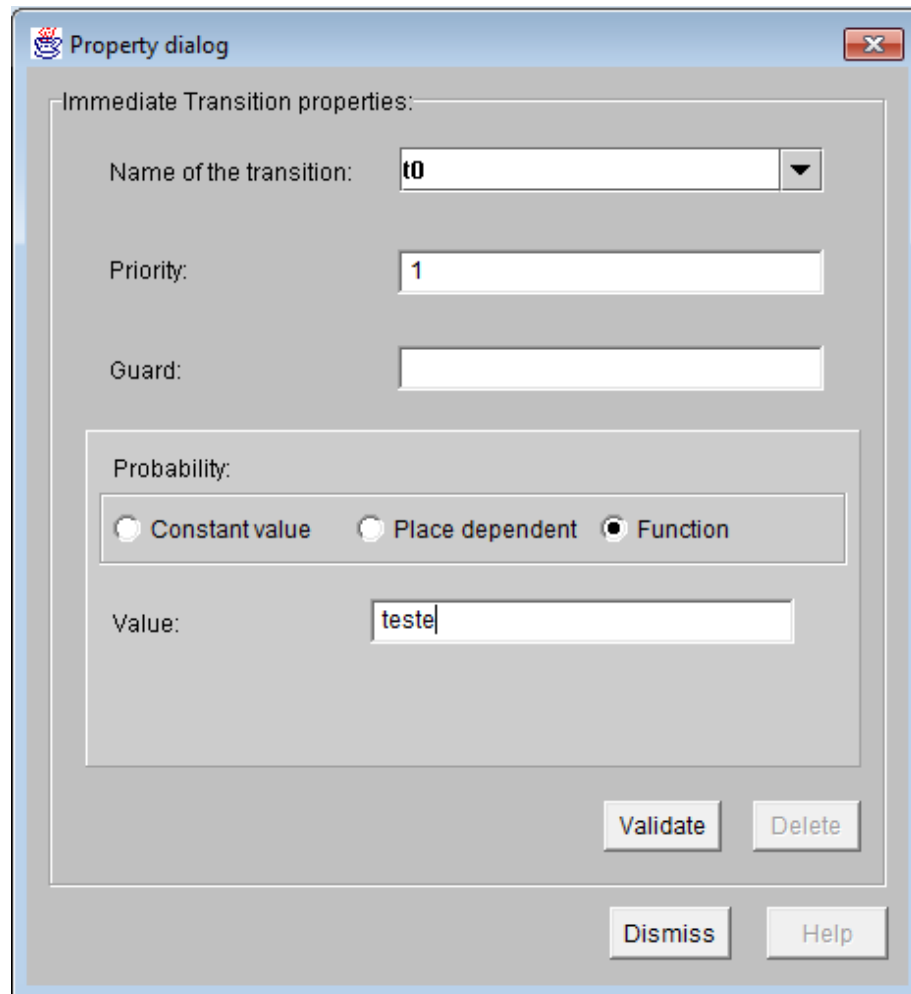


Ilustração 10: Propriedades de uma transição

Para recuperar a quantidade de *tokens* em um lugar, deve-se usar a função “*mark()*” como exemplificado a seguir:

```
double umaFuncaoSuaAqui() {
    if(mark("P2") == 1) return (1.0);
    else return (0.0);
}
```

O nome da variável que é passado como argumento para a função “*mark()*” deve ser exatamente como foi nomeada, inclusive maiúsculas e minúsculas.

4.4 Analisando a rede

Crie um novo projeto e faça a rede como na Ilustração 11 com um token em P0. Pode-se notar que o lugar P0 fica vermelho, isso significa que há um ou mais *tokens* nele. Clicando no menu “*Analysis Editor → Analysis*”, abrirá uma janela como na Ilustração 12.

Deve-se escolher cuidadosamente quais as saídas desejadas, pois apesar de haver o botão “*delete*” ele não funciona para remover um tipo de saída adicionado. Para isso deve-se fechar a janela, abri-la novamente e adicionar somente as saídas desejadas.

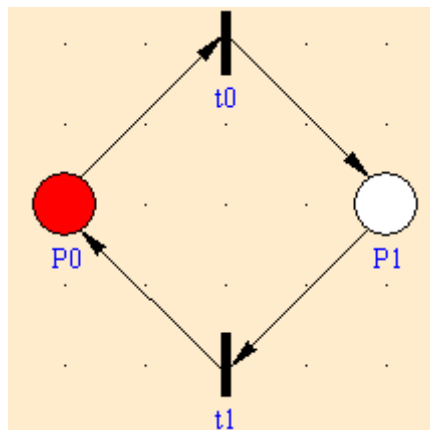


Ilustração 11: Outro exemplo de rede de Petri

Para cada item selecionado há uma quantidade de parâmetros que devem ser ajustados de acordo com a necessidade do usuário, esses parâmetros podem ser modificados ao selecionar um desses itens, como na Ilustração 13 por exemplo.

