UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

---

DISSERTAÇÃO
apresentada à UTFPR
para obtenção do grau de

MESTRE EM CIÊNCIAS

por

ERIK EUGÊNIO KÜNZLE

IMPLEMENTAÇÃO E VALIDAÇÃO DE UMA FERRAMENTA COMPUTACIONAL PARA ANÁLISE DE REDES DE PETRI TEMPORAIS

Orientador:

Prof. Dr. LUIS ALLAN KÜNZLE **INF, UFPR - BRASIL**

Banca Examinadora

Presidente:

Prof. Dr. PAULO CÉZAR STADZISZ **CPGEI, UTFPR - BRASIL**

Examinadores:

Prof. Dr. FABIANO SILVA **INF, UFPR - BRASIL**

Prof. Dr. RICARDO LÜDERS **CPGEI, UTFPR - BRASIL**

Dezembro de 2006

ERIK EUGÊNIO KÜNZLE

IMPLEMENTAÇÃO E VALIDAÇÃO DE UMA FERRAMENTA COMPUTACIONAL PARA ANÁLISE DE REDES DE PETRI TEMPORAIS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do grau de Mestre em Ciências.

Orientador:
Prof. Dr. LUIS ALLAN KÜNZLE

Curitiba - PR - Brasil
Dezembro de 2006

Sumário

Lista de Figuras

Lista de Tabelas

Agradecimento

Gostaria de agradecer de forma especial a Deus que me proporcionou a paz necessária para a realização deste trabalho.

Ao orientador Professor Doutor Luis Allan Künzle, pela orientação, supervisão, confiança e paciência que me concedeu para o desenvolvimento desta pesquisa e por todo o apoio prestado durante este tempo.

Aos membros da banca examinadora por aceitar avaliar este trabalho e por suas valiosas contribuições.

Ao amigo Evangivaldo, pela colaboração, apoio, pesquisa e sugestões que foram fundamentais na elaboração e consolidação do trabalho.

A todos os meus amigos e familiares, pela sua ajuda, apoio e carinho.

Notação

$\mathbf{A}$	Matriz intervalar
A	Matriz real
$\underline{\mathsf{A}}$	Matriz real limitante inferior
$\overline{\mathsf{A}}$	Matriz real limitante superior
$\mathbf{x}$	Vetor intervalar $n \times 1,\ n = 1, 2, \ldots$
x	Vetor real
$\underline{\mathsf{x}}$	Vetor real limitante inferior
$\overline{\mathsf{x}}$	Vetor real limitante superior
$f(x)$	Função real
$f(\mathbf{x})$	Função real com domínio intervalar
$\mathbf{F}$	Função intervalar
$\mathscr{U}$	Conjunto universo
$\mathbb{N}$	Conjunto dos naturais
$\mathbb{Z}$	Conjunto de inteiros
$\mathbb{Q}$	Conjunto dos números racionais
$\mathbb{R}$	Conjunto dos números reais
$\mathbb{IR}$	Conjunto dos intervalos reais

Abreviaturas

- ADC - *Analog-Digital Converter* - Conversor Analógico-Digital
- ARM - *Advanced RISC Machine* - Processador RISC avançado
- C - Matriz de incidência
- CPGEI - Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial
- CTP - Vetor de lugares pertencentes a um conflito
- DAC - *Digital-Analog Converter* - Conversor Digital-Analógico
- ET - *Enabled Transitions* - Vetor de transições habilitadas
- ETP - *Past Enabled Transitions* - Vetor de transições habilitadas anteriormente
- FT - *Firable Transitions* - Vetor de transições disparáveis
- IIF - *Interchange File Format* - Intercâmbio de informação entre aplicações
- IM - Matriz de transições
- IMR - Matriz intervalar
- LCD - *Liquid Crystal Display* - Monitor de cristal líquido
- LSIP - Laboratório de Sistemas Inteligentes de Produção
- M - Vetor de marcação
- Mo - Vetor de marcação inicial
- PN - *Petri Nets*
- Pos - Matriz de incidência posterior
- Pre - Matriz de incidência prévia

- PS/2 - *Personal System/2* - Sistema de computador pessoal
- PT - Vetor de conexões entre transições iniciais e transições finais
- RdP/RdPs - Rede de Petri/Redes de Petri
- RISC - *Reduced Instructions Set Computer* - Computador com um conjunto reduzido de instruções
- RPT/RPTs - Rede de Petri Temporal/Redes de Petri Temporais
- RTC - *Real Time Communication* - Comunicação em tempo real
- SRAM - *Static Random Access Memory* - Mémoria estática de acesso aleatório
- TT - Matriz de intervalos de tempo global
- TTh - *Thread Time* - Tempo da *thread*
- UML - *Uniform Modeling Language* - Linguagem de modelagem
- USB - *Universal Serial Bus* - Barramento serial universal

Resumo

As Redes de Petri (RdPs) são um tipo particular de grafos bipartidos dirigidos que oferecem um ambiente uniforme para a modelagem, análise formal e simulação de sistemas a eventos discretos.

O presente trabalho consiste no desenvolvimento de uma ferramenta computacional de análise de Redes de Petri Temporais. A ferramenta permite ao usuário construir graficamente uma RdP, simular sua execução de maneira automática ou passo a passo, e escolher entre os caminhos a seguir a partir de um conflito. A análise temporal é baseada em tempo global e a metodologia empregada utiliza os fundamentos da álgebra intervalar. A ferramenta permite obter resultados como: diagrama de classes, grafo de classes para redes seguras, análise temporal usando tempo global, seqüências de disparo válidas para alcançar uma determinada marcação sem simular a execução da rede e calcular intervalos de disparo de transições usando álgebra intervalar.

Um estudo de caso foi modelado para a validação da ferramenta. Consiste no desenvolvimento de um sistema embarcado que mostra informações de temperatura e pressão em seu *display*. Tais informações são enviadas de maneira aleatória por uma máquina externa utilizando uma interface de comunicação serial. O sistema operacional tem como base o uso de *threads* para executar a aplicação, e para fazer isso utiliza um algoritmo de escalonamento tipo *Round-Robin*, que organiza a execução das *threads* de forma seqüencial a partir da ordem de criação, levando em consideração o nível de prioridade de cada *thread*. O objetivo do estudo de caso foi traduzir o funcionamento do escalonador, bem como o funcionamento interno de cada *thread* em Redes de Petri Temporais para analisar temporalmente o comportamente do sistema como um todo, verificando o tempo de execução, sincronismo entre as *threads*, e se possível, propor melhorias no *software* embarcado.

A ferramenta foi desenvolvida utilizando um ambiente integrado de desenvolvimento para o sistema operacional *Windows* com base na linguagem pascal.

Palavras-Chave: Rede de Petri Temporal, Álgebra Intervalar, Tempo Global, Simuladores, Sistema Embarcado, Sistema Operacional X, *Threads* e Pascal.

Abstract

Petri nets (PN) are a particular type of directed bipartite graphs that offer an uniform environment for the modeling, formal analysis, and simulation of discrete events systems.

The present work consists of the development of a computational tool for the analysis of Time Petri Nets. It allows the user to construct a PN graphically, to simulate its execution in automatic way or by step, and to choose among the ways to follow in a conflict situation. The time analysis is based on global time and the employed methodology uses the concepts of interval analysis. The tool allows to get several results, as diagram of class, graph of class for safe nets, time analysis using global time, valid sequences of transitions firings to reach a specific marking without simulating the execution of the net and to calculate intervals of transistions firing being used interval analysis.

A case study was shaped for the validation of the tool. It consists of the development of an embedded system that shows the information of temperature and pressure in its display. Such information are sent in random way from an external machine using the serial interface. The operating system uses threads to execute the application, and to make this, it uses an Round-Robin algorithm for scheduling, that organizes the execution of threads in a sequential way, taking in account the priority level of each thread. The objective of the study case was to translate the operation of the scheduler, as well as the internal functioning of each thread in Time Petri Nets to analyze the behavior of the system as a whole, verifying the execution time, synchronism between threads, and if possible, to consider improvements in embedded software.

The tool was developed using an integrated developing environment for the Windows operating system based on pascal language.

Keywords: Time Petri Nets, Interval Analysis, Global Time, Simulation, Embedded Systems, Operational System X, Threads e Pascal.

1 Introdução

As crescentes necessidades do mundo atual fazem com que a eficiência, produtividade e segurança sejam importantes aspectos que devem ser levados em consideração ao projetar um sistema. Para alcançar estes objetivos, é necessário levar em conta técnicas de modelagem e análise que possibilitem o aproveitamento eficiente dos recursos de tecnologia atualmente disponíveis. A aplicação da teoria de Redes de Petri como técnica para modelagem, análise, controle e projeto de Sistemas a Eventos Discretos tem se mostrado eficiente para sistemas desta natureza [Gom96].

A maioria das ferramentas computacionais de Redes de Petri permite sua edição gráfica e simulação. Na base de dados das ferramentas de Rede de Petri [Wor06], de um total de 66 ferramentas, 20 possibilitam construir graficamente a rede e simular sua execução levando em consideração aspectos temporais. Em geral, cada ferramenta implementa uma metodologia específica de análise.

Esta dissertação teve como objetivo o desenvolvimento de uma aplicação que permite modelar graficamente uma Rede de Petri, simular sua execução tanto em modo passo-a-passo, no qual o usuário intervém para decidir situações de conflito, quanto em modo automático, onde os conflitos são decididos de forma aleatótia. A análise temporal implementada é baseada em uma nova abordagem no tratamento do tempo, que é o conceito de tempo global, associado ao formalismo da álgebra intervalar. Esta abordagem foi desenvolvida no Laboratório de Sistemas Inteligentes de Produção do CPGEI [LLK05].

Para validar a ferramenta foi realizado um estudo de caso envolvendo a modelagem e a análise de um software de controle de um sistema embarcado. Este estudo de caso foi baseado nos sistemas desenvolvidos no Laboratório de Inovação e Tecnologia de Sistemas Embarcados (LIT) da UTFPR.

O sistema apresentado foi desenvolvido para a plataforma *Windows*, utilizando um ambiente integrado de desenvolvimento que tem como base a linguagem pascal e disponibiliza

os recursos necessários para programação orientada a objetos.

Organização da Dissertação

No primeiro capítulo são apresentados os formalismos com os quais as metodologias, implementadas na ferramenta, foram definidas. Temos portanto uma breve descrição do formalismo de Redes de Petri e Redes de Petri Temporais, considerando suas definições, propriedades e características. Além disso é apresentada uma noção básica da álgebra intervalar, usada para calcular os intervalos de disparo das transições na metodologia de tempo global.

No capítulo seguinte, é apresentado a metodologia de análise temporal baseada em tempo global, utilizada como base nos algoritmos de análise da ferramenta. Neste capítulo também é apresentado um método para redução das classes de Rede de Petri Temporal, bem como um algoritmo que usa elementos dos métodos de ordem parcial para obter uma rede reduzida. Esse algoritmo foi implementado na ferramenta.

A descrição da ferramenta desenvolvida é detalhada no capítulo 4, onde são apresentadas as principais características, as estruturas de dados, o caso de uso, as classes específicas criadas para o ambiente gráfico do sistema e o funcionamento dos algoritmos que fornecem a base de cálculo e de análise da Rede de Petri modelada.

O capítulo 5 descreve o estudo de caso que foi modelado para validar aspectos temporais da ferramenta com o intuito de traduzir o funcionamento de um *software* embarcado em modelos de Rede de Petri Temporal, analisando temporalmente o comportamente do sistema embarcado como um todo.

Por último, procede-se à apresentação das conclusões do trabalho, conjuntamente com algumas sugestões para novos tópicos a serem investigados em trabalhos futuros.

No Anexo é feita uma breve apresentação das ferramentas computacionais de análise baseadas em Redes de Petri, atualmente disponíveis, que proporcionam ao usuário criar modelos para análise e simulação e que levam em consideração aspectos temporais [Wor06, Pai04].

No Apêndice é apresentado o código fonte do *software* embarcado desenvolvido no estudo de caso.

2 Formalismos de Base

Neste capítulo serão apresentados os dois formalismos que constituem a base teórica do método de análise implementado na ferramenta computacional: as Redes de Petri e a Álgebra Intervalar.

2.1 Redes de Petri

Esta seção apresenta alguns conceitos sobre Redes de Petri, necessários apenas para uniformizar a linguagem usada no texto. Também é feita uma introdução aos diferentes modelos de Redes de Petri com extensão de tempo, principalmente, as Redes de Petri Temporais, por serem a base desse trabalho.

2.1.1 Redes de Petri

As Redes de Petri constituem uma poderosa ferramenta formal para modelagem, análise, verificação e validação de propriedades de sistemas concorrentes, sistemas sincronizantes e sistemas compartilhados [Mur89].

Estrutura

Definição 2.1 (`Rede de Petri`) *Uma Rede de Petri Lugar/Transição é uma quádrupla* $N = \langle P, T, Pre, Pos \rangle$, *sendo:*

1. P, *o conjunto finito e não vazio de lugares*
2. T, *o conjunto finito e não vazio de transições*
3. $P \cap T = \emptyset$

4. $Pre : P \times T \longrightarrow \mathbb{N}$, *a função incidência de entrada, sendo* $\mathbb{N}$ *o conjunto dos números naturais (peso dos arcos de entrada). Pré-condições ligando lugares às transições.*

5. $Pos : T \times P \longrightarrow \mathbb{N}$, *a função incidência de saída, sendo* $\mathbb{N}$ *o conjunto dos números naturais (peso dos arcos de saída). Pré-condições ligando transições a lugares.*

Definição 2.2 (`Rede de Petri Ordinária`) *Uma RdP é dita ordinária se suas funções de incidência podem assumir apenas valores* 0 *e* 1. *Ou seja,* $\forall t \in T$, $\forall p \in P$:

$$\begin{cases} Pre(p,t) \in \{0, 1\} \\ Pos(t,p) \in \{0, 1\} \end{cases}$$

Definição 2.3 (`Conjuntos Característicos`) *Seja* N *uma RdP,* $t \in T$ *e* $p \in P$. *Os seguintes conjuntos são então, definidos:*

Conjunto dos lugares de entrada de t*:* $\quad {}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$

Conjunto dos lugares de saída de t*:* $\quad t^\bullet = \{p \in P \mid Pos(t,p) > 0\}$

Conjunto das transições de entrada de p*:* $\quad {}^\bullet p = \{t \in T \mid Pos(t,p) > 0\}$

Conjunto das transições de saída de p*:* $\quad p^\bullet = \{t \in T \mid Pre(p,t) > 0\}$

Definição 2.4 (`Representação Matricial`) *Uma RdP pode ser definida matricialmente por meio de suas matrizes características. Seja* m *o número de lugares de* P *e* n *o número de transições de* T. *As matrizes características de uma RdP são:*

1. *matriz incidência de entrada,* $\mathtt{Pre} = [Pre(p_i,t_j)]_{m \times n}$.

2. *matriz incidência de saída,* $\mathtt{Pos} = [Pos(p_i,t_j)]_{m \times n}$.

3. *matriz incidência,* $\mathtt{C} = \boldsymbol{Pos} - \boldsymbol{Pre}$

$i = 1, \ldots, m,\ j = 1, \ldots, n$

Os vetores coluna (linha) das matrizes `Pre` e `Pos` serão denotados como:

$\mathtt{Pre}(.,t) = \mathtt{Pre}(t) \quad (\mathtt{Pre}(p,.) = \mathtt{Pre}(p))$
$\mathtt{Pos}(.,t) = \mathtt{Pos}(t) \quad (\mathtt{Pos}(p,.) = \mathtt{Pos}(p))$

Comportamento

A dinâmica de uma Rede de Petri, em termos de transição de estado, pode ser representada por sua marcação e a evolução da marcação.

Definição 2.5 (*Marcação*) *A marcação de uma RdP é uma aplicação associando um inteiro não negativo a cada lugar da rede. A marcação pode ser representado por um vetor* $m-dimensional$*, sendo* m *o número de lugares da rede.*

Definição 2.6 (*Rede de Petri Marcada*) *Uma RdP marcada é o par* $\langle N, m_0 \rangle$*, sendo* m_0 *a marcação inicial da rede*

Definição 2.7 (*Habilitação de Transição*) *Uma transição* $t \in T$ *estará habilitada, ou sensibilizada, por uma marcação* m *se, somente se,* $\forall p \in {}^\bullet t$*, for verificada a seguinte condição:*

$$m \geq Pre(t). \tag{2.1}$$

O conjunto de transições habilitadas por uma marcação m, é denominado por $\mathscr{H}$.

Definição 2.8 (*Mudança de Marcação*) *Se* t *é uma transição habilitada por uma marcação* m_0*, então* t *pode disparar. O disparo da transição* t *retira marcas de* ${}^\bullet t$ *e coloca marcas em* $t^\bullet$*, levando a rede a uma nova marcação,* m_1*. A evolução da marcação na rede, é descrita pela equação:*

$$m_1 = m_0 + Pos(t) - Pre(t) = m_0 + C(t) \tag{2.2}$$

Para uma marcação m_k*, alcançável a partir de* m_0*, pelo disparo do vetor quantidade de disparos de transições,* $q \in (\mathbb{Z}^+)^n$*, a equação (2.2) passa à denominada equação de estado, como segue,*

$$m_1 = m_0 + Cq \tag{2.3}$$

Definição 2.9 (*Alcançabilidade*)*. O conjunto de todas as marcações alcançáveis, a partir da marcação inicial* m_0*, pelo disparo de seqüências de transições, é denominado conjunto de alcançabilidade de* m_0 *e denotado* $\mathscr{R}(m_0)$*.*

A figura 1 mostra a estrutura de uma RdP e a tabela 1 apresenta sua nomenclatura.

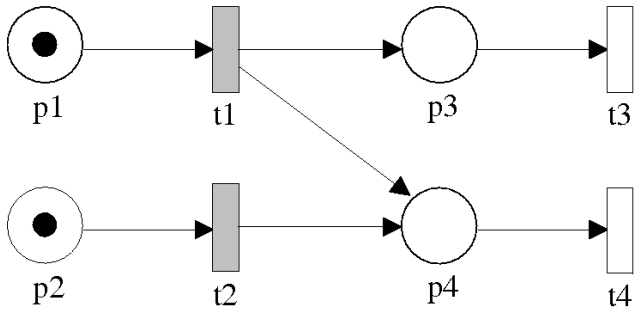

Figura 1: Exemplo de uma RdP

Tabela 1: Nomenclatura da RdP da figura 1

RdP	
Lugares	p1, p2, p3 e p4
Transições	t1, t2, t3 e t4

Propriedades

Definição 2.10 (`Rede Reversível`) *Uma RdP é dita reversível se, e somente se, a partir de qualquer marcação alcançável da rede é possível alcançar a marcação inicial.*

Ao longo deste texto, as redes reversíveis também serão tratadas como redes cíclicas, embora uma rede cíclica, não necessariamente seja reversível.

Definição 2.11 (`Rede Consistente`) *Uma RdP é dita consistente se, e somente se, existe uma seqüência de disparo, envolvendo todas as transições da rede, tal que a rede seja reversível.*

Definição 2.12 (`Rede K-Limitada`) *Uma RdP é dita* $k-$`limitada` *se, e somente se, qualquer que seja sua marcação, qualquer lugar da rede, tem capacidade limitada a* k *marcas. Quando* $k=1$ *a rede é dita segura.*

Definição 2.13 (`Grafo Marcado`) *Um grafo marcado é uma RdP ordinária tal que*

$$|^{\bullet} p_i| = 1 \;\; e \;\; |p_i^{\bullet}| = 1, \;\; \forall \, p_i \in P$$

isto é, cada lugar da rede tem apenas um arco de entrada e um arco de saída.

Grafo marcado permite a modelagem de concorrência e sincronismo, mas não conflito.

Definição 2.14 (`Livre Escolha`) *Uma rede de livre escolha é uma RdP ordinária tal que*

$$| p_i^\bullet | > 1 \Rightarrow |^\bullet t_k | = 1, \ \forall t_k \in p_i^\bullet$$

isto é, se duas transições t_i e t_j compartilham o mesmo lugar de entrada p, então, p é o único lugar de entrada de t_i e t_j.

2.1.2 Redes de Petri com Tempo

As Redes de Petri foram propostas sem qualquer interferência do tempo em seu comportamento. Porém, considerar que não existe uma especificação de quando uma transição habilitada irá disparar é o mesmo que admitir um indeterminismo sobre quando será esse disparo. As diversas formas de especificar o tempo nas Redes de Petri mostram formas distintas de reduzir esse indeterminismo.

Um aspecto importante nessas formas de especificação do tempo é como o tempo é interpretado. Este aspecto é chamado de *semântica do tempo* e há, basicamente, duas formas distintas:

- **Tempo de Sensibilização** que determina o tempo que deve transcorrer desde o momento de habilitação de uma transição até o momento de disparo. O disparo ocorre de forma atômica, ou seja, disparo instantâneo.
- **Tempo de Disparo** no qual o disparo de uma transição ocorre em três fases: retirada da marca, disparo e colocação da marca.

Para determinados sistemas é importante que o disparo de uma transição habilitada deva ocorrer num determinado tempo. Nestes casos, a semântica do tempo é chamada de *semântica forte*, em oposição à *semântica fraca*, cujo disparo de uma transição não é obrigatório, porém, se ocorrer, deve ser num determinado tempo.

As Redes de Petri com tempo foram introduzidas por Ramchandani [Ram74] e Merlin [Mer74] no início dos anos 70. Desde então, um grande número de diferentes modelos de Redes de Petri para especificar sistemas temporizados têm sido propostos na literatura, cada um deles sendo usado para uma aplicação específica. Geralmente, os modelos se diferenciam em aspectos tais como: tipo de temporização, localização da restrição temporal e propriedade

da restrição [CMS99]. Independente desses aspectos, pode-se resumir as extensões temporais das Redes de Petri em quatro grandes categorias:

- **Redes de Petri Temporais** (*Time Petri Nets*). A restrição temporal é um intervalo de tempo associado a cada transição. Foram inicialmente usadas na descrição de protocolos de comunicação. Entretanto, seu campo de aplicação tem-se ampliado para áreas como: manufatura, sistemas de tempo real, validação e verificação de sistemas [Nid94, BM82, BV95, MGV00].

- **Redes de Petri Temporizadas** (*Timed Petri Nets*). Permitem que transições ou lugares retenham marcas durante um intervalo de tempo. São comumente usadas na análise de sistemas de manufatura [Fre82, Sif77].

- **Marcas Temporizadas**. A restrição temporal é associada à marca ao se propagar na rede. São largamente usadas em problemas de produção [Cel82, TYC95].

- **Modelos Probabilísticos**. São as chamadas Redes de Petri Estocásticas e são usadas para estimação de desempenho de grandes sistemas. Uma probabilidade é associada a um dos elementos da rede, transição, lugar ou marca [Mur89].

Ao longo desse texto serão analisadas as Redes de Petri proposta por Merlin e Faber [Mer74], denominadas de Redes de Petri Temporais (RPTs). Este modelo mostra-se adequado para expressar a maioria dos requisitos temporais, além de ser mais genérico que as demais extensões temporais [BV95, MGV00].

2.1.3 Redes de Petri Temporais

A idéia básica das RPTs é associar um intervalo para cada transição. Se uma transição é habilitada de forma contínua durante um tempo mínimo, então ela pode disparar. Esse tempo mínimo é uma espécie de atraso no estado. Se a transição se mantiver sensibilizada por um tempo igual ao valor máximo do intervalo, então, ela deve disparar porque atingiu seu tempo máximo de sensibilização. O intervalo, portanto, indica uma incerteza na data de disparo da transição, caso ela permaneça sensibilizada de forma contínua.

Definição 2.15 (`Rede de Petri Temporal`) *Uma Rede de Petri Temporal é uma tripla* $\Gamma = (N, \boldsymbol{m}_0, \boldsymbol{i}^s)$*, sendo:*

- N, *uma RdP ordinária*
- $m_0 : P \rightarrow \mathbb{N}$, *a marcação inicial*
- $\mathbf{i}^s : T \rightarrow (\mathbb{Q}^+ \times (\mathbb{Q}^+ \bigcup \{\infty\})$, *função intervalos estáticos, sendo* $\mathbb{Q}^+$ *o conjunto dos números racionais positivos, juntamente com o zero*[1]

A função $\mathbf{i}^s$ associa a cada transição $t \in T$, um intervalo $\mathbf{i}^s(t) = [\delta^s(t), \Delta^s(t)]$, sendo $\delta^s(t)$ o tempo mínimo de sensibilização e $\Delta^s(t)$ o tempo máximo de sensibilização da transição t, e verificando as seguintes condições.

- $0 \leq \delta^s(t) \leq \Delta^s(t)$
- $0 \leq \delta^s(t) < \infty$
- $0 \leq \Delta^s(t) \leq \infty$
- $\begin{cases} \delta^s(t) \leq \Delta^s(t), \text{ se } \Delta^s(t) \neq \infty \\ \delta^s(t) < \Delta^s(t), \text{ se } \Delta^s(t) = \infty \end{cases}$

Se $\Delta^s(t)$ é finito, tem-se a semântica forte do tempo, pois, para um tempo de habilitação igual a $\Delta^s(t)$, a transição t deve disparar.

A habilitação de uma transição $t \in T$ deve satisfazer à condição 2.1. Porém, estando a transição habilitada, não significa que ela pode ser disparada. O disparo dessa transição só ocorrerá se for decorrido, no mínimo, um tempo de habilitação igual ao tempo mínimo de sensibilização $\delta^s(t)$. Suponha, por exemplo, que uma transição t foi sensibilizada no instante θ. O disparo de t não ocorrerá antes do instante $\theta + \delta^s$, e nem depois do instante $\theta + \Delta^s$. Assim, o disparo de t ocorrerá no intervalo $[\theta + \delta^s, \theta + \Delta^s]$. Durante esse período as marcas permanecem em seus lugares, podendo inclusive, serem removidas pelo disparo de outra transição. O disparo de uma transição é instantâneo (não consome tempo).

Nas RPTs, uma marcação não contém informação suficiente para descrever o comportamento do estado do sistema. O estado deve, também, incluir informações do tempo. Esta informação é dada por um relógio (*clock*) e é carregada em cada transição habilitada. A dupla <marcaçã, intervalos de disparo das transições habilitadas> se constitui em uma classe de estados.

[1] A escolha de $\mathbb{Q}^+$ diferencia da proposta inicial de Merlin e Faber [Mer74, MF76] e em [BD91], nas quais a função $\mathbf{i}^s$ está definida em $\mathbb{R}$ e $\mathbb{N}$, respectivamente.

A maior parte dos trabalhos de análise de RPT, via alcançabilidade, utiliza o conceito de classe de estados, que representa uma compactação de estados contínuos no tempo. As classes de estados podem ser representadas por fórmulas lógicas temporais lineares e foram introduzidas por [BM82, BD91]. A idéia é agrupar todos os estados, mantidos continuamente sob uma mesma marcação. Dessa forma, o uso das RPTs como formalismo para modelagem e análise de sistemas dependentes do tempo tornou-se mais atrativo e adequado para diferentes aplicações [Vic01, XHD02, BV95, MGV00, TYC95, WD00, LLK05].

Definição 2.16 (`Classe de Estado`) *Uma classe de estado de uma RPT é definida pelo par* $CS = \langle \mathtt{m}, D \rangle$*, sendo,*

- $\mathtt{m}$*, a marcação*
- D*, um conjunto de inequações que expressam os intervalos de disparos das transições habilitadas por* $\mathtt{m}$*. Esse conjunto é também chamado de domínio dos disparos dinâmicos.*

Mesmo utilizando o conceito de classe de estado, existem diferentes extensões para as RPTs. Cada qual com uma aplicação específica [BV95, TYC95, BBAC04, PZ91, LLK05].

2.2 Álgebra Intervalar

A álgebra intervalar é um ferramenta matemática introduzida por R. M. Moore no começo dos anos 60 [Moo95] e foi inicialmente utilizada para solucionar problemas relacionados com erros em computação científica. Nos últimos anos, entretanto, ela tem sido aplicada em outras áreas, tais como Robótica, Controle Robusto, Otimização, entre outras.

É apresentado a seguir o conjunto de intervalos e as operações básicas sobre seus elementos, bem como, alguns conceitos algébricos e topológicos da álgebra intervalar. Para uma maior compreensão recomenda-se consultar [JKDW95, HW04, Neu90, Moo95].

2.2.1 Intervalos

Um *intervalo* é definido como sendo o conjunto

$$\mathbf{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$$

sendo, $\underline{x}$ e $\overline{x}$ denominados limites inferior e superior do intervalo $\mathbf{x}$, respectivamente.

O espaço dos intervalos, $\mathbb{IR}$, é então,

$$\mathbb{IR} = \{[\underline{x}, \overline{x}] \mid \underline{x} \leq \overline{x},\ \underline{x}, \overline{x} \in \mathbb{R}\}$$

cada elemento de $\mathbb{IR}$ é chamado intervalo.

Sejam $\mathbf{x} = [\underline{x},\overline{x}]$, $\mathbf{y} = \left[\underline{y},\overline{y}\right]$ e $\mathbf{z} = [\underline{z},\overline{z}]$ intervalos.

A distância entre dois intervalos é definida como

$$\mathbf{d}_{dist}(\mathbf{x},\ \mathbf{y}) = \max\{|\overline{x} - \overline{y}|, |\underline{x} - \underline{y}|\}.$$

A interseção de dois intervalos $\mathbf{x}$ e $\mathbf{y}$ nem sempre é um intervalo e é definida como segue,

$$\mathbf{x} \cap \mathbf{y} = \{k \in \mathbb{R} \mid k \in \mathbf{x} \text{ e } k \in \mathbf{y}\}$$

Se $\mathbf{x} \cap \mathbf{y} \neq \emptyset \implies \mathbf{x} \cap \mathbf{y} = [\max\{\underline{x}, \underline{y}\}, \min\{\overline{x}, \overline{y}\}]$

A união de dois intervalos $\mathbf{x}$ e $\mathbf{y}$ nem sempre é um intervalo e é definida por

$$\mathbf{x} \cup \mathbf{y} = \{k \in \mathbb{R} \mid k \in \mathbf{x} \text{ ou } k \in \mathbf{y}\}$$

Para tornar o conjunto de intervalos fechado em relação à operação união, é definida a operação união intervalar, como segue,

A união intervalar de dois intervalos $\mathbf{x}$ e $\mathbf{y}$, denotada por $\sqcup$, é o menor intervalo que contém $\mathbf{x}$ e $\mathbf{y}$, como segue,

$$\mathbf{x} \sqcup \mathbf{y} = [\mathbf{x} \cup \mathbf{y}] = [\min\{\underline{x}, \underline{y}\}, \max\{\overline{x}, \overline{y}\}]$$

Estrutura Algébrica

`Operações básicas`:

$\mathbf{x} + \mathbf{y} = \left[\underline{x} + \underline{y},\ \overline{x} + \overline{y}\right]$

$\mathbf{x} - \mathbf{y} = \left[\underline{x} - \overline{y},\ \overline{x} - \underline{y}\right]$

$\mathbf{x} \cdot \mathbf{y} = \left[\min\{\underline{x}\,\underline{y},\ \underline{x}\,\overline{y},\ \overline{x}\,\underline{y},\ \overline{x}\,\overline{y}\},\ \max\{\underline{x}\,\underline{y},\ \underline{x}\,\overline{y},\ \overline{x}\,\underline{y},\ \overline{x}\,\overline{y}\right]$

$\mathbf{x} / \mathbf{y} = \left[\min\{\underline{x}/\,\underline{y},\ \underline{x}/\,\overline{y},\ \overline{x}/\,\underline{y},\ \overline{x}/\,\overline{y}\},\ \max\{\underline{x}/\,\underline{y},\ \underline{x}/\,\overline{y},\ \overline{x}/\,\underline{y},\ \overline{x}/\,\overline{y}\}\right],\ \ \mathbf{y} \neq 0$

`Propriedades`:

$$\mathbf{x} + \mathbf{y} = \mathbf{x} + \mathbf{x}$$
$$\mathbf{x} + (\mathbf{y} + \mathbf{z}) = (\mathbf{x} + \mathbf{y}) + \mathbf{z}$$
$$\mathbf{x} + 0 = \mathbf{x}$$
$$1 \cdot \mathbf{x} = \mathbf{x}$$
$$\mathbf{x}\,(\mathbf{y} + \mathbf{z}) \subseteq \mathbf{x} \cdot \mathbf{y} + \mathbf{x} \cdot \mathbf{z}$$

2.2.2 Vetores e Matrizes

- Um vetor intervalar $\mathbf{x} \in \mathbb{IR}^n$ é um vetor cujos n componentes (coordenadas), são intervalos $\mathbf{x}_i \in \mathbb{IR}$.

  O vetor intervalar pode ser denotado como:

  - produto cartesiano: $\mathbf{x} = \mathbf{x}_1 \times \mathbf{x}_2 \times \ldots \times \mathbf{x}_n$
  - vetor de componentes reais: $\mathbf{x} = (\mathbf{x}_1 \times \mathbf{x}_2 \times \ldots \times \mathbf{x}_n)^T$

  Um vetor real $\mathsf{x} \in \mathbb{R}^n$ é dito contido em $\mathbf{x}$ se, somente se, $\forall\, \mathsf{x}_i \in \mathbf{x}_i$ $(i = 1, \ldots, n)$.

  Operações com vetores:

  $$\alpha \cdot \mathbf{x} = \alpha\mathbf{x}_1 \times \alpha\mathbf{x}_2 \times \ldots \times \alpha\mathbf{x}_n$$
  $$\mathbf{x}^T \cdot \mathbf{y} = \mathbf{x}_1 \cdot \mathbf{y}_1 \times \ldots \times \mathbf{x}_n \cdot \mathbf{y}$$
  $$\mathbf{x} + \mathbf{y} = \mathbf{x}_1 + \mathbf{y}_1 \times \ldots \times \mathbf{x}_n + \mathbf{y}_n$$

- Uma matriz intervalar $\mathbf{A} \in \mathbb{IR}^{m \times n}$ é uma matriz cujos elementos são intervalos.

  Uma matriz real $\mathsf{A} \in \mathbb{R}^{m \times n}$, com elementos a_{ij}, é dita contida na matriz intervalar $\mathbf{A}$ $(\mathsf{A} \in \mathbf{A})$, se, somente se $a_{ij} \in \mathbf{A}_{i,j}$ para todo $i = 1, \ldots, m$ e $j = 1, \ldots, n$.

2.3 Conclusão

A inclusão de parâmetros temporais ampliam a capacidade de modelagem das Redes de Petri a sistemas cujo comportamento depende do tempo, tais como os sistemas tempo-real, os sistemas embarcados, entre outros. Neste capítulo foram apresentados os conceitos das Redes de Petri e das Redes de Petri Temporais, além de elementos básicos da Álgebra Intervalar, que serão utilizados nos próximos capítulos.

3 Análise das Redes de Petri Temporais Usando Tempo Global

Este capítulo apresenta a metodologia de tempo global [LLK06, LLK05] para o tratamento do tempo para a classe de Redes de Petri Temporais. Esta metodologia desenvolvida no Laboratório de Sistemas Inteligentes de Produção do CPGEI, é a base do desenvolvimento da ferramenta computacional apresentada neste trabalho.

3.1 Análise de Alcançabilidade Usando Tempo Global

O tempo global considera um relógio global que é disparado com o início da execução da rede e é incrementado pelo intervalo de tempo de disparo das transições. O uso do relógio global aparece em trabalhos que tratam o estado de forma densa, como em [YS97, Lil99], ou naqueles em que o estado é tratado de forma compacta, como em [XHD02, WD00, LLK05]. Em todos estes trabalhos existe uma função que, como um relógio global, acumula o tempo a partir do estado inicial de execução da rede.

A abordagem utilizada implementada na ferramenta se difere das demais que utilizam o tempo global, nos seguintes pontos:

- incorpora o tempo absoluto (global) aos intervalos dinâmicos de disparo das transições;
- as regras para verificar se uma transição habilitada é disparável ou não são aplicadas diferentemente às transições persistentes e recém-habilitadas;
- cria para o tempo global um grafo de classes limitado.

3.1.1 Classe de Estado de uma RPT com Tempo Global

A classe de estado de uma RPT é definida por sua marcação $\mathbf{m}$ e um vetor de intervalos de disparos, composto por transições habilitadas pela marcação $\mathbf{m}$.

Formalmente, uma classe de estado S de uma RPT, é definida pelo par $S = \langle \mathbf{m}, \mathbf{i} \rangle$, sendo:

- $\mathbf{m}$, a marcação atual da rede;
- $\mathbf{i}$, vetor de intervalos dinâmicos dos tempos globais de disparo das transições habilitadas pela marcação $\mathbf{m}$.

O vetor de intervalos de tempo $\mathbf{i}$ é diferente do domínio de disparos, definido para a classe de estado, porque os tempos são absolutos e definem os intervalos em que as transições devem disparar, caso não sejam desabilitadas pelo disparo de outra transição. Durante a evolução da rede o vetor de intervalos varia, tanto por causa do número de transições habilitadas pela marcação, quanto pelo instante de disparo de alguma das transições habilitadas. Na classe inicial S_0, o intervalo $\mathbf{i}_0$ é definido pelos intervalos estáticos das transições habilitadas nesta classe. Por exemplo, para a rede mostrada na figura 2, a classe inicial S_0 é caracterizada como segue:

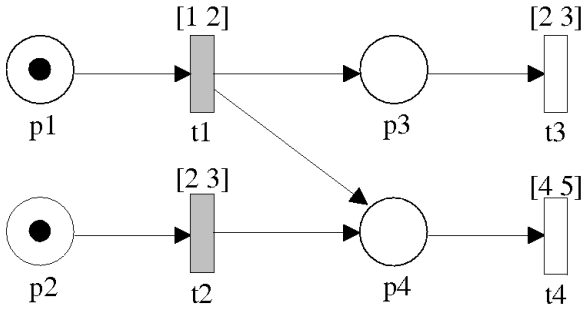

Figura 2: Rede de Petri Temporal

$$S_0 = \langle \mathbf{m}_0, \mathbf{i}_0 \rangle$$

$$\mathbf{m}_0 = [1\ 1\ 0\ 0]^T$$

$$\mathbf{i}_0 = \begin{bmatrix} \mathbf{i}^S(t_1) \\ \mathbf{i}^S(t_2) \end{bmatrix} = \begin{pmatrix} [1,\ 2] \\ [2,\ 3] \end{pmatrix}$$

3.1.2 Disparabilidade de um Conjunto de Transições

Para uma classe $S = \langle \mathtt{m}, \mathtt{i} \rangle$ de uma RPT, o conjunto de transições habilitadas pela marcação $\mathtt{m}$, é denotado por $\mathscr{H}$. Algumas das transições pertencentes a $\mathscr{H}$, podem não disparar na classe S. Isto ocorre devido à semântica forte do tempo. Assim, o fato de uma transição estar habilitada em uma determinada classe, não determina necessariamente que ela irá disparar nessa classe, pois uma outra transição poderá ser disparada antes do término do tempo mínimo de sensibilização da referida transição. Dessa forma, deve-se estabelecer uma condição de disparabilidade. Essa condição é expressa através da definição de transição disparável, como segue.

Definição 3.1 (`Transição Disparável`) *Seja $\mathscr{H}$, o conjunto de transições habilitadas em uma classe S de uma RPT. Uma transição t_j é dita disparável em S se, e somente se,*

$$\delta^s(t_j) \leq \min\{\Delta^s(t)\} \tag{3.1}$$

sendo, $t \in \mathscr{H}$ e

- $\delta^s(t_j)$ *o limite inferior do intervalo de tempo para t_j disparar;*
- $\Delta^s(t)$ *o limite superior do intervalo de disparo de cada transição $t \in \mathscr{H}$.*

sendo $\delta^s(t_j)$ e $\Delta^s(t)$ relativos ao início de execução da rede.

O conjunto das transições disparáveis em cada classe é denotado por $\mathscr{D}$.

3.1.3 Regra de Disparo e Mudança de Classe de Estado

Nesta abordagem considera-se que a mudança de classe de estado da rede ocorre devido ao disparo de alguma transição. O instante de disparo de uma transição varia com o limite superior mínimo das transições disparáveis na referida classe. Logo, o intervalo de disparo de uma transição pode diferir do intervalo dinâmico da transição. O intervalo de disparo de uma transição disparável é definido.

Definição 3.2 (`Intervalo de Disparo`) *Seja $t_j \in \mathscr{D}_0$ em uma classe S_0. O disparo de t_j, na classe S_0, ocorrerá no intervalo de disparo i^d, definido como segue.*

$$i^d(t_j) = \left[\delta_0(t_j),\, \min\{\Delta_0(t)\}\right] = \left[\delta^s(t_j),\, \min\{\Delta^s(t)\}\right] \tag{3.2}$$

sendo, $t \in \mathscr{H}_0$.

O intervalo de disparo, portanto, é limitado pelo instante em que a transição é sensibilizada para disparar e a condição de disparabilidade (3.1).

Seja $S_1 = \langle \mathfrak{m}_1, \mathbf{i}_1 \rangle$ uma classe alcançada, a partir de S_0, pelo disparo de t_j com o intervalo de $\mathbf{i}_0^d = [\delta^d, \Delta^d]$. Quando t_j dispara, a marcação da rede e o valor dos intervalos das transições habilitadas são alterados, como segue:

1. Remoção e adição de marcas nos lugares de entrada e saída da transição t_j, conforme equação (2.2);

2. Os intervalos de disparo das transições habilitadas após o disparo de t_j são atualizados. Isto ocorre de diferentes maneiras, como segue:

   (a) para transições persistentes, ou seja, transições habilitadas antes e depois do disparo de t_j, ou seja, $\forall\, t \neq t_j$ e t persistente, então,

   $$\mathbf{i}_1(t) = \Big[\max\{\delta^d, \delta^s(t)\},\, \Delta^s(t)\Big] \tag{3.3}$$

   (b) para transições habilitadas após o disparo de t_j, as chamadas recém-habilitadas, têm-se:

   $$\mathbf{i}_1(t) = \mathbf{i}^d + \mathbf{i}^s(t) \tag{3.4}$$

Na equação (3.3) nota-se que uma transição que persiste habilitada quando há uma mudança de classe, pode ter seu intervalo de disparo, na nova classe, alterado pelo intervalo da transição que disparou. Já pela equação (3.4), as transições que são recém-habilitadas, têm como intervalo de disparo a soma do seu intervalo estático e o intervalo de disparo da transição que leva a rede à nova classe. Caso a transição que disparou volte a ficar habilitada, então, ela será tratada como recém-habilitada.

As equações (3.3) e (3.4) possibilitam que o conjunto de transições habilitadas em S_1 seja separado em dois conjuntos disjuntos, dados por:

- conjunto das transições herdadas ou transições que permanecem habilitadas quando da mudança de classe, de S_0 para S_1, a partir do disparo de t_j, ou seja,

  $$\mathscr{P}_1 = \{t \mid t \in \mathscr{H}_1 \wedge (\mathscr{H}_0 \setminus \{t_j\})\} \tag{3.5}$$

- conjunto das transições novas ou transições recém habilitadas:

$$\mathscr{N}_1 = \{\mathscr{H}_1 \setminus \mathscr{P}_1\} \tag{3.6}$$

Nas equações (3.5) e (3.6) nota-se que, caso uma transição disparada em S_0 continue habilitada em S_1, ela será tratada como uma transição nova.

Considerando a rede mostrada na figura 2, e disparando a transição t_1 em qualquer instante pertencente ao intervalo [1, 2], a rede alcança a classe S_1, com as seguintes características:

$S_1 = \langle \mathtt{m}_1, \mathbf{i}_1 \rangle$;

$\mathtt{m}_1 = [0\ 1\ 1\ 1]^T$;

$\mathscr{H}_1 = \{t_2, t_3, t_4\}$, transições habilitadas;

$\mathbf{i}_1(t_2) = \mathbf{i}_0(t_2) = [1,\ 2]$, transição persistente;

$\mathbf{i}_1(t_3) = \mathbf{i}^d(t_1) + \mathbf{i}^s(t_3) = [1,\ 2] + [2,\ 3] = [3,\ 5]$, transição recém-habilitada;

$\mathbf{i}_1(t_4) = \mathbf{i}^d(t_1) + \mathbf{i}^s(t_4) = [1,\ 2] + [4,\ 5] = [5,\ 7]$, transição recém-habilitada;

$$\mathbf{i}_1 = \begin{pmatrix} \emptyset \\ [1,\ 2] \\ [3,\ 5] \\ [5,\ 7] \end{pmatrix};$$

$\mathscr{D}_1 = \{t_2\}$, transição disparável.

A verificação da condição de disparabilidade de uma transição, via intervalos dinâmicos com tempo global, nem sempre expressa a verdade. As operações com intervalos guardam ou expressam situações que não ocorrem no plano real. No caso das RPTs, isso pode levar a falsos cenários de execução de seqüências de transições. Para ilustrar, será considerada a RPT mostrada na figura 3.

Na classe inicial apenas a transição t_1 está habilitada, logo, seu disparo habilitará, no mesmo instante, as transições t_2 e t_3 e seus intervalos de disparo serão [6, 13] e [8, 16], respectivamente. Ambas estarão disparáveis, com intervalos de disparos iguais a [6, 13] para

t_3 e [8, 13] para t_2, como t_2 e t_3 compartilham o mesmo lugar de entrada, elas estariam em conflito, como mostrado na figura 4.

Assumindo que, por exemplo, o tempo de disparo de t_1 pertence ao intervalo [4, 10], em $\theta(t_1) = 10$, o intervalo de disparo da transição t_3 seria [12, 13], e o intervalo de disparo de t_2 seria [14, 16]. Portanto, para qualquer instante de disparo da transição t_2, apenas a transição t_3 será disparável e não haverá conflito entre as transições t_2 e t_3. Esta é uma situação em que a soma de dois intervalos mostra a existência de um conflito que na realidade não existe.

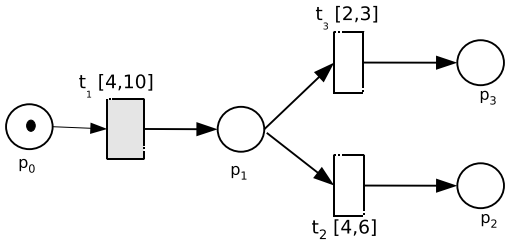

Figura 3: Exemplo de uma RPT para análise de tempo global

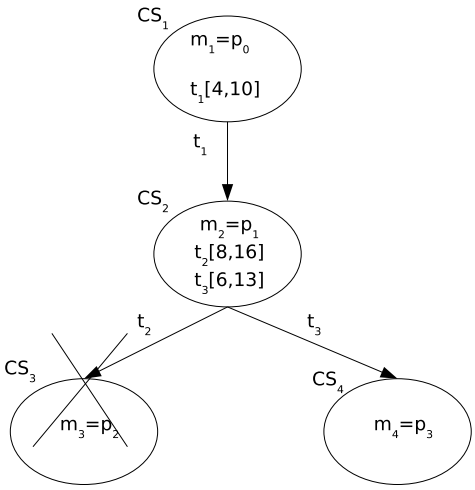

Figura 4: Grafo de classes para a rede da figura 3

Neste caso, a soma do intervalo [4, 10] com o intervalo [4, 6] gera um intervalo que considera o disparo de t_1 entre os instantes 4 e 10. Formalmente, a soma de dois intervalos $\mathbf{x}$ e $\mathbf{y}$ é definida como sendo:

$$\mathbf{x}+\mathbf{y} = \{x+y \mid x \in \mathbf{x}, y \in \mathbf{y}\}$$

Assim, aparecem situações como, por exemplo, t_3 considera que o disparo de t_1 ocorreu no instante 10, logo o intervalo de t_3 seria [12, 13]. Enquanto para t_2, o disparo de t_1

ocorreu no instante 6, por exemplo, logo o intervalo de disparo de t_2 seria $[10, 12]$. Como $[12, 13] \cap [10, 12] \neq \emptyset$, ambas são disparáveis [1].

Para evitar que uma transição não disparável seja incluída entre as disparáveis e com isso levar à validação de falsas execuções, como no exemplo anterior, a definição de transição disparável (3.1) é redefinida, separando as transições persistentes das recém-habilitadas. Assim, uma transição $t_j \in \mathscr{H}$, em uma classe S de uma RPT, é considerada disparável se, e somente se,

$$\begin{aligned} \delta^s(t_j) &\leq \Delta^s(t), \quad \text{para } t \in \mathscr{N} \\ \delta(t_j) &\leq \Delta(t), \quad \text{para } t \in \mathscr{P} \end{aligned} \tag{3.7}$$

Para t_j ser disparável ela tem que atender duas condições de disparabilidade, uma no seu domínio estático, outra no domínio dinâmico. No exemplo, portanto, usando a condição (3.7), verifica-se que a transição t_2 é disparável pelo seu intervalo dinâmico, mas não o é pelo seu intervalo estático.

Além da necessidade de verificar duas condições de disparabilidade, a análise usando tempo global pode ter um grafo de classes ilimitado, mesmo sendo a rede cíclica e limitada, como mostra a figura 5.

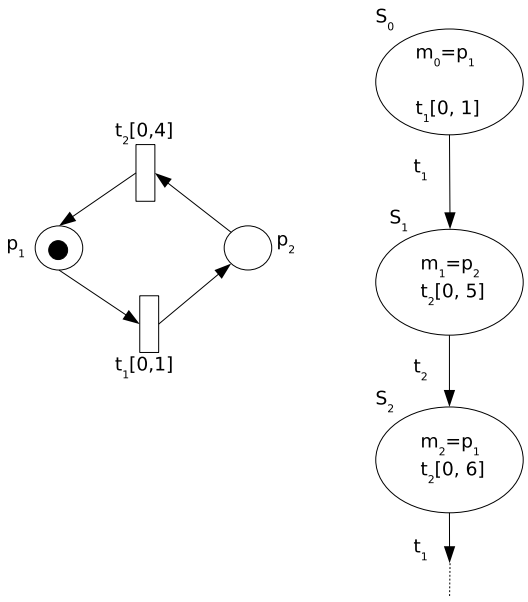

Figura 5: Exemplo de um grafo de classes ilimitado

[1] O fato de existir uma interseção entre os intervalos, significa que a condição de disparabilidade é satisfeita.

3.2 Equivalência de Estados

Um dos problemas dos métodos enumerativos usando tempo global é a geração ilimitada de classes, pois sendo o tempo global, as classes sempre terão seus domínios temporais ampliados.

Na figura 6 é apresentada uma RPT e seus grafos de alcançabilidade: quando a é 'não temporizada', figura 6a, e considerando as informações temporais, figura 6b. Para a rede com informação temporal e com marcação $2p_3$, pode ser alcançado por diferentes intervalos de tempo. Disparando a seqüência $\sigma_1 = t_1, t_2, t_3$, a partir da classe S_0, a rede alcança $S_1 = \langle 2p_3, [3,6] \rangle$, enquanto o disparo da seqüência $\sigma_2 = t_1, t_3, t_2$ leva a $S_2 = \langle 2p_3, [3,7] \rangle$, logo $S_1 \neq S_2$.

Sob o ponto de vista lógico, as classes S_1 e S_2 são iguais, visto que têm a mesma marcação, porém S_2 pode ser alcançada em 7 unidades de tempo, enquanto S_1, é alcançada, no máximo, em 6 unidades de tempo. Caso o interesse seja a classe em que apenas o lugar p_3 tenha marca, pode-se considerar que as classes S_1 e S_2 são equivalentes, pois $\mathbf{m}_1 = \mathbf{m}_2$ e $\mathbf{i}_1 \subseteq \mathbf{i}_2$. Logo, S_1 e S_2 passariam a ser uma só classe, conforme a figura 6(b).

Definição 3.3 (`Classes Equivalentes`) *Sejam $S_j = \langle m_j, i_j \rangle$ e $S_k = \langle m_k, i_k \rangle$, duas classes alcançáveis de uma RPT. S_j e S_k são equivalentes se, somente se,*

$$\begin{cases} m_j = m_k \\ i_j \subseteq i_k \\ \mathscr{H}_j = \mathscr{H}_k \end{cases} \tag{3.8}$$

3.2.1 Rede Reversível ou Cíclica

Um caso especial de equivalência ocorre quando a rede é repetitiva. Neste caso, a cada ciclo da rede tem-se classes com mesma marcação e com intervalos de disparo proporcionais, como definido a seguir.

Definição 3.4 (`Vetor Intervalos Equivalentes`) *Seja uma RPT cíclica com período de repetição $d \in \mathbb{IQ}^+$. Dadas duas classes $S_0 = \langle m_0, i_0 \rangle$ e $S_n = \langle m_n, i_n \rangle$ separadas por k ciclos e sendo $m_0 = m_n$, os intervalos de disparos i_0 e i_n são relacionados da seguinte forma*

$$i_k = i_0 + dk \quad k = 0, 1, \ldots \tag{3.9}$$

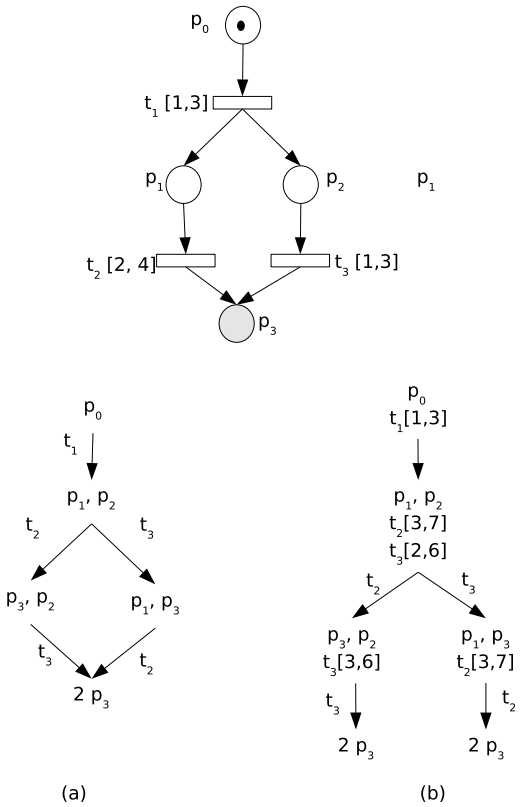

Figura 6: Grafo de alcançabilidade (a) rede não temporizada (b) rede temporal

sendo $\boldsymbol{d}$ *o período do ciclo*

Exemplo 1 *Seja* q_1 *o vetor de disparo tal que*

$$S_0 \xrightarrow{q_1} S_1.$$

sendo $S_1 = \langle m_0, i_1 \rangle$ *Assim,*

$$i_1 = i_0 + d$$

e $d = i_1 \ominus i_0$

Para $S_2 = \langle m_1, i_2 \rangle$

$$i_2 = i_1 + d = i_0 + 2d$$

$$\vdots$$

$$i_k = i_0 + d.k$$

Assim, as classes com mesma marcação e com vetores intervalos de disparos proporcionais, são equivalentes e o termo $\mathbf{d}k$ pode ser assumido como um vetor intervalar de disparo

associado à transição t_j, cujo disparo leva a S_n, tal que $\mathfrak{m}_0 = \mathfrak{m}_k$ ou seja,

$$S_0 \xrightarrow{t_j} S_k.$$

e,

$$i^d(t_j) = \mathbf{d}k$$

A definição de classes equivalentes para uma rede cíclica evita que, mesmo para uma rede segura, a análise via tempo global leve a um espaço de estado finito.

3.3 Métodos de Redução para Rede de Petri Temporal

3.3.1 Análise das RPTs via Alcançabilidade

A análise via alcançabilidade é baseada na geração do grafo de alcançabilidade para um modelo formal do sistema. Os vértices do grafo de alcançabilidade são todos os estados alcançáveis a partir do estado inicial, e as arestas são as transições que levam de um estado a outro. Se o sistema tem um espaço de estado finito, então, ele pode ser gerado automaticamente [Val94, God96].

Entretanto, muitos sistemas, apesar de terem um espaço de estado finito, são intratáveis devido ao elevado número de estados. Em alguns casos, o tamanho do espaço de estado pode crescer exponencialmente em relação ao tamanho do sistema. Esse problema é conhecido como explosão do número de estados [God96].

Uma maneira de resolver o problema da explosão do número de estados é através da análise parcial de alcançabilidade, que possibilita uma análise automática, rápida e eficiente. Existem vários métodos que permitem esta análise reduzindo, de maneira inteligente, o espaço de busca a partir da exploração de estruturas padrões dos sistemas, tais como: concorrência ou paralelismo, sincronização, entre outras.

Existem várias maneiras de agrupar esses métodos, uma delas é agrupá-los em dois grandes grupos: os métodos de redução do espaço de estado baseados na semântica de ordem parcial e os métodos de redução do modelo, inspirados nas regras de redução para as Redes de Petri não temporizadas. No primeiro grupo estão os algoritmos de análise que verificam determinadas propriedades da rede, sem explorar todo o espaço de estado. Nesse grupo estão os métodos que exploram particularidades dos sistemas, tais como, independência entre transições, conjuntos de transições persistentes, planificação da rede,

entre outras [God96, Val92, WG93, SB97, DBDS94]. No segundo grupo, estão os métodos de redução que estenderam as regras que permitem agrupar transições e lugares da rede, mantendo propriedades como: limitação, segurança e vivacidade no modelo reduzido [SB96, Ber87, JM92, BB02]. Em ambos os grupos se consegue uma redução considerável no tamanho do espaço de estados, tornando tratável a análise via alcançabilidade.

Na próxima seção, é apresentado um algoritmo que usa elementos dos métodos de ordem parcial para obter uma rede reduzida. Este algoritmo explora estruturas regulares da rede e através de agrupamento de lugares e transições obtem uma rede reduzida, representada por uma forma matricial de suas transições. Além do agrupamento das estruturas concorrentes, que permite gerar uma rede equivalente sem ou com poucas transições persistentes, a rede reduzida possibilita a análise via alcançabilidade com o número de estados bastante reduzido em relação à rede original.

3.3.2 Método de Redução via Relação de Ordem

O princípio básico desse método é garantir que, se existe duas ou mais possibilidades de ocorrências concorrentes em uma RPT, elas podem ser representadas por uma única ocorrência, desde que sejam mantidas as condições lógicas e temporais da rede. Dessa forma, pode-se limitar o número de ocorrências redundantes e transformar a rede original num modelo equivalente bem mais simples.

Uma outra idéia na qual o método se baseia é que a ocorrência de uma transição depende, fundamentalmente, da ocorrência dos seus antecessores. Agrupando as transições da rede, segundo sua ordem de precedência, o método ordena todas as seqüências de transições originadas nas transições inicialmente habilitadas.

Explorando esses dois princípios o algoritmo produz uma rede reduzida, que capta e agrupa estruturas da rede que podem ser executadas concorrentemente, respeitando as dependências entre ocorrências. Essas estruturas são identificadas através de uma matriz de transições que apresenta, de forma ordenada, todas as transições da rede original.

Toda transformação obtida é baseada no seguinte ponto de vista: se em uma RPT existir dois estados S_1 e S_2, tal que, S_2 é alcançável a partir de S_1, então existe um caminho equivalente entre S_1 e S_2 e que contém todos os demais caminhos.

Baseado nesse ponto de vista, é construído uma estrutura algébrica, como segue:

Matriz de Transições

A matriz de transições relaciona as transições da rede, segundo suas relações de precedência na estrutura. A matriz de transições é obtida a partir do conceito de matriz de caminhos da teoria de grafos orientados [LL04] e sua redução a partir de técnicas que exploram a existência do paralelismo e do conflito estrutural em uma RPT.

Para gerar a matriz de transições, a rede é tratada como sendo uma rede acíclica [RK04]. Para isso, serão formadas algumas estruturas, explicadas nas subseções seguintes.

Conjunto de Transições

Sejam T e P, os conjuntos de transições e lugares de uma RPT, respectivamente. Os conjuntos T e P podem ser particionados em subconjuntos disjuntos, segundo a ordem de precedência das transições e dos lugares na rede. Assim, os seguintes pares de conjuntos serão formados:

- $P_0 = \{p \in P \mid {}^\bullet p = \emptyset\}$
  $T_0 = \{t_{0_i} \in T \mid {}^\bullet t_{i_0} \subseteq P_0\}$ ou $T_0 = \{t_{i_0} \in T \mid Pre(t_{0_i}) \leq \mathtt{m}({}^\bullet t_{0_i})\}$
- $P_1 = \{p \in P \setminus P_0 \mid {}^\bullet p \subseteq T_0\}$
  $T_1 = \{t_{1_i} \in T \setminus T_0 \mid \exists (t_{0_i}, t_{1_i}),\ t_{0_i} \precsim t_{1_i},\ t_{0_i} \in T_0\}$
- $P_2 = \{p \in P \setminus \{P_0 \bigcup P_1\} \mid {}^\bullet p \subseteq T_1\}$
  $T_2 = \{t_{i_2} \in T \setminus \{T_1 \bigcup T_2\} \mid \exists (t_{0_i}, t_{1_i}, t_{2_i}),\ t_{0_i} \precsim t_{1_i} \precsim t_{2_i},\ t_{0_i} \in T_0, t_{i_1} \in T_1\}$

$\vdots$

- $P_n = \{p \in P \setminus \bigcup_{k=0}^{k=n-1} P_k \mid {}^\bullet p \subseteq T_{n-1}\}$
  $T_n = \{t_{n_i} \in T \setminus \bigcup_{k=0}^{k=n-1} T_k \mid$
  $\exists (t_{0_i}, \ldots, t_{n_i}),\ t_{0_i} \precsim \cdots, t_{n_i},\ t_{0_i} \in T_0, \cdots, t_{n-1_i} \in T_{n-1}\}$

sendo, $t_a \precsim t_b$ a ordem de precedência entre as transições t_a e t_b, isto significa que, t_a precede t_b, ou seja, $t_a^\bullet \cap {}^\bullet t_b \neq \emptyset$.

Para uma rede marcada, P_0 é o conjunto dos lugares com marcas, enquanto que T_0 é conjunto das transições habilitadas pela marcação inicial ou, é o conjunto das transições fonte, caso $P_0 = \emptyset$ [Mur89].

O conjunto T_n contém todas as conexões entre cada transição fonte ou transição inicialmente habilitada e todas as suas transições suscedentes. Portanto, se há um caminho entre uma transição t pertencente ao conjunto T_0 e uma transição objetivo, ou transição que leva a uma marcação desejada, esse caminho será composto por elementos de T_n.

Cardinalidade e Enumeração dos Conjuntos de Transições

A cardinalidade de cada conjunto T_k $(k = 0, \ldots, n)$, é indicada pelo seu número de transições, ou o número de *úplas* que ele contém e é denotada por $\sharp T_k$. Enquanto a cardinalidade de T é igual ao número de transições da rede.

$$T_k = \{\underbrace{(t_{01}, \ldots, t_{k1})}_{T_{k1}}, \ldots, \underbrace{(t_{0i}, \ldots, t_{mk})}_{T_{km}}\}$$

$$T = T_1 \cup T_2 \cup \ldots T_n$$

Dentre os conjuntos T_k, o conjunto T_0 é o mais precedente de todos, ou seja, os elementos desse conjunto precedem estruturalmente todos os demais elementos de T. Na outra extremidade da ordem de precedência, está o conjunto T_n, cujos os elementos são os mais suscedentes de todos.

Baseado na ordem de precedência, pode-se enumerar os elementos dos conjuntos T_k. Qualquer elemento dos subconjuntos $T \setminus \{T_0, T_n\}$, são ordenados segundo sua posição na *úpla*. Porém, se um determinado elemento pertence a uma *úpla* do conjunto T_n e esse mesmo elemento pertence a uma *úpla* de um conjunto $T \setminus T_n$, então esse elemento será ordenado segundo a sua ordem em T_n[2].

Para ordenar os conjuntos T_k uma função f é definida, como segue:

$$f : T \longrightarrow \mathbb{N} \begin{cases} k = 0: & f(t_{01}) = 1, & f(t_{02}) = 2, & \cdots, & f(t_{0m}) = m \\ & & \cdots & & \\ k = n: & f(t_{n1}) = p - r, & f(t_{n2}) = p - r + 1, & \cdots, & f(t_{nr}) = p \end{cases}$$

sendo $m = \sharp T_0$, $r = \sharp T_n$ e $p = \sharp T$.

[2]Para evitar a busca pela ordem de elementos pertencentes a mais de um conjunto T_k, deve-se começar a enumeração a partir dos elementos de T_n.

Matriz Parcial

Cada subconjunto T_k, com suas transições já ordenadas, é transformado em uma matriz com número de linhas igual à cardinalidade de T_k e o número de colunas igual à cardinalidade de T. A posição das transições nas colunas seguem a ordem de precedência na rede, ou seja, se t_1 ocupa a coluna 2 e t_6 ocupa a coluna 4, então $t_1 \precsim t_6$ ou entre t_1 e t_6 não há dependência. A matriz parcial é, então, definida como segue.

Definição 3.5 *(`Matriz Parcial`) Seja uma RPT e T o conjunto de suas transições. Para cada conjunto $T_k \subseteq T$ e $T_k = \{\underbrace{(t_{01},\ldots,t_{k1})}_{T_{k1}},\ldots,\underbrace{(t_{0i},\ldots,t_{km})}_{T_{km}}\}$ tem-se uma matriz parcial M_k, booleana e de ordem $r \times n$, sendo r a cardinalidade do conjunto T_k e n a cardinalidade de T. Os elementos da matriz M_k são definidos pela seguinte relação,*

$$M_k(i,j) = \begin{cases} 1, & \textit{se } t_i \precsim t_j \\ 0, & \textit{caso contrário} \end{cases} \tag{3.10}$$

sendo, $t_i, t_j \in T_{ki},\ i = 1,\ldots,r$

Matriz de Transições

A matriz de transições é composta pelas matrizes parciais da rede. A matriz terá ordem $m_T \times n$, sendo, m_T a soma das cardinalidades dos T_k conjuntos e n a cardinalidade de T. Logo, a matriz de transições é obtida como segue,

$$\mathtt{M}_T = \begin{bmatrix} \mathtt{M}_0 \\ \mathtt{M}_1 \\ \vdots \\ \mathtt{M}_n \end{bmatrix} \tag{3.11}$$

A matriz $\mathtt{M}_T$ apresenta todas as relações de precedência entre as transições da rede. Se a matriz de transições apresenta duas ou mais linhas, nas quais os últimos elementos pertencem à mesma coluna, então estas linhas podem ser agrupadas, pois representam diferentes ramificações para se chegar a uma mesma transição. Após o agrupamento a linha resultante conterá todas as precedências dessa transição.

Após todos os agrupamentos possíveis das linhas, a matriz assume a forma quadrática de ordem igual ao número de transições da rede e nessa forma passa a ser um caso particular da

matriz fecho transitivo, definida na teoria dos grafos para representar relações de precedência [LL04].

Redução da Matriz de Transições

A matriz M_T será submetida a um processo de redução, agrupando linhas e colunas que representam transições independentes ou conflitantes. O objetivo é transformar a matriz M_T numa matriz reduzida equivalente, em que suas linhas representam junções de caminhos que levam ao mesmo objetivo. O processo de redução utiliza o seguinte procedimento:

1. Repetir o passo 2, enquanto as condições forem atendidas.
2. Variar i de 1 a n:
   (a) Se $\mathtt{M}_T(i+1,i) = 0$
   (b) Verificar se existe precedente comum, CASO SIM:
      i. Agrupar as linhas e colunas i e $i+1$ de M_T, de modo que, para $j = 1,2,\ldots,n$:
         A. Somar linha i com $i+1$ Se $\mathtt{M}_T(i,j) = \mathtt{M}_T(i,j) + \mathtt{M}_T(i+1,j) \geq 1$, fazer $\mathtt{M}_T(i,j) = 1$.
         B. Somar coluna i com $i+1$ Se $\mathtt{M}_T(j,i) + \mathtt{M}_T(j,i+1) \geq 1$, fazer $\mathtt{M}_T(j,i) = 1$.
      ii. Anular linha e coluna *i+1*
3. Fazer $\mathtt{M}_I = \mathtt{M}_T$.
4. Verificar se t_a e t_b podem ser agrupadas

   Se $\mathbf{i}(t_a) \cap \mathbf{i}(t_b) \neq \emptyset$

   - Se $t_c \precsim t_a$ e $t_c \precsim t_b$
     - Se $\mathtt{Pre}(t_a) = \mathtt{Pre}(t_b)$ e $^\bullet(\mathtt{Pre}(t_a)) =^\bullet (\mathtt{Pre}(t_b)) = t_c$, então, $t_{ab} = t_a \vee t_b$ e,

       $$\mathbf{i}(t_{ab}) = [\min\{\delta(t_a), \delta(t_b)\}, \min\{\Delta(t_a), \Delta(t_b)\}]$$

     - Se $\mathtt{Pre}(t_a) \cap \mathtt{Pre}(t_b) \neq \emptyset$
       Transformar a estrutura
     - Se $\mathtt{Pre}(t_a) \cap \mathtt{Pre}(t_b) = \emptyset$

(a) Se ${}^\bullet(\mathtt{Pre}(t_a)) =^\bullet (\mathtt{Pre}(t_b)) = t_c$

* Se $(\mathtt{Pos}(t_a))^\bullet = (\mathtt{Pos}(t_b))^\bullet = t_d$, então, $t_{ab} = t_a \parallel t_b$ e,

$$\mathbf{i}(t_{ab}) = [\max\{\delta(t_a), \delta(t_b)\},\ \max\{\Delta(t_a), \Delta(t_b)\}]$$

* Caso contrário, então, $t_{ab} = t_a \parallel t_b$ e,

$$\mathbf{i}(t_{ab}) = [\min\{\delta(t_a), \delta(t_b)\},\ \max\{\Delta(t_a), \Delta(t_b)\}]$$

(b) Se $t_a,\ t_b \in T_0$, então, $t_{ab} = t_a \parallel t_b$ e,

$$\mathbf{i}(t_{ab}) = [\min\{\delta(t_a), \delta(t_b)\},\ \max\{\Delta(t_a), \Delta(t_b)\}]$$

- Se $\precsim = t_a \precsim t_b = \emptyset$
  - Se $t_a,\ t_b \in T_0$, então, $t_{ab} = t_a \parallel t_b$ e,

$$\mathbf{i}(t_{ab}) = [\min\{\delta(t_a), \delta(t_b)\},\ \max\{\Delta(t_a), \Delta(t_b)\}]$$

Exemplo 3.1 *Seja uma RPT cujas úplas geradas por cada subconjunto do conjunto de transições, são como segue:*

$$T_0 = \{t_1\}$$
$$T_1 = \{(t_1, t_2),\ (t_1, t_3),\ (t_1, t_5)\}$$
$$T_2 = \{(t_1, t_2, t_4),\ (t_1, t_3, t_5),\ (t_1, t_5, t_6)\}$$
$$T_3 = \{(t_1, t_2, t_4, t_6),\ (t_1, t_3, t_5, t_6)\}$$

A partir das matrizes parciais $\mathtt{M}_k$, geradas pelos conjuntos acima, forma-se a matriz de transições $\mathtt{M}_T$.

$$\mathtt{M}_T = \left(\begin{array}{l} \mathtt{M}_0 = \left\{ \begin{array}{cccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{array} \right. \\ \mathtt{M}_1 = \left\{ \begin{array}{cccccc} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \end{array} \right. \\ \mathtt{M}_2 = \left\{ \begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{array} \right. \\ \mathtt{M}_3 = \left\{ \begin{array}{cccccc} 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{array} \right. \end{array} \right), \quad \mathtt{M}_T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

A matriz de transições apresenta todas as relações de precedência entre as transições da rede. Aplicando o algoritmo de redução, apresentado na seção anterior, obtém-se a matriz de transições com todas as relações de precedência entre as transições.

Na matriz $\mathtt{M}_T$ as transições t_2 e t_3, atendem ao requisito para serem agrupadas, pois ambas têm mesma precedência, ou seja,

$t_1 \precsim t_2$ e $t_1 \precsim t_3$ ou $t_1 = {}^\bullet t_2 = {}^\bullet t_3$

As transições t_2 e t_3 só dependem, da transição t_1. Essas duas transições podem ser agrupadas, pois são independentes.

Entretanto, as transições t_4 e t_5, apesar de verificarem a primeira condição para serem agrupadas, uma não depende da outra, elas não possuem mesma precedência, ou seja,

$t_1 \precsim t_4$ e $t_3 \precsim t_5$

As transições t_4 e t_5 podem até serem independentes, mas não temporalmente.

Após o agrupamento das transições t_2 e t_3, a matriz tem sua ordem reduzida. A enumeração das transições é então atualizada e o novo processo de redução da rede é verificado.

Na segunda verificação, as transições t_4 e t_5 passam a atender as condições de agrupamento, pois a fusão das transições t_2 e t_3 passa a ser precedente dessas transições.

Após fundir t_4 com t_5 a matriz $\mathtt{M}_T$ assume a forma $\mathtt{M}_I$, como segue:

$$\mathtt{M}_T = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \qquad \mathtt{M}_I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Substituindo em $\mathtt{M}_I$, os elementos de valor 1 pelos respectivos intervalos estáticos equivalentes, tem-se a matriz $\mathbf{C}$ dos intervalos da rede reduzida equivalente.

3.4 Aplicação

Na figura 7 é mostrada uma RPT cíclica e seu grafo de alcançabilidade é mostrado na figura 8.

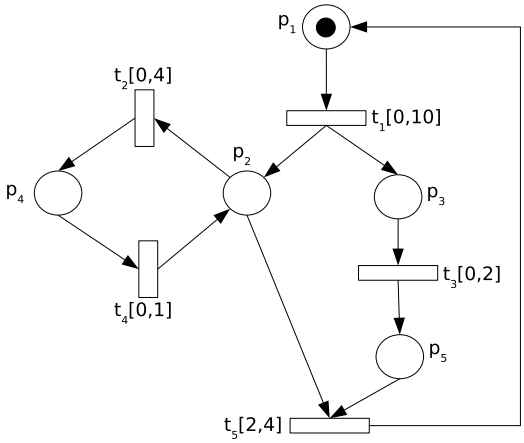

Figura 7: RPT para Aplicação

Não assumindo a equivalência entre classes, a rede mostrada na figura 7, gera um espaço de classes infinito, mostrado, parcialmente na figura 8. Porém, considerando a equivalência entre classes que têm mesma marcação e possuem intervalos que guardam uma relação de inclusão, obtém-se um grafo equivalente finito, capaz de gerar todos as classes da rede. Para o grafo mostrado na figura 8, foram identificadas as seguintes classes equivalentes:

- $S_1 \cong S_4$

$$\begin{cases} \mathtt{m}_1 = \mathtt{m}_4 \\ \mathbf{i}_1 \subseteq \mathbf{i}_4 \\ \mathscr{H}_4 = \mathscr{H}_1 \end{cases}$$

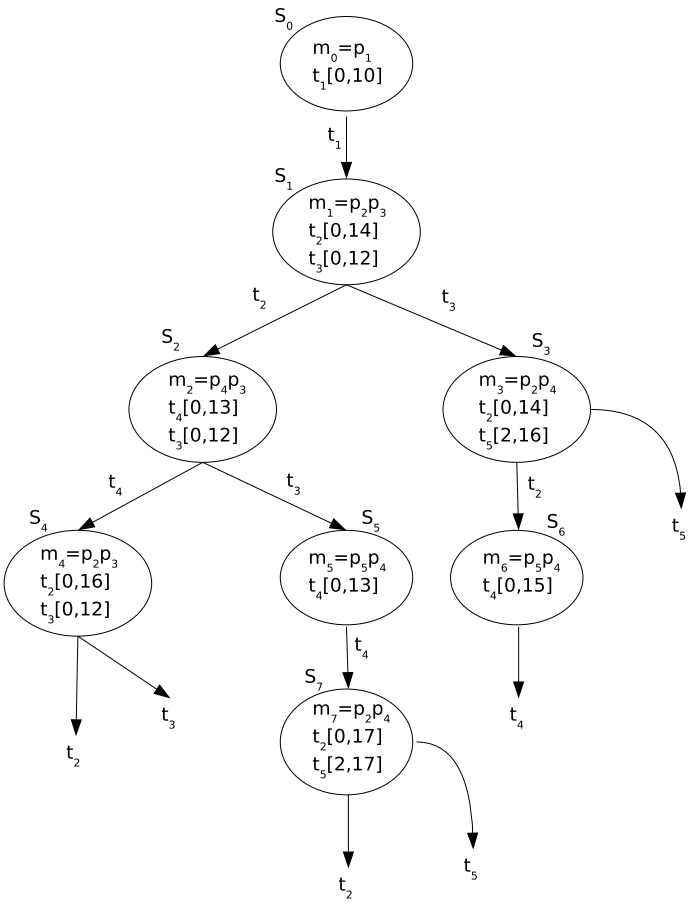

Figura 8: Grafo de alcançabilidade de classes para a rede da figura 7

Na classe S_4, disparando a transição t_2, a rede assume a mesma marcação que S_1 e o intervalo de disparo $\mathbf{i}_4$ pode ser escrito em função de $\mathbf{i}_1$. Na realidade a sub-rede formada por $p_2t_2p_4t_4$ forma uma rede cíclica, logo, a equação (3.9) relaciona $\mathbf{i}_4$ e $\mathbf{i}_1$, como segue:

$$\mathbf{i}_4 = \mathbf{i}_0 + \mathbf{d}k_1 = \begin{pmatrix} [0,\ 14] \\ [0,\ 12] \end{pmatrix} + \underbrace{\begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix}}_{\mathbf{i}^d(t_4)} k_1$$

sendo $k_1 = 0, 1, 2, \ldots$.

A classe S_4 pode ser agrupada a S_1 e a transição t_4 que alcança S_4 a partir da classe S_2, terá como intervalo de disparo o vetor intervalar $\mathbf{d} = \begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix}$

- $S_3 \cong S_7$.

$$\begin{cases} \mathbf{m}_3 = \mathbf{m}_7 \\ \mathbf{i}_3 \subseteq \mathbf{i}_7 \end{cases}$$

A classe S_7 pode, ser gerada a partir de S_3 pela seguinte relação:

$$\mathbf{i}_7 = \mathbf{i}_3 + \mathbf{d}k_2 = \begin{pmatrix} [0,\ 14] \\ [2,\ 16] \end{pmatrix} + \begin{pmatrix} [0,\ 3] \\ [0,\ 1] \end{pmatrix} k_2$$

sendo $k_2 = 0, 1, 2, \ldots$.

Assim, o disparo de t_4 para levar a rede de S_5 para a classe S_7 (agora S_3), é igual ao valor $\mathbf{d}$, ou seja,

$$\mathbf{i}_5^d(t_4) = \begin{pmatrix} [0,\ 3] \\ [0,\ 1] \end{pmatrix}$$

- $S_5 \cong S_6$

Semelhantemente, as classes S_5 e S_6 têm seus intervalos relacionados , como segue:

$$\mathbf{i}_6 = \mathbf{i}_5 + \mathbf{d}k_1 = [0,\ 13] + [0,\ 2]\, k_2$$

O intervalo de disparo de t_2, para levar S_3 à classe S_6 (agora S_3), assume o valor de $\mathbf{d}$, ou seja,

$$\mathbf{i}_5^d(t_4) = \begin{pmatrix} [0,\ 3] \\ [0,\ 1] \end{pmatrix}$$

- Ao agrupar a classe S_7 à classe S_3, há duas alternativas para alcançar a classe S_0, , a partir dessas classes. Essas duas alternativas são fundidas, porém, preservando as restrições de tempo de cada uma, conforme mostrado na figura 9.

Assim, o grafo de alcançabilidade que teria dimensão ilimitada, devido à presença de ciclos internos, passa a ter uma estrutura limitada e ser capaz de gerar todo o espaço de estados da rede, como mostrado na figura 9. Por exemplo, analisando a seqüência $\sigma_1 = t_1 t_2 t_4 t_3 t_5$ tem-se as seguintes classes:

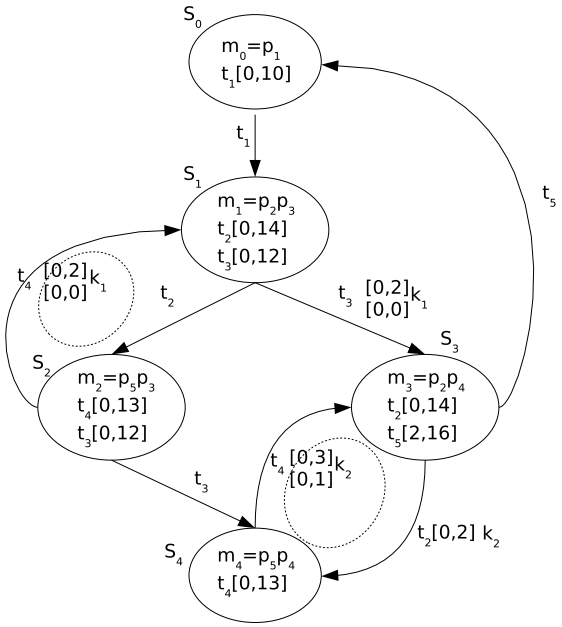

Figura 9: Grafo de alcançabilidade equivalente à rede da figura 7. As transições de classe onde não aparece o intervalo de disparo da transição é porque obedece a regra geral de transição de classe 3.2

1. $S_0 = \begin{cases} \mathtt{m}_0 = p_1 \\ \mathbf{i}_0(t_1) = [0,\ 10] \end{cases}$

   Só há uma transição habilitada e seu disparo ocorrerá no intervalo $\mathbf{i}^d = [0,\ 10]$

2. $S_1 = \begin{cases} \mathtt{m}_1 = p_2, p_3 \\ \mathbf{i}_1 = \begin{pmatrix} \mathbf{i}(t_2) \\ \mathbf{i}_1(t_3) \end{pmatrix} = \begin{pmatrix} [0,\ 14] \\ [0,\ 12] \end{pmatrix} \end{cases}$

   Nessa classe há duas transições habilitadas e ambas podem disparar. O intervalo de disparo de cada uma é $\mathbf{i}^d = [0,\ 12]$. Disparando t_2, tem-se,

3. $S_2 = \begin{cases} \mathtt{m}_2 = p_3, p_4 \\ \mathbf{i}_2(t_3) = [0,\ 12] \\ \mathbf{i}(t_4) = [0,\ 13] \end{cases}$

   Disparando t_4, seu intervalo de disparo será $\mathbf{i}^d = [0,\ 12]$ e carregará a informação $\begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix} k_1$ a ser acrescida à classe S_1, de modo a formar o estado S_1^1, como segue,

4. $S_1^1 = \begin{cases} \mathtt{m}_1 = p_2, p_3 \\ \mathbf{i}_1^1 = \mathbf{i}_1 + \begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix} = \begin{pmatrix} [0,\ 16] \\ [0,\ 12] \end{pmatrix} \end{cases}$

   O disparo de t_3 ocorrerá no intervalo $i^d = [0,\ 12]$ e carregará a informação $\begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix} k_1$ a ser acrescida ao estado S_3. Assim,

5. $S_3^1 = \begin{cases} \mathtt{m}_3 = p_2, p_4 \\ \mathbf{i}_3^1 = \begin{pmatrix} \mathbf{i}(t_2) \\ \mathbf{i}(t_5) \end{pmatrix} = \mathbf{i}_3 + \begin{pmatrix} [0,\ 2] \\ [0,\ 0] \end{pmatrix} k_1 = \begin{pmatrix} [0,\ 16] \\ [2,\ 16] \end{pmatrix} \end{cases}$

   A transição t_5, portanto, irá disparar no intervalo de tempo $[2,\ 16]$, para $k_1 = 1$ contado a partir do início da execução da rede.

3.5 Conclusão

Neste capítulo foi apresentado o conceito de tempo global para análise de RPTs e um método enumerativo que permite avaliar os limites de tempo decorrido entre ocorrências de uma seqüência de transições em uma RPT. Usando elementos da álgebra intervalar e tratando o tempo de forma absoluta, este método possibilita reduzir a complexidade da enumeração das classes em relação a outros métodos enumerativos, e também possibilita a enumeração de um grafo de classes finito quando o tempo global é usado para análise.

4 Projeto da Ferramenta Computacional

Muitas ferramentas computacionais foram desenvolvidas com base no formalismo das Redes de Petri e suas diversas classes. Algumas proporcionam ao usuário a criação de modelos para análise em uma classe específica e outras permitem criar e analisar modelos em mais de uma classe. A maior parte das ferramentes existentes trabalha com as classes Lugar/Transição e Temporizada, cada uma oferecendo de forma distinta uma variedade de características. No Anexo é apresentado um estudo comparativo das características mais importantes na descrição e análise de Redes de Petri disponíveis nas principais ferramentas hoje existentes.

Baseando-se nas pesquisas realizadas na área de Redes de Petri do laboratório LSIP, pertencente ao programa de pós-graduação da UTFPR, viu-se a necessidade de criar uma ferramenta gráfica que possibilitasse a modelagem e análise de Redes de Petri Temporais, tendo como principal objetivo a implementação de um jogador que usasse o conceito de tempo global para o tratamento do tempo.

Neste capítulo foi utilizado o termo simulação equivalente à simulação de execução da rede, para representar o disparo de transições de uma RdP, que movimenta marcas de um estado para outro, mostrando o comportamento dinâmico da RdP.

4.1 Características Necessárias

A ferramenta desenvolvida foi denominada PNAnalyzer e pode ser categorizada como sendo um simulador que se baseia nos conceitos formais de modelagem da RdP Lugar/Transição, da RdP Temporal e da Metodologia de Tempo Global, tendo diversas características, entre elas:

- Usar o conceito de tempo global para redes com tempo;
- Criar a rede graficamente;
- Ampliar e reduzir a visualização da rede;
- Imprimir e exportar a rede;
- Simular a execução da rede passo a passo, o usuário escolhe o caminho a ser percorrido;
- Simular a execução da rede automaticamente, o sistema resolve conflitos de forma aleatória;
- Permitir que a mesma seqüência, no modo automático, seja simulada novamente;
- Definir um tempo de parada, permitindo que a simulação da rede pare caso o limite mínimo ou máximo de uma transição disparável seja alcançado;
- Visualizar transições habilitadas, transições disparáveis e regiões de conflito durante a simulação da rede;
- Visualizar os intervalos de tempo das transições durante a simulação da rede;
- Visualizar as classes durante a simulação da rede;
- Determinar os caminhos com estimação de tempo sem simular a execução da rede, fornecendo o estado inicial e o estado final;
- Gerar o grafo de classes para redes seguras.

Tais características compreendem o mínimo necessário para que a ferramenta seja utilizada para analisar uma RdP ordinária ou RPT. A figura 10 apresenta estas características.

4.2 Arquitetura

A ferramenta foi desenvolvida para a plataforma *Windows* utilizando um ambiente integrado de desenvolvimento que utiliza a linguagem pascal e disponibiliza os recursos necessários para a programação orientada a objetos.

O sistema operacional *Windows* dispõe de um conjunto de classes para o desenvolvimento de aplicações e algumas destas foram utilizadas na criação da ferramenta. Um conjunto de

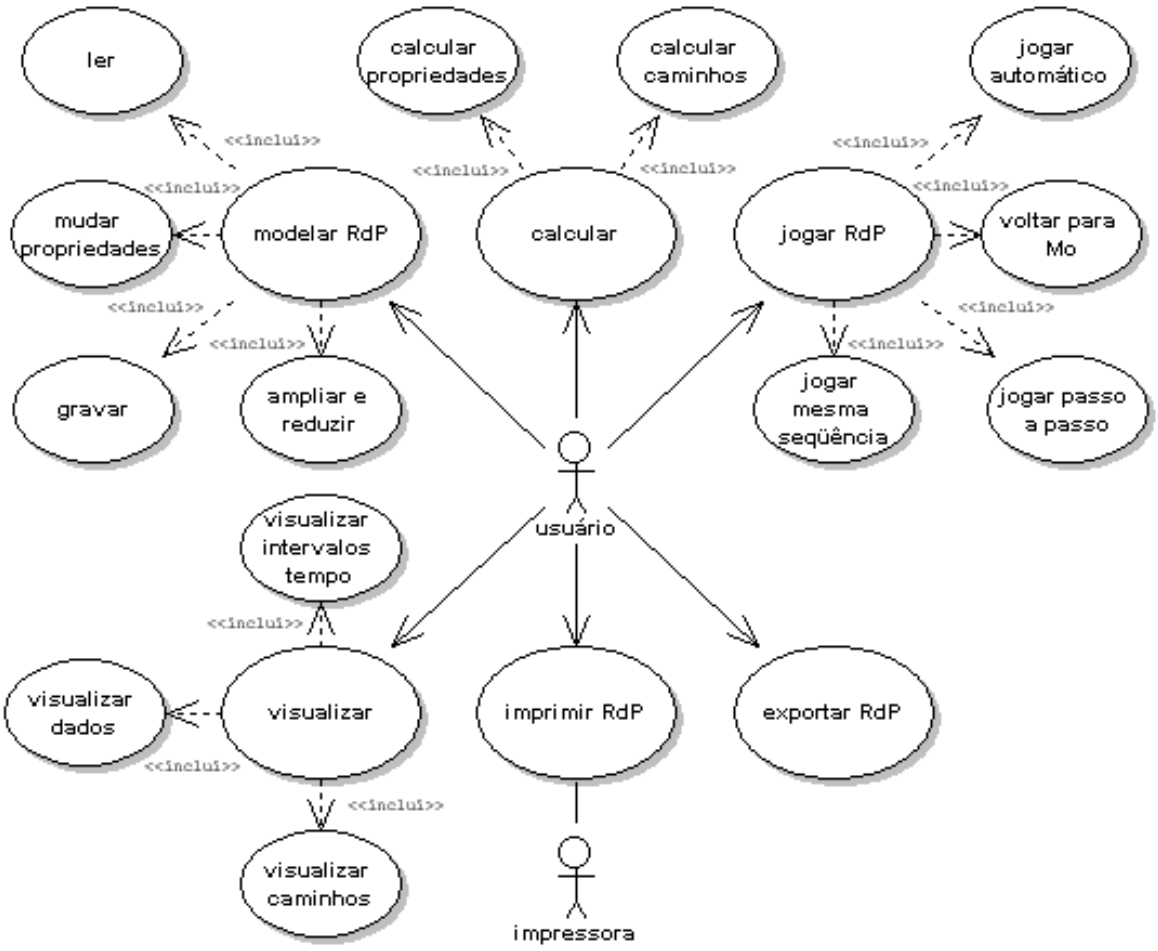

Figura 10: Casos de uso da ferramenta PNAnalyzer

classes específicas foi criado para disponibilizar ao usuário um ambiente gráfico que permitisse de forma natural modelar uma Rede de Petri. As classes são:

- TGraphObject = class(TPersistent)
- TGraphLink = class(TGraphObject)
- TGraphNode = class(TGraphObject)
- TPlaceNode = class(TGraphNode)
- TTransitionNode = class(TGraphNode)
- TClassNode = class(TGraphNode)
- TGraphConstraints = class(TPersistent)

- TGraphScrollBar = class(TPersistent)
- TGraphObjectList = class(TList)
- TPetriNet = class(TCustomControl)

A figura 11 mostra as classes específicas criadas para representar graficamente uma RPT e sua estrutura, assim como a ligação com as classes utilizadas pelo *Windows*.

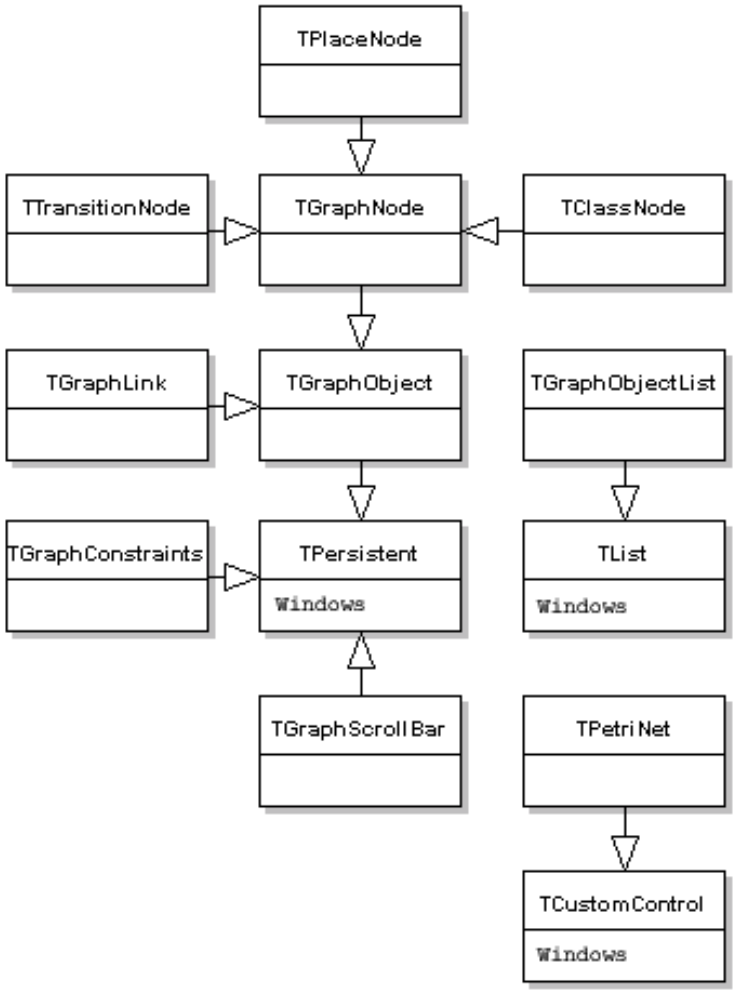

Figura 11: Diagrama de classes da ferramenta PNAnalyzer

A classe TGraphObject é a base para todos os objetos gráficos que aparecem no editor da ferramenta, contém atributos e operações que definem lugares, transições e arcos. Esta classe determina as características do desenho, tais como, espessura e cor da borda, cor de fundo, visibilidade, estado, tipo de fonte e o texto a ser apresentado.

A classe TGraphLink é a base para todos os objetos de ligação (arcos) da rede. Contém atributos que identificam a ligação entre o objeto de saída e o de entrada, pontos de início e fim do arco. Esta classe define o desenho do arco, seta e peso. As marcas, inferior e superior, delimitam este desenho.

A classe TGraphNode é a base para os demais objetos da rede (lugares e transições). Seus atributos e operações identificam a localização do objeto no editor gráfico, o alinhamento, layout e desenho do texto apresentado.

A classe TPlaceNode define o conceito de lugar da RdP na ferramenta. Define, também, o formato e desenho do objeto no editor gráfico.

A classe TTransitionNode define o conceito de transição da RdP na ferramenta. Define, também, o formato e desenho do objeto no editor gráfico.

A classe TClassNode define o conceito de classe de estado da RdP na ferramenta. Define, também, o formato e desenho do objeto na área de visualização de classes de estado durante a simulação da rede.

A classe TGraphConstraints define o limite inferior e superior do editor gráfico. Este limite determina a área na qual se pode desenhar e se ajusta automaticamente para cada objeto gráfico inserido no editor.

A classe TGraphScrollBar mostra e controla uma barra de rolagem horizontal e vertical no editor gráfico. O sistema operacional *Windows* dispõe de uma classe específica para o controle da barra de rolagem, mas foi necessário criar uma nova classe que disponibilizasse um controle mais específico, fazendo com que as barras de rolagem se ajustassem de acordo com as dimensões da RdP modelada.

A classe TGraphObjectList mantém a coleção de objetos gráficos que aparecem no editor da ferramenta. Foi necessário criar uma classe específica para gerenciar os objetos do tipo TGraphObject.

A classe TPetriNet pode ser denominada como *editor gráfico* porque define o controle e visualização dos objetos criados para representar uma Rede de Petri. Permite selecionar objetos, ampliar ou reduzir sua visualização, mostrar partes escondidas da rede através das barras de rolagem horizontal e vertical, mostrar uma grade para organizar os objetos durante a edição, imprimir ou exportar a rede. Contém em sua estrutura todos os atributos e operações necessárias para a análise e simulação da rede.

Algumas destas classes herdaram características de classes do ambiente integrado de desenvolvimento, que encapsula os atributos e operações para a programação na plataforma *Windows*. Segue a descrição destas classes.

A classe TPersistent encapsula o comportamento comum para todos os objetos assinados por outros objetos. É usada como uma classe de base quando se declara objetos que não são componentes e que tenham capacidade de *streaming*.

A classe TList, a qual armazena um vetor de ponteiros, é frequentemente usada para manter uma lista de objetos. Apresenta atributos e operações para adicionar, excluir, reorganizar, localizar, acessar e ordenar objetos na lista.

A classe TCustomControl é uma classe que empacota os objetos de tela da plataforma *Windows* que executam seus próprios desenhos. É uma das duas classes básicas de controle que desenha em sua própria superfície. Instâncias de seus descendentes podem receber foco e servir como recipientes.

Para representar algebricamente uma RdP foram criadas matrizes e vetores, que definem sua estrutura e armazenam informações durante a simulação da rede, conforme a lista abaixo:

1. Informações referentes à estrutura de uma RdP
    - Mo [Lugares]
    - M [Lugares]
    - Pre [Lugares, Transições]
    - Pos [Lugares, Transições]
    - C [Lugares, Transições]
    - CTP [Lugares]
2. Informações referentes à Metodologia de Tempo Global
    - TT [Transições, 2][1]
3. Informações sobre as transições habilitadas, habilitadas anteriormente e disparáveis
    - ET [Transições]
    - ETP [Transições]

[1]Informações sobre a Metodologia de Tempo Global

- FT [Transições]

4. Informações referentes ao processo de redução

   - IM [Transições, Transições]
   - IMR [Transições, Transições]
   - PT [Transições]

Os vetores Mo e M definem, respectivamente, o vetor de marcação inicial e o vetor de marcação de uma RdP. Este último armazena a marcação da rede em um determinado estado durante a simulação da rede.

As matrizes Pre, Pos e C definem, respectivamente, a matriz de incidência prévia, a matriz de incidência posterior e a matriz de incidência de uma RdP.

O vetor CTP define os lugares que fazem parte de um conflito na RdP. A partir desta informação, a ferramenta mostra as regiões de conflito na estrutura da rede. Estas regiões são formadas por lugares, transições e arcos. Suas colunas representam os lugares da rede e cada uma é composta de 0 (zero) ou 1 (um), indicando que o lugar pertence ao conflito.

A matriz TT armazena os intervalos de tempo global das transições disparáveis durante a simulação da rede. A primeira coluna contém o limite inferior e a segunda o limite superior.

Os vetores ET (*Enabled Transitions*), ETP (*Enabled Transitions Past*) e FT (*Fireable Transitions*) definem, respectivamente, as transições habilitadas no estado atual, as transições habilitadas no estado anterior e as transições disparáveis durante a simulação da rede. Suas colunas representam as transições da rede e cada uma é composta de 0 (zero) ou 1 (um), indicando que a transição está habilitada (ET), habilitada anteriormente (ETP) ou disparável (FT).

A matriz IM representa a ordem de precedência entre as transições da rede. Com esta matriz é possível identificar paralelismo entre as transições, conflito e transições em série. É obtida a partir de regras descritas no apêndice 3.3.2. A criação da matriz IMR (Matriz Intervalar) é feita a partir desta matriz.

A matriz IMR (Matriz Intervalar) contém a representação da RdP Temporal Reduzida. Seus elementos são intervalos resultantes do agrupamento de transições paralelas e conflitantes segundo o algoritmo apresentado na seção 3.3.2.

O vetor PT contém todas as conexões entre as transições iniciais, habilitadas pela marcação inicial, e as transições finais, que levam à marcação final.

O uso destes vetores e matrizes foi feito através de algoritmos, implementados seguindo o conceito da RdP Temporal e da Metodologia de Tempo Global. A seguir é descrito o funcionamento de cada um deles.

4.3 Algoritmos

Os algoritmos são explicados usando o clássico problema do jantar de cinco filósofos, proposto por [Dij68], figura 12. A rede original foi alterada, colocando-se intervalos de tempo nas transições para explicar o funcionamento das regras proposta pela Metodologia de Tempo Global. Esses algoritmos foram escritos em pseudo-código, contendo o necessário para explicar o funcionamento da ferramenta.

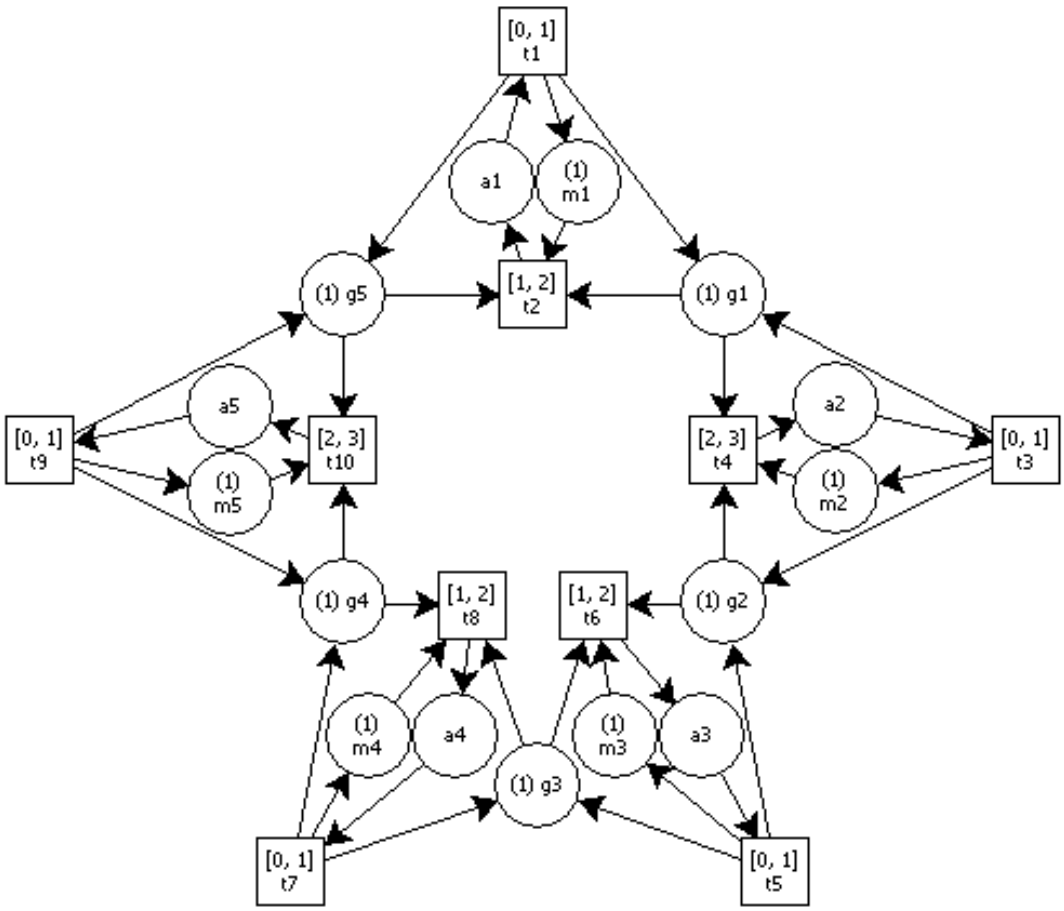

Figura 12: Problema dos filósofos

Os algoritmos foram agrupados de acordo com a área na qual executam suas atividades na ferramenta, da seguinte forma:

1. Relacionados com a estrutura da Rede de Petri Temporal
    - Calcular
    - CriarCTP
    - CriarET
    - CriarFT
    - CriarM(Estado Atual)
2. Relativos à simulação da Rede de Petri Temporal
    - VerificarFT
    - Executar
    - VerificarEscolhaTransicaoDisparavel(Modo de Execução)
    - VerificarTempoParada
    - EscolhaAleatoria
3. Relacionados com a metodologia de Tempo Global
    - CalcularTempo
4. Relativos ao processo de redução
    - CalcularCaminhos
    - CriarIM
    - Recursividade(Transicao, Coluna)
    - ReduzirIM
    - VerificarVertice(Transicao, Coluna)
    - CriarPT
    - CompararCaminho(Vetor1, Vetor2)

4.3.1 Algoritmo - *Calcular*

Este algoritmo cria as matrizes (Pre, Pos, C e TT) e o vetor (Mo) que representam, respectivamente, a estrutura e o estado inicial da rede que está modelada no editor da ferramenta. Somente a partir de sua execução é que a ferramenta habilita os processos de simular a execução, de mostrar informações sobre a estrutura da rede e de visualizar o estado das transições (habilitadas, disparáveis e regiões de conflito). De acordo com o cálculo do tempo global, a matriz TT é inicializada com os intervalos estáticos das transições, sendo atualizada durante a simulação da rede.

Código do algoritmo *Calcular*

```
início
  inicializar o vetor Mo
  inicializar as matrizes Pre, Pos, C e TT

  para (cada objeto gráfico) faça // objetos modelados no editor da ferramenta
    se (objeto for do tipo lugar) então
      posicao recebe a numeração interna do objeto lugar

      se (quantidade de marcas for maior que zero) então
        Mo[posicao] recebe a quantidade de marcas do objeto lugar
      fim se
    fim se

    se (objeto for do tipo arco) então
      se (arco não for bidirecional) então
        se (origem do arco for um objeto do tipo lugar) então
          posLugar recebe a numeração interna do objeto lugar apontado pela
            origem do arco
          posTransicao recebe a numeração interna do objeto transição apontado pelo
            destino do arco

          Pre[posLugar, posTransicao] recebe o peso do arco
        fim se

        se (origem do arco for um objeto do tipo transição) então
          posLugar recebe a numeração interna do objeto lugar apontado pelo
            destino do arco
          posTransicao recebe a numeração interna do objeto transição apontado pela
            origem do arco

          Pos[posLugar, posTransicao] recebe o peso do arco
        fim se
      senão
        se (origem do arco for um objeto do tipo lugar) então
          posLugar recebe a numeração interna do objeto lugar apontado pela
            origem do arco
          posTransicao recebe a numeração interna do objeto transição apontado pelo
            destino do arco
        fim se

        se (origem do arco for um objeto do tipo transição) então
          posLugar recebe a numeração interna do objeto lugar apontado pelo
            destino do arco
          posTransicao recebe a numeração interna do objeto transição apontado pela
            origem do arco
        fim se

        Pre[posLugar, posTransicao] recebe o peso do arco
```

```
        Pos[posLugar, posTransicao] recebe o peso do arco
      fim se

      C[posLugar, posTransicao] recebe Pos[posLugar, posTransicao] -
                                       Pre[posLugar, posTransicao]
    fim se

    se (objeto for do tipo transição) então
      posicao recebe a numeração interna do objeto transição

      TT[posição, 0] recebe o limite de tempo inferior do objeto transição
      TT[posição, 1] recebe o limite de tempo superior do objeto transição
    fim se
  fim para

  copiar o vetor Mo para o vetor M

  executar o procedimento CriarET
  executar o procedimento CriarCTP
  executar o procedimento CriarFT

  joga_rede recebe o retorno da função VerificarFT
fim.
```

Executando este algoritmo para o exemplo apresentado na figura 12, tem-se o resultado apresentado nas tabelas 2, 3, 4 e 5.

Tabela 2: Marcação inicial (M_0) gerada pelo algoritmo *Calcular*

Mo														
a_1	a_2	a_3	a_4	a_5	m_1	m_2	m_3	m_4	m_5	g_1	g_2	g_3	g_4	g_5
0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

4.3.2 Algoritmo - *CriarET*

Este algoritmo cria o vetor ET que representa as transições habilitadas do estado no qual a rede se encontra, e o vetor ETP que representa as transições que foram habilitadas no estado anterior. Baseia-se no vetor M e nas matrizes Pré e Pos para verificar quais transições estão habilitadas. É chamado pelo algoritmo Calcular, definindo as transições no estado inicial e pelo algoritmo Executar, definindo as transições durante a simulação da rede. Suas colunas representam as transições da rede e cada uma é composta de 0 (zero) ou 1 (um), indicando que a transição está habilitada (ET) ou habilitada anteriormente (ETP).

Tabela 3: Matriz de incidência prévia (Pré) gerada pelo algoritmo *Calcular*

Pré (PxT)										
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
a_1	1	0	0	0	0	0	0	0	0	0
a_2	0	0	1	0	0	0	0	0	0	0
a_3	0	0	0	0	1	0	0	0	0	0
a_4	0	0	0	0	0	0	1	0	0	0
a_5	0	0	0	0	0	0	0	0	1	0
m_1	0	1	0	0	0	0	0	0	0	0
m_2	0	0	0	1	0	0	0	0	0	0
m_3	0	0	0	0	0	1	0	0	0	0
m_4	0	0	0	0	0	0	0	1	0	0
m_5	0	0	0	0	0	0	0	0	0	1
g_1	0	1	0	1	0	0	0	0	0	0
g_2	0	0	0	1	0	1	0	0	0	0
g_3	0	0	0	0	0	1	0	1	0	0
g_4	0	0	0	0	0	0	0	1	0	1
g_5	0	1	0	0	0	0	0	0	0	1

Tabela 4: Matriz de incidência posterior (Pós) gerada pelo algoritmo *Calcular*

Pos (PxT)										
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
a_1	0	1	0	0	0	0	0	0	0	0
a_2	0	0	0	1	0	0	0	0	0	0
a_3	0	0	0	0	0	1	0	0	0	0
a_4	0	0	0	0	0	0	0	1	0	0
a_5	0	0	0	0	0	0	0	0	0	1
m_1	1	0	0	0	0	0	0	0	0	0
m_2	0	0	1	0	0	0	0	0	0	0
m_3	0	0	0	0	1	0	0	0	0	0
m_4	0	0	0	0	0	0	1	0	0	0
m_5	0	0	0	0	0	0	0	0	1	0
g_1	1	0	1	0	0	0	0	0	0	0
g_2	0	0	1	0	1	0	0	0	0	0
g_3	0	0	0	0	1	0	1	0	0	0
g_4	0	0	0	0	0	0	1	0	1	0
g_5	1	0	0	0	0	0	0	0	1	0

Tabela 5: Matriz de intervalos no estado inicial (TT) gerada pelo algoritmo *Calcular*

TT									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
	[1, 2]		[2, 3]		[1, 2]		[1, 2]		[2, 3]

Código do algoritmo *CriarET*

```
início
  copiar o vetor ET para o vetor ETP

  inicializar o vetor ET

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    para (j recebe valores entre [0, quantidade de lugares da rede - 1]) faça
      se (Pre[j, i] >= 1) então
        ET[i] recebe 1

        se ((M[j] = 0) ou (M[j] - Pre[j, i] < 0)) então
          ET[i] recebe 0

          sair do laço mais interno (j)
        fim se
      fim se

      se ((Pre[j, i] = 0) e (Pos[j, i] >= 1)) então
        ET[i] recebe 1
      fim se
    fim para
  fim para
fim.
```

A figura 13 apresenta as transições habilitadas no estado inicial. Executando o algoritmo *CriarET* para o exemplo apresentado na figura 12, tem-se o resultado apresentado na tabela 6.

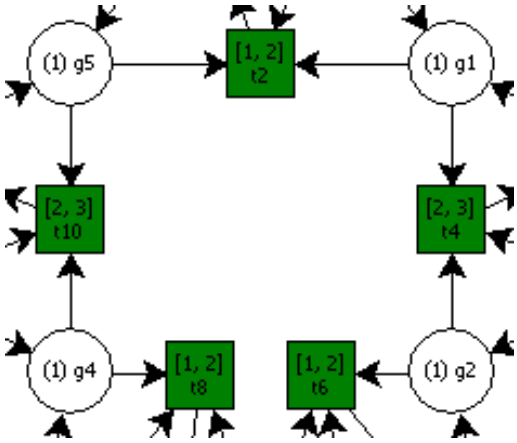

Figura 13: Figura parcial da rede de Petri da figura 12 com as transições habilitadas no estado inicial

Tabela 6: Vetor de transições habilitadas no estado inicial, gerado pelo algoritmo *CriarET*

ET									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
0	1	0	1	0	1	0	1	0	1

4.3.3 Algoritmo - *CriarCTP*

Este algoritmo cria o vetor CTP que define os lugares que fazem parte de um conflito, caso exista, na RdP. É chamado pelo algoritmo Calcular, definindo os lugares no estado inicial, e pelo algoritmo Executar, definindo os lugares durante a simulação da rede. Suas colunas representam os lugares da rede e cada uma é composta de 0 (zero) ou 1 (um), indicando se o lugar pertence ao conflito.

Código do algoritmo *CriarCTP*

```
início
  inicializar o vetor CT

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    para (j recebe valores entre [0, quantidade de lugares da rede - 1]) faça
      se (Pre[j, i] >= 1) então
        CT[i] recebe 1

        se ((M[j] = 0) ou (M[j] - Pre[j, i] < 0)) então
          CT[i] recebe 0

          sair do laço mais interno (j)
        fim se
      fim se

      se ((Pre[j, i] = 0) e (Pos[j, i] >= 1)) então
        CT[i] recebe 1
      fim se
    fim para
  fim para

  inicializar o vetor CTP

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se (CT[i] = 1) então
      para (cada objeto gráfico) faça // objetos modelados no editor da ferramenta
        se (objeto for do tipo arco) então
          se (origem do arco for um objeto do tipo lugar) então
            posLugar recebe a numeração interna do objeto lugar apontado pela
              origem do arco
            posTransicao recebe a numeração interna do objeto transição apontado pelo
              destino do arco

            se (posTransicao = i) então
              CTP[posLugar] recebe CTP[posLugar] + 1
            fim se
          fim se
        fim se
      fim para
    fim se
  fim para
fim.
```

A figura 14 apresenta a região de conflito no estado inicial. Executando este algoritmo para o exemplo apresentado na figura 12, tem-se o resultado apresentado na tabela 7.

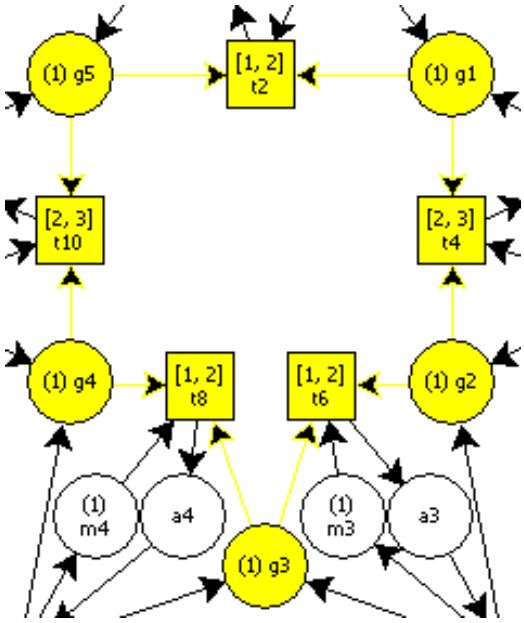

Figura 14: Figura parcial da rede de Petri da figura 12 com conflitos no estado inicial

Tabela 7: Vetor dos lugares pertencentes ao conflito (CTP), no estado inicial, gerado pelo algoritmo *CriarCTP*

CTP														
a_1	a_2	a_3	a_4	a_5	m_1	m_2	m_3	m_4	m_5	g_1	g_2	g_3	g_4	g_5
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

4.3.4 Algoritmo - *CriarFT*

Este algoritmo cria o vetor FT que representa as transições disparáveis segundo as regras estabelecidas na Metodologia de Tempo Global apresentada no capítulo 3, baseando-se nos vetores ET e ETP. É chamado pelo algoritmo Calcular, definindo as transições no estado inicial e pelo algoritmo Executar, definindo as transições durante a simulação da rede. Suas colunas representam as transições da rede e cada uma é composta de 0 (zero) ou 1 (um), indicando se a transição é disparável.

Código do algoritmo *CriarFT*

```
início
  inicializar o vetor FT

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    para (j recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se ((ET[i] = 1) e (ET[j] = 1)) então
        FT[i] recebe 1
```

```
          se (limite de tempo inferior da transição i >
              limite de tempo superior da transição j) então
            FT[i] recebe 0

            sair do laço mais interno (j)
          fim se
        fim se
      fim para
    fim para

    para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se ((FT[i] = 1) e (ETP[i] = 0) e (ET[i] = 1)) então
        para (j recebe valores entre [0, quantidade de transições da rede - 1]) faça
          se ((FT[j] = 1) e (ETP[j] = 0) e (ET[j] = 1)) então
            se (limite de tempo inferior da transição i >
                limite de tempo superior da transição j) então
              FT[i] recebe 0

              sair do laço mais interno (j)
            fim se
          fim se
        fim para
      fim se
    fim para
  fim.
```

A figura 15 mostra as transições disparáveis t_2, t_4, t_6, t_8 e t_{10}, verificadas pelo algoritmo para o estado inicial, conforme a tabela 8. Após o disparo da transição t_2, a transição t_1 se torna disparável e as transições t_6 e t_8 permaneceram disparáveis, conforme a figura 16. Disparando a transição t_1 na seqüência, a transição t_2 se torna novamente disparável e as transições t_6 e t_8 continuaram disparáveis. Nota-se também que, após o disparo de t_1, as transições t_4 e t_{10} ficaram habilitadas, mas não ficaram disparáveis porque seus tempos globais estão fora do limite das outras transição disparáveis t_2, t_6 e t_8, conforme a figura 17 e a tabela 9.

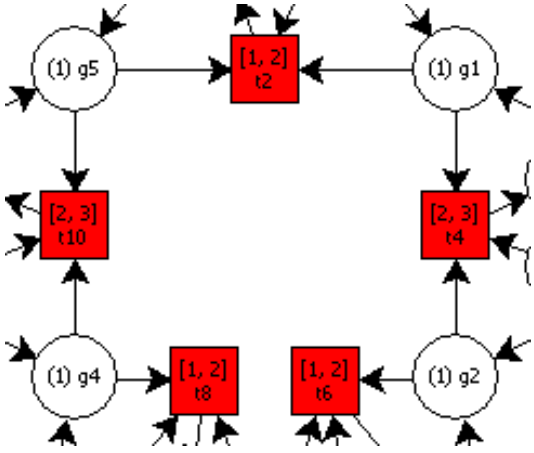

Figura 15: Figura parcial da rede de Petri da figura 12 com as transições disparáveis no estado inicial

Tabela 8: Vetor das transições disparáveis (FT), no estado inicial, gerado pelo algoritmo *CriarFT*

FT									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
0	1	0	1	0	1	0	1	0	1

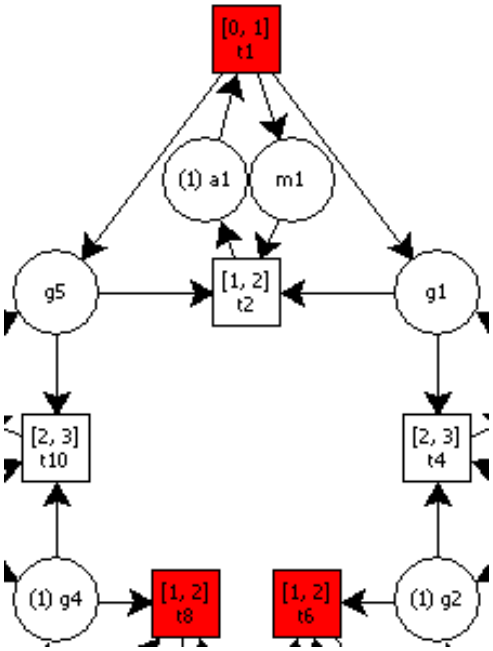

Figura 16: Figura parcial da rede de Petri da figura 12 com as transições disparáveis após disparo de t_2

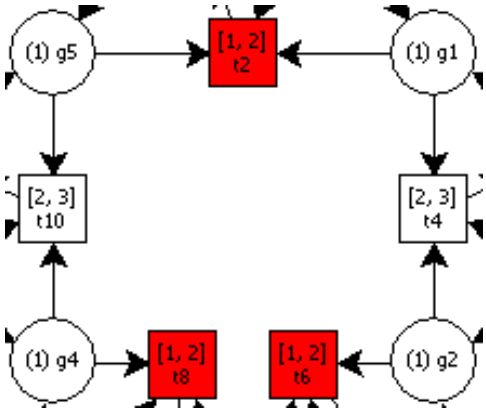

Figura 17: Figura parcial da rede de Petri da figura 12 com as transições disparáveis após disparo de t_1

4.3.5 Algoritmo - *VerificarFT*

O algoritmo *VerificarFT* informa a existência de transições disparáveis na rede para cada estado, baseando-se no vetor FT. É chamado pelo algoritmo Calcular para habilitar os

Tabela 9: Matriz de intervalos após a seqüência de disparo t_2 e t_1 a partir do estado inicial

TT									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
	[2, 4]		[3, 5]		[1, 2]		[1, 2]		[3, 5]

botões de execução automática ou passo a passo da ferramenta e pelo algoritmo Executar para decidir se a simulação da rede pode continuar.

Código do algoritmo *VerificarFT*

```
início
  resultado_funcao recebe falso

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se (FT[i] = 1) então
      resultado_funcao recebe verdadeiro

      sair do laço (i)
    fim se
  fim para
fim.
```

De acordo com o exemplo apresentado na figura 15, as transições disparáveis são t_2, t_4, t_6, t_8 e t_{10}, e o resultado deste algoritmo será verdadeiro, informando que existem transições disparáveis.

4.3.6 Algoritmo - *Executar*

Este algoritmo é responsável pelo processo que simula a execução da rede nos modos passo a passo ou automático. Foi criado para disponibilizar ao usuário um controle maior sobre a simulação da rede, por exemplo, se o usuário escolher o modo automático e em determinado momento parar a simulação, este pode continuar a partir do estado no qual a rede se encontra, bastando escolher uma das transições disparáveis. No momento que não houver mais a possibilidade de disparo, a ferramenta pára a simulação automaticamente. Além disso, a ferramenta dispõe de um tempo de parada, isto quer dizer, se durante a simulação o limite inferior ou superior de uma das transições alcançar este tempo, a simulação é finalizada.

Código do algoritmo *Executar*

```
início
  se ((existe transição disparável) e (tempo de parada não foi alcançado)) então
    executar o procedimento CriarM(Estado Atual)
    executar o procedimento CriarET
    executar o procedimento CriarCTP
    executar o procedimento CalcularTempo
    executar o procedimento CriarFT

    joga_rede recebe o retorno da função VerificarFT
    resultado_funcao recebe joga_rede
  senão
    resultado_funcao recebe falso
  fim se
fim.

Obs.: São executadas duas funções no teste acima. Em 'existe transição disparável' é chamada a função
      VerificarEscolhaTransicaoDisparavel passando como parâmetro o modo de execução e em
      'o tempo de parada não foi alcançado' é chamada a função  VerificarTempoParada.
```

4.3.7 Algoritmo - *VerificarEscolhaTransicaoDispáravel(Modo de Execução)*

Este algoritmo verifica se existe uma ou mais transições disparáveis. Caso o parâmetro recebido seja o modo passo a passo, é feita uma verificação para saber se o usuário selecionou uma destas transições; caso contrário, é chamado o algoritmo que escolhe de forma aleatória uma delas para disparo. Baseia-se no vetor FT para verificar as transições e é chamado pelo algoritmo Executar para verificar se a rede pode continuar a simulação.

Código do algoritmo *VerificarEscolhaTransicaoDispáravel(Modo de Execução)*

```
início
  resultado_funcao recebe falso
  exite_transicao_disparavel recebe falso

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se (FT[i] = 1) então
      se ((passo a passo) e (objeto selecionado for uma transição disparável) e
          (transição selecionada = i)) então
        transicao_para_disparo recebe i
        resultado_funcao recebe verdadeiro

        sair do laço (i)
      fim se

    existe_transicao_disparavel recebe verdadeiro
  fim para

  se ((automático) e (existe_transicao_disparavel)) então
    transicao_para_disparo recebe o retorno da função EscolhaAleatoria
    resultado_funcao recebe verdadeiro
  fim se
fim.
```

De acordo com o exemplo apresentado na figura 15, as transições disparáveis são t_2, t_4, t_6, t_8 e t_{10}, e se o modo for passo a passo, o usuário poderá escolher uma delas, senão, a ferramenta fará a escolha aleatoriamente.

4.3.8 Algoritmo - *VerificarTempoParada*

O algoritmo verifica se o tempo de parada definido na ferramenta foi alcançado, baseando-se no vetor FT e na matriz TT. A simulação da rede é finalizada quando o limite inferior ou superior de uma das transições disparáveis for maior ou igual ao tempo de parada. É chamado pelo algoritmo Executar para verificar se a simulação da rede pode continuar.

Código do algoritmo *VerificarTempoParada*

```
início
  resultado_funcao recebe verdadeiro

  se (tempo de parada configurado > 0) então
    para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se (FT[i] = 1) então
        se (((está configurado para verificar o limite inferior) e
            (TT[i, 0] >= tempo de parada configurado))
           ou
            ((está configurado para verificar o limite superior) e
            (TT[i, 1] >= tempo de parada configurado))) então
          Resultado_funcao recebe falso

          sair do laço (i)
        fim se
      fim se
    fim para
  fim se
fim.
```

A figura 18 mostra o estado no qual a simulação da rede foi parada quando o tempo, definido em 12 unidades, foi alcançado por uma das transições disparáveis e, neste caso, foi a transição t_{10} que alcançou este tempo. A tabela 10 mostra os intervalos de tempo das transições que permaneceram disparáveis, obtidos pelo disparo aleatório das transições habilitadas, simulando a rede no modo automático.

Tabela 10: Vetor dos intervalos de tempo das transições que permaneceram disparáveis (TT)

TT									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
		[6, 10]					[7, 11]		[8, 12]

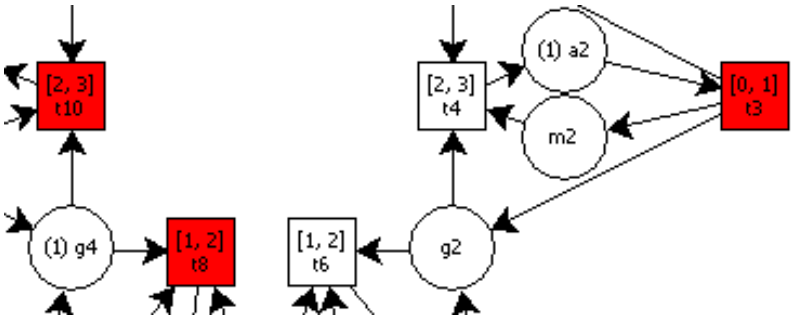

Figura 18: Figura parcial da rede de Petri da figura 12 com as transições que permaneceram disparáveis

4.3.9 Algoritmo - *CriarM(Estado Atual)*

O algoritmo *CriarM(Estado Atual)* atualiza o vetor M a partir da marcação na qual a rede se encontra. É chamado pelo algoritmo *Executar* para armazenar o estado atual de marcações durante a simulação da rede.

Código do algoritmo *CriarM(Estado Atual)*

```
início
  inicializar o vetor M

  para (i recebe valores entre [0, quantidade de lugares da rede - 1]) faça
    M[i] recebe Estado Atual[i] +
                Pos[i, transicao_para_disparo] -
                Pre[i, transicao_para_disparo]

    se (M[i] < 0) então
      M[i] recebe 0
    fim se
  fim para
fim.
```

A figura 19 mostra a nova marcação da rede após o disparo da transição t_2 da figura 15 e a tabela 11 mostra os novos valores. Vê-se que uma marca foi colocada no lugar a_1 e as marcas dos lugares m_1, g_1 e g_5 foram retiradas.

Tabela 11: Marcação após o disparo da transição t2, atualizada pelo algoritmo *CriarM(Estado Atual)*

M														
a_1	a_2	a_3	a_4	a_5	m_1	m_2	m_3	m_4	m_5	g_1	g_2	g_3	g_4	g_5
1	0	0	0	0	0	1	1	1	1	0	1	1	1	0

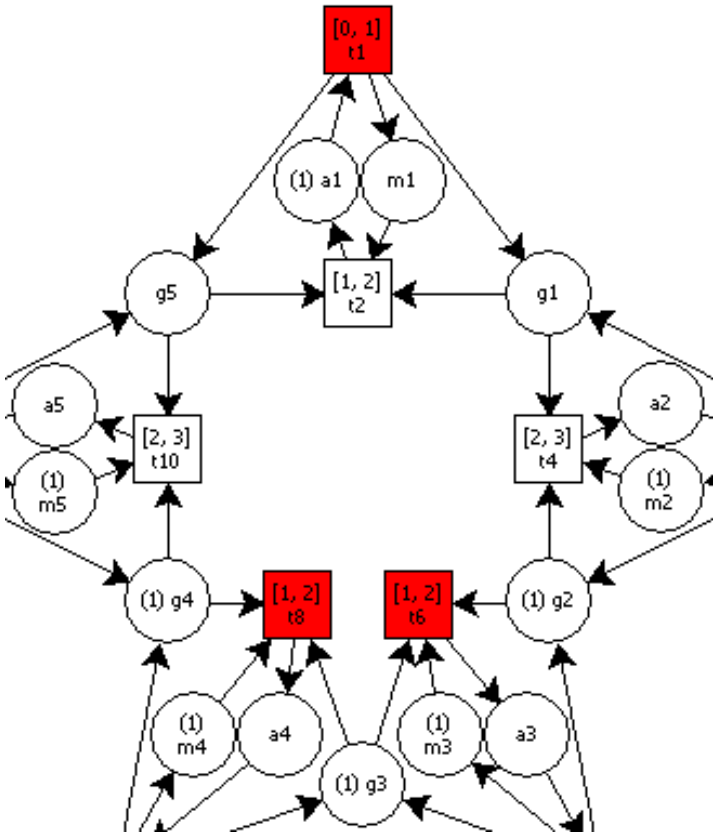

Figura 19: Figura parcial da rede de Petri da figura 12 com a nova marcação após o disparo de t_2

4.3.10 Algoritmo - *CalcularTempo*

O algoritmo *CalcularTempo* atualiza a matriz TT segundo as regras estabelecidas na Metodologia de Tempo Global apresentada no capítulo 3, baseando-se nos vetores ET e ETP. É chamado pelo algoritmo Executar para armazenar o tempo global das transições durante a simulação da rede.

Código do algoritmo *CalcularTempo*

```
início
  primeira_transicao_habilitada recebe 0

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se (ETP[i] = 1) então
      primeira_transicao_habilitada recebe i

      sair do laço (i)
    fim se
  fim para

  tempo_maximo_limite_superior recebe TT[primeira_transicao_habilitada, 1]
```

```
  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se ((tempo_maximo_limite_superior > TT[i, 1]) e (ETP[i] = 1)) faça
      tempo_maximo_limite_superior recebe TT[i, 1]
    fim se
  fim para

  ETP[transicao_para_disparo] recebe 0

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se ((ETP[i] = 0) e (ET[i] = 1)) então
      TT[i, 1] recebe limite superior da transição apontada por i +
                      tempo_maximo_limite_superior
    fim se
  fim para

  tempo_minimo_limite_inferior recebe TT[transicao_para_disparo, 0]

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se ((ETP[i] = 1) e (ET[i] = 1) e (TT[i, 0] < tempo_minimo_limite_inferior)) então
      TT[i, 0] recebe tempo_minimo_limite_inferior
    fim se

    se ((ETP[i] = 0) e (ET[i] = 1)) então
      TT[i, 0] recebe limite inferior da transição apontada por i +
                      tempo_minimo_limite_inferior
    fim se
  fim para
fim.
```

A figura 20 mostra as transições disparáveis t_2, t_4, t_6, t_8 e t_{10} e a tabela 12 mostra o tempo global para cada transição, no estado inicial. Após o disparo da transição t_2, a transição t_1 se torna disparável com o tempo global de [1, 3] e as transições t_6 e t_8 permaneceram com o tempo global de [1, 2], conforme a tabela 13, e disparando a transição t_1 na seqüência, a transição t_2 se torna novamente disparável com o tempo global de [2, 4] e as transições t_6 e t_8 continuaram disparáveis com o tempo global de [1, 2]. Nota-se, também, que após o disparo de t_1, as transições t_4 e t_{10} ficaram habilitadas, mas não ficaram disparáveis porque seus tempos globais estão fora do limite das outras transição disparáveis t_2, t_6 e t_8, conforme a tabela 14.

Tabela 12: Matriz de intervalos (TT) no estado inicial

TT									
t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
	[1, 2]		[2, 3]		[1, 2]		[1, 2]		[2, 3]

Tabela 13: Matriz de intervalos (TT) após o disparo de t_2 a partir do estado inicial

TT									
t_1 (nova)	t_2	t_3	t_4	t_5	t_6 (antiga)	t_7	t_8 (antiga)	t_9	t_{10}
[1, 3] = [1, 2] + [0, 1]					[1, 2]		[1, 2]		

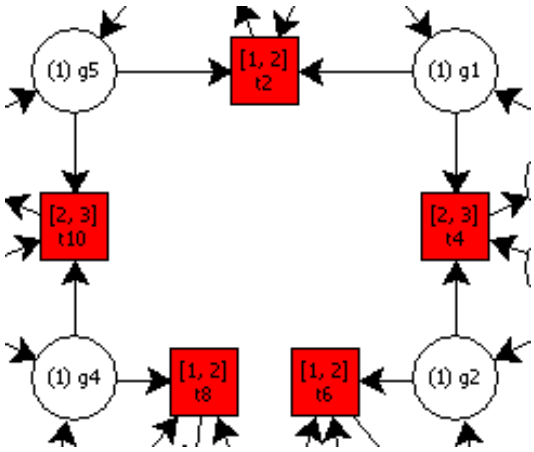

Figura 20: Figura parcial da rede de Petri da figura 12 com as transições disparáveis no estado inicial

Tabela 14: Matriz de intervalos (TT) após a seqüência de disparo t_2 e t_1 a partir do estado inicial

TT				
t_1	t_2 (nova)	t_3	t_4 (nova)	t_5
	$[2, 4] = [1, 2] + [1, 2]$		$[3, 5] = [1, 2] + [2, 3]$	
t_6 (antiga)	t_7	t_8 (antiga)	t_9	t_{10} (nova)
$[1, 2]$		$[1, 2]$		$[3, 5] = [1, 2] + [2, 3]$

4.3.11 Algoritmo - *EscolhaAleatoria*

O algoritmo é chamado quando a ferramenta está configurada para simular a execução da rede no modo automático, retornando, caso exista, uma das transições disparáveis. Para esta escolha aleatória, é gerado um número entre 0 e 100 que será dividido pelo número de transições disparáveis no momento da escolha. Por exemplo, se o número gerado for 67 e houver 4 transições disparáveis, o algoritmo dividirá o número 100 por 4 e obterá quatro setores, sendo o primeiro de 0 à 24, o segundo de 25 à 49, o terceiro de 50 à 74 e o quarto de 75 à 100, escolhendo a terceira transição para disparo porque o número gerado pertence ao terceiro setor. É chamado pelo algoritmo VerificarEscolhaTransicaoDisparavel a cada passo da simulação da rede.

Código do algoritmo *EscolhaAleatoria*

```
início
  inicializar o vetor FTC

  qtde_transicoes_disparaveis recebe 0
```

```
  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    se (FT[i] = 1) então
      mudar o tamanho do vetor FTC para qtde_transicoes_disparaveis + 1

      FTC[qtde_transicoes_disparaveis] recebe i

      qtde_transicoes_disparaveis recebe qtde_transicoes_disparaveis + 1
    fim se
  fim para

  resultado_funcao recebe FTC[0]

  se (qtde_transicoes_disparaveis > 1) então
    numero_gerado recebe escolha entre 0 e 100

    posicao recebe o número inteiro da divisão
                (numero_gerado/(100/qtde_transicoes_disparaveis))

    resultado_funcao recebe FTC[posicao]

    se (posicao > (quantidade de elementos do vetor FTC - 1)) então
      resultado_funcao recebe FTC[posicao - 1]
    fim se
  fim se
fim.
```

4.3.12 Algoritmo - *CalcularCaminhos*

O algoritmo *CalcularCaminhos* cria as matrizes IM, IMR e PT.

Código do algoritmo *CalcularCaminhos*

```
início
  se (matriz IM ainda não foi calculada) então
    executar o procedimento CriarIM
    executar o procedimento ReduzirIM
  fim se

  executar o procedimento CriarPT
fim.
```

A rede da figura 21 será utilizada para explicar os algoritmos CriarIM, ReduzirIM e CriarPT.

4.3.13 Algoritmo - *CriarIM*

Este algoritmo, chamado pelo algoritmo CalcularCaminhos, cria a matriz IM que representa a ordem de precedência entre as transições da rede e, para isso, segue as regras descritas na seção 3.3.2.

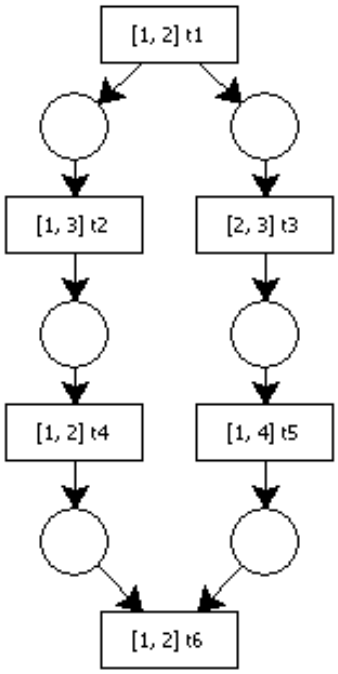

Figura 21: Rede criada para explicar os algoritmos envolvidos no cálculo de caminhos

Código do algoritmo *CriarIM*

```
início
  indice recebe -1

  inicializar a matriz IMVertices
  inicializar a matriz IM

  copiar a matriz Pre para matriz IMPre

  para (i recebe valores entre [0, quantidade de lugares da rede - 1]) faça
    para (j recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se (Mo[i] > 0) e (IMPre[i, j] = 1)) então
        IMPre[i, j] recebe 0
      fim se
    fim para
  fim para

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    IM[i, i] recebe i + 1

    executar o procedimento RecursividadeIM(i, i)
  fim para

  copiar a matriz IM para a matriz IMR
  executar o procedimento ReduzirIM
fim.
```

A execução do algoritmo *CriarIM* para o exemplo apresentado na figura 21 gera a tabela 15. Esta tabela é a mesma que foi formada no exemplo 3.1 do capítulo 3.

Tabela 15: Matriz de Transições (IM) gerada pelo algoritmo *CriarIM*

IM					
1					
1	2				
1		3			
1	2		4		
1		3		5	
1	2	3	4	5	6

4.3.14 Algoritmo - *RecursividadeIM(Transicao, Coluna)*

RecursividadeIM é um algoritmo recursivo que faz uma busca exaustiva para criar a matriz IM que contém a representação de todos os caminhos da rede. É chamado pelo algoritmo CriarIM.

Código do algoritmo *RecursividadeIM(Transicao, Coluna)*

```
início
  para (t recebe valores entre [0, quantidade de transições da rede - 1]) faça
    para (l recebe valores entre [0, quantidade de lugares da rede - 1]) faça
      se ((Pos[l, Transicao] >= 1) e (IMPre[l, t] >= 1)) então
        se (par t/Coluna ainda não foi analisado) então
          IM[t, Coluna] recebe Coluna + 1

          executar o procedimento RecursividadeIM(t, Coluna)
        fim se
      fim se
    fim para
  fim para
fim.

Obs.: Em "par t/Coluna ainda não foi analisado" é chamada a função VerificarVertice,
      passando como parâmetros a Transição e a Coluna.
```

4.3.15 Algoritmo - *ReduzirIM*

Este algoritmo, chamado pelo algoritmo CalculaCaminhos, cria a matriz IMR (Matriz Intervalar) segundo o algoritmo definido na seção 3.3.2.

Código do algoritmo *ReduzirIM*

```
início
  repete_laço recebe verdadeiro

  enquanto (repete_laço = verdadeiro) faça
    repete_laço recebe falso

    para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
      para (j recebe valores entre [i + 1, quantidade de transições da rede - 1]) faça
        se (IMR[j, i] = '') então
          para (k recebe valores entre [i + 1,
                quantidade de transições da rede - 1]) faça
            se (IMR[j, k] <> '') então
              se (j = k) então
                se (IMR[i, k] = '') então
                  para (linha recebe valores entre [0,
                        quantidade de transições da rede - 1]) faça
                    se (IMR[linha, k] <> '') então
                      se (IMR[linha, i] = '') então
                        IMR[linha, i] recebe IMR[j, j]
                      senão
                        IMR[linha, i] recebe IMR[linha, i] + '-' + IMR[j, j]
                      fim se
                    fim se
                  fim para

                  para (linha recebe valores entre [0,
                        quantidade de transições da rede - 1]) faça
                    IMR[linha, k] recebe ''
                  fim para

                  para (coluna recebe valores entre [0,
                        quantidade de transições da rede - 1]) faça
                    se (IMR[j, coluna] <> '') então
                      se (IMR[i, coluna] = '') então
                        IMR[i, coluna] recebe IMR[j, coluna]
                      senão
                        IMR[i, coluna] recebe IMR[i, coluna] + '-' + IMR[j, coluna]
                      fim se
                    fim se
                  fim para

                  para (coluna recebe valores entre [0,
                        quantidade de transições da rede - 1]) faça
                    IMR[j, coluna] recebe ''
                  fim para

                  repete_laço recebe verdadeiro

                  sair dos laços (k, j e i), executando novamente o laço enquanto
                fim se
              senão
                sair do laço (k)
              fim se
            fim se
          fim para
        fim se
      fim para
    fim para
  fim enquanto
fim.
```

A execução do algoritmo *ReduzirIM* para o exemplo apresentado na figura 21, gera a tabela 16. Esta tabela é a mesma que foi formada no exemplo 3.1 do capítulo 3.

Tabela 16: Matriz de Intervalos (IMR) gerada pelo algoritmo *ReduzirIM*

IMR			
1			
1	2-3		
1	2-3	4-5	
1	2-3	4-5	6

4.3.16 Algoritmo - *VerificarVertice(Transicao, Coluna)*

O algoritmo *VerificarVertice(Transicao, Coluna)* verifica se um vértice já foi analisado durante o algoritmo de recursividade. Após a análise de um vértice e suas relações, este é armazenado em uma matriz chamada IMVertices. É chamado pelo algoritmo RecursividadeIM.

Código do algoritmo *VerificarVertice(Transicao, Coluna)*

```
início
  resultado_funcao recebe verdadeiro

  para (i recebe valores entre [0, indice]) faça
    se ((IMVertices[i, 0] = Transicao) e (IMVertices[i, 1] = Coluna)) então
      resultado_funcao recebe falso

      sair do laço (i)
    fim se
  fim para

  se (resultado_funcao = verdadeiro) então
    indice recebe indice + 1

    IMVertices[indice, 0] recebe Transicao;
    IMVertices[indice, 1] recebe coluna;
  fim se
fim.
```

4.3.17 Algoritmo - *CriarPT*

Este algoritmo, chamado pelo algoritmo CalcularCaminhos, cria o vetor PT que contém todas as conexões entre as transições iniciais, habilitadas pela marcação inicial, e as transições finais, que levam à marcação final.

Código do algoritmo *CriarPT*

```
início
  inicializar a matriz PT

  para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
    para (j recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se (IMR[i, j] <> '') então
        PT[i] recebe PT[i] + IMR[i, j] + ' '
      fim se
    fim para
  fim para

  repete_laço recebe verdadeiro

  enquanto (repete_laço = verdadeiro) faça
    repete_laço recebe falso

    para (i recebe valores entre [0, quantidade de transições da rede - 1]) faça
      se (PT[i] <> '') então
        para (j recebe valores entre [i + 1, quantidade de transições da rede - 1]) faça
          se (PT[j] <> '') então
            retorno_comparacao recebe CompararCaminho(PT[i], PT[j])

            se (retorno_comparacao = 1) então
              PT[i] recebe ''
            fim se

            se (retorno_comparacao = 2) então
              PT[j] recebe ''
            fim se

            se ((retorno_comparacao = 1) ou (retorno_comparacao = 2)) então
              repete_laço recebe verdadeiro

              sair dos laços (j e i), executando novamente o laço enquanto
            fim se
          fim se
        fim para
      fim se
    fim para
  fim enquanto
fim.
```

A executação do algoritmo *CriarPT* para o exemplo apresentado na figura 21, gera a tabela 17.

Tabela 17: Vetor PT para o exemplo da figura 21, gerado pelo algoritmo *CriarPT*

PT					
1-2-3-4-5-6					

4.3.18 Algoritmo - *CompararCaminho(Vetor1, Vetor2)*

O algoritmo compara os elementos do Vetor1 com os elementos do Vetor2. Retorna 1 se os elementos do Vetor1 estiverem contidos no Vetor2, retorna 2 se os elementos do Vetor2 estiverem contidos no Vetor1 e 0 se ambos falharem. É chamado pelo algoritmo CriarPT para verificar se um caminho está contido em outro.

Código do algoritmo *CompararCaminho(Vetor1, Vetor2)*

```
início
  resultado_funcao recebe 0
  pesquisa recebe verdadeiro

  para (i recebe valores entre [0, quantidade de elementos do Vetor1 - 1]) faça
    se (pesquisa = verdadeiro) então
      pesquisa recebe falso

      para (j recebe valores entre [0, quantidade de elementos do Vetor2 - 1]) faça
        se (Vetor1[i] = Vetor2[j]) então
          pesquisa recebe verdadeiro

          sair do laço mais interno (j)
        fim se
      fim para
    fim se
  fim para

  se (pesquisa = falso) então
    pesquisa recebe verdadeiro

    para (i recebe valores entre [0, quantidade de elementos do Vetor2 - 1]) faça
      se (pesquisa = verdadeiro) então
        pesquisa recebe falso

        para (j recebe valores entre [0, quantidade de elementos do Vetor1 - 1]) faça
          se (Vetor2[i] = Vetor1[j]) então
            pesquisa recebe verdadeiro

            sair do laço mais interno (j)
          fim se
        fim para
      fim se
    fim para

    se (pesquisa = verdadeiro) então
      resultado_funcao recebe 2
    fim se
  senão
    resultado_funcao recebe 1
  fim se
fim.
```

4.4 Conclusão

Neste capítulo foi detalhada a estrutura da ferramenta computacional que implementa o método de tempo global para RPTs, além de oferecer um ambiente gráfico para edição, simulação e apresentação de resultados. As estruturas de dados desenvolvidas refletem os modelos conceituais da metodologia, permitindo, de forma simples, a implementação de novos algoritmos para análise.

5 Estudo de Caso - Sistema Embarcado

5.1 Introdução

O sistema embarcado utilizado como estudo de caso consiste em um sistema de *hardware* composto por uma placa de avaliação eAT55 para arquitetura ARM baseada no processador AT91M55800 da Atmel com *core* ARM7TDMI de 32 *bits*. O *hardware* compreende também SRAM, Flash, ADC, DAC, RTC, Interfaces (seriais, USB, PS/2 e LCD), além de disponibilizar o barramento do processador para placas de expansão. O *core* ARM é utilizado por diferentes fabricantes de processador, tais como, Intel, Motorola, Atmel, Samsung, Sharp, entre outros [ee06]. O protótipo desenvolvido para este estudo de caso pode ser visto na figura 22.

A aplicação de controle do sistema embarcado interage com o *X Real Time Kernel* (um núcleo operacional de tempo real), desenvolvido pela empresa eSysTech. Este núcleo consiste de um módulo de *software* que cumpre o papel de sistema operacional para o sistema embarcado, permitindo a execução de múltiplas tarefas. A principal área de aplicação do *X Real Time Kernel* são os chamados *Deeply Embedded Systems* com severas restrições temporais e de recursos computacionais. É importante salientar que o estudo de caso se baseou no conceito de não preempção do sistema operacional, onde as *threads* são executadas em seqüência pelo escalonador de acordo com o nível de prioridade de cada uma. Ou seja, a *thread* fica em execução enquanto não informar sua liberação ao sistema operacional [eXK06].

Para o desenvolvimento da aplicação do estudo de caso foi utilizado o ambiente de desenvolvimento da *IAR Systems* (o *IAR Embedded Workbench* para ARM). É um ambiente integrado de desenvolvimento com ferramentas para gerenciamento de projetos e um editor, sendo também um compilador C e C++ otimizado [Sys06].

A máquina externa foi emulada por um programa que envia, de forma aleatória, infor-

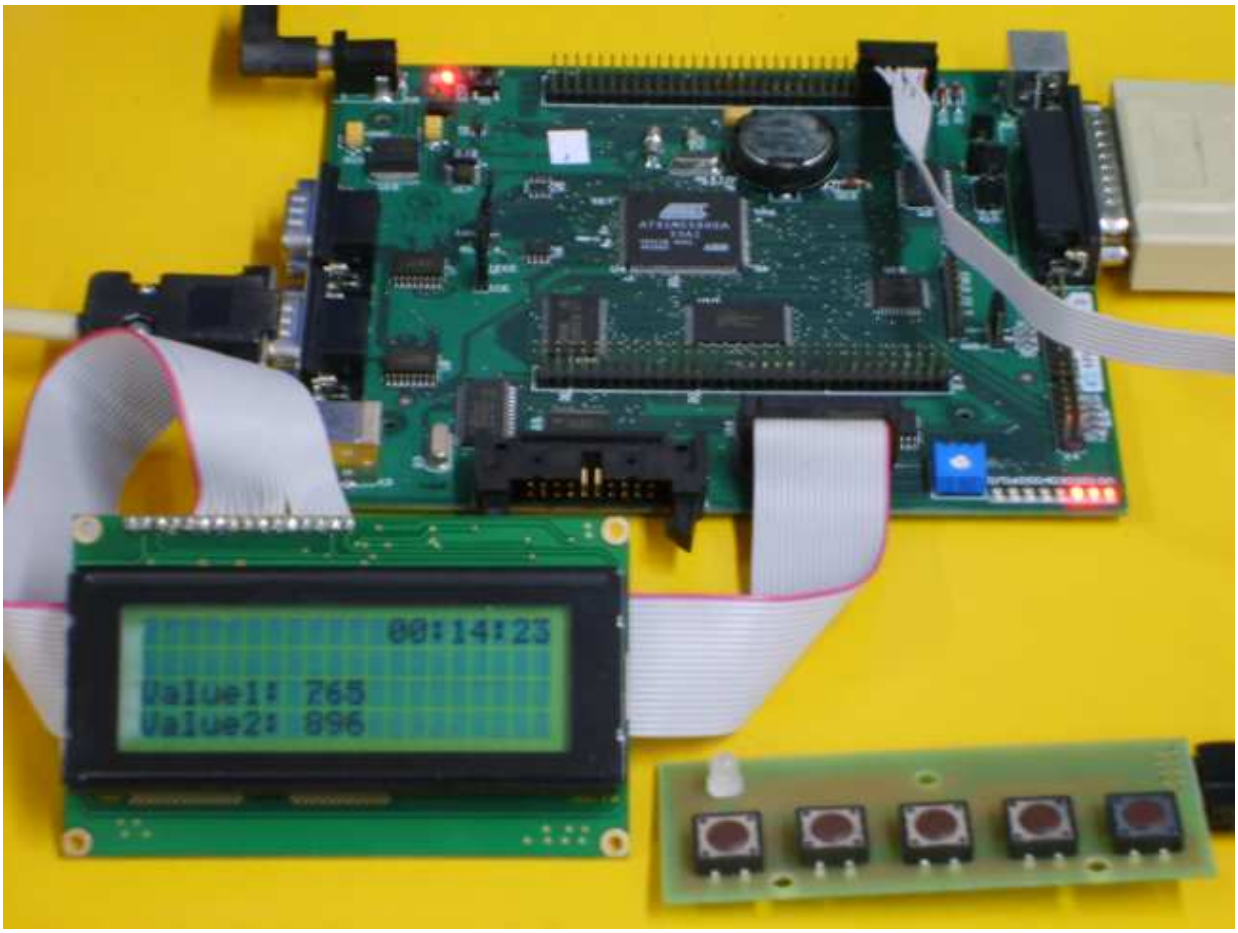

Figura 22: Placa de avaliação eAT55

mações de temperatura e pressão, que consiste de seqüências de 8 à 2048 *bytes* para testar o recebimento de dados no sistema embarcado, além de enviar a condição de ligado ou desligado da máquina externa. Este programa foi desenvolvido em um ambiente integrado de desenvolvimento para o sistema operacional *Windows* com base na linguagem pascal. A figura 23 mostra a tela de emulação do programa.

5.1.1 Escopo da aplicação

A aplicação desenvolvida para o estudo de caso consistiu em um sistema embarcado que mostra informações de temperatura e pressão em seu *display*. Tais informações são enviadas de maneira aleatória por uma máquina externa utilizando comunicação através de uma porta serial. O sistema monitora se a transmissão de dados foi suspensa ou se a máquina externa foi desligada, apresenta mensagens no *display*, mostra a hora e permite que o usuário visualize as últimas informações recebidas. Estas opções são escolhidas a partir de um menu. A figura

Figura 23: Máquina externa - emulada por *software*

24 mostra os casos de uso do sistema embarcado.

O sistema operacional tem como base o uso de *threads* para executar a aplicação e, para tanto, utiliza um algoritmo de escalonamento tipo *Round-Robin*, que organiza a execução das *threads* de forma seqüencial a partir de sua ordem de criação, levando em conta seu nível de prioridade.

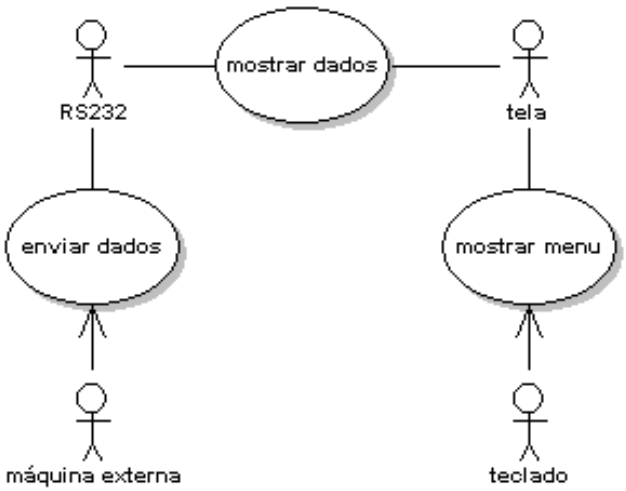

Figura 24: Casos de uso do sistema embarcado

5.1.2 Objetivo do estudo de caso

O objetivo foi traduzir o funcionamento do escalonador, bem como o funcionamento interno de cada *thread* em Redes de Petri Temporais para analisar, no simulador, o comportamento de cada *thread*, bem como do sistema como um todo, verificando o tempo de execução, sincronismo entre as *threads*, e se possível, propor melhorias no *software* embarcado e, principalmente, mostrar o potencial de análise da metodologia de tempo global, conforme o esquema apresentado na figura 25.

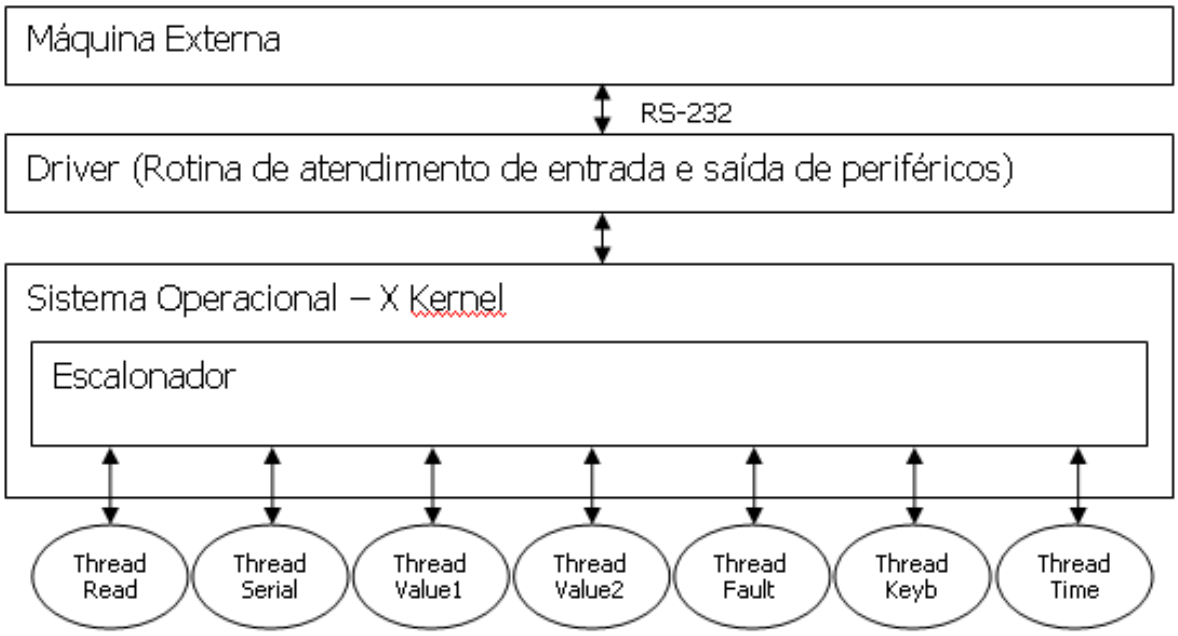

Figura 25: Esquema representativo entre a máquina externa e o sistema embarcado

5.2 Estrutura do *Software*

Para suprir as necessidades do escopo do sistema embarcado proposto, são necessárias diversas rotinas: ler dados da porta serial RS-232, armazenar a informação recebida em uma lista e mostrá-la no *display*, mostrar erros da máquina externa no *display*, mostrar a hora no *display* e ler o teclado. Essas rotinas, para sua execução, dependem da troca de mensagens entre si. A figura 26 mostra as rotinas e as mensagens entre elas.

Cada rotina foi implementada como *thread*. A aplicação embarcada teve portanto inicialmente 6 *threads*:

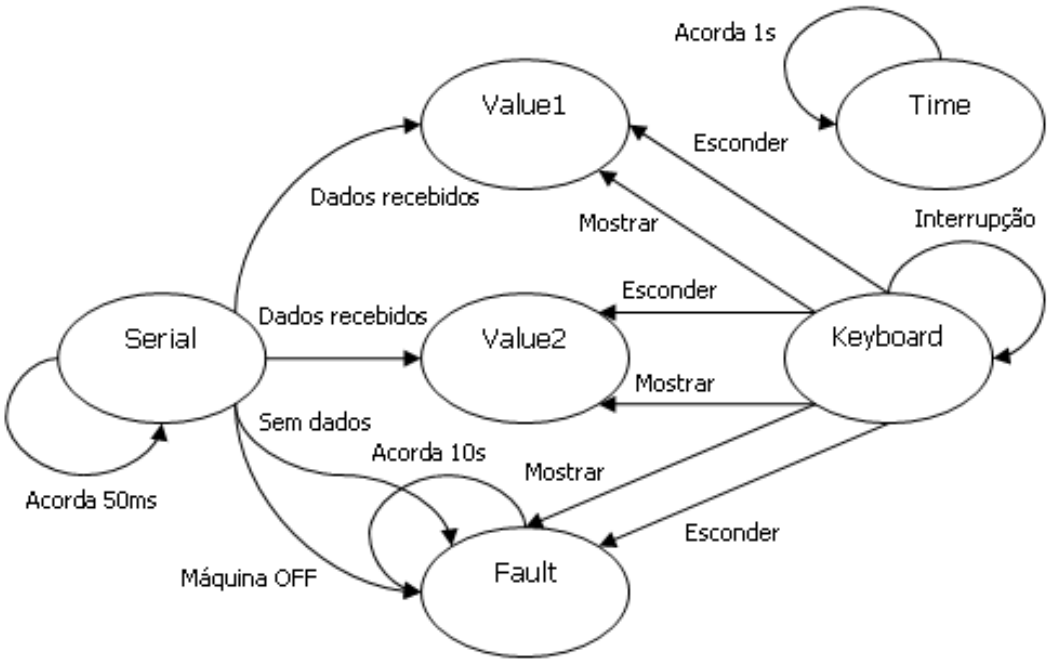

Figura 26: *Threads* que inicialmente foram propostas para o sistema embarcado

- *Thread Serial* - Lê dados da porta serial, processa e encaminha os dados para a *thread* respectiva. Periódica. Acorda automaticamente.
- *Thread Value1* - Armazena a informação da temperatura em uma lista e a mostra no *display*. Esporádica. Executada somente se chegar uma mensagem.
- *Thread Value2* - Armazena a informação da pressão em uma lista e a mostra no *display*. Esporádica. Executada somente se chegar uma mensagem.
- *Thread Fault* - Mostra erro do dispositivo externo no *display*. Esporádica. Executada somente se chegar uma mensagem.
- *Thread Time* - Mostra a hora no *display*. Periódica. Acorda automaticamente.
- *Thread Keyboard* - Lê o teclado. Esporádica. Executada se ocorrer uma interrupção de teclado.

A placa de avaliação disponibiliza uma interface de comunicação serial assíncrona RS-232. Com isso, a partir do *device driver*, pode-se criar um *buffer* para recebimento de dados com tamanho variado. Este *buffer*, figura 27, é circular, ou seja, se a quantidade de dados

recebidos for maior que o tamanho do *buffer* criado, os dados serão sobrepostos. Caracteriza-se dessa forma um fluxo de dados conhecido como *streaming* de *bytes*.

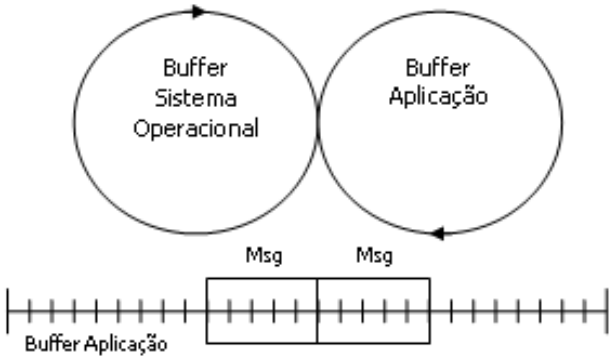

Figura 27: *Buffer* circular

Para garantir que os dados não sejam sobrepostos, mais uma *thread* foi criada. Ela retira os dados do *buffer*, gerenciado pelo sistema operacional, e os coloca em um *buffer* da aplicação, que pode ser melhor gerenciado para evitar a perda de dados. Esta nova *thread*, chamada *Thread Read*, é periódica e acorda automaticamente a cada TTh ms. Este tempo mínimo foi calculado utilizando a seguinte fórmula:

TTh = bu/Tx, onde:

- TTh - Tempo da *thread* em milisegundos (ms)
- bu - *Buffer* do sistema operacional em *bytes* (b)
- Tx - Taxa de transferência em *bytes* por milisegundos (b/ms)

Caso a *thread* seja acordada em 50ms é necessário criar um *buffer* de 240 *bytes*, com a taxa de transferência definida em 38400 *bits*/s, ou seja, 4,8 *bytes*/ms. Assim, esta *thread* tem tempo suficiente para retirar os dados do *buffer* do sistema operacional e colocá-los no *buffer* da aplicação.

A figura 28 mostra o novo conjunto de *threads* do sistema embarcado, onde os arcos representam as mensagens trocadas entre as *threads*.

O simulador foi utilizado para a construção do modelo em Redes de Petri da aplicação embarcada. A primeira rede criada, conforme mostrado na figura 29, representa a troca de mensagens entre as *threads*. Nota-se que as *threads read*, *serial*, *fault* e *time* podem ser

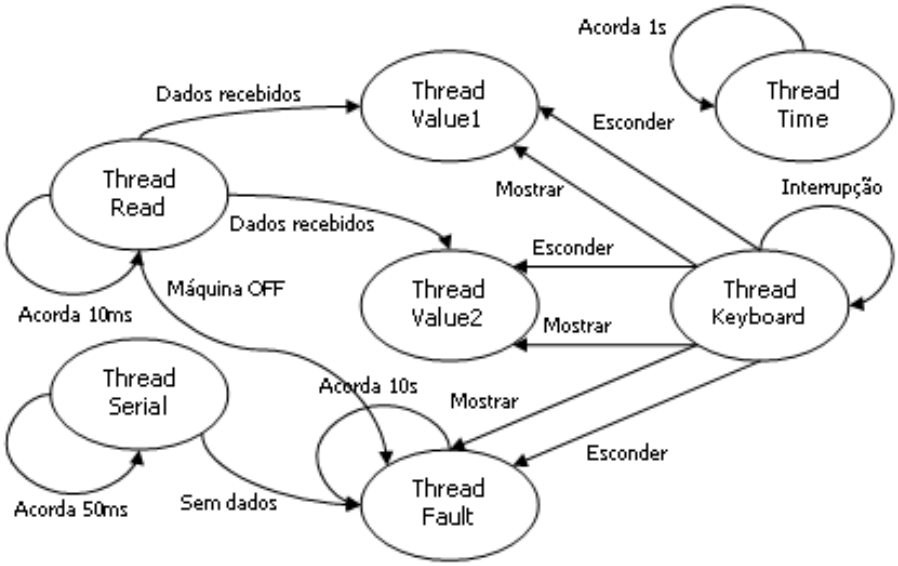

Figura 28: *Threads* do sistema embarcado

acordadas de acordo com a programação temporal do escalonador, respeitando os tempos definidos para cada *thread*. A exceção é a *thread keyboard* que depende do pressionamento de uma tecla para gerar a interrupção de teclado a ser executada. As demais *threads*, *value1* e *value2*, são chamadas através de mensagens disparadas por outras *threads*.

Os intervalos de tempo associados a cada transição correspondem ao tempo de execução de cada *thread*. Para tanto, foi necessário medir, no protótipo, o tempo gasto por cada *thread*, além do tempo consumido pelo escalonador, que corresponde ao tempo entre a finalização de uma *thread* e a chamada de outra. Esses tempos medidos estão apresentados na tabela 18. As transições *wake up read, serial, fault, time* e *keyboard interruption* correspondem a decisões do escalonador e não têm tempo associado.

A segunda rede proposta foi criada para representar o funcionamento do escalonador (figura 30). Ela sequencializa a execução das *threads* através de um escalonador tipo *Round-Robin* Isto corresponde, na prática, a um procedimento que decide os conflitos da rede, mostrados na figura 29, que estavam representados pelas transições t4, t5, t6, t12 e t17.

A rede de Petri que modela o escalonador está representada nas figuras 31 e 32. A simulação do escalonador, na ferramenta, permite ao usuário ter uma noção clara da troca de mensagens entre as *threads*, a partir do sistema operacional.

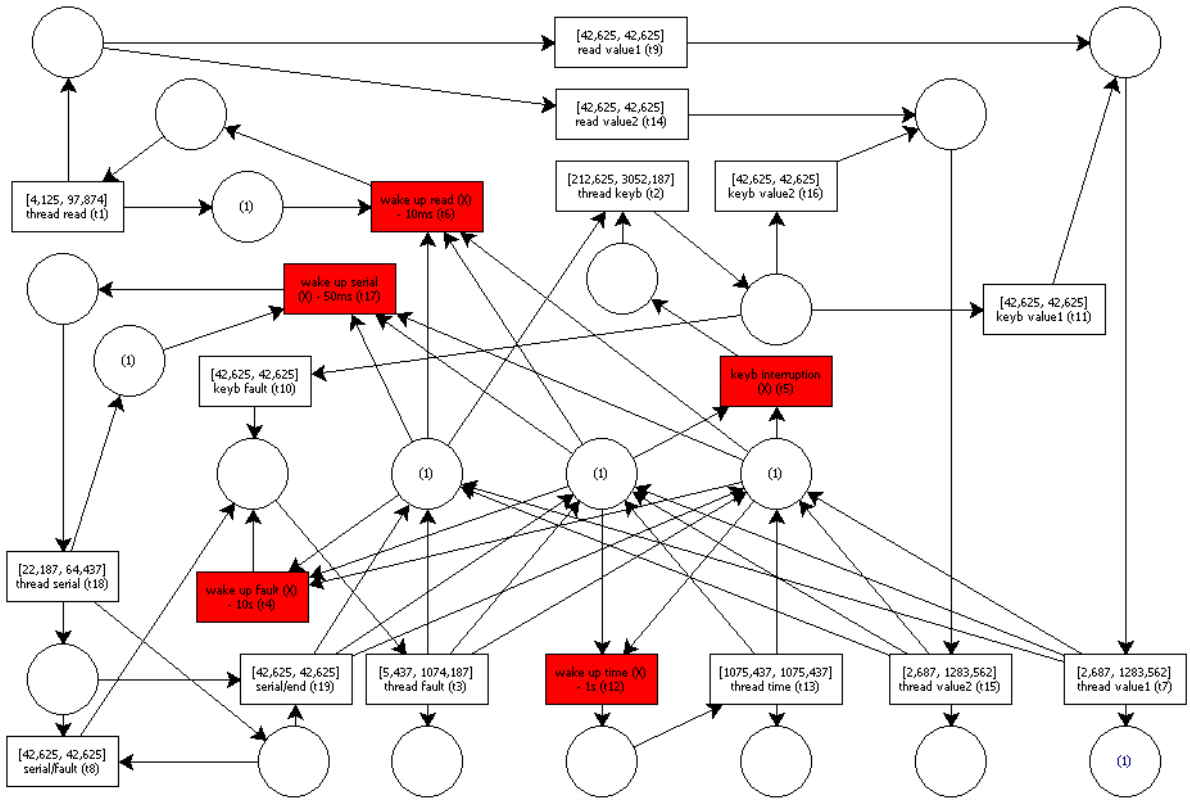

Figura 29: Mensagens entre as *threads*

5.3 *Threads* e Algoritmos

As Redes de Petri a seguir mostram o funcionamento interno de cada *thread*, bem como o algoritmo que representa seu comportamento esperado.

Cada uma das *threads* foi modelada em redes de Petri, sendo que cada algoritmo desenvolvido e implementado foi baseado no seu respectivo modelo.

Cada subseção a seguir mostra o modelo em RdP e o algoritmo correspondente.

5.3.1 *Thread Serial*

A *thread* lê dados da porta serial, processa e encaminha os dados para a *thread* respectiva. É periódica e acorda automaticamente. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 33 e o seu código é apresentado a seguir.

Tabela 18: Tempos medidos na execução do sistema embarcado

Thread	**Condição**	**Tempo (ms)**
serial	lendo 0 *bytes*	22,187
serial	lendo 8 *bytes* da RS-232	64,437
read	*buffer* da aplicação vazio	4,125
read	*buffer* da aplicação com 8 *bytes*	97,874
value1 e *value2*	tratar mensagem MOSTRAR	2,687
value1 e *value2*	tratar mensagem ESCONDER	3,437
value1 e *value2*	mostrar dados no *display*	1283,562
fault	tratar a mensagem MOSTRAR	5,437
fault	tratar a mensagem ESCONDER	6,187
fault	tratar a mensagem MÁQUINA DESLIGADA	132,312
fault	tratar a mensagem ACORDAR	422,187
fault	mostrar dados no *display*	1074,187
keyboard	sair do menu principal	212,625
keyboard	sair do menu histórico	213,375
keyboard	sair do menu histórico e voltar para o menu principal	2333,875
keyboard	selecionar o menu histórico	2515,062
keyboard	selecionar o menu principal	3052,187
time	mostrar a hora no *display*	1075,437
-	tempo entre a saída de uma *thread* e a entrada em outra	42,625

Código do algoritmo da *Thread Serial*

```
zerar fault

enquanto (verdade) faça
  receber mensagem
  ler dados da serial e colocar no buffer da aplicação

  se (sem dados) então faça
    fault recebe fault + 1

    se (fault = 8) então faça
      enviar mensagem para a Thread Fault
      zerar fault
    fim se
  senão
    zerar fault
  fim se
fim enquanto
```

5.3.2 *Thread Read*

A *thread* retira os dados do *buffer*, gerenciado pelo sistema operacional, e os coloca em um *buffer* da aplicação, que pode ser melhor gerenciado para evitar a perda de dados. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 34 e o seu código é apresentado a seguir.

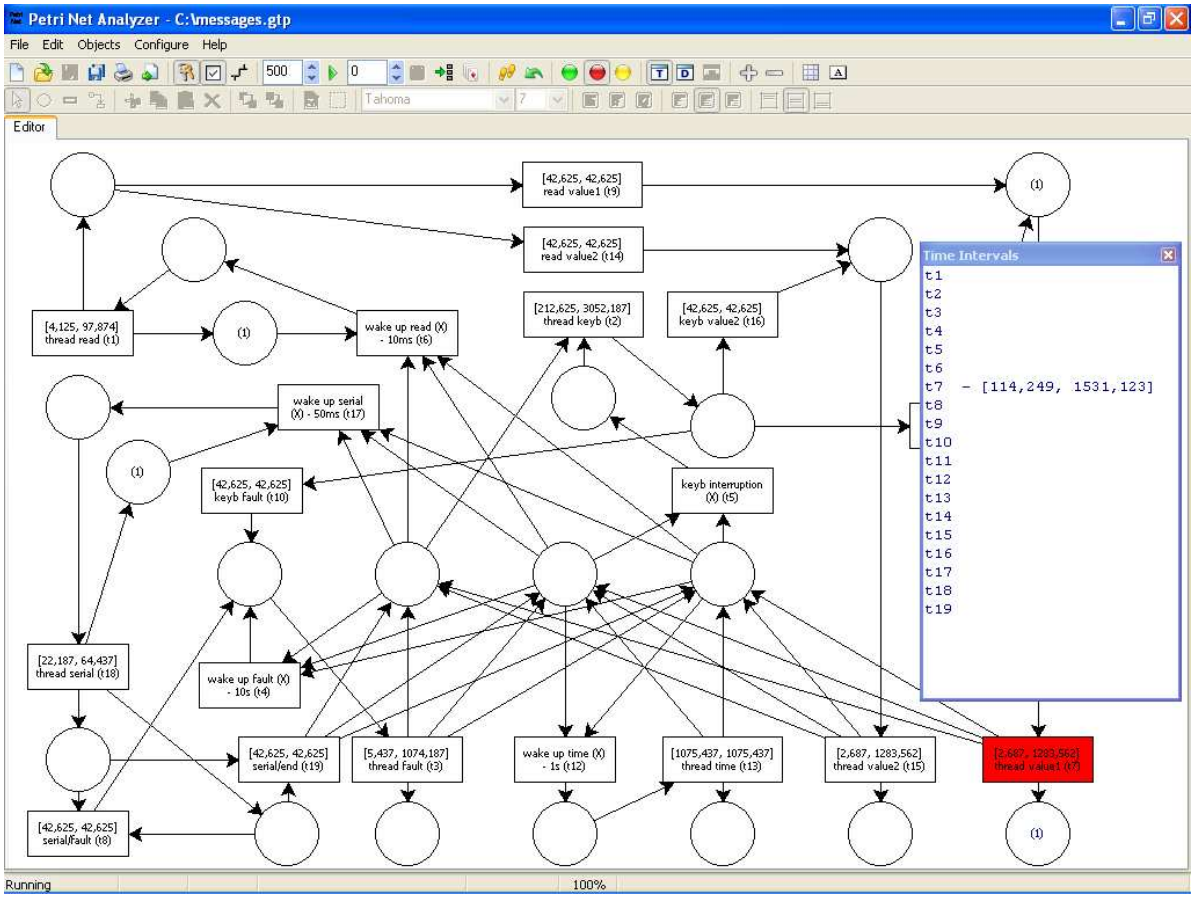

Figura 30: Análise temporal da RdP da figura 29

Código do algoritmo da *Thread Read*

```
enquanto (verdade) faça
  receber mensagem

  se (mensagem no buffer) então
    se (máquina desligada) então
      enviar mensagem para a Thread Fault
    fim se

    se (temperatura) então
      enviar mensagem para a Thread Value1
    fim se

    se (pressão) então
      enviar mensagem para a Thread Value2
    fim se

    retirar mensagem do buffer
  fim se
fim enquanto
```

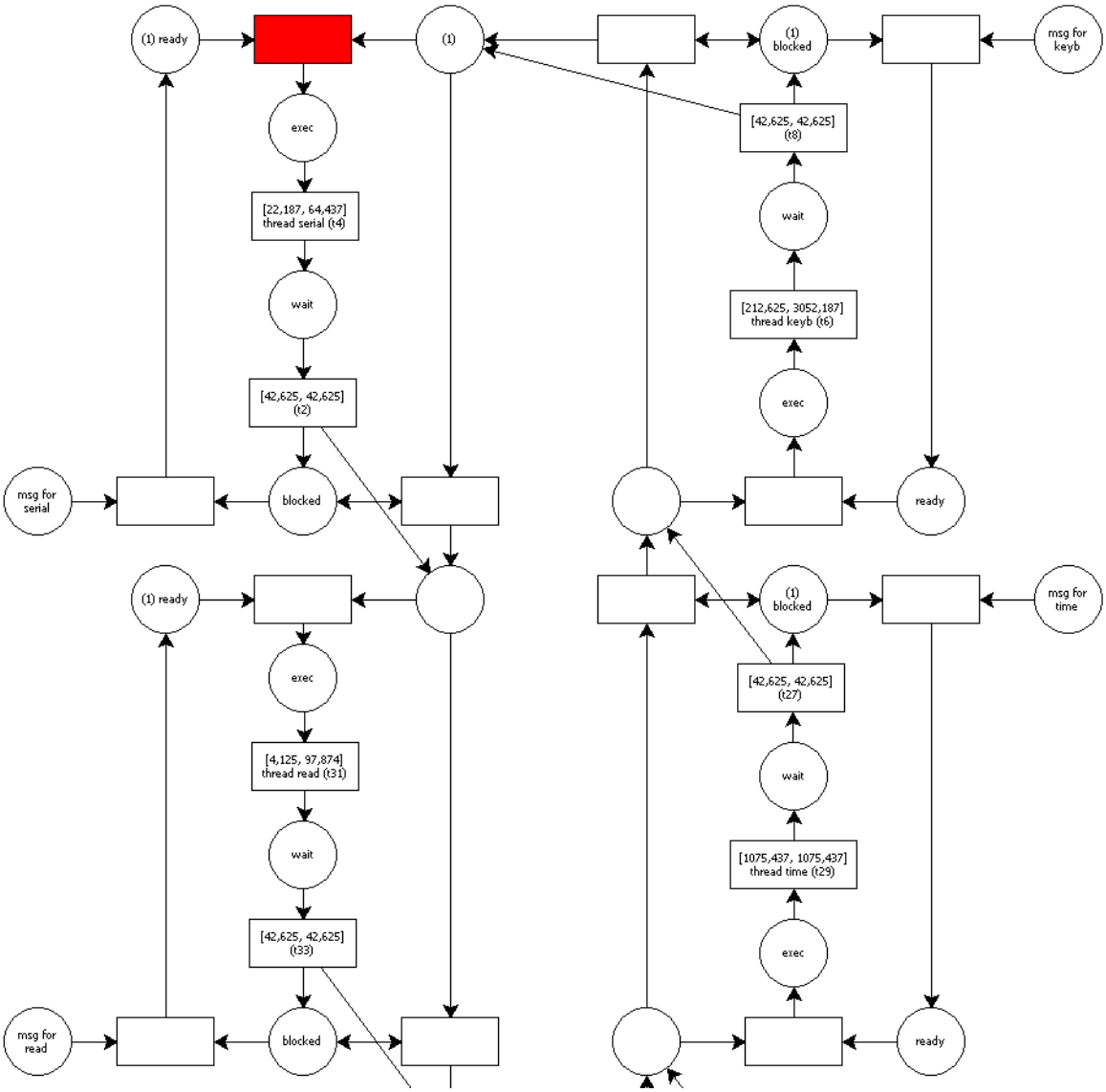

Figura 31: Escalonador - parte 1

5.3.3 *Thread Value1*

A *thread* armazena a informação da temperatura em uma lista e a mostra no *display*. É esporádica, sendo executada somente se chegar uma mensagem. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 35 e o seu código é apresentado a seguir.

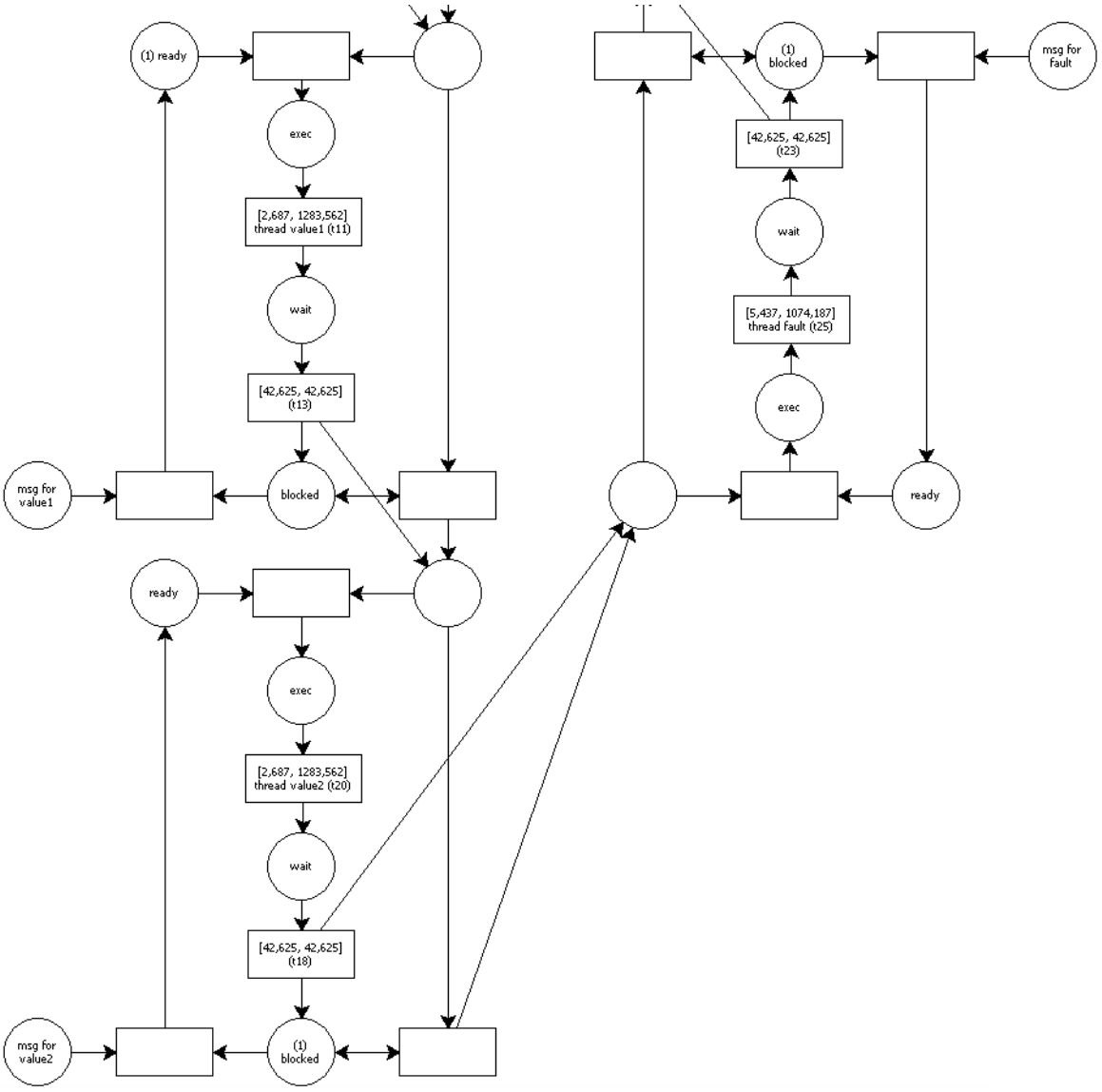

Figura 32: Escalonador - parte 2

Código do algoritmo da *Thread Value1*

```
enquanto (verdade) faça
  receber mensagem

  se (mostrar mensagem) então
    ativar a visualização da temperatura no display
  fim se

  se (esconder mensagem) então
    desativar a visualização da temperatura no display
  fim se

  se (dados recebidos) então
    mostrar a temperatura no display
    armazenar a informação no histórico
  fim se
fim enquanto
```

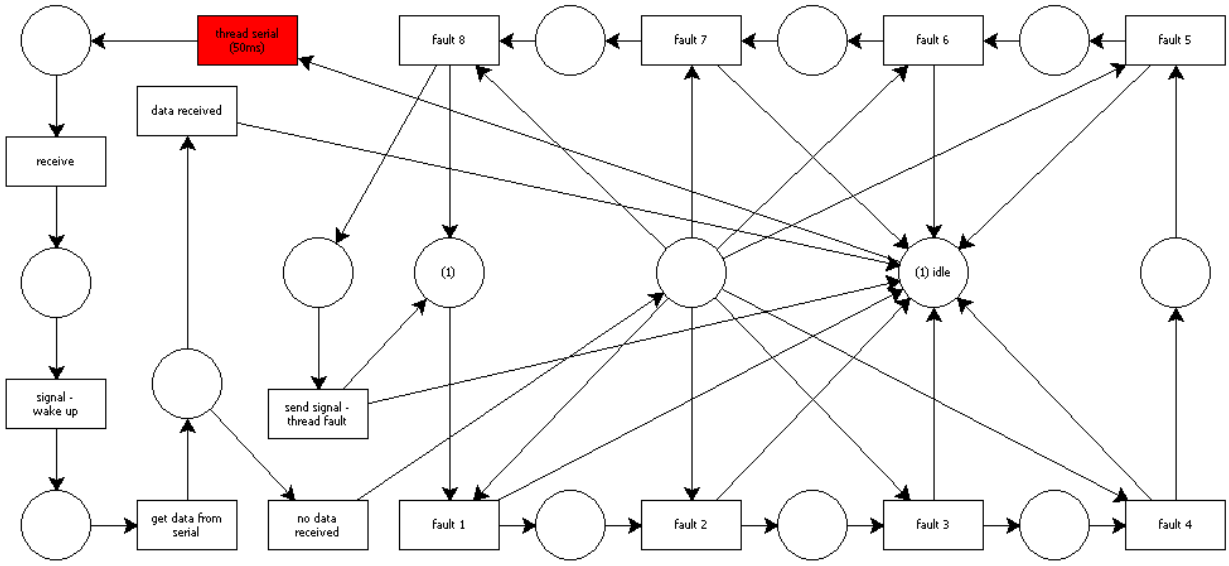

Figura 33: *Thread Serial*

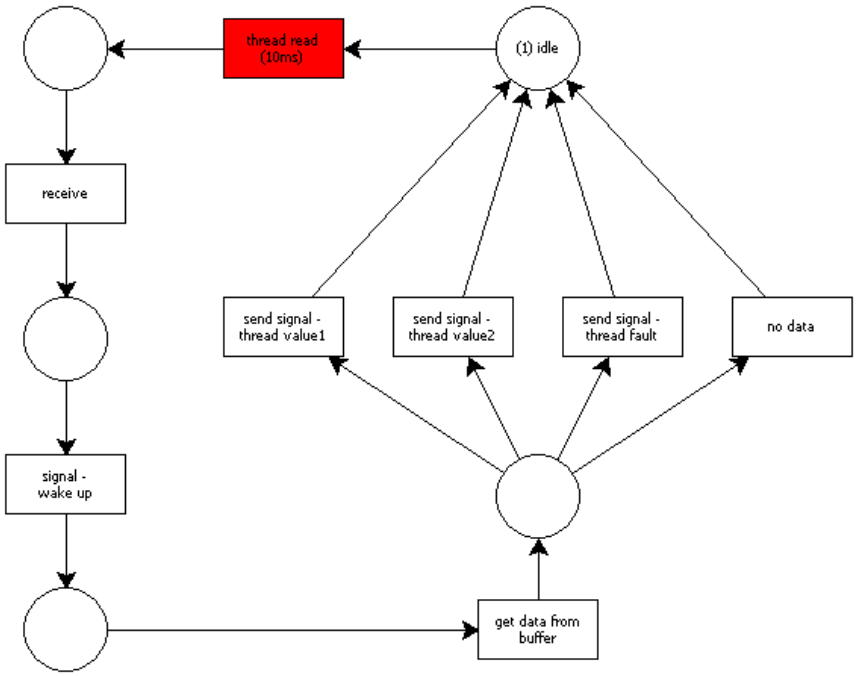

Figura 34: *Thread Read*

5.3.4 *Thread Value2*

A *thread* armazena a informação da pressão em uma lista e a mostra no *display*. É esporádica, sendo executada somente se chegar uma mensagem. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 36 e o seu código é apresentado a seguir.

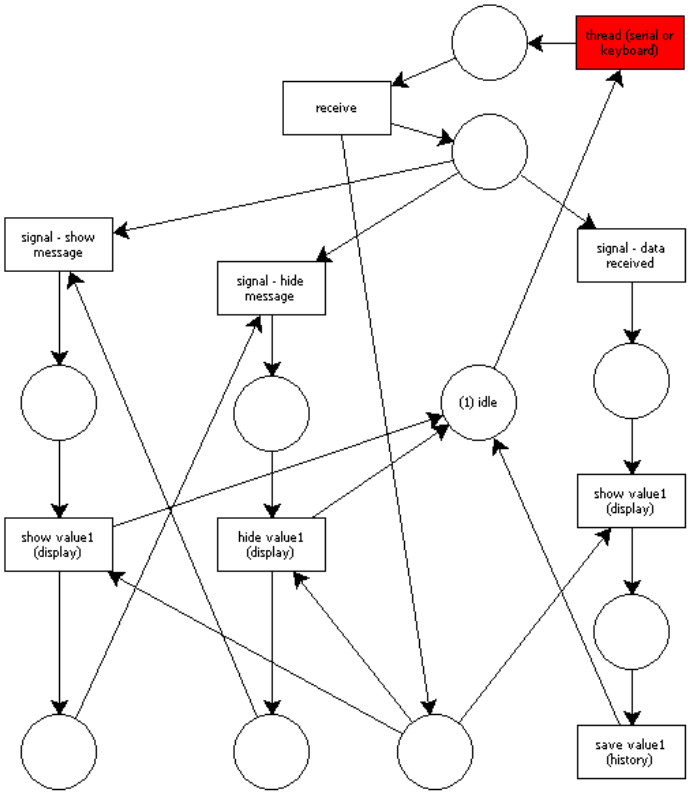

Figura 35: *Thread Value1*

Código do algoritmo da *Thread Value2*

```
enquanto (verdade) faça
  receber mensagem

  se (mostrar mensagem) então
    ativar a visualização da pressão no display
  fim se

  se (esconder mensagem) então
    desativar a visualização da pressão no display
  fim se

  se (dados recebidos) então
    mostrar a pressão no display
    armazenar a informação no histórico
  fim se
fim enquanto
```

5.3.5 *Thread Fault*

A *thread* mostra erro do dispositivo externo no *display*. É Esporádica, sendo executada somente se chegar uma mensagem. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 37 e o seu código é apresentado a seguir.

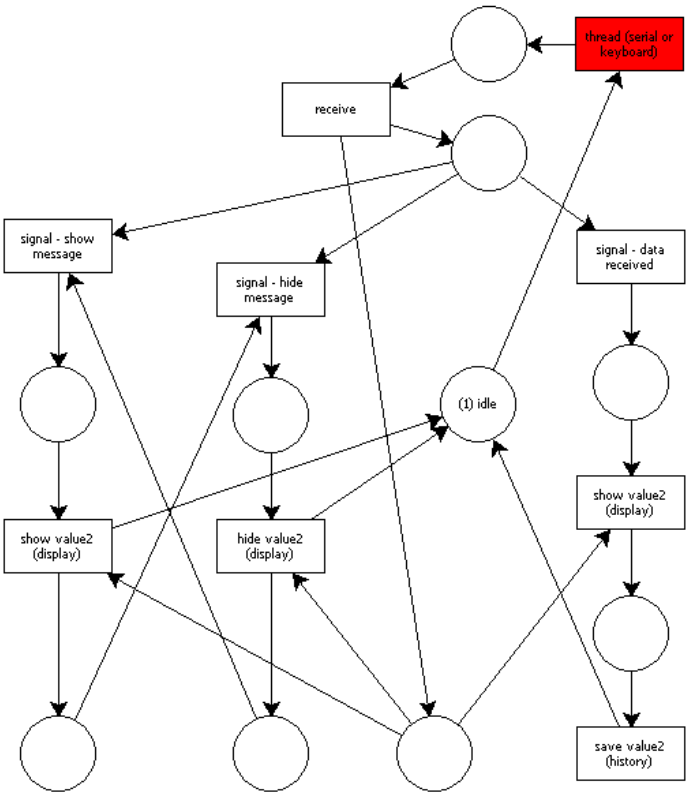

Figura 36: *Thread Value2*

Código do algoritmo da *Thread Fault*

```
enquanto (verdade) faça
  receber mensagem

  se (mostrar mensagem) então
    ativar a visualização da quantidade de ciclos que a serial esteve ociosa
  fim se

  se (esconder mensagem) então
    desativar a visualização da quantidade de ciclos que a serial esteve ociosa
  fim se

  se (acordar) então
    limpar a área de visualização da quantidade de ciclos
  fim se

  se (máquina desligada) então
    mostrar a informação ?Off? no display
  fim se

  se (sem dados) então
    somar a quantidade de ciclos
    mostrar a quantidade de ciclos no display
  fim se
fim enquanto
```

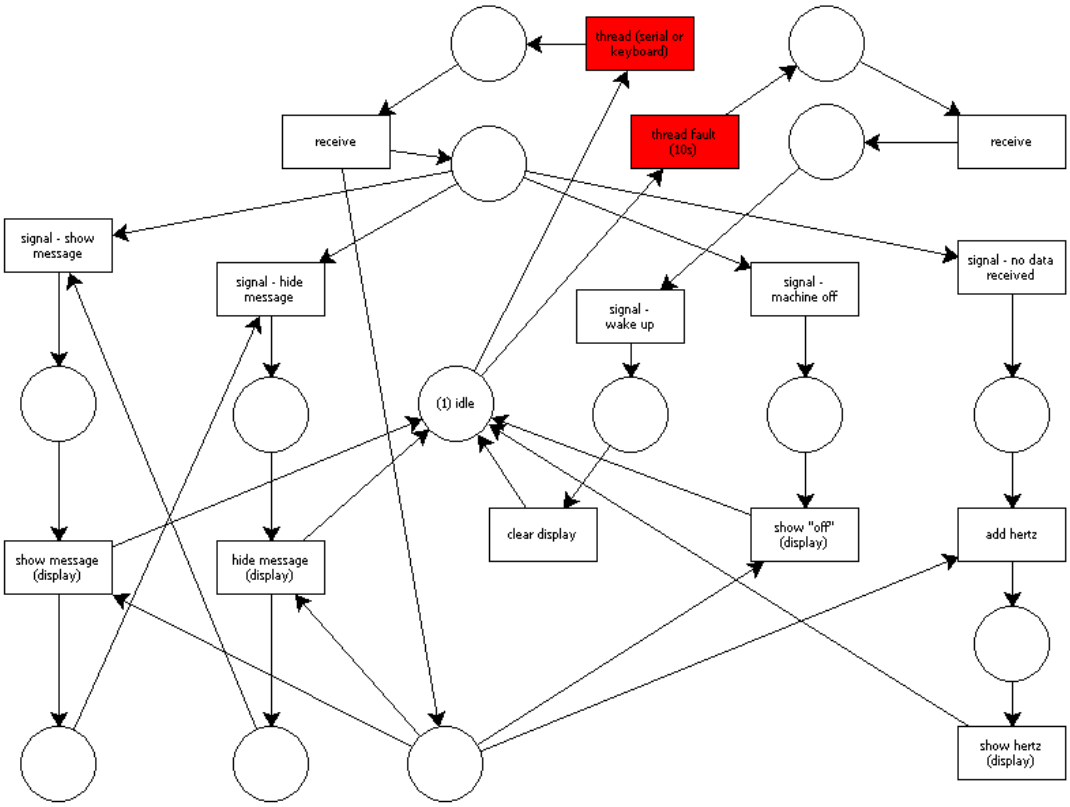

Figura 37: *Thread Fault*

5.3.6 *Thread Time*

A *thread* mostra a hora no *display*. É Periódica e acorda automaticamente. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 38 e o seu código é apresentado a seguir.

Código do algoritmo da *Thread Time*

```
enquanto (verdade) faça
  receber mensagem

  se (mostrar mensagem) então
    ativar a visualização da hora
  fim se

  se (esconder mensagem) então
    desativar a visualização da hora
  fim se

  se (acordar) então
    pegar a hora do sistema
    mostrar a hora no display
  fim se
fim enquanto
```

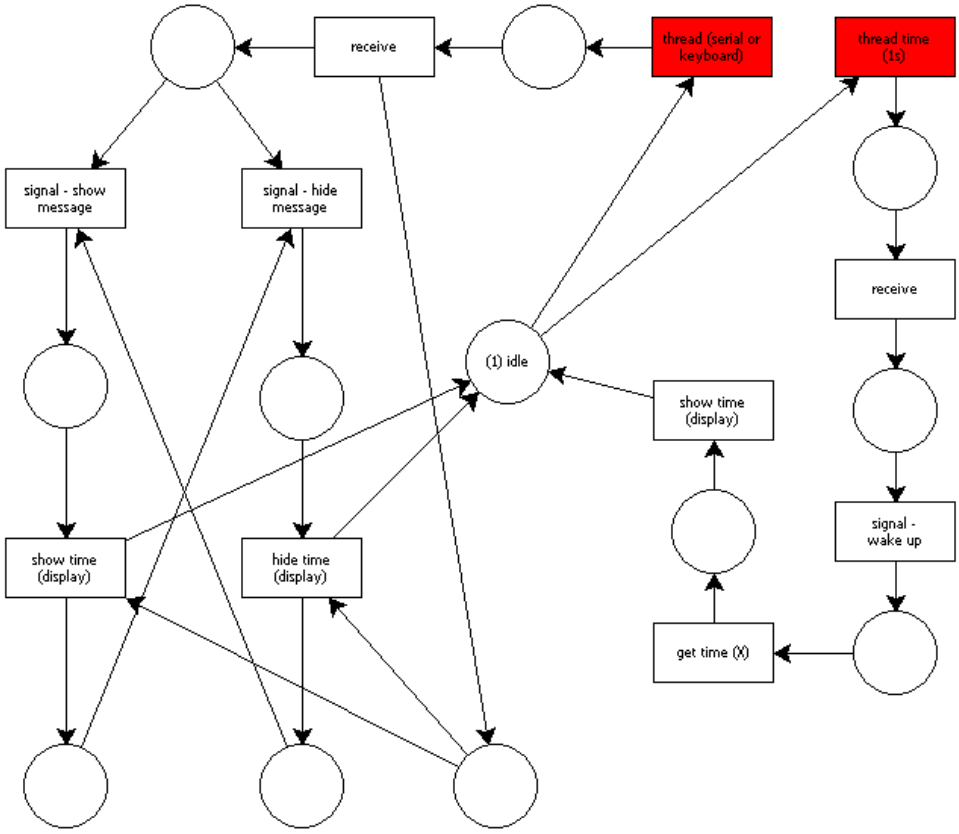

Figura 38: *Thread Time*

5.3.7 *Thread Keyboard*

A *Thread Keyboard* foi, primeiramente, construída utilizando o conceito de máquina de estados para a escolha de opções a partir de um menu, guardando o estado atual no qual se encontra a escolha do usuário. O código deste algoritmo está apresentado a seguir.

Código do algoritmo da *Thread Keyboard* utilizando máquina de estados

```
estado recebe ocioso

enquanto (verdade) faça
  receber mensagem

  se (estado = ocioso)
    se (tecla = 1) então
      enviar mensagem para desativar a visualização da temperatura para a Thread Value1
      enviar mensagem para desativar a visualização da pressão para a Thread Value2
      enviar mensagem para desativar a visualização da quantidade de ciclos
        para a Thread Fault
      mostrar o menu principal

      estado recebe menu_principal
    fim se
  fim se
```

```
se (estado = menu_principal) então
  se (tecla = 1) então
    mostrar o menu histórico

    estado recebe menu_historico
  fim se

  se (tecla = 2) então
    enviar mensagem para ativar a visualização da temperatura para a Thread Value1
    enviar mensagem para ativar a visualização da pressão para a Thread Value2
    enviar mensagem para ativar a visualização da quantidade de ciclos
      para a Thread Fault

    estado recebe ocioso
  fim se
fim se

se (estado = menu_historico) então
  se (tecla = 1) então
    mostrar a lista de temperaturas armazenadas no histórico

    estado recebe lista_temperatura
  fim se

  se (tecla = 2) então
    mostrar a lista de pressões armazenadas no histórico

    estado recebe lista_pressao
  fim se

  se (tecla = 3) então
    mostrar o menu principal

    estado recebe menu_principal
  fim se

  se (tecla = 4) então
    enviar mensagem para ativar a visualização da temperatura para a Thread Value1
    enviar mensagem para ativar a visualização da pressão para a Thread Value2
    enviar mensagem para ativar a visualização da quantidade de ciclos
      para a Thread Fault

    estado recebe ocioso
  fim se
fim se

se (estado = lista_temperatura) então
  se (tecla = 1) então
    rolar a lista de valores para cima
  fim se

  se (tecla = 2) então
    rolar a lista de valores para baixo
  fim se

  se (tecla = 3) então
    mostrar o menu histórico

    estado recebe menu_historico
  fim se

  se (tecla = 4) então
    enviar mensagem para ativar a visualização da temperatura para a Thread Value1
    enviar mensagem para ativar a visualização da pressão para a Thread Value2
    enviar mensagem para ativar a visualização da quantidade de ciclos
      para a Thread Fault

    estado recebe ocioso
  fim se
fim se
```

```
  se (estado = lista_pressao) então
    se (tecla = 1) então
      rolar a lista de valores para cima
    fim se

    se (tecla = 2) então
      rolar a lista de valores para baixo
    fim se

    se (tecla = 3) então
      mostrar o menu histórico

      estado recebe menu_historico
    fim se

    se (tecla = 4) então
      enviar mensagem para ativar a visualização da temperatura para a Thread Value1
      enviar mensagem para ativar a visualização da pressão para a Thread Value2
      enviar mensagem para ativar a visualização da quantidade de ciclos
        para a Thread Fault

      estado recebe ocioso
    fim se
  fim se
fim enquanto
```

Quando da modelagem do comportamento desta *thread* utilizando o formalismo das Redes de Petri, notou-se que este algoritmo poderia ser construído de forma diferente, atendendo de maneira mais simples o funcionamento esperado. Isto mostra que o uso de um formalismo adequado permite construir de maneira mais eficiente algoritmos que representam o comportamento esperado das *threads*. O modelo do comportamento da *thread* está modelado pela rede de Petri da figura 39 e o seu código é apresentado a seguir.

Código do algoritmo da *Thread Keyboard* baseado na RdP

```
enquanto (verdade) faça
  receber mensagem

  se (tecla = 1) então
    enviar mensagem para desativar a visualização da temperatura para a Thread Value1
    enviar mensagem para desativar a visualização da pressão para a Thread Value2
    enviar mensagem para desativar a visualização da quantidade de ciclos
      para a Thread Fault

    repete recebe verdadeiro

    enquanto (repete = verdadeiro) faça
      mostrar o menu principal
      receber mensagem

      caso (tecla) faça
        1:
          enquanto (repete = verdadeiro) faça
            mostrar o menu histórico
            receber mensagem

            caso (tecla) então
              1:
                mostrar a lista de temperaturas armazenadas no histórico

                enquanto (repete = verdadeiro) faça
```

```
                receber mensagem

                caso (tecla) faça
                  1:
                    rolar a lista de valores para cima
                  2:
                    rolar a lista de valores para baixo
                  3:
                    sair do laço
                  4:
                    repete recebe falso
                fim caso
              fim enquanto
            2:
              mostrar a lista de pressões armazenadas no histórico

              enquanto (repete = verdadeiro) faça
                receber mensagem

                caso (tecla) faça
                  1:
                    rolar a lista de valores para cima
                  2:
                    rolar a lista de valores para baixo
                  3:
                    sair do laço
                  4:
                    repete recebe falso
                fim caso
              fim enquanto
            3:
              sair do laço
            4:
              repete recebe falso
          fim caso
        fim enquanto
      2:
        repete recebe falso
    fim caso
  fim enquanto
fim se

enviar mensagem para ativar a visualização da temperatura para a Thread Value1
enviar mensagem para ativar a visualização da pressão para a Thread Value2
enviar mensagem para ativar a visualização da quantidade de ciclos
  para a Thread Fault
fim enquanto
```

5.4 Análise Temporal

Para exemplificar o uso da ferramenta em análise temporal, tomemos como base a RdP que modela a troca de mensagens entre as *threads*, com seus respectivos tempos de execução conforme a figura 29. A geração do grafo de classes dessa rede permite obter o grafo mostrado na figura 40. Neste grafo os nodos representam marcações e os arcos os disparos de transições que ocasionam a mudança de marcação. Associado ainda a cada arco estão os intervalos de tempo que delimitam as datas mínima e máxima para o disparo da respectiva transição.

A partir do grafo é possível reconstruir todos os cenários possíveis de comportamento do

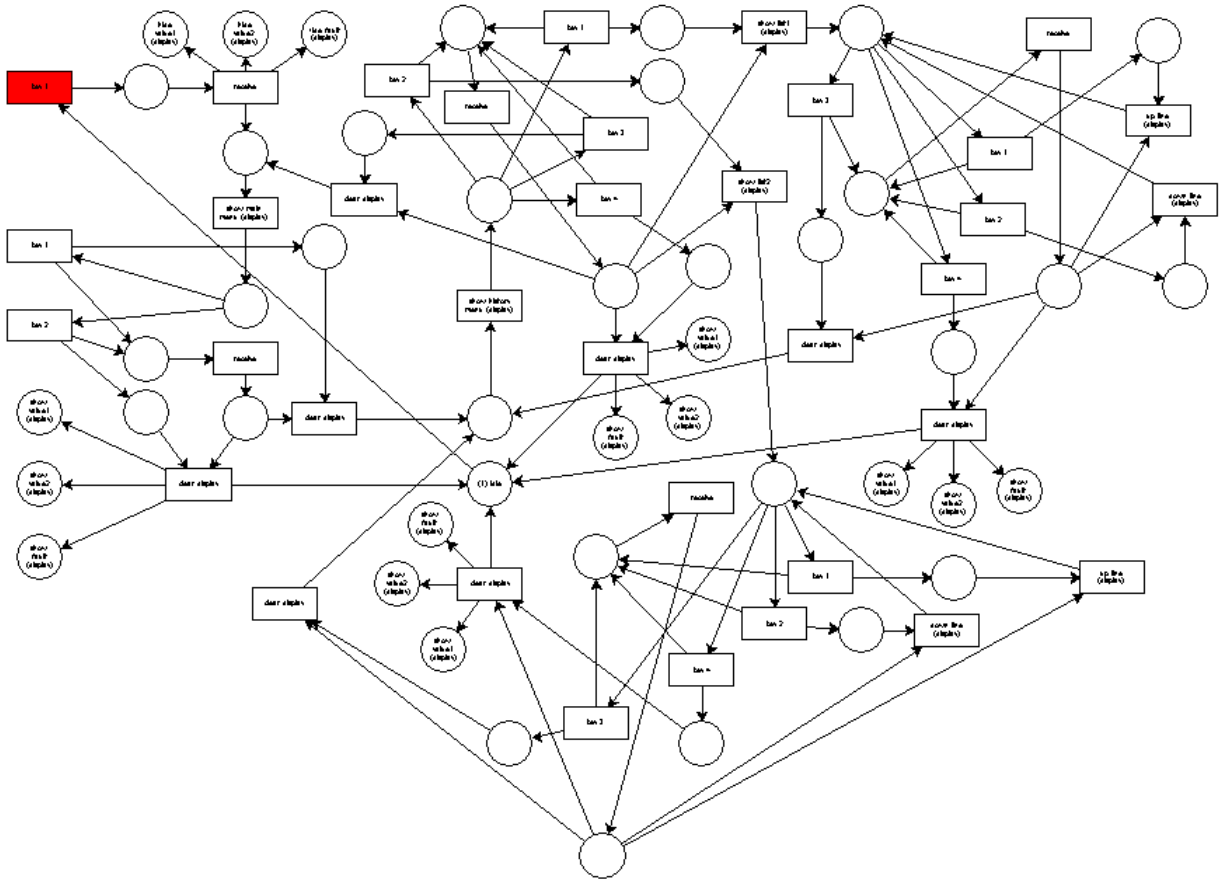

Figura 39: *Thread Keyboard*

sistema e calcular os intervalos de tempo que representam a duração destes. Por exemplo, a partir do protótipo, o tempo medido para a leitura de 8 *bytes* na interface serial, a gravação destes no *buffer* da aplicação, a leitura deste *buffer*, sua análise e a apresentação do resultado no *display* consome 1531,562 ms. Este cenário corresponde a dois ciclos sucessivos do grafo. O primeiro com os disparos de {t17, t18, t19} e o segundo com {t6, t1, t9, t7}. A soma dos intervalos produz como resultado [114, 1532] conforme pode ser verificado na figura 30, que corresponde a análise temporal deste cenário. O mesmo resultado poderia ser obtido com a simulação e análise da RdP do escalonador.

5.5 Conclusão

O estudo de caso apresentado neste capítulo mostrou duas grandes vantagens do uso da ferramenta. A primeira está associada ao uso de um método formal dedicado a sistemas dinâmicos a eventos discretos para a modelagem de sistemas embarcados baseados em *threads*.

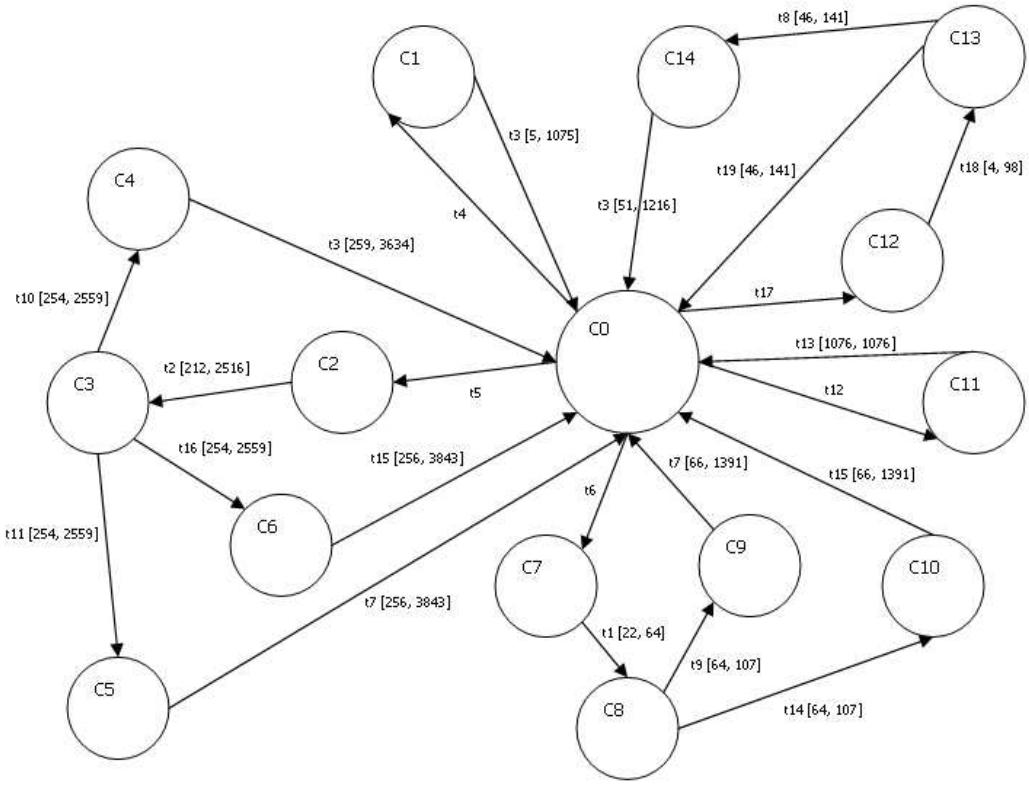

Figura 40: Grafo de classes da figura 29

O modelo formal obtido possibilita a geração de algoritmos melhor estruturados, conforme mostrado na seção 5.3.7[1]. A segunda está na análise temporal baseada em tempo global, que fornece intervalos temporais mais precisos, o que permite avaliar se as restrições temporais do sistema são satisfeitas pelo modelo.

[1]Todo o código fonte do sistema embarcado está apresentado no Apêndice.

6 Conclusão e Trabalhos Futuros

Este trabalho teve como objetivos a implementação de uma ferramenta computacional para edição e análise de Redes de Petri Temporais e a validação do uso desta ferramenta através de um estudo de caso. No ambiente de edição são oferecidos ao usuário mecanismos gráficos para inclusão, exclusão e cópia de lugares, transições e arcos, tratados individualmente ou de forma agrupada. A ferramenta permite ao usuário jogar a rede tanto em modo passo-a-passo, no qual são mostradas as transições habilitadas a serem disparadas e o usuário escolhe qual evento deve ocorrer, quanto em modo automático, no qual as transições são disparadas desde que habilitadas e um mecanismo de escolha aleatória decide as situações de conflito. A análise temporal do comportamento leva em consideração os intervalos temporais de sensibilização associados às transições. Esta análise foi implementada tendo por base a metodologia de tempo global, onde um relógio global, zerado no início da simulação, data os disparos das transições. A metodologia de tempo global, desenvolvida no Laboratório de Sistemas Inteligentes de Produção do CPGEI, utiliza os conceitos e fundamentos da álgebra intervalar. Os resultados da simulação da rede são apresentados ao usuário na forma de intervalos temporais que representam a duração mínima e máxima de cenários de simulação ou na forma enumerativa, gerando um grafo de classes cujos nodos representam estados da rede com mesma marcação e intervalos de disparo de transições equivalentes.

A análise temporal impôs a criação de estruturas de dados que fossem capazes de representar a parte estrutural das redes de Petri, através de matrizes e vetores de inteiros. Além disso, foram necessárias matrizes e vetores intervalares, tanto para a representação dos intervalos de sensibilização das transições quanto para os algoritmos e resultados parciais da metodologia de tempo global. No texto da dissertação foram apresentados os principais algoritmos que traduzem os métodos de análise implementados na atual versão da ferramenta. As estruturas de dados adotadas permitem que novos métodos sejam implementados e não se limitam à metodologia de tempo global.

A validação da ferramenta foi efetuada através de um estudo de caso que consistiu na

modelagem do comportamento de um sistema embarcado com restrições temporais. A ferramenta foi utilizada para modelar o comportamento de cada uma das *threads* que executam as diferentes funções do sistema e para modelar o comportamento do escalonador de tarefas utilizado para ativar as diferentes *threads*. Para obter os diferentes tempos associados aos eventos do sistema, um modelo real foi implementado, desde a estrutura de *hardware*, que consistiu em uma placa de avaliação eAT55 para arquitetura ARM, até a geração do código das *threads* e do escalonador. Este código foi construído baseado nos modelos em redes de Petri desenvolvidos na ferramenta computacional e foi compilado junto com o sistema operacional X e embarcado na placa. Os intervalos de tempo utilizados para a análise temporal foram medidos no sistema embarcado real. A primeira versão da ferramenta gerou um trabalho aceito na Sessão *Tools* da Conferência *QEST 2005* [KLK05].

O estudo de caso mostrou que a ferramenta é adequada para aplicações reais de modelagem e análise de sistemas dinâmicos a eventos discretos com restrições temporais. Os resultados obtidos com a ferramenta permitem validar os modelos tanto comportamental quanto temporalmente. Em termos comportamentais, a modelagem das diferentes tarefas de um sistema em redes de Petri permitem a geração de algoritmos melhor estruturados. No aspecto temporal, a análise dos diferentes caminhos do grafo de classes permite identificar se os ciclos mais longos violam ou não as restrições temporais da aplicação.

A estrutura de dados concebida para a ferramenta permite que novos algoritmos sejam incluídos de forma simples. Na atual versão da ferramenta apenas uma parte da metodologia de tempo global foi implementada. Como trabalhos futuros, deverão ser incluídos novos métodos de análise da metodologia, como por exemplo a redução do grafo de classes, a análise de alcançabilidade temporal, entre outros. Além disso, as técnicas clássicas de análise estrutural por invariantes e por enumeração do espaço de estados.

ANEXO – Ferramentas de Redes de Petri

As RdP são um formalismo que pode ser representado graficamente, permitindo visualizar diversos comportamentos do sistema modelado, como por exemplo concorrência, situações de bloqueio, recursos compartilhados, sincronização, entre outros.

Na sua forma mais simples, permitem a descrição qualitativa dos sistemas. Com o desenvolvimento de extensões às RdPs, nomeadamente através da introdução de eventos e sinais externos, as RdPs tornaram-se uma ferramenta importante de modelação de modelos físicos.

Todo este potencial oferecido pelas RdPs, só poderá ser concretizado na resolução de problemas reais, se a sua utilização for suportada por ferramentas computacionais, que permitam, entre outras funcionalidades:

- A utilização de uma interface gráfica para a criação de modelos;
- A possibilidade de simulação dos modelos com animação gráfica de marcas;
- A possibilidade de realização de simulações rápidas;
- Informação estatística sobre o desempenho do sistema modelado;
- A análise de propriedades das RdP, como por exemplo espaço de estados e invariantes;

Existem numerosas ferramentas baseadas em RdP para a análise e simulação gráfica de sistemas a eventos discretos. A sua apresentação exaustiva está fora do âmbito desta dissertação. No entanto, considera-se importante apresentar, de forma sucinta, algumas das ferramentas atualmente disponíveis para a modelação baseada em RdP. A avaliação das ferramentas de Redes de Petri apresentadas neste Anexo foram baseadas em [Pai04].

As tabelas 19, 20, 21 e 22 apresentam diversas informações relativas as 20 ferramentas de análise e de simulação de RdP Temporal, atualmente disponíveis [Wor06].

Nas classes de redes suportadas, o significado da nomemclatura utilizada é o seguinte:

- RdP básicas - RdPs cujas marcas não têm qualquer informação associada (não são coloridas) e que são representadas por pontos negros (*black tokens*);
- RdP temporizadas - RdPs que permitem a modelação de tempo, podendo ser utilizadas na análise do desempenho de sistemas; os tempos envolvidos são, em princípio, fixos;
- RdP estocásticas - RdPs em que as transições podem ter tempos estocásticos associados, modelados por variáveis aleatórias, normalmente descritas por funções de densidade de probabilidade exponenciais;
- RdP estocáticas generalizadas - RdP que possibilita a definição de tempos de disparo das transições fixos ou estocásticos, e suportando eventualmente a definição de tipos de dados abstratos;
- RdP Lugar/Transição - RdP na qual é permitida a existência de múltiplas marcas nos lugares e mais do que um arco entre um lugar e uma transição ou entre uma transição e um lugar;
- RdP Coloridas - RdP de alto-nível, em que as marcas possuem estruturas de dados associadas e os arcos possuem expressões associadas que determinam as marcas envolvidas no disparo das transições;
- RdP Orientadas a Objetos - RdP com suporte para conceitos orientados a objetos.

Nas principais características das redes suportadas, o significado da nomenclatura utilizada é o seguinte:

- Editor gráfico - ferramenta que possibilita a construção de redes de Petri.
- Simulação com animação - possibilidade de simular o modelo através da simulação um passo de cada vez;
- Simulação rápida - possibilidade de simular o modelo sem interação por parte do usuário;
- Espaço de estados - Conjunto de todos os estados para os quais a rede pode evoluir;

- Invariantes de Lugar - Um invariante de lugar é uma função linear dependente da marcação dos lugares, cujo valor é uma constante que depende apenas da marcação inicial da rede;

- Invariantes de Transição - Um invariante de transição é uma sequência de disparos de transições que não modifica a marcação da rede;

- Redução de redes - técnica que permite reduzir o tamanho da rede, visando diminuir a complexidade da análise, através da aplicação de regras de redução, de forma que, a rede inicial e a rede reduzida satisfazem um mesmo conjunto de propriedades;

- Análise de desempenho Simples - a análise de desempenho simples permite o acesso a estatísticas relativas às RdP simuladas, tais como, número médio de marcas em uma posição, número de marcas que entraram e saíram de uma posição, número de vezes que uma transição foi disparada, entre outras;

- Análise de desempenho Avançado - a análise de desempenho avançada permite além das funcionalidades da "Análise de desempenho Simples", a definição de estatísticas relativas ao sistema particular que está para ser modelado, a sua exportação para outras aplicações e pode também incluir a sua representação gráfica.

- Análise estrutural - análise baseada na matriz de incidência da rede de Petri, para verificar algumas das suas propriedade;

- Geração de Código - geração de código para execução e ou análise de propriedades da rede de Petri;

- Interchange File Format - formato baseado em XML para intercâmbio de informação entre aplicações.

Tabela 19: Principais características das Aplicações para Redes de Petri Temporais

	ALPHA/Sim	Artiflex	CPN Tools	ExSpect	F-net
Classes de RdP suportadas					
RdP de Alto Nível	X	X	X	X	
RdP Estocásticas				X	X
RdP Lugar/Transição		X		X	
RdP Orientadas a Objetos		X			
RdP Outros tipos					
RdP Temporizadas	X	X	X	X	X
Principais Características					
Editor Gráfico	X	X	X	X	X
Simulação com animação	X	X	X	X	X
Simulação rápida	X	X	X	X	X
Espaços de estados			X		X
Invariantes de Lugar					X
Invariantes de Transição					X
Redução					
Análise de desempenho simples	X		X	X	X
Análise de desempenho avançado		X		X	X
Análise estrutural		X			X
Geração de código					
Interchange File Format			X		
Outras				X	

Tabela 20: Principais características das Aplicações para Redes de Petri Temporais

	GreatSPN	HPSim	Income Suite	JFern	Opera
Classes de RdP suportadas					
RdP de Alto Nível	X		X	X	
RdP Estocásticas	X	X			
RdP Lugar/Transição		X	X	X	
RdP Orientadas a Objetos				X	
RdP Outros tipos					
RdP Temporizadas	X	X	X		X
Principais Características					
Editor Gráfico	X	X	X	X	X
Simulação com animação	X	X	X	X	X
Simulação rápida	X	X	X	X	X
Espaços de estados	X			X	X
Invariantes de Lugar	X				X
Invariantes de Transição	X				X
Redução					X
Análise de desempenho simples	X	X		X	X
Análise de desempenho avançado	X				X
Análise estrutural					X
Geração de código					
Interchange File Format				X	X
Outras	X		X		

Tabela 21: Principais características das Aplicações para Redes de Petri Temporais

	PACE	PEP	Petri .NET Simulator	PN Toolbox	PNTalk
Classes de RdP suportadas					
RdP de Alto Nível	X	X			X
RdP Estocásticas	X			X	
RdP Lugar/Transição	X	X	X	X	
RdP Orientadas a Objetos					
RdP Outros tipos	X			X	
RdP Temporizadas	X	X	X	X	X
Principais Características					
Editor Gráfico	X	X	X	X	X
Simulação com animação	X	X	X	X	X
Simulação rápida	X		X	X	X
Espaços de estados		X		X	
Invariantes de Lugar		X		X	
Invariantes de Transição		X		X	
Redução	X	X			
Análise de desempenho simples		X	X	X	X
Análise de desempenho avançado				X	
Análise estrutural				X	
Geração de código					
Interchange File Format		X		X	
Outras	X	X	X	X	

Tabela 22: Principais características das Aplicações para Redes de Petri Temporais

	Renew	SEA	TimeNET	Tina	Visual Object Net ++
Classes de RdP suportadas					
RdP de Alto Nível	X	X	X		
RdP Estocásticas			X		
RdP Lugar/Transição	X		X	X	X
RdP Orientadas a Objetos	X				
RdP Outros tipos				X	X
RdP Temporizadas	X	X	X	X	X
Principais Características					
Editor Gráfico	X	X	X	X	X
Simulação com animação	X	X	X	X	X
Simulação rápida	X	X	X	X	X
Espaços de estados			X	X	
Invariantes de Lugar			X	X	
Invariantes de Transição				X	
Redução					
Análise de desempenho simples			X		X
Análise de desempenho avançado			X		
Análise estrutural			X		X
Geração de código					
Interchange File Format	X		X		
Outras	X	X		X	

APÊNDICE – Código Fonte do Sistema Embarcado

```
#include "stdio.h"
#include "DDLX\DDLX_atmel_devices.h"
#include "DDLX\DDLX_atmel_rtc.h"
#include "DDLX\DDLX_LCD.h"
#include "eSysTech\DD_keyb.h"
#include "eSysTech\DD_serial_232.h"
#include "X\xl.h"
#include "X\x_shell.h"

X os(15, 5, 0);
CX_CommPort dbg_comm(8*1024, 8, 32, 300);

#ifdef X_TRACE
  CX_Trace x_trace(11, TR_END);
#endif

enum EDisplayState {TURN_OFF = 0,  TURN_ON};
enum EMessageType  {SHOW_MESSAGE = 1, HIDE_MESSAGE, RECEIVED_DATA, NO_RECEIVED_DATA, MACHINE_OFF};
enum EMenuState    {IDLE = -1, OPT_MAIN, OPT_HISTORY, LIST_VALUE1, LIST_VALUE2};

Byte   g_led = 0;
Byte   g_buffer[6144];
Card8  g_off = 0;
Card16 g_count = 0;
Int16  g_p = 0;

const Card8 cFAULT = 20;

// Classe C_DISPLAY - Classe base para controle do display

class C_DISPLAY {
  public:
    Card8 turn;
  public:
    C_DISPLAY();

    void clear();
    void onoff(Card8 pvalue);
};

C_DISPLAY::C_DISPLAY() {
  turn = 1;
  g_lcd.Command(LCD_DISP_ON);
  g_lcd.Command(LCD_CLEAR);
};

void C_DISPLAY::clear() {
  g_lcd.Command(LCD_CLEAR);
};

void C_DISPLAY::onoff(Card8 pvalue) {
  turn = pvalue;
};

// Classe C_DISPLAYTIME - Classe para mostrar a hora no display
```

```
class C_DISPLAYTIME : public C_DISPLAY {
  public:
    void cleararea();
    void show(char *pvalue);
};

void C_DISPLAYTIME::cleararea() {
  if (turn) {
    g_lcd.GoToLineCol(0, 12);
    g_lcd.WriteString("        ");
  }
}

void C_DISPLAYTIME::show(char *pvalue) {
  if (turn) {
    cleararea();
    g_lcd.GoToLineCol(0, 12);
    g_lcd.WriteString(pvalue);
  }
};

// Classe C_DISPLAYVALUE1 - Classe para mostrar um valor

class C_DISPLAYVALUE1 : public C_DISPLAY {
  public:
    void cleararea();
    void show(char *pvalue);
};

void C_DISPLAYVALUE1::cleararea() {
  if (turn) {
    g_lcd.GoToLineCol(2, 0);
    g_lcd.WriteString("                    ");
  }
}

void C_DISPLAYVALUE1::show(char *pvalue) {
  if (turn) {
    cleararea();
    g_lcd.GoToLineCol(2, 0);
    g_lcd.WriteString("Value1:");
    g_lcd.GoToLineCol(2, 8);
    g_lcd.WriteString(pvalue);
  }
};

// Classe C_DISPLAYVALUE2 - Classe para mostrar um valor

class C_DISPLAYVALUE2 : public C_DISPLAY {
  public:
    void cleararea();
    void show(char *pvalue);
};

void C_DISPLAYVALUE2::cleararea() {
  if (turn) {
    g_lcd.GoToLineCol(3, 0);
    g_lcd.WriteString("                    ");
  }
}

void C_DISPLAYVALUE2::show(char *pvalue) {
  if (turn) {
    cleararea();
    g_lcd.GoToLineCol(3, 0);
    g_lcd.WriteString("Value2:");
    g_lcd.GoToLineCol(3, 8);
    g_lcd.WriteString(pvalue);
  }
};
```

```
// Classe C_DISPLAYFAULT - Classe para mostrar uma falha

class C_DISPLAYFAULT : public C_DISPLAY {
  public:
    void cleararea();
    void show(char *pvalue);
    void showoff();
};

void C_DISPLAYFAULT::cleararea() {
  if (turn) {
    g_lcd.GoToLineCol(0, 0);
    g_lcd.WriteString("           ");
  }
}

void C_DISPLAYFAULT::show(char *pvalue) {
  if (turn) {
    cleararea();
    g_lcd.GoToLineCol(0, 0);
    g_lcd.WriteString(pvalue);
  }
};

void C_DISPLAYFAULT::showoff() {
  if (turn) {
    g_lcd.GoToLineCol(3, 17);
    g_lcd.WriteString("OFF");
  }
}

// Classe C_DISPLAYMENU - Classe para mostrar o menu de opções

class C_DISPLAYMENU : public C_DISPLAY {
  public:
    void showop_main();
    void showop_history();
    void exit();
};

void C_DISPLAYMENU::showop_main() {
  clear();
  g_lcd.GoToLineCol(0, 0);
  g_lcd.WriteString("MAIN");
  g_lcd.GoToLineCol(2, 0);
  g_lcd.WriteString("1. History 2. Exit");
};

void C_DISPLAYMENU::showop_history() {
  clear();
  g_lcd.GoToLineCol(0, 0);
  g_lcd.WriteString("HISTORY");
  g_lcd.GoToLineCol(2, 0);
  g_lcd.WriteString("1. Value1 2. Value2");
  g_lcd.GoToLineCol(3, 0);
  g_lcd.WriteString("3. Back 4. Exit");
};

void C_DISPLAYMENU::exit() {
  clear();
};

// Thread Serial - Lê dados da porta serial e coloca em um buffer da aplicação
// Periódica, trabalha diretamente com o Hardware

void threadSerial(Word arg1, Word arg2) {
  Card8 *led_ptr = (Card8 *) 0x7300000;
  Card8 fault = 0;
  Int16 aval, read;
  PutMsg msg;
```

```
  os.PeriodicMsg(50*MSEC);

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    g_led ^= 1;
    *led_ptr = g_led;

    aval = g_serial_232.StartRx();
    read = g_serial_232.ReadBytes(&g_buffer[g_p], aval);
    g_p += read;

    if (aval == 0) {
      fault += 1;

      if (fault == cFAULT) {
        msg.command = NO_RECEIVED_DATA;
        os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
        fault = 0;
      }
    } else {
      fault = 0;
    }
  }
}

// Thread Read - Lê dados do buffer da aplicação e encaminha via mensagem para a Thread respectiva
// Periódica

void threadRead(Word arg1, Word arg2) {
  Card8 *led_ptr = (Card8 *) 0x7300000;
  Int16 i, l;
  PutMsg msg;

  os.PeriodicMsg(10*MSEC);

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    g_led ^= 2;
    *led_ptr = g_led;

    if ((g_p != 0) && (g_buffer[0] <= g_p)) {
      l = g_buffer[0];

      if ((char) g_buffer[1] == 'X') {
        msg.command = MACHINE_OFF;
        os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
        g_off = 1;
      } else {
        msg.byte_array[0] = RECEIVED_DATA;

        for (i=2; i<l; i++) {
          msg.byte_array[i-1] = g_buffer[i];
        }

        msg.byte_array[i-1] = '\0';

        if ((char) g_buffer[1] == 'T') {
          os.Put(os.GetTId("Value1"), (PutMsg *) &msg);
          g_off = 0;
        } else if ((char) g_buffer[1] == 'P') {
          os.Put(os.GetTId("Value2"), (PutMsg *) &msg);
          g_off = 0;
        } else if ((char) g_buffer[1] == 'M') {
          os.Put(os.GetTId("Keyboard"), (PutMsg *) &msg);
        }
      }

      for (i=l; i<g_p; i++) {
```

```
        g_buffer[i-l] = g_buffer[i];
      }

      g_p -= l;
    }
  }
}

// Thread Value1 - Armazena seu valor em uma lista e mostra no display
// Esporádica, executa somente se chegar uma mensagem

void threadValue1(Word arg1, Word arg2) {
  C_DISPLAYVALUE1 g_displayvalue1;
  Card8 command;
  PutMsg msg;

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    command = msg.byte_array[0];

    if (command == SHOW_MESSAGE) {
      g_displayvalue1.onoff(TURN_ON);
    } else if (command == HIDE_MESSAGE) {
      g_displayvalue1.onoff(TURN_OFF);
    } else if (command == RECEIVED_DATA) {
      g_displayvalue1.show((char *) &msg.byte_array[1]);
    }
  }
}

// Thread Value2 - Armazena seu valor em uma lista e mostra no display
// Esporádica, executa somente se chegar uma mensagem

void threadValue2(Word arg1, Word arg2) {
  C_DISPLAYVALUE2 g_displayvalue2;
  Card8 command;
  PutMsg msg;

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));
    command = msg.byte_array[0];

    if (command == SHOW_MESSAGE) {
      g_displayvalue2.onoff(TURN_ON);
    } else if (command == HIDE_MESSAGE) {
      g_displayvalue2.onoff(TURN_OFF);
    } else if (command == RECEIVED_DATA) {
      g_displayvalue2.show((char *) &msg.byte_array[1]);
    }
  }
}

// Thread Fault - Mostra erro do dispositivo externo no display
// Esporádica, executa somente se chegar uma mensagem

void threadFault(Word arg1, Word arg2) {
  C_DISPLAYFAULT g_displayfault;
  Card8 *led_ptr = (Card8 *) 0x7300000;
  Card32 command;
  char *value;
  PutMsg msg;

  os.PeriodicMsg(10000*MSEC);

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    command = msg.command;

    g_led ^= 4;
```

```
    *led_ptr = g_led;

    if (command == SHOW_MESSAGE) {
      g_displayfault.onoff(TURN_ON);
    } else if (command == HIDE_MESSAGE) {
      g_displayfault.onoff(TURN_OFF);
    } else if (command == WAKE_UP_MSG) {
      g_displayfault.cleararea();
    } else if (command == MACHINE_OFF) {
      g_displayfault.showoff();
    } else if (command == NO_RECEIVED_DATA) {
      g_count++;
      sprintf(value, "NO DATA-%03d", g_count);
      g_displayfault.show(value);
    }
  }
}

// Thread Time - Mostra a hora no display
// Periódica

void threadTime(Word arg1, Word arg2) {
  C_DISPLAYTIME g_displaytime;
  Card8 *led_ptr = (Card8 *) 0x7300000;
  Card32 command;
  Calendar calTime;
  char *time;
  PutMsg msg;

  os.PeriodicMsg(1000*MSEC);

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    command = msg.command;

    g_led ^= 8;
    *led_ptr = g_led;

    if (command == SHOW_MESSAGE) {
      g_displaytime.onoff(TURN_ON);
    } else if (command == HIDE_MESSAGE) {
      g_displaytime.onoff(TURN_OFF);
    } else if (command == WAKE_UP_MSG) {
      calTime = g_rtc.GetTimeDate();
      sprintf(time, "%02d:%02d:%02d", calTime.hour, calTime.minute, calTime.second);
      g_displaytime.show(time);
    }
  }
}

// Thread Keyboard - Lê o teclado
// Esporádica, executa se acontecer uma interrupção de teclado
// Executa ações de acordo com uma máquina de estado

void threadKeyboard(Word arg1, Word arg2) {
  C_DISPLAYMENU g_displaymenu;
  Int8 state = IDLE;
  char option;
  PutMsg msg;

  while (1) {
    os.Receive(&msg, sizeof(PutMsg));

    option = (char) msg.byte_array[1];

    if (state == IDLE) {
      if (option == '1') {
        msg.byte_array[0] = HIDE_MESSAGE;
        os.Put(os.GetTId("Value1"), (PutMsg *) &msg);
        os.Put(os.GetTId("Value2"), (PutMsg *) &msg);
```

```
        msg.command = HIDE_MESSAGE;
        os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
        g_displaymenu.showop_main();
        state = OPT_MAIN;
      }
    } else if (state == OPT_MAIN) {
      if (option == '1') {
        g_displaymenu.showop_history();
        state = OPT_HISTORY;
      }

      if (option == '2') {
        g_displaymenu.exit();
        msg.byte_array[0] = SHOW_MESSAGE;
        os.Put(os.GetTId("Value1"), (PutMsg *) &msg);
        os.Put(os.GetTId("Value2"), (PutMsg *) &msg);
        msg.command = SHOW_MESSAGE;
        os.Put(os.GetTId("Fault"), (PutMsg *) &msg);

        if (g_off == 1) {
          msg.command = MACHINE_OFF;
          os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
        }

        state = IDLE;
      }
    } else if (state == OPT_HISTORY) {
      if (option == '3') {
        g_displaymenu.showop_main();
        state = OPT_MAIN;
      }

      if (option == '4') {
        g_displaymenu.exit();
        msg.byte_array[0] = SHOW_MESSAGE;
        os.Put(os.GetTId("Value1"), (PutMsg *) &msg);
        os.Put(os.GetTId("Value2"), (PutMsg *) &msg);
        msg.command = SHOW_MESSAGE;
        os.Put(os.GetTId("Fault"), (PutMsg *) &msg);

        if (g_off == 1) {
          msg.command = MACHINE_OFF;
          os.Put(os.GetTId("Fault"), (PutMsg *) &msg);
        }

        state = IDLE;
      }
    }
  }
}

int main() {
  *(Byte*) 0x7300000 = 0x0;

  os.Init();
  g_lcd.Init();
  g_serial_232.Init(38400);

  os.CreateThread(threadSerial, 0, 0, "Serial", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadRead, 0, 0, "Read", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadValue1, 0, 0, "Value1", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadValue2, 0, 0, "Value2", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadFault, 0, 0, "Fault", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadTime, 0, 0, "Time", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.CreateThread(threadKeyboard, 0, 0, "Keyboard", 1024, ARM_CODE | FIQ_ENABLE | IRQ_ENABLE, 7);
  os.Start();
}
```

Referências Bibliográficas

[BB02] H. Boucheneb and G. Berthelot. Contraction of the itcpn state space. *Eletronic Notes in Theoretical Computer Science*, 65:1–15, 2002.

[BBAC04] P. Bonhomme, G. Berthelot, P. Agaline, and S. Calvez. Verification technique for time Petri nets. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 4278–4283, 2004.

[BD91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using petri nets. volume 17, pages 259–273, 1991.

[Ber87] G. Berthelot. Transformations and decompositions of nets. In *Advances in Petri nets 1986, part I on Petri nets: central models and their properties*, pages 359–376, 1987.

[BM82] B. Berthomieu and M. Menasche. A state enumeration approach for analyzing time petri nets. In *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, Italy, september 1982.

[BV95] G. Bucci and E. Vicario. Compositional validation of time-critical systems using communicating time Petri nets. *IEEE Transactions on Software Engeneering*, 21(12):969–992, 1995.

[Cel82] J. Celko. Time token design methodology. volume 2(10), pages 889–895. Software - Pratice and Experience, 1982.

[CMS99] A. Cerone and A. Maggiolo-Schettini. Time-base expressivity of time petri nets. volume 216, pages 1–53. Theorical Computer Science, 1999.

[DBDS94] S. Duri, U. Buy, R. Devarapalli, and S. M. Shatz. Application and experimental evaluation of state space reduction methods for deadlock analysis in ada. *ACM Transactions on Software Engineering and Methodology*, 3(4):340–380, 1994.

[Dij68] E. Dijkstra. *Programming Language*. New York Academic Press, 1968.

[ee06] eSysTech eAT55. *eAT55 - Placa de avaliação para processadores ARM*. http://www.esystech.com.br/produtos/hard/eAT55.htm, 2006.

[eXK06] eSysTech X Kernel. *X - Kernel de Tempo Real para Aplicações Embarcadas*. http://www.esystech.com.br/produtos/XKernel/XKernel.htm, 2006.

[Fre82] S. French. *Sequencing and Scheduling - An Introdution to the Mathematics of the Job-Shop*. Jonh Wiley, 1982.

[God96] P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems*. PhD thesis, Universite de Liege, 1996.

[Gom96] Luís F. S. Gomes. *As Redes de Petri Reactivas e Hierárquicas - integração de formalismo no projecto de sistemas reactivos de tempo-real*. PhD thesis, UNL-FCT, 1996.

[HW04] E. Hansen and G. W. Walster. *Global Optimization Using Interval Analysis*. Marcel Dekker, 2004.

[JKDW95] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Methods and Applications of Interval Analysis*. SIAM - Studies in Applied Mathematics, 1995.

[JM92] E. Y. Juan and T. Murata. A technique of state space search based on unfolding. *Kluwer Academic Publishers*, (9):1–22, 1992.

[KLK05] E. E. Künzle, E. A. Lima, and L. A. Künzle. Global time petri net analyzer. In *QEST 05*, Torino, Italy, September 2005.

[Lil99] J. Lilius. Efficient state space search for time petri nets. In *In Proc. MFCS'98 Workshop on Concurrency*, volume 18, 1999.

[LL04] S. Lipschutz and M. Lepson. *Matemática Discreta*. Bookman, 2004.

[LLK05] E. A. Lima, R. Lüders, and L. A. Künzle. Análise de redes de Petri temporais usando tempo global. In *VII Simpósio Brasileiro de Automação Inteligente - SBAI*, number 7, 2005.

[LLK06] E. A. Lima, R. Lüders, and L. A. Künzle. Análise de redes de Petri temporais via Álgebra intervalar. In *XVI Congresso Brasileiro de Automática - CBA*, number 16, 2006.

[Mer74] P. Merlin. *A Study of Recoverability of Computer Systems*. PhD thesis, University of California IRVINE, 1974. available from Ann Arbor: Univ Microfilms, No. 75–11026.

[MF76] P. Merlin and D. J. Farber. Recoverability of communication protocols. *IEEE Transaction on Communication*, 24(9), 1976.

[MGV00] L. Montano, F. J. Garci, and J. L. Villarroel. Using time Petri net formalism for specification, validation, and code generation in robot-control applications. *The International Journal of Robotics Research*, 19(1):59–76, 2000.

[Moo95] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM - Studies in Applied Mathematics, 1995.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. volume 77, pages 541–580. Proceedings of IEEE, 1989.

[Neu90] A. Neumaier. *Interval Methods for Systems of Equation*. Cambridge Univ. Press, 1990.

[Nid94] M. Nidd. Time extensions of petri nets. Technical report, 1994.

[Pai04] Rui Manuel Carvalho Pais. Geração de executores e analisadores de redes de petri. Master's thesis, Universidade Nova Lisboa, 2004.

[PZ91] L. Popova-Zeugmann. On time petri nets. volume 27, pages 227–244, 1991.

[Ram74] R. Ramchandani. *Analysis of Asyncronous Concurrent Systems by Time Petri Nets*. PhD thesis, Cambridge, Mass.: MIT, Dept. Electrical Engineering, 1974. Project MAC TR-120.

[RK04] P. Ramachandran and M. Kamath. A sufficient condition for reachability in a general Petri net. *Discrete Event Dynamic Systems*, 14(3):251–266, 2004.

[SB96] R. H. Sloan and U. A. Buy. Reduction rules for time Petri nets. *Acta Informatica*, 33(7):687–706, 1996.

[SB97] R. H. Sloan and U. Buy. Stubborn sets for real-time Petri nets. *Formal Methods in System Design: An International Journal*, 11(1):23–40, July 1997.

[Sif77] J. Sifakis. Etude du comportement permanent des réseaux de petri temporisés. pages 165–184, 1977.

[Sys06] IAR Systems. *IAR Embedded Workbench*. http://www.iar.se/, 2006.

[TYC95] J. J. P. Tsai, S. J. Yang, and Y. H. Chang. Timming constraint Petri nets and their application to schedulability analsis of real-time systems specifications. *IEEE Transactions on Software Engeneering*, 21(1):32–49, 1995.

[Val92] Antti Valmari. A stubborn attack on state explosion. *Form. Methods Syst. Des.*, 1(4):297–322, 1992.

[Val94] Antii Valmari. State of the art report: Stubborn sets. *Petri Net Newsletter*, (46):6–14, apr 1994.

[Vic01] Enrico Vicario. Static analysis and dynamic steering of time-dependent systems. *IEEE Transactions on Software Engeneering*, 27(8):728–748, 2001.

[WD00] J. Wang and Y. Deng. Reachability analysis of real-time systems using time Petri nets. In *IEEE Transactions on Systems , Man, and Cybernetics - Part B*, volume 30, pages 725 – 73, 2000.

[WG93] P. Wolper and P. Golefroid. Partial-order methods for temporal verification. *Lecture Notes in Computer Science, Springer-Verlag*, pages 1–14, 1993.

[Wor06] Petri Nets World. *Petri Nets Tools Database*. http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html, 2006.

[XHD02] D. Xu, X. He, and Y. Deng. Compositional schedulability analysis of real-time systems using time Petri nets. *IEEE Trans. Soft. Engineering*, 28(10):984–996, 2002.

[YS97] T. Yoneda and B.-H. Schlingloff. Efficient verification of parallel real-time systems. In *Formal Methods in System Design*, volume 11, pages 187–215, 1997.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo