Análise de Sistemas Complexos Baseada na Decomposição de Sistemas de G-Nets

por

Angelo Perkusich

Tese apresentada ao Curso de Doutorado em Engenharia Elétrica da Universidade Federal da Paraíba, como requisito parcial às exigências para a obtenção do título de Doutor em Ciências, no domínio da Engenharia Elétrica

Área de Concentração: Processamento da Informação

Universidade Federal da Paraíba

1994

P447a Perkusich, Ângelo.
Análise de sistemas complexos baseada na decomposição de sistemas de G-Nets / Ângelo Perkusich. - Campina Grande, 1994.
144 f.

Tese (Doutorado em Engenharia Elétrica) - Universidade Federal da Paraíba, Centro de Ciências eTecnologia, 1994.
Referências.
"Orientação : Prof. Dr. Misael Elias de Morais, Prof. Dr. Shi-Kuo Chang".

1. Análise de Sistema - Qualidade. 2. Redes Petri. 3. Tese - Engenharia Elétrica. I. Morais, Misael Elias de. II. Chang, Shi-Kuo. III. Universidade Federal da Paraíba - Campina Grande (PB). IV. Título

CDU 621.3:004.5(043)

ANÁLISE DE SISTEMAS COMPLEXOS BASEADA NA
DECOMPOSIÇÃO DE SISTEMAS G-NETS

ANGELO PERKUSICH

Tese Aprovada em 23.08.1994

MISAEL ELIAS DE MORAIS, Dr.-Ing., UFPB
Orientador

SHI-KUO CHANG, Ph.D., Univ. of Pittsburgh
Co-Orientador

TADAO MURATA, Ph.D., Univ. of Illinois at Chicago
Componente da Banca

JORGE MOREIRA DE SOUZA, Dr.Etat, CpQD/TELEBRÁS/UNICAMP
Componente da Banca

DAVID SIMONETTI BARBALHO, Dr., UFRN
Componente da Banca

ULRICH SCHIEL, Dr.ver.nat., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Agosto - 1994

Para Ligia, Mirko and Andrei.

Resumo

Análise de Sistemas Complexos Baseada na Decomposição de Sistemas de G-Nets

por Angelo Perkusich

Quando especificando, concebendo e analizando sistemas complexos, é necessário adotar uma methodologia composicional ou modular. Esta metodologia deve permitir ao projetista a habilidade de verificar propriedades locais de módulos ou componentes individuais de um sistema, e também, permitir a verificação do comportamento correto entre componentes interagindo. A aplicação de redes de Petri para a modelagem e verificação de sistemas, ao nível de especificação e concepção, é bem conhecida. A despeito de poderesos mecanismos de estruturaçao disponíveis na teoria de redes de Petri, para a construção do modelo de um sistema complexo, o projetista poderá ainda se defrontar com o problema da explosão de estados, quando analizando e verificando grandes sistemas. Neste trabalho introduzimos uma metodologia de análise composicional para um tipo de redes de Petri de alto-nível denominada *G-Nets*. Ainda mais, tendo em mente o objetivo de aplicar esta metodologia para sistemas atuais, nós abordamos a introdução sistemática de propriedades de tolerância a falhas na concepção de um componente. A aplicação e exemplificação da metodologia introduzida é na concepção de sistemas de software.

Abstract

Analysis of G-Net Systems Based Upon Decomposition

by Angelo Perkusich

When specifying, designing and analyzing complex systems, it is necessary to adopt a modular or compositional methodology. This methodology shall allow the designer the ability to verify local properties of individual modules or components in the system, and also shall allow the verification of the correct behavior of interacting components. The application of Petri nets for the modeling and verification of systems, at specification and design levels are well know. Despite of powerful structuring mechanisms available in the Petri nets theory for the construction of the model of complex systems, the designer is still likely to face the problem of state explosion, when analyzing and verifying large systems. In this work we introduce a compositional analysis methodology for a kind of high level Petri nets called *G-Nets*. Moreover, having in mind the objective of applying this methodology for actual systems, we also address the systematic introduction of fault-tolerant properties in the design of a component. The application and exemplification of the introduced analysis methodology is on the design of software systems.

Agradecimentos

Gostaria de agradecer aos meus orientadores Shi-Kuo Chang e Misael Elias de Morais por proverem direção e encorajamento em todos os momemntos. Agradeço a Shi-Kuo Chang por suas sugestões a este trabalho e por prover um ambiente de pesquisa e trabalho que possibilitaram sua conclusão. Muitos membros do Center for Parallel and Distributed Systems (CPDIS) da Universidade de Pittsburgh e do Departamento de Engenharia Elétrica (DEE) da Universidade Federal da Paraíba contribuíram para esta tese. Agradeço a Daniel Mossé do CPDIS pelo seu incentivo, comentários, e por coloborar na revisão desta tese. Agradeço a Antonio Marcus Nogueira Lima do DEE por seu contínuo suporte a este trabalho, e por sua disponibilidade para discutir e ajudar em muitos dos problemas que encontramos durante a pesquisa. Agradeço a Mary Conley, cujo humor e ajuda em diferentes circunstâncias tornaram possível uma estada mais agradável em Pittsburgh. Agradeço ainda a Péricles Rezende de Barros e Gurgip Singh Deep por seus incentivos.

Sou grato a Afonso Costa Silva que sempre encorajou a finalização desta tese, e por sua paciente ajuda na revisão de alguns capítulos desta tese, agradeço principalmente por sua amizade.

Agradeço ainda a Jorge Cesar Abrantes de Figueiredo como colega de pesquisa e amigo por seu constante incentivo e colaboração.

Agradeço a minha mãe e irmãs por todo amor e suporte que me deram durante os últimos 34 anos.

Não teria sido possível completar esta tese sem a ajuda de minha esposa Lígia. Agradeço por ela ter assumido responsabilidades adicionais sempre que estive pressionado, e por nunca duvidar que eu completaria esta tese. Finalmente, agradeço aos meus filhos, Mirko e Andrei, por proverem energia, excitação, e humor quando estes foram mais necessários.

Esta pesquisa foi financiada em parte pelo CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) através da bolsa 201463/91-1.

CONTEÚDO

LISTA DE FIGURAS

Capítulo 1
INTRODUÇÃO

A aplicação de uma abordagem distribuída tem sido mais e mais utilizada para tratar com a complexidade associada à concepção de grandes sistemas de software. Um sistema distribuído de software é composto por processos comunicantes, ou componentes, que resolvem cooperativamente um problema comum [100]. Estes processos formam unidades de programa, que comunicam-se entre si por meio de mensagens passantes. Processos podem ser alocados a agentes de processamento (computadores) e escalonados para execução em um ambiente fracamente acoplado. A concepção de um sistema distribuído de software, assim como no caso de muitos outros sistemas complexos, apresenta inúmeras dificuldades.

Do ponto de vista de engenharia de software, diferentes problemas podem surgir durante as fases (ciclo de vida) de desenvolvimento de software, ou seja, durante a definição de requisitos, especificação, concepção, projeto, codificação e teste. Formalismos de modelagem para especificação e concepção de um sistema de software devem considerar que um sistema distribuído coordena processos assíncronos. Portanto, estes formalismos devem suportar a coordenação e comunicação de atividades em diferentes níveis de abstração.

Nesta tese consideramos que um sistema distribuído de software é composto por um conjunto fixo de processos, definidos durante a fase de concepção. Dois objetivos principais devem ser alcançados em um sistema distribuído de software, os quais são: alto desempenho e tolerância a falhas. Alto desempenho depende do paralelismo inerente ao suporte de comunicação, a arquitetura da rede de comunicação. O segundo objetivo é prover tolerância a falhas, o que neste caso pode em princípio ser realizado porque, em um sistema distribuído os recursos são inerentemente redundantes.

1.1 Conceitos Básicos

Um dos principais aspectos a ser considerado quando especificando e concebendo sistemas complexos distribuídos de software é a inerente concorrência que estes sistemas apresentam. Além disto, deve-se considerar propriedades estruturais do sistema, assim como o comportamento dos componentes do sistema e as mensagens de comunicação

trocadas entre estes componentes. Ainda mais, técnicas formais devem ser utilizadas quando especificando e concebendo este tipo de sistemas, pois técnicas formais possuem sólidas bases para a análise da validade da especificação, de forma que erros podem ser detectados ainda durante a fase de concepção. Além disto as técnicas formais facilitam a modificação, re-concepção, manutenção e reusabilidade.

Redes de Petri e Lógica Temporal são modelos com sólida base formal, possibilitando a especificação e concepção de sistemas complexos distribuídos de software.

1.1.1 Redes de Petri

Uma rede de Petri é um grafo direcionado bipartido, mais um estado inicial denominado marcação inicial [77]. O grafo direcionado consiste de dois tipos de nós, denominados *lugares* e *transições*. Os nós em uma rede de Petri são relacionados (conectados) por arcos rotulados com pêsos (inteiros positivos). Um arco não pode relacionar componentes do mesmo tipo. Graficamente lugares são representados por círculos e transições por retângulos. Um lugar p é entrada para uma transição t se existe um arco direcionado conectando o lugar à transição, neste caso o lugar é um *lugar de entrada*. Um lugar p é saída para uma transição, se existe um arco direcionado conectando a transição ao lugar, neste caso o lugar é um *lugar de saída*. O grafo direcionado define a estrutura de um sistema representado por uma rede de Petri. A definição informalmente introduzida para redes de Petri é também denominada grafo de suporte ou estrutura da rede. Uma marcação atribui a cada lugar p um número k, inteiro não negativo, de elementos denominados fichas. Fichas são representadas por pontos pretos. Quando um número k, não negativo, de fichas é atribuído ao lugar, dizemos que o lugar está marcado com k fichas. Uma marcação é um vetor com o mesmo número de lugares que a estrutura da rede. O comportamento do sistema modelado pela estrutura da rede pode ser caracterizado pelo movimento de fichas pelos lugares, quanto a rede é *executada*. Este movimento de fichas caracteriza o comportamento dinâmico do sistema em termos de estados e suas mudanças. De modo a mover fichas transições disparam. Uma transição deve estar *habilitada* na marcação corrente para poder disparar. Uma transição é dita habilitada se todos os lugares de entrada são marcados, por pelo menos, o mesmo número de fichas definido pelo pêso associado aos arcos conectando estes lugares à transição. Uma transição habilitada pode disparar. Quando uma transição dispara, o número de fichas associados aos pesos dos arcos de entrada são removidas dos lugares de entrada, e depositadas nos lugares de saída de acordo com os pesos associados aos arcos saindo da transição, conectando os lugares de saída. Este movimento de fichas pela rede é também

conhecido como *jogo de fichas.*

A análise de redes de Petri é principalmente baseada no grafo de alcançabilidade ou em técnicas algébricas lineares. Um *grafo de alcançabilidade* representa o conjunto de estados alcançáveis, e pode ser usado para verificar uma variedade de propriedades, p.e., se a rede é livre de bloqueio mortal (*deadlock*). Técnicas algébricas lineares são utilizadas para calcular *invariantes.* A idéia é representar a rede por uma matriz de incidência e marcações por vetores de controle. Esta representação pode então ser utilizada para caracterizar a dinâmica do sistema utilizando-se a equação de controlabilidade utilizada em teoria de sistemas de controle. Portanto, pode-se então derivar equações algébricas lineares, cujas soluções características são denominadas invariantes. Dois tipos de invariantes podem então ser identificados: *invariantes de lugar* e *invariantes de transição.* Para uma introdução detalhada a estes e outros aspectos relacionados a redes de Petri o leitor pode referir-se à [77, 87]. No Capítulo 2 apresentamos uma revisão dos conceitos básicos relacionados a redes de Petri, entretanto esta introdução esta orientada a fornecer um embasamento teórico para temas que serão abordados nesta tese.

Podemos identificar duas direções principais para pesquisas utilizando redes de Petri: teoria *pura* de redes de Petri e teoria *aplicada* de redes de Petri. No primeiro caso, a pesquisa enfoca principalmente os fundamentos teóricos de redes de Petri, enfatizando as bases matemáticas e lógicas da teoria. Mesmo considerando-se que os resultados produzidos por esta direção de pesquisa são úteis, muitas técnicas e conceitos introduzidos são difíceis de serem aplicados a problemas práticos, pricipalmente devido a complexidade dos sistemas atuais. A segunda direção de pesquisa enfoca a aplicação de redes de Petri à modelagem e análise de sistemas. Deste ponto de vista, as redes de Petri tem sido aplicadas com sucesso à muitas áreas diferentes como protocolos de comunicação, sistemas distribuídos, e sistemas flexíveis de manufatura. Para lidar com a complexidade destes sistemas torna-se freqüentemente necessário extender o modelo básico de redes de Petri. Entretanto, estas extensões dificultam ou mesmo não permitem a aplicação de muitas das técnicas desenvolvidas na teoria de redes de Petri.

As duas direções de pesquisa não levaram ao desenvolvimento de uma metodologia eficiente que explore completamente as capacidades de análise e as aplicações práticas de redes de Petri para sistemas complexos atuais. Um dos objetivos desta tese é mostrar nossa contribuição com vistas a prover uma ligação entre estas duas direções de pesquisa.

1.1.2 Lógica Temporal

Lógica temporal provê um sistema formal para descrever a ocorrência de eventos no tempo. Lógica temporal pertence à classe de lógica modal originalmente desenvolvida para tratar com diferentes "modos" da verdade. Uma asserção p pode ser falsa considerando-se circunstâncias presentes, mas a asserção *possivelmente* p pode ser verdadeira se existem outras circunstâncias em que a assrção p é verdadeira. Em um sistema de lógica temporal as modalidades são interpretadas temporalmente, de modo a descrever como valores verdadeiros de asserções variam com o tempo. Modalidades temporais incluem *alguma vez* p, a qual é verdadeira para o instante presente no tempo se existe um momento futuro em que p torna-se verdadeiro, e *sempre* q, o qual é verdadeiro se q é verdadeiro para todos os instantes futuros.

Um sistema de lógica temporal especifica a sintaxe de fórmulas legais e a semântica para interpreta-las. A lógica temporal que discutiremos é lógica proposicional ordinária, com conectivos temporais para implicitamente lidar com o tempo. A semântica de lógica temporal pode ser representada por grafos direcionados, denominadas estruturas. Intuitivamente, os *estados* de uma estrutura (vértices do grafo) denotam possíveis mundos ou instantes no tempo, e são rotulados a partir de um conjunto de proposições atômicas que são verdadeiras para o estado. Os arcos do grafo denotam a passagem do tempo: um arco de um estado s para um estado t indica que o mundo t é um possível sucessor do mundo s. Portanto, cada caminho saindo do estado denota possíveis futuros para o estado. O sistema de lógica temporal é dito *linear no tempo* se as estruturas sobre as quais as fórmulas são interpretadas são cadeias finitas unidirecionais, i.e., cada estado possui um único futuro, e *alternativo no tempo* se as estruturas são grafos arbitrários (sujeitos à restrição de que cada estado tenha pelo menos um sucessor, i.e., cada mundo tem pelo menos um futuro). Lógica temporal linear no tempo é extensivamente discutida em [74]. No caso de lógica temporal alternativa no tempo (*branching time temporal logics*) o tempo é visto como uma ordenação parcial de uma coleção de instantes discretos no tempo, possibilitando a especificação de diversos futuros alternativos em qualquer instante de tempo.

A combinação de redes de Petri e lógica temporal tem sido assunto de pesquisa nos últimos anos [6, 18, 19, 73, 94, 95, 96, 107]. Com base nestas pesquisas podemos identificar duas abordagens para combinar redes de Petri e lógica temporal, abordagem *axiomática* e *verificação de modelo*. Na abordagem axiomática a análise de um sistema de rede é completamente reduzida à uma questão lógica. Verificação de modelo é um tipo de método híbrido, combinando lógica com métodos de análise convencionais.

Estamos interessados na abordagem por verificação de modelo, combinada com análise de alcançabilidade. Uma razão para isto é porque o grafo de alcançabilidade para uma rede de Petri pode ser diretamente utilizado como semântica para fórmulas em lógica temporal. Ainda mais, fórmulas em lógica temporal podem ser utilizadas como consultas para questionar o grafo de alcançabilidade.

1.2 Especificação e Concepção de Sistemas Complexos

Usualmente a concepção de sistemas complexos é baseada numa abordagem *top-down* ou *bottom-up*. A concepção *top-down* de um sistema inicia-se com uma descrição de alto nível do sistema. Esta descrição de alto nível é então redefinida através de sucessivos refinamentos, pela substituição de partes do modelo por modelos mais detalhados. Nesta abordagem, o projetista trata com sub-problemas relativamente pequenos, que podem ser distribuídos entre diversos projetistas. No caso da abordagem *bottom-up*, inicia-se com a concepção dos módulos de mais baixo nível do sistema, os quais são então abstraídos para obter a concepção de alto nível do sistema.

Entre outros formalismos adotados para a especificação e concepção de sistemas distribuídos de software, redes de Petri [77] e lógica temporal [40, 74] estão entre os mais aplicados. Como anteriormente discutido, redes de Petri, assim como lógica temporal, tem sido aplicados para a modelagem de diversos sistemas que podem ser caracterizados como distribuídos, concorrentes e assíncronos. Uma vez que nosso principal interresse está na aplicação destes formalismos para sistemas distribuídos de software, introduziremos de modo informal a aplicação destes dois formalismos para este tipo de sistemas.

Para modelar um sistema de software distribuído complexo, podem ser aplicadas técnicas de abstração e refinamento. No caso de redes de Petri, refinamento significa a substituição de uma transição ou um lugar por uma rede. Quando este tipo de substituição ocorre, as propriedades da rede refinada podem ou não ser preservadas. Por exemplo, a rede original poderia ser *segura* e *viva*, no sentido de que sempre alguma transição estaria habilitada. Mas, algumas propriedades podem ser alteradas ou novas definidas. Isto ocorre pois as redes refinadas, substituindo o lugar ou a transição, podem ser capazes de executar ações que não estavam definidas no modelo primário. Em [15] apresenta-se uma revisão relativa a preservação de comportamento e equivalência para refinamentos em redes de Petri.

Os maiores problemas quando modelando grandes sistemas utilizando redes elementares, ou redes lugar/transição são: para um grande sistema as redes de Petri podem se tornar proibitivamente grandes; noções de estruturas de dados e dependência de dados

refinadas são perdidas, e; em muitas formulações de redes de Petri somente *sistemas fechados* são descritos, de modo que a interação com algum tipo de ambiente não pode ser adequadamente expressada. Com a introdução do conceito de *fichas individuais*, representando elementos de dados fluindo na rede, foi possível modelar componentes comuns uma só vez, e atribuir fichas distinguíveis para cada componente idêntico. Este desenvolvimento levou a uma classe de redes de Petri denominda *redes de Petri de alto-nível*, incluindo *redes Predicado/Transição (redes PrT)* [45, 48] *redes de Petri Coloridas (redes CP)*, [59] *redes Relação (Redes-Rel)* [91], e *redes de Petri com fichas individuais* [92]. Entretanto, mesmo com as redes de Petri de alto-nível, a modelagem de sistemas pode ainda ser difícil, pois não são providos meios nestes modelos para especificar a *estrutura* de um sistema. Por exemplo, não há conceito de hierarquia. Para remediar este problema, outras extensões às redes de Petri de alto-nível, incluindo o conceito de hierarquia tem sido propostas. Os modelos de rede de Petri resultantes são denominados redes de Petri hierárquicas, incluindo *redes CP hierárquicas* [57, 59] e *redes PrT hierárquicas* [79]. Mesmo considerando que as construções hierárquicas introduzidas não extendem o poder de modelagem de redes de alto-nível, ferramentas de estruturação são introduzidas de modo a facilitar a construção de modelos de grandes sistemas na prática [57].

O problema de gerenciar grandes sistemas é principalmente uma questão de modularidade. Portanto, seria desejável compor grandes redes a partir de redes menores, de modo que a semântica da grande rede pudesse ser deduzida a partir das redes menores. Intuitivamente isto pode ser caracterizado pela habilidade em *esconder* detalhes internos das redes e considerarem-se somente aqueles necessários.

Lógica temporal, assim como redes de Petri é um modelo maduro com sólida base formal. Apesar do fato de que redes de Petri permitem ao usuário modelar precisamente a estrutura e o comportamento de um sistema, principalmente quando usando redes de alto-nível, não é possível ao projetista especificar ou atribuir aquelas propriedades que ele deseja. Em redes de Petri temos que construir um modelo e então verificar ou provar que o modelo de rede satisfaz as propriedades desejadas. Por outro lado, lógica temporal permite ao usuário atribuir propriedades ao sistema por meio de axiomas e fórmulas. Estas fórmulas nada dizem a respeito da estrutura do sistema, mas sim identificam propriedades importantes. No caso de programas concorrentes devemos ser capazes de atribuir propriedades para os processos e relacioná-las com os vários estados durante a execução dos processos em um ambiente distribuído. Lógica temporal prove esta capacidade, e é um modelo bastante conhecido para a especificação de sistemas

concorrentes [40, 74].

1.3 Motivação

Em [35, 36, 37, 38] introduziu-se o conceito de *G-Nets* e sistemas de *G-Nets*. *G-Nets* são uma abordagem baseada em redes de Petri para a especificação e concepção modular de sistemas distribuídos de informação. Esta abordagem é uma integração da teoria de redes de Petri com a abordagem de engenharia de software baseada em objetos para a concepção de sistemas. A motivação para esta integração é criar uma ponte entre o tratamento formal de redes de Petri e uma abordagem modular, orientada a objetos para especificação, concepção e prototipagem de sistemas complexos de software.

A notação de *G-Nets* incorpora noções de módulo e estrutura de sistema à redes de Petri, promovendo abstração, encapsulamento e fraco acoplamento entre módulos. Estas características tornam *G-Nets* uma ferramenta mais aplicável para a especificação e concepção de sistemas de software complexos, suportando modificação incremental de um sistema.

Uma especificação ou concepção baseada em *G-Nets* consiste de um conjunto independente e fracamente acoplado de módulos (*G-Nets*) organizado em termos de diferentes estruturas de sistema. Estruturas recursivas podem também ser especificadas. Uma *G-Net* é encapsulada de forma tal que um módulo pode somente acessar outro módulo através de um mecanismo bem definido, denomindado abstração de *G-Net*, e nenhuma *G-Net* pode diretamente afetar a estrutura interna de um outro módulo. Como indicado por Booch [12], dificilmente podemos conceber um sistema complexo corretamente na primeira tentativa, então a concepção de um sistema complexo de software é um processo evolutivo, no qual repetidas alterações são necessárias. Além disto, como argumentado por Luqi [72], os benefícios potencias da prototipagem dependem de modo crítico da habilidade de modificar o comportamento do protótipo com esforço substancialmente menor que o requerido para modificar e produzir software. As características modulares de *G-Nets* proveem os suporte necessário para a concepção incremental e modificação sucessiva.

Uma especificação baseada em *G-Net* pode ser diretamente executada em um ambiente distribuído, onde uma *G-Net* é a unidade natural para distribuição e execução [24]. O mecanismo de execução permite que múltiplas instâncias de invocação de uma mesma*G-Net* possam ser executadas simultaneamente. A execução de uma *G-Net* por múltiplos agentes computacionais permitem que a execução paralela.

Do ponto de vista teórico uma técnica formal de transformação foi introduzida em

[35, 38]. Esta técnica permite que uma especificação em *G-Nets* seja traduzida para um conjunto de *redes PrT*, onde cada *rede PrT* corresponde a um método na especificação em *G-Net*, e apresenta uma semântica equivalente à da última. Por meio desta transformação, as técnicas de análise formal desenvolvidas para redes PrT podem ser utilizadas para analizar especificações em *G-Net*. Infelizmente existem algumas limitações para esta técnica formal de transformação. A limitação mais séria é o fato de que esta técnica não compartilha das carcterísticas modulares e de abstração de *G-Nets*. Portanto, quando aplicadas para analizar sistemas complexos de *G-Net*, o projetista muito provavelmente deverá defrontar-se com o conhecido problema de *explosão de estados*, o qual é considerado um dos maiores problemas da análise em redes de Petri [77].

1.4 Escopo e Principais Resultados desta Tese

Esta tese explora a possibilidade de utilizar a decomposição natural de um sistema distribuído, para desenvolver uma metodologia modular de análise para *G-Nets* e sistemas de *G-Nets*. De modo a cumprir esta tarefa, primeiro formalizamos o conceito de *G-Nets* e sistemas de *G-Nets*, assim como sua decomposição. Introduzimos restrições estruturais e comportamentais na comunicação entre *G-Nets* em um sistema de *G-Nets*. Estas restrições permitem que apliquemos o paradigma assume/garante [74] de modo a desenvolver uma metodologia modular de análise para sistemas de *G-Nets*. Esta metodologia modular de análise permite que o projetista raciocine considerando componentes, processos ou módulos de software, e suas interações com um ambiente. Quando concebendo um componente, suposições são feitas a respeito do comportamento do ambiente, de modo que o comportamento local de um componente pode ser especificado e verificado. Quando concebendo componentes que compõe o ambiente de um outro componente, o projetista deve garantir que estes componentes comportam-se como assumido. Na verdade, o projetista garante a concordância do ambiente com respeito ao comportamento ambiental assumido.

Uma vez que estamos discutindo a concepção de sistemas distribuídos de software reais, devemos também considerar aspectos relacionados a tolerância a falhas, p.e., autoproteção de componentes e diversidade de concepção [1, 61]. Isto pode ser justificado devido ao fato de que o sistema como um todo, ou ao menos parte dele, será codificado por programadores humanos, os quais podem cometer erros, que por sua vez podem levar o sistema a falhar. Dentro deste contexto introduzimos uma metodologia para sistematicamente introduzir propriedades de tolerância a falhas na concepção de um

sistema de software.

As principais contribuições desta tese são detalhadas a seguir.

Formalização de conceito de G-Nets e sistemas de G-Nets. A formalização que introduzimos nesta tese possibilita a aplicação de uma metodologia modular de análise para *G-Nets* ou sistemas de *G-Nets*. Ao invés de utilizar uma técnica de transformação como a introduzida em [35, 38], definimos precisamente todos os componentes notacionais utilizados em *G-Nets* e sistemas de *G-Net*. Definimos também a semântica destes elementos notacionais de modo que possamos decompor *G-Nets* e definir técnicas de análise próprias.

Decomposição de G-Nets. A técnica de decomposição que introduzimos define claramente a interface entre *G-Nets* e um fraco acoplamento entre elas. Portanto, podemos impor restrições estruturais e comportamentais à esta interface de modo a permitir composição modular para *G-Nets*. Nossa abordagem de decomposição melhora a estrutura das redes permitindo a clara definição de um protocolo de comunicação entre redes em termos de suas habilidades funcionais, e não somente em termos de sua estrutura gráfica, provendo ao projetista uma visão abstrata da rede.

Formalização do conceito de reentrância e outras propriedades estruturais e comportamentais para G-Nets e sistemas de G-Nets. A introdução de restrições comportamentais e estruturais nos permite raciocinar a respeito de *G-Nets* como componentes individuais, que devem satisfazer determinadas suposições comportamentais. A possibilidade de reinicialização e terminação são as principais propriedades que uma *G-Net* deve satisfazer. Reinicialização garante que a rede sempre pode voltar ao seu estado inicial. Terminação garante que a realização de uma determinada funcionalidade em termos de uma rede, pode sempre atingir um estado no qual uma mensagem é retornada para o ambiente. Além disto, introduzimos e formalizamos o conceito de *G-Net* dinâmicamente livre de conflito. Esta propriedade permite que assumamos que diferentes processos em um sistema distribuído modelado por um sistema de *G-Nets* possam progredir independentemente.

Definição e formalização de uma metodologia modular de análise. A metodologia, baseada no paradigma assume/garante permite que raciocine-se em termos de componentes individuais e seus ambientes. A definição de uma abordagem modular com uma base formal para *G-Nets* e sistemas de *G-Nets* torna possível a aplicação destas a sistemas complexos. Além disto, a substituição e reutilização de componentes de software modelados por *G-Net* são simplificados, desde que a interação ou equivalência de interface entre o velho e o novo componente permanece a mesma, não sendo necessário

executar nenhuma análise global adicional.

Aplicação dos conceitos de sistemas de G-Net a um sistema complexo real. Além de exemplos como a solução utilizando *G-Nets* para o problema do produtor/consumidor e a interação na arquitetura cliente/servidor, introduzimos a aplicação de nossa metodologia de concepção e análise para o controle distribuído de um sistema de trens.

Introdução de metodologias para embutir propriedades de tolerância a falhas na concepção de sistemas distribuídos de informação e controle. A premissa de que um sistema distribuído de software deve fornecer serviços confiáveis impõe a necessidade de introduzir propriedades de tolerância a falhas na concepção de uma *G-Net*. Estas propriedades evitam a interferência no comportamento de um componente, quando erros ambientais ocorrem. Em outras palavras, deve ser possível a detecção de um comportamento faltoso do ambiente com respeito ao assumido, quando concebendo um componente, de modo que esta falta não seja propagada para o componente objeto e outros com os quais este interage. Além disto, devemos vislumbrar como redundância baseada, em diversidade de concepção, pode ser introduzida na concepção de um sistema de *G-Nets*. Nesta tese introduzimos tanto uma metodologia sistemática para embutir auto-proteção na concepção de uma *G-Net*, como técnicas para implementar diversidade de concepção.

Definição de uma ferramenta para auxiliar o projetista na análise de G-Nets e sistemas de G-Nets. De modo a auxiliar o projetista a analizar um sistema de *G-Nets*, introduzimos uma ferramenta para a análise de sistemas de *G-Nets*. Utilizamos para tanto o método do grafo de alcançabilidade. Nossa ferramenta constrói o grafo de alcançabilidade de uma *G-Net* baseando-se na especificação da rede e de suposições a respeito do comportamento do ambiente. Além disso, implementamos algorítmos para auxiliar o projetista na análise deste grafo de alcançabilidade.

1.5 Estrutura da Tese

O restante desta tese e organizado como segue-se. No Capítulo 2 revisamos os conceitos básicos relacionados com redes de Petri, lógica temporal e tolerância a falhas em software. No Capítulo 3 apresentamos as bases de *G-Nets* e sistemas de *G-Nets*. Introduzimos uma formalização para *G-Nets* que possibilita a aplicação de uma abordagem modular/composicional para verificação. Neste capítulo exemplificamos a aplicação de sistemas de *G-Nets* para modelar o problema produtor/consumidor, a interação cliente/servidor, e o problema de controle de um sistema de trens. No Capítulo 4 introduzimos a decomposição para sistemas de *G-Nets* e introduzimos ainda uma abordagem para a verificação destes sistemas. Apresentamos como decompor *G-Nets* com o objetivo

de executar análise e introduzir propriedades de tolerância a falhas. Propriedades comportamentais e estruturais são também definidas. No Capítulo 5 introduzimos uma ferramenta de verificação bem como sua aplicação para o problema produtor/consumidor. Por fim, no Capítulo 6 apresentamos a conclusão desta tese.

Capítulo 2

CONCEITOS BÁSICOS

2.1 Introdução

Neste capítulo introduzimos aos conceitos básicos de redes de Petri e lógica temporal. O objetivo principal é revisar os conceitos básicos relacionados com estas duas teorias. Discutimos também a aplicação de metodologias composicionais baseadas em redes de Petri e lógica temporal para análise e verificação de sistemas. O nível de detalhe variará através das secções, algumas apresentarão as definições formais exatas, enquanto outras objetivarão prover uma visão intuitiva através de descrições informais e exemplos.

Uma vez que nesta tese discutiremos a especificação e concepção de sistemas de software complexos reais, introduziremos também conceitos relacionados com tolerância à falhas em sistemas de software.

2.2 Preliminares

O objetivo principal desta secção é o de introduzir os conhecimentos básicos relacionados com modelos de redes de Petri aos quais nos referiremos nesta tese. Após introduzirmos uma classificação básica para os sistemas de rede de Petri, como definido em [11], introduziremos alguns conceitos relacionados à sistemas de redes de Petri. A classificação para sistemas de rede é:

1. *Sistemas de rede de baixo-nível* onde os lugares são marcados por fichas não estruturadas. Neste caso temos os seguintes grupos principais:

   (a) sistemas de rede de baixo-nível cujos lugares são marcados por no máximo uma ficha não estruturada. Neste caso lugares representam *condições.*

   (b) sistemas de rede de baixo-nível cujos lugares são marcados por várias fichas não estruturadas. Por exemplo, sistemas de rede cujos lugares representam contadores.

