

Adauto Trigueiro de Almeida Filho
Clerton Ribeiro de Araújo Filho
Hugo Parente Lima

Opulus, um editor/simulador para redes de Petri
Lugar/Transição

Campina Grande

Agosto 2008

Adauto Trigueiro de Almeida Filho
Clerton Ribeiro de Araújo Filho
Hugo Parente Lima

Opulus, um editor/simulador para redes de Petri
Lugar/Transição

Projeto de Pesquisa de Monografia apresentada,
como exigência parcial para obtenção do título
de Bacharel em Ciências da Computação, à
Banca Examinadora da Universidade Federal de
Campina Grande.

Orientador:
Jorge Abrantes

UNIVERSIDADE FEDERAL DE CAMPINA GRANDE

Campina Grande
Agosto 2008

Projeto de Pesquisa de Monografia sob o título “*Opulus, um editor/simulador para redes de Petri Lugar/Transição*”, defendida por Aduino Trigueiro de Almeida Filho, Clerton Ribeiro de Araújo Filho e Hugo Parente Lima, apresentada como exigência parcial para obtenção do título de Bacharel em Ciências da Computação, à Banca Examinadora da Universidade Federal de Campina Grande.

BANCA EXAMINADORA

Prof. Dr. Jorge Abrantes
Orientador

Prof. Dr. Antão Moura
Professor

Sumário

Lista de Figuras

Resumo

Abstract

1	Introdução	p. 9
1.1	Definição Geral	p. 9
1.2	Áreas de Aplicação	p. 9
1.3	A disciplina	p. 11
1.4	Motivação	p. 11
1.5	A Ferramenta Opulus	p. 12
2	Fundamentação Teórica	p. 13
2.1	Conceitos básicos	p. 13
2.2	Regras de disparo	p. 15
2.3	Redes de Petri temporizadas	p. 16
2.4	Redes com Arcos inibidores	p. 16
2.5	Redes com capacidade	p. 17
2.6	Análise	p. 18
2.6.1	Grafo de Cobertura	p. 18
2.7	Propriedades comportamentais	p. 19
2.7.1	Alcançabilidade	p. 19

2.7.2	Limitação	p. 20
2.7.3	Vivacidade	p. 20
2.7.4	Persistência	p. 21
2.7.5	Reversibilidade	p. 21
3	A Ferramenta Oplus	p. 22
3.1	Requisitos	p. 22
3.1.1	Requisitos funcionais	p. 22
3.1.2	Requisitos não funcionais	p. 23
3.2	Arquitetura	p. 23
3.3	Ferramentas Utilizadas	p. 24
3.3.1	GCC e MinGW	p. 24
3.3.2	Qt4	p. 24
3.3.3	libxslt	p. 24
3.3.4	Boost Graphs Library	p. 24
3.3.5	Graphviz	p. 25
3.3.6	CMake	p. 25
3.3.7	Doxygen	p. 25
3.4	Implementação	p. 25
3.4.1	Simulação de Redes de Petri	p. 25
3.4.2	Infraestrutura para plugins	p. 26
3.4.3	Geração do Grafo de Cobertura	p. 27
3.4.4	Propriedades comportamentais	p. 29
	Anexo A – Processo de desenvolvimento	p. 31
A.1	Descrição do processo	p. 31
	Anexo B – The GNU General Public License	p. 33

Lista de Figuras

1.1	Rede de Petri representando uma relação de causalidade: seqüencialidade. . .	p. 10
1.2	Rede de Petri representando uma relação de causalidade: conflito.	p. 10
1.3	Rede de Petri representando uma relação de causalidade: concorrência. . . .	p. 11
1.4	Rede de Petri representando uma relação de causalidade: sincronização. . . .	p. 11
2.1	Exemplo produtor/consumidor	p. 17
2.2	Grafo de cobertura	p. 19
3.1	Arquitetura da ferrameta	p. 24
3.2	Diagrama de classes do módulo OpulusCore.	p. 26
3.3	Rede de petri com infinitas marcações, transições em verde estão ativas . . .	p. 28
3.4	Grafo de cobertura da rede 3.3 sem capacidade.	p. 29
3.5	Grafo de cobertura da rede 3.3 contendo capacidade de 2 fichas no lugar <i>P1</i> . .	p. 29
3.6	Plugin de geração de grafo de cobertura	p. 30

Resumo

Esta monografia tem por objetivo descrever a ferramenta — *Opulus* —. Em cursos introdutórios de Redes de Petri, em geral, não há uma ferramenta, simples e intuitiva, que auxilie o professor a demonstrar grande parte dos conceitos de Redes de Petri lugar/transição. Diante desse problema, a ferramenta descrita nesta monografia foi desenvolvida visando oferecer um editor de fácil utilização para criação redes de petri lugar/transição. Além disso, possibilita ao usuário realizar análise e simulações, de forma assistida ou automática, na rede. O desenvolvimento da ferramenta proporcionou ao acadêmicos envolvidos o contato com a teoria de Redes de Petri, além do contato com uma nova linguagem de programação e frameworks.

Abstract

[Resumo em inglês, quando o resumo estiver fechado]

1 Introdução

1.1 Definição Geral

A teoria das redes de Petri teve início em 1962 com a tese de doutorado de Carl Adam Petri, intitulada de *Kommunikation mit Automaten* [1], na faculdade de Matemática e Física da Universidade de Darmstadt, localizada na antiga Alemanha Ocidental. Redes de Petri é uma técnica de modelagem gráfica que permite um bom nível de abstração comparada com outros modelos gráficos. É um modelo tipo estado-evento, onde cada evento provoca a mudança no estado do modelo. Uma rede de Petri também pode ser considerada como um tipo de grafo (grafo bipartido) que permite modelar as propriedades estáticas de um sistema em eventos discretos.

1.2 Áreas de Aplicação

Sendo uma ferramenta gráfica e matemática de modelagem, o seu domínio de aplicação se torna amplo, sendo aplicadas condizentemente em sistemas assíncronos com elevada concorrência ou paralelismo. [2] As redes de Petri têm uma ampla aplicação na ciência da computação, dentre as principais áreas podemos destacar: sistemas operacionais, redes de computadores, sistemas distribuídos, banco de dados e muitas outras. Em sistemas operacionais as redes de Petri podem ser usadas na modelagem e verificação de várias características como a concorrência, verificação de conflitos, sincronização de processos, compartilhamento de recursos, entre outros [3].

Outra área em que o uso de redes de Petri demonstra ser muito interessante é a modelagem de protocolos de comunicação em redes de computadores. Geralmente nos protocolos, as transições são nítidas, por exemplo, na recepção ou transmissão de mensagem. E situações onde a exclusão mútua está presente também são comuns. Um exemplo deste caso acontece quando se precisa escolher um receptor de uma mensagem dentre vários. Situações como essas são modeladas utilizando as redes de Petri [4].

Além de aplicações na ciência da computação existem outras áreas também fazem uso de redes de Petri. Por exemplo, na modelagem de sistemas flexíveis de manufatura, automação de manufatura em geral e na modelagem e análise de processos de negócios [5].

Diante do exposto, pode-se ver como esta ferramenta matemática e gráfica de modelagem pode prover uma visualização de alta abstração e um acompanhamento dinâmico dos problemas modelados.

As Redes de Petri podem ser divididas em redes de alto nível e redes de baixo nível. As redes de baixo nível são as redes cujo o significado de suas fichas não são diferenciáveis, a não ser pela estrutura da rede à qual estão associadas. As rede de alto nível são aquelas em que as fichas possuem alguma semântica associada, viabilizando a sua diferenciação [6].

As Redes de Petri lugares/transição são redes de baixo nível. Elas são compostas por três elementos principais: lugares, transições e arcos. As transições, representadas por retângulos, simbolizam ações, ou eventos, que ocorrem no sistema e possuem pré-condições que vão permitir seu disparo, acarretando em uma mudança no estado do sistema e gerando pós-condições em consequência desta mudança. Os lugares representados pelos círculos são responsáveis por representar o estado atual do sistema. Os arcos são elos que ligam um lugar e uma transição, são direcionados e podem possuir um peso associado.

O modelo nos permite expressar com facilidade relações de causalidade como, por exemplo: seqüencialidade (1.1), conflito (1.2), concorrência (1.3) e sincronização (1.4).

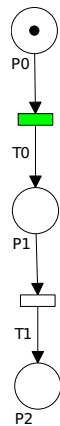


Figura 1.1: Rede de Petri representando uma relação de causalidade: seqüencialidade.

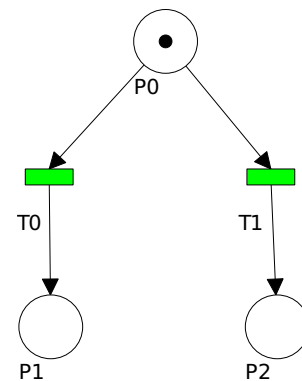


Figura 1.2: Rede de Petri representando uma relação de causalidade: conflito.

Rede de Petri é, portanto, um formalismo que nos permite modelar com grande expressividade sistemas discretos.

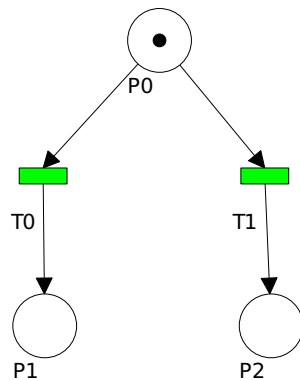


Figura 1.3: Rede de Petri representando uma relação de causalidade: concorrência.

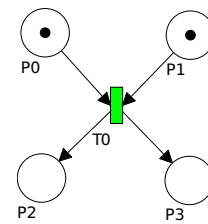


Figura 1.4: Rede de Petri representando uma relação de causalidade: sincronização.

1.3 A disciplina

A disciplina Redes de Petri é oferecida periodicamente pela Unidade Acadêmica de Sistemas de Computação, do Centro de Engenharia Elétrica e Informática – Universidade Federal de Campina Grande, ministrada pelo Prof. Dr. Jorge Abrantes, proporciona aos alunos que a cursam um contato inicial com a teoria de Redes de Petri.

1.4 Motivação

Há inúmeras ferramentas para a edição e simulação de Redes de Petri lugar/transição. Entretanto, a maioria delas não contemplavam todos os conceitos vistos na disciplina supracitada.

Diante de tal situação, os autores desta monografia, desenvolveram a idéia de construir uma ferramenta que contemplasse, de forma mais abrangente os conceitos de Rede de Petri lugar/transição.

A motivação do desenvolvimento surgiu da necessidade de se ter uma ferramenta para edição e simulação de redes de Petri lugar/transição que fosse de fácil utilização pelos alunos da disciplina, para que a mesma fosse utilizada para auxiliar o professor durante o curso.

Em Projeto em Computação I e Projeto em Computação II foi dada continuidade ao desenvolvimento da ferramenta, visto que na disciplina de Redes de Petri não houve tempo suficiente para a conclusão do projeto.

1.5 A Ferramenta Opulus

A ferramenta Opulus é um editor/simulador de Redes de Petri lugar/transição. A ferramenta possibilita que o usuário alvo, ou seja, estudantes de cursos introdutórios de Redes de Petri, exercite a modelagem de problemas com as redes lugar/transição.

Ela é composta por um espaço para edição das redes, uma visão geral da rede produzida em uma tela miniaturizada e um painel que expõe as propriedades de cada item da rede que está sendo editada.

A ferramenta proporciona ao usuário a possibilidade de fazer simulações na rede de petri lugar/transição de forma manual, selecionando a transição desejada, ou de forma automática. Na simulação automática o usuário poderá disparar uma ou N transição dependendo da sua escolha.

Com os plugins existentes na ferramenta o usuário poderá gerar a árvore de cobertura da rede de petri e verificar propriedades comportamentais como limitação e reversibilidade.

2 *Fundamentação Teórica*

2.1 Conceitos básicos

Uma Rede de Petri possui um conjunto de elementos capaz de descrever as diversas partes de sistemas típicos estudados pela Ciência da Computação, tais como: concorrência, controle, conflitos, sincronização e compartilhamento.

Uma Rede de Petri é composta pelos seguintes elementos [7]:

- Lugares ou Places: representam uma condição, uma atividade ou um recurso;
- Fichas, Marcas ou Tokens: representam o estado de um sistema;
- Transições: representam um evento;
- Arcos: indicam os lugares de entrada ou saída para as transições.

Portanto, ainda sem considerarmos a existência dos tokens, podemos definir formalmente a estrutura de uma Rede de Petri como uma 4-upla $R = (P, T, F, W)$ onde [8]:

- $P = p_1, p_2, \dots, p_m$ é o conjunto finito de lugares.
- $T = t_1, t_2, \dots, t_n$ é o conjunto finito de transições.
- $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$ os conjuntos P e T são disjuntos e não vazios.
- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto finito dos arcos.
- $W : F \rightarrow \mathbb{N}$ são os pesos dos arcos.

