UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

LEMMA 2000 – Uma linguagem para análise e representação de protocolos para diagnósticos em Medicina

por

Guilherme Tomaschewski Netto

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de Mestre
em Ciência da Computação

Prof. Dr. José Palazzo Moreira de Oliveira
Orientador

Porto Alegre, dezembro de 2000.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

NETTO, Guilherme Tomaschewski

LEMMA 2000 – Uma linguagem para análise e representação de protocolos para diagnóstico em Medicina / por Guilherme Tomaschewski Netto. – Porto Alegre : PPGC da UFRGS, 2000.

91 f. il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2000. Orientador: Oliveira, José Palazzo Moreira de.

1. Redes de Petri. 2. Otimização e Análise de Processos médicos. I. Oliveira, José Palazzo Moreira de. II. Título.

Aos meus pais e esposa pelo permanente apoio e incentivo.

Agradecimentos

Agradeço sinceramente a oportunidade e a confiança do Prof. Dr. José Palazzo Moreira de Oliveira.

Aos professores e alunos do Departamento de Eletrônica e Informação do Politécnico de Milão pelo apoio e incentivo no desenvolvimento do trabalho em especial à, Mauro Pezzè e Luciano Baresi pela orientação durante o período de trabalho em Milão e Balaji Mohanaradhakrishnan pelo apoio e auxílio.

Ao Prof. Medhi Jazayeri pela bolsa que possibilitou trabalhar por dez meses no Politécnico de Milão.

À Universidade Católica de Pelotas por financiar este curso de pós-graduação.

À Universidade Federal do Rio Grande do Sul pela oportunidade concedida.

“E para se chegar, onde quer que seja,
aprendi que não é preciso dominar a força,
mas a razão. É preciso, antes de mais nada, querer.”

Amir Klink

“Se fizéssemos aquilo que somos capazes,
literalmente surpreenderíamos a nós mesmos.”

Tomas A. Edson

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A qualidade dos serviços de saúde tornou-se um tema relevante e cada vez mais esforços são dedicados para definir metodologias e ferramentas para medir e assegurar a qualidade. São exigidos novos métodos para aperfeiçoar os processos de saúde, garantindo assim uma alto padrão de qualidade utilizando os recursos disponíveis. A otimização da utilização destes recursos de forma a preservar a qualidade do atendimento bem como baixar os custos requer modelos rigorosos dos processos médicos.

Neste contexto apresentamos LEMMA 2000 (Language for an Easy Medical Model Analysis) uma notação destinada a modelar processos médicos, desenvolvida em cooperação com o Politécnico de Milão..

Esta notação disponibiliza aos médicos elementos simples e intuitivos para representar os modelos de diagnóstico. Simultaneamente um modelo de Redes de Petri Temporizadas é gerado automaticamente. Deste modo os modelos de LEMMA podem ser validados e analisados através de simulações e testes.

O objetivo desta dupla modelagem é permitir aos médicos ganhar todos os benefícios de uma notação formal sem a necessidade de conhecer esta notação abstrata. Os usuários administram os elementos mais simples da notação, enquanto redes de Petri asseguram formalidade e capacidades de validação. Desta maneira LEMMA integra notações formais e intuitivas superando os problemas de ambos enfoques.

A definição da notação LEMMA 2000 foi apoiada pela implementação de um ambiente para projetar os modelos. Este ambiente além de permitir a administração dos elementos da notação menos formal permite a geração e analise dos modelos mais formais.

Palavras-chave: Redes de Petri, Otimização e Análise de processos médicos.

TITLE: "LEMMA 2000 – A LANGUAGE FOR ANALYSING AND REPRESENTING PROTOCOLS FOR DIAGNOSES IN MEDICINE"

Abstract

The quality of health services has become an important subject. Efforts have been dedicated more and more to define methodologies and tools to measure and to guarantee quality. People demand new methods in order to improve health processes. This way a high standard of quality is guaranteed by using available resources. The optimization of using these resources in such a way as to keep the quality of attendance as well as to reduce expenses, needs strict patterns of medical processes.

In this context we present LEMMA 2000 a notation for modeling medical processes developed in co-operation with the Politecnico di Milano.

This notation provides doctors with common and intuitive elements to represent the patterns of diagnoses. At the same time a model of Timed Petri nets is created automatically. Because of this the patterns of LEMMA can be validated and analysed through simulations and tests.

The aim of this double modelling is to allow doctors to gain all the benefits of a formal notation without the need to know this abstract notation. The users manage the most simple elements of the notation while Petri nets guarantee formality and ways of validation. This way LEMMA integrates formal and intuitive notations overcoming problems of both focuses.

The definition of LEMMA 2000 notation was supported by the implementation of an environment for projecting the patterns. This environment not only allows the management of the elements of the less formal notation but it also allows the creation and analysis of most formal patterns.

Keywords: Optimization and analysis of medical processes, Petri nets.

1 Introdução

A qualidade dos sistemas e processos tornou-se um problema proeminente e cada vez mais são dedicados esforços a fim de definir metodologias e ferramentas para medir e assegurar a qualidade tanto nos processos quanto em sua representação[Don88] [Don 90]. Pois, para que falhas ou erros nos processos dentro de determinadas instituições sejam minimizados é necessário que sejam conhecidos estes processos e para conhecê-los devemos criar modelos que representem com veracidade a realidade.

Através da representação da realidade em modelos formais poderemos obter um conjunto de conhecimento satisfatório para a validação destes. Nas palavras de lorde Kelvin, "... quando não se pode medir, quando não se pode expressar em números, o conhecimento que se tem é escasso e insatisfatório."

Sistemas médicos são extremamente complexos, caros, e críticos. Reduzir os custos é importante para oferecer serviços médicos à população inteira. Porém, redução de custo não deve afetar a qualidade do serviço. A modelagem de processos de diagnóstico e terapêuticos são extremamente importantes para identificar gargalos, treinar pessoal novo, e monitorar a qualidade dos processos médicos. A complexidade de diagnóstico e processos terapêuticos requer idiomas de modelagem para os médicos que devem poder entender e descrever tal processo. Por outro lado, este idioma deve ser preciso e não-ambíguo, evitar enganos e ser extremamente consistente. Eles também têm de apoiar técnicas de análise para conferir a justeza da descrição e monitorar os processos atuais. Finalmente eles têm que permitir acesso fácil à muita informação requerida durante os processos diagnóstico. Podem ser satisfeitas para tais exigências com idiomas diferentes: um idioma de "*front-end*" que satisfaça as exigências dos médicos, e um idioma de "*background* " que satisfaça a modelagem e exigências de análise, e pode ser conectado adequadamente com bancos de dados existentes.

Este trabalho cria técnicas e tecnologias para apoiar os processos médicos. Começará a partir de idioma dual que foi o resultado de uma colaboração entre doutores médicos da *Università Roma* e engenheiros do Departamento de Eletrônica e Informação de *Politecnico di Milano*. Tal enfoque consiste de um idioma, chamado LEMMA, para modelar processos diagnóstico; e um idioma formal, rede de Petri temporizadas, que apóia as capacidades de análise exigidas. O enfoque está baseado em um processo de customisação construído em teoria de gramática de grafos que permite que o idioma de modelagem possa ser otimizado de acordo com as exigências dos doutores médicos e o processo diagnóstico ser modelado.

As etapas de projeto e desenvolvimento deste trabalho foram feitas no Politécnico de Milão, Departamento de Eletrônica e Informação no período de maio de 1999 à fevereiro de 2000 sob orientação dos Prof. Mauro Pezzè e Prof. Luciano Baresi.

Estes dez meses de trabalho foram financiados pelo Projeto Alfa que colaborou com uma bolsa e passagens.

A seguir é mostrada a organização deste texto.

No capítulo 2 é feita uma breve descrição de algumas características dos protocolos médicos e como eles devem ser representados.

No capítulo 3 são descritas as redes de Petri, seus tipos e características, bem como sua aplicação na representação dos protocolos médicos.

No capítulo 4 é mostrada a notação LEMMA e a segunda versão desta chamada de LEMMA 2000. São descritos os elementos e as análises em que esta notação auxilia a execução.

No capítulo 5 é mostrada a correspondência entre os elementos da notação LEMMA 2000 e as redes de Petri equivalentes.

No capítulo 6 desenvolve-se a descrição do editor gráfico encarregado por gerenciar os modelos representados mediante a notação LEMMA 2000, bem como a criação do arquivo necessário para a correspondência com as redes de Petri.

No capítulo 7 é mostrada uma aplicação do editor gráfico, utilizando como caso de estudo alguns protocolos de diagnóstico em neurologia.

Por fim tem-se uma conclusão e uma relação de trabalhos futuros.

2 Protocolos Médicos

Neste capítulo são apresentados os protocolos médicos para descrever os processos clínicos, listando as habituais aplicações e as propriedades a serem satisfeitas. Em um segundo momento, são demonstradas as principais técnicas de especificação de tais protocolos, considerando suas vantagens e desvantagens.

2.1 Os protocolos médicos

Segundo [Bar94], o Instituto de Medicina norte-americano assim define os protocolos médicos:

No âmbito médico, não existe uma rigorosa definição dos protocolos, mas podem ser tratados como uma documentação do processo de decisão do médico e intercâmbio de informação entre este e o paciente. Conforme cada área médica existe uma maior especificidade na representação destes processos de decisão, havendo sinais bem determinados como exames de sangue. Em outras áreas, como psiquiatria, o processo de tomada de decisão é muito sutil, pois baseia-se mais na experiência do especialista, sua capacidade de interpretar os sinais e traços de um caso clínico específico.

Estas definições dos protocolos médicos são necessárias por vários motivos:

- Melhoramento qualitativo da cura: é o principal objetivo da aplicação destas representações, a possibilidade de dispor de um padrão de comportamento no qual todos os médicos farão referência. Assegura uma uniformidade no tratamento do paciente na cura de uma síndrome universalmente reconhecida.
- Assistência ao médico e paciente: os protocolos oferecem a possibilidade de enquadrar a situação clínica de um paciente em um contexto no qual foram especificados os procedimentos clínicos e servem de suporte ao médico e paciente na decisão entre as diversas alternativas que se apresentam.
- Ensino: oferecem aos médicos uma série de situações ou casos com um conjunto de instruções operativas, não só para o auxílio de um caso mais raro, mas também para os estudos e o aprendizado de uma área nova.
- Redução de despesas.
- Melhor alocação de recursos: utilizando protocolos que representam diretamente recursos usados no processo de decisão ou cura (como leitos, laboratórios, médicos), é possível uma melhor otimização destes,

tornando possível agendamentos mais precisos, diminuindo o tempo médio de permanência do paciente nos ambientes de saúde.

É desejado que os protocolos satisfaçam algumas propriedades, sendo um elenco destas características mostrado na Tabela 2.1[Ins92]. Sua descrição, informal e abstrata, coloca em evidência a falta de rigor nos seguintes ítens:

- Validade, aplicabilidade, flexibilidade clínica são propriedades gerais dos protocolos, requerem que a decisão do médico seja apropriada para o caso em exame, e aplicável em situações reais.

TABELA 2.1 - Propriedades dos protocolos médicos

Protocolos Médicos	
Propriedades	**Descrição**
Validade	Os protocolos são válidos se conduzem a resultados satisfatórios, tanto em termos de melhoramento da saúde dos pacientes bem como em relação à otimização dos recursos e a redução dos custos.
Reprodutibilidade	Um protocolo é considerado reproduzível se especialistas diversos, perante a mesma situação clínica, fornecem a mesma indicação. Isto deve suceder tanto no desenvolvimento do protocolo quanto na sua aplicação.
Aplicabilidade Clínica	Os protocolos devem ser aplicados em um número significativo de pacientes.
Flexibilidade Clínica	Os protocolos devem permitir adaptação segundo as características das estruturas clínicas locais e dos pacientes.
Clareza	Deve-se utilizar uma linguagem não-ambígua, com termos claros e símbolos ou notações de fácil compreensão.
Multidisciplinaridade	No desenvolvimento dos protocolos devem participar todos os representantes dos setores que vêm a influenciar o processo em exame.
Atualizabilidade	O desenvolvimento deve permitir a possibilidade de atualizações e modificações.
Documentação	O processo de desenvolvimento de protocolos devem ser meticulosamente documentados.

- A propriedade de clareza é atribuída ao tipo de especificação do protocolo, ou seja, ao método de representação escolhido.

- As propriedades de reprodutibilidade e multidisciplinaridade são próprias do processo de desenvolvimento, ou seja, da atividade ou especialidade que levou à definição de um protocolo.
- As propriedades restantes de agendamento e documentação aplicam-se unicamente ao processo de desenvolvimento e representação de um protocolo.

É definido como critério de revisão: "declaração desenvolvida sistematicamente, que pode ser utilizada para assegurar a adequada utilização de intervenção médica"[Ins92], observando claramente a necessidade de compilar a análise, verificação e simulação automática nos modelos de representação.

2.2 Especificação dos Protocolos Médicos

Entende-se a especificação de um protocolo médico pela realização de um modelo mais genérico, de uma representação de um processo mediante uma particular notação ou linguagem. Devem também ser observadas algumas propriedades referentes a estes, como foi citado anteriormente.

Os métodos de especificação de um protocolo variam notoriamente a respeito do grau de formalidade da notação escolhida: se é utilizada linguagem natural, tabelas, fluxogramas, descrições algorítmicas, para que se aproximem a modelos mais matemáticos ou formais como gramática de grafo ou técnicas orientadas a objeto. Uma breve descrição das características de cada técnica de especificação é apresentada a seguir.

Linguagem natural

A linguagem natural é largamente utilizada para representar os protocolos médicos para servir de ponto de partida para outros métodos de representação. Sua grande vantagem é ser facilmente compreendida por médicos e outros profissionais da saúde, eliminado as partes mais técnicas, podendo ser utilizada também com pacientes e seus familiares.

A maior desvantagem é o alto grau de informalidade, que impede completamente qualquer verificação e análise.

Descrição Algorítmica

A seqüência lógica das operações a serem executadas em um determinado contexto vem ilustrada mediante algoritmos. A definição dos algoritmos é obtida por

meio de refinamentos sucessivos de uma descrição do problema em linguagem natural. Uma especificação desse tipo vem a ser utilizada para efetuar confronto entre protocolos e eventuais conflitos no processo. A desvantagem principal é a pouca legibilidade dos modelos e a necessidade de quem o modela, de conhecer tanto a notação quanto os procedimentos clínicos.

Diagramas e Tabelas

Existem também métodos de especificação dos protocolos em forma gráfica ou tabular: esquemas de blocos, tabelas de decisão, diagramas de influência, etc. Estas representações aumentam a legibilidade dos modelos e são ajuda de muita valia para a assimilação dos conceitos representados, mas sua correta interpretação é condicionada ao conhecimento das regras sintático-semânticos das notação utilizada e não tem a capacidade de acrescentar a contribuição pessoal que cada médico pode oferecer com sua própria experiência.

Gramática de Grafos

É possível especificar um protocolo médico mediante a utilização de produções da gramática de grafos [GP93, Goe92] na qual são utilizados símbolos característicos. A Teoria dos Grafos engloba o estudo das relações entre entidades ou objetos que possuem características relevantes para um dado problema ou situação. Um grafo é formado por nós e arestas que são as ligações entre estes nós. O modelo proposto é seguramente válido devido ao grau de formalidade da notação, mas o resultado final é, em geral, muito complexo, de difícil compreensão também a um profissional especializado nestes formalismos.

Sistemas de Suporte à Decisão

As notações gráficas também são usadas para definir sistemas de suporte à decisão. Isto pode ser realizado através de transformações entre representações informais e linguagens fortemente estruturadas, aderidas a uma descrição dos comportamentos possíveis numa situação clínica considerada.

Como existe uma ampla possibilidade de modelagem e um vasto campo de aplicação, a fase de definição do sistema e de qualquer modificação resulta num trabalho muito custoso. Isto leva a uma estreita colaboração entre os informatas e médicos.

Técnicas Orientadas a Objeto

Um outro enfoque leva à aplicação de técnicas orientadas a objeto para a gestão da informação em um ambiente clínico. É bastante óbvia a necessidade de se escolher uma notação ou uma linguagem que se adapte ao objetivo da modelagem. Existem várias técnicas de modelagem orientadas a objeto, cada uma com seu grau de formalidade. As vantagens e defeitos desta técnica não irão destacá-las em relação as outras. As vantagens de uma notação formal são pagas em termos de conhecimento informático necessário aos usuários.

A experiência mostra que a modelagem de protocolos médicos se dá, em geral, com notações "informais", que resultam em modelos mais simples para a consulta feita pelos usuários médicos. Já os enfoques formais gerados por profissionais de informática têm resultados finais mais complexos de difícil interpretação aos usuários especialistas em outros âmbitos: médicos, clínicos ou cirúrgicos.

Pode-se concluir que tanto modelos formais quanto informais não estabelecem um conjunto de benefícios que possibilite a elaboração de modelos de protocolos os quais satisfaçam as necessidades de representação. Logo, pode-se resolver este problema com a aplicação dos dois enfoques, aproveitando as vantagens das duas notações.

3 Especificação formal com Redes de Petri

3.1 Redes de Petri

Em diversos modelos de representação, os aspectos formais são modelados com Redes de Petri, cujas características e representações veremos a seguir.

Redes de Petri [Mac 96] são uma notação gráfica formal bem apropriada para modelar uma variedade larga de sistemas. Combinam simplicidade (representação gráfica) e uma armação matemática rigorosa para descrever sistemas de atividades assíncronas, paralelas e não-determinísticas.

Uma rede de Petri, Figura 3.1, é um gráfico bipartido dirigido com um estado

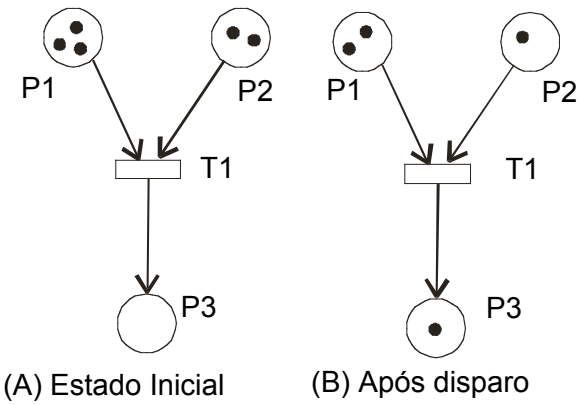


FIGURA 3.1-Rede de Petri Elementar

inicial, onde nodos são divididos em lugares, representados por meio de círculos, e em transições, representadas como retângulos pequenos. Transições definem os eventos que caracterizam os sistemas; lugares são pré e pós-condições associadas com eventos. Arcos conectam cada lugar a transições ou transições a lugares.

A rede da Figura 3.1 inclui transição T1 e três lugares: P1, P2, e P3. Lugares P1, P2, prefixos de T1, definem a pré-condição que permitem o evento T1 acontecer. O lugar P3, o pós-condição de T1, identifica a condição após o disparo de T1.

Aos lugares são associados *tokens* ou marcas, pequenos círculos pretos. Estas marcas darão a condição necessária para que uma transição seja disparada, visto que para que isto aconteça, em cada lugar prefixado deve existir uma marca. O disparo da transição remove uma marca de cada lugar prefixado e produz uma ficha em cada lugar do pós-fixado.

Na Figura 3.1(a), a transição T1 é habilitada e seu disparo remove uma ficha de lugares P1 e P2, produzindo uma ficha em lugar P3 (Figura 1(b)). Desde que o lugar P1 contém três fichas e P2 duas fichas, a transição T1 estaria habilitada e poderia disparar duas vezes.

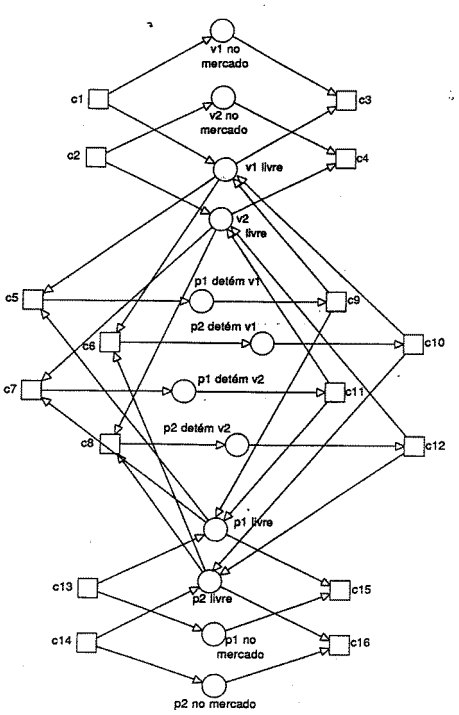


FIGURA 3.2 A. Rede de Petri Elementar[Heu90]

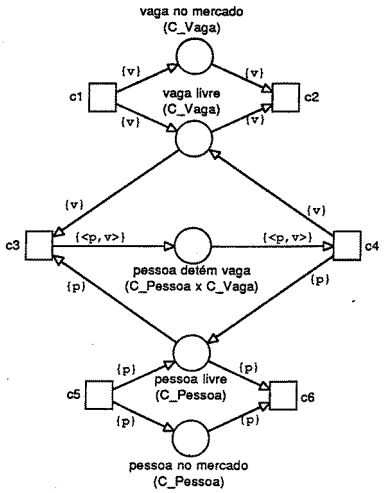


FIGURA 3.2 B. Rede de Petri Compacta[Heu90]

Além de uma definição rigorosa de execução, a estrutura matemática permite que as redes de Petri sejam analisadas rigorosamente. A capacidade de análise permite aos desenhistas conferir automaticamente os modelos contra propriedades específicas.

Existe ainda a extensão, que permite uma melhor abstração desta representação: são as chamadas redes de Petri de alto nível, também conhecidas como redes compactas, redes coloridas, redes de predicado/transição e redes hierárquicas.

A diferença básica entre redes de Petri de alto nível e redes de Petri elementares está na caracterização das marcas. Nas redes de Petri elementares, as marcas são elementos do tipo inteiro positivo. Nas redes de Petri de alto nível as marcas são diferenciadas por parâmetros, que permitem a individualização destas, podendo ser feita a especificação de um objeto por sua cor, ou mesmo, podendo ser representadas as marcas não por um único objeto, mas por um conjunto de objetos [Mac96][Heu90]. Este conjunto de objetos pode representar características bem específicas de cada marca. Nas redes temporizadas, uma dessas características é o tempo em que esta marca permanece em cada transição.

As redes de alto nível foram uma evolução da elementares, visto que sua representação gera modelos muito complexos para aplicações onde processos se repetem.

Na figura 3.2 A, pode-se observar um exemplo de uma rede de Petri elementar que representa o comportamento de duas pessoas e duas vagas no mercado. Pode-se notar que a representação para as vagas e pessoas são idênticas, mas se fazem necessárias para distinguir cada pessoa e cada vaga. Seria mais conveniente registrar todos os processos semelhantes em uma mesma rede, o que é mostrado na figura 3.2 B, onde uma rede compacta contorna este problema, representando processos semelhantes com a mesma rede distinguindo somente as marcas. A estas marcas são acrescentadas características, ou predicados, que farão a distinção entre elas: por exemplo, qual pessoa detém a vaga, e qual vaga é detida.

Assim, é possível obter uma representação mais simples sem perder os aspectos formais, que são o ponto forte e justificam sua utilização para realizar determinadas validações.

3.2 Utilização das Redes de Petri na Especificação dos Protocolos Médicos

3.2.1 As Vantagens de Uma Representação Formal

As principais vantagens de uma representação formal [GFM94] derivam dos fundamentos matemáticos nos quais se embasam.

Uma notação formal é dotada de uma semântica unívoca, é precisa e obriga a definição de todos os detalhes: assim, é evitada a presença de pontos não-especificados. É possível compilar análises de maneira automática e sobre o modelo construído, realizar execuções ou simulações que podem ser utilizadas para a validação desta especificação.

3.2.2 Escolha dos Modelos

Em uma atenta análise da documentação relativa à especificação dos protocolos médicos, é possível derivar princípios gerais para a modelagem com redes de Petri.

Em primeiro lugar é identificado o paciente como ator do processo, e é agregado um conjunto de atividades fundamentais a uma série de ligações ou precedências entre eles. Um paciente pode entrar em um processo a partir de um determinado ponto e alcançar, após cumprir um itinerário determinado, algum ponto de saída determinado, dito saída do processo.

O estado inicial do paciente é descrito por uma cartela clínica onde se reportam os dados clínicos associados a este. Esta cartela é identificável mediante um código unívoco ligado ao ingresso no processo onde contém as informações necessárias para percorrer toda a rede. Usando as redes de Petri de alto nível a cartela clínica é representada por um *token* dotado de uma oportuna estrutura.

O ingresso no processo representa a fase na qual são inseridos os sintomas encontrados no paciente (geralmente através de uma visita médica). Este é realizado com um lugar de entrada, no qual são marcados os *tokens* relativos aos pacientes que devem ser submetidos à análise, e uma transição a qual dá inicio ao processo de diagnóstico.

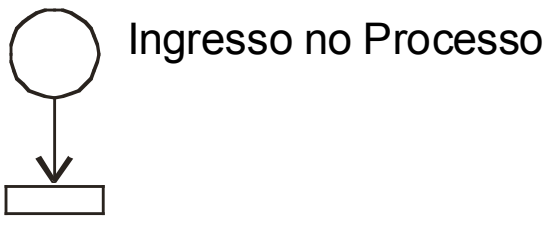


FIGURA 3.3 – Ingresso no processo

O itinerário seguido pelo paciente durante a averiguação do diagnóstico é reconduzido ao caminho do *token* que representa a cartela clínica na rede de Petri.

As decisões sobre as futuras atividades a serem desenvolvidas são determinadas pelo êxito de uma combinação de sintomas estabelecidos no momento de ingresso do paciente no processo. Todas as atividades ou pontos de decisão são realizados mediante um lugar que recolhe um *token* o qual chega de uma transição proveniente de um processo anterior e o envia a uma transição de saída. O número de saídas depende dos diferentes resultados ou decisões assumidas.

Entre as atividades representadas a, execução de um exame tem uma importância particular. Tal operação é esquematizada como o envio de uma requisição de execução a um determinado Laboratório e a resposta do resultado obtido por ele.

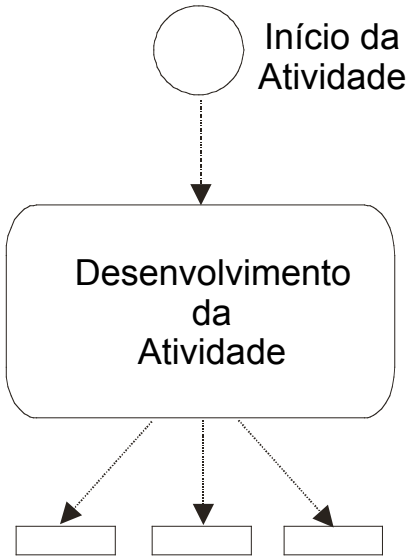


FIGURA 3.4 – Modelo geral de uma atividade

Observa-se que todas as requisições de um certo tipo serão feitas a um mesmo Laboratório que é uma sub-rede de Petri apta a realizar tal operação.

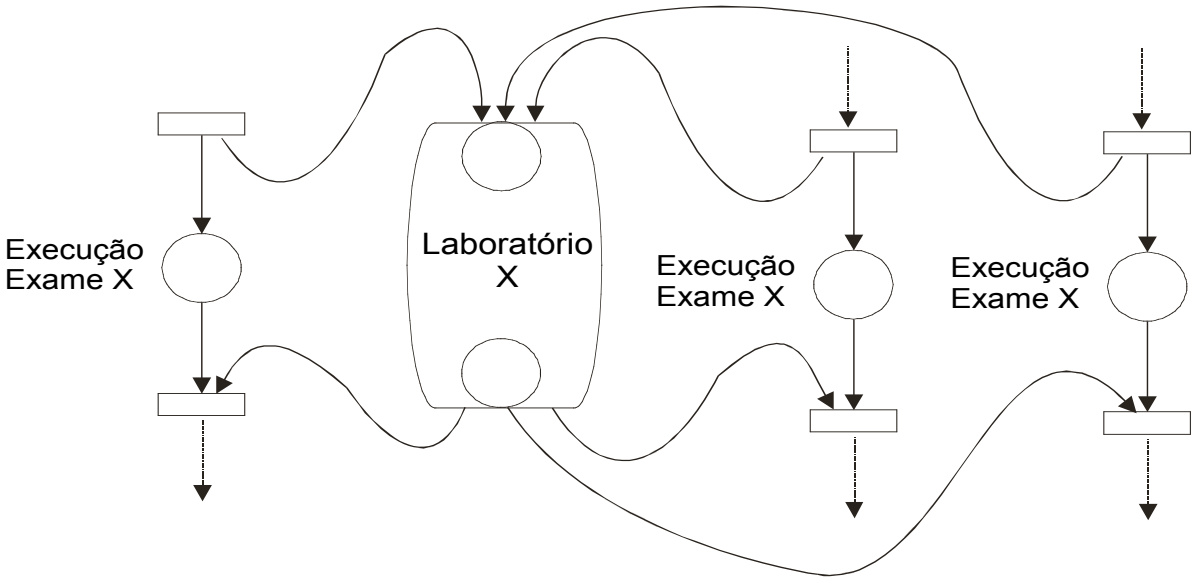


FIGURA 3.5 – Execução de um exame

O diagnóstico considera-se concluído quando o *token* que representa a cartela clínica do paciente chega a um lugar terminal ao qual são ligados os arcos de saída.

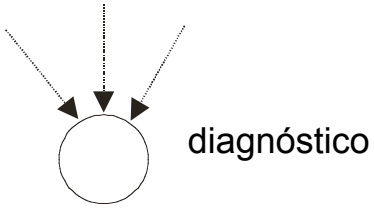


FIGURA 3.6 – Fim do processo

Nota-se que cada atividade é projetada de modo que possa ser conectada a outra facilmente: em um ingresso encontra-se um lugar em correspondência com uma transição. A conexão entre duas atividades se dá simplesmente pela transição de saída com um lugar de ingresso.

Observação sobre a utilização das redes de Petri

A modelagem dos protocolos médicos com redes de Petri provou uma grande capacidade para cumprir todos os quesitos ou princípios descritos no capítulo anterior.

Na figura 3.7 é mostrada como exemplo uma definição parcial de um protocolo de diagnóstico[Bar94a]. Uma descrição da estrutura da rede é desenvolvida no próximo capítulo, onde se introduz a origem da notação LEMMA.

Fica claro que, neste modelo, são aproveitadas as vantagens de uma notação formal. Não se pode, neste ponto, considerar resolvido o problema da modelagem dos processo clínico da parte do pessoal médico, que geralmente não é qualificado sob o ponto de vista informático.

O problema maior consiste na escassa legibilidade do modelo devido a complexidade topológica e de codificação dessa rede. Um processo, por mais simples que se possa conceber, necessitará de um número muito elevado de lugares e transições, também de um número elevado de instruções contidas nos predicados.

Deve-se observar, ainda que a manutenção(realização e modificação) desta rede requer um tempo elevado.

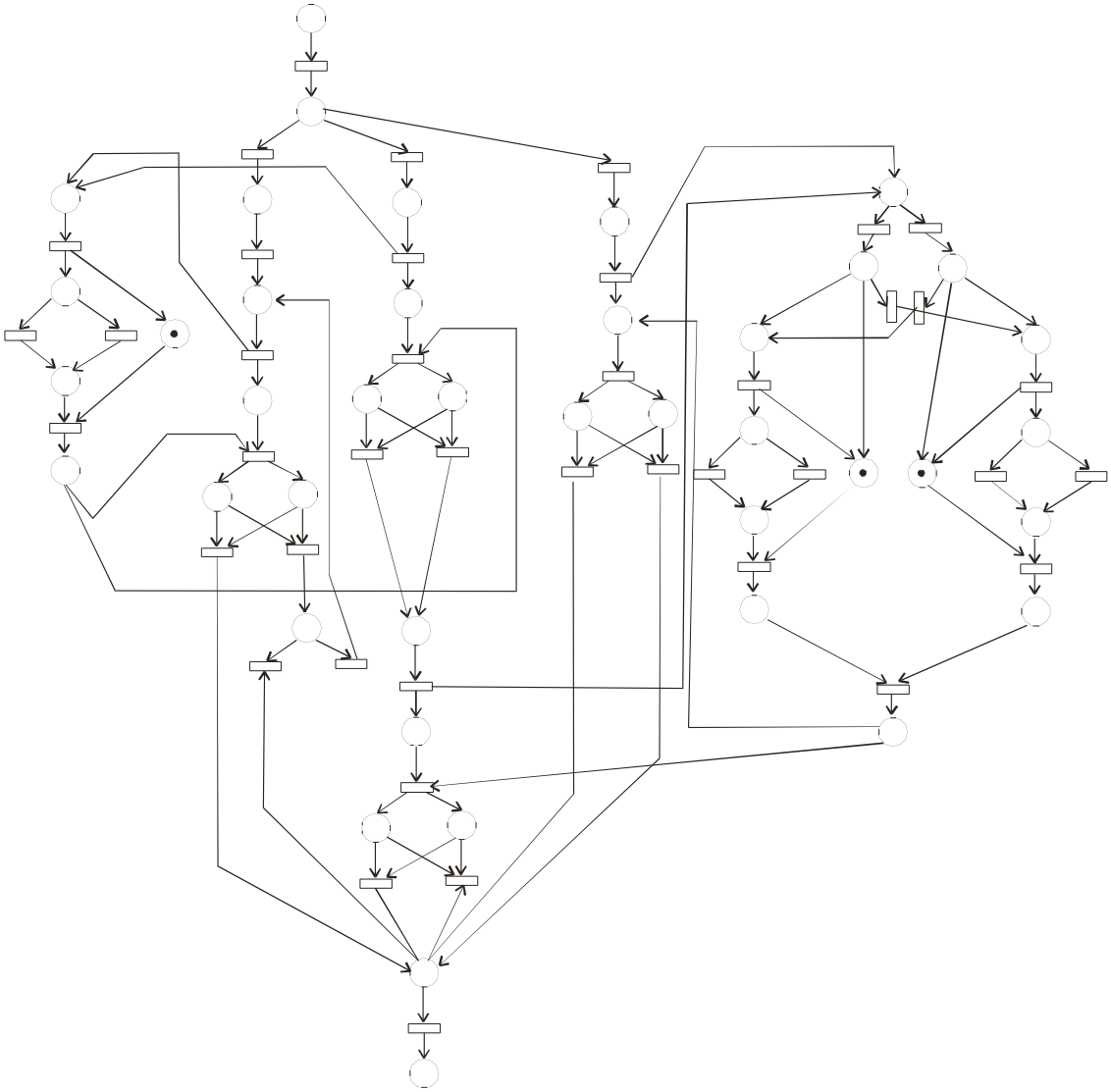


FIGURA 3.7 – Representação de um processo clínico com redes de Petri

4 Notação Lemma

Neste capítulo é ilustrado o processo de desenvolvimento que levou a definir a notação simplificada LEMMA 2000(Language for na Easy Medical Model Analysis) para a especificação dos processos clínicos. Esta notação teve sua primeira versão desenvolvida pelos engenheiros do Politécnico de Milão a qual é mostrada na primeira parte deste capítulo, na seqüência será descrita a versão 2000.

4.1 A Notação

Sobre a base da modelagem feita com redes de Petri existe a necessidade de uma notação para especificar os protocolos médicos que seja de fácil compreensão e utilização por usuários não-informatas, mas que ao mesmo tempo, permita a análise e verificação de uma linguagem formal.

O foco de desenvolvimento concentra-se em associar sub-redes de Petri a unidades funcionais evidenciadas por gráficos. Então, são individualizadas sub-redes que apresentam uma estrutura similar ou uma funcionalidade bem específica. Um exemplo da abstração das sub-redes em tais "blocos" é mostrada na figura 4.1. As sub-redes estão marcadas com retículas diversas, para evidenciá-las conforme sua utilização.

O ponto forte desta notação é a capacidade de acumular duas propriedades:

- Simplicidade de uso: os seus elementos consistem em uma especificação gráfica de processos clínicos de fácil compreensão e elevada legibilidade.
- Formalidade dos modelos: consiste em desfrutar todas as vantagens de uma notação formal, especialmente no campo da análise.

Cabe salientar que este modelo de rede de Petri foi desenvolvido pelo Politécnico de Milão[Bar94a] e serviu de base para a elaboração da primeira versão da notação LEMMA.

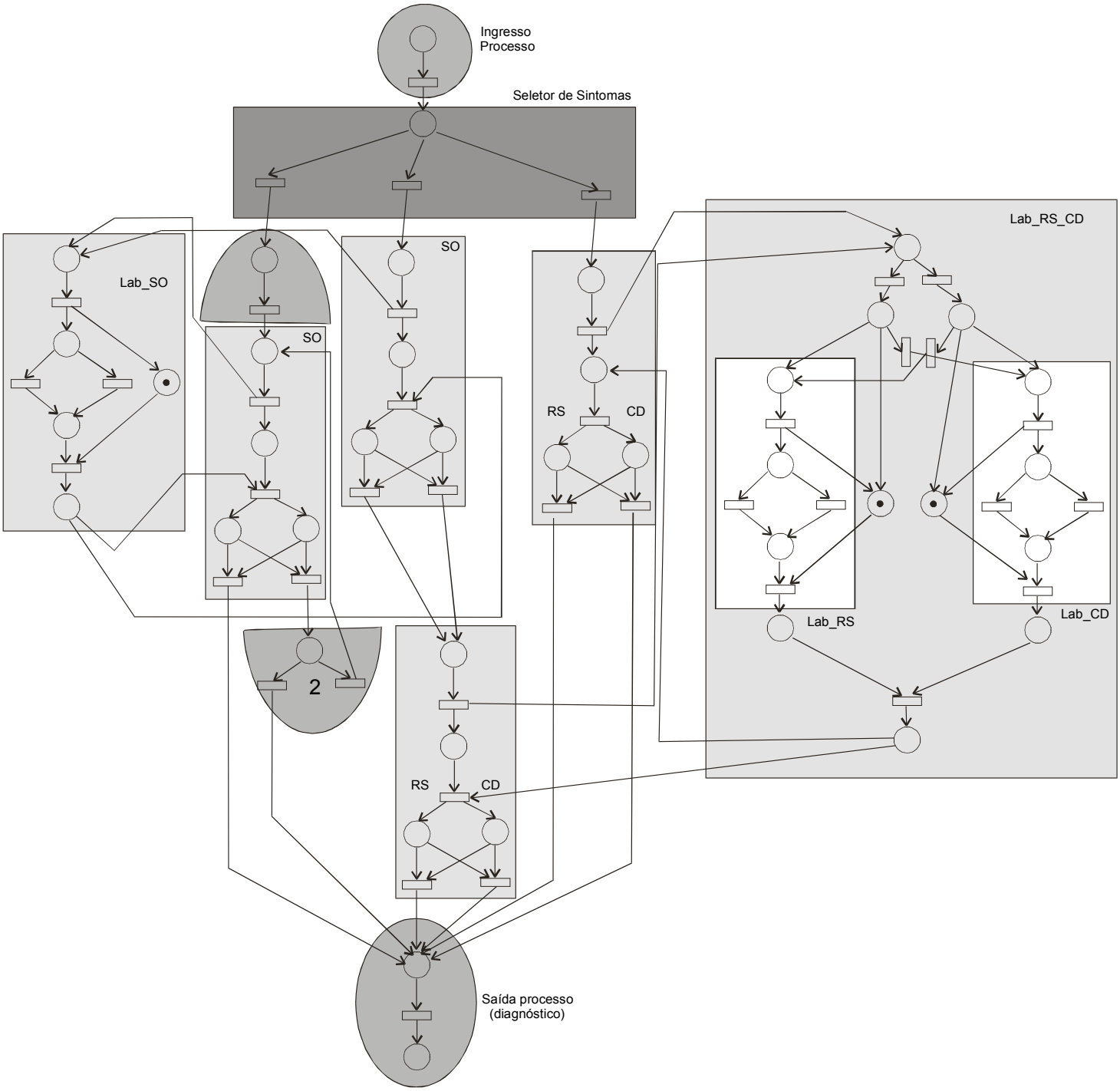


FIGURA 4.1 A Notação

4.2 Especificação dos Elementos da Notação

Um modelo realizado com a notação LEMMA é constituído dos seguintes elementos:

- um conjunto de blocos ou elementos que constituem as atividades a serem desenvolvidas ou um ponto onde deve ser feita uma decisão
- um conjunto de conexões que ligam os blocos
- um conjunto de sintomas relativos ao processo representado.

A fim de compreender plenamente um modelo de um processo, são apresentadas as definições que refletem precisamente a escolha da modelagem.

- sintomas do processo: são as categorias que reúnem sintomatologias de um mesmo tipo relativas ao processo diagnóstico
- combinação de sintomas: conjunto de sintomas relativos a um paciente
- êxito: resultado determinado pela execução de um exame
- combinação de êxitos: conjunto de resultados relativos a uma seqüência de exames reunidos em um único resultado.

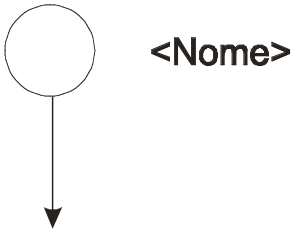
A seguir são mostrados as propriedades dos elementos da notação LEMMA.

4.3 LEMMA

Esta versão da notação foi desenvolvida no departamento de Engenharia de Software do Politécnico de Milão. Os elementos estão descritos separadamente.

Nome: Ingresso no Processo

Símbolo:



Descrição: Elemento que representa o ingresso no processo. Um processo pode possuir muito pontos de entrada. Por exemplo, um paciente pode ter efetuado alguma análise, a qual permita entrar no processo em um ponto mais avançado.

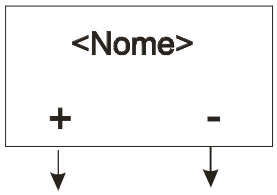
Ponto de Ingresso: Nenhum

Saídas: 1

Parâmetros: Nome de identificação

Nome: Exame(2 saídas)

Símbolo:



Descrição: Elemento que representa a execução do exame identificado pelo nome assinalado. Neste elemento, sempre se terá êxito positivo ou negativo.

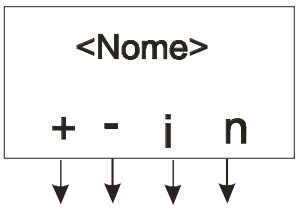
Ponto de Ingresso: 1

Saídas: 2 (êxito positivo, negativo)

Parâmetros: Nome de identificação

Nome: Exame (4 saídas)

Símbolo:



Descrição: Elemento que representa a execução do exame identificado pelo nome assinalado. Os êxitos possíveis neste caso são 4: positivo, negativo, incerto e nulo.

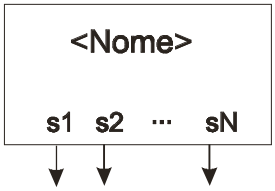
Ponto de Ingresso: 1

Saídas: 4 (êxito positivo, negativo, incerto, nulo)

Parâmetros: Nome de identificação

Nome: Conjunto de Exames

Símbolo:



Descrição: Elemento que representa o desenvolvimento de um conjunto de exames com uma ordem de execução não-conhecido. A determinação do número de saídas é feita pelo usuário. Este deve informar, a cada saída, quais são as combinações de êxitos que levam a esta.

Ponto de Ingresso: 1

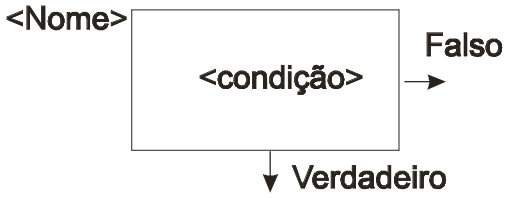
Saídas: N

Parâmetros: Nome que identifica o conjunto de exames.

Devido ao número elevado de possibilidades de êxitos, este modelo de conjunto de exames pode tornar-se muito complexo e difícil de gerir. A solução proposta é ordenar os êxitos dos vários exames segundo uma hierarquia especificada pelo usuário.

Nome: Seletor de Sintomas

Símbolo:



Descrição: Elemento que representa um ponto no diagnóstico, no qual é decidido qual caminho dever ser percorrido pelo paciente.

Ponto de Ingresso: 1

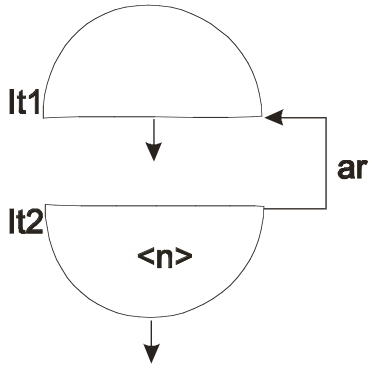
Saídas: 2 (verdadeiro, falso)

Parâmetros: Nome de identificação

Condição: combinação de sintomas

Nome: Repetidor

Símbolo:



Descrição: Este elemento regula o fluxo de pacientes entre dois pontos do processo. Indica que deve repetir o percurso compreendido entre o elemento IT1 e o inferior IT2, no máximo N vezes, passando pelo arco de retorno. Após terminadas as repetições o fluxo passa pelo arco de saída. Pode acontecer de ser repetido um número menor que o indicado; por exemplo um exame deve ser refeito somente com o êxito negativo, e este teve um resultado positivo na primeira execução.

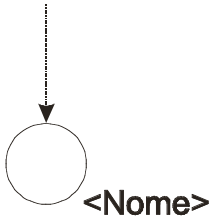
Ponto de Ingresso: 2

Saídas: 2 (+ arco de retorno)

Parâmetros: Número de repetições

Nome: Saída do processo

Símbolo:



Descrição: Indica um ponto terminal do processo. Isto significa que o paciente tem um diagnóstico e pode ser aplicado um tratamento adequado. Um processo pode ter vários pontos de saída.

Ponto de Ingresso: 1

Saídas: Nenhum

Parâmetros: Nome de identificação, normalmente relaciona a um diagnóstico ou conclusão.

Mesmo tendo sido inspirada em modelos de protocolos médicos, esta notação possui alguns problemas. Com sua aplicação em alguns modelos de protocolos de neurologia foram verificadas algumas deficiências como:

- Falta de um elemento que possibilite seleção múltipla;
- Todos os protocolos devem sempre ser representados em um único modelo, deveria existir um elemento hierárquico que possibilitasse o encapsulamento de submodelos;
- O elemento conjunto de exames é de difícil utilização e pouco flexível tornando sua aplicação muito restrita;
- Os símbolos gráficos são muito parecidos, o que torna os modelos representados de difícil leitura e compreensão.

Na próxima etapa, é discutida a nova notação em que foram feitas várias alterações no modelo original a fim de resolver tais problemas.

4.4 Lemma 2000

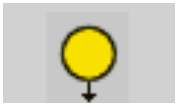
Nesta etapa estão listados os elementos da notação LEMMA 2000, os quais foram inspirados no modelo anterior, com algumas alterações na representação gráfica e o acréscimo de alguns elementos novos.

4.4.1 Elementos e Grupos

Foram feitas algumas alterações nos símbolos utilizados para a representação dos elementos da notação em relação à versão anterior. Foi criado um padrão para diferenciar os elementos em grupos divididos em categorias tipo exames, que são sempre representados por retângulos; elementos de controle do fluxo de execução, que são representados por losangos; elementos iniciais e terminais por círculos; e, por último o elemento hierárquico que é representado por um retângulo com as bordas arredondadas. Também para diferenciar cada elemento dentro dessas categorias, foi utilizada uma cor que os identifica, como pode ser observado nas ilustrações abaixo.

Nome: Ingresso no processo

Símbolo:



Descrição: Elemento que representa o ingresso no processo. Um processo pode possuir muito pontos de entrada. Por exemplo, um paciente pode ter efetuado alguma análise permitindo entrar em um ponto mais avançado.

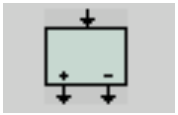
Ponto de ingresso: Nenhum

Saídas: 1

Parâmetros: Nome: nome de identificação deste processo

Nome: Exame (2 saídas)

Símbolo:



Descrição: Elemento que representa a execução do exame identificado pelo nome assinalado. Neste elemento, sempre teremos êxito positivo ou negativo.

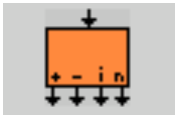
Ponto de ingresso: 1

Saídas: 2 (êxito positivo, negativo)

Parâmetros: Nome: nome que identifica o exame

Nome: Exame (4 saídas)

Símbolo:



Descrição: Elemento que representa a execução do exame identificado pelo nome assinalado. Os êxitos possíveis neste caso são 4: positivo, negativo, incerto e nulo.

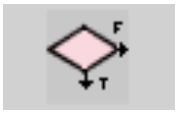
Ponto de ingresso: 1

Saídas: 4 (êxito positivo, negativo, incerto e nulo)

Parâmetros: Nome: nome que identifica o exame

Nome:Seletor de sintomas

Símbolo:



Descrição: Elemento que representa um ponto no diagnóstico, onde é decido qual caminho dever ser percorrido pelo paciente. A decisão é tomada no confronto entre os sintomas encontrados no paciente e uma combinação indicada pelo usuário que inseriu este elemento no modelo. Esta combinação pode ser visualizada e editada em uma caixa mostrada no momento da inserção do elemento ou quando for solicitado pelo usuário.

Ponto de ingresso:1

Saídas: 2 (verdadeiro, falso)

Parâmetros: Nome: nome dado pelo usuário.

Condição: Combinação de sintomas.

Nome: Repetidor

Símbolo:

Descrição: Este elemento regula o fluxo de pacientes entre dois pontos do processo. Indica que deve repetir-se o percurso compreendido entre o elemento a que o arco de retorno está ligado e o elemento anterior ao repetidor no máximo N vezes, passando pelo arco de retorno. Após terminadas as repetições o fluxo passa pelo arco de saída. Pode acontecer de ser repetido um número menor que o indicado; por exemplo, um exame deve ser repetido somente com o êxito negativo, e este teve um resultado positivo na primeira execução.

Ponto de ingresso: 1

Saídas: 2 (normal e arco de retorno)

Parâmetros: N: número de repetições.

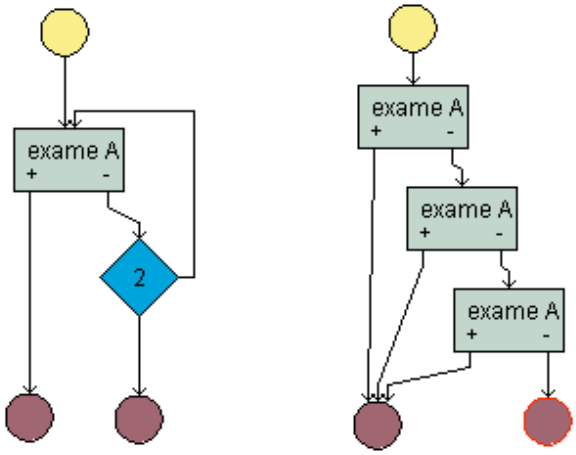
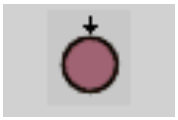


FIGURA 4.2 – Exemplo de modelo com repetidor e seu equivalente

Nome: Saída do processo

Símbolo:



Descrição: Indica um ponto terminal do processo. Isto significa que o paciente tem um diagnóstico e pode se aplicado um tratamento adequado. Um processo pode ter vários pontos de saída.

Ponto de ingresso: 1

Saídas: Nenhuma

Parâmetros: Nome (nome informado pelo usuário)

Elemento Hierárquico e sua utilização

Nome: Elemento hierárquico

Símbolo:

Descrição:

Modelos de protocolos médicos, via de regra, são muito complexos e extensos. Por mais simples que seja uma notação, representar tais modelos é sempre uma tarefa não-trivial e de difícil interpretação. Para reduzir tal problema foi introduzido um elemento hierárquico que possibilita o encapsulamento de modelos menores. Este elemento possui um ponto de ingresso e *n* pontos de saída. À medida que o usuário acrescenta elementos de saída no subgráfico, são incluídas setas no elemento hospedeiro, como mostra a figura abaixo.

Visto que é possível criar cascatas de seletores, é agregada a possibilidade de serem utilizados vários pontos de saída. Ainda é possível utilizar tal elemento como um seletor múltiplo, característica muito importante para um modelo com ramificações muito grandes.

Ponto de ingresso: 1

Saídas: Indeterminadas

Parâmetros:Nome (informado pelo usuário)

Submodelo:

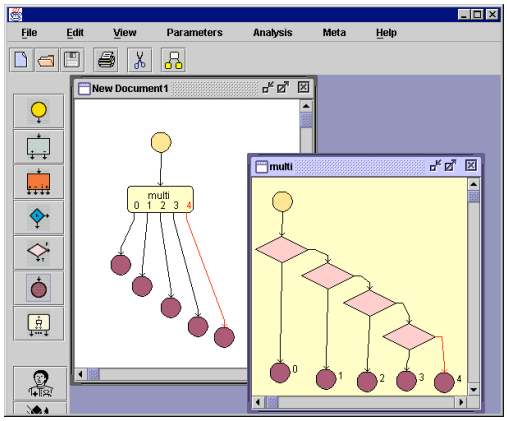


FIGURA 4.3 – Elemento Hierárquico

4.5 Análise

Nesta seção são descritas as análises efetuadas sobre os processos clínicos especificados mediante a notação LEMMA 2000. Para que o modelo obtido seja a representação de um processo significativo devem ser respeitadas algumas regras que são definidas a seguir.

Definição 4.1 - Uma conexão é dita válida se liga qualquer saída a qualquer ponto de ingresso.

Definição 4.2 - Um processo diagnóstico definido mediante a notação LEMMA 2000 é dito **válido** se:

- Tem pelo menos um ponto de ingresso;
- Tem pelo menos um ponto de saída;
- Todas as saídas dos blocos estão conectadas;
- Todos os blocos, exceto Ingresso, têm pelo menos uma conexão de ingresso.

Definição 4.3 - Um processo diagnóstico definido mediante a notação LEMMA 2000 é dito **correto** se:

- É válido;
- Não existem ciclos(Definição 4.4);
- Não existem caminhos não percorridos, relativos aos seletores e os sintomas definidos.

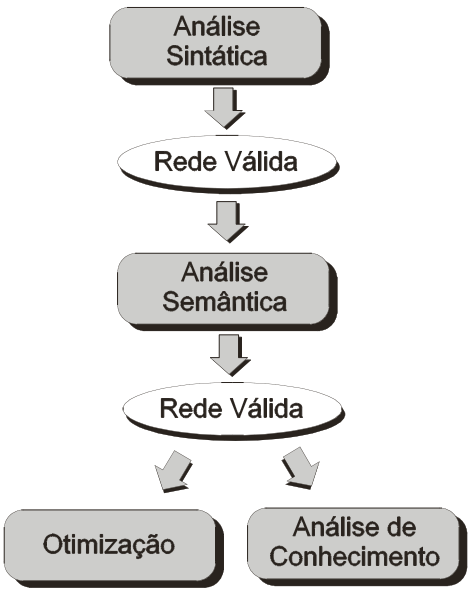


FIGURA 4.4 – Esquema Lógico de Análise

Definição 4.4 - Considerando um processo descrito com a notação LEMMA 2000, define-se ciclo como um caminho obtido sem percorrer os arcos de retorno de um repetidor, cujo elemento final coincide com o inicial.

Esquema lógico da Análise

Sobre um modelo LEMMA 2000, podem ser efetuados quatro conjuntos de análises: os primeiros dois são efetuados em cascata e os segundos podem ser efetuados em uma ordem não-precisa. A figura 4.4 mostra um esquema lógico com estas análises.

Os quatro tipos de análise são:

Análise Sintática

O escopo desta análise é verificar a validade da rede em questão. É verificada a existência de todos os elementos constituintes de um modelo.

O termo sintático refere-se ao fato de que não existam erros na utilização dos elementos, isoladamente, e de que se respeitem algumas regras para que, deste modo, possam vir a representar efetivamente um processo. Na figura 4.5 é mostrado um processo não-válido onde são incluídas duas saídas desconectadas.

Análise Semântica

O escopo desta análise é verificar a corretude da rede em questão, dada a validade comprovada com a análise sintática. Verifica-se que a notação seja utilizada de modo a obter um modelo que represente um processo significativo. Um exemplo de processo não-correto é mostrado na figura 4.6.

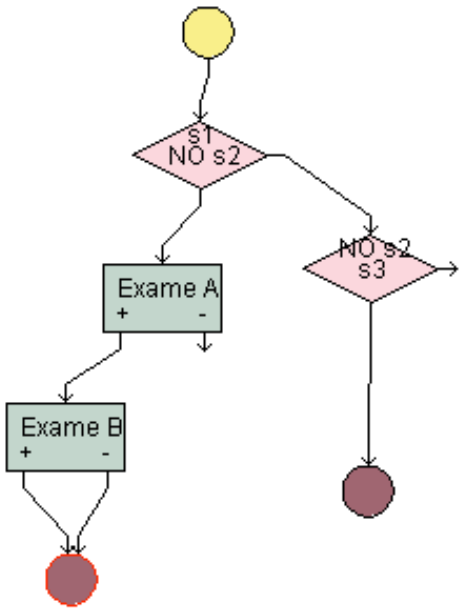


FIGURA 4.5 – Exemplo de um processo não-válido

.

O termo semântico é utilizado porque nesta análise são investigadas as erradas aplicações das relações (conexões) entre os elementos do modelo que levam a uma situação não-correta do ponto de vista topológico ou lógico.

A verificação de caminhos não percorridos é necessária, pois com a utilização de um número elevado de seletores, pode ocorrer uma combinação de sintomas no qual tem um valor *booleano* indeterminável. Um exemplo é mostrado na figura 4.7, no qual a saída verdadeira do seletor 3 não será ativada, pois a configuração de sintomas a esta associada habilita a saída verdadeira do seletor 1.

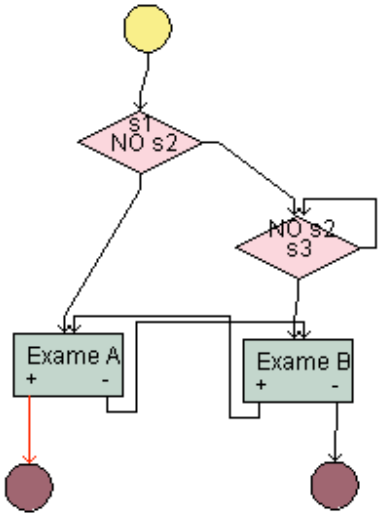


FIGURA 4.6 – Exemplo de processo não-correto: foram introduzidos dois ciclos com as conexões entre exame A e B, e com a saída falsa de Seletor 2.

Análise de Conhecimento

Permite levantar informações relevantes de caráter geral sobre o processo em exame. Podem ser de diversos tipos:

Considerações descritivas: desfrutando das características gráficas da notação é possível revelar de modo imediato:

- Os ponto de ingresso;
- Os pontos de diagnóstico ou saída a outros processos;
- Os pontos de decisão;
- A ordem de grandeza do número de exames;

- Os percursos de diagnósticos que levam a uma dada situação.

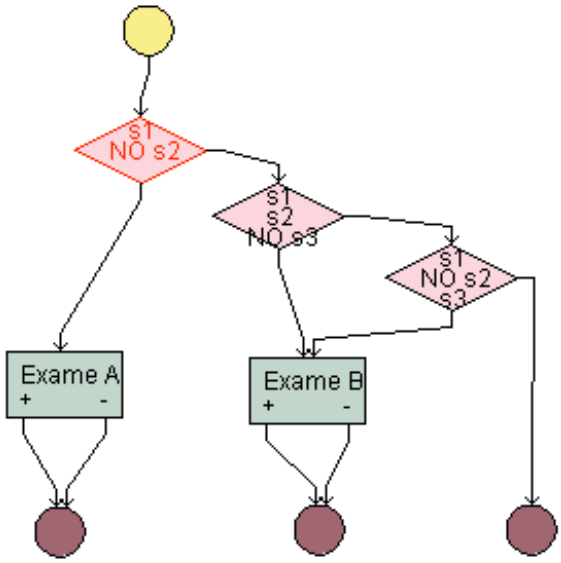


FIGURA 4.7 – Exemplo de má utilização dos seletores: não existe nenhuma configuração de sintomas atribuíveis ao paciente que possa ativar a saída verdadeira do Seletor 3.

Todas estas informações podem ser colhidas da leitura direta do modelo da parte do usuário médico, também sem o auxílio de instrumentos informatizados, graças à expressividade e o poder de síntese oferecida pela notação LEMMA 2000.

Com estas informações, podem ser feitas estimativas de custos e de utilização dos recursos e, sobre estes, efetuar algumas otimizações.

Otimização

Desfrutando das informações disponíveis no modelo, é possível ajudar os médicos a efetuarem observações e otimizações nos diversos caminhos em que o paciente percorre no interior de um processo de diagnóstico.

5 Semântica Lemma

Neste capítulo é apresentada a semântica operacional de cada elemento da notação LEMMA, seus modelos correspondentes em redes de Petri de alto nível.

5.1 Correspondência semântica dos elementos

Dado um processo descrito mediante a notação LEMMA, é possível reconduzir a uma Rede de Petri de alto nível correspondente. Isto é possível pois para cada elemento da notação abstrata existe uma rede de Petri equivalente. O modelo assim definido poderá usufruir da simplicidade da representação LEMMA e o formalismo das redes de Petri.

Para cada transição são associados predicados e ações. Recordando, um predicado é uma condição para que a transição seja executada.

Ações

- Ação de transferência do conteúdo de um token de um lugar para outro. È sempre indicado com a::trasf.
- Ação de requisição de um exame, transfere o código de um paciente para um laboratório. É indicada com a::req
- Ação de atribuição do êxito de um exame. È indicada com a::attr_ex

Predicados

O predicado de controle do campo código é utilizado para sincronizar os tokens que esperam um paciente.

p::cód

<nome1>.cod == <nome2>.cód

predicado de controle do campo êxito de um lugar.

p::êxito

<nome1>.êxito == <valor_êxito>

Onde os predicados não são especificados é sempre atribuído um valor verdadeiro.

5.1.1Ingresso no Processo

O lugar “início” define a entrada no processo, onde são incluídos as marcas que representam os pacientes bem como seus predicados que representam um conjunto de sintomas.

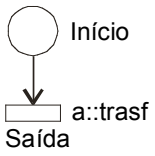


FIGURA 5.1 - Ingresso no processo

5.4.1 Exame de duas saídas

O lugar de início recebe as marcas que vêm dos blocos precedentes;. A transição “requisição” transfere uma marca a uma sub-rede “laboratório”, que realiza o desenvolvimento de um exame propriamente dito e uma marca ao lugar “controle”. O “laboratório” genérico encarregasse de receber todas as requisições para o exame e após verificar se o número de usuários não superou o limite, analisa e retorna o resultado. Na transição “resposta” a marca em “controle” memoriza o resultado do exame sincronizado pelo código do

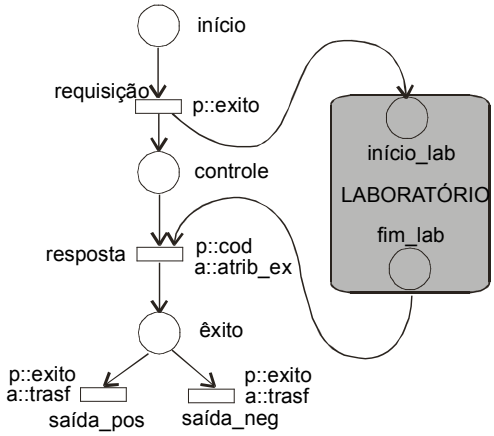


FIGURA 5.2 - Exame com duas saídas

paciente. A saída deste processo dar-se-á por duas transições disparadas conforme o resultado do exame analisando os atributos das marcas.

Laboratório

Um laboratório genérico recebe a requisição de um serviço no lugar "inicio_lab". A transição "verificação" encarrega-se de controlar o número máximo de usuários utilizando as marcas armazenadas em "N_vagas". Este número de usuários é iniciado no momento da definição do exame. A seguir é efetuada a tomada de decisão onde será atualizado o predicado da marca que representa o resultado do exame, que neste caso pode ser negativo ou positivo. A transição "restauração" conduz a marca ao final do processo deste laboratório e libera mais um usuário para N_vagas.

5.1.3 Exame de 4 saídas

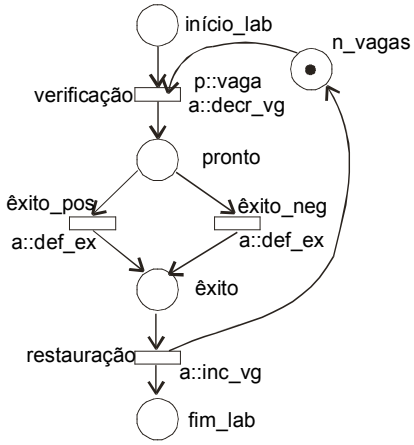


FIGURA 5.3 – Laboratório genérico

O comportamento desta rede é exatamente o mesmo da rede do exame de duas saídas. A diferença principal consiste no acréscimo de mais duas saídas. Foram introduzidas as alternativas, incerto e nulo. Também neste caso só uma destas saídas poderá ser ativada.

O laboratório empregado nesta rede diferencia-se do anterior exatamente no ponto onde é tomada a decisão do resultado, entre os lugares "pronto" e "êxito".

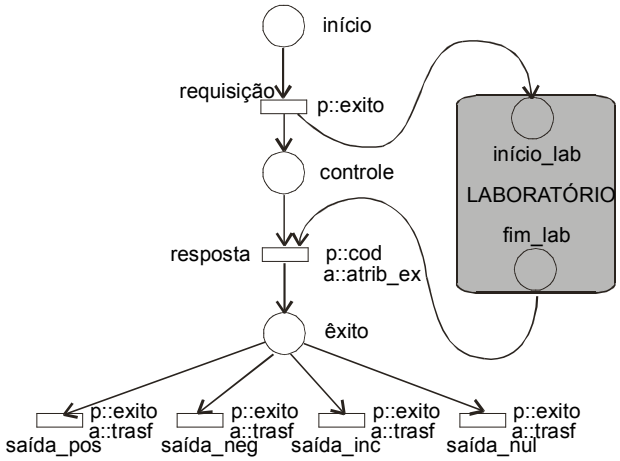


FIGURA 5.4 - Exame com 4 saídas

O laboratório utilizado nesta rede diferencia-se do visto anteriormente no ponto de escolha do êxito, como é mostrado na figura 5.5.

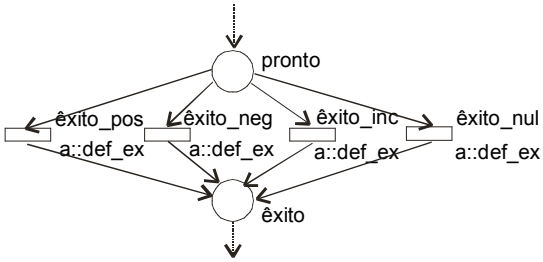


FIGURA 5.5 – Laboratório para exame com 4 saídas

5.1.4 Seletor de sintomas

Este elemento é constituído simplesmente de um lugar que recebe os tokens que chegam e duas transições, saída_verdadeira e saída_falsa, que se ocupam de estabelecer qual caminho deverá ser percorrido. Tal saída é efetuada avaliando a combinação de sintomas ligada ao paciente e as ligadas aos predicados de transições de saída.

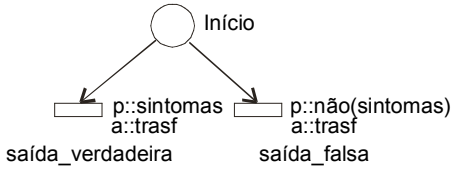


FIGURA 5.6 - Seletor de sintomas

5.1.5 Repetidor

Este elemento recebe alguns predicados e ações específicas:

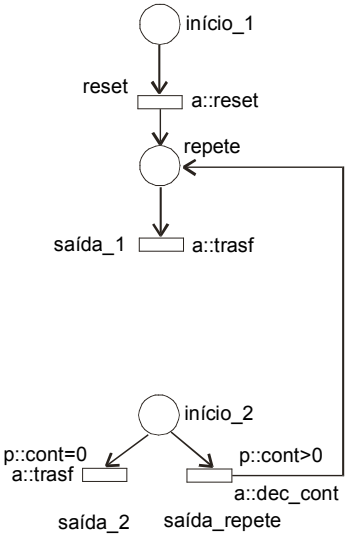


FIGURA 5.7 - Repetidor

Os predicados que guardam o controle do valor do contador cont que memoriza o número de repetições que devem ser efetuadas são:

p::cont=0

<nome>.cont == 0

p::cont>0

<nome>.cont > 0

As ações que se ocupam por atualizar o valor do contador são:

a::reset

<nome>.cont = N;

a::dec_cont

<nome>.cont--;

A rede correspondente ao repetidor é composta por duas partes: a primeira inicia com o lugar incio_1 e a segunda no lugar início_2. A primeira parte tem a função de inicializar o contador através da transição "reset" com o número de repetições a serem efetuadas. A segunda parte tem a função de controlar o final do ciclo de repetições, através da transições saída_2, ativada pelo predicado p::cont=0 e saída_repete que é ativada pelo predicado p::cont>0 e executa uma ação de decrementar cont. Os elemento ligados entre estas duas partes serão sempre executadas no máximo N+1 vezes.

5.1.6 Saída do Processo

Esta rede somente tem a função de receber tokens que chegam no lugar início e transferi-los ao lugar "fim_processo".

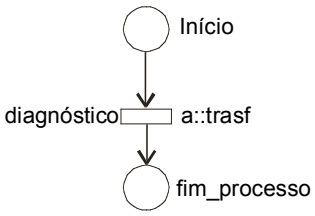


FIGURA 5.8 – Saída do processo

6 Protótipo

Neste capítulo é descrito o projeto do protótipo para a experimentação de LEMMA 2000. Este protótipo foi concebido para a verificação do enfoque conceitual já apresentado. A aplicação disponibiliza funcionalidades para a gestão gráfica da notação, para a construção dos modelos, e para sua análise e animação.

É preciso gerenciar a correspondência entre os dois modelos, um sintático expresso mediante a notação LEMMA e um semântico relativo as redes de Petri de alto nível. Para resolver este problema existem duas alternativas, a primeira com o desenvolvimento de uma ferramenta completa partindo do zero ou a utilização de ferramentas já existentes. A escolha foi feita pela utilização de alguns recursos disponíveis e a implementação completa do módulo Editor Gráfico.

Um modelo LEMMA 2000 pode ser realizado utilizando-se um Editor gráfico e o modelo formal é gerado através da aplicação das regras sintático/semânticas definidas no *metaeditor* e visualizadas pelo editor *Cabernet*, ferramentas estas desenvolvidas no Departamento de Eletrônica e Informação, Politécnico de Milão.

A princípio é apresentada a tecnologia utilizada e imediatamente após é descrita implementação do protótipo e sua funcionalidade.

6.1Tecnologia Utilizada

Neste trabalho foram utilizadas diversas ferramentas que auxiliaram tanto na implementação do editor gráfico quanto na integração de todos os componentes do processo de criação de um modelo Lemma desde a modelagem com a notação menos formal até as redes de Petri. Nesta seção estão descritas todas as ferramentas utilizadas e como se integram.

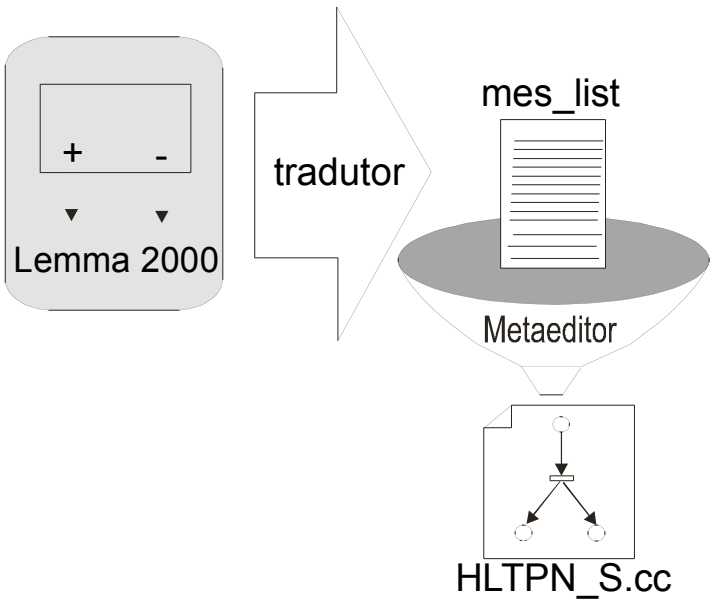


FIGURA 6.1 – Esquema de tradução LEMMA p/ Cabernet

O metaeditor é a principal ferramenta, responsável pela transformação dos modelos representados com a notação LEMMA 2000 para as redes de Petri de alto nível. Este módulo lê um arquivo texto num padrão pré-estabelecido que o editor gráfico de LEMMA cria chama do mes_list, e alimentado pelas regras sintático/semânticas descritas nos capítulos anteriores, gera o arquivo correspondente no padrão Cabernet [PES 94], a ferramenta utilizada para manipular as redes de Petri [Mer 95]. A figura 6.1 mostra este processo.

O metaeditor é um instrumento que precisa ser configurado para que possa compreender a informação contida no arquivo de entrada e consiga criar um modelo consistente. Para isto é necessário que as regras de transformação de um modelo para outro sejam inseridas na sua configuração.

O processo de configuração do meta editor é mostrado na Figura 6.2, e consiste na criação de dois arquivos para cada regra ou elemento de LEMMA. Estes dois arquivos são: um grafo que representa a gramática gerada e um arquivo texto que contém os atributos deste grafo, estas regras são mostradas na seção 6.3 deste capítulo.

Estes dois arquivos devem ser convertidos em um só, através de um conversor chamado fig2trad. Este conversor transforma os dois arquivos de entrada em um só com extensão *.trad*, que é um arquivo texto com um código C++ que deve ser compilado. Para esta tarefa foi utilizado um compilador C chamado CYGNUS.

Após todos este processo teremos um arquivo chamado LEMMAtrad.exe. Este arquivo é o responsável pela configuração do metaeditor.

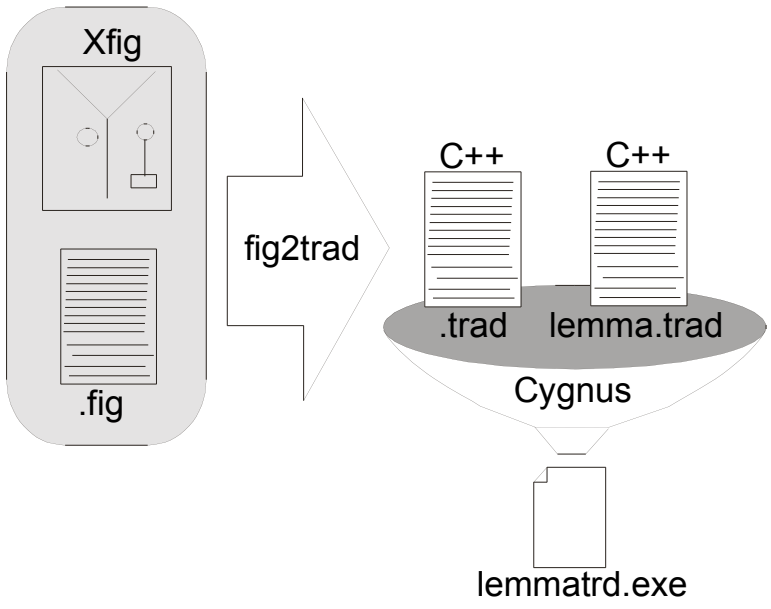


FIGURA 6.2- Esquema de geração dos arquivos para Metaeditor

Para a implementação deste projeto foi escolhida a linguagem JAVA. O maior interesse despertado por esta linguagem foi a portabilidade. Visto que programas escritos em Java podem ser executados em qualquer plataforma que possua seu interpretador e também ao fato de utilizar outras ferramentas, este recurso pode ser muito útil para a integração dos modelos gerados futuramente.

Também o fato de ser orientada a objetos pesou bastante na escolha da linguagem a ser utilizada, pois esta característica facilitou muito a implementação do editor gráfico, bem como as bibliotecas gráficas existentes para esta linguagem.

6.2 Editor Gráfico

Neste segmento é descrito o projeto do módulo desenvolvido para dar suporte a notação Lemma. É apresentada a estrutura completa da aplicação, bem como todos os seus módulos e seu comportamento.

6.2.1 Interface e utilização

Este editor foi desenvolvido para executar a gestão dos modelos representados por LEMMA. Seu projeto considera dois estágios de utilização: módulo usuário e módulo de desenvolvimento.

Por usuário final entende-se, neste caso, os médicos que utilizam o módulo para definir, modificar e analisar os processos clínicos. O segundo, compreende as etapas a serem desenvolvidas por um profissional de informática que possa gerenciar as etapas de criação dos modelos formais, visto que as ferramentas responsáveis por este trabalho necessitam de um treinamento específico e têm um grau de dificuldade avançado.

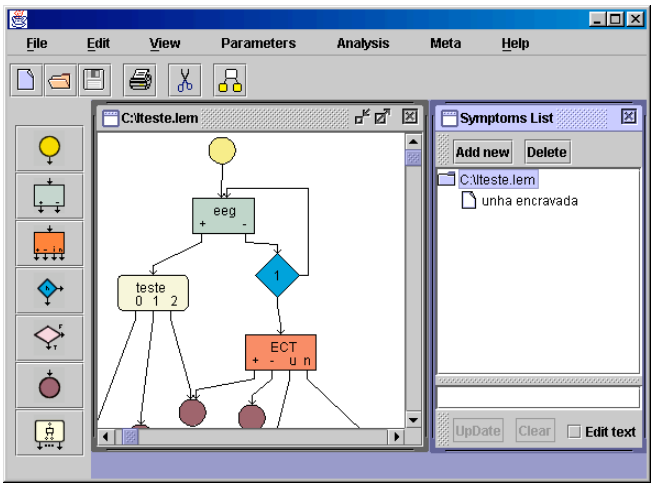


FIGURA 6.3 – Interface Editor Gráfico

Esquema do Editor

Na Figura 6.3 é mostrado uma cópia da interface do editor. Esta é dividida em blocos que facilitam a implementação e seguem o padrão de interface utilizados pelos sistemas operacionais conhecidos. Cada bloco é mostrado a seguir:

-Barra de Menu: parte superior da janela principal e contém os seguintes itens:

File, Edit, View, Parameters, Analysys, Meta, Help.

- Barra de ferramentas: logo abaixo da barra de menus, contém algumas funções de gestão dos modelos como, novo trabalho, abrir um modelo existente, salvar, imprimir, excluir elemento, e expandir um elemento hierárquico.

- Barra de botões: para criar um modelo LEMMA é necessária a inserção dos elementos da notação . Nesta barra estão mostrados os elementos da notação.

- Área de trabalho: localizada ao centro, representada na figura 6.3 por uma janela onde podem ser inseridos os elementos da notação. É possível a edição de vários modelos simultâneos. Bem como janelas de diálogo, onde são exibidas informações como sintomas do paciente e operadores booleanos aplicados ao elemento seletor.

6.2.2 Estrutura dos Elementos

Cada elemento da notação Lemma foi representado por uma classe ou sub-classe A figura 6.4 mostra um diagrama de classes padrão UML. Neste diagrama pode-se observar na parte superior, os elementos responsáveis pela gestão dos editor como: janelas, barras de ferramentas, menus. Na parte inferior estão as classes que representam os elementos da notação Lemma, bem como sua correspondência.

É importante observar que a classe DrawCanvas é uma sub-classe de Canvas

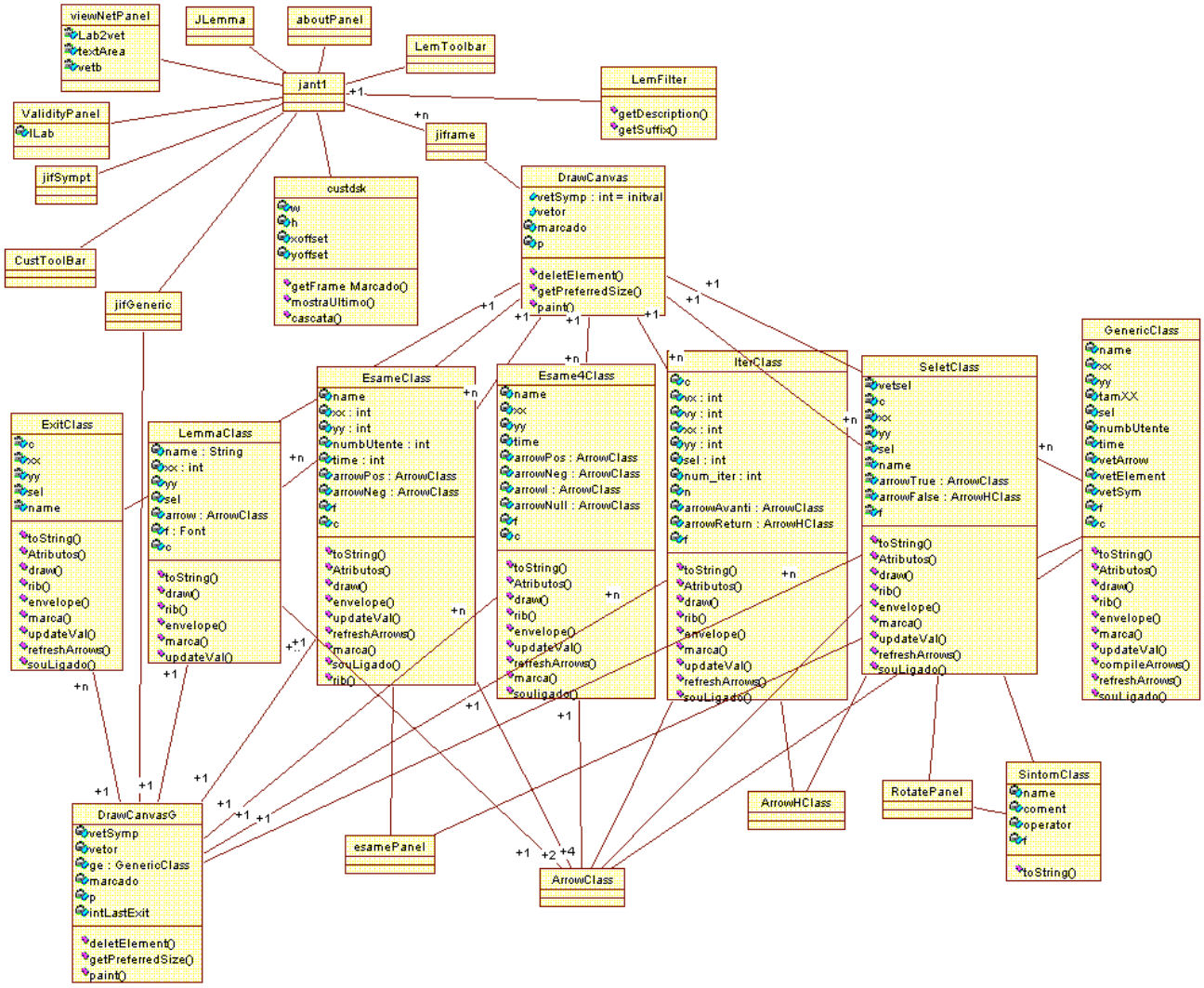


FIGURA 6.4 – Diagrama de Classes - UML

que gerencia o elementos inseridos, ou seja, quando é inicializada uma nova janela de trabalho é instanciado um objeto desta sub-classe. Ela possui duas estruturas de dados principais, que são responsáveis por armazenar os dados referentes aos modelos gerados. Estas estruturas usam uma sub-classe da Classe Container, chamada de Vector,

que implementa um vetor dinâmico, ou seja, sem limitação do número máximo de elementos e sem restrição quanto ao tipo destes.

A primeira estrutura chama-se vetor, este vetor armazena cada objeto instanciado das classes que representam os elementos da notação Lemma. Logo ele é responsável pela estocagem dos modelos gerados.

O segundo vetor armazena uma lista de objetos instanciados de SintomClass, estes objetos representam os sintomas relativos a um processo de diagnóstico. Para cada modelo criado deve ser incluída sua lista de sintomas.

Inserção de elementos

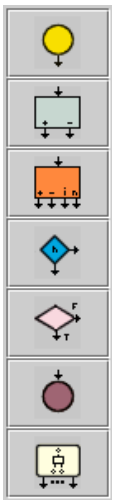


FIGURA 6.5

Os elementos da notação LEMMA 2000 podem ser inseridos em um modelo através do acionamento dos respectivos botões exibidos na figura ao lado. Esta é uma barra de ferramentas que contém todos os elementos da notação e para inseri-los em um modelo basta clicar no botão e inserir no plano de desenho com outro clique. A edição de cada elemento pode ser feita através do duplo clique diretamente no plano de desenho, onde será aberta uma janela de diálogo. Para cada elemento existe uma classe que o representa dentro do modelo orientado à objetos.

Estes objetos são armazenados em um vetor que já foi citado anteriormente.

Gestão de arquivos

FIGURA 6.6

Para a gestão dos modelos representados no editor estão a disposição as funções de manipulação de arquivos. Funções comuns como salvar, abrir e imprimir. A figura ao lado mostra as opções que fazem parte do menu FILE, não serão feitos comentários específicos de cada função pois seu nome já é suficiente para demonstrar sua aplicação. Cabe salientar que funções como Open, save, print, também estão disponíveis na barra de ferramenta superior.

Para efetua
vetor de elemento e o vetor de sintomas. Para isto, em cada classe que representa os elementos da notação e na classe SintomClass foi implementado um método chamado toString que tem a função de transformar os dados de cada objeto instanciado, em seqüências de caracteres(strings). Este método é necessário para que seja possível executar a gravação dos dados de forma serial utilizando o pacote java.io, que implementa os métodos FileInputStream e FileOutputStream, para gravação e leitura dos dados.

Edição de Sintomas

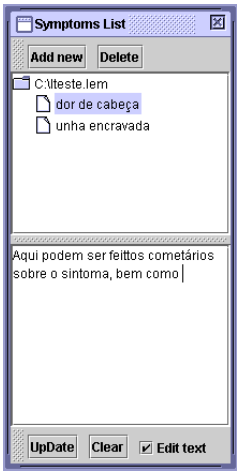


Figura 6.7

Os elementos seletores desviam o fluxo de execução dentro de um modelo avaliando um conjunto de sintomas. A figura ao lado representa a janela de configuração destes sintomas. Na metade superior são exibidas as listas de sintomas. São inseridas uma para cada modelo representado e uma para cada elemento hierárquico.

Cada sintoma é representado por um objeto instanciado da classe SintomClass e a lista é armazenada em um vetor como já foi citado.

Como é possível trabalhar com várias janelas simultâneas, esta janela também possibilita a visualização das listas de sintomas de cada janela que esteja aberta.

Na metade inferior é mostrado o campo de comentários, onde para cada sintoma é possível que o usuário descreva o significado deste sintoma e as características. Este recurso é bastante importante para a representação de modelos mais consistentes.

Configuração dos Elementos Seletores

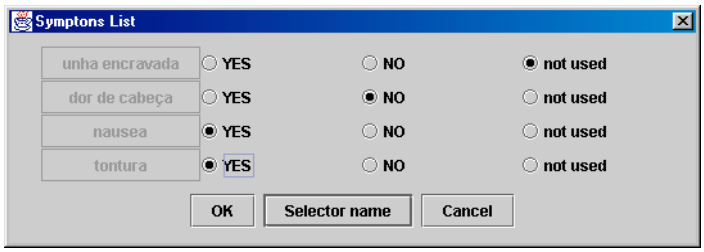


FIGURA 6.8 – Lista de sintomas dos seletores

Os elementos seletores necessitam de uma programação prévia para que possam funcionar adequadamente. Para que seja feito o desvio no fluxo de execução em um processo clínico é avaliado um conjunto de sintomas que podem ser selecionados para cada elemento através de uma janela de configuração mostrada na figura 6.8.

Quando selecionamos no modelo um elemento seletor com um clique duplo, é instaciada a classe RotatePanel, que é uma janela como é mostrada na Figura 6.8. Nesta

janela são inseridos todos os sintomas inseridos no modelo e que estão armazenados em vetSymp da classe Drawcanvas.

Na janela de configuração é então listado todos os sintomas possíveis e estão disponíveis três opções para cada um. A primeira é se este será incluído na lista de testes do seletor, a segunda se será incluído com o operador lógico de negação e a última exclui o determinado sintoma dos testes deste elemento específico. Com a confirmação através do botão OK, esta configuração é armazenada no vetor vetsel implementado dentro da classe SeletClass, esta quando instanciada representa o elemento Seletor.

Para o exemplo da Figura 6.8 o teste lógico para o seletor em questão seria:

Não(dor de cabeça) AND náusea AND tontura

Expansão do Elemento hierárquico

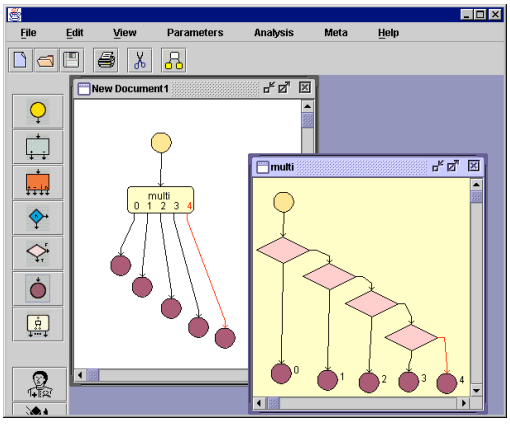


FIGURA 6.9 – Elemento Hierárquico

Como já foi citado anteriormente este elemento possui o recurso de incluir processos clínicos inteiros dentro de sua estrutura. Para editar esta estrutura basta marcar o elemento hierárquico desejado e clicar no botão mais a esquerda da barra de ferramentas superior. Esta ação abre uma janela para e edição dos processos deste item.

Este elemento é representado pela classe GenericClass. Esta classe assemelha-se com as demais classes de elementos da notação, implementando os métodos comuns a todas, mas contém dois vetores em sua estrutura. Estes vetores tem a mesma função dos da classe DrawCanvas, servem para armazenar os elementos inseridos bem como a lista de sintomas específica para este elemento.

A janela de edição do elemento hierárquico diferencia-se das demais por possuir um fundo amarelado. Seus recursos são os mesmos da janela de edição normais, com a exceção de que só é possível a inclusão de um elemento de entrada no processo, pois este item que faz a ligação com os elementos anteriores no diagrama principal.

Outra restrição é a impossibilidade de incluir um elemento hierárquico dentro de outro. O editor só possibilita um nível de execução com este item.

Esta restrição deve-se a complexidade do algoritmo de geração do arquivo de integração com metaeditor. A implementação deste algoritmo é mostrado no anexo B. Para tornar possível a implementação de um código que permita mais níveis de hierarquia, talvez seja necessária a criação de um algoritmo recursivo.

Para cada elemento de saída é incluída uma seta de conexão na representação o elemento hierárquico no diagrama principal. A figura 6.9 mostra uma janela de edição com uma cascata de seletores ramificando em cinco saídas que são representadas por cinco conectores no diagrama principal.

Análise de um processo

FIGURA 6.10

Este menu disponibiliza alguns recursos de teste e análise dos modelos representados como: Validação e corretude, e por último um item que demonstra os resultados encontrados nos itens anteriores. Os resultados da análise também são mostrados em uma janela logo após o termino dos testes.

Em 4.3 foi mostrado como é feita a análise de Validade e Corretude de um modelo Lemma. Quando um destes itens é disparado instancia a classe ValidityPanel onde é implementado o método Ilab, este é o responsável por percorrer o vetor de elementos testando-os conforme as regras já discutidas (4.3).

Correspondência com metaeditor

FIGURA 6.11

Na figura ao lado são exibidos o recursos de comunicação com o metaeditor. A opção Send Net dispara a geração do arquivo de comunicação chamado mes_list. Este arquivo contém comandos que representam os modelos LEMMA codificados nos comandos que irão ser compilados pelo metaeditor. As duas opções finais informam para o meta editor informações afim de tornar possível a simulação em Cabernet.

A construção do
discutido na seção seguint

6.2.3 Módulo Tradutor

Este módulo realiza a tradução de um modelo expresso mediante a notação LEMMA em um equivalente com redes de Petri de alto nível. A tradução é feita mediante a utilização do metaeditor previamente programado através de uma série de regras sintático/semânticas.

A forma de comunicação com o editor gráfico e o metaeditor é um arquivo texto chamado mes_list. Este arquivo é constrído utilizando as regras mostradas na tabela 6.1. Para cada elemento inserido na notação é disparado um determinado comando. Por exemplo, para que seja inserido um seletor de sintomas, são utilizados três comandos:

generic@Addselet@26@"sintomas"

generic@Addconn@26@<proximo elemento>

generic@Addconn@26@<proximo elemento>

Esta seqüência de comandos, quando interpretado pelo metaeditor insere um elemento seletor no modelo de redes de Petry com suas devida conxões. No Anexo A é mostrado um arquivo mês_list completo.

Tabela 6.1 - Comandos de Produção

Comandos de Produção			
n	**Sintática**	**Semântica**	**Descrição**
0	Axiom	Axiom	É o axioma da gramática
1	SY_AddStart	SE_AddStart	Cria um bloco de ingresso no processo
2	SY_ AddEnd	SE_AddEnd	Cria um bloco de saída do processo
3	SY_Conn	SE_Conn	Efetua a conexão entre dois blocos
4	SY_ AddSelet	SE_AddSelet	Cria um bloco tipo seletor
5	SY_ AddIter	SE_AddIter	Cria um bloco tipo repetidor
6	SY_ AddLab2	SE_AddLab2	Adiciona um exame de duas saídas
7	SY_ AddExam2	SE_AddExam2	Cria um exame de duas saídas
8	SY_AddLab4	SE_AddLab4	Adiciona um exame de 4 saídas
9	SY_ AddExam4	SE_AddExam4	Cria um exame de quatro saídas

6.3 Regras sintáticas e Semânticas

Aqui são mostrados os conjuntos de regras de produção sintático/semântico dos elementos da notação LEMMA para a construção das respectivas redes de Petri. Para cada elemento é mostrada a produção sintática e semântica, representas pelos grafos e uma parte textual, que representam os arquivos necessários para o funcionamento do metaeditor. Os grafos foram desenvolvidos no editor Xfig, que em conjunto com a parte textual, são compilados pelo instrumento fig2trad, gerando um arquivo C++, que irá possibilitar o correto funcionamento do metaeditor. O esquema de construção foi mostrado no início do capítulo.

Esta representação utilizada é um gramática de grafos em "Y"[Mer 94][Bar 94b], que é um modelo que se presta para transformações de contexto devido ao acréscimo de uma dimensão que representa justamente o modelo de transformação. A figura 6.12 mostra o esquema de produção com esta notação mostrando seus três elementos:

- Grafo esquerdo: grafo de partida que será substituído após a produção;
- Grafo direito: sub-grafo que substitui o da esquerda;
- Transformação de contexto: define como o contexto do sub-grafo esquerdo deverá ser transformado para o direito. Tipicamente é um conjunto de conexões.

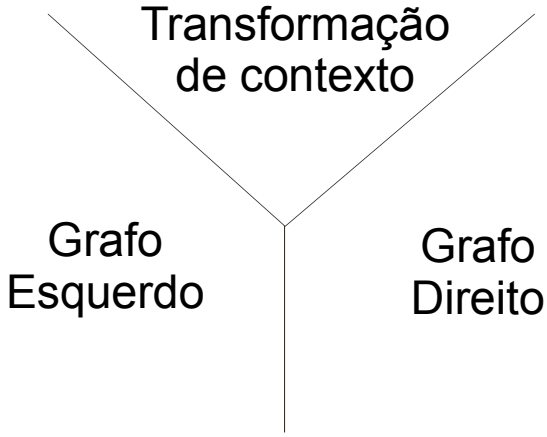


FIGURA 6.12 – Representação em Y das gramáticas de grafo

Na figura 6.14 é ilustrada a evolução de uma rede de Petri após a utilização da produção da Figura 6.13 sobre o lugar P2.

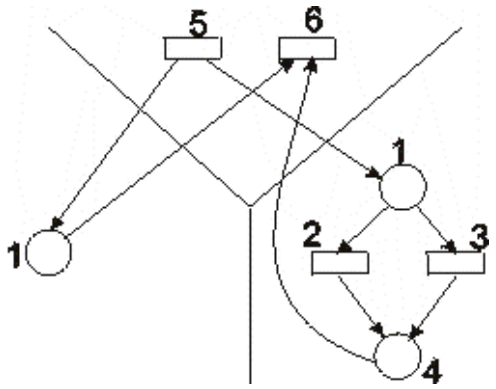


FIGURA 6.13 – Exemplo de produção para rede de Petri

Cada nodo interno da produção é identificado com um índice que é utilizado para o cálculo dos atributos. Os atributos são os valores associados aos nodos dos grafos que especificam o nome, o tipo e outras características a eles associadas. A ligação entre os nodos do grafo e seus atributos é feita por meio de uma parte textual que pode ser escrita para cada produção. Esta parte textual contém instruções do tipo *índice.atributo=....*

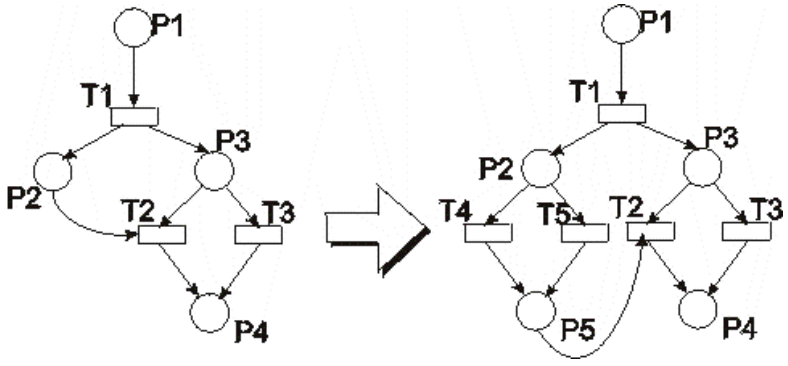
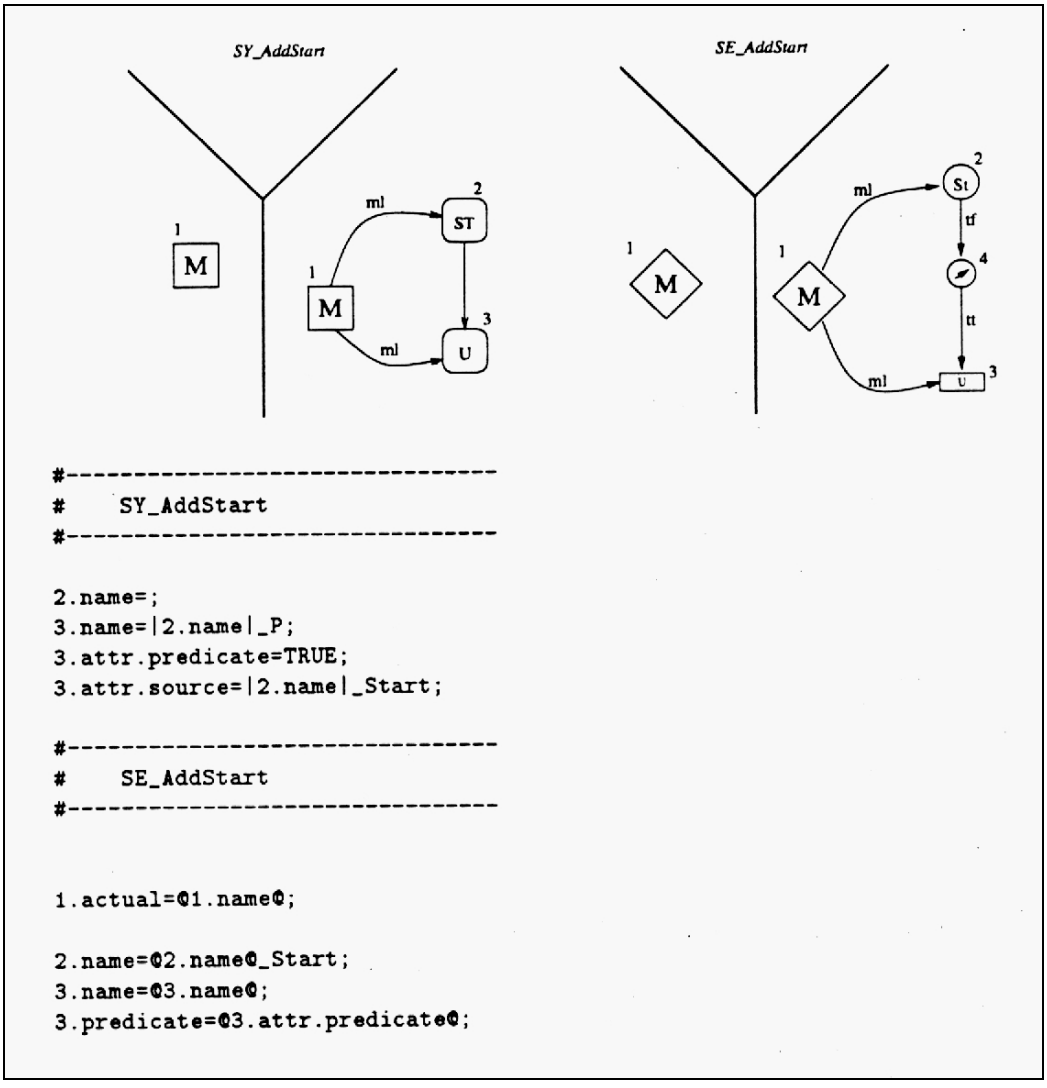


FIGURA 6.14 – Aplicação da produção no lugar P2

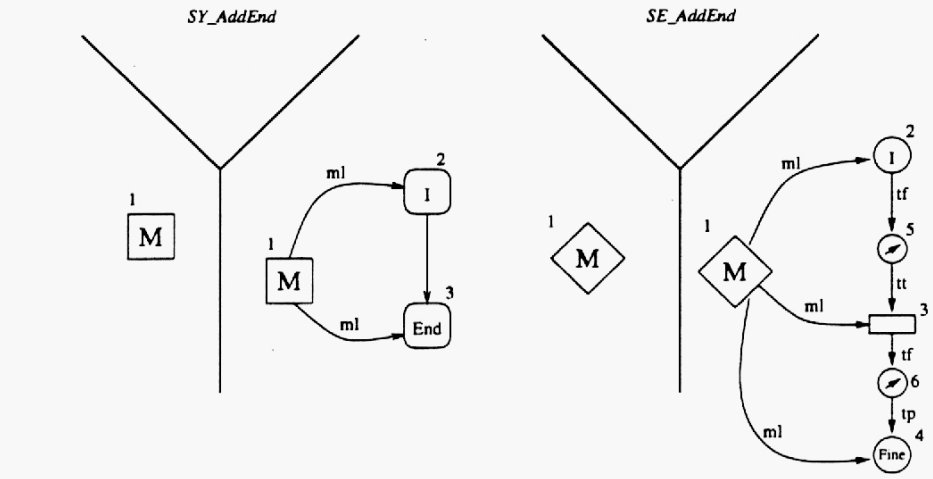
Ingresso no Processo




```
#--------------------------------
#    SY_AddStart
#--------------------------------

2.name=;
3.name=|2.name|_P;
3.attr.predicate=TRUE;
3.attr.source=|2.name|_Start;

#--------------------------------
#    SE_AddStart
#--------------------------------


1.actual=@1.name@;

2.name=@2.name@_Start;
3.name=@3.name@;
3.predicate=@3.attr.predicate@;
```

FIGURA 6.15 – Regras para Ingresso

Saída do Processo




```
#--------------------------------
#    SY_AddEnd
#--------------------------------

2.name=;
3.name=|2.name|_Fine;

#--------------------------------
#    SE_AddEnd
#--------------------------------

1.actual=@1.name@;

2.name=@2.name@_In;
3.name=@2.name@_transOut;
4.name=@3.name@;
3.action={@3.name@.value=|2.name|.value;};
```

FIGURA 6.16 – Regras para Saída

Conexão

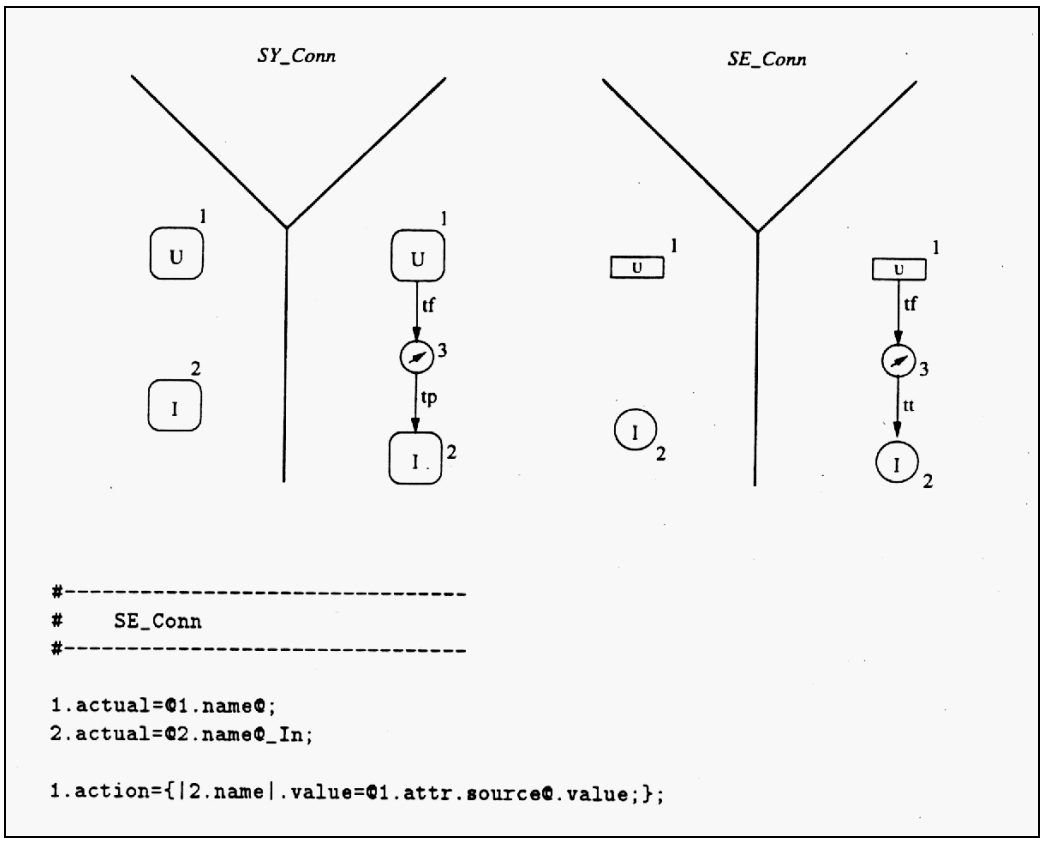
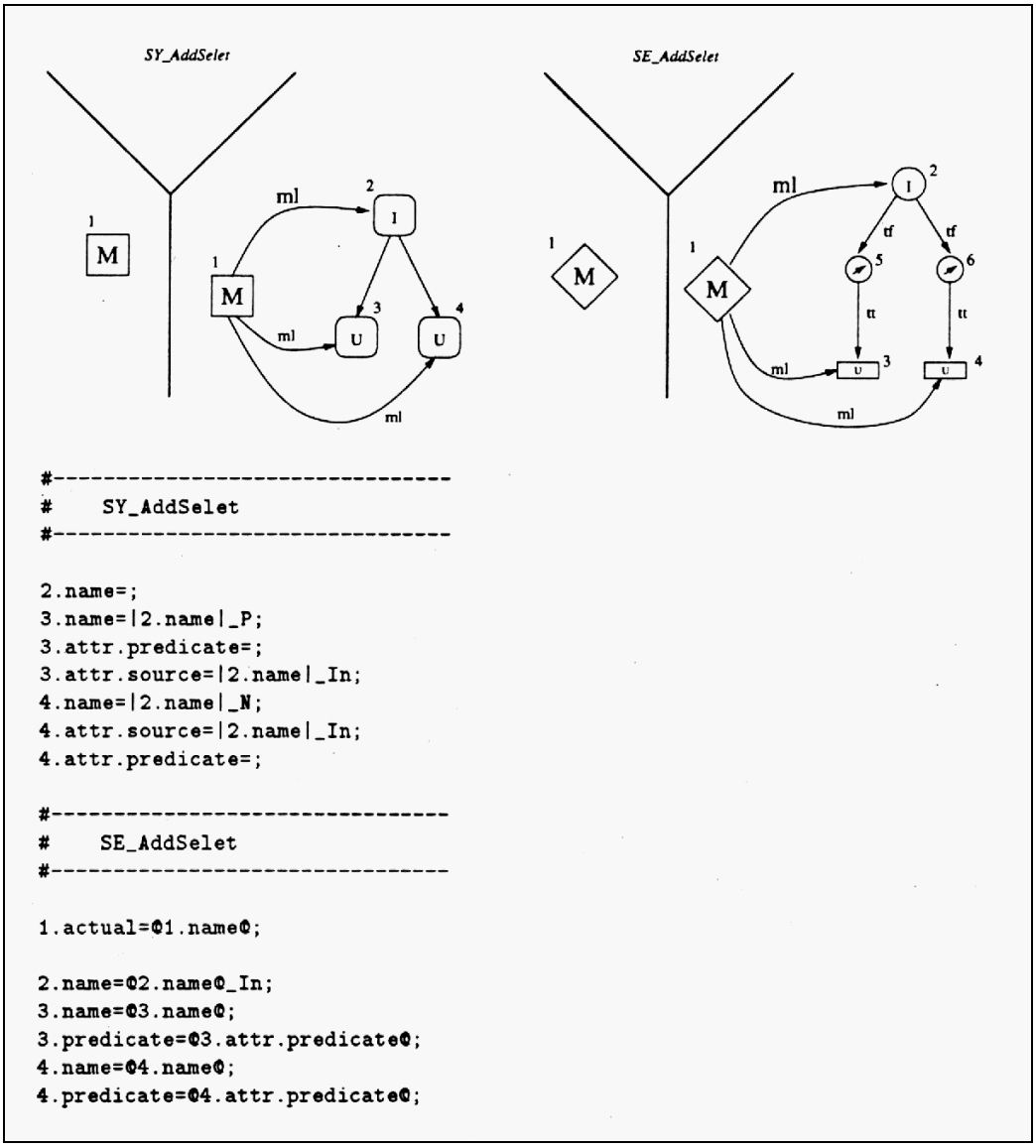


FIGURA 6.17 – Regras para conexões

Seletor




```
#--------------------------------
#    SY_AddSelet
#--------------------------------

2.name=;
3.name=|2.name|_P;
3.attr.predicate=;
3.attr.source=|2.name|_In;
4.name=|2.name|_N;
4.attr.source=|2.name|_In;
4.attr.predicate=;

#--------------------------------
#    SE_AddSelet
#--------------------------------

1.actual=@1.name@;

2.name=@2.name@_In;
3.name=@3.name@;
3.predicate=@3.attr.predicate@;
4.name=@4.name@;
4.predicate=@4.attr.predicate@;
```

FIGURA 6.18 – Regras para Seletor

Repetidor

```
#--------------------------------
#    SE_AddIter
#--------------------------------

1.actual=@1.name@;

2.name=@2.name@_1_In;
2.ASGGNode=@5.id@;
3.name=@2.name@_Reset;
3.ASGGNode=@5.id@;
4.name=@2.name@_Ritorno;
4.ASGGNode=@3.id@;

#----- reset contatore
3.action={|4.name|.value=|2.name|.value;
          |4.name|.value.ciclo=@3.attr.numero@;}
5.name=@4.name@;
6.name=@2.name@_In;
7.name=@7.name@;

#----- contatore cicli nullo
7.predicate=|6.name|.value.ciclo==0;
8.name=@6.name@;

#----- contatore ciclo > 0
8.predicate=|6.name|.value.ciclo>0;

#----- decremento contatore
8.action={|4.name|.value=|6.name|.value;
          |4.name|.value.ciclo--;}
```

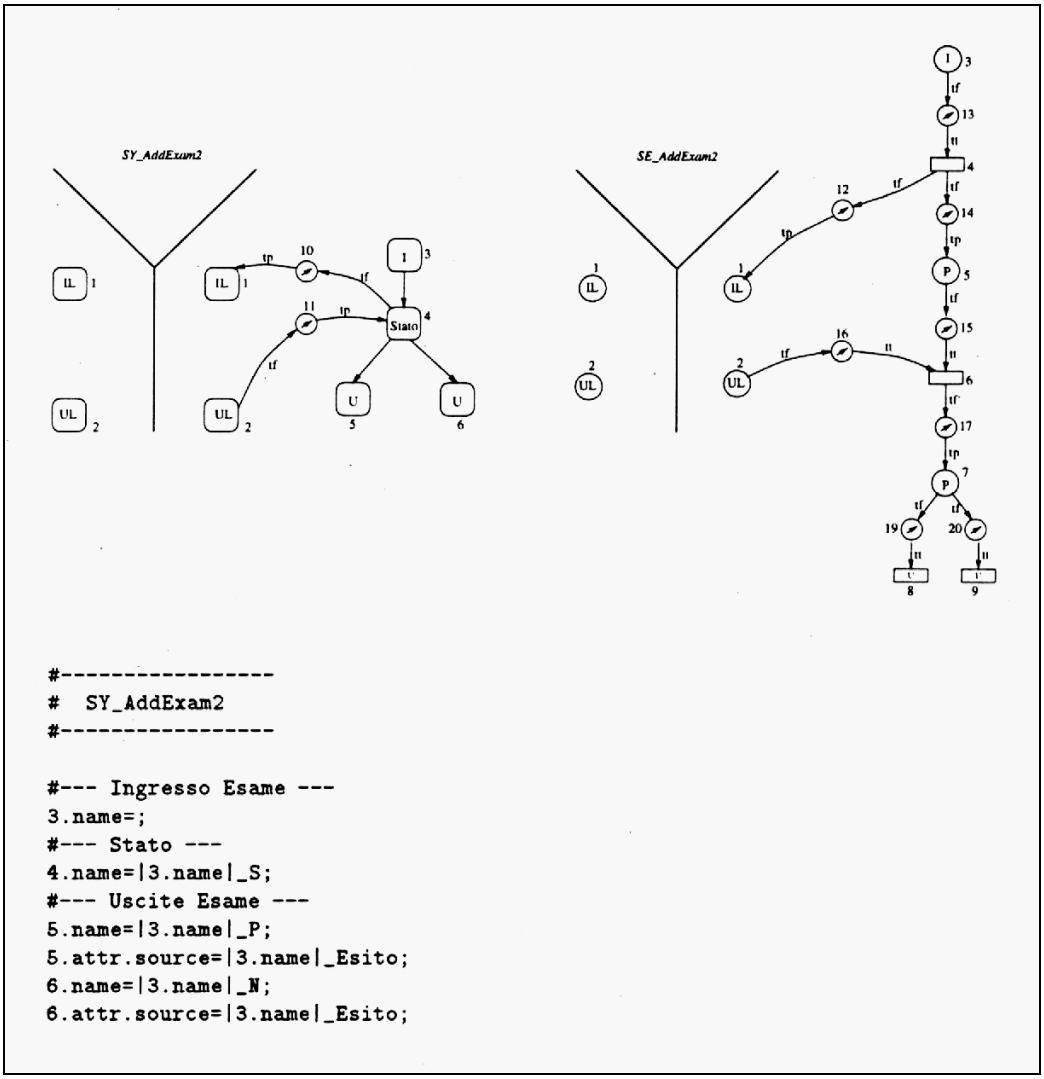
FIGURA 6.20 – Regras para Repetidor

Laboratório

```
#--------------------------------
#    SE_AddLab2
#--------------------------------

1.actual=@1.name@;

#--- ingresso laboratorio ---
2.name=@2.name@_IL;
2.ASGGNode=@3.id@;
3.name=@2.name@_verUt;
3.predicate=@2.name@_Utenti.value>0;
3.action={@2.name@_StExam.value=@2.name@_IL.value;
          @2.name@_Utenti.value--;};

#--- esecuzione esame ---
4.name=@2.name@_StExam;
5.name=@2.name@_pos;
5.action={@2.name@_EndExam.value=@2.name@_StExam.value;
          @2.name@_EndExam.value.esito=POSITIVO;};
6.name=@2.name@_neg;
6.action={@2.name@_EndExam.value=@2.name@_StExam.value;
          @2.name@_EndExam.value.esito=NEGATIVO;};
7.name=@2.name@_EndExam;
8.name=@2.name@_Libera;
8.action={@2.name@_UL.value=@2.name@_EndExam.value;
          @2.name@_Utenti.value++;};

#--- uscita laboratorio ---
9.name=@2.name@_UL;
9.ASGGNode=@4.id@;
10.name=@2.name@_Utenti;
10.createtoken=;
10.token=@2.attr.Nutenti@;
```

FIGURA 6.21 – Regras para Laboratório

Exame de duas saídas




```
#-----------------
#  SY_AddExam2
#-----------------

#--- Ingresso Esame ---
3.name=;
#--- Stato ---
4.name=|3.name|_S;
#--- Uscite Esame ---
5.name=|3.name|_P;
5.attr.source=|3.name|_Esito;
6.name=|3.name|_N;
6.attr.source=|3.name|_Esito;
```

FIGURA 6.22 – Regras para Exames

```
#------------------------------
#    SE_AddExam2
#------------------------------

#--- individua laboratorio ---
1.actual=@1.name@;
2.actual=@2.name@;

#--- ingresso esame ---
3.name=@3.name@_In;

#--- corpo esame ---
4.name=@3.name@_GoLab;
4.action={@3.name@_esec.value=@3.name@_In.value;
          @1.name@.value.cod=@3.name@_In.value.cod;
          @1.name@.value.esito=0;};
5.name=@3.name@_esec;
5.ASGGNode=@4.id@;
6.name=@3.name@_RetLab;
6.ASGGNode=@4.id@;
6.predicate=@2.name@.value.cod==@3.name@_esec.value.cod;
6.action={@3.name@_Esito.value=@3.name@_esec.value;
          @3.name@_Esito.value.esito=@2.name@.value.esito;};
7.name=@3.name@_Esito;
7.ASGGNode=@4.id@;

#--- uscite esame ---
8.name=@5.name@;
8.predicate=@3.name@_Esito.value.esito==POSITIVO;
9.name=@6.name@;
9.predicate=@3.name@_Esito.value.esito==NEGATIVO;
```

FIGURA 6.23 – Regras para Exames

7 Estudo de Caso

Neste capítulo abordaremos ás representações dos protocolos de diagnóstico em Neurologia e as dificuldades encontradas na sua representação com a Metodologia LEMMA bem como a aplicação de LEMMA 2000. A primeira etapa são mostrados os modelos elaborados com a primeira versão, que deu embasamento para a construção da versão final. A segunda etapa mostra a aplicação de LEMMA 2000.

7.1 Descrição do estudo de caso

Os "Modelos Neurológicos" foram extraídos de [WSG 93]. No futuro também serão representados outros modelos de diagnóstico de áreas de saúde distintas, afim de avaliarmos a necessidade da criação e adaptação das notações.

Para facilitar a representação dos protocolos de diagnóstico devido a extensão da nomeclatura utilizada e a complexidade dos diagramas utilizamos algumas abreviaturas que são listadas abaixo.

Lista de Abreviaturas

PwNS Pacient with Neurological Sympstoms

NS Neurologic Symptoms

FND Focal Neurological Disturbances

SC State of consciusness

EF Episode of falling

RS Resolution of Sysptoms

\+ Normal

\- Abnormal

ANS Associated neurologic Symptoms

SS Sleep Study

TIA Transient Ischemic Attack

NEF neurologic examination findings

DS duration of symptoms

TC temporal course

PID Progression of initial deficit

TPR Temporal pattern or recurrence

CNW Continued neurologic worsening

RwNS Reassess with neurodiagnostic studies

SI Spontaneous improvement

7.2 Diagramas

Análise de decisão clínica é um processo complicado, complexo, e sistemático; porém, freqüentemente é executado quase automaticamente por clínicos experientes. Antes que o julgamento clínico seja reduzido a um processo intuitivo automático, este deve ser executado muitas vezes sistematicamente e analiticamente. Para alcançar esta meta, é possível estruturar análise de decisão em regras lógicas e seqüentes. Estes são definidos como árvores de decisão ou algoritmos clínicos. O princípio básico de algoritmos é aquele inerente no processo diagnóstico e está representada a análise de pensamento lógico. Pelo uso de algoritmos, pode ser transformada a "arte de neurologia" em um processo analítico, seqüente, racional. Em [WSG 93] os protocolos de diagnóstico em neurologia estão representados com árvores decisão que são mostradas a seguir:

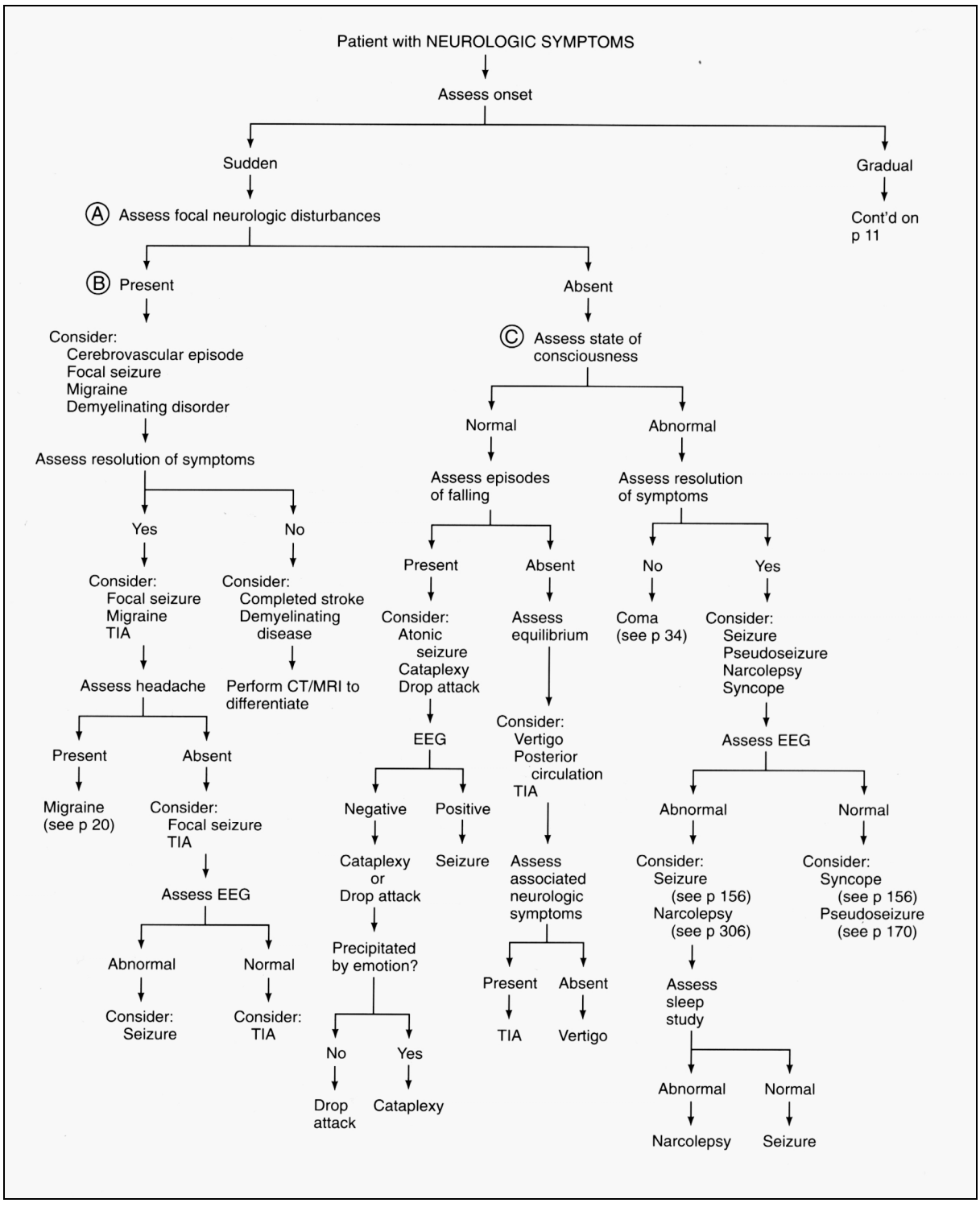


FIGURA 7.1 – Árvore de Decisão "*Neurologic Symptoms*" [WSG 93]

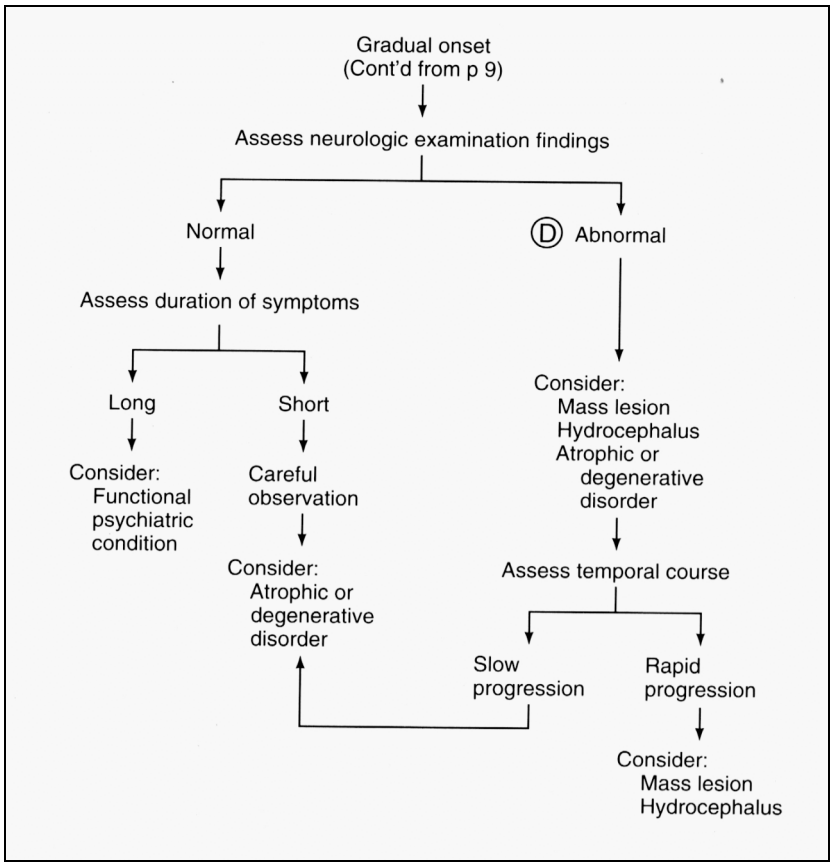


FIGURA 7.2 - Árvore de Decisão "*Gradual Onset*" [WSG 93]

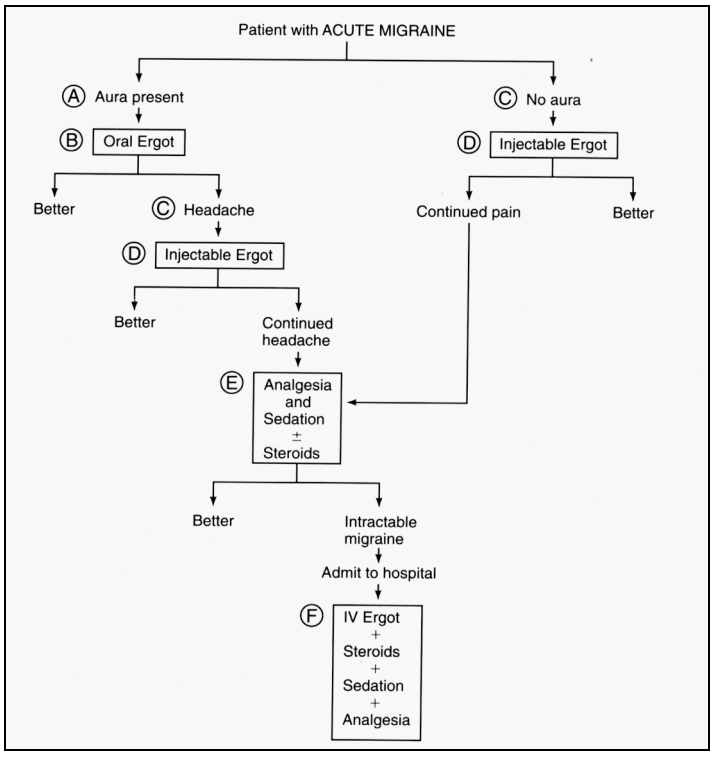


FIGURA 7.3 - Árvore de Decisão "*Acute Migraine*" [WSG 93]

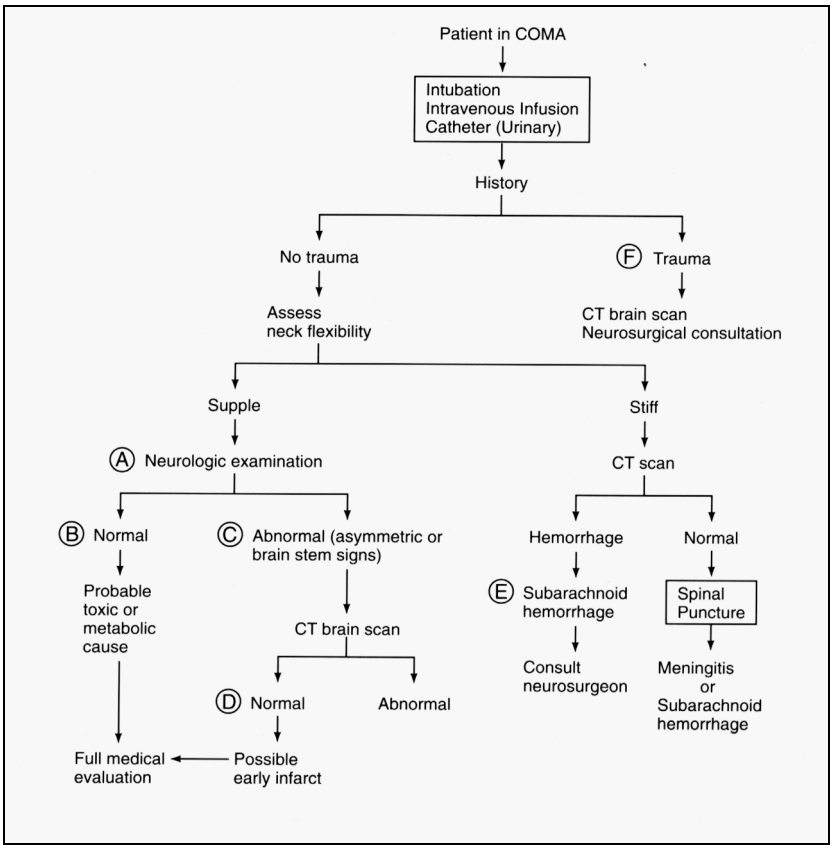


FIGURA 7.4 - Árvore de Decisão "*COMA*" [WSG 93]

7.3 Utilização da Notação LEMMA 2000

Nesta aplicação utilizamos os três diagramas mostrados anteriormente e devido a grande capacidade de representação que LEMMA 2000 mostrou nos testes feitos, foi possível representa-los somente com um único diagrama. A seguir mostramos este diagrama bem como a "explosão" dos elementos hierárquicos.

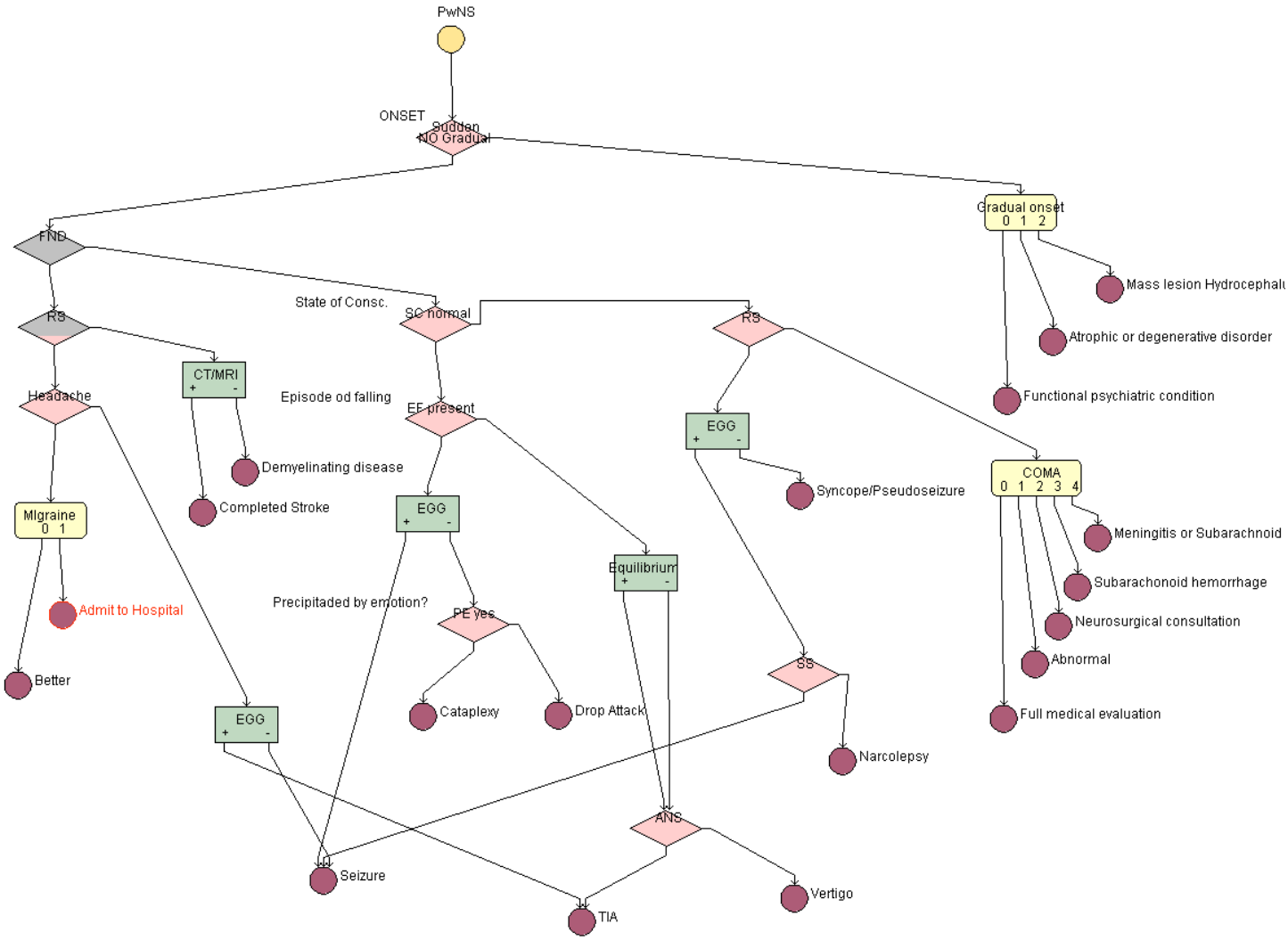


FIGURA 7.5 – Gráfico "Sintomas Neurológicos"

7.4 Análise do Modelo

O modelo é:

LEMMA-válido: são presentes os elementos de ingresso e saída do processo e todos os elementos estão ligados corretamente;

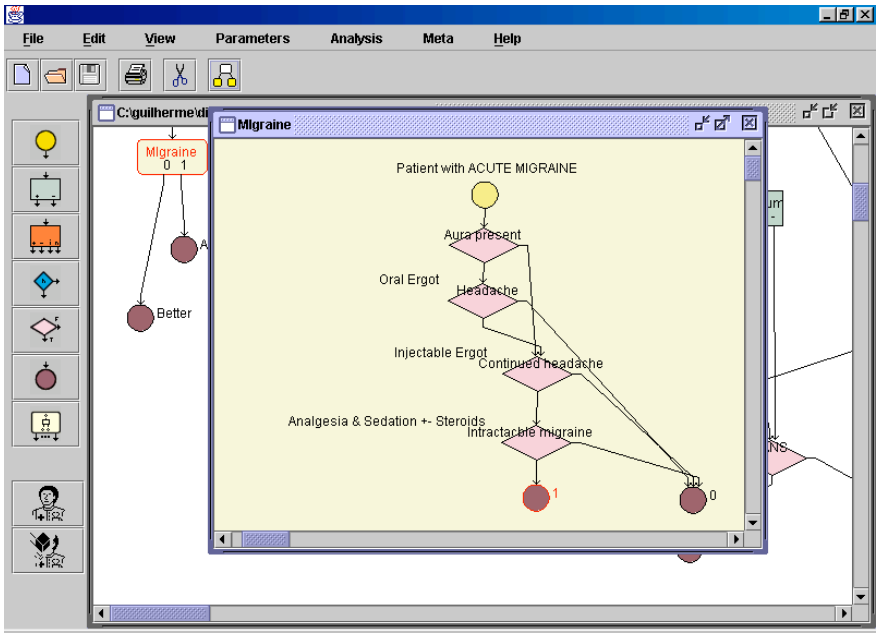


FIGURA7.6 – Subgráfico - Migraine

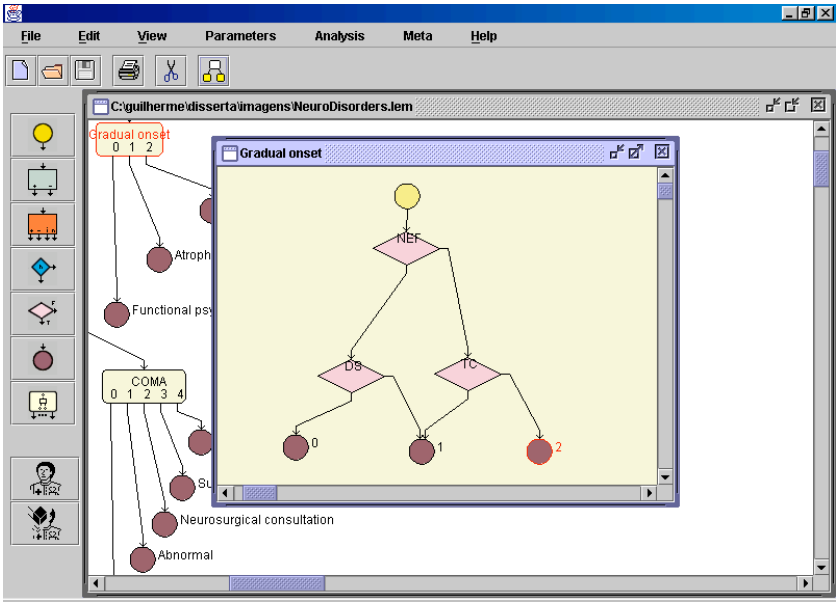


FIGURA 7.7 – Subgráfico “Gradual Onset”

LEMMA-correto: não existem ciclos nem caminhos não percorridos, esta análise foi efetuada através da rede de petri equivalente submetida a testes com Cabernet[PES 94].

7.5 Rede de Petri equivalente

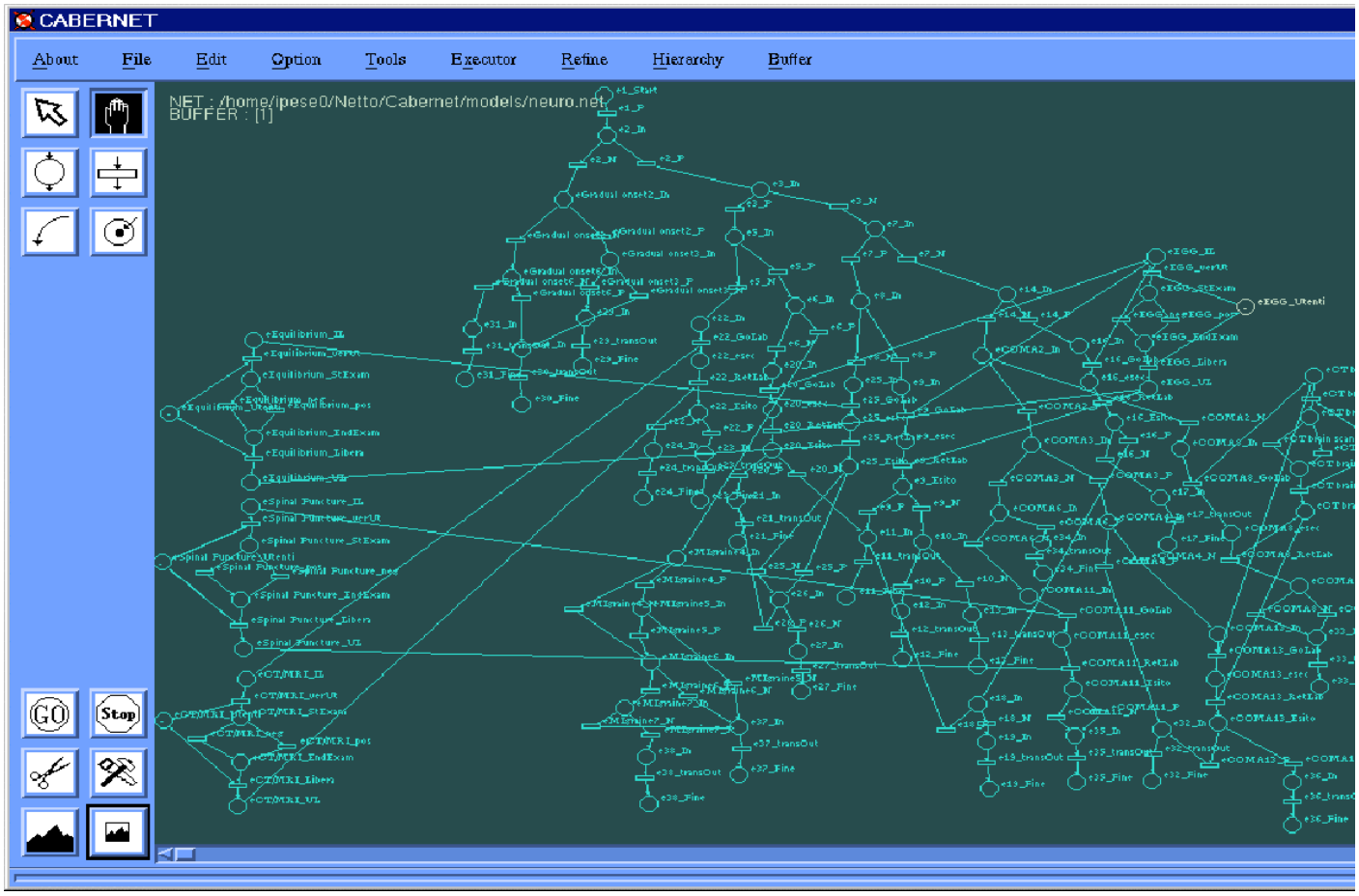


FIGURA 7.9 – Rede de Petri Equivalente

Na figura 7.9 é mostrada a rede de petri correspondente ao modelo mostrado na figura 7.5. Devido a dificuldade de compreensão do resultado obtido, podemos notar diferença de expressividade e legibilidade entre estas duas notações.

8 Conclusão

Este trabalho teve como tema principal a especificação formal e análise de processos clínicos. Em particular foram estudadas a utilização de notações gráficas e formais, concentrando sempre nestas duas características indispensáveis para alcançar os objetivos de:

- Elevada legibilidade do modelo;
- Automação da análise e da verificação das propriedades.

Como foi visto em todos os capítulos, o trabalho desenvolveu-se em torno da aplicação de modelos formais para uma melhor definição dos processos clínicos e a expectativa é de uma contribuição para a melhoria da qualidade de cura dos pacientes.

No decorrer do desenvolvimento do trabalho foram introduzidos os seguintes tópicos:

- Especificação de processos clínicos mediante a utilização de uma notação formal;
- Especificação de LEMMA 2000, notação simplificada para modelagem de processos clínicos. Esta acumula as propriedades de ser formal graças a sua correspondência semântica com as redes de Petri;

Ao estudo dos aspectos teóricos está ligada a implementação de uma ferramenta que se encarrega-se de gerenciar a notação LEMMA 2000 e estabelecer a correspondência com as rede de Petri. Logo foram implementados:

- Um editor gráfico para a especificação e gestão dos processos clínicos modelados com LEMMA 2000 realizado com a linguagem JAVA.
- E o módulo responsável pelo funcionamento do metaeditor, realizado mediante uma técnica de programação específica.

O instrumento baseia-se na construção de redes de Petri a partir da especificação de um processo mediante a notação LEMMA 2000. Onde é possível efetuar verificações de propriedades, análises e animações baseadas na execução das correspondentes redes de Petri.

Como contribuições este trabalho mostrou uma nova versão para a notação Lemma, que veio resolver vários problemas, entre eles ícones pouco elaborados, elementos não configuráveis e um editor mais elaborado.

Os modelos gerados para os protocolos de diagnóstico em neurologia tornaram-se mais simples e mais legíveis comparados com os testes efetuados com a primeira versão. Isto foi possível devida a utilização do elemento hierárquico, que agregou um elevado grau de expressividade e simplicidade a notação.

Em uma análise para trabalhos futuros, poderá ser pensada a possibilidade de introdução do conceito de cálculos de custo nos processos, bem como análise de alocação de recursos para otimização da utilização destes nos hospitais visando uma redução do tempo de permanência do cliente e redução de custos.

Um outro tópico a ser abordado é a construção de uma ferramenta que possibilite a criação de grupos de trabalho virtuais, em que profissionais poderiam utilizar a notação Lemma 2000 para representar os protocolos clínicos de forma mais dinâmica, compartilhando suas experiências, ou seja, criando um ambiente de ensino cooperativo.

Anexo 1 Código de Interface entre Lemma 2000 e Cabernet

```
formalism@LEMMA
new@gtn
generic@start@gtn@1
generic@selet@gtn@2@"Sudden"&&!("Gradual")@!("Sudden"&&!("Gradual"))
generic@selet@gtn@3@"FND"@!("FND")
generic@selet@gtn@Gradual onset2@NEF@!(NEF)
generic@selet@gtn@Gradual onset3@DS@!(DS)
generic@selet@gtn@Gradual onset6@TC@!(TC)
generic@selet@gtn@5@"RS"@!("RS")
generic@selet@gtn@6@"Headache"@!("Headache")
generic@selet@gtn@7@"SC normal"@!("SC normal")
generic@selet@gtn@8@"EF present"@!("EF present")
generic@lab2@gtn@EGG@1
generic@exam2@EGG_IL@EGG_UL@9
generic@selet@gtn@10@"PE yes"@!("PE yes")
generic@end@gtn@11
generic@end@gtn@12
generic@end@gtn@13
generic@selet@gtn@14@"RS"@!("RS")
generic@selet@gtn@COMA2@HIstory@!(HIstory)
generic@selet@gtn@COMA3@Neck Flex.@!(Neck Flex.)
generic@selet@gtn@COMA4@NE normal@!(NE normal)
generic@selet@gtn@COMA6@CT scan- HEMO.@!(CT scan- HEMO.)
generic@lab2@gtn@CT brain scan@1
generic@exam2@CT brain scan_IL@CT brain scan_UL@COMA8
generic@lab2@gtn@Spinal Puncture@1
generic@exam2@Spinal Puncture_IL@Spinal Puncture_UL@COMA11
generic@exam2@CT brain scan_IL@CT brain scan_UL@COMA13
generic@exam2@EGG_IL@EGG_UL@16
generic@end@gtn@17
generic@selet@gtn@18@"SS"@!("SS")
generic@end@gtn@19
generic@exam2@EGG_IL@EGG_UL@20
generic@end@gtn@21
generic@lab2@gtn@CT/MRI@1
generic@exam2@CT/MRI_IL@CT/MRI_UL@22
generic@end@gtn@23
generic@end@gtn@24
generic@lab2@gtn@Equilibrium@1
generic@exam2@Equilibrium_IL@Equilibrium_UL@25
generic@selet@gtn@26@"ANS"@!("ANS")
generic@end@gtn@27
generic@selet@gtn@MIgraine4@Aura present@!(Aura present)
generic@selet@gtn@MIgraine5@Headache@!(Headache)
generic@selet@gtn@MIgraine6@Continued headache@!(Continued headache)
generic@selet@gtn@MIgraine7@Intractacble migraine@!(Intractacble migraine)
generic@end@gtn@29
generic@end@gtn@30
generic@end@gtn@31
generic@end@gtn@32
generic@end@gtn@33
generic@end@gtn@34
generic@end@gtn@35
generic@end@gtn@36
generic@end@gtn@37
generic@end@gtn@38
generic@conn@1_P@2
generic@conn@9_P@11
generic@conn@9_N@10
generic@conn@16_P@18
generic@conn@16_N@17
generic@conn@20_P@21
generic@conn@20_N@11
generic@conn@22_P@23
generic@conn@22_N@24
generic@conn@25_P@26
generic@conn@25_N@26
generic@conn@2_P@3
```

```
generic@conn@2_N@Gradual onset2
generic@conn@3_P@5
generic@conn@3_N@7
generic@conn@5_P@6
generic@conn@5_N@22
generic@conn@6_P@MIgraine4
generic@conn@6_N@20
generic@conn@7_P@8
generic@conn@7_N@14
generic@conn@8_P@9
generic@conn@8_N@25
generic@conn@10_P@12
generic@conn@10_N@13
generic@conn@14_P@16
generic@conn@14_N@COMA2
generic@conn@18_P@11
generic@conn@18_N@19
generic@conn@26_P@21
generic@conn@26_N@27
generic@conn@Gradual onset2_P@Gradual onset3
generic@conn@Gradual onset2_N@Gradual onset6
generic@conn@Gradual onset3_P@29
generic@conn@Gradual onset3_N@30
generic@conn@Gradual onset6_P@30
generic@conn@Gradual onset6_N@31
generic@conn@COMA8_P@33
generic@conn@COMA8_N@33
generic@conn@COMA11_P@35
generic@conn@COMA11_N@35
generic@conn@COMA13_P@32
generic@conn@COMA13_N@36
generic@conn@COMA2_P@COMA3
generic@conn@COMA2_N@COMA8
generic@conn@COMA3_P@COMA4
generic@conn@COMA3_N@COMA6
generic@conn@COMA4_P@32
generic@conn@COMA4_N@COMA13
generic@conn@COMA6_P@34
generic@conn@COMA6_N@COMA11
generic@conn@MIgraine4_P@MIgraine5
generic@conn@MIgraine4_N@MIgraine6
generic@conn@MIgraine5_P@MIgraine6
generic@conn@MIgraine5_N@37
generic@conn@MIgraine6_P@MIgraine7
generic@conn@MIgraine6_N@37
generic@conn@MIgraine7_P@38
generic@conn@MIgraine7_N@37
save@gtn
compile@gtn
quit
```

Anexo 2 Código Java

```
package gui;
import javax.swing.*;
import javax.swing.tree.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import java.io.*;


class viewNetPanel extends JPanel{
  private Vector Lab2vet = new Vector(10,3);
  final JTextArea textArea = new JTextArea();
  private Vector vetb = new Vector(10,3);

  public viewNetPanel(Vector vet, Vector vetSynt){
    LemmaClass in = new LemmaClass(0,0);
    EsameClass es = new EsameClass(0,0);
    Esame4Class es4 = new Esame4Class(0,0);
    SeletClass s = new SeletClass(0,0);
    ExitClass ex = new ExitClass(0,0);
    IterClass it = new IterClass(0,0);
    GenericClass ge = new GenericClass(0,0);

    Object ob = new Object();
    Object ob2 = new Object();
    int vetsize = vet.size();
    int cont = 1;
    vetb = vet;
    this.setLayout(new BorderLayout());
    textArea.setEditable(false);
    JScrollPane scroller2 = new JScrollPane(textArea);
    scroller2.setPreferredSize(new Dimension(200,600));
    add(scroller2,BorderLayout.CENTER);
    textArea.append("formalism@LEMMA" + "\n");
    textArea.append("new@gtn" + "\n");
    int ccx = (int)'"';
    char aspas = (char)ccx;
    for(int i=0; i < vetsize; i++){
       ob = vet.elementAt(i);
       if (ob.getClass() == in.getClass()){
         in = (LemmaClass)ob;
         textArea.append("generic@start@gtn@" + cont++ + "\n");

       };
       if (ob.getClass() == ex.getClass()){
         ex = (ExitClass)ob;
         textArea.append("generic@end@gtn@" + cont++ + "\n");

       };
       if (ob.getClass() == es.getClass()){
         es = (EsameClass)ob;
         if(!existeEsame2(es.name)){
           textArea.append("generic@lab2@gtn@" + es.name + "@"
+ es.numbUtente +"\n");
           Lab2vet.addElement(es.name);
         };
```

```
                textArea.append("generic@exam2@" + es.name + "_IL@" +
es.name + "_UL@" + cont++ + "\n");
              };
              if (ob.getClass() == it.getClass()){
                it = (IterClass)ob;
                textArea.append("generic@iter@gtn@" + cont++ + "@" +
it.num_iter + "\n");

              };
              if (ob.getClass() == s.getClass()){
                s = (SeletClass)ob;
                SintomClass sint = new SintomClass("");
                String at1="", at2="",op = "";
                for(int ii2=0;ii2 < s.vetsel.size();ii2++){
                  sint = (SintomClass)s.vetsel.elementAt(ii2);
                  if(sint.operator.compareTo("NO") != 0){
                    at1 += op + aspas + sint.name + aspas;
                  }
                  else{
                    at1 += op + "!(" + aspas + sint.name + aspas +
")";
                  };
                  op = "&&";
                };
                at2 = "!(" + at1 + ")";
                textArea.append("generic@selet@gtn@" + cont++ + "@" +
at1 + "@" + at2 +"\n");

              };
              if (ob.getClass() == ge.getClass()){
                ge = (GenericClass)ob;
                cont++;
                //Elementos de generic class
                addGeneric(ge);
              };
            };
            int aux, aux2;
            //Conecta inicios ----------------------------------------
------
            for(int i=0; i < vetsize; i++){
               ob = vet.elementAt(i);
               if (ob.getClass() == in.getClass()){
                 in = (LemmaClass)ob;
                 aux = i+1;
                 aux2 = in.arrow.prox + 1;
                 int iter = testaIter(vet,in.arrow.prox);
                 ob = vet.elementAt(in.arrow.prox);
                 if(iter != 0)
                   textArea.append("generic@conn@" + aux + "_P@" + iter
+ "_1\n");
                 else
                   if(ob.getClass() == ge.getClass()){
                     ge = (GenericClass)ob;
                     String fg = firstGeneric(ge);
                     textArea.append("generic@conn@" + aux + "_P@" +
ge.name + fg + "\n");
                   }
                   else
                     textArea.append("generic@conn@" + aux + "_P@" +
aux2 + "\n");
```

```
            };
         };
         //Conecta Esame2 ----------------------------------------
-----
         for(int i=0; i < vetsize; i++){
            ob = vet.elementAt(i);
            if (ob.getClass() == es.getClass()){
              es = (EsameClass)ob;
              aux = i+1;
              aux2 = es.arrowPos.prox +1;
              int iter = testaIter(vet,es.arrowPos.prox);
              ob = vet.elementAt(es.arrowPos.prox);
              if(iter != 0)
                textArea.append("generic@conn@" + aux + "_P@" + iter
+ "_1\n");
              else
                if(ob.getClass() == ge.getClass()){
                  ge = (GenericClass)ob;
                  String fg = firstGeneric(ge);
                  textArea.append("generic@conn@" + aux + "_P@" +
ge.name + fg + "\n");
                }
                else
                textArea.append("generic@conn@" + aux + "_P@" + aux2
+ "\n");
              //-----------------------------------------------------
-----
              aux2 = es.arrowNeg.prox +1;
              iter = testaIter(vet,es.arrowNeg.prox);
              ob = vet.elementAt(es.arrowNeg.prox);
              if(iter != 0)
                textArea.append("generic@conn@" + aux + "_N@" + iter
+ "_1\n");
              else
                if(ob.getClass() == ge.getClass()){
                  ge = (GenericClass)ob;
                  String fg = firstGeneric(ge);
                  textArea.append("generic@conn@" + aux + "_N@" +
ge.name + fg + "\n");
                }
                else
                  textArea.append("generic@conn@" + aux + "_N@" +
aux2 + "\n");
            };
         };
         //Conecta Seletore ---------------------------------------
-------
         for(int i=0; i < vetsize; i++){
            ob = vet.elementAt(i);
            if (ob.getClass() == s.getClass()){
              s = (SeletClass)ob;
              aux = i+1;
              aux2 = s.arrowTrue.prox + 1;
              int iter = testaIter(vet,s.arrowTrue.prox);
              ob = vet.elementAt(s.arrowTrue.prox);
              if(iter != 0)
                textArea.append("generic@conn@" + aux + "_P@" + iter
+ "_1\n");
              else
                if(ob.getClass() == ge.getClass()){
```

```
                    ge = (GenericClass)ob;
                    String fg = firstGeneric(ge);
                    textArea.append("generic@conn@" + aux + "_P@" +
ge.name + fg + "\n");
                  }
                  else
                    textArea.append("generic@conn@" + aux + "_P@" +
aux2 + "\n");

                //-----------------------------------------
                aux2 = s.arrowFalse.prox +1;
                iter = testaIter(vet,s.arrowFalse.prox);
                ob = vet.elementAt(s.arrowFalse.prox);
                if(iter != 0)
                  textArea.append("generic@conn@" + aux + "_N@" + iter
+ "_1\n");
                else
                  if(ob.getClass() == ge.getClass()){
                    ge = (GenericClass)ob;
                    String fg = firstGeneric(ge);
                    textArea.append("generic@conn@" + aux + "_N@" +
ge.name + fg + "\n");
                  }
                  else
                    textArea.append("generic@conn@" + aux + "_N@" +
aux2 + "\n");
              };
            };
            //Conecta Iteratore ----------------------------------------
--------
            for(int i=0; i < vetsize; i++){
               ob = vet.elementAt(i);
               if (ob.getClass() == it.getClass()){
                 it = (IterClass)ob;
                 aux = i+1;
                 aux2 = it.arrowReturn.prox +1;
                 textArea.append("generic@conn@" + aux + "_1_P@" + aux2
+ "\n");
                 aux2 = it.arrowAvanti.prox +1;
                 textArea.append("generic@conn@" + aux + "_P@" + aux2 +
"\n");
               };
            };
            //Conecta Generic ------------------------------------------
------
            for(int i=0; i < vetsize; i++){
               ob = vet.elementAt(i);
               if (ob.getClass() == ge.getClass()){
                 ge = (GenericClass)ob;
                 conectGeneric(ge);
               };
            };
            textArea.append("save@gtn" + "\n");
            textArea.append("compile@gtn" + "\n");
            textArea.append("quit" + "\n");


            //save mes_list --------------------------------------------
------
            try{
```

```
    FileOutputStream fi = new FileOutputStream("mes_list");
    PrintStream ou = new PrintStream(fi);
    String lin = textArea.getText();
    int cc = 0;
    for(int i=0;i< lin.length();i++){
      if(lin.substring(i,i+1).compareTo("\n") == 0){
        ou.println(lin.substring(cc,i));
        cc=i+1;
      };
    };
    ou.close();
  }catch(IOException e){
    JOptionPane.showMessageDialog(null, "Error");
  }
}
private String daNome(Object o){
  String ss=null;
  LemmaClass in = new LemmaClass(0,0);
  EsameClass es = new EsameClass(0,0);
  Esame4Class es4 = new Esame4Class(0,0);
  SeletClass s = new SeletClass(0,0);
  ExitClass ex = new ExitClass(0,0);
  IterClass it = new IterClass(0,0);
  if (o.getClass() == in.getClass()){
    in = (LemmaClass)o;
    ss = in.name;
  };
  if (o.getClass() == ex.getClass()){
    ex = (ExitClass)o;
    ss = ex.name;
  };
  if (o.getClass() == es.getClass()){
    es = (EsameClass)o;
    ss = es.name;
  };
  return ss;
}
private int testaIter(Vector vvet,int prox){
  IterClass it = new IterClass(0,0);
  Object o = new Object();
  int a = 0;

  for(int i=0; i < vvet.size(); i++){
     o = vvet.elementAt(i);
     if (o.getClass() == it.getClass()){
       it = (IterClass)o;
       if(it.arrowReturn.prox == prox)
          a = i+1;
     };
  };
  return a;
}
private String firstGeneric(GenericClass ge){
  Object o = new Object();
  LemmaClass in = new LemmaClass(0,0);
  String a = "";

  for(int ix=0; ix < ge.vetElement.size(); ix++){
    o = ge.vetElement.elementAt(ix);
    if (o.getClass() == in.getClass()){
```

```
                in = (LemmaClass)o;
                int ite = testaIter(ge.vetElement,in.arrow.prox);
                if(ite == 0)
                  a = Integer.toString(in.arrow.prox + 1);
                else
                  a = Integer.toString(ite) + "_1";
              };
            };
            return a;
          }
          //-Metodo conecta elementos de generic-----------
          private void conectGeneric(GenericClass ge){
            EsameClass es = new EsameClass(0,0);
            Esame4Class es4 = new Esame4Class(0,0);
            SeletClass s = new SeletClass(0,0);
            IterClass it = new IterClass(0,0);
            ArrowClass arr = new ArrowClass(0,0,0,0);
            ExitClass ex = new ExitClass(0,0);
            Object ob = new Object();
            Object obb = new Object();
            GenericClass ge2 = new GenericClass(0,0);
            int aux, aux2, aux3;
            String g = ge.name;
            //Conecta  Esame2 ----------------------------------------
-----
            for(int i=0; i < ge.vetElement.size(); i++){
               ob = ge.vetElement.elementAt(i);
               if (ob.getClass() == es.getClass()){
                 es = (EsameClass)ob;
                 aux = i+1;
                 aux2 = es.arrowPos.prox +1;
                 int iter = testaIter(ge.vetElement,es.arrowPos.prox);
                 ob = ge.vetElement.elementAt(es.arrowPos.prox);
                 if(iter != 0)
                   textArea.append("generic@conn@" + g + aux + "_P@" +
g + iter + "_1\n");
                 else
                   if(ob.getClass() == ex.getClass()){
                     ex = (ExitClass)ob;
                     for(int ix=0;ix < ge.vetArrow.size();ix++){
                       arr = (ArrowClass)ge.vetArrow.elementAt(ix);
                       if(arr.nameG == Integer.parseInt(ex.name)){
                         obb = vetb.elementAt(arr.prox);
                         if(obb.getClass() != ge.getClass()){
                           aux3 = arr.prox + 1;
                           textArea.append("generic@conn@" + g + aux +
"_P@" + aux3 + "\n");
                         }
                         else{
                           ge2 = (GenericClass)obb;
                           String fg = firstGeneric(ge2);
                           textArea.append("generic@conn@" + g + aux +
"_P@" + ge2.name + fg + "\n");
                         };
                       }
                     }
                   }
                   else
                   textArea.append("generic@conn@" + g + aux + "_P@" +
g + aux2 + "\n");
```

```
                //-------------------------------------------------
-----
                aux2 = es.arrowNeg.prox +1;
                iter = testaIter(ge.vetElement,es.arrowNeg.prox);
                ob = ge.vetElement.elementAt(es.arrowNeg.prox);
                if(iter != 0)
                  textArea.append("generic@conn@" + g + aux + "_N@" +
g + iter + "_1\n");
                else
                  if(ob.getClass() == ex.getClass()){
                    ex = (ExitClass)ob;
                    for(int ix=0;ix < ge.vetArrow.size();ix++){
                      arr = (ArrowClass)ge.vetArrow.elementAt(ix);
                      if(arr.nameG == Integer.parseInt(ex.name)){
                        obb = vetb.elementAt(arr.prox);
                        if(obb.getClass() != ge.getClass()){
                          aux3 = arr.prox + 1;
                          textArea.append("generic@conn@" + g + aux +
"_N@" + aux3 + "\n");
                        }
                        else{
                          ge2 = (GenericClass)obb;
                          String fg = firstGeneric(ge2);
                          textArea.append("generic@conn@" + g + aux +
"_N@" + ge2.name + fg + "\n");
                        };
                      }
                    }
                  }
                  else
                    textArea.append("generic@conn@" + g + aux + "_N@"
+ g + aux2 + "\n");
                };
            };
            //Conecta Seletore ----------------------------------------
-------
            for(int i=0; i < ge.vetElement.size(); i++){
               ob = ge.vetElement.elementAt(i);
               if (ob.getClass() == s.getClass()){
                 s = (SeletClass)ob;
                 aux = i+1;
                 aux2 = s.arrowTrue.prox + 1;
                 int iter = testaIter(ge.vetElement,s.arrowTrue.prox);
                 ob = ge.vetElement.elementAt(s.arrowTrue.prox);
                 if(iter != 0)
                   textArea.append("generic@conn@" + g + aux + "_P@" +
g + iter + "_1\n");
                 else
                   if(ob.getClass() == ex.getClass()){
                     ex = (ExitClass)ob;
                     for(int ix=0;ix < ge.vetArrow.size();ix++){
                       arr = (ArrowClass)ge.vetArrow.elementAt(ix);
                       if(arr.nameG == Integer.parseInt(ex.name)){
                         obb = vetb.elementAt(arr.prox);
                         if(obb.getClass() != ge.getClass()){
                           aux3 = arr.prox + 1;
                           textArea.append("generic@conn@" + g + aux +
"_P@" + aux3 + "\n");
                         }
                         else{
```

```
                        ge2 = (GenericClass)obb;
                        String fg = firstGeneric(ge2);
                        textArea.append("generic@conn@" + g + aux +
"_P@" + ge2.name + fg + "\n");
                      };
                    }
                  }
                }
                else
                  textArea.append("generic@conn@" + g + aux + "_P@"
+ g + aux2 + "\n");

              //-----------------------------------------
              aux2 = s.arrowFalse.prox +1;
              iter = testaIter(ge.vetElement,s.arrowFalse.prox);
              ob = ge.vetElement.elementAt(s.arrowFalse.prox);
              if(iter != 0)
                textArea.append("generic@conn@" + aux + "_N@" + iter
+ "_1\n");
              else
                if(ob.getClass() == ex.getClass()){
                ex = (ExitClass)ob;
                  for(int ix=0;ix < ge.vetArrow.size();ix++){
                    arr = (ArrowClass)ge.vetArrow.elementAt(ix);
                    if(arr.nameG == Integer.parseInt(ex.name)){
                      obb = vetb.elementAt(arr.prox);
                      if(obb.getClass() != ge.getClass()){
                        aux3 = arr.prox + 1;
                        textArea.append("generic@conn@" + g + aux +
"_N@" + aux3 + "\n");
                      }
                      else{
                        ge2 = (GenericClass)obb;
                        String fg = firstGeneric(ge2);
                        textArea.append("generic@conn@" + g + aux +
"_N@" + ge2.name + fg + "\n");
                      };
                    }
                  }
                }
                else
                  textArea.append("generic@conn@" + g + aux + "_N@"
+ g + aux2 + "\n");
              };
          };
          //Conecta Iteratore ------------------------------------
--------
          for(int i=0; i < ge.vetElement.size(); i++){
             ob = ge.vetElement.elementAt(i);
             if (ob.getClass() == it.getClass()){
               it = (IterClass)ob;
               aux = i+1;
               aux2 = it.arrowReturn.prox +1;
               textArea.append("generic@conn@" + g + aux + "_1_P@" +
g + aux2 + "\n");
               aux2 = it.arrowAvanti.prox +1;
               ob = ge.vetElement.elementAt(it.arrowAvanti.prox);
               if(ob.getClass() == ex.getClass()){
                   ex = (ExitClass)ob;
                   for(int ix=0;ix < ge.vetArrow.size();ix++){
```

```
                    arr = (ArrowClass)ge.vetArrow.elementAt(ix);
                    if(arr.nameG == Integer.parseInt(ex.name)){
                      obb = vetb.elementAt(arr.prox);
                      if(obb.getClass() != ge.getClass()){
                        aux3 = arr.prox + 1;
                        textArea.append("generic@conn@" + g + aux +
"_P@" + aux3 + "\n");
                      }
                      else{
                        ge2 = (GenericClass)obb;
                        String fg = firstGeneric(ge2);
                        textArea.append("generic@conn@" + g + aux +
"_P@" + ge2.name + fg + "\n");
                      };
                    }
                  }
                }
                else
                  textArea.append("generic@conn@" + g + aux + "_P@" +
g + aux2 + "\n");
              };
            };
          }
          //-Metodo inclui elementos de generic-----------
          private void addGeneric(GenericClass ge){
            EsameClass es = new EsameClass(0,0);
            Esame4Class es4 = new Esame4Class(0,0);
            SeletClass s = new SeletClass(0,0);
            IterClass it = new IterClass(0,0);
            LemmaClass in = new LemmaClass(0,0);
            ExitClass ex = new ExitClass(0,0);
            Object ob = new Object();
            int cont = 1;
          for(int ix=0; ix < ge.vetElement.size(); ix++){
               ob = ge.vetElement.elementAt(ix);
               String g = ge.name;
               if (ob.getClass() == es.getClass()){
                 es = (EsameClass)ob;
                 if(!existeEsame2(es.name)){
                   textArea.append("generic@lab2@gtn@" + es.name + "@"
+ es.numbUtente +"\n");
                   Lab2vet.addElement(es.name);
                 };
                 textArea.append("generic@exam2@" + es.name + "_IL@" +
es.name + "_UL@"+ g + cont++ + "\n");
               };
               if (ob.getClass() == it.getClass()){
                 it = (IterClass)ob;
                 textArea.append("generic@iter@gtn@"+ g + cont++ + "@"
+ it.num_iter + "\n");
               };
               if (ob.getClass() == in.getClass()){
                 cont++;
               };
               if (ob.getClass() == ex.getClass()){
                 cont++;
               };
               if (ob.getClass() == s.getClass()){
                 s = (SeletClass)ob;
                 SintomClass sint = new SintomClass("");
```

```
                String at1="", at2="",op = "";
                for(int ii2=0;ii2 < s.vetsel.size();ii2++){
                  sint = (SintomClass)s.vetsel.elementAt(ii2);
                  if(sint.operator.compareTo("NO") != 0){
                    at1 += op + sint.name;
                  }
                  else{
                    at1 += op + "!(" + sint.name + ")";
                  };
                  op = "&&";
                };
                at2 = "!(" + at1 + ")";
                textArea.append("generic@selet@gtn@"+ g + cont++ + "@"
+ at1 + "@" + at2 +"\n");
              };
          };

        }
        //-----------------------------------------------
        private boolean existeEsame2(String s){
          String sc;
          boolean f = false;
          for(int i=0; i<Lab2vet.size();i++){
            sc = (String)Lab2vet.elementAt(i);
            if(sc.compareTo(s) == 0)
              f = true;
          }
          return f;
        }}//fim da classe
```

Bibliografia

[BAR 97a] BARESI, L. et al. **LEMMA:** A language for Easy Medical Models Analysis. [S.l.:s.n],1997.

[BAR 97b] BARESI, L. **Formal Customization of Graphical Notations.** Milano: Dipartimento di Eletronica e Informazione, Politecnico di Milano, 1997. PhD thesis

[BOP 97] BARESI, L.; ORSO A.; PEZZE, M. Introducing Formal Methods in industrial Pactice. In : International Conference on Software Engineering, 20., 1997. **Proceedings…** [S.l.]: IEEE-Computer Society Press, 1997.

[DON 88] DONABEDIAN, A. The Quality of Care:How Can It Be Assessed? **JAMA**, [S.l.], v.260, n. 12, 1988.

[DON 90] DONABEDIAN. A. The Seven Pilars of Quality. **Arch. Pathol. Lab. Med.,**[S.l.], v.114, n.8, 1990.

[GEA 99] GEARY, David M. **Graphic Java 2, Mastering the JFC**. Boston:Sun Microsystems Press, 1999.

[GFM94] GUEZZI, C. et al. **Ingegneria del Software**. [S.l.:s.n.], 1994.

[NET 98] NETTO, G. T. **Formalização de Protocolos de Diagnóstico de Síndromes Neurológicas com LEMMA**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1998.

[HEU 90] HEUSER, C. **Modelagem Conceitual de Sistemas:** Redes de Petri. Campinas: R. Vieira, 1991.

[KNO 98] KNOPLICH, J. **Atualização na Saúde.** Disponível em : <http://www.medonline.com.br/medonline4/knoplish.html>. Acesso em: set. 1999.

[MAC 96] MACIEL, P. **Introdução às Redes de Petri e Aplicações**. Campinas: Instituto de Computação/UNICAMP, 1996. 187p. Trabalho apresentado na 10. Escola de Computação.

[MER 94] MERLI, L. **Uno strumento per la definizione de notazioni diagrammatiche**. Milano: Dipartimento di Eletronica e Informazione , Politecnico di Milano, 1994.

[NIB 99] NÚCLEO DE INFORMÁTICA EM BIOCIÊNCIAS (NIB). **Ensino à Distância**. Disponível em: <http://www.uol.com.br/intramed/nib/ensino.htm>. Acesso em: out. 1999.

[PER 89] PERES, E. M. **Uma proposta para a anotação informal de redes de petri de alto nível:** trabalho individual. Porto Alegre: CPGCC da UFRGS, 1989.

[PER 89] PERES, E.M. **Intergrando aspectos formais e informais em uma linguagem de anotação de redes de petri**: trabalho individual. Porto Alegre: CPGCC da UFRGS, 1989. 43 p.

[PES 94] PEZZÈ, M.; SILVA, S. **Cabernet User Manual**. Milano: Politecnico di Milano, 1994. (Raporto interno 47-94).

[VIG 96] VICCARI, R.M.; GIRAFFA, L.M**.** Intelligent Tutoring Systems: functional approach x agents approach. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, 13., 1996, Curitiba**. Advances in Artificial Intelligence**: proceedings. Berlin: Springer-Verlag, 1996. p. 235.

[VIC 99] VICCARI, R.M. **Questões Pedagógicas e Softwares Educacionais**. Disponível em: <http://osse.inf.ufrgs.br/ensino/documentos/pedag.html>. Acesso em: ago .1999.

[WHO96] WHO (WORLD HEALTH ORGANIZATION). **Diagnostic and Management Guidelines for Mental Disorders in Primary Care:** ICD-10**.** Göttingen: Hogrefe & Huber Publishers, 1996.

[SPO96] SPODICK, Edward F. **The evolution of distance learning**. Disponível em: <http://sqzm14.ust.hk/distance/evolution-distance-learning.htm>. Acesso em: out. 1999.

[WSG93] WEINSBERG, L.; STRUB, R.; GARCIA, C**. Decision Making in Adult Neurology**. 2nd ed.[S.l.]: Mosby Publishers, 1993.