FERRAMENTAS COMPUTACIONAIS PARA A ENGENHARIA: UML E C++ EM UMA APLICAÇÃO COM REDES DE PETRI

Lincoln C. Zamboni [1] ; Sergio V. D. Pamboukian [2] ; Edson A. R. Barros [3]; Osvaldo Ramos Tsan Hu [4]; Melanie Lerner Grinkraut [5]

Universidade Presbiteriana Mackenzie, Escola de Engenharia
Departamentos de Engenharia Elétrica, Engenharia de Produção e Tecnologia Elétrica
Rua da Consolação, 930 Prédios 4 (Sala 204) e 6
01302-907 – São Paulo – SP
[1] lincoln.zamboni@mackenzie.br
[2] sergiop@mackenzie.br
[3] prof_edson@mackenzie.br
[4] oshu@yahoo.com
[5] mlgrinkraut@mackenzie.br

Abstract: *Many resources are free of charge available through the Internet in a lot of different sites. There are offered, for instance, tools for the development of softwares that embrace all theirs life cycle, from the requirements analysis to the tests and maintenance. In the development of some applications for engineering such offer is not insignificant. Languages such as the already established UML and C++ are available for the development of scientific applications in a fast way, with attractive graphic interfaces and easy manipulation. The student must take into account that such languages shall be present in the conception of the systems, since his first steps in the Engineering course, independently of the future professional attributions that may come to get. In this work it is developed, using UML and C++, the project of an editor of Petri Nets, as well as tools for their analysis, supplying didactic guidelines for the application of object oriented methods in classroom.*

Key-words: *Computer Science, Programming, UML, C++, Petri Nets.*

Resumo: *Muitos recursos são disponibilizados gratuitamente por meio da Internet nos seus mais diversos sítios. Oferecem-se, por exemplo, ferramentas para o desenvolvimento de softwares que abrangem todo o seu ciclo de vida, desde a análise de requisitos até os testes e manutenção. No desenvolvimento de aplicações para a engenharia tal oferta não é insignificante. Linguagens como as já consagradas UML e C++ estão disponíveis para o desenvolvimento de aplicações científicas de forma rápida, com interfaces gráficas atraentes e de fácil manipulação. Tais linguagens devem estar presentes na concepção dos sistemas por parte do estudante desde seus primeiros passos no curso de engenharia, independente das futuras atribuições profissionais que o mesmo venha a ter. Neste trabalho é desenvolvido, com o uso de UML e C++, o projeto de um editor de Redes de Petri bem como ferramentas para sua análise, fornecendo diretrizes didáticas para aplicação dos métodos orientados a objetos em sala de aula.*

Palavras-chave: *Computação, Programação, UML, C++, Redes de Petri*

1. INTRODUÇÃO

A mentalidade científica é formada nas primeiras etapas dos cursos de engenharia, nas quais os alunos recebem formação sólida e completa de disciplinas tais como Computação, Desenho, Matemática e Física. Algumas considerações são feitas nos próximos parágrafos de forma a fortalecer a inserção de disciplinas ligadas à produção de software nos semestres iniciais de todos os cursos de engenharia.

O Conselho Nacional de Educação, pelas Diretrizes Curriculares Nacionais dos Cursos de Graduação de Engenharia, resolução CNE/CES 11/2002, salienta que:

- a composição do núcleo de conteúdos básicos deve ter cerca de 30% da carga horária mínima e versar sobre os tópicos Informática, Expressão Gráfica, Matemática, Física etc.;
- a composição do núcleo de conteúdos profissionalizantes deve ter cerca de 15% da carga horária mínima e versar sobre os tópicos Algoritmos e Estruturas de Dados, Compiladores, Métodos Numéricos etc.

Diretrizes estratégicas são estabelecidas para o planejamento do ensino da engenharia em um contexto mais local e, juntamente com as Diretrizes Curriculares Nacionais dos Cursos de Graduação em Engenharia, norteiam os rumos futuros. Dentre tais diretrizes estratégicas observa-se:

- a manutenção de recursos didáticos e tecnológicos que garantam o ensino e a aprendizagem;
- a ampliação da oferta de cursos e programas, utilizando-se de métodos científicos;
- a implantação de programas de educação e treinamento a distância;
- o aprimoramento da iniciação científica na graduação.

A Internet disponibiliza excelentes recursos nos seus mais diversos sítios. Estes recursos envolvem manuais técnicos, boletins técnicos, tutoriais e ferramentas profissionais. As ferramentas são fornecidas, dentre outras, em versões licenciadas gratuitas, versões licenciadas para testes e versões acadêmicas. Estas versões não trazem ônus algum ao estudante de engenharia. A informática é privilegiada com tais ferramentas e, em especial, a informática aplicada à engenharia. Citam-se alguns exemplos:

- *NUMERICAL RECIPES*, bibliotecas para análise numérica e algorítmica nas linguagens *C*/*C++*, *Pascal* e *FORTRAN* (77 e 90). Possui publicações em livros especializados da própria *Numerical Recipes* e estes livros contém as implementações dos algoritmos. Oferece extensão para o *MATLAB*. É encontrada em *http://www.nr.com*;
- *MATFOR*, um conjunto de bibliotecas desenvolvidas especialmente para uso de cientistas e engenheiros de forma a incrementar os softwares com visualizações dinâmicas, processamentos numéricos e processamentos estatísticos. É oferecida para as linguagens *C++*, *FORTRAN*, *VB* e *C#*. Oferece extensão para o *MATLAB*. Está disponível em *http://www.ancad.com*;
- *Turbo C++ Explorer* e *Turbo Delphi Explorer*, versões gratuitas de ambientes de desenvolvimento integrado para as linguagens *C*/*C++* e *Object Pascal*, respectivamente. Estão disponíveis em *http://www.codegear.com*;
- *Visual C++ 2005 Express Edition*, versão gratuita de ambiente de desenvolvimento integrado para a linguagem C/C++. Está disponível em *http://www.microsoft.com*.
- *ENTERPRISE ARCHITECT*, uma ferramenta para modelagem de softwares com o uso dos diagramas da *UML*. Disponível em *http://www.sparxsystems.com*;
- *Eclipse*, uma plataforma de desenvolvimento aberta cujos projetos são focados na estrutura extensível e em ferramentas (*UML*, C++ etc.) para o gerenciamento do ciclo de vida do software. A plataforma é apoiada por universidades, instituições

de pesquisa e diversos desenvolvedores que a estendem e a complementam. Está disponível gratuitamente em *http://www.eclipse.org*.

- *Dev-Cpp*, versão gratuita de ambiente de desenvolvimento integrado para a linguagem *C*/*C++*. O ambiente e pacotes estão disponíveis em *http://www.bloodshed.net*, sendo que muitos outros estão disponíveis em *http://devpaks.org*;
- *MATLAB*, uma linguagem de alto nível em ambiente interativo que permite executar tarefas computacionais intensivas. Está disponível em *http://www.mathworks.com*;
- *Mathematica*, é um sistema integrado que permite um fluxo de trabalho sem precedentes e com coerência, confiança e inovação. Em lugar de requerer *toolkits* diferentes para trabalhos diferentes, o *Mathematica* foi concebido com a visão de ser o ambiente de computação técnico completo. Possui ferramentas de conexão que permitem aos usuários do *MATLAB* fazer chamadas ao *Mathematica* diretamente e vice-versa. Está disponível em *http://www.wolfram.com*.

Existem outras excelentes ferramentas para o desenvolvimento de softwares para a engenharia. Aqui se citaram apenas algumas que podem ser utilizadas pelo estudante de engenharia sem ônus.

O presente trabalho enfoca:

- a introdução às Redes de Petri e o cálculo de invariantes através de algoritmos já conhecidos da Álgebra Linear, integrando tais conhecimentos;
- o desenvolvimento, sem ônus algum, de um software para a edição das Redes de Petri e cálculo de invariantes;
- a orientação para os cursos do núcleo de conteúdos básicos, em especial a informática aplicada à engenharia, no desenvolvimento de software.

As seguintes ferramentas foram utilizadas:

- Borland C++ Builder, um ambiente integrado de desenvolvimento para C++. Foi encontrado em *http://www.borland.com*;
- *LMD Tools*, um conjunto de componentes *VCL* e *.NET* para ambientes de desenvolvimento integrado como o *Borland Delphi* e o *Borland C++Builder*. Foi encontrado em *http://www.lmdtools.com*;
- *AddFlow*, um componente para desenho de diagramas de propósitos gerais. Possui versões *ActiveX* e *.NET*. Foi encontrado em *http://www.lassalle.com*.

2. UML E C++

A padronização é a coluna vertebral para as aplicações mundialmente bem sucedidas. Ela traz inúmeras vantagens e muita comodidade e segurança para os usuários, além da economia de recursos. Isto não é diferente para a *UML* (*Unified Modeling Language*) e *C++*.

Segundo OMG (2008), responsável pelos padrões da *UML*, esta é uma linguagem gráfica para a visualização, especificação, construção e documentação dos artefatos de um sistema fortemente baseado em software. Oferece um modo padrão para descrever sistemas, incluindo tanto as partes abstratas (conceituais e funcionais) como as partes concretas (comandos em linguagem de programação, por exemplo).

A *UML* é empregada como sinônimo de métodos orientados a objetos. Tal emprego é pertinente, mas não preciso, pois a *UML* é uma linguagem e não um conjunto de métodos. Sua especificação, juntamente com hiperlinks para diversas ferramentas gratuitas, pode ser encontrada gratuitamente em *http://www.omg.org*.

LEE e TEPFENHART (2001) reforçam que a grande quantidade de diferentes notações na modelagem de sistemas impediu o progresso da mesma. A ampla aceitação da *UML* e de

seus padrões eliminou a maior parte deste impedimento e o sucesso da *UML* está ligado, principalmente, pela disponibilidade de todos os ícones do ciclo de desenvolvimento de software, necessários para capturar conceitos e/ou mecanismos valiosos para a resolução dos problemas reais. A *UML* é uma linguagem em que seu próprio ciclo de utilização é dinâmico e extensível, pois oferece a capacidade de estender a notação para conceitos e/ou mecanismos ainda não definidos.

STROUSTRUP (1991) descreve *C++* como uma linguagem de programação multiparadigmas, pois suporta a programação procedural (*Procedural Programming*), a modular (*Modular Programming*), a abstração de dados (*Data Abstraction)* e a orientada a objetos (*Object-Oriented Programming*). ZAMBONI, PAMBOUKIAN e BARROS (2008) escrevem que em 1997, o *ANSI* (*American National Standards Institute*) instituiu os padrões para a linguagem *C++* e que a *ISO* (*International Organization for Standardiztion*) gerou recentemente a norma ISO/IEC 14882:2003 que especifica exigências para implementações da linguagem C++ e da sua biblioteca padrão. A linguagem *C++* traz as seguintes vantagens:

- orientação a objeto: permite ao programador projetar aplicações sob a ótica do encapsulamento, da hereditariedade e do polimorfismo, possibilitando a reutilização de uma maneira mais lógica e produtiva;
- portabilidade: pode-se praticamente compilar um código *C++* em quase todos os tipos de computadores e sistemas operacionais sem fazer mudanças profundas no mesmo;
- sintaxe compacta: o código escrito em *C++* é mais curto do que o escrito em outras linguagens e traz ao programador experiente a elegância e pureza encontrada nas notações matemáticas;
- velocidade: o código gerado por um compilador *C++* é muito eficiente e adequado no que tange a dualidade da *C++* como uma linguagem de alto nível e como uma linguagem de baixo nível;
- compiladores e ambientes gratuitos: é o caso do *Borland C++ Compiler*, do *Bloodshed Dev-C++*, do *GCC* e *G++*, do *Digital Mars C++*, do *DJGPP*, do *Mingw32*, do *Intel C++ Compiler* etc. Esses e diversos outros compiladores e ambientes integrados podem ser baixados da Internet;
- bibliotecas genéricas: é o caso da biblioteca padrão *STL* (*Standard Template Library*), uma coleção de contêineres, algoritmos e iteradores muitíssimo utilizada em computação.

3. REDES DE PETRI

O marco inicial do desenvolvimento da teoria das Redes de Petri foi a dissertação de doutorado de Carl Adam Petri em 1962 pela Universidade de Bonn, na Alemanha. Petri construiu uma rede completamente assíncrona de células idênticas as quais passavam informação para as suas vizinhas conforme necessário. Esta dissertação tem hoje apenas um interesse histórico. Um grupo de pesquisadores do *MIT* (*Massachussetts Institute of Technology*), impulsionado por Anatol W. Holt, formou as bases das Redes de Petri entre 1968 e 1976.

Uma Rede de Petri Lugar-Transição (*PTN - Place Transition Net*) ou, mais simplesmente, uma Rede de Petri, é uma quádrupla (P, T, IN, OUT), onde:

- P (*Places*) é o conjunto finito dos lugares, $P = \{p_1, p_2, ..., p_n\}$;
- T (*Transitions*) é o conjunto finito das transições, $T = \{t_1, t_2, ..., t_m\}$;
- $P \cup T \neq \phi$ e $P \cap T = \phi$;
- $IN : P \times T \rightarrow N$ é a função de entrada (*INput*);

- $OUT : P \times T \rightarrow N$ é a função de saída (*OUTput*);
- N é o conjunto dos números naturais, $N = \{0,1,2,...\}$.

Uma Rede de Petri Marcada é uma dupla (R, M), onde:

- R é uma Rede de Petri;
- $M : P \rightarrow N$ é a função de marcação.

A "Figura 1" ilustra uma Rede de Petri Marcada, retirada de LIU e HOROWITZ (1989). A "Figura 2" mostra suas respectivas funções *IN*, *OUT* e *M* em forma matricial, isto é,

- $IN : \{1,2,...,n\} \times \{1,2,...m\} \rightarrow N$, onde $IN(i, j) = IN(p_i, t_j)$;
- $OUT : \{1,2,...,n\} \times \{1,2,...m\} \rightarrow N$, onde $OUT(i, j) = OUT(p_i, t_j)$;
- $M : \{1,2,...,n\} \times \{1\} \rightarrow N$, onde $M(i,1) = M(p_i)$.

Observa-se nas matrizes anteriores que foram mantidos os mesmos nomes das funções correspondentes. Tal sobrecarga de nomes é muito útil desde que não gere ambigüidades.

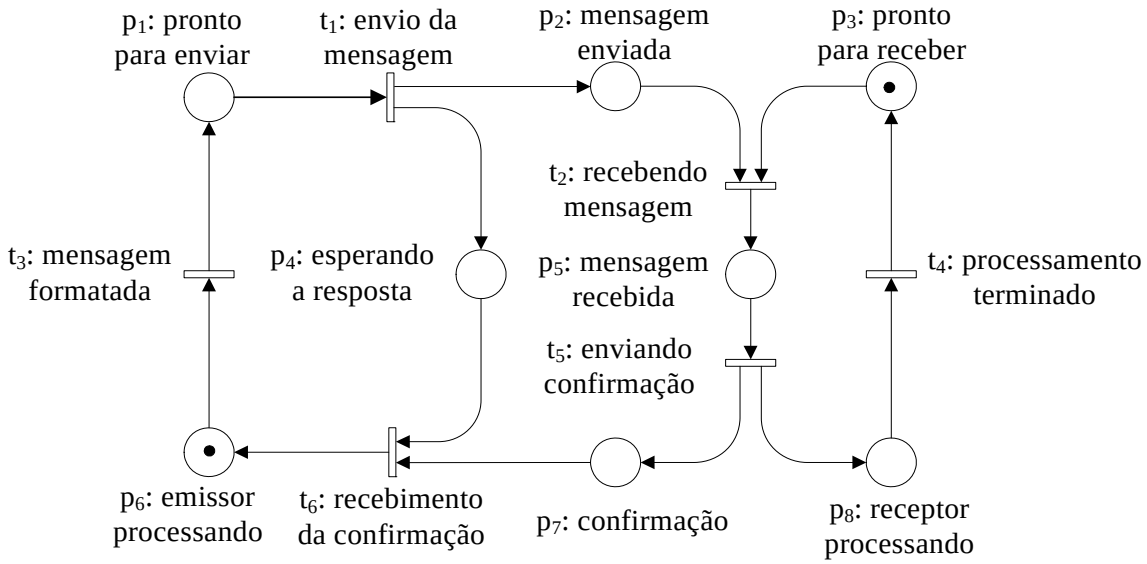

Figura 1 - Rede de Petri de um sistema emissor-receptor.

$$IN = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array} \quad OUT = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \quad M = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{array} \begin{array}{c} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \end{array}$$

Figura 2 – Matrizes *IN*, *OUT* e *M* do sistema emissor-receptor.

Uma transição t_j está armada se, e somente se, $M(p_i) \geq IN(p_i, t_j)$, $\forall p_i \in P$; caso contrário ela está desarmada. Exemplos de transições armada e desarmada são dados na "Figura 3".

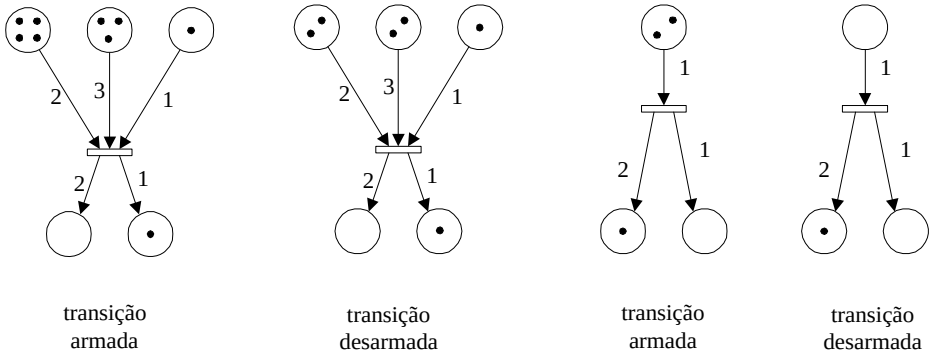

Figura 3 – Transições armadas de desarmadas.

A Equação Fundamental para as Redes de Petri é conseguida da forma que se segue.

Seja (*R*,*M*) uma Rede de Petri Marcada e t_j uma transição armada. O disparo desta transição resulta em uma nova marcação $NM(p_i)$ de forma que $NM(p_i) = M(p_i) + OUT(p_i,t_j) - IN(p_i,t_j),\ \forall p_i \in P$. Esta é a regra de disparo das transições armadas para as Redes de Petri Marcadas. A "Figura 4" ilustra a regra de disparo.

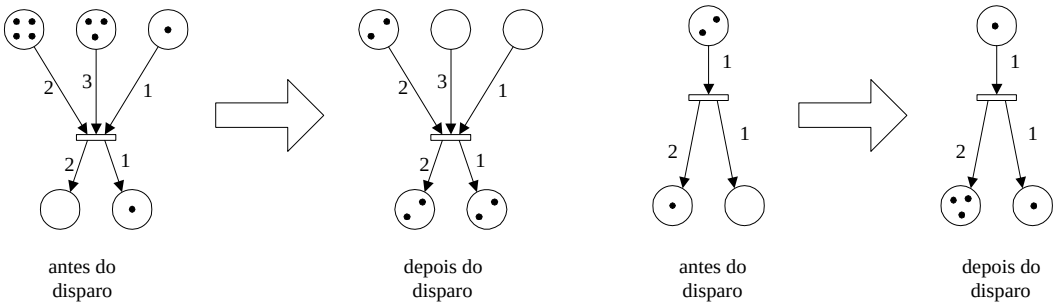

Figura 4 – Disparo de uma transição armada.

Utilizando a função de disparo *s* (*shot*) da transição t_j, definida por $s: T \to N$, onde $s(t_k) = \begin{cases} 1, \text{se } k = j \\ 0, \text{se } k \neq j \end{cases}$, pode-se escrever, em notação matricial, a seguinte equação: $NM = M + (OUT - IN) \cdot s$ ou, mais simplesmente, $NM = M + C \cdot s$.

Utilizando uma seqüência de matrizes de disparo de transições armadas $s^1, s^2, \ldots, s^p$ e a equação matricial $NM = M + C \cdot s$, encontramos: $M^1 = M + C \cdot s^1$, $M^2 = M^1 + C \cdot s^2, \ldots$, $M^p = M^{p-1} + C \cdot s^p$. Substituindo a primeira equação na segunda equação e o resultado desta substituição na terceira e o resultado desta substituição na quarta e assim sucessivamente, chegamos à seguinte equação: $M^p = M + C \cdot (s^1 + s^2 + \ldots + s^p)$. Chamando $M^p = M'$ e $\bar{s} = s^1 + s^2 + \ldots + s^p$, encontramos finalmente: $M' = M + C \cdot \bar{s}$, onde $M \geq 0$ e $\bar{s} \geq 0$.

4. INVARIANTES EM REDES DE PETRI E SUA OBTENÇÃO

Segundo VALETTE (1992):

- *f* é um componente conservativo de uma Rede de Petri se, e somente se, é uma solução de $f^T \cdot C = 0$. Um invariante de lugar é a função linear $f^T \cdot M' = f^T \cdot M$, onde *f* é um componente conservativo;

- $\bar{s}$ é um componente repetitivo de uma Rede de Petri se, e somente se, for uma solução de $C \cdot \bar{s} = 0$. Um invariante de transição é um componente repetitivo para o qual existe uma seqüência de disparos de transições armáveis (as transições estarão armadas para o disparo) associada a este.

Observa-se que:

- um componente conservativo não depende da marcação inicial *M*. Já um invariante de lugar é dependente da marcação inicial;
- um componente repetitivo não depende da marcação inicial *M*. Já um invariante de transição é dependente da marcação inicial.

Excelentes livros descrevem algoritmos de como se resolver o problema de cálculo de uma base para o espaço vetorial das soluções de um sistema linear homogêneo. Dois são clássicos: STRANG (1988) e NOBLE e DANIEL (1986). Aqui segue-se o relatado em VALETTE (1992).

Considera-se inicialmente o sistema linear homogêneo $f^T C = 0$. Utilizando as matrizes elementares escreve-se $C' = ECD$, onde E é o produto de todas as matrizes elementares que multiplicam a matriz C pela esquerda (esta matriz produz operações entre linhas na matriz C); D é o produto de todas as matrizes elementares que multiplicam a matriz C pela direita (esta matriz produz operações entre colunas na matriz C); e $C' = \left[\begin{array}{c|c} S & S' \\ \hline zeros & zeros \end{array}\right]$, onde $S = \begin{bmatrix} S_{11} \neq 0 & S_{12} & S_{13} & \dots & S_{1r} \\ 0 & S_{22} \neq 0 & S_{23} & \dots & S_{2r} \\ 0 & 0 & S_{33} \neq 0 & \dots & S_{3r} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & S_{rr} \neq 0 \end{bmatrix}$ e S' é uma matriz sem nenhum aspecto especial.

A matriz C' é obtida da seguinte forma:

Parte-se da matriz $C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & \dots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \dots & C_{2m} \\ C_{31} & C_{32} & C_{33} & \dots & C_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \dots & C_{nm} \end{bmatrix} \begin{matrix} \ell_1 \\ \ell_2 \\ \ell_3 \\ \vdots \\ \ell_n \end{matrix}$. Nesta matriz *C*, executa-se, para *i = 1, 2, 3,... , n-1*, as seguintes combinações lineares entre linhas (matrizes elementares multiplicadas pela esquerda da matriz *C*):

$$\begin{cases} \ell_1 \leftarrow \ell_1 \\ \ell_2 \leftarrow \ell_2 \\ \quad \vdots \\ \ell_i \leftarrow \ell_i \\ \ell_{i+1} \leftarrow C_{ii}\ell_{i+1} - C_{i+1,i}\ell_i \\ \ell_{i+2} \leftarrow C_{ii}\ell_{i+2} - C_{i+2,i}\ell_i \\ \quad \vdots \\ \ell_n \leftarrow C_{ii}\ell_n - C_{ni}\ell_i \end{cases}$$

Estas combinações servirão para, a partir do elemento pivô $C_{ii} \neq 0$ da linha *i*, zerarmos os elementos da coluna também *i* que estão na banda inferior, isto é, $C_{i+1,i}, C_{i+2,i}, \dots, C_{n,i}$.

Caso o pivô seja nulo ($C_{ii} = 0$), ou se permuta a linha *i* com uma das linhas *i+1, i+2,... , n*, de tal forma a obter um novo pivô não nulo (a permutação entre as linhas é uma multiplicação de uma matriz elementar pela esquerda), ou se permuta a coluna *i* com uma das colunas *i+1, i+2,... , m*, para a mesma finalidade (a permutação entre as colunas é uma multiplicação de uma matriz elementar pela direita).

O processo de obtenção da matriz C' estará terminado quando não se tem mais nenhum candidato a pivô.

Deve-se observar que esta não é a única forma de, a partir da matriz *C*, obter-se a matriz C'.

Como as matrizes *E* e *D*, onde $C' = ECD$, possuem inversa, pode-se escrever $C = E^{-1}C'D^{-1}$. Assim,

$f^T C = 0 \Rightarrow$ (considerando $C = E^{-1}C'D^{-1}$)

$f^T(E^{-1}C'D^{-1}) = 0 \Rightarrow$ (usando a propriedade associativa)

$(f^T E^{-1})C'D^{-1} = 0 \Rightarrow$ (usando a propriedade da transposta do produto)

$(E^{-1^T} f)^T C'D^{-1} = 0 \Rightarrow$ (chamando $f' = E^{-1^T} f$)

$f'^T C'D^{-1} = 0 \Rightarrow$ (multiplicando ambos os membros por *D*)

$(f'^T C'D^{-1})D = 0D \Rightarrow$ (usando a propriedade associativa e da matriz nula)

$f'^T C'(D^{-1}D) = 0 \Rightarrow$ (usando as propriedades da matriz inversa)

$f'^T C' = 0 \Rightarrow$ (como $f' = E^{-1^T} f \Leftrightarrow f'^T = (E^{-1^T} f)^T \Leftrightarrow f'^T = f^T E^{-1}$)

$(f^T E^{-1})C' = 0 \Rightarrow$ (como $C' = ECD$)

$(f^T E^{-1})(ECD) = 0 \Rightarrow$ (usando a propriedade associativa)

$f^T(E^{-1}E)CD = 0 \Rightarrow$ (usando as propriedades da matriz inversa e da identidade)

$f^T CD = 0 \Rightarrow$ (multiplicando ambos os membros por D^{-1})

$(f^T CD)D^{-1} = 0D^{-1} \Rightarrow$ (usando as propriedades associativa, matriz inversa e matriz nula)

$f^T C = 0$.

Logo, conclui-se que $f^T C = 0 \Leftrightarrow f'^T C' = 0$, onde: $C' = ECD$ e $f = E^T f'$ (pois $f' = E^{-1^T} f \Leftrightarrow f'^T = (E^{-1^T} f)^T \Leftrightarrow f'^T = f^T E^{-1} \Leftrightarrow f'^T E = f^T \Leftrightarrow f = E^T f'$), significando que os sistemas $f^T C = 0$ e $f'^T C' = 0$ têm as mesmas soluções e que a relação entre *f* e f' é afetada apenas pelas operações entre linhas da matriz *C* (pois *E* é o produto de todas as matrizes elementares que multiplicam a matriz *C* pela esquerda).

O sistema $f'^T C' = 0$, reduz-se a
$$\begin{cases} S_{11} f_1' = 0 \\ S_{12} f_1' + S_{22} f_2' = 0 \\ S_{13} f_1' + S_{23} f_2' + S_{32} f_3' = 0 \\ \vdots \\ S_{1r} f_1' + S_{2r} f_2' + S_{3r} f_3' + \ldots + S_{rr} f_r' = 0 \\ S_{1,r+1} f_1' + S_{2,r+1} f_2' + S_{3,r+1} f_3' + \ldots + S_{r,r+1} f_r' = 0 \\ S_{1,r+2} f_1' + S_{2,r+2} f_2' + S_{3,r+2} f_3' + \ldots + S_{r,r+2} f_r' = 0 \\ \vdots \\ S_{1n} f_1' + S_{2n} f_2' + S_{3n} f_3' + \ldots + S_{rn} f_r' = 0 \end{cases}.$$

Assim, percebe-se que as soluções do sistema $f'^T C' = 0$ são:

$$f^{'} = f^{'}_{r+1}\underbrace{\begin{bmatrix} \underline{zeros} \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{v_{r+1}} + f^{'}_{r+2}\underbrace{\begin{bmatrix} \underline{zeros} \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{v_{r+2}} + f^{'}_{r+3}\underbrace{\begin{bmatrix} \underline{zeros} \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}}_{v_{r+3}} + \ldots + f^{'}_{n}\underbrace{\begin{bmatrix} \underline{zeros} \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}}_{v_{n}}.$$

Verifica-se também que as matrizes (vetores) $v_{r+1}, v_{r+2}, v_{r+3}, \ldots, v_n$ geram o espaço vetorial das soluções de $f'^T C' = 0$ e que são linearmente independentes, melhor dizendo, formam uma base para tal espaço (base canônica).

Por outro lado, considerando-se

$\theta_{r+1}E^T v_{r+1} + \theta_{r+2}E^T v_{r+2} + \theta_{r+3}E^T v_{r+3} + \ldots + \theta_n E^T v_n = 0 \Rightarrow$

$E^T(\theta_{r+1}v_{r+1} + \theta_{r+2}v_{r+2} + \theta_{r+3}v_{r+3} + \ldots + \theta_n v_n) = 0 \Rightarrow$ (multiplicando por $E^{T^{-1}}$)

$\theta_{r+1}v_{r+1} + \theta_{r+2}v_{r+2} + \theta_{r+3}v_{r+3} + \ldots + \theta_n v_n = 0 \Rightarrow$ (os vetores formam uma base)

$\theta_{r+1} = \theta_{r+2} = \theta_{r+3} = \ldots = \theta_n = 0$, logo os vetores $E^T v_{r+1}, E^T v_{r+2}, E^T v_{r+3}, \ldots, E^T v_n$ são linearmente independentes.

Como $f = E^T f' = E^T(f^{'}_{r+1}v_{r+1} + f^{'}_{r+2}v_{r+2} + f^{'}_{r+3}v_{r+3} + \ldots + f^{'}_{n}v_n) =$

$f^{'}_{r+1}E^T v_{r+1} + f^{'}_{r+2}E^T v_{r+2} + f^{'}_{r+3}E^T v_{r+3} + \ldots + f^{'}_{n}E^T v_n$, logo $E^T v_{r+1}, E^T v_{r+2}, E^T v_{r+3}, \ldots, E^T v_n$ geram o espaço vetorial das soluções de $f^T C = 0$.

Assim, conclui-se que os vetores $E^T v_{r+1}, E^T v_{r+2}, E^T v_{r+3}, \ldots, E^T v_n$ formam uma base para o espaço vetorial das soluções de $f^T C = 0$.

Segundo VALETTE (1992), as soluções positivas (base positiva) são, em geral, as mais interessantes. Este interesse é evidente para os invariantes de transição. É então útil favorecer as combinações de linhas positivas a partir da eliminação.

O algoritmo simplificado, implementado no presente trabalho, para a busca de soluções, sempre que possível positivas, é composto de 5 etapas:

1. INICIALIZAÇÃO DA MATRIZ PARTICIONADA: iniciamos com a matriz particionada (ou aumentada) $C \mid i_n$, onde C é a primeira partição e i_n é a segunda, para ser usada nas etapas sucessivas.

2. REMOÇÃO ENQUANTO EXISTE UMA COLUNA COM UM ÚNICO ELEMENTO NÃO NULO: enquanto encontrarmos, na primeira partição, uma coluna com um único elemento não nulo, removemos esta coluna juntamente com a linha da matriz particionada que contém este elemento, obtendo uma nova matriz particionada para ser usada nas etapas sucessivas. Esta remoção é válida, pois deveríamos zerar a coluna do pivô.

3. COMBINAÇÃO ENTRE LINHAS QUANDO EXISTE UMA COLUNA COM UM ÚNICO ELEMENTO MAIOR QUE ZERO OU UM ÚNICO MENOR QUE ZERO: se encontrarmos na primeira partição uma coluna que possua um único elemento maior que zero (todos os outros são menores ou iguais a zero) ou um único elemento menor que zero (todos os outros são maiores ou iguais a zero), então efetuamos, na matriz particionada, combinações lineares entre a linha que contém este único elemento e as outras, de forma a anularmos os outros elementos da coluna,

obtendo uma nova matriz particionada para ser usada nas etapas sucessivas. Retornamos para 2.

4. COMBINAÇÃO ENTRE LINHAS QUANDO EXISTE UMA COLUNA COM DOIS OU MAIS ELEMENTOS MAIORES QUE ZERO E DOIS OU MAIS MENORES QUE ZERO: se encontrarmos na primeira partição uma coluna que contenha dois ou mais elementos maiores que zero e dois ou mais elementos menores que zero, então selecionemos um elemento maior (menor) que zero. Efetuamos na matriz particionada combinações lineares entre a linha que contém o maior (menor) e as outras, exceto uma, que contém os menores (maiores), obtendo uma nova matriz particionada para ser usada nas etapas sucessivas. Retornamos para 3.

5. COMBINAÇÃO ENTRE LINHAS QUANDO EXISTE UMA COLUNA COM TODOS OS ELEMENTOS MAIORES QUE ZERO OU TODOS MENORES QUE ZERO: se encontrarmos na primeira partição uma coluna na qual todos os seus elementos possuem o mesmo sinal, então selecionemos um não nulo destes elementos e efetuemos, na matriz particionada, combinações lineares entre a linha que contém este não nulo e as outras, de forma a anularmos os outros elementos desta coluna, obtendo uma nova matriz particionada para ser usada nas etapas sucessivas. A base não compreenderá mais as soluções positivas. Retornamos para 2.

A "Figura 5" ilustra por uma Rede de Petri, um resumo essencial das idéias pertinentes ao presente trabalho quanto ao cálculo de invariantes, com suas ligações algébricas calcadas em transformações lineares, matrizes, matriz em escada, teorema das combinações lineares, núcleo de uma transformação etc.

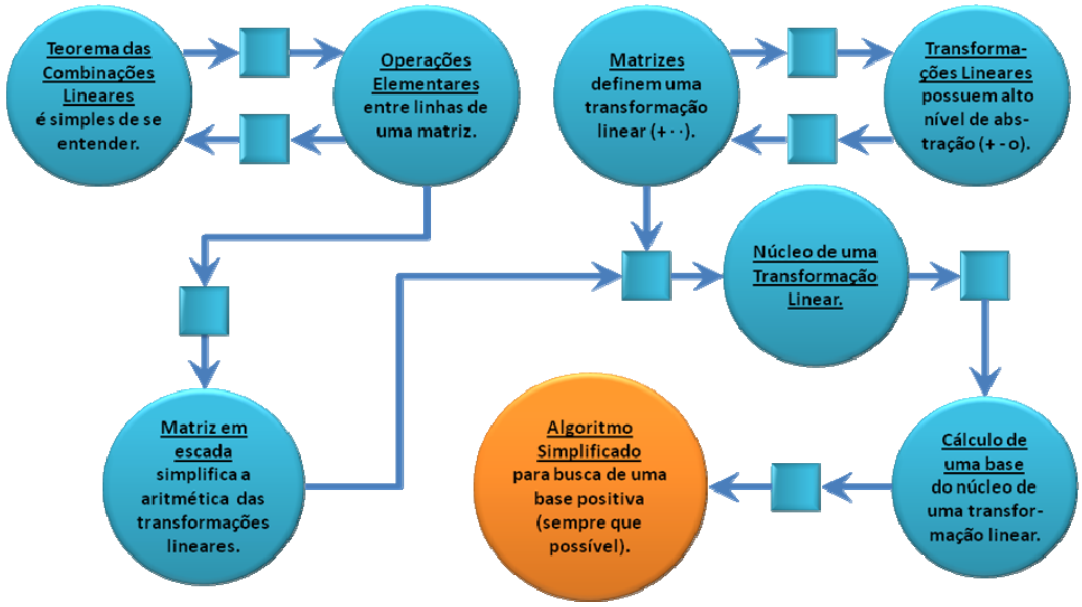

Figura 5 – Cálculo de Invariantes em Redes de Petri.

5. DIAGRAMAS DE CASO DE USO E INTERFACE GRÁFICA

Serão utilizados neste trabalho, dentre os 9 diagramas da *UML*, apenas 3: o de Casos de Uso, o de Classes e o de Atividade. Tais diagramas estão, para os estudantes de engenharia, entre os mais importantes. Não que os outros não o sejam, é que aqui se segue um enfoque mais tradicional para tais cursos que, anos atrás, valiam-se, e ainda se valem atualmente, dos Fluxogramas ISO 5807 quando do ensino e aprendizado de programação.

Os Diagramas de Caso de Uso possibilitam, através de uma linguagem simples, a compreensão do comportamento externo de um sistema. Tal simplicidade é inerente de forma que o sistema seja compreendido por qualquer pessoa envolvida no desenvolvimento do mesmo. Os casos de uso descrevem a visão do sistema por uma perspectiva do usuário:

- operam com qualidades e relações, e não com a realidade sensível: é abstrato;
- são fáceis de se manejar, de se moldar: é flexível;
- são espontâneos e não se atém à fórmula estabelecidas: é informal.

Os Diagramas de Caso de Uso mapeiam os requisitos do sistema e são base para os demais diagramas da UML. Eles estão intimamente ligados à interface gráfica do sistema. A "Figura 6" ilustra o Diagrama de Casos de Uso para o sistema Invariantes em Redes de Petri e a "Figura 7" mostra a interface gráfica desenvolvida para o mesmo sistema.

Os Diagramas de Casos de Uso evidenciam os atores e os casos de uso propriamente ditos. Os atores são representados pelo uso do símbolo de um boneco 🚹 e são os papéis desempenhados pelos diversos usuários que se valerão das funcionalidades do sistema. Os atores não são necessariamente uma pessoa, podem ser um *hardware* ou um *software* que interage com o sistema. Os casos de uso são representados pelo símbolo de uma oval ⬭ e são os serviços, as tarefas ou funções utilizadas pelos usuários do sistema. O sistema é representado pela caixa retangular que contém os casos de uso.

Na "Figura 6" observa-se os atores *Aluno* e *Professor* que são especializações do ator *Usuário*. Tal fato é indicado pelo uso da seta contínua com ponta vazada em forma de triângulo eqüilátero ⇑. Os casos de uso *Ajustar Lugares*, *Ajustar Transições* e *Ajustar Arcos* são especializações do caso de uso *Ajustar Elementos*. O caso de uso *Ajustar Elementos* estende a funcionalidade do caso de uso *Desenhar Rede* e tal fato é indicado pelo uso da seta tracejada com ponta de 60° ← e com a palavra *extend* entre os símbolos << e >> (estereótipo). Os casos de uso *Calcular Matriz IN*, *Calcular Matriz OUT* e *Calcular INVARIANTE* incluem o caso de uso *Mostrar Matriz* e tal fato é indicado pela mesma seta da extensão, porém com a palavra *include* entre os símbolos << e >> (estereótipo).

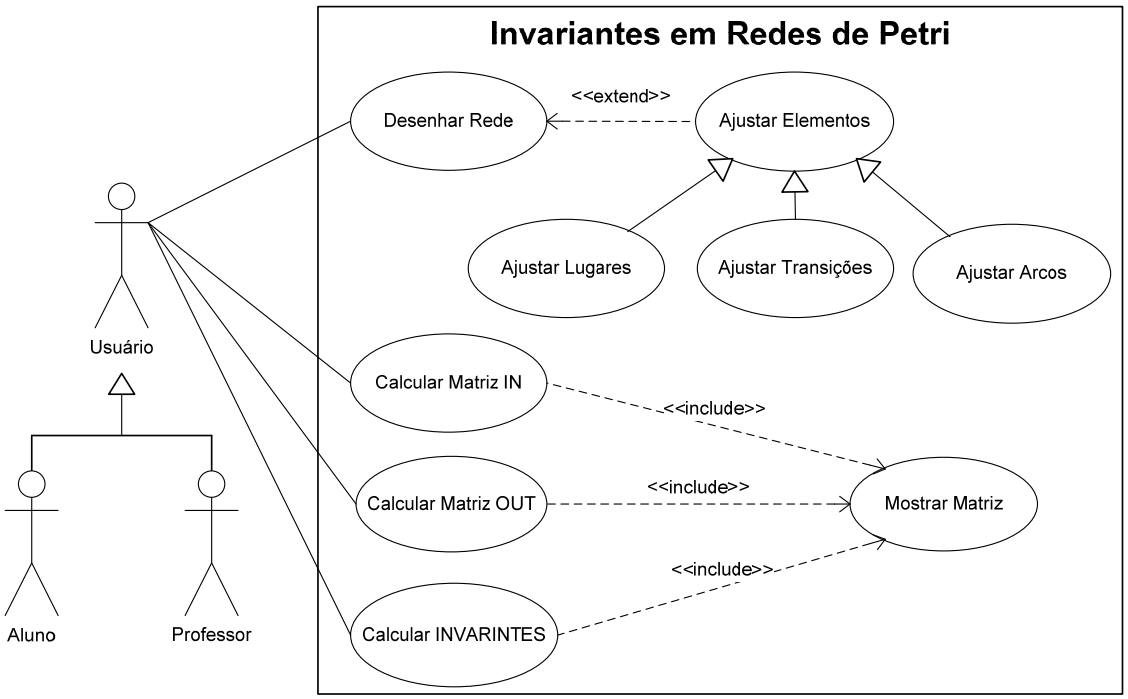

Figura 6 – Diagrama de Caso de Uso para o sistema.

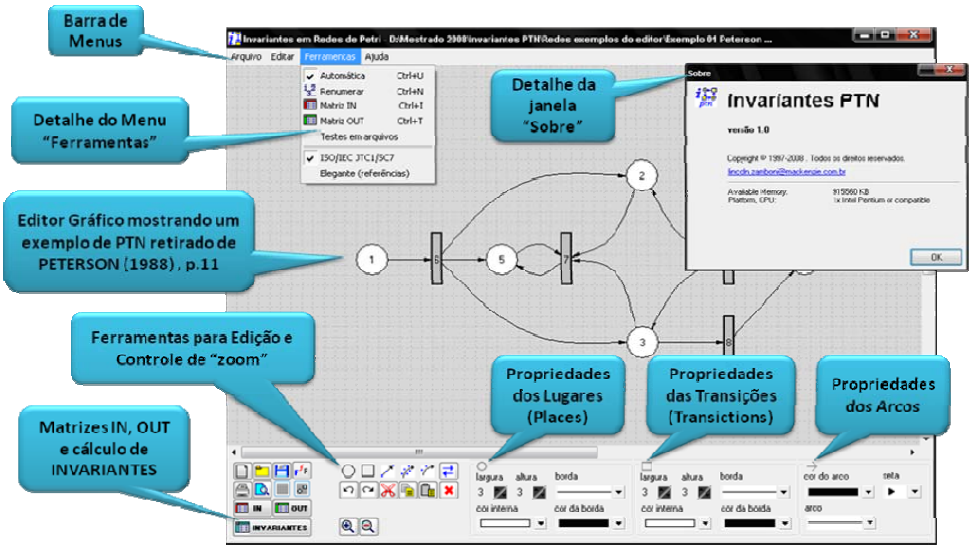

Figura 7 – Interface do sistema Invariantes em Redes de Petri.

Na "Figura 8" mostra-se, como exemplo, a funcionalidade descrita pelo caso de uso *Calcular INVARIANTES* e que é efetivada pelo aperto do botão *INVARIANTES* na interface gráfica. Observa-se também uma janela mostrando uma matriz com os invariantes. Tal janela proveio da funcionalidade descrita pelo caso de uso *Mostrar Matriz*, que é incluído também pelos casos de uso *Calcular Matriz IN* e *Calcular Matriz OUT*.

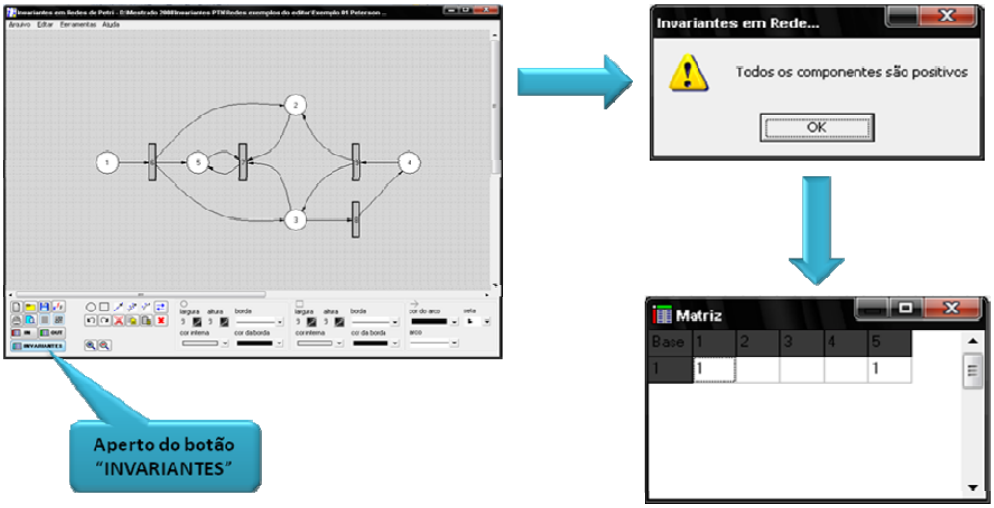

Figura 8 – Exemplo de cálculo de invariantes com o uso do sistema.

6. DIAGRAMA DE CLASSES E ALGORITMO SIMPLIFICADO

Os Diagramas de Classe mostram as diferentes classes que compõem um sistema e como elas se relacionam umas com as outras. O propósito destes diagramas é evidenciar a estrutura estática do sistema que é modelado e, para tal, mostram as classes em conjunto com os seus atributos e operações, assim como as relações estáticas entre elas e quais as classes que conhecem outras classes ou que fazem parte de outra classe, porém não mostram as chamadas de métodos entre elas.

A "Figura 9" mostra um trecho do Diagrama de Classes para a implementação da matriz particionada e também do algoritmo simplificado. O código em C++ para tal implementação pode ser visto na "Figura 11".

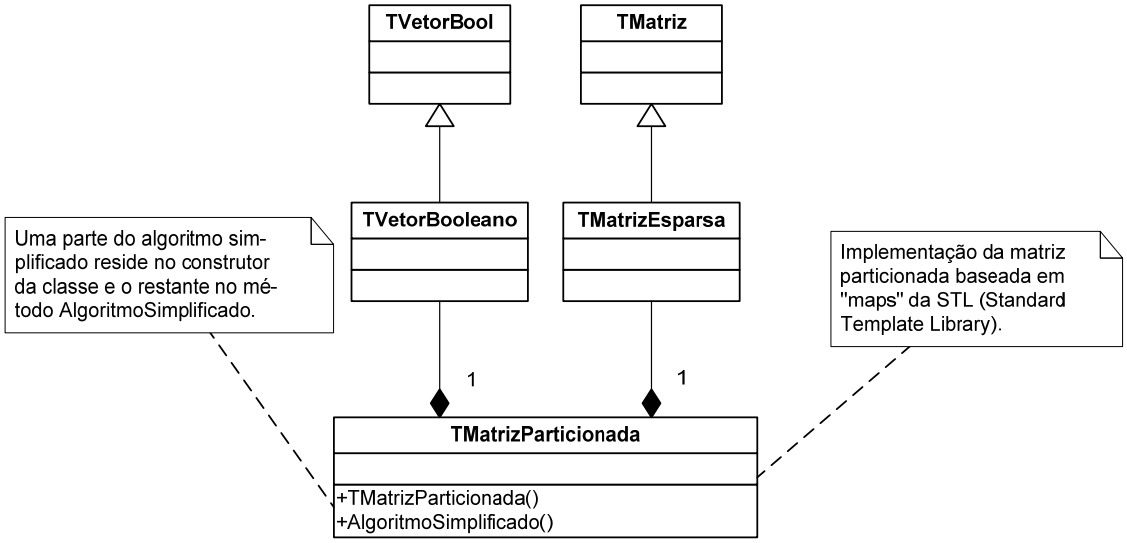

Figura 9 – Trecho do Diagrama de Classes para a matriz particionada.

7. DIAGRAMA DE ATIVIDADE

A “Figura 10” mostra o Diagrama de Atividade para a inicialização da matriz particionada.

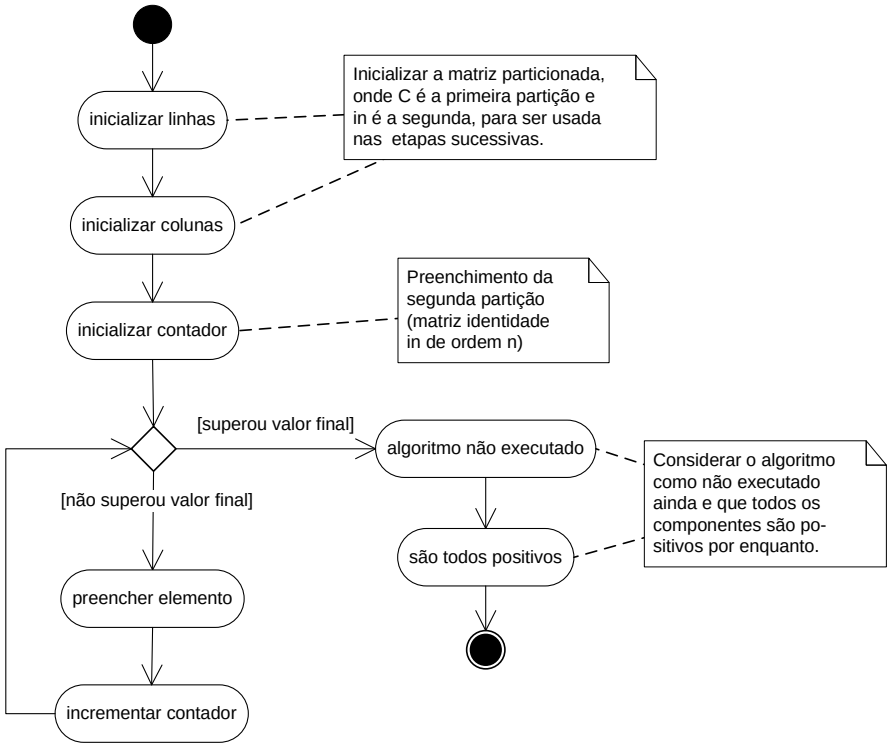

Figura 10 – Diagrama de Atividade para inicialização da matriz particionada.

8. CÓDIGO FONTE EM C++

A “Figura 11” mostra trechos do código fonte escrito em C++ para o sistema. Cabe salientar que o código fonte abrange aproximadamente 2700 linhas e seria inviável sua exposição aqui.

```
#ifndef UInvariantesH
#define UInvariantesH
#include <fstream>
#include <map>
using namespace std;
//---------------------------------------------------------------------------
// Implementação da matriz particionada baseada em "maps" da STL (Standard Template Library).
// É uma solução moderna para o caso de matrizes esparsas, no que tange a novidade e a difi-
// culdade da implementação.
// A matriz particionada é composta de dois vetores booleanos (LinRemDeC e ColRemDeC do tipo
// TVetorBooleano) e uma matriz esparsa (Cin do tipo TMatrizEsparsa). Os vetores booleanos são
// utilizados para a marcação das linhas e das colunas removidas. A matriz esparsa é utilizada
// para representar as duas partições envolvidas no algoritmo simplificado (matriz de incidên-
// cia e matriz identidade de ordem n. Esta última transformar-se-á na matriz dos componentes
// da base de invariantes após a aplicação do algoritmo simplificado.
//---------------------------------------------------------------------------
typedef unsigned int TIndice;
// Tipo para representar os índices dos vetores e das matrizes.
//---------------------------------------------------------------------------
typedef map<TIndice, bool, less<TIndice> > TVetorBool;
class TVetorBooleano: private TVetorBool{
  // ... código omitido
};
//---------------------------------------------------------------------------
typedef map<TIndice, int, less<TIndice> > TMatriz;
class TMatrizEsparsa: private TMatriz{
  // ... código omitido
};
//---------------------------------------------------------------------------
class TMatrizParticionada{
  public:
  TMatrizParticionada(TMatriz::key_type lin, TMatriz::key_type col);
  // Contrutor da matriz particionada.
  TMatrizParticionada(const char *Arq, const bool Transp = false);
    // Contrutor da matriz particionada a partir de um arquivo ".cnn". O parâmetro Transp
  // sinaliza se será lida a matriz transposta ou não.
  // ... código omitido
  private:
  // Implementação de um vetor booleano baseado em "maps" da STL(Standard Template Library).
  // Os elementos "true" do vetor booleano terão seus valores alocados na memória e os "false"
  // não terão.
  //   Exemplo:
  //   O vetor v[1]=false v[2]=true v[3]=true v[4]=false terá apenas os elementos [2] e [3]
  //   alocados na memória. Os elementos restantes não serão alocados.
  TVetorBooleano LinRemDeC, ColRemDeC;

  // Marcação, respectivamente, das linhas e das colunas removidas. "false" significa não re-
  // movida e "true" significa removida.
  // Os elementos não nulos da matriz esparsa terão seus valores alocados na memória e os nu-
  // los não terão.
  //   Exemplo:
  //   A matriz
  //   a[1,1]= 11 a[1,2]= 0 a[1,3]= 0
  //   a[2,1]= 0 a[2,2]= 22 a[2,3]= 0
  //   a[3,1]= 0 a[3,2]= 0 a[3,3]= 0
  //   terá apenas os elementos [1,1] e [2,2] alocados na memória. Os restantes não serão
  //   alocados.
  // A conversão dos índices duplos da matriz esparsa para índices simples do "map" e vice-
  // versa é feita da seguinte forma:
  //   [linha, coluna] -> [c*(linha-1)+coluna]
  //   [indice/c, indice%c] <- [indice]
  TMatrizEsparsa Cin;
  // C é a primeira partição (matriz de incidência) e in é a segunda partição (matriz iden-
  // tidade de ordem n).
  // ... código omitido
};
//---------------------------------------------------------------------------
#endif
```

Figura 11 – Trecho do código fonte.

9. CONSIDERAÇÕES FINAIS

Este trabalho mostra como é possível o desenvolvimento de ferramentas computacionais de extrema utilidade, para aplicações em engenharia, sem qualquer ônus para o desenvolvedor. O estudante de engenharia possui diversas ferramentas a seu dispor, em geral obtidas através de downloads gratuitos feitos via Internet. A UML ajuda a organizar as idéias e informações do projeto e as linguagens de programação, bibliotecas de funções e ambientes gráficos disponíveis permitem a implementação das mesmas. Devemos salientar que a presença de disciplinas que envolvem computação e programação é essencial nas primeiras etapas dos cursos de Engenharia, para que o aluno possa ter conhecimento suficiente para desenvolver soluções automatizadas de seus problemas acadêmicos, visando também à utilização futura em sua vida profissional.

REFERÊNCIAS BIBLIOGRÁFICAS

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML - guia do usuário**. Rio de Janeiro: Editora Campus, 2000.

LEE, R. C.; TEPFENHART, W. M. **UML and C++ - a practical guide to object-oriented development**. São Paulo: MAKRON Books, 2001.

LIU, Lung-Chun; HOROWITZ, Ellis. **A formal model for software project manegement**. IEEE Transactions on Software Engineering, v. 14, n. 10, p. 1280-1293, out., 1989.

NOBLE, B.; DANIEL, J. W. **Álgebra linear aplicada**. Rio de Janeiro: Prentice-Hall do Brasil, 1986.

OMG. Object Management Group, Inc. Disponível em: <http//www.omg.org >. Acesso em: 20 abr. 2008.

PETERSON, J. L. **Petri net theory and the modeling of systems**. New Jersey: Prentice-Hall, Inc., 1981.

STRANG, G. **Linear algebra and its applications**. 3. ed. San Diego: Harcourt Brace Jovanovich, Publishers, 1988.

STROUSTRUP, Bjarne. **The C++ programming language**. 2. ed. New Jersey: Addison-Wesley Publishing Company, 1991.

VALETTE, Robert**. Les réseaux de Petri**. L.A.A.S./C.N.R.S. Toulouse. mai., 1992.

ZAMBONI, L. C.; PAMBOUKIAN, S. V. D.; BARROS, E. A. R. **C++ para universitários**. 2. ed. São Paulo: Páginas & Letras, 2008.