Ao se modelar um sistema através de Redes de Petri procura-se representar os estados antes e depois de cada evento neste sistema. O elemento responsável por indicar qual o estado de uma Rede de Petri é um token (marca). Portanto, uma Rede de Petri possui uma marcação inicial [8], que pode ser descrita através da seguinte notação:

- Marcação de uma rede de Petri é uma função $M : P \rightarrow \mathbb{N}$.
- $M(p)$ é a marcação do lugar p (número de marcas ou fichas contidas em p). A evolução da marcação simula o comportamento dinâmico de um sistema.

A marcação inicial faz parte do projeto do modelo. Uma falha na definição da marcação inicial pode resultar em um modelo inconsistente ou com falhas de execução.

Além da marca (token) presente dentro de um lugar, existe um peso associado a cada arco. Este peso é muito importante e muitas vezes é necessário durante o projeto do modelo [7].

Assim, podemos apresentar uma definição mais completa de uma Rede de Petri como uma 5-upla $PN = (P, T, F, W, M_0)$ [2]:

- $P = p_1, p_2, \dots, p_m$ é o conjunto finito de lugares.
- $T = t_1, t_2, \dots, t_n$ é o conjunto finito de transições.
- $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$ os conjuntos P e T são disjuntos e não vazios.
- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto finito dos arcos.
- $W : F \rightarrow \mathbb{N}$ são os pesos dos arcos.
- $M_0 : P \rightarrow \mathbb{N}$ é a marcação inicial.

Rede de Petri é uma estrutura de rede mais uma marcação inicial, é portanto, a 2-upla $PN = (N; M_0)$ [2].

Os elementos de P (lugares) são também chamados de p-elementos. Os elementos de T (transições) são também chamados de t-elementos.

Seja $t \in T$:

- $t = \{p \in P \mid (p, t) \subseteq F\}$ é o pré-conjunto de t (entradas de t).
- $t^* = \{p \in P \mid (t, p) \subseteq F\}$ é o pós-conjunto de t (saídas de t).

Seja $p \in P$:

- $p = t \in T \mid (t, p) \subseteq F$ é o pré-conjunto de p (entradas de p).
- $p^* = t \in T \mid (p, t) \subseteq F$ é o pós-conjunto de p (saídas de p).

2.2 Regras de disparo

A função de uma transição é representar um evento, ou seja, a transição é responsável por modificar a marcação de uma Rede de Petri, portanto, modificar o estado de uma Rede de Petri. Esta ocorrência só é válida quando existem tokens nos lugares de entrada de uma transição. Sendo assim, ao modificar o estado ou a marcação de uma Rede de Petri, basta disparar as transições existentes.

Para simular a dinâmica de um sistema, a marcação, ou estado, de uma PN evolui de acordo com a regra de disparo enunciada a seguir [2]:

1. (Pre-condição): Uma transição t está habilitada (ou é gatilhável) se cada lugar de entrada p de t contém pelo menos $w(p,t)$ fichas, onde $w(p,t)$ é o peso do arco de p para t
2. Uma transição habilitada pode ou não disparar (ou gatilhar).
3. (Pos-condição): Ao disparar t , $w(p,t)$ fichas são removidas de cada entrada p de t e $w(t,p)$ fichas são acrescentadas a cada saída p de t , onde $w(t,p)$ é o peso do arco de t para p [2].

Exemplo 2.1 [2]:

Exemplo de uma Rede de Petri

$$PN = (P, T, F, W, t, M_0)$$

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4, t_5\}$$

$$F = \{(p_1, t_1), (t_1, p_2), (t_1, p_3), (p_2, t_2), (t_2, p_4), (p_4, t_3), (t_3, p_2), (p_3, t_4), (p_4, t_4), (t_4, p_5), (p_5, t_5), (t_5, p_1)\}$$

$$W = \{[(p_1, t_1), 1], [(t_1, p_2), 1], [(t_1, p_3), 2], [(p_2, t_2), 1], [(t_2, p_4), 1], [(p_4, t_3), 1], [(t_3, p_2), 1], [(p_3, t_4), 2], [(p_4, t_4), 1], [(t_4, p_5), 1], [(p_5, t_5), 1], [(t_5, p_1), 1]\}$$

$$M_0 = \{(p_1, 1), (p_2, 0), (p_3, 0), (p_4, 0), (p_5, 0)\}$$

A estrutura da rede de Petri descrita é a quadrupla $N = (P, T, F, W)$, P , T , F e W conforme definidos. A rede PN é, por conseguinte, a dupla $(N; M_0)$.

Uma vez subentendida a ordem em que os lugares aparecem na notação das marcações, podemos indicar a marcação inicial da rede acima simplesmente por $M_0 = (1, 0, 0, 0, 0)$.

2.3 Redes de Petri temporizadas

Redes de Petri com transições disparam desde que os lugares de entrada possuam tokens suficientes para a ocorrência do disparo, no entanto, pode ser necessário associar ao disparo um retardo. Sendo assim, para cada transição atribuímos um valor de tempo.

Em relação às Rede de Petri Temporizadas [8] podemos extrair os seguintes significados:

- Tempo de aquisição de um recurso.
- Tempo de utilização de um recurso.
- Tempo em que não se necessita de um recurso.

Uma rede de lugar/transição temporizada é uma 6-tupla (P, T, F, W, t, M_0) , em que $P, T, F, W \in M_0$ representam respectivamente um conjunto de lugares, um conjunto de transições, um conjunto de arcos, funções peso e marcação inicial. t é uma função de tempo $t : T \rightarrow 1, 2, \dots$ mapeando cada lugar na rede nos números naturais [2].

A principal vantagem de uma Rede de Petri temporizada é a relação de tempo, não existente numa rede lugar/transição. No entanto, trabalhar com uma Rede de Petri temporizada significa poder alterar a definição de estado da rede. A relação de tempo altera os estados/marcações alcançáveis.

2.4 Redes com Arcos inibidores

A Figura 2.1 mostra um exemplo de um sistema produtor/consumidor com prioridade. O consumidor A tem prioridade de consumo, B só pode consumir se não houver produtos no buffer A, e houver produtos no buffer B. Isto só pode ser modelado com a introdução de um arco inibidor. Este é representado por uma linha cheia ou tracejada com um pequeno círculo na ponta. O arco inibidor habilita uma transição quando não há fichas no lugar de entrada por ele conectado com esta transição. A transição não está habilitada se há alguma ficha no lugar de entrada conectado com ela por um arco inibidor. Com relação aos demais lugares, a regra de disparo não se altera. O arco inibidor permite, por conseguinte, representar teste de zero, o que aumenta o poder de modelagem de uma PN. Uma PN com arco inibidor é uma rede de Petri estendida [2].

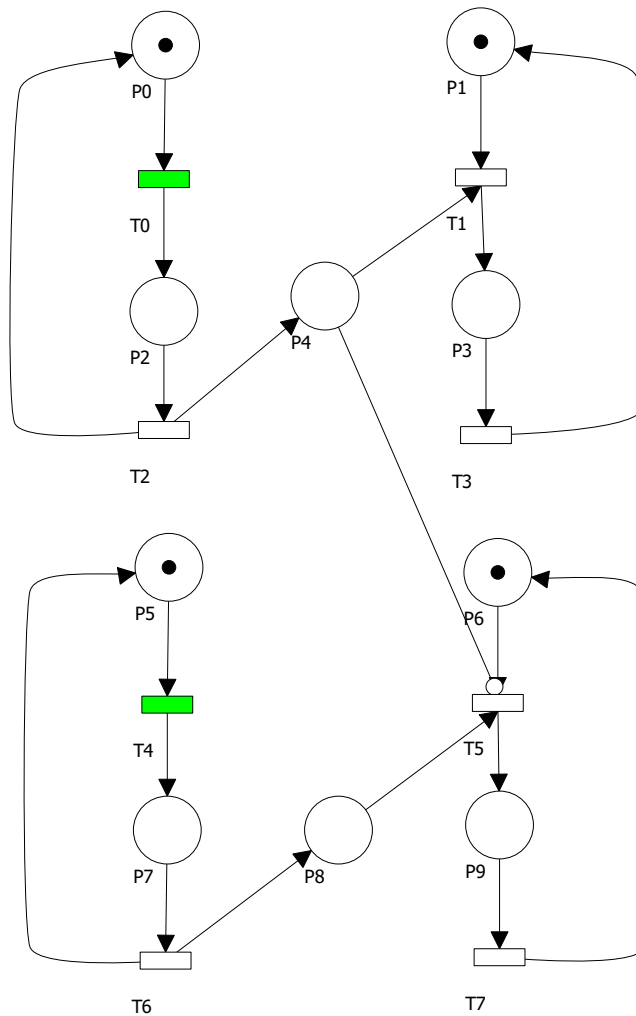


Figura 2.1: Exemplo produtor/consumidor

2.5 Redes com capacidade

A Rede de Petri em que cada lugar pode acomodar um número ilimitado de fichas, como já foi definido anteriormente, é uma Rede de Petri de capacidade infinita, mas se ela tiver um número limitado de fichas em que cada lugar pode acomodar pode-se denominar como uma Rede de Petri de capacidade finita.

Uma Rede de Petri de capacidade finita formalmente é definida como uma 6-tupla (P, T, F, W, K, M_0) , onde:

- $P = p_1, p_2, \dots, p_m$ é o conjunto finito de lugares.
- $T = t_1, t_2, \dots, t_n$ é o conjunto finito de transições.
- $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$ os conjuntos P e T são disjuntos e não vazios.

- $F \subseteq (P \times T) \cup (T \times P)$ é o conjunto finito dos arcos.
- $W : F \rightarrow \mathbb{N}$ são os pesos dos arcos.
- $K : P \rightarrow \mathbb{N}$ é uma função capacidade.
- $M_0 : P \rightarrow \mathbb{N}$ é a marcação inicial.

As regras de disparos descritas anteriormente, que se aplica a Redes com capacidade infinita, é uma regra de transição fraca.

A redes de capacidade infinita aplica-se a regra de transição estrita. Nesta, a fim de que a quantidade de fichas em cada lugar não exceda sua capacidade, a pre-condição (parte 1 da regra de disparo) fica modificada para:

Uma transição t está habilitada se cada lugar de entrada p de t contém, pelo menos, $w(p, t)$ fichas, onde $w(p, t)$ é o peso do arco de p para t e se cada lugar de saída p de t contém, no máximo, $K(p) - w(t, p)$ fichas, onde $K(p)$ é a capacidade do lugar p e $w(t, p)$ é o peso do arco que vai de t para p . [2]

2.6 Análise

2.6.1 Grafo de Cobertura

Dada uma rede de Petri $PN = (N; M_0)$, a partir da marcação inicial podemos obter novas marcações, tantas quantas forem as transições habilitadas; a partir de cada uma das novas marcações podemos obter outras e assim por diante. Isto resulta na representação das marcações por uma árvore.

Se a rede for ilimitada, a árvore crescerá infinitamente. Para fazê-la limitada, introduzimos um símbolo especial ω que pode ser interpretado como infinito, e que tem as seguintes propriedades: para todo inteiro $n : \omega > n, \omega \pm n = \omega$ e $\omega \geq \omega$.

O algoritmo para obtenção de um grafo de cobertura apresentado a seguir, aplica-se a redes ilimitadas.

1. Rotule a marcação inicial M_0 de raiz e sinalize-a como nova.
2. Enquanto existirem marcações novas, faça o seguinte:
 - (a) Escolha uma nova marcação M .

- (b) Se M for idêntica a outra marcação no caminho da raiz até M , sinalize M com velho e vá para uma outra marcação.
- (c) Se nenhuma transição estiver habilitada em M , sinalize M com fim.
- (d) Enquanto existirem transições habilitadas em M , para cada transição habilitada t em M faça o seguinte:
 - i. Obtenha a marcação M' que resulta do disparo de t em M .
 - ii. Se, no caminho da raiz até M , existir uma marcação M'' tal que $M'(p) \geq M''(p)$ para cada lugar p e $M' \neq M''$, então substitua $M'(p)$ por ω para cada p tal que $M'(p) > M''(p)$.
 - iii. Introduza M' como um nó, desenhe um arco com rótulo t , de M para M' e rotule M' com novo.

O grafo de cobertura de uma rede de Petri $PN = (N; M_0)$ é um grafo orientado rotulado $G(V, E)$. V é o conjunto de todos os nós com rótulos distintos da árvore de cobertura. E é o conjunto de arcos rotulados com transições simples t_k ; representa todos os possíveis disparos de transições simples tais que $M_i(t_k)M_j$, onde M_i e M_j estão em V . Na Figura 2.2 mostramos um exemplo de grafo de cobertura.

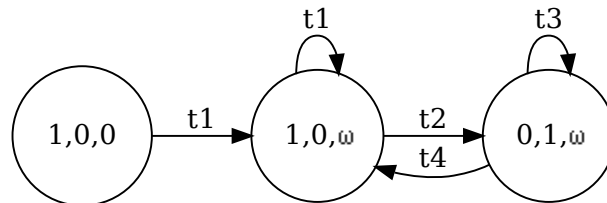


Figura 2.2: Grafo de cobertura

2.7 Propriedades comportamentais

2.7.1 Alcançabilidade

Indica a possibilidade de alcançarmos um determinado conjunto de marcações após disparos de transições a partir de uma marcação inicial [7].

Uma marcação M_n é alcançável a partir de M_0 se existe uma sequência de disparos que transforma M_0 em M_n .

O conjunto de todas as possíveis marcações alcançáveis a partir de M_0 em uma rede de Petri $PN = (N; M_0)$ é denotado por $R(N, M_0)$ ou $R(M_0)$.

O problema da alcançabilidade para Redes de Petri é o problema de determinar se $M_n \in R(M_0)$ para uma dada marcação M_n numa rede (N, M_0) .

Às vezes, estamos interessados apenas nas marcações de um subconjunto de lugares.

Demonstrou-se que o problema da alcançabilidade é decidível, embora em casos gerais tenha alto custo computacional [2].

2.7.2 Limitação

Uma Rede de Petri pode ser dita como k -limitada. Em uma Rede de Petri 3 -limitada, nenhum dos lugares pode exceder o número de tokens definido por k , neste caso, três [7].

Um lugar p de uma rede de Petri $PN = (N, M_0)$ é k -limitado se para toda marcação alcançável $M \in R(M_0)$, $M(p) \leq k$.

Se p é 1 -limitado, dizemos também que p é binário.

Uma rede de Petri PN é limitada se para cada lugar p da rede existe um inteiro k tal que p seja k -limitado.

Uma rede de Petri PN é k -limitada se o número de fichas em qualquer lugar não excede um número finito k para qualquer marcação M alcançável a partir de M_0 , ou seja, $M(p) \leq k$ para todo p , para todo $M \in R(M_0)$ [2].

2.7.3 Vivacidade

A vivacidade indica se uma Rede de Petri está livre de impasses (deadlocks). Neste caso não existe a ocorrência de uma transição que não possa ser disparada. Uma transição morta é uma transição que não pode mais ser disparada. Uma Rede de Petri morta se encontra em impasse total, ou seja, todas as transições estão mortas [7].

Uma rede de Petri PN é viva se, qualquer que seja a marcação atingida a partir de M_0 , for possível disparar qualquer transição da rede, mediante alguma sequência de disparos.

Uma rede de Petri viva garante uma operação livre de bloqueios [2].

2.7.4 Persistência

Persistência indica que para qualquer par de transições habilitadas, o disparo de uma delas não desabilita o disparo da outra [7].

Numa rede persistente, uma transição uma vez habilitada, permanecerá habilitada até disparar.

Segue uma definição equivalente:

Uma rede de Petri (N, M_0) é persistente se, para toda $M \in R(M_0)$ para todo par de transições distintas $t, t' \in T$, $M(t > e M(t' >$ implicam $M(tt' >$ (e, por simetria, $M(t't >)$) [2].

2.7.5 Reversibilidade

Quando existe o retorno à marcação inicial ou uma outra marcação qualquer diz-se que a Rede de Petri é reversível [7].

Uma rede de Petri (N, M_0) é reversível se, para toda marcação M de $R(M_0)$, M_0 for alcançável a partir de M .

Numa rede reversível, pode-se sempre voltar ao estado inicial. Em muitas aplicações, isto não é necessário.

Uma marcação M' é um estado de passagem se, para toda marcação M de $R(M_0)$, M' for alcançável a partir de M [2].

3 *A Ferramenta Opulus*

3.1 Requisitos

3.1.1 Requisitos funcionais

Para o planejamento dos requisitos funcionais foram levados em consideração principalmente: o curto período de tempo disponível para o desenvolvimento da ferramenta e sua aplicação como ferramenta de ensino. Desta forma os requisitos funcionais que nortearam o desenvolvimento da ferramenta foram:

- Redes de Petri com capacidade – O aplicativo deve permitir que o usuário defina capacidades máxima (2.5) aos lugares de uma Rede de Petri;
- Arcos inibidores – O aplicativo deve permitir que o usuário crie arcos inibidores (2.4);
- Arcos ponderados – O aplicativo deve permitir que o usuário defina pesos aos arcos de uma Redes de Petri;
- Transições temporizadas – O aplicativo deve suportar transições temporizadas como descrito na seção 2.3 e todas as suas conseqüências nos métodos de análise;
- Grafo de cobertura – O aplicativo deve ser capaz de gerar o grafo de cobertura de uma Rede de Petri utilizando o algoritmo descrito na seção 2.6.1;
- Transformação de rede com capacidade para rede sem capacidade: dada uma rede que possui lugares com capacidade o aplicativo deve ser capaz de gerar uma rede equivalente sem o uso de lugares com capacidade;
- Transformação de rede com arco inibidor para rede sem arco inibidor: dada uma rede com arcos inibidores o aplicativo deve ser capaz de gerar uma rede equivalente sem arcos inibidores;

- Verificar propriedades comportamentais da rede descritas na seção 2.7, tais como: alcançabilidade, limitação, vivacidade, reversibilidade e persistência;
- Importar redes do Pipe2 [9].

Os requisitos referentes a transformação de rede com capacidade para rede sem capacidade e rede com arco inibidor para rede sem arco inibidor não foram contemplados. O principal motivo para esta decisão foi a baixa prioridade desses requisitos.

3.1.2 Requisitos não funcionais

Alguns requisitos não funcionais foram decididos pelo cliente do projeto, outros foram levantados pela própria equipe de desenvolvimento e em seguida aprovados pelo cliente:

- Portabilidade entre as plataformas Linux™, BSD e variantes e Windows™;
- Suporte a internacionalização e localização;
- Métodos de análise implementados através de plugins;
- Distribuído sob uma licença livre;
- A versão para Windows™ deve possuir um instalador;

Todos os requisitos funcionais estão contemplados na versão mais recente do Opulus.

3.2 Arquitetura

O Opulus utiliza uma arquitetura de 2 camadas: interface do usuário e lógica de negócio.

No total existem 2 módulos no Opulus:

- OpulusCore – Biblioteca dinâmica (libopuluscore.so) que provê toda a lógica de negócio e desconhece qualquer coisa a respeito da interface com o usuário;
- OpulusGUI – Biblioteca dinâmica (libopulusgui.so) que provê uma interface gráfica com o usuário, tem como dependência o módulo OpulusCore;

Tanto o módulo OpulusCore quanto o módulo OpulusGUI podem ser estendidos através de plugins.

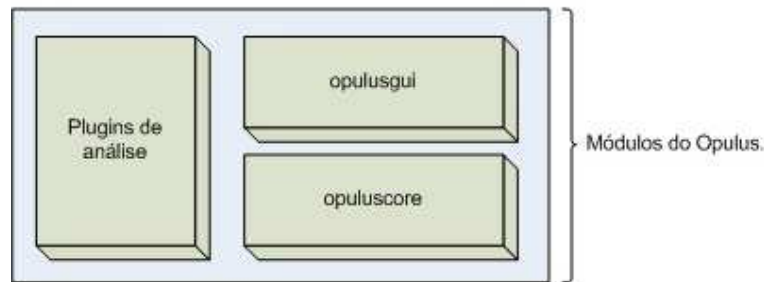


Figura 3.1: Arquitetura da ferramenta

3.3 Ferramentas Utilizadas

Todas as ferramentas utilizadas no desenvolvimento do Opulus são distribuídas gratuitamente na internet como software livre, com exceção do Inno Setup [10] que não é software livre mas é distribuído gratuitamente.

3.3.1 GCC e MinGW

Foram utilizados os compiladores C++ GCC para Linux e MinGW para Windows, porém o projeto foi escrito em ISO/C++ e deve ser compilável em qualquer compilador C++ moderno.

3.3.2 Qt4

Qt4 [11] é um framework multiplataforma feito pela Trolltech™ que inclui uma biblioteca de classes C++ para interface gráfica, internacionalização, XML, banco de dados, scripting, rede e multimídia. O Opulus faz uso intensivo desta biblioteca.

3.3.3 libxslt

Libxslt é uma biblioteca escrita em C utilizada para fazer transformações XSL [12], o projeto utiliza XSL para transformar arquivos XML no formato do Pipe2 para o formato utilizado no Opulus.

3.3.4 Boost Graphs Library

Boost Graphs Library [13] (BGL) é uma biblioteca para grafos usada pelo plugin de propriedades comportamentais (3.4.4) para calcular os componentes fortemente conectados do grafo de cobertura de uma Rede de Petri.

3.3.5 Graphviz

Graphviz [14] é uma coleção de programas para visualização de grafos utilizado pelo Oplus para a visualização do grafo de cobertura.

3.3.6 CMake

CMake [15] é uma ferramenta de meta-construção (*meta-building tool*) multiplataforma utilizado para gerar scripts para as principais ferramentas de construção de cada plataforma. No caso do Oplus, ele é utilizado para gerar Makefiles para o GCC no Linux™ e Windows™.

3.3.7 Doxygen

Doxygen [16] é utilizado para extrair a documentação da API (*Application Programming Interface*) a partir do código fonte do projeto.

3.4 Implementação

Para implementar o Oplus, foi escolhida a linguagem C++ [17] devido à fluência nesta linguagem por um dos membros da equipe e à vontade de aprender por parte do resto da equipe. Além disso, utilizar o framework Qt4, faz de C++ uma linguagem ideal para a construção de aplicativos que exigem uma rica interface gráfica que se comporte como aplicativos nativos em todos os sistemas operacionais suportados.

3.4.1 Simulação de Redes de Petri

O diagrama de classes do módulo OplusCore¹ é mostrado na figura 3.2.

Incluindo classes abstratas, o software tem apenas apenas 10 classes. A principal entidade é Petrinet, a qual contém todos os itens da Rede de Petri (classes derivadas de Item) e uma marcação (classe Marking).

A simulação é implementada usando o idioma C++ *Resource Acquisition Is Initialization* [17]. A classe Simulation guarda a marcação inicial da Rede de Petri no momento de sua inicialização, a modifica quantas vezes precisar no decorrer da simulação e depois retorna à marcação inicial da Rede de Petri no momento de sua destruição. Como o leitor já deve ter

¹Classes relativas ao carregamento de plugins não estão contempladas neste diagrama.

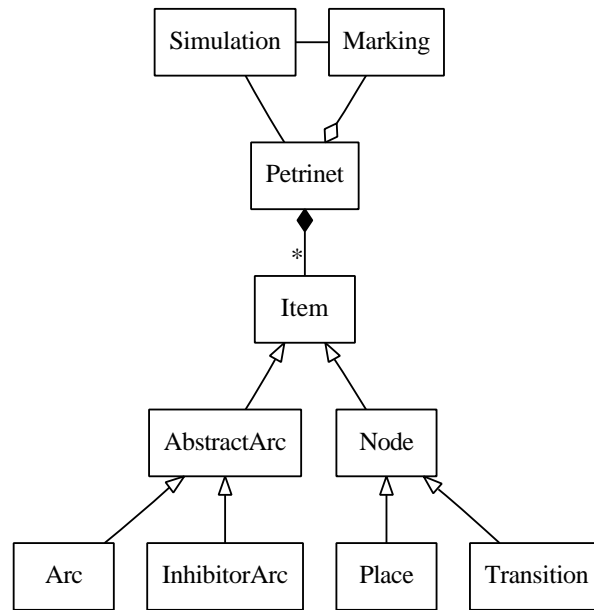


Figura 3.2: Diagrama de classes do módulo OpulusCore.

notado, as fichas são guardadas na entidade *Marking* ao invés da entidade *Place*. O que parece ser baixa coesão, característica típica de um design pobre, é na verdade uma estratégia para facilitar a implementação de plugins de análise que precisam saber a marcação da Rede de Petri em vários momentos da análise.

3.4.2 Infraestrutura para plugins

Plugins, no Opulus, são bibliotecas dinâmicas que contêm uma classe que implementa a interface *Analyser*. Tal interface foi concebida com o intuito de ser a mais simples e flexível possível como mostrado na listagem 3.1.

Listagem 3.1: core/analyser.h

```

1 class Analyser {
2 public:
3     /**
4     * Plugin internal name.
5     * @note This string CAN NOT be translated.
6     */
7     virtual QString internalName() const = 0;
8     /// The plugin name, can be translated.
9     virtual QString name() const = 0;
10    /// This method is called (in another thread) when an analysis is
        requested.
  
```

```

11     virtual void analyse(PetriNet* pn, AnalysisReporter* reporter) =
        0;
12     /// This method is called before the analysis start.
13     virtual bool setup(QWidget* parentWidget) = 0;
14     /// This method is called when an analysis is finished.
15     virtual void finish(QWidget* parentWidget) = 0;
16 };
17
18 Q_DECLARE_INTERFACE(Analyser, "opulus.sourceforge.net.Analyser/1.0")

```

Assim que o aplicativo inicia, o módulo OpulusGUI varre o diretório `INSTALL_PREFIX/share/opulus/plugins`² e pede para o módulo OpulusCore carregar todas as bibliotecas dinâmicas que ele encontrou. Para cada plugin carregado com sucesso, é adicionada uma entrada no menu “Análise” com o nome informado pelo próprio plugin.

Apesar dos plugins serem carregados pelo módulo OpulusCore, eles podem opcionalmente ter uma interface com o usuário, porém eles não tem acesso ao módulo OpulusGUI. Plugins podem, por exemplo, exibir uma janela de diálogo antes de serem executados para que o usuário possa configurar o plugin. É possível também exibir uma outra janela de diálogo ao final do processamento para mostrar os resultados de uma análise.

Para evitar que o processamento dos plugins obstruísse o processamento dos eventos oriundos da interface do usuário eles são executados em uma thread separada e reportam seu estado através do objeto *AnalysisReporter*.

3.4.3 Geração do Grafo de Cobertura

A implementação do grafo de cobertura usa o algoritmo descrito na seção 2.6.1 com uma pequena mudança para suportar redes com capacidade. O algoritmo modificado:

1. Rotule a marcação inicial M_0 de raiz e sinalize-a como nova.
2. Enquanto existirem marcações novas, faça:
 - (a) Escolha uma nova marcação M .
 - (b) Se M for idêntica a outra marcação no caminho da raiz até M , sinalize M com velho e vá para uma outra marcação.

²INSTALL_PREFIX é o diretório onde o Opulus foi instalado.

- (c) Se nenhuma transição estiver habilitada em M , sinalize M com fim.
- (d) Enquanto existirem transições habilitadas em M , para cada transição habilitada t em M faça o seguinte:
 - i. Obtenha a marcação M' que resulta do disparo de t em M .
 - ii. Se no caminho da raiz até M existir uma marcação M'' tal que $M'(p) \geq M''(p)$ para cada lugar p e $M' \neq M''$ e p **não possui um limite de capacidade**, então substitua $M'(p)$ por ω para cada p tal que $M'(p) > M''(p)$.
 - iii. Introduza M' como um nó, desenhe um arco com rótulo t , de M para M' , e rotule M' com novo.

Suponha a rede da figura 3.3. Se $P1$ não tiver capacidade definida a rede pode possuir infinitas marcações e seu grafo de cobertura (Figura 3.4) precisará da notação ω , caso contrário ele seria infinito.

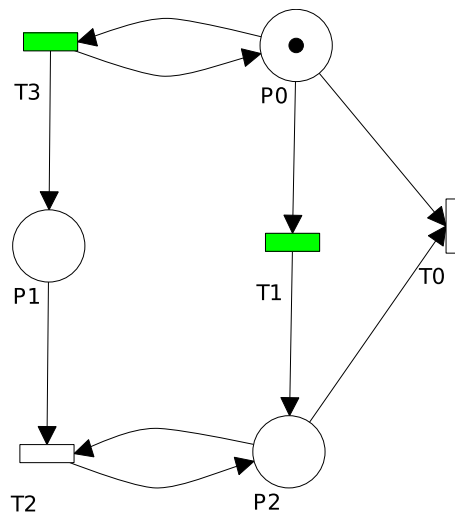


Figura 3.3: Rede de petri com infinitas marcações, transições em verde estão ativas

Agora suponha que $P1$ tenha capacidade igual à 2. Neste caso a rede não pode mais possuir infinitas marcações e o uso da notação ω iria apenas causar perda de informação sobre possíveis marcações da rede. Por isso na hora de tentar aplicar a notação ω a existência de capacidade de um lugar é levado em conta.

O plugin mostra os resultados em uma janela de diálogo, indicando as marcações mortas, a marcação inicial e opções de aumentar/reduzir zoom, como pode ser visto na figura 3.6.

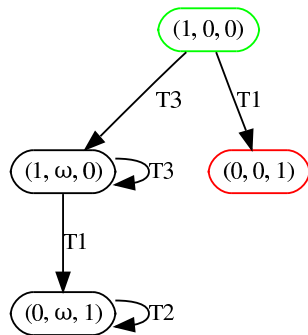


Figura 3.4: Grafo de cobertura da rede 3.3 sem capacidade.

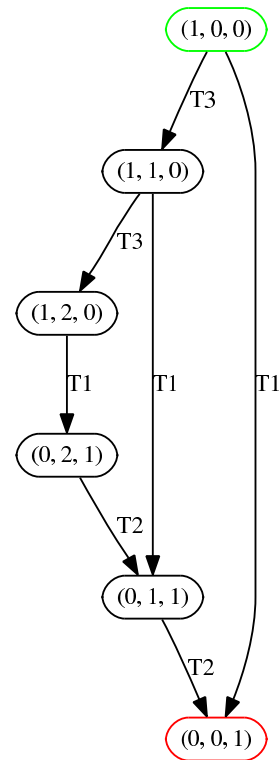


Figura 3.5: Grafo de cobertura da rede 3.3 contendo capacidade de 2 fichas no lugar $P1$.

3.4.4 Propriedades comportamentais

Este plugin verifica duas propriedades comportamentais da rede: limitação e reversibilidade.

Para saber se uma rede é limitada basta usar o mesmo algoritmo usado para gerar o grafo de cobertura. Se existir pelo menos uma marcação com ω , a rede é ilimitada, caso contrário ela é k -limitada, onde k é o maior número de fichas encontrado em um lugar de uma marcação.

O processo de saber se uma rede é reversível também passa pelo algoritmo usado para gerar o grafo de cobertura. Consiste apenas de verificar se o grafo de cobertura possui apenas um componente fortemente conectado. Se isso ocorrer significa que, a partir de qualquer marcação, eu posso retornar para a marcação inicial.

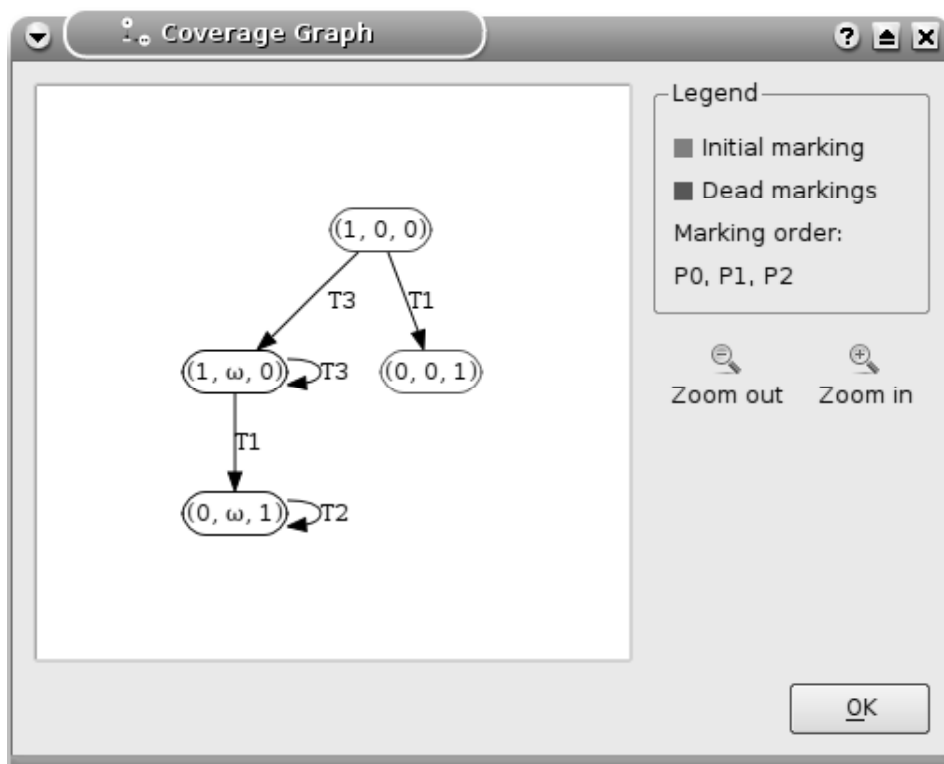


Figura 3.6: Plugin de geração de grafo de cobertura

ANEXO A – Processo de desenvolvimento

O desenvolvimento do projeto foi baseado no processo XP1 [18]. Inicialmente, será apresentado um panorama do processo XP1. Logo após, será apresentado o contexto do projeto. A seguir, serão definidos os elementos do processo (quem faz o quê, quando e como) no projeto desenvolvido.

A.1 Descrição do processo

O processo de desenvolvimento empregado nas disciplinas de Projeto em Computação I e Projeto em Computação II foi o XP1 [18], elaborado por um grupo de alunos e professores da UFCG. Esse processo é baseado nas práticas de Extreme Programming (XP) e, por conseguinte, bastante indicado quando os requisitos funcionais do software a ser desenvolvido não estiverem bem definidos, pois permite a realização de mudanças a um custo menor.

Dentre suas principais características:

- É baseado nas práticas de Extreme Programming (XP), mas com algumas mudanças e simplificações.
- Pretende abraçar as tarefas imprescindíveis: aquelas que precisam ser feitas em qualquer processo de desenvolvimento de software, por mais simples que seja. XP1 é um processo magro.
- Pretende ser tão simples que as chances de sucesso em aplicá-lo por alunos das disciplinas sejam muito altas. Portanto, vários aspectos que constam de um processo de software completo estão propositadamente ausentes (traceability, por exemplo).
- Foi especialmente elaborado para levar em consideração características especiais do ambiente universitário em que alunos não trabalham no projeto 40 horas por semana e nem sempre terão muitas horas compartilhadas para realizar o trabalho.

O XP1 também define papéis durante o processo e suas respectivas responsabilidades:

- Cliente – responsável por descrever a funcionalidade desejada, estando disponível para pequenas interrupções a qualquer momento para refinar as Stories e tirar dúvidas que surgirem durante o desenvolvimento; descrever os requisitos não-funcionais do software; definir o plano de release de software; descrever os testes de aceitação para cada User Story e por escolher User Stories para cada iteração.
- Desenvolvedor – responsável por ajudar o cliente a estimar as user stories e requisitos não-funcionais do processo; elaborar um projeto arquitetural e estimar o tempo aproximado de desenvolvimento de user stories; dividir user stories em tarefas e estimar precisamente o tempo de desenvolvimento; elaborar o esquema lógico dos dados; escrever o código das tarefas baseado nos testes de aceitação automáticos recebidos; executar atividades de integração de tarefas realizadas à base central de código do projeto; conversar com o cliente antes do desenvolvimento da user story para detalhar o que deverá ser implementado.
- Gerente – responsável por conduzir as atividades de planejamento; avaliar riscos descobertos e lidar com estes; manter o progresso do projeto; auxiliar na coleta de informações auxiliares.

O XP1 ainda define e descreve as diversas atividades a serem desenvolvidas durante o processo, tais como: análise, planejamento, projeto (design), testes, documentação, integração e gerência.

ANEXO B – The GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation’s software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
 - (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
 - (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
 - (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

- (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
- (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indi-

cate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.
7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

Referências Bibliográficas

- [1] PETRI, C. A. *Kommunikation mit Automaten*. Tese (Doutorado) — Institut für instrumentelle Mathematik, Bonn, 1962.
- [2] BRAGA, J. D. M.; PERKUSICH, A.; FIGUEIREDO, J. C. A. de. *Redes de Petri*. 1999.
- [3] LINO, F. G. de O.; SZTAJNBERG, A. Analisador e simulador de redes de petri. *Anais do XXVII Congresso da SBC*, 2007.
- [4] FRANCÊS, C. R. L. *Introdução às Redes de Petri*. 2003.
- [5] GARBI, G. P.; GRANDINETTI, F. J. *Application of Petri Net to identification and transport system for three different products*. 2006.
- [6] MARRANGHELLO, N. *Redes de Petri: Conceitos e Aplicações*. 2005.
- [7] PENHA, D. O. et al. Modelagem de sistemas computacionais usando redes de petri: aplicação em projeto, análise e avaliação. *Anais da IV Escola Regional de Informática RJ/ES.*, 2004.
- [8] MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541–580, 1989. Disponível em: <<http://dx.doi.org/10.1109/5.24143>>.
- [9] BLOOM, J. et al. *Platform Independent Petri net Editor 2*. Disponível em: <<http://pipe2.sourceforge.net/>>. Acesso em: 23 jul. 2008.
- [10] INNO Setup. Disponível em: <<http://jrsoftware.org/isinfo.php>>. Acesso em: 23 jul. 2008.
- [11] QT Cross-Platform Application Framework. Disponível em: <<http://trolltech.com/products/qt/>>. Acesso em: 23 jul. 2008.
- [12] W3C. *XSL Transformations (XSLT)*. 1997. Disponível em: <<http://www.w3.org/TR/xslt>>. Acesso em: 23 jul. 2008.
- [13] BOOST C++ Libraries - The Boost Graph Library. Disponível em: <http://www.boost.org/doc/libs/1_35_0/libs/graph/doc/index.html>. Acesso em: 23 jul. 2008.
- [14] GRAPHVIZ. Disponível em: <<http://www.graphviz.org/>>. Acesso em: 23 jul. 2008.
- [15] CMAKE Cross Platform Make. Disponível em: <<http://www.cmake.org/>>. Acesso em: 23 jul. 2008.
- [16] DOXYGEN. Disponível em: <<http://www.doxygen.org/>>. Acesso em: 23 jul. 2008.

- [17] STROUSTRUP, B. *The C++ Programming Language, Third Edition*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997. ISBN 0201889544.
- [18] SAUVÉ, J. P. et al. *XPI: Um Processo de Desenvolvimento*. Disponível em: <<http://jacques.dsc.ufcg.edu.br/projetos/common/xp1/xp1.html>>. Acesso em: 23 jul. 2008.