2. *Sistemas de rede* cujos lugares são marcados por fichas estruturadas, freqüentemente conhecidos como *redes de alto-nível.* No caso destas redes, a informação pode ser associada às fichas.

Diversos sistemas de redes podem ser definidos para cada um dos níveis, entre outros exemplos de redes pertencendo ao nível 1 citamos: sistemas Condição/Evento (C/E) [87, 91]; sistemas de Redes Elementares (RE) [99, 105]; sistemas 1-seguros [49]; e sistemas Lugar/Transição (P/T) [93]. O leitor interessado pode referir-se à [11, 14, 77, 87, 91] para uma introdução detalhada às diferentes classes de redes de Petri.

No caso de sistemas de alto nível os principais são as *redes de Petri de Alto-Nível* como: *redes Predicado/Transição (redes PrT)* [45, 48]; *redes de Petri Coloridas (redes CP)* [59, 60]; *redes Relação (Redes-Rel)* [91], e *redes de Petri com Fichas Individuais* [92]. Apesar do fato de que as diferentes redes de alto-nível apresentarem diferentes capacidades de expressão introduziremos somente as redes PrT, devido ao fato deste tipo de redes apresentarem um alto grau de relacionamento com *G-Nets*, as quais serão introduzidas no Capítulo 3.

Independentemente do nível do sistema, uma rede de Petri é representada por um grafo bipartido possuindo dois tipos de elementos: lugares e transições. Lugares e transições são relacionados por relações de fluxo, representadas por arcos conectando lugares a transições, e transições a lugares. No que segue introduziremos para o caso de sitemas de baixo nível os sistemas de redes elementares (*RE*) e sistemas P/T, e para o caso de redes de alto-nível as redes PrT.

Sistemas de Redes Elementares (RE)

Antes de introduzirmos o conceito de *sistemas RE* introduziremos os conceitos mais elementares relativos a redes de Petri, iniciando pelas *redes Condição/Evento.*

Definição 2.1 *Redes Condição/Evento (C/E)*
Uma *rede C/E* é a tripla $\mathcal{N} = (P, T; F)$, de modo que:

1. P e T são conjuntos finitos tais que $P \cap T = \emptyset$ e $P \cup T \neq \emptyset$. Na representação gráfica para uma rede, os elementos pertencentes a P são representados por círculos e são denominados lugares, e os elementos pertencentes a T são representados por retângulos e denominados transições.

2. $F \subseteq (P \times T) \cup (T \times P)$ é tal que: $domínio(F) \cup faixa(F) = P \cup T$, e é denominada *relação de fluxo.* Graficamente está relação é representada por arcos orientados.

Na Definição 2.1 P é o conjunto de *P-elementos*, T é o conjunto de *T-elementos*, e F é a *relação de fluxo*. Ainda mais, define-se $X = P \cup T$ como o conjunto de *elementos* da rede. A condição 1 significa que P e T formam uma partição de X, e que X, e portanto a rede, não pode ser vazia. A condição 2 força a rede a não conter elementos isolados[1].

Definição 2.2 *Matriz de incidência*
Para uma rede C/E com n transições e m lugares, a *matriz de incidência* $\mathcal{I}$ é uma matriz $n \times m$. Uma entrada típica é dada por:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{2.1}$$

tal que:

$$a_{ij} = \begin{cases} -1 & \text{se existe um arco da transição } i \text{ para seu lugar de saída } j \\ 0 & \text{se não existe arco algum entre a transição } i \text{ e o lugar } j \\ 1 & \text{se existe um arco para a transição } i \text{ de seu lugar de entrada } j \end{cases}$$

Uma rede C/E pode então ser representada graficamente por um grafo direcionado, e algébricamente por uma matriz de incidência. Na Figura 2.1 apresentamos uma rede C/E e sua matriz de incidência.

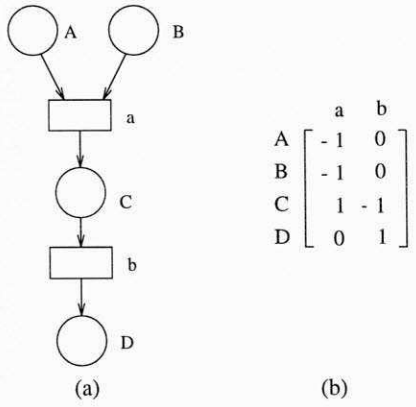

Figura 2.1: Representação de uma rede, (a) grafo, (b) matriz de incidência

Definição 2.3 *Pré e pós-conjunto*

---

[1] Deve-se observar que procurou-se utilizar a mesma nomenclatura adotada em inglês, isto é, P para lugares, do inglês *places*.

1. O *pré-conjunto* de um elemento $p \in X$ é denotado por:

$$\bullet x = \{y \in X : (y, x) \in F\}. \tag{2.2}$$

2. O *pós-conjunto* é denotado por:

$$x\bullet = \{y \in X : (x, y) \in F\}. \tag{2.3}$$

Portanto, para $Y \subseteq X$, $\bullet Y = \bigcup_{y \in Y} \bullet y$, and $Y\bullet = \bigcup_{y \in Y} y\bullet$. Isto é o pré-conjunto (pós-conjunto) é a união dos pre-conjuntos (pós-cunjuntos) de elementos.

Baseando-se na Definição 2.1 definen-se algumas propriedades estáticas para uma rede C/E. Por exemplo, uma rede de Petri $\mathcal{N}$ é dita ser *pura* se e somente se $\forall x \in X \bullet x \cap x\bullet = \emptyset$, i.e., não há arco conectando ao mesmo tempo um lugar de entrada e saída à uma transição. Para outras propriedades refira-se a [89].

Sistemas RE são um tipo de sistema de redes bastante básico e são definidos como segue.

Definição 2.4 *Sistemas de Rede Elementares (RE)*
Um *sistema RE* é definido pela quádrupla $\mathcal{N} = (B, E; F, c_{in})$ onde:

1. $(B, E; F)$ é uma rede.

2. $c_{in} \subseteq B$.

Elementos de B são denominados *condições*, elementos de E são denominados *eventos*, $(B, E; F)$ é denominada *rede suporte de* $\mathcal{N}$, e c_{in} é denominado *caso inicial* de $\mathcal{N}$, e $c_{in} \neq \emptyset$. Em sistemas *RE* condições podem somente ser verdadeiras ou falsas. Um evento (e) pode somente ocorrer se todas as pre-condições são verdadeiras e todas as pós-condições são falsas. Quando um evento ocorre todas as pré-condições deixam de ser verdadeiras e todas as pós-condições passam a ser verdadeiras.

A rede de suporte captura a estrutura do sistema modelado por $\mathcal{N}$. A evolução (ou execução) do sistema, e portanto seu comportamento, é definida aplicando-se uma regra de transição, a qual especifica as condições nas quais um evento pode ocorrer, e como a ocorrência de eventos modifica o estado das condições.

Definição 2.5 *Regra de transição*
Considerando-se que $\mathcal{N} = (B, E; F)$ é uma rede, $e \in E$ e $c \subseteq B$.

1. e é dito habilitado em c, denotado por $c[e\rangle_{\mathcal{N}}$, se e somente se $\bullet e \subseteq c$, and $e\bullet \cap c \neq \emptyset$.

2. Se e está habilitado em c, então a ocorrência de e leva de c para c', o que é denotado por $c[e\rangle_{\mathcal{N}} c'$, se e somente se $c' = (c - \bullet e) \cup e\bullet$

Redes Lugar/Transição (P/T)

Redes *Lugar/Transição (P/T)* estão entre os modelos de rede mais aplicados. Em redes P/T, lugares podem ser marcados por um ou mais fichas não estruturadas, as quais, na maioria dos casos representam contadores. Lugares possuem uma capacidade definida, a qual indica o número máximo de fichas que um lugar pode receber. Nesse caso os arcos possuem um peso associado, o qual indica o número de fichas que podem fluir através deles a cada ocorrência (disparo) das transições envolvidas[93].

Definição 2.6 *Redes Lugar/Transição*
A tupla $\mathcal{N} = (P, T; F, K, W, M_0)$ é denominada uma rede Lugar/Transição se e somente se:

1. $(P, T; F)$ é uma rede onde os P-elementos são denominados lugares e os T-elementos são denominados transições.

2. $K : P \to \mathbb{N}^+ \cup \{\infty\}$ é a função de capacidade.

3. $W : F \to \mathbb{N}^+$ é a função de peso.

4. $M_0 \to \mathbb{N}$ é uma função de marcação inicial satisfazendo $\forall p \in P : M_0(p) \leq K(P)$.

Uma rede P/T tal que $\forall p \in P : K(p) = \infty$ e $\forall f \in F : W(f) = 1$ pode ser denotada simplesmente por $\mathcal{N} = (P, T; F, M_0)$ e e denominada um *rede de Petri ordinária.*

Definição 2.7 *Regra de transição*
Considerando-se que $\mathcal{N} = (P, T; F, K, W, M_0)$ é uma rede P/T.

1. A função $M : P \to \mathbb{N}$ é dita uma marcação de $\mathcal{N}$ se e somente se $\forall p \in P : M_0(p) \leq K(p)$.

2. Um transição $t \in T$ está habilitada em M se e somente se $\forall p \in P : W(p, t) \leq M(p) \leq K(p) - W(t, p)$.

3. Se $t \in T$ é uma transição habilitada na marcação M, então t pode ocorrer, resultando em uma nova marcação M' dada pela equação: $M'(p) = M(p) - W(p,t) + W(t,p)$, $\forall p \in P$.

4. A ocorrência ou disparo de t altera a marcação M em uma nova marcação M', e é denotada por $M[t\rangle M'$.

5. $[M_0\rangle$ é a classe de alcançabilidade (para a frente), e é definida como o menor conjunto de marcações de $\mathcal{N}$ tal que: $M_0 \in [M_0\rangle$, e se $M_1 \in [M_0$ e $M_1[t\rangle M_2$ para alguma $t \in T$, então $M_2 \in [M_0\rangle$.

Antes de introduzirmos outros conceitos relacionados com sistemas de rede, vamos discutir a regra de transição ou disparo como apresentada na Definição 2.7. Esta regra de disparo determina o comportamento da rede P/T em termos de estados do sistema (marcações) e suas mudanças. Uma marcação é representada por um vetor coluna transposto, p.e. $[M(p_1), M(p_2), \cdots, M(p_n)]$, onde $M(p_i)$ é a marcação do lugar p_i. Esta evolução de estados permite simular o comportamento dinâmico do sistema modelado por uma rede P/T. Quando uma transição está habilitada, por exemplo, a transição a na Figura 2.2, ela pode disparar. Quando a transição dispara, fichas são removidas dos lugares de entrada e depositadas nos lugares de saída. No exemplo da Figura 2.2, duas fichas são removidas do lugar A, uma vez que o peso do arco conectando-o à transição a é dois, e uma ficha é removida do lugar B (o peso um associado ao arco conectando o lugar B a transição a está implícito). Após as fichas serem removidas dos lugares de entrada, uma ficha é depositada no lugar de saída C. Em outras palavras, o sistema evolui do estado $[2,1,0,0]$, para o estado $[0,0,1,0]$.

O conceito de matriz de incidência para redes P/T é bastante similar ao introduzido para sistemas C/E na Definição 2.5. A maior diferença é que para redes P/T, aos arcos pode ser associado um peso que pode ser maior que um.

Definição 2.8 *Matriz de incidência para uma rede P/T*
Para uma rede P/T com n transições e m lugares, a matriz de incidência $\mathcal{I}$ é uma matriz de inteiros $n \times m$ e uma entrada típica é dada por:

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{2.4}$$

onde $a_{ij}^{+} = w(i,j)$, com $w \in W$, é o peso do arco da transição i para o lugar de saída j, e a_{ij}^{-} é o peso do arco para a transição i do seu lugar de entrada j.

Análogamente as redes C/E, uma rede P/T pode ser representada por um grafo orientado, e algebricamente por uma matriz de incidência. Apresentamos na Figura 2.2 um exemplo de uma rede P/T e sua matriz de incidência.

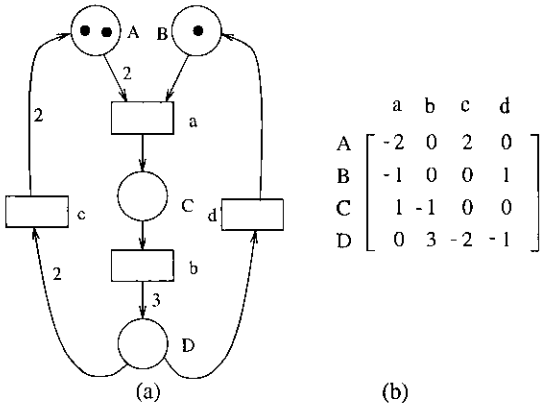

Figura 2.2: Representação de uma rede P/T, (a) grafo, (b) matriz de incidência

Entre outras propriedades para redes de Petri [11, 77] uma das mais importantes no contexto desta tese é o conceito de transição *livre de conflito*, no qual um lugar somente é entrada para uma única transição.

Definição 2.9 *Transição estaticamente livre de conflito*
Considerando-se que T é um conjunto de transições e $\mathcal{N}$ é uma rede P/T. Uma transição $t \in T$ é dita estar *estaticamente livre de conflito* se e somente se:

$$\bullet t \cap \bullet t' = \emptyset, \text{ com } t \neq t', \forall t \in T \text{ e } \forall t' \in T$$

O conceito de transição estaticamente livre de conflito é aplicado à estrutura da rede, mas é possível que, mesmo quando uma rede possui transições em conflito estático (ou estrutural), não será o caso no comportamento dinâmico do sistema [50], isto é, pode ocorrer que as transições nunca estejam habilitadas simultamente para as possíveis marcações da rede .

Definição 2.10 *Transição dinamicamente livre de conflito*
Uma rede de Petri é dita *dinamicamente livre de conflito* se e somente se para qualquer marcação alcançável M existem dois pares distintos $\langle habilita_1, t_1 \rangle$, $\langle habilita_2, t_2 \rangle$ tal que $habilita_1$ habilita t_1 para disparo em M, $habilita_2$ habilita t_2 para disparo em M, e o disparo de t_1 com a tupla $habilita_1$ disabilita o disparo de t_2 com a tupla $habilita_2$.

Logo, uma transição é dita estaticamente livre de conflito se esta não possui nenhum lugar de entrada em comum com outras transições. Observe que a propriedade estática

pode ser facilmente verificada analizando a topologia da rede, enquanto que o caso dinâmico, depende de estados que podem ser dinamicamente alcançados, ou seja, do comportamento do sistema.

Redes Predicado/Transição (redes PrT)

Redes Predicado/Transição, também denominadas redes PrT no que segue-se, foram introduzidas no início da década de 80. Nesta seção apresentamos uma revisão dos conceitos fundamentais e características das redes PrT. Para o leitor interessado em um entendimento profundo das propriedades algébricas e lógicas das redes PrT, as seguintes referências são recomendadas [45, 47, 48, 57]. Assim como no caso dos sistemas de baixo-nível, um rede PrT possui uma estrutura - um grafo direcionado bipartido $(P, T; F)$ como na Definição 2.6. A principal diferença entre redes PrT e redes de baixo-nível é o fato de que as fichas não são mais *anônimas*, mas podem ser objetos estruturados com valores associados, e mais ainda, o disparo de transições pode ser controlado ou restrito pela imposição de condições relacionadas ao valor das fichas. Esta característica possibilita que o projetista possa representar ou modelar sistemas em um nível de abstração mais alto de que com os sistemas de baixo-nível.

Podemos agora formalmente introduzir o conceito de redes PrT assim como outras importantes definições.

Definição 2.11 *Rede Predicado/Transição*
Uma rede PrT é definida pela tupla $\mathcal{N} = (P, T; F, A, W, R, M_0)$ onde:

1. $(P, T; F)$ é uma estrutura de rede.

2. $A = (C, \Phi)$ é uma estrutura de suporte para $\mathcal{N}$, onde C é um conjunto finito de constantes e Φ é um conjunto de funções parcias definidas sobre C, i.e., para qualquer $f \in F$, $C \times \cdots \times C \rightarrow C$. Um *termo* é um dos seguintes: uma constante em C, a faixa de uma variável sobre C, ou $f(v_1, \cdots, v_m)$ onde f é uma função m-ária de símbolos em Φ e cada v_i $(1 \leq i \leq m)$ é um termo.

3. $W : F \rightarrow D^+_{fin}$ é um mapeamento de um conjunto de arcos F em um conjunto de rótulos, onde cada rótulo é um conjunto de tuplas e cada tupla é da forma $(d_1, \cdots, d_n)$ na qual cada d_i $(1 \leq i \leq n)$ é um termo (parêntesis são usualmente ignorados para tuplas com aridade[2] $D^+_{fin} = D^1 \cup \cdots \cup D^{max}$ é a união finita do

[2] Usamos o termo *aridade* como tradução para o termo em inglês *arity*.

conjunto de tuplas, e D_i^+ $(1 \leq i \leq max)$ representa o conjunto finito de tuplas com aridade i (max é "comprimento máximo" das tuplas). A aridade dos rótulos deve ser a mesma, e a aridade de um predicado é definida como sendo a aridade dos rótulos inscrevendo seu arco incidente.

4. $R : T \rightarrow RE$ é um mapeamento de um conjunto de transições T para um conjunto de expressões relacionais, onde uma expressão relacional é construída a partir de varios operadores e termos. Um expressão relacional vazia é também incluída em RE. Todas as variáveis ocorrendo em uma expressão relacional pertencem a um dos rótulos inscrevendo arcos incidentes com a transição contendo a expressão relacional.

5. $M_0 : P \rightarrow C^*_{fin}$ é o mapeamento de um conjunto de predicados P para um conjunto de fichas, e é denominada marcação inicial, onde $C^*_{fin} = C^0 \cup C^1 \cup \cdots \cup C^{max}$ é a união finita de um conjunto de tuplas constantes. Uma marcação M de $\mathcal{N}$ é um mapeamento $M : P \rightarrow C^*_{fin}$.

Mostramos na Figura 2.3 um rede PrT simples. Esta rede poderia representar um procedimento com parâmetros a e c, e se x>y a saída é o valor de y neste caso x=a e y=c. De acordo com a Definição 2.11 temos para esta rede PrT:

$P = \{P1, P2\}$; $T = \{t1\}$; $F = \{(P1, t1), (t1, P2)\}$; $A = (\{\text{a,c}\},\{\ \})$;
$W(P1, t1) = \{\text{x,y}\}$; $W(P2, t1) = \text{y}$; $R(t1) = \text{x} > \text{y}$;
$M_0(P1) = \{\text{a,c}\}$; $M_0(P2) = \{\ \}$;

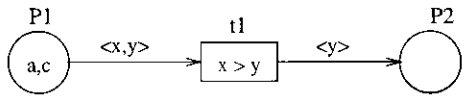

Figura 2.3: Uma rede predicado transição

Uma ficha $\tau = \langle a_1, a_2, \cdots, a_r \rangle$ em um lugar $p \in P$ denota o fato de que o predicado $P(x_1, x_2, \cdots, x_r)$, correspondente àquele lugar, para aquela instância particular da tupla de argumentos contida na ficha é verdadeiro.

Uma marcação M pode ser obtida a partir de outra M' pelo disparo de transições. De modo a introduzir a semântica para o disparo de transições, vamos primeiramente definir os conjuntos de entrada e saída para uma transição.

Definição 2.12 *Conjuntos de entrada/saída para uma transição*
Para cada transição $t \in T$, um *conjunto de entrada* $I(t)$ e um *conjunto de saída* $O(t)$ são introduzidos:

$$\begin{aligned} I(t) &= \{p \in P | (p,t) \in F\}, \\ O(t) &= \{p \in P | (t,p) \in F\}. \end{aligned} \tag{2.5}$$

$I(t)$ e $O(t)$ são respectivamente os conjuntos de lugares de *entrada* e *saída* de uma transição t.

Definição 2.13 *Conjuntos de entrada/saída para um lugar*
Para cada lugar $p \in P$, um *conjunto de entrada* $I(p)$ e um *conjunto de saída* $O(p)$ são introduzidos:

$$\begin{aligned} I(p) &= \{t \in T | (t,p) \in F\}, \\ O(p) &= \{t \in T | (p,t) \in F\}. \end{aligned} \tag{2.6}$$

$I(p)$ e $O(p)$ são respectivamente os conjuntos de transições de *entrada* e *saída* para o lugar p.

Com base nas Definições 2.12 e 2.13 podemos definir a regra de disparo para redes PrT, como segue-se.

Definição 2.14 *Regra de disparo*
Uma transição $t \in T$ está *habilitada* e pode disparar sempre que:

1. Cada lugar de entrada $p \in I(t)$ contém pelo menos tantas fichas quantas especificadas pelo rótulo (p,t).

2. Os valores das fichas ocorrendo nos lugares de entrada possuem valores que satisfazem a fórmula inscrita na transição, se houver alguma.

3. A capacidade de cada lugar de saída não é excedida pelo disparo da transição.

É importante notar que a regra de disparo na Definição 2.14 utiliza o conceito de *capacidade de um lugar*. A capacidade de um lugar define o número máximo de fichas que podem ser depositadas em um dado lugar. Note que o conceito de capacidade de lugar pode também ser aplicado a redes P/T.

Quando uma transição $t \in T$ está habilitada ela pode ser *disparada*, pela remoção do lugar de entrada $p \in I(t)$ do número de fichas especificadas no arco (p, t), e adicionando a cada lugar de saída $p' \in O(t)$ o número de fichas especificadas no rótulo do arco (t, p'). Deve-se notar que a ausência de fórmula inscrita à transição significa que o disparo depende somente do número de fichas nos lugares de entrada e da capacidade dos lugares de saída.

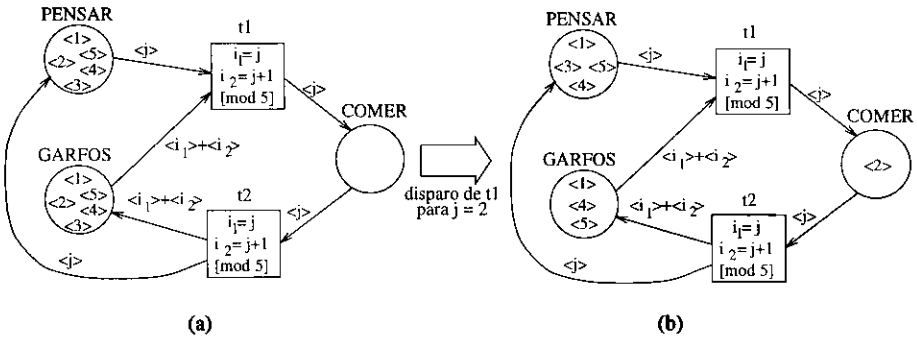

Figura 2.4: Rede PrT para o problema de *livre de bloqueio* dos cinco filósofos de Dijkstra

Com o objetivo de ilustrar a regra de disparo consideremos o exemplo apresentado em [51], o qual modela o bem conhecido problema dos cinco filósofos de Dijkstra com ausência de *bloqueio* [87]. Apresentamos na Figura 2.4 a rede PrT como solução para este problema antes do disparo da transição *t1* com $j = 2$. Na Figura 2.4 mod representa a operação *módulo*. Para esta rede PrT a variável j possui a tripla *FILÓSOFO* podendo assumir valores inteiros definidos no intervalo $[1, n]$; a variável i possui tipo *GARFO* e assume valores interiros definidos no intervalo $[1, n]$. Em (a) mostra-se a marcação inicial M_0 ; M_0 indica que todos os filósofos estão pensando e todos os garfos não estão sendo utilizados. Se a transição t_1 dispara para $j = 2$, e alcança-se a marcação M mostrada em (b); M indica que os filósofos 1,3 e 5 estão pensando e o filósofo 2 está comendo, e os garfos 1,4 e 5 não estão sendo utilizados. Observando, neste exemplo pode-se notar as vantagens de utilizar-se redes PrT na modelagem do problema ao invés de redes P/T, de fato a rede apresentada na Figura 2.4 é independente do número de filósofos, e então sua dimensão não cresce com o número de filósofos. Note-se que mesmo que as variáveis j e i na Figura 2.4 possuem a mesma faixa de possíveis valores, estas

possuem tipos diferentes, e portanto não podem ser confundidas.

De modo similar aos casos previamente introduzidos uma rede PrT pode ser representada por sua matriz de incidência.

Definição 2.15 *Matriz de incidência para redes PrT*
Dado que C é a matriz de incidência de uma rede PrT com n transições e m lugares, a linha i $(1 \leq i \leq m)$ de C corresponde ao lugar $p_i \in P$, e a coluna j $(1 \leq j \leq m)$ de C corresponde à transição $t_j \in T$, uma entrada c_{ij} é definida por:

$$c_{ij} = \begin{cases} -s & \text{se } s \text{ é o rótulo do arco } (p_i, t_j) \in F \\ s & \text{se } s \text{ é o rótulo do arco } (t_j, p_i) \in F \\ 0 & \text{para qualquer outro caso} \end{cases} \tag{2.7}$$

Na Figura 2.5 apresentamos a matriz de incidência para a rede Figura 2.4 como exemplo.

$$\begin{array}{c} \\ T \\ F \\ E \end{array} \begin{array}{c} \begin{array}{cc} \boldsymbol{t_1} & \boldsymbol{t_2} \end{array} \\ \begin{bmatrix} -\langle j \rangle & \langle j \rangle \\ -\langle i_1 \rangle - \langle i_2 \rangle & \langle i_1 \rangle + \langle i_2 \rangle \\ \langle j \rangle & -\langle j \rangle \end{bmatrix} \end{array}$$

Figura 2.5: Matriz de incidência para a rede da Figura 2.4

2.3 Modelagem de Sistemas Utilizando Sistemas de Rede

Nesta seção fazemos alguns comentários genéricos sobre a aplicação de sistemas de rede para a modelagem e análise sistemas complexos. A introdução das redes de alto-nível proporcionou a possibilidade de utilizar uma abordagem mais adequada para a modelagem de sistemas complexos, neste caso entende-se por complexo um sistema com um grande número de estados, mas em muitos casos não pode-se confiar somente nas abordagens estruturadas Disponíveis. Isto pode ser justificado pois, um sistema modelado por uma rede de alto-nível pode ser vista como o dobramento (*folding*) de um sistema de baixo-nível, e e em alguns casos, não é necessário utilizar redes de alto-nível. Portanto, mesmo considerando-se que o modelo é mais compacto, técnicas de estruturação e modularização mais poderosas devem ser introduzidas. A solução para este problema

é introduzir-se hierarquias e construções modulares no contexto de sistemas de redes. Para tanto, algumas metodologias e técnicas estão disponíveis para prover a possibilidade de construir o modelo do sistema de forma modular e hierárquica. Valette [108] introduz uma metodologia baseada em refinamentos sucessivos para ser aplicada na construção de redes de baixo-nível. Ele introduziu o conceito de *blocos bem formados*. A idéia básica é definir redes bem formadas com o comportamento desejado, p.e. vivacidade (*liveness*) ou limitabilidade (*boundness*), e então usa-las para construir uma rede mais complexa. Se a a rede é construída usando este tipo de blocos, as propriedades desejadas para o sistema podem ser preservadas, e em conseqüência a rede complexa resultante não precisa ser analisada. Murata [103] generalizou a metodologia introduzida por Vallete de modo a incluir também abstração. Infelizmente, esta abordagem pode somente ser aplicada para redes ou blocos com um par de transições para entrada e saída. Valette [2, 109] introduziu outra abordagem que possibilitava a fusão de lugares e transições, de modo que algumas propriedes possam ser preservadas. Para detalhes sobre estas técnicas o leitor pode referir-se a [8, 67, 81, 80].

Outras construções modulares [102] baseiam-se na idéia geral de construir-se uma rede modelando um sistema de forma modular, e deduzir propriedades do sistema somente pela análise de seus componentes menores. A razão para introduzir-se este tipo de abordagem é que de um modo geral, a composição de sub-redes genéricas não possibilita a preservação de determinadas propriedades (especialmente vivacidade) ao nível da rede global. Diferentemente das abordagens previamente introduzidas, esta não introduz restrições às redes a serem compostas. Por exemplo, a restrição de haver somente um par de transições de entrada e saída, é suficiente para estruturalmente restringir um meio (uma terceira rede), a qual é usada para compor duas outras redes.

Para redes de alto-nível existem algumas metodologias que permitem construções hierárquicas e modulares também. He and Lee [55] introduzem uma metodologia para construção de redes PrT hierárquicas. A metodologia baseia-se em técnicas de transformação e regras para aplicar estas técnicas. Transformações equivalentes são discutidas por Genrich em [46], basicamente elas baseiam-se em regras de redução, as quais serão introduzidas na Secção 2.4. Christenssen and Petrucci [25] assim como Jensen [60] apresentam metodologias modulares para a análise de redes de Petri Coloridas.

Broy e Stricher [16] combinam redes de alto-nível com modelos funcionais, de modo a tentar unificar e combinar diferentes modelos para sistemas distribuídos. A idéia básica é estudar fragmentos de redes de alto-nível, denominadas *componentes de rede*, onde cada rede pode incluir arcos sem alvos ou fontes definidos. Estes arcos são considerados

como entradas e saídas para o ambiente destes fragmentos de redes, de forma que eles podem ser livremente conectados a arcos de outras redes ou da mesma rede. Cheraibar [22] introduz o conceito de redes de Petri Coloridas reentrantes. Este modelo permite alguns tipos de composições com preservação de propriedades, como por exemplo escolha e composição seqüencial. Mostra-se também como uma rede pode ser substituída por outra reentrante. Uma rede reentrante é composta ou substituída por outra, mantendo-se mesmos lugares especias de interface e impondo algumas restrições estruturais e comportamentais à rede.

Battiston et al [9] definem uma nova classe de redes de alto-nível que utiliza técnicas de análise algébrica. O conceito de redes de Autômatos Superpostos 1-Seguros (*1-safe Superposed Automata nets (SA)*) é utilizado para obter o modelo de um sistema. Neste caso cada componente identificado do sistema é modelado por uma máquina de estados, portanto representando componentes seqüenciais. Composição é conseguida através da superposição de transições. Fichas fluindo na rede possuem duas partes, um nome, o qual não é alterado devido ao disparo de transições, e uma parte de dados, a qual pode ser alterada devido ao disparo das transições.

PROTOB foi introduzido em [7], e é um sistema **CASE**[3] juntamente com um conjunto integrado de ferramentas para especificação, modelagem, prototipagem e implementação de sistemas distribuídos, baseando-se em um paradigma de ciclo de vida operacional para o software. PROTOB suporta uma metodologia orientada a objetos similar ao HOOD - *Hierarchical Object Oriented Design* [98]- o qual divide o sistema em uma hierarquia de objetos para melhorar a compreenssão e simplificar sua modificação e o reuso de elementos do sistema. Cada objeto é representado por uma rede PROT, que é uma rede de Petri de alto-nível com a adição de um novo elemento denominado **PORTA** (*GATE*), o qual é utilizado para enviar fichas para outras redes PROT.

2.4 Métodos de Análise

Nesta seção introduziremos de modo informal técnicas e métodos de análise para sistemas de rede. Como advogado em [60], a maneira mais direta de análise é simulação, a qual em muitos aspectos é bastante similar ao teste e execução de programas.

Simulação é extremamente útil para o entendimento e depuraçao de um sistema de redes. Este aspecto é particularmente relevante durante a fase de concepção e validação prematura de um grande sistema complexo. Entretanto por meios de simulação, é

[3] CASE origina-se do inglês *Computer Aided Software Engineering.*

impossível obter-se um completa prova ou verificação das propriedades dinâmicas de um sistema, devido a complexidade espacial e temporal. Portanto, é muito importante vislumbrar métodos formais de análise (i.e., métodos que são baseados em técnicas de prova matemática).

Duas classes de propriedades podem ser verificadas ou analisadas para sistemas de redes, propriedades estáticas e propriedades dinâmicas. Nesta apresentação enfocaremos a análise de propriedades estáticas e dinâmicas para sistemas de redes de alto-nível, em especial redes PrT. No caso de redes de baixo-nível o leitor interessado pode consultar as referências [11, 14, 77, 87, 91].

Propriedades Estáticas

Propriedades estáticas ou estruturais podem ser derivadas da definição da rede em questão – sem considerar as seqüências de disparo ou ocorrência das transições. Propriedades estáticas são principalmente importantes para caracterizar redes com alguma tipo de propriedade especial.

Para redes de baixo-nível, tal como redes P/T, é possível definir diversas propriedades estáticas que possam ser verificadas. Murata [77] apresenta uma excelente introdução à estas propriedades estáticas. Entre as principais citam-se: vivacidade estrutural (*structural liveness*), controlabilidade, repetitividade, e consistência. Jensen [60] introduziu o conceito de uniformidade e máquina de estado para redes de Petri Coloridas.

Propriedades Dinâmicas

Propriedades dinâmicas ou comportamentais caracterizam o comportamento de redes individuais, p.e., se é possível ou não alcançar uma marcação na qual nenhuma transição estaria habilitada. A verificação de propriedades dinâmicas pode ser extremamente difícil quando nenhum método formal é disponível, isto pois o número de possíveis combinações de casos a serem simulados pode ser proibitivo. Portanto, é muito importante definirem-se métodos formais para a análise dos vários tipos de sistemas de redes.

Exemplos de propriedades comportamentais são: alcançabilidade, limitabilidade, vivacidade, reversibilidade, estados originais (*home states*), cobertura (*coverability*), persistência, distância sincrônica (*synchronic distance*), e justiça. Todas estas propriedades dinâmicas são discutidas em [77]. Métodos de análise para sistemas de rede podem ser classificados nos seguintes três grupos:

1. invariantes de lugar e transição

2. técnicas de redução e decomposição, e

3. método da árvore de alcaçabilidade (cobertura).

No que segue-se discutiremos informalmente as principais técnicas de análise para redes de Petri.

2.4.1 Invariantes de Lugar e Transição

O método do invariante é conhecido por pelo menos duas vantagens: primeiramente, a análise pode ser executada em sub-redes locais ignorando-se como o sistema global comporta-se; segundo, este método é aplicável tanto para redes de baixo-nível como redes de alto-nível.

A ídeia básica é analisar o comportamento dinâmico de um sistema através de equações lineares. Entretanto, como enfatizado em [77], a solução destas equações é um tanto quanto limitada. Isto deve-se a característica não determinística do comportamento de modelos de sistemas de redes, e devido a restrição de que as soluções devem pertencer ao conjunto dos inteiros não negativos, no caso de redes de baixo-nível. A seguir introduziremos somente os aspectos conceituais da análise de invariantes. Para detalhes matemáticos refira-se a [77].

No caso da análise de invariantes definem-se equações de estado para o sistema. Uma marcação M_k é escrita como um vetor coluna m. A $j^{ésima}$ entrada de M_k denota o número de fichas no lugar j imediatamente após a $k^{ésima}$ seqüência de disparo. A $k^{ésima}$ seqüência de disparo pode ser vista como um vetor de controle u_k. O vetor de controle u_k é um vetor coluna $n \times 1$ com $n-1$ zeros e uma entrada não nula, um 1 na $j^{ésima}$ posição indica que a transição j dispara no $k^{ésimo}$ disparo. Uma vez que a $i^{ésima}$ linha na matriz de incidência C denota a mudança de uma marcação como resultado do disparo da transição i, podemos escrever a seguinte equação de estado:

$$M_k = M_{k-1} + C^T u_k, \quad k = 1, 2, \cdots. \tag{2.8}$$

Como detalhado em [77], duas equações podem ser derivadas da Equação 2.8. Uma é denominada de *P-invariantes* para invariantes de lugar, e a segunda é denominada *T-invariantes* para invariantes de transição.

Definição 2.16 *Vetor de disparo*
Um vetor de disparo x é um vetor coluna $n \times 1$ de inteiros não negativos.

onde a $i^{ésima}$ entrada de x denota o número de vezes que a transição i deve disparar para transformar uma marcação M_0 para M_d.

Definição 2.17 *P-invariantes*
Dado que C é a matriz de incidência para um sistema de redes, e dado que x é um vetor de disparo, um P-invariante é uma solução inteira para o sistema de equações homogêneas:

$$C^T x = 0 \tag{2.9}$$

Definição 2.18 *T-invariantes*
Dado que C é a matriz de incidência para um sistema de redes, e dado que x é um vetor de disparo, um T-invariante é uma solução inteira para o sistema de equações homogêneas:

$$Cx = 0 \tag{2.10}$$

Informalmente um P-invariante corresponde a uma seqüência de disparo que não altera a soma das fichas nos lugares, e um T-invariante corresponde a uma seqüência de disparo que não altera a marcação da rede. A análise através de invariantes é um método bastante poderoso tanto para executar análise estrutural como comportamental [75, 77].

No caso de redes de alto-nível surgem algumas dificuldades adicionais para aplicar-se a análise de invariantes. No caso de redes PrT temos que tratar com a substituição de variáveis e outros formalismos relacionados à tuplas. De qualquer modo, é póssivel aplicar-se a análise de invariantes para redes PrT, Genrich [47] discute em detalhes as bases matemáticas da análise de invariantes para redes PrT. Para aplicações práticas veja [52, 78, 86] no caso de redes PrT, e [59] no caso de redes de Petri Coloridas.

2.4.2 Redução

Para simplificar a análise de grandes sistemas de redes, é freqüentemente necessário reduzir o modelo para um mais simples. Deve-se notar que esta redução deve garantir a preservação de propriedades. Existem diversas diferentes técnicas para transformar ou reduzir um sistema de redes.

A idéia básica por tráz destas transformações está em escolher um ou mais tipos de propriedades que queremos investigar (p.e. vivacidade ou limitabilidade). Então, definimos um conjunto de regras para redução, que quando aplicadas podem simplificar o sistema de rede – sem alterar as propriedades que estão sendo investigadas. Usualmente, as regras são locais, no sentido de que cada uma delas permite que uma sub-rede seja substituída por uma outra mais simples. Está além do escopo desta tese discutir todas as regras de redução disponíveis para redes de baixo-nível, mas o leitor pode referir-se a [67, 77].

Para redes de alto-nível existem também várias técnicas de redução disponíveis. As referências [46, 55], entre outras discutem em detalhes a aplicação de técnicas de redução para redes de Petri Coloridas e redes PrT respectivamente. Vamos discutir por meio de um exemplo a aplicação de redução para redes PrT. Uma regra típica é esboçada na Figura 2.6[4].

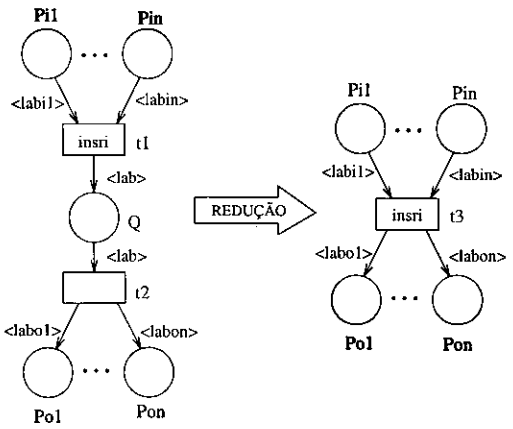

Figura 2.6: Regra de redução simples, preservando diferentes propriedades

No exemplo, substitui-se a sub-rede que possui um lugar Q, duas transições *t1* e *t2*, e dois arcos idênticos com o rótulo $\langle lab \rangle$, por uma sub-rede com somente uma transição *t3*. Especifica-se também como a nova sub-rede é inserida na rede original. No exemplo, *t3* terá um arco de entrada para cada um de *t1* e um arco de saída para cada arco de saída de *t2*. Cada novo arco possui o mesmo rótulo e o mesmo lugar correspondente ao lugar removido. A inscrição (ou expressão) inscrita em *t3* é idêntica àquela de *t1*. Finalmente, pode-se especificar que a regra somente pode ser aplicada quando certos pré-requisitos forem satisfeitos. No exemplo, requer-se que Q, *t1*, *t2* tenham somente os arcos mostrados. Q deve possuir uma marcação inicial vazia e a inscrição em *t2*

[4] Este exemplo é adaptado de um introduzido por Jensen [59] para redes de Petri Coloridas.

deve ser verdadeira, ou vazia. Além disto, o rótulo $\langle lab \rangle$ deve possuir determinadas características particulares [55].

Após definir-se um conjunto de regras de redução é necessário mostrar que este conjunto é consistente. Isto significa que devemos provar que as regras nunca alteram o conjunto de propriedades que estamos investigando. Finalmente, podemos então aplicar estas regras à rede que desejamos analisar. Esta redução resulta em uma rede que mantém as mesmas propriedades da rede original. Portanto, podemos investigar a rede reduzida ao invés da rede original. Um conjunto de regras de redução deve ainda ser poderoso, possibilitando uma redução significativa, isto é, capaz de produzir uma rede que é mais simples de ser investigada que o rede original.

Deve-se notar que a definição de um conjunto de regras de redução e sua validação deve ser executada uma só vez. Isto significa que uma nova regra de redução e a investigação da rede reduzida deve ser executada cada vez que uma rede tenha que ser analisada. A validação é freqüentemente uma tarefa complexa e monótona, porque devem ser consideradas todas as possíveis reduções que podem ser aplicadas à uma rede. Entretanto, note-se que o usuário pode aplicar um conjunto existente de regras sem conhecimento de como é executada sua validação.

Infelizmente, os resultados obtidos por muitos métodos de redução são não construtivos – no sentido de que a ausência de uma determinada propriedade na rede reduzida não diz muita coisa a respeito de porque a propriedade não está presente na rede original.

2.4.3 Árvore the Alcançabilidade/Cobertura

Para a descrição da árvore de alcançabilidade/cobertura tomaremos a definição de Murata [77]. Dada um sistema de rede $\mathcal{N}$, a partir da marcação inicial M_0, podemos obter tantas marcações quantas forem as transições habilitadas. A partir de cada nova marcação podemos então alcançar novas marcações. Este processo resulta em uma árvore de marcações. Para esta árvore, nós representam marcações geradas a partir de M_0 (a raiz da árvore) e seus sucessores, e cada arco da árvore representa o disparo de uma transição, o qual transforma uma marcação em outra. Entretanto, a representação em árvore crescerá indefinidamente no caso da rede não ser limitada. De modo a manter a árvore finita, introduz-se um símbolo especial ω, o qual pode ser considerado como *infinito*. Para redes de baixo-nível este símbolo apresenta a propriedade de que para cada inteiro não negativo n $\omega > n$, $\omega \pm n$ e $\omega \geq \omega$. A árvore de cobertura para um sistema de rede $\mathcal{N}$ e uma marcação inicial M_0, pode ser construída aplicando-se o algoritmo apresentado em [77].

No caso de um sistema limitado, a árvore de cobertura é denominada árvore de alcançabilidade, uma vez que esta contém todas as possíveis marcações alcaçáveis. Neste caso todos os problemas de análise podem ser resolvidos pela análise da árvore de alcançabilidade. A desvantagem deste método reside no fato de ser um método exaustivo. Entretanto, de modo geral, os problemas de alcançabilidade e vivacidade não podem ser resolvidos somente com a árvore de cobertura, isto devido a introdução do símbolo ω.

Para um sistema de rede, o grafo de cobertura é definido pelo grafo direto rotulado $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Onde o conjunto de nós $\mathcal{V}$, é o conjunto de de todos os nós rotulados distintos na árvore de cobertura, e o conjunto de arcos $\mathcal{E}$, é o conjunto de arcos rotulados com uma única transição t_k, representando todos os possíveis disparos únicos de transições, de forma que $M_i[t_k\rangle M_j$, onde M_i e M_j estão em $\mathcal{V}$. Por exemplo, para a rede mostrada na Figura 2.7 a árvore de cobertura e o grafo de cobertura são mostrados na Figura 2.8 (a) e (b), respectivamente.

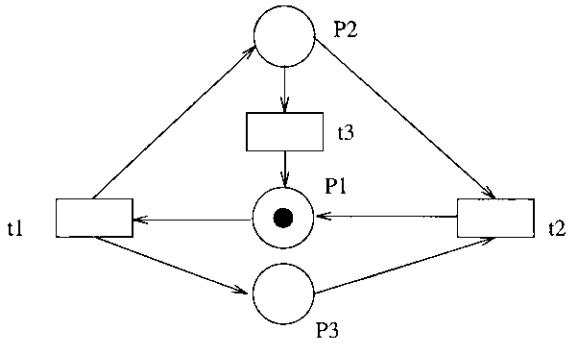

Figura 2.7: Uma rede P/T

Para redes de Petri Coloridas uma discussão detalhada a respeito da construção da árvore de alcaçabilidade e/ou cobertura pode ser encontrada em [56]. No caso de redes PrT uma discussão profunda sobre árvore de alcaçabilidade e/ou cobertura pode ser encontrada em [71]. Rauhamaa [90] apresenta uma coletânea de métodos eficientes para análise de alcançabilidade para redes PrT.

2.5 Lógica Temporal Ramificada no Tempo

Nesta seção introduzimos os conceitos de *Computation Tree Logic*, CTL, [26]. Antes de formalmente introduzirmos CTL, discutiremos os conceitos básicos relacionados à lógica temporal como discutidos por Emerson em [40]. Discutiremos também aspectos relacionados com *verificação de modelo* (*model checking*) para CTL.

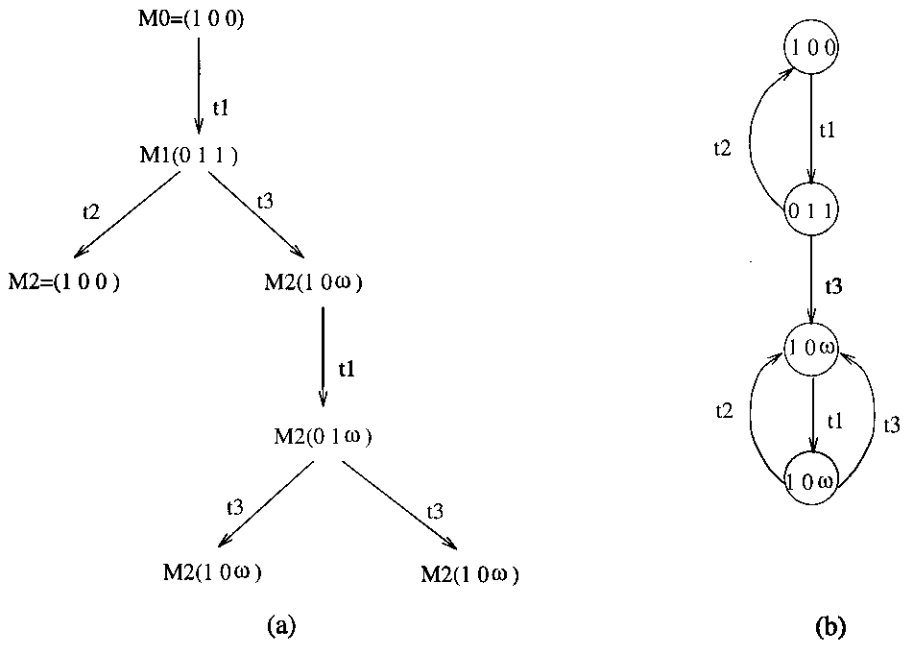

Figura 2.8: (a) Árvore de cobertura e (b) Grafo de cobertura para a rede da Figura 2.7

2.5.1 Introdução a CTL

Lógica temporal prove um sistema formal para descrever ocorrências de eventos no tempo. Ela pertence à classe de lógicas modais que foram originariamente introduzidas para tratar com diferentes "modos" da verdade. Uma asserção p pode ser falsa considerando-se circunstâncias atuais, mas a asserção *possivelmente* p será verdadeira se existirem outras circunstâncias nas quais p é verdadeira. Em um sistema de lógica temporal, modalidades são interpretadas temporalmente, de modo a descrever como valores verdadeiros de asserções podem variar com o tempo. Tipicamente modalidades temporais incluem *alguma vez* p, o que é verdadeiro no instante presente de tempo de existe um momento futuro no qual p torna-se verdadeira e *sempre* q a qual é verdadeira se q é verdadeira em todos os momentos futuros.

Um sistema de lógica temporal especifica a sintaxe de fórmulas válidas, e a semântica para interpreta-las. A lógica temporal que discutiremos é lógica proposicional ordinária com conectivos temporais para implicitamente lidar com tempo. A semântica de lógica temporal pode ser representada por grafos direcionados, e são denominados estruturas. Intuitivamente, os *estados* de uma estrutura (vértices do grafo) denotam possíveis mundos ou instantes de tempo, e são rotulados com proposições atômicas que são verdadeiras para o estado. Os arcos do grafo denotam a passagem do tempo: um arco de um estado s para um estado t indica que o mundo t é um possível sucessor do mundo

s. Portanto, cada caminho saindo de um estado denota possíveis futuros para o estado. O sistema de lógica temporal é dito *linear no tempo* se as estruturas sobre as quais as formulas são interpretadas são cadeias finitas unidirecionais, i.e., cada estado possui um único futuro, e *alternativo no tempo* se estruturas são grafos arbitrários (sujeitos à restrição de que cada estado tenha pelo menos um sucessor, i.e., cada mundo tem pelo menos um futuro). Lógica temporal linear no tempo é extensivamente discutida em [74]. No caso de lógica temporal alternativa no tempo (*branching time temporal logics*) o tempo é visto como uma ordenação parcial de uma coleção de instantes discretos no tempo, possibilitando a especificação de diversos futuros alternativos em qualquer instante de tempo. No que segue discutiremos a lógica CTL que é um tipo de lógica temporal alternativa no tempo.

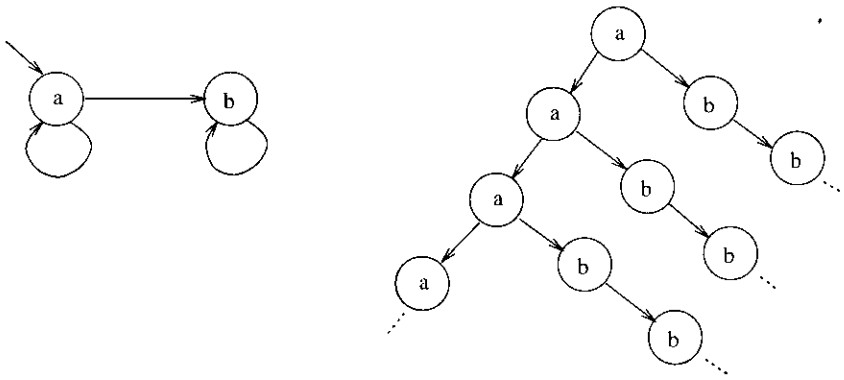

Figura 2.9: Grafo de transição de estado e sua respectiva árvore computacional

Para ilustrar a aplicação de CTL, considere que alguém deseja raciocinar sobre um sistema de transições M. Se algum estado de M é selecionado como *estado inicial*, então M pode ser conceitualmente desmembrado em uma árvore infinita, contendo este estado inicial como raiz (no lado esquerdo da Figura 2.9 temos um sistema de transições arbitrário, e do lado direito temos sua árvore computacional). Uma vez que caminhos na árvore representam possíveis comportamentos ou computações do sistema de transições, a árvore infinita obtida é denominada *árvore computacional* de M. CTL contém operadores especiais para descrever árvores conputacionais.

Fórmulas em CTL são classificadas como sendo fórmulas de estado (aquelas verdadeiras para estados da estrutura) ou fórmulas de caminho (aquelas verdadeiras ao longo de um caminho infinito da estrutura). Duas modalidades de caminho são definidas, **A** e **E**, as quais podem ser prefixadas à uma fórmula de caminho p, resultando em uma fórmula de estado. Intuitivamente a fórmula de estado $\mathbf{A}p$ é verdadeira no estado p se ela é verdaeira para todos os caminhos iniciando no estado, e $\mathbf{E}p$ é verdadeira para al-

gum caminho iniciando no estado. As modallidades para construir fórmulas de caminho são as mesmas do caso linear [74]: **G**(sempre), **F**(alguma vez), **X**(próximo instante), **U**(até). Estas modalidades descrevem a ordenação dos eventos no tempo, ao longo de um caminho computacional e possuem o seguinte significado intuitivo:

1. $\mathbf{G}\varphi$ ("φ válida globalmente") significa que φ é verdadeira em qualquer lugar ao longo do caminho.

2. $\mathbf{F}\varphi$ ("φ válida alguma vez no futuro") é verdadeira para um caminho se existir um estado no caminho para o qual a fórmula φ é verdadeira.

3. $\mathbf{X}\varphi$ ("φ válida na próxima vez") significa que φ é verdadeira no segundo estado do caminho, i.e., o estado imediatamente seguinte ao presente estado.

4. $\varphi\,\mathbf{U}\,\psi$ ("φ válida até que ψ seja válida") significa que existe algum estado no caminho para o qual ψ é verdadeira naquele estado, e para todos os estados prévios φ é verdaeira.

Cada fórmula na lógica pode ser verdadeira ou falsa em um dado estado. Uma proposição atômica é verdadeira se a variável de estado corespondente à proposição é verdadeira no estado. A validade de uma fórmula é construída à partir de conectivos booleanos dependendo das sub-fórmulas da forma usual. Uma fórmula cujo operador de alto nível é um operador temporal com quantificador de caminho universal (existencial) é verdadeira sempre que todos os caminhos (algum caminho) iniciando no estado possuirem a propiedade requerida pelo operador de modalidade.

Formalmente CTL é definida para um sistema de transições, tal qual uma máquina de estado finita. Dado que $M = \langle S, I, R\rangle$ é um sistema de transições. Um *caminho* em M é uma seqüência infinita de estados $\pi = s_o s_1 s_2 \cdots$ tal que para todo $i \in \mathbb{N}$, $R(s_i, s_{i+1})$. Escrevemos $M, s \models \varphi$ para indicar que a fórmula φ é verdadeira no estado s de M. A relação é indutivamente definida por:

1. Se φ é a proposição atômica correspondente à variável de estado v, então $s \models \varphi$ se e somente se $v \in s$.

2. $s \models \neg\varphi$ se e somente se não é o caso que $s \models \varphi$. $s \models \varphi \wedge \psi$ se somente se $s \models \varphi$ e $s \models \psi$. Outros conectivos proposicionas são manipulados de forma similar.

3. $s \models \mathbf{EX}\varphi$ se e somente se existe um caminho $\pi = s_o s_1 s_2 \cdots$ iniciando em $s = s_0$, tal que $s \models \varphi$.

4. $s \models \mathbf{EG}\varphi$ se e somente se existe um caminho π iniciando em s o qual para todo estado s' em π, $s' \models \varphi$.

5. $s \models \mathbf{E}(\varphi \,\mathbf{U}\, \psi)$ se e somente se existe um caminho $\pi = s_o s_1 s_2 \cdots$ iniciando em $s = s_0$ e algum $i \geq 0$ tal que $s_i \models \psi$ e para todo $j < i$, $s_j \models \varphi$.

A semântica dos operadores com quantificadores de caminho universais pode ser definida em termos dos acima apresentados. Por exemplo considere que, $\mathbf{AG}\varphi$ é equivalente a $\neg\mathbf{E}(\mathit{true}\,\mathbf{U}\,\neg\varphi)$ e $\mathbf{A}(\varphi\,\mathbf{U}\,\psi)$ é equivalente a $\neg\mathbf{E}(\neg\varphi\,\mathbf{U}\,(\neg\psi \wedge \neg\varphi)) \wedge \neg\mathbf{EG}\neg\psi$. Finalmente, podemos escrever que $M \models \varphi$ para indicar que todo estado inicial de M satisfaz a fórmula φ.

2.6 Verificação de Modelo e Verificação Modular Baseada em CTL

No caso de CTL foram desenvolvidos procedimentos para eficientemente verificar um sistema de transição de estados [27, 28]. Entretanto estes procedimentos sofrem com o problema da explosão do espaço de estados. Uma abordagem para atacar este problema é adotar *verificação simbólica de modelo.*

Existem diversas lógicas, principalmente modais, que podem ser aplicadas para verificação simbólica de modelo. Burch et al introduziram uma abordagem genérica na qual todas as outras baseiam-se [17], e pode manipular com mais de 10^{20} estados. Entretanto existem dois pontos negativos principais. O espaço de estado simbólico parece ser extremamente complicado mesmo para exemplos bastante simples, como unidades lógicas e aritméticas com operador *e* e *ou exclusivo*. O segundo ponto negativo reside no fato de que alguém pode argumentar que muitos sistemas distribuídos muito provavelmente são bem mais complexos que operações de uma unidade lógica e aritmética, e que na maioria dos casos práticos a verificação simbólica do modelo é impraticável do ponto de vista computacional. Ainda mais, acreditamos que a verificação simbólica não pode suplantar os métodos convencionias para sistemas de rede. Principalmente considerando-se que os eventos presentes em um sistema distribuído não apresentam propriedades tão convenientes como operações de adição e ou exclusivo apresentam, como apresentado no exemplo introduzido em [73].

De modo a tratar com o tipo de propriedades presentes em um sistema distribuído, um método óbvio para tentar evitar a explosão do espaço de estados é tirar vantagem

da decomposição natural que estes sistemas apresentam. O objetivo é verificar propriedades de componentes individuais, inferir se estas propriedades são válidas para o sistema como um todo, e usa-las para deduzir propriedades adicionais do sistema. Ainda mais, quando verificando propriedades de componentes, pode-se tornar necessário fazer suposições sobre o comportamento do ambiente com o qual o componente interage ou interagirá. Esta abordagem foi inicialmente utilizada para lógica temporal por Pnueli [88], utilizando um paradigma denominado *assume/garante.*

Para reduzir a complexidade da verificação automática de modelos duas abordagens podem ser consideradas. A primeira inclui métodos para reduzir o grafo global de estados, ou expandir somente as partes necessárias do grafo de estado global. A segunda classe é denominada composicional; neste caso propriedades de componentes individuais são verificadas, e propriedades do sistema como um todo são deduzidas destas. Entre outros resultados mais significativos para verificação composicional autómatica para lógica temporal estão aqueles apresentados por Josko [29, 62, 63, 64] e Long [53, 54].

2.7 Relacionamento entre Lógica Temporal e Redes de Petri

Lógica, e neste contexto lógica temporal, pode ser combinada com sistemas de rede com o objetivo de análise de duas formas diferentes, denominadas abordagem axiomática e abordagem de verificação de modelo. A diferença reside na quantidade de lógica utilizada. No método axiomático o problema de análise de redes e inteiramente reduzido a uma questão lógica. Verificação de modelo é um tipo de método híbrido, combinando lógica com a análise tradicional para sistemas de rede.

Na abordagem axiomática presume-se que a rede e suas proriedades podem ser expressadas em lógica. A questão a ser respondida neste caso é se as propriedades sendo verificadas possuem ou não dedução à partir da descrição da rede. Esta verificaçao pode ser conseguida aplicando-se um provador de teoremas.

Um verificador de modelo verifica se uma dada fórmula é verdadeira ou falsa para dado modelo. No caso da análise de redes o modelo pode ser um grafo de alcançabilidade ou algum derivado, e fórmulas expressam propriedades dinâmicas da rede, como por exemplo a ausência de *bloqueio.*

Nosso interesse está principalmente no relacionamento entre modelos de rede e lógica temporal sob o ponto de vista de verificação de modelo. Podemos ver a aplicação de verificação de modelo para sistemas de rede como um meio de prover consultas para fazer perguntas sobre o grafo de alcançabilidade da rede. Desta forma a lógica utilizada no verificador de modelo pode ser vista como uma linguagem de consultas.

Tomemos um exemplo derivado de [106] para informalmente introduzir os conceitos do uso de lógica temporal como linguagem de consulta. Considere o sistema elementar (SE) mostrado na Figura 2.10, representando um protocolo de comunicação introduzido por Diaz in [39]. O grafo de alcançabilidade (conhecido para sistemas elementares como *grafo de casos*) possui 62 casos[5]. Podemos dizer que a verificação manual de que se determinada propriedade é válida ou não está muito longe de ser uma tarefa simples. Este é o caso quando temos disponível apenas uma representação textual do grafo. Seria muito mais simples elaborar questões relevantes a respeito do grafo utilizando lógica temporal como linguagem de consulta.

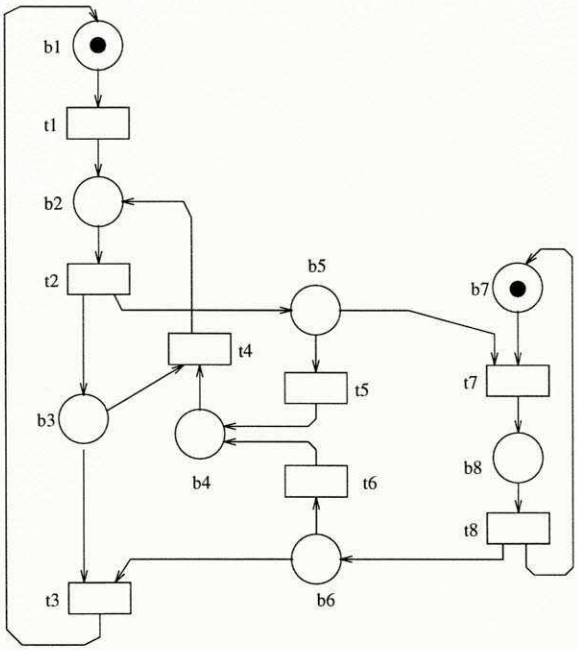

Figura 2.10: Um sistema elementar representando um protocolo de comunicações

Consideremos a seguinte propriedade, que é denominada *comportamento repetitivo*. Informalmente ela diz que:

para todos os caminhos sempre que b1 e b7 são verdadeiros, então sempre é o caso que b1 e b7 serão verdadeiros novamente

Esta propriedade pode ser expressada em CTL por

[5] Apresentaremos uma representação mais compacta do problema, possuindo um espaço de estado mais reduzido, uma vez que o nosso objetivo é introduzir o conceito de verificação de modelo para sistemas de rede.

$$\mathbf{AG}((b_1 \wedge b_7) \rightarrow \mathbf{AGEF}(b_1 \wedge b_7)) \tag{2.11}$$

De modo a aplicar a verificação de modelo baseada em CTL para verificar se a fórmula descrita em 2.11 é verdadeira, construímos o grafo de alcançabilidade para a rede da Figura 2.10, com proposições associadas a cada lugar. Neste caso o conjunto de proposições é $\{b1, b2, b3, b4, b5, b6, b7, b8\}$. Então, verificamos se o grafo de alcançabilidade é um modelo para a fórmula. O grafo de alcaçabilidade para a rede da Figura 2.10 é mostrada na Figura 2.11. Do grafo de alcaçabilidade está claro que existe um caminho partindo do estado $\{b1, 0, 0, 0, 0, 0, b7, 0\}$ para si mesmo. Entretanto, para sistemas mais complexos esta verificação não é tão óbvia. Introduziremos a seguir informalmente como esta verificação pode ser executada. O procedimento detalhado para esta verificação é apresentado no Capítulo 4.

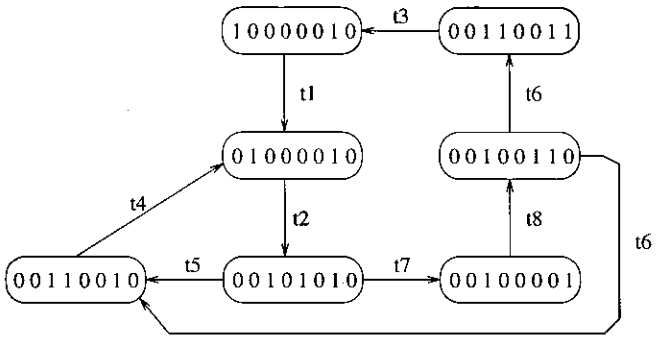

Figura 2.11: Grafo de alcaçabilidade para o sistema elementar mostrado na Figura 2.10

Para verificar se uma fórmula em lógica temporal é válida, por exemplo aquela representada pela Equação 2.11, computamos a a transição fechada (*transitive closure*) da matriz de incidência do grafo de alcançabilidade mostrado na Figura 2.11. Com base na matriz de transição fechada é possível verificar se uma fórmula é válida ou não. Do grafo de alcaçabilidade podemos, por exemplo, verificar a validade da Equação 2.11. Observe os seguintes disparos de transições: $t1 \rightarrow t2 \rightarrow t7 \rightarrow t8 \rightarrow t6 \rightarrow t3$, $t2 \rightarrow t5 \rightarrow t4$ e $t2 \rightarrow t7 \rightarrow t8 \rightarrow t6$. A partir deste disparos de transição é possível observar que uma vez que o estado $\{1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\}$ é alcançado, isto é a expressão $b1 \wedge b_7$ é verdadeira, entao é sempre possível encontrar um caminho no grafo de alcaçabilidade que leve a este mesmo estado. Esta informação pode ser facilmente extraída da transição fechada RG^* do grafo de alcaçabilidade RG. O maior problema desta abordagem á a explosão de estados. Para tratar com este problema Burch et al introduziram a aplicação de *diagramas de decisão binários*, e mais tarde Macmillan [73] aplicou este conceito para definir uma abordagem

para *verificação simbólica de modelo*. A idéia básica é utilizar uma representação mais compacta para o grafo de alcaçabilidade, por meio de um diagrama de decisão binário. Como discutido anteriormente, é nossa opinião que a aplicabilidade destes métodos parece estar restrita a casos muito simples. Ainda mais, a abordagem é extremamente dependente da ordenação dos símbolos (relacionados às proposições), de modo a obter uma representação por um diagrama de decisão binário ótimamente compacto para o grafo de alcaçabilidade. Na verdade, não é simples definir de forma ótima a ordenação dos eventos em um sistema distribuído. De qualquer forma, o leitor interessado pode referir-se a [106] para uma discussão teórica sobre o relacionamento entre sistemas de rede e lógica temporal.

2.8 Aspectos de Tolerância a Falhas

A crescente popularidade dos sistemas de computação tem possibilitado sua aplicação para a solução de uma grande variedade de problemas que englobam desde simples aplicações até tarefas complexas e críticas como controle de tráfego aéreo. Assim como qualquer outro tipo de sistema, os sistemas de computação estão sujeitos a falhas e, dependendo da natureza do sistema, as conseqüências introduzidas pelas falhas podem variar de uma simples inconveniência até uma catástrofe. Neste último caso aspectos relacionados com o grau de confiança na operação correta do sistema e sua capacidade em tolerar falhas ganharam uma grande importância nos últimos anos. Desta forma, considerando o atual estágio da tecnologia, os aspectos relacionados com a tolerância a falhas em sistemas de computação são essenciais.

Um sistema é tolerante a falhas se ele mantém sua capacidade funcional e desempenho completo quando da presença de falhas. Para sistemas complexos de *hardware*, metodologias bem definidas relacionadas com a confiabilidade, segurança e tolerância a falhas foram desenvolvidas [97]. No entanto, com a proliferação dos sistemas de computação e consequentemente a crescente utilização de aplicativos de *software*, o interesse no desenvolvimento de metodologias de tolerância a falhas para *software* vem crescendo significativamente.

No caso de sistemas de software as metodologias definidas para os sistemas de hardware não podem ser facilmente aplicadas principalmente por dois motivos. Primeiramente, é muito difícil corrigir automaticamente erros cometidos por programadores [1]. Em segundo lugar, é quase impossível prever todas as mudanças de comportamento que possam ocorrer no ambiente com o qual o software interage. Estas mudanças de comportamento que podem ser causadas por exemplo pela substituição de uma interface com

especificações ligeiramente diferentes das originais, podem causar um comportamento incorreto na operação do software do sistema.

Redes de Petri tem sido eficientemente aplicadas para analizar e introduzir propriedades e esquemas de tolerância a falhas [10, 68, 70]. Além de ser uma ferramenta formal, as redes de Petri permitem modelar de forma unificada hardware, software e aspectos do comportamento humano. Desta forma possibilitando utilizar um único formalismo para combinar estes três componentes em um único modelo.

Para sistemas de software complexos, deve-se considerar uma abordagem modular ou composicional que permita a introdução sistemática de propriedades de tolerância a falhas no projeto de um componente. No caso dos sistemas de software atuais, esta abordagem deve também incluir a habilidade de acomodar o projeto orientado a objetos [21]. Isto é devido a crescente popularidade da metodologia de projeto orientado a objetos no desenvolvimento de sistemas de controle em tempo real visando a reutilização de componentes de software.

Confiabilidade de um sistema é tradicionalmente definida como a probabilidade que o sistema desempenhará de forma correta suas funções por um período determinado de tempo considerando-se um conjunto de restrições ambientais [1, 68].

Segurança de um sistema é definida como a probabilidade de que o sistema ou desempenhará suas funções corretamente ou as descontinuará, de modo que operações sendo executadas em outros sistemas associados a ele não sejam afetadas em caso de falha [61].

De acordo com Avizienis e Laprie [5] os termos *falha*, *erro* e *falta* podem ser definidos por:

> Uma *falta* ocorre no sistema quando os serviços sendo executados desviam-se de sua especificação, onde a *especificação* do serviço é uma descrição do que se espera do serviço. A falta ocorre pois o sistema está em um estado de erro: um *erro* é a parte do estado do sistema que é sujeito a faltas. A causa de um erro é uma *falha*. Um erro é então a manifestação de uma falha no sistema, e uma falta é o efeito de um erro no serviço.

No caso de sistemas dedicados, por exemplo sistemas de controle aéreo, o controle do sistema não pode ser interrompido abruptamente quando uma falha ocorre [69, 70]. Portanto, devem ser embutidas no sistema capacidades de resposta a falhas de *hardware*, falhas de *software* e talvez condições ambientais não esperadas. Estas respostas podem ser classificadas como:

tolerância a falhas é definida como a capacidade de um sistema de software continuar desempenhando completamente suas funções mesmo na presença de falhas de operação.

seguro significa que o sistema tenta limitar o total do prejuízo causado por uma falha. Neste caso, nenhuma tentativa é feita para satisfazer as especificações funcionais, exceto quando for necessário garantir segurança.

falho suave é o sistema que continua operando, mas provendo índices de desempenho degradados ou capacidades funcionais reduzidas, até a remoção da falha.

vital significa que o sistema garante que algumas funcionalidades e respectivos desempenhos são cumpridas durante um determinado intervalo de tempo, depois do qual o sistema pode ser totalmente desabilitado em um estado seguro.

Um sistema é tolerante a falha quanto ele pode prevenir que falhas causem faltas [1]. As principais técnicas para garantir tolerância a falhas para *software* são:

Revisão e teste do software. Esta é a metodologia tradicional de engenharia de software e não é considerada neste artigo, pois assume-se que erros de programadores e mudanças no ambiente não podem ser totalmente evitadas ou previstas.

Verificação formal do software. Baseado em algum método formal de verificação, o projetista verifica se o modelo satisfaz as propriedades definidas para o sistema durante a análise.

Implementar n-versões do software. Uma ou mais versões do *software* são produzidas se a confiança na primeira versão é pequena. Neste caso o projetista pode tanto descartar a versão original, usar todas as versões e votar os resultados, ou ainda adotar um teste de aceitação em tempo de execução que determine em qual versão confiar.

Deve-se ainda considerar quais propriedades de tolerância a falhas um componente de *software* deve possuir. Um componente é somente o formalismo para um módulo. Durante a fase de projeto, um componente pode ser tornado *auto-protegido* ou *auto-verificante*. Um componente é dito auto-protegido se ele pode se proteger de interferência em seu comportamento, devido a falhas em outros componentes, ou clientes.

Se ocorrer um erro em um cliente, o cliente deve falhar ou pelo menos tratar seu erro, desta forma evitando interferência no comportamento de outros componentes que interagem com ele. Quando ocorre uma ação auto-protetora, o componente que detecta o erro deve recusar interferência em seu comportamento. Este tipo de ação pode ser implementada através do envio de uma mensagem ao cliente, indicando que a ação não será executada como solicitado, caso contrário um estado de erro será alcançado. Um componente é dito auto-verificante se ele pode detectar e sinalizar seu erro. Desta forma, sua falha não interferirá no comportamento de outros componentes que fazem uso de seus serviços ou em componentes que o componente faltoso interage. Para tanto o componente faltoso deve gerar sinais de erro explícitos toda vez que um erro é detectado e não pode ser corrigido.

Considerando métodos formais para analizar e introduzir propriedades e esquemas de tolerância a falhas em sistemas de *software*, redes de Petri estão entre aqueles mais aplicáveis.

Leveson [68, 70] aplica redes de Petri temporizadas [76] para a modelagem e análise de sistemas em tempo real críticos seguros, por exemplo sistemas de transporte metroviário e ferroviário. Belli [10] introduz uma metodologia para sistematicamente introduzir propriedades de tolerância a falhas no projeto de sistemas de software. O projeto do sistema é construído a partir de uma rede Predicado/Transição (Pr/T). A introdução destas propriedas é conseguida isolando-se caminhos seqüenciais no comportamento da Pr/T modelando o sistema. Estes caminhos são então traduzidos por expressões regulares que geram um código. Sempre que um erro ocorre o mecanismo de codificação pode detectar ou mesmo corrigir erros, dependendo do grau de redundância do código. A maior desvantangem desta abordagem é que ela só pode ser aplicada a caminhos seqüenciais do comportamento do sistema ou do componente de software.

Capítulo 3

G-NETS E SISTEMAS DE G-NETS

3.1 Introdução

Neste capítulo introduzimos os conceitos básicos e definições formais relacionados a *G-Nets* e sistemas de *G-Nets*. Sistemas de *G-Nets* são uma abordagem para a especificação, concepção e modelagem de sistemas complexos de software. Um sistema de *G-Nets* é composto de um certo número de *G-Nets* [35, 38], cada uma das quais pode ser vista como um módulo ou um objeto e suas operações associadas, denominadas métodos [13, 58, 98]. Os métodos proveem mecanismos bem definidos para esconder detalhes de suas realizações, e um conjunto bem definido de elementos de interface prove a comunicação entre outras *G-Nets*.

Uma *G-Net* é composta por duas partes: um lugar especial denominado *Lugar Genérico de Chaveamento* (GSP)[1], e uma *Estrutura Interna* (IS)[2] O GSP é um mecanismo para abstrair a estrutura interna de um módulo, servindo como interface de acesso única.

Neste capítulo introduzimos ainda tres exemplos. O primeiro é uma solução para o problema produtor/consumidor. O segundo é a representação da interação de alto-nível para a arquitetura cliente/servidor. O último exemplo, e o mais complexo, é a aplicação de *G-Nets* para modelar o inter-travamento (*interlocking*) para um sistema de controle distribuído para trens. Este exemplo mostra como *G-Nets* podem ser aplicadas para construir incrementalmente o modelo de um bloco de controle para um sistema de trens.

3.2 Definição de Sistemas de G-Nets

Nesta seção introduzimos os conceitos básicos de *G-Nets* e sistemas de *G-Nets* de acordo com [35, 38]. Introduzimos também novas notações para a representação gráfica e de-

---

[1] GSP origina-se da denominação em inglês *Generic Switching Place*.

[2] IS origina-se da denominação em inglês *Internal Structure*.

finição formal como apresentado em Perkusich et al[3]. Esta versão melhorada permite uma clara definição da rede, uma vez que métodos, inscrições e declarações de atributos são tornadas explicítas, desta forma permitindo melhor legibilidade. Além, disto as novas definições possibilitam o desenvolvimento de métodos de análise locais a *G-Nets*.[82].

Antes de introduzirmos formalmente os conceitos de *G-Nets* e sistemas de *G-Nets*, faremos uma introdução informal. Esta introdução informal visa prover ao leitor uma intuição sobre os conceitos básicos.

3.2.1 Introdução Informal para G-Nets e Sistemas de G-Nets

Um princípio de engenharia de software amplamente aceito na concepção de sistemas, advoga que um sistema deve ser composto por um conjunto de módulos independentes. Cada módulo no sistema esconde detalhes internos de suas atividades de processamento. Além disto, módulos comunicam-se através de interfaces bem definidas [41]. A notação de *G-Nets* prove um forte suporte para este princípio. Um *G-Net*, G, é composta por duas partes: um lugar especial denominado *Lugar Genérico de Chaveamento* (GSP), e uma *Estrutura Interna* (IS). O GSP prove a abstração para um módulo, servindo como única interface entre a *G-Net* e outros módulos. A estrutura interna é uma rede de Petri modificada, e representa a realização interna detalhada da aplicação modelada. A notação de *G-Nets* está bastante próxima àquela introduzida no Capítulo 2. Entre outras características esta notação permite ao usuário indicar comunicação entre *G-Nets* e terminação. A notação para *G-Nets* é apresentada na Figura 3.1, e explicada como a seguir:

1. A estrutura interna da rede (IS) é definida por um retângulo com as bordas arredondadas, definindo o limite da estrutura interna.

2. O GSP é indicado por uma elipse no canto superior esquerdo do retângulo definindo os limites da IS. A inscrição $GSP(nome\_da\_rede)$ define o nome da rede, para ser referida por outras *G-Nets*.

3. O retângulo com as bordas arredondados no canto superior direito do limite da IS é utilizado para identificar os métodos e atributos da rede, onde:

[3] De fato, a notação e definição formal para *G-Nets* e sistemas de *G-Nets* é uma versão melhorada em relação àquela apresentada em [38, 82]

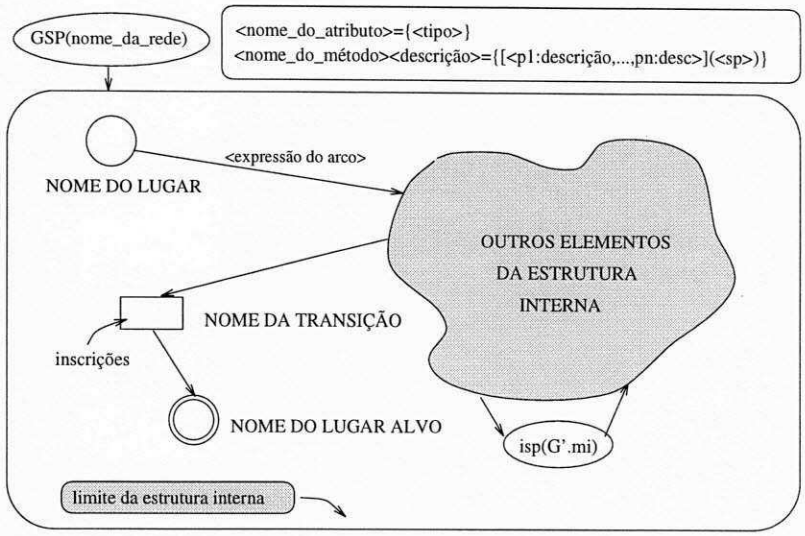

Figura 3.1: Notação utilizada para uma *G-Net*

- $\langle nome\_do\_atributo = \{\langle tipo \rangle\}$ define um atributo para rede, onde:

  $\langle nome\_do\_atributo \rangle$ é o nome do atributo, e

  $\langle tipo \rangle$ é o tipo para o atributo, o qual está restrito ao conjunto dos inteiros não negativos.

- $\langle nome\_do\_método \rangle$ é o nome para um método, $\langle descrição \rangle$ é uma descrição para o método. $\langle$ *p1:descrição*, $\cdots$, *pn:descrição* $\rangle$ é uma lista de argumentos para o método. Por fim $\langle sp \rangle$ é o nome do *lugar inicial* para o método.

4. Um círculo representa um lugar normal.

5. Uma elipse na estrutura interna representa um *lugar de chaveamento para instanciação* (*instantiated switching place (isp)*). O *isp* é utilizado para prover comunicação *inter-G-Net*. A inscrição $isp(G'.mi)$ indica a invocação da rede G' com método mi.

6. Um retângulo representa uma transição, a qual pode ter uma inscrição associada. Esta inscrição pode tanto ser uma atribuição como uma restrição de disparo. Utilizaremos a notação padrão da Linguagem C tanto para as atribuições como para as restrições de disparo.

7. Um círculo duplo representa um lugar de terminação ou *lugar alvo*.

8. Lugares e transições são conectados por arcos direcionados que podem carregar uma expressão.

O *GSP* em uma *G-Net* G, denotado por $GSP(nome\_da\_rede)$ na elipse da Figura 3.1, unicamente identifica o módulo. O *GSP* contém a declaração para um ou mais métodos executáveis (descritos no retângulo com as bordas arredondadas na Figura 3.1), especificando as funções, operações ou serviços definidas para a rede, e um conjunto de atributos especificando propriedades passivas do módulo (quando definidas). A estrutura detalhada e o fluxo de informação de cada método são definidos por uma rede de alto-nível modificada na estrutura interna. Mais especificamente, um método define os parâmetros de entrada, a marcação inicial correspondente à rede de alto-nível interna (o estado inicial da execução). A coleção de métodos e os atributos (de definidos) proveem a abstração ou visão externa do módulo.

Na estrutura interna, lugares representam *primitivas* e transições, juntamente com os arcos, representam conexões or relações entre as primitivas. Na definição original [35], estas primitivas podiam ser ações, predicados, entidades de dados e *lugares genéricos para chaveamento isp's*. Nesta tese consideraremos que estas primitivas podem ser predicados ou lugares genéricos para chaveamento. A razão para tanto é restringir a semântica das primitivas a uma forma tratável formalmente. Um conjunto de lugares especiais denominados *Lugares Alvo* representa o estado final de uma execução, e os resultados (se algum) a serem retornados. Uma transição, juntamente com os arcos, define uma sincronização e coordena a transferência de informação entre seus lugares de entrada e saída. Na Seção 3.2.2 estes conceitos serão introduzidos de forma mais específica e formal.

Dada uma *G-Net* G, um *isp* em G é denotado por $isp(G_{nome}.mtd)$ (ou simplesmente $isp(G)$, se nenhuma ambigüidade ocorrer) onde G_{nome} é uma identificação única para G, e mtd é um método definido para G. Um $isp(G_{nome}.mtd)$ denota uma instânciação da *G-Net* G, i.e., uma instância de invocação de G baseada no método mtd. Portanto, executando a primitiva *isp* implica na invocação de G (através do envio de fichas para G) utilizando um método especificado. As fichas contém os parâmetros necessários para definição das fichas na marcação inicial da rede invocada. Esta interação entre *G-Nets* pode ser comparada ou método de chamada remota de procedimento.

O *isp* serve como mecanismo primário para a conexão ou relacionamento entre diferentes *G-Nets* (módulos). Embutindo um *isp* de uma *G-Net* em um nível inferior em uma *G-Net* em um nível supeior, especifica uma configuração hierárquica. Ainda mais, embutindo um *isp* de uma *G-Net* especificando um servidor em uma *G-Net* especifi-

cando um cliente resulta em uma relação cliente-servidor [35]. O interesse maior nesta tese está neste último tipo de relacionamento entre *G-Nets*.

Antes de continuarmos com esta apresentação, introduziremos um exemplo. Na Figura 3.2 apresentamos um exemplo de um sistema de *G-Nets* composto de duas *G-Nets*. Não estamos preocupados neste ponto com a funcionalidade das redes. A rede à esquerda é denominada *G1*, possuindo um método definido, o qual é *ma:processa a={[a:inteiro](P1)}*. Isto significa que somente um método está definido para *G1*, e que não há atributos definidos. O método *ma* recebe um inteiro a, e uma ficha com este valor é depositado no lugar inicial *P1*. Temos quatro lugares definidos, os quais são $\{P1,P2,P3,P4\}$, e duas transições, $\{t1,t2\}$. Os lugares *P1* e *P2* são lugares normais, e o lugar *P4* é uma lugar alvo. O lugar *P2* é um *isp*. A transição *t1* não tem inscrição alguma. Isto significa que a ficha depositada no lugar *P1* é removida sem nenhuma restrição, e é depositada nos lugares *P2* e *P3*, após o disparo da transição *t1*. A ficha depositada no lugar *P2* inicializa a invocação da rede $G2$. Após a rede $G2$ terminar sua execução, o resultado é retornado a $G1$ através do lugar alvo *P4* em $G2$. Finalmente, a transição *t2* de $G1$ dispara e a soma formal de a (a ficha em *P3*) e o valor retornado y (a ficha em *P2*) é calculada e depositado no lugar *P4*. Uma vez que *P4* é um lugar alvo, $G1$ termina sua execução e o valor resultante é retornado para a rede que invocou $G1$ (não representada na figura). Na Seção 3.2.2 detalhamos a semântica operacional desta rede.

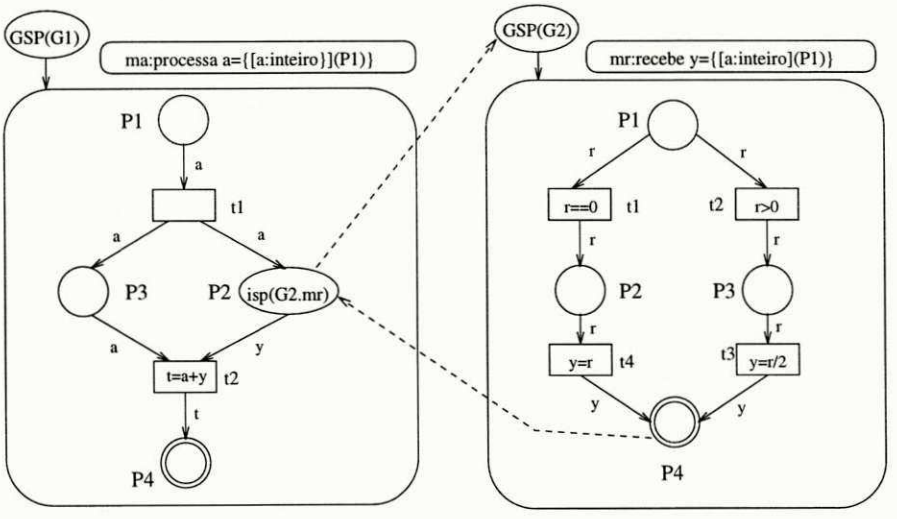

Figura 3.2: Duas *G-Nets* conectadas através de um *isp*

Um ambiente para editar e executar (simular) sistemas de *G-Nets* foi desenvolvido e e introduzido em [24, 35, 38]. A execução de uma especifição em *G-Net* é conduzida

por um conjunto concorrente de processos denominados agentes. Uma *G-Net* pode ser associada a dois modos de execução, *modo de instanciação* ou *modo servidor*. No modo de instânciação, a funcão de um agente é executar somente *uma* instância de uma invacação de uma *G-Net* baseando-se no método especificado. Um novo agente é criado para cada instânciação de uma *G-Net*, e o agente é terminado quando a instância da *G-Net* termina. No modo servidor, somente um agente é associado com a *G-Net* especificando um servidor. O agente é denominado um servidor, quando ele é responsável por todas as requisições dos clientes. Agentes comunicam-se através de passagem de ficha, como descrito acima. Nesta tese nos referiremos somente ao modo servidor.

No que segue-se formalizamos as definições relacionada a *G-Nets* e sistemas de *G-Net*.

3.2.2 Introdução a G-Nets e sistemas de G-Nets

O principal objetivo de formalizar os conceitos introduzidos na seção anterior é permitir a análise formal de um sistema de *G-Nets*. As definições que introduziremos são uma versão refinada daquelas introduzidas em [35, 38]. Formalmente um sistema de *G-Nets* é definido como a seguir:

Definição 3.1 *Sistema de G-Nets*
Um *sistema de G-Nets* (GNS) é uma tripla $GNS = (TS, GS, AS)$, onde

1. TS é uma coleção de fichas dinamicamente geradas durante a a execução do sistema;

2. GS é um conjunto de *G-Nets*; e

3. AS é um conjunto descentralizado, de agentes computacionais concorrentes, executando um sistema de *G-Nets*.

Definição 3.2 *G-Net*
Uma *G-Net* é a dupla $G = (GSP, IS)$, onde

1. GSP é um *Lugar Genérico de Chaveamento (GSP)* provendo uma abstração para a *G-Net*; e

2. IS é a *Estrutura Interna*, a qual é uma rede PrT modificada.

Definição 3.3 *Lugar Genérico de Chaveamento (GSP)*
Dado $GSP \in G$ ser um lugar genérico de chaveamento. Um GSP é definido por (NID, MS, AS), onde

1. NID é uma identificação única (nome) da *G-Net* G;

2. MS é um conjunto de métodos; e
   $\forall mtd_i \in MS$, $mtd_i = (\mathsf{m\_nome_i}, \mathsf{m\_argumentos_i}, \mathsf{m\_iniciador_i})$, onde

   $\mathsf{m\_nome_i}$ é um nome para mtd_i.

   $\mathsf{m\_argumentos_i}$ é uma tupla de variáveis especificando a ficha de entrada para o método mtd_i.

   $\mathsf{m\_iniciador_i}$ é um mapeamento de $\mathsf{m\_argumentos_i}$ para a *marcação inicial* da rede pertencendo a estrutura interna associada ao método mtd_i.

3. AS é um conjunto de atributos, e $\forall as_i \in AS, as_i \in \mathbb{N}$.

A pair da Definição 3.3 dizemos que um GSP é unicamente identificado por um nome denotado por NID, abstraindo um conjunto de métodos denotado por MS. Os métodos definem como a estrutura interna pode ser executada. Na Definição 3.3 m_iniciador define como a informação em m_argumentos é transformada em fichas do tipo correspondente. Estas fichas serão depositadas, no caso de mais de um argumento ser definido, no lugar inicial para o método.

Definição 3.4 *Estrutura Interna*
A *estrutura interna* de uma *G-Net*, $G.IS$, é uma estrutura de rede, isto é, um grafo direcionado bipartido definido por $G.IS = (\Sigma, P, T, I, O)$, onde

1. Σ é uma estrutura consistindo de algum tipo de predicados, juntamente com o conjunto de relações e operações definidas para estes predicados.

2. P é um conjunto finito e não vazio de lugares, denotados por círculos.

3. T é um conjunto de transições, denotadas por retângulos.

4. $I : T \rightarrow P^{\infty}$ é denominada função de entrada, definindo inscrições para transições e arcos de entrada.

5. $O : T \rightarrow P^{\infty}$ é denominada função de saída, definindo inscrições para transições e arcos de saída.

A definição acima introduzida para *G-Nets* é similar àquela introduzida para redes PrT como introduzidas no Capítulo 2. De fato podemos dizer que a estrutura interna de uma *G-Net* é uma rede PrT modificada. A maior diferença é que agora podemos associar uma ação, ou função aos lugares. De qualquer modo, a estrutura interna (IS) de uma *G-Net* pode ser transformada para uma rede PrT.

Como mostrado na Seção 3.2.1 temos tres tipos diferentes de lugares na estrutura interna de uma *G-Net*. A seguir introduzimos as definições associadas com estes diferentes tipos de lugares.

Definição 3.5 *Conjunto de lugares*
Dado que $G.IS$ é a estrutura interna de uma *G-Net*, O conjunto de lugares $P \in G.IS$ é definido por $P = (\mathcal{ISP}, \mathcal{NP}, \mathcal{GP})$, onde

1. $\mathcal{ISP}$ é um subconjunto *isp's*.

2. $\mathcal{NP}$ é um conjunto de lugares normais.

3. $\mathcal{GP}$ é um subconjunto de lugares alvo.

A definição acima apresentada para o conjunto de lugares $P \in G.IS$ é necessária pois cada subconjunto de lugares apresenta diferente semântica. Além disto, $\mathcal{ISP}$ e $\mathcal{GP}$ possuirão uma importância significativa para decomposição, análise e verificação de sistemas de *G-Nets*, como será mostrado no Capítulo 4. Agora, detalharemos o significado de cada um destes subconjuntos de lugares.

Um $isp \in \mathcal{ISP}$ prove mecanismos utilizados em sistemas de *G-Nets* para implementar interconexão entre *G-Nets*. Um *isp* definido em uma net G e invocando uma rede G' é denotado por $isp(G'm)$ e é definido pela quádrupla:

$$isp(G'm) = (NID, mtd, \text{ação\_antes}, \text{ação\_após})$$

NID é o identificador único de G', $mtd \in G'.GSP.MS$, ação_após e ação_antes são ações primitivas, cuja função é atualizar a *seqüência de propagação da ficha*, a qual introduziremos a seguir. Mais especificamente, um método $mtd \in G.MS$ define os parâmetros de entrada, e a marcação inicial da rede de Petri interna (define o estado inicial da execução). O conjunto de *Lugares Alvo* representa o estado final de execução para cada método, e os resultados a serem retornados. A coleção de métodos e atributos

(se algum estiver definido) prove a abstração ou visão externa do módulo. Impomos a restrição de que o conjunto de lugares alvo deve ser não vazio. Esta restrição é necessária para garantir que um método sempre alcance um estado final, esta restrição será melhor discutida no Capítulo 4.

Definição 3.6 *Ficha*
Dado G ser uma *G-Net*, e tkn ser uma ficha, então

1. tkn é uma tupla de *itens* da forma $tkn = \langle seq, scor, d_1, \cdots, d_q \rangle$, onde $tkn.seq$ é a *seqüência de propagação* da ficha, $tkn.scor \in$ (antes, após) é a *cor de estado*, e $tkn.d_i$, $i \in \mathbb{N}$, é a mensagem associada à ficha. O item $tkn.seq$ é uma seqüência de $\langle NID, isp, PID \rangle$, onde NID é a identificação da *G-Net*, isp é o nome de um ISP e PID é um identificador único de processo. A ficha está disponível para habilitação e disparo de transições quando $scor =$ após.

2. Se $tkn = \omega$, ele pode ser casado com qualquer seqüência.

A seqüência de propagação da ficha somente é alterada em um ISP ou GSP. Quando uma *G-Net* G é invocada por um ISP isp_i em outra *G-Net* G' (quando isp_i recebe uma ficha), a tripla $\langle G', isp_i, Pid_{G'} \rangle$, onde $Pid_{G'}$ é o identificador do processo executando G', é associada à seqüência de propagação da ficha antes dela ser enviada à *G-Net* G. Esta tripla indica que quando a execução de G termina, a ficha resultante deve ser retornada ao lugar identificado por isp_i in *G-Net* G'. O identificador de processo é necessário para distinguir para qual instância de execução de G' a ficha sendo retornada pertence. Quando a ficha de entrada é recebida pelo GSP G, a triple $\langle 0, 0, Pid_G \rangle$ é associada à seqüência de propagação da ficha, indicando que o agente responsável pela execução da invocação é identificado por Pid_G. Devido ao fato de Pid_G ser único, a seqüência de propagação é também única. A estrutura da ficha em *G-Nets* não somente garante que todas as fichas pertencendo à uma instância de execução de uma *G-Net* têm a mesma seqüência de propagação, mas também contém a história completa de propagação das fichas, a qual governa as interações entre os processos executando um sistema de *G-Nets*.

Devido ao campo de seqüência de propagação associado à ficha, mais de uma execução de uma *G-Net* pode ser executada simultaneamente. Isto pois diferentes seqüências de execução (invocações) podem ser unicamente identificadas. Na verdade, como dito anteriormente, a ficha carrega a história de execução de um sistema de *G-Nets*.

Como mencionado anteriormente a cor de estado da ficha pode assumir dois possíveis valores, tanto antes ou após. Uma ficha é dita estar *disponível* se sua $scor =$ após. Caso

contrário, ela é dita estar *indisponível.* Quando uma ficha é depositada em um lugar, sua cor de estado é antes, e após a ação primitiva (se definida) ser executada a cor da ficha é alterada para após, indicando que a ficha está pronta para ser utilizada para disparos de transições. O campo de mensagem de uma ficha é uma lista de indivíduos dependente da aplicação.

Um *lugar normal* $NP \in \mathcal{NP}$ possui as regras para manipular a ficha. Quando uma ficha é depositada em um NP, uma ação associada ao lugar é executa, esta execução baseia-se nos parâmetros especificados no campo msg. O resultado da ação é associado ao msg da ficha e a *cor de desvio* da ficha é definida. A cor de desvio da ficha serve para definir para qual transição de saída a ficha está disponível. Portanto, a ação associada aos lugares normais executa a mesma funcão das expressões associadas aos arcos e inscrições em transições para redes PrT. Finalmente o campo *scor* da ficha é atualizado para *scor* $\leftarrow$ após, e a ficha está disponível para habilitação e disparo de transições.

Um lugar alvo $GP_i \in \mathcal{GP}$ *deve pertencer* à marcação final de $G.IS$. Para cada método, *mtd*, os lugares alvo devem ser alcançaveis e devem ser únicos para cada método. Portanto, a informação resultante da execução pode ser retornada à *G-Net* que invocou.

O mecanismo de disparo de transição para *G-Nets* é definido pelas seguintes *regras de disparo*:

Definição 3.7 *Regras de disparo para transições*
Dado G ser uma *G-Net*, as regras de disparo de transição são definidas por:

1. Uma transição t é dita *habilitada* se e somente se cada lugar $p \in I(t)$ possui ao menos uma ficha, seu conteúdo satisfaz a condição definida por $I(p,t)$, e

   (a) todas as fichas envolvidas possuem a mesma seqüência de propagação;

   (b) todas as fichas envolvidas possuem suas *scor*= após;

   (c) o número de fichas definida em $O(t)$ é menor que a capacidade dos lugares.

2. Uma transição habilitada t pode *disparar* e quando dispara:

   (a) uma ficha satisfazendo $I(p,t)$ é removida de cada $p \in I(t)$;

   (b) uma ficha cuja seqüência de propagação é a mesma que a das fichas removidas de $I(t)$, cujas *scor* = após, e cuja campo de mensagem é definido por $O(t,p)$ é depositada em cada lugar $p \in O(t)$.

A invocação de uma *G-Net* define os mecanismos para iniciar a execução das estruturas internas, baseando-se na mensagem associada à ficha.

Definição 3.8 *Invocação de uma G-Net*
Dado G ser uma *G-Net*, a invocação da rede G baseada no método $mtd \in G.MS$ é conduzida da seguinte forma:

1. Determine a marcação inicial de G com base na definição para o método (o conteúdo das fichas depende da ficha de entrada);

2. Dispare as transições habilitadas, se alguma;

3. Invoque as primitivas habilitadas, se alguma;

4. Repita (2)-(3) até que um lugar alvo seja alcaçado;

5. Envie o resultado da execução (se definida por *mtd*) para a rede invocando.

De modo a clarear os conceitos introduzidos acima, apresentaremos um exemplo simples de um sistema de *G-Nets*. Para a rede mostrada na Figura 3.2 temos as seguintes definições.

$\boxed{G.GSP = (G1, AS, MS)}$

$AS = \emptyset$
$MS = \{ma : processa\ a = \{[a : inteiro](P1)\}\}$, a rede $G1$ possui um método definido, ma, recebendo como entrada um inteiro a; a marcação inicial para ma é o lugar $P1$.

$\boxed{G.IS = (P, T, I, O)}$

$P = \{P1, P2, P3, P4\}$, $\mathcal{GP} = \{P4\}$, $\mathcal{ISP} = \{P2\}$ e $\mathcal{NP} = \{P1, P3\}$.
$T = \{t1, t2\}$
$I(P1, t1) = O(t1, P3) = O(t1, P2) = I(P3, t2) = \{a\}$;
$I(P2, t2) = \{y\}$; $O(t2, P4) = \{t\}$.

3.3 Exemplificação da Aplicação de G-Nets

Nesta seção apresentamos a aplicação de sistemas de *G-Nets* para modelar a solução para o problema produtor/consumidor, assim como a interação cliente/servidor. A notação utilizada nestes exemplos, assim como em outros, segue aquela introduzida na Seção 3.2.1.

3.3.1 Modelagem para o Problema Produtor/Consumidor

O problema produtor/consumidor é um problema bastante conhecido. Ele consiste da sincronização entre um ou mais produtores e um ou mais consumidores. No exemplo que mostraremos assumimos que inicialmente, um produtor é capaz de produzir n items, e que o consumidor é capaz de consumir um item por vez. O exemplo que mostraremos é uma versão ligeiramente modificada em relação àquela apresentada em Perkusich et al[4] [82].

Na Figura 3.3 mostra-se a *G-Net* $G(P)$ para o produtor. De acordo com as definições introduzidas na Seção 3.2.2 temos:

$\boxed{G.GSP = (S, AS, MS)}$

$AS = \emptyset$

$MS = \{mp : produz = \{[n : número\ inteiro\ de\ itens](PR)\}\}$, a rede P possui um método definido, mp, o qual recebe como entrada um inteiro n; a marcação inicial para mp é o lugar PR.

$\boxed{G.IS = (P, T, I, O)}$

$P = \{PR, ISP1, ISP2, GP\}$, $\mathcal{GP} = \{GP\}$, $\mathcal{ISP} = \{ISP1, ISP2\}$ e $\mathcal{NP} = \{PR\}$.

$T = \{rs, nr, cr, co, pr\}$

todas as funções de entrada/saída são definidas para n, exceto $I(pr, GP) = \omega$.

Para esta rede um método é definido e é denominado *mp*. A função do método *mp* é produzir n itens para serem consumidos. Quando $G(P)$ é invocada através de *GSP(P)* uma ficha, juntamente com o campo n, expressando o número de itens a produzir é depositada no lugarPR. A ação associada ao lugar simplesmente decrementa o campo n. Se $n < 0$, a transição *pr* dispara e a invocação de $G(P)$ termina quando o *lugar alvo* *GP* é alcaçado. Contrariamente, isto é $n \geq 0$, a transição *rs* dispara, a ficha alcança o lugar *isp(C.ms)*, e a rede C é invocada utilizando o método *ms*. Esta invocação serve para consultar o estado de $G(C)$, de modo a garantir se ela está pronta ou não para consumir um item. Se a rede $G(C)$ não está pronta, a transição *nr* dispara, e a rede $G(C)$ é consultada novamente. De outro modo, o consumidor está pronto e a transição *cr* dispara. Após o disparo de *cr* uma ficha é depositada no lugar *isp(C.mc)* e a rede C é invocada com o método *mc*, juntamente com o item a ser consumido. Quando a ficha é retornada pela rede C a transição *co* dispara e uma ficha é novamente depositada no lugar *PR*.

---

[4] Na verdade, modificamos a estrutura interna da rede, através da adição de um lugar adicional para sincronização.

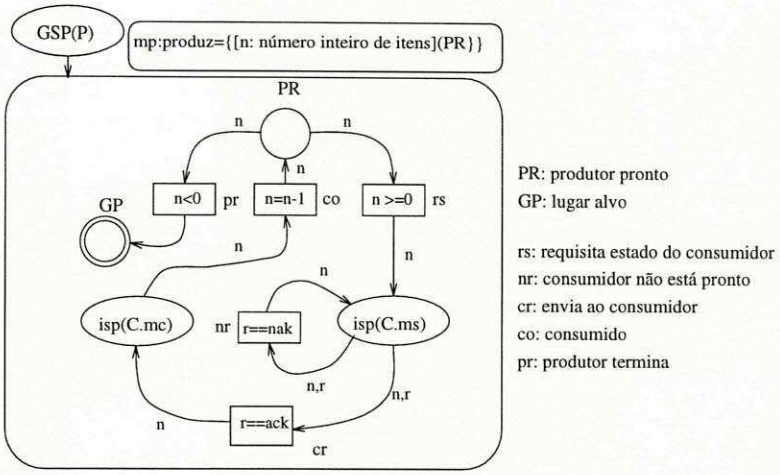

Figura 3.3: *G-Net* $G(P)$ modelando o produtor

A *G-Net* modelando o consumidor é mostrada na Figura 3.4. Dois métodos são definidos para $G(C)$: método estado (*ms*), e método consome (*mc*). Quando $G(C)$ é invocada através *GSP(C)*, se o método é *ms* uma fichha é depositada no lugar *IN*. Dependendo do estado de $G(C)$, o qual pode ser tanto consumindo como pronta para consumir, a transição *na* ou *sa* disparará. A escolha entre *na* e *sa* baseia-se nos estados representados pelo lugar *RC* (pronto para consumir) e lugar *CO* (consumindo). Se *RC* está marcado, a transição *sa* dispara e uma ficha alcança o *lugar alvo GPS* com um campo *reconhecido* (*ack*) associado a ele. Contrariamente, um campo *não reconhecido* (*nak*) é associado à ficha. Após o disparo de *sa*, uma ficha é depositada no lugar de entrada *WP*, e o consumidor está pronto para consumir. Quando $G(C)$ é invocada com o método consome, *mc*, uma ficha é depositada no lugar *TC*. Uma vez que *WP* foi previamente marcado, após uma execução com sucesso do método *ms*, a transição *sc* dispara e o lugar *CO* é marcado. Neste ponto $G(C)$ indica ao produtor que a ação requerida está sendo processada ou ainda está em processamento. Isto é porque quando a transição *sc* dispara uma ficha é depositada no lugar *CO* e outra no lugar alvo *GPC*. Deve ser notado que a transição *ac* pode ou não disparar antes que o produtor receba a ficha de retorno de *GPC*. Após o disparo de *ac*, uma ficha é removida de *CO* e uma outra é removida de *RC'*, e uma ficha é depositada em *RC*. Logo $G(C)$ está pronta para consumir um novo item.

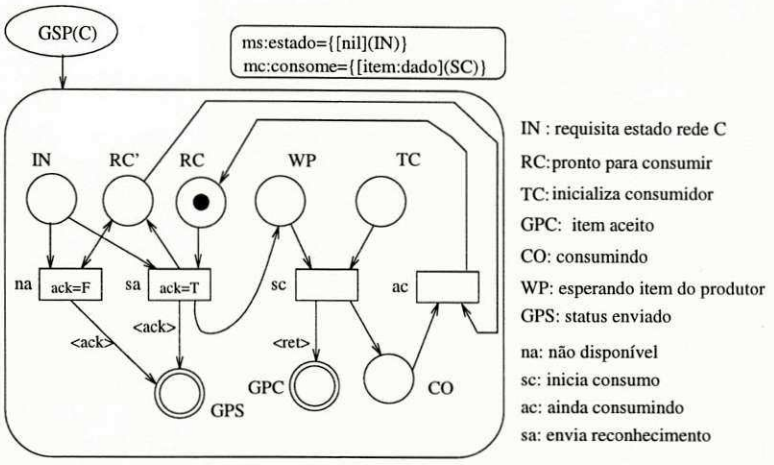

Figura 3.4: *G-Net* $G(C)$ modelando o consumidor

3.3.2 Modelagem da Interação Cliente/Servidor

Apresenta-mos agora a modelagem para a interação cliente/servidor utilizando *G-Nets*. O servidor deve atender requisições de clientes provendo um *buffer* para requisições pendentes. O cliente pode requisitar vários serviços ao servidor, dependendo do serviço a ser executado. Após uma requisição com sucesso, o cliente envia os dados para o servidor, de modo a inicializar a execução da requisição pendente. Quando o servidor recebe uma requisição, ele verifica se pode aceitar a requisição, caso haja espaço disponível no buffer, ou rejeita a requisição em caso contrário. Se o buffer de requisições pendentes não está cheio, o servidor coloca a informação relativa ao serviço no buffer. Esta informação é utilizada para selecionar o serviço requisitado a ser processado quando os dados são enviados pelo cliente. A disponibilidade de espaço no buffer é representada por um atributo, *BS*, associado ao servidor. Quando da inicialização da execução de uma requisição, o servidor atualiza o atributo relacionao ao buffer, e inicia a execução do serviço. Após executar a requisição, o servidor envia um reconhecimento positivo ao cliente.

A interação cliente/servidor como descrita acima, é modelada para o i-ésimo cliente pela *G-Net* $G(C)$, apresentada na Figura 3.5, e pela *G-Net* $G(S)$ modelando o servidor, como apresentado na Figure 3.6. O processo inicia-se quando o cliente ten uma requisição para enviar para o servidor. Neste caso, duas fichas são depositadas no lugar *CR* na *G-Net* $G(C)$. Uma das fichas contém a identificação do serviço (*serviço*) e a outra contém os dados necessários à execução do serviço especificado (*dado*). A transição *es* dispara e uma ficha é depositada no lugar *isp(S.ms)*, com a identificação de serviço, *serviço*, associada a ela. Então, a *G-Net* $G(S)$ é invocada com o método

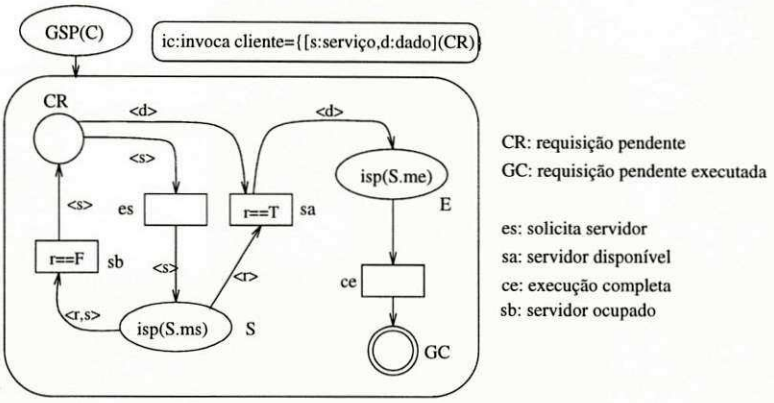

Figura 3.5: *G-Net* modelando o cliente

ms para verificar se o servidor está disponível. Uma ficha é depositada no lugar SV na *G-Net* $G(S)$. A disponibilidade de espaço no buffer determinará qual transição disparará, i.e., *bn* or *ba*. Se não há espaço disponível no buffer, a transição *bn* dispara e uma ficha é depositada no lugar GR e a invocação termina. Neste caso, após a invocação, a transição *sb* na *G-Net* $G(C)$ dispara, e uma ficha é depositada no lugar CR, e o processo é repetido até que pendência pelo serviço seja reconhecida. Se há espaço no buffer, a transição *ba* em $G(S)$ dispara e fichas são depositadas nos lugares PR e GR. A atributo de capacidade do buffer é decrementado de um e a invocação termina. Neste caso, a transição *sa* dispara, e o dado é obtido do lugar CR, uma ficha é depositada no lugar *isp(S.mc)*, invocando a *G-Net* $G(S)$ com método *mc*. Então, uma ficha é depositada no lugar TR na *G-Net* $G(S)$ e a transição *ex* dispara e a requisição pendente é processada. Além disto o atributo de capacidade do buffer, BS, é incrementado de um. Após o processamento da requisição (uma ficha no lugar ER), a transição *ac* dispara, um reconhecimento é enviado para o cliente, terminando a invocação. A transição*ce* na *G-net* $G(C)$ dispara, e uma ficha é depositada no lugar alvoGC, e a interação termina.

3.4 Embutindo Atributos Inteiros

De modo a prover meios para analizar uma dada *G-Net* através de técnicas tradicionais de redes de Petri, p.e., análise de alcançabilidade e análise de invariantes, mostramos como um atributo pode ser embutido na estrutura interna de uma da *G-Net*. Para conseguirmos embutir um dado atributo é necessário seguir um procedimento em dois passos como mostrado a seguir. A transformação é ilustrada com base no modelo servidor/cliente como introduzido na seção 3.3.2.

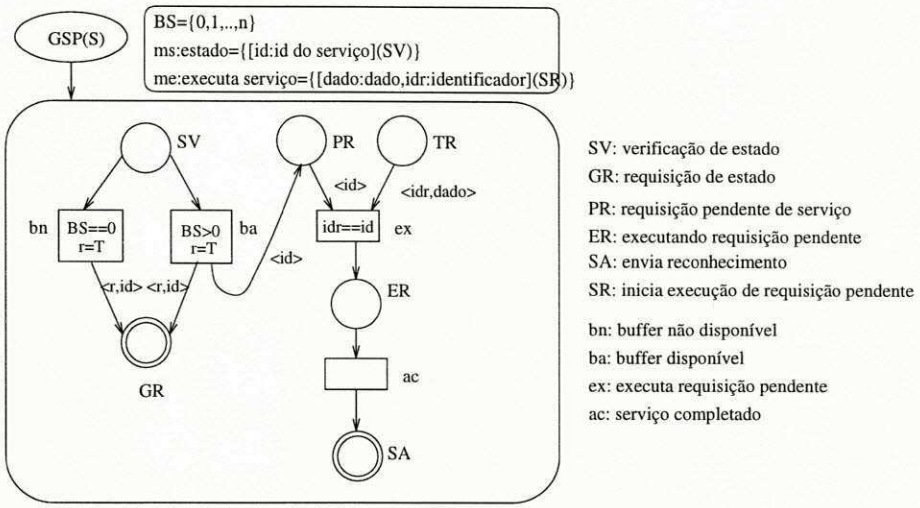

Figura 3.6: *G-Net* modelando o servidor

Embutindo Atributos

O exemplo que mostraremos baseia-se na *G-Net S* modelando o servidor, como mostrado na Figura 3.6. Na Figura 3.7 mostramos a *G-Net S* após o atributo BS ter sido embutido na estrutura interna da rede.

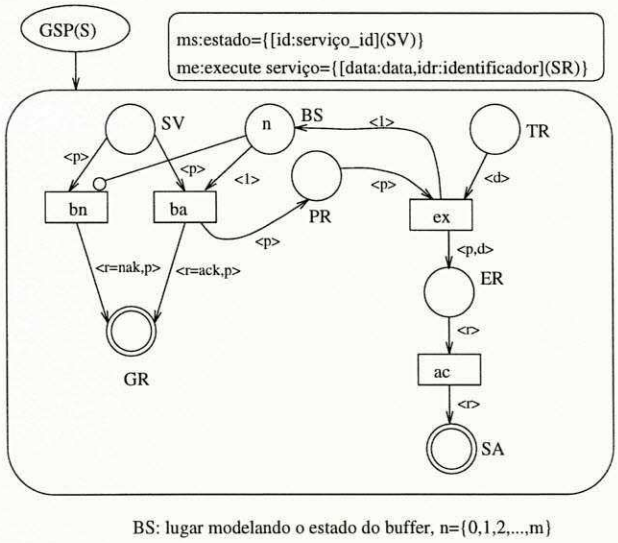

Figura 3.7: *G-Net* modelando o servidor com o atributo BS embutido na rede

Para embutir um atributo na estrutura interna de uma *G-Net*, o projetista deve verificar quais disparos de transições são dependentes do valor do atributo, e adicionar um arco inibidor e arcos regulares com pesos de acordo com estas restrições de disparo.

No caso na *G-Net* para o servidor adicionamos um arco inibidor para a transição bn, de forma que um *nak* só é retornado quando quando não há mais espaço no buffer, e adicionamos um arco com peso um do lugar BS para a transição ba para representar a decrementação de um toda vez que o servidor aceita uma requisição. Além disto, cada vez que o servidor inicia o atendimento de uma requisoção, disparando a transição ex, uma ficha é depositada no lugar BS, para indicar que existe uma posição livre no buffer de requisição.

Remoção de Arcos Inibidores

Para implementar a funcionalidade de arco inibidor $\overline{(p,t)}$ e eliminá-lo, uitilizamos o procedimendo mostrado na Figura 3.8.

Procedimento Elimina_Arco_Inibidor

1. **Cria** um lugar $p'; m(p') = k - \mu(p); k \in \mathbb{N}$
2. **Cria** arcos de modo que $\bullet p' = p\bullet$ e $p'\bullet = \bullet p$
3. **Cria** dois arcos $(p',t),(t,p')$, com $w(p',t) = w(t,p') = k - w\overline{(p,t)} + 1$

Figura 3.8: Procedimento para eliminar um arco inibidor

No procedimento para remover um arco inibidor, k é a capacidade do lugar, i.e., $k \geq \mu(p)$, para todas as marcações da rede em consideração. No caso em que o lugar não é limitado, um valor k que é maior que a marcação do lugar durante a operação normal pode ser utilizado. Esta remoção de arco é mostrada na Figura 3.9.

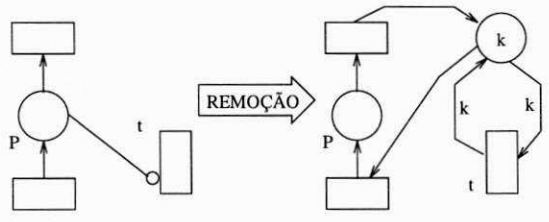

Figura 3.9: Examplo da remoção de um arco inibidor

O procedimento apresentado anteriormente freqüentemente cria um laço entre um lugar e uma transição. O propósito deste laço é habilitar/desabilitar a transição com base na marcação de algum lugar, sem modificar o fluxo de fichas entre eles. Para garantir que esta condição é satisfeita quando já existe um arco entre os dois nós em questão para os quais o laço é adicionado, o algoritmo mostrado na Figura 3.10 é aplicado.

Procedimento Incorpora_Multiplicidade

1. **Se** c_{tp} e $w(t,p) = w(p,t) \neq 0$
1.1. **Se** $q > w(t,p)$, faça $w(t,p) = w(p,t) = q$
1.2. **Se** $q \leq w(t,p)$, **Faça** nada
2. **Se** $c_{tp} > 0$
2.1. **Se** $c_{tp} \geq q$, **Faça** nada
2.2. **Se** $c_{tp} < q$, faça $w(t,p) = q - c_{tp}, w(p,t) = q$
3. **Se** $c_{tp} < 0$
3.1. **Se** $-c_{tp} > q$, **Faça** nada
3.2. **Se** $-c_{tp} \leq q$, faça $w(p,t) = q, w(t,p) = q + c_{tp}$

Figura 3.10: Procedimento para incorporar o efeito da multiplicidade

O algorítmo mostrado na Figura 3.10 ajusta os pesos dos arcos do laço de modo que tanto a condição de habilitação como o laço são satisfeitos e o fluxo de fichas entre o lugar e a transição é mantido. No caso em que existam arcos implementando a condição que cobre a condição que o laço implementaria, nenhum novo arco é adicionado.

A eliminação do arco inibidor entre o lugar BS e a transição bn na Figura 3.7 é apresentada na Figura 3.11.

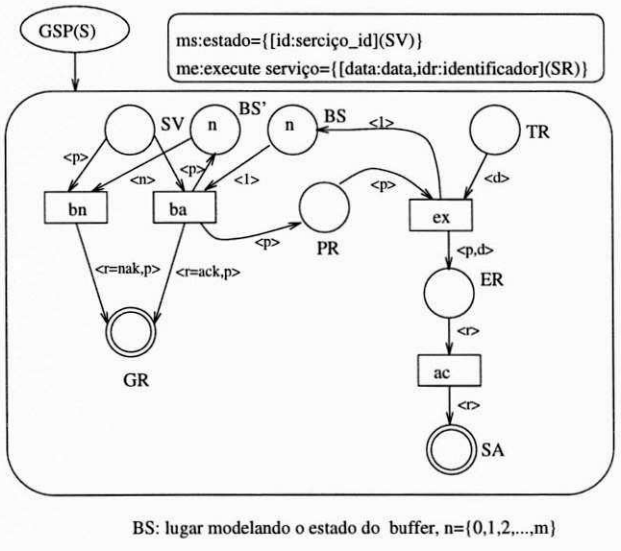

Figura 3.11: *G-Net* modelando o servidor com o arco inibidor removido

3.5 Controle Distribuído de um Sistema de Trens

De modo a ilustrar a aplicação de *G-Nets* para a modelagem de um sistema complexo real, introduziremos o problema do inter-travamento de trens. O problema a ser estu-

dado é o controle seguro de trens em um ambiente distribuído. A seguir introduziremos o problema e mostraremos como modelar o sistema utilizando o conceito de *G-Nets*. Este exemplo é complexo o suficiente para mostrar as vantagens da utilização de uma metodologia de concepção incremental como *G-Nets*, para contruir o modelo de um sistema.

3.5.1 Definição do Problema

O sistema de controle completo de um sistema de trens pode ser apresentado como uma hierarquia, como mostrado an Figura 3.12. No nível mais alto da hierarquia está o planejamento de *rotas de trens*. Neste nível decide-se a rota para os trens de modo que este possam alcançar seus destinos de forma segura. O *escalonamento de trens* é responsável pela definição do melhor escalonamento para as diferentes rotas na via, de modo a otimizar o fluxo de trens e evitar conflitos de rotas. Por rota entendemos os possíveis caminhos que podem ser estabelecidos na via. O *controle de rotas de trens* garante o fluxo de trens com rotas definidas de forma segura. No nível de *alocação de percursos para trens* alocam-se os caminhos, ou rotas, associadas aos trens. A *sinalização e comandos de chaves* executa o controle dos dispositivos conectados à via, assim como sinais e chaves.

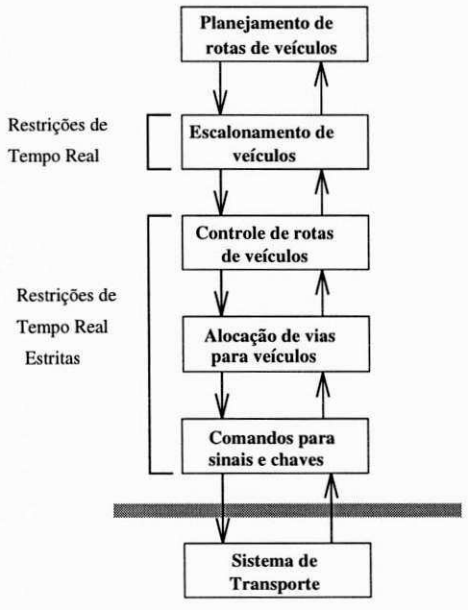

Figura 3.12: Hierarquia para um sistema de trens

A forma usual de resolver o problema de controle de trens é dividir a via em partes, denominadas *blocos*. O controle deste tipo de sistema é tradicionalmente centralizado.

Nesta tese consideraremos que o controle é distribuído. Cada bloco é controlado por um dispositivo denominado *controlador de bloco*. Entre outras funções, como roteamento, a função mais importante é o *inter-travamento* de trens [20, 34, 44]. O inter-travamento garante o trânsito seguro de trens através da via, permitindo aos trens troca de via. Além disto, ele controla o estado dos sinais associados à via. Estes sinais, quando *verdes*, permitem que um trem deixe uma seção. Quando os sinais estão *vermelhos* o trem deve parar.

Mostramos na Figura 3.13 um diagrama para o sistema de controle de trens. Assumimos que o trem possui dois pontos de referência, seu início e sem fim. Como discutido anteriormente, a via é dividida em blocos, e um controlador é associado a cada um destes blocos. Este controlador, denaminado *controlador de bloco*, implementa os tres níveis mais baixos da hierarquia apresentada anterirmente. Como pode-se ver pela Figura 3.13 estabelecemos um caminho de comunicação entre os controladores de bloco, através de canais de comunicação. Esta comunicação é necessária para que a informação associada com um dado trem possa ser enviada (recebida) quando um trem deixa (entra) um bloco sob controle de um determinado controlador de bloco.

Nesta apresentação concentraremos nossa atenção à modelagem e implementação das *funções de inter-travamento* para o controlador de bloco.

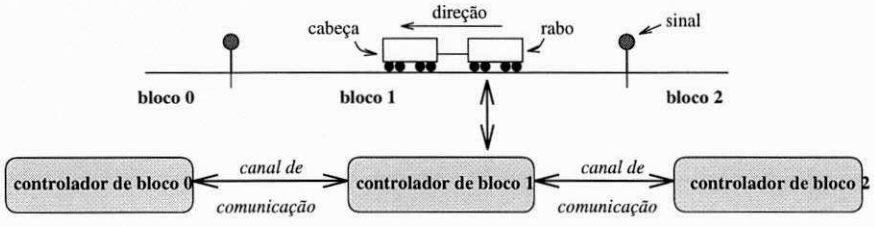

Figura 3.13: Parte de uma via e os blocos

3.5.2 Concepção do Controle Distribuído para os Blocos

Para conceber os controladores de bloco, a melhor abordagem é utilizar partes pré-definidas. Cada parte pré-definida corresponde a diferentes tipos de seções ou percursos possíveis. Uma *seção* é um modelo para diferentes padrões que podem existir em uma via de trem. Baseado neste modelos para seções, é possível conceber incrementalmente o modelo do sistema utilizando uma abordagem *bottom-up*. Para tanto definimos tres tipos de seções: *elementares*, *básicas* e *conexões*.

Seções elementares são as menores entidades do sistema. As seções elementares são mostradas na Figura 3.14. Neste caso somente o movimento unidirecional de trens é

permitido. As linhas pontilhadas representam os possíveis caminhos para os trens. Na entrada de uma seção temos um sinal (representado pelo círculo vazio), e uma seta indicando a direção permitida para a entrada de um trem. Ao fim da seção temnos um sensor (representado pelo círculo preto), que gera um sinal quando um trem deixa a seção. Temos ainda um seta preta indicando a direção permitida para a saída do trem que entrou na seção.

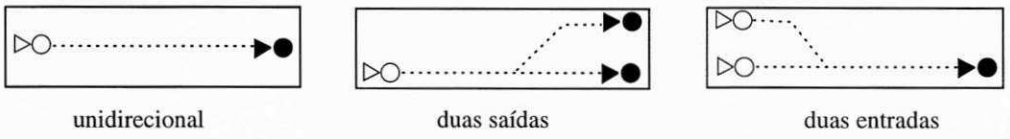

Figura 3.14: Seções elementares

Seções básicas apresentam uma complexidade funcional maior que as elementares. Elas permitem o movimento de trens em ambas as direções, mas não permitem a mudança de direção uma vez que o trem entrou na seção. Todas as seções básicas podem ser construídas através da composição das tres seções elementares. Na Figura 3.15 as seções básicas são apresentadas. Além das seções apresentadas temos que definir conexões. As conexões permitem que trens troquem de via. Definimos dois tipos de conexão, como mostramos na Figura 3.16. O significado das linhas pontilhadas e outros símbolos é o mesmo definido para as seções elementares.

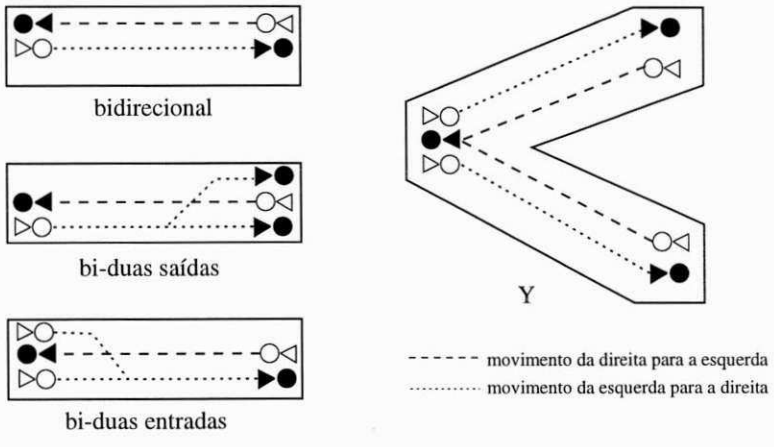

Figura 3.15: Seções básicas

3.5.3 Modelos de Rede de Petri para as Seções

O modelo e a metodologia para construir modelos de rede de Petri – redes P/T no caso – para este tipo de sistema são bem conhecidos e não serão detalhados nesta tese. De

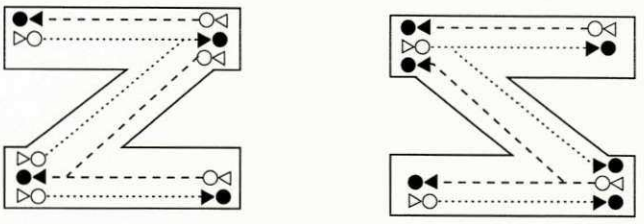

Figura 3.16: Conexões

qualquer modo, introduziremos os conceitos básicos e algumas redes P/T modelando diferentes tipos de seções. O leitor pode referir-se a [80, 103, 108] para uma introdução a metodologias estruturadas para construção de modelos utilizando redes P/T, e a [8, 81, 109] para modelagem de sistemas de transporte, em especial trens.

Introduziremos os modelos de redes P/T para algumas das seções do sistema de tres como apresentado em [8, 81].

Na Figura 3.17 mostramos o modelo de rede P/T para a seção simples (unidirecional). Discutiremos primeiramente o significado dos lugares e transições.

- M representa um trem em moviemento.
- S representa que o trem alcançou o fim da seção ou está parado.
- F representa que a seção está livre. A capacidade deste lugar é um, uma vez que somente um trem é permitido por vez na seção.
- C representa que a seção pode ser cruzada por um trem vindo de outra seção – este lugar só é necessário quando a seção pertence a uma chave ou cruzamento.
- Transição *en* dispara quando o trem entra na seção.
- Transição *es* dispara quando o trem alcança o fim da seção, saindo da mesma, ou parando.
- Transição *ex* dispara quando o trem deixa a seção.

Para explicar o comportamento da seção unidirecional introduziremos o conceito de *requisitos operacionais*. Um requisito operacional define uma regra que deve ser verificada para o correto e seguro movimento de trens ao longo da seção. No caso das seções elementares definimos os seguintes requisitos operacionais:

OR1: É sempre o caso que existe no máximo um trem em movimento ou parado em uma seção elementar.

OR2: É sempre o caso que um trem que entra em uma seção sai dela.

Observe que apesar do fato destes requisitos operacionais serem locais às seções, eles devem ser válidos para o sistema como um todo. Este requisito será discutido em detalhes ainda nesta seção. A rede P/T modelando a seção unidirecional e verificando a OR1 e OR2 é mostrada na Figura 3.17. Para verificar o correto comportamento de acordo com as regras operacionais aplicamos a análise de invariantes como discutida no Capítulo 2, para detalhes refira-se a [8, 81].

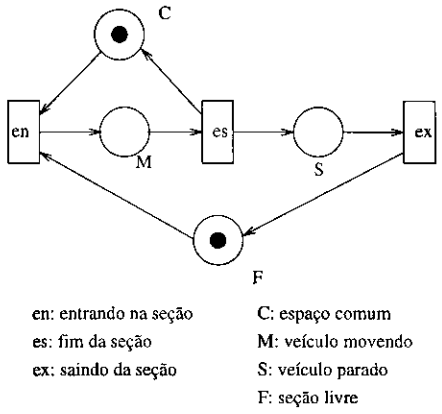

Figura 3.17: Rede P/T modelando uma seção unidirecional

A partir dos invariantes de lugar é possível derivar a seguinte equação:

$$M(M) + M(S) + M(F) = 1 \tag{3.1}$$

A Equação 3.1 verifica o requisito operacional OR1. Isto pois é sempre o caso em que, no máximo, existe somente uma ficha no conjunto de lugares $\{M,P\}$. Isto significa que quando a seção não está livre, $M(F) = 0$, temos no máximo um trem parado ou em movimento na seção.

A partir dos invariantes de transição é possível mostrar que é sempre io caso que a seqüência de disparo das transições é $en \rightarrow es \rightarrow ex$. Claramente, esta seqüência de disparo verifica o requisito operacional OR2. O significado desta seqüência de disparo (invariante) é que um trem que entrou na seção sempre sai. Esta seqüência também garante que um trem entrando pela esquerda (direita) deixa a seção pela direita (esquerda), sem troca de direção.

Na Figura 3.18 mostramos o modelo de rede P/T para uma seção elementar de duas entradas. Assim como no caso da seção unidirecinal, os requisitos operacionais OR1 e OR2 devem ser verificados. A partir dos invariantes de transição, é direto mostrar que as seqüências de disparo são: *en1*→*es*→*ex* e *en2*→*es*→*ex*, o que claramente verifica OR2. A partir dos invariantes de lugar é possível derivar a seguinte equação:

$$M(M1) + M(M2) + M(S) + M(F) = 1 \tag{3.2}$$

A partir da Equação 3.2 é fácil ver que o número de fichas nos lugares da rede, representando um trem em movimento após entrar pela *entrada 1* (transição *en1*) (representado pelo lugar *M1*), ou um trem em movimento após entrar pela entrada 2 (transição *en2*) (representado pelo lugar *M2*), ou que a seção está livre (representado pelo lugar *F*), ou ainda que o trem está parado na seção (representado pelo lugar *S*), deve ser sempre igual a um. Isto significa que há ou um trem *em movimento*, *parado* ou ainda a *seção está livre*. Portanto a regra operacional OR1 é verificada.

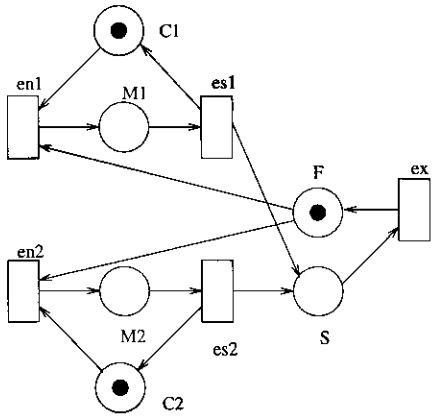

Figura 3.18: Rede P/T modelando uma seção com duas entradas

Na Figura 3.19 mostramos modelo de rede P/T para a seção elementar com duas saídas. Similarmente ao caso da seção unidirecional e ao caso da seção elementar com duas entradas é possível verificar OR2. A partir dos invariantes de transição é possível identificar que as possíveis seqüências de disparo são: *en*→*es*→*ex1* e *en*→*es*→*ex2*, o que garante que todos os trens que entram na seção saem da seção. Além disto, pode ser mostrado que os invariantes de lugar para esta seção são os mesmos que os da seção unidirecional, e portanto OR1 é verificada.

Para construir os modelos para as outras seções, i.e., seções básicas e conexões, pode-se adotar uma abordagem estruturada. Esta abordagem baseia-se na fusão de lugares. Este procedimento garante que os invariantes de lugar para cada uma das

redes fundidas permanecerão inalterados e novos invariantes de lugar que aprarecerão podem ser facilmente derivados pela concatenação dos invariantes de lugar das redes originais. Provas detalhadas podem ser encontradas em [8, 81, 80].

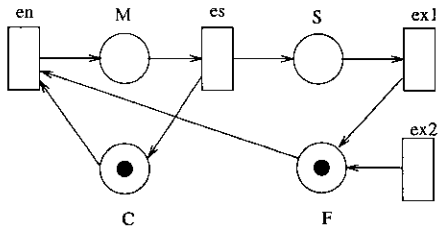

Figura 3.19: Rede P/T modelando uma seção com duas saídas

Para ilustrar o procedimento de fusão de lugares, mostramos a rede P/T resultante modelando uma seção bidirecional, e o modelo da seção bidirecional com duas saídas. O modelo da seção bidirecional é mostrado na Figura 3.20, e o modelo para a seção bidirecional com duas saídas é mostrado na Figura 3.21.

Explicaremos informalmente a construção destes dois tipos de seção. No caso da seção bidirecional tomamos duas seções unidirecionais, uma controlando o movimento de trens da direita para a esquerda, e outra controlando o movimento de trens da esquerda para a direita. Como mostramos na Figura 3.20 os lugares *Cr* e *Fl* (resultando no lugar *C*), e os lugares *Cl* e *Fr* (resultando no lugar *F*), são fundidos para obter o modelo da seção bidirecional. Neste caso os invariantes, lugar e transição, são obtidos por composição, e não precisam ser calculados novamente. Pode ser mostrado que os invariantes de lugar para esta rede são obtidos por *justaposição* dos invariantes das redes originais, e os invariantes de transição são obtidos por concatenação. Justaposição significa que os invariantes serão sobrepostos considerando lugares de mesmo nome. Deve-se notar que este procediemento é possível pois a marcação dos lugares envolvidos é a mesma. Concatenação significa que os invariantes de transição serão concatenados. Pela concatenação dos invariantes de transição é possível mostrar que as seqüências de disparo das transição sempre serão $enr \rightarrow esr \rightarrow exr$ e $enl \rightarrow esl \rightarrow exl$. Estas seqüências garantem que quando um trem entra na seção pela esquerda (direita) ele sempre deixa a seção pela saída da esquerda (direita) sem troca de direção. Então a regra operacional OR2 é verificada.

A partir da análise dos invariantes de lugar pode-se mostrar que OR1 ainda é válida. Informalmente, pode-se verificar que o número de fichas no conjunto de lugares $\{Mr, Sr, C, F, Ml, Sl\}$ será sempre um, o que garante que temos ou um e somente um trem movendo-se ou parado, ou ainda que não há trem na seção.

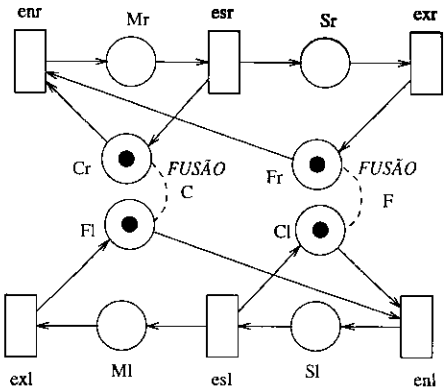

Figura 3.20: Rede P/T modelando uma seção bidirecional

Para o caso da seção bidirecional com duas saídas a verificação da propriedade OR1 é a mesma que para a seção bidirecional. Pode-se mostrar que as duas seções possuem os mesmos invariantes de lugar. No caso da propriedade OR2 as seguintes seqüências de disparo podem ocorrer:

1. $enr \rightarrow esr \rightarrow exr$

2. $enr \rightarrow esr \rightarrow exr'$

3. $enl \rightarrow esl \rightarrow exl$

As seqüências 1. e 2. garantem que um trem que entra na seção por *enr* sempre saíra por *exr* ou *exr'*. A terceira seqüência de disparo (3.) garante que um trem que entre por *enl* sempre saíra por *exl*. Estas seqüências de disparo garantem também que os trens não mudam de direção, uma vez que entram na seção com uma direção definida.

É possível construir todos os modelos de rede P/T para as seções apresentadas nas Figuras 3.15 e 3.16. No caso das seções do tipo conexão uma outra regra operacional deve ser definida.

OR3: é sempre o caso que existe somente um trem em movimento em uma *conexão* durante a troca de vias.

Informalmente a propriedade ou regra operacional acima, diz que quando não há trem trocando de via, é possível haver mais de um trem, movendo-se ou parado na conexão.

Outros tipos de seções podem ser definidas, p.e. cruzamento, mas não é nosso objetivo modelar exaustivamente cada tipo de seção. Na verdade, a discussão acima

teve o propósito de introduzir um sistema complexo real. No que segue apresentaremos um abordagem baseada em *G-Nets* para representar/modelar o problema de inter-travamento em um ambiente de controle distribuído.

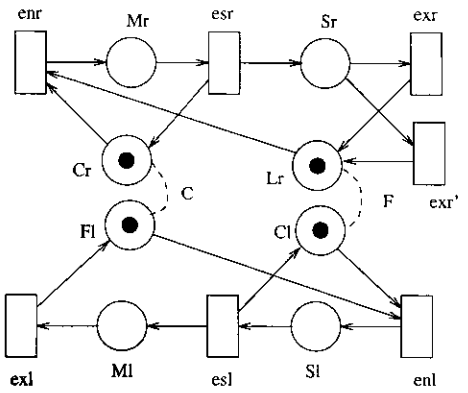

Figura 3.21: Rede P/T modelando uma seção bidirecional com duas saídas à esquerda

3.5.4 Modelo Baseado em G-Net para as Seções

Existem duas possíveis abordagens para construir modelos de *G-Nets* para cada uma das seções básicas e conexões mostradas nas Figuras 3.15 e 3.16. A solução mais direta é encapsular as redes P/T para cada uma das seções. Entretanto, como será mostrado, esta abordagem sofre com o fato de que as redes podem se tornar muito grandes e complexas. A segunda abordagem consiste em definir-se uma *G-Net* que será responsável pela sincronização das outras redes necessárias ao modelo. Esta última abordagem resulta em um modelo multi-nível para as seções, o qual é bem mais simples de ser entendido, concebido e analisado. A aplicação desta abordagem multi-nível para obter os modelos das seções resulta em um modelo cuja complexidade é reduzida por um grande fator. Note que nesta seção estamos enfatizando a construção das redes; a validação (análise) das mesmas será discutida no Capítulo 4.

Outro ponto que deve ser enfatizado é o fato de que consideraremos tanto na abordagem por encapsulamento como multi-nível aspectos de comunicação. Por exemplo, consideraremos a necessidade de estabelecer comunicação entre os níveis de alocação e comando (veja Figura 3.12). Neste caso introduziremos de forma explícita meios pelos quais a realização de uma seção possa informar ao comando a necessidade de alterar a posição de uma chave ou um sinal associado à seção. Esta comunicação é abstraída ou talvez negligenciada nos modelos para as seções utilizando redes P/T. É nossa opinião que uma representação explícita da interação entre os diferentes níveis é absolutamente

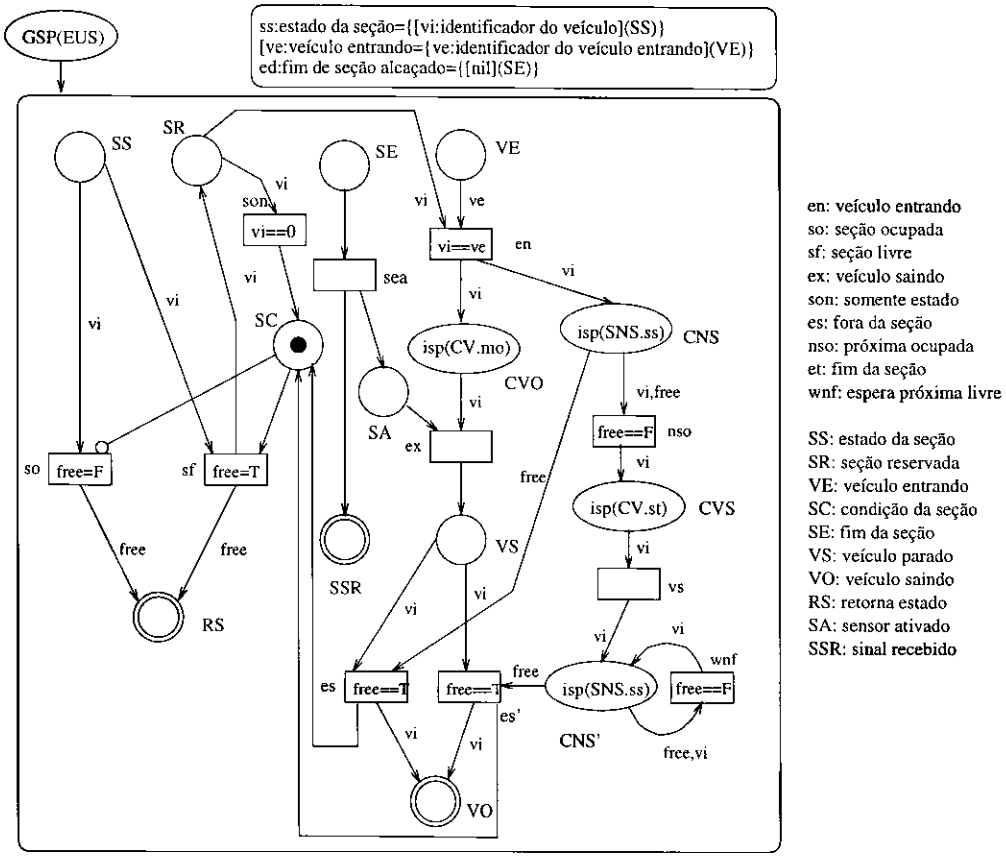

Figura 3.22: *G-Net* modelanto uma seção unidirecional

essencial para concepção de um controle distribuído. Isto dito, introduziremos primeiramente a abordagem por encapsulamento.

Encapsulamento dos Modelos de Redes P/T

No caso da abordagem por encapsulamento iniciaremos com o modelo elementar para uma seção unidirecional. Na Figura 3.22 mostramos o modelo para a seção unidirecional.

Definimos para a *G-Net EUS* (Elementary Unidirectional Section), modelando a seção unidirecional, tres métodos:

1. O método *ss* é responsável por informar o estado da seção e reserva-la. Este método tem um parâmetro de entrada, denominado *vi*, isto é, o idetificador do trem. O lugar inicial é *SS*.

2. O método *ve* implementa a funcionalidade de um trem entrando em uma seção. Define-se um parâmetro de entrada, *ve*, o identificador do trem entrando na seção.

O lugar inicial para o método *ve* é VE.

3. O método *ed* recebe uma ficha indicando que o fim da via foi antingido, não há nenhum parâmetro de entrada para este método. O lugar inicial é SE.

Estes tres métodos juntos permitem que os níveis superiores aloquem a seção para um determinado trem e a verificação do estado da seção. Estes métodos permitem também que os níveis inferiores indiquem que o fim de via foi alcaçado. Vamos introduzir agora de modo informal o comportamento da seção.

Para verificar o estado da seção, a *G-Net* $G(EUS)$ é invocada com método *ss*. Então uma ficha, com o identificador do trem, é depositada no lugar SS. Se o lugar SC possui uma ficha, a capacidade de SS é um (somente um trem é permitido na seção), a transição *so* dispara, e atribui-se falso à variável *livre*. A ficha alcaça o lugar alvo RS, e uma ficha é retornada a rede que invocou, informando que a seção está ocupada. No caso de o lugar SC estar marcado (indicando que a seção está livre), a transição sf dispara, e atribui-se verdadeiro à variável *livre*. A ficha atinge o lugar alvo RS e é retornada a rede que invocou. Mais ainda, uma ficha com o identificador do trem *vi* é depositada no lugar SR. Duas situações podem então ocorrer. Primeiro, quando $vi = 0$, significa que a seção não deve ser reservada para o trem *vi*, o objetivo da invocação é somente determinar o estado da seção. Neste caso a transição *son* dispara, e uma ficha é depositada no lugar SC indicando que a seção está livre. De forma contrária, a ficha indicando o identificador do trem permanece no lugar SR, indicando que a seção está reservada para o trem *vi*. Para indicar que um trem está entrando na seção a *G-Net* $G(SEU)$ é invocada com método *ve*. Então, uma ficha é depositada no lugar VE. A transição *en* dispara se o identificador do trem entrando *ve* é o mesmo que *vi*, previamente depositada no lugar SR. Se o identificador do trem não é o mesmo, uma falha ocorreu no sistema, e neste caso uma ação de recuperação deve ser tomada. Este aspecto será discutido no Capítulo 4. Entretanto, por hora assumamos que $ve = vi$. Após o disparo da transição *en*, os lugares CVO ($isp(CV.mo)$) e CNS ($isp(SNS.ss)$) serão marcados. A invocação $isp(CV.mo)$ pode ser interpretada de duas formas. Primeiro, ela pode representar o envio de um sinal de movimento para o trem, ou a mudança do estado do sinal no início da seção para "verde". A invocação $isp(SNS.ss)$ verifica se a próxima seção na rota do trem está livre ou não. Se a próxima seção não está livre, a transição *nso* dispara, e uma ficha é depositada em CVS ($isp(CV.st)$). Este *isp* invoca uma rede de controle ($G(CV)$, não representada) para tanto informar ao trem que este deve parar ao fim da seção, ou que o estado do sinal ao fim da seção deve ser alterado para "vermelho". Se a

próxima seção não está livre, uma ficha é depositada no lugar CNS' ($isp(SNS.ss)$) e o estado da próxima seção é verificado até que esta esteja livre. Retornando ao ponto em que o trem entrou na seção, a transição *en* disparou. O trem manter-se-á movendo, e portanto eventualmente o sensor associado ao fim da seção detectará que o trem atingiu o fim da seção. Neste ponto, uma rede de nível inferior invoca $G(EUS)$ com método *ed*, a transição *sea* dispara, e o lugar SA (indicando que o sinal de fim de seção foi recebido) é marcado. Portanto a transição *ex* pode disparar. Após o disparo da transição *ex* uma ficha é depositada no lugar VS. Se a próxima seção está livre a transição *es* dispara, e uma ficha contendo o identificador do trem é depositada no lugar alvo VO. Além disto uma ficha é depositada no lugar SC indicando que a seção está livre.

Na Figura 3.23 mostramos o modelo de *G-Net* encapsulado para a seção bidirecional. Para esta rede temos os seguintes métodos definidos:

1. O método *ss* informa o estado da seção. Ele possui dois parâmetros de entrada, denominados *vi* (identificador do trem), e *d*, definindo a direção permitida para o próximo trem a entrar na seção. O lugar inicial para o método *ss* é SS.

2. O método *vr* (*vl*), indica que um trem está entrando pela direita (esquerda), seu parâmetro de entrada, tanto para *vr* e *vl*, é *ve* (trem entrando). O lugar inicial para o método *vr* (*vl*) é VER (VEL).

3. O método *er* (*el*) quando invocado indica que o trem atingiu o fim da seção, não há parâmetro para este método. O lugar inicial para o método *er* (*el*) é SER (SEL).

A descrição do comportamento da *G-Net* $G(SBB)$ (*section basic bidirecional*) é análoga à descrição do comportamento para a *G-Net* $G(EUS)$. A diferença é que a direção do trem deve ser informada, de modo a permitir a entrada pela direita ou pela esquerda. Como pode ser visto a realização interna dos métodos *vr* e *vl* é a mesma que para o método *ve* para a *G-Net* $G(EUS)$.

Na Figura 3.24 mostramos a *G-Net* modelando uma seção bidirecional com duas saídas. Como pode ser visto na figura, é óbvio que, apesar da abordagem por encapsulamento ser possível de ser aplicada, a rede resultante pode ser muito grande e complexa. Mesmo considerando que ainda é possível analisar uma rede deste tipo, é muito provável que o problema de explosão de estados ocorra. Ainda mais, não estamos nos aproveitando das caracteristicas de *G-Nets*. Nós não apresentamos os modelos para

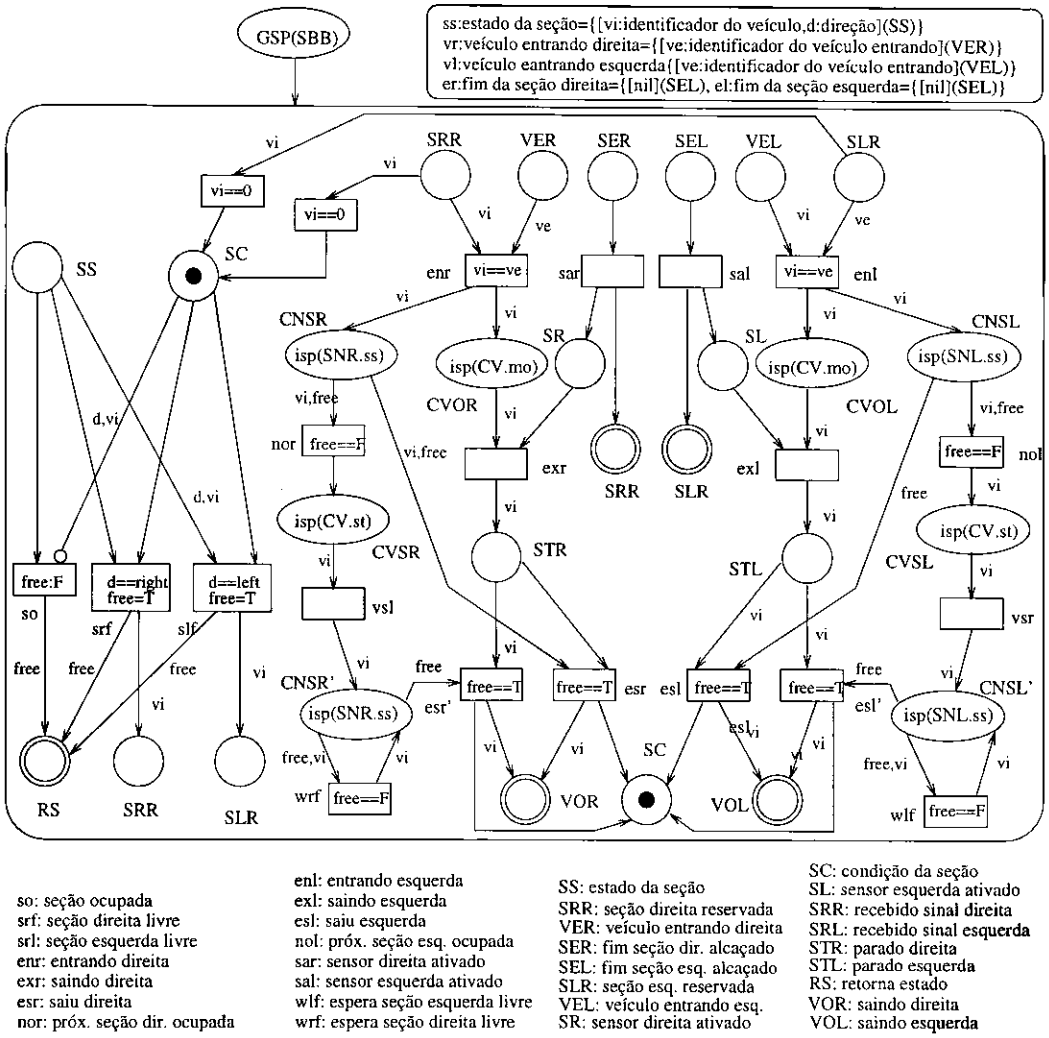

Figura 3.23: *G-Net* modelando uma seção bidirecional

as conexões, mas é evidente que as redes serão muito maiores e mais complexas que para a seção bidirecional com duas saídas. Portanto, a abordagem por encapsulamento torna-se proibitiva para componentes muito complexos. De qualquer modo, é possível encapsular todos os modelos de redes P/T para as seções previamente definidas.

Por esta razão na próxima seção mostraremos como adotar uma abordagem multi-nível para construir os diferentes modelos para as seções.

3.5.5 Modelagem Multi-Nível Baseada em G-Nets para as Seções

Com o aumento da complexidade funcional das seções, a abordagem por encapsulamento dos modelos de *G-Net* torna-se impratícavel. Portanto, adotaremos uma abordagem alternativa. Tomando por base os modelos elementares, apresentamos somente o modelo

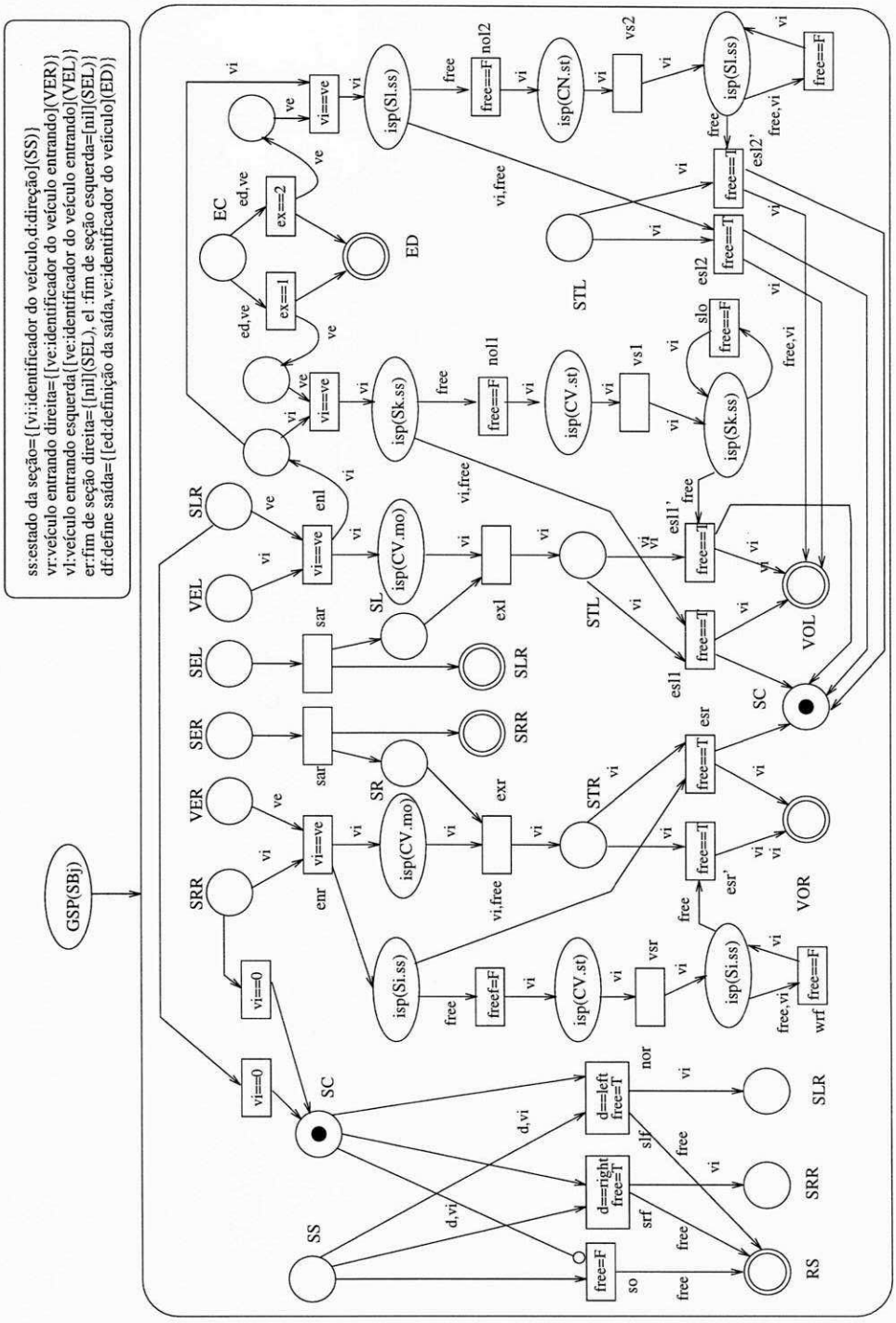

Figura 3.24: *G-Net* modelando uma seção bidirecional com duas saídas

unidericional (vide Figura 3.22), podemos conceber um sistema de *G-Nets* modelando as seções básicas e conexões. Apresentaremos somente o modelo para seção bidirecional. Para os outros tipos de seção a abordagem é mesma.

A idéia é utilizar um sistema de *G-Nets* para modelar as seções básicas e conexões. Mostramos na Figura 3.25 uma visão de alto nível para o sistema de *G-Nets* realizando uma seção bidirecional.

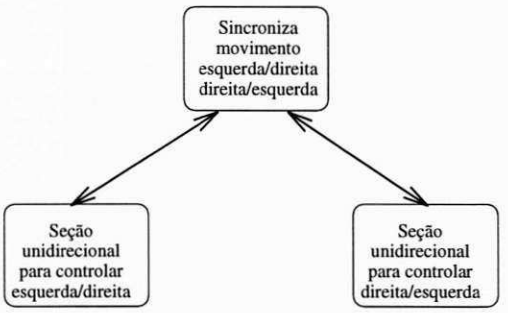

Figura 3.25: Uma concepção de alto-nível multi-nível para a seção bidirecional

Para permitir a concepção multi-nível da seção bidirecional, modificaremos o modelo da seção unidirecinal como mostrado na Seção 3.5.4. O modelo modificado para a seção unidirecinal é mostrado na Figura 3.26, e é muito similar àquele apresentado na Figura 3.22. De fato removemos o método que informava o estado da seção, pois esta é uma função de sincronização, e deve er embutida na *G-Net* responsável pela sincronização. O comportamento desta rede é bastante similar ao descrito para a *G-Net* da Figura 3.22, portanto não o discutiremos. Para construir o modelo da seção bidirecional necessitamos duas destas redes: uma para controlar o movimento de trans da esquerda para a direita, e outra para controlar o movimento da direita para a esquerda.

A rede de sincronização é apresentada na Figura 3.27. A *G-Net* $G(BI)$ sincroniza as outras duas redes. Para esta rede definimos os seguintes métodos:

1. O método ss informa o estado da seção. Ele possui dois parâmetros de entrada, denominados identificador do trem vi, e d (direção), a qual pode ser rl para o movimento da direita para a esquerda, e lr para o movimento da esquerda para a direita. O lugar inicial para o método ss é RS.

2. O método sv recebe como parâmetro de entrada o identificador do trem entrando na seção (ve), e a direção permitida (d) para o trem.

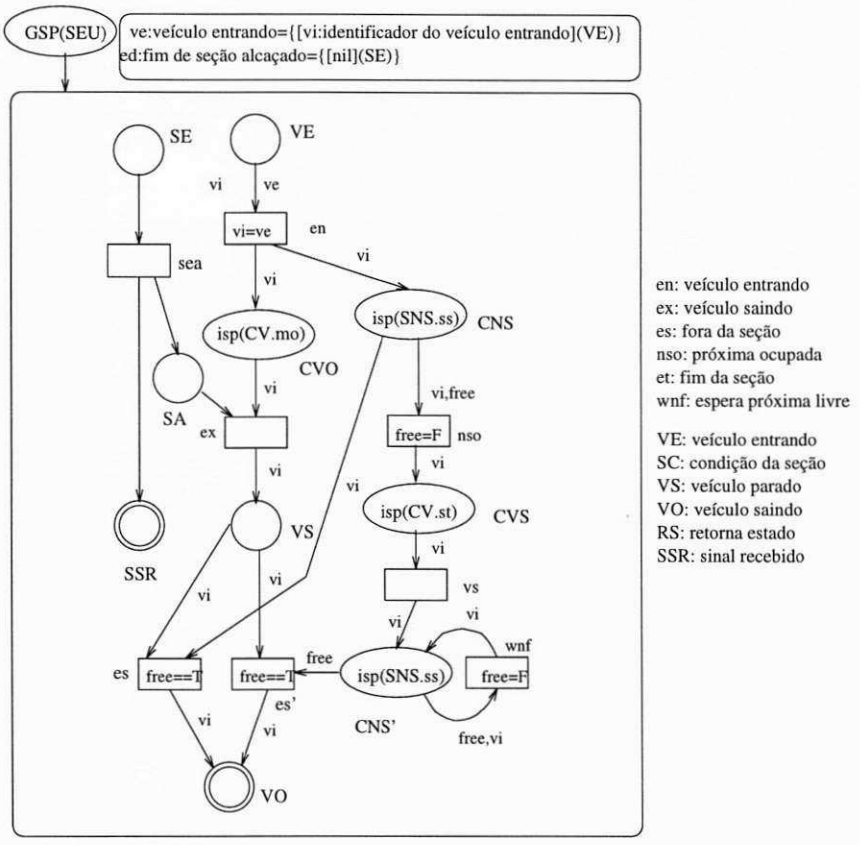

Figura 3.26: A *G-Net* modificada modelando a seção unidericional

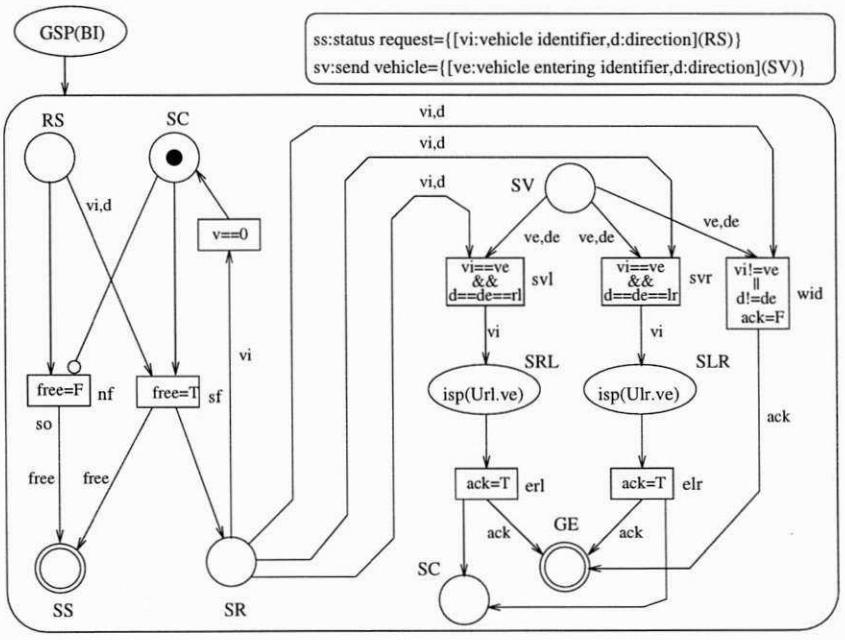

Figura 3.27: Sistema de *G-Nets* modelando a seção bidirecional

A seguir descrevemos o comportamento da *G-Net* $G(BI)$. Quando $G(BI)$ é invocada com método ss, uma ficha (com parâmetros vi e d) é recebida, então duas fichas são depositadas no lugar RS, uma para vi e outra para d. Se a seção está ocupada (o lugar SC não está marcado) a transição so dispara e atribui-se falso à variável $livre$. A ficha atinge o lugar alvo SS e a ficha é retornada, informando que a seção está ocupada. No caso em que o lugar SC, a transição sf dispara, e uma ficha é removida do lugar SC, e atribui-se verdadeiro à variável $livre$. Ainda mais, vi e d são depositadas no lugar SR, indicando que a seção está reservada para o trem vi, com direção permitida d. A ficha com $livre$ =verdadeiro alcança o lugar alvo SS, e é retornada à rede que invocou. Se $vi = 0$, isto significa que somente o estado da rede é necessário. Então a transição so dispara, removendo vi e d do lugar SR, e depositando uma ficha em SC, indicando que a seção está livre. Quando o método sv é invocado, uma ficha ve indicando o identificador do trem entrando, e outra indicando a direção do trem de, são depositadas no lugar SV. Neste ponto tres casos diferentes podem ocorrer. Primeiro, se $vi \neq ve$ e $d \neq de$, a variável ack (reconhecido) é tomada *falsa*, a transição wid dispara e uma ficha com $ack = falso$ é retornada, indicando que a requisição não pode ser atendida. Além disto uma ficha vi e d são depositadas de volta no lugar SR. No caso em que $vi = ve$ e $d = de = rl$ $(d = de = lr)$ a transição svl (svr) dispara e a *G-Net* para o controle do movimento da direita para a esquerda é invocada, $isp(Url.ve)$ (esquerda para a direita, $isp(Ulr.ve)$). Quando a rede $G(Url)$ $(G(Ulr))$ termina sua execução, as transições erl (elr) disparar, e uma ficha ack, é retornada. Também após o disparo da transição erl (elr) uma ficha é depositada no lugar SC, para indicar que a seção está livre.

Como pode ser visto pelas Figuras 3.23, 3.26 e 3.27 a concepção multi-nível é muito mais eficiente para a concepção de um sistema complexo, do que a abordagem por encapsulamento. Esta conclusão considera somente os aspectos de concepção e legibilidade. A abordagem multi-nível provará se mais eficiente ainda para análise e para introdução de tolerância a falhas.

Capítulo 4

DECOMPOSIÇÃO E ANÁLISE DE SISTEMAS DE G-NETS

4.1 Introdução

Neste capítulo introduzimos uma abordagem de decomposição para *G-Nets*. A abordagem de decomposição que será introduzida evita problemas estruturais na interface entre *G-Nets*, e possibilta análise formal. Para esta decomposição permitir a composição de *G-Nets* de forma estruturada, temos que definir mecanismos pelos quais seja possível considerar diferentes componentes de um modelo, com base em uma estrutura rigorosa, permitindo ao projetista um melhor controle sobre o gerenciamento da complexidade de concepção de um sistema. Deste modo então, diferentes partes de um modelo podem ser independentemente consideradas. Mais ainda, análise, reuso, e correção deve ser localizada e executada ao nível de componentes, desde que a interface entre estes componentes mantenha-se imutável [83]. Para poder usufruir das vantagens e benefícios de uma abordagem modular, um componente deve apresentar as seguintes características:

- A visão externa de um componente deve ser fracamente acoplada, de modo que a independência entre componentes pode ser a mais alta possível e somente alguns relacionamentos bem definidos são permitidos.

- Externamente um componente deve apresentar um alto nível de coesão funcional, de forma que sua contribuição e participação no sistema como um todo seja claramente definida.

Como discutimos no Capítulo 3, *G-Nets* e sistemas de *G-Nets* apresentam as características acima. De modo a obter vantagem dos conceitos acima, os quais são inerentes às *G-Nets*, temos que definir um protocolo de alto-nível, o qual determina como *G-Nets* são conectadas (interagem). Sistemas de redes de Petri podem ser conectados tanto pela fusão de transições, fusão de lugares, ou por arcos [16]. Como é bem conhecido, a fusão de transições facilita a análise do modelo, pois muitas propriedades são preservadas neste caso [103, 108]. Entretanto, as redes ficarão fortemente acopladas, não sendo

possível então considerar cada rede como um sistema autônomo, principalmente pelo fato de que seus comportamentos serão *misturados*, e em conseqüência não é possível decidir localmente se uma transição estará ou não habilitada, com base apenas no conhecimento sobre o estado local da rede em questão. Composição de redes pela fusão de lugares corresponde a comunicação pelo compartilhamento de variáveis. Apesar de que este método de composição não apresenta as mesmas desvantagens da fusão de transições, é necessário prover-se sincronização adicional entre as redes fundidas. Esta sincronização adicional, através da adição de lugares ou transições adicionais, pode levar a necessidade de analisar o sistema como um todo novamente, principalemente pelo fato de que novas seqüências de disparo de transições podem aparecer, apesar do fato de que as seqüências de disparo anteriores são preservadas. O terceiro método para compor redes, que é através de arcos, corresponde ao mecanismo de *mensagem passante.*

Outra abordagem para compor redes é o uso de outra rede como um meio de comunicação. A vantagem de utilizar uma abordagem como esta é o fato de que restrições podem então ser impostas a rede de comunicação, ou parte das redes comunicantes, representado o meio de comunicação, ao invés de impor restrições as redes compostas. Para possibilitar este tipo de composição, as redes envolvidas na comunicação devem apresentar determinadas restrições estruturais e comportamentais, de modo a permitir preservação de comportamento adequada. A abordagem adotada nesta tese é bastante similar a esta.

Assumimos somente a crição estática de processos. Esta suposição é necessária para definir a conexão entre as redes a priori. Apesar de que esta suposição possa parecer muito restritiva, é nossa opinião que na maioria das aplicações, é possível definir a priori os processos que irão interagir. Dito isto, comecemos a definir nossa abordagem de decomposição para *G-Nets.*

Os elementos que são utilizados como base para a decomposição de uma *G-Net* são o *GSP*, *isp*, e *lugares alvo.* Estes lugares serão denominados de agora em diante de *lugares de interface.* Uma decomposição para cada um destes elementos é então introduzida. Definimos também o conceito de reentrância para uma *G-Net* decomposta. A abordagem de decomposição para *G-Nets* possibilita não somente a introdução do conceito de reentrância, mas permite também a possibilidade de aplicarmos análise formal para verificar o comportamento de uma *G-Net* em um sistema de *G-Nets.* Os aspectos de análise são considerados na Seção 4.8.

A abordagem que introduziremos para decompor *G-Nets* segue os principios básicos da abordagem orientada a objetos. Portanto, a unidade funcional das redes pode ser

definida durante a concepção, de modo que as redes são componentes provendo alguns serviços para outras redes, e aplica a outras redes os serviços oferecidos. Como discutido acima, o *GSP* de uma *G-Net* provê a abstração com a definição explícita dos métodos (serviços) disponíveis para outras redes. Além disto, os *isp's* e os *lugares alvo* possibilitam que uma *G-Net* possa ser invocada e os resultados processados possam ser retornados. Os princípios gerais que devem ser satisfeitos quando *G-Nets* estão se comunicando é bastante similar ao protocolo implemetado em arquiteturas cliente-servidor, como discutido no Capítulo 3, e consiste dos seguintes passos:

1. A *G-Net* invocadora requer um serviço.

2. As *G-Nets* invocadas aceitam ou não a requisição.

3. Quando um serviço solicitado é aceito, a rede invocada atende a invocação e provê os resultados, de outra forma a *G-Net* invocadora deve invocar novamente.

4. A rede invocadora recupera o resultado.

De fato no passo 4 o resultado processado é enviado de volta para a rede que invocou, que deve estar esperando pelo resultado. A *G-Net* invocadora pode ser vista como um cliente e a *G-Net* invocada como um servidor.

Este protocolo provê comunicação entre *G-Nets* e é muito parecido com o *rendevouz* em Ada. A aplicação deste tipo de protocolo na concepção de sistemas bem estruturados tem sido amplamente demonstrada e não será adicionalmente discutada.

Nas seções seguintes detalharemos os conceitos de *G-Net* decomposta assim como de *G-Net* decomposta reentrante. Apresentamos como os elementos de uma *G-Net* são decompostos. A decomposição e a reentrância são ilustradas utilizando o exemplo do produtor/consumidor introduzido no Capítulo 3. Na Seção 4.8 introduzimos uma metodologia composicional para a verificação de sistemas de *G-Nets*. O objetivo é aproveitar o encapsulamento natural de uma *G-Net* em um sistema de *G-Nets*. O objetivo principal desta metodologia é evitar o problema da explosão de estados quando analizando um sistema complexo. De modo a atingir este objetivo, cada *G-Net*, *módulo*, é localmente analizado para verificar suas propriedades locais, p.e., limitabilidade de algum lugar ou vivacidade de alguma transição. Considerando que cada *G-Net* interage com outras em um sistema de *G-Nets*, definimos o conceito de ambiente. Para analizar o comportamento de uma *G-Net* assumiremos o comportamento do ambiente como impondo

algumas restrições ao comportamento local. Por outro lado, cada *G-Net exportará* algumas de suas propriedades para o ambiente. Estas propriedas devem ser verificadas mais tarde, quando analizando o ambiente interagindo com a *G-Net* analizada. O princípio básico a ser adotado é a partição natural do espaço de estado do problema e aplicar um paradigma do tipo assume/garante para verificar o comportamento de um sistema de *G-Nets*. *Computation Tree Logic* (CTL) é utilizada como uma linguagem de consulta para extrair a abstração de uma *G-Net* e expressar as propriedades do ambiente.

4.2 Decomposição de G-Nets

O objetivo em decompor uma *G-Net* é o de prover um claro acoplamento estrutural entre *G-Nets*. Este acoplamento estrutural impõe restrições na comunicação entre *G-Nets*. Além disto, com a decomposição dos elementos de interface permitimos que o projetista conceba e analize o sistema de forma incremental através da adição de novos componentes. Para fazer isto, a composição de *G-Nets* deve ser uma operação estável. Portanto, a composição de uma *G-Net* invocadora com uma *G-Net* invocada deve, após a composição, ainda ser uma rede invocadora-invocada. Isto pode ser conseguido através da preservação de propriedades dos componentes, *G-Nets*, de modo que as propriedades do sistema como um todo podem ser deduzidas das propriedades de seus componentes.

A decomposição de uma *G-Net* G resulta em uma rede decomposta Gd. Nesta tese a decomposição resultará em uma *rede reentrante*. O conceito de redes reentrantes foi inicialmente introduzido em [22] e os requisitos básicos que Gd deve respeitar são estruturais, p.e., restrições gráficas, e comportamentais, p.e., espaços originais (*home spaces*) e reversibilidade [77]. A definição de reentrância para *G-Nets* é apresentada com detalhes na Seção 4.6.1.

Antes de definirmos a decomposição dos elementos de uma *G-Net* introduziremos formalmente o conceito de *G-Net* decomposta.

Definição 4.1 *G-Net decomposta*
A decomposição de uma *G-Net* G é a tupla $Gd = (IS; \mathcal{GSP}; \mathcal{GP}; \mathcal{ISP})$, onde

IS é a estrutura interna representada por uma rede de Petri modificada.

$\mathcal{GSP}$ é a decomposição de um *GSP*.

$\mathcal{GP}$ é um conjunto de lugares alvo.

$\mathcal{ISP}$ é a decomposição do conjunto de lugares alvo.

Como o leitor pode notar a tupla formada por $(\mathcal{GSP}; \mathcal{GP}; \mathcal{ISP})$, corresponde aos elementos involvidos na interação entre *G-Nets*. A decomposição de um GSP $(\mathcal{GSP})$ define que lugares da estrutura interna recebe fichas quando uma *G-Net* é invocada com um método definido. O conjunto de lugares de chaveamento para instânciação $(\mathcal{ISP})$ define os lugares responsáveis por iniciar a comunicação com outra *G-Net* e receber o resultado de volta, quando existente. O conjunto de lugares alvo $(\mathcal{GP})$ representa os lugares que devem ser marcados quando a invocação de uma *G-Net* é completada, portanto estes lugares são resposnsáveis por enviar os resultado a quem invocou. Estes conjuntos de lugares são os lugares de interface de uma *G-Net*.

4.3 Decomposição dos elementos de uma G-Net

Nesta seção introduzimos a decomposição para cada um dos elementos de uma *G-Net*.

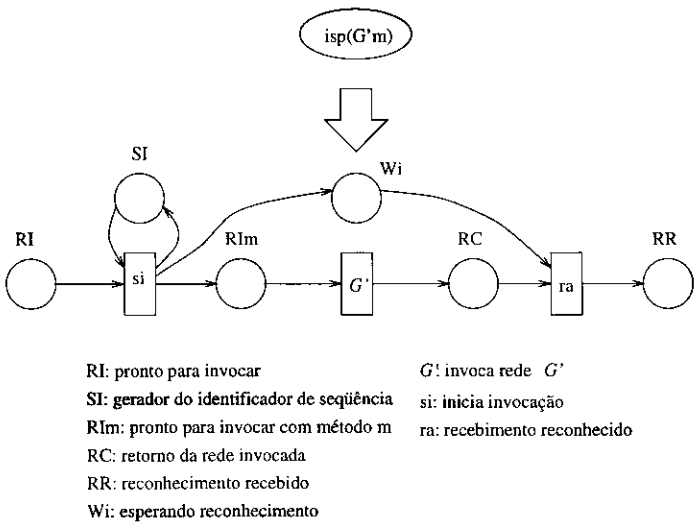

Figura 4.1: Decomposição de um $isp(G'm)$

Um $isp(G'm)$, representando a invocação de uma rede G', usando o método m será decomposto como mostrado na Figura 4.1. O lugar *RI* recebe uma ficha da(s) transição(ões) de entrada, que não está(ão) representada(s) na Figura 4.1. A transição *si* dispara e o campo *seq* da ficha é atualizado, através de um identificador único que, neste caso, é um valor inteiro n, da ficha no lugar *SI*, a qual é incrementada toda vez que *si* dispara. A ficha de saída é, então, colocada no lugar *RIm*. Esta ficha terá então um identificador único mais uma mensagem definindo a marcação inicial da rede G'. Além disto, uma ficha será depositada em *Wi*. A transição *G'* pode então disparar. Este disparo corresponde à invocação da rede G'. Quando a rede G' atinge seu *lugar*

alvo, a(s) ficha(s) retornando é(são) depositada(s) no lugar RC e a transição ra dispara, resultando em uma ficha no lugar RR. A função deste lugar é manter a informação sobre o estado da rede que está executando a invocação e é usado para introduzir propriedades de tolerância a falhas e recuperação em caso de erro.

No caso do $GSP(G')$, a decomposição é mostrada na Figura 4.2. O lugar inicial, SP, recebe as fichas correspondentes à marcação inicial relativa ao método m. A transição correspondente ao método especificado disparará e as fichas serão depositadas no conjunto de lugares iniciais, que é representado pelos retângulos pontilhados na Figura 4.2.

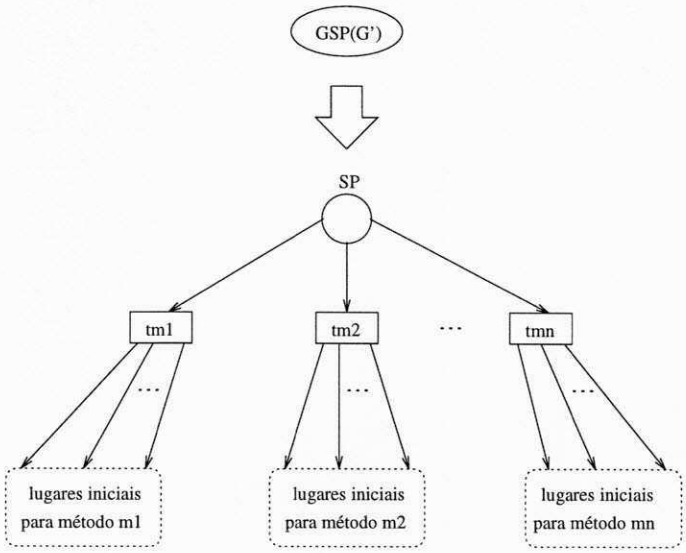

Figura 4.2: Decomposição de um $GSP(G')$

Para completar a decomposição de uma *G-Net*, é necessário decompor ou transformar os lugares normais e transições. No que se segue, apresentaremos a decomposição introduzida em [35, 38].

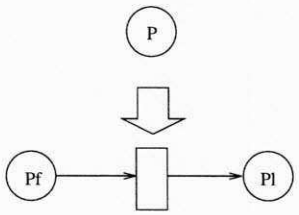

Figura 4.3: A transformação de um lugar normal

A Transformação de lugares normais e transições é bastante intuitiva. Um lugar normal em uma *G-Net* pode ser visto como um mapeamento, denotado por $A(p)$, entre o conjunto de parâmetros de entrada $I(p) = \langle seq, x_1, x_2, \ldots, x_q \rangle$, (carregado pela ficha

de entrada, uma ficha de entrada é aquela que $sc =$ após), e o conjunto de parâmetros de saída $O(p) = \langle seq, y_1, y_2, \ldots, y_r \rangle$, o qual é associado à ficha de saída. Tal lugar pode ser facilmente transformado para uma representação em *PrT-net* como mostra-se na Figura 4.3. Observe que a seqüência de propagação da ficha permanece inalterada. Um transição, **t**, em uma *G-Net* pode ser facilmente transformada para a *PrT-net* como mostrado na Figura 4.4.

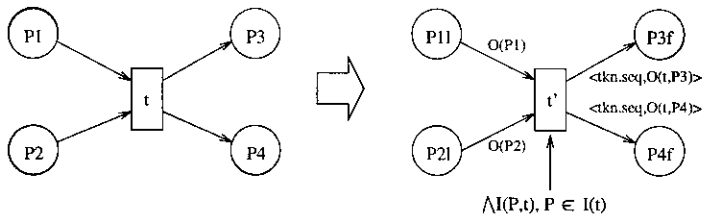

Figura 4.4: Transformação de uma transição

Deve ser enfatizado que um lugar normal deve ser decomposto somente se uma primitiva é associada a ele. De outra forma ele somente representa proposições ou predicados, de modo não são necessárias outras decomposições.

4.4 Decomposição do Modelo Produtor/Consumidor

Com o objetivo de ilustrar a metodologia de decomposição que introduzimos, mostramos a decomposição do modelo para o produtor/consumidor. Na Figura 4.5, apresenta-se a decomposição do produtor. Neste caso, a rede $G(P)$, mostrada na Figura 3.3, resulta na rede $Gd(P)$ mostrada na Figura 4.5. Uma vez que para o produtor só há um método definido, qual seja *produza*, o $GSP(P)$ tem somente uma transição, que é *mp*. Note que os lugares *W1* e *W2*, obtidos da decomposição de $isp(C.ma)$ e $isp(C.mc)$, quando marcados representam que $G(P)$ está esperando pela resposta de $G(C)$.

Na Figura 4.6, apresenta-se a decomposição da *G-Net*, modelando o produtor. Neste caso, a rede $G(C)$, mostrada na Figura 3.4, resulta na rede $Gd(C)$, como mostra-se na Figura 4.6. Para esta rede, dois métodos são definidos: verifica estado (ms) e consome (mc). Então, quando decompondo $GSP(C)$, obtêm-se duas transições que são *ms* e *mc*. Quando a transição *ms* dispara, uma ficha é depositada no lugar *IN*. Desta forma, a parte da rede responsável por verificar e retornar o estado do consumidor modelado pela rede $Gd(C)$ é executada. De outra forma, uma ficha é depositada no lugar *TC*, que correponde à parte responsável pelo consumo de um determinado ítem.

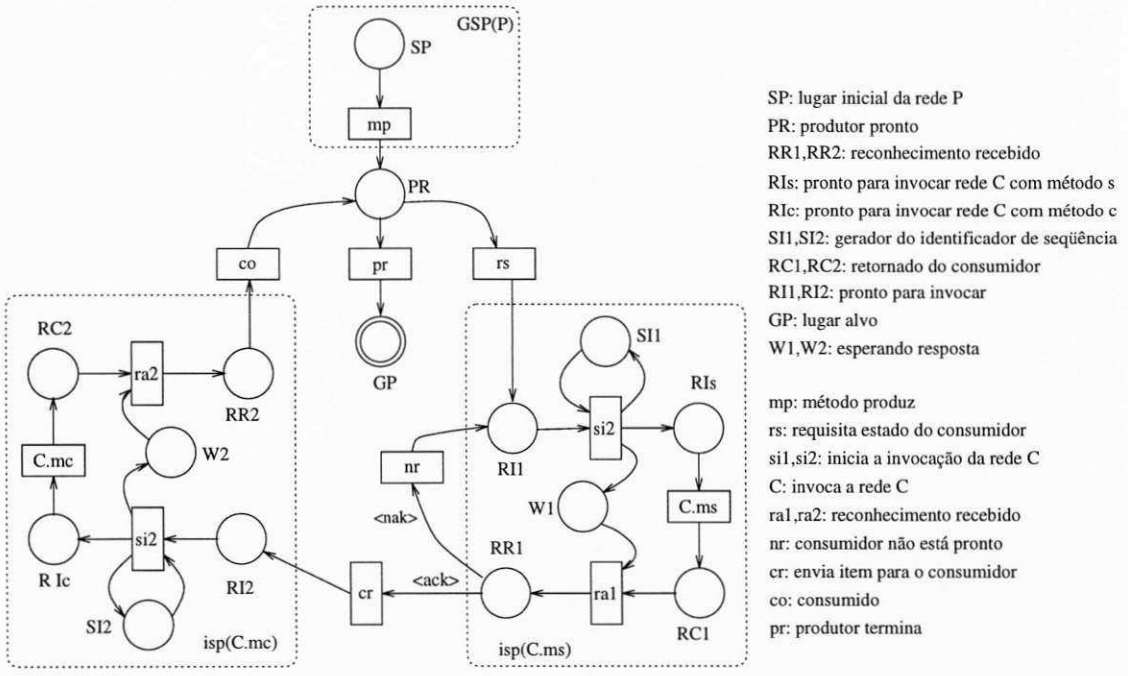

Figura 4.5: Decomposição de $G(P)$ modelando o consumidor resultando em $Gd(P)$

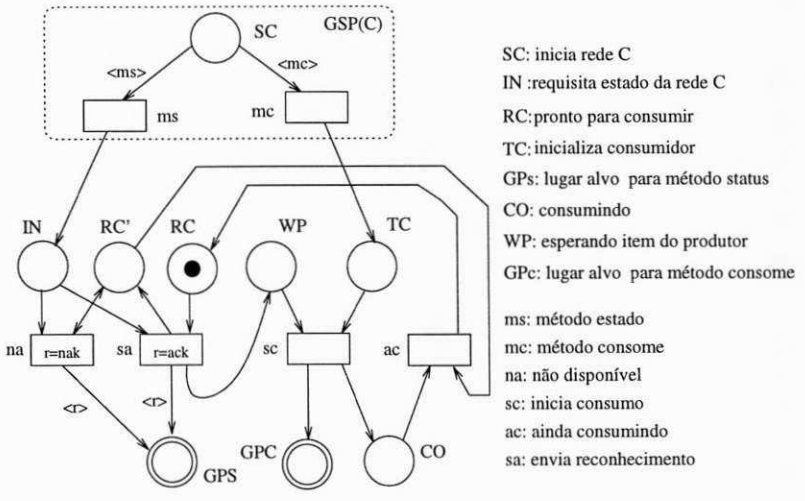

Figura 4.6: Decomposição de $G(C)$ modelando o consumidor resultando em $Gd(C)$

4.5 Estruturando a Comunicação Entre-G-Nets

Antes de definir a comunicação entre *G-Nets*, apresentaremos alguns dos conceitos introduzidos no Capítulo 2, bem como alguns novos conceitos e notações que serão utilizados ao longo do Capítulo.

Considerando uma *G-Net* decomposta Gd e uma marcação inicial M_0, definida por $Gd = (IS; \mathcal{GSP}; \mathcal{GP}; \mathcal{ISP}; M_0)$ (veja a difinição 4.1), as seguintes notações e definições serão utilizadas.

Definição 4.2 *Número de Ocorrências de uma Transição*
Dada uma seqüência de disparo de transições $\sigma \in T^*$, $\eta(\sigma, t)$ é o número de ocorrências da transição t na seqüência de disparos da transição σ.

Definição 4.3 *Sucessor de uma Marcação*
$M \longrightarrow M'$ se existe $\sigma \in T^*$ tal que $M[\sigma\rangle M'$.

Definição 4.4 *Conjunto de Transições Disparáveis Restrito a uma Marcação*
Dada uma *G-Net* decomposta Gd, $L(Gd, M) = \{\sigma \in T^*; M[\sigma\rangle\}$, e $L(Gd, M)|T' = \{\sigma|T'; \sigma \in L(Gd, M)\}$

Definição 4.5 *Prefixo de uma Seqüência de Disparos*
Seja $\sigma \in T^*$ e $T' \subset T$, então:

$\sigma \mid T' \in T^*$ é a subcadeia de σ obtida através da remoção dos elementos $T \backslash T'$.

se σ, $\sigma' \in T^*$, $\sigma' \leq \sigma$ se σ' é um prefixo de de σ, i.e. se existe um $\sigma'' \in T^*$ tal wue $\sigma'.\sigma'' = \sigma$.

Definição 4.6 *Transições livres de conflito em conjunto*
Seja $\mathcal{T}_{m1}$, $\mathcal{T}_{m2}$, e $Gd = (IS; \mathcal{IP}; \mathcal{GP}; \mathcal{ISP})$, com $\mathcal{T}_{m1} \in IS$ e $\mathcal{T}_{m2} \in IS$, $\mathcal{T}_{m1}$, e $\mathcal{T}_{m2}$ ser um conjunto de transições pertencendo ao $método_1$ e $método_2$ respectivamente, estes dois conjuntos de transições são denominados *livre de conflito em conjunto* se e somente se

$\bullet t \cap \bullet t' = \emptyset$, com $t \neq t'$, $\forall\, t \in \mathcal{T}_{m1}$ e $\forall\, t' \in \mathcal{T}_{m2}$.

Informalmente a Definição 4.6 garante que cada método em uma *G-Net* não está em conflito com outros.

4.6 Propriedades Estruturais e Comportamentais

Como discutido anteriormente, para permitir a composição de *G-Nets* temos que introduzir algumas restrições estruturais e comportamentais às redes sendo compostas. Introduziremos e discutiremos primeiro o conceito de reentrância para *G-Nets*

4.6.1 Formalização do Conceito de Reentrância

Reentrância é um dos conceitos principais que introduzimos às *G-Nets*. O objetivo de introduzir reentância em *G-Nets*, é permitir preservação de propriedades quando compondo *G-Nets*. As propriedades básicas que devem ser satisfeitas por uma *G-Net* decomposta são reversibilidade[1] e estados originais [77].

Definição 4.7 *G-Net Decomposta Reentrante*
Seja G uma *G-Net* e $Gd = (IS; \mathcal{GSP}; \mathcal{GP}; \mathcal{ISP})$ ser sua decomposição. Gd é *reentrante* se e sómente se:

P1 $\bullet SP = \emptyset$, com $SP \in \mathcal{GSP}$, isto é, não existem arcos para um SP.

P2 $\forall GP_i \in \mathcal{GP}, GP_i\bullet = \emptyset$, isto é, não existem arcos saindo do lugar alvo.

P3 Para cada dois métodos m_i e m_j temos que, $GSP_{m_i} \neq GSP_{m_j}$, isto é, cada método definido para uma *G-Net* termina em um lugar alvo diferente.

P4 $\forall GSP_{m_i} \in \mathcal{GP}$, $\forall \sigma_{m_i} \in T^* : M_0[\sigma_{m_i}\rangle M(GSP_{m_i})$, isto é, para um dado método seu lugar alvo é sempre alcaçável, independentemente da seqüência de disparos.

P5 É sempre o caso que a rede seja reversível.

P6 Os conjuntos de transições pertencendo a cada método são livres de conflito em conjunto.

Se as propriedades P1 a P6 são verificadas em uma *G-Net* decomposta, dizemos que ela é reentrante. Obviamente a propriedade P1 é sempre verificada, uma vez que esta

[1] Como discutido no Capítulo 2, uma rede de Petri (N, M_0) é reversível, se para cada marcação M em $R(M_0)$, M_0 é alcançável a partir de M, onde $R(M_0)$ é o conjunto de todas as marcações alcançáveis a partir de M_0 para a rede (N, M_0). Portanto, em uma rede reversível é sempre possível alcançar a marcação inicial. Uma marcação M' é denominada estado original se para cada M em $R(M_0)$, $M^{'}$, $M^{'}$ é alcaçável a partir de M.

é parte do GSP decomposto na rede original. A seguir discutiremos cada uma das propriedades introduzidas e discutiremos suas importâncias para a definição de *G-Net* decomposta reentrante.

A Propriedade **P1** garante que quando uma *G-Net* G é invocada com método m_i somente os lugares iniciais associados ao método serão marcados quando a transição $t_{m_i} \in T_{methods}$ dispara na *G-Net* decomposta Gd, veja Figura 4.2.

A Propriedade **P2** garante que a *G-Net* decomposta Gd não tem mais transições a disparar, para um determinado método, quando o lugar alvo é alcançado, para uma invocação específica identificada pelo campo *tkn.seq* associado a ficha.

A Propriedade **P3** restringe a existência de um e somente um lugar alvo para cada método. Esta restrição é necessária para garantir que o resultado é sempre devolvido para a rede invocadora correta. Além disto, como veremos na Seção 4.8, esta restrição é também necessária para a abordagem de análise que será introduzida.

A Propriedade **P4** garante que a invocação de uma *G-Net* decomposta sempre termina, e considerando as propriedades **P2** e **P3**, o resultado sempre é devolvido ao invocador.

A Propriedade **P5** é muito importante, pois deve ser sempre o caso em que uma *G-Net* decomposta sempre retorna ao seu estado inicial, de modo que uma nova invocação para um determinado método sempre ocorre.

A Propriedade **P6** tem imporância crucial para a metodologia de análise que será introduzida na Seção 4.8. Ela garante que as execuções dos métodos definidos em uma *G-Net* decomposta podem ser tratadas independentemente.

Baseando-se na definição de reentrância podemos definir claramente como a comunicação entre *G-Nets* ocorrerá. O conjunto de elementos de interface ($\mathcal{IE}$) implementa esta função.

Com base na decomposição de um *isp*, como mostrado na Seção 4.2, definimos o conceito de requisição de serviço, ou simplesmente *G-Net* invocadora como segue.

Definição 4.8 *G-Net invocadora*
Seja $GdR = (IS_R; \mathcal{GSP}_R; \mathcal{GP}_R; \mathcal{ISP}_R)$ uma *G-Net* decomposta invocadora GdS, e que $isp_i \in GdR$ é um *isp* invocando GdS com método m_{GdS_i}, temos que:

1. Para cada invocação de GdS com método m_i através do $isp_i \in GdR$, existe um lugar de invocação de retorno $W_i \in isp_i$ tal que:

   $W_i \in si\bullet$, $\bullet W_i = \{si\}$, e
   $W_i \in \bullet ra$, $\bullet ra = \{W_i\}$.

2. M é uma marcação de GdR tal que $M(W_i) = 0$ para qualquer lugar de invocação de retorno W_i.

O lugar de espera W_i de um *isp* corresponde ao estado da *G-Net* invocadora decomposta GdR, está esperando pelo resultado da *G-Net* decomposta de serviço GdS, representada pela transição G' no *isp*, para completar o serviço requisitado e devolver o resultado.

É muito importante que definamos quando um *isp* completará sua execução, representando que o serviço requisitado foi completado com sucesso, e como o controle é retornado a IS (estrutura interna) do invocador.

Lema 4.1 *Terminação Clara de uma G-Net decomposta*
Uma *G-Net* decomposta termina claramente sua execução, independentemente do número de invocações.

Prova.
A prova é direta, baseando-se nas propriedades de reentrância P4, P5, e P6 na Definição 4.7, na Definição 4.6 e no fato de que a seqüência *tkn.seq* é única. □

Proposição 4.1 *Terminação do isp*
Sejam GdR, e GdS serem *G-Nets* decompostas invocadora e invocada repectivamente, e dado $isp_{RS} \in P_{GdR}$ ser um *isp* invocando GdS com método m_i. Se GdS é reentrante o *isp* sempre recebe o resultado de retorno.

Prova.
A prova da proposição 4.1 é imediata. Considerando as seqüências de disparo para as *G-Nets* invocadora e invocada, temos que:

1. $\sigma_r = \sigma_{br}.\sigma_{si}.\sigma_{G'}.\sigma_{ra}.\sigma_{ar}$.

2. $\sigma_{G'} = \sigma_c$.

onde:

σ_r é um conjunto de seqüências de disparo de transições livre de conflito de GdR.

σ_c é um conjunto de seqüências de disparo de transições livre de conflito de GdS para alcançar um lugar alvo em GdS para o método invocado m_i.

$\sigma_{G'}$ é a abstração do conjunto de seqüências de disparo de transições livre de conflito de *GdS*.

σ_{br} é a seqüência de disparo de transições em *GdR* antes que *GdS* retorne o resultado.

σ_{ar} é a seqüência de disparo de transições em *GdR* após *GdS* retornar o resultado.

σ_{si} é o disparo da transição *si* no $isp_i \in GdR$.

σ_{ra} é o disparo da transição *ra* no $isp_i \in GdR$.

G' é a abstração da execução da rede decomposta *GdS*.

2. Na Proposição 4.1 pode ser garantido pela Propriedade **P3** na Definição 4.7 e pelo Lema 4.1, de modo que podemos reescrever que:

$$\sigma_r = \sigma_{rb}.\sigma_{si}.\sigma_c.\sigma_{ra}.\sigma_{ar} \tag{4.1}$$

A expressão acima garante que quando um *isp* é alcançado a *G-Net* respectiva é invocada com método m_i, e que o resultado é sempre retornado, de modo que o *isp* retorna o controle para a rede invocadora. □

Uma *G-Net* invocada oferece serviços aos invocadores, então ela aceita requisições, processa estas requisições e provê os resultados. Os lugares de inteface de uma rede invocada casa com os lugares da rede invocadora, isto é ela casa com os *isp's* decompostos da invocadora. O lugar inicial (SP) no GSP e os lugares alvo são utilizados para ser fundidos com os lugares *si* e *ra*, no *isp* decomposto quando substituindo a G' no *isp*.

Considerando que a rede invocada de serviço ou simplesmente *G-Net* de serviço é reentrante podemos definir a rede de serviço.

Definição 4.9 *G-Net de Serviço*

Seja *GdS* uma *G-Net* decomposta, com lugar inicial SP e lugar alvo GP_{m_i} para o método $method_i$, *GdS* é uma rede de serviço se e somente se para todos os métodos definidos temos:

1. *GdS* é reentrante.

2. Existe um invariante de lugar mínimo I tal que $I(SP) = I(GP_{m_i}) = 1$, $\forall\ m_i \in$ *método.*

3. M_0 é uma marcação de GdS tal que $\forall\, p \in P[I(p)\rangle 0 \Longrightarrow M_0(p) = 0]$.

4. Para qualquer marcação M e M' tal que $M(p) = M_0(p)$ para $p \in P\backslash\{SP\}$ para $M \longrightarrow M'$, existe M'' tal que $M' \longrightarrow M''$, $M''(SP) = 0$ e $M''(GP_{m_i}) = M(SP)$, $\forall m_i \in$ *métodos*.

Na definição acima $P \in IS_{GdS}$. A condição 4. na Definição 4.9 garantes que a o identificador de seqüência da ficha enviada é seguramente propagado através da rede de serviço, e atinge o lugar alvo com a mesma seqüência de propagação.

Uma *G-Net* de serviço decomposta pode depositar fichas nos seus lugares alvo somente após remover a ficha do lugar inicial, e ela pode mover qualquer número de fichas do seu lugar inicial para os lugares alvos.

Proposição 4.2 *Terminação da Rede de Serviço*
Seja $GdS = (IS; \mathcal{GSP}_S; \mathcal{GP}_S; \mathcal{ISP}_S; M_0)$ ser uma *G-Net* de serviço decomposta, M ser uma marcação de GdS tal que $M(SP) > 0$ e $M(p) = M_0(p)$ para $p \in P\backslash\{SP\}$, m_i e o método invocado com a transição t_{m_i} como a transição de método, $\bullet GP_{m_i} = \{tGP_{m_i}\}$ a transição de entrada de um lugar alvo associado ao método m_i, e $\sigma \in L(GdS, M)$.

1. $\eta(\sigma, tGP_{m_i}) \leq \eta(\sigma, t_{m_i})$.

2. $\exists\, \sigma' \in T^*[\sigma.\sigma' \in L(GdS, M) \wedge \eta(\sigma.\sigma'(tGPm_i \leq \eta(\sigma, t_{m_i})]$.

Prova.
1. Considerando que $I.M_0 = 0$, temos $I.M = I.M(SP)$. Seja M' ser tal que $M[\sigma\rangle M'$. Então $\eta(\sigma, t_{m_i}) = M(GP_{m_i}) - M'(GP_{m_i}) = I.M - M'(SP) = I.M' - M'(SP) \geq M'(GP_{m_i}) = \eta(\sigma, t_{m_i})$.
2. resulta do ponto 4) na Definição 4.9, desde que $M''(GP_{m_i}) = M(GP_{m_i})$ é equivalente a $\eta(\sigma.\sigma', tGP_{m_i}) = \eta(\sigma.\sigma', t_{m_i})$, e podemos considerar que $M(SP) = \eta(\sigma, t_{m_i}) = \eta(\sigma.\sigma', t_{m_i})$. □

Podemmos ver uma *G-Net* de serviço como um conjunto de redes provendo serviços (definidos pelos métodos) juntamente com os seus lugares de interface. Portanto, os lugares de interface juntamente com as transições de método, e os lugares alvo, e suas respectivas transições, coordenam a aceitação de uma requisição e como a informação resultante é enviada a rede invocadora.

4.7 Relação de Equivalência

A relação de equivalência entre duas *G-Nets* decompostas é muito importante, pois baseado-se nesta relação, refinamento, abstração, e reusabilidade de uma *G-Net* concebida são facilitados. Aqui, estamos falando a respeito de equivalência de interface, isto é, desde que duas *G-Nets* decompostas são *equivalentes a nível de interface* elas podem ser utilizadas como parte da concepção de um sistema, mantendo a mesma visão externa, considerando-se somente os aspectos estruturais.

Definição 4.10 *Equivalência de Interface*
Sejam Gd_1 e Gd_2 duas *G-Net* decompostas satisfazendo as propriedades de reentrância. Gd_1 e Gd_2 são equivalentes a nível de interface se e somente se:

1. $\mathcal{IP}_{Gd_1} = \mathcal{IP}_{Gd_2}$,

2. $\mathcal{GSP}_{Gd_1} = \mathcal{GSP}_{Gd_2}$, e

3. domínio($\mathcal{IP}_{Gd_1} \cup \mathcal{GSP}_{Gd_1}$)= domínio($\mathcal{IP}_{Gd_2} \cup \mathcal{GSP}_{Gd_2}$).

Na definição acima as condições 1 e 2 indicam que as redes tem os mesmos lugares iniciais e lugares alvo, isto é, são compatíveis estruturalmente. A condição 3 garante que as interfaces das redes possuem o mesmo domínio. Estas três propriedades garantem que as redes são intercambiáveis a nível de interface. O leitor deve observar que até este ponto não discutimos se as redes possuem comportamentos equivalentes, este ponto será discutido na Seção 4.8.

4.8 Análise de Sistemas de G-Nets

Com o objetivo de análise, um sistema de *G-Nets* será considerado como a composição de um certo número de *G-Nets*. Considerando o objetivo de evitar a explosão de estados, um método evidente para analisar um sistema de *G-Nets* é decomposição. O objetivo é verificar propriedades de componentes individuais, verificar se estas propriedades são válidas para o sistema como um todo e usá-las para deduzir propriedades adicionais do sistema.

A metodologia que será apresentada combina análise lógica e comportamental. A análise comportamental é aplicada para verificar o comportamento local. Neste passo podemos considerar tanto a análise por invariantes ou a análise através da árvore de alcançabilidade. Preferiremos a aplicação da análise baseada na árvore de alcançabilidade

pela razão de que podemos extrair da árvore de alcançabilidade o comportamento externo ou observado de uma *G-Net*.

A análise lógica baseia-se no paradigma *assume/garante* [88] para sistemas de transição de estado, como discutido no Capítulo 2. Esta idéia foi empregada com sucesso em trabalhos mais recentes como [29, 53, 54, 62, 63, 64, 66].

Vemos uma *G-Net* decomposta *Gd* como sendo constituída de duas partes: a interface e sua realização interna, como discutimos na Seção 4.2. A interface descrve o comportamento visível de fora, a abstração, e a segunda parte descreve como os métodos definidos para uma dada *G-Net* são implementados através de redes de Petri. Como uma *G-Net* interage em um sistema de *G-Nets*, o comportamento de uma *G-Net*, ou sua visão decomposta, depende da reação de outras *G-Nets*, que formam seu ambiente. Logo, a interface ou abstração pode conter algumas restrições a respeito das reações esperadas do ambiente. Portanto, as interfaces são definidas por um par (*assm*, *comm*), onde *comm* descreve as propriedades dos serviços que serão garantidas pela *G-Net*, desde que o ambiente satisfaça as suposições definidas em *assm*. Como a interface ou abstração é o comportamento visível ou observado de uma dada *G-Net*, a parte interna (sua realização) deve satisfazer este comportamento. Portanto, no processo de concepção, a correção da implementação deve ser verificada.

Suposições e compromissos podem ser representados por fórmulas em lógica temporal. Geralmente lógica de primeira ordem seria necessária para descrever os serviços de uma *G-Net*, uma vez que a realização dos métodos é através de uma rede PrT modificada, mas podemos separar a interação pura (protocolo), a qual pode ser descrita utilizando somente lógica temporal proposicional, da especificação do serviço em si, a qual trata com a computação de dados, onde lógica de primeira ordem é necessária. Logo, em muitos casos é possível separar a parte proposional, descrevendo a interação, e a parte de primeira ordem descrevendo as transformações de dados. Então, a análise de correção pode ser executada em dois passos, quais sejam: primeiro verifica-se a interação, p.e., através da verificação de modelos; e em segundo lugar verifica-se a correção das transformações de dados da estrutura interna pela aplicação de metodologias de análise bem conhecidas para redes PrT, como análise de invariantes e alcançabilidade.

Baseado nesta análise em dois passos, cada *G-Net* é localmente analizada, considerando restrições ambientais e propriedades globais, como *deadlock*, podem se verificadas.

De modo a analisar um sistema de *G-Nets*, propomos outro nível de decomposição além do introduzido no Capítulo 4. Neste nível, os lugares iniciais, lugares alvo e lugares de chaveamento serão desdobrados. Entende-se por desdobramento a substituição de

lugares na interface de uma *G-Net*, de modo que as inscrições associadas aos arcos possam ser eliminadas. Por exemplo, para cada inscrição nos arcos de saída de um lugar inicial para o GSP criaremos um lugar correspondente, de modo que as inscrições não serão mais necessárias. Esse processo de desdobramento é necessário para eliminarem-se as inscrições dos arcos associados aos lugares de interface. Além disso, associaremos uma proposição a cada lugar na interface de uma *G-Net*.

Para verificar se a parte de implementação satisfaz as propriedades impostas para a interação entre um *G-Net* e seu ambiente, construímos um modelo (estrutura de Kripke, como introduzido no Capítulo 2), representando a implementação, na qual a especificção ou proriedades impostas possam ser interpretadas. Tomamos o grafo-de-caso (grafo de alcançabilidade) da *G-Net* decomposta como uma estrutura de Kripke.

Nosso alvo é considerar componentes de software modelados por meio de *G-Nets*, consideramos que estas *G-Nets* são componentes assíncronos que comunicam-se por meio de ações sincronizadas como em CSP. Portanto, assumimos que não existe dependência entre as unidades de relógio dos diferentes componentes, isto é, não existe um relóogio global.

Considerando o exemplo do produtor/consumidor modelado pelas *G-Nets* $G(C)$ e $G(P)$, introduzido no Capítulo 3. Para verificar a correção de $G(C)$ e $G(P)$ empregamos o seguinte princípio.

> mostrando que $G(C)$ satisfaz $(assm_{G(P)}, spec_{G(C)})$ e $(, assm_{G(C)})$ e que $G(P)$ satisfaz $(asss_{G(C)}, spec_{G(P)})$ e $(, assm_{G(P)})$ podemos deduzir a especificação $(, spec_{G(C)} \wedge spec_{G(P)})$ para o sistema global $G(P) \parallel G(C)$.

4.8.1 Análise Local

A análise local é executada segundo o procedimento abaixo:

Passo 1. decomponha cada *isp* e GSP.

Passo 2. decomponha, se necessário, os lugares na estrutura interna de modo que transições modelem ações e lugares modelem estados.

Passo 3. verifique as propriedades da rede analisando a árvore de alcançabilidade ou análise de invariantes.

Passo 4. verifique as propriedades de reentrância na árvore de alcançabilidade.

A realização dos passos 1. e 2. já foi discutida no Capítulo 4. No passo 3., utiliza-se a análise baseada na árvore de alcançabilidade ou invariantes. O passo 4. depende da formalização do conceito de reentrância. Intuitivamente, no caso do passo 4., o que desejamos verificar é se os lugares alvos são sempre alcáçaveis, deteminar se a marcação dos lugares iniciais e finais são inicialmente vazias e verificar se a rede é reversível.

4.8.2 Análise da Interação

A análise da interação entre *G-Nets* é executada de acordo o seguinte procedimento:

Passo 1. desdobre os elementos da interface das *G-Nets* decompostas em um sistema de *G-Nets*.

Passo 2. extraia de cada rede uma estrutura de Kripke correspondente ao comportamento da interface de cada *G-Net* decomposta.

Passo 3. introduza um estado adicional para cada lugar de saída de um *isp* correspondendo aos rótulos nos arcos de saída.

Passo 4. introduza um estado adicional para cada lugar de saída de um *isp* correspondendo a cada transição disparável após o elemento de interface ser atingido.

Passo 5. verifique se as propriedades de interação, expressadas em CTL, são satisfeitas.

O objetivo do passo 4. é identificar qual transição dispara antes que um lugar de interface seja atingido. Isto ficará claro no exemplo do produtor/consumidor como será apresentado a seguir. O desdobramento de cada *G-Net* decomposta é necessário para definir o rotulamento de transições de interface e associação de proposições aos lugares de interface. O rotulamento de transições é necessário para definir um operador de composição que execute a fusão de transições com mesmo rótulo. A associação de proposições aos lugares de interface é necessária para a extração de uma estrutura de Kripke, caracterizando o comportamento externo de cada *G-Net* decomposta. O objetivo é definir uma abstração para o comportamento de cada *G-Net* decomposta e aplicar uma técnica de verificação baseado em lógica temporal, como discutido no Capítulo 3, e desta forma, verificar o comportamento composto de um sistema de *G-Nets*.

No que segue, discutiremos o desdobramento dos elementos na interface de uma *G-Nets* assim como a extração das estruturas de Kripke. Ilustraremos a apresentação, utilizando o problema produtor/consumidor.

Desdobramento dos Elementos na Interface

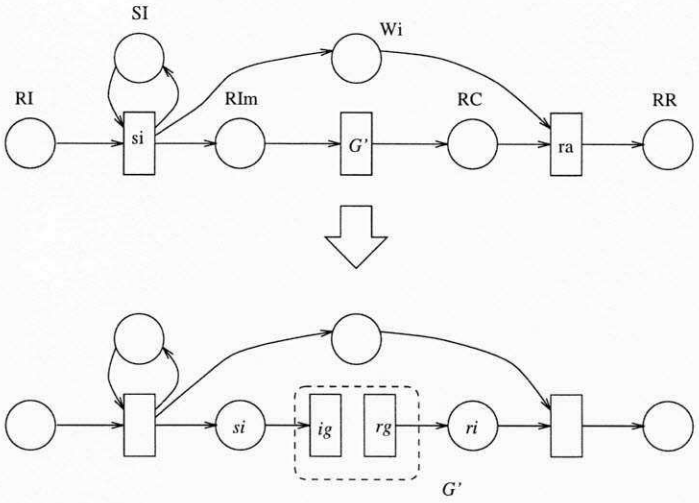

Figura 4.7: Desdobramento de G' e associação de proposições

Para o procedimento de desdobramento, definimos um conjunto de proposiçoes, PL, e um conjunto de rótulos, TL, a serem assoiciados aos lugares e transições dos elementos de interface em uma *G-Net* decomposta.

Os elementos de interface em uma *G-Net* decomposta são os lugares iniciais no GSP decomposto, os lugares alvo e a transição G' no *isp* decomposto, como introduzido no Capítulo 4.

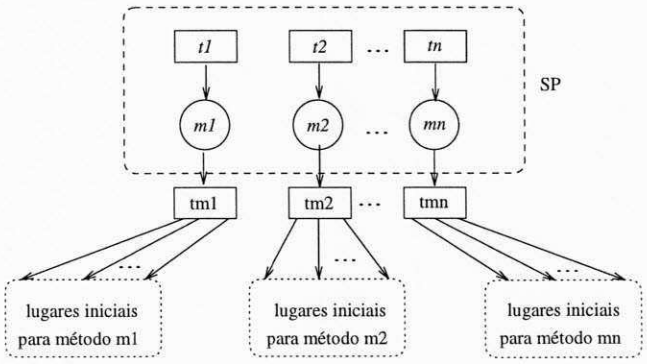

Figura 4.8: Desdobramento de um lugar inicial

Para o *isp*, a transição G' será desdobrada em duas, como mostrado na Figura 4.7.

A transição *ig* representa a invocação da rede G' e a transição *rg* representa o fim da invocação da rede G'. Para cada *isp* em um conjunto de *G-Nets* decompostas, um rótulo diferente do conjunto TL será associado às correspondentes transições *ig* e *rg*. Associamos, também, proposições, definidas em PL, aos lugares **RIm** e **RG**.

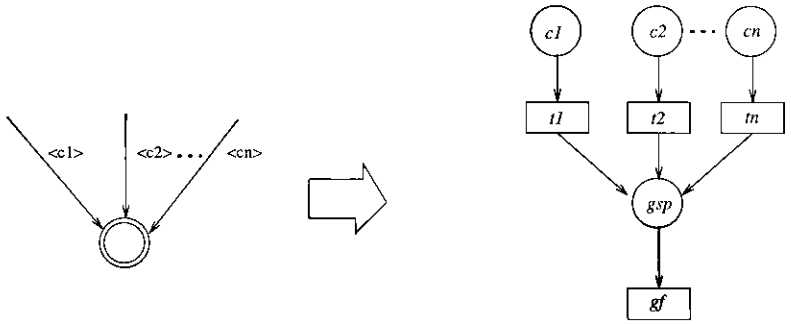

Figura 4.9: Desdobramento de um lugar alvo

Desdobramos, também, o lugar inicial em cada GSP como mostrado na Figura 4.8. O lugar inicial SP é desdobrado de acordo com as inscrições associadas aos arcos de saída. Este desdobramento resultará em um lugar para cada método. Para cada um desses lugares, associa-se uma proposição, definida em PL. Introduzimos também uma transição e um arco de entrada para cada um destes lugares. Esta transições são rotuladas com os mesmos rótulos usadas para as transições representando a invocação de uma *G-Net* decomposta. Por exemplo, a transição *t1* no GSP decomposto terá o mesmo rótulo da transição *ig* em um *isp*, que invoque a rede abstraída por este GSP.

O último desdobramento está relacionado com os lugares alvo. Na Figura 4.9, o desbramento genérico de um lugar alvo é aprsentado. Assumiremos um número arbitrário n de arcos de chegada para um lugar alvo. Neste caso, para cada método, deve-se definir um e somente um lugar alvo. Na Figura 4.9, as inscrições $\langle c1\rangle, \langle c2\rangle \cdots \langle cn\rangle$ definem qual será a mensagem associada à ficha a ser retornada. Relembrando o problema produtor/consumidor, estas inscrições podem ser para o método status tanto $\langle ack\rangle$ ou $\langle nak\rangle$, indicando se o consumidor está ou não pronto para receber um ítem para ser consumido. A decomposição ser 'a constituída de um lugar para cada condição. Cada um desses lugares terá uma proposição associada. Além disso, associa-se uma transição com arcos de chegada a cada um desses lugares e um lugar com arcos de entrada associados a estas transições. Por fim, introduz-se uma transição com um arco de entrada proveniente deste lugar intermediário. Esta transição será rotulada com o mesmo rótulo atribuído à transição *rg*, representando o fim da invocação da rede G' no *isp* decomposto.

Apresenta-se nas Figuras 4.10 e 4.11, o desdobramento dos elementos de interface,

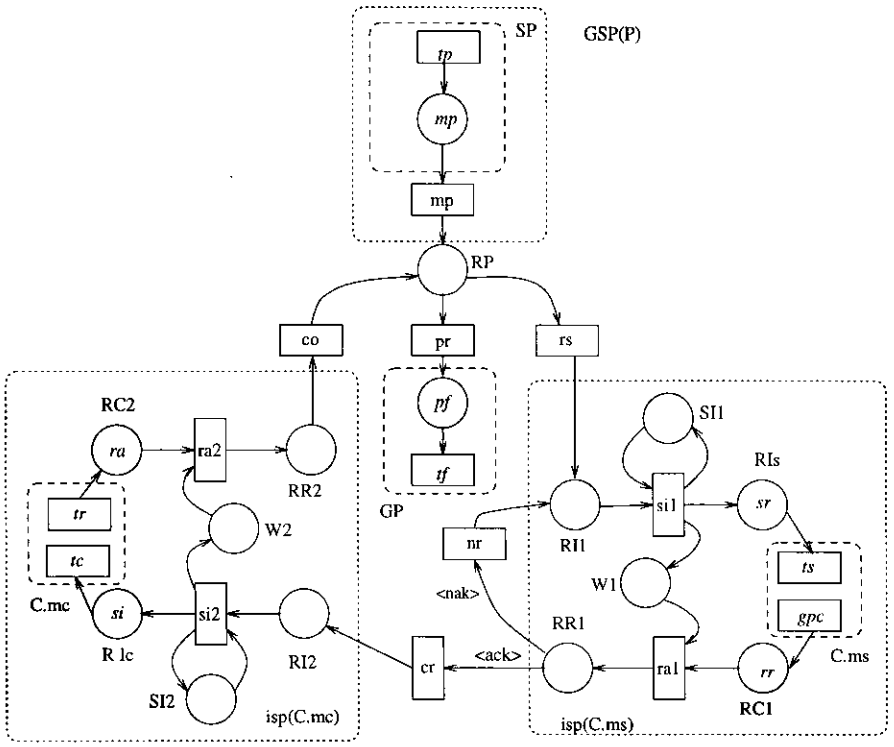

Figura 4.10: Decomposição com desdobramento para o produtor

isto é, os lugares alvo, lugares iniciais e as transições de invocação G', para o produtor e para o consumidor, respectivamente.

Análise do Comportamento de Interação

A análise do comportamento da interação tem sua base nos conceitos de lógica temporal introduzidos no Capítulo 3 e no procedimento de desdobramento previamente introduzido. As proposições associadas aos lugares de interface, como por exemplo *ms* e *nak* na Figura 4.13, são usados para gerar estruturas de Kripke representando o comportamento externo de uma *G-Net*. Estas estruturas de Kripke são então usadas para verificar as propriedades de interação entre *G-Nets* expressadas por fórmulas em lógica temporal.

Apresenta-se nas Figuras 4.12 e 4.13, as estruturas de Kripke para o problema produtor/consumidor. Estas estruturas são obtidas no passo 2. do procedimento de análise de interação. Deve-se enfatizar que os estados *nak* e *ack* são introduzidos pela aplicação do Passo 3. do procedimento de análise. Além disto, os estados *pr* e rs são aqueles introduzidos de acordo com Passo 4. do procedimento de análise. Neste caso está claro que as transições *pr* ou rs são aquelas que são disparadas antes dos lugares de inteface

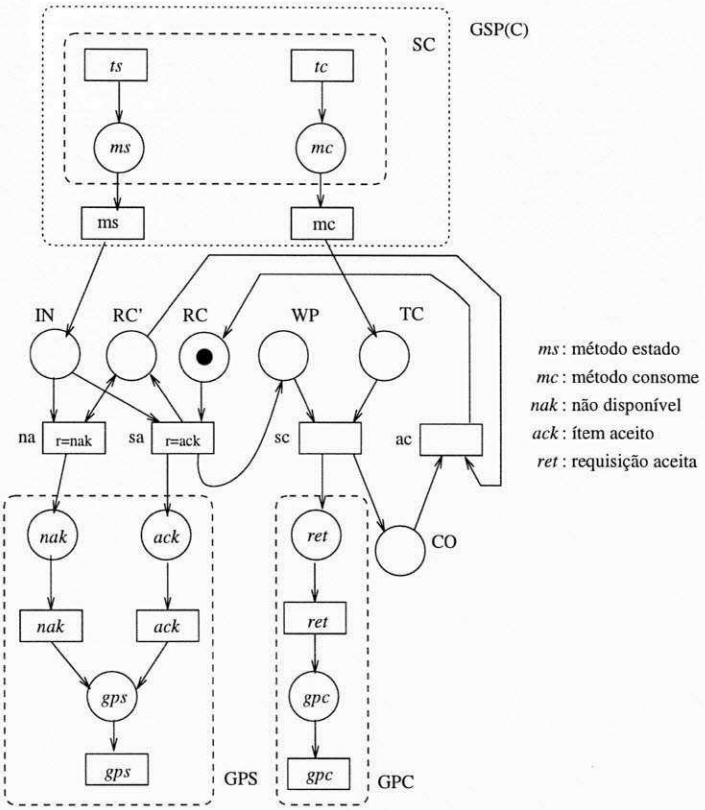

Figura 4.11: Decomposição com desdobramento para o consumidor

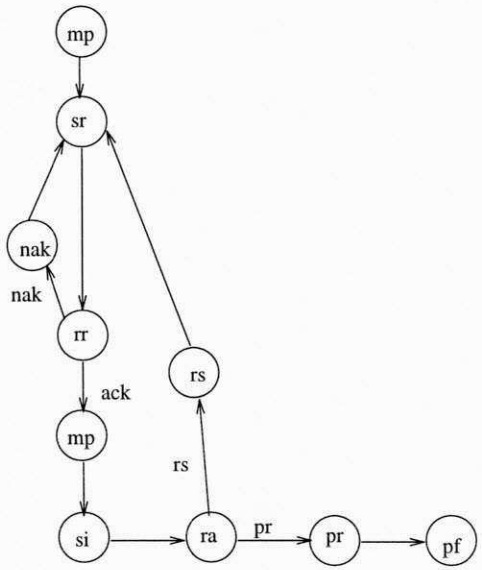

Figura 4.12: Estrutura de Kripke para o comportamento do produtor

$isp(C.ms)$ e GP são alcançados.

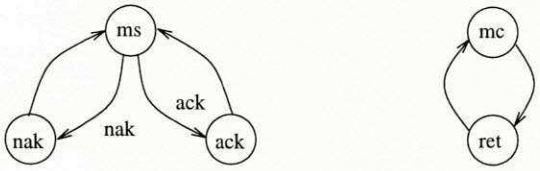

Figura 4.13: Estrutura de Kripke para o comportamento do consumidor

No caso do problema produtor/consumidor, duas propriedades definindo a interação entre o produtor e o consumidor podem ser informalmente descritas por:

somente envie um ítem se um reconhecimento é recebido

sempre após receber um reconhecimento envie um ítem

As declarações acima podem ser expressas em CTL usando as proposições associadas aos elementos de interface nas Figuras 4.10 e 4.11 por:

1. **AG**(sr **U** nak): é sempre o caso que sr (requisição de status) ocorre até que nak (não reconhecimento) ocorre.

2. **AG**($ack \rightarrow si$): é sempre o caso que se um ack (reconhecimento) ocorre um si (envia item) ocorre.

As propriedades (1) e (2) podem ser validadas para as estruturas de Kripke para o produtor e para o consumidor (Figuras 4.12 e 4.13 respectivamente). Por exemplo, a propriedade (2) garante que nenhum produtor adquirirá o direito de enviar um item e não o fará. Se a propriedade (2) não for satisfeita pela consumidor um *dead-lock* pode ocorrer.

Os algoritmos para a análise de sistemas de *G-Nets* serão introduzidos no Capítulo 5.

4.9 Aspectos de Tolerância a Falhas

Nesta seção discutimos como introduzir propriedades e esquemas de tolerância a falhas na concepção de sistemas de *G-Nets*.

4.9.1 Introdução de Esquemas de Recuperação

A introdução de mecanismos de recuperação para *G-Nets*, considera o *isp* decomposto, uma vez que ele é o elemento resposável por iniciar a executar a comunicação entre *G-Nets*. A ideia é adicionar a cada *isp* retsrições de tempo correspondendo ao tempo encapsulado relacionado com a *G-Net* invocada, neste caso o *pior caso de execução*. As restrições de tempo são computadas através de análise temporal e desempenho utilizando o modelo FTPN [82, 32, 30, 31]. Esta análise pode ser exeutada em tempo de execução ou *off-line* de modo a identificar o tempo necessário para que a invocação de um objeto termine. Portanto é necessario prover meios para medir o tempo decorrido entre o início e o fim de uma invocação. No que segue mostramos como o *isp* pode ser utilizado para esta função [21, 84, 33].

Na Figura 4.14 mostramos um *isp* tolerante a falhas. A invocação da *G-Net* G' utilizando o método m é representada pela transição rotulada com G'. Uma restrição de tempo definida pelo intervalo $[ti, ta]$ é atribuída à transição G', onde, ti e ta são respectivamente os tempos de execução mínimo e máximo associados à invocação da rede G' utilizando o método m. O lugar RI recebe uma ficha da transição de entrada t', então a transição si dispara e ocampo $tkn.seq$ é atualizado de acordo com um valor inteiro n, obtido do lugar SI, o qual é incrementado de um toda vez si dispara. A ficha de saída é então deposita no lugar RIm. Esta ficha terá uma seqüência única de identificação juntamente com um conjunto de fichas definindo a marcação inicial de G'. Além disto, uma ficha é depositada no lugar Wi. A transição G' pode então disparar, correspondendo a invocação da rede G'. Quando a rede G' alcanç seu lugar alvo as fichas ou ficha de retorno são depositadas no lugar RC e a transição ra dispara, resultando em uma ficha no lugar RR. Ainda mais, o lugar Wi pode ser utilizado para medir o tempo de uma invocação. Para tanto basta observar quanto tempo a ficha permanecerá no lugar Wi até o fim da execução. Se, por alguma razão, a transição G' não dispara, uma transição externa to que representa o time-out disparará após o tempo definido por t_a, i.e., a transição to gatilhará um bloco de recuperação o qual poderia ser a invocação de uma *G-Net* responsável pela recuperação da *G-Net* invocada G'. O processo de invocação é recomeