UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

SimRP - SIMULADOR DE REDES DE PETRI FLEXÍVEL COM GERAÇÃO DE CÓDIGO VHDL

MARCOS MARTINS MELO

ORIENTADOR: CARLOS HUMBERTO LLANOS QUINTERO

DISSERTAÇÃO DE MESTRADO EM SISTEMAS MECATRÔNICOS

PUBLICAÇÃO: ENM.008/2006

BRASÍLIA: JULHO - 2006

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA MECATRÔNICA

SimRP - SIMULADOR DE REDES DE PETRI FLEXÍVEL COM GERAÇÃO DE CÓDIGO VHDL.

MARCOS MARTINS MELO

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS.

APROVADA POR:

Professor Carlos Humberto Llanos Quintero, Dr (ENM – UNB).

(Orientador)

Prof. Walter de Brito Vidal Filho, Dr (ENM – UNB).

(Examinador interno)

Prof. Maurício Ayala Rincón, Dr (MAT – UNB).

(Examinador externo)

BRASÍLIA/DF, 14 DE JULHO DE 2006.

FICHA CATALOGRÁFICA

MELO, MARCOS MARTINS

SimRP - SIMULADOR DE REDES DE PETRI FLEXÍVEL COM GERAÇÃO DE CÓDIGO VHDL [Distrito Federal] 2006.

xvii, 171 p., 210 x 297 mm (ENM/FT/UNB, Mestrando, Sistemas Mecatrônicos, 2006).

Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.

Departamento de Engenharia Mecânica.

1. Redes de Pedri 2. Simuladores

3. VHDL 4. GPL

I. ENM/FT/UNB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Melo, M. M. (2006). SimRP - SIMULADOR DE REDES DE PETRI FLEXÍVEL COM GERAÇÃO DE CÓDIGO VHDL. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-008/2006, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 171 p.

CESSÃO DE DIREITOS

AUTOR: Marcos Martins Melo

TÍTULO: SimRP - SIMULADOR DE REDES DE PETRI FLEXÍVEL COM GERAÇÃO DE CÓDIGO VHDL.

GRAU: Mestre ANO: 2006

Marcos Martins Melo

QNA 11 casa 05 – Taguatinga Norte.

72110-110 Brasília – DF – Brasil.

"Os caminhos para o poder humano e para o conhecimento humano correm lado a lado e são quase os mesmos; não obstante, por conta do inveterado e pernicioso hábito de insistir nas abstrações, é mais seguro começar a desenvolver as ciências dos fundamentos que têm relação com a prática e deixar a parte ativa ser como o selo que se imprime e determina a contrapartida contemplativa."

Francis Bacon

DEDICATÓRIA

Dedico este trabalho, primeiramente, a Deus, que me concedeu sabedoria, paz e saúde.

Dedico-o, ainda à minha amada família, que me motivou e possibilitou que eu pudesse vencer mais este desafio e ao meu Professor e Orientador Carlos Humberto Llanos; ao meu amigo de mestrado Evandro; e, principalmente, a minha Madrinha Maria que sempre me motivou.

AGRADECIMENTOS

Agradeço, primeiramente, a Deus que me possibilitou saúde e sabedoria para a realização deste feito.

Agradeço à madrinha Maria Borges e a minha família que foi compreensiva e fundamental na execução deste projeto.

Em especial, gostaria de agradecer ao meu orientador Carlos Humberto Llanos Quintero e a todos os meus professores que me dotaram de conhecimentos necessários para o bom andamento do projeto.

Ao meu amigo de mestrado Evandro e a todos que participaram de alguma forma desta conquista, o meu sincero agradecimento.

RESUMO

O objetivo desta dissertação é implementar uma ferramenta de CAD para a descrição e simulação de Redes de Petri (RdPs) baseadas em software livre, chamada de SimRP. Esta ferramenta de CAD possibilitará a descrição e simulação de diversas RdPs, ou seja, Ordinárias, Temporais, Temporizadas e Interpretadas, além disso, O SimRP disponibilizará a opção de geração de código VHDL (uma linguagem de descrição de hardware), a partir de um modelamento de uma RdP interpretada. Adicionalmente o SimRP permite a verificação de diversas propriedades tais como rede marcada reiniciável, rede marcada viva, rede não pura e detecção de conflitos/deadlocks.

Uma característica importante do SimRP é sua flexibilidade, Os usuários poderão descrever uma RdP Ordinária e mais tarde converte-la para outro tipo de rede estudada neste trabalho. Deste modo, os usuários poderão acrescentar atributos a uma RdP ordinária para representar e simular uma RdP Temporal (por exemplo). Estes atributos são facilmente adicionados e deletados pela interface gráfica.

O SimRP foi desenvolvido na arquitetura cliente-servidor para garantir a melhor usabilidade/portabilidade. Tanto a instalação quanto a configuração são feitas no servidor. Neste caso, o SimRP necessita apenas de um browser na sua estação de trabalho.O simulador foi desenvolvido baseado na licença GPL, isto permite acesso ilimitado a todas funções do sistema e código fonte, permitindo que os usuários modifiquem e adaptem o sistema de acordo com sua necessidade, sem necessitasse de licença.

Palavras-Chave: Redes de Petri, Simulação, eventos discretos, VHDL.

ABSTRACT

The objective of this dissertation is to implement a description and simulation CAD tool for Petri Nets (PNs) based on open source, called SimRP. This CAD tool will be able to describe and simulate several PNs namely Ordinary, Temporal, Temporized and Interpreted types. Besides this, SimRP provides an option to generate VHDL code (a hardware description language), taking a model of an Interpreted PN. Additionally, SimRP allows several properties verification such as Restartable PN, Marked-alive PN, Not-pure PN and conflicts/deadlocks detection.

An important feature of SimRP is its flexibility, since users can describe an ordinary PN and afterwards converts it to another type of network studied here. Thus, users can add certain attributes to an ordinary PN in order to represent and simulate a Temporal PN (for example). These attributes are easily added/deleted through a graphical interface.

The SimRP was developed in client-server architecture in order to guarantee the best usability/portability. Both system installation and configuration are made on a server. In this case, a SimRP user needs only a browser in his/her workstation. Given that the simulator was developed under GPL license, this provides an unlimited use to all of its resources, including source code, allowing any user to use, modify, and adapt it to their needs without license requires.

Keywords: Petri Networks , Simulation, discrete events, VHDL.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

CAD	*Computer Aided Design* – Projeto Assistido por Computador.
DNS	*Domain Name System* – Sistema de Nome de Domínio
FPGA	*Field Programmable Gate Array* – Arranjo de Portas Programáveis no Campo
GPL	*General Public License GPL* – Licença Pública Geral.
HDL	*Hardware Description Languagem* – Linguagem de Descrição de Hardware
UML	*Unified Modelling Language* – Linguagem de Modelamento Unificado
URL	Uniform Resource Locator – Localizador de Pesquisa Unificada.
VHDL	*VHSIC Hardware Description Languagem* – Linguagem de Descrição de Hardware VHSIC.
RdPs	Redes de Petri
SEDs	Sistemas de Eventos Discretos

1. INTRODUÇÃO

Redes de Petri (RdPs) são um modelo matemático desenvolvido por C.A Petri [16]. As redes de Petri são especialmente apropriadas para modelar sistemas onde diferentes eventos podem estar acontecendo (acionados por eventos). RdPs podem ser aplicadas para sistemas que podem ser caracterizados como concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos, estocásticos [17]. Na área de automação RdPs têm ampla aplicabilidade para modelar os sistemas acionados a Eventos Discretos (SED-Sistemas de Eventos Discretos) [18], onde é muito importante caracterizar, estudar, simular e/ou verificar o comportamento do sistema. Um ponto importante na área de automação é determinar se o sistema está livre de conflitos no processo de execução de alguma tarefa determinada.

Uma propriedade importante das RdPs é sua clara e simples representação gráfica, mediante um grafo composto de lugares, transições e arcos. Aplicações importantes de RdPs podem ser encontradas em diferentes áreas como avaliação de desempenho, protocolos de comunicação, análise de sistemas distribuídos, bancos de dados distribuídos, programação concorrente e paralela, sistemas tolerantes a falhas, circuitos assíncronos, sistemas operacionais, projeto de compiladores, automação de escritórios, linguagens formais, programação lógica, sistemas de manufatura, entre outras [17].

A presente dissertação tem por objetivo o estudo das Redes de Petri e a construção de uma ferramenta CAD de descrição e simulação de código aberto baseado na licença GPL, denominada SimRP. Esta ferramenta de CAD será capaz de descrever e simular RdPs dos tipos Ordinária, Interpretada, Temporal e Temporizada. Além do anterior, a ferramenta possui a opção de gerar código em VHDL (uma linguagem de descrição de hardware) a partir do modelo de uma rede de Petri interpretada.

Uma característica importante do SimRP é sua flexibilidade, dado que o usuário pode descrever uma RdPs Ordinária e, posteriormente, implementar a sua conversão para um dos outros tipos de redes estudados nesta dissertação. Desta maneira, o usuário pode acrescentar certos atributos para que uma rede de Petri ordinária passe a representar uma rede de Petri

Temporal (por exemplo). Estes atributos são facilmente acrescentados por intermédio de uma interface gráfica, que permite editar os atributos de RdPs.

Uma estratégia para garantir a melhor usabilidade/portabilidade do SimRP foi desenvolvê-lo na arquitetura de três camadas, onde toda a instalação e configuração do sistema é realizada no servidor. Neste caso, um usuário do SimRP só precisa ter instalado um *browser* no seu computador ou estação de trabalho.

O simulador foi desenvolvido em licença GPL e possibilitará o uso ilimitado de todos seus recursos, inclusive do código fonte, permitindo que outras pessoas utilizem, modifiquem e adaptem a suas necessidades, gratuitamente, e sem a exigência de licenças.

1.1 GENERALIDADES

Com o objetivo de descrever e visualizar o comportamento de eventos discretos, Carlos Petri [16] propôs um modelo chamado de Rede de Petri. Em princípio este modelo foi idealizado para descrever a comunicação entre autômatos [1]. Mas devido ao seu formalismo e capacidade de descrever eventos de alta complexidade o modelo apresenta-se como uma alternativa para descrever eventos em diversas áreas, como por exemplo, informática , economia, não se limitando o modelo à área de tecnologia[2].

Modelando o comportamento de um evento, torna-se possível simulá-lo e compreendê-lo, verificando se o comportamento é realmente o esperado, torna-se possível testá-lo antes de ser implementado, verificando assim a presença de conflitos estruturais e *deadlocks*[1].

O licenciamento GPL define quatro liberdades em relação a um software[3]:

a) Liberdade de executar o programa para qualquer fim.

b) Liberdade de estudar como o programa funciona e poder adaptá-lo para as suas necessidades. O acesso ao código-fonte é um pré-requisito para esta liberdade.

c) Liberdade de redistribuir cópias de modo que se possa beneficiar o próximo.

d) Liberdade de aperfeiçoar o programa, e liberar as suas melhorias, de modo que toda a comunidade se beneficie. O acesso ao código-fonte é um pré-requisito para esta liberdade.

O software que possue este licenciamento são chamados de *software livre* ou *open source*.

O software gratuito ou *software Free* [3], não permite as quatro liberdades definidas na licença GPL; este tipo de licenciamento define apenas uma permissão de utilização, não permitindo o estudo de código, adaptação da aplicação e distribuição do produto. Este tipo de licença pode ainda restringir o tipo de uso, definindo, por exemplo que a utilização está restrita ao meio acadêmico.

1.2 OBJETIVOS DO TRABALHO

1.2.1 **Objetivos gerais**:

a) Desenvolver uma ferramenta de CAD (*Computer Aided Design* – Projeto Assistido por Computador) para simulação de três tipos de RdPs baseado nos conceitos de Software Livre com licenças GPL (*General Public License GPL)*, com documentação seguindo os princípios e premissas da Engenharia de Software e UML (*Unified Modeling Language*) [19];

b) Propiciar a integração entre as RdPs e a linguagem de descrição de hardware VHDL, (*VHSIC Hardware Description Languagem*) [14], para possibilitar a implementação de controladores em FPGAs (*Field Programmable Gate Arrays*).

1.2.2 **Objetivos específicos**:

a) Criar um simulador de sistemas de eventos discretos (SEDs) modelados em RdPs.

b) Possibilitar a simulação de RdPs do tipo Ordinárias, Interpretadas, Temporais e Temporizadas.

c) Implementar a característica de flexibilidade, que consiste na possibilidade de converter descrições de RdPs Ordinárias para RdPs do tipo Interpretadas, Temporais e/ou Temporizadas.

d) Gerar código VHDL por intermédio de definição de uma RdPs Interpretada, objetivando a geração automática de controladores;

e) Verificar algumas propriedades pertinentes a uma RdPs, rede viva, rede iniciável e trasição viva.

f) Detectar certos conflitos estruturais e *deadlock* em tempo de simulação.

g) Proporcionar condições necessárias para que outras pessoas possam dar continuidade ao desenvolvimento do simulador.

1.3 MOTIVAÇÃO DESTE TRABALHO

A teoria de RdPs é cada vez mais utilizada, devido ao seu simples formalismo matemático e sua capacidade de simular problemas complexos. A flexibilidade do seu modelo possibilita que seus conceitos sejam utilizados por outras áreas do conhecimento, não se limitando às áreas de tecnologia [2].

Na pesquisa dos simuladores existentes foram encontrados diversos aplicativos que simulam Redes de Petri. A grande maioria dos simuladores pesquisados são softwares proprietários ou softwares gratuitos. Os simuladores encontrados com licença GPL não possuíam uma boa documentação.

O uso de software livre vem sendo a cada dia mais difundido e utilizado, devido as suas características de compartilhamento de conhecimento, redução de custo e disseminação de tecnologia. O Governo Federal do Brasil vem estimulando o uso destes softwares nos órgãos do governo e nas instituições de ensino [20].

Mesmo existindo simuladores de Redes de Petri, baseados em Software Livre, estes não possuem uma boa documentação e, em geral, são desenvolvidos sem levar em consideração premissas da Engenharia de Software. Isto torna, na maioria das vezes, o entendimento mais complexo e difícil que o próprio desenvolvimento.

Por outro lado, existe na comunidade de sistemas digitais o interesse de desenvolver ferramentas de CAD que permitam a geração automática de descrições de sistemas/circuitos para Linguagens de Descrição de Hardware como VHDL, *Verilog, SystemC* [21], entre outras. Por exemplo, dada uma Rede de Petri Interpretada, que modela um controlador para um sistema de manufatura, deseja-se a geração automática da sua descrição numa linguagem como VHDL. Neste caso, a descrição inicial do sistema é feita num modelo formal e de mais alto nível de abstração (como é o caso de Redes de Petri), o qual permite resolver e/ou aliviar os problemas clássicos do projeto de sistemas digitais complexos, tais como a validação e verificação formal (automática e/ou manual) dos mesmos.

Esta dissertação além de apresentar os conceitos relacionados à Rede de Petri, contribui com a implementação de um simulador baseado em licença GPL - *General Public License GPL* – Licença Pública Geral (Software Livre), disponibilizando a sua análise e documentação.

1.4 METODOLOGIA APLICADA NESTE TRABALHO

Foi realizado um estudo teórico dos diferentes modelos de RdPs. Dentre os modelos possíveis foram escolhidos aqueles mais utilizados para aplicações de automação e controle (vide capítulo 2).

Por outro lado, foram estudados diversos simuladores de Rede de Petri, para verificar suas características (vide capítulo 3). Este estudo serviu de base para a especificação do SimRP.

O simulador foi implementado utilizando os princípios e premissas da Engenharia de Software e a metodologia utilizada será UML para a documentação e Orientada a Objeto para o desenvolvimento da ferramenta.

O modelo escolhido para o desenvolvimento da aplicação foi a arquitetura de três camadas, onde toda a instalação e configuração da aplicação são realizadas no servidor, não necessitando a instalação e configuração de nenhum aplicativo com exceção do navegador *web*, nas máquinas dos usuários.

Este modelo contribui para uma melhor gerência da aplicação, facilitando a atualização da aplicação. Como todo o processamento será realizado no servidor, não serão necessários grandes recursos de máquinas por parte dos usuários e nem a instalação de produtos nas estações.

1.5 ESPECIFICAÇÃO DO PROJETO

O simulador foi desenvolvido em PHP. Esta linguagem permite a implementação da arquitetura de três camadas e não precisa da instalação de nenhum software proprietário para seu funcionamento, diferente da linguagem de programação Java que necessita da instalação de Software: Máquina Virtual Java, para seu funcionamento, este Software de uso Livre, porém não é GPL.

O servidor que disponibilizará a aplicação poderá ser instalado e executado em diversos sistemas operacionais, como por exemplo: Linux, Windows, SUN, AIX, entre outros. Será necessária a instalação do serviço de publicação de página Apache [22] com o módulo de interpretação PHP, também se faz necessária a instalação do banco de dados MySQL [23].

A estação dos usuários do sistema somente necessitará ter instalado um navegador *web*, seja Internet Explorer, Fire Fox ou um qualquer outro, em qualquer sistema operacional.

1.6 ESTRUTURA DO TRABALHO

Esta dissertação está dividida em seis capítulos. Além desta introdução, no capítulo II são enfatizados os conceitos relacionados à teoria de RdPs, mais especificamente das redes objeto deste trabalho, detalhando suas características, propriedades e taxonomias. Além do anterior, são introduzidos os conceitos básicos da Linguagem de Descrição de Hardware VHDL.

No capítulo III são apresentadas as análises dos simuladores pesquisados, além das suas vantagens e desvantagens.

No capítulo IV é apresentado o simulador SimRP junto com a descrição das suas funcionalidades e uma descrição da estrutura básica da implementação.

No capítulo V são apresentados os resultados obtidos com o SimRP na simulação das RdPs.

No capítulo VI conclui-se a dissertação e apresentam-se sugestões para trabalhos futuros com vistas à continuidade deste projeto.

Além dos capítulos, o trabalho contém cinco apêndices – apêndice 1: Diagramas do Sistema, apêndice 2: Descrição dos casos de uso, apêndice 3: Script de Banco de Dados, no apêndice 4: Manual de Utilização do sistema, apêndice 5: Código VHDL gerado pelo SimRP.

2. REFERENCIAL TEÓRICO

2.1 INTRODUÇÃO

Neste capítulo serão apresentados os conceitos básicos utilizados na realização deste trabalho. Redes de Petri são bastante utilizadas para modelamento de Sistemas de Eventos Discretos (SED). Um sistema de Eventos Discretos se caracteriza pela mudança de estado, que ocorre sempre em instantes precisos. Além do anterior, os valores das variáveis de estado mudam bruscamente. Nos sistemas de eventos contínuos a mudança de estado pode ocorrer a qualquer instante, com poucas variações no valor das variáveis de estado. São exemplos de sistemas de eventos discretos os sistemas digitais, filas de serviços (por exemplo, aplicados a sistemas operacionais), processos de manufatura, entre outros [1].

Sistemas Discretizados são sistemas contínuos observados em instantes discretos ("Sistemas amostrados"). Neste caso, o objetivo é buscar um comportamento em um instante discreto de tempo [1].

As RdPs são representações matemáticas que objetivam o modelamento de eventos discretos. Por meio deste modelamento é possível analisar o comportamento de um SED, verificando possíveis conflitos, *deadlocks* e até mesmo propriedades que permitem o melhoramento do modelo [2][5].

Por outro lado, existe o problema da geração de sistemas de controle e automação usando técnicas de sistemas digitais avançadas. Dado o aumento da complexidade dos sistemas e da capacidade de integração, existe a necessidade de pesquisa de novas metodologias de projeto objetivando fornecer soluções efetivas de custo baixo, que sejam capazes de cumprir com as exigências de capacidade de processamento e comunicação.

Sistemas em chip (*System on chip* – SoC) e sistemas de chips programáveis (*System on a programmable chip SoPC*) aparecem como alternativas tecnológicas para integração de uma grande variedade de algoritmos necessários para tarefas como processamento de dados e comunicações, automação e controle, entre outras. Uma solução obtida para enfrentar o problema da alta complexidade dos sistemas digitais (por exemplo, dos SoC) é a de elevar o nível de abstração na especificação e descrição dos mesmos. Tradicionalmente, os projetistas têm usado linguagens como VHDL e Verilog para especificar e descrever circuitos digitais.

As linguagens de descrição de hardware como VHDL e Verilog possibilitam o modelamento e construção de sistemas digitais por intermédio de descrições com alto nível de abstração, com estruturas sintática e semântica semelhantes a das linguagens de programação como Pascal e C. Este tipo de definição facilita o projeto de sistemas digitais complexos. Por outro lado, estes tipos de linguagens são amplamente usados no projeto de sistemas baseados em Arquiteturas Reconfiguráveis.

Arquiteturas Reconfiguráveis são baseadas em microcontroladores/microprocessadores de uso geral e Dispositivos Lógicos Programáveis (PLDs). Entre os PLDs mais utilizados estão os FPGAs (*Field Programmable Gate Arrays*). Uma descrição do hardware de um sistema (usando uma linguagem como VHDL) pode ser sintetizada (usando uma ferramenta CAD) e repassada para a FPGA. Este processo é chamado de reconfiguração da FPGA. A reconfiguração pode ser implementada ainda em tempo real [15].

Dispositivos como FPGAs permitem a implementação de algoritmos diretamente em hardware. A configuração destes dispositivos é realizada por um arquivo binário (*bitstream*). Em sistemas convencionais (modelo de von Neumann) a programação é realizada por software, sendo carregada dentro da RAM da CPU. A semântica da programação em software é procedural. Em FPGAs a programação é feita por *Configware*, compilada para um código de reconfiguração binário e carregada na FPGA. Isto caracteriza um novo paradigma de programação (*Lógica Programável*).

A semântica do *Configware* é estrutural (não procedural), onde a descrição do circuito é compilada por meio de ferramentas de posicionamento (*placement*) e roteamento (*routing*). Em FPGAs existem milhares e milhões de blocos lógicos configuráveis (*configurable logic blocks* – CLBs) que são, normalmente, de largura de palavra de um bit. Uma alternativa recente para RL é a Computação Reconfigurável (*Reconfigurable Computing* – PC) usando rDPAs (*Reconfigurable Data Path Units*), cada uma tendo uma largura de palavra de vários bits (por exemplo 32 bits). Deve ficar claro que uma rDPU não é uma CPU dado que não possui contador de programa (PC) [27]. A programação desta última alternativa usa contadores de dados (*data counters*), os quais estão localizados em bancos de memória de auto-seqüenciamento.

Tanto a alternativa de lógica reconfigurável (RL) como a de programação reconfigurável (PC) oferecem a possibilidade de realizar reconfiguração dinâmica e parcial. Dispositivos oferencidos por Xilinx [42] permitem a reconfiguração dinâmica e parcial. Neste caso, pode ser realizada a reconfiguração de uma parte da FPGA enquanto o resto do dispositivo continua funcionando normalmente. O código de reconfiguração pode ser carregado via interfase serial/paralela ou armazenando uma memória de reconfiguração.

Um outro lado, o nível de abstração de um modelo com RdP é mais alto que o nível de abstração oferecido por linguagens tradicionais como VHDL e Verilog. O projetista pode, facilmente, modelar um sistema de manufatura, junto com o controle do mesmo usando RdPs. O anterior faz importante a geração automática de uma descrição do controlador (numa linguagem de menor nível de abstração como VHDL ou C), partindo da sua descrição em RdP. Dado que o interesse deste trabalho é a geração automática de descrições de controladores para Arquiteturas Reconfiguráveis a linguagem alvo escolhida foi VHDL[15].

2.2 CONCEITOS BÁSICOS DE MODELAGEM DE SISTEMAS

Segundo [2], os conceitos básicos na modelagem de um sistema baseado numa abordagem por eventos discretos são:

a) **Eventos**: são instantes de observação e de mudanças de estado.

b) **Atividades**: são caixas-pretas utilizadas para recuperar e esconder a evolução do sistema físico entre dois eventos.

c) **Processos**: são seqüências de eventos e atividades independentes.

d) **Cooperação**: antes do ponto de sincronização procura-se descrever uma independência de processos, fazendo com que os processos concorram a um objetivo comum.

e) **Competição**: os processos devem ter acesso a um dado recurso para realizar uma tarefa, tratando-se, portanto, de exclusão mútua.

f) **Pseudoparalelismo**: os eventos nunca são simultâneos e o paralelismo é apenas aparente, tendo seus ventos ordenados por um relógio comum.

g) **Paralelismo verdadeiro**: os eventos ocorrem simultaneamente, um processador é alocado para cada tarefa independente.

Estes conceitos têm ampla aplicabilidade no modelamento de sistemas de manufatura (o foco de aplicação deste trabalho).

2.3 CONCEITOS BÁSICOS DE REDES DE PETRI

Nesta seção serão abordados os conceitos sobre RdPs, assim como os diferentes tipos abordados na literatura.

2.3.1 Elementos básicos de uma RdPs

Segundo [1] e [14], as RdPs podem ser divididas em três elementos básicos:

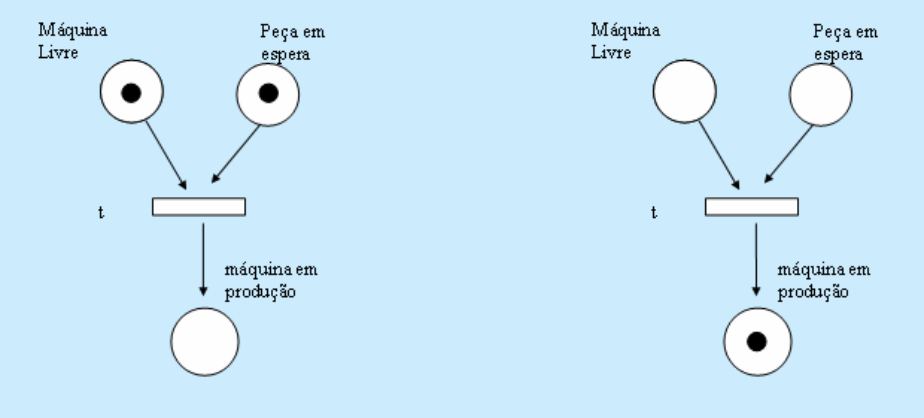
a) **Lugar**: indica uma condição, um estado parcial, alocação de recurso ou um estado de espera. É representado por um círculo.

b) **Transição**: indica um evento que pode ocorrer no sistema. É representado por um retângulo.

c) **Ficha**: está associada a um lugar e indica se determinada condição é verdadeira. É representada por um ponto num lugar.

A figura 2.1 mostra como os elementos básicos de uma RdPs se relacionam. O modelamento abaixo se refere a um determinado processo de fabricação de uma determinada peça.

Para esta fabricação é necessário que os lugares referentes a “Máquinas Livre” e “Peça em Espera” possuam recursos ou fichas (vide figura 2.1 (a)), determinando que a máquina está livre e que a peça já está disponível. Satisfeita esta condição a RdPs, pode passar para o outro estado que representa a máquina em produção (vide figura 2.1 (b)).



(a) **(b)**

Figura 2.1 - RdPs [1]

2.3.2 Comportamento dinâmico

A ocorrência de um evento no sistema está diretamente associada a um disparo de transição que consiste em três passos [4]:

a) Após o evento retiram-se as fichas do lugar de entrada para indicar que a condição não é mais verdadeira (como mostrado na figura 2.1 (b)).

b) Adicionar a ficha no lugar de saída após a ocorrência do evento (figura 2.1 (b)).

c) A retirada de fichas de um determinado lugar e o acréscimo de determinadas fichas em outro, indicam que uma condição *t* foi satisfeita e o evento associado àquela transição, foi realizada.

2.3.3 Modelamento de diferentes interações entre processos usando Redes de Petri

a) **Modelamento de uma Seqüência**: descreve uma seqüência de disparo que representa o evento de passagem de uma fase para outra, definindo um encadeamento de eventos e atividades [2].

A figura 2.2 ilustra este encadeamento; *J1* representa a primeira e *J2*; a segunda fase. As fichas correspondem aos recursos consumidos em cada fase.

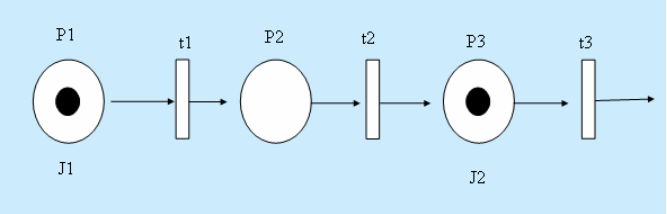
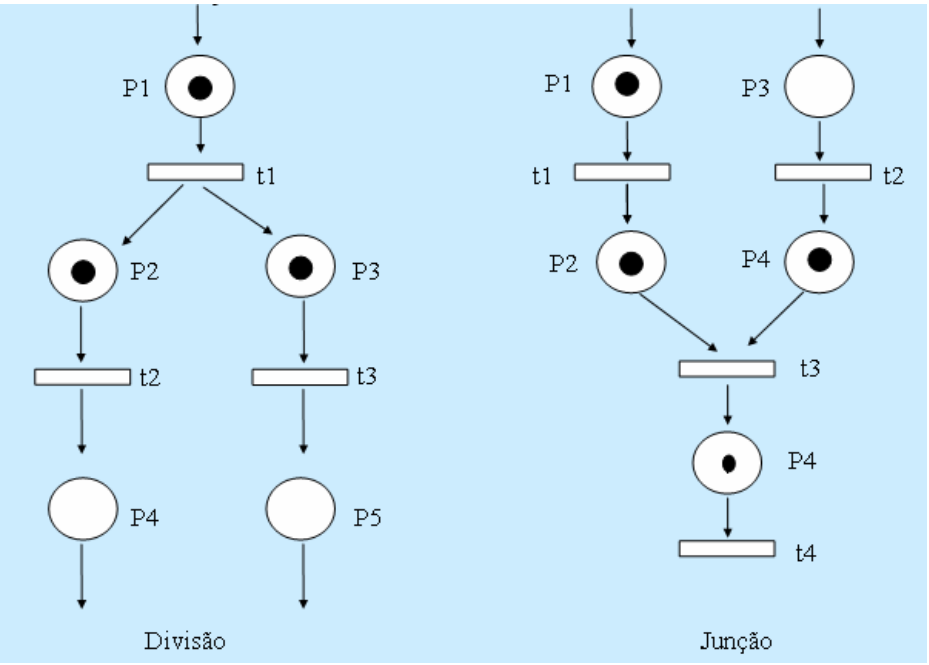


Figura 2.2 - Seqüência de Processos [2]

b) **Modelamento de Evoluções síncronas e assíncronas**: A figura 2.3 (a) apresenta uma ficha, no lugar *P1*. Ao ser disparada a transição t1 duas fichas aparecerão em *P2* e *P3* simultaneamente. O aparecimento das duas fichas em *P2* e *P3* é dependente do disparo da transição *t1*. Neste caso, os eventos em *P2* e *P3* ocorrem de maneira síncrona. O mesmo pode ser observado na figura 3.b, onde o desaparecimento de fichas em *P2* e *P3* ocorre de maneira simultânea (síncrona).

As evoluções assíncronas não estabelecem relação entre execução de transições que alterarão as marcações, podendo ser executadas a qualquer instante sem depender de um sincronismo [14].

A figura 2.3 (a) descreve a divisão e a junção de recursos. Após a divisão do recurso (feito sincronamente) as fichas evoluem independentemente (de maneira assíncrona) [13].

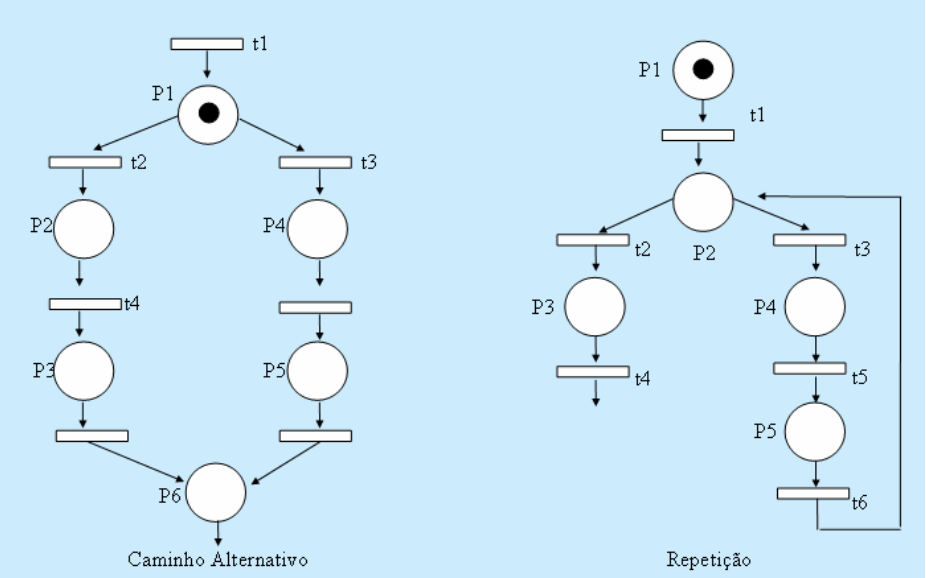


(a) **(b)**

Figura 2.3 - Divisão e junção de recursos[3]

c) **Modelamento de Caminhos alternativos e Repetição**: O caminho alternativo caracteriza-se na situação em que a RdPs pode evoluir por um caminho em determinado momento, e por um outro, em outro momento [6][1]. O anterior é determinado pela definição de uma RdPs ordinária (vide seção 2.5). Para implementar os caminhos alternativos, são necessárias pelo menos duas transições como mostrado na figura 2.4 (a).

A repetição se caracteriza pela existência de uma transição que possibilita retornar ao começo de uma atividade. Esta característica pode representar, por exemplo, uma determinada etapa de um processo de fabricação onde a mesma pode se repetir inúmeras vezes. A figura 2.4 (b) mostra este modelamento.



(a) **(b)**

Figura 2.4 – Caminhos Alternativos e Repetição[1]

d) **Modelamento de Alocação de Recursos**: a alocação de recursos apresenta-se com um recurso fundamental na modelagem de uma RdP, pois permite simular um *buffer* armazenando recursos para uma determinada atividade[8]. Este caso pode ser observado na figura 2.5, onde um recurso está alocado para ser utilizado em *Pr*. Uma vez *t1* seja disparada, a transição será sensibilizada já que *P2* e *Pr* possuem recursos. Quando *t3* for disparado o recurso consumido, anteriormente, será reposto em *Pr*. Um outro processo pode estar sendo executado no sistema (vide *t4* e *P5*). Neste caso, somente um processo pode ser executado num determinado momento.

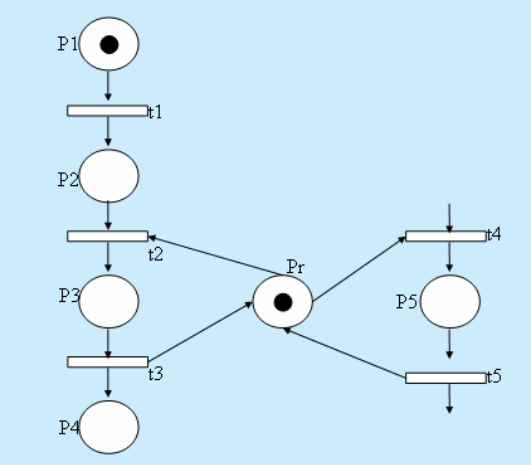


Figura 2.5 - Alocação de Recursos[1]

2.4 ASPECTOS FORMAIS DAS RdPs

Segundo [7], [1] e [14], uma RdP é uma quádrupla definida por:

$R = \langle P,T,Pre,Post \rangle$, onde:

- P é um conjunto finito de lugares de tamanho n.
- T é um conjunto finito de transições de tamanho m.
- $Pre : P \times T \rightarrow N$ é a aplicação de entrada (lugares precedentes ou incidência anterior), com N sendo o conjunto dos números naturais. *Pre* representa os lugares que são precedentes a uma transição (conjunto de arcos de entrada a uma transição). A matriz associada à condição *Pre* é definida por n linhas (representando lugares) e m colunas (representado transições).
- $Post: P \times T \rightarrow N$ é a aplicação de saída (lugar seguinte ou incidência posterior). *Post* representa as saídas (representados como arcos) de uma transição para lugares da RdP. A matriz associada à aplicação *Post* é definida por n linhas (representando lugares) e m colunas (representado transições).

Uma RdP definida dessa maneira é conhecida como **RdP Ordinária**.

Para cada "arco" ou componentes das matrizes associadas às aplicações *Pre* e *Post* pode ser atribuído um peso referente ao número de recursos necessários para sensibilizar uma transição.

Uma rede marcada [7] é uma dupla definida por: N = <R, M>, onde:

- *R* é uma RdP;
- *M* é uma marcação dada por uma aplicaçào $M : P \rightarrow N$;

A marcação representa a distribuição das fichas nos lugares da RdP num momento determinado. Pode ser definida uma Marcação Inicial para a RdP, a partir da qual a RdP evoluirá para novas marcações, de acordo com as aplicações *Pre* e *Post*.

Para exemplificar os conceitos de matrizes *Pre* e *Post* a figura 2.6 mostra uma RdP. Esta figura apresenta um exemplo de modelamento de uma RdPs, marcada com estado inicial $M = [0 \quad 3 \quad 0]^T$.

As matrizes *Pre* e *Post* da mesma são mostradas nas figuras 2.7 (a) e 2.7 (b). As linhas das matrizes representa os lugares da RdP e as colunas o conjunto das transições da mesma (vide figura 2.6).

2.4.1 Matriz de incidência

Uma matriz de incidência *C* representa o balanço das fichas numa RdPs. Os elementos da matriz indicam quantas fichas serão retiradas ou adicionadas, na execução de uma determinada transição[11]. Da mesma maneira que as matrizes *Pre* e *Post*, esta é definida por *n* linhas (representando lugares) e *m* colunas (representado transições).

Por intermédio desta representação é possível saber quais transições foram sensibilizadas tendo como parâmetro uma marcação inicial e uma marcação final. Também é possível saber qual será a marcação final (estado final), passando como parâmetro a marcação inicial e as transições sensibilizadas.

O comportamento de uma matriz de incidência é definido da seguinte forma:

- Quando o elemento a_{ij} > 0, indica que a relação da transição com aquele lugar é de acréscimo de *n* fichas, onde, *n* é igual ao valor de a_{ij} ;
- Quando o elemento a_{ij} < 0, indica que a relação da transição com aquele lugar é de retirada de *n* fichas; onde, *n* é igual ao valor de a_{ij} ;
- Quando o elemento a_{ij} = 0, indica que aquela transição não possui relação com aquele lugar.

A matriz de incidência *C* pode ser obtida por meio da equação 1:

$$C = Post - Pre \qquad (1)$$

A figura 2.7 (c) mostra a matriz de incidência correspondente à figura 2.6 e calculada usando a equação 1 e a $M = [0\ 3\ 0]^t$. A figura 2.7 (d) mostra o vetor-coluna (M) de marcação.

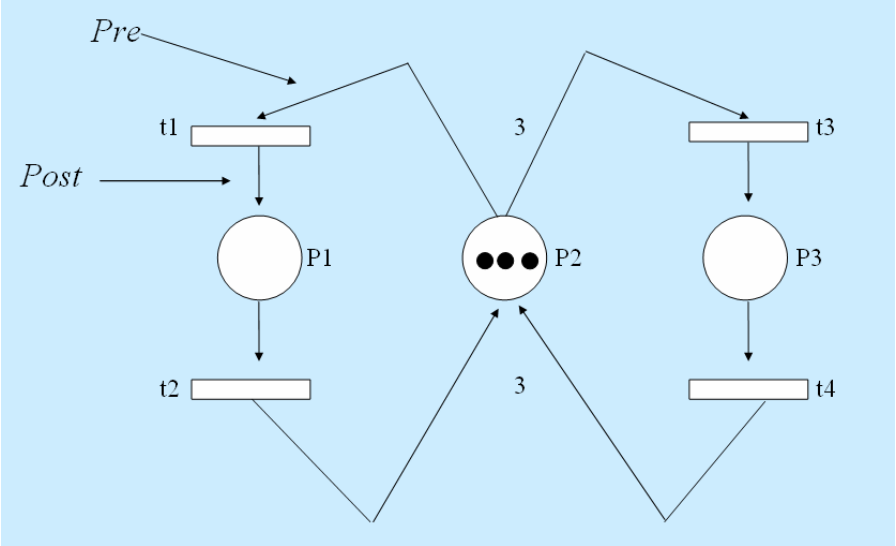


Figura 2.6 – Exemplo de RdPs [1]

$$Pre = \begin{array}{c} \\ p1 \\ p2 \\ p3 \end{array} \begin{array}{c} \begin{array}{cccc} t1 & t2 & t3 & t4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

(a)

$$Post \begin{array}{c} \begin{array}{cccc} t1 & t2 & t3 & t4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ p1 \\ p2 \\ p3 \end{array}$$

(b)

$$C = \begin{array}{c} \begin{array}{cccc} t1 & t2 & t3 & t4 \end{array} \\ \begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & -3 & 3 \\ 0 & 0 & 1 & -1 \end{bmatrix} \end{array} \begin{array}{c} p1 \\ p2 \\ p3 \end{array}$$

(c)

$$M = \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix}$$

(d)

Figura 2.7 - Definição formal da RdPs

2.4.2 Transição Sensibilizada

Segundo [4], dada uma marcação *M,* para que uma transição *t* fique sensibilizada ou habilitada, tem que satisfazer a condição mostrada na equação 2:

$$\forall p \in P, M(p) \geq Pre(p,t) \qquad \textbf{(2)}$$

A equação 2 significa que se nos lugares precedentes da transição tiver um número de fichas maior ou igual ao peso do "arco" referente ao elemento do conjunto Pre, que liga a este lugar a transição, esta transição poderá ser disparada.

2.4.3 Disparo de uma Transição

Segundo [8], se uma determinada transição t estiver sensibilizada por uma marcação M, será obtida uma nova marcação M' com o disparo desta transição. Esta nova marcação poderá ser obtida por meio da equação 3:

$$M' = M - Pre[.,t]_i + Post[.,t]_i = M + C[.,t]_i \qquad (3)$$

Onde $Pre[.,t]_i$, $Post[.,t]_i$ e $C[.,t]_i$ representam vetores da i-ésima coluna.

O disparo de uma transição consiste em retirar as fichas de cada lugar precedente da transição (*Pre*(p,t)), de acordo com o peso do arco e adicionar as fichas no lugar procedente da transição (*Post*(p,t)), de acordo com o peso do arco.

Em uma RdPs ordinária, somente uma transição pode ser disparada de cada vez, mesmo tendo outras transições sensibilizadas [2].

A figura 2.8 mostra este comportamento, observe que na figura 2.8 (a) o lugar P1 possui 2 fichas e o arco referente a *Pre*(p1,t1)=2, logo a transição t1 está sensibilizada e o recurso que estava em P1 vai para P2 com 03 fichas, como o peso estabelecido por *Pre*(p1,t1)=3 e ilustrado na figura 2.8 (b).

Por definição, quando o peso do arco não é declarado no modelamento, o mesmo terá sempre o peso 1.

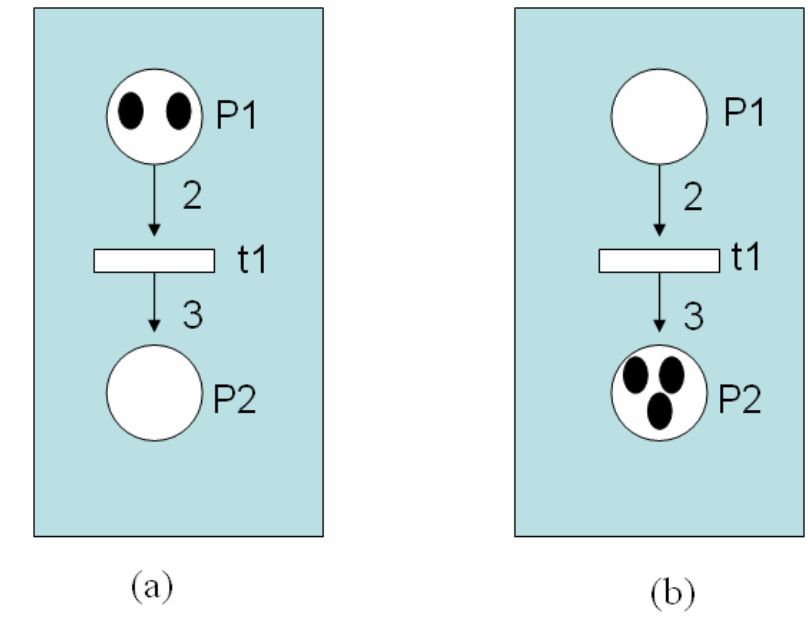


Figura 2.8 - Exemplo de transição sensibilizada

No exemplo da figura 2.8 a *Pre*(1,1)=2 e a *Post*(2,1)=3.

2.4.4 Conflito e Paralelismo

Na execução de uma RdP podem acontecer problemas. Por exemplo, duas transições usando o mesmo recurso. Neste caso as duas transições estariam sensibilizadas, porém quando uma das transições for disparada, automaticamente a outra não estará mais sensibilizada. Este tipo de problema é denominado de conflito. O conflito caracteriza-se pela existência de dada marcação que possua duas ou mais possibilidades, excludentes, de evolução. No caso do paralelismo, todas as atividades poderão ser executadas ao mesmo tempo [11].

O paralelismo e conflito podem ser do tipo:

a) **Conflito Estrutural**: duas transições *t1* e *t2* estão em conflito estrutural se, e somente se, elas têm ao menos um lugar *p* de entrada em comum, como indicado na condicão 4:

$$\exists p \in P, Pre(p,t_1) \neq 0 \qquad e \qquad Pre(p,t_2) \neq 0. \quad (4)$$

Isto pode ser compreendido como o caso de um lugar estar fornecendo um mesmo recurso a duas transições.

b) **Conflito Efetivo**: duas transições *t1* e *t2* estão em conflito efetivo para uma marcação *M* se e somente se ambas estão em conflito estrutural e estão sensibilizadas, como indicado na condição 5:

$$M \geq \Pr e(.,t_1) e M \geq \Pr e(.,t_2). \qquad (5)$$

Este problema é mostrado na figura 2.9.

c) **Bloqueio fatal** (***deadlock***): Este problema acontece quando uma dada seqüência de disparos, a partir da marcação inicial, conduz a uma situação em que não há qualquer transição sensibilizada [2].

d) **Paralelismo Estrutural**: duas transições *t1* e *t2* são paralelas, estruturalmente, se não possuem nenhum lugar de entrada em comum, como demonstrado na equação 6:

$$\mathrm{Pr}\,e(.,t_1) \quad \bullet \quad \mathrm{Pr}\,e(.,t_2) = 0. \quad (6)$$

onde $\bullet$ representa o produto escalar entre os vetores coluna correspondentes.

e) **Paralelismo efetivo**: Duas transições *t1* e *t2* são paralelas para uma marcação *M* se, e somente se, são paralelas estrutural e se cumpre que:

$$M \geq \mathrm{Pr}\,e(.,t_1) \quad e \quad M \geq \mathrm{Pr}\,e(.,t_2). \quad (7)$$

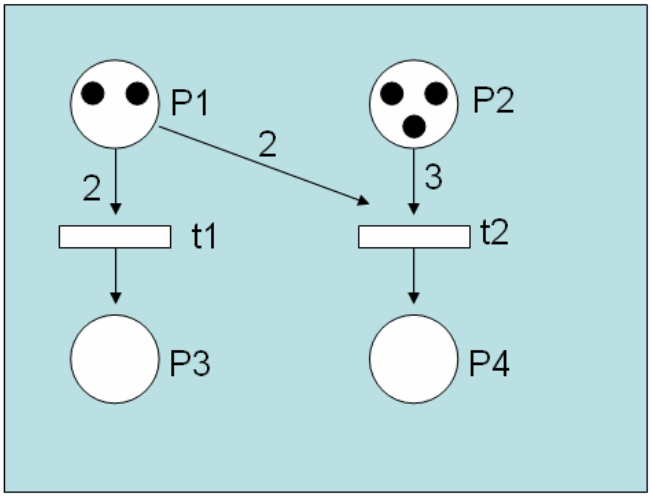


Figura 2.9 - Exemplo de conflito efetivo

2.4.5 Seqüência de Disparo

Seqüência de disparo é uma sucessão de disparos consecutivos levando de uma marcação M para uma determinada marcação M'. Para que a seqüência possa ser executada, a primeira transição tem que estar sensibilizada, após o disparo da primeira transição, a segunda transição da seqüência tem que estar sensibilizada e assim por diante, até que seja executada a última transição da seqüência [13].

Numa seqüência de disparo, cada transição será disparada na ordem estabelecida. Somente uma transição é disparada por vez (vide propriedade das RdPs Ordinárias). A seqüência de disparo pode ser representada como:

$$M \xrightarrow{t_a \; t_X \; t_j} M^n \qquad (8)$$

Onde M é a marcação inicial, M^n a marcação final. A seqüência indicada acima da seta representa a ordem de disparo.

Após o disparo desta seqüência, surge o vetor *S*. Este vetor também é chamado de vetor característico da seqüência *S*, que representa o número de ocorrências de uma transição *t*. O vetor *S* apenas informa quantas vezes cada transição foi disparada, entretanto não estabelece a ordem deste disparo.

Simplificando a equação 3, obtemos a equação 9 que também pode ser chamada de equação fundamental das RdPs. Ela pode ser utilizada tanto para realizar uma seqüência de disparo quanto para verificar se um determinado estado pode ser alcançado, dado um estado inicial e obter quais e quantas vezes cada transição deve ser disparada para chegar a este estado.

$$M' = M + C * S \qquad com \quad M \geq 0, S \geq 0. \qquad (9)$$

Onde * e o operador de multiplicação matricial.

A figura 2.10 mostra a seqüência de disparo $M^o \xrightarrow{t1\ \ t2\ \ t3} M'$ da RdP ilustrada na figura 2.6. M^i representa as diferentes marcações obtidas a partir do disparo das transições da seqüência.

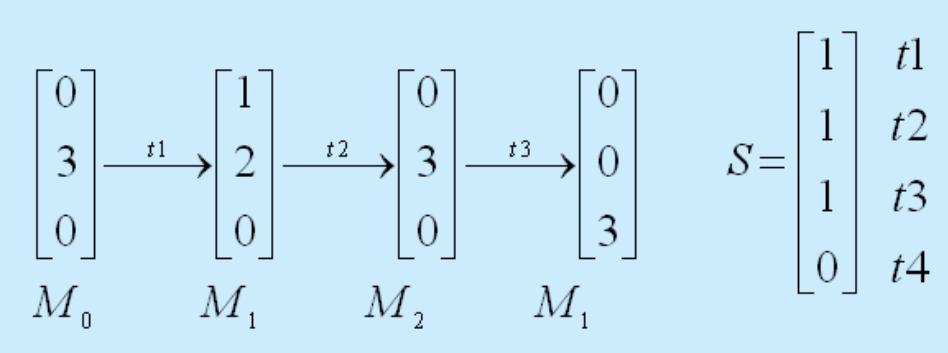


Figura 2.10 – Seqüência de disparo [2]

Pode ser observado que a transição *t1*, t2 e *t3* foram disparadas uma única vez. A transição *t4* não foi disparada na seqüência de disparo definida (vide matriz S da figura 2.10).

A figura 2.11 exemplifica o uso da equação fundamental. O objetivo é achar o vetor *S* que permite sair do estado inicial *M0* e chegar ao estado *M1* como observado na figura 2.10.

$$\begin{bmatrix}1\\2\\0\end{bmatrix} = \begin{bmatrix}0\\3\\0\end{bmatrix} + \begin{bmatrix}1 & -1 & 0 & 0\\-1 & 1 & -3 & 3\\0 & 0 & 1 & -1\end{bmatrix} * \begin{bmatrix}t1\\t2\\t3\\t4\end{bmatrix}$$

$$t1 - t2 = 1 \quad t3 = t4 = 0 \qquad t2 = {}^{0}\!/_{0}$$

$$s = \begin{bmatrix}t2+1\\ {}^{0}\!/_{0}\\0\\0\end{bmatrix}$$

Figura 2.11 - Exemplo de aplicação da equação Fundamental

Devido ao fato de que neste exemplo a RdP não possui o mesmo número de transições e lugares, a equação resulta em um sistema não-determinístico, e, portanto, com solução não única. A solução do sistema contém um conjunto soluções possíveis que pode ser observado na figura 2.10. Para chegar ao estado *M1*, saindo de *M0*, possui pelo menos duas soluções: $S = [1 \quad 0 \quad 0 \quad 0]^T$ ou $S = [2 \quad 1 \quad 0 \quad 0]^T$.

2.4.6 Árvores de alcançabilidade

Uma árvore de alcançabilidade torna possível enumerar, exaustivamente, as marcações alcançáveis possíveis em uma RdP partindo de uma marcação inicial. Estas marcações são descritas por um gráfico do tipo árvore, onde os nós são vetores de marcações, alcançados sucessivamente e alternadamente pela rede. Os arcos correspondem às transições executadas [14].

A figura 2.12 ilustra a árvore de alcançabilidade da RdP definida na figura 2.6; pode-se observar que a árvore descreve todos os estados atingíveis daquela rede. Este método apresenta-se como alternativa para a construção de simuladores.

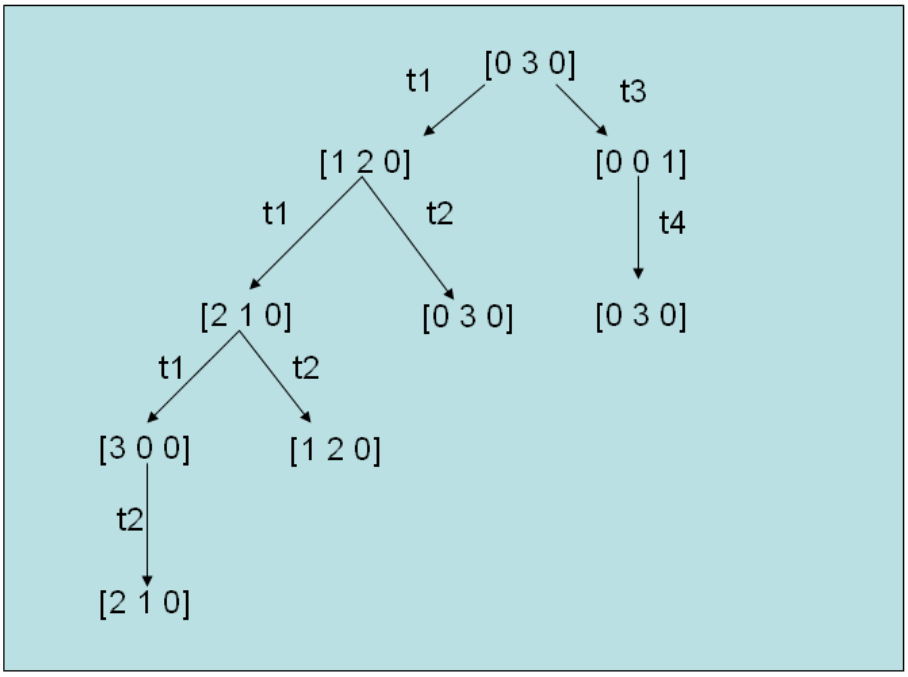


Figura 2.12 - Árvore de alcançabilidade

2.4.7 Conjunto de Marcações Acessíveis

É um conjunto de marcações que podem ser alcançadas a partir de um estado inicial por intermédio de uma seqüência de disparos [1].

Este conjunto é representado pela equação 10 onde A representa o conjunto de marcações acessíveis:

$$A(R,M) = \{M_i, \exists S \quad M \xrightarrow{s} M_i\} \qquad (10)$$

Se o conjunto for finito, poderá ser representado na forma de grafo $G_{A(R,M)}$.

2.5 PROPRIEDADES DA RdPs

Nesta seção serão definidas as propriedades das RdPs.

2.5.1 Rede Marcada k-limitada

O conceito de rede limitada corresponde ao fato de que um sistema físico é sempre limitado. Entretanto, podem-se utilizar uma RdPs não limitada quando se quer avaliar o desempenho de um sistema independentemente dos seu limites de elementos de armazenamento intermediários [12]. A seguir serão dados alguns conceitos relacionados com redes limitadas:

a) **Lugar *k*-limitado e binário**: todo lugar *p* de uma rede marcada *N* será *k*-limitado se, e somente se:

$$\forall M' \in A(R,M), \quad M'(p) \leq k. \qquad (11)$$

Quando *k*=1, o lugar é denominado de binário. Para a marcação inicial da rede descrita na figura 2.6 ($M = p_2^3$ ou $M = [0 \quad 3 \quad 0]^t$), o lugar p_3 é binário, enquanto p_1 e p_2, são 3-limitadas;

b) **RdPs marcada *k*-limitada e rede marcada binária**: uma rede marcada *N* é *k-limitada*, se, e somente se, todos seus lugares são *k-limitados*;

Uma rede marcada *N* é binária, se, e somente se, todos seus lugares forem binários. Neste caso, podem ser chamadas também de redes *salvas* ou *seguras*.

2.5.2 Rede marcada viva

Uma importante característica das RdPs é a propriedade de *vivacidade*. Por meio desta propriedade podemos observar diversas características como citado a seguir:

a) **Transição quase-viva**: para que uma transição seja considerada quase-viva é necessária uma seqüência de disparos tal que:

$$\exists S \mid M \xrightarrow{s} M' \quad e \quad M' \xrightarrow{t} \qquad (12)$$

logo, para ser considerada uma transição quase-viva deverá ser possível que uma transição *t* seja sensibilizada a partir de uma marcação M, obtendo-se uma nova marcação *M'*, usando uma seqüência de disparo S [7]. Esta propriedade pode ser observada na figura 2.13 se o disparo das transições seguir a ordem de disparo *t*1, *t*2 e *t*3. Neste caso *t*2 e *t*3 são quase vivas dado que seus disparos dependerão de certas marcações apropriadas.

b) **Transição viva**: uma transição pode ser considerada viva, se e somente se uma transição *t* puder ser sensibilizada por qualquer marcação *M'* obtida por uma seqüência de disparo *S* como descrito na equação 14 [7], este comportamento pode ser observado na figura 2.13:

$$\forall M' \in A(R,M), \exists s \mid M' \xrightarrow{st} \qquad (13)$$

Neste exemplo *t1* é uma transição viva.

c) **Rede marcada viva**: uma RdPs marcada somente poderá ser considerada viva se, e somente se, todas as suas transições forem vivas. Logo, esta propriedade assegura que não haja nenhum bloqueio gerado pela estrutura da rede [8]. Este comportamento pode ser observado na figura 2.14.

d) **Rede marcada reiniciável**: uma rede marcada só poderá ser considerada reiniciável se, e somente se, a partir de qualquer marcação M'de $G_{A(R,M)}$, seja possível encontrar uma seqüência de disparos capaz de levar ao estado inicial. Esta propriedade é utilizada em redes com comportamentos repetitivos, podendo ser verificada por meio da equação 15. Esta propriedade pode ser observada na figura 2.14.

$$\forall M' \in A(R,M), \quad \exists s \mid \quad M' \xrightarrow{s} M \qquad (14)$$

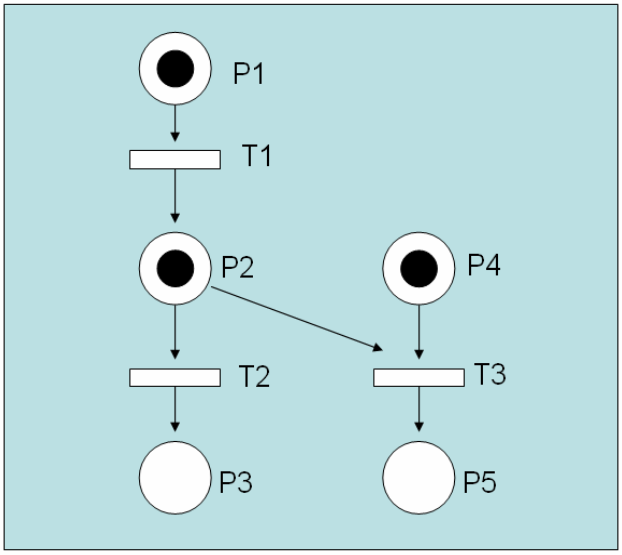


Figura 2.13 - Exemplo de transição quase-viva

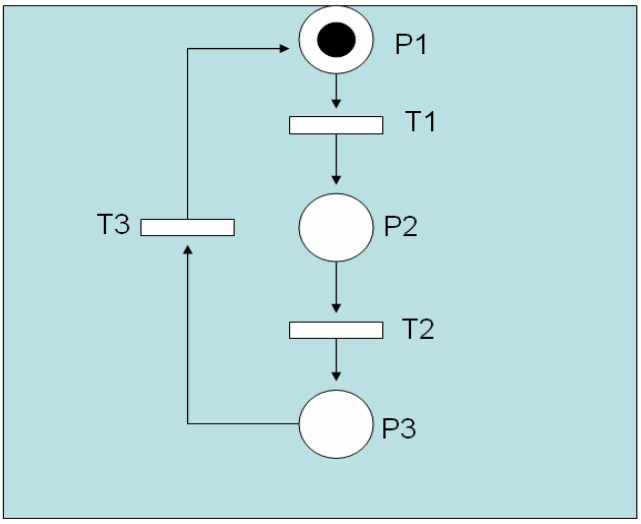


Figura 2.14 - Rede reiniciável

2.5.3 Propriedades Estruturais

Segundo [2], estas propriedades são independentes da marcação ou do estado inicial; referem-se diretamente às propriedades derivadas da estrutura da rede. A aplicação destas propriedades permite informações adicionais sobre o comportamento dinâmico de uma RdPs. A seguir serão definidas algumas destas propriedades.

a) **Componentes conservativos, invariantes de lugar**: pode-se observar uma invariância de lugar sempre que a soma de recursos (fichas) de dois ou mais lugares permanecerem constantes independentemente de qualquer seqüência de disparo. Esta propriedade pode ser verificada por meio da equação 16.

$$\forall M \in A(R, M_0), \qquad M(p_a) + M(p_b) = M_0(p_a) + M_0(p_b) \qquad (15)$$

Uma invariante linear de lugar é uma função linear da marcação dos lugares cujo valor é uma constante que depende apenas da marcação inicial da rede. Com isto, é possível verificar, independentemente de qualquer que seja a evolução da rede, suas restrições sobre os estados e atividades do sistema.

Na figura 2.15 podemos observar um exemplo de invariante de lugar, onde, *Mi (Pt1) + M1(Pt2) = M0(Pt1)+M0(Pt2)* e invariante de transição, onde *S = t2t3*.

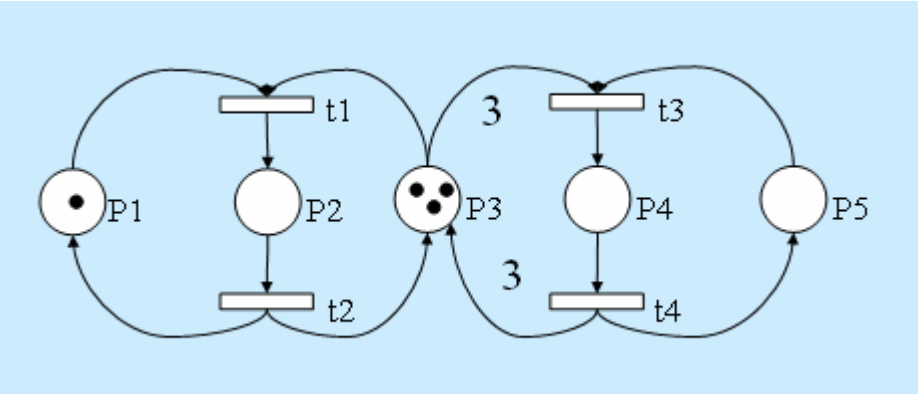


Figura 2.15 - Exemplo de invariante

b) **Componentes repetitivos, invariantes de transição**: podem-se observar invariâncias de transições sempre que for disparado. Esta propriedade é facilmente verificada aplicando-se a equação 9 se $M = M'$, logo $C*S = 0$.

Neste caso, todas as soluções da equação 9 são chamadas de componentes repetitivos.

2.5.4 Transições neutras ou identidade

Uma transição neutra não interfere no comportamento da rede, seu disparo não modifica em nada a marcação da rede. Uma rede que possui este comportamento é chamada de *RdP* não-pura. Este comportamento poderá ser observado sempre que após um disparo de transição a função *Pre*(.,t), for igual à função *Post*(.,t) [3].

A figura 2.16 exemplifica este comportamento, pois a transição *t*5 quando disparada não altera a marcação da RdP.

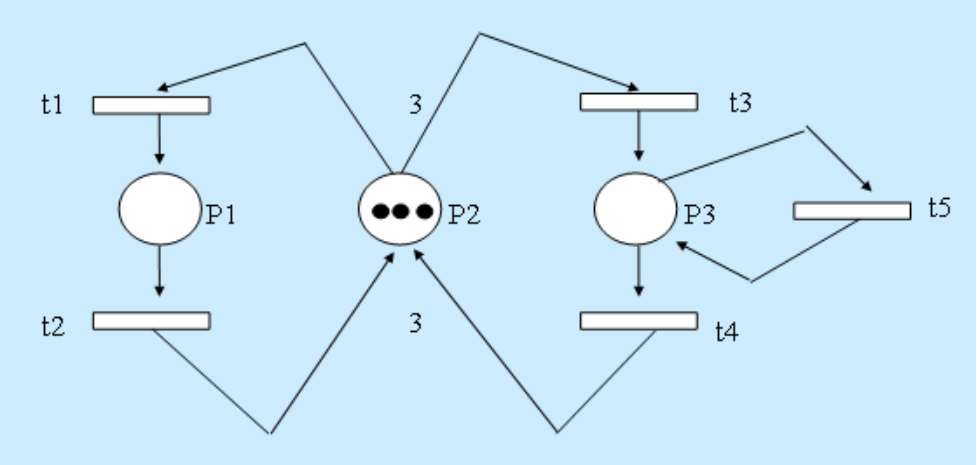


Figura 2.16 - Rede dePetri não pura

2.6 TAXONOMIA DAS RdPs

Nesta seção são descritos os tipos de Redes de Petri objeto deste trabalho e que são referenciados na literatura. Devido à complexidade dos modelos existentes, surge a necessidade da evolução das Redes de Petri, adicionando ao seu modelo mecanismos capazes de modelar ambientes levando em consideração o tempo, ambiente externo, dados estatísticos, aspectos dinâmicos, entre outros.

2.6.1 RdPs Interpretadas

Estas se caracterizam por introduzirem variáveis às transições da rede, podendo com isto representar condições e ações existentes no sistema. Tais variáveis podem interagir com o ambiente externo, como por exemplo, sensores (em sistemas de automação e de manufatura) [9].

Diferentemente das RdPs ordinárias, que para disparar uma transição basta que a mesma esteja sensibilizada, as redes interpretadas para ter uma transição disparada, tem que estar sensibilizadas e satisfazer a condição que a elas foi atribuída. Com isto, torna-se possível o disparo de várias transições ao mesmo tempo [1].

Segundo [1], "interpretar uma RdP, implica em dar sentido concreto a um modelo matemático", a RdPs interpretada estrutura-se em duas partes:

a) **Controle**: descreve todos os encadeamentos potenciais de eventos e de atividades, este controle é descrito pelas Redes de Petri Ordinárias;

b) **Dados**: é a parte operativa do sistema. Descreve ao mesmo tempo as estruturas de dados internas ao sistema e os cálculos que são feitos sobre estes dados, sem especificar em quais instantes eles são realizados.

A figura 2.17 ilustra a interação da RdP com o ambiente externo.

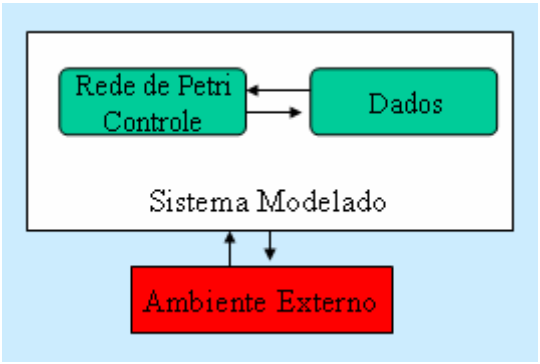


Figura 2.17 - Interação da rede com o ambiente externo [1]

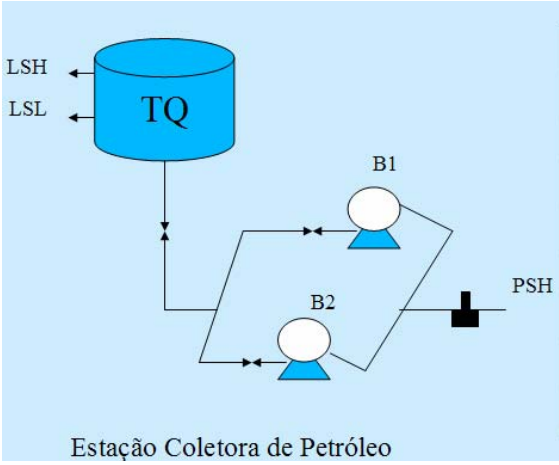
A RdP descreve apenas a parte de controle do sistema. A marcação da rede fornece, portanto, o estado do controle.

O estado do sistema, estado em que a RdPs interpretada é descrita pela marcação associada ao estado dos dados. Logo:

- *Estado rede interpretada = marcação + estado dados*;
- *Estado dados = estado variáveis internas + tempo transcorrido*.

A figura 2.18 apresenta um exemplo de RdPs interpretada, descrevendo uma estação coletora de petróleo, onde um determinado tanque possui dois sensores que controlam o volume deste reservatório. O controle é realizado por duas bombas que bombeiam óleo para dentro do tanque, sempre que o sensor de nível baixo do tanque é acionado, e desliga sempre que o sensor de nível alto do tanque é acionado.

Somente é ligada uma bomba de cada vez, a segunda bomba somente será ligada caso a primeira bomba não ligue após dois segundos do acionamento do sensor. Existe também um sensor que controla a pressão da bomba como mostrada na figura 2.18 [1].



t_i	Nome	(C_i, A_i)
t_1	B2_Resp_Desligada	(;B2_ligada=0^lsl=0)
t_2	B2_Rec_Sinal_Liga	(liga_B2=1;liga_b2=0)
t_3	B2_Resp_Ligada	(;B2_ligada=1^lsh=0)
t_4	B2_Rec_Sinal_Desl	(desl_b2=1;desl_b2=0)
t_5	B1_Resp_Desligada	(;B1_ligada=0^lsl=0)
t_6	B1_Rec_Sinal_Liga	(liga_B1=1;liga_b1=0)
t_7	B1_Resp_Ligada	(;B1_ligada=1^lsh=0)
t_8	B1_Rec_Sinal_Desl	(desl_b1=1;desl_b1=0)

Figura 2.18 - Exemplo estação coletora de petróleo [1]

A figura 2.19 apresenta o modelamento da estação coletora de petróleo apresentada na figura 2.18.

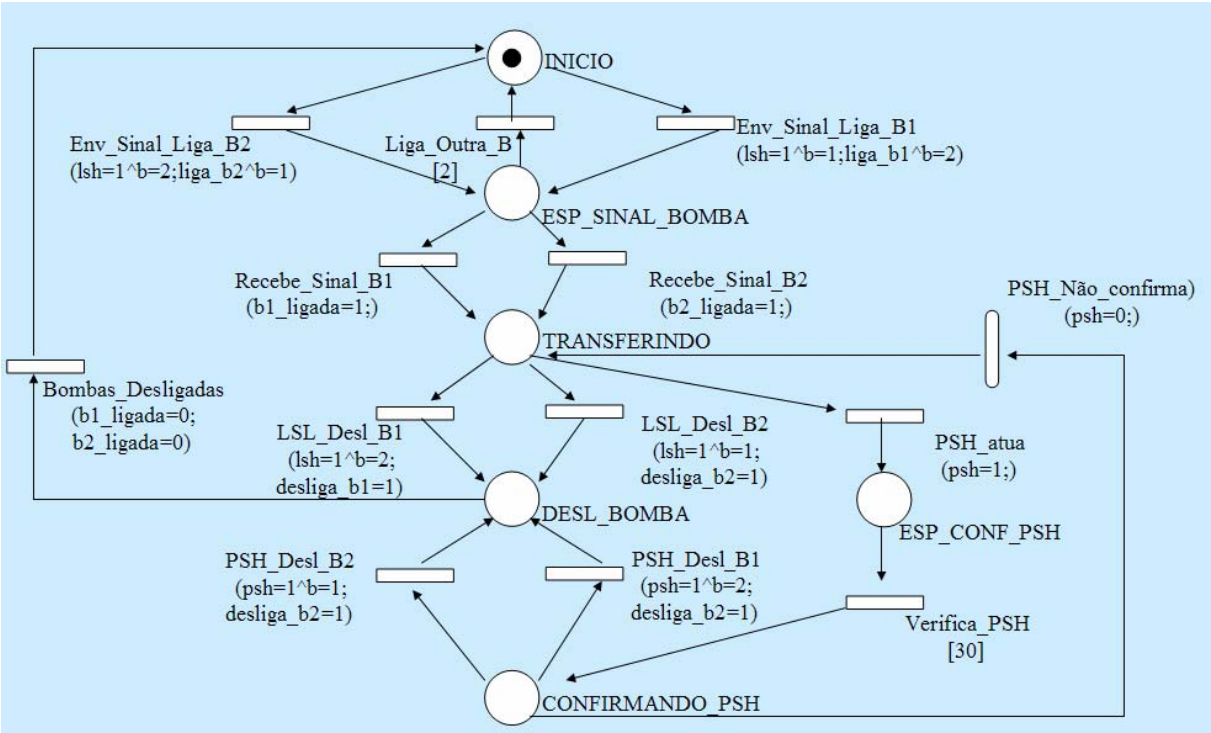


Figura 2.19 – Modelamento da RdPs Interpretada [1]

2.6.2 RdPs Temporais e Temporizadas

Tendo o tempo como parte do controle as Redes de Petri, temporal e temporizada, modelam o comportamento dos seus processos levando em conta estas características.

a) **RdPs Temporais**: para este modelamento a cada transição é associada um par de datas $(\theta_{min}, \theta_{max})$, onde θ_{min} e θ_{max} indicam a duração mínima e máxima, correspondente, de sensibilização da transição do disparo [9], como ilustrado na figura 20. θ_{min} e θ_{max} representam datas referentes a um relógio comum, que representa o tempo absoluto de toda a rede.

Pode-se associar um intervalo [*a,a*] a uma transição para representar uma duração *a*. Se a transição estiver no tempo τ , a mesma irá disparar no tempo $\tau + a$, caso nesta data continue sensibilizada pela transição [1].

Segundo [12], formalmente uma RdP temporal é definida por par $N_{tl} = \langle N, I \rangle$, onde:

- *N* é uma RdPs Ordinária;
- $\theta(t) = [\theta_{\min}(t), \theta_{\max}(t)]$ é uma função que para cada transição *t* associa uma duração de sensibilização. A duração *θ(t)* é definida como um intervalo fechado racional.

Na figura 2.20, a transição *t*1 só está sensibilizada entre os valores temporais (absolutos) um (1) e dois (2). Neste tipo de rede se assume de maneira implícita que existe um relógio único no sistema que define os momentos em que as transições estarão sensibilizadas.

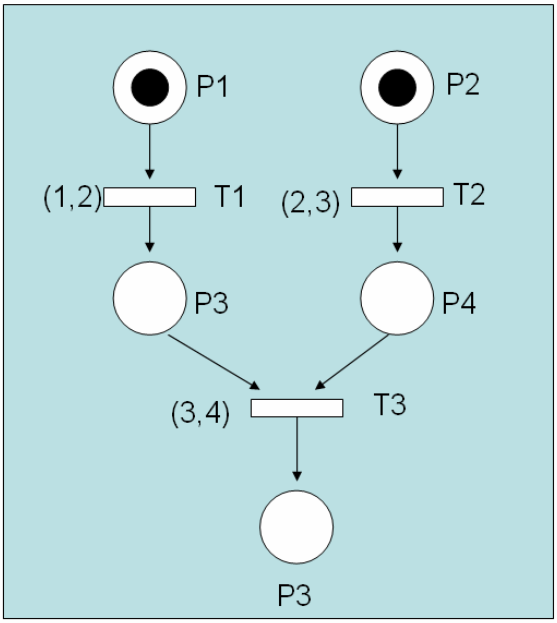


Figura 2.20 - Redes Temporais

b) **RdPs Temporizadas:** neste modelo cada transição associa-se a uma duração a fim que a transição seja disparada, logo quando uma ficha chega a um determinado lugar em uma data τ, ela só deixará o lugar após $\tau' = \tau + \theta$ instantes [1].

Segundo [1], formalmente uma RdPs temporizada é definida por par $N_{tl} =< N, \Theta_f >$, onde:

- N é uma RdP ordinária;
- $\Theta_f : T \rightarrow Q^+$ é a função de duração de disparo, que associa a cada transição um número racional positivo que descreve a duração do disparo.

A figura 2.21 apresenta um exemplo de uma RdPs temporizada. Neste caso, em um momento dado (t = τ_1) um recurso está disponível em P1, sensibilizando $t1$. Entretanto, este recurso (ficha) só estará disponível em P2 um tempo $\theta 1$ depois. Isto pode ser interpretado como um retardo introduzido por $t1$ na transferência da ficha de P1 para P2.

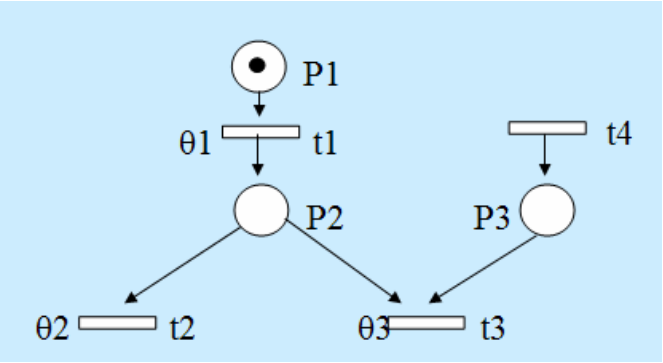
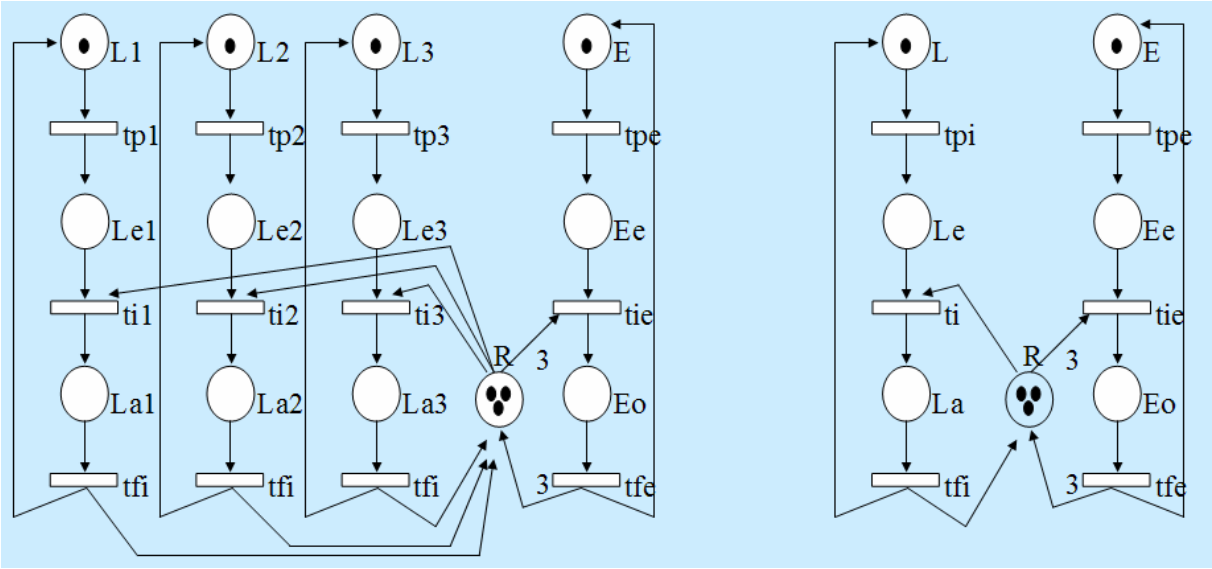


Figura 2.21 – Exemplo de RdPs Temporizada [1]

2.7 OUTROS TIPOS DE RdPs

Nesta seção serão descritos, sucintamente, outros tipos de RdPs referenciadas na literatura. Estas taxonomias não foram contempladas no desenvolvimento do SimRP, devido à complexidade da implementação das outras taxonomias escolhidas para o simulador. Neste trabalho é apresentada esta taxonomia para auxiliar no desenvolvimento de trabalhos futuros**.**

Um importante tipo de RdPs são aquelas consideradas de alto nível. Estas redes objetivam simplificar seu modelamento. O comportamento de leitores e escritores pode ser um bom exemplo para explicar seu funcionamento, já que o comportamento dos leitores é sempre o mesmo, independente de ser o leitor *1* ou o leitor *n*, como representado na figura 2.22. Nesta figura pode ser observado um conjunto de leitores (L1, L2 e L3) que interagem com um único escritor (E).



(a) **(b)**

Figura 2.22 - Redes dobradas [1]

Nesta figura podemos observar que o comportamento dos leitores (L1, L2 e L3) é o mesmo. Neste caso, o que se modifica e a interação entre estes leitores e o escritor (vide figura 2.22

(a)). Pode ser visto que cada leitor L_i executa várias tarefas de acordo com o disparo de transições que levam para novos lugares na RdP. Da mesma maneira o escritor executa suas respectivas tarefas como descrito na figura 2.22.

Se tivesse um número muito grande de leitores, ficaria muito difícil entender o comportamento desta rede. Surgiu então o conceito de dobrar a rede, que consiste em agrupar os lugares e transições que juntos apresentam comportamento similar, como demonstrado na rede da figura 2.22 (b), onde os três leitores são agrupados num único leitor (L). Com isto, a modelagem torna-se menos complexa e facilita a visualização do comportamento da rede. Com o dobramento da rede (agrupamento de elementos da mesma), o próximo passo é diferenciar o comportamento de cada leitor individualmente na rede dobrada [12].

A seguir são apresentados os modelos que se baseiam na idéia de dobramento de Rede, como no caso de redes coloridas, redes predição-transição e redes a objeto.

2.7.1 RdPs Coloridas

As RdPs coloridas estabelecem uma cor (números inteiros ou conjunto de etiquetas) diferente para representar cada um dos processos dobrados. Como conseqüência, a cada lugar é associado um conjunto de cores das fichas que podem pertencer a este lugar, e a cada transição é associado um conjunto de cores correspondentes às diferentes maneiras de disparar uma transição [1].

A RdPs Colorida é uma sêxtupla, definida por:

- P é um conjunto finito de lugares;
- T é um conjunto finito de transições;

- C_{or} é um conjunto finito de cores;
- C_{SC} é um conjunto finito de cores que cada lugar e cada transição associa um sub-conjunto de C_{or} (as cores possíveis para este lugar ou esta transição): $C_{sc}:P \cup T \rightarrow P(C_{or})$;
- W é uma função de incidência (equivalente a C = Post – Pre); cada elemento de $W(p,t) : C_{SC}(t) \times C_{SC}(p) \rightarrow N$;
- M_0 é uma marcação inicial que associa, para cada lugar e para cada cor possível neste lugar um número de fichas $M_0(p):C_{SC}(P) \rightarrow N$.

A figura 2.23 apresenta um exemplo de modelamento de RdPs Colorida. Este trata do modelamento do comportamento de um processo de usinagem que consiste em três máquinas: maq1, maq2, maq3; e três peças: pc1, pc2, pc3. No exemplo, a rede foi dobrada como mostrada na figura 22, Na figura 24 também é apresentada a definição formal desta rede.

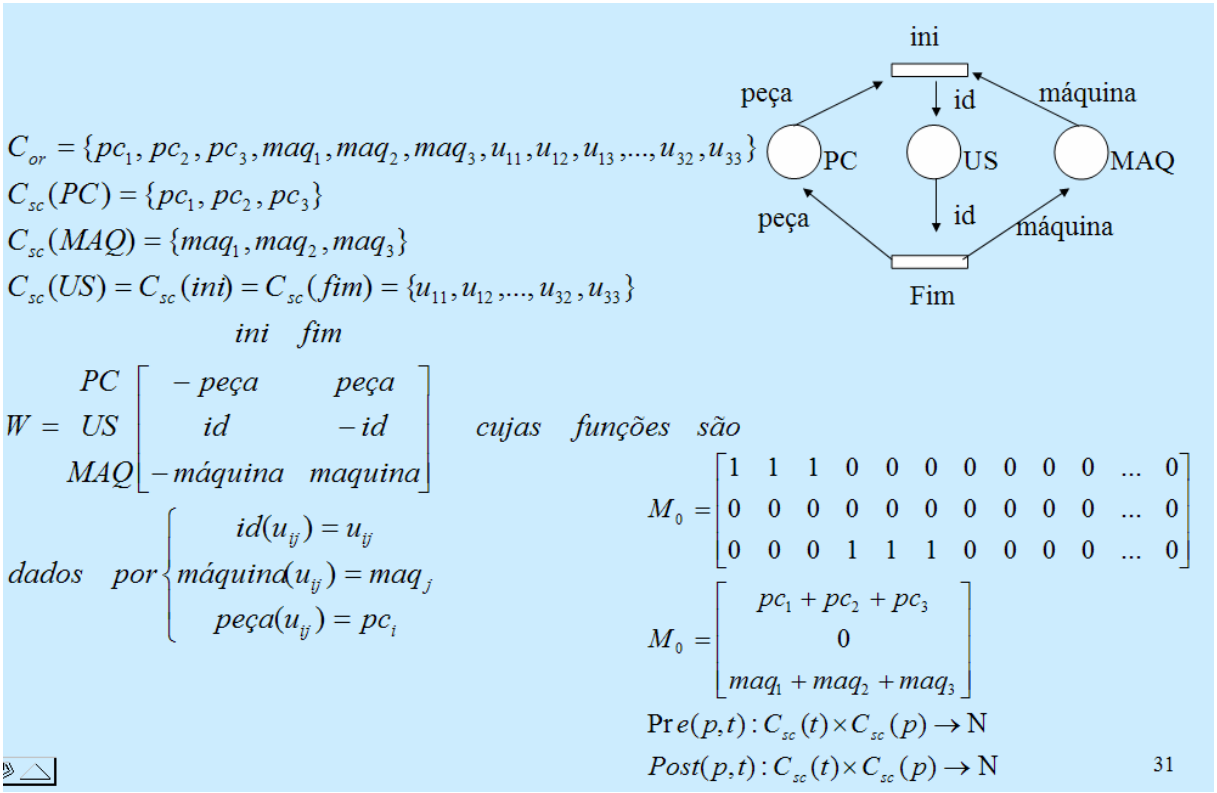


Figura 2.23 -Exemplo de RdPs Coloridas

Esta modelagem pode se tornar inviável computacionalmente, quando o número de cores for elevado, já que as funções Pre e Post são formadas pelo produto cartesiano do conjunto de cores [2].

2.7.2 RdPs Predicado-Transição

As transições são consideradas como regras num sistema lógico-proporcional (sem variáveis), e o poder de descrição é aumentado substituindo-se por regras da lógica da primeira ordem (com variáveis) [1].

Com isto, uma transição deverá descrever uma família de eventos e não somente um evento como no caso das redes coloridas.

A RdPs Predicato-Transição é definida formalmente por uma quíntupla, como mostrado na figura 2.24.

$$N_{pt} =< P, T, V, \mathrm{Pre}, Post, C_{onst}, A_{ct}, A_{ta}, M_0 >$$

- P é o conjunto de lugares, T é o conjunto de transição
- V é um conjunto de variáveis
- Pre e Post
- Const é um conjunto de constantes
- $A_{ct} : T \rightarrow L_c(C_{onst}, v)$
- $A_{ta} : T \rightarrow L_a(C_{onst}, v)$
- M_0 Estado inicial, associada a cada lugar p de P uma soma formal de n-uplas de constantes

Figura 2.24 – Definição formal da rede predicato-transição

A figura 2.25 apresenta o modelamento em RdPs Predicato-transição, do exemplo apresentado no item anterior – RdPs Coloridas.

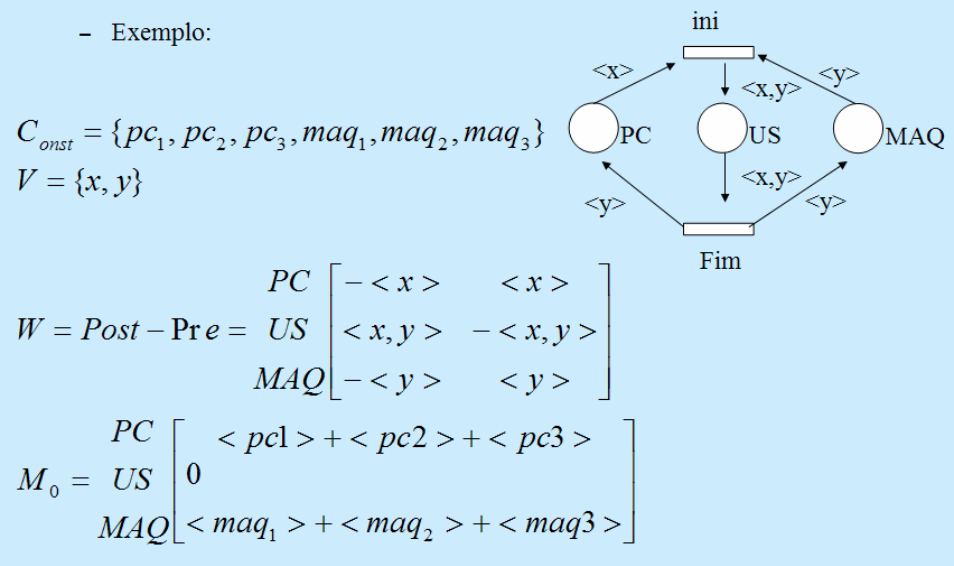


Figura 2.25 - Exemplo de Rede Predicato-Transição [1]

Este modelamento apresenta-se mais interessante do que o anterior, já que a função Pre e Post, não mais consiste em uma operação cartesiana, mas sim de conjunto de variáveis que utilizadas quando necessárias no decorrer do desenvolvimento da rede.

No caso das RdPs predicato-transição, as transições de uma RdPs ordinária são consideradas como regras num sistema de lógica proposicional (sem variáveis), e o poder de descrição é aumentado substituindo-se por regras lógicas de primeira ordem (regras de variáveis). Com isto, a representação fica mais concisa.

Uma regra (transição) descreve, então, uma família de eventos e não mais somente um evento. A família é definida pelo conjunto de substituições possíveis de variáveis por valores. Ao invés de regras do tipo:

- Se uma peça pc_1 e máquina M_2, fazer usinagem; u_{12}**;**
- Se uma peça pc_2 e máquina M_2, fazer usinagem; u_{22}**;**
- Se uma peça pc_i e máquina M_j, fazer usinagem; u_{ij}**;**

Tem-se regras do tipo:

Se uma peça <x> é uma máquina <y>, fazer uma usinagem <u>, onde as variáveis <x>, <y> e <u> assumirão, respectivamente, valores no conjunto de constante $\{pc_i\}$ descrevendo as peças em espera, o conjunto M_j das máquinas livres, e o conjunto $\{u_{ij}\}$ das opções a serem realizadas.

2.7.3 RdPs a Objeto

Este modelamento utiliza-se do conceito de objetos, muito utilizado no meio computacional e na engenharia de Software. A RdPs a Objetos utiliza-se dos conceitos da RdPs Predicato-Transição, adicionado ao modelo conceitos de Classes de Objetos, atributos, métodos e herança. Conceitos abstraídos da teoria de Orientação a Objeto [1].

Formalmente uma RdPs a Objetos é definida por uma 9-upla, como mostrada na figura 2.26.

$$N_o =< P, T, C_{lass}, V, P_{re}, P_{ost}, A_{tc}, A_{ta}, M_0 >$$

- Class conjunto finito de classe e objetos;
- P é um conjunto de Lugares
- T conjunto de transições
- V Conjunto de variáveis
- A_{tc} é uma aplicação que a cada transição uma condição fazendo intervir as variáveis formais associadas aos arcos de entrada e aos atributos das classes correspondentes.
- A_{ta} é uma aplicação que cada transição associa uma ação fazendo intervir as variáveis formais associadas aos arcos de entrada e aos arcos de entrada e aos atributos das classes correspondentes.
- M_0 Marcação Inicial

Figura 2.26 - Definição formal da RdPs a Objetos [1]

Para ilustrar melhor a RdPs a Objetos, a figura 2.28 apresentam o modelamento em RdPs a Objeto, do mesmo exemplo utilizado nos dois tópicos acima – Redes Coloridas e Predicato-Transição:

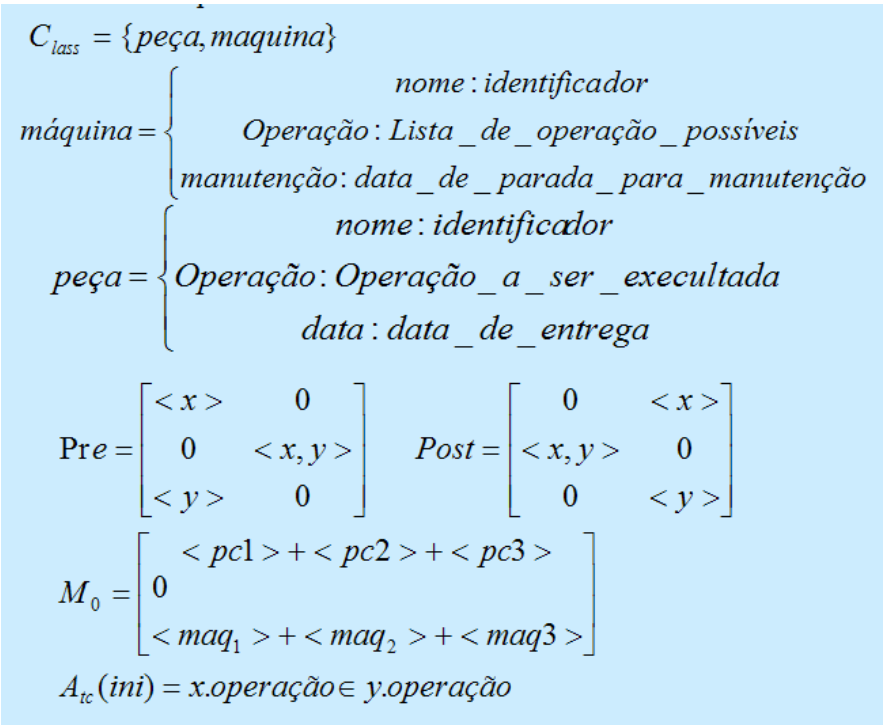


Figura 2.27 - Exemplo de RdPs a Objetos [1]

Note que este modelamento é bem similar ao da RdPs Predicato-Transição, contudo mais poderoso, já que agrega atributos capazes de identificar, por exemplo, o nome, operações possíveis, data de manutenção entre outros.

2.8 LINGAGEM DE ESPECIFICAÇÃO DE HARDWARE VHDL

Nesta seção serão explicados, sucintamente, alguns conceitos sobre a linguagem de descrição de hardware VHDL. O Projeto Very Higd Speed Integrated Circuits (VHISC) foi desenvolvido no ano de 1980 pelo Governo do Estados Unidos da América, com o objetivo de desenvolver circuitos integrados de alta velocidade e realçar a descrição projetos eletrônicos. Em uma tentativa de padronizar a descrição de um projeto de Hardware, o IEEE – Institute of Electrical and Eletronic Enginneering apoio à criação do VHDL – VHSIS Hardware Descripion language em 1993 o VHDL foi aprovado, atualizado e definido com o padrão IEEE 1076.1[14].

O Objetivo da linguagem é estruturar de forma simples a descrição de circutios digitais, usando estruturas sintáticas e semânticas similares às linguagens de programação de alto nível como Pascal e C. Como toda linguagem de programação, o VHDL possui elementos léxicos, operadores lógicos, tipo de dados, operadores e sentenças lógicas.

2.8.1 Elementos léxicos básicos da linguagem

O VHDL não é *case-sensitive*, os identificadores são palavras reservadas ou nomes definidos pelo programador; uma linha de comentário é definida por dois hífens "- -"; os números podem ser expressos no formato decimal inteiro, inteiro, decimal exponencial, bases definidas entre 2 e 16, números com ponto flutuante que neste caso estarão definidos como real. Os caracteres utilizados na linguagem são correspondentes à tabela ASCII e as string são as concatenações destes caracteres, alem desses tipos de dados o VHDL possui outro patrão, o bit strings, que são conversões para definir um padrão em diferentes bases, por conversão, cada base tem uma letra no início que a identifica, exemplo B Binário e X hexadecimal, neste caso o valor X "16" é 16 em hexadecimal.

2.8.2 Tipo de Dados

Segundo [15], os tipos de dados primitivos definidos na linguagem possibilitam a construção de tipos derivados.

a) **Dados primitivos**:

- Inteiro: (*integer*), com uma faixa entre –2147483647 a +2147483647;
- Pontos flutuantes (*real*), que variam entre –1E38 a +1E38;

- Físicos que são tipos numéricos representativos de algumas grandezas físicas, como massa, comprimento, tempo ou tensão elétrica.

b) **Dados derivados**:

Entre os tipos derivados, temos os tipos enumerados, que são conjuntos ordenados de identificadores ou caracteres, *arrays*, que são conjuntos indexados de elementos de algum tipo primitivo e os *records* que são conjuntos de tipos de dados primitivos distintos. Existem ainda, outros tipos de dados derivados, como os *subtypes* e os *attributes*, como é mostrado na figura 2.28.

```
TYPE octal_digit IS (‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’);
TYPE word IS ARRAY (31 downto 0) OF BIT
TYPE instruction IS RECORD
        opcode: processor_op;
        address_mode: mode;
        operand1: INTEGER RANGE 0 TO 15;
        operand2: INTEGER RANGE 0 TO 15;
END RECORD;
```

Figura 2.28 - Exemplo de dados primitivos

2.8.3 Objetos

Na descrição VHDL, quando um objeto é nomeado, ele assume um valor de acordo com o tipo de dado a que foi associado. As três classes de objetos são: as constantes, as variáveis e os sinais [15]. Exemplo:

CONSTANT real:= 2.71828;

VARIABLE count: INTEGER:=0;

2.8.4 Expressões e operadores

Semelhantemente a uma linguagem de programação, as expressões em VHDL são como combinações de tipos primários e operadores. Tipos primários incluem nomes de objetos, literais, chamadas a funções e expressões entre parênteses. Os operadores estão divididos em Lógicos (and, or, nand, nor, xor, not); relacionais (=, /=, <, <=, >, >=) e aritméticos (+, -, *, /, **).

2.8.5 Sentenças

No VHDL como nas linguagens de programação, as sentenças são estruturas sintáticas, a linguagem contém várias facilidades para modificação do estado dos objetos e para controle do fluxo de execução dos modelos [14]. Na figura 2.29 é mostrada a sentença de repetição mais utilizada como IF mostrado na figura 2.29 (a), o CASE mostrado na figura 2.29 (b) e WHILE na figura 2.29 (c).

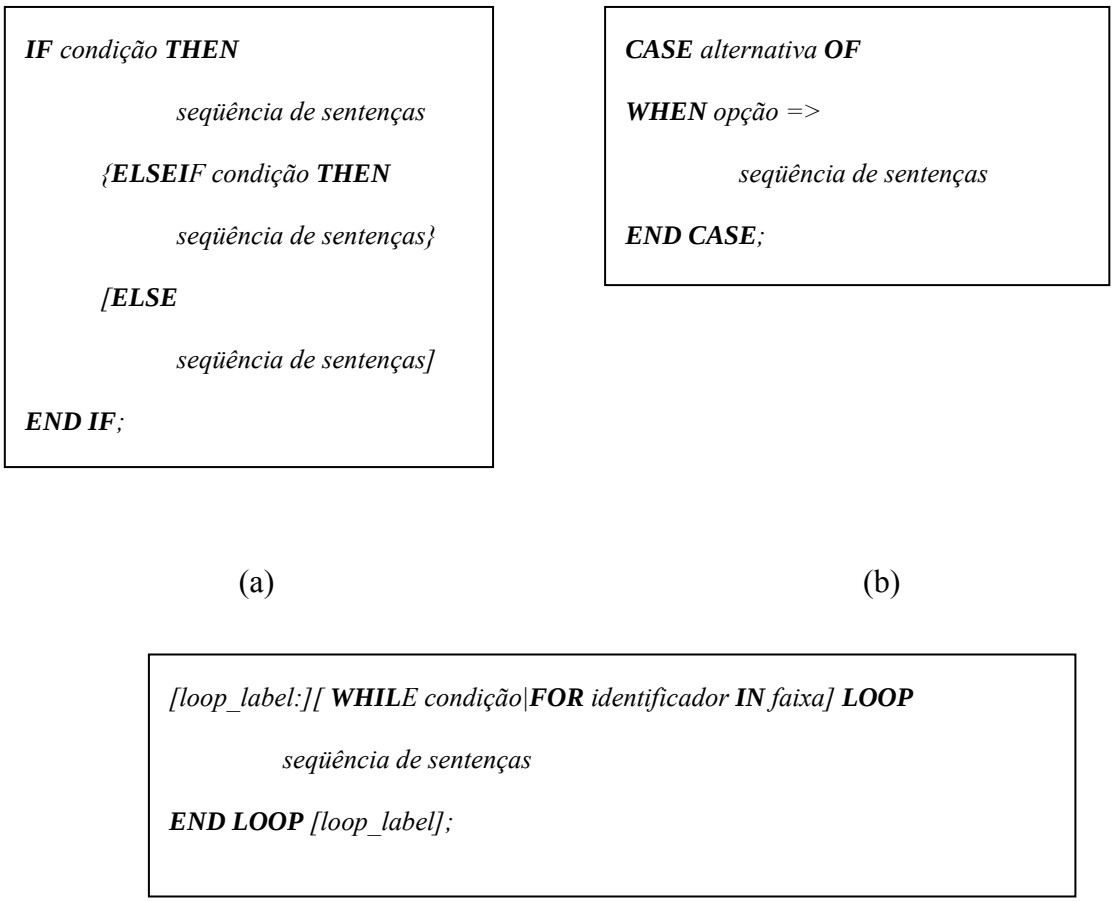
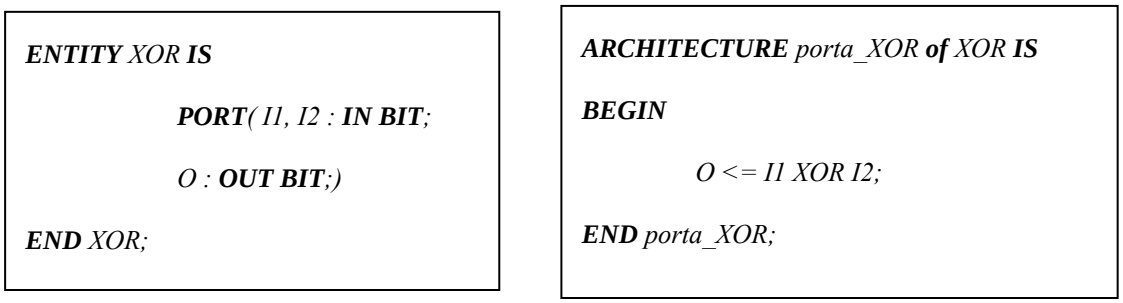


Figura 2.29 - Exemplo de Sentenças

2.8.6 Estrutura da Linguagem

Segundo [14], uma especificação VHDL pode ser dividida em duas partes fundamentais. A primeira define a interface e a segunda, seu comportamento. A interface é definida por meio da declaração *entity*, nela são declarados os nomes dos componentes e suas portas de entrada

e saída. O exemplo da figura 2.30 (a) apresenta a declaração de uma entididade chamada *XOR*, que possui duas entradas e uma saída.



```
ENTITY XOR IS
        PORT( I1, I2 : IN BIT;
        O : OUT BIT;)
END XOR;
```

(a)

```
ARCHITECTURE porta_XOR of XOR IS
BEGIN
        O <= I1 XOR I2;
END porta_XOR;
```

(b)

Figura 2.30 - Exemplo de Estrutura

Por meio da declaração *architecture*, define-se o comportamento, e o funcionamento do circuito. São definidas as relações entre entrada e saída por meio algorítmico[14]. O exemplo da figura 2.30 (b) define o comportamento de uma porta *XOR*, a entidade, foi definida no exemplo anterior.

Para realizar a conexão dos componentes de uma estrutura, é utilizado no VHDL o conceito de *sinais*. Para declarar um sinal no VHDL, é necessário associá-lo a um tipo de dado, é possível também atribuir valores no momento da inicialização, como demonstrado no exemplo a seguir: *SIGNAL A: BIT := '0';*

A unidade primária de descrição do comportamento de um circuito é o processo (*process*). O processo caracteriza-se por uma seqüência de códigos que podem ser ativados em resposta à mudança de estados. O processo é executado concorrentemente quando mais de um processo é ativado ao mesmo tempo. Existem dois estados possíveis para um processo: *suspenso* e *ativo*. Todos os processos do modelo podem ficar ativos em qualquer instante [15]. O exemplo da figura 2.31 demonstra a declaração de um processo.

```
PROCESS
BEGIN
        WAIT ON A, B UNTIL Enable = '1';
        T <= A AND B;
END PROCESS;
```

Figura 2.31 - Exemplo de declaração de um processo

Segundo [15], a mudança entre os estados de um processo é controlada por uma sentença chamada *wait*. Quando o processo executa a sentença *wait* ele é suspenso e as condições para sua reativação são ajustadas. As sentenças *wait* podem ser utilizadas da seguinte forma:

- *WAIT*: faz com que o processo fique suspenso indefinidamente. Depois da execução da sentença, o processo não irá ser ativado novamente durante a simulação;
- *WAIT ON signal-list*: suspende o processo, definindo uma lista de sinais que deverão ser sensibilizados para sua reativação;
- *WAIT UNTIL condition*: especifica uma condição, que quando verdadeira, reativa o processo;
- *WAIT FOR time-expression*: determina um tempo que o processo deve permanecer suspenso, para, então, ser reativado;

O VHDL também possibilita atribuir tempo aos sinais; a temporização permite que os valores de tempo sejam atribuídos aos respectivos sinais. A determinação de tempos pode ser controlada pela sentença *after*. O exemplo da figura 2.32 mostra esta funcionalidade:

```
SIGNAL S: INTEGER := 0;
P1: PROCESS
BEGIN
        S <= 1 AFTER 1 ns;
        S <= 2 AFTER 2 ns;
        WAIT;
END PROCESS;
```

Figura 2.32 - Exemplo da sentença AFTER

Segundo [15], o VHDL é uma linguagem genérica para descrever sistemas digitais de hardware não fazendo alusão acerca da tecnologia ou da metodologia usada para descrever o sistema. Uma abstração dos sistemas digitais de hardware é usada como um dos principais fundamentos da linguagem. Neste conceito inclui comportamento, temporização e características estruturais dos sistemas digitais que integram uma linguagem singular, que permite uma ampla faixa de opções para descrição e esquematização.

A linguagem baseia-se em dois modelos interdependentes:

b) **O modelo comportamental**: divide um fluxo de dados e algorítmico, nele a relação E/S define-se através de funções booleanas, modelado um circuito através da escrita de um programa que defina o seu comportamento;

c) **O modelo estrutural**: define um circuito através de seus componentes e a ligação entre eles;

2.8.7 O modelo comportamental

A estrutura e seu comportamento estão presentes em todos os modelamentos. A linguagem também possibilita o uso de bibliotecas, com comportamentos pré-definidos, entretanto, comportamento é diretamente integrado na linguagem e o projetista tem a opção de misturar estrutura e comportamento em qualquer instância do modelo [15].

A transformação de valores discretos de entrada em valores discretos de saída, aplicando-se um número de operações ou transformações nos valores de entrada, define um sistema digital, este comportamento.

2.8.8 O modelo estrutural

Aumentando a complexidade de um modelo de sistema digital, a representação de um sistema discreto pode se tornar difícil de gerenciar. Muitos sistemas digitais são desenhados pela combinação de vários subsistemas [15]. A figura 2.33 exemplifica este conceito.

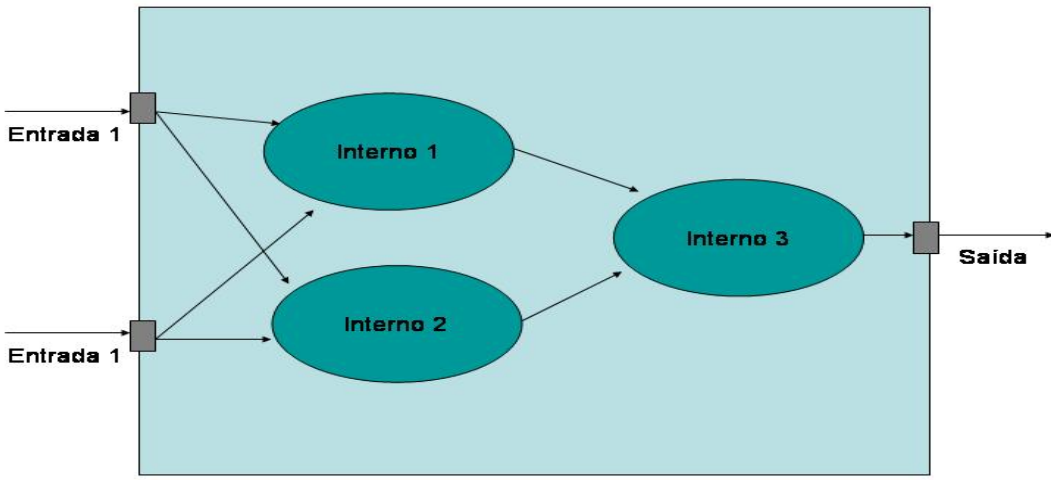


Figura 2.33 - Exemplo de um modelo estrutural

Do ponto de vista de uma descrição comportamental, a descrição de Máquinas de Estados Finitos (*Finite State Machines* - **FSMs**) é um bom exemplo.

Uma FSM é definida por uma sêxtupla {X, Z, Q, T, S, E} onde:

- X é o conjunto de entradas da FSM.
- Z é o conjunto de saídas da FSM.
- Q é o conjunto de estados internos da FSM.
- T é a função de transição de estado que representa um conjunto de mapeamentos parciais: T: (X, Q) -> Q.
- S é a função de saída que representa um conjunto de mapeamentos parciais: S: (X, Q) -> Z (Máquina de Mealy) ou S: Q -> Z (Máquina de Moore).
- E é o estado inicial.

FSMs são amplamente utilizadas em engenharia para projeto de controladores. As mesmas podem descrever o comportamento de um sistema de automação, onde o sistema emite sinais de controle dependendo das entradas e do estado atual do sistema.

Do ponto de vista de implementação do circuito, uma FSM pode ser representada como um bloco combinacional mais um bloco de memória para armazenar os estados presentes (vide figura 2.34).

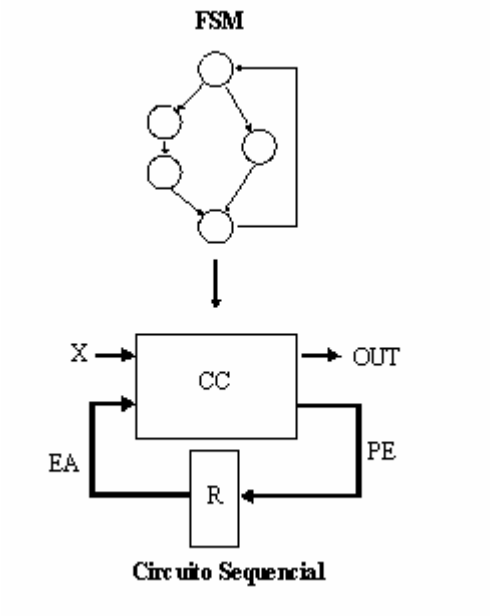


Figura 2.34 - Exemplos de máquinas de estados

FSMs pode ser representada em VHDL através de processos. Na figura 2.35 é apresentado um exemplo de implementação de máquina de Moore em código VHDL, no desenvolvimento do SimRP, foi adotado o modelo de Máquina de Mooore na geração de código VHDL, a figura

2.35 (a) apresenta a declaração da entidade, a figura 2.35 (b) apresenta o modelamento da entidade e a figura 2.35 (c), a definição do processo.

```
LIBRARY ieee;
ENTITY moore3 IS
    PORT (Clk, Reset        : IN  STD_LOGIC;
          Input             : IN BIT;
          Output            : OUT STD_LOGIC_VECTOR (1 DOWNTO 0)
         );
END moore3;
```

(a)

```
ARCHITECTURE comportamental OF moore3 IS
    TYPE TIPO_ESTADO IS (estado_A, estado_B, estado_C);
    SIGNAL estado: TIPO_ESTADO;
BEGIN
    PROCESS (Reset, Clk)
    BEGIN
        IF reset = '1' THEN
            estado <= estado_A;
      --ELSIF Clk'EVENT THEN
        ELSIF Clk'EVENT AND clk = '1' THEN
            CASE estado IS
                WHEN estado_A =>
                    IF Input = '0' THEN estado <= estado_B;
                    ELSIF Input = '1' THEN estado <= estado_C;
                END IF;
                WHEN estado_B =>
                    IF Input = '0' THEN estado <= estado_B;
                    ELSIF Input = '1' THEN estado <= estado_C;
               END IF;
                • • •
```

(b)

```
PROCESS (estado, Input)
    BEGIN
        CASE estado IS
            WHEN estado_A =>
                output <= "01";
            WHEN estado_B =>
            • • •
        END CASE;
    END PROCESS;
END comportamental;
```

(c)

Figura 2.35 - Exemplo de implementação de Máquina de estados em VHDL

2.9 CONCLUSÃO

Este capítulo apresentou os conceitos de Sistemas Discretos, RdPs e da Linguagem VHDL. Neste sentido foram abordos alguns aspectos formais de RdPs que são importantes no contexto deste trabalho. Outros aspectos teóricos podem ser estudados na literatura, tais como utilização de gramáticas e regras de reescrita para formalizar RdPs [1].

O critério de escolha das taxonomias a serem desenvolvidas no SimRP foi a grande utilização destas em outros simulados, e no caso específico da RdPs Interpretadas é a sua semelhança com o comportamento das Máquinas de Estados Finitos. Para resumir os pontos importantes estudados neste capítulo são feitos os seguintes comentários:

a) Para os objetivos de gerar descrições de controladores automaticamente a RdPs Interpretadas se assemelham bem a modelos de Máquinas de Estados Finitos (*FSMs – Finite State Machines*).

b) As taxonomias temporais e temporizadas possibilitam atribuir variável de tempo, permitindo assim realizar simulações mais precisas.

c) A linguagem VHDL possui uma definição bem semelhante a das linguagens de programação estruturadas, permitindo assim implementar sistemas digitais complexos de forma mais simples e rápida. FSMs podem ser descritas em VHDL facilmente. Descrições nesta linguagem de RdPs Interpretadas podem ser geradas, de maneira automática, mapeando-as como FSMs (vide seção 2.5.1).

d) A possibilidade de acrescentar às Redes de Petri Interpretadas cláusulas temporais permite enriquecer a capacidade de representação deste modelo, esta funcionalidade não pertence à versão 1.0 do SimRP, mas está sendo colocado como proposta de trabalhos futuros. Mais informações sobre esta funcionalidade poderá ser obtida no artigo referente à bibliografia catalogada na posição [24].

3. SIMULADORES DE REDES DE PETRI PESQUISADOS

3.1 INTRODUÇÃO

Neste capítulo são analisadas algumas ferramentas de simulação de redes de Petri. Neste contexto é importante verificar o estado da arte na área de simulação de RdPs que suportem os modelos escolhidos (conforme discutido no Capítulo 2). Isto é, as RdPs Interpretadas, Temporais e Temporizadas. Para o desenvolvimento deste trabalho foram pesquisados vários simuladores de Redes de Petri, quatro destas ferramentas se destacaram em relação às outras por suas características e funcionalidade.

Nesta análise foi verificada a coerência com relação à teoria de RdPs (isto é, se os modelos adotados nas ferramentas concordam plenamente com o aspectos teóricos discutidos no capítulo 2), características de interface (interfase amigável/não-amigável), documentação do sistema e manuais de uso.

A seguir serão analisadas de maneira objetiva quatro (4) simuladores de RdPs encontrados na pesquisa correspondente.

3.2 TimeNET

Esta é uma ferramenta para a edição, simulação e verificação de Redes de Petri Temporizadas, desenvolvida em C++, compatível com os sistemas Operacionais Linux SuSE Linux 7.2 Professional, SunOS 5.7-5.9 (Solaris), Sun workstations e Windows XP, O *TimeNet* é uma ferramenta de uso livre; foi desenvolvida em 1991 pela *Technische Universität Berlin* [29].

a) Vantagens:

- A ferramenta possui interface gráfica amigável para edição e simulação, ou seja, a mesma possibilita a edição e simulação das RdPs através de comandos intuitivos e uso de objetos gráficos.
- Verificação de propriedades/conflitos referentes às RdPs tais como rede marcada viva, rede marcada reiniciável, aém de conflitos estruturais, bloqueio fatal (vide capítulo 2, seção 2.5**).**
- Ferramenta de uso gratuito (*software gratuito*).

b) Desvantagens:

- É limitada à simulação das RdPs Temporizadas, não permitindo a simulação de outras taxonomias.
- Para a instalação da ferramenta no Linux ou no Solares são necessários conhecimentos avançados nos respectivos sistemas operacionais e a instalação de dependências relacionadas ao compilador GCC[34].
- Não disponibiliza o código fonte.
- Não foi encontrado manual de uso da ferramenta e nem a documentação do sistema.

A figura 3.1 apresenta a interface gráfica da ferramenta de edição e simulação de RdPs Temporizadas *TimeNet*.

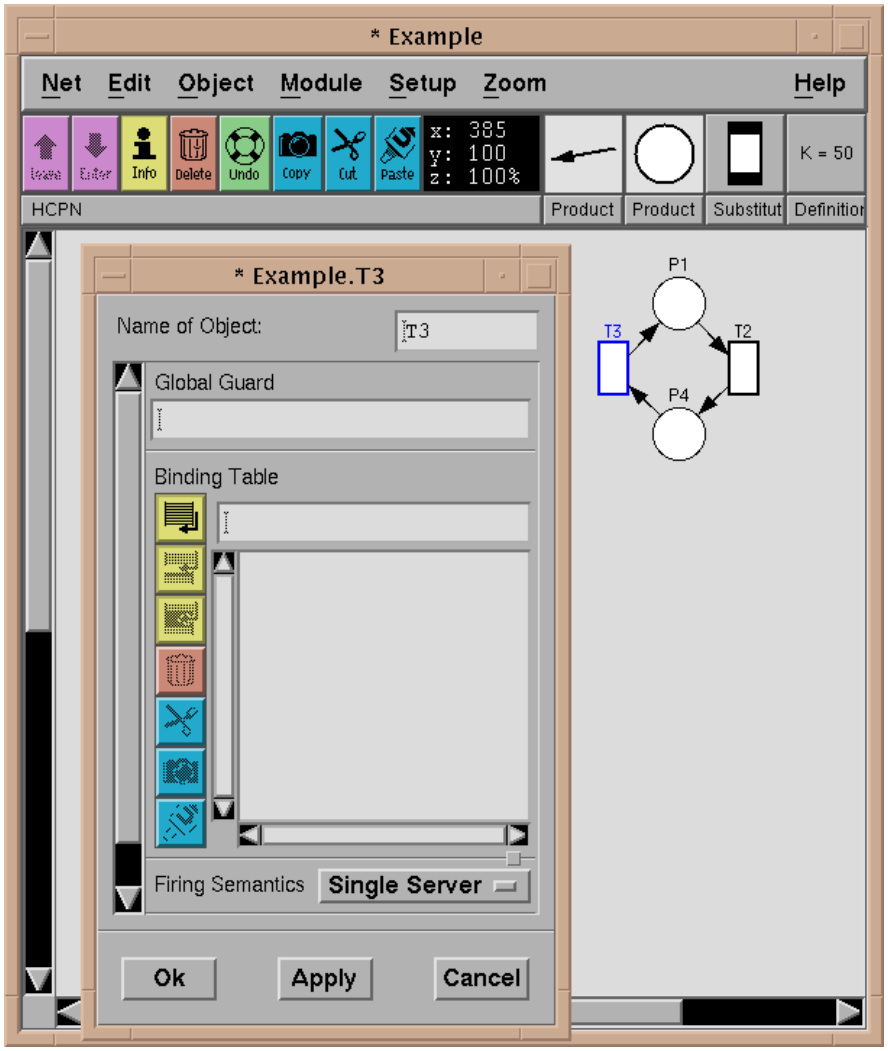


Figura 3. 1 - Interface gráfica do TimeNet

3.3 ARP

Esta é uma ferramenta para a edição, simulação e verificação de RdPs Ordinárias e temporizadas, desenvolvida na linguagem de programação Pascal é compatível com o sistema operacional Windows com MSDOS. O ARP é uma ferramenta de uso livre (*software gratuito*), que foi desenvolvida no ano 1995 pelo laboratório de Engenharia Elétrica da Universidade de Santa Catarina [30].

.

a) Vantagens:

- Verificação *deadlock* (vide capítulo 2, seção 2.6.1).
- Verificação de propriedades referentes à taxonomia das redes temporizadas e ordinárias como a propriedade de K-limitada, vivacidade (vide capítulo 2 seção 2.6).
- Compatibilidade com a teoria de RdPs, descrita no capítulo 2.
- Instalação simplificada.
- Manual uso acessível na internet.
- Ferramenta de uso gratuito (*software gratuito*, capítulo 1, vide seção 1.1).

b) Desvantagens:

- É limitada à taxonomias ordinária e temporal.
- Não disponibiliza o código fonte.
- Interface pouco amigável. Exige que o usurário faça seu modelamento através de arquivo texto; para este modelamento é necessário que o usuário conheça previamente a sintaxe da linguagem de entrada.
- Não possui portabilidade para outros sistemas operacionais.

A figura 3.2 apresenta a interface gráfica da ferramenta de edição e simulação de Redes de Petri Temporizadas ARP2

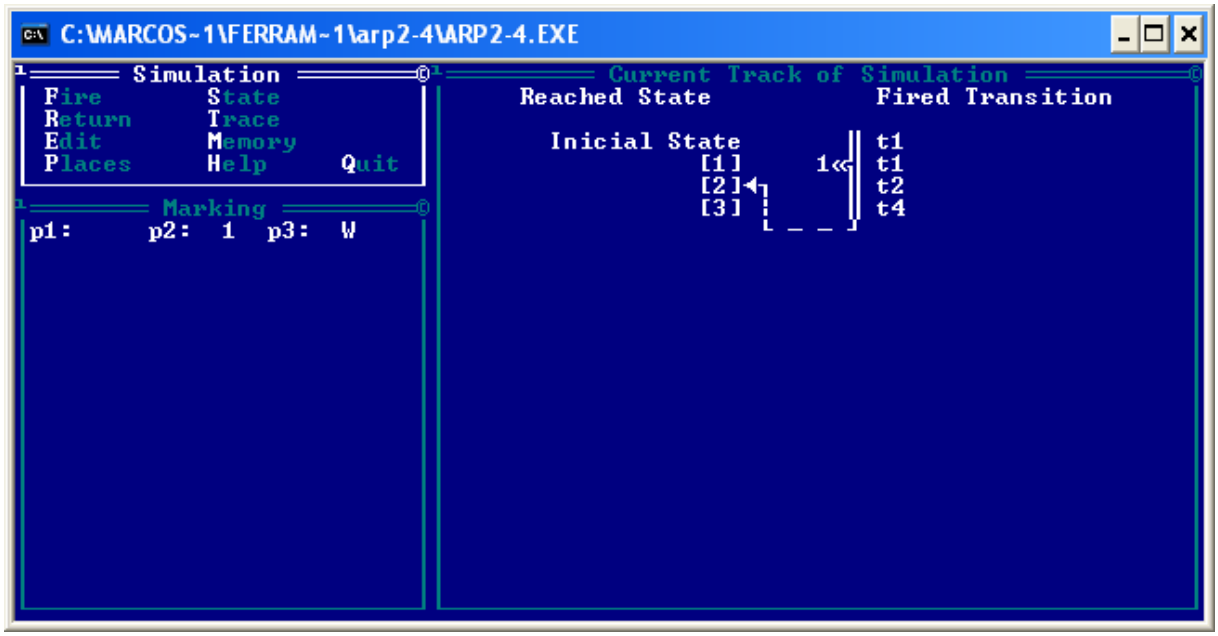


Figura 3. 2 - Interface do ARP2

3.4 PetriSim

Esta é uma ferramenta para a edição, simulação e verificação de Redes de Petri Ordinárias e Temporizadas, desenvolvida na linguagem de programação Pascal é compatível com o sistema operacional Windows com MSDOS. O PetriSim é uma ferramenta de uso livre (*software gratuito*), foi desenvolvida no ano 2004 pela laboratório *University of Malta* [31]

a) Vantagem:

- Verificação *deadlock*.
- Verificação de propriedades referentes à taxonomia das redes temporizadas e ordinárias como a propriedade de K-limitada e vivacidade**.**
- Plenamente compatível com a teoria de RdPs discutida no capítulo 2.
- Instalação simplificada.

- Manual uso acessível na internet.
- Ferramenta de uso gratuito.

b) Desvantagem:

- É limitada a duas taxonomias ordinárias e temporizadas.
- Não disponibiliza o código fonte.
- Interface pouco amigável, o usuário necessita de conhecer os comandos e as funcionalidades para operar a ferramenta (**comandos textuais e gráficos).**
- Não possui portabilidade para outros sistemas operacionais.

A figura 3.3 apresenta a interface gráfica da ferramenta de edição e simulação de Redes de Petri Temporizadas PetriSim.

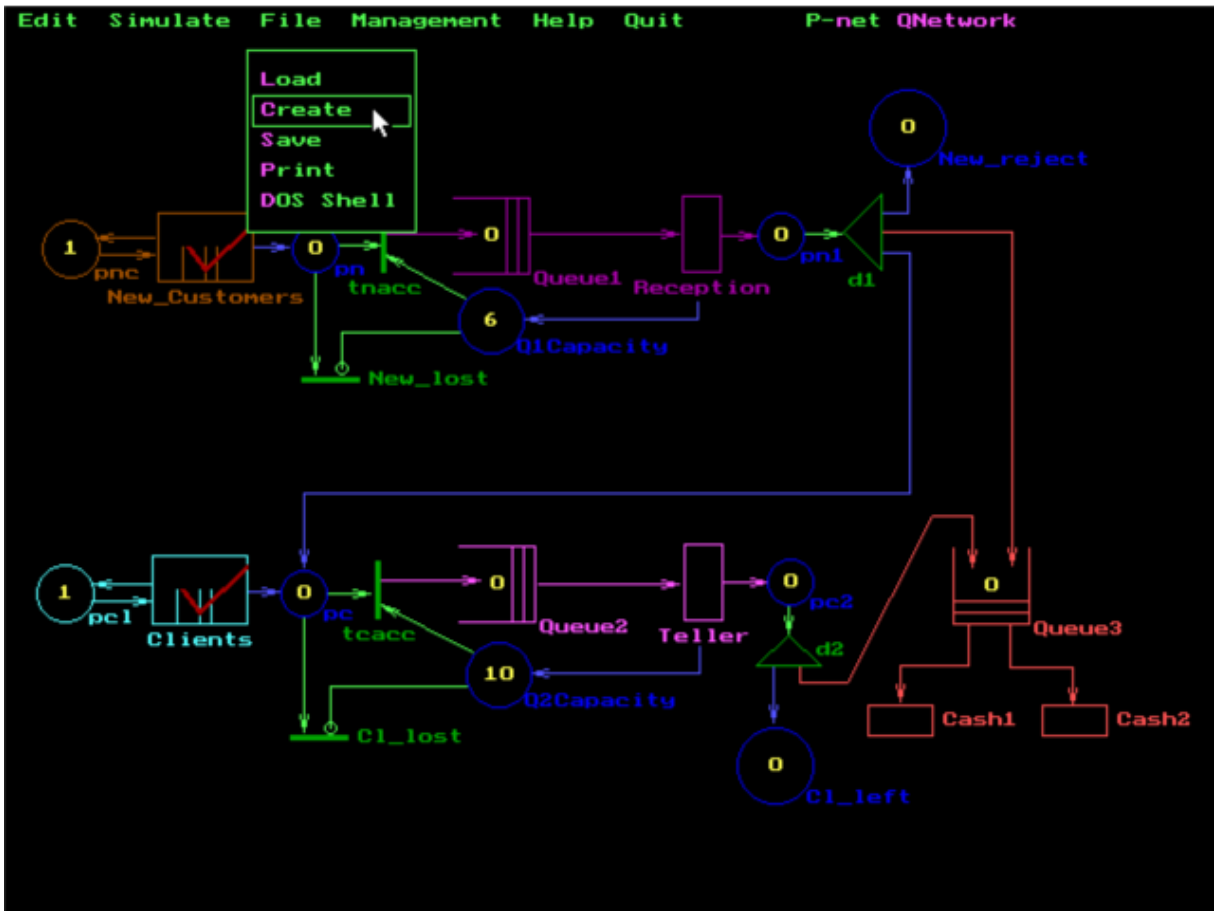


Figura 3. 3 - Interface gráfica do PetriSim

3.5 Visual Objekt Net ++

Esta é uma ferramenta para edição e simulação de eventos contínuos e discretos utilizando-se Redes de Petri Ordinárias e Temporizadas, desenvolvida na linguagem de Object C++ é compatível com o sistema operacional Windows 98 ou superior, O Virtual Objekt Net ++ é uma ferramenta de uso livre; foi desenvolvida pela *Ilmenau University of Technology* [32].

a) Vantagens:

- Verificação *deadlock* e conflitos.
- A ferramenta possui interface gráfica amigável para edição e simulação. Neste caso a mesma possibilita a edição e simulação das RdPs através de comandos intuitivos e uso objetos gráficos.
- Verificação de propriedades referentes à taxonomia das redes temporizadas e ordinárias.
- Instalação simplificada.
- Ferramenta de uso gratuito (*software gratuito*).

b) Desvantagem:

- É limitada a duas taxonomias: temporizadas e ordinárias.
- Não disponibiliza o código fonte.
- Utiliza conceitos não presentes na teoria de Redes de Petri, como por exemplo, a simulação de RdPs Ordinárias sem a definição de uma seqüência de disparos, quando um conflito estrutural é encontrado, a mesma escolhe de forma aleatória o caminho a percorrer.
- Não possui portabilidade com outros sistemas operacionais.

A figura 3.4 apresenta a interface gráfica da ferramenta de edição e simulação de Redes de Petri Visual *Objekt Net*++.

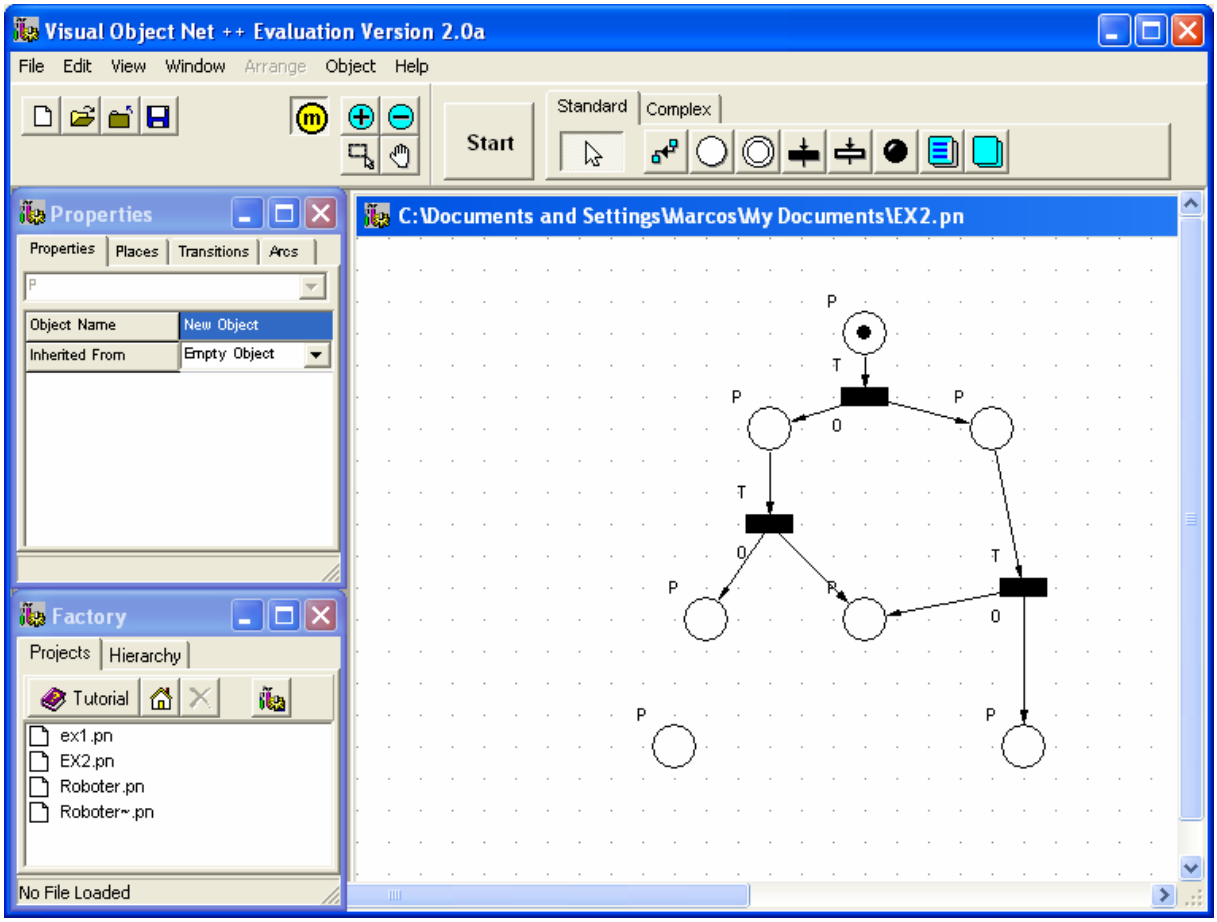


Figura 3. 4 - Interface do Objekt Net ++

3.6 CONCLUSÃO

A análise de outros aplicativos que editam e simulam o comportamento de eventos discretos utilizando recursos da teoria de Redes de Petri teve como objetivo verificar seus funcionamentos e características, para servir de base no desenvolvimento do SimRP.

Na pesquisa foi observado um grande número de simuladores de Redes de Petri disponíveis na WEB, mas na sua grande maioria tratavam-se de protótipos não robustos, sem documentação e referência.

Os quatro simuladores descritos neste capítulo foram o que apresentaram maior qualidade entre os simuladores pesquisados, todos apresentaram uma limitação quanto ao uso de taxonomias e limitações de funcionalidade. Nenhum dos simuladores estudados simula RdPs Interpretadas e Temporais.

Não foi observado em nenhum simulador as funcionalidades propostas pelo SimRP (**simular RdPs ordinárias, temporais e temporizadas de maneira flexível**). Adicionalmente, não foi encontrado um simulador de boa qualidade com licenciamento GPL que motivasse a continuidade do seu desenvolvimento.

Como pode ser visto, as ferramentas estudadas foram desenvolvidas usando diferentes tipos de linguagens de programação. A escolha da linguagem de programação PHP para o desenvolvimento do Simulador SimRP foi feita pelos seguintes motivos:

a) Esta linguagem é de software livre (com licença GPL).

b) É uma linguagem orientada para implementações na arquitetura de três (3) camadas (cliente, servidor e banco de dados).

c) Possui *plugins* de acesso direto a banco de dados como MySQL [23], PostgreSQL [35], Oracle [36].

d) É uma linguagem interpretada de maior compatibilidade com os navegadores.

e) Embora seja uma linguagem orientada para aplicações *web*, a sua sintaxe é bem semelhante à linguagem C.

f) Não necessita de software proprietários adicionais para seu funcionamento. Neste sentido a linguagem PHP tem vantagem sobre a linguagem JAVA [37] dado que esta última precisa de uma máquina virtual JAVA para seu funcionamento. Neste caso, a máquina virtual JAVA não é software livre (sem licença GPL).

g) Esta linguagem garante total portabilidade para diferentes sistemas operacionais, dado que utiliza tecnologia WEB, necessitando apenas de um navegador WEB *(browser)* para seu funcionamento.

h) Facilita a utilização por parte dos usuários, já que os mesmos não necessitam da instalação do software nas máquinas.

i) Facilita a atualização do programa (versão do software), pois somente é necessário aplicar as atualizações no servidor.

4. ANÁLISE, PROJETO E IMPLEMENTAÇÃO DO SIMULADOR

4.1 Introdução

Neste capítulo é descrito o ciclo de projeto do simulador SimRP. Como descrito no capítulo 3, a linguagem escolhida foi o PHP, seguindo os conceitos da teoria de Orientação a Objetos. Vantagem da orientação a objetos sobre o Projeto Estruturado de Sistemas é que tudo o ciclo do projeto (análise, projeto e implementação) usa os conceitos fundamentais da orientação a objetos tais como encapsulamento, herança, comunicação por meio de mensagens (usando os métodos das classes), entre outros.

Além da linguagem PHP, foi utilizada a linguagem Java Script [38], sendo que a implementação total do sistema usa a arquitetura de três camadas: cliente, servidor, banco de dados (como demonstrado na figura 4.1).

A metodologia utilizada para o projeto e documentação do sistema foi o RUP (*Rational Unified Process*), desenvolvida pela Rational [39], esta metodologia vem se tornando a mais utilizada para descrever e documentar sistemas, devido sua compatibilidade com a UML.

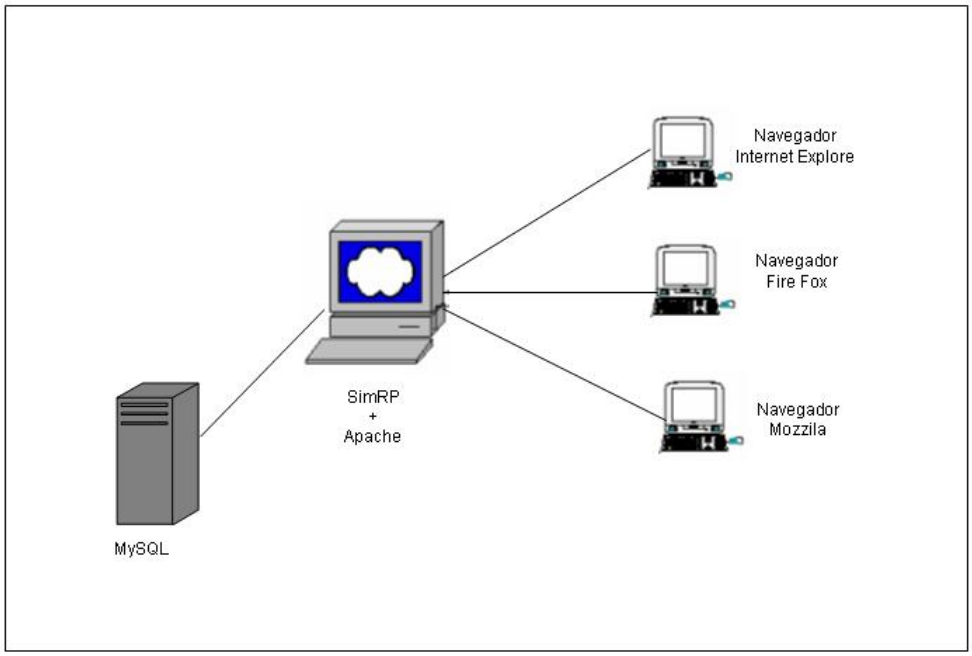


Figura 4.1 - Arquitetura do Sistema

O módulo servidor é o responsável por todo o processamento da aplicação; para o seu funcionamento é necessário que seja instalado e habilitado o serviço de publicação de página Apache[22], com o módulo de interpretação do PHP. Também se faz necessária a instalação do banco de dados MySQL [23], todos os aplicativos supracitados são compatíveis com os sistemas operacionais Linux e Windows e são registrados com GPL.

O servidor é responsável por publicar uma página para a entrada de dados no simulador (interface com o usuário), que implementará a tarefa de entrada de dados. Neste caso, a interface somente é responsável por tratar a entrada de dados e retornar o resultado já processado. Por outro lado, todo o processamento da aplicação (simulação e verificação de propriedades) é realizado no servidor.

No cliente (ou usuário do sistema) é necessária a instalação de um navegador WEB. O sistema será acionado através da URL (apontando para o servidor do SimRP), que deverá ser definida no momento da instalação do servidor. Para que este seja mais intuitivo o acesso ao servidor é interessante utilizar um servidor DNS.

4.2 Análise do Projeto

O simulador está projetado para simular as Redes de Petri Ordinárias, Interpretadas, Temporizadas e Temporais e gerar códigos VDHL a partir de descrições de RdPs Interpretadas. Também é possível verificar propriedades pertinentes ao modelamento de uma Rede de Petri como Vivacidade, Redes Iniciável e Rede não Pura (vide capítulo 2 seção 2.5), além de descobrir conflitos estruturais e *deadlocs* (vide capítulo 2 seção 2.4.4).

4.2.1 Limitações do Sistema

O simulador não foi projetado para simular o comportamento de Redes de Petri Estocásticas e Nebulosas, Coloridas e de Auto-Nível, conforme definições dadas no Capítulo 2 (vide seção 2.7).

O simulador somente gerará código VHDL das Redes de Petri modeladas como Interpretadas, devido a sua semelhança às máquinas de estados finitos (FSMs, vide capítulo 2 seções 2.8.8). O tratamento para as demais taxonomias tratadas neste trabalho é deixado como sugestão para trabalhos futuros.

O desenvolvimento do simulador será dividido em módulos, possibilitando a inclusão de módulos adicionais como descrito acima.

4.2.2 Principais Funcionalidades do Sistema

O objetivo do sistema é simular o comportamento de RdPs para os tipos propostos anteriormente e gerar código VHDL do comportamento de uma Rede de Petri Interpretada. Para que este objetivo fosse alcançado, foram implementadas as seguintes funcionalidades:

a) **Inserir Rede de Petri**: esta funcionalidade é responsável por definir o modelamento de uma Rede de Petri no sistema, descrevendo seu comportamento e atribuindo uma taxonomia quando for o caso.
b) **Editar Taxonomia:** esta funcionalidade possibilita que seja atribuída uma taxonomia a uma Rede de Petri Ordinária, caso não tenha sido definido no momento do modelamento na funcionalidade Inserir Rede de Petri.
c) **Visualizar Rede de Petri:** esta funcionalidade permite a visualização do modelamento da rede, e com isto, poder analisar se a mesma não foi modelada de modo equivocado.
d) **Editar Rede de Petri:** esta funcionalidade oportuniza que uma Rede de Petri, já modelada no sistema possa ser alterada ou apagada.
e) **Definir Seqüência de Disparos:** esta funcionalidade permite que seja definida uma seqüência de disparo para uma Rede de Petri Ordinária.
f) **Verificar Propriedades:** esta funcionalidade possibilita verificar se a Rede de Petri é *iniciável*, *viva*, *não-pura* ou se existe a presença de conflitos estruturais e *deadlocks*.
g) **Gerar Código VHDL:** esta funcionalidade permite a geração de um código VHDL através do modelamento de uma Rede de Petri Interpretada.
h) **Simular Rede de Petri:** esta funcionalidade possibilita acompanhar a simulação de uma RdP modelada no sistema de acordo com sua taxonomia. Nesta funcionalidade também é possível verificar a presença de *deadlock*.

Nos tópicos seguintes serão detalhadas as funcionalidades do sistema além do seu comportamento e estrutura.

4.2.3 Diagramas de Caso de Uso

A figura 4.2 apresenta o diagrama de caso principal do sistema. O diagrama descreve seu comportamento e funcionalidades.

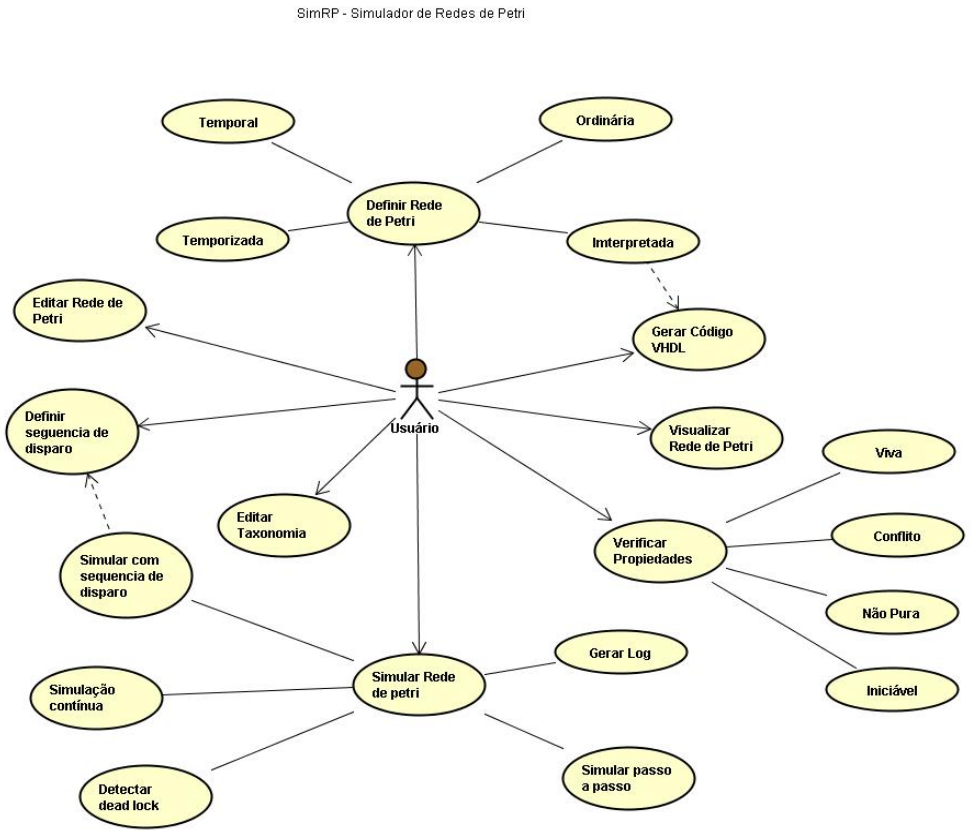


Figura 4.2 - Diagrama de caso de uso do SimRP

O apêndice 1 apresenta a análise do sistema e os seus respectivos diagramas. No apêndice 2 encontram-se as tabelas com os detalhamentos dos casos de uso. No apêndice 4 é descrito o *script* de banco de dados e no apêndice 5 é apresentado o manual de utilização do sistema.

4.3 Projeto do sistema

Nesta seção será apresentado o projeto do SimRP e os principais diagramas de *Classe* e de Seqüência.

4.3.1 Diagrama de classe

Na figura 4.3 pode ser observado o modelamento do SimRP com suas respectivas classes. Na figura 4.4, podem ser visualizados os atributos referentes a cada classe de dados. Estas classes foram definidas para armazenar informação (dados pertinentes ao modelamento de uma RdP).

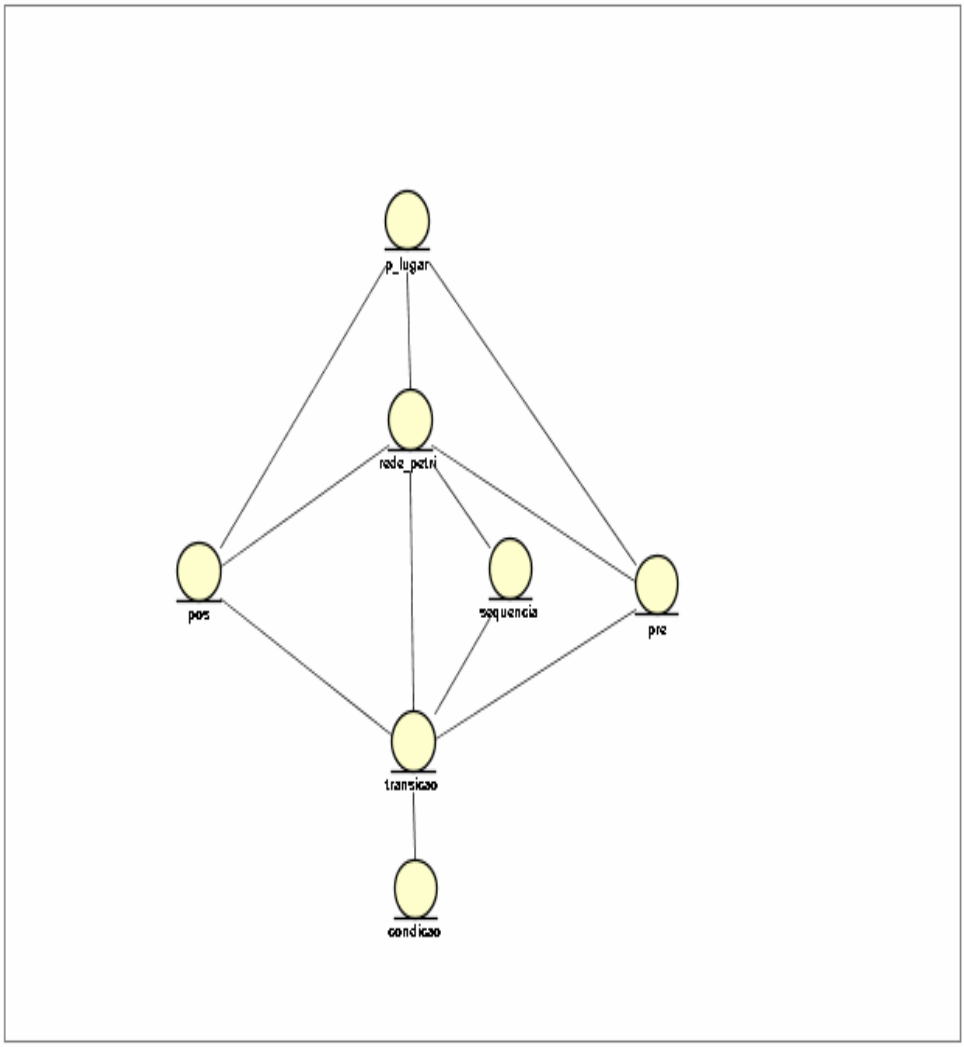


Figura 4.3 - Diagrama de classe de dados

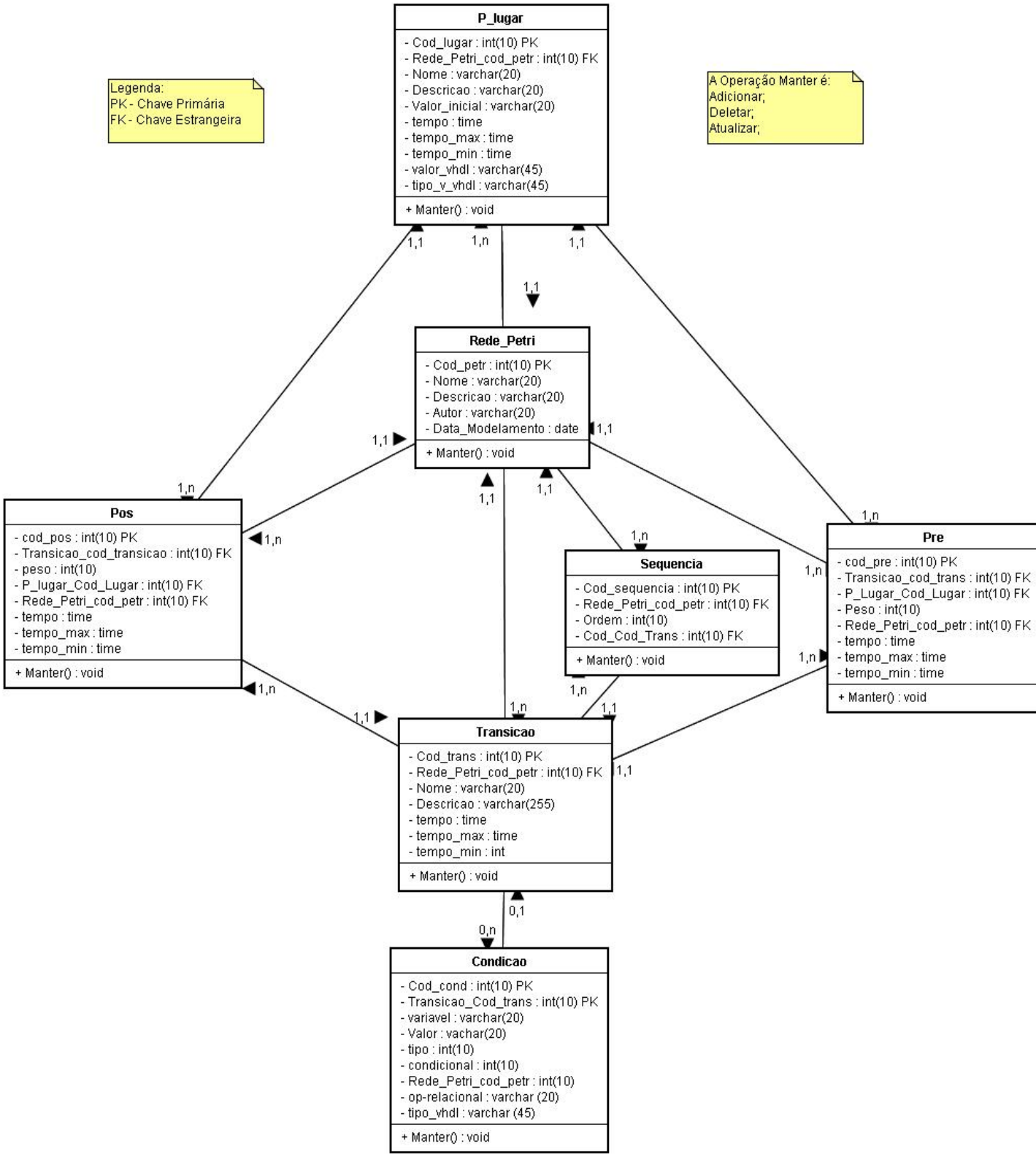


Figura 4.4 - Diagrama físico do Bando de Dados

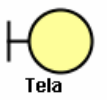
No diagrama apresentado na figura 4.4, também podem ser observados os tipos de dados, relacionamentos, chaves primárias e chaves estrangeiras.

4.3.2 Diagramas de seqüências

Os diagramas de seqüência apresentados nas figuras 4.5 a 4.11, descrevem o funcionamento dos casos de uso apresentados na figura 4.2. O detalhamento destes casos de uso pode ser observado no apêndice 2. Estes diagramas incluem *classes de dados*, *classes de tela* (classes definidas para receber e apresentar informações para o usuário), e *classes de controle* (classes definidas para tratar as informações e execução de tarefas de simulação e verificação de propriedades de RdPs).

Para melhor entendimento do diagrama, os seguintes símbolos padrão do UML são usados para representar os três diferentes tipos de classes:

a) Classe de Tela:



b) Classe de Controle:

c) Classe de Dados:

a) **Definir Rede de Petri**: neste caso a tarefa consiste em definir a topologia da RdP (lugares, transições, pré-condições, pós-condições e marcação inicial). A figura 4.5 mostra o diagrama de seqüência respectivo.

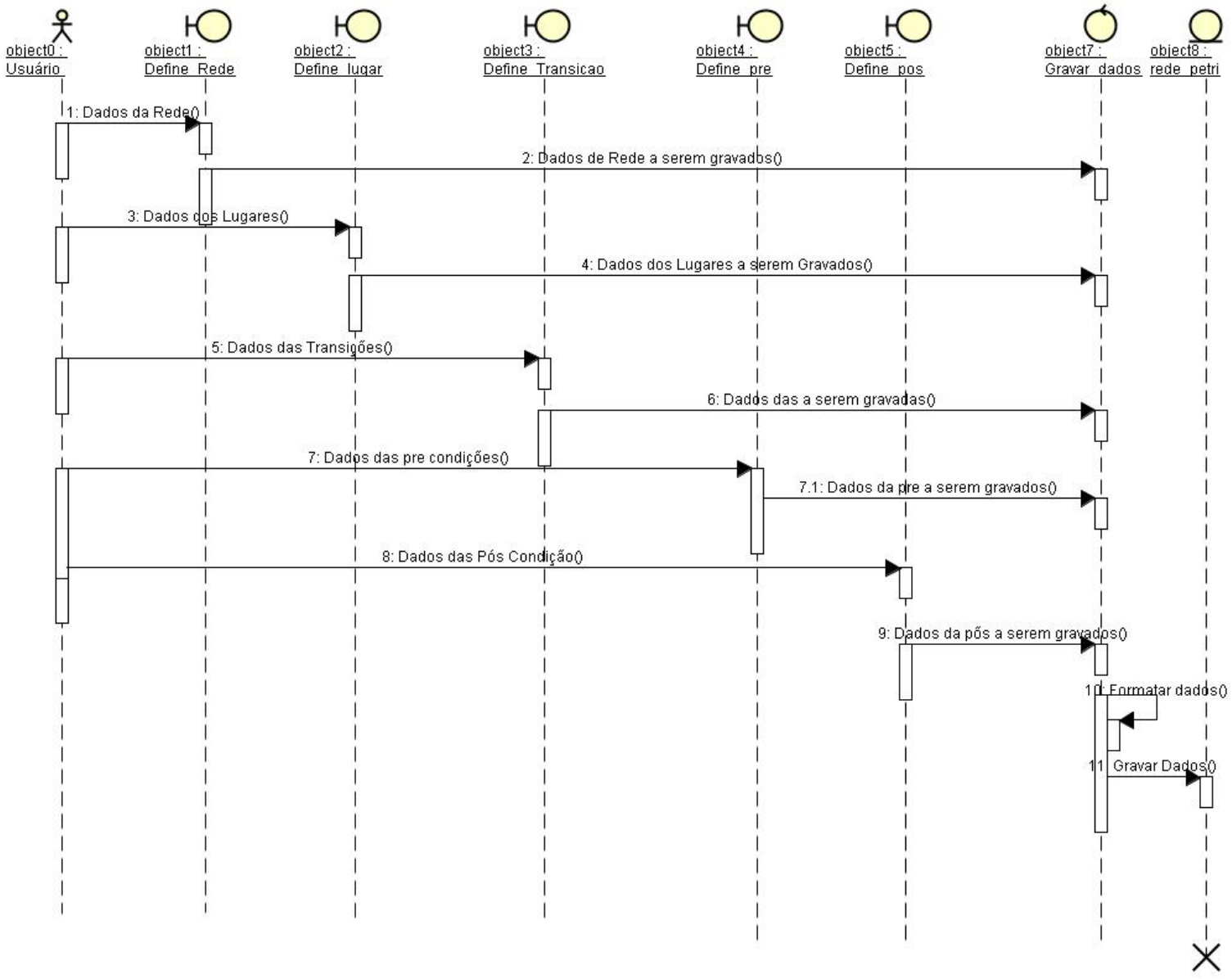


Figura 4.5 - Diagrama de Seqüência: Definir Rede de Petri

b) **Taxonomia**: neste caso são definidos os atributos referentes às RdPs Interpretadas, Temporais e Temporizadas (condições, e características de tempo). A figura 4.6 mostra o diagrama de seqüência respectivo.

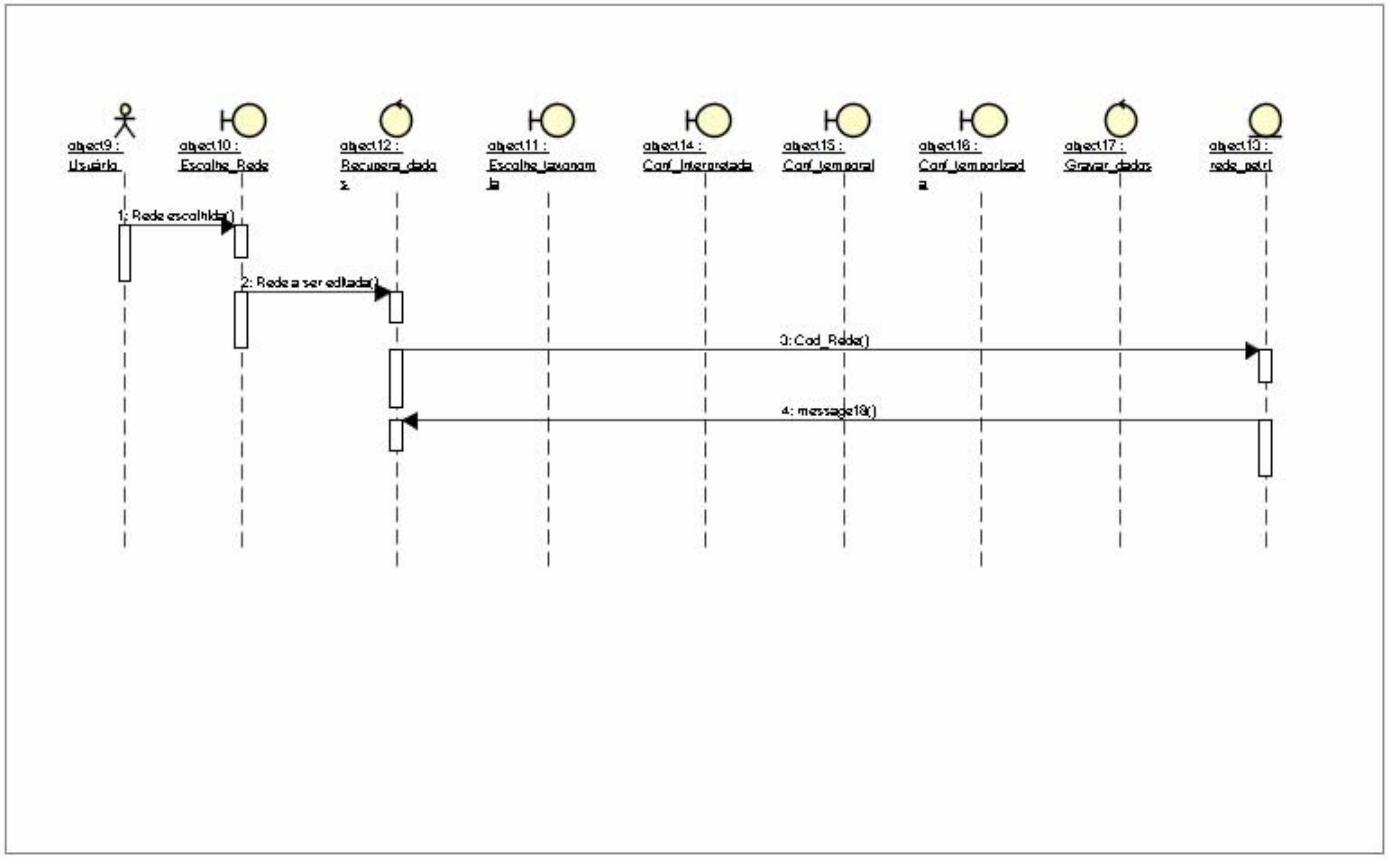


Figura 4.6 - Diagrama de Seqüência: Taxonomia

c) **Visualizar**: neste caso são mostrados para o usuário os dados pertinentes à RdP sendo tratada. A figura 4.7 mostra o diagrama de seqüência respectivo.

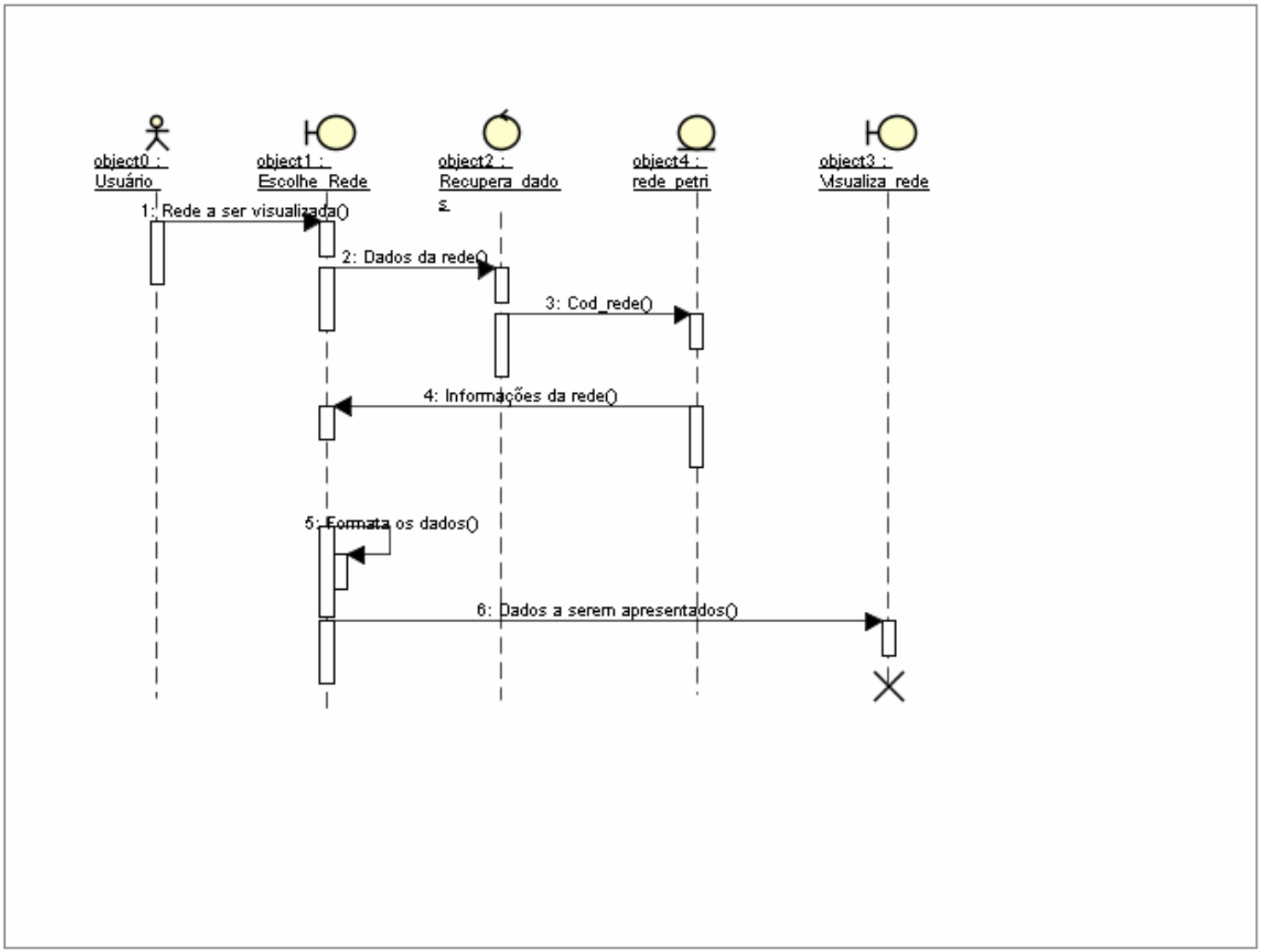


Figura 4.7 – Diagrama de Seqüência: Visualizar

d) **Edita**: tarefas como modificar, editar, apagar são introduzidas no SimRP. A figura 4.8 mostra o diagrama de seqüência respectivo.

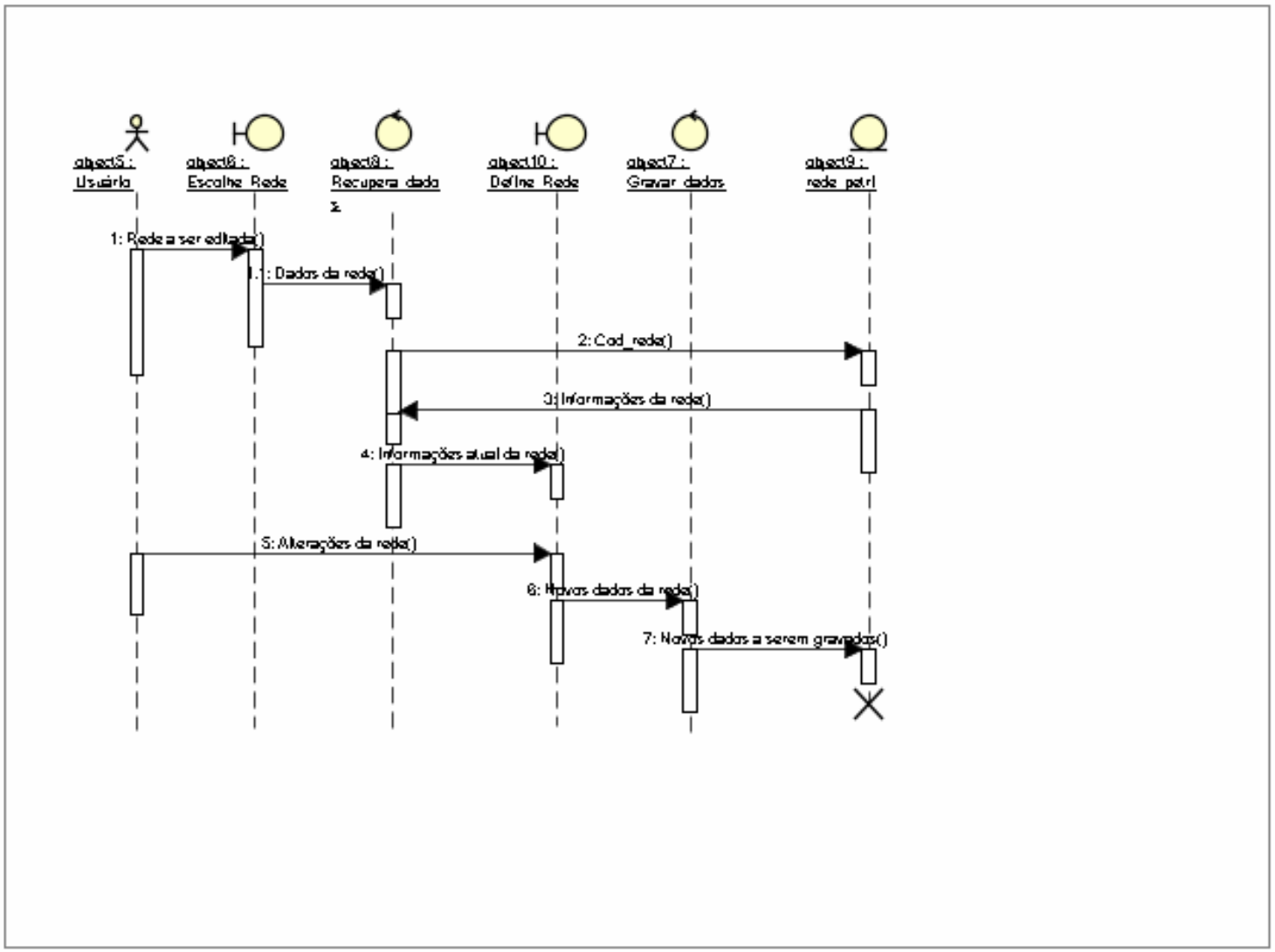


Figura 4.8 - Diagrama de seqüência: Edita

e) **Seqüência de Disparos**: as tarefas de definir uma seqüência de disparo a ser utilizadas na simulação são mostradas no diagrama de seqüência da figura 4.9.

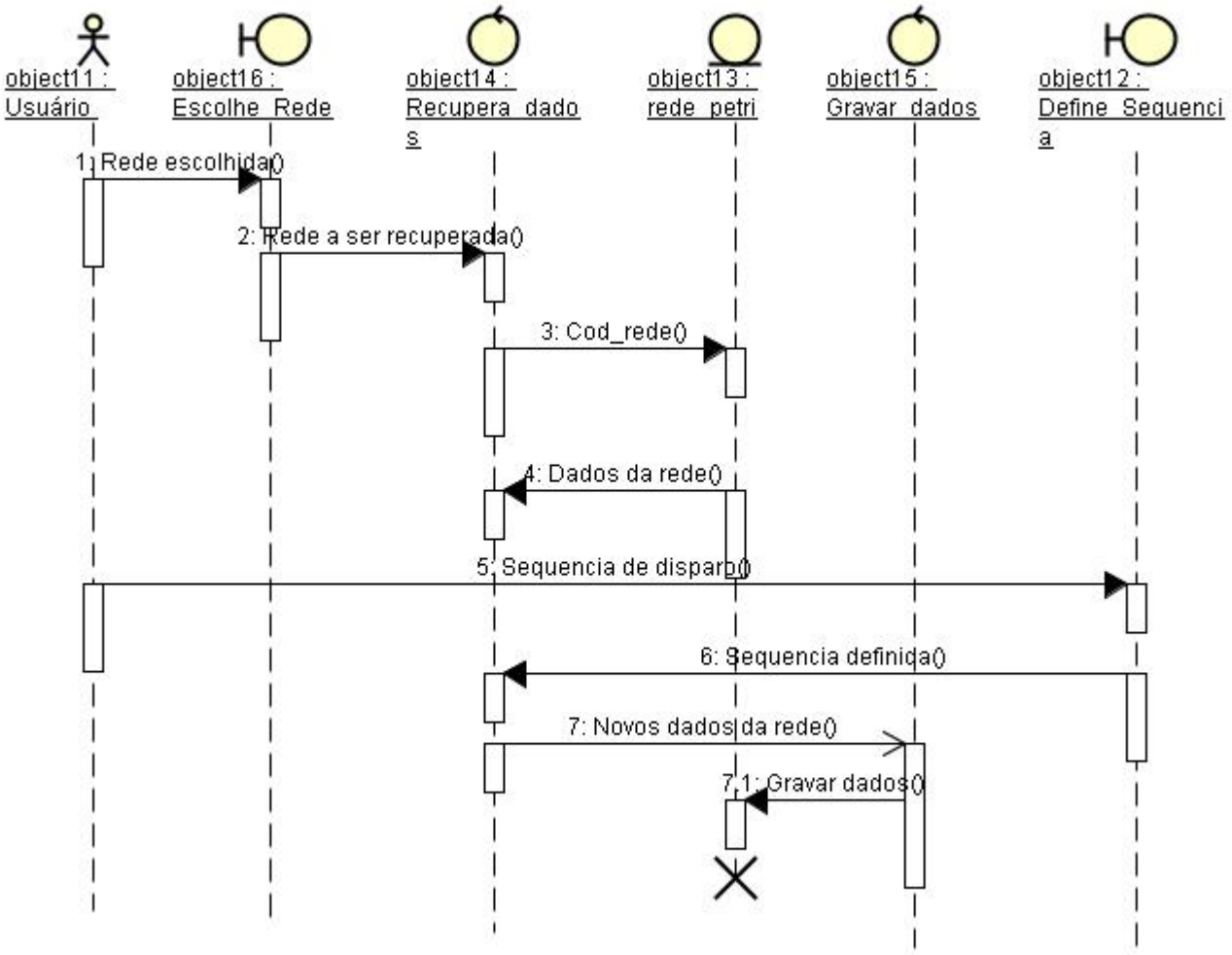


Figura 4.9 - Diagrama de seqüência: Seqüência de disparos

f) **Propriedades**: a verificação de propriedades de uma RdP é mostrada no diagrama de seqüência da figura 4.10.

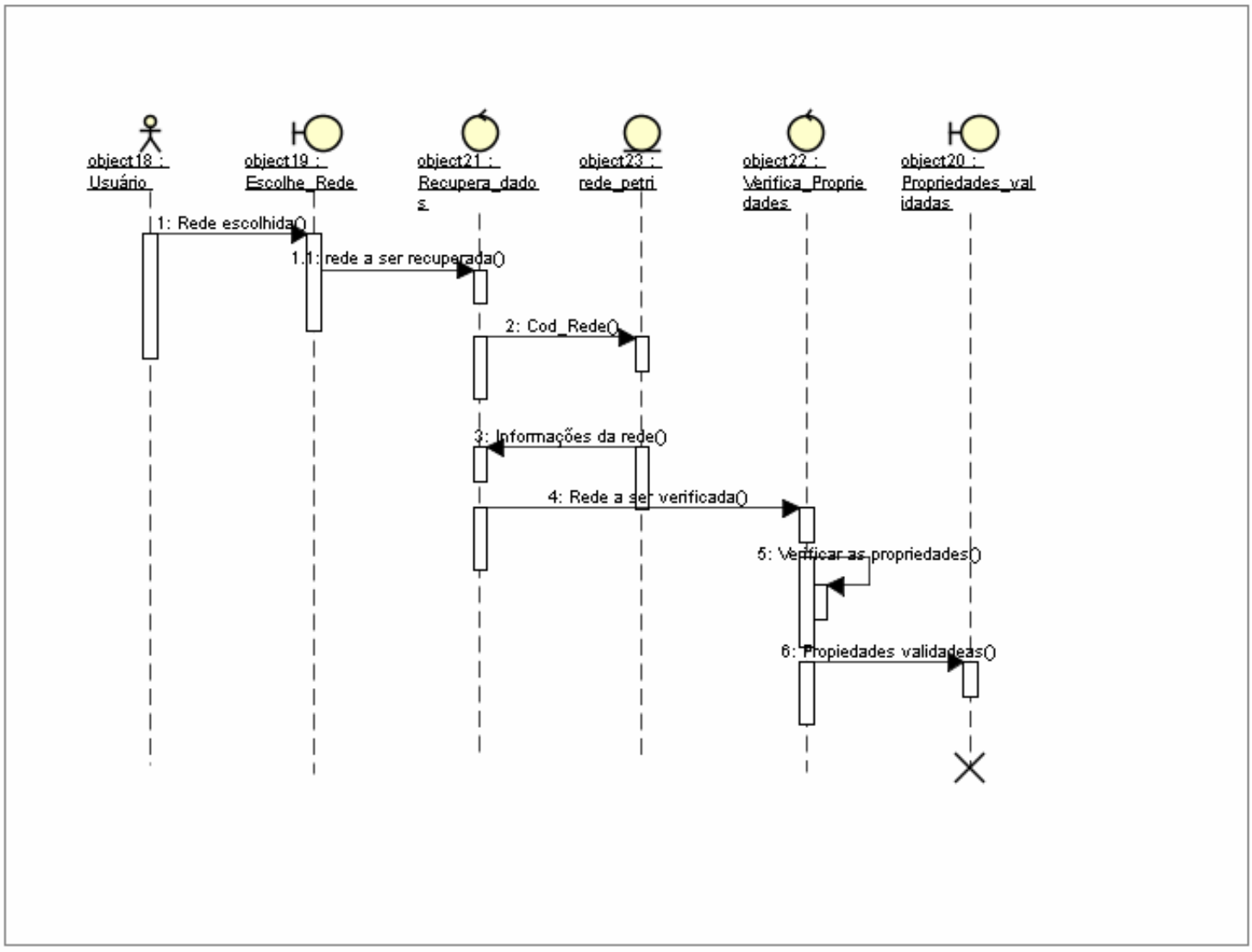


Figura 4.10 - Diagrama de seqüência: Propriedades

g) **Simulação**: A tarefa de simulação da RdP é mostrada na figura 4.11. Este diagrama inclui a geração de código VHDL para as RdPs Interpretadas.

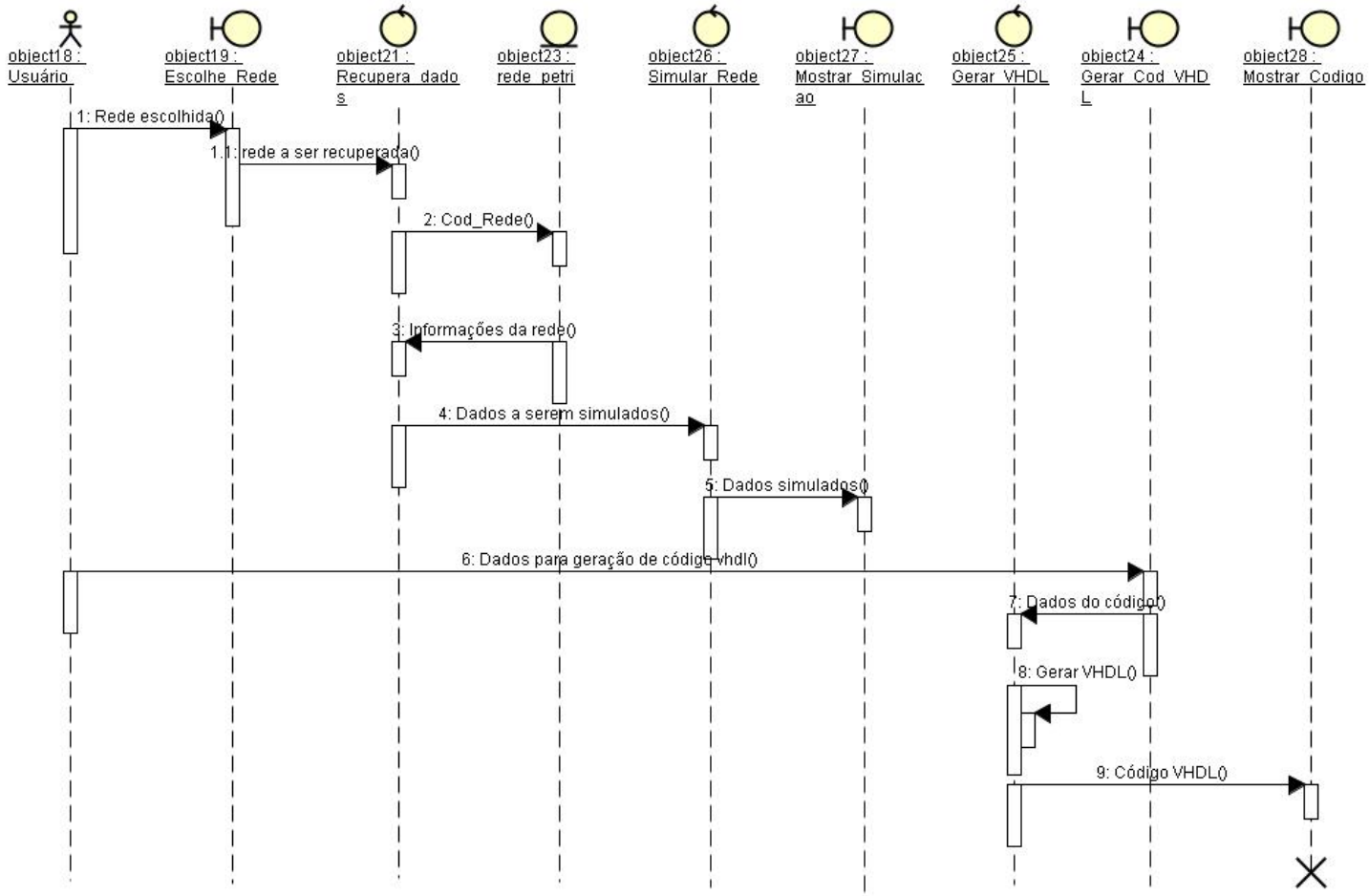


Figura 4.11 – Diagrama de seqüência: Simulação

4.4 Projeto de Implementação do SimRP

Nesta seção será detalhada a implementação do SimRP.

4.4.1 Algoritmo base para a simulação da RdPs

Para a implementação do SimRP foi necessária a implementação do algoritmo descrito na figura 4.5, este algoritmo serviu de base para a implementação da simulação de todas as RdPs abordadas neste trabalho.

O desenvolvimento deste algoritmo utilizou os conceitos apresentados no capítulo 2 (seção 2.5.6). O conceito de árvore de alcançabilidade foi usado devido à necessidade de simular passo a passo cada marcação, mantendo o controle da seqüência de disparo executada para alcançar uma nova marcação. A própria característica da teoria de árvore de alcançabilidade permitem determinar quais e quantas vezes cada transição foi disparada.

A figura 4.12 detalha este algoritmo. Neste caso a árvore de alcançabilidade é implementada parcialmente tendo em conta as pré e pós-condições.

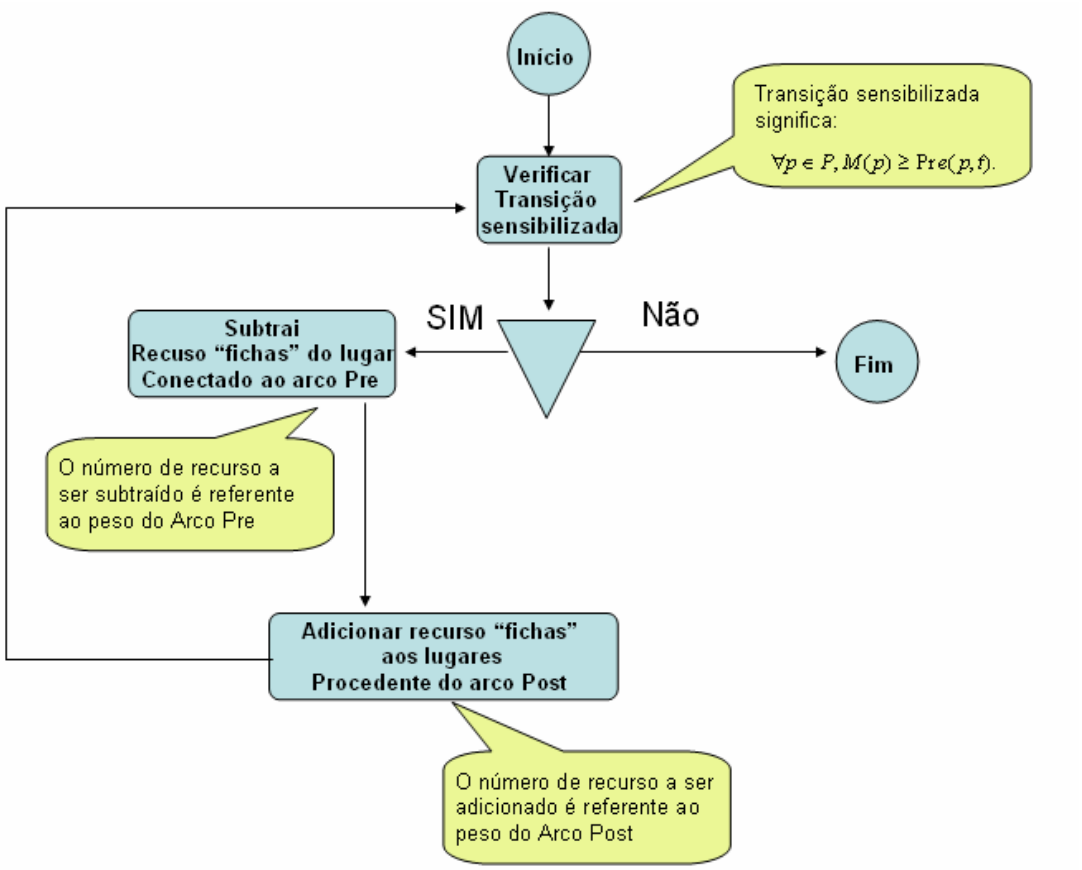


Figura 4.12 - Algoritmo base

O SimRP utiliza-se do banco de dados para armazenar todas as características do modelos RdPs utilizados no SimRP. Isto diminui a performance do simulador, contudo permite o armazenamento dos atributos do modelo de uma maneira flexível (característica importante para simular diferentes tipos de RdPs).

Utilizando o conceito de Banco de Dados Relacionais [40], foi possível implementar a flexibilidade, no contexto de representar diferentes modelos de RdPs. Isto é possível dada a

possibilidade de associar a um modelo geral uma ou mais características referentes as RdPs Interpretadas, Temporais ou Temporizadas.

Por uma questão teórica, o SimRP condiciona o modelamento a uma única taxonomia. Isto é, somente são simulados os modelos de RdPs nas suas definições teóricas, como descritas no capítulo 2.

Para o desenvolvimento da simulação das RdPs Interpretadas foram utilizados os conceitos descritos no capítulo 2 seção 2.6.1 e o algoritmo descrito na figura 4.12, associado a campos que definem as *condições* associadas às transições. Neste caso o número de fichas ou recursos de uma certa RdPs é limitada a, no máximo, uma por lugar.

Para descrever uma *condição*, podem-se utilizar os operadores condicionais *E*, *OU* e *Não*. A utilização do Java Script no desenvolvimento do simulador permite que os valores das *condições* possam ser alterados no momento da simulação.

No caso do desenvolvimento da simulação das RdPs com representação de tempo, foram utilizados os conceitos descritos no capítulo 2 seção 2.6.2. Adicionalmente, foi utilizado o algoritmo descrito na figura 4.12. Neste caso, o Java Script foi utilizado para garantir a atualização da tela a cada segundo (não foi possível definir um tempo menor, pois não seria possível visualizar as alterações na RdPs).

Para a simulação das RdPs ordinárias foi necessária a definição de uma seqüência de disparo, como descrito no capítulo 2 seção 2.4.5.

Para a geração dos códigos VHDL foi utilizado um *template* de um código VHDL para implementação de máquinas de estados. Para fazer um programa na linguagem VHDL, se fez necessária à criação de campos como: tipo de dados de entrada e saída na definição de uma RdPs interpretada.

4.4.2 Verificação de propriedades e conflitos no SimRP

As propriedades verificadas no SimRP são: *rede marcada viva* , *RdP pura*, *RdP reiniciável*. Além do anterior o SimRP detecta *conflitos estruturais* e *deadlocks*.

A verificação das propriedades foi desenvolvida utilizando o algoritmo descrito na figura 4.12, o simulador percorre toda a RdPs, analisando as propriedades descritas no capítulo 2 seção 2.5.

Para a verificação da propriedade *rede marcada viva* (vide capítulo 2 seção 2.5.2) foi utilizado o algoritmo apresentado na figura 4.12, através deste a simulador percorre toda a RdP verificando se todas as marcações são acessíveis a partir de uma marcação inicial. Deve se ter em consideração que a RdP será via para uma seqüência de disparo *S* (para o caso de RdP ordinárias). Para o caso das outras taxonomias suportadas pelo SimRP esta propriedade é verificada pelo comportamento da RdP em tempo de simulação.

A propriedade *RdP Pura* (vide capítulo 2 seção 2.5.4) foi verificada a partir do algoritmo mostrado na figura 4.12 que percorre toda a RdP verificando se nenhum lugar tem a *Pre* = *Pos*.

A propriedade *RdP Reiniciável* (vide capítulo 2 seção 2.5.2) foi verificada a partir do algoritmo, que *percorre* a RdP verificando se uma simulação retorna a sua marcação inicial.

Para a verificação do conflito estrutural (vide capítulo 2 seção 2.4.4), o sistema verifica se algum lugar compartilha o mesmo recurso com duas ou mais transições.

Para a verificação do *deadlock* (vide capítulo 2 seção 2.4.4), o sistema verifica se a partir de uma seqüência de disparos e de uma marcação inicial, conduz a uma situação em que não há qualquer transição sensibilizada.

4.2 CONCLUSÃO

O simulador SimRP foi desenvolvido com o objetivo de ser flexível, sendo possível atribuir características novas à RdP tratada. Por exemplo, uma vez que seja definida uma RdP Interpretada é possível a sua conversão para uma RdP Interpretada pela eliminação dos atributos associados às *condições* das transições. O mesmo pode ser feito com os outros tipos de RdP (temporais e temporizadas). As mudanças podem ser efetivadas de maneira fácil por intermédio da ferramenta de edição da RdP.

A arquitetura de três camadas facilita a utilização da ferramenta, já que não se faz necessária a instalação do simulador nas máquinas clientes para o uso. Esta característica também facilita a manutenção do sistema, além de garantir a interoperabilidade entre os sistemas operacionais (portabilidade do sistema).

O licenciamento GPL irá possibilitar o desenvolvimento contínuo da ferramenta, o compartilhamento do conhecimento, além do uso irrestrito da solução.

5. RESULTADOS OBTIDOS

Neste capítulo são apresentado os resultados obtidos com o uso do SimRP. Maiores informações sobre o funcionamento do SimRP, podem ser obtidas no Manual do Sistema – Anexo.

5.1 Simulação de RdPs Ordinárias

Para exemplificar a simulação de uma RdPs Ordinária, será utilizado o exemplo ilustrado na figura 5.1.

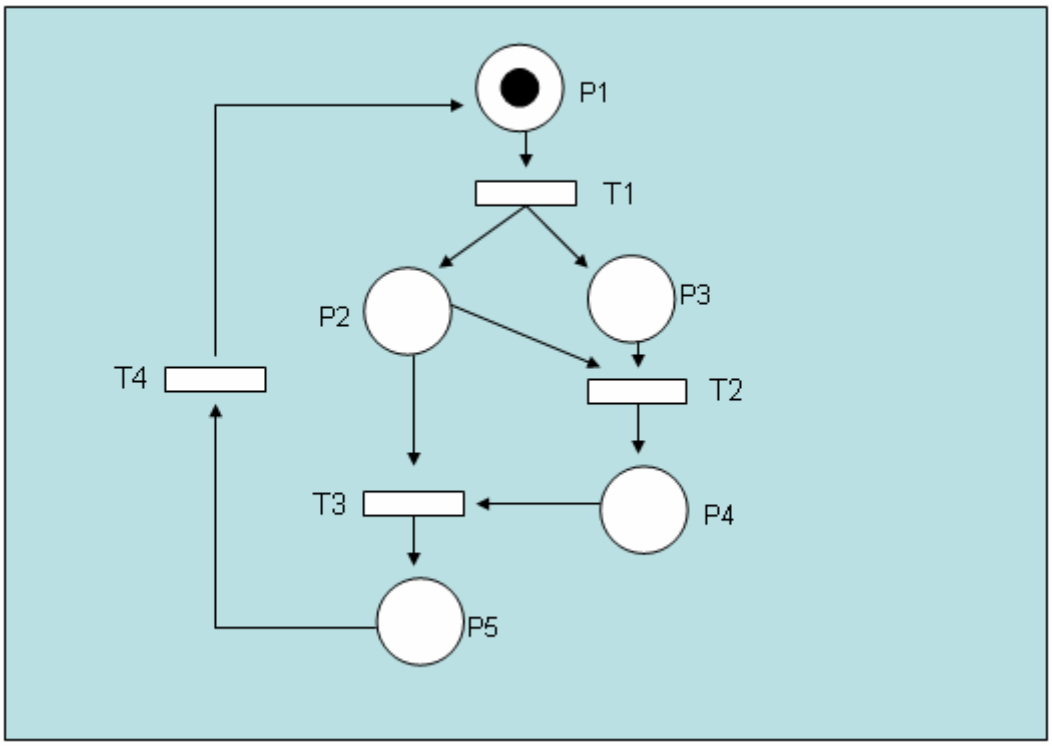
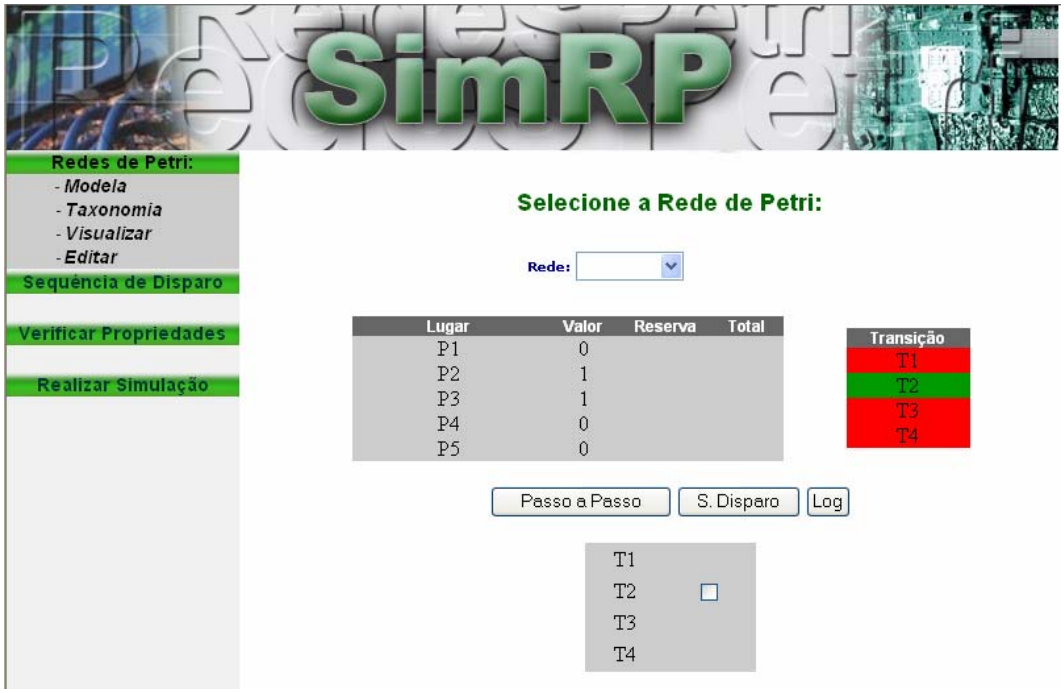


Figura 5. 1 -Modelamento de RdPs Ordinária

Neste exemplo, podemos observar uma rede com cinco lugares: P1, P2, P3, P4, P5 e com quatro transições: T1, T2, T3 e T4, a marcação inicial desta rede é M = { 1 0 0 0 0 }.

Na figura 5.2 podemos observar a simulação desta rede após ter sido modelada no SimRP, para esta simulação foi definida a seguinte seqüência de disparo: S = { T1,T2,T3,T4 }. A figura 5.2 a) apresenta a RdPs na sua marcação inicial e a figura 5.2 b) após o primeiro disparo.

(a)



(b)

Figura 5. 2 - Simulação da RdPs Ordinária

Na figura 5.3 é apresentado o modelamento de uma Rdp complexa que descreve o funcionamento de uma guarita com dois potões eletrônicos e na figura 5.4 a simulação desta Rdp.

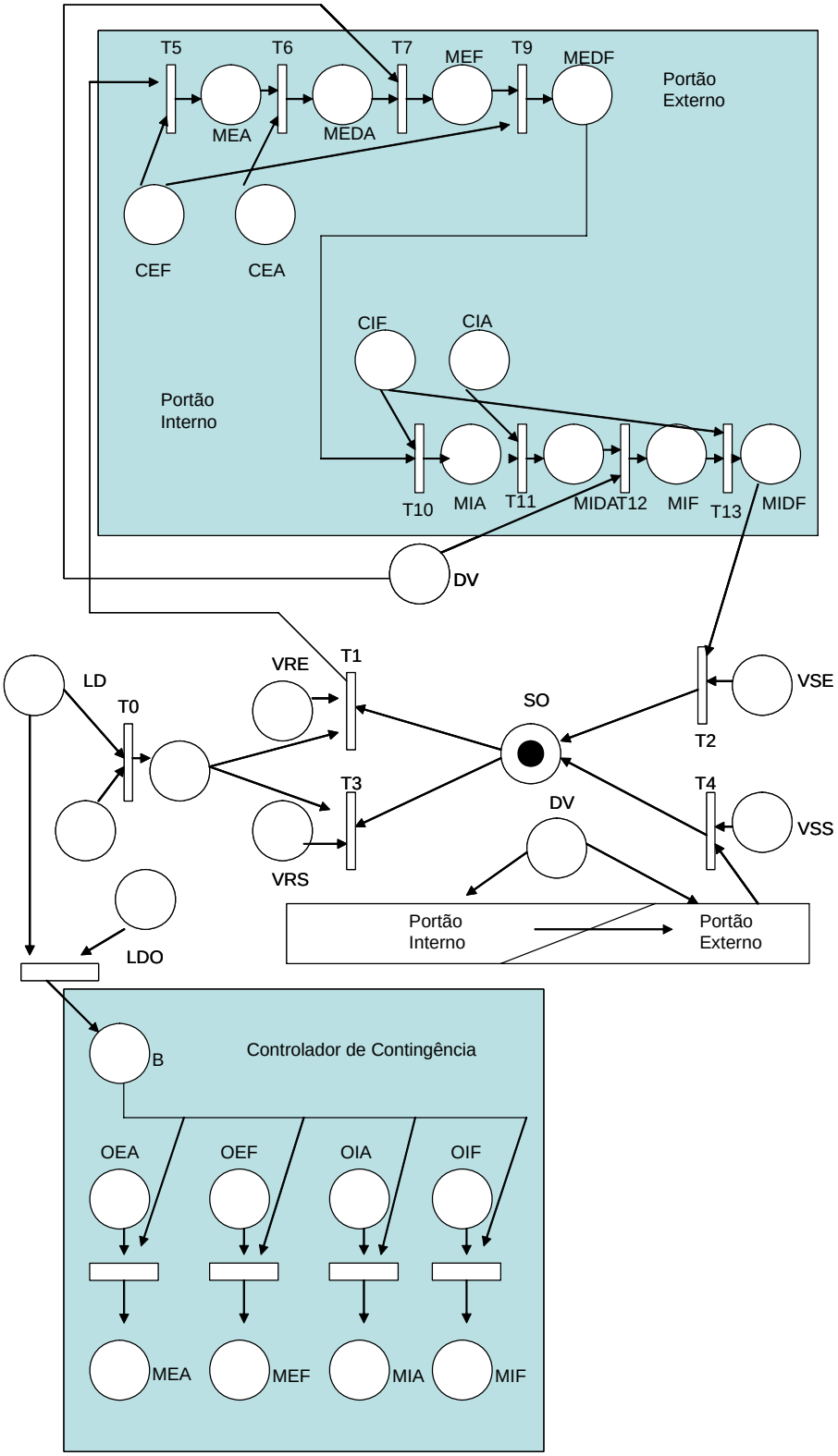


Figura 5. 3 - RdP Portão Eletrônico

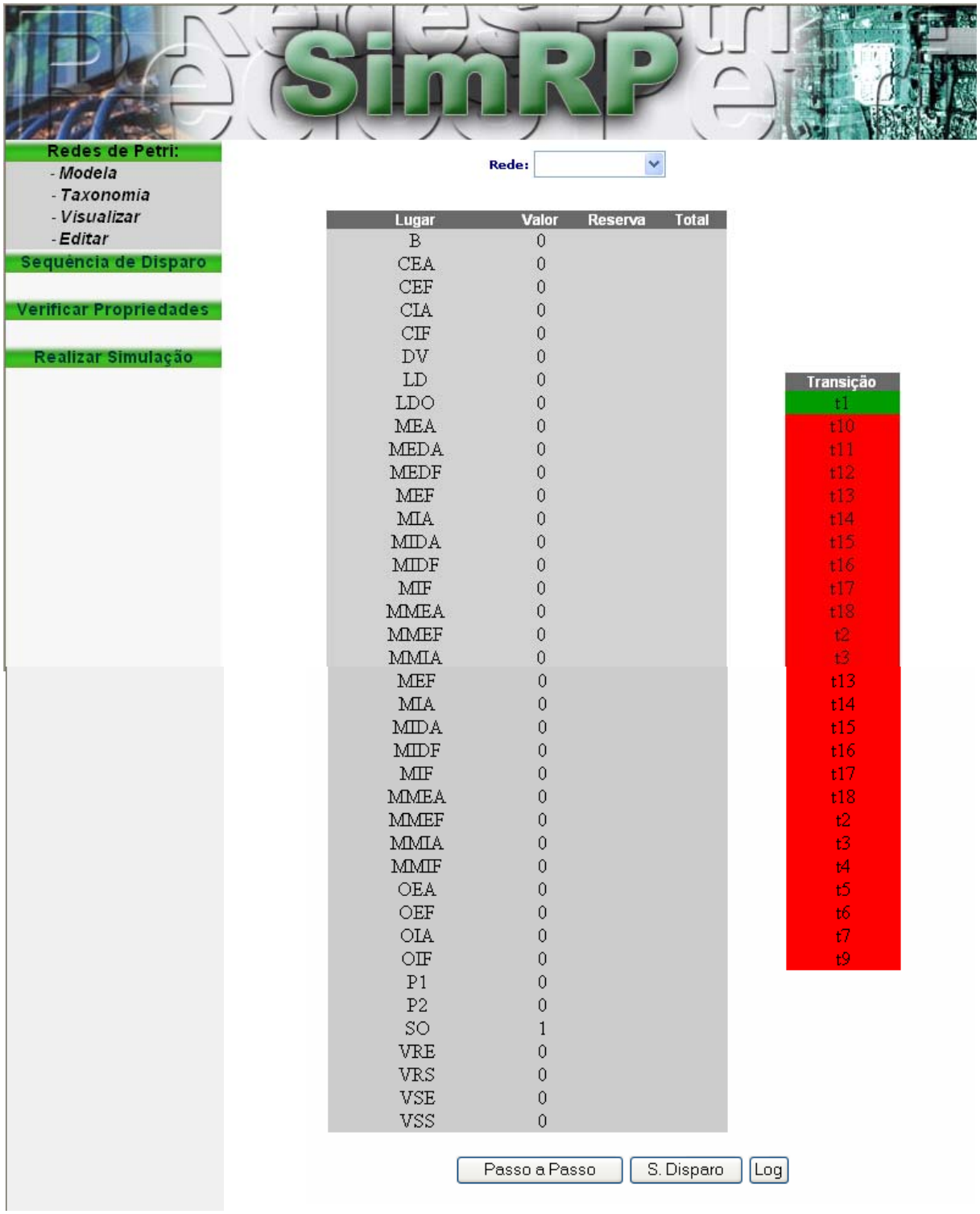


Figura 5. 4 - Exemplo Complexo de RdP Ordinária

5.2 Verificação das Propriedades

Analisando as propriedades referentes ao modelamento da RdPs da figura 5.1, podemos observar a existência de um *conflito estrutural* e de *deadlock*, como observado na figura 5.5.

Figura 5. 5 - Verificação das Propriedades

5.3 Modelamento da RdPs Temporal

Na figura 5.6 podemos observar o modelamento de uma RdPs Temporal.

Figura 5. 6 - Modelamento da RdPs Temporal

5.4 Modelamento da RdPs Temporizada

Na figura 5.7 podemos observar o modelamento de uma RdPs Temporizada.

Figura 5. 7 - Modelamento da RdPs Temporizada

5.5 Exemplo para uma RdP interpretada e a geração automática de código VHDL

Para exemplificar a simulação de uma RdP Interpretada (vide capítulo 2 seção 2.6.1) e a geração de código VHDL (vide capítulo 2 seção 2.8), utilizaremos parte de um exemplo extraído de [2]. Este descreve o comportamento de um controlador manual de dois portões automáticos.

Para ilustrar este comportamento, a figura 5.8 apresenta o descrição deste controlador através de uma RdP Ordinária. Para melhor entendimento, utilizaremos as siglas LD – Chave geral de energia ligada, LDO – *Módulo automático desativado*, OEA – *Comando portão externo abrir,* OEF - *Comando portão externo fecha*r, OIA - *Comando portão interno abrir*, OIF - *Comando portão interno fechar*, MEA – *Motor externo abrindo*, MEF - *Motor externo fechando*, MIA - *Motor interno abrindo* e MIF *Motor externo fechando*.

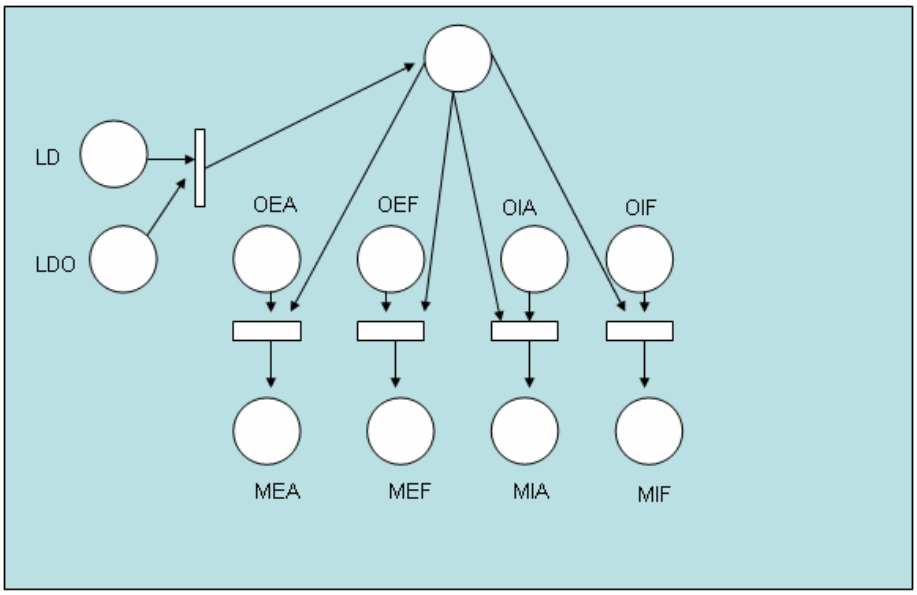


Figura 5. 8 - Exemplo de controlador em RdP Ordinária

Este mesmo exemplo pode ser simulado utilizando uma RdP Interpretada. A figura 5.9 apresenta este exemplo de controlador descrito em uma RdP Interpretada. Nesta figura podemos observar que os lugares (OEA, OEF, OIA e OIF) foram associados a uma condição nas transições correspondentes; que no caso deste exemplo será verdadeira se o valor for igual ao valor 1 (vide figura 5.7).

Neste caso, a transição OEA estará sendo habilitada e o recurso "ficha" será adicionada em MEA, representando o motor do portão externo abrindo. A RdP Interpretada da figura 5.9 foi obtida a partir da RdP previamente editada no SimRP (figura 5.8). Neste caso, a edição foi realizada eliminando os lugares respectivos e atribuindo as condições às transições.

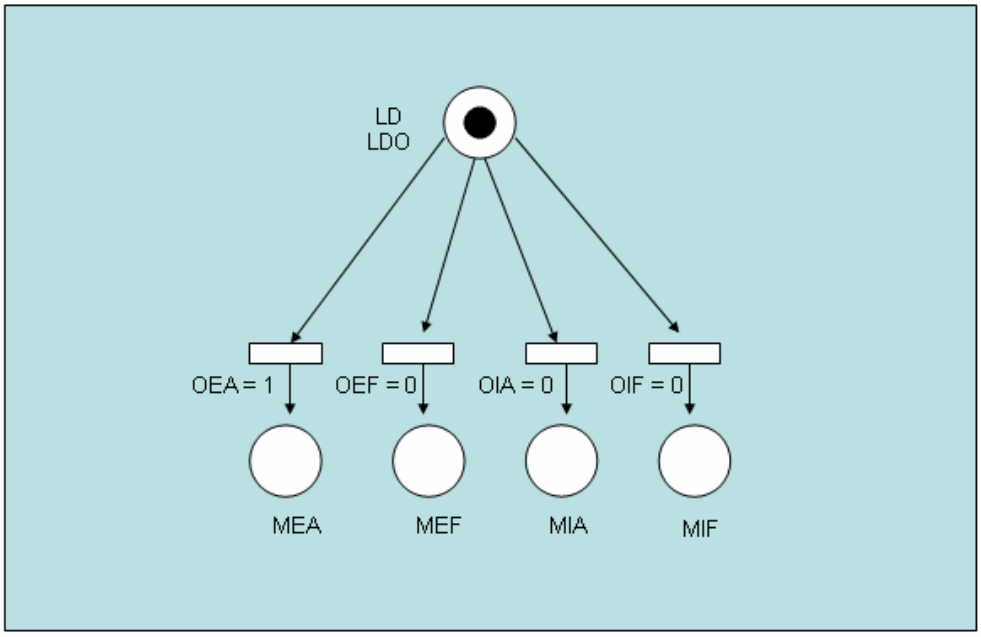


Figura 5. 9- Exemplo de Controlador descrito em uma RdP Interpretada

A figura 5.10 apresenta um passo de simulação do exemplo apresentado na figura 5.9 no SimRP.

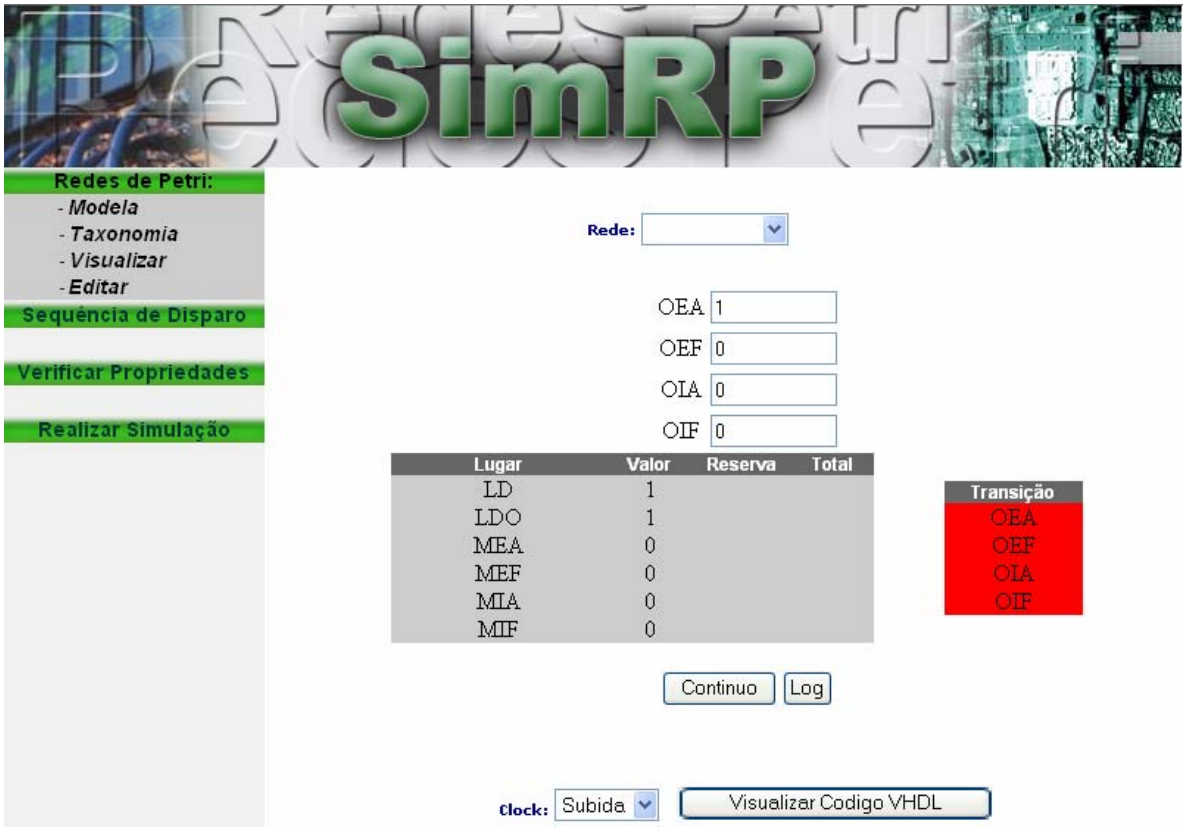


Figura 5. 10 - Tela de Simulação do SimRP

Na figura 5.11 é apresentado a síntese do código VHDL gerado pelo SimRP no aplicativo Quartus [41], nesta figura, podemos verificar informações como o total de elementos lógicos gerados, pinos utilizados na síntese do código. Mais informações sobre resultados da síntese pode ser visto na figura 5.12.

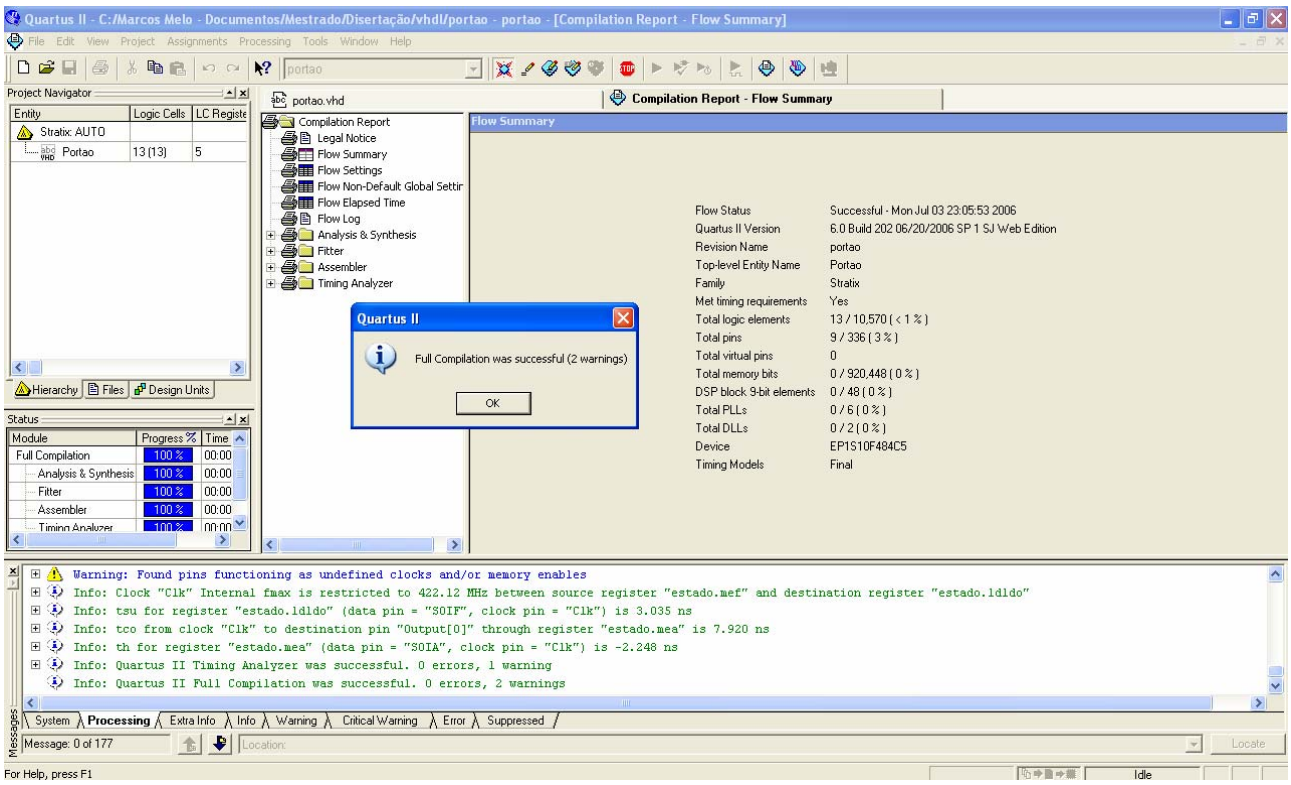


Figura 5. 11- Sintetização do código pelo Quartus

Para a geração de código VHDL no SimRP o usuário pode escolher se a FSM será sensível a borda de subida/descida do *clock*; tipos de dados para entradas/saídas (*bit, stdlogic*, *stdlogic_vector*); definir valores de saídas para a FSM (estes valores não podem ser definidos na RdP Interpretada).

O código VHDL gerado para o exemplo é mostrado no apêndice 5.

	Resource	Usage
1	Logic cells	8
2	Total combinational functions	8
3	Total 4-input functions	3
4	Total 3-input functions	1
5	Total 2-input functions	4
6	Total 1-input functions	0
7	Total 0-input functions	0
8	Combinational cells for routing	0
9	Total registers	5
10	I/O pins	9
11	Maximum fan-out node	Clk
12	Maximum fan-out	5
13	Total fan-out	36
14	Average fan-out	2.12

Figura 5. 12 – Dados gerados a partir da síntese

O código gerado pelo SimRP também é sintetizável pela ferramenta ISE da Xilinx [42].

5.6 CONCLUSÃO

Neste capítulo foram apresentados os resultados obtidos com o SimRP. Os diferenciais verificados neste simulador diferencia o SimRP das demais ferramentas estudas. Este capítulo mostrou alguns exemplos para ilustrar a flexibilidade de SimRP para simular os modelos de RdPs propostos neste trabalho. No caso de trabalhar com RdP Interpretadas pode ser observado a flexibilidade da ferramenta dado que o usuário pode ter várias opções de converter lugares em condições da RdP Interpretada.

Por outro lado foram mostrados os resultados de síntese da RdP Interpretada usando a ferramenta Quartus [41].

6. CONCLUSÃO E TRABALHOS FUTUROS

6.1 O SISTEMA SimRP

O simulador de RdPs foi desenvolvido com a finalidade de ser uma ferramenta CAD de descrição e simulação de código aberto baseado na licença GPL. Esta ferramenta de CAD é capaz de descrever e simular redes de Petri do tipo ordinária, interpretada, temporal e temporizada. Além do anterior, a ferramenta possui a opção de gerar código em VHDL (uma linguagem de descrição de hardware) a partir do modelo de uma rede de Petri interpretada.

Para facilitar o uso e não ser restritiva em relação a um determinado sistema operacional o simulador foi desenvolvido na arquitetura de três camadas, onde a instalação e configuração do sistema são realizadas no servidor. O acesso à aplicação por parte dos clientes é feito através de um *browser* de internet.

O SimRP alcançou os objetivos propostos quanto à flexibilidade (não observada em outras ferramentas de simulação de RdPs estudadas) e a possibilidade de gerar código VHDL para o caso de descrições de controladores usando RdPs (vide exemplo de RdP Interpretada no capítulo 5).

Para a implementação do SimRP foram tidas em conta as definições formais das RdP dadas na literatura estudada. No processo de estudo de outros simuladores (vide capitulo 3) pode ser observado que nem sempre eram seguidas as definições teóricas de RdP. Por exemplo, no Visual Objekt Net++ (que simula RdP ordinárias), no caso de se detectar um conflito estrutural, o simulador disparava aleatoriamente uma das transições sensibilizadas.

Nos testes mostrados no capítulo 5 o SimRP mostrou-se operacional, sendo necessário ainda o seu teste mais consistente para detectar *bugs* e outros tipos de problemas.

6.2 PERSPECTIVAS FUTURAS

O desenvolvimento contínuo deste trabalho torna-se possível, devido ao seu licenciamento ser GPL, para complementar as funcionalidades do simulador e torná-lo mais completo e profissional. Para melhorar a implementação do SimRP são dadas as seguintes sugestões:

a) Implementação de outras taxonomias; não foi contemplado neste trabalho o desenvolvimento de outras taxonomias como as RdP Coloridas, Predicado- Transição, Nebulosas, entre outras.

b) Interface gráfica com simulação através das representações gráficas da teoria de Redes de Petri como os círculos, retângulos e setas.

c) Inclusões de RdPs com modelos híbridos. Por exemplo, permitir que numa RdP Interpretada possam ser incluídos características temporais nas condições das transições, como descrito em [24].

d) Geração automática de descrições de RdPs em outras linguagens de programação.

.

6.3 CONSIDERAÇÕES FINAIS

No mercado existem vários Softwares que simulam o comportamento de uma RdP, contudo a maioria destas ferramentas são softwares proprietários ou software gratuito. Como conseqüência não existe a liberdade de compartilhamento de código, impedindo o desenvolvimento colaborativo de novas funcionalidades, adaptação do sistema de acordo com as necessidades de cada instituição e o compartilhamento do conhecimento gerado no desenvolvimento da solução. Neste sentido, espera-se que o desenvolvimento do SimRP venha a contribuir com o espírito colaborativo envolvido nos conceitos de Software Livre.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Cardoso , J. e R. Valente, Rede de Petri, Editora da UFSC, 1997.

[2] Morais, Cícero Couto e Plínio de Lauro Castrucci, Engenharia de Automação Industrial, Editora LTC, Rio de Janeiro, 2001.

[3] Siqueira, Sérgio Amadeu, Software Livre: a luta pela liberdade do conhecimento, Editora Fundação Perseu Abramo, 2004.

[4] Liu, Wang e Rust, Carsten e Stappert, Friedhelm,A SIMULATION PLATFORM FOR PETRI NET MODELS OF DYNAMICALLY, Universität Paderborn, 2005.

[5] Carsten e Stappert, Friedhelm e Schamberger, Stefan, Integrating Load Balancing Into Petri-Net Based Embedded, Universität Paderborn, 2005.

[6] López, Oscar e Laguna, Miguel A. e García, Francisco, Representación de Requisitos mediante Redes de Petri Coloreadas, Universidad de Valladolid eUniversidad de Salamanca, 2002.

[7] Murata, Tadau, Petri Nets: Properties, Analysis and Applications, IEEE, 1989

[8] Llorens, Marisa e Oliver, Javier, Redes Reconfigurables Controladas por Marcado: Redes de Petri con Câmbios Dinámicos Estructurales, UPV e Universidad de Valladolid eUniversidad de Salamanca, 2005.

[9] Carrasco, Eliana, ESTRUCTURAS GENERALIZADAS PARA CONTROLADORES LÓGICOS MODELADAS MEDIANTE REDES DE PETRI, Universidad Nacional de Colombia sede Medellín, 2002.

[10] Barros, Tomaz C e Santos, Edval J. P. e Melo, Israel L.,SISTEMAS DE PRODUÇÃO EM LOTES: MODELAGEM E ANÁLISE COM BASE EM REDES DE PETRI T-TEMPORIZADAS E EQUAÇÕES DIFERENCIAIS, Universidade Federal de Pernambuco, 2003.

[11] Aarhus, Denmark, Petri Nets 2000, Introductory Tutorial Petri Nets, UNIVERSITY OF AARHUS, 2000.

[12] Apostila: Ejemplos basados en Redes de Petri Temporales y Autómatas Finitos Temporizados, UTN-FRBB.

[13] Moo, Sergio Antonio Pérez, Modelado y Control de Sistemas Híbridos con Redes de Petri Difusas y Redes Neuronales, INSTITUTO POLITÉCNICO NACIONAL, 2002.

[14] Fernandes, João Miguel Lobo, Rede de Petri e VHDL na especificação de controladores paralelos, Universidade do Minho, 1999.

[15] Marra, Felipe Walcarenchi e Coelho, Victor e Nedjah, Nadia, Tradutor de C para VHDL – C2VHDL,UERJ – Universidade do Estado do Rio de Janeiro, 2005.

[16] C.A.Petri, Kommunication mit Automaten. Schriften des IIM Nr.2, Institut für Instrumentelle Mathematik, Bonn, 1962. Traduzida para o inglês como: Communication with Automata, Technical Report RADC-TR-65-377, Griffiths Air Force Base, New York, Vol.1, Suppl.1, 1966.

[17] T.Murata, Petri Nets: Properties, Analysis and Applications, Proceedings of the IEEE, 77(4)541-580, april de 1989.

[18] Cassandras, Chistos G., Stéphane Lafortune, Introduction to Discrete Event Systems, Ed. Springer, New York, 1999.

[19] Fowler, Martine e Kendall Scott, UML Essencial, Um breve guia para linguagem-padrão de modelamento de objetos, Segunda Edição, Ed. Bookman, Rio de Janeiro, 2000.

[20] http://www.softwarelivre.gov.br/diretrizes, acessado em 01/05/2006.

[21] Calazans , Ney Laert Vilar, Alessandro Noriaki Ide, Edson Ifarraguirre Moreno, Taciano Ares Rodolfo e Fernando Gehm Moraes, Tutorial e Diretivas para Captura de Projeto, Validação e Prototipação de Módulos de Hardware Descritos em SystemC, FACULDADE DE INFORMÁTICA PUCRS, 2003

[22] http://httpd.apache.org/docs/2.2, acessado em 20/12/2005.

[23] http://dev.mysql.com/doc/refman/4.1/pt/index.html, acessado em 20/12/2005.

[24] Zapata, German e Eliana Carrasco, Estructuras Generalizadas para Controles Lógicos

Modelados Mediante Rede de Petri, Universidad Nacional de Colômbia sede Medellín, 2002.

[25] Braga, André Luiz Sordi, VANNGen – Uma Fermanta CDA Flexível Para A Implementação de Redes Neurais Artificiais Em Hardware, Universidade de Brasília, 2005.

[26] Hart, David Augustus, Petri Net, Cornell University, 2000.

[27] Liu, Wang Yan, Carsten Rust e Friedhelm Stappert, Simulation Platform for Net Models of Dynamically Modifiable embedded Systems, Universitat Pderborn, 2002

[28] www.teses.usp.br/teses/disponiveis/ 3/3132/tde-30032001-150611/, acessada em 05/06/2006.

[29] http://pdv.cs.tu-berlin.de/~timenet/, acessada em 05/06/2206

[30] http://www.eng.uerj.br/~ldmm/controle%20de%20processos/manual-pt.html, acessada em 05/06/2006.

[31] http://staff.um.edu.mt/jskl1/petrisim/htmlman/manual5.html#c42, acessada em 05/06/2006.

[32] Manual do Visual Objekt Net ++

[34] http://gcc.gnu.org/, acessada em 05/06/2006.

[35] http://www.postgresql.org/, acessada em 05/06/2006.

[36] http://www.oracle.com/global/br/index.html, acessada em 05/06/2006.

[37] http://java.sun.com/, acessada em 05/06/2006.

[38] http://www.javascript.com/, acessada em 05/06/2006.

[39] www.portaljava.com/home/modules. php?name=News&file=article&sid=425, acessada em 05/06/2006.

[40] www.mysqlbrasil.com.br/treinamentos/banco, acessada em 05/06/2006.

[41] www.altera.com, acessada em 05/06/2006.

[42] www.xilinx.com, acessada em 05/06/2006.

Apêndice 1 – Diagramas do Sistema

1. Diagramas de Classe

1.1 Define Rede de Petri

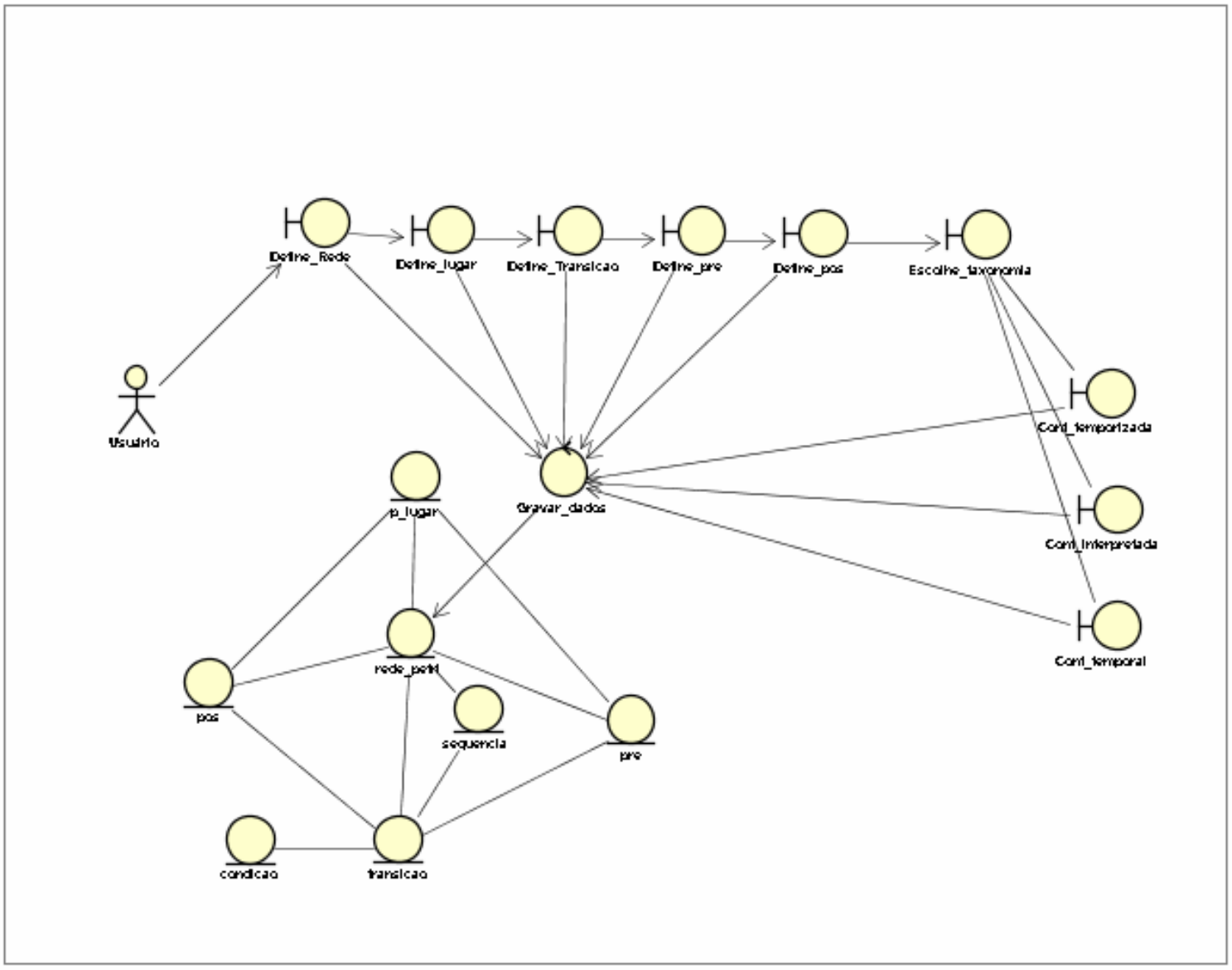


Diagrama de Classe: Define Rede de Petri

1.2 Taxonomia

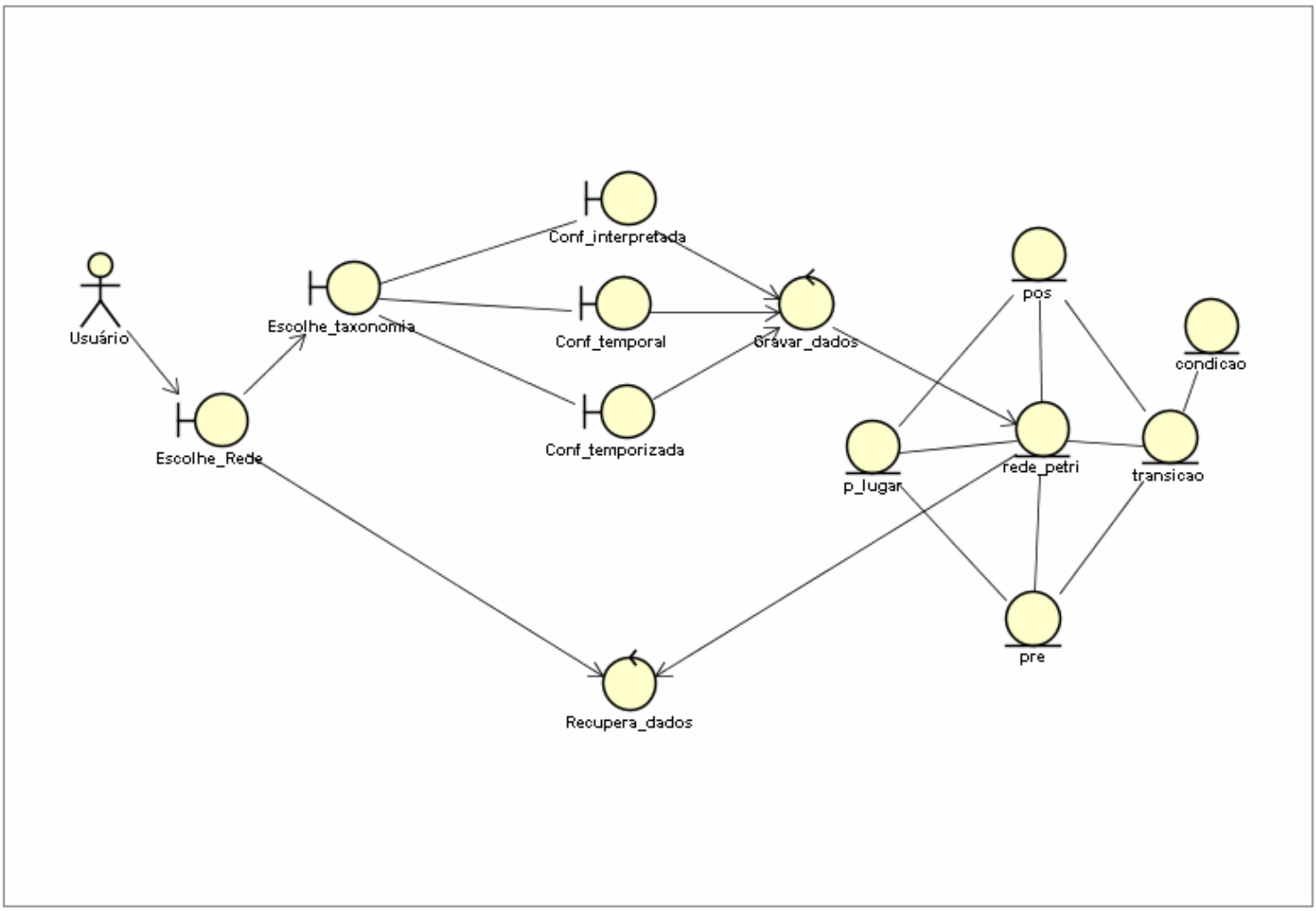


Diagrama de Classe: Taxonomia

1.3 Visualizar

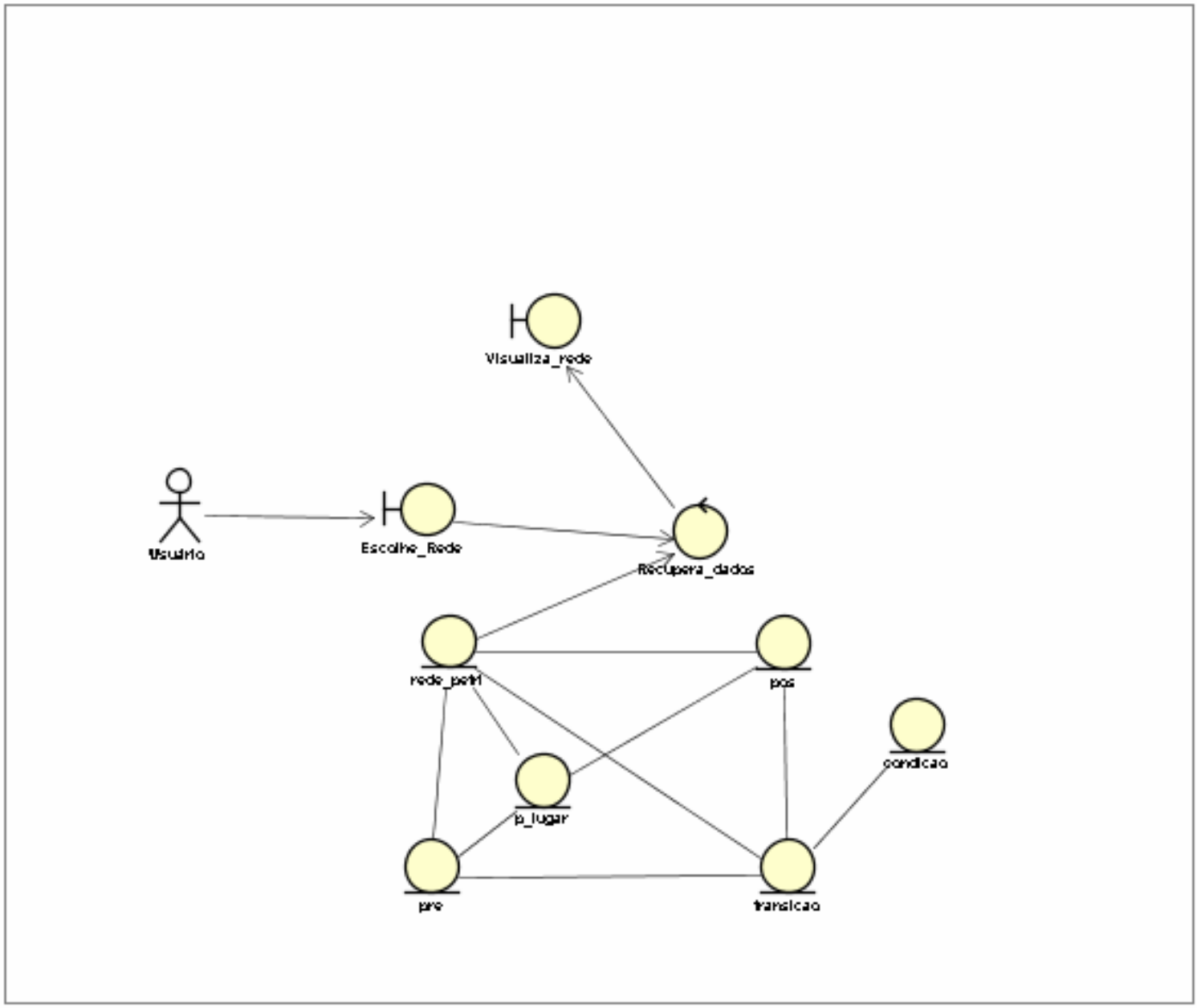


Diagrama de Classe: Visualizar

1.4 Editar

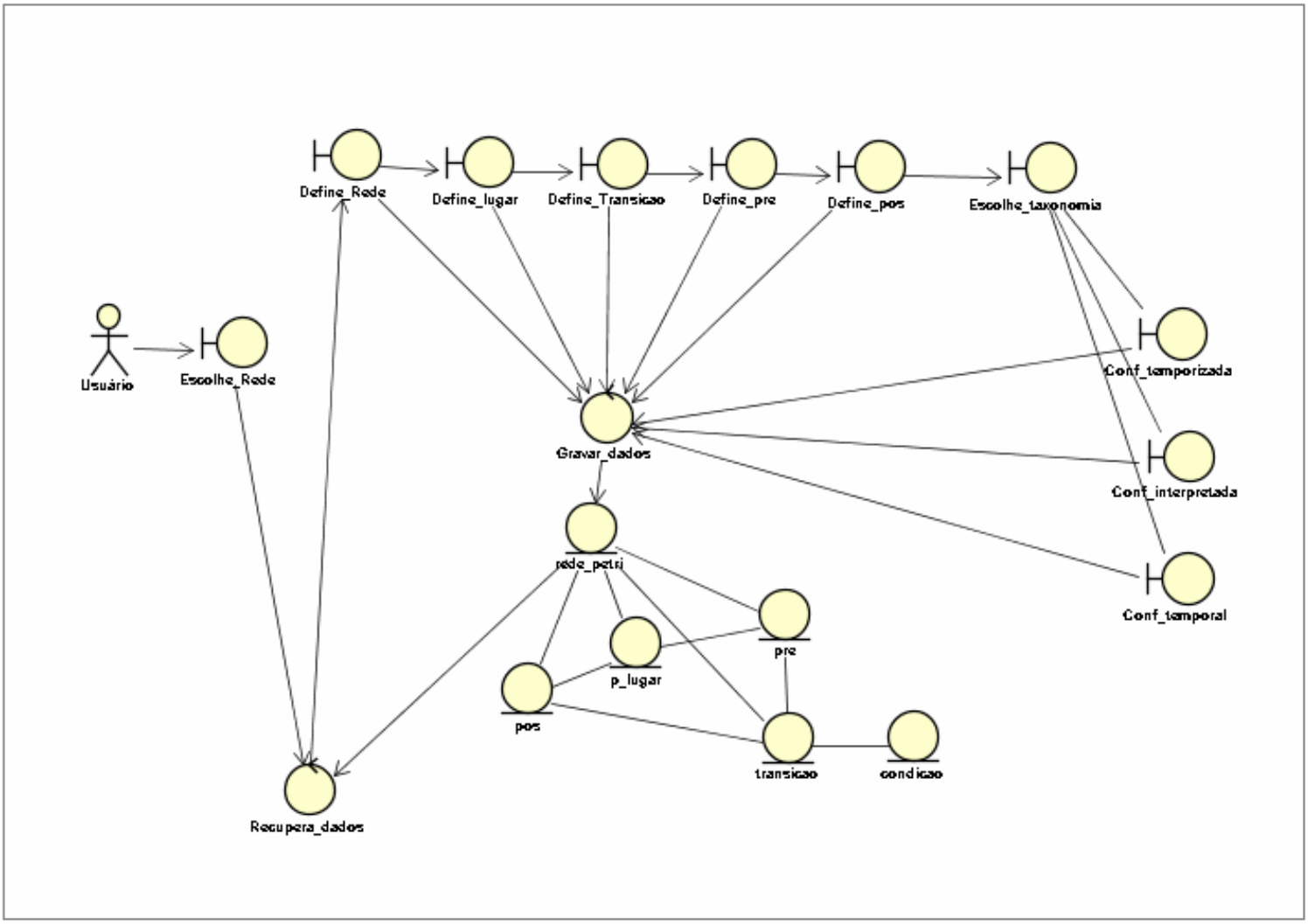


Diagrama de Classe: Editar

1.5 Seqüência de Disparo

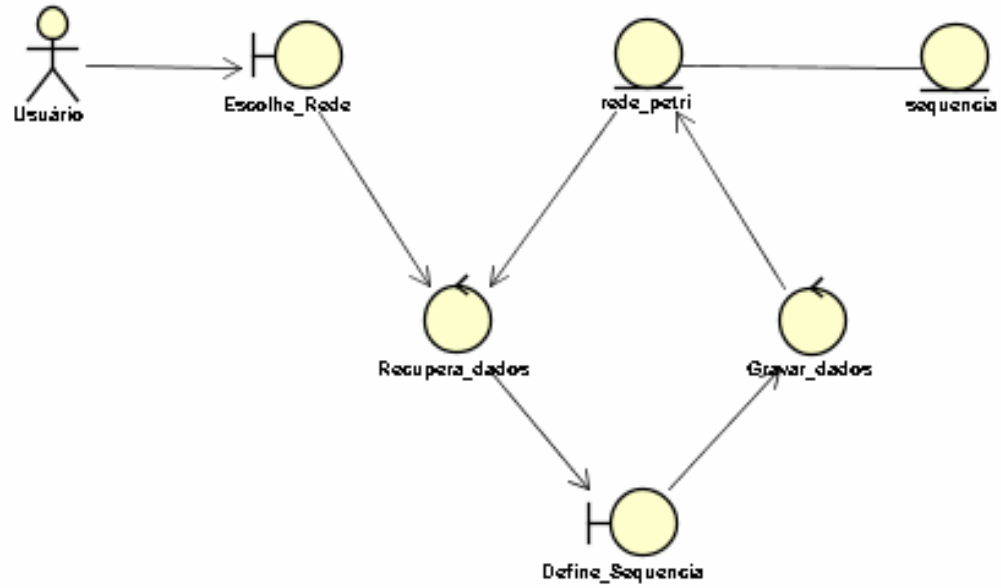


Diagrama de Classe: Definir Seqüência de Disparo

1.6 Verificar Propriedades

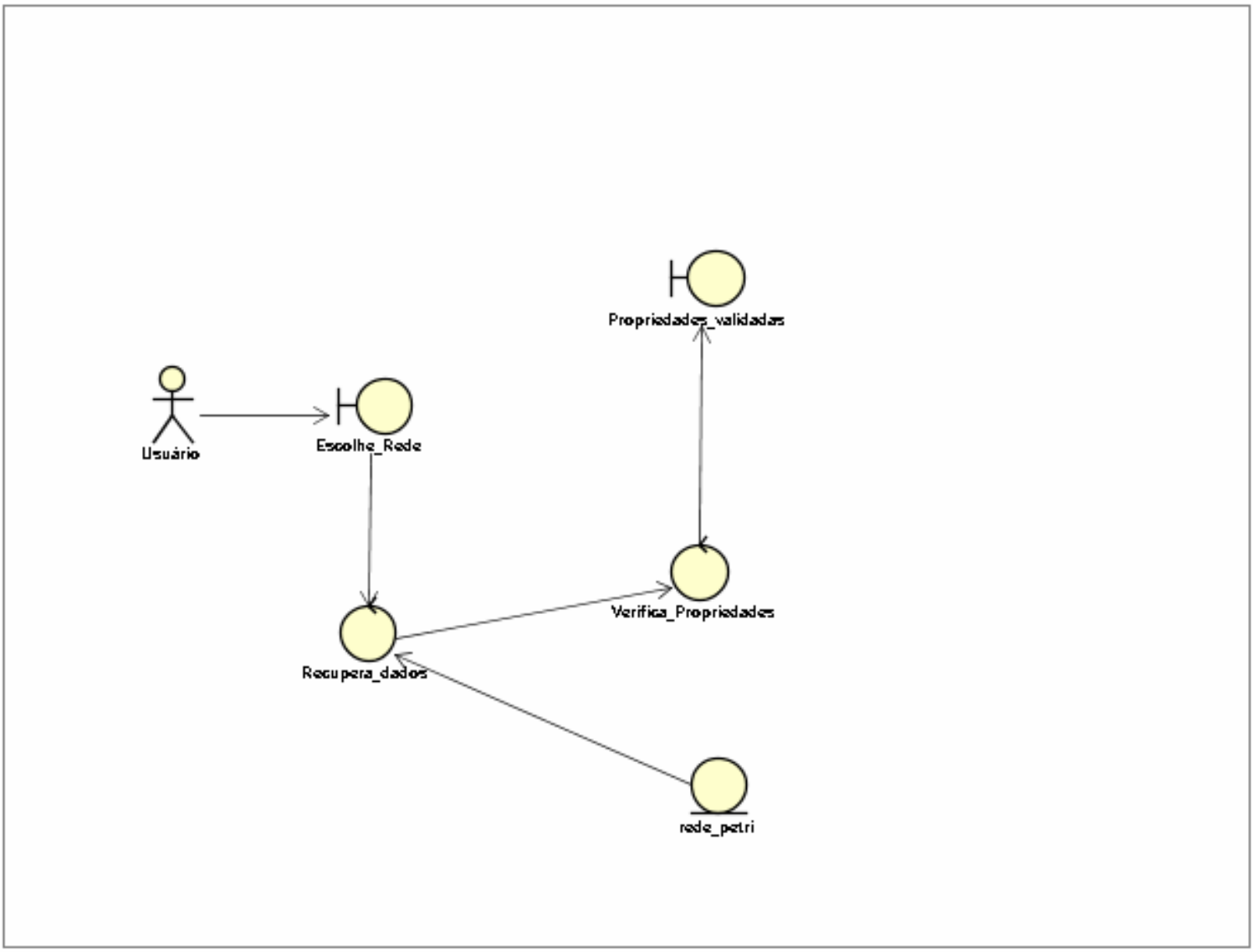


Diagrama de Classe: Propriedades

1.7 Simulação

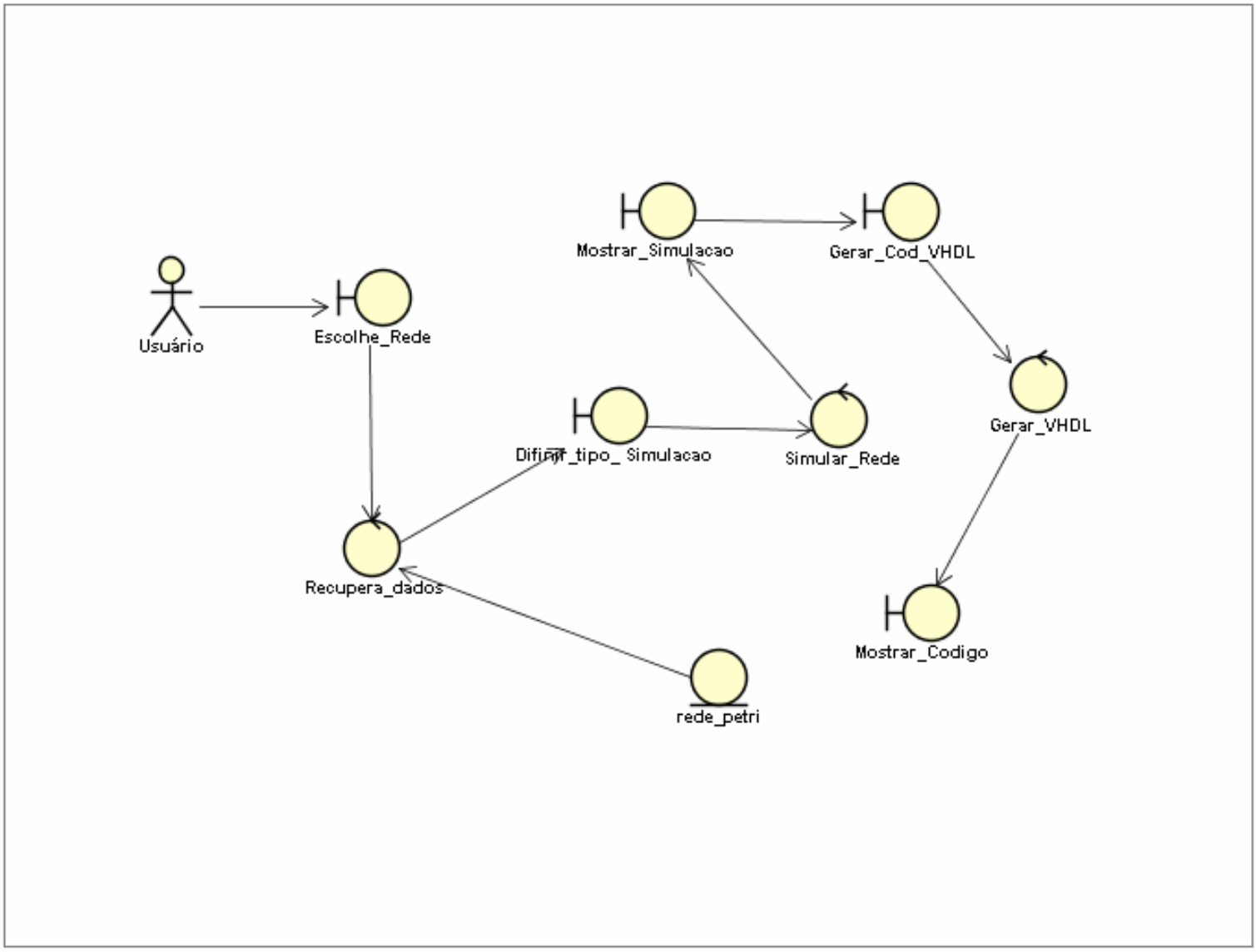


Diagrama Classe: Simulação

Apêndice 2 – Descrição dos casos de uso

As tabelas de 1 a 11 detalham os casos de uso descritos na figura 31.

Caso de Uso: Definir Rede	
Descrição	Neste caso de uso o usuário poderá modelar as Redes de Petri ordinárias, é possível também definir as outras taxonômicas como temporal, temporizada e interpretada além de definir as configurações para a geração de código VHDL
Autor:	Usuário
Fluxo principal	- O usuário solicita a função Define Rede; - O usuário apresenta a tela de edição de rede; - O sistema solicita os dados iniciais da rede; - O sistema apresenta a tela de edição de lugares; - O sistema solicita o conjunto de lugares da rede, o conjunto P; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição de transição; - O sistema solicita o conjunto de transição da rede, o conjunto T; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição das pré-condições; - O sistema solicita o conjunto das pré-condições da rede, o conjunto Pre; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição de pós-condição;

	- O sistema solicita o conjunto de pós-condições da rede, o conjunto pós-condição; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição das taxonomias;
Fluxo Alternativo	Não se aplica a este caso de uso.
Pré-condição	Não se aplica a este caso de uso.
Pós-condição	Não se aplica a este caso de uso.
Observação:	Neste modelamento é possível definir características de outras taxonomias.

Tabela 1 - Caso de uso definir rede

Caso de Uso: Temporal	
Descrição	Neste caso de uso o usuário pode incluir características das redes temporais as redes ordinárias definidas no sistema.
Autor:	Usuário
Fluxo principal	- Após a definição de rede ordinária; - O sistema apresenta a tela de edição das redes temporais; - O sistema solicita os dados desta rede; - O sistema armazena as informações no banco de dados.
Fluxos Alternativos	Não se aplica a este caso de uso.
Pré-condição	Rede ordinária já modelada
Pós-condição	Não se aplica a este caso de uso.
Observação:	Caso já tenha sido definida uma outra taxonomia a Rede de Petri escolhida, não será possível utilizar este caso de uso.

Tabela 2 -Caso de Uso: Temporais

Caso de Uso: Temporizadas	
Descrição	Neste caso de uso o usuário pode incluir características das redes temporizadas as redes ordinárias definidas no sistema.
Autor:	Usuário
Fluxo principal	- Após a definição de rede ordinária; - O sistema apresenta a tela de edição das redes temporizadas; - O sistema solicita os dados desta rede; - O sistema armazena as informações no banco de dados.
Fluxos Alternativos	Não se aplica a este caso de uso.
Pré-condição	Rede ordinária já modelada
Pós-condição	Não se aplica a este caso de uso.
Observação:	Caso já tenha sido definida uma outra taxonomia a Rede de Petri escolhida, não será possível utilizar este caso de uso.

Tabela 3 - Caso de Uso: Temporizadas

Caso de Uso: Interpretadas	
Descrição	Neste caso de uso o usuário pode incluir características das redes Interpretadas nas redes ordinárias definidas no sistema.
Autor:	Usuário
Fluxo principal	- Após a definição de rede ordinária; - O sistema apresenta a tela de edição das redes Interpretadas; - O sistema solicita os dados desta rede; - O sistema armazena as informações no banco de dados.
Fluxos Alternativos	Não se aplica a este caso de uso.

Pré-condição	Rede ordinária já modelada
Pós-condição	Não se aplica a este caso de uso.
Observação:	Caso já tenha sido definida uma outra taxonomia a Rede de Petri escolhida, não será possível utilizar este caso de uso.

Tabela 4 - Caso de Uso: Interpretadas

Caso de Uso: Editar Rede de Petri	
Descrição	Neste caso de uso o usuário poderá editar as redes de petri definidas anteriormente pelo caso de uso Definir Rede.
Autor:	Usuário
Fluxo principal	- O usuário solicita a função Editar Rede; - O usuário apresenta a tela de edição de rede; - O sistema solicita os dados iniciais da rede; - O sistema apresenta a tela de edição de lugares; - O sistema solicita o conjunto de lugares da rede, o conjunto P; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição de transição; - O sistema solicita o conjunto de transição da rede, o conjunto T; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição das pré-condições; - O sistema solicita o conjunto das pré-condições da rede, o conjunto Pre; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição de pós-condição; - O sistema solicita o conjunto de pós-condições da rede, o conjunto

	pós-condição; - Os dados serão armazenados no banco de dados; - O sistema apresenta a tela de edição das taxonomias;
Fluxo Alternativo	Não se aplica a este caso de uso.
Pré-condição	Redes de Petri já modeladas pelo sistema.
Pós-condição	Não se aplica a este caso de uso.
Observação:	Neste caso de uso também é possível deletar as redes de petri.

Tabela 5 - Editar Redes de Petri

Caso de Uso: Gerar código VHDL	
Descrição	Neste caso de uso o usuário poderá definir os atributos necessários para a geração de um código VHDL a partir do modelamento de uma rede de petri interpretada.
Autor:	Usuário
Fluxo principal	- O sistema apresenta a tela de edição das redes Interpretadas; - Na apresentação desta tela, está disponível a opção gerar código VHDL, caso abilitada esta opção; - O sistema apresenta a tela de edição para os atributos de geração de código VHDL; - O sistema armazena as informações no banco de dados.
Fluxos Alternativos	Não se aplica a este caso de uso.
Pré-condição	O modelamento de uma rede de petri interpretada.
Pós-condição	Simulação da rede modelada.
Observação:	Este caso de uso não é responsável diretamente pela a geração do código VHDL, mas sim edição dos parâmetros necessários para

	esta geração, a geração propriamente dita é realizado pelo caso de uso Simular rede de petri.

Tabela 6 - Gerar código VHDL

Caso de Uso: Definir seqüência de disparo	
Descrição	Neste caso de uso o usuário poderá definir a seqüência de disparo às redes de petri ordinárias.
Autor:	Usuário
Fluxo principal	- O sistema apresenta a tela de seqüência de disparo; - O sistema solicita a ordem da seqüência; - O sistema armazena as informações no banco de dados.
Fluxo Alternativo	Não se aplica a este caso de uso.
Pré-condição	Rede de petri ordinária já modelada.
Pós-condição	Não se aplica a este caso de uso.
Observação:	Caso seja definida uma seqüência de disparo a uma rede de petri que não seja ordinária, está seqüência é ignorada.

Tabela 7 - Definir seqüência de disparo

Caso de Uso: Visualizar rede de petri	
Descrição	Neste caso de uso o usuário poderá visualizar o modelamento de uma rede de petri já definida no sistema.
Autor:	Usuário
Fluxo principal	- O sistema apresenta a tela para a escolha da rede; - O sistema recupera os dados no banco de dados referente a esta rede; - O sistema apresenta os dados.

Fluxo Alternativo	Não se aplica a este caso de uso.
Pré-condição	Rede de petri já modelada.
Pós-condição	Não se aplica a este caso de uso.
Observação:	

Tabela 8 - Visualizar rede de petri

Caso de Uso: Editar Taxonomia	
Descrição	Neste caso de uso o usuário poderá editar e adicionar características adicionais na rede de petri ordinária, mudando assim sua taxonomia.
Autor:	Usuário
Fluxo principal	- O sistema apresenta a tela de edição das Taxonomias; - O sistema solicita os dados da taxonomia; - O sistema armazena as informações no banco de dados.
Fluxo Alternativo	Não se aplica a este caso de uso.
Pré-condição	Rede de petri já modelada
Pós-condição	Não se aplica a este caso de uso.
Observação:	Cada modelamento de rede de petri somente poderá possuir um tipo de taxonomia.

Tabela 9 - Editar Taxonomia

Caso de Uso: Verificar Propriedades	
Descrição	Neste caso de uso o usuário poderá verificar as propriedades: viva, pura, conflito e iniciáveis.
Autor:	Usuário

Fluxo principal	- O sistema apresenta a tela para a seleção de redes; - O sistema verifica as propriedades; - O sistema apresenta as propriedades válidas modificando a cor da propriedade em questão.
Fluxo Alternativo	Na verificação da propriedade caso seja encontrada algum Deadlock, o sistema apresentará na tela uma mensagem de aviso.
Pré-condição	Modelamento de rede de petri já definido no modelo
Pós-condição	Não se aplica a este caso de uso.
Observação:	

Tabela 10 - Verificar propriedades

Caso de Uso: Simular Redes de Petri	
Descrição	Neste caso de uso o usuário poderá simular os modelamentos de redes de petri definidos no sistema. A simulação poderá ser passo a passo, por seqüência de disparo ou continua.
Autor:	Usuário
Fluxo principal	- O sistema apresenta a tela de seleção de rede de petri; - O sistema solicita recupera no banco de dados os dados referente a rede de petri; - O sistema pergunta a qual o tipo de simulação desejado; - O sistema apresenta o resultado da simulação;
Fluxo Alternativo	Caso a rede de petri simulada for interpretada e tiver os parâmetro de geração de código VHDL, será disponibilizada a opção de geração de código.
Pré-condição	Modelamento de rede de petri já definida no sistema.
Pós-condição	Não se aplica a este caso de uso.

Tabela 11 - Simulação de rede de petri

Apêndice 3 – Script do Banco de Dados

```
DROP TABLE IF EXISTS `condicao`;
CREATE TABLE `condicao` (
  `Cod_cond` int(10) unsigned NOT NULL auto_increment,
  `Transicao_Cod_trans` int(10) unsigned NOT NULL,
  `variavel` varchar(20) default NULL,
  `valor` varchar(20) default NULL,
  `tipo` int(10) unsigned default NULL,
  `condicional` int(10) unsigned default NULL,
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `op_relacional` varchar(45) default '0',
  `tipo_vhdl` varchar(45) default NULL,
  PRIMARY KEY  (`Cod_cond`,`Transicao_Cod_trans`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `p_lugar`;
CREATE TABLE `p_lugar` (
  `Cod_Lugar` int(10) unsigned NOT NULL auto_increment,
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `Nome` varchar(20) default NULL,
  `Descricao` varchar(255) default NULL,
  `Valor_inicial` varchar(20) default '0',
  `tempo` time default NULL,
  `tempo_max` time default NULL,
  `tempo_min` time default NULL,
  `valor_vhdl` varchar(45) default NULL,
  `tipo_v_vhdl` varchar(45) default NULL,
  PRIMARY KEY  (`Cod_Lugar`,`Rede_Petri_cod_petr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `pos`;
CREATE TABLE `pos` (
  `cod_pos` int(10) unsigned NOT NULL auto_increment,
  `Transicao_Cod_trans` int(10) unsigned NOT NULL,
  `Peso` int(10) unsigned default '1',
  `P_Lugar_Cod_Lugar` int(10) unsigned NOT NULL,
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `tempo` time default NULL,
  `tempo_max` time default NULL,
  `tempo_min` time default NULL,
  PRIMARY KEY
(`cod_pos`,`Transicao_Cod_trans`,`P_Lugar_Cod_Lugar`,`Rede_Petri_cod_petr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='InnoDB free: 11264 kB;
(`Transicao_Cod_trans` `Transicao_Red';

DROP TABLE IF EXISTS `pre`;
CREATE TABLE `pre` (
  `cod_pre` int(10) unsigned NOT NULL auto_increment,
  `Transicao_Cod_trans` int(10) unsigned NOT NULL,
  `P_Lugar_Cod_Lugar` int(10) unsigned NOT NULL,
  `Peso` int(10) unsigned default '1',
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `tempo` time NOT NULL,
```

```
  `tempo_max` time NOT NULL,
  `tempo_min` time NOT NULL,
  PRIMARY KEY  (`cod_pre`,`Transicao_Cod_trans`,`P_Lugar_Cod_Lugar`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='InnoDB free: 11264 kB;
(`Transicao_Cod_trans` `Transicao_Red';

DROP TABLE IF EXISTS `rede_petri`;
CREATE TABLE `rede_petri` (
  `cod_petr` int(10) unsigned NOT NULL auto_increment,
  `Nome` varchar(20) default NULL,
  `Descricao` varchar(255) default NULL,
  `Autor` varchar(20) default NULL,
  `Data_Modelamento` date default NULL,
  PRIMARY KEY  (`cod_petr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

DROP TABLE IF EXISTS `sequencia`;
CREATE TABLE `sequencia` (
  `Cod_sequencia` int(10) unsigned NOT NULL auto_increment,
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `Transicao_Cod_trans` int(10) unsigned NOT NULL,
  `Ordem` int(10) unsigned default NULL,
  PRIMARY KEY
(`Cod_sequencia`,`Rede_Petri_cod_petr`,`Transicao_Cod_trans`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 COMMENT='InnoDB free: 11264 kB;
(`Rede_Petri_cod_petr`) REFER `dbrede';

DROP TABLE IF EXISTS `transicao`;
CREATE TABLE `transicao` (
  `Cod_trans` int(10) unsigned NOT NULL auto_increment,
  `Rede_Petri_cod_petr` int(10) unsigned NOT NULL,
  `Nome` varchar(20) default NULL,
  `Descricao` varchar(255) default NULL,
  `tempo` time default NULL,
  `tempo_MAX` time default NULL,
  `tempo_MIN` time default NULL,
  PRIMARY KEY  (`Cod_trans`,`Rede_Petri_cod_petr`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;</PRE></BODY></HTML>
```

Apêndice 4 – Manual de Usabilidade do Sistema

Este apêndice tem por objetivo auxiliar os usuários a operar de melhor forma o Simulador SimRP. Na figura 1 é apresentada a tela principal do simulador, o qual foi desenvolvido de tal forma que a qualquer momento da utilização é possível acessar qualquer funcionalidade do sistema através das opções do menu localizado à esquerda da página.

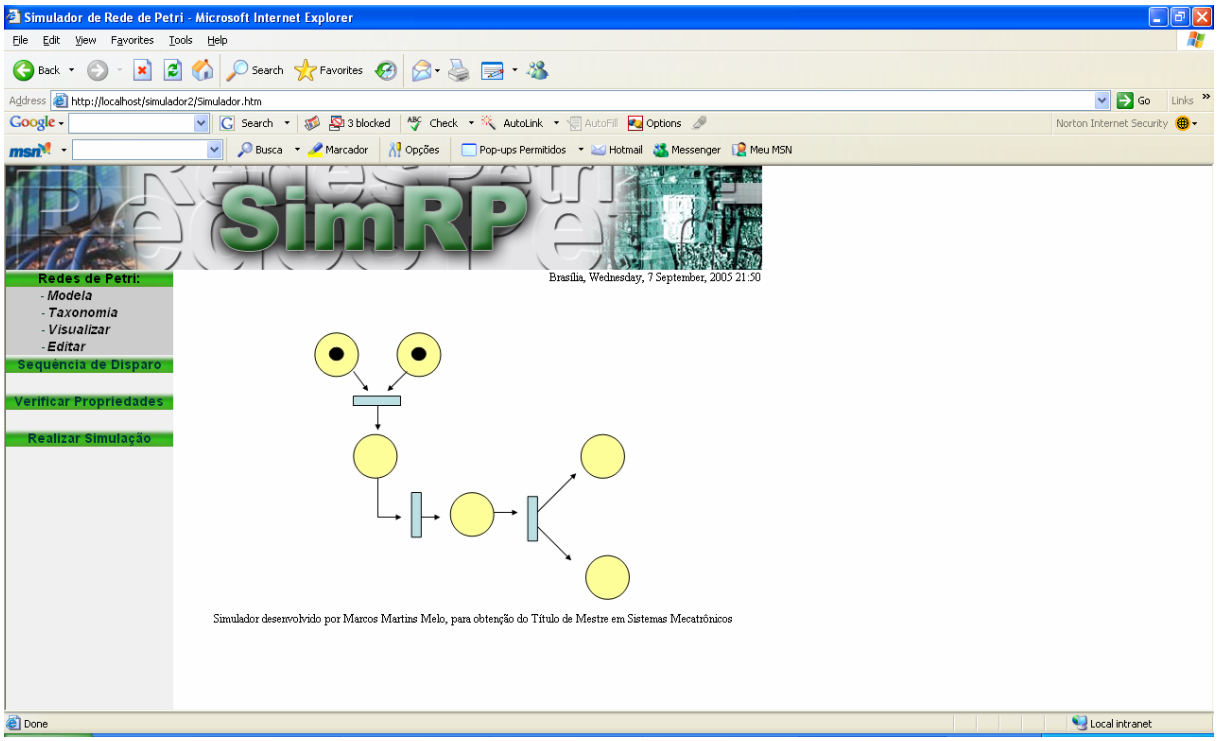


Figura 1 – Tela Principal do Sistema

Para iniciar o modelamento das Redes de Petri, selecione a opção Modela, localizada no menu. Após esta seleção o sistema irá solicitar os dados iniciais para o modelamento de uma rede como demonstrado na figura 2.

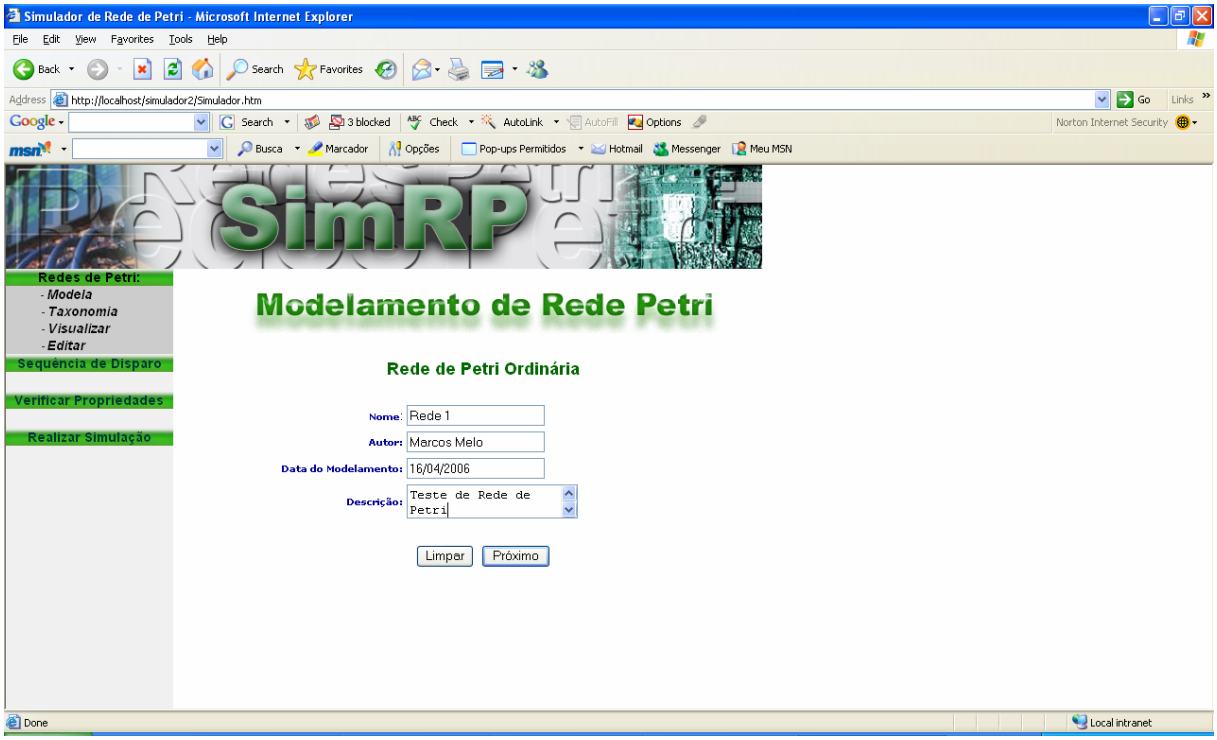


Figura 2 - Tela de edição de rede

Após o preenchimento dos dados solicitados na figura 2, selecione o botão Próximo. O sistema solicitará então os dados referentes ao conjunto de lugares como demonstrado na figura 3.

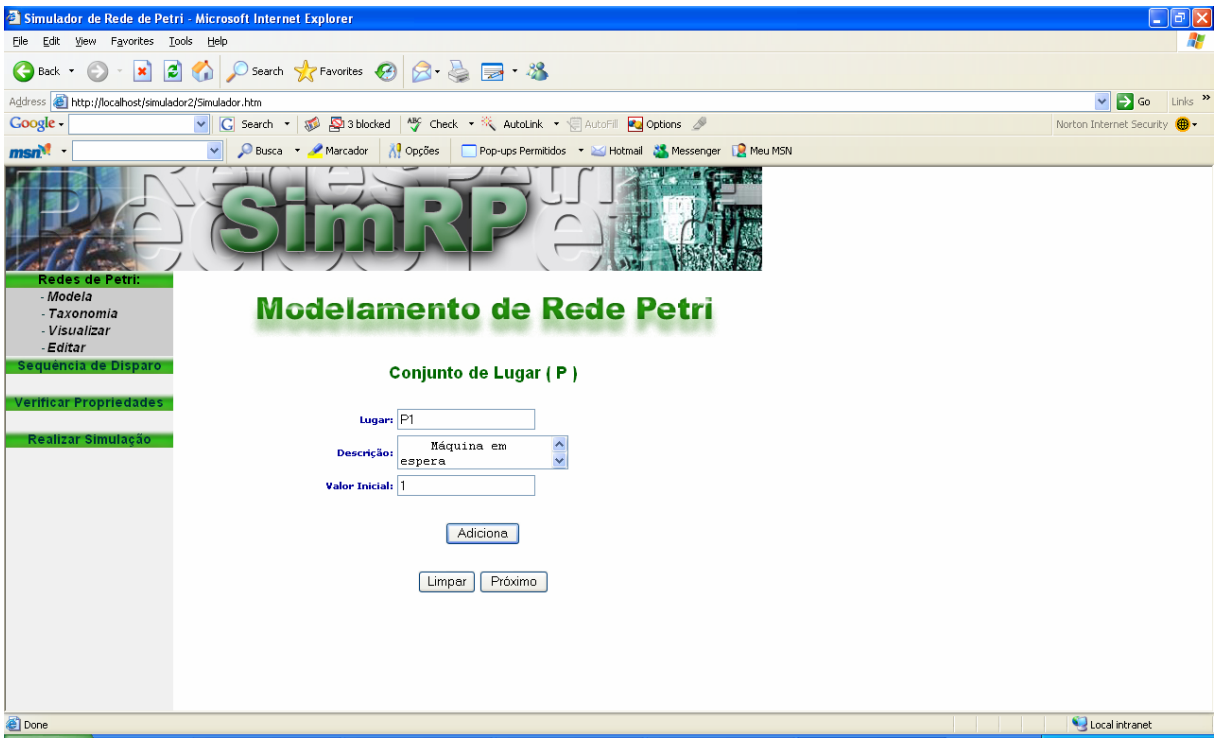


Figura 3 – Tela de edição de Lugares

Para cada lugar adicionado ao modelamento é necessário selecionar a opção Adicionar. O próximo passo é definir o conjunto de transição como demonstrado na figura 4.

Figura 4 - Tela de edição das transições

Em seguida o sistema irá solicitar o conjunto das pré-condições, como demonstrado na figura 5.

Figura 5 – Tela de edição das pré-condições

Após este passo o sistema irá solicitar o conjunto das pós condições como demonstrado na figura 6.

Figura 6 – Tela de edição das pós-condições

No momento do modelamento também é possível acrescentar os atributos referentes a uma das taxonomias citadas na figura 7, caso a intenção seja o modelamento de uma rede ordinária basta selecionar a opção concluir.

Caso tenha necessidade de atribuir atributos referentes a uma taxonomia posteriormente, basta selecionar a qualquer momento a opção Taxonomia do Menu à esquerda da tela que tela demonstrada na figura 7, será apresentada.

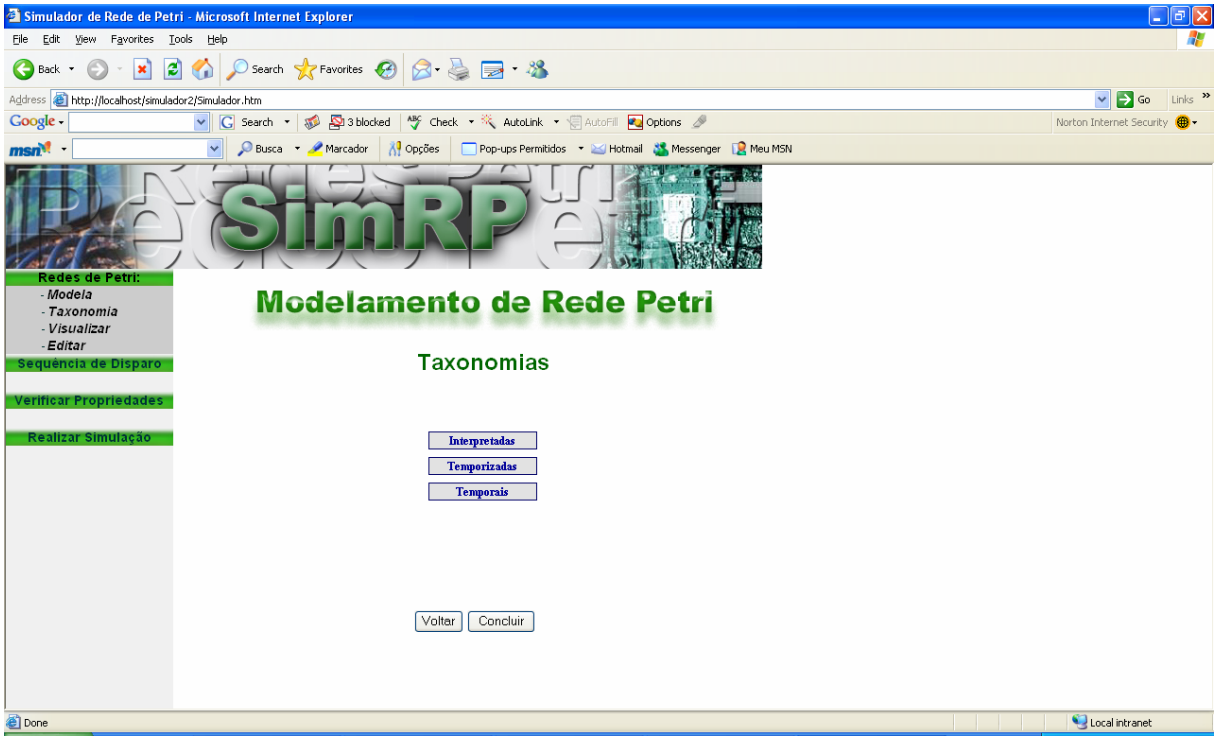


Figura 7 - Tela de edição das Taxonomias

A tela da figura 8 é responsável pela configuração das Redes de Petri interpretadas, está tela também oferece a opção de configuração da geração de código VHDL.

Figura 8 - Tela de edição das redes interpretadas

Na figura 9 é apresentada a tela de configuração das Redes de Petri temporizadas e na figura 10 a tela de configuração das Redes de Petri temporais.

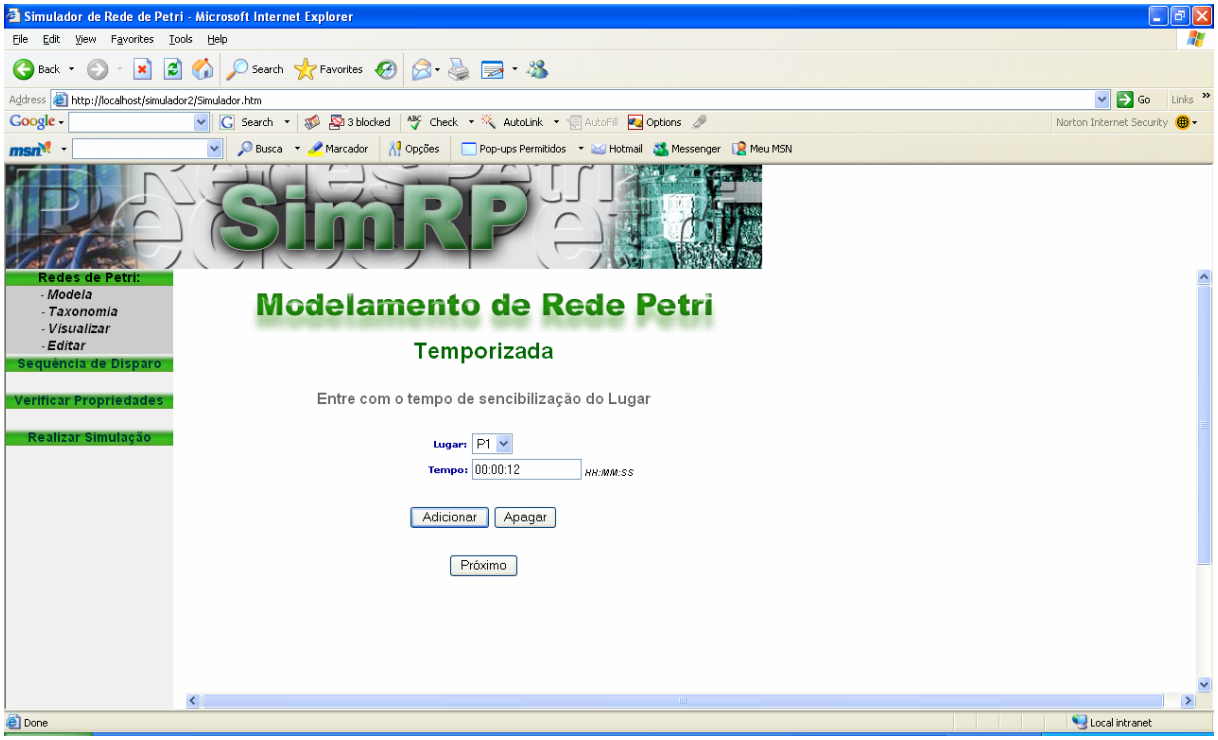


Figura 9 - Tela de edição das redes temporizadas

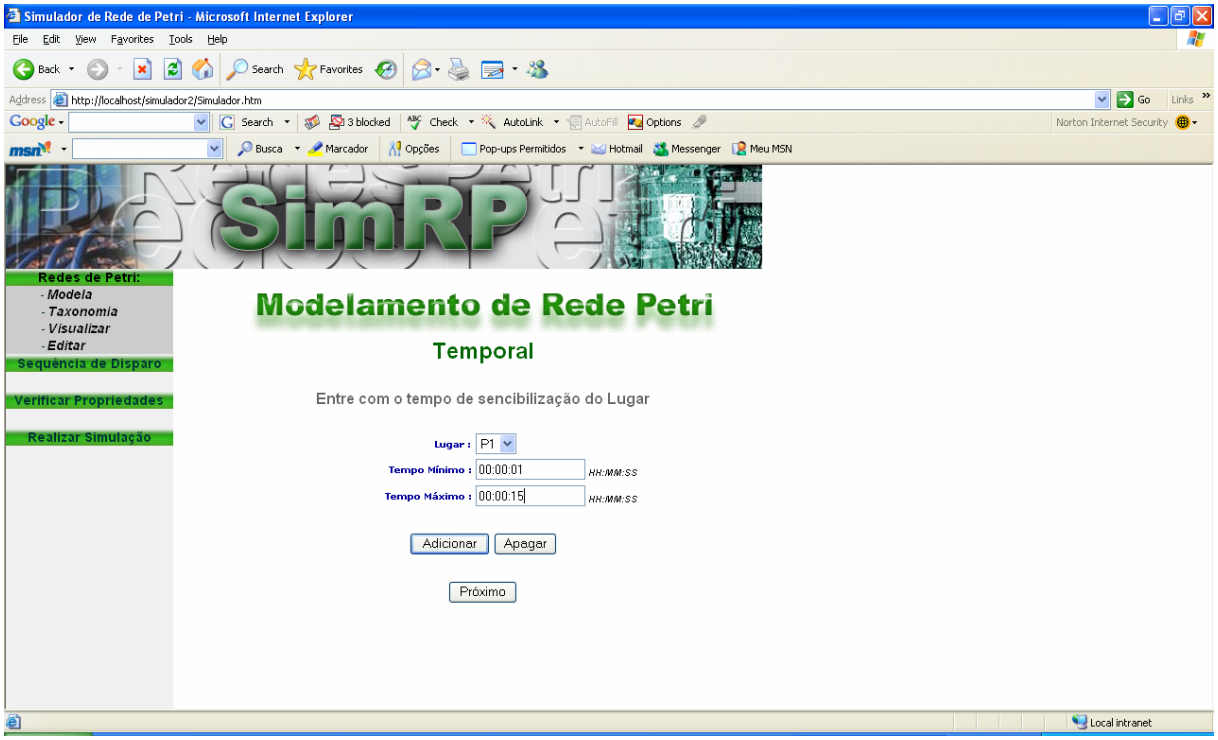


Figura 10 - Tela de edição das redes temporais

Quando a opção do menu Taxonomia, Visualizar, Editar, Seqüência de Disparo, Verificar Propriedades e Simulação é selecionada, o sistema solicita a escolha da rede como demonstrada na figura 11.

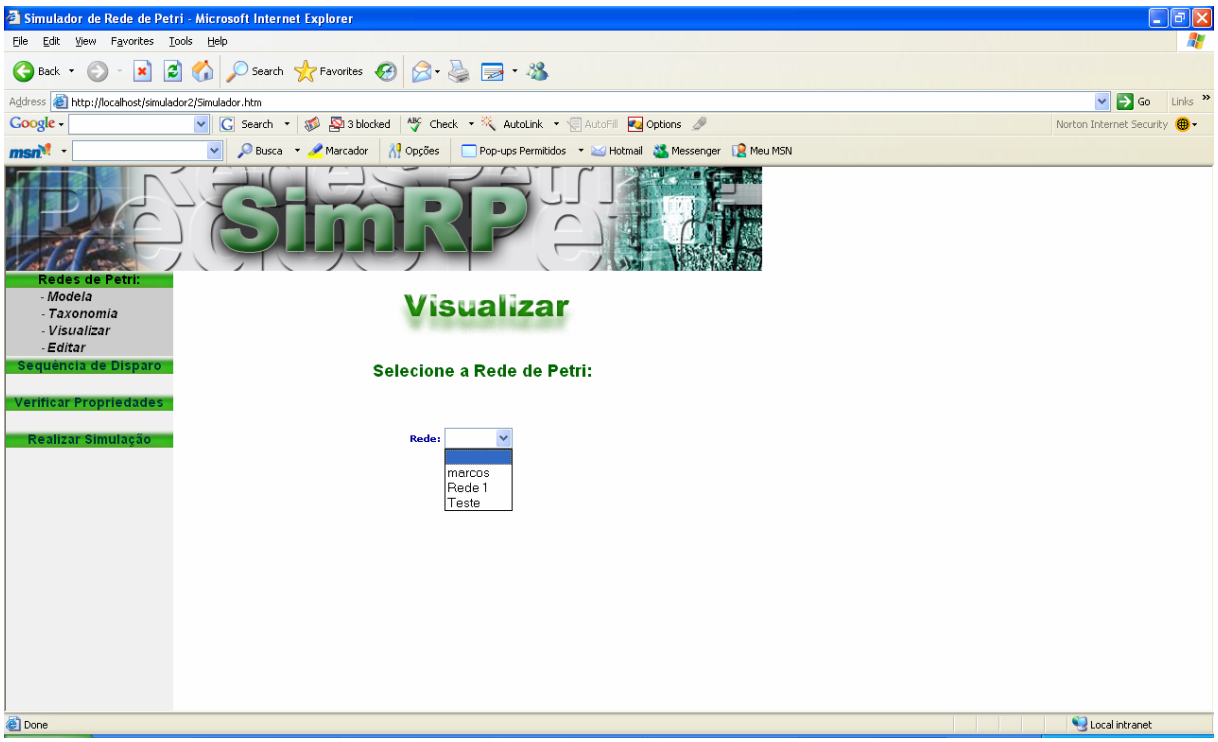


Figura 11 - Tela de Seleção

A figura 12 apresenta o modelamento de uma Rede de Petri, obtida através da opção Visualizar.

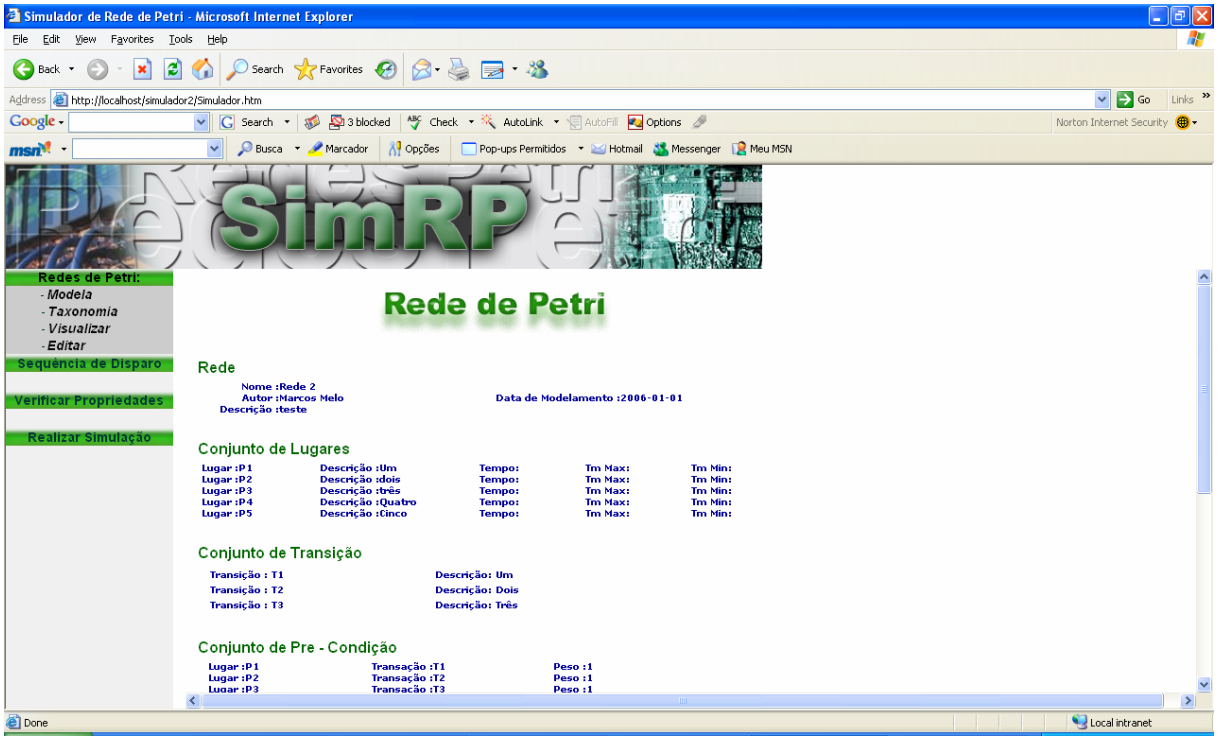


Figura 12 - Tela visualizar Redes de Petri

A figura 13 demonstra a tela de edição do modelamento de uma Rede de Petri, nesta opção é possível editar ou excluir os modelamentos.

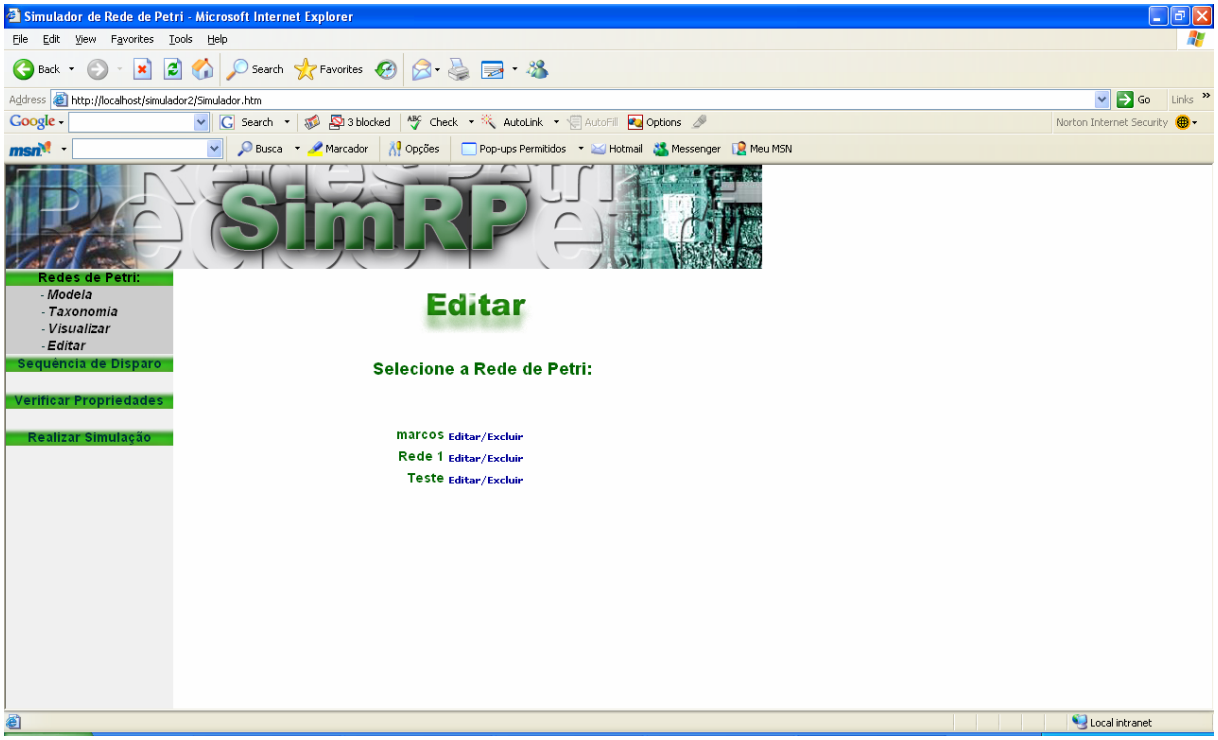


Figura 13 - Tela de edição de rede

Para as Redes de Petri Ordinárias, é possível definir uma seqüência de disparo como demonstrado na figura 14.

Figura 14 - Tela de edição de seqüência de disparo

O simulador permite a verificação das propriedades da Rede de Petri, como demonstrado na figura 15, quando uma propriedade é verificada é atribuída uma cor verde ao retângulo correspondente a propriedade, os conflitos e deadlocks, também podem ser observados nela funcionalidade.

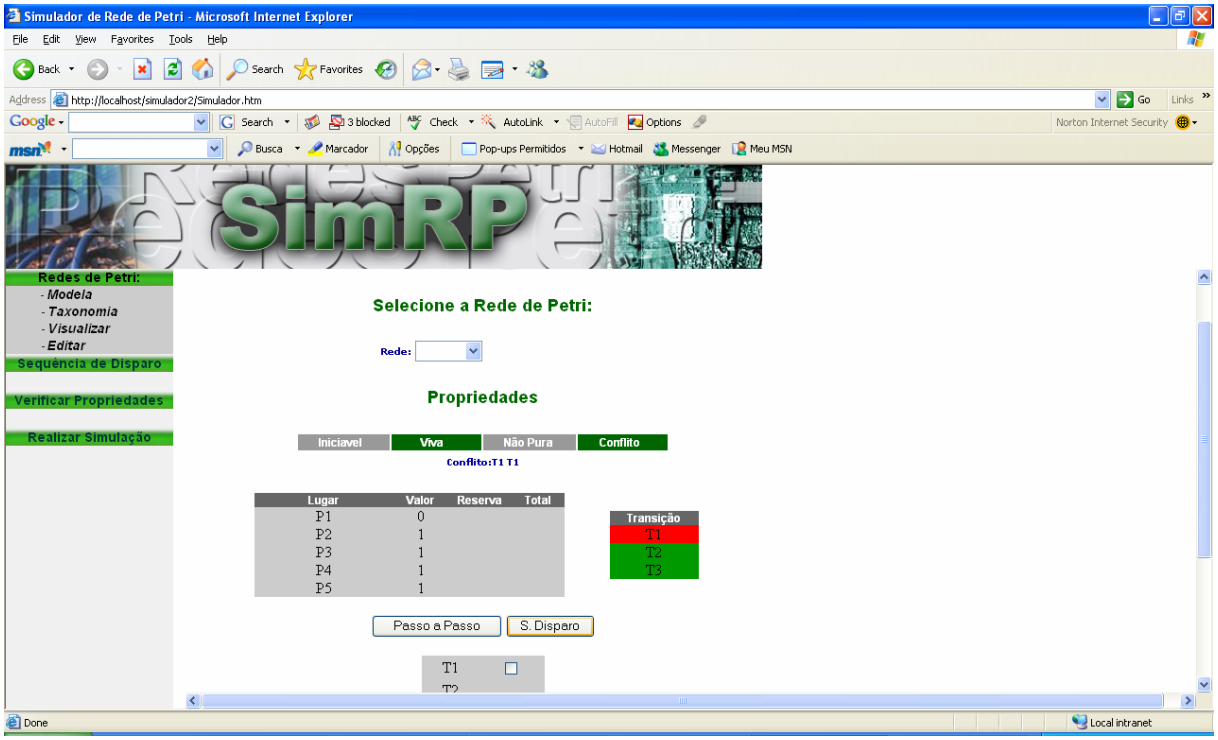


Figura 15 – Tela de verificação de propriedade

A simulação de modelamento, vai depender da sua taxonomia, no caso das Redes de Petri Ordinárias existe a opção seqüência de disparo e Passo a passo, no primeiro caso a simulação seguirá a seqüência definida na opção Seqüência de Disparo, como demonstrado na figura 16.

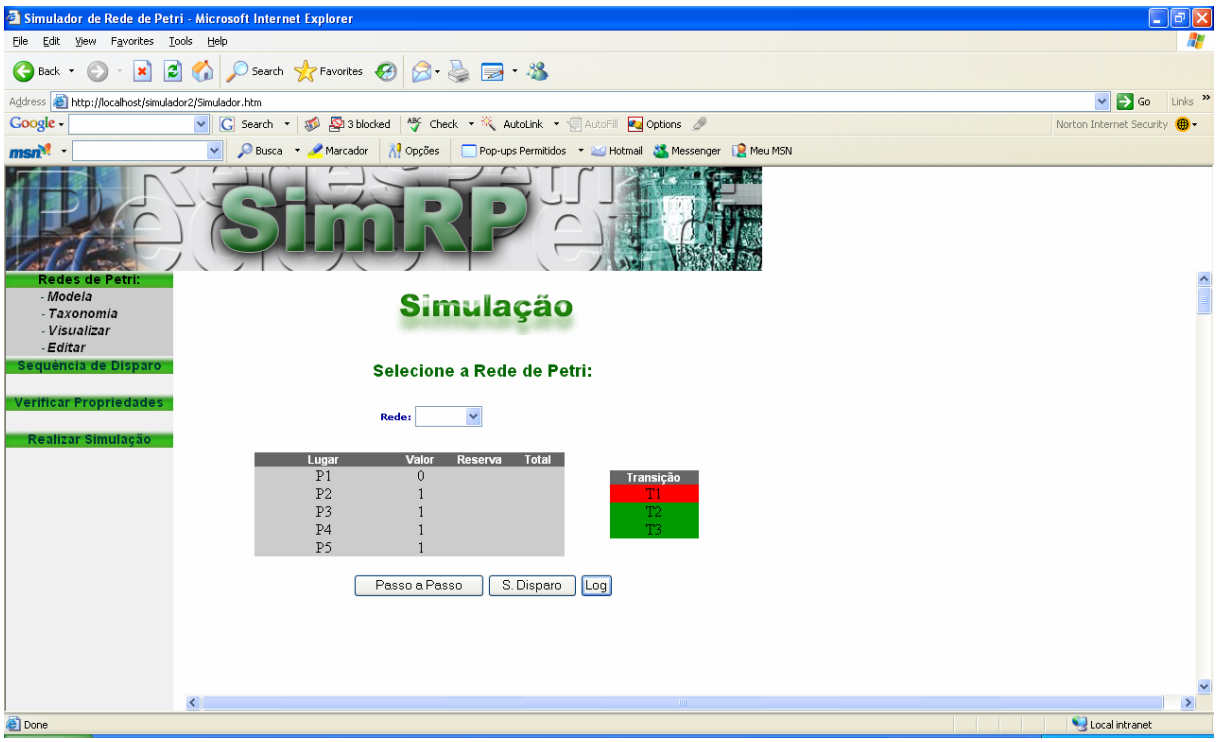


Figura 16 - Tela de simulação

No caso da simulação Passo a passo, o sistema irá apresentar as transições sensibilizadas naquele momento e solicitará que o usuário defina qual transição será utilizada, como demonstrada na figura 17.

Na simulação também está disponível a opção de Log, onde é possível verificar o comportamento anterior da Rede de Petri.

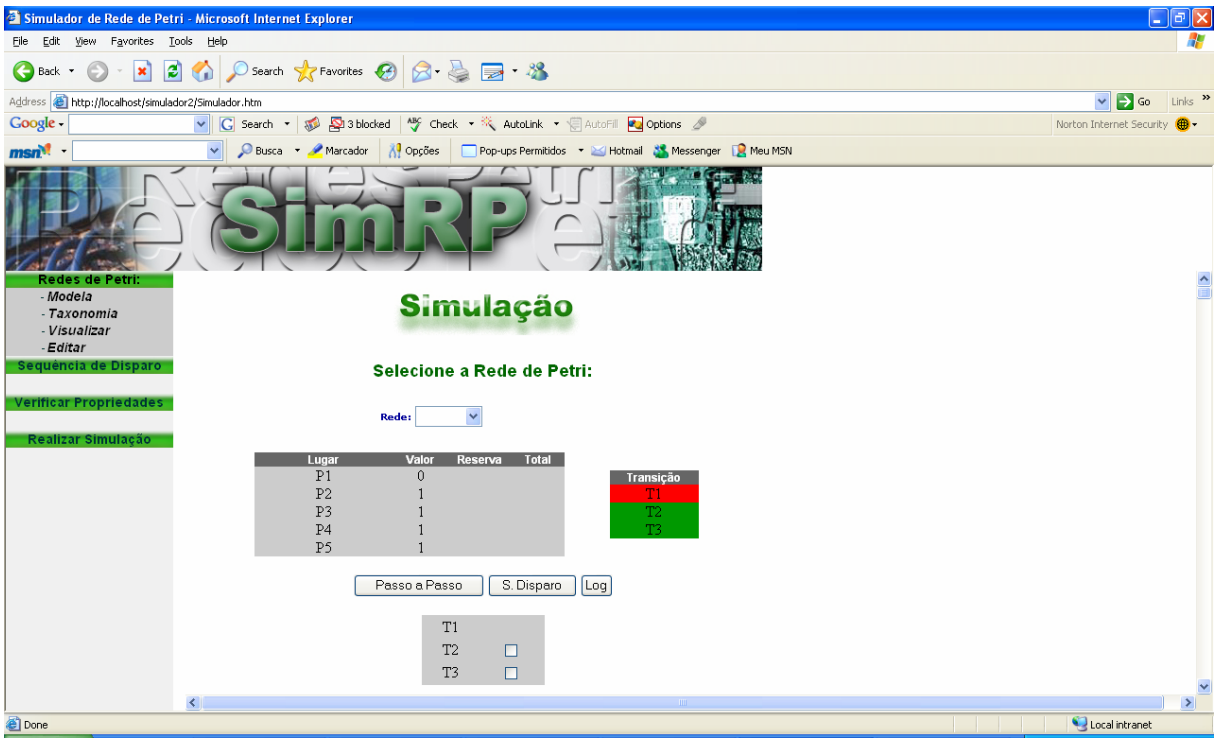


Figura 17 - Tela de simulação passo a passo

Na simulação das taxonomias Interpretadas, Temporal e Temporizadas, somente está disponível a opção de simulação contínua como demonstrado na figura 18, No caso específico da simulação das Redes Interpretadas, o valor das variáveis associadas às transição podem ser alteradas em tempo de execução, nesta opção também é possível visualizar o código VHDL gerado a partir desta simulação. Na figura 19 é apresentado o código fonte gerado a partir deste modelamento.

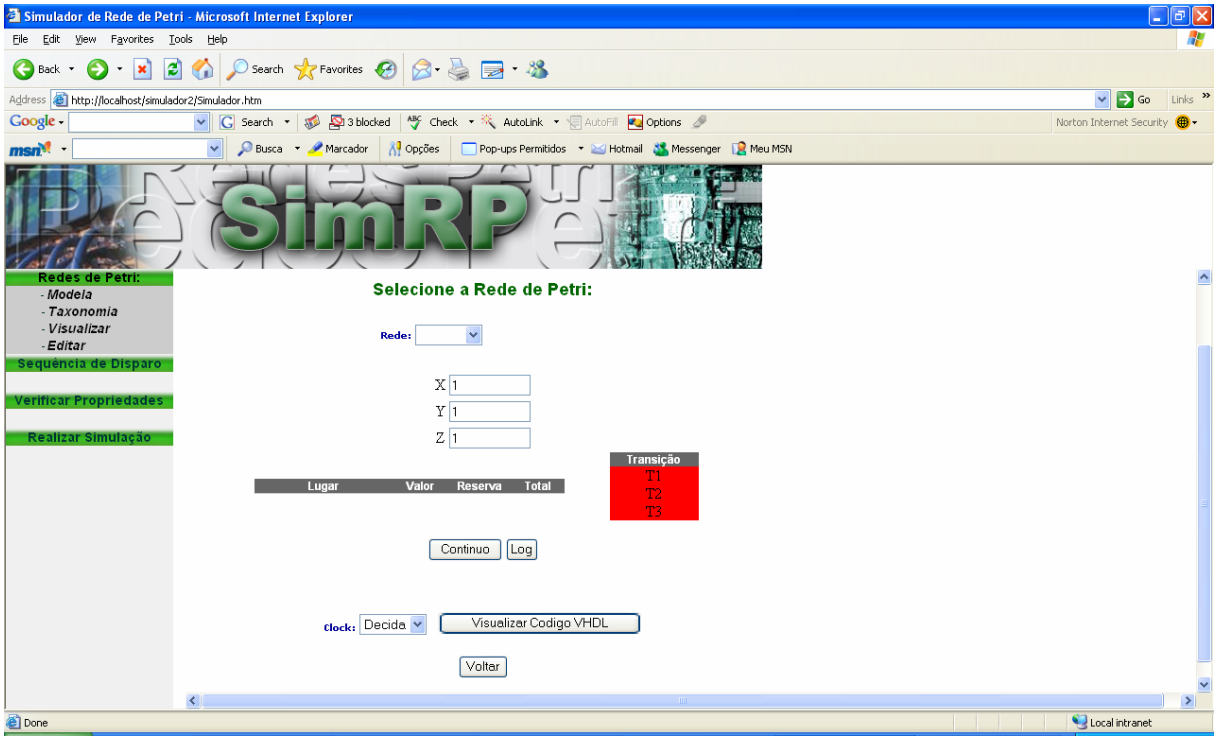


Figura 18 - Tela de simulação rede interpretada

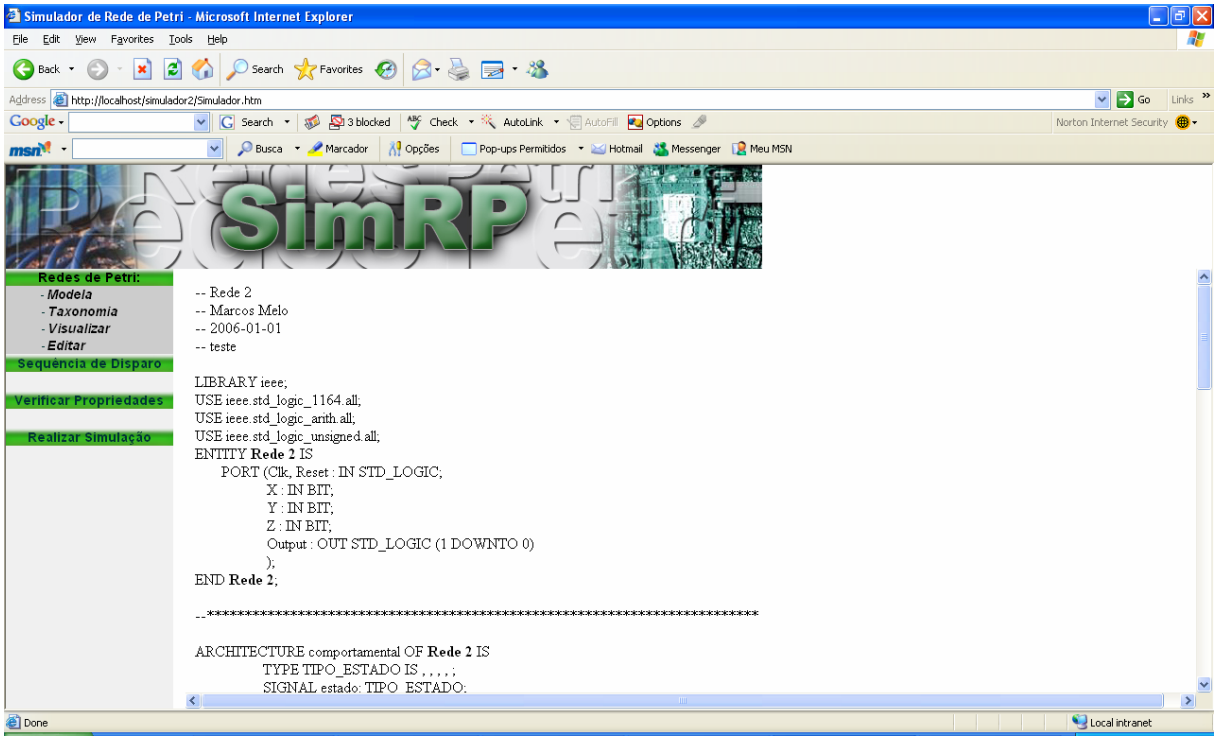


Figura 19 – Tela de geração de código VHDL

Apêndice 5 – Código VHDL gerado pelo SimRP

```
-- Portão
-- Marcos Melo
-- 2006-01-01
-- teste de RP interpretada

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;
ENTITY Portao IS
     PORT (Clk, Reset : IN STD_LOGIC;
               SOEA : IN BIT;
               SOEF : IN BIT;
               SOIA : IN BIT;
               SOIF : IN BIT;

                 Output : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
               );
END Portao;

--*********************************************************************

ARCHITECTURE comportamental OF Portao IS
            TYPE TIPO_ESTADO IS (LDLDO, MEA, MEF, MIA, MIF);
            SIGNAL estado: TIPO_ESTADO;
BEGIN
          PROCESS (Reset, Clk)
          BEGIN
                    IF Reset  =  '1'  THEN
                         estado <= LDLDO ;
                    -- ELSIF Clk 'EVENT THEN
                           ELSIF Clk' EVENT AND clk = '1' THEN
```

```
                    CASE estado IS


                        WHEN LDLDO =>
                                IF SOEA = '1' THEN
                                        estado <= MEA;
                        END IF;

                        WHEN MEA =>
                                IF SOEF = '1' THEN
                                        estado <= MEF;
                        END IF;

                        WHEN MEF =>
                                IF SOIA = '1' THEN
                                        estado <= MIA;
                        END IF;

                        WHEN MIA =>
                                IF SOIF = '1' THEN
                                        estado <= MIF;
                        END IF;

                         WHEN OTHERS =>
                                estado <= LDLDO;
                    END CASE;
                END IF;
        END PROCESS;
--*************************************************************************
--parte combinacional para as saidas
--*************************************************************************
PROCESS (estado, SOEA, SOEF, SOIA, SOIF)
        BEGIN
                CASE estado IS
```

```
                    WHEN LDLDO =>
                        output <= "001";
                    WHEN MEA =>
                        output <= "010";
                    WHEN MEF =>
                        output <= "011";
                    WHEN MIA =>
                        output <= "100";
                    WHEN MIF =>
                        output <= "101";

                END CASE;
        END PROCESS;
END comportamental;
```