Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica
Departamento de Telemática

MODELAGEM E SIMULAÇÃO DE PROTOCOLOS DE COMUNICAÇÃO

MATEUS CONRAD BARCELLOS DA COSTA
Bacharel em Ciências da Computação

Orient: Prof. Dr. WALTER DA CUNHA BORELLI

Tese apresentada à Faculdade de Engenharia Elétrica - FEE/UNICAMP como parte dos requisitos exigidos para a obtenção do título de *Mestre em Engenharia Elétrica.*

Campinas, 18 de dezembro de 1995.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

C823m

Costa, Mateus Conrad Barcellos da
Modelagem e simulação de protocolos de comunicação / Mateus Conrad Barcellos da Costa.-- Campinas, SP: [s.n.], 1995.

Orientador: Walter da Cunha Borelli.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Redes de petri. 2. Redes de computação - Protocolos. 3. Simulação (Computadores). 4. Sistemas de parâmetros distribuidos. I. Borelli, Walter da Cunha. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. III. Título.

AGRADECIMENTOS

Agradeço a todas as pessoas e instituições que contribuíram para a realização deste trabalho. Sobretudo,

Ao *DT/FEE* e à *UNICAMP* pela excelente infraestrutura oferecida, à *FAPESP* (pelo projeto temático) e ao *CNPq* pelo apoio financeiro.

Ao meu orientador *Prof. Dr. Walter da Cunha Borelli.*

À banca examinadora, *Dr. Luis Carlos Trevelin, Dr. Maurício Magalhães e Dr. Shusaburo Motoyama.*

Aos colegas e amigos *Alencar, Fábio, Marcelo Segatto, Nádia, Thales, Dinalva, Flávia, Nestor e Denise.*

À toda a minha família, sobretudo à minha mãe *Edna*, meu falecido pai *Euzébio*, e meu querido irmão *Maucrice.*

À *Luciana*, para quem eu dedico este trabalho.

RESUMO

Com o intuito de cooperar para o desenvolvimento de ferramentas de auxílio ao projeto de sistemas concorrentes e distribuídos, particularmente sistemas de telecomunicações, esta dissertação apresenta a proposta de um simulador para a verificação por acompanhamento de sistemas computacionais baseados em processos comunicantes.

O simulador, baseado em modelos de estados e transições, utiliza como estrutura interna de simulação um modelo de rede de Petri de alto Nível. O modelo de rede Petri utilizado para a estrutura do simulador incorpora características das redes de Petri de Predicados e Transições (P/T nets) e das redes de Petri Numéricas.

A aplicação do simulador está particularmente voltada para a verificação de especificações SDL, o que implicou no estabelecimento de regras e esquemas de modelagem para a transformação das especificações SDL para o modelo de RP proposto e utilizado como base para implementação do simulador.

ABTRACT

In order to cooperate to the development of computer aid tools for the design of concurrent/distributed systems (e.g., telecommunication systems), this dissertation presents a project of a simulator for trace verification of communicating process based systems.

The simulator, based on the State-Transition model, uses as its internal structure a suggested model of high level Petri net derived from the P/T and Numerical Petri nets models. Its aplication is more concerned to the simulation of SDL specifications and a set of tranform rules to the modelling of SDL specifications to the Petri Net model, were established.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

A construção de sistemas computacionais cujas atividades são executadas concorrentemente é tarefa bastante complexa. Especificar um sistema com essa característica (uma central de comutação telefônica, por exemplo) e que atenda as exigências de mercado, pode vir a ser um enorme transtorno, caso projetistas não façam uso de linguagens de especificação formais. Estas linguagens tém a função de padronizar, facilitem e estabelecer um maior grau de consistência ao desenvolvimento de sistemas concorrentes.

O uso de linguagens de especificação formais para o projeto (*design*) de sistemas concorrentes, tem crescido acentuadamente. Estas linguagens não só proporcionam um grau maior de confiabilidade (eliminação de ambiguidades) que linguagens naturais, como também agilizam o processo de desenvolvimento, incorporando as especificações características como modularidade, reusabilidade e encapsulamento.

Entretanto, mesmo fazendo uso de linguagens de especificação formais, a tarefa de se projetar sistemas concorrentes de grande e médio porte continua sujeita a uma quantidade considerável de erros, tanto de concepção quanto de especificação, além de erros do escopo sintático e semântico da linguagem. Ferramentas de auxílio ao desenvolvimento de sistemas que utilizam essas linguagens, em geral, possuem recursos de análise do código fonte que detectam erros tanto de sintaxe como de semântica das especificações produzidas. Contudo, os erros contextuais cometidos nestas especificações são difíceis de se detectar e faceis de serem gerados.

Geralmente, as únicas ferramentas disponíveis para o rastreamento de erros deste tipo são simuladores. Com eles é possível acompanhar um pseudo funcionamento do sistema especificado, verificar detalhes de seu comportamento dinâmico e testar hipóteses que só seriam verificadas com o sistema já implementado.

Essas possibilidades que simuladores oferecem os tornam importantes par-

ceiros para projetistas de sistemas de grande complexidade como é o caso dos sistemas de telecomunicações atuais. Nosso objetivo central neste trabalho de dissertação, é desenvolver uma ferramenta de simulação que permita a verificação por acompanhamento de especificações formais. Nossa proposta é a apresentação do protótipo para a construção de um simulador de sistemas orientados a processos comunicantes voltado para a simulação de especificações de sistemas elaboradas com a linguagem SDL[1], linguagem formal para especificação de sistemas de comutação.

A simulação de especificações SDL pode ser feita com o uso de três técnicas diferentes a seguir:

1. Traduzindo a especificação para um programa em linguagem de programação como C ou Pascal, e gerando um programa executável que represente o comportamento da especificação.

2. Construindo uma máquina de processamento simbólico para a própria linguagem SDL.

3. Transformar a especificação SDL em um modelo de estados e transições e construir uma máquina que processe o modelo produzido pela transformação.

Dentre estas três possibilidades, a utilização de um modelo de estados e transições intermediando a especificação e a simulação da mesma foi a mais conveniente. Isto decorreu da decisão pela utilização de Rede de Petri[23] como modelo de estados e transições. Esta ferramenta apresenta facilidades para modelagem de sistemas descritos em SDL e grande potencial para análise e simulação de sistemas modelados.

Para analisarmos um sistema concorrente, partimos de sua especificação formal, feita na linguagem SDL, estabelecemos a utilização de redes de Petri como ferramenta de modelagem, estabelecemos regras de transformação do sistema original para o modelo em redes de Petri e, por fim, desenvolvemos um protótipo de um programa simulador de sistemas modelados com as regras estabelecidas.

1.1 Modelagem de Especificações Formais

Técnicas de verificação de sistemas concorrentes são geralemente baseadas em modelagem. Aplicando um método de modelagem, podemos extrair do

[1]SDL: Specification and Description Language. Linguagem desenvolvida e padronizada pelo ITU (antigo CCITT). Neste trabalho estaremos nos referindo a recomendação Z.100 de 1988.

sistema original uma abstração que focalize as partes do sistema das quais desejamos obter informações. A partir de então, podemos tecer formas apropriadas de análise do sistema no foco da abstração.

Para obtermos este modelo abstrato, geralmente fazemos uso de ferramentas de modelagem matemáticas. Nestas ferramentas, esperamos encontrar propriedades as quais possamos associar às propriedades comportamentais e estruturais do sistema modelado.

Tanto SDL quanto redes de Petri incorporam o modelo de estados e transições. SDL, como linguagem de especificação propriamente dita, facilita a especificação e a descrição de sistemas por pessoas. No entanto, o objetivo da linguagem é representar sistemas estaticamente, isto é, o que é descrito em SDL, são os programas cujas instâncias serão os processos do sistema. Isto dificulta a análise de propriedades dinâmicas do mesmo. Por outro lado, a própria definição de redes de Petri está atrelada a uma determinada dinâmica de execução, o que possibilita uma análise tanto estrutural como comportamental de sistemas de uma maneira eficiente.

O processo de verificação de corretude de sistemas concorrentes que estamos interessados, envolve frequentemente certo tipo de raciocínio sobre o espaço de estados do sistema. Por exemplo, provar que um sistema está livre de impasses (*dead-lock free*) requer que mostremos que não é possível alcançar, partindo o sistema de um estado inicial, algum estado no qual nenhuma transição, para um outro estado, possa ser executada. Raciocínio este simples, todavia muito frequentemente impraticável devido a explosão combinatorial do espaço de estados de sistemas concorrentes. Até mesmo um sistema concorrente simples pode gerar muitas centenas ou milhares de estados. Além disso, com o aumento linear do grau de concorrência do sistema, rapidamente o número de estados torna-se absurdamente grande.

O número de estados diferentes alcançáveis por um sistema concorrente representa um número equivalente de situações em que o sistema pode se encontrar em um dado momento de seu funcionamento. O grande número de situações possíveis para estes sistemas tornam meio cegos, por assim dizer, os projetistas que não dispoem de uma ferramenta de simulação e de verificação que os auxilie. Para verificarmos isto, basta que tentemos rastrear manualmente um pequeno programa concorrente com três ou quatro processos independentes e comunicantes.

Quando se fala em verificação de corretude de sistemas utilizando modelagem, deve-se ter em mente a abrangência limitada da verificação de um sistema feita a partir de uma abstração do mesmo. Certamente estamos interessados em um nível de verificação ou na verificação de alguns aspectos do sistema. Este trabalho está voltado para a verificação de corretude dos aspectos computacionais dos sistemas desenvolvidos. Isto é, estamos inte-

ressados em avaliar o comportamento interno dos processos e a comunicação realizada entre os mesmos.

Esse escopo de verificação tem se tornado cada dia mais importante, haja visto que os protocolos da camada de aplicação de redes têm se tornado cada vez mais sofisticados em função das possibilidades oferecidas pelo uso de sistemas de comunicação digitais que propiciam altas velocidades de transmissão de dados com baixíssimas taxas de erros.

1.2 Modelos de Estados e Transições

A utilização do modelo de máquinas de estados e transições, para o desenvolvimento do simulador foi uma escolha feita pelo fato de que as especificações SDL também incorporam este paradigma. Este é um importante aspecto para a construção de um simulador, haja visto que a ferramenta de simulação deve refletir da melhor maneira possível o comportamento do sistema que está sendo simulado.

Uma máquina de estados e transições modela os possíveis estados do sistema. Em cada estado, existe um conjunto limitado de unidades de informação válidas que podem ser recebidas ou transmitidas. Comparando-se a unidade de informação recebida, quando o sistema se encontra em um determinado estado, com as possíveis entradas deste estado, é feito um chaveamento e uma função de transição específica é determinada e o sistema atinge um outro estado. O modelo de máquina de estados e transições é relativamente simples e fácil de desenvolver, embora apresente o problema de explosão combinatorial de estados.

A ferramenta de modelagem utilizada como base para a construção do simulador é um tipo de rede de Petri. Redes de Petri incorporam ao modelo de estados e transições, um grande poder de descrição, que torna o modelo mais próximo da especificação formal. Além disso, redes de Petri possuem propriedades bem definidas com paralelos práticos no comportamento do sistema modelados e servem como forma de avaliação do mesmo.

1.3 Projeto Temático "Redes Metropolitanas Multi Serviços"

Este trabalho foi originado no projeto temático FAPESP intitulado "Redes Metropolitanas Multi-Serviços"[2] .

[2]Projeto Redes Metropolitanas Multi Serviços: Proc. FAPESP 91/3660-0, setembro de 1992 a setembro de 1994, desenvolvido no Departamento de Telemática / UNICAMP/FEE

O projeto foi motivado pela possibilidade de diminuição de custos de implementação destas redes através do uso de técnicas de comutação distribuída, levando parte da central multi-serviços para perto dos assinantes. O projeto também previu a utilização de uma série de políticas de estruturação dentre as quais podemos destacar:

- Utilização de técnicas modernas de comutação;
- meios de transmissão de faixa larga como fibras ópticas;
- técnicas de telefonia celular entre os assinantes e os nós de comutação da rede;
- regras de encaminhamento não convencionais e utilização de protocolos específicos a serem definidos.

As atividades de pesquisa previstas no projeto foram:

A. Especificação, validação e simulação de protocolos;

B. Esquemas de acesso;

C. Encaminhamento de tráfego: Análise de desempenho;

D. Acesso aos nós de comutação via rádio: técnicas de roteamento alternativo;

E. Acesso aos nós de comutação via rádio: interferência de co-canais pela aplicação de técnicas de roteamento alternativo.

1.4 Organização da Dissertação

Um dos focos principais do desenvolvimento desta dissertação é o estudo de Redes de Petri, feito de maneira direcionada para a necessidades de modelagem e simulação. A teoria de redes de Petri, bem como a definição do modelo proposto formam o tema do capítulo 2.

O próximo ponto a ser visto é a forma como especificações SDL são transformadas para o modelo de redes de Petri. Este é o tema do capítulo 3. No capítulo 4 é apresentada a estrutura do SIM, programa simulador de sistemas concorrentes baseado em redes de Petri, bem como os aspectos relacionados a construção de programas simuladores.

No capítulo 5 é apresentada a modelagem e simulação de uma central de comutação telefônica previamente especificada em SDL. São também estabelecidas comparações entre simulações baseadas no esquema de modelagem

proposto e simulações baseadas na especificação formal propriamente dita. O capítulo de conclusão faz um apanhado geral dos objetivos previstos e os alcançados neste trabalho.

Capítulo 2

Redes de Petri

2.1 Introdução

Rede de Petri[23] é uma ferramenta matemática para a modelagem de sistemas, com grande potencialidade para a descrição de sistemas de processamento de informação caracterizados como concorrentes, assíncronos, paralelos, distribuídos, não determinísticos e/ou estocásticos. Pelo seu forte apelo gráfico, a ferramenta pode também ser usada em substituição a diagramas de fluxo de controle e de dados.

A teoria de redes de Petri visa desenvolver técnicas que nos permitam:

- *Criar modelos de sistemas utilizando redes de Petri.* O que siginifica que teremos uma representação matemática e uma abstração funcional do mesmo;

- *Analisar sistemas modelados como redes de Petri.* Análises destas redes de Petri podem, a partir de modelos, revelar importantes informações sobre a estrutura e o comportamento dinâmico dos mesmos.

Redes de Petri de uma maneira geral e sobretudo aquelas chamadas redes de Petri de alto nível fornecem uma eficiente e expressiva estrutura para a concepção de simuladores de sistemas concorrentes. No que diz respeito a protocolos de comunicação ela se torna ainda mais atrativa. Isto se deve principalmente ao importante uso de modelos baseados em máquinas de estados em especificações de protocolos e serviços para sistemas distribuídos, inclusive para padronizações ISO e do ITU (anteriormente CCITT).

A aplicação de redes de Petri está diretamente ligada a modelagem. Em muitas áreas de estudo, um fenômeno não é estudado diretamente, mas sim através de um modelo do objeto ou sistema em questão. Um modelo é uma representação daquilo que definimos como características importantes do

objeto de estudo. Quando estamos utilizando esta representação, temos o desejo de obter novos conhecimentos a respeito do sistema modelado sem que para isso tenhamos de correr riscos, perder tempo ou dinheiro, e ainda livrarmo-nos de algum outro inconveniente da manipulação do próprio sistema.

Especificações SDL fornecem uma visão estática do sistema. Com o modelo de rede de Petri extraido das especificações é possível obter conhecimentos estruturais de especificações e, principalmente, conhecimentos a respeito do comportamento dinâmico de sistemas especificados em SDL, tanto através da análise de propriedades como através de simulação.

Este capítulo primeiramente introduz os conceitos básicos da teoria de redes de Petri. Em seguida, são apresentados tipos de rede de Petri com maiores facilidades de modelagem. Por fim é apresentada a estrutura de rede de Petri utilizada na implementação do simulador.

2.1.1 Desenvolvimentos Iniciais

O aparecimento de redes de Petri se deu com o trabalho de doutorado de Carl Adam Petri, submetido em 1962 à faculdade de Matemática e Física da Universidade Técnica de Darmstadt, Alemanha. Em seguida ao trabalho de C. A. Petri, aplicações da Teoria de redes de Petri aparecem em [10]. A partir destes trabalhos iniciais, uma série aparentemente inesgotável de trabalhos ligados às redes de Petri tem sido produzida. Uma razão para isso é o vasto espectro de aplicação dessas redes.

Dentre as aplicações de redes de Petri[1] duas áreas de grande sucesso são a avaliação de desempenho e protocolos de comunicação. Uma característica do estudo de redes de Petri e verificação de suas aplicações é que este sempre depende de uma ferramenta computacional de auxílio associada.

[1] Atualmente podem ser encontrados trabalhos que utilizam redes de Petri nas áreas de: sistemas de software distribúidos, bancos de dados distribuídos, programas paralelos e distribuídos, sistemas de tempo real, linhas de produção flexíveis e sistemas de controle industrial, sistemas de eventos discretos, sistemas de memória multiprocessada, computação baseada em fluxo de dados, sistemas de tolerância a falhas, lógica programável e vetores VLSI, circuitos assíncronos, compiladores e sistemas operacionais, linguagens formais, programação lógica, redes locais, sistemas de legislação, fatores humanos, redes neurais, filtros digitais e modelos de decisão.

2.2 Definições Básicas

2.2.1 Estrutura de uma Rede de Petri

Uma rede de Petri é composta de quatro partes: um conjunto de lugares P, um conjuntos de transições T, uma função de entrada I e uma função de saída O. As funções de entrada e saída relacionam lugares e transições. A função de entrada I é um mapeamento de uma transição t_j à uma coleção de lugares $I(t_j)$, conhecidos como lugares de entrada de t_j. A função de saída O mapeia uma transição t_j à uma coleção de lugares $O(t_j)$, conhecidos como lugares de saída de t_j.

As entradas e saídas de uma transição são *bags* de lugares. *Bag* é uma generalização de conjunto que permite múltiplas ocorrências de um elemento em uma mesma *bag*. O uso de bags ao invés de conjuntos para entradas e saídas de transições permite que um lugar seja uma múltipla entrada ou uma múltipla saída de uma mesma transição.

Definição 2.1: *Uma estrutura de rede de Petri C é uma $4-tupla$, $C = (P, T, I, O)$, onde $P = \{p_1, p_2, \ldots, p_n\}$ é um conjunto finito de lugares, $n \geq 0$. $T = \{t_1, t_2, \ldots, t_m\}$ é um conjunto finito de transições, $m \geq 0$. O conjuntos de lugares e o conjunto de transições são disjuntos, $P \cap T = \phi$. $I : T \rightarrow P$ é uma função de entrada, um mapeamento de transições à "bags" de lugares. $O : T \rightarrow P$ é a função de saída, um mapeamento de transições a "bags" de lugares. A cardinalidade do conjunto P é n, e a cardinalidade do conjunto T é m. Um lugar p_i é um lugar de entrada de uma transição t_j se $p_i \in I(t_j)$. Um lugar p_i é um lugar de saída de uma transição t_j se $p_i \in O(t_j)$.*

As funções de entrada e saída podem ser estendidas para o mapeamento de lugares a *bags* de transições. Define-se que uma transição t_j é uma entrada de um lugar p_i se p_i é um lugar de saída de t_j. Uma transição t_j é uma saída de um lugar p_i se este for uma entrada da transição t_j.

2.2.2 Grafo de Rede de Petri

O desenvolvimento de trabalhos em redes de Petri é realizado geralmente com base na definição 2.1. Contudo, a representação gráfica desta estrutura é a principal forma de representação utilizada para ilustrar os conceitos da teoria.

Um grafo de rede de Petri é uma representação de sua estrutura como um multigrafo direcionado e bipartido. Correspondendo a definição da estrutura da rede, o grafo de rede de Petri possui dois tipos de nós. Estes tipos de nós são: círculos, que representam os lugares da rede, e barras, que representam

as transições. Arcos direcionados conectam lugares e transições. Arcos que se direcionam de um lugar para uma transição definem o lugar como sendo um lugar de entrada da transição. Múltiplas entradas de uma transição são indicados por múltiplos arcos que partem de lugares e chegam á transição. Um lugar de saída é indicado por arcos partindo de uma transição e chegando no lugar. Da mesma forma, múltiplas saídas de uma transição são indicadas por múltiplos arcos partindo da transição e chegando aos lugares de saída da mesma.

Uma rede de Petri é um multi grafo por este pemitir múltiplos arcos de um nó para outro. Em adição, desde que os arcos são direcionados, o grafo é dito direcionado. Considerando-se ainda que seus nós são divididos em dois conjuntos distintos, e que os arcos unem sempre um elemento de um conjunto a outro de outro conjunto, dizemos que o grafo é bipartido. Assim, a denominação multigrafo direcionado bipartido é justa.

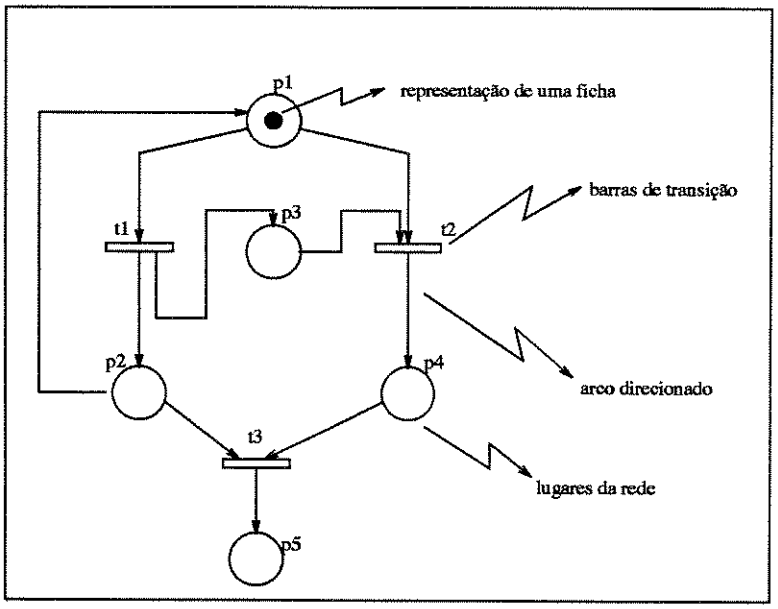

Figura 2.1: Exemplo de grafo de rede de Petri

2.2.3 Marcação de uma Rede de Petri

Uma marcação μ é uma associação de fichas (*tokens*) a lugares da rede. Uma ficha, tal como lugares e transições, é um conceito primitivo para redes de Petri. Estas fichas, pode-se dizer, residem dentro dos lugares da rede.

A quantidade de fichas e a posição das mesmas com relação aos lugares pode mudar durante a execução da rede de Petri. As fichas são usadas para

definir a execução de uma rede de Petri, ou seja, para demonstrar a dinâmica dos sistemas modelados por estruturas de rede de Petri.

Definição 2.2: *Uma marcação μ de uma rede de Petri $C = (P, T, I, O)$ é uma função que relaciona o conjunto de lugares P a inteiros não negativos, ou seja, $\mu : P \rightarrow N$.*

A marcação μ pode também ser definida como um n-vetor $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$ onde $n = \mid P \mid$ e cada $\mu_i \in N$. O vetor μ fornece para cada lugar p_i o número de fichas que este contém. Uma estrutura $M = (P, T, I, O, \mu)$ é chamada de rede de Petri Marcada. Em um grafo de rede de Petri, fichas são representadas por pequenos círculos preenchidos e posicionados dentro dos lugares da rede. Alguns autores consideram a marcação inicial como sendo parte da própria definição da estrutura de uma rede de Petri. Além desta consideração, pode-se encontrar na literatura a definição de redes de Petri como sendo um grafo na sua concepção mais primitiva. Assim, uma definição alternativa para redes de Petri pode ser a seguinte:

Definição 2.3:

Uma rede de Petri é uma $5 - upla, PN = (P, T, F, W, M_0)$, *onde:*
$P = \{p_1, p_2, \ldots p_m\}$, *é um conjunto finito de lugares;*
$T = \{t_1, t_2, \ldots t_n\}$, *é um conjunto finito de transições;*
$F \subseteq (P \times T) \cup (T \times P)$ *é um conjunto de arcos;*
$W : F \rightarrow \{1, 2, 3, \ldots\}$ *é uma função de peso dos arcos;*
$M_0 : P \rightarrow \{0, 1, 2, 3, \ldots\}$ *é uma marcação inicial;*
$P \cap T = \phi$ *e* $P \cup T \neq \phi$.

A função de peso dos arcos elimina a necessidade de termos mais de um arco de mesmo sentido ligando dois nós da rede. Assim, os lugares de entrada e de saída de uma transição podem ser definidos como conjuntos e não mais como "*bags*". Um arco que parte de um lugar p_i e chega a uma transição t_j, cujo peso é 3, tem o mesmo significado de três arcos partindo de p_i e chegando em t_j , quando não se considera a função peso.

Espaço de Estados de uma rede de Petri

Um estado e uma marcação em uma rede de Petri são maneiras diferentes de dizermos a mesma coisa. Pelo menos em redes de Petri clássicas. O disparo de uma transição acarreta uma mudança de estado de uma rede de Petri se o mesmo acarretou uma mudança na marcação da rede.

O *Espaço de Estados* de uma RP é o conjunto de todas as suas marcações. A mudança de estado provocada pelo disparo de uma transição é definida pela função δ chamada de função de próximo estado. Duas sequências resultam da execução de uma rede de Petri: a *sequência de marcações* e a *sequência de transições*.

Em uma rede de Petri de alto nível o estado não corresponde exatamente a definição de marcação. Nestas redes, um estado da rede é definido com relação a todos os seus componentes e não só com relação à presença de fichas nos lugares da rede.

2.2.4 Regras de Habilitação e Disparo

A execução de um rede de Petri é controlada pelo número e distribuição das fichas na mesma. Uma rede de Petri é executada através do disparo de transições. O disparo de uma transição implica na remoção de fichas de seus lugares de entrada e criação de novas fichas que são distribuídas nos seus lugares de saída. Uma transição só poderá disparar se esta estiver habilitada. Uma transição t_j está habilitada se cada um dos seus lugares de entrada possuir fichas em número maior ou igual ao número de arcos que os liga à transição t_j. Múltiplas fichas são necessárias para múltiplos arcos. Isto é equivalente a dizer que o número de fichas presentes em cada lugar de entrada de uma transição t_j deve ser maior ou igual ao peso do arco que liga o lugar à transição t_j para que esta fique habilitada.

Considerando a definição 2.3, para simular o comportamento dinâmico dos sistemas modelados, a marcação (estado) da rede é modificada de acordo com as seguintes regras de disparo:

1. uma transição é dita habilitada se cada lugar de entrada possui pelo menos $w(p,t)$ fichas, onde $w(p,t)$ é o peso do arco que liga o lugar p à transição t;

2. Estando habilitada, uma transição pode ou não disparar;

3. Um disparo de uma transição t remove $w(p,t)$ fichas de cada lugar de entrada de t, e adiciona $w(t,p)$ fichas em cada lugar de saída p da transição t, onde $w(t,p)$ é o peso do arco que liga a transição t ao lugar de saída p.

Segundo o paradigma de eventos e condições, lugares representam condições e transições representam eventos. Uma transição possui um conjunto de lugares de entrada e um conjunto de lugares de saída representando respectivamente, as pré-condições e as pós-condições do evento (transição). As pré-condições são avaliadas segundo a presença de fichas nos lugares de entrada da transição e o peso dos arcos que ligam esses lugares à transição.

Na realidade, lugares de entrada oferecem uma forma de avaliação das pré-condições (alojando fichas). Típicas interpretações de lugares e transições são apresentadas na tabela da figura 2.1:

Lugares de Entrada	Transição	Lugares de saída
Pré-condições	evento	Pós-condições
Dados de entrada	Passo computacional	Dados de Saída
Recursos necessitados	Tarefa	Recursos liberados
Buffers	Processadores	Buffers

Transições sem lugares de entrada são chamadas de *transições fonte* e estão sempre habilitadas. Transições sem lugares de saída são chamadas de *"sink"*. Um par (t, p) é chamado de "self loop" quando p é ao mesmo tempo lugar de entrada e de saída de t. Uma rede que não possui "self loops" é chamada de rede de Petri *Pura*.

Para a regra de habilitação apresentada, é assumido que cada lugar da rede pode acomodar um número ilimitado de fichas. Estas redes de Petri são referenciadas como redes de Petri com capacidade infinita. Para a modelagem de muitos sistemas entretanto, faz-se necessário limitar a capacidade dos lugares a um determinado número de fichas. Redes com essa característica são chamadas de *redes com capacidade finita*.

Para estas redes, uma nova condição de habilitação aparece: para uma transição ficar habilitada, o número de fichas em cada lugar de saída p não deve exceder $k(p) - w(t, p)$, onde $k(p)$ é a capacidade do lugar p e $w(t, p)$ é o peso do arco.

A regra de restrição de capacidade é chamada de *regra de transição estrita*, enquanto a anterior, sem restrição, é chamada simplesmente de regra de transição (fraca). Dada uma rede de capacidade finita (N, M_0), é possível aplicar a regra de transição estrita nesta, ou equivalentemente, a regra de transição fraca à rede (N', M_0') obtida de (N, M_0) pela transformação de lugares complementares, assumindo que a rede é pura. A transformação de lugares complementares é feita pelos seguintes passos:

passo 1: adicionar um lugar complementar para cada lugar p' da rede; a marcação inicial de p' é dada por $M_0(p') = k(p) - M_0(p)$.

passo 2: Entre cada transição t e lugares complementares p', construir novos arcos (t, p') ou (p', t) onde $w(t, p') = w(p, t)$ e $w(p', t) = w(t, p)$, de tal maneira que a soma das fichas nos lugares p e p' é igual à capacidade $k(p)$ para cada lugar p, antes e depois do disparo da transição t.

Teorema: *Seja (N, M_0) uma rede pura de capacidade finita, onde a regra de transição restrita possa ser empregada. Seja (N', M_0') uma rede obtida de (N, M_0) através da transformação de lugares complementares, onde a regra*

de transição fraca pode ser aplicada. Então, estas duas redes são equivalentes no sentido de que elas possuem todos os mesmos conjuntos de sequências de disparo possíveis.

O teorema acima nos permite utilizar apenas a regra de transição fraca, visto que podemos obter sempre uma rede de Petri com capacidade infinita equivalente[2] a uma dada rede de Petri com capacidade finita.

Conceito de sub redes

Em algumas aplicações poderemos estar interessados em marcações de um subconjunto de lugares da rede sem nos preocuparmos com a disposição de fichas em outros lugares. Este subconjunto de lugares e suas respectivas marcações podem ser consideradas em separado do restante da rede e a análise de propriedades pode ser feita com relação às sub-marcações consideradas.

2.3 Análise de redes de Petri

Modelos de redes de Petri podem ser usados para representar o "design" de sistemas concorrentes de uma maneira natural e conveniente. A modelagem pura e simples de um sistema não possui muita utilidade se não for usada posteriormente com algum propósito.

Neste trabalho nosso maior interesse com relação a redes de Petri é delinear o projeto de uma ferramenta de simulação. Entretanto nesta seção, iremos introduzir algumas importantes propriedades de redes de Petri chamadas comportamentais, que como o próprio nome já diz são propriedades que dizem respeito ao processo de execução das redes de Petri.

Propriedades comportamentais de uma rede de Petri são aquelas que dependem da marcação inicial da rede. Estas propriedades, também referenciadas como propriedades dependentes da marcação (*marking-dependent properties*), assumem um papel importante na análise de sistemas concorrentes, visto que as propriedades comportamentais (dinâmicas) de um sistema concorrente não são conhecidas na sua totalidade por seus projetistas.

A seguir serão vistas as principais propriedades de uma rede de Petri.

2.3.1 Segurança (*Safeness*)

Para modelagem de equipamentos de hardware, uma das mais importantes propriedades é a *segurança*. Um lugar de uma rede de Petri é seguro se o

[2]equivalente do ponto de vista do grafo de alcançabilidade

número de fichas que este comporta nunca excede de 1. Uma rede de Petri é segura se todos os seus lugares o são.

Def 2.6: *Um lugar $p_i \in P$ de uma rede $C = (P, T, I, O)$ com marcação inicial μ é segura se para todas as marcações $\mu' \in R(C, \mu), \mu(p_i) \leq 1$. Uma rede de Petri é segura se todos os seus lugares forem seguros.*

Sendo um lugar seguro, o número de fichas neste é sempre 1 ou 0. Um lugar com essa característica pode ser implementado utilizando um flip-flop. Em uma rede de Petri segura uma transição habilitada possui uma ficha em cada um dos seus lugares de entrada e zero fichas em seus lugares de saída, o que motiva a interpretação de lugares como sendo condições. Sob uma definição lógica, uma condição é sempre verdadeira (1) ou falsa (0). Assim, em uma rede de Petri segura, cada lugar pode ser considerado segura sob o ponto de vista de eventos e condições.

Desde que um lugar não seja uma múltipla entrada ou uma múltipla saída de uma transição (e os pesos dos arcos não excedam de 1), é possível força-lo a ser um lugar seguro. Um lugar p_i o qual desejamos tornar seguro, deve ser suplementado por um outro lugar p_i'. As transições ligadas a p_i são modificadas da seguinte maneira:

1. Se $p_i \in I(t_j)$ e $\notin O(t_j)$ então adicionar p_i a $O(t_j)$;
2. Se $p_i \in O(t_j)$ e $\notin I(t_j)$ então adicionar p_i a $I(t_j)$;

O objetivo deste novo lugar é representar a condição "p' está vazio". p_i e p_i' são complementares; p_i possui uma ficha apenas se p_i' não possui uma ficha e vice-versa. Qualquer transição que remova uma ficha do lugar p_i deve depositar uma ficha no lugar p_i' e vice-versa. A marcação inicial deve ser modificada de modo que provenha uma ficha ou em p_i ou em p_i'.

2.3.2 Limitabilidade *(Boundedness)*

Segurança é um caso especial de limitabilidade. Um lugar de uma rede de Petri é k-limitado quando k é o número máximo de fichas que podem ser armazenadas em p_i. Um lugar $p_i \in P$ de uma rede de Petri $C = (P, T, I, O)$ com uma marcação inicial μ é k-limitado se para todas $\mu' \in R(C, \mu)$, $\mu'(p_i) \leq k$. Um lugar limitado a uma ficha é um lugar seguro. Desde que exista um número finito de lugares, podemos tomar um limite máximo k entre todos os limites de cada lugar e definir a rede como k-segura.

2.3.3 Conservação *(Conservation)*

Em uma rede conservativa, fichas não são criadas nem destruídas. A relação de conservação é muito forte, com ela temos imediatamente que

$I((t_j) = O(t_j), j = 1 \ldots n_t$.

2.3.4 Alcançabilidade

Alcançabilidade (*reachability*), é um conceito fundamental para o estudo de propriedades dinâmicas de qualquer sistema. Em redes de Petri este conceito é definido em função do conjunto de marcações que podem ser assumidas por uma rede a partir de uma marcação inicial.

O disparo de uma transição habilitada irá mudar a distribuição de fichas na rede de acordo com as regras de execução estabelecidas na seção 2.2.4. Uma sequência de disparos irá resultar em uma sequência de marcações.

Uma marcação M_n é dita alcançável a partir de uma marcação M_0, se existir uma sequência de disparos que, partindo de M_0, levem a rede à marcação M_n. Esta sequência de disparos será denotada por $\sigma = M_0 t_1 M_1 t_2 M_2 \ldots t_n M_n$ ou apenas $\sigma = t_1 t_2 \ldots t_n$.

O conjunto de todas as possíveis marcações alcançáveis por uma rede de Petri C a partir de M_0 é denotado por $R(C, M_0)$. O conjunto de todas as possíveis sequências de disparo de uma rede de Petri C a partir de uma marcação M_0 é denotada por $L(C, M_0)$. O problema da alcançabilidade para uma rede de Petri resume-se em descobrir se $M_n \in R(C, M_0)$, dada uma marcação qualquer M_n para a rede C.

2.3.5 Vivacidade *(Liveness)*

O conceito de vivacidade está diretamente ligado a total ausência de deadlocks em sistemas operacionais. Uma rede de Petri (N, M_0) é dita *viva* (ou equivalentemente, M_0 é dita uma marcação viva) caso, não importando a marcação que será alcançada a partir de M_0, seja sempre possível disparar uma transição, dando continuidade à sequência de disparos.

Uma rede de Petri viva garante que o sistema modelado vai operar livre de *dead locks* e por isso vivacidade é uma propriedade ideal para muitos sistemas. No entanto, torna-se impraticável verificar esta propriedade para sistemas como os operacionais e/ou distribuídos. Commoner em [4] relaxa a condição de vivacidade e define níveis de vivacidade como os seguintes. Uma transição t de uma rede (N, M_0) é dita:

0) L0-viva ou morta, se t nunca for disparada em qualquer sequência de disparos possível em $L(M_0)$;

1) L1-viva ou pontencialmente disparável, se t pode ser disparada pelo menos uma vez em alguma sequência de disparos em $L(M_0)$;

2) L2-viva se, dado algum inteiro positivo k, t pode disparar pelo menos k em alguma sequência de disparos em $L(M_0)$;

3) L3-viva, se t aparece infinitamente em alguma sequência de disparo em $L(M_0)$;

4) L4-viva ou viva se t é L1-viva em toda marcação $M \in R(M_0)$.

Uma rede de Petri é dita $Lk - viva$ se toda transição na rede é $Lk - viva$ para $k = 1, 2, 3, 4$. Uma rede de Petri L4-viva corresponde a primeira definição de vivacidade.

2.3.6 Persistência *(Persistence)*

Um rede de Petri é dita persistente se para quaisquer duas transições habilitadas, o disparo de uma não irá desabilitar a outra. Em outras palavra, uma transição habilitada irá permanecer habilitada até que ela dispare.

2.3.7 Distância Sincrônica *(Sincronic Distance)*

A relação de distância Sincrônica é um dos conceitos fundamentais introduzidos por C. A. Petri. Compreende uma métrica relacionada ao grau de dependência entre a ocorrência de dois eventos em um sistema de eventos e condições. Em uma rede de Petri, a distância sincrônica entre duas transições t_1 e t_2 é definida como: $d_{12} = max_\sigma|\sigma(t_1) - \sigma(t_2)|$, onde σ é a sequência de disparos começando na marcação μ de $R(\mu_0)$ e $\sigma(t_1)$ é o número de vezes que a transição t_1 disparou em σ.

2.4 Redes de Petri de Alto Nível

O modelo de rede de Petri apresentado como definição é chamado de modelo Clássico e é a base para a definição das redes de Petri de Alto Nível. Redes de Petri de Alto Nível diferem do modelo clássico, principalmente, nas condições de disparo de transições e na caracterização das fichas. Eles incorporam às redes de Petri Clássicas um maior poder descritivo e no entanto não perdem a coerência do modelo apresentado. Dentre estas redes de alto nível veremos dois tipos que deram origem à rede de Petri utilizada no projeto do simulador.

2.4.1 P/T Nets: Redes de Predicados e Transições

Como no modelo Clássico, no modelo de Predicados e Transições temos os seguintes elementos:

1. um conjunto de lugares;
2. um conjunto de transições;
3. um conjunto de arcos ligando lugares e transições;
4. um conjunto de fichas;

As diferenças com relação as redes de Petri Clássicas seguem dos seguintes pontos[16]:

1. Os lugares são *k-limitados* e as fichas podem ser descritas como objetos com valores associados;
2. Aos arcos são associadas variáveis. As variáveis associadas aos arcos que chegam a uma transição são chamadas de *variáveis de entrada*. As variáveis associadas aos arcos que saem das transições são chamadas *variáveis de saída*;
3. Uma expressão booleana chamada *fórmula seletora* é associada a cada transição. Os argumentos da fórmula seletora são os valores das variáveis de entrada da transição em questão.
4. As variáveis, por sua vez, obtém seu valores a partir de predicados sobre às fichas residentes nos lugares de entrada associados.
5. Uma transição pode ser vista como um procedimento que atribui valores às variáveis de saída com base nas variáveis de entrada. No cálculo dos valores atribuídos às variáveis de saída, constantes, variáveis e funções podem ser usadas.
6. As condições de habilitação de uma transição estão associadas aos seguintes pontos:
   - todas as variáveis de entrada possuem valores;
   - a fórmula seletora é avaliada como verdadeira;
   - os valores das variáveis de saída não provocarão a entrada de um número excessivo de fichas nos lugares de saída no disparo de uma barra de transição;

Neste modelo é importante ressaltar os seguintes pontos:

(1) As condições de disparo e habilitação de uma transição são mais fortes, visto que elas consideram as fichas como objetos e consequentemente consideram a informação contida nesses objetos na habilitação da transição. Há também a consideração sobre o estouro da capacidade de conter fichas nos lugares de saída.

(2) Embora na operação de disparo de uma transição possam ser consideradas constantes, variáveis e funções que não as variáveis de entrada e saída, as condições de habilitação e disparo estão basicamente ligadas à informação contida nas fichas. Não existe nenhuma restrição ao disparo independente do fluxo de fichas e da informação contida nessas fichas;

2.4.2 Redes de Petri Numéricas (RPNs)

Redes de Petri Numéricas são uma generalização do modelo clássico que retém seus princípios básicos, entidades e modos de operação, adicionando a este um maior poder descritivo. Suas maiores vantagens estão na capacidade de descrição de sistemas exibindo concorrência, provendo técnicas de análise que delineiam o comportamento do sistema modelado. Uma rede de Petri Numérica possui as seguintes extensões acrecentadas ao modelo clássico[17]:

1. Fichas podem ter uma natureza e um valor;
2. Existe uma memória de leitura e escrita associada à rede;
3. As condições de habilitação e disparo são associadas aos arcos individualmente e são independentes entre si;
4. As condições de habilitação das transições referem-se às fichas existentes nos lugares de entrada e as variáveis na memória referentes à transição;
5. A ação de disparo de transições incluem a retirada de fichas dos lugares de entrada, a entrada de fichas nos lugares de saída e operações na memória da rede;

Como no modelo de Predicados e Transições, as fichas nas redes de Petri Numéricas carregam consigo uma quantidade de informação maior que no modelo clássico. O disparo de uma transição está associado a uma função de transição que se refere às condições das fichas nos lugares de entrada da transição.

Uma diferença básica entre os dois modelos é a existência nas redes de Petri Numéricas de uma memória associada à rede que fornece restrições

de habilitação independentes das fichas residentes nos lugares de entrada da transição.

Como em todos os modelos de rede de Petri, apenas uma transição pode disparar de cada vez. O disparo de uma transição corresponde às seguintes operações indivisíveis:

- Fichas são removidas dos lugares de entrada de acordo com suas funções de transição individuais;
- a operação de transição é suspensa;
- Fichas são inseridas nos lugares de saída de acordo com as suas funções de transição específicas;

2.5 Redes de Petri com Restrições de Tempo

Redes de Petri que possuem restrições de tempo são apresentadas em dois modelos distintos: *redes temporizadas de Ranchandani e as redes com tempo de Merlin.*

Redes temporizadas de Ranchandani são obtidas do modelo clássico pela associação de um tempo de disparo a cada transição da rede. A regra de disparo é modificada considerando que o intervalo de tempo de disparo de uma transição é iniciado ao mesmo tempo em que a transição fica habilitada e que o disparo se completa após o término do intervalo de tempo associado à transição. Isso significa que o disparo de uma transição não é mais um evento instantâneo. Este tipo de modelo é usado principalmente para a avaliação de desempenho de sistemas. O segundo e mais interessante modelo para análise e simulação de protocolos de comunicação é a rede de Petri com tempo de Merlin[3](RPT).

2.5.1 Redes de Petri com Tempo

As RPTs são mais genéricas que as redes de Ranchandani. Nelas um par de número não negativos (V_{min}, V_{max}) é associado a cada transição. O primeiro valor, V_{min} fornece o tempo mínimo que a transição deve permanecer habilitada antes do seu disparo. V_{max} fornece o tempo máximo em que a transição deve disparar se esta permanecer habilitada durante o intervalo

[3]Time Petri Net

$(0, V_{max})$. Como consequência disso, se a transição permanecer continuamente habilitada ela irá disparar em V_{max} em todo caso. Se uma transição for habilitada no tempo μ, ela não pode disparar antes de $\mu + V_{min}$ e deve disparar antes de $\mu + V_{max}$ a menos que esta tenha sido desabilitada na ocorrência do disparo de uma outra transição.

Note que o comportamento desta regra de disparo permite a descrição perfeita de *time-out*. Com esta regra, pode-se definir o tempo de espera do *time-out* quando este é acionado. Além disso, as RPT também permitem modelar com os valores de V_{min} e V_{max} as possíveis incertezas do tempo atual no qual o *time-out* vai terminar.

Em geral, estas redes são mais difíceis de analisar devido a:

1. o número infinito de possíveis tempo que uma transição pode disparar entre (V_{min}, V_{max});

2. quando uma transição dispara, este disparo induz subtas restrições em transições que permanecem habilitadas.

A primeira solução completa para analisar este tipo de rede se baseia na construção de um conjunto de **marcações alcançáveis previsíveis** da RPT.

Redes de Petri com restrições de tempo ocupam lugar de destaque como ferramenta de análise de desempenho em protocolos de comunicação. No entanto este não é o nosso objeto de estudo pelo menos no escopo desta dissertação. E por isso, encerramos aqui este breve parênteses sobre essas redes.

Cabe ainda lembrar que a capacidade de adaptação de modelos computacionais de redes de Petri permite que possamos acrescentar restrições de tempo a um modelo de rede de alto nível, o que pode ser interessante para tornar mais completa uma ferramenta de simulação e a princípio justifica esta apresentação superficial do modelo.

2.6 Proposta de rede de Petri de Alto Nível

Usualmente, entidades cooperantes/comunicantes em um sistema distribuído são representados por conjuntos de processos comunicantes. Esses processos e a forma com que eles se comunicam entre si são os elementos que estamos interessados em modelar e simular. Por esse motivo, é nosso desejo obter um modelo de uma rede de Petri de alto nível que permita-nos que (a partir de uma especificação SDL):

1. Identificados cada um dos processos, descrever os mesmos como um modelo de rede;

2. conectar explícitamente as descrições individuais dos processos para obter um modelo global do sistema distribuído;

3. Ter uma metodologia de análise para o modelo do sistema;

Além disso, estamos também interessados em construir uma ferramenta de simulação que compreenda os modelos obtidos utilizando a rede de Petri estabelecida. Assim o tipo de rede estabelecida engloba características que a tornam compatível tanto com os requisitos de modelagem de uma descrição formal feita em SDL quanto os requisitos para a construção da ferramenta de simulação.

Como já mencionado, a rede de Petri elaborada é baseada no modelo clássico com associações explícitas de características dos modelos de predicados e transições e das redes numéricas.

Nas redes de Predicados e Transições, as condições de habilitação relacionadas às fichas são dispostas de maneira mais clara que nas redes de Petri Numéricas. Estas condições são implementadas através das variáveis de entrada presentes nos arcos da transição.

Na definição do modelo foi utilizada a mesma forma de avaliação das fichas nos lugares de entrada do modelo de Predicados e Transições. Assim a rede possui variáveis de entrada com predicados relacionados às fichas residentes nos lugares de entrada.

Uma memória global da rede também foi utilizada e permite a definição de pré condições e pós condições de disparo das transições que são independentes da natureza e do fluxo de fichas existentes na rede.

O modelo definido apresenta as seguintes características e entidades a serem incorporadas ao modelo clássico apresentado em 2.1:

1. Variáveis de entrada associadas aos arcos: Quando o arco parte de um lugar para uma transição. Estas variáveis possuem um *valor* lógico e um predicado para o cálculo desse valor lógico. Estes predicados podem ter parâmetros quantidades de fichas, suas cores e seus valores. O conjunto de fichas presentes no lugar associado ao mesmo arco é utilizado para o cálculo dos predicados das variáveis.

2. Variáveis de Saída associadas aos arcos: Quando o arco parte de uma transição para um lugar. Estas variáveis são *regra pós disparo* que definem quais e quantas fichas serão transferidas para o lugar de saída associado a variável.

3. Possui uma função lógica associada a cada transição chamada *Fórmula Seletora* que avalia as variáveis de entrada segundo um certo predicado. Se o predicado for avaliado como verdadeiro a fórmula seletora retorna Verdadeiro, caso contrário retorna Falso. O resultado da Fórmula Seletora constitui uma das *pré condições* de disparo.

4. Possui uma *Memória Global* onde podem ser armazenadas variáveis inteiras, lógicas e literais. Estas variáveis são utilizadas tanto em testes pré disparo de transições como também são manipuladas em ações pós disparo.

5. Cada transição possui uma fórmula seletora (associada as suas variáveis de entrada e consequentemente às fichas dos lugares de entrada) e um conjunto de pré condições associadas as variáveis existentes na memória global. Assim suas condições de habilitação e de disparo são funções tanto das fichas residentes em seus lugares de entrada, como das variáveis globais consideradas nas pré condições . As ações pós disparo são dependentes das variáveis de saída da transição (no que diz respeito ao fluxo de fichas) e das pós condições associadas a variáveis da memória global e a transição em questão.

Mais suscintamente, a estrutura e características da rede de Petri definida serão as seguintes:

- Um conjunto de lugares que podem conter fichas;
- Um conjunto de fichas que possuem uma natureza representada por sua cor e um valor que distingue fichas de uma mesma natureza;
- Um conjunto de barras de transição;
- Um conjunto de arcos orientados ligando os lugares e as barras de transição;
- Conjuntos de variáveis de entrada independentes associados aos arcos de entrada das transições;
- Conjuntos de variáveis de saída independentes associadas aos arcos de saída das transições;
- Um função lógica chamada *fórmula seletora* associada a cada barra, que avalia as variáveis de entrada de acordo com um determinado predicado;
- Uma memória global que permite leitura e escrita;

- Conjuntos de pré condições associadas a cada transição individualmente e envolvendo as variáveis da memória global;
- Conjuntos de Pós condições associadas a cada transição e envolvendo as variáveis da memória global.

Como exemplo de utilização do modelo de rede de Petri estabelecido, a seção seguinte traz a modelagem de um protocolo de comunicação simples conhecido como bit alternante.

2.7 Representação do Protocolo *Bit Alternante*

O protocolo *Bit Alternante* controla dois caminhos de transferência de dados com o esquema envia recebe. Antes de enviar uma mensagem, o processo emissor espera a chegada do reconhecimento da mensagem que foi enviada anteriormente. A perda de uma mensagem é corrigida através de retransmissões. Para o controle das mensagens, essas são numeradas com números de sequência módulo dois. A modelagem de um caminho é simétrica ao outro. Neste exemplo, estará sendo feita a modelagem de apenas um caminho.

A figura 2.2 mostra o grafo da rede de Petri do protocolo com os lugares, barras de transição e arcos.Os arcos serão identificados com os pares ordenados, (L,T) para os arcos de entrada e (T,L) para os arcos de saída das barras.Para cada arco existe um conjunto de variáveis com predicados ou ações. Para cada transição serão apresentadas as pré e pós condições de disparo.

A fórmula seletora, para esta descrição de protocolo é igual para todas as barras: todas as variáveis de entrada devem possuir valor válido. Elas não serão citadas na descrição.

2.7.1 Fichas e Variáveis da Rede

A representação do protocolo necessitou das seguintes variáveis globais:

- X1;
- T1 - temporizador;
- T2 - temporizador;

As fichas podem ser de 5 cores diferentes mas não possuem valores. São elas:

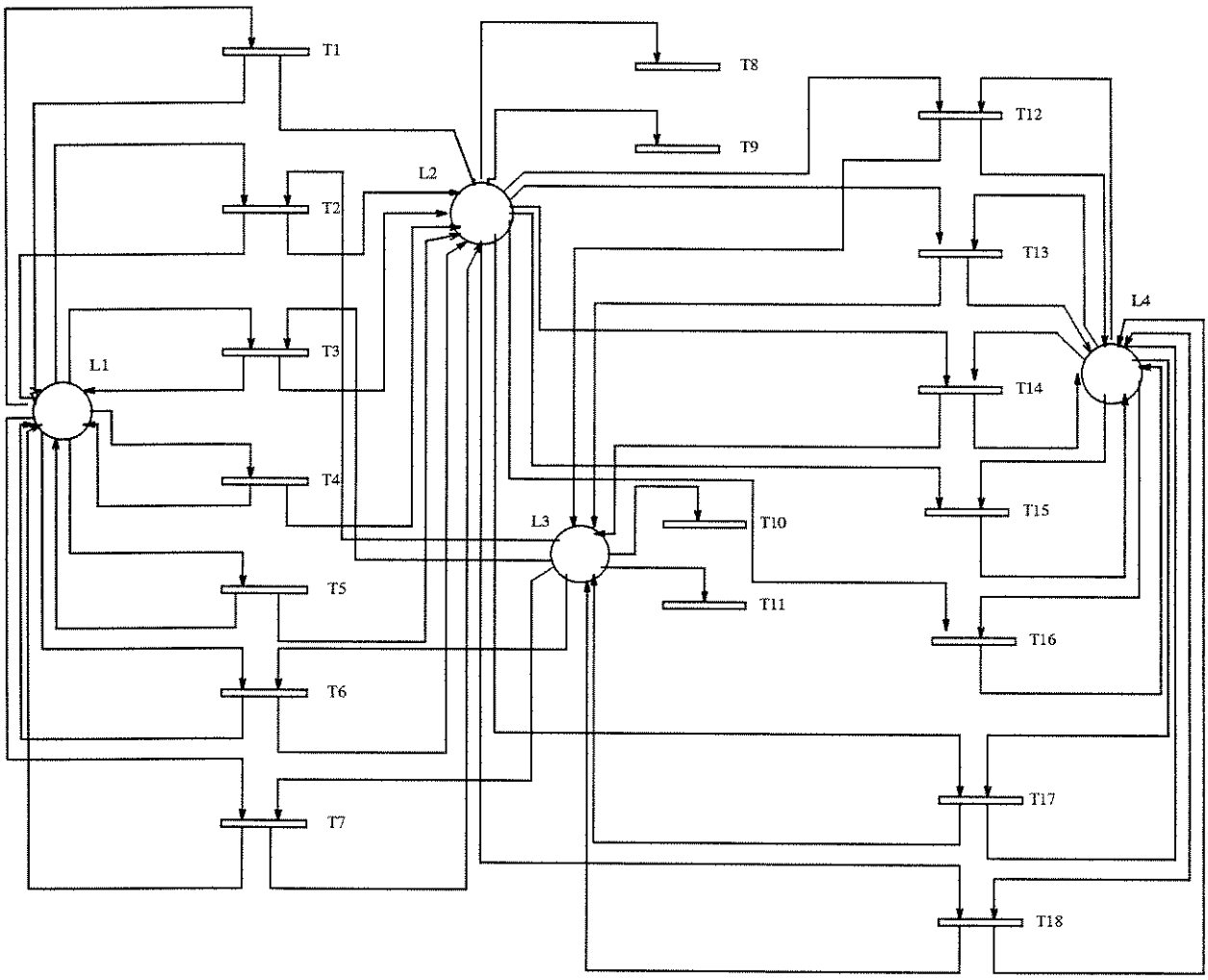

Figura 2.2: Rede de Petri do Protocolo Bit Alternante

- S1: bit 1;
- S2: bit 0;
- S3: mensagem;
- S4: fim de transmissâo;
- S5: espera da primeira mensagem;

2.7.2 Predicados das Variáveis de entrada

Predicados das variáveis dos arcos que partem do lugar L1:

1. arco (L1,T1): v1:1 S3 (Existe 1 ficha de cor S3 em L1?)
2. arco (L1,T2): v1:1 S1;
3. arco (L1,T3): v1: 1 S2
4. arco (L1,T4): v1: 1 S1;
5. arco (L1,T5): v1: 1 S2;
6. arco (L1,T6): v1: 1 S1;
7. arco (L1,T7): v1: 1 S2;

Predicados das variáveis dos arcos que partem do lugar L2:

1. arco (L2,T8): v1: 1 S1; v2: 1 S3;
2. arco (L2,T9): v1: 1 S2; v2: 1 S3;
3. arco (L2,T12): v1: 1 S1; v2: 1 S3;
4. arco (L2,T13): v1:1 S1; v2: 1 S3;
5. arco (L2,T14): v1: 1 S2 v2: 1 S3
6. arco (L2,T15): v1: 1 S4;
7. arco (L2,T16): v1: 1 S4;
8. arco (L2,T17): v1: 1 S1; v2:1 S3
9. arco (L2,T18): v1: 1 S2; v2:1 S3;

Predicados das variáveis dos arcos que partem do lugar L3:

1. arco (L3,T2): v1:1 S1;
2. arco (L3,T3): v1: 1 S2
3. arco (L3,T6): v1: 1 S1;
4. arco (L3,T7): v1: 1 S2;
5. arco (L3,T10): v1: 1 S1;
6. arco (L3,T11): v1: 1 S2;

Predicados das variáveis dos arcos que partem do lugar L4:

1. arco (L4,T12): v1: 1 S5;
2. arco (L4,T13): v1:1 S1;
3. arco (L4,T14): v1: 1 S2;
4. arco (L4,T15): v1: 1 S1;
5. arco (L4,T16): v1: 1 S2;
6. arco (L4,T17): v1: 1 S2;
7. arco (L4,T18): v1: 1 S1;

2.7.3 Ações das Variáveis de Saída

Ações das variáveis de saída dos arcos que chegam no lugar L1:

1. arco (T1,L1): v1: 1 S1 (insere uma ficha de cor S1 em L1);
2. arco (T2,L1): v1: 1 S2;
3. arco (T3,L1): v1: 1 S3;
4. arco (T4,L1): v1: 1 S1;
5. arco (T5,L1): v1: 1 S2;
6. arco (T6,L1): v1: 1 S3;
7. arco (T7,L1): v1: 1 S3;

Ações das variáveis de saída dos arcos que chegam no lugar L2:

1. arco (T1,L2): v1: 1 S1; v2: 1 S3;
2. arco (T2,L2): v1: 1 S2; v2: 1 S3;
3. arco (T3,L2): v1: 1 S1; v2: 1 S3;
4. arco (T4,L2): v1: 1 S1; v2: 1 S3;
5. arco (T5,L2): v1: 1 S2; v2: 1 S3;
6. arco (T6,L2): v1: 1 S4;
7. arco (T7,L2): v1: 1 S4;

Ações das variáveis de saída dos arcos que chegam no lugar L3:

1. arco (T12,L3): v1: 1 S1;
2. arco (T13,L3): v1: 1 S1;
3. arco (T14,L3): v1: 1 S2;
4. arco (T17,L3): v1: 1 S1;
5. arco (T18,L3): v1: 1 S2;

Ações das variáveis de saída dos arcos que chegam ao lugar L4:

1. arco (T12,L4): v1: 1 S2;
2. arco (T13,L4): v1: 1 S2;
3. arco (T14,L4): v1: 1 S1;
4. arco (T15,L4): v1: 1 S5;
5. arco (T16,L4): v1: 1 S5;
6. arco (T17,L4): v1: 1 S2;
7. arco (T18,L4): v1: 1 S1;

2.7.4 Pré Condições de Disparo das Barras de Transição

As pré condições estão associadas as barras de transição. Nem todas as barras possuem pré condições:

Barra T1: (1) X1 = 0 (a variável X1 possui valor 0?); (2) T1=OFF (o temporizador T1 está OFF);

Barra T4: (1) T2 = OFF; (2) T1 = OFF;

Barra T5 (1) T2 = OFF; (2) T1= ON;

2.7.5 Pós Condições de Disparo das Barras de Transição

Barra T1: (1) T1 = ON (atribui à variável T1 o valor ON); (2) T2 = ON;

Barra T2: (1) T2 = ON;

Barra T3: (1) T2 = ON;

Barra T4: (1) T2 = ON;

Barra T5: (1) T2 = ON;

Barra T6: (1) X1 = 1; (2) T1 = OFF;

Barra T7: (1) X1 = 1; (2) T1 = OFF;

Barra T8: (1) T2 = OFF;

Barra T9: (1) T2 = OFF;

Barra T10: (1) T2 = OFF;

Barra T11: (1) T2 = OFF;

Barra T15: (1) X1 = 0;

Barra T16: (1) X1 = 0;

2.7.6 Descrição da Rede

Transições

T1 : Envia a mensagem inicial;

T2 : Recebimento da confirmação OK1 (ficha s1);

T3 : Recebimento da confirmação OK2 (ficha s2);

T4 : Retransmissão 1;

T5 : Retransmissão 2;

T6 : Fim da transmissão após recebimento de OK1;

T7 : Fim da transmissão após o recebimento de OK2;

T8 : Perda de mensagem 1;

T9 : Perda de mensagem 2;

T10 : Perda de reconhecimento 1;

T11 : Perda de reconhecimento 2

T12 : Primeira mensagem reconhecida;

T13 : Envio de OK1;

T14 : Envio de OK2;

T15 : Recebe FIM2;

T16 : Recebe FIM1;

T17 : Deteção de mensagem repetida 1;

T18 : Deteção de mensagem repetida 2;

Funcionamento

O transmissor pode executar as seguintes operações:

- Início da transmissão (transição T1);
- Recebimento de OK1 e envio de mensagem do tipo 2 (transição T2);
- Recebimento de OK2 e envio de mensagem do tipo 1 (transição T3);
- Retransmissões (transições T4 e T5);
- Envio de fim de transmissão (transição T6 e T7);

o meio de transmissão apresenta duas ações: perda de mensagem e perda de reconhecimento para as mensagens do tipo 1 e do tipo 2. Estas ações são representadas por barras de transição sem lugares de saída (T8, T9, T10 e T11). O recepetor pode executar as seguintes ações:

- sincronização do receptor com o transmissor no início de uma transmissão (transição T12);
- envio de OK1 e OK2 (transições T13 e T14);
- sinalização de final de transmissão (transições T15 e T16);
- deteção de mensagens repetidas do tipo 1 e do tipo 2 (transições T17 e T18);

Se o transmissor tiver no lugar L1 uma ficha S1 a mensagem a ser enviada será do tipo 1. Se o receptor (Lugar L4) contiver uma senha S1, isso indica que ele espera que a próxima mensagem será do tipo 1.

A ficha S4 foi utilizada para representar o final da transmissão. Utilizamos também a senha S5 para o sincronismo do receptor com o transmissor no início de uma transmissão.

Capítulo 3

Modelagem de Especificações SDL com Redes de Petri

O objetivo deste capítulo é introduzir os conceitos básicos de descrição de sistemas com SDL[1] e estabelecer regras e esquemas de modelagem utilizando a rede de Petri definida. No final as regras de modelagem são aplicadas a um exemplo de sistema descrito em SDL.

3.1 Algumas Características de SDL

A linguagem SDL incorpora três paradigmas básicos:

1. Os sistemas são descritos segundo o *ponto de vista funcional.*
2. A execução de atividades concorrentes é obtida utilizando o *modelo orientado a processos.*
3. A comunicação entre processos se dá através de *troca de mensagens discretas.*

O comportamento funcional de sistemas de chaveamento baseados em SDL focaliza a interação entre o conjunto de processos comunicantes. O comportamento dinâmico de cada processo é modelado como uma máquina de estados finitos estendida que trabalha autônoma e concorrentemente aos outros processos.

A cooperação entre processos é feita assincronamente, através da troca de mensagens discretas (*signals*). Para viabilizar a troca de mensagens, cada processo possui uma única fila de entrada, que recebe as mensagens enviadas pelos outros processos.

A linguagem SDL também permite a definição de macros, serviços e procedimentos. Estas construções, visam facilitar a especificação de processos. Como já mencionado, a estrutura dos processos é concebida como uma máquina de estados finitos estendida. Para possibilitar isto, a linguagem possui um conjunto de símbolos formais. Alguns dos principais símbolos gráficos são apresentados na tabela da **figura 3.1**.

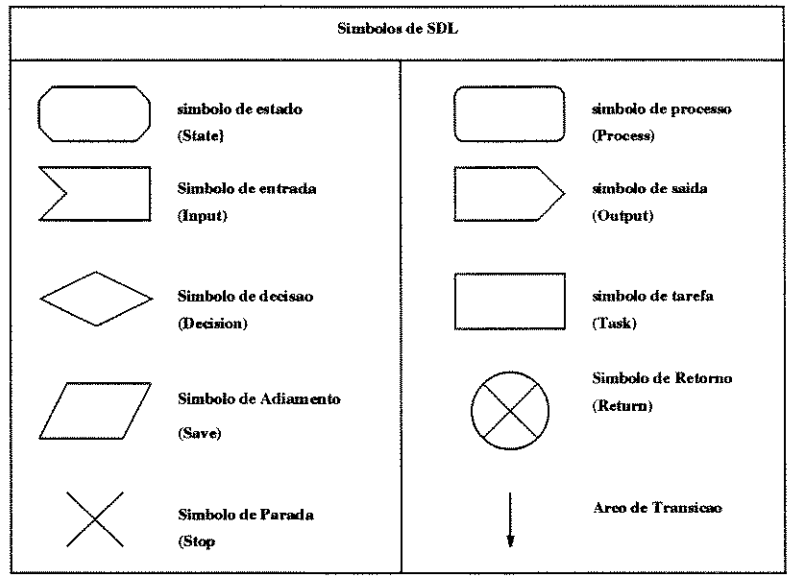

Figura 3.1: Símbolos gráficos da linguagem SDL

3.2 Modelagem com Redes de Petri

A linguagem SDL tem sido usada, em geral, para especificar sistemas concorrentes. Um esquema de modelagem capaz de modelar as especificações SDL, deve ser capaz de modelar os requisitos básicos de sistemas baseados em processos concorrentes comunicantes. Antes de apresentarmos a modelagem direta de SDL para Rede de Petri, mostraremos como a ferramenta pode ser usada de uma maneira mais genérica, na modelagem de programas concorrentes.

3.2.1 Representação de processos em redes de Petri

Um processo é descrito por um programa. Este programa representa dois aspectos separados do processo: computação e controle. Por computação

entendemos o conjunto de todas as operações lógicas e aritméticas, processamento de entrada e saída e manipulações genéricas da memória do sistema e de seu valores.

Controle não está ligado à forma como as computações são feitas, mas sim, à ordem em que estas são executadas. Uma forma padrão utilizada para representar a estrutura de controle de programas são os diagramas de fluxo ou fluxogramas.

Fluxogramas que representam o controle do programa e não especificam as computações realizadas são chamados de não interpretados e podem representar o controle de qualquer programa. Dessa forma, se conseguirmos representar fluxogramas com rede de Petri estaremos aptos a representar programas não interpretados com redes de Petri.

Um diagrama de fluxo, sob certo ponto de vista é muito semelhante a uma rede de Petri. Fluxogramas são formados de nós (computações e decisões) e arcos ligando os nós e estabelecendo a ordem de execução das computações.

A instrução corrente de um programa é identificada por um contador de programa. Este contador pode ser representado no fluxograma por uma ficha que se movimentaria pelos nós, com a sua posição correspondendo à próxima instrução que será executada. Assim, a ficha se movimenta à medida que as instruções correspondentes aos nós são executadas.

Representando nós do fluxograma por lugares na rede e os arcos por transições, obteremos uma rede de Petri que difere do fluxograma por alguns aspectos: no modelo obtido, as transições é que representam as ações, enquanto que nos fluxogramas, as ações são representadas pelos nós. Além disso, se a ficha de controle da execução parar, isso ocorrerá entre os nós e não dentro dos nós. Sendo assim, uma forma mais correta de se representar fluxogramas com redes de Petri é representar os nós destes por transições da rede e os arcos por lugares.

Cada arco do fluxograma é mapeado a um único lugar na rede. Os nós são mapeados de acordo com seu tipo: Computação ou decisão. Uma computação é mapeada como uma única transição enquanto que decisões são mapeadas a tantas transições quantos forem os caminhos possíveis de serem seguidos após a decisão.

Nesta transformação, o significado dos lugares está ligado à idéia da inserção da ficha de controle nos fluxogramas, como sendo o contador de programa. Neste caso, uma ficha residindo em um lugar significa que o programa está pronto para executar a próxima computação, representada pelas transições de saída do lugar.

Para se interpretar uma rede de Petri, devemos prover uma interpretação para cada transição. Transições que representam ações computacionais possuem um única saída e uma única entrada não existindo conflitos de execução.

Ações de decisão introduzem conflitos entre transições. Uma escolha entre as transições habilitadas para saber qual irá disparar deve ser feita. Esta escolha pode ser tanto aleatória quanto direcionada por algum mecanismo externo de decisão.

Paralelismo

Definida uma forma de se representar um único processo com redes de Petri, paralelismo e concorrência podem então ser introduzidos. Suponha dois processos concorrentes. Cada processo representado por um rede de Petri. Neste caso, a rede de Petri composta pelas duas redes particulares podem representar a execução concorrente entre os dois processos. A marcação inicial será composta de duas fichas, cada uma delas representando a posição inicial dos contadores de programa de cada processo.

Sincronização

Paralelismo e concorrência só se tornam interessantes para a solução de um problema caso os processos possam cooperar em conjunto para a solução do problema. Tal cooperação requer ou o compartilhamento da informação e recursos entre processos, ou a troca de mensagens entre os processos. No primeiro caso, o compartilhamento deve ser controlado para garantir a execução correta do sistema.

O segundo caso, a troca de mensagens, muito embora não requeira mecanismos de controle, necessita de uma série de cuidados na sua implementação de forma a não permitir principalmente, a ocorrência de impasses no sistema.

Como nosso objetivo é modelar processos que utilizam troca de mensagens não nos ateremos muito tempo com problemas de compartilhamento de recursos. Contudo, apresentaremos a título de ilustração a modelagem de um mecanismo de sincronização para a solução do problema de exclusão mútua.

Este problema surge quando, por exemplo, dois processos tentam realizar operações mutuamente exclusivas (por ex., ler e escrever o valor de uma variável ao mesmo tempo). Para resolver o conflito criamos uma exclusão mutua para as regiões de código que realizam as operações de código conflitantes.

Exclusão mútua é uma técnica para a definição de um código de entrada e saída para trechos de código que permite que no máximo um processo possa fazer acesso ao objeto compartilhado de cada vez. O código que necessita ser envolvido pela exclusão mútua é chamado de seção crítica. A figura 3.2 representa a solução para o problema utilizando rede de Petri.

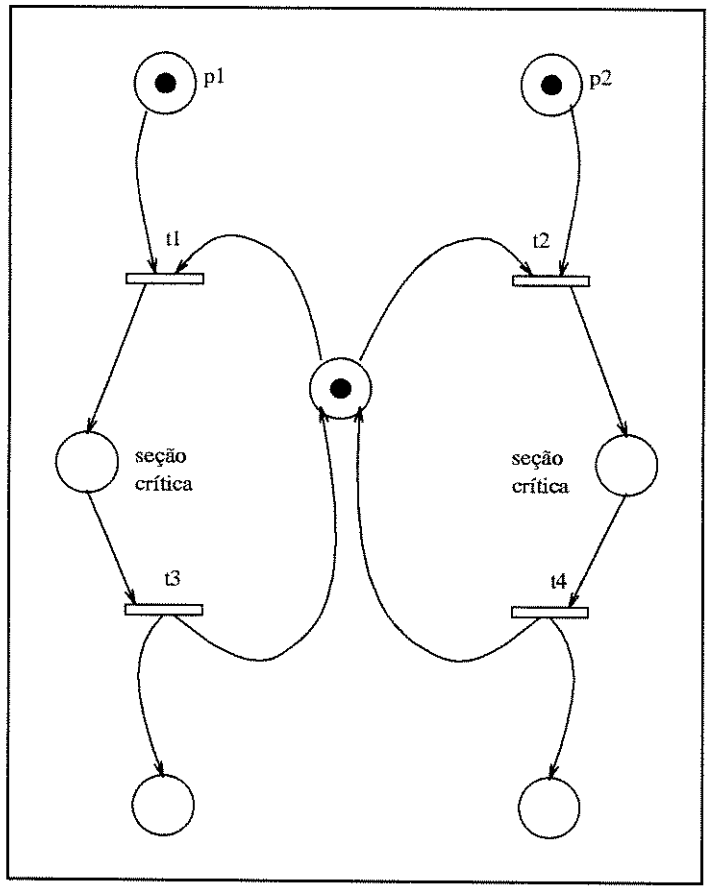

Figura 3.2: Resolução do problema de exclusão mutua

3.2.2 Sincronização e Cooperação entre processos utilizando troca de Mensagens

Um sistema baseado em troca de mensagens é uma coleção de processos que se comunicam através de mensagens. Duas operações primitivas sob mensagens são possíveis: *enviar* e *receber*. A primitiva *enviar* é dita não bloqueante, e a primitiva *receber* é dita bloqueante. Isto significa que quando uma operação receber é executada, o processo fica parado até receber a mensagem requisitada. Quando um processo executa a operação *enviar*, ele simplesmente envia a mensagem e não espera que o processo destinatário a receba. Os esquemas de controle e reconhecimento do recebimento de mensagens podem ser estabelecidos a partir destas duas operações básicas. Troca de mensagens é a maneira praticada por SDL para coordenar as atividades concorrentes de seus processos e compartilhar informações entre os mesmos.

Na definição da linguagem, cada processo descrito em SDL possui apenas uma fila de entrada onde são recebidas as mensagens. Esta filas de processos podem ser representadas por lugares de uma rede de Petri que co-existam nas sub redes que representam os processos comunicantes.

3.3 Transformação de especificações SDL em Redes de Petri

A modelagem de SDL com redes de Petri é um processo de abstração de sintaxes e semânticas formais comportadas pela linguagem. As regras aqui apresentadas para a transformação de SDL para redes de Petri são baseadas nas regras apresentadas em [17] específicas para as redes de Petri numéricas. Companrando com as regras apresentadas em [17], as regras para o processo de abstração aqui apresentadas diferem com relação a modelagem de tarefas aqui consideradas, o que não é feito em [17].

Algumas definições necessárias para o entendimento das regras:

1. A associação das barras de transição aos seus lugares de entrada e de saída (pré e pós condições) é chamada de *função de transição*;

2. As fichas são rotuladas pela sua cor. Esta cor pode ser um nome ou apenas um número. As fichas unitárias tem a função de controle da rede de Petri[1]. Elas são utilizadas para identificar o estado em que se encontram os processos, para a ativação de processos, chamadas de

[1]Estas fichas unitárias se assemelham a mecanismos de controle de execução de programas em processadores (contadores de programa)

procedimentos, ativação de macros e serviços. As cores das fichas são os própios nomes dos sinais que elas representam.

3. Os rótulos dos arcos de entrada das transições precedidos do sinal $\sim$, significam que a transição estará habilitada quando a ficha no início da fila de entrada não for da cor definida no rótulo.

3.3.1 Modelagem do Save

Em SDL, quando um processo necessita de consumir os sinais que chegam na sua fila de entrada em ordem diferente da ordem de chegada dos mesmos, fazemos uso da instrução **Save**.

O símbolo Save ou símbolo de adiamento, instrui o processo a reter a passagem de um determinado sinal na fila de entrada. Caso contrário, isto é, sinais que não são explicitamente salvos e que não são mencionados em alguma entrada são consumidos quando chegam no começo da fila, como se fosse uma transição implícita nula, permanecendo o processo no mesmo estado.

Tal como símbolos de entrada, um símbolo Save segue após um símbolo de estado. O save contém o nome dos sinais que deverão ser salvos em um estado, e é válido somente para o estado em que o processo se encontra. Caso necessário outros Saves devem ser especificados para outros estados.

O uso do Save provoca uma excessão no modo **FIFO** (*First In First Out*) das filas de entrada dos processos. Esta excessão nas redes de Petri equivalentes é obtida através de modelagem. A figura 3.3 mostra como o uso do Save é transcrito em um rede de Petri. Nesta modelagem, fica claro que o símbolo que é retirado do começo da fila é inserido no final da mesma sem provocar uma mudança de estado.

3.3.2 Regras para transformação de SDL em redes de Petri

1. *Cada* **fila de entrada** *de cada processo em SDL pode ser representada por um lugar na rede de Petri.* Assim, cada processo possui um lugar que representa sua fila de entrada. Os sinais de entrada são modelados como fichas. A espera de um sinal de entrada possui uma função de transição cuja uma das pré condições é a presença da ficha que representa o sinal de entrada, no lugar que representa a fila de entrada do processo.

   Sinais de saída são representados por fichas inseridas na fila de entrada do processo destino.

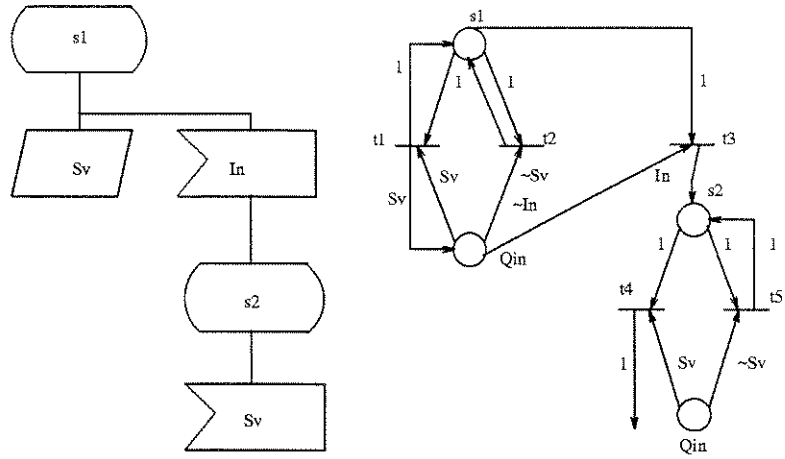

Figura 3.3: Modelagem do Símbolo Save

2. *Cada símbolo* **start, stop, procedure, return** *e de* **estado** *é representado como um lugar na rede.* A presença de uma ficha unitária em um lugar que representa um símbolo **start**, significa que o processo ainda não foi iniciado mas que está pronto para iniciar. O símbolo **procedure** é representado diretamente pelo lugar que representa o primeiro símbolo do procedimento referenciado. Por exemplo, se o procedimento inicia como um estado denominado *Idle*, a sua chamada corresponderá à inserção de uma ficha de controle no lugar *Idle*.

3. *Cada* **transição** *em SDL equivale a uma barra de transição na rede de Petri equivalente.*

4. *Uma* **transição nula** *em SDL pode ser representada por uma barra de transição e uma função de transição de estado mapeada a partir do lugar correpondente à fila de entrada do processo.* A ocorrência de uma transição nula provoca o consumo da ficha que representa o estado ativo e a imediata inserção de uma ficha no mesmo estado. A ocorrência desse evento não tem implicações sobre o fluxo de execução do sistema. Porém, a execução de uma transição nula pode ser usada para o controle e contabilização da execução.

5. *Fluxos de controle podem ser representados como uma série de funções de transição de estados mapeadas entre barras de transição e lugares.*

6. *Símbolos de* **tarefa** *podem ser descartados no modelo de rede de Petri.* Tarefas são usadas para executar operações sobre variáveis do sistema e outras funções como a ativação de temporizadores. Estes símbolos são

geralmente descartados no modelo de rede de Petri por não representarem mudanças no fluxo de execução do sistema modelado. Contudo isto não é sempre verdade.

A existência de decisões cujos predicados são variáveis declaradas na especificação SDL, tornam as tarefas operações que modificam o caminho de execução do sistema e que portanto devem ser representadas de alguma forma. Quando tarefas ativam temporizadores, esta ação também poderá influir radicalmente na atividade do sistema e também deve ser representada.

Assim, quando uma **tarefa** manipula variáveis, ela é representada por ações pós disparo da barra de transição que representa a transição em que a tarefa é ativada. Quando uma *tarefa* ativa um temporizador, esta ativação é representada pelo envio de uma ficha (*requisição*) para a fila de entrada do processo temporizador.

7. *Todo símbolo de entrada (***input***) pode ser representado como uma função de transição mapeada do lugar que representa a fila de entrada para a barra de transição correspondente.*

8. *Cada símbolo de adiamento (***save***) pode ser representado por duas funções de transição mapeadas da barra de transição para o lugar que representa a fila e vice-versa.* O símbolo **save** tem a função de adiar o consumo de sinais em um determinado estado para que estes sinais possam ser consumidos quando o sistema estiver em outro estado.

9. *Cada símbolo de saída (***output***) pode ser representado por uma função de transição mapeada da barra de transição correspondente ao lugar que representa a fila de entrada do processo destino.*

10. *Cada símbolo de decisão (***decision***) pode ser representado por um lugar associado a várias barras de transição por diferentes funções de transição correspondentes a todos os possíveis caminhos da decisão.*

A tabela da **figura 3.4** mostra graficamente a tradução dos símbolos e dos principais processamentos indivisíveis de SDL para os seus respectivos modelos em rede de Petri. Os arcos que partem de uma transição são rotulados com a cor da senha que será inserida no lugar de saída. Os arcos que chegam a uma transição são rotulados com as cores das fichas que devem estar presentes no lugar de entrada de onde parte o arco, para que a barra de transição fique habilitada.

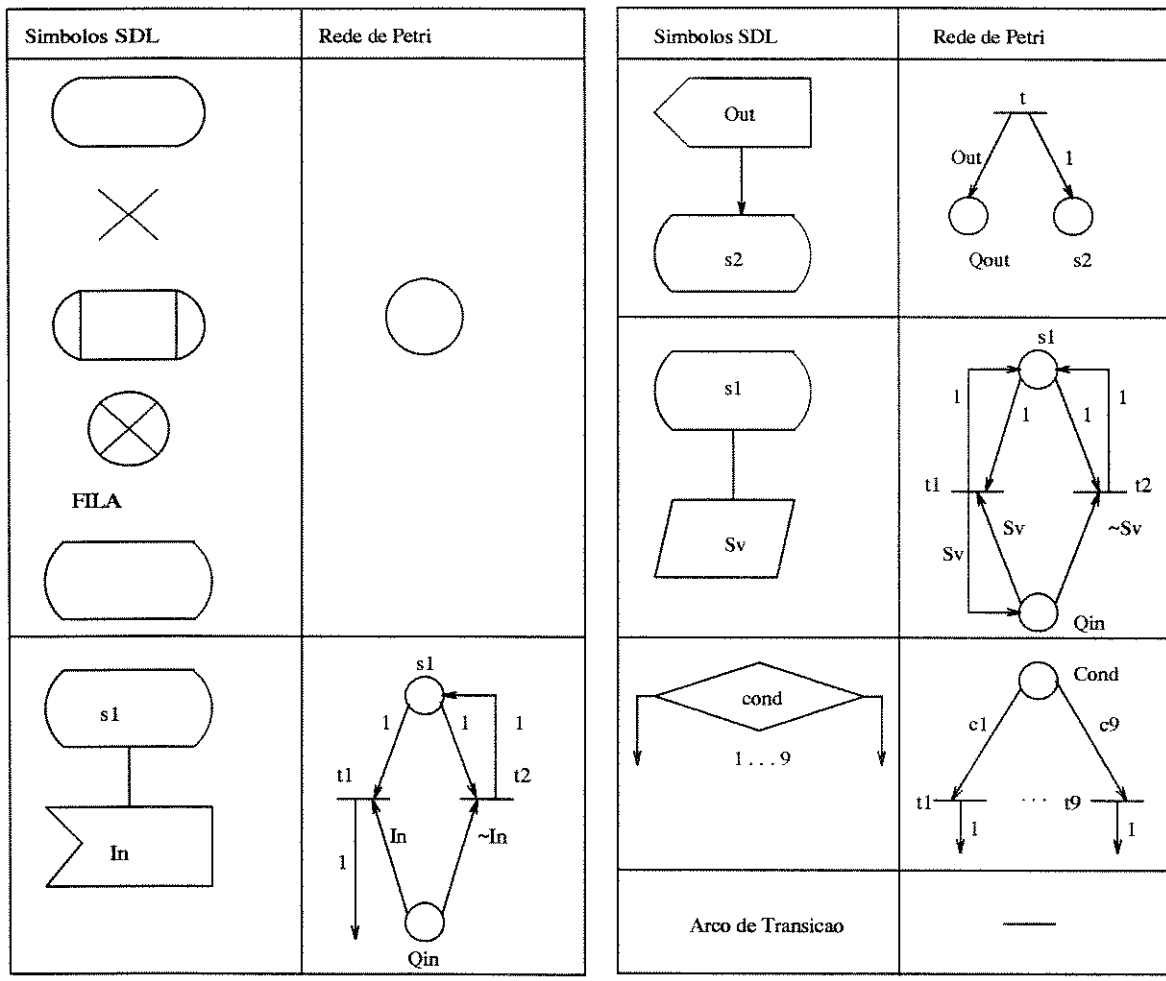

Figura 3.4: Regras de Trasformação de SDL para Rede de Petri

3.3.3 Modelagem de Temporizadores

Os temporizadores em SDL são ativados pela execução de tarefas. O temporizador é requisitado para enviar um sinal específico para o processo requisitante ao término do tempo especificado relativo ao valor da variável **now** .

O esquema para modelagem de temporizadores desenvolvido considera estes como sendo processos e, como processos, possuem uma fila de entrada e se comunicam com os demais processos da mesma maneira, indiferentemente.

A **figura 3.5** modela um temporizador que envia sinais de TimeOut para dois processos requisitantes. O temporizador é iniciado com uma ficha unitária no lugar Idle e fica esperando o envio de sinais de requisição para envio de TimeOut. Ao receber o sinal de um dos processos, o temporizador vai para o estado de envio de TimeOut (**TimeOutP1 ou TimeOutP2**). No estado de envio de TimeOut, caso o processo requisitante queira cancelar o TimeOut, este deve enviar o sinal de cancelamento para o temporizador. Com a chegada do cancelamento de TimeOut (*CancelP1 e CancelP2*), o temporizador suspende o envio e retorna para o estado Idle.

Se o sinal de cancelamento não chega antes do vencimento do "tempo" de **TimeOut**, o sinal é enviado para o processo requisitante e o temporizador vai para o estado **Idle**. Se o pedido de cancelamento chega, o temporizador vai para o estado **Idle**, e uma mensagem de cancelamento de **TimeOut** é enviada para o Ambiente externo ao sistema.

Esta modelagem não considera realmente tempos, contudo permite que possam ser modeladas situações equivalentes, as de vencimento do TimeOut e cancelamento de TimeOut.

3.4 Exemplo de Aplicação das Regras de Transformação

Nesta seção utilizaremos a especificação de um sistema simples chamado Demon Game[2] para demonstrar as regras de transformação de especificações SDL para redes de Petri.

O sistema DemonGame é um exemplo de aplicação integrante do software SDT Design Tool da Telelogic, adquirido pelo projeto temático Fapesp "Redes Metropolitanas Multi Serviços". Este software é constituido de um conjunto de ferramentas para construção de sistemas que reune as funções de: especificação, análise, simulação e geração de código C.

[2]Exemplo do SDT Design Tool

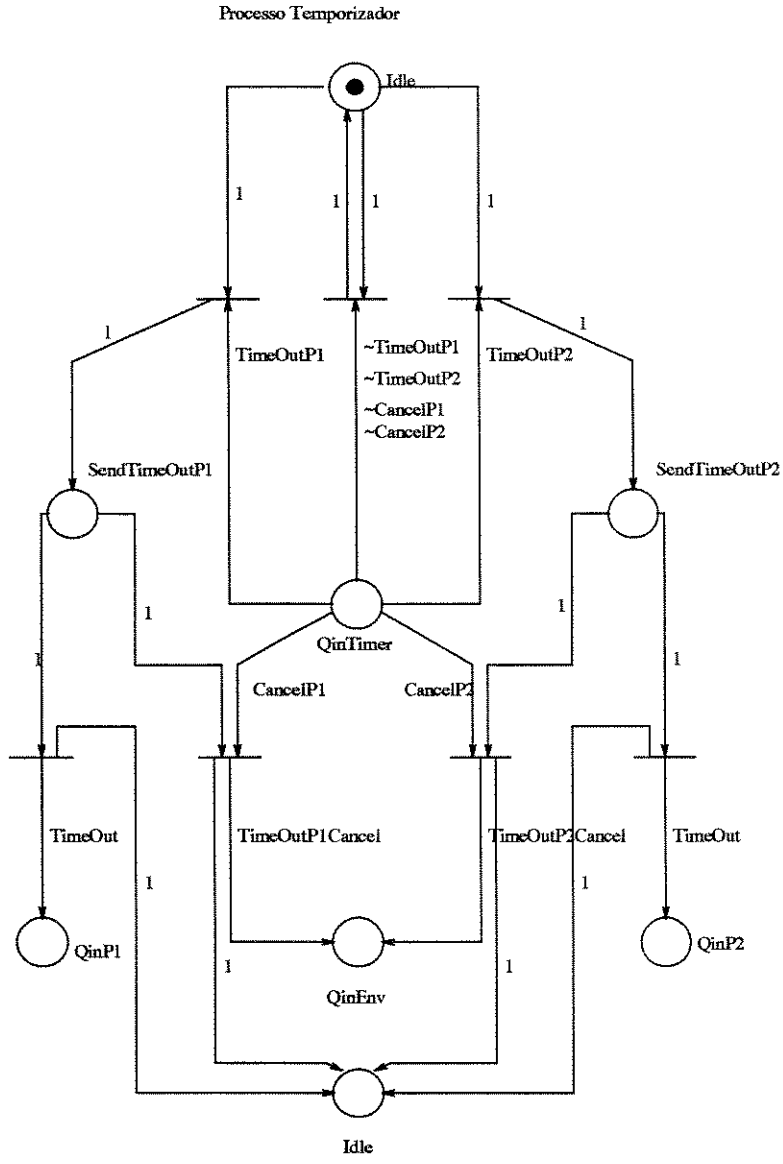

Figura 3.5: Modelagem de um processo temporizador para dois processos requisitantes de TimeOut. Obs: lugares com o mesmo nome são representações de um único lugar

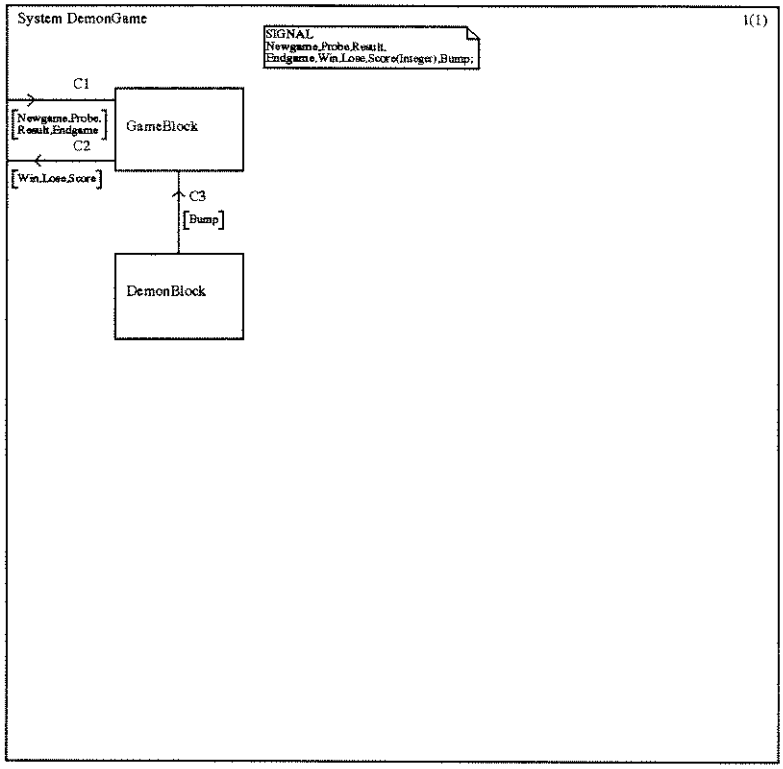

Figura 3.6: Sistema DemonGame: Blocos Funcionais

3.4.1 Descrição de um sistema em SDL

A especificação SDL tem início com a divisão do sistema em blocos funcionais. Esta divisão tem a função de reduzir a complexidade da especificação dividindo o sistema em subsistemas menores inteligados através de **Canais de comunicação**.

Os blocos funcionais são representados graficamente por retângulos contendo o nome do bloco funcional. Na especificação, o ambiente externo também é representado e interage com o sistema. Os canais de comunicação são representados por linhas direcionadas ligando entre si, os blocos e o ambiente. Sobre as linhas que representam os canais, são definidos os sinais que irão ser transportados por eles. Estes sinais são também declarados na página de especificação do sistema.

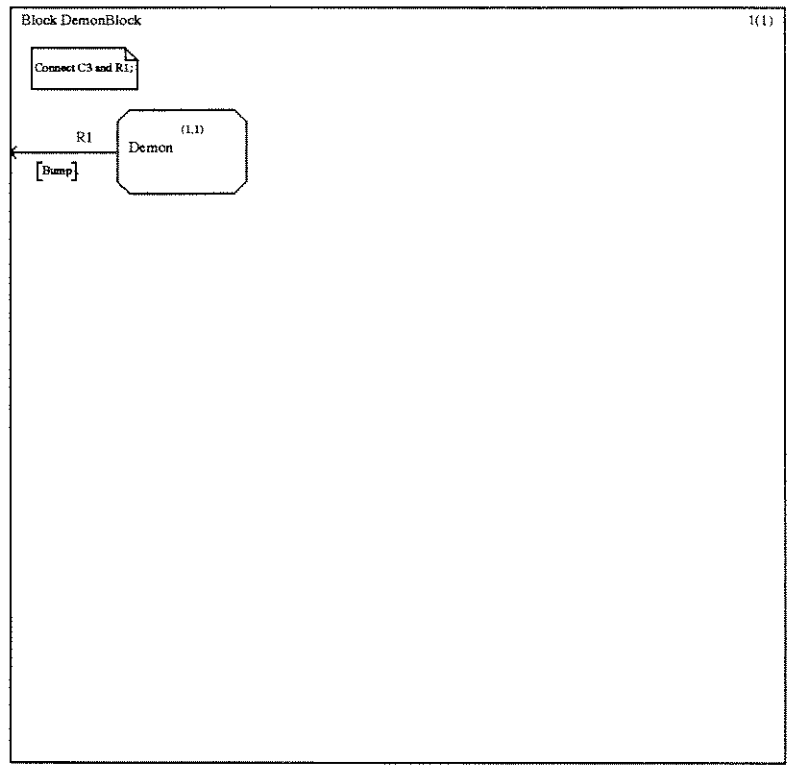

Figura 3.7: Especificação do bloco DemonBlock

Na especificação do Demon Game o sistema foi partido em dois blocos funcionais, o **bloco GameBlock** e o **bloco DemonBlock**. Os dois blocos se comunicam através do canal C3. O bloco GameBlock se comunica com o ambiente através dos canais C1 e C2. É possível ainda, dividir os blocos funcionais em subblocos e estes podem ser ainda novamente repartidos, não

havendo nenhuma imposição da linguagem para o término dos refinamentos em blocos.

Em seguida, vem a descrição de nível de abstração mais baixo que a dos blocos funcionais. É a especificação do conjunto de processos pertencentes a cada bloco. Nesta divisão os processos são representados por retângulos com vértices arrendondados contendo o nome do processo que representa.

A ligação estre os processos é feita por rotas de comunicação. Semelhante a especificação dos blocos, o retângulo maior que contorna toda a especificação, representa o ambiente externo ao processo. Os processos são ligados a este ambiente através de rotas de comunicação.

Neste nível de especificação, são definidas conexões entre os canais e as rotas de comunicação. A **figura 3.7** mostra a especificação do bloco DemonBlock que contém apenas o processo Demon. Os números entre parênteses dentro dos símbolos de processo indicam repectivamente, o número de instâncias do processo que serão criadas quando o sistema é iniciado e, o número máximo de instâncias do processo que poderão ser criadas durante a execução do sistema.

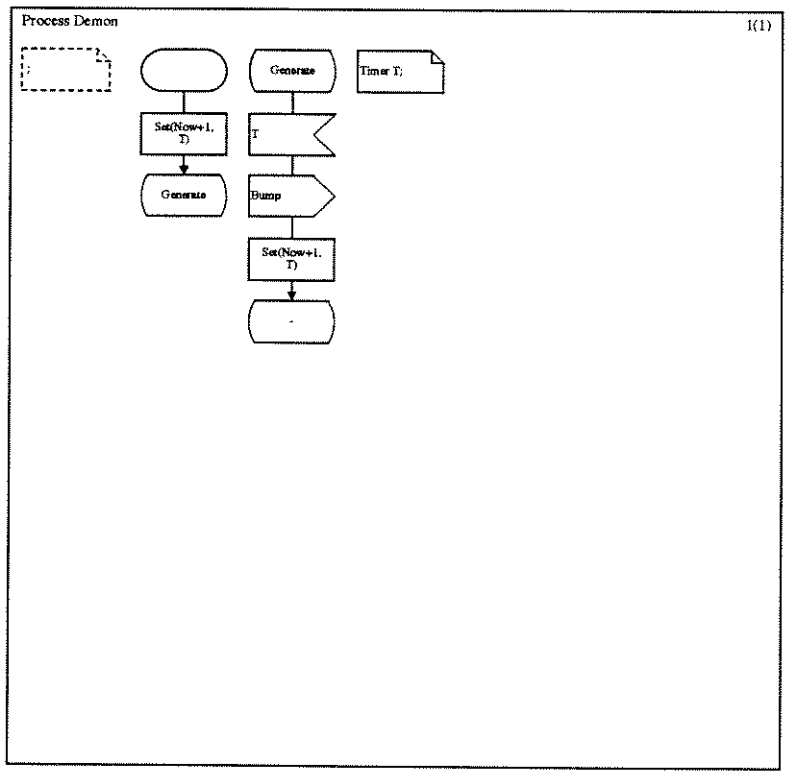

Figura 3.8: Especificação do processo Demon

A figura 3.9 refina o bloco GameBlock com seus processos e

suas rotas de comunicação. No bloco GameBlock é definido o sinal GameOver local ao bloco GameBlock.

O próximo nível de abstração desta abordagem "*top - down*" é a especificação de cada processo com a utilização dos símbolos básicos da linguagem. Do nível de especificação de processos, podemos descer ainda mais com o uso de serviços, macros e procedimentos. Estas estruturas executam sequencialmente dentro do processo em que se encontram.

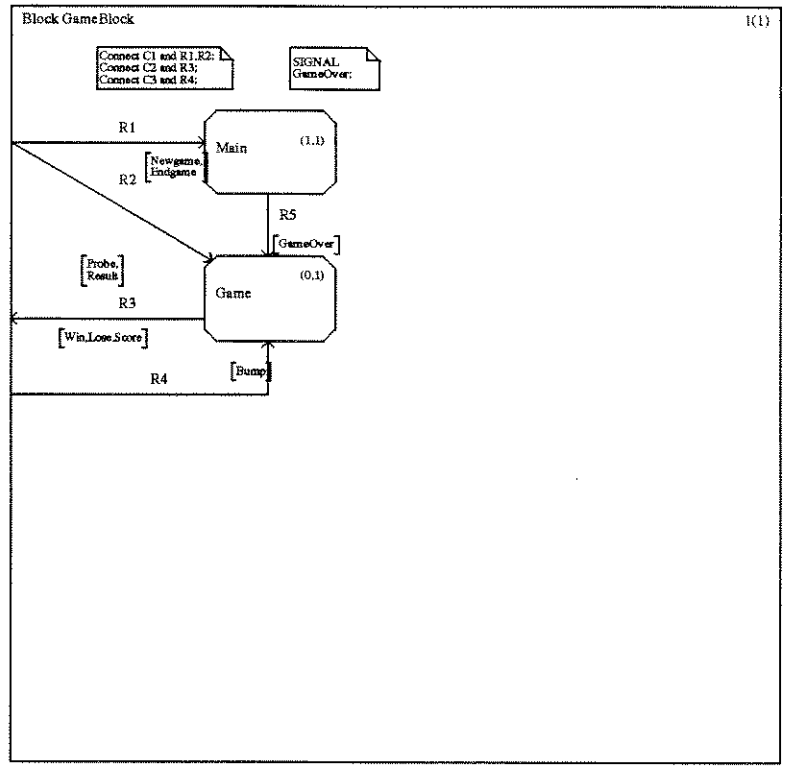

Figura 3.9: Especificação do Bloco GameBlock

3.4.2 Rede de Petri do Sistema Demon Game

A transformação para a rede de Petri equivalente, é feita levando-se em conta a divisão de processos do sistema. Assim, na figura 3.12 podemos identificar as sub redes que representam os processos Main, Game, Demon e o Timer. Na realidade, a rede não possui esta divisão entre os processos, e estes são ligados das seguintes maneiras:

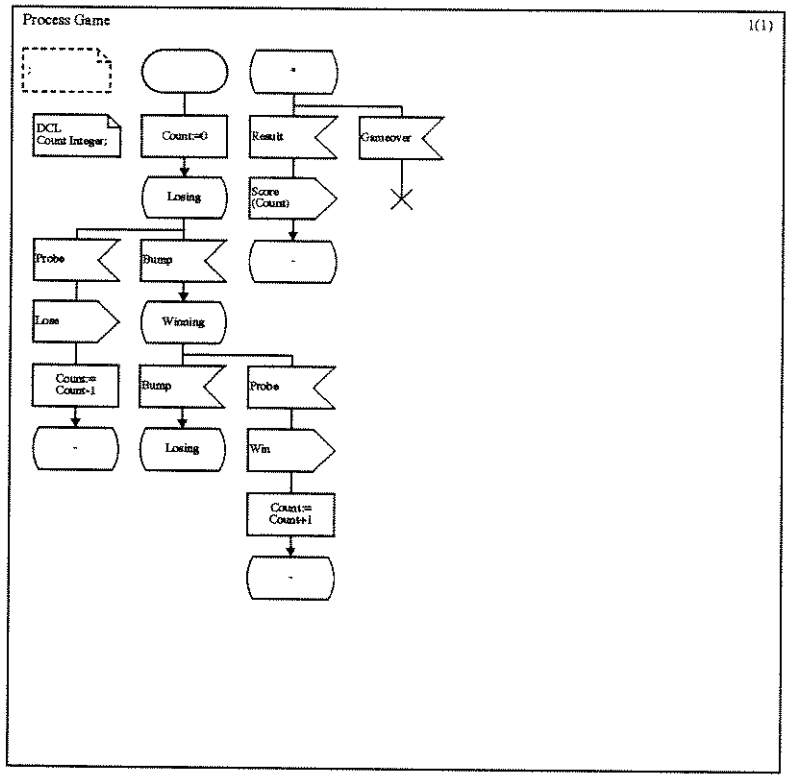

Figura 3.10: Especificação do Processo Game

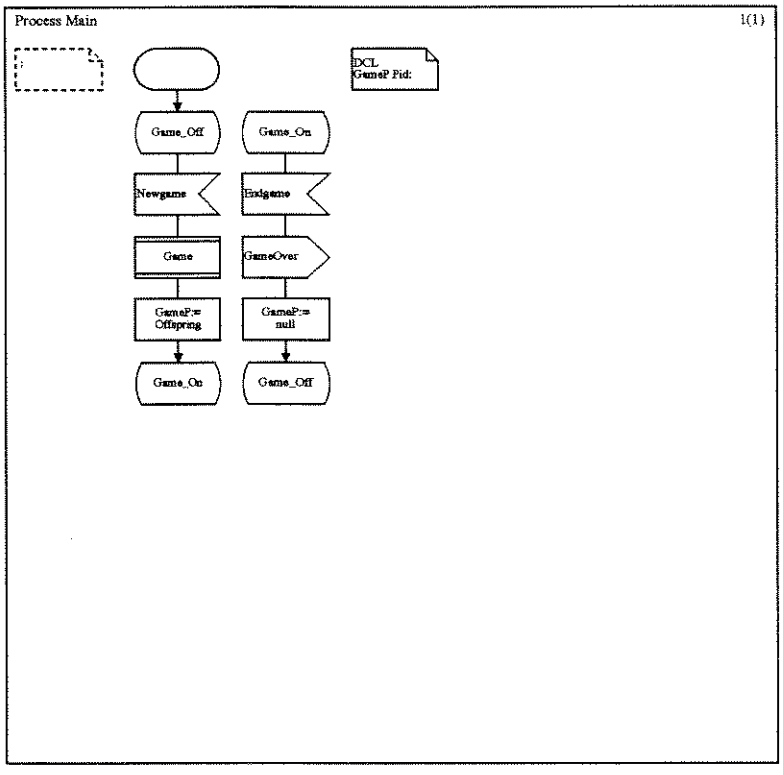

Figura 3.11: Especificação do processo Main

1. Através dos lugares que representam as filas de entrada dos processos. Os lugares que representam filas de entrada de processos ficam sendo lugares de saída de barras de transições de outros processos. Desta maneira é possível simular a troca de mensagens entre os processos comunicantes.

2. Através dos lugares que representam o estado inicial do processo, ou o símbolo start. Estes lugares são também lugares de saída de barras de transições do seu processo criador (*pai*) e recebem fichas unitárias. Desta forma podemos simular a ativação de um processo por outro.

Os lugares que representam as filas de entrada dos processos são os seguintes:

QinMain: fila de entrada do processo Main;

QinGame: fila de entrada do processo Game;

QinDemon: fila de entrada do processo Demon;

QinTimer: fila de entrada do processo Timer;

Todos os símbolos de estado da descrição em SDL são representado por lugares com o mesmo nome do estado na descrição formal. As cores das fichas da rede do sistema Demon Game possuem o seguinte significado:

ng: representa o sinal NewGame que é enviado do ambiente para o processo Main para iniciar o sistema. A recepção deste sinal leva o processo Main para o estado GameOn e cria o processo Game;

eg: representa o sinal EndGame transmitido do ambiente para o processo Main, quando o usuário quer terminar a execução do sistema. A chegada deste sinal permite o envio do sinal GameOver para o processo Game.

go: ficha que representa o sinal GameOver. Quando o processo Game recebe este sinal, ele para. Neste sistema, o símbolo stop poderia ser modelado como um lugar, como indica a regra 2 da seção 3.1.1. Contudo, esta representação do símbolo stop pode ser descartada se levarmos em conta que o processo não se encontrará mais ativo, após o disparo da barra de transição cuja pré condição é a chegada de go.

Probe: sinal enviado pelo ambiente durante o jogo. Se este sinal chega quando o processo Game está no estado Losing, o processo Game continua em Losing e envia o sinal lose para o ambiente. Se Game estiver em Winning e receber Probe, ele continua em Winning, incrementa a variável count e envia o sinal Win para o ambiente.

Bump: sinal enviado pelo processo Demon ao processo Game à chegada do sinal T do temporizador no processo Demon. Se este sinal chega quando o processo Game está no estado Winning, Game vai para o estado Losing. Se Game estiver em Losing e receber Bump, ele vai para o estado Winning.

Win e Lose: estas fichas são enviadas para o ambiente para indicar que o jogador ganhou ou perdeu pontos, respectivamente.

T: Sinal gerado pelo temporizador e transmitido para o processo *Demon*.

Result: sinal enviado pelo ambiente que requisita o envio do sinal Score para o ambiente.

Score: o sinal Score é enviado do processo Game para o ambiente. O parâmetro count do sinal é atribuído ao valor da ficha de cor Score.

Modelagem do Ambiente

Após o término da modelagem do sistema em rede de Petri, o ambiente externo ao sistema pode ser especificado também como parte integrante da rede de Petri. Isto é feito criando-se um processo que transmita e receba os sinais transmitidos e recebidos pelo ambiente que se deseja modelar.

Como os demais processos da especificação, o ambiente deve possuir uma fila de entrada, e as ligações necessárias para a sua interação com o sistema. A figura 3.13 mostra uma modelagem de um ambiente para o a interação com o Demon Game.

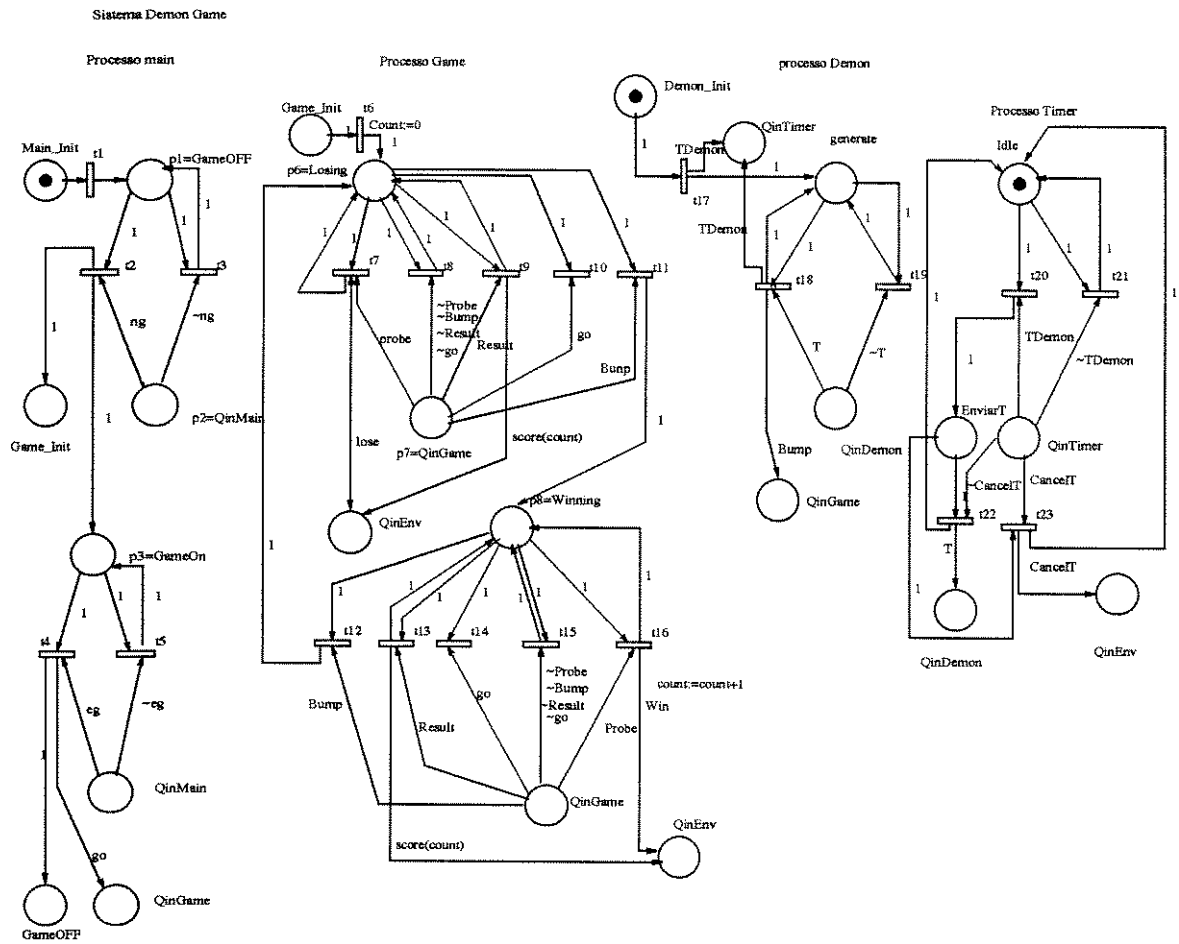

Figura 3.12: Rede de Petri do sistema DemonGame

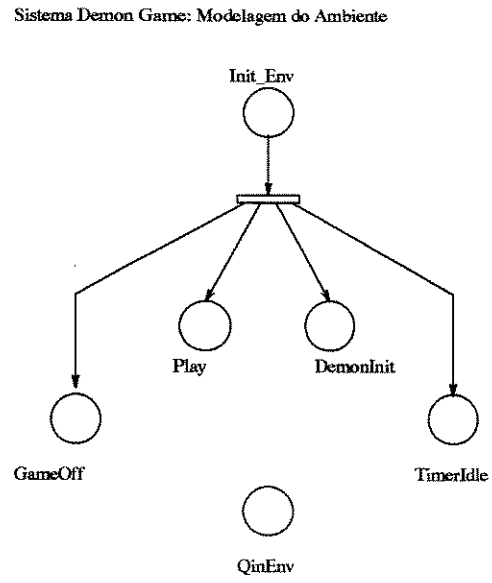

Figura 3.13: Modelagem de um ambiente para o sistema Demon Game

Capítulo 4

Protótipo de um Simulador de Sistemas Concorrentes

Este capítulo aborda o projeto e a implementação do protótipo de um simulador de sistemas concorrentes SIM, programa cuja função é simular sistemas previamente modelados com a utilização da rede de Petri proposta na Seção 2.6.

O desenvolvimento do programa simulador de sistemas concorrentes SIM, teve seu andamento em paralelo com a definição do modelo de redes de Petri utilizado e com o estudo das necessidades impostas pela própria sintaxe/semântica das especificações SDL.

A idéia central do programa é possibilitar a execução de sistemas modelados com a rede de Petri definida, estabelecendo um esquema de processos comunicantes semelhante ao de SDL, e atendendo a um conjunto de requisitos básicos de um simulador.

Para possibilitar a execução de modelagens feitas segundo a estrutura de rede proposta, foram definidas como integrantes da implementação funções e estruturas de dados que possibilitassem representar internamente ao simulador esses modelos. A definição desses objetos pode ser vista como a definição da estrutura mais interna do simulador, a partir da qual foram implementadas as funções e estruturas de simulação propriamente ditas.

4.1 Aspectos de Simulação e Simuladores

Simulação é uma das melhores e mais naturais estratégias para obtermos informações a respeito de qualquer sistema dinâmico.

No escopo de sistemas computacionais, ferramentas de simulação tornam-se mais interessantes ainda quando a execução do sistema real necessita de componentes de hardware ou de software que não estão diponíveis. Isto ocorre geralmente na fase de especificação de sistemas. É nesta etapa do ciclo de vida de sistemas concorrentes ou distribuídos que as linguagens de especificação formal como SDL são usadas. É nesta etapa também que o desenvolvimento de tais sistemas está sujeito a ocorrências de erros de difícil identificação.

Uma forma de amenizar este problema é fazer o uso de simuladores que possibilitem a investigação de especificações nos ajudando a evitar que a fase de implementação seja iniciada com erros de especificação.

Um simulador consiste em um esquema que possibilita a execução de especificações fora do seu ambiente real de execução. Este esquema pode ser um programa executável que modele um sistema ou um programa que extraia um modelo arquitetural do sistema a partir da especificação completa e possibilite a sua execução. Neste último caso o programa geralmente faz associações de informações adicionais ao modelo extraído da especificação de forma a atender um conjunto de objetivos da simulação.

O modelo arquitetural extraído da especificação deve conter as informações necessárias para que os objetivos da simulação sejam alcançados. Ao invés de trabalhar diretamente com o modelo arquitetural da especificação, o simulador pode trabalhar com uma abstração deste, utilizando para isso uma outra ferramenta de modelagem que permita a verificação de aspectos da especificação não avaliáveis a partir do modelo ou linguagem utilizada na especificação.

Foi esta estratégia utilizada para a construção do simulador SIM, isto é, a ferramenta de simulação implementada é um programa que tem como entrada um modelo em rede de Petri que é uma abstração do sistema que se deseja simular. Sobre o modelo de entrada do sistema, o simulador executa uma série de funções que permitem a visualisação e avaliação do sistema simulado.

Simuladores também podem ser usados para o estabelecimento de conjuntos de testes de sistemas. Testar um sistema significa checar se uma implementação está correta no que diz respeito à sua especificação. Para a realização desta tarefa, é necessário estabelecer o conjunto de testes que serão desenvolvidos. O desenvolvimento do conjunto de teste possui três estágios:

1. Examinar cuidadosamente a especificação a ser testada;
2. desenvolver um conjunto de propósitos do teste;
3. desenvolver um conjunto de testes que cubram todos os propósitos pré-estabelecidos;

Tanto no primeiro quanto no segundo estágio do desenvolvimento de testes, o uso de simuladores pode ser de grande valia.

4.1.1 Algumas Características e Requerimentos de um Simulador

Na construção de um simulador, aspectos como representação da configuração de hardware, tempo e escalonamento podem estar relacionados com os objetivos da simulação. Por se tratar de uma linguagem de especificação, SDL oferece algumas facilidades para o desenvolvimento de uma ferramenta de simulação que deseja simular especificações feitas utilizando a linguagem. Em especificações SDL, por exemplo, o tempo para a ocorrência de transições entre estados é considerado nulo.

Questões relacionadas ao escalonamento de processos pelo sistema operacional e características do sistema que irá suportar o simulador são de grande importância e em geral difíceis de se estimar quando situações que envolvem tempo necessitam de ser representadas em simulações. Para o caso de simulação de especificações SDL entretanto, os problemas temporais não geram complicações.

Dentro deste contexto o simulador deve prover ao usuário um conjunto de funções dentre as quais podemos citar:

Redirecionamento de entrada e saída: É necessário que haja várias possibilidades de entrada e saída, entre as quais podemos destacar:

- possibilidade de se ter um arquivo de comandos para o simulador como forma de entrada. Isto permite um rápido caminho para se repetir uma determinada seção de simulação;
- possibilidade de se criar um arquivo *trace*. O arquivo *trace* poderá ser usado para se estudar o comportamento da especificação.

Política de Escalonamento: Durante a simulação deve ser possível executar todos os caminhos possíveis da especificação.

Navegação: Permitir a navegação significa que o simulador deve oferecer a capacidade de movimentação (visualização) dentro das várias partes e estados envolvidos na simulação.

Gerenciamento de complexidade: O gerenciamento de complexidade deve garantir a execução de partes da especificação.

Informação de status: As informações de status devem incluir, sobretudo, estados em que o simulador e o sistema simulado se encontram, estados de filas existentes, etc.

Alterações no sistema: O simulador deve permitir ao usuário fazer alterações no status do sistema e na sua estrutura.

Suporte à linguagem: O simulador deve garantir que todos os recursos existentes em SDL possam ser usados nas especificações simuladas.

4.2 Estrutura Interna

Os dois principais componentes básicos do simulador não poderiam deixar de ser outros senão os Lugares e as Barras de Transições. Além desses dois elementos, para a representação e execução do modelo proposto foram definidos os seguintes elementos:

- Variáveis de Entrada;
- Memória Global;
- Variáveis de Saída;
- Pré-condições;
- Pós-condições;
- Função de atualização de variáveis de entrada;
- Fórmula Seletora;
- Função de avaliação de pré-condições;
- Função de habilitação;

- Função de execução dos predicados das variáveis de saída;
- Função de execução de pós-condições;
- Função de disparo;

4.2.1 Lugares

Um lugar da rede é uma estrutura de dados que armazena as seguintes informações:

Identificação: A identificação de um lugar da rede é composta por um nome e por um identificador único;

Conjunto de transições de entrada: São identificadores de barras de transições ligadas ao lugar com um arco partindo da transição e chegando ao lugar.

Conjunto de transições de saída: são identificadores de barras de transição ligadas ao lugar com um arco partindo do lugar e chegando à transição.

Lista de fichas: é uma estrutura de lista encadeada para o armazenamento de fichas da rede.

Os lugares são agrupados em uma única lista encadeada à medida que eles vão sendo inseridos na rede.

4.2.2 Barras de Transição

Uma barra de transição ou transição da rede é um estrutura que contém as seguintes informações:

Identificação: Semelhantemente aos lugares, um transição contém um nome e um identificador único;

Identificador para a fórmula seletora: É um identificador para a formula seletora associada a transição;

Conjunto de lugares de entrada: São os endereços de todos os lugares de entrada da transição;

Conjunto de lugares de saída: São os endereços de todos os lugares de saída da transição;

Conjunto de pré-condições: É o conjunto de todas as pré-condições de habilitação da transição;

Conjunto de pós-condições: É o conjunto de todas as pós-condições ou ações de disparo da transição;

Assim como os lugares, as transições também são armazenadas em uma lista encadeada. Tanto os identificadores de transições de entrada ou saída de lugares como os identificadores de lugares de entrada e saída das transições são referências explícitas (apontadores) de endereços das listas encadeadas.

4.2.3 Variáveis de Entrada

As variáveis de entrada são implementadas como conjuntos individuais de variáveis associadas a cada arco de entrada da transição, ou seja, cada par *(Lugar de Entrada, Transição)* possui um conjunto privativo de variáveis de entrada.

Uma variável de entrada possui os seguintes campos:

- Predicado: O predicado da variável de entrada é um teste sobre as fichas residentes no lugar de entrada associado ao mesmo arco da variável. Os predicados podem ser do seguinte tipo:
  - o número de fichas no lugar associado é n;
  - o número de fichas de cor c no lugar associado é n;
  - o número de fichas de cor c e valor v no lugar associado é n;
- Valor: o valor de uma variável de entrada é o resultado do teste do predicado;

Vale lembrar que a cada arco está associado um conjunto de variáveis, o que permite testes de predicados como por exemplo: *No lugar de entrada existem 3 fichas de cor azul e 1 ficha de cor verde.*

4.2.4 Variáveis de Saída

Da mesma forma que as variáveis de entrada, as variáveis de saída são conjuntos associados individualmente a cada arco de saída das barras de transição. Uma variável de saída indica a ação do

disparo sobre as fichas. A ação de uma variável de saída diz respeito ao movimento/criação de fichas no momento do disparo de transições. As ações podem ser do tipo:

- movimentar n fichas de cor c do lugar $l1$ para o lugar associado;
- movimentar n fichas de cor c e valor v do lugar $l1$ para o lugar associado;
- inserir n fichas de cor c no lugar associado;
- inserir n fichas de cor c e valor v no lugar associado;

Como as variáveis de saída são conjuntos, combinações dessa variáveis dão flexibilidade ao movimento das fichas na rede.

4.2.5 Memória Global

A *Memória Global* é um conjunto de variáveis (inteiras, lógicas e literais) comuns a todas as operações de pré e pós condições da rede. Estas variáveis são criadas independentemente das barras e lugares da rede. Existe também um conjunto de operações sobre elas que podem ser utilizadas nas pré e pós condições. As funções sobre as variáveis globais são as seguintes:

FindX(X): Encontra a variável X na memória global;

SetXy(X,y): Set X com y: atribui o valor parâmetro y à variável X;

SumXY(X,Y): Soma as variáveis X e Y e lança o valor em X;

MultXY(X,Y): Multiplica as variáveis X e Y e lança o valor na variável X;

Eval(X,Y, op): *op* é um operador do conjunto $\{>, <, =, !\}$. O operador ! significa *diferente de*; os demais tem o próprio significado aritimético. A função Eval avalia se X *op* Y é verdade.

4.2.6 Pré-Condições

As pré-condições são predicados associados às variáveis globais da rede. Na verificação de habilitação de uma barra de transição suas pré condições são examinadas com relação ao estado atual

das variáveis globais envolvidas. Cada barra de transição possui um conjunto independente de pré condições que podem referenciar variáveis em comum. Basicamente as pré condições são formadas com a função Eval, podendo ser do tipo:

- $X = Y$: Eval(X, Y, =);
- $X > Y$: Eval(X, Y, >);
- $X < Y$: Eval(X, Y, <);
- $X \; ! \; Y$: Eval(X, Y, !);

4.2.7 Pós Condições

As pós condições são as ações realizadas sobre as variáveis globais da rede no disparo de uma barra de transição. Estas ações compreendem:

- atribuições de valores a variáveis;
- soma de duas variáveis;
- multiplicação de duas variáveis;
- atribuições de valores de variáveis a outras variáveis;

4.2.8 Atualização de Variáveis de Entrada

Para a habilitação de uma transição, os valores das variáveis de entrada são testados. Este valores, no momento do teste de habilitação necessitam estar atualizados. Para esta atualização existe a função *Atualiza Variáveis de Entrada (AtualizaInputVar)*. Esta função avalia os predicados das variáveis de entrada com base nas fichas contidas nos lugares de entrada associados e "seta" os valores das variáveis como *Verdadeiro* ou *Falso* de acordo com o resultado da avaliação.

4.2.9 Fórmula Seletora

A *Fórmula Seletora (SelectForm* constitui um teste dos conjuntos de variáveis de entrada de uma transição de acordo com um predicado. Trata-se de uma função lógica cujo valor de retorno é um

dos critérios de avaliação do teste de habilitação da barra associada. Cada barra possui apenas uma fórmula seletora que é a responsável por testar seus conjuntos de variáveis de entrada. Os predicados das fórmulas seletoras podem ter as seguintes construções:

- todas as variáveis de entrada possuem valor verdadeiro;
- todas as variáveis do conjunto V possuem valor verdadeiro;
- pelo menos uma variável do conjunto V possui valor verdadeiro;

4.2.10 Avaliação de Pre-Condições

A avaliação de pré-condições junto com a fórmula seletora constituem os dois critérios do teste de habilitação de uma transição. O conjunto de pre-condições de uma barra de transição são avaliadas pela função *avalia pré-condições (EvalPreCond)*. Se uma das pré-condições falhar, a função retorna um valor *Falso*. Se todas as pré-condições forem verdadeiras, a função retorna o valor verdadeiro.

4.2.11 Função de Habilitação

Para o disparo de uma transição, esta deve estar habilitada. Uma transição estar habilitada significa que os dois critérios de habilitação devem ser atendidos, ou seja, a *fórmula seletora* e a função *avalia pré condições*, ambas devem retornar o valor *Verdadeiro*. Sendo assim, a função de habilitação de transições consiste apenas na conjunção das duas funções citadas. Se uma delas retornar *Falso* a transição não está habilitada e a função de teste retorna *Falso* também. Se as duas funções retornarem *Verdadeiro* á função de habilitação retorna *Verdadeiro* e a transição poderá ser disparada.

4.2.12 Execução de ações de Variáveis de Saída

A execução de ações de variáveis de saída é desempenhada pela função *Ações de Disparo*. Esta função é responsável pela execução de cada variável de saída de uma transição. Como cada par *(Transição, Lugar de saída)* possui um conjunto de variáveis de saída, para a execução de todas as ações de variáveis de saída, a função deve ser

executada tantas vezes quanto forem o número de elementos desse conjunto e para todos os lugares.

4.2.13 Execução de Pós-Condições

A execução de pós-condições de disparo é desempenhada pela função *Executa Pós Condições (Action)*. Esta função exerce ações sobre as variáveis globais da rede associadas as expressões das pós-condições. De acordo com o identificador da pós condição, a função executa uma das operações de manipulação da memória global (atribuição, soma e multiplicação).

4.2.14 Função de Disparo

O disparo de uma transição é desempenhado pela função *Disparar (Fire)*. Esta função só é executada sobre transições habilitadas. Ela é responsável pela execução das funções *Ações de Disparo* e *Executa Pós-Condições* para todas as variáveis de saída e pós-condições da transição.

Para uma visualização melhor do disparo de uma transição considere o disparo da transição T_1 da figura 4.1. Avaliados os predicados das variáveis de entrada (s_2 em P_1 e s_1 em P_2), estas foram "setadas" com valor *Verdadeiro*. Assim, a fórmula seletora e a pré-condição ($x < 1$) são verdadeiras o que implica que a transição está habilitada. Quando a ação de disparo é executada, as fichas dos lugares de entrada são removidas, a pós condição ($x = 0$) é executada e a ficha s_4 é depositada no lugar P_3.

4.3 Estruturas e Funções de Simulação

As funções e estruturas de simulação implementadas podem ser divididas em dois grupos, um relacionado a *Processos* e outro relacionado a *Estados*. As construções relacionadas a estados, usa tanto funções e estruturas internas quanto implementações relacionadas a processos.

4.3.1 Processos

As informações sobre processos são mantidas em uma estrutura chamada de *bloco de controle do processo (bcp)*. Esta estrutura possui

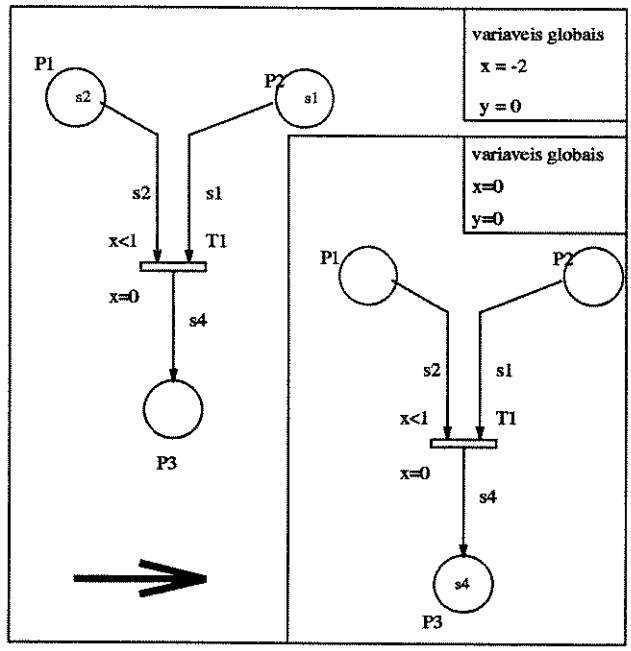

Figura 4.1: Disparo da transição T1

as seguintes informações:

Identificador: É um Identificador único do processo;

Nome: Nome do processo;

Transição apta: É a transição do processo que está apta para ser executada. Esta transição é uma referência à barra de transição correspondente da rede;

Identificador da fila de entrada: Variável que identifica o lugar correspondente à fila de entrada do processo;

Conjunto de transições: É o conjunto que identifica todas as transições da rede que pertencem ao processo;

Conjunto de Estados: É o conjunto de identificadores dos lugares da rede que representam os estados que o processo pode assumir;

Sobre esta estrutura, são realizadas as operações que veremos a seguir.

Verificação do estado do processo

Para que um processo execute, é necessário que alguma transição deste esteja habilitada. A função *Está Pronto* verifica se uma das transições do processo está habilitada. Quando a função encontra uma transição habilitada, ela atribui ao campo de informação *Transição apta* o identificador da mesma. Como cada processo sempre possui no máximo uma única transição habilitada, a procura é interrompida assim que uma transição habilitada é encontrada.

Alocação

A função responsável pela alocação dos processos é a *Alocar (Allocate)*. Esta função verifica se cada um dos processos está pronto para executar através da função *Está Pronto (IsReady)*. Em caso afirmativo, o identificador do processo pronto é inserido na fila de execução.

A ordem de alocação dos processo é estabelecida na definição do esquema de alocação que será utilizado na simulação. Os esquemas de alocação serão visto posteriormente.

Execução dos processos

A função de execução *(Execute)* dos processos retira os processos da fila de prontos e executa suas transições que estiverem aptas, respeitando o esquema de execução definido na seção de simulação.

Para alocação e execução de processo existe a *Fila de Prontos*, uma estrutura de dados onde os processos alocados são inseridos para serem executados. Quando um processo estiver na fila de prontos o mesmo pode ser desabilitado ou alterar sua função de transição. Quando o processo fica desabilitado, ele é retirado da fila sem executar. Quando ele altera sua função de transição, a sua execução irá depender do critério de execução definido para a seção de simulação.

4.3.2 Estados

As operações realizadas sobre os estados constituem basicamente da geração de sequências de estados do sistema e da visualização do estado atual do sistema. Para permitir a geração de estados foi definida uma estrutura composta dos seguintes campos de informação:

Identificador Id: É um identificador que é único para cada estado;

Origens do estado: É um conjunto de identificadores de estados que deram origem ao estado *Id*;

Estados dos processos: É um conjunto que indica o estado de cada processo no estado *Id*. Estado de processo é um estado que um processo pode estar durante a execução do sistema. O processo *Main* do sistema *Demon Game*, por exemplo, pode estar ou inativo, ou nos estados *Game On* e *Game Off*;

Estados das filas dos processos: São os estados em que se encontram as filas dos processos, isto é, a disposição e quantidade de fichas presentes nas filas dos processos;

Saídas do Estado: É o conjunto das possíveis saídas do estado *Id*. Estas saídas correspondem às transições aptas dos processos do sistemas no estado *Id*;

A geração de sequências obtem os estados do sistema através da função *Obtem Estado (GetState)*. No início de uma sequência, esta função examina o sistema e gera o primeiro estado com suas respectivas saídas. A partir de então essas saídas são executadas e novos estados são gerados. Quando um estado gerado já existe, este não é criado novamente e uma nova origem é adicionada ao estado existente. A visualização do *Estado* do sistema é desempenhada pelas função *Informações de Processos (ProcessInformation)* e complementada com informação da memória global pela função *Estado Atual*. A função *Informações de Processos* apresenta as informações contidas no conjunto de blocos de controle de processos e indica se o sistema está inativo, parado ou ativo.

4.4 Redes de Petri: Facilidades para a construção de um Simulador

O uso de redes de Petri como base para a construção do simulador propiciou que este possuisse de maneira natural um conjunto básico de requisitos e características necessárias a um simulador de especificações formais de sistemas. Estes requisitos foram apresentados na seção 4.1.1 e vamos agora ver como eles são implementados no simulador.

4.4.1 Política de Escalonamento

Todo o processo de simulação é dirigido pelo disparo de transições. Uma ordem de execução de processos é nada mais que uma ordem de execução de transições. Cada processo possui sempre, no máximo uma transição habilitada[1] . No simulador, as transições possuem uma identificação do processo a qual pertencem. Estabelecendo uma ordem de execução de processos estaremos indiretamente estabelecendo uma ordem de execução de transições. Esta característica nos da uma certa flexibilidade para estabelecermos ordens de execução sem perda da consistência da execução do modelo. Estabelecemos no simulador esquemas de escalonamento de processos que podem ser com *ordens repetitivas invariantes, ordens repetitivas variantes* e *ordens com prioridades.* Variando a política de escalonamento, conseguimos obter os possíveis caminhos de execução do sistema. Veremos a seguir como é o funcionamento destes esquemas e a maneira como eles são obtidos.

Ordens Repetitivas Invariantes

Este esquema de escalonamento é obtido definindo-se no início da seção de simulação uma ordem fixa de execução de processos. Sejam por exemplo os processos $\{p_1,\ p_2,\ p_3,\ p_4\}$, os processos componentes do sistema S. Seja a ordem de execução definida como $E =< p_1,\ p_2,\ p_4,\ p_3\} >$. No esquema de ordens fixas invariantes, a ordem E estabelecida significa que estando o processo p_1 pronto, este será o primeiro a ser inserido na fila de execução. Se p_1 não estiver pronto e p_2 estiver, p_2 é que será o primeiro a entrar na fila de execução. O processo de alocação de processos na fila de execução segue então desta maneira, sendo que p_3 será o último a entrar na fila, caso esteja pronto. Após a execução de um processo, o estado dos processos remanescentes na fila de execução é verificado. Quando a fila de execução estiver vazia, os processos podem ser novamente escalados repetindo-se a ordem de execução estabelecida. A fila de execução dos processos tem uma associação direta com a fila de transições habilitadas no modelo. Quando o processo é executado, a transição habilitada do processo é disparada. Em seguida, para a execução do próximo processo, a transição habili-

[1]Isto se deve ao fato de que internamente os processos SDL são sequenciais não permitindo a existência de mais de uma transição habilitada ao mesmo tempo para não ocasionar conflitos de decisão.

tada correspondente a este é examinada. Caso esteja habilitada, o processo é executado. Caso contrário, o mesmo é retirado da fila. O processo se repete até que a fila esteja vazia. Daí então uma nova escalação pode ser realizada.

Ordens Repetitivas Variantes

Observe que, no esquema de ordens repetitivas invariantes, se um processo que estiver na fila de execução após a execução de outro, continuar pronto para a execução mas alterar a transição habilitada, ele será retirado da fila sem executar. As implicações desse comportamento devem ser analisadas pelo usuário. Caso este deseje alterar este esquema de maneira a garantir que o processo que alterou a transição habilitada seja executado, ele pode optar pela ordem repetitiva variante. Este esquema permite que a a transição habilitada do processo varie, sem que este seja retirado da fila de execução por esse motivo. Ordens de execução variantes são obtidas com a verificação de todas as transições dos processos remanescentes na fila após uma execução. Para cada processo, se existir pelo menos uma transição habilitada, ele permanece na fila, senão é retirado.

Ordens com Prioridades

Para este esquema, prioridades são estabelecidas para os processos. Estas prioridades são utilizadas para definir qual processo será executado. O processo de maior prioridade é examinado, se houver alguma transição habilitada, este será escalado e executado. Se nenhuma transição estiver habilitada o processo com prioridade imediatamente inferior é examinado. do mesmo modo. A lista de processos é examinada do mesmo modo sempre que um processo é executado. Isso significa que após a execução do processo p_i, ele continua pronto e é o de maior prioridade, será novamente escalado e executado.

Como pode ter sido observado, todas os três esquemas de escalação de processos estabelecidos possuem um paralelo com a manipulação direta de disparo de transições da rede que representa o modelo do sistema. Este fato não só garante a integridade da execução do sistema como também facilita a criação destes esquemas, seu aperfeiçoamento, e a criação de novos esquemas.

Um outro esquema de execução facilmente implementável é um

de ordem aleatória, no qual teríamos uma função de distribuição de probabilidades associadas aos processos. A ordem de execução dos mesmo então seguiria esta distribuição.

4.4.2 Gerenciamento de Complexidade

A execução de partes de uma especificação é limitada pelo conceito de processo. O simulador garante a execução de um processo em separado, mas não menos que isso. Caso o usuário queira simular a execução de partes de um processo, estas partes devem ser modeladas separadamente. A execução de um processo único pode ser obtida de duas maneiras: carregando-se o processo em separado ou o sistema inteiro.

Simulação de um Processo carregado em separado

Neste caso o processo que se deseja executar é isolado em um arquivo de entrada como se fosse um sistema completo. As filas de entrada dos processos com os quais o processo em questão se comunica, bem como os lugares iniciais de processos filhos, são criados. O envio de sinais para o processo pode ser simulado com o uso da função enviar-sinal.

No nível da rede de Petri, o que ocorre é a execução da rede de Petri equivalente ao processo em questão mais as suas ligações com os demais processos, formados de filas de entrada e de lugares iniciais de processos filhos do processo analisado.

A interações no sentido *processos comunicantes* $\rightarrow$ *processo analisado* são feitas com enviar-sinal, cuja função é inserir fichas especificadas no lugar correspondente à fila de entrada do processo. As interações no sentido *processo analisado* $\rightarrow$ *demais processos* podem ser observadas pelo conteúdo das filas dos processos e seus lugares iniciais.

Nesta simulação, o uso da função consumir-sinais também é necessário, pois os processos comunicantes não existem e por isso não consomem os sinais depositados nas suas pseudo-filas de entrada. Esta função também pode ser usada para desativar processos filhos, retirando o sinal do seu lugar inicial.

Simulação em Separado carregando o sistema inteiro

Carregando-se o sistema inteiro também é possível acompanhar a execução de um processo específico. Tanto a função enviar-sinal como a função primitiva executar-transição podem ser usadas quando se deseja verificar a execução isolada de um processo. A função enviar-sinal é usada com o mesmo intuito que na execução do processo isolado. Com ela é possível depositar sinais esperados na fila de entrada dos processos em questão. A função primitiva executar-transição permite escolher uma transição para ser executada dentro de todas as transições habilitadas do sistema. Assim podemos escolher para executar sempre transições do processo que se deseja analisar. Cabe aqui salientar que, quando utilizamos a função enviar-sinal com o sistema inteiro carregado, a execução deste como um todo pode perder a sua consistência.

Mais uma vez, modelos de rede de Petri possibilitam a obtenção de facilidades de simulação. Sem muito esforço um gerenciamento de complexidade básico de análise de sistemas modelados foi implementado cumprindo assim mais um requisito de uma ferramenta de simulação.

4.4.3 Informação de Status

A informação de status do sistema é formada por informações globais e informações específicas de cada processo constituinte do sistema.

Informações Globais

A informação de status do sistema é o reflexo da conjunção dos estados de cada processo[2] e do instante da simulação.

Temos como informações globais os seguintes dados:

Estado do Sistema: O estado do sistema diz respeito a sua condição de execução. Podendo ser:

- Ativo: Quando o sistema está no estado Ativo, significa que há pelo menos 1 processo que possa ser executado,

[2]o significado de estado varia no contexto deste texto. Aqui estado significa o conjunto de instâncias dos diversos componentes do processo e não o estado específico ou ponto de execução em que o processo se encontra.

ou seja, existe pelo menos um processo no sistema que possua uma transição habilitada;

- Parado: Este estado indica que não há processos prontos para executar no sistema, mas pelo menos um dos seus processos se encontra em algum estado;
- Inativo: Neste estado, todos os processos do sistema estão inativos, ou seja nao se encontram em nenhum estado. Tal estado indica que suas atividades foram terminadas completamente;

Informações de Processos

Para cada processo do sistema são apresentadas as seguintes informações:

Estado do Processo: Corresponde ao estado em que o processo se encontra. Estado aqui tem o mesmo significado que em SDL. Um processo modelado possui geralmente o mesmo número de estados que a sua especificação SDL.

Transição Habilitada: Se o processo se encontra pronto para execução, a transição habilitada correspondente é apresentada;

Fila de Entrada: Este campo apresenta um "*dump*" da fila de entrada do processo, que permite ao usuário visualisar os sinais presentes na mesma;

Com os dados apresentado na função Informações de Status é possível que o usuário tenha uma visão completa do estado em que o sistema se encontra, podendo com isso avalia-lo de imediato ou definir os próximos passos da seção de simulação.

4.4.4 Alterações no Sistema

Alterações no sistema podem ser feitas de duas formas: no arquivo fonte do modelo do sistema ou durante a seção de simulação. Na alteração do arquivo fonte o usuário pode alterar o sistema da forma que achar conveniente. O simulador não oferece nenhuma facilidade para tais alterações.

Na segunda opção o usário irá alterar o sistema carregado no simulador. Para esta operação, existem funções básicas que permitem alterações no modelo de rede Petri, e funções derivadas destas

que permitem alterações em um nível de abstração mais alto. As funções básicas de alteração compreendem as operações de inserção e eliminação de lugares e transições da rede. A seguir veremos como estas funções operam:

Operações Básicas de Alteração

São 4 as operações básicas de alteração:

inserir-lugar: Esta função insere um lugar na rede de Petri. Para ser executada, necessita dos seguintes parâmetros:

- Identificador do lugar. É o identificador interno que cada lugar da rede deve possuir. Este identificador é único para cada lugar e deve ser um número inteiro.
- Nome do lugar. É o nome que caracteriza o lugar criando uma correspondência entre este a o objeto equivalente da especificação. Este campo de informação é uma string de caracteres;
- Transições de entrada do lugar. Todas as transições que irão ser conectadas a este lugar no sentido transição $\rightarrow$ lugar são as transições de entrada do lugar, e devem ser especificadas na inserção de um novo lugar. Para cada ligação (transição $\rightarrow$ lugar), deve ser especificado o conjunto de variáveis de saída. Observe que são as variáveis de saída que são especificadas neste caso, haja visto que o sistema flui na verdade da transição para o lugar, e variáveis especificadas serão variáveis de saída da transição que será ligada ao lugar.
- Transições de saída do lugar. Todas as transições que irão ser conectadas a este lugar no sentido lugar $\rightarrow$ transição são as transições de saída do lugar, e devem ser especificadas na inserção de um novo lugar. Neste caso, para cada ligação (lugar $\rightarrow$ transição) deve ser especificado o conjunto de varíaveis de entrada relacionado ao par.
- Identificador de Processo. Após a criação do novo lugar é necessário associa-lo a um dos processo componentes do sistema. Na verdade, cada processo possui um conjunto de identificadores dos lugares da rede que o compõe. Identificado o processo a que pertence o novo lugar, o

identificador do mesmo é inserido no conjunto de identificadores de lugares do processo.

inserir-transição: inserir-transição é uma função similar a inserir-lugar. O objeto a ser inserido é uma barra de transição, que como um lugar, deve conter, um identificador único, um nome, ligações seus lugares de entrada, ligações com seus lugares de saída e o identificador do processo a que pertencem. Novamente para cada ligação (lugar → transição) estabelecida, deve ser especificado o conjunto de variáveis de entrada associado. Para as ligações (transição → lugar) o conjunto de variáveis de saída é que deve ser indicado. As outras informações associadas a uma transição e que devem ser definidas na inserção de uma nova são:

- Pre-condições
- Pos-condições
- Fórmula Seletora

eliminar-lugar: Esta função, literalmente desfaz tudo que a função inserir-lugar faz. Dado um identificador do lugar, todas as suas ligações com transições são desfeitas, seu identificador é desvinculado do processo a que pertencia e seus dados são eliminados. Não é permitida a eliminação de lugares do tipo fila.

eliminar-transição: Esta função faz também o processo inverso da função inserir-transição, eliminado as ligações e variáveis associadas, seu vínculo com o processo a que pertence e seus dados.

Para garantir a consistência entre especificação e modelo é conveniênte que, antes de alterar o modelo, o usuário faça as alterações na especificação, e só depois, aplicando as regras de transformação, altere o modelo em rede de Petri.

4.4.5 Simulação Automática

No simulador foram estabelecidas duas formas de geração automática de estados. Uma consiste na geração da árvore de estados completa do sistema a partir de um estado inicial. A outra consiste

na geração de sequência de estado segundo o esquema de execução com prioridades.

A geração da árvore de alcançabilidade começa com um estado inicial. A partir deste estado, o simulador dispara suas saídas gerando assim novos estados. Caso o estado gerado já exista, uma ligação entre o novo estado gerador e o estado já existente é feita.

Após a execução de uma saída, o sistema recupera o estado em que se encontrava e executa a proxima saída, procedendo dessa maneira até que as saídas tenham se esgotado. Esgotadas as saídas, o próximo estado da da sequência da lista é avaliado e suas saídas (se existirem) são executadas. Infelizmente, este processo se depara com a explosão de estados. Este fato foi observado com maior clareza na simulação do sistema de comutação telefônica, cuja árvore de alcançabilidade possuia um número de estado da ordem de dezenas de milhares.

No contexto do simulador implementado, a simulação automática utilizando o algorítmo da árvore de estados alcançáveis deve ser utilizada comedidamente se quisermos evitar a explosão do espaço de estados.

Definindo um estado inicial, limitando o número de estados gerados utilizando esquemas de geração de estado poderemos obter informações sobre o comportamento do sistema sem nos depararmos com alguns milhares de estados. Podemos por exemplo, submeter a geração de estados ao esquema de prioridade já discutido na seção sobre política de escalonamento. Dessa maneira a árvore de estados irá crescer segundo o esquema de prioridades, ou seja, sempre uma única saída de cada estado gerado será executada. Esta saída será aquela relacionada ao processo de maior prioridade. Dessa maneira obtemos uma forma de gerar automaticamente caminhos de execução passíveis de análise. O comando *sequência de estados (seq-est)* do monitor utiliza a geração de sequência com o esquema de prioridades.

4.5 Exemplo de Simulação

Nesta seção serão realizadas simulações para exemplificar como os recursos do simulador podem ser utilizados para a análise de um sistema . Para submeter sistemas ao SIM, opós terem sido modelados com as regras de transformação para rede de Petri estabelecidas, um arquivo ASCII que representa o modelo deve ser

gerado. Este arquivo deve possuir o seguinte formato:

Número de processos
proc $_1$ *QinProc* $_1$
Número de transi,ões
$t_1 \ldots t_n$
Número de estados
$e_1 \ldots e_n$
$proc_2 \ldots proc_n$
Número total de transições da rede
$t_1 t_n$
Número de variáveis da rede
$v_1 \ldots v_n$
Número de sinais utilizados na rede
$s_1 \ldots s_n$

Na especificação de cada transição estão identificados seus lugares de entrada e variáveis de entrada associadas a cada par, lugares de saída e variáveis de saída associadas a cada par, bem como a fórmula seletora e as pre e pos condições de disparo. O exemplo que será utilizado aqui será o Demon Game cuja modelagem pode ser vista na figura 3.12.

4.5.1 Simulação do sistema Demon Game

Esta simulação tem o objetivo de ilustrar os comandos do simulador bem como a maneira conveniente de usá-los. Ao executar a chamada do programa sim, teremos de início o "menu" de opções do simulador:

```
******* SIM ********
** Simulador de protocolos de comunicacao **
 1) Inserir barras de transicao
 2) Inserir lugares
 3) Criacao de Variaveis Globais da RP
 4) Mostrar rede
 5) Remover barra de transicao
 6) Remover lugar
 7) Alterar variaveis do sistema
 8) Ler arquivo de rede
 9) Gerar grafo de estados
10) Monitor de simulacao
 0) Finalizar
Opcao Desejada: 8
nome do arquivo da rede:demo.rpm
Inicializacao da rede:
Nome do lugar:Init_Env
Numero de fichas do lugar Init_Env:1
cor do token: ctr
Valor do token:0
terminar inicializacao?[s,n]s
```

Escolhida a opção de leitura de arquivo de rede, o nome do arquivo deve ser indicado. Posteriormente, a inicialização da rede pode ser feita. A rede foi iniciada com um ficha de controle no lugar Init_Env. A partir deste ponto podemos entrar na opção **monitor de simulação**. Esta opção do simulador ativa um módulo do programa que permite ao usuário realizar seções de simulação utilizando os comandos implementados. Um deste comandos é o **help**. Este comando apresenta os demais comandos do simulador e o conjunto de sinais utilizados no sistema ativo.

```
Proximo comando:help
Lista de comandos:
[alocar]          : aloca processos
na fila de prontos

[definir-esquema] : definir esquema
de alocacao execucao

[enviar-sinal]    : envia um sinal
 para um lugar da rede

[executar]        : executar processos
alocados

[reiniciar]       : reinicializa rede
[nova-transicao]  : dispara um transicao
habilitada

[estado-atual]    : fornece o estado do sistema

[seq-est]         : gera sequencia de
 estados a partir de um estado inicial
 e grava em arquivo

[ate-parar]       : aloca e executa
ate que nao haja mais alocacoes

[end]             : termina a execucao do monitor

Lista de cores dos sinais:
ctr
ng
eg
go
win
lose
probe
bump
result
score
fim
T
Tdemon
CancelT
Tcanceled
Proximo comando:
```

No início da simulação o esquema de execução dos processos deve ser definido. Para isso usamos o comando *definir-esquema.* Este comando apresenta as três opções de ordens de execução implementadas, e em seguida pede a ordem de prioridades dos processos.

```
Proximo comando:definir-esquema
Entre com o esquema de execucao:
Ordens Repetivas Invariantes: (1)
Ordens Repetitivas Variantes: (2)
Ordens com Prioridades:       (3)
 Escolha uma das opcoes acima: 1
Ordem de execucao dos processos:
Processo Main :: Pid: 1
Processo Game :: Pid: 2
Processo Demon :: Pid: 3
Processo Timer :: Pid: 4
Processo Ambiente :: Pid: 5
prioridade maxima: 0
```

```
Prioridade 0 : 2
Prioridade 1 : 3
Prioridade 2 : 4
Prioridade 3 : 1
Prioridade 4 : 5
Proximo comando:
```

Neste ponto podemos iniciar a execução de nosso sistema, usando o comando **alocar** e posteriormente o comando *executar*.

```
Proximo comando:alocar
Processos alocados:
Ambiente
Proximo comando:executar
Disparando transicao Iniciar_jogo
Proximo comando:
```

Inicialmente o único processo alocado foi o Ambiente. Vejamos agora, utilizando o comando *estado-atual* qual a situação do sistema.

```
Proximo comando:estado-atual
Estado dos processos:
==========================================
Processo Main :: Pid: 1
Estado atual: GameOff
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Game :: Pid: 2
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Demon :: Pid: 3
Estado atual: DemonInit
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: Iniciar_Demon
==========================================
Processo Timer :: Pid: 4
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Ambiente :: Pid: 5
Estado atual: Play
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
------------------------------------------
------------------------------------------

Estado do sistema: ATIVO
Variaveis globais da rede:
Variavel v0 - Valor: 0
Variavel v1 - Valor: 1
Variavel m1 - Valor: -1
Variavel points - Valor: 0
Proximo comando:
```

O comando estado-atual mostra o estado de cada processo, suas filas e a transição que estiver habilitada. Por fim o estado do sistema é indicado bem como suas variáveis globais. O processo **Main** está no estado **GameOff** a espera do sinal *new game* para dar início ao jogo. Através do comando *enviar-sinal* podemos enviar o sinal *new game* para o processo Main.

```
Proximo comando:enviar-sinal
Destino:QinMain
cor do sinal:ng
valor associado ao sinal ng: 0
Proximo comando:alocar
Processos alocados:
Demon
Main
Proximo comando:executar
Disparando transicao Iniciar_Demon
Disparando transicao Ir_Para_GameOn
Proximo comando:estado-atual
Estado dos processos:
==========================================
Processo Main :: Pid: 1
Estado atual: GameOn
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Game :: Pid: 2
Estado atual: Losing
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Demon :: Pid: 3
Estado atual: Generate
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Timer :: Pid: 4
Estado atual: TimerIdle
Sinais na  fila de entrada:
cor do sinal: Tdemon
Numero de sinais na fila: 1
Transicao Habilitada: Ir_Para_Envio_de_T
==========================================
Processo Ambiente :: Pid: 5
Estado atual: Play
```

```
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
-----------------------------------------
-----------------------------------------

Estado do sistema: ATIVO
Variaveis globais da rede:
Variavel v0 - Valor: 0
Variavel v1 - Valor: 1
Variavel m1 - Valor: -1
Variavel points - Valor: 0
```

O comportamento do sistema DemonGame é o seguinte: Existe um temporizador que envia sinais T para o processo **Demon** como um sistema de "clock". Quando o processo **Demon** recebe o sinal do temporizador, este envia um sinal *bump* para o processo **Game**. O sinal bump tem a função de alternar o estado do processo **Game**. Se este estiver no estado **Loosing** vai para **Winning** e vice-versa. O ambiente deve interagir com o jogo enviando sinais *probe* para o processo *Game*. Se Game estiver em Winning, o jogador ganha 1 ponto (acrecido na variável *points*). Se estiver em Loosing, ele perde um ponto.

Como não contamos com um temporizador automático, toda vez que o processo **Demon** enviar um sinal *bump*, ele envia um novo sinal de requisição para o temporizador. Isto gera um sistema cíclico que deve ser controlado. Na simulação o envio de um sinal do temporizador pode ser inibido pelo pedido de cancelamento do sinal, *cancelT*.

Usando o comando *ate-parar* podemos executar o sistema um número determinado de vezes. Assim, dependendo do número de vezes que executa, o processo Game poderá parar ou em **Loosing** ou em **Winning**.

```
Proximo comando:ate-parar
Indicar ponto de parada[s/n]:s
Numero de interacoes antes de parar:7
Processos alocados:
Timer
Execucao[ 1 ]
Disparando transicao Ir_Para_Envio_de_T
Processos alocados:
Timer
Execucao[ 2 ]
Disparando transicao Enviar_T
Processos alocados:
Demon
Execucao[ 3 ]
Disparando transicao Enviar_Bump
Processos alocados:
Game
Timer
Execucao[ 4 ]
Disparando transicao Ir_para_Winning
Disparando transicao Ir_Para_Envio_de_T
Processos alocados:
Timer
Execucao[ 5 ]
Disparando transicao Enviar_T
Processos alocados:
Demon
Execucao[ 6 ]
Disparando transicao Enviar_Bump
Processos alocados:
Game
Timer
Execucao[ 7 ]
Disparando transicao Ir_para_Losing
Disparando transicao Ir_Para_Envio_de_T
Proximo comando:enviar-sinal
Destino:QinGame
cor do sinal:probe
valor associado ao sinal probe: 0
Proximo comando:alocar
Processos alocados:
Game
Timer
Proximo comando:executar
Disparando transicao Enviar_sinal_lose
Disparando transicao Enviar_T
Proximo comando:alocar
Processos alocados:
Demon
Proximo comando:executar
Disparando transicao Enviar_Bump
Proximo comando:alocar
Processos alocados:
Game
Timer
Proximo comando:executar
Disparando transicao Ir_para_Winning
Disparando transicao Ir_Para_Envio_de_T
Proximo comando:enviar-sinal
Destino:QinGame
cor do sinal:probe
valor associado ao sinal probe: 0
```

```
Proximo comando:enviar-sinal
Destino:QinGame
cor do sinal:probe
valor associado ao sinal probe: 0
Proximo comando:ate-parar
Indicar ponto de parada[s/n]:s
Numero de interacoes antes de parar:5
Processos alocados:
Game
Timer
Execucao[ 1 ]
Disparando transicao Ir_para_Losing
Disparando transicao Enviar_T
Processos alocados:
Game
Demon
Execucao[ 2 ]
Disparando transicao Enviar_sinal_lose
Disparando transicao Enviar_Bump
Processos alocados:
Game
Timer
Execucao[ 3 ]
Disparando transicao Enviar_sinal_lose
Disparando transicao Ir_Para_Envio_de_T
Processos alocados:
Timer
Execucao[ 4 ]
Disparando transicao Enviar_T
Processos alocados:
Demon
Execucao[ 5 ]
Disparando transicao Enviar_Bump
Proximo comando:estado-atual
Estado dos processos:
==========================================
Processo Main :: Pid: 1
Estado atual: GameOn
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Game :: Pid: 2
Estado atual: Losing
Sinais na  fila de entrada:
cor do sinal: bump
Numero de sinais na fila: 1
Transicao Habilitada: Ir_para_Winning
==========================================
Processo Demon :: Pid: 3
Estado atual: Generate
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Timer :: Pid: 4
Estado atual: TimerIdle
Sinais na  fila de entrada:
cor do sinal: Tdemon
Numero de sinais na fila: 1
Transicao Habilitada: Ir_Para_Envio_de_T
==========================================
Processo Ambiente :: Pid: 5
Estado atual: Play
Sinais na  fila de entrada:
cor do sinal: lose
cor do sinal: lose
cor do sinal: lose
Numero de sinais na fila: 3
Transicao Habilitada: nenhuma
------------------------------------------
------------------------------------------

Estado do sistema: ATIVO
Variaveis globais da rede:
Variavel v0 - Valor: 0
Variavel v1 - Valor: 1
Variavel m1 - Valor: -1
Variavel points - Valor: -3
```

Todos os sinais *probe* enviados chegaram ao processo Game quando este estava no estado Loosing. Executando uma única vez o processo Game vai agora para o estado Winning. Se um sinal *probe* depois disso o "jogador" ira fazer 1 ponto aumentando de score para -2.

```
Proximo comando:alocar
Processos alocados:
Game
Timer
Proximo comando:executar
Disparando transicao Ir_para_Winning
Disparando transicao Ir_Para_Envio_de_T
Proximo comando:enviar-sinal
Destino:QinGame
cor do sinal:probe
valor associado ao sinal probe: 0
Proximo comando:alocar
Processos alocados:
Game
Timer
Proximo comando:executar
Disparando transicao Enviar_sinal_Win
Disparando transicao Enviar_T
Proximo comando:estado-atual
Estado dos processos:
==========================================
Processo Main :: Pid: 1
Estado atual: GameOn
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Game :: Pid: 2
Estado atual: Winning
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
```

```
Processo Demon :: Pid: 3
Estado atual: Generate
Sinais na  fila de entrada:
cor do sinal: T
Numero de sinais na fila: 1
Transicao Habilitada: Enviar_Bump
==========================================
Processo Timer :: Pid: 4
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Ambiente :: Pid: 5
Estado atual: Play
Sinais na  fila de entrada:
cor do sinal: lose
cor do sinal: lose
cor do sinal: lose
cor do sinal: win
Numero de sinais na fila: 4
Transicao Habilitada: nenhuma
------------------------------------------
------------------------------------------

Estado do sistema: ATIVO
Variaveis globais da rede:
Variavel v0 - Valor: 0
Variavel v1 - Valor: 1
Variavel m1 - Valor: -1
Variavel points - Valor: -2
```

Se o usuário quiser reiniciar o sistema, basta usar o comando *reiniciar*. Para sair do monitor de simulação basta usar o comando *end* e o sistema volta para o "menu" de opções, onde pode ler novos arquivos ou alterar o modelo lido.

Nesta seção de simulação foram apresentados todos os comandos de simulação exceto o gerador de sequência de estados e a opção *Gera grafo de estados*. Estas opções serão vistas na simulação do sistema de comutação. telefônica.

Capítulo 5

Simulação de um Protocolo de Comunicação

Este capítulo tem como objetivo consolidar a aplicação do que foi desenvolvido neste trabalho convalidando os valores obtidos com o desenvolvimento teórico e prático da dissertação e aplicando os resultados deste desenvolvimento na análise de um protocolo de comunicação, tema que originou a elaboração do trabalho.

5.1 Modelagem do Telephone Exchange

Esta seção relata o processo de modelagem do sistema Telephone Exchange. Tal como o Demon Game, o Telephone Exchange faz parte do conjunto de exemplos que acompanham a ferramenta SDT.

Na sua especificação original, este sistema oferece os serviços de chamadas telefônicas locais e interurbanas para 50 assinantes. A sua primeira divisão funcional é formada pelos seguintes blocos:

Bloco Subscriber Handling A: A função deste bloco é gerenciar o estabelecimento da ligação no lado do assinante que faz a chamada;

Bloco Subscriber Handling B: Monitora o lado do assinante que recebe a chamada;

Bloco Connection Device: Faz o controle e a sincronização dos sinais trocados entre o subscriber Handling A e o Subscriber Handling B;

Bloco Account: Faz a contabilização das chamadas;

Para a modelagem desta especificação, como em qualquer processo de abstração, alguns detalhes foram modificados e outros desconsiderados. As principais diferenças entre o modelo em rede de Petri e a especificação original são:

1. *O bloco Account não será modelado.* Isto se deve ao fato de que o bloco Account não possui aspectos relevantes que o tornem necessário para a modelagem.

2. *A modelagem em rede de Petri possibilitará a existência de 1 assinante A e outro assinante B.* Nesta modelagem, temos como objetivo a verificação do comportamento dos processos na ocorrência de uma chamada telefônica. A criação de mais de uma instância dos processos não será abordada.

A **figura 6.1**[1] apresenta a visão geral do sistema na sua especificação original, com os quatro blocos funcionais e os processos que os compõe. A seguir serão apresentadas as especificações dos processos de cada bloco e suas respectivas modelagens em Rede de Petri.

5.1.1 Modelagem do Subscriber Handling A

O bloco Subscriber Handling A é o conjunto de processos responsável pelo monitoramento das ligações telefônicas do lado de quem faz a chamada. Os processos que fazem parte do bloco Subscriber Handling A são os seguintes:

Processo Monitor 1

Na especificação em SDL, o processo Monitor1 pode criar as várias instâncias do processo Subscriber Sensing. A sua modelagem em rede de Petri resume-se a espera do sinal Init, que o ambiente envia para ativar o sistema. Com a chegada do sinal Init, o processo Subscriber Sensing pode ser ativado. A **figura 6.2** mostra a especificação em SDL do processo Monitor_1, e a **figura 6.3** a sua respectiva tranformação em rede de Petri.

Processo Subscriber Sensing

Na especificação SDL, este processo cria durante a sua primeira transição uma instância do processo Tone Transmission que verifica se o número de

[1]As figuras da descrição e modelagem do sistema Telephone Exchange estão no Apêndice B

ligações estabelecidas ultrapassou o limite (25). Este aspecto será desconsiderado. Assim, a modelagem em rede de Petri do Subscriber Sensing Segue os seguintes passos:

1. ativação do processo Tone Transmission e espera do sinal *Rec Off* no estado **Idle**;

2. com a chegada de *Rec Off*, o Subscriber Sensing envia o sinal *Rec Lift* para o processo Tone Transmission, ativa o temporizador, e vai diretamente para o estado **Wait_for_First_Digit**. a partir desse ponto, a modelagem segue exatamente como a especificação.

As **figuras 6.4, 6.5 e 6.6** mostram a especificação SDL do Processo Subscriber Sensing e as figuras **6.7 e 6.8** mostram a sua modelagem com rede de Petri.

Processo Tone Transmission

No processo Tone Transmission o teste sobre o número de ligações feitas é desconsiderado. O processo vai do estado **Free** diretamente para o estado **Dial Tone Phase** (após a chegada do sinal RecLift). Nesta transição, o processo Digit Reception é ativado e o sinal **Dial_Start** é transmitido ao ambiente. No estado **Dial_Tone_Phase** , o processo espera a chegada do sinal *First Digit*, ou os sinais *RecRepl* e *No_Digit*. Deste estado em diante, a modelagem segue a especificação, exceto nos seguintes pontos:

- as tarefas No_Of_DR = No_Of_DR + 1 e No_Of_DR =No_Of_DR − 1 são descartadas;

- os sinais *Start e Stop Debit* não são transmitidos;

A especificação deste processo é mostrada nas **figuras 6.9, 6.10, 6.11 e 6.12**. a modelagem é mostrada na **figuras 6.13, 6.14 e 6.15**.

Processo Digit Reception

Para o processo Digit_Reception, a maior diferença entre a modelagem e a especificação é a ausência do teste para verificar se o número da chamada de longa distância é formado de 8 dígitos. Sua especificação é apresentada nas **figuras 6.16 e 6.17**, e a modelagem é apresentada na **figura 6.18**.

5.1.2 Modelagem do Subscriber Handling B

O bloco Subscriber Handling B faz o controle das atividades do sistema no lado de quem recebe a chamada. Para tanto, o bloco possui dois processos: o processo Monitor_3 e o B_Subscriber. A especificação do processo Monitor_3 é apresentada na **figura 6.19** e, o B_Subscriber e mostrado na **figura 6.20**. A modelagem dos dois processos é apresentada na **figura 6.21**.

5.1.3 Modelagem do Connection Device

O bloco Connection device é formado por dois processos: Transfer_1 e Transfer_2. Eles são responsáveis pela transferência dos sinais de estabelecimento e término de ligações do Subscriber_Handling_A para o Subscriber_Handling_B *(Disconnection)* e vice-versa *(Connection, Engaged, Called e B_Repl)*. A Especificação do Transfer_1 é apresentada na **figura 6.22** e Transfer_2 na **figura 6.24**. A **figura 6.23** apresenta a especificação da macro Transfer, utilizada no processo Transfer_1. A modelagem dos dois processos com rede de Petri pode ser vista na **figura 6.25**.

5.2 Simulação do Telephone Exchange

5.2.1 Simulação 1

Representacao da situação em que uma pessoa retira o fone do gancho do aparelho telefônico e não disca nenhum número.

O estado inicial do sistema tem ativos os processos Monitor_1, Timer, Monitor_3, Transfer_1 e Transfer_2 e Environment. Estes processos estão prontos para executar suas transições iniciais que os levará para seus estados Idle. Quando o processo Monitor_1 estiver em Idle, o recebimento do sinal **init** enviado pelo ambiente vai tornar o sistema pronto para fazer a ligação entre o assinante A e o assinante B.

A chegada do sinal **init** no processos Monitor_1 provoca a ativação do processo Subscriber Sensing. Quando o processo Subscriber Sensing está no estado SSA_Idle, o sinal **rec_off** que representa a retirada do fone do gancho é enviado para o mesmo.

O tratamento do sinal **rec_off** ativa o processo Tone Transmission que por sua vez ativa o processo Digit Reception. Como nem um dígito e enviado, o sistema volta a uma situação normal através do esquema de time out.

Dentro do monitor de simulação, após a definição do esquema de execução dos processos. O estado do sistema pode ser visto com o comando estado-atual.

```
******* SIM ********
** Simulador de protocolos de comunicacao **
 1) Inserir barras de transicao
 2) Inserir lugares
 3) Criacao de Variaveis Globais da RP
 4) Mostrar rede
 5) Remover barra de transicao
 6) Remover lugar
 7) Alterar variaveis do sistema
 8) Ler arquivo de rede
 9) Gerar sequencia de  estados
10) Monitor de simulacao
 0) Finalizar
Opcao Desejada:10
Proximo comando:definir-esquema
Entre com o esquema de execucao:
Ordens Repetivas Invariantes: (1)
Ordens Repetitivas Variantes: (2)
Ordens com Prioridades:       (3)
 Escolha uma das opcoes acima: 1
Ordem de execucao dos processos:
Processo Monitor_1 :: Pid: 1
Processo SubscriberSensing :: Pid: 2
Processo Tone_Transmission :: Pid: 3
Processo Digit_Reception :: Pid: 4
Processo Monitor_3 :: Pid: 5
Processo BSubscriber :: Pid: 6
Processo Transfer_1 :: Pid: 7
Processo Transfer_2 :: Pid: 8
Processo Timer :: Pid: 9
Processo Environment :: Pid: 10
prioridade maxima: 0
Prioridade 0 : 2
Prioridade 1 : 3
Prioridade 2 : 4
Prioridade 3 : 7
Prioridade 4 : 8
Prioridade 5 : 1
Prioridade 6 : 5
Prioridade 7 : 6
Prioridade 8 : 9
Prioridade 9 : 10
Ordem de execucao:
p1 p2 p3 p6 p7 p0 p4 p5 p8 p9
Proximo comando:estado-atual
Estado atual da rede:
Estado dos processos:
----------------------------------------
----------------------------------------
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 1
----------------------------------------
----------------------------------------
Processo SubscriberSensing :: Pid: 2
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
```

```
-----------------------------------------
Processo Tone_Transmission :: Pid: 3
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
-----------------------------------------
-----------------------------------------
Processo Digit_Reception :: Pid: 4
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
-----------------------------------------
-----------------------------------------
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 49
-----------------------------------------
-----------------------------------------
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
-----------------------------------------
-----------------------------------------
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 58
-----------------------------------------
-----------------------------------------
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 62
-----------------------------------------
-----------------------------------------
Processo Timer :: Pid: 9
Estado atual: TimerInit
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 68
-----------------------------------------
-----------------------------------------
Processo Environment :: Pid: 10
Estado atual: Env_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 72
-----------------------------------------
-----------------------------------------

Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 0
Variavel x - Valor: 0
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor: 0
```

Com a utilização do comando enviar-sinal, o sinal **init** é enviado para para a fila do processo Monitor_1, e, com o processo Subscriber Sensing já ativo o sinal rec_off é enviado para o mesmo.

```
Proximo comando:alocar
Processos alocados:
Transfer_1
Transfer_2
Monitor_1
Monitor_3
Timer
Environment

Proximo comando:executar
Disparando transicao IniciarTransfer_1
Disparando transicao Iniciar_Transfer_2
Disparando transicao Iniciar_Monitor1
Disparando transicao Iniciar_Monitor_3
Disparando transicao Iniciar_Timer
Disparando transicao Iniciar_ambiente

Proximo comando:enviar-sinal
Destino:QinMonitor1
cor do sinal:init
valor associado ao sinal init: 0

Proximo comando:alocar
Processos alocados:
Monitor_1
Proximo comando:executar
Disparando transicao Ativar_SSA

Proximo comando:alocar
Processos alocados:
SubscriberSensing

Proximo comando:executar
Disparando transicao Iniciar_SSA
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:rec_off
valor associado ao sinal rec_off: 0
Proximo comando:estado-atual
Estado atual da rede:
Estado dos processos:
-----------------------------------------
-----------------------------------------
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
```

```
----------------------------------------
----------------------------------------
Processo SubscriberSensing :: Pid: 2
Estado atual: SSA_Idle
Sinais na  fila de entrada:
cor do sinal: rec_off
Numero de sinais na fila: 1
Transicao Habilitada: 5
----------------------------------------
----------------------------------------
Processo Tone_Transmission :: Pid: 3
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Digit_Reception :: Pid: 4
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Timer :: Pid: 9
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Environment :: Pid: 10
Estado atual: Env_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------

Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 0
Variavel x - Valor: 0
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor: 0
Proximo comando:alocar
Processos alocados:
SubscriberSensing
Proximo comando:executar
Disparando transicao Ativar_ToneTrans

Proximo comando:alocar
Processos alocados:
Tone_Transmission
Timer
Proximo comando:executar
Disparando transicao Iniciar_Tone_Transmission
Disparando transicao Ir_para_SendTimeOutSS
Proximo comando:alocar
Processos alocados:
Tone_Transmission
Timer
Proximo comando:executar
Disparando transicao Ir_para_Dial_Tone_Phase
Disparando transicao Enviar_TimeOut
```

Com o processo Subscriber Sensing no estado "Wait for First Digit", o processo Tone Transmission em "Dial Tone Phase" e o processo Digit Reception sendo iniciado, o sistema espera por digitos na fila de entrada do processo Subscriber Sensing, o que não ocorre permitindo assim o envio de time_out pelo Timer. O Tratamento do sinal **time_out** leva o sistema ao estado de espera de ligações e envia o sinal **dial_stop** para o Ambiente.

```
Proximo comando:estado-atual
Estado atual da rede:
Estado dos processos:
----------------------------------------
----------------------------------------
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
```

```
Processo SubscriberSensing :: Pid: 2
Estado atual: WffD
Sinais na  fila de entrada:
cor do sinal: time_outSSA
Numero de sinais na fila: 1
Transicao Habilitada: 9
----------------------------------------
----------------------------------------
Processo Tone_Transmission :: Pid: 3
Estado atual: DialTonePhase
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Digit_Reception :: Pid: 4
Estado atual: DigitRecept_Init
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: 37
----------------------------------------
----------------------------------------
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Timer :: Pid: 9
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Environment :: Pid: 10
Estado atual: Env_Idle
Sinais na  fila de entrada:
cor do sinal: dial_start
Numero de sinais na fila: 1
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------

Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 0
Variavel x - Valor: 0
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor: 0
Proximo comando:ate-parar
Processos alocados:
SubscriberSensing
Digit_Reception
Disparando transicao Tratar_TimeOut
Disparando transicao Iniciar_digit_Reception
Processos alocados:
Tone_Transmission
Disparando transicao Retornar_para_Free
Processos alocados:
Digit_Reception
Disparando transicao Terminar_DR
Processos alocados:
Proximo comando:estado-atual
Estado atual da rede:
Estado dos processos:
----------------------------------------
----------------------------------------
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo SubscriberSensing :: Pid: 2
Estado atual: SSA_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Tone_Transmission :: Pid: 3
Estado atual: Free
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Digit_Reception :: Pid: 4
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Idle
Sinais na  fila de entrada:
Fila vazia
```

```
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Timer :: Pid: 9
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------
Processo Environment :: Pid: 10
Estado atual: Env_Idle
Sinais na  fila de entrada:
cor do sinal: dial_start
cor do sinal: dial_stop
Numero de sinais na fila: 2
Transicao Habilitada: nenhuma
----------------------------------------
----------------------------------------

Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 0
Variavel x - Valor: 0
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor: 0
```

5.2.2 Simulação 2

Simulação do estabelecimento de ligação entre o assinante A e o assinante B. O sistema, em seu estado inicial recebe do ambiente os sinais rec_off e os dígitos do número 430361. O tratamento destes sinais leva o sistema até o estado em que o assinante B está sendo chamado. Não existe time out para o estado onde o assinante B está sendo chamado.

Para a simulação desta situação será usado o comando *seq-est*. Este comando gera a sequência de estados do sistema a partir de um estado inicial até que este atinja um estado **Parado** ou **Inativo**. A sequência de estados é armazenada em arquivo. A seguir é apresentada a seção de simulação correspondente.

```
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:rec_off
valor associado ao sinal rec_off: 0
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 4
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 3
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 0
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 3
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 6
Proximo comando:enviar-sinal
Destino:QinSSA
cor do sinal:digit
valor associado ao sinal digit: 1
Proximo comando:estado-atual
Estado dos processos:
========================================
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
========================================
Processo SubscriberSensing :: Pid: 2
Estado atual: SSA_Idle
Sinais na  fila de entrada:
cor do sinal: rec_off
cor do sinal: digit
cor do sinal: digit
```

```
cor do sinal: digit
cor do sinal: digit
cor do sinal: digit
cor do sinal: digit
Numero de sinais na fila: 7
Transicao Habilitada: Ativar_ToneTrans
=========================================
Processo Tone_Transmission :: Pid: 3
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Digit_Reception :: Pid: 4
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Timer :: Pid: 9
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
=========================================
Processo Environment :: Pid: 10
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
-----------------------------------------
-----------------------------------------

Estado do sistema: ATIVO
Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 0
Variavel x - Valor: 0
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor: 0
Proximo comando:seq-est
```

Neste instante o sistema se encontra no estado 54 da sequência de estados gerada, a espera que o assinante B atenda o telefone. Este evento pode ser simulado com o envio do sinal *rec_off* para o processo **SubscriberB**. Com o envio deste sinal a conexão será estabelecida.

```
Proximo comando:enviar-sinal
Destino:QinBSubscriber
cor do sinal:rec_off
valor associado ao sinal rec_off: 0
Proximo comando:seq-est
```

Usando mais uma vez o comand seq-est, o sistema estabelece a conexao entre os assinantes.

```
::=======================================::
Estado ID: 1 :: Estado Step: 0
Processo Monitor_1 ::
Estado: Monitor1_Idle
Processo SubscriberSensing ::
Estado: SSA_Idle
Processo Tone_Transmission ::
Estado: RingTonePhase
Processo Digit_Reception ::
Estado: Wait_for_Termination
Processo Monitor_3 ::
Estado: Monitor3_Idle
Processo BSubscriber ::
Estado: Calling
Sinal no inicio da fila: rec_off
Processo Transfer_1 ::
Estado: Transfer1_Idle
Processo Transfer_2 ::
Estado: Transfer2_Idle
Processo Timer ::
Estado: TimerIdle
Processo Environment :: Inativo
Sinal no inicio da fila: dial_start
Variaveis Globais
      nDigits -> 6
       digito -> 1
            x -> 10
           v1 -> 1
           v0 -> 0
 No_of_Digits -> 6
    TemB_AbNo -> 430361
estado: 2
::=======================================::
Estado ID: 2 :: Estado Step: 1
Processo Monitor_1 ::
```

```
Estado: Monitor1_Idle
Processo SubscriberSensing ::
Estado: SSA_Idle
Processo Tone_Transmission ::
Estado: RingTonePhase
Processo Digit_Reception ::
Estado: Wait_for_Termination
Processo Monitor_3 ::
Estado: Monitor3_Idle
Processo BSubscriber ::
Estado: SpeechB
Processo Transfer_1 ::
Estado: Transfer1_Idle
Processo Transfer_2 ::
Estado: Transfer2_Idle
Sinal no inicio da fila: b_repl
Processo Timer ::
Estado: TimerIdle
Processo Environment :: Inativo
Sinal no inicio da fila: dial_start
Variaveis Globais
      nDigits -> 6
       digito -> 1
            x -> 10
           v1 -> 1
           v0 -> 0
 No_of_Digits -> 6
    TemB_AbNo -> 430361
Origem: Estado 1
Transicao: 55
::--------------------------------::
estado: 3
::================================================::
Estado ID: 3 :: Estado Step: 2
Processo Monitor_1 ::
Estado: Monitor1_Idle
Processo SubscriberSensing ::
Estado: SSA_Idle
Processo Tone_Transmission ::
Estado: RingTonePhase
Sinal no inicio da fila: b_repl
Processo Digit_Reception ::
Estado: Wait_for_Termination
Processo Monitor_3 ::
Estado: Monitor3_Idle
Processo BSubscriber ::
Estado: SpeechB
Processo Transfer_1 ::
Estado: Transfer1_Idle
Processo Transfer_2 ::
Estado: Transfer2_Idle
Processo Timer ::
Estado: TimerIdle
Processo Environment :: Inativo
Sinal no inicio da fila: dial_start
Variaveis Globais
      nDigits -> 6
       digito -> 1
            x -> 10
           v1 -> 1
           v0 -> 0
 No_of_Digits -> 6
    TemB_AbNo -> 430361
Origem: Estado 2
Transicao: 67
::--------------------------------::
estado: 4
::================================================::
Estado ID: 4 :: Estado Step: 3
Processo Monitor_1 ::
Estado: Monitor1_Idle
Processo SubscriberSensing ::
Estado: SSA_Idle
Processo Tone_Transmission ::
Estado: SpeechA
Processo Digit_Reception ::
Estado: Wait_for_Termination
Processo Monitor_3 ::
Estado: Monitor3_Idle
Processo BSubscriber ::
Estado: SpeechB
Processo Transfer_1 ::
Estado: Transfer1_Idle
Processo Transfer_2 ::
Estado: Transfer2_Idle
Processo Timer ::
Estado: TimerIdle
Processo Environment :: Inativo
Sinal no inicio da fila: dial_start
Variaveis Globais
      nDigits -> 6
       digito -> 1
            x -> 10
           v1 -> 1
           v0 -> 0
 No_of_Digits -> 6
    TemB_AbNo -> 430361
Origem: Estado 3
Transicao: 34
::--------------------------------::
```

O envio do sinal *rec_on* para o processo BSubscriber desfaz a conexão simulando o fim da conversa. Usando o comando *ate-parar* a desconexão é realizada e o sistema vai para o estado apresentado a seguir.

```
Proximo comando:ate-parar
Proximo comando:estado-atual
Estado dos processos:
==========================================
Processo Monitor_1 :: Pid: 1
Estado atual: Monitor1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo SubscriberSensing :: Pid: 2
Estado atual: SSA_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
```

```
==========================================
Processo Tone_Transmission :: Pid: 3
Estado atual: Free
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Digit_Reception :: Pid: 4
Estado atual: Inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Monitor_3 :: Pid: 5
Estado atual: Monitor3_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo BSubscriber :: Pid: 6
Estado atual: inativo
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Transfer_1 :: Pid: 7
Estado atual: Transfer1_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Transfer_2 :: Pid: 8
Estado atual: Transfer2_Idle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Timer :: Pid: 9
Estado atual: TimerIdle
Sinais na  fila de entrada:
Fila vazia
Transicao Habilitada: nenhuma
==========================================
Processo Environment :: Pid: 10
Estado atual: inativo
Sinais na  fila de entrada:
cor do sinal: dial_start
cor do sinal: digit_finish
cor do sinal: dial_stop
cor do sinal: ring_start
cor do sinal: ring_start
cor do sinal: TimeOutSSACanceled
cor do sinal: TimeOutSSACanceled
cor do sinal: TimeOutSSACanceled
cor do sinal: TimeOutSSACanceled
cor do sinal: TimeOutSSACanceled
cor do sinal: rec_off
cor do sinal: ring_stop
Numero de sinais na fila: 12
Transicao Habilitada: nenhuma
------------------------------------------
------------------------------------------
```

```
Estado do sistema: PARADO
Variaveis globais da rede:
Variavel nDigits - Valor: 0
Variavel digito - Valor: 1
Variavel x - Valor: 10
Variavel v1 - Valor: 1
Variavel v0 - Valor: 0
Variavel No_of_Digits - Valor: 0
Variavel TemB_AbNo - Valor:0
```

Capítulo 6

Objetivos, Resultados e Análise Conclusiva

Esta dissertação foi o resultado do esforço equilibrado em cada uma de suas etapas. Seus resultados e contribuições podem ser sintetisados em dois objetos: A proposta da ferramenta de simulação, que longe de ser a última palavra sobre o tema, servirá sobretudo, como material de referência a novas pesquisas e projetos; e a dissertação por ela mesma, sob o ponto de vista didático. Neste capítulo, será aberta uma discussão final com o propósito de levantar os aspectos relevantes sobre o que foi pretendido e alcançado nestas duas direções.

6.1 Análise de Objetivos e Contribuições

O objetivo do capítulo 1 desta dissertação foi inserir o leitor no contexto geral do trabalho. A partir de então, a cada capítulo, contribuições podem ser tiradas individualmente.

No capítulo dois, o estudo de redes de Petri nos levou aonde queriamos chegar, ou seja, no estabelecimento de um modelo computacional de rede de Petri de alto nível consistente, que comportasse a modelagem de sistemas previamente especificados em SDL. Este modelo possibilitou:

1. a modelagem de sistemas especificados em SDL;

2. a implementação do simulador.

Além disso, neste capítulo foram agrupados conceitos, com os quais foi permido a construção de funções que fizessem cumprir os requisitos requeridos em um programa simulador.

Cabe aqui salientar que a teoria de redes de Petri pode e deve, como trabalho futuro, ser mais explorada, visando a construção de novas funções que aumentem a capacidade de análise do simulador.

Definido o modelo de rede de Petri, o próximo problema a ser resolvido era verificar se o modelo definido era realmente capaz de modelar especificações SDL, e se era, como faze-lo. Este foi o objetivo do capítulo 3, que culminou na determinação de um conjunto de regras de transformação de SDL para redes de Petri e construções de modelagens básicas para tal fim.

Determinado o modelo de rede de Petri e comprovada a sua capacidade de modelagem de especificações SDL, chegamos à implementação do simulador, tema do capítulo quatro e objetivo central do trabalho. Como contribuição encontrada neste capítulo podemos colocar uma única, porém extensa: o projeto e protótipo do simulador.

Nos moldes em que se encontra, o sistema não pode ser dito operacional, mesmo porque o objetivo não era a construção de um software de qualidade comercial e sim a definição das bases para construção de um. Mesmo assim, como pode ter sido verificado, o programa atende aos requisitos determinados, é consistente com o modelo de rede implementado, é passível de alterações e permitiu a simulação de uma extensa parte de um sistema de comutação telefônica.

A arquitetura e os conceitos inseridos no protótipo podem ser usados integralmente para a implementação de sistemas simuladores futuros. Mesmo a sua implementação pode ser incrementada facilmente, de maneira a permitir o uso efetivo como ferramenta de simulação por outras pessoas.

Por fim, no capítulo 5 apresentamos um processo de simulação e análise de um protocolo de uma rede de telefonia usando o simulador implementado. Este capítulo teve como objetivo principal verificar a potencialidade do simulador. Comparando o processo de simulação realizado com as seções de simulação realizadas no SDT Design Tool, podemos verificar a equiparação das funcionalidades básicas (execução de transições, informações de "status") deste último com o sistema implementado. Uma vantagem do simulador implementado, é a capacidade de realizar simulações mais versáteis por este dar um maior grau de liberdade para a execução de comandos de simulação e também pelo fato de redes de Petri utilizarem o conceito de fichas para caracterizar a dinâmica de execução dos sistemas modelados.

6.2 Trabalhos Futuros

O potencial de descrição e análise de redes de Petri é algo incontestável, que se reflete em qualquer trabalho que as utilize como ferramenta de modela-

gem. Utilizar mais o potencial imbutido na ferramenta de simulação definida e implementada certamente pode originar uma grande gama de trabalhos futuros. Estes trabalhos podem ser motivados e partir de vários pontos. Alguns podem ser desde de já estabelecidos:

1. Criação de um algoritmo de simulação baseado na relação definida por [11] chamada de *Simulação Ótima*; A partir de então poderíamos aplicar a verificação de propriedades como vivacidade e "*Coverability*", sem nos depararmos com o problema de explosão de estados.

   A simulação completa de um sistema representa o comportamento de uma rede em todos os possíveis caminhos de execução. As grandes desvantagens com respeito a este tipo de simulação consiste no grande número de caminhos de execução e no tamanho do grafo de alcançabilidade da rede. A solução apresentada por em [11], [12] e [13] para encontrar pequenas simulações que possibilitem a verificação do comportamento dinâmico de redes é a *Simulação Ótima*. Segundo [11] exitem quatro razões para que o mecanismo tenha ganho este nome:

   Expressividade: Grande número das propriedades comportamentais são comuns às simulações "completa" e "ótima", o que nos permite concluir que ambas capturam as mesmas propriedades comportamentais da rede.

   Minimizada: A simulação "ótima" não pode ser reduzida sem que perca poder de expressividade.

   Eficiente: os históricos de execução são construídos usando os menores caminhos de execução possíveis. O que implica que eventos serão observados na sua mais rápida ocorrência.

   Única Alternativa: não existe outro tipo de simulação que seja ao mesmo tempo expressiva, minima e eficiênte.

2. aplicação da métrica definida por Petri como distância Sincrônica que nos permitiria obter o grau de sincronismo entre as diversas transições e processos de uma rede;

Existem alguns passos que permitirão que a ferramenta implementada se torne operacional:

1. Viabilização de um mecanismo de tradução automática de especificações SDL para o modelo de rede de Petri estabelecido. Este é o maior entrave para que a ferramenta se torne utilizável, pois o processo de modelagem manual de sistemas em SDL para redes de Petri é trabalhosa e pode tornar inviável o restante do processo de verificação;

2. Construção de uma interface gráfica para o simulador. Esta outra medida viabilizaria ainda mais o uso da ferramenta na análise de protocolos extensos;

3. ampliação da capacidade do simulador de simular a criação de instâncias de processos. Esta facilidade permitirá uma verificação mais completa do sistema.

6.3 Nota Final

Concluiu-se que os temas estudados e apresentados nesta dissertação tem grande importância para o desenvolvimento de sistemas computacionais e de telecomunicações.
'E esperado que este trabalho venha contribuir como material de consulta, e como instrumento de divulgação dos temas abordados. Espera-se que pesquisadores de diferentes graus de conhecimento sobre o assunto consigam (ou tenham conseguido) fazer uma boa leitura do texto de modo a obter informações para levar adiante suas idéias e projetos.

Apêndice A - Implementação do Simulador

O programa simulador **SIM** foi implementado em linguagem C e sistema operacional Unix (SunOs) utilizando estações de trabalho SUN do laboratório do dep. de Telemática da FEE/UNICAMP.

A implementação do programa foi dividida nos seguintes modulos:

sim.c Implementação de funções de relacionadas a habilitação e disparo de transições e interface do programa;

learq.c Implementa a entrada de dados do programa;

inpvar.c Implementações de operações sobre Variáveis de Entrada;

outvar.c implementações de operações sobre Variáveis de Saída;

fila.c Implentações de operações de manipulação de filas e listas dinâmicas;

processos.c Implementações de manipulação de processos e geração de estados do simulador;

monitor.c Implementação do sistema monitor para a simulação;

tokens.c Implementações de funções de manipulação/movimentação de fichas.

As principais estruturas de dados utilizadas no programa estão definidas no *header file* sim.h. Dentre estas estruturas, apresentaremos aqui a definição de transições, lugares, processos e estados:

Estrutura de um lugar

```
typedef struct PLACE{
char name[30];
int peso,Id;
int nTokens;
queue Tokens; /* ponteiro para a lista de tokens presentes no lugar */
int InpVarIndP;  /*  indice das variaveis de entrada ( com relacao a transicao) associada ao lugar */
```

```
int OutVarIndP; /* indice da variavel de saida (com relacao a transicao) associada ao lugar */

                void  *Tin[n], *Tout[n]; /* vetores de ponteiros para as transicoes */
int ntin,ntout; /* numero de transicoes de entrada e de saida do lugar */
 struct PLACE* next;
}place;
```

Estrutura de uma transição

```
typedef struct TRANSITION{
char name[50];
        int peso, Id;
                int vaux[10]; /* vetor auxiliar para operacoes genericas */
                int lvaux; /* comprimento do vetor vaux */
                int nplin,nplout;
int SelectForm,  /* identificador da formula seletora */
nPreCond,nPosCond;
cond    PreCond[10],
                        PosCond[10];
        pPlace PlaceIn[n], PlaceOut[n];
          struct TRANSITION* next;

}transition;
```

Estrutura de um Processo

```
typedef struct BCP{
int Pid,Tap;

char PName[30];
char Qin[30];
                int nTrans;
int Trans[50];
                int nEstados;
char Estados[30][30];
}bcp;
```

Estrutura de um Estado

```
typedef struct ESTADO{
                  saida saidas[maxproc];
                  int nsaidas;
                  origem origens[maxproc];
                  int norigens;
                  int *gvar;
                  int estado_processo[maxproc];
                  info estado_filas[maxproc][maxFila];
                  int estado_Id,
                  step, nfichas[maxproc];

                 }estado;
```

Apêndice B - Modelagem do Sistema Telephone Exchange

Este apêndice apresenta a rede de Petri equivalente a especificação dos processos do sistema Telephone Exchange. Serão apresentas primeiro as especificações em SDL de cada processo modelado e em seguida se modelo em rede de Petri. Vale lembrar que apenas o bloco Account não foi modelado, como foi mencionado no cap. 5.

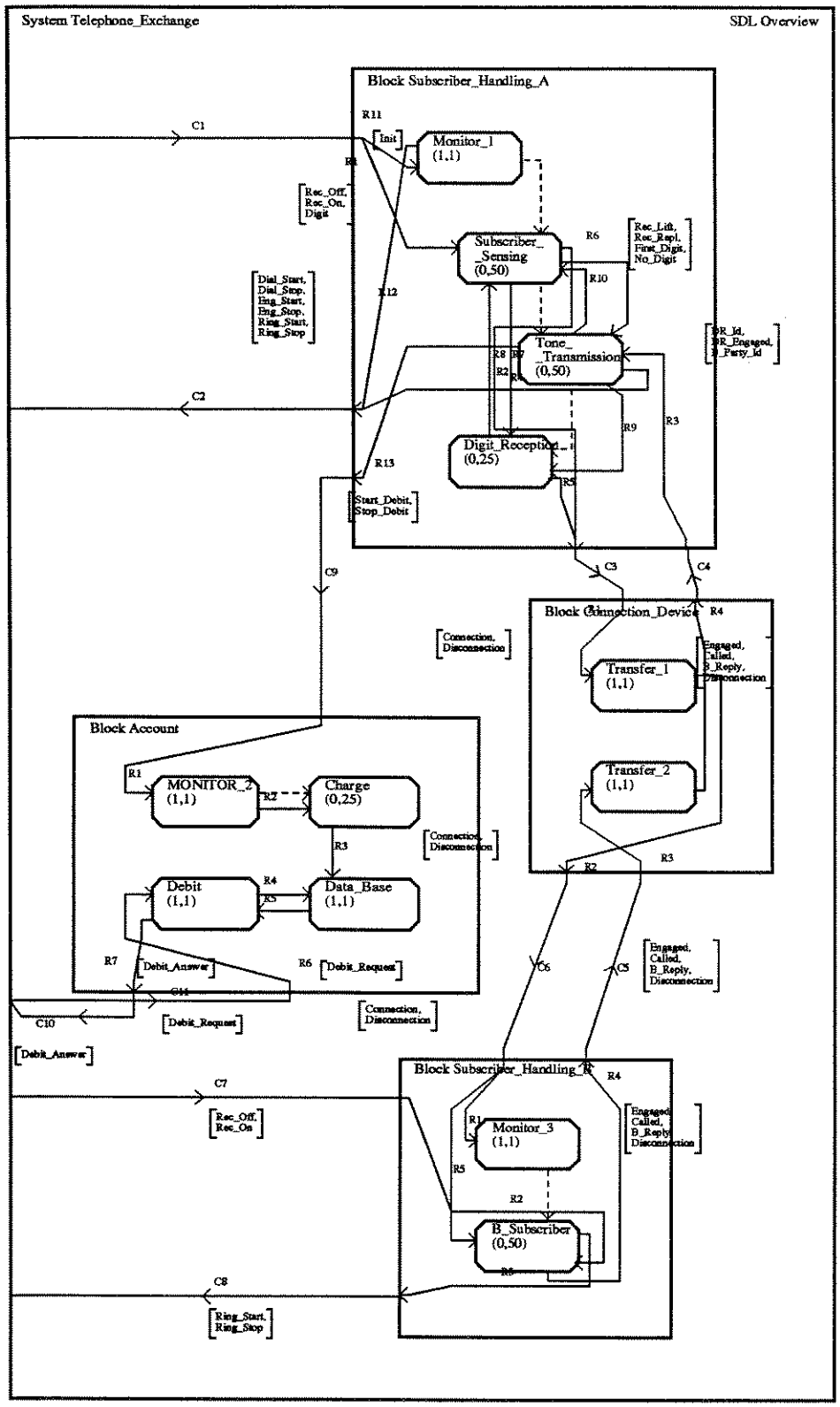

Figura 6.1: Sistema Telephone Exchange

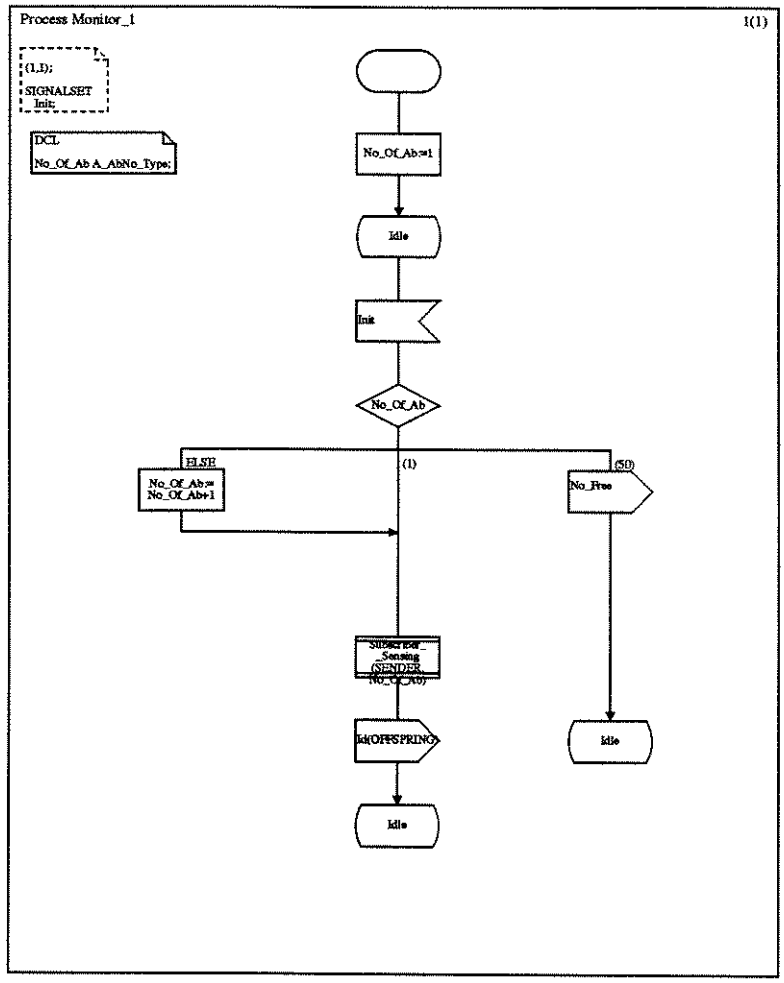

Figura 6.2: Especificação do processo Monitor 1

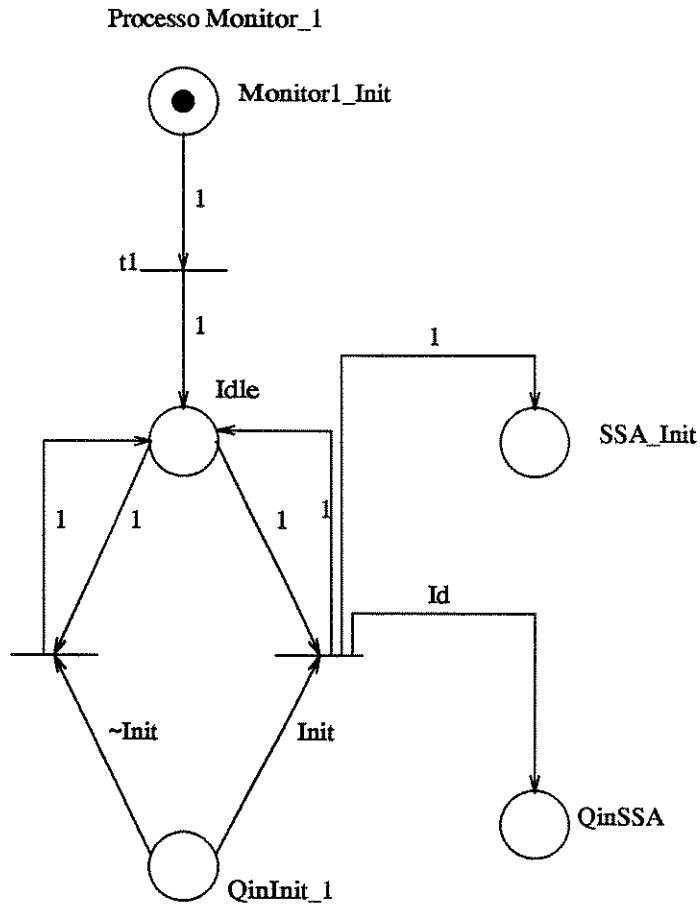

Figura 6.3: Modelagem do processo Monitor 1

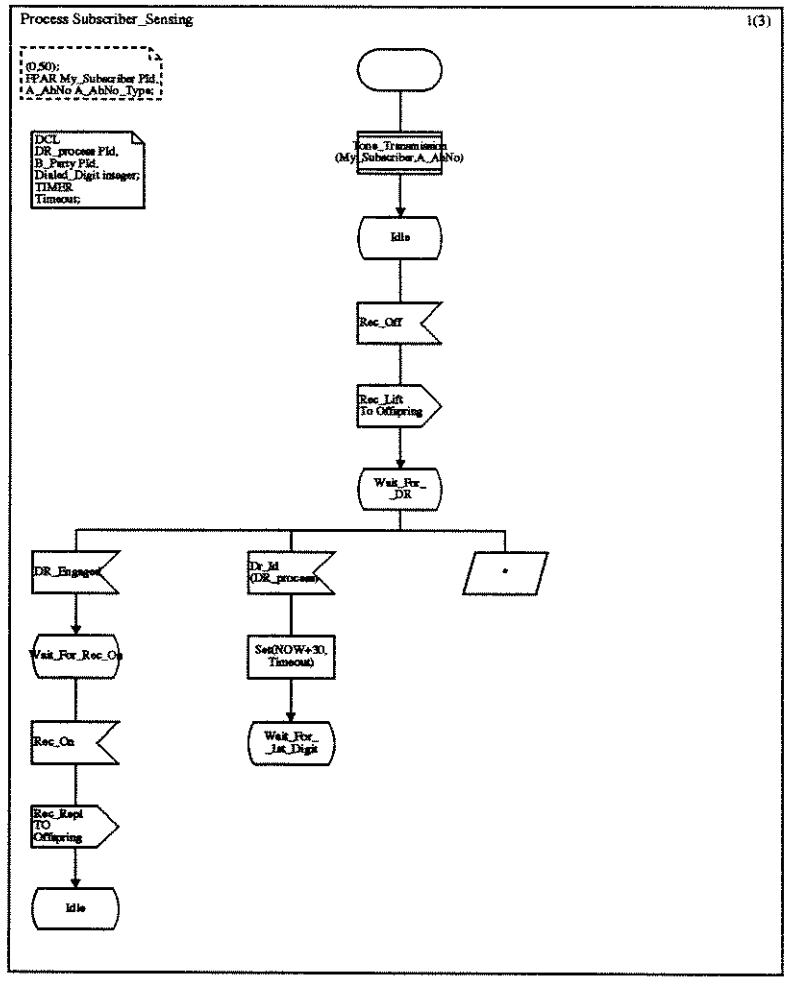

Figura 6.4: Especificação do processo Subscriber Sensing

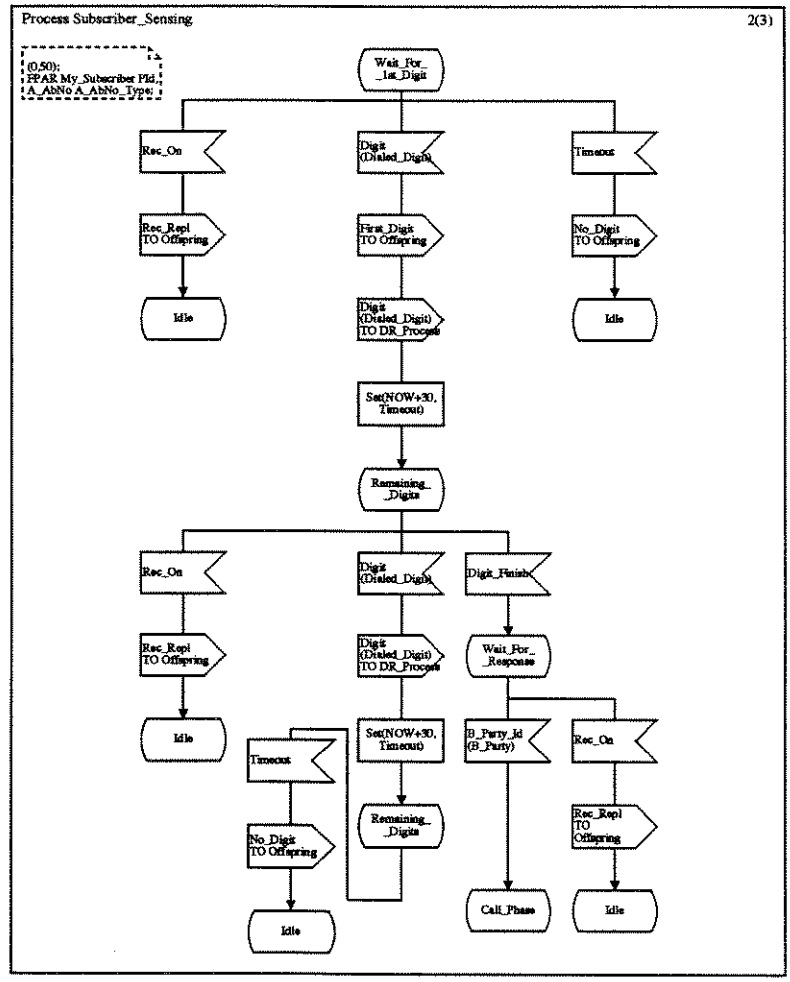

Figura 6.5: Especificação do processo Subscriber Sensing

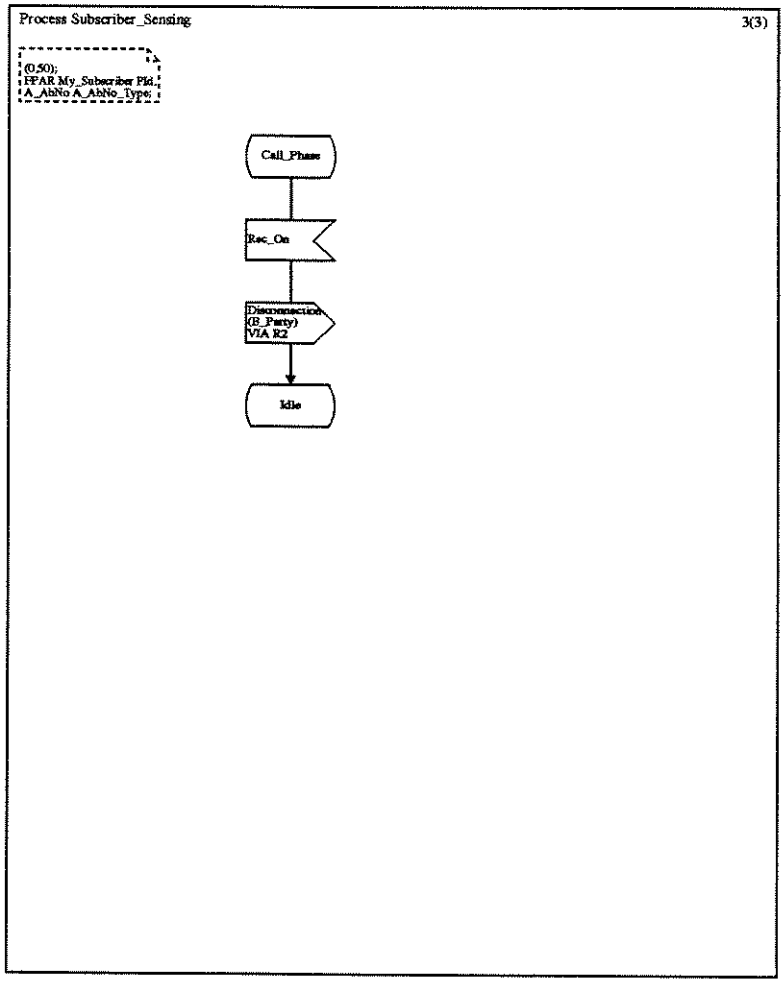

Figura 6.6: Especificação do processo Subscriber Sensing

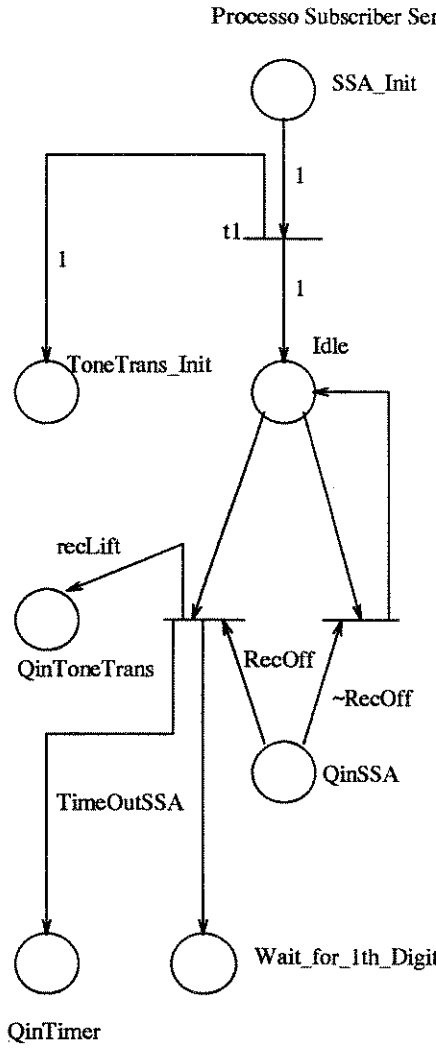

Figura 6.7: Modelagem do processo Subscriber Sensing

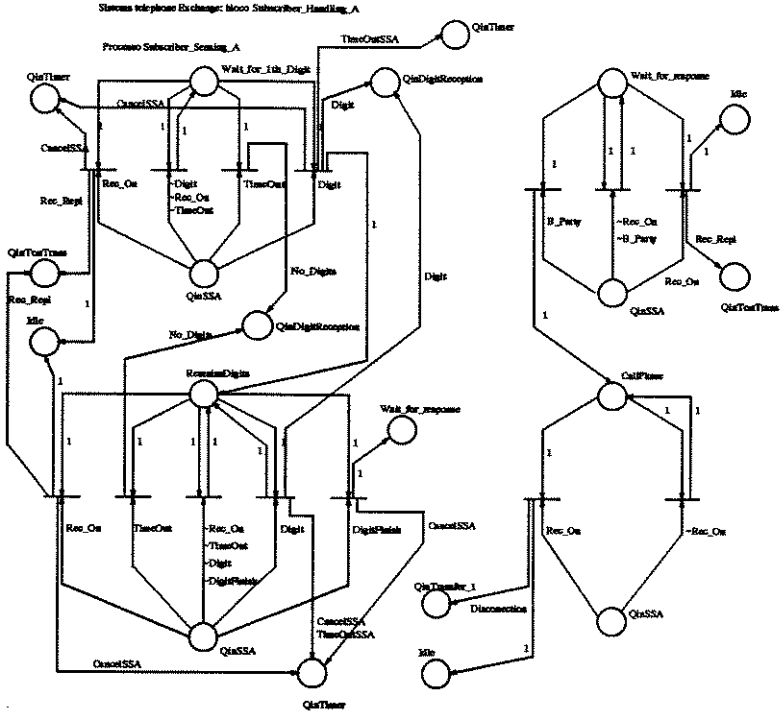

Figura 6.8: Modelagem do processo Subscriber Sensing

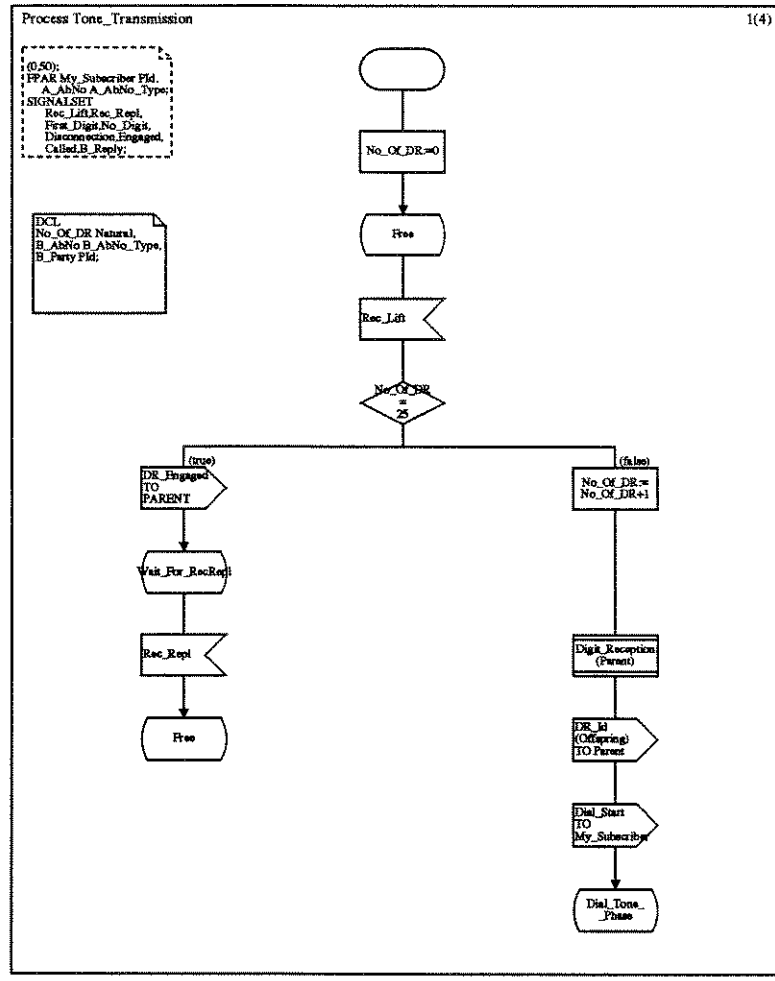

Figura 6.9: Especificação do Processo Tone Transmission

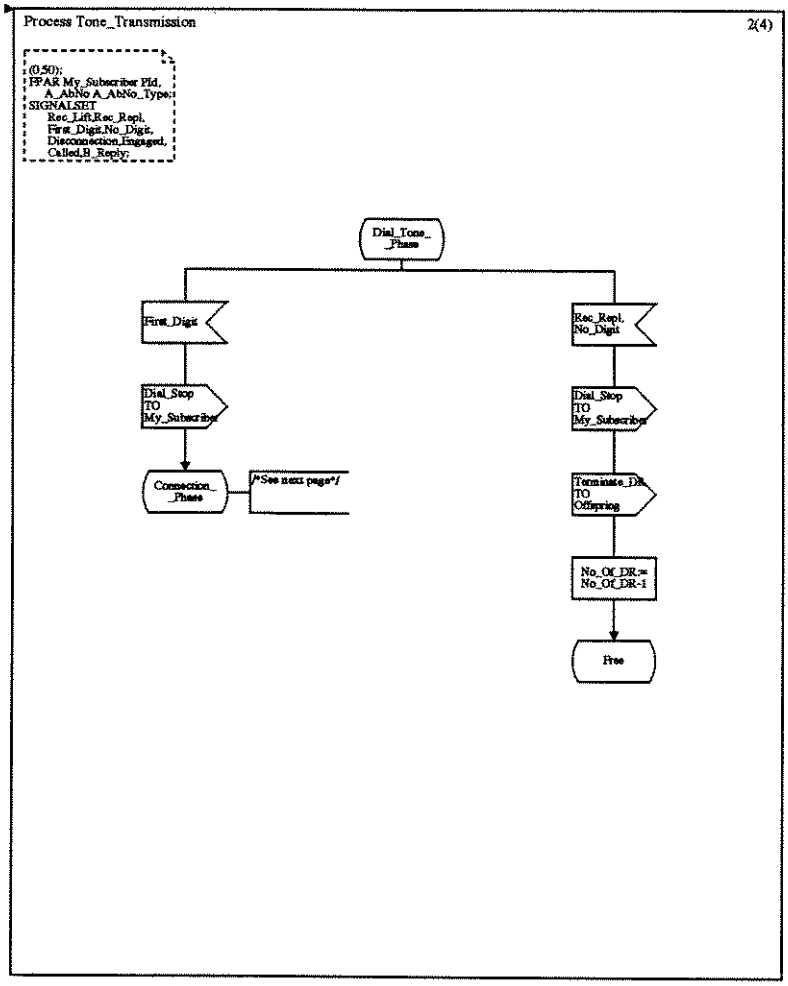

Figura 6.10: Especificação do Processo Tone Transmission

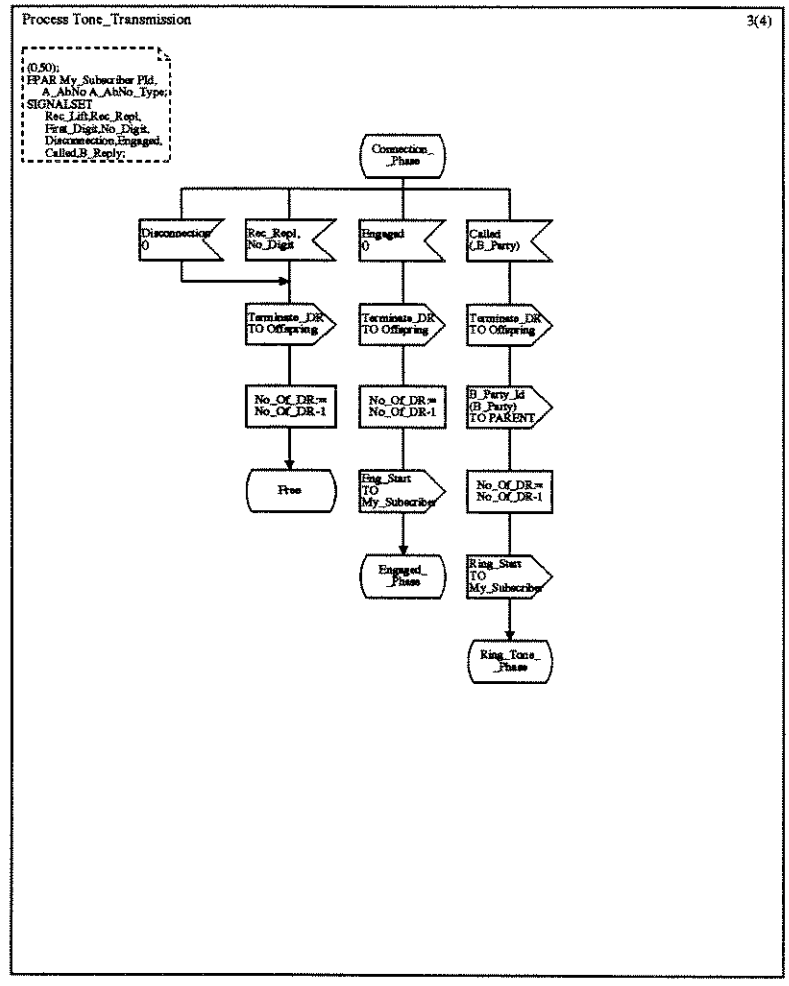

Figura 6.11: Especificação do Processo Tone Transmission

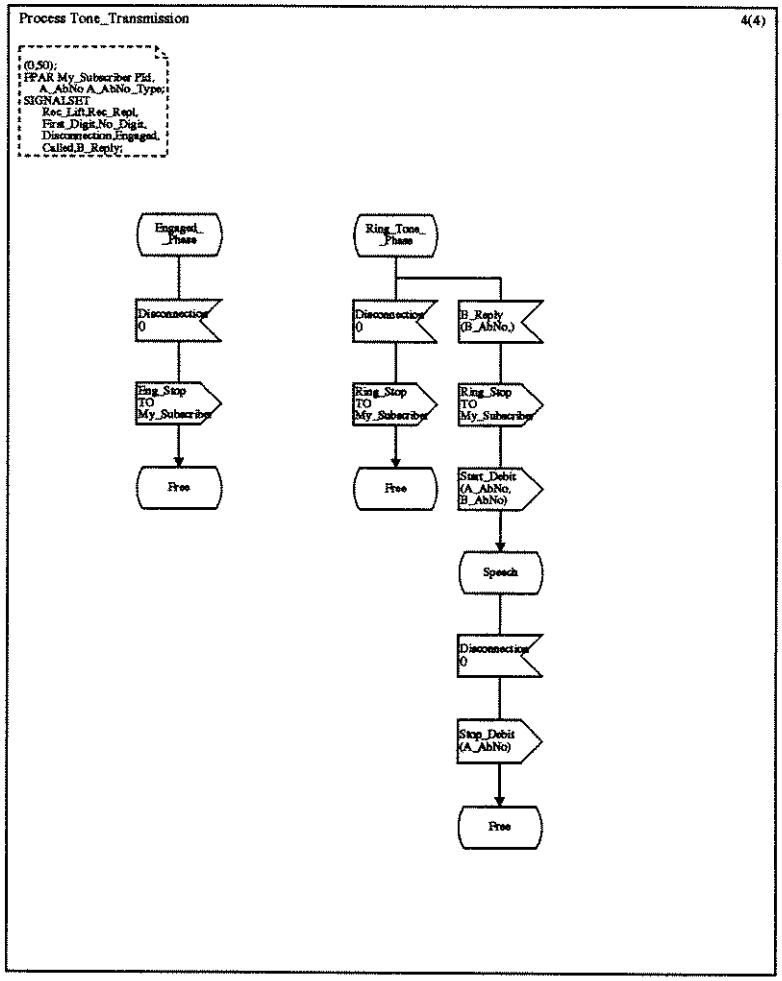

Figura 6.12: Especificação do Processo Tone Transmission

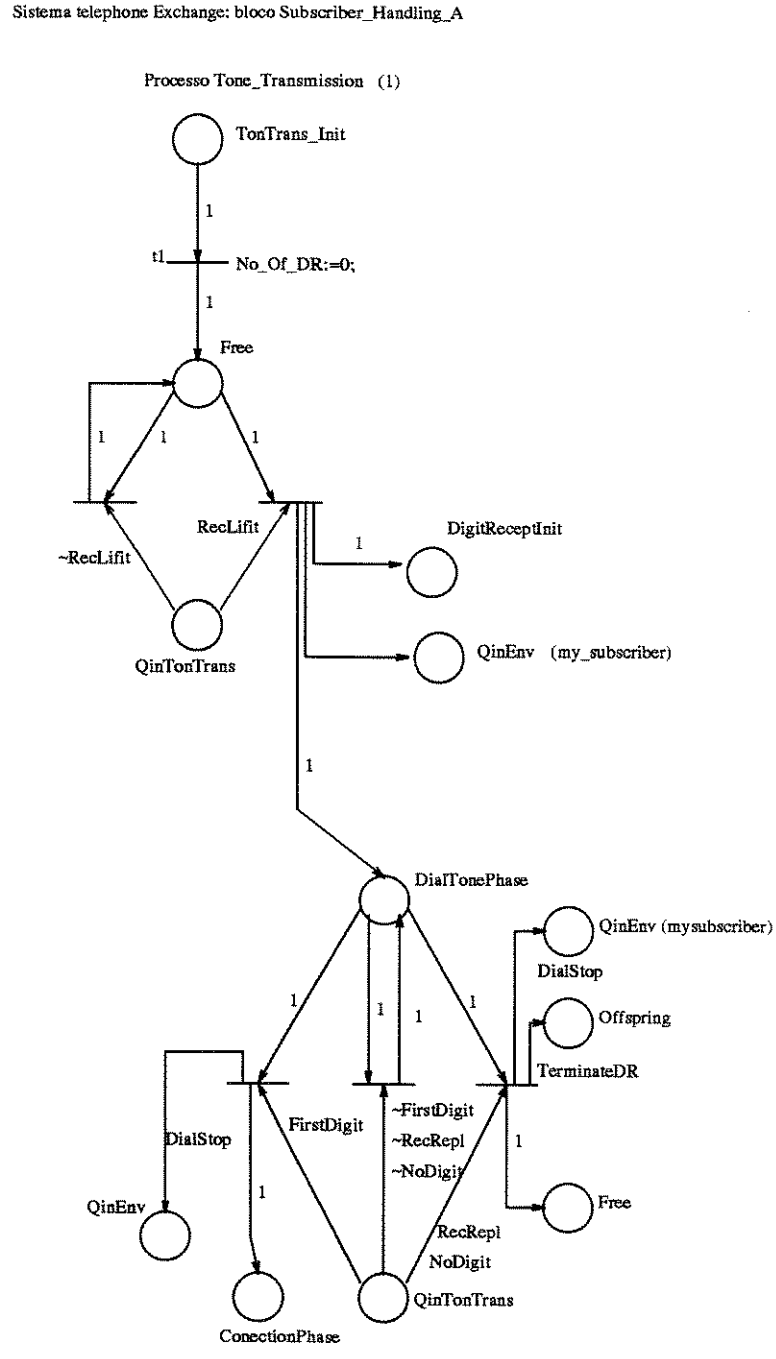

Figura 6.13: Modelagem do Processo Tone Transmission

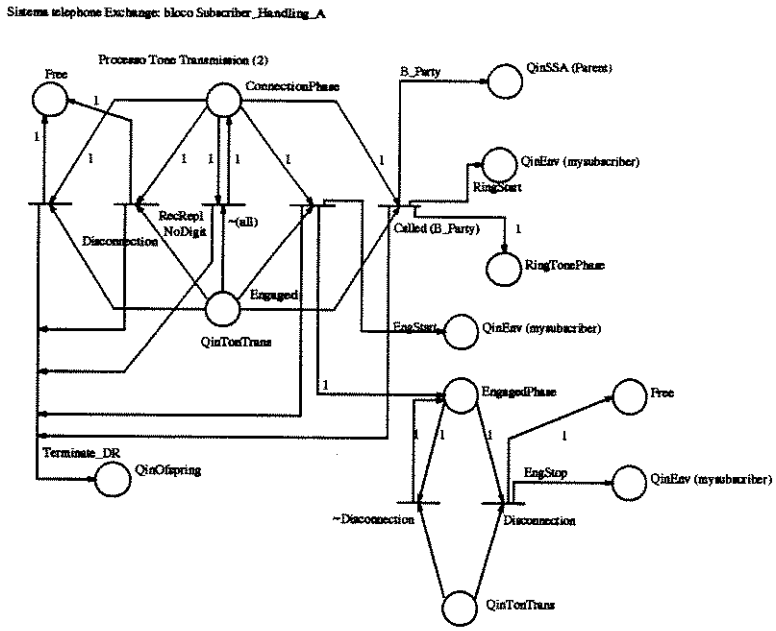

Figura 6.14: Modelagem do Processo Tone Transmission

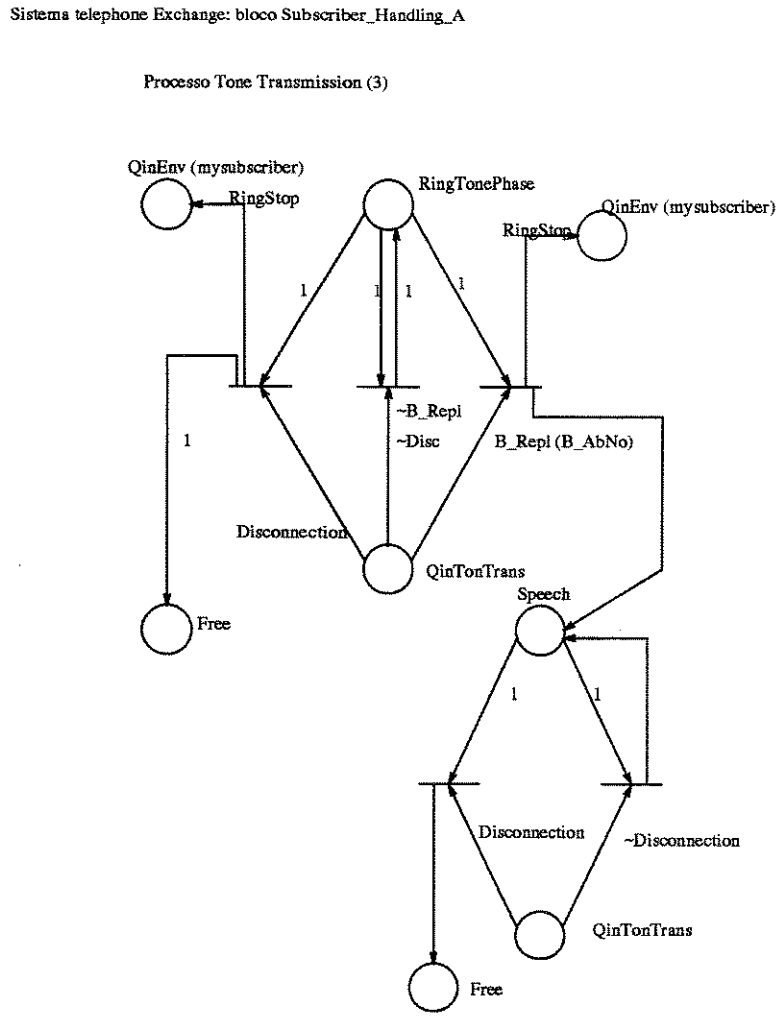

Figura 6.15: Modelagem do Processo Tone Transmission

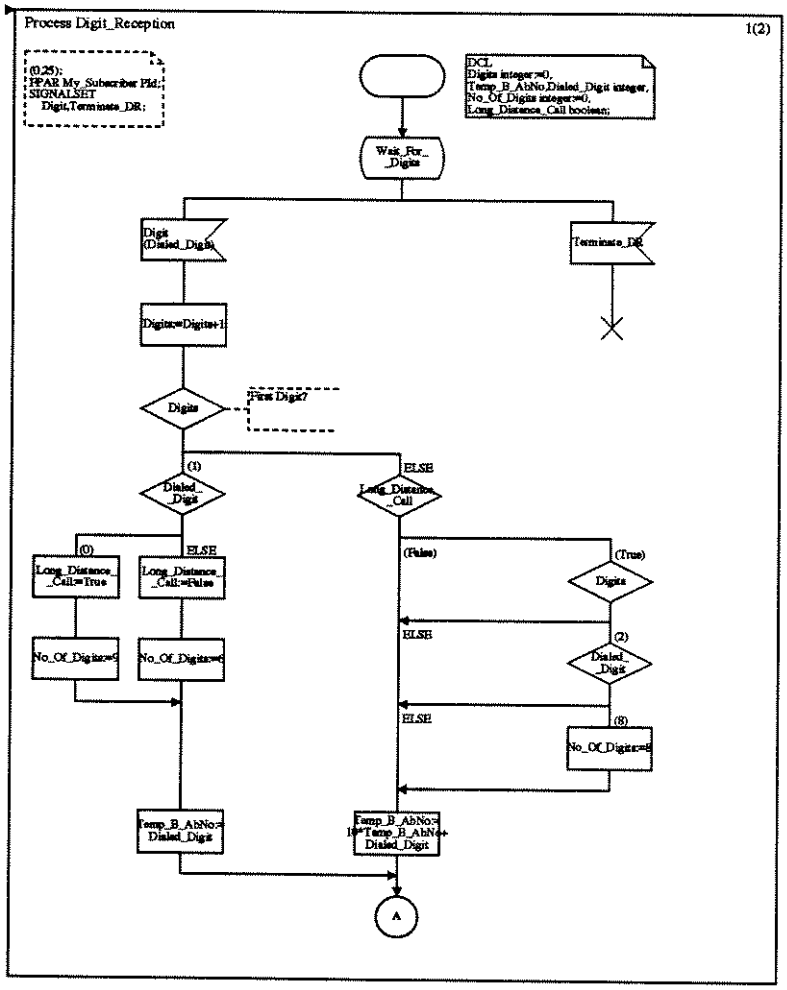

Figura 6.16: Especificação do processo Digit Reception

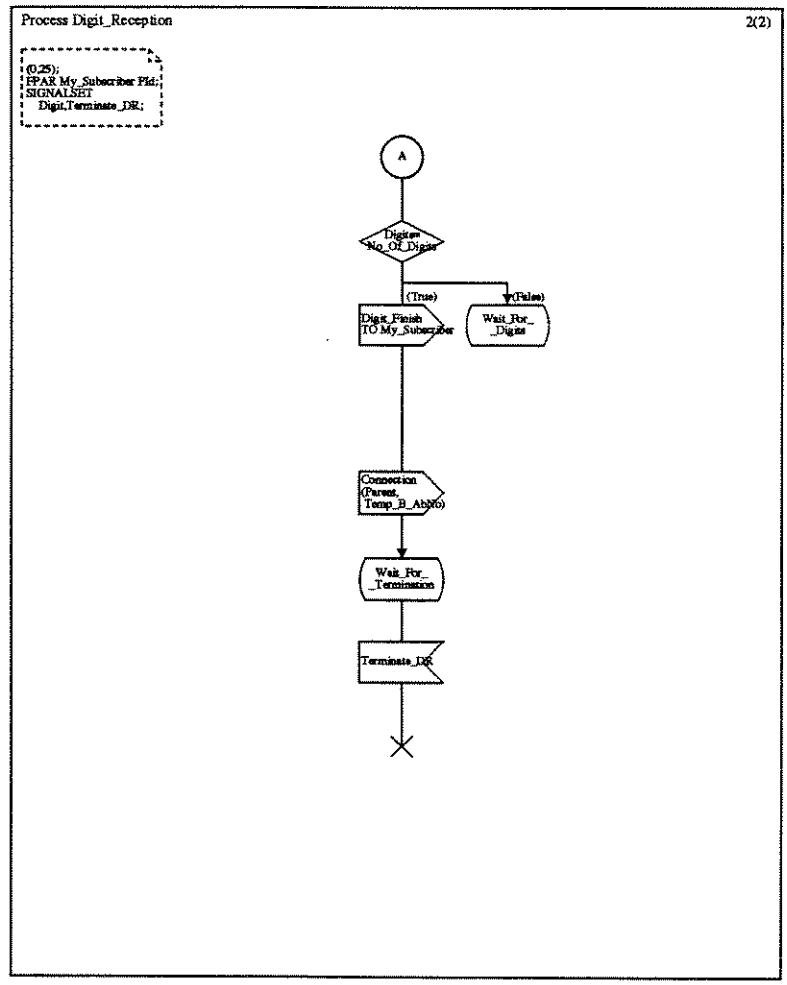

Figura 6.17: Especificação do processo Digit Reception

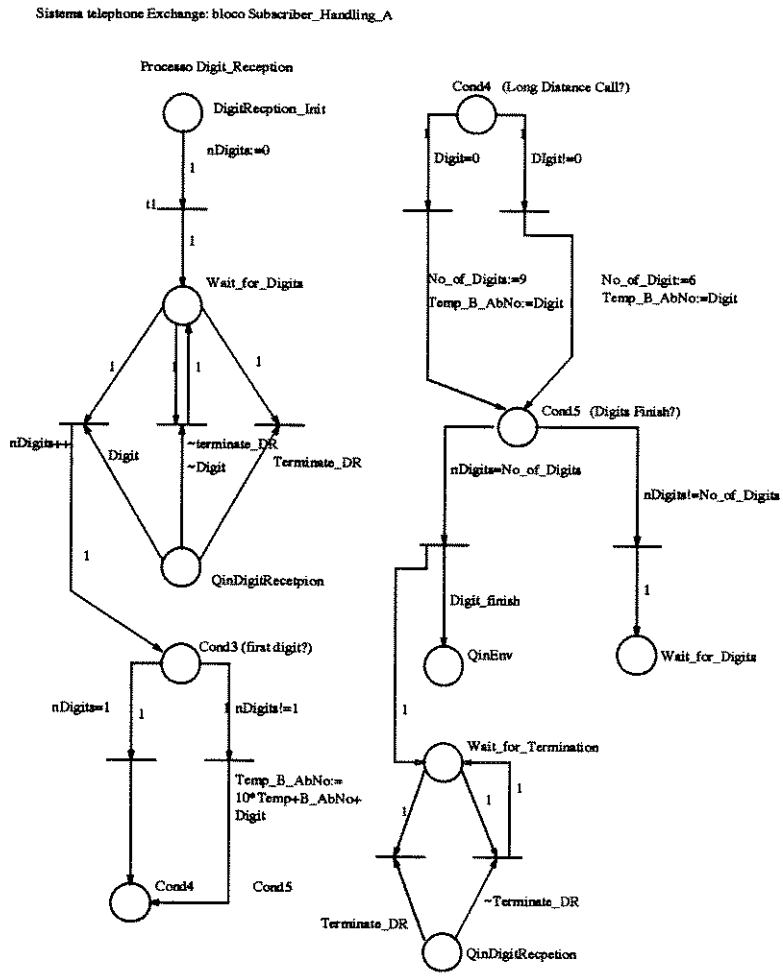

Figura 6.18: Modelagem do processo Digit Reception

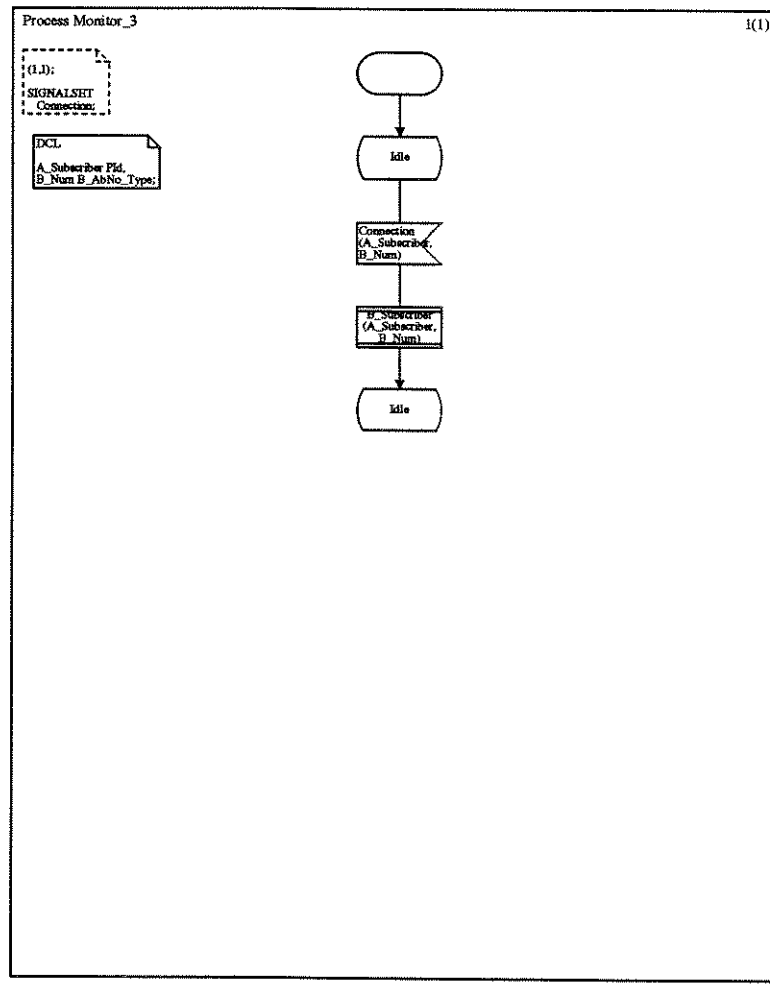

Figura 6.19: Especificação do processo Monitor 3

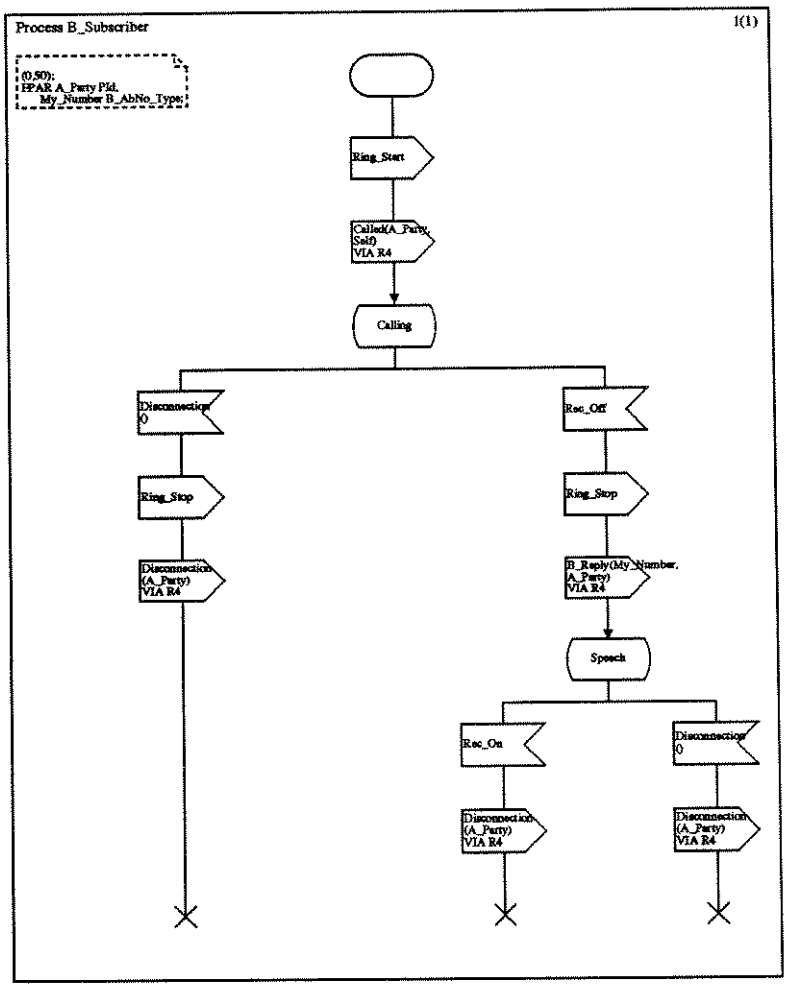

Figura 6.20: Especificação do processo B Subscriber

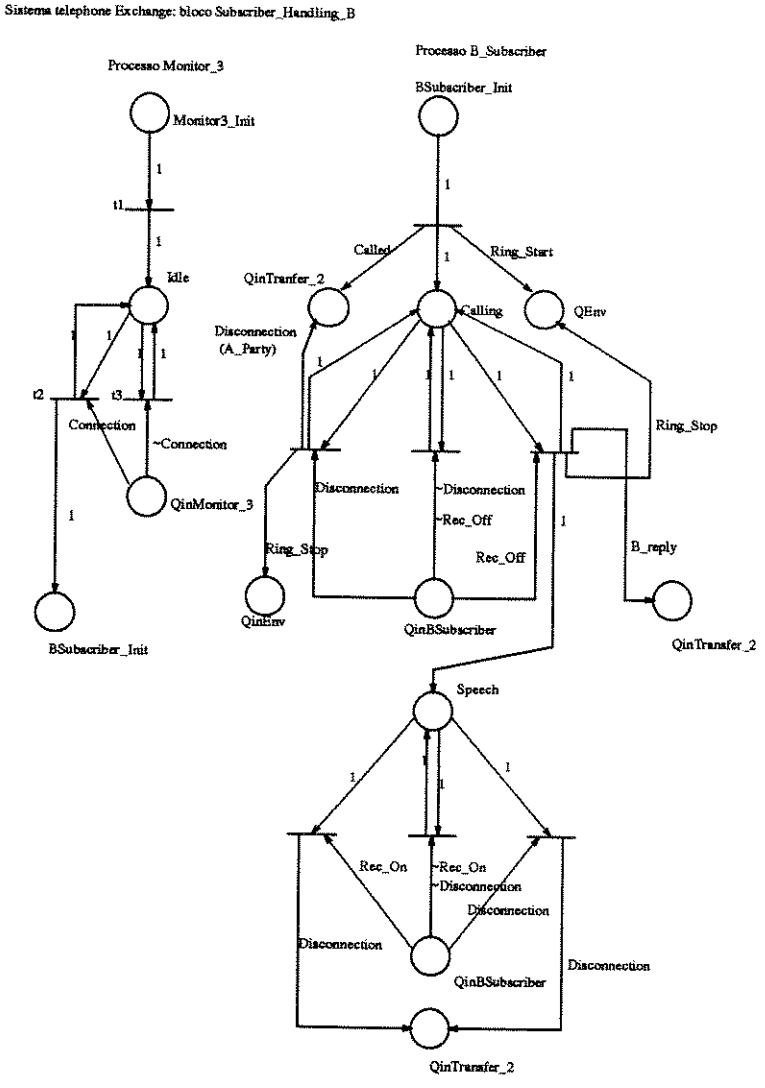

Figura 6.21: Modelagem dos processos Monitor3 e SubscriberB

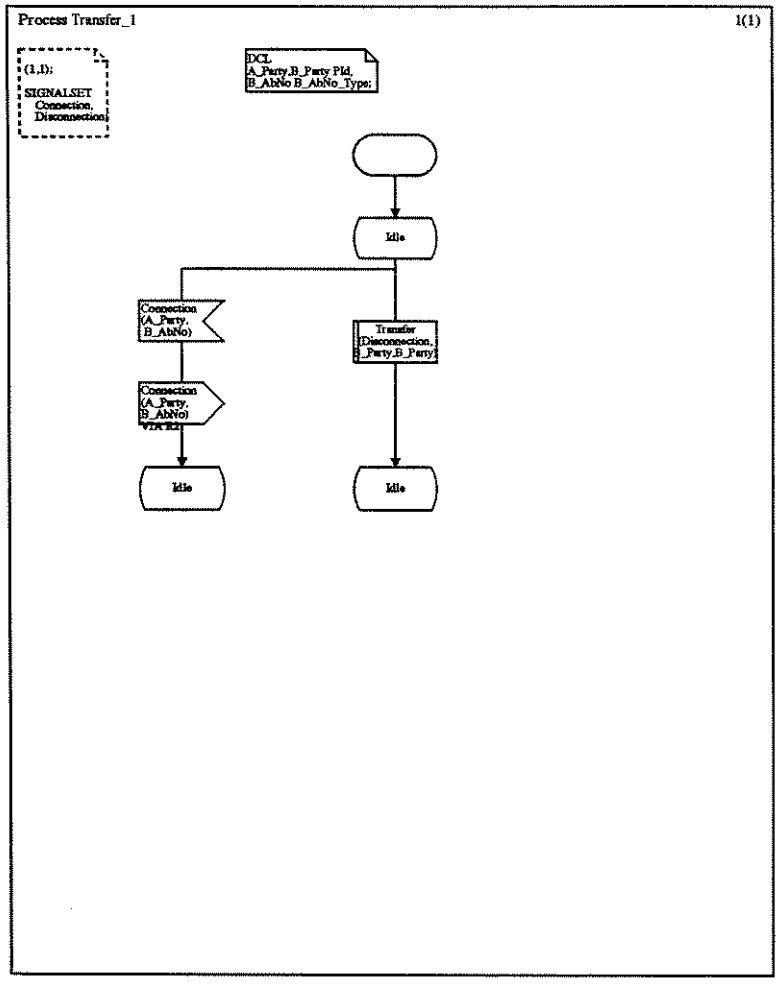

Figura 6.22: Especificação do processo Transfer 1

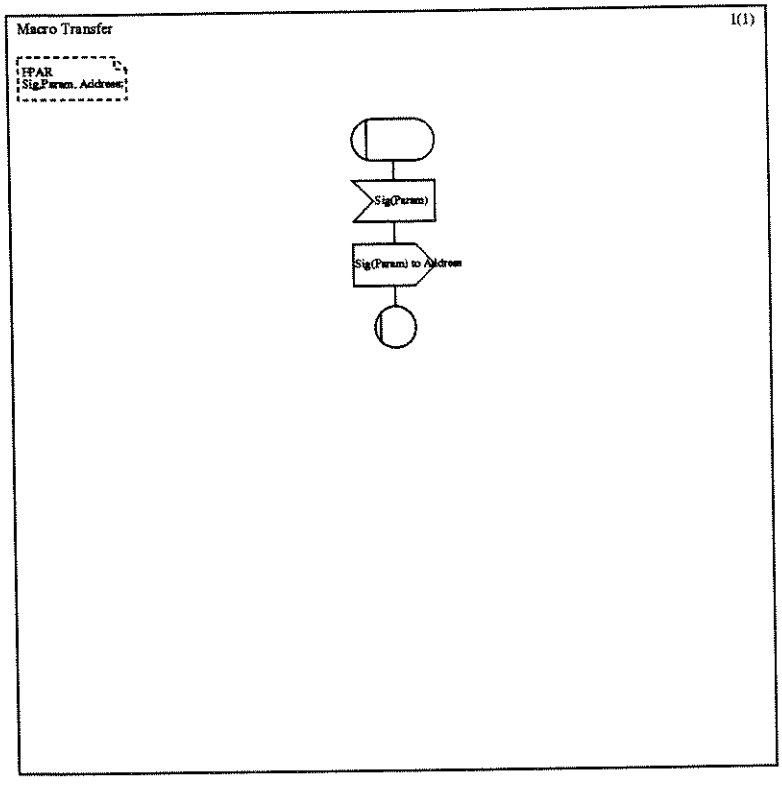

Figura 6.23: Especificação da macro Transfer

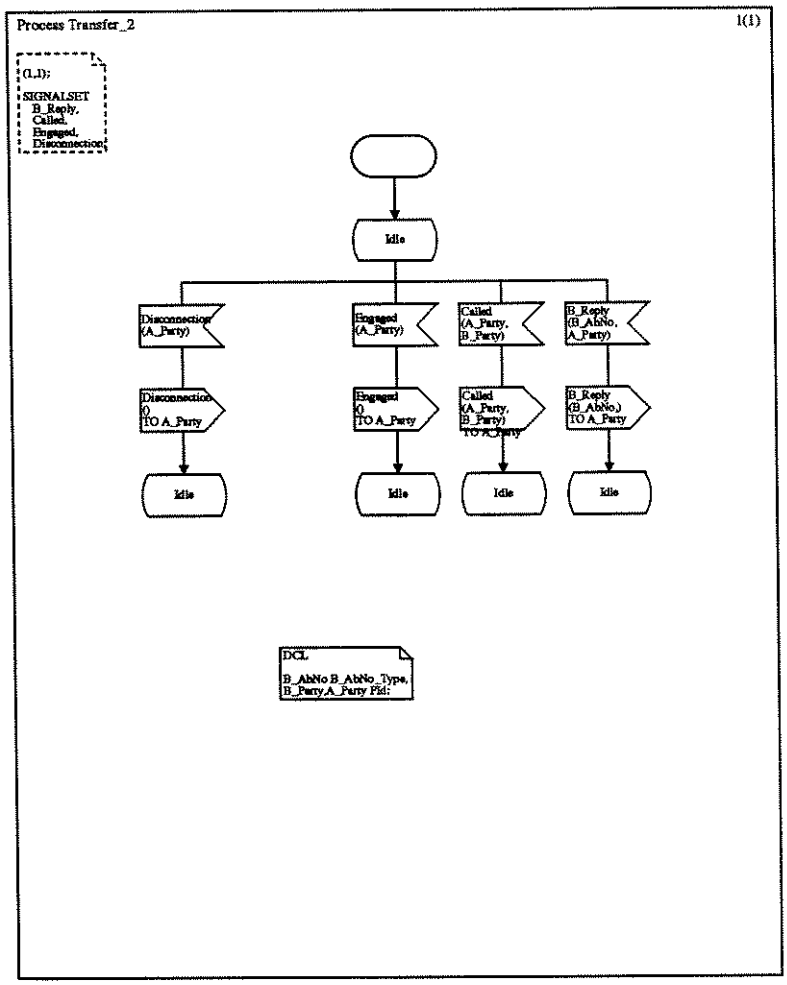

Figura 6.24: Especificação do processo Transfer 2

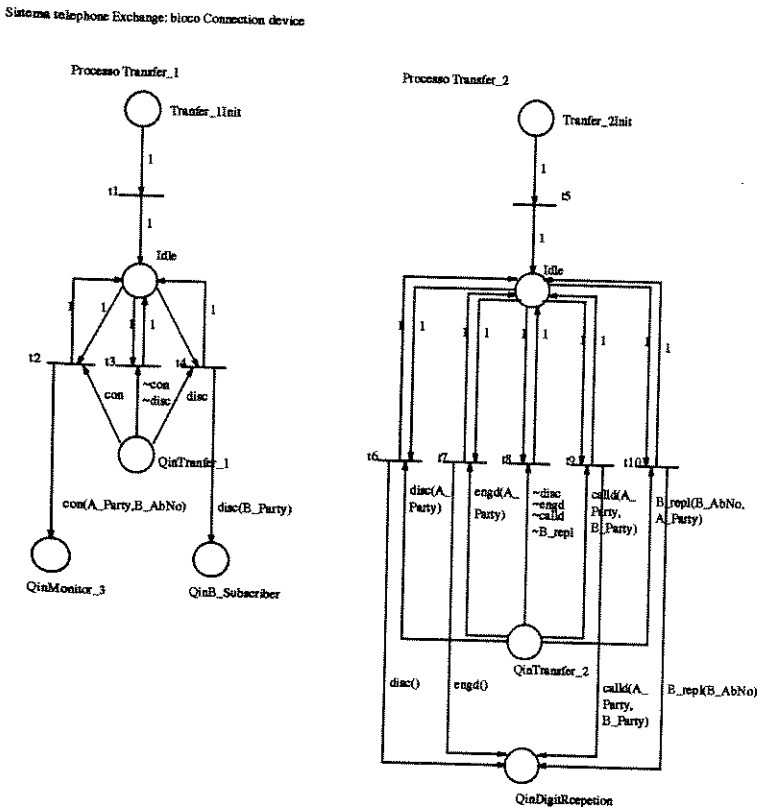

Figura 6.25: Modelagem dos processos Tranfer 1 e Transfer 2

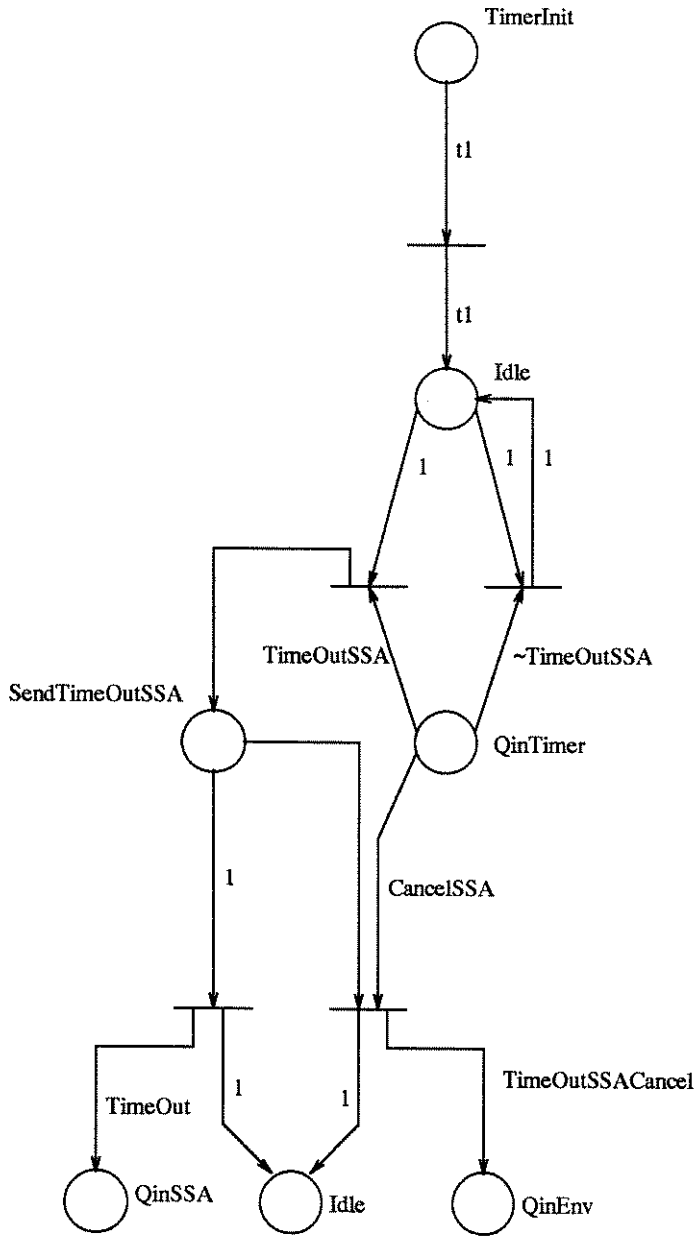

Figura 6.26: Modelagem do Temporizador para TimeOut do Processo Subscriber Sensing

Bibliografia

[1] Belina F., Hogref D. ,Sarma A.. *SDL with Applications from Protocol Specification*, Prentice Hall International (UK) Ltda, 1991. Prentice Hall International (UK) Ltda, 1991.

[2] Borelli W. C. et alli.;*Validation of SDL Communication Protocols by means of Petri Nets.* Telecon'85. Rio de Janeiro, dezembro de 1985.

[3] Borelli, W. C. et alli. *Conversor Automático de protocolos de comunicação em SDL para rede de Petri.* V Simpósio Brasileiro de Telecomunicações. Campinas, setembro de 1987.

[4] Commoner F. *Deadlock in Petri Nets.* Wakefield, applied Data Research, Inc., Report CA-7206-2311, 1972.

[5] Dahler J. at al. *A graphical Tool for the design and prototyping of Distributed Systems.*

[6] Damasceno, B. C.. *Validação de Protocolos de Complexos de Comunicação através de Redes de Petri Numéricas*; Tese de Mestrado (Orientador: W. C. Borelli). FEE/UNICAMP, julho de 1989.

[7] Diaz, M.. *Modeling and Analysis of Communication and Cooperation Protocols Using Petri Nets Based Models*; Computer Networks; n. 6. 1982.

[8] Ek A., Ellsberger J.. *A Dynamic Analysis Tool for SDL.* SDL'91: Evolving Methods, 1991.

[9] Hogrefe, Dieter. *Simulation of a Large SDL system: Some Experiences and a Proposed Method.* Universidade de Hamburgo, 1987.

[10] Holt A. W., Saint H., Shapiro R., Warshall. *Final Report of the information System Theory Project.* Tech. Rep. RADCTR-65-377, vol. 1, 1966.

[11] Janicki, Rysard. *Optimal Simulations, Nets and Reachability Graphs.* LNCS Advances on Petri Nets.

[12] Janicki Rysard e Koutny M. *Towards a Theory of Simulation for verification of Concurrent Systems.* LNCS 366, Springer 1989, 73-88.

[13] Janicki, Rysard. *Optimal Simulations for verification of Concurrent Systems.* Tecnical Reports n 89 - 05, McMaster University, Hamilton, Ontario, 1989.

[14] Kalnins A. *Global State Based Automatic Test Generation for SDL.* SDL'91 Evolving Methods, 1991.

[15] Karlsson J., Ek A. *SSI - An SDL SImulation Tool.* SDL'89: Evolving Methods, 1989.

[16] Kettunen, Esa; Lindqvist, Markus. *Towards Practicality of Predicate/Transition Petri Net Reachability Analisys of SDL.* SDL Estate of the Art and Future Trends, 1987.

[17] Kim HWan C. et alli. *The Automated Verification of SDL Specification Using Numerical Petri nets.* Elsevier Science Publishers B. V. (North Holland), 1991.

[18] Lin Y., Wuu G.. *A constrained Approach for Temporal Intervals in the Analysis of Timed Transitions.* IFIP'91, 1991.

[19] Luo G., Das A. and Bochmann G. V.. *Test Selection Based in SDL Specifications with Save.* SDL'91: Evolving Methods, 1991.

[20] Magill E. H., Smith D.G.. *The automated Checking of Specifications Writen in SDL.* SDL'89: Evolving Methods, 1989.

[21] Marton, M.. *Validação de Protocolos de Comunicação Através de Redes de Petri Temporizadas*; Tese de Mestrado (Orientador: W.C.Borelli). FEE/UNICAMP, fevereiro de 1989.

[22] Nguyen H. T., Jackson L. N., Parker K. R.. *Reachability Graph Generator for SDL.* SDL'89: Evolving Methods, 1989.

[23] Peterson, J.L.. *Petri Net Theory and the Modelling of Systems.*

[24] Saraco, Roberto at al. *Telecomunication Systems Engineering using SDL*, Editora North-Holland 1989 .

[25] Sarma A.. *Modelling Broadband ISDN Protocols with SDL.* SDL'91: Evolving Methods, 1991.

[26] *SDT 2.3 User's Guide.* TeleLOGIC Malmo AB. Suécia, 1993.

[27] *SDT 2.3 Reference Manual vol. 1 e 2.* TeleLOGIC Malmo AB. Suécia, 1993.

[28] Sixtensson A., Petterson S., Johansson A.. *Executable specifications in SDL for validating Protocol Equipment.* SDL'91: Evolving Methods, 1991.

[29] Souissi Y.. *A Modular Approach for the Validation of Communication Protocols using FIFO Nets.* IFIP'91, 1991.

[30] Symons, F. J. W. *The Verification of Communication Protocols Using Numerical Petri Nets.* A. T. R. Vol. 14 num. 1, 1980.

[31] Tadao L. *Petri Nets: Properties, Analysis and Aplications.* Procedings of the IEEE, vol. 77, num. 4, abril, 1989.

[32] Tamura, R. T.. *Um método para a validação de protocolos de comunicação especificados em SDL utilizando Rede de Petri.* Tese de mestrado (Orientador: S. Motoyama). FEE, Unicamp, Brasil, 1988.

[33] Wilts, H. J. e Tilanus, P. A. J.. *Test Design on SDL Simulation*

[34] Wohlin C.. *Performance Analysis of SDL Systems from SDL Descriptions.* SDL'91: Evolving Methods, 1991.