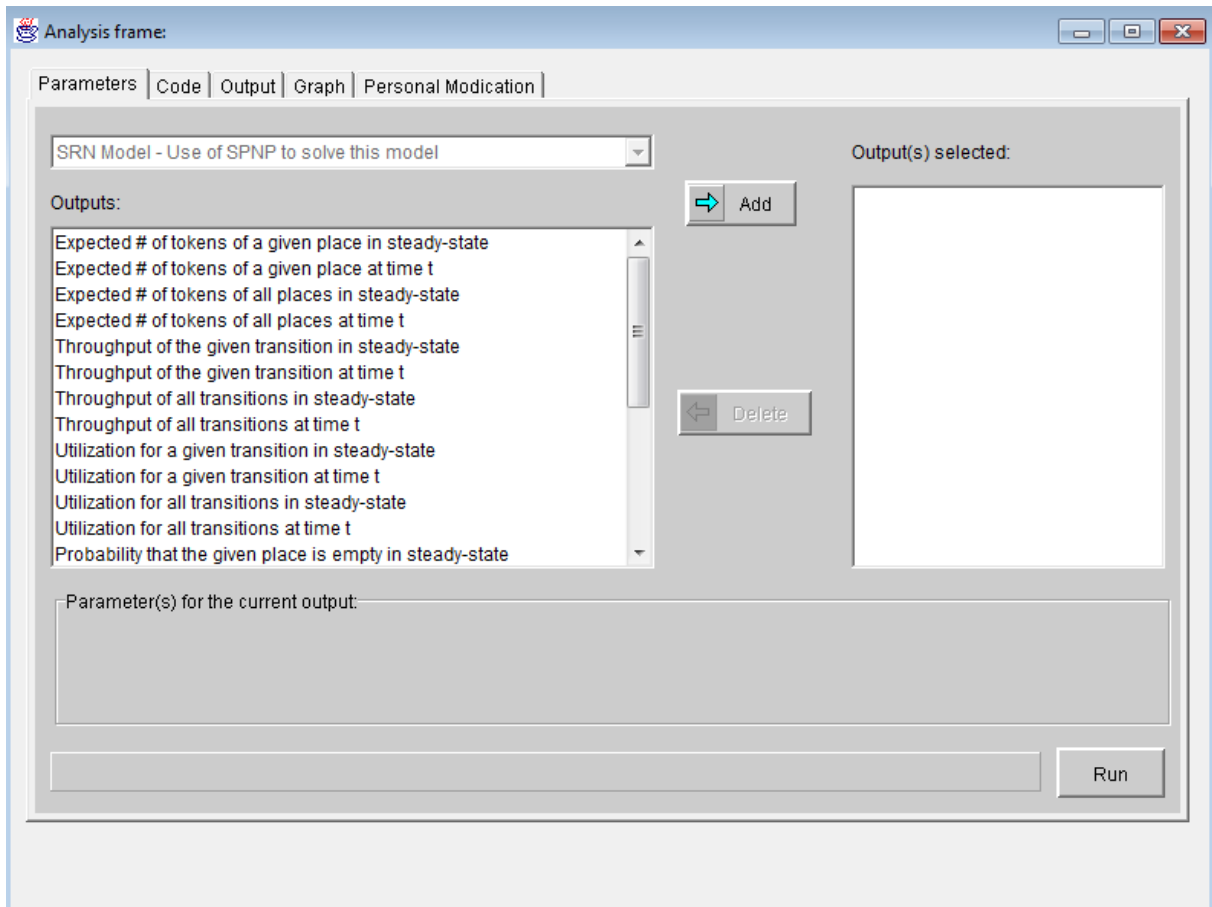


Ilustração 12: Janela de análise de rede

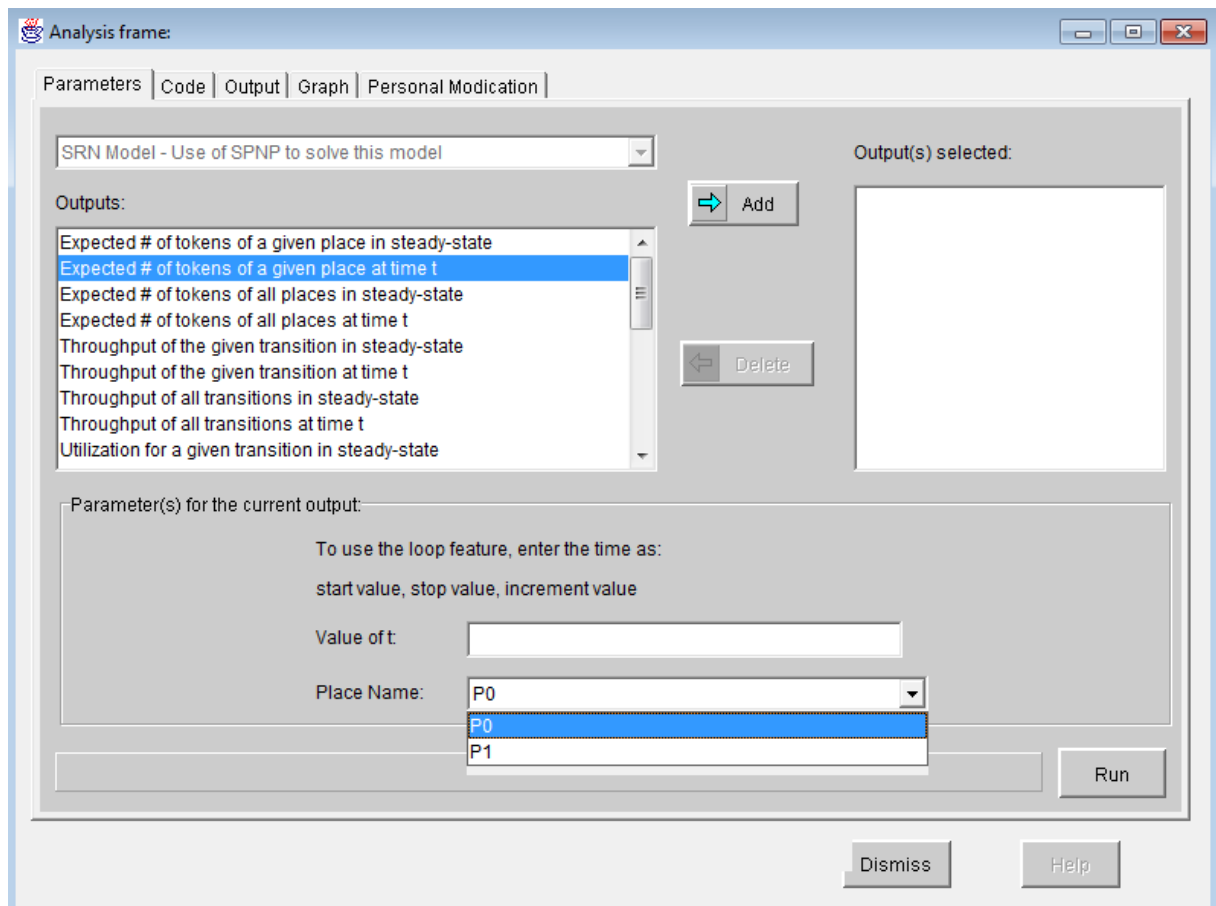


Ilustração 13: Exemplo de análise

4.4.1 Erros comuns

Muitos erros podem ser retornados, mas o mais comuns são:

- Erro no código. Solução: reveja suas funções escritas em C;
- Programa fechando/travando. Solução: salve antes de iniciar a análise.
- Erro de execução do “run”. Verifique se na janela “Interface SPNP” surgiram mensagens de erro como na Ilustração 14. Solução: salve seu projeto e reinicie a ferramenta.
- Encontrado estado absorvente. Solução: modifique o código gerado manualmente como explicado no tópico a seguir.

```

Interface SPNP
at spnpGui.AnalysisFrame_removeOutput_actionAdapter.actionPerformed(A
analysisFrame.java:3981)
at com.borland.jbcl.util.ActionMulticaster.dispatch(ActionMulticaster.ja
va:67)
at com.borland.jbcl.view.BeanPanel.processActionEvent(BeanPanel.java:139
)
at com.borland.jbcl.view.ButtonView.processMouseReleased(Compiled Code)
at com.borland.jbcl.view.BeanPanel.processMouseEvent(Compiled Code)
at java.awt.Component.processEvent(Compiled Code)
at java.awt.Container.processEvent(Compiled Code)
at com.borland.jbcl.view.BeanPanel.processEvent(Compiled Code)
at java.awt.Component.dispatchEventImpl(Compiled Code)
at java.awt.Container.dispatchEventImpl(Compiled Code)
at java.awt.Component.dispatchEvent(Component.java:2289)
at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:1944
)
at java.awt.LightweightDispatcher.processMouseEvent(Container.java:1732)
at java.awt.LightweightDispatcher.dispatchEvent(Container.java:1645)
at java.awt.Container.dispatchEventImpl(Compiled Code)
at java.awt.Component.dispatchEvent(Component.java:2289)
at java.awt.EventQueue.dispatchEvent(EventQueue.java:258)
at java.awt.EventDispatchThread.run(EventDispatchThread.java:68)
Value of command : make -f C:\Snp-Gui\snp\Makerun.dos SPN=Tutorial1

```

Ilustração 14: Exemplo de erro comum

4.4.2 Modificação pessoal

Clicar em “*Personal Modification*” permite ver e modificar o código C gerado. O código costuma ser gerado no seguinte modo:

- Inicia-se com os tradicionais “*include*” como qualquer código C que faça importação de bibliotecas;
- Declaração de variáveis;
- Protótipo de funções;
- Opções do programa SPNP-Gui;
- Definições da rede (lugar, transição, arcos, funções de cardinalidade etc);
- Definições das funções.

Como dito antes, é comum o erro de estado absorvente. Quando já se cria uma rede sabendo da existência desses estados, pode-se modificar o código para inserir o comando “`iopt(IOP_SSMETHOD, VAL_POWER);`” na função “`options()`” ficando:

```
void options() {
```

```

        iopt(IOP_SSMETHOD, VAL_POWER);
    }

```

Mais opções podem ser encontradas no manual oficial da ferramenta (V6.0) em *Chapter 4.1 Function: options()* e em *Chapter 7 – Available Options*.

4.4.3 Examinando a saída ou “*output*”

Ao clicar na aba *output*, se não houve erros, poderá ser visto uma mensagem como a que segue:

```

gcc -c -g -DDEBUG_FLAG=0 -I"C:\Snpn-Gui\snpn\include" -I.
Tutorial.c

gcc -g -o Tutorial.snp Tutorial.o -lsnp6_mingw -lm

SPNP version 6.1.2a

The analysis is starting.

Log messages in file Tutorial.log

Output result in file Tutorial.out

```

Clicar com o botão direito no texto abre um menu que dá acesso aos dados de saída e ao *log* por exemplo. Cabe ao usuário saber como analisar os dados que serão exibidos, pois os dados retornados dependem da rede criada e, portanto, o pesquisador deve saber o que esperar.

4.4.4 Examinando o código ou “*code*”

Após rodar a simulação, o SPNP-Gui libera o código C gerado para a rede de Petri em questão. Já inclui a(s) saída(s) desejada(s) e suas funções personalizadas.

4.4.5 Animação

Apesar de já ter conseguido fazer funcionar (poucas vezes), não foi possível encontrar um método que sempre faça a animação funcionar corretamente. Por tal motivo é preferível não tentar usar essa parte do programa, pois uma vez que é

gerado um erro ao tentar animar, não é mais garantido que o programa continue rodando de maneira correta.

4.4.6 Gerando gráfico

Modificar a opção em “*Output / Function chosen*” fará com que as opções dessa janela mudem de acordo com a necessidade de cada saída/função escolhida. Inicialmente já há uma opção selecionada em “*Output / Function chosen*”, mas ela, na verdade não está, de fato, selecionada, então se a função pretendida for a que já está em exibição, o ideal é que a opção seja escolhida pelo usuário para que fique realmente selecionada.

Depois de preenchidas as opções como o usuário preferir, basta clicar em “run” para que seja gerado um gráfico. A Ilustração 15 mostra um exemplo de configuração para que seja gerado um gráfico e a Ilustração 16 mostra um exemplo de um gráfico gerado.

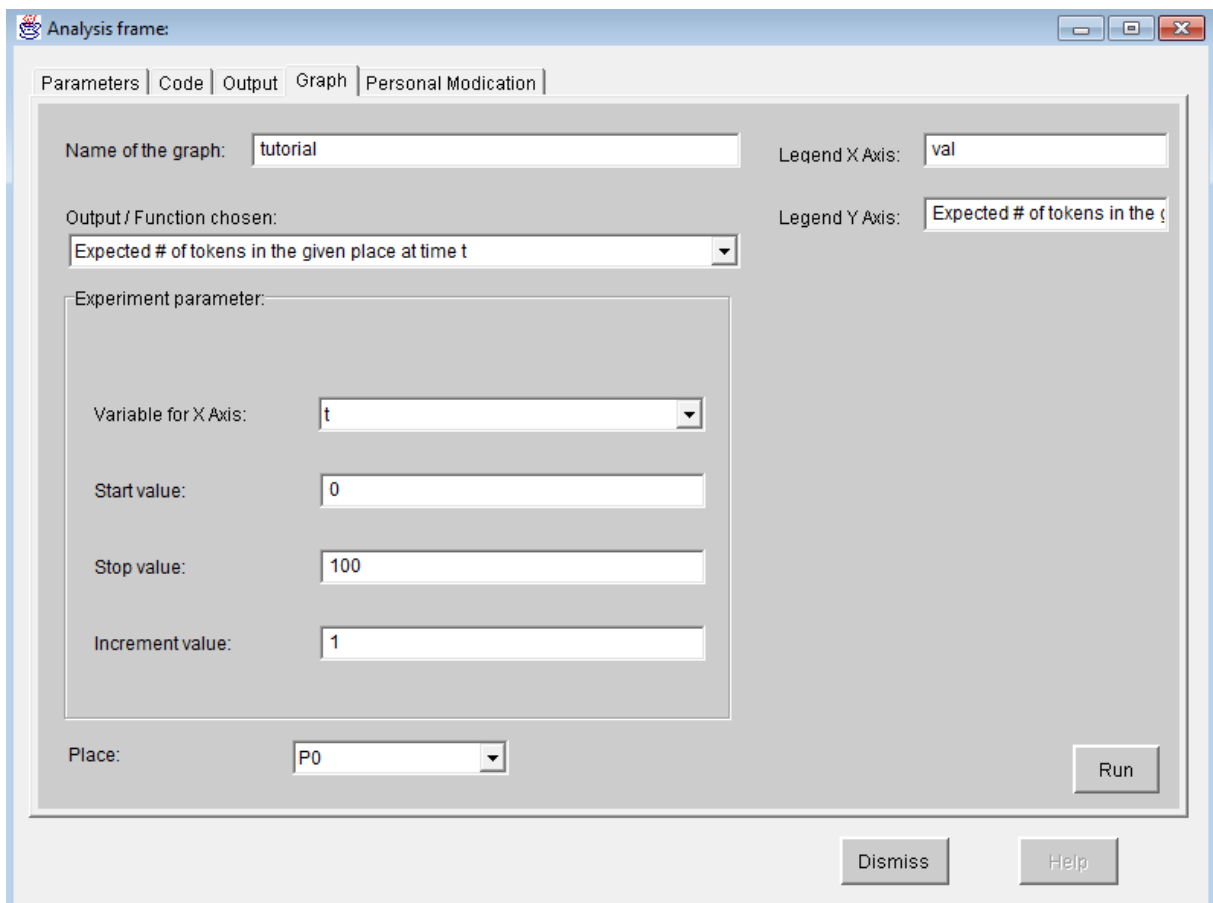


Ilustração 15: Janela de geração de gráfico

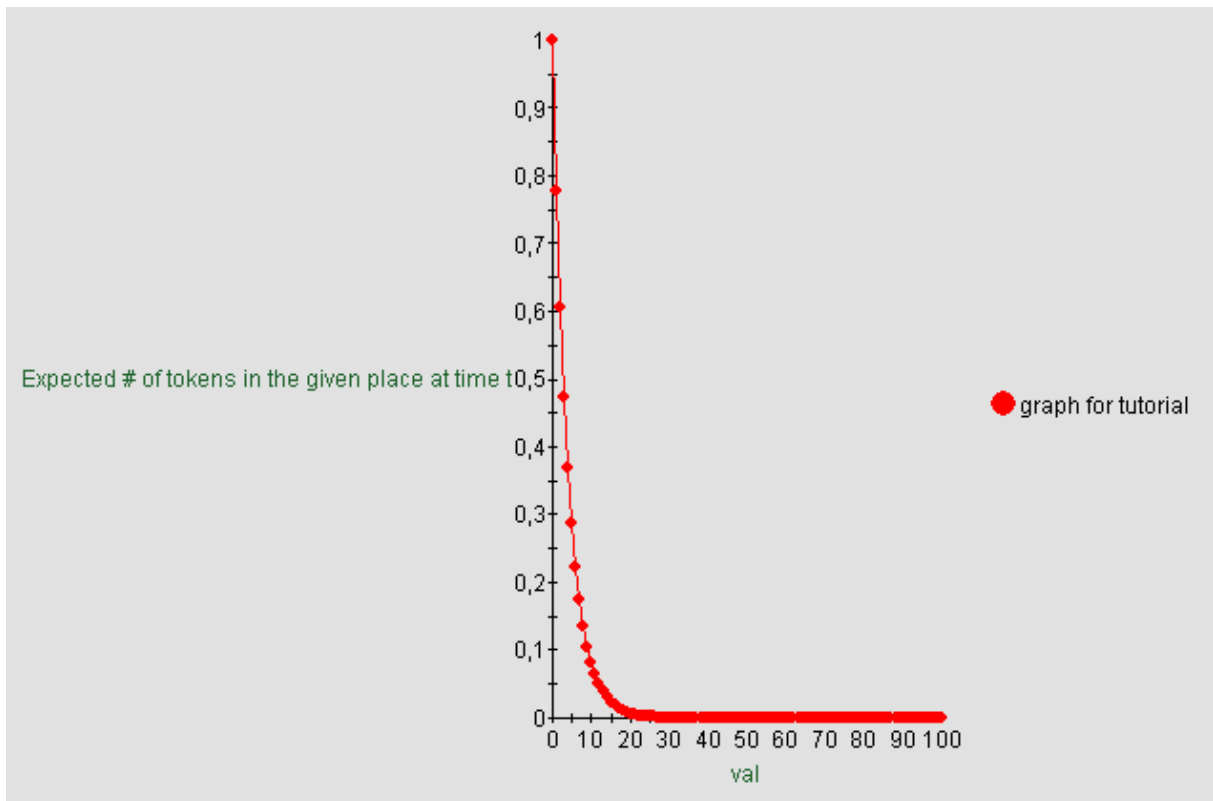


Ilustração 16: Exemplo de gráfico gerado

4.5 Validação

Como dito antes, o *SPNP-Gui* é uma ferramenta que modela *SPNs* (redes de petri estocásticas servindo de abstração para os cálculos matemáticos. Para que a ferramenta seja considerada válida ela deve retornar resultados, pelo menos, próximos de outras ferramentas do mesmo tipo. Deve, também, ser capaz de realizar cálculos matemáticos com precisão. O valor exato dessa precisão depende da necessidade do pesquisador, logo a ferramenta pode ser válida para alguns pesquisadores, mas não para outros. O código-fonte referente a cada caso pode ser encontrado no Apêndice A.

4.5.1 EDSPN

Reproduzindo o modelo *EDSPN* (*extended deterministic stochastic Petri net* – rede de Petri estocástica determinística estendida) encontrado em [1], da Ilustração 17, tem-se um modelo que suas métricas de confiabilidade e inconfiabilidade podem ser calculadas de acordo com a Tabela 1.

Tabela 1. : Expressões analíticas e métricas

Atributo	Expressão analítica	Métrica
Confiabilidade	$R(t) = \exp(-\lambda_x t)$	mark("X_ON")=1
Inconfiabilidade	$R(t) = 1 - \exp(-\lambda_x t)$	mark("X_ON")=0

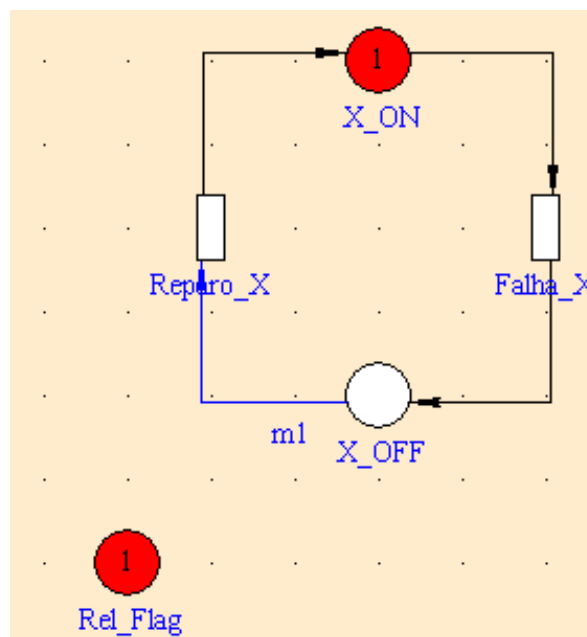


Ilustração 17: EDSPN sem cobertura de falhas para o bloco básico X

Fazendo uma análise da rede através do *SPNP-Gui* verificando a probabilidade de “X_ON” para tempos t tem-se os seguintes resultados já comparados com os valores das expressões analíticas na Tabela 2. Os valores das expressões analíticas foram calculados com a calculadora do Google (que obteve resultados com mais casas decimais do que os resultados encontrados com *Action Script 3* e *C#*), os valores complementares foram omitidos por serem iguais a “1 – val” onde 'val' é o valor encontrado.

Tabela 2. : Comparação de resultados

t	$R(t)=\exp^{(-\lambda_x t)}$	mark("X_ON")=1
1	0,367879441	0.367879441171
2	0,135335283	0.135335283237
3	0,0497870684	0.0497870683679
4	0,0183156389	0.0183156388887
5	0,006737947	0.00673794699909
6	0,00247875218	0.00247875217667
7	0,000911881966	0.000911881965555
8	0,000335462628	0.000335462627903
9	0,000123409804	0.000123409804087
10	$4,53999298 \times 10^{-5}$	4.53999297625e-005
12	$6,14421235 \times 10^{-6}$	6.14421235333e-006
14	$8,31528719 \times 10^{-7}$	8.31528719104e-007
16	$1,12535175 \times 10^{-7}$	1.12535174719e-007
18	$1,52299797 \times 10^{-8}$	1.52299797447e-008
20	$2,06115362 \times 10^{-9}$	2.06115362244e-009
25	$1,38879439 \times 10^{-11}$	1.3887943865e-011
30	$9,35762297 \times 10^{-14}$	7.26965505719e-014

4.5.2 Análise de desempenho

Molloy [4] sugeriu a rede de Petri da Ilustração 18 para estudo tanto probabilístico quanto estudo de cadeias de Markov que podem ser geradas de acordo com essa rede e o crescimento dessa cadeia de acordo com o aumento do número de *tokens* iniciais.

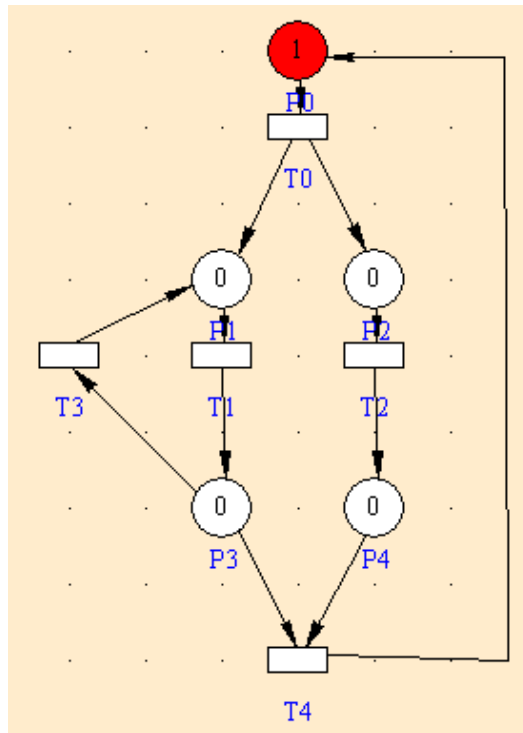


Ilustração 18: Molloy

A Tabela 3 ilustra quantidade de marcações tangíveis de acordo com a análise feita pelo *SPNP-Gui* e por Molloy.

Tabela 3. : Alcançabilidade

<i>Tokens</i> iniciais	Molloy	<i>SPNP-Gui</i>
1	5 marcações tangíveis	5 marcações tangíveis
2	14 marcações tangíveis	14 marcações tangíveis
3	30 marcações tangíveis	30 marcações tangíveis
4	??	55 marcações tangíveis

4.5.3 Análise de desempenho de *software*

É possível verificar a probabilidade de um dado algoritmo ser finalizado. Um exemplo de modelo baseado em um algoritmo [7] pode ser visto na Ilustração 19 que modela o seguinte código-fonte:

```

A: statements;
PARBEGIN
    B1: statements;
    B2: IF cond THEN
        C: statements;
    ELSE
        DO
            D: statements
        WHILE cond;
    IFEND
PAREND
    
```

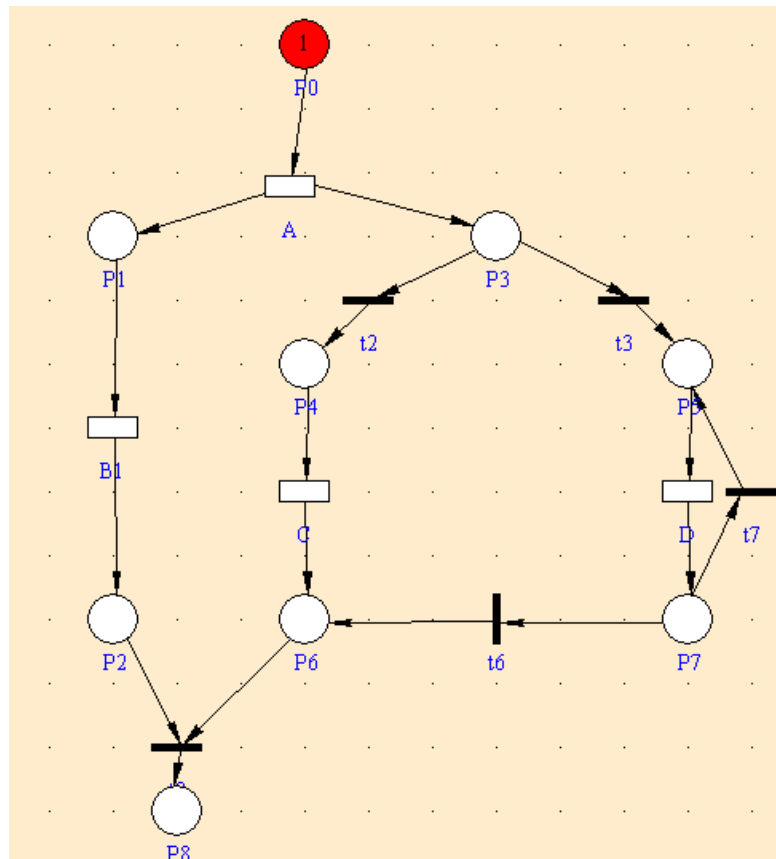


Ilustração 19: Performance de software

Verificar a probabilidade de haver um *token* em P8 significa verificar a probabilidade de um *software* ter finalizado sua execução. A probabilidade de haver um *token* em P8 pode ser calculada na ferramenta, que retornará os seguintes valores para intervalos de tempo *t*:

Tabela 4. Probabilidade de finalização

Tempo (t)	Esperado (<i>tokens</i>) *100%
1	0.0182310471254
2	0.0942596098853
3	0.211172679123
4	0.340887817442
5	0.464569402965
6	0.573112422652
7	0.663776939689
8	0.73720940978
9	0.795529184722
10	0.841270992809
12	0.904446340365
14	0.942219290161
16	0.964769103826
18	0.978303726839
20	0.986499508005
25	0.995697676386
30	0.998558990719

35	0.999500077889
40	0.999822919421
45	0.99993648136

Como pode ser visto, com o decorrer do tempo, aumenta a probabilidade do algoritmo ter finalizado independente de ter entrado ou não no *loop*.

4.5.4 Modelo *Hardware-Software*

O modelo *Hardware-Software*, como visto na Ilustração 20, foi criado para verificar a confiabilidade, disponibilidade, inconfiabilidade e a indisponibilidade [1] de um sistema computacional onde há a possibilidade de falha tanto do *hardware* quanto do *software* que pode falhar ou deixar de estar ativo devido a uma falha no *hardware*.

Tabela 5. *Hardware-Software*:

Tempo	Conf./Disp.	Inconf./Indisp.	Hard. Ativo	Soft. Ativo
0	1	0	1	1
0.1	0.951229424501	0.0487705754993	0.975309912028	0.951229424501
0.2	0.904837418036	0.095162581964	0.951229424501	0.904837418036
0.3	0.860707976425	0.139292023575	0.927743486329	0.860707976425
0.4	0.818730753078	0.181269246922	0.904837418036	0.818730753078
0.5	0.778800783071	0.221199216929	0.882496902585	0.778800783071
0.6	0.740818220682	0.259181779318	0.860707976425	0.740818220682
0.7	0.704688089719	0.295311910281	0.839457020769	0.704688089719
0.8	0.670320046036	0.329679953964	0.818730753078	0.670320046036

0.9	0.637628151622	0.362371848378	0.798516218759	0.637628151622
1	0.606530659713	0.393469340287	0.778800783071	0.606530659713

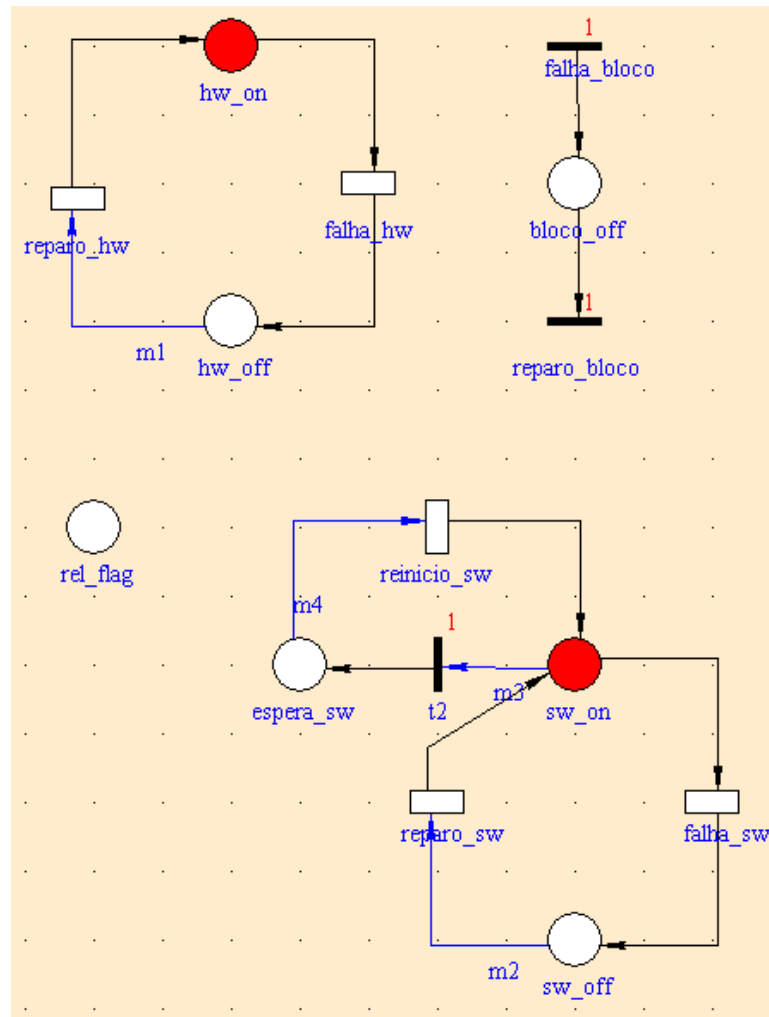


Ilustração 20: Hardware-Software

Como visto no estudo original, a tendência do sistema como um todo é de se tornar menos confiável com o decorrer do tempo, dado que o *hardware* tende a se deteriorar e que o *software* tende a falhar por erro próprio ou por estar esperando que o *hardware* seja reparado.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

A ferramenta *SPNP-Gui* se mostrou eficiente ao retornar dados precisos quando utilizada como ferramenta de modelagem e análise de dados matemáticos. Também se mostrou rápida ao demorar, geralmente, menos de 1s (em um processador com 8 núcleos de 2.93GHz) para gerar os dados mostrados nesta monografia. Deve-se levar em conta que a ferramenta foi feita em uma época em que multiprocessamento nos computadores pessoais ainda não era comum e que, por isso, provavelmente não está otimizada para utilizar todos os núcleos do processador.

O *SPNP-Gui*, como foi visto, é capaz de modelar redes de Petri para análises matemáticas, para verificação de confiabilidade e tolerância a falhas como um todo, além de servir também para análise de cadeias de Markov advindas dos modelos em rede de Petri.

Uma característica essencial na ferramenta é a liberdade de edição das funções, da própria rede gerada e de opções sobre a continuidade da análise no caso de encontrar estados indesejados na rede. Isto dá ao pesquisador a opção de poder parar o processamento quando é encontrado um estado absorvente, por exemplo, ou de continuar tendo em mente que há um estado desse tipo, mas que, mesmo assim, pretende continuar estudando a rede modelada.

Por possuir poucos botões e menus, a ferramenta tende a se tornar familiar e intuitiva com pouco tempo de uso. Os botões com desenhos dos elementos da rede de Petri e menus bem descritos ajudam o entendimento e uso prático da mesma. Acostuma-se a montar redes de Petri, criar novos modelos dentro de um mesmo projeto, analisar dados, criar e modificar funções.

5.1.1 Qualidades diferenciais da ferramenta

O *SPNP-Gui* gera um código-fonte com toda a rede que foi modelada de maneira visual e torna essa função passível de modificação por parte do usuário. O pesquisador pode inserir uma função de qualquer complexidade dentro da rede e pode modificá-la a qualquer momento, inclusive após as análises para uma rápido teste, se achar necessário. Isso se torna possível porque o código-fonte final contém todas as funções geradas pelo usuário.

Por ser uma interface visual para o pacote *SPNP*, o pesquisador não precisa mais saber algo de programação para criar o mais simples modelo em redes de Petri ou para realizar manutenção de grande parte do modelos gerados.

Rápido retorno de resultados.

Indicação precisa da linha do código C em que está localizado o erro.

Pode-se estudar a parte visual com o código-fonte gerado para aprender como usar o *SPNP* (parte não visual) de maneira comparativa.

5.1.2 Problemas graves

O sistema operacional alvo no momento da criação da ferramenta já possui mais de dez anos, isso gera incompatibilidades (com sistemas operacionais posteriores da mesma empresa) e travamentos que levam a perda dos modelos nos usuários que não tenham o hábito de salvar com frequência.

A montagem gráfica da rede de Petri, por vezes, é trabalhosa pois os arcos nem sempre estão apontando para onde o usuário imagina (geralmente alguns *pixels* de distância), quando se está criando o modelo, e isso faz com que o arco não seja criado quando não aponta precisamente para um lugar ou para uma transição.

5.2 Trabalhos futuros

A experiência com a ferramenta tornou possível visualizar alguns aspectos que podem se tornar alvo de pesquisa. Quer para melhoria e divulgação do *SPNP-Gui*, ou para o uso prático em dispositivos que não sejam computadores pessoais.

Em todos os casos é preciso entrar em contato com o autor para solicitar permissão de uso da ferramenta e de divulgação dos resultados.

5.2.1 Nova interface

O núcleo SPNP é bastante consistente e robusto, nunca retornando resultados inesperados nem travando o funcionamento do computador. Porém, a interface atual chega a ser problemática quando utiliza símbolos padrões (como o “x” para fechar janelas) mas que não realizam sua função para a qual o usuário está acostumado.

Tendo em vista que a ferramenta consiste de duas partes onde uma delas é bastante confiável e outra não tão robusta, torna-se um possível caso de estudo a geração de uma nova interface. Mesmo que seja baseada na interface atual (visando melhor adaptação de usuários antigos), ela deve conter menos erros e se torne visualmente mais agradável e consistente.

5.2.2 Versão da ferramenta na *Internet*

Caso o autor da ferramenta considere interessante, pode-se criar um *site* que através de controle de usuário – nome e senha – dê acesso imediato à ferramenta salvando os dados do modelo em algum servidor. Isso tornaria a ferramenta independente de plataforma, de instalação e garantindo que o núcleo SPNP, rodando no servidor, não possa ser desviado para uso não autorizado em algum programa que o utilize, por exemplo.

O *site* pode dar ao usuário a opção de tornar seus modelos públicos o que facilita a troca de modelos de redes de Petri, além de possibilitar a ajuda mútua entre pesquisadores.

Pode também haver controle de versão nos modelos, o que evita que modelos públicos sejam danificados irreversivelmente por algum usuário mal intencionado, além disso o usuário modificador pode ser registrado e tomar-se-ia as providências cabíveis pela tentativa de sabotagem.

5.2.3 Estudo de uso do pacote SPNP em sensores

Com o SPNP-Gui pode-se criar uma rede de Petri que analise a confiabilidade, inconfiabilidade, disponibilidade e indisponibilidade, por exemplo, de um sensor. Essa rede gerada talvez possa ser inserida em algum sensor juntamente com o pacote SPNP. O sensor deve ser capaz de processar algoritmos em C, para que de tempos em tempos ele recalcule a possibilidade dele ainda estar em correto funcionamento. Desse modo, por exemplo, um sensor de monitoramento de temperatura em uma usina nuclear pode estar monitorando a temperatura da usina e, ao mesmo tempo, recalculando a própria confiabilidade dado que uma mudança na temperatura do ambiente pode ter tornado o próprio sensor menos confiável.

Bibliografia

[1] FERNANDES S. M. M. ***Avaliação de Dependabilidade de Sistemas com Mecanismos Tolerantes a Falhas: Desenvolvimento de um Método Híbrido Baseado em EDSPN e Diagrama de Blocos***. Tese de Doutorado na Universidade Federal de Pernambuco, 2007, Recife, Brasil.

[2] GADOMSKI A. M. ***An approach to an Unified Engineering Meta-Ontology: Universal Domain Paradigm: SPG Representation of a Domain-of-Activity***. e-paper, the ENEA Agency MKEM Server, <http://erg4146.casaccia.enea.it/wwwerg26701/gad-diag0.htm>, última atualização em 15 de Agosto de 2006, Itália.

[3] Markov. A. A. ***Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga.Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete***, 2ª edição, tom 15, pp 135-156, 1906, Rússia.

[4] MOLLOY M.K. ***Performance Analysis Using Stochastic Petri Nets***, IEEE Trans. Comput., C-31 (9),pp. 913–917, 1982, Estados Unidos.

[5] PETRI, C. A. ***Kommunikation mit Automaten***. Faculty of Mathematics and Physics, 1965, Technische Universität Darmstadt, Alemanha.

[6] ROBBINS, S. P. ***O processo administrativo: integrando teoria e pratica***. São Paulo: Atlas, 1981 (1990, 1a. ed. 4a .tiragem), Brasil.

[7] TRIVEDI, K.S. SPNP-Gui <http://people.ee.duke.edu/~kst/>. ***SPNP-Gui e manual do pacote SPNP***. Última visualização em 8 de novembro de 2011, Estados Unidos.

[8] U.S. Department of Defense, ***Electronic Reliability Design Handbook, U.S. Military Handbook MIL-HDBK-338B***, 1998, Estados Unidos.

[9] ZIMMERMANN A, FREIHEIT J, GERMAN R, HOMMEL G. ***Petri net modelling and performability evaluation with TimeNET 3.0*** 11º Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Lecture

Notes in Computer Science, vol. 1786, pp. 188-202, 2000, Schaumburg, Estados Unidos.

Apêndice A

Códigos C de redes de Petri

Código referente à Ilustração 17

```
#include <stdio.h>

#include "user.h"

int m1 ();

double lambda = 1;

double mi = 1;

/* ===== OPTIONS ===== */

void options() {

    iopt(IOP_SSMETHOD, VAL_POWER);
}

/* ===== DEFINITION OF THE NET ===== */

void net() {

    /* ===== PLACE ===== */

    place("X_ON");

    init("X_ON",1);

    place("X_OFF");

    place("Rel_Flag");

    init("Rel_Flag",1);

    /* ===== TRANSITION ===== */
```

```
/* Timed Transitions */

rateval("Falha_X",lambda);

rateval("Reparo_X",mi);

/* ===== ARC ===== */

/* Input Arcs */

iarc("Falha_X","X_ON");

viarc("Reparo_X","X_OFF",m1);

/* Output Arcs */

oarc("Falha_X","X_OFF");

oarc("Reparo_X","X_ON");

}

/* CARDINALITY Functions */

int m1 () {
    if(mark("Rel_Flag")==1) { return 2; }
    else { return 1; }
}

/* ===== DEFINITION OF THE FUNCTIONS ===== */

int assert() {}

void ac_init() {}

void ac_reach() {}

double rfunc(){
    return mark("X_ON");
}
```



```

void ac_final() { int i;
    /* Análise transiente com múltiplos pontos temporais*/
    for ( i = 1; i < 10; i++ ) {
        solve( (double) i );
        pr_expected("probability X_ON: ", rfunc);
    }

    for ( i = 10; i < 20; i += 2 ) {
        solve( (double) i );
        pr_expected("probability X_ON: ", rfunc);
    }

    for ( i = 20; i < 50; i += 5 ) {
        solve( (double) i );
        pr_expected("probability X_ON: ", rfunc);
    }
    pr_std_average(); //exibe as medias
}

```

Código referente à Ilustração 18

```

#include <stdio.h>
#include "user.h"

/* ===== OPTIONS ===== */
void options() {
    iopt(IOP_PR_RGRAPH, VAL_YES);
    iopt(IOP_PR_MC, VAL_YES);
    iopt(IOP_PR_PROB, VAL_YES);
}

/* ===== DEFINITION OF THE NET ===== */
void net() {
    /* ===== PLACE ===== */
    place("P0");
}

```

```
init("P0",1);
place("P1");
place("P2");
place("P3");
place("P4");

/* ===== TRANSITION ===== */

/* Timed Transitions */
rateval("T0",0.25);
rateval("T1",0.25);
rateval("T2",0.25);
rateval("T3",0.25);
rateval("T4",0.25);

/* ===== ARC ===== */
/* Input Arcs */
iarc("T0","P0");
iarc("T1","P1");
iarc("T2","P2");
iarc("T4","P4");
iarc("T4","P3");
iarc("T3","P3");

/* Output Arcs */
oarc("T0","P1");
oarc("T0","P2");
oarc("T1","P3");
oarc("T2","P4");
oarc("T3","P1");
oarc("T4","P0");
}

/* ===== DEFINITION OF THE FUNCTIONS ===== */
int assert() { }
```

```
void ac_init() {
    /* Information on the net structure */
    pr_net_info();
}

void ac_reach() {
    /* Information on the reachability graph */
    pr_rg_info();
}

void ac_final() {
    int loop;
    solve(INFINITY);
    pr_std_average();
}
```

Código referente à Ilustração 20

```
#include <stdio.h>
#include "user.h"

int m1();
int m2();
int m3();
int m4();

int falhaBloco();
int reparoBloco();

/* ===== OPTIONS ===== */
void options() {
    iopt(IOP_SSMETHOD, VAL_POWER);
}
```

```
int m1(){
    if(mark("Rel_Flag")==1) { return 2; }
    else { return 1; }
}

int m2(){
    if(mark("Rel_Flag")==1) { return 2; }
    else { return 1; }
}

int m3(){
    if(mark("Hw_ON")==0) { return 1; }
    else { return 2; }
}

int m4(){
    if(mark("Rel_Flag")==1) { return 2; }
    else { return 1; }
}

int falhaBloco(){
    if( (mark("Hw_ON") == 0 || mark("Sw_ON")== 0) && mark("Bloco_OFF")== 0)
        { return (1); }
    else
        { return (0); }
}

int reparoBloco(){
    if( (mark("Hw_ON")>0 && mark("Sw_ON")>0 ) && mark("Bloco_OFF")== 1 )
        { return (1); }
    else
        { return (0); }
}

int t1Guard() { return 1; }

/* ===== DEFINITION OF THE NET ===== */
```

```
void net() {
/* ===== PLACE ===== */
    place("Hw_ON");
    init("Hw_ON",1);
    place("Rel_Flag");
    init("Rel_Flag",1);
    place("Espera_Sw");
    place("Sw_OFF");
    place("Sw_ON");
    init("Sw_ON",1);
    place("Bloco_OFF");
    place("Hw_OFF");

/* ===== TRANSITION ===== */
    imm("Falha_Bloco");
    guard("Falha_Bloco", falhaBloco);
    priority("Falha_Bloco",1);
    probval("Falha_Bloco",1);
    imm("Reparo_Bloco");
    guard("Reparo_Bloco", reparoBloco);
    priority("Reparo_Bloco",1);
    probval("Reparo_Bloco",1);
    imm("t1");
    guard("t1", t1Guard);
    priority("t1",1);
    probval("t1",1);

    rateval("Reparo_Hw",0.25);
    rateval("Falha_Hw",0.25);
    rateval("Reinicio_Sw",0.25);
    rateval("Reparo_Sw",0.25);
    rateval("Falha_Sw",0.25);

/* ===== ARC ===== */
    iarc("Falha_Hw","Hw_ON");
```

```
    iarc("Falha_Sw","Sw_ON");
    iarc("Reparo_Bloco","Bloco_OFF");

    viarc("Reparo_Hw", "Hw_OFF", m1);
    viarc("Reparo_Sw", "Sw_OFF", m2);
    viarc("t1", "Sw_ON", m3);
    viarc("Reinicio_Sw", "Espera_Sw", m4);

    oarc("Falha_Hw","Hw_OFF");
    oarc("Reparo_Hw","Hw_ON");
    oarc("Reparo_Sw","Sw_ON");
    oarc("Falha_Sw","Sw_OFF");
    oarc("t1","Espera_Sw");
    oarc("Reinicio_Sw","Sw_ON");
    oarc("Falha_Bloco","Bloco_OFF");
}

/* ===== DEFINITION OF THE FUNCTIONS ===== */

int assert() { }

void ac_init() { }

void ac_reach() { }

double confFunc() {
    if(mark("Bloco_OFF") == 0) { return (1.0);}
    else { return(0.0);}
}

double inconfFunc() {
    if(mark("Bloco_OFF") == 1) { return (1.0);}
    else { return (0.0);}
}

double hwOn() {
    if(mark("Hw_ON") == 1) return (1.0);
```

```
        else return (0.0);
    }

double swOn() {
    if(mark("Sw_ON") == 1) return (1.0);
    else return (0.0);
}

void ac_final() {
    double i;
    for(i = 0; i < 1; i+=0.1) {
        solve((double)i);
        pr_expected("Confiabilidade e disponibilidade (Bloco OFF = 0):", confFunc);
        pr_expected("Inconfiabilidade e indisponibilidade (Bloco OFF = 1):",
inconfFunc);
        pr_expected("Hardware ativo:", hwOn);
        pr_expected("Software ativo:", swOn);
    }
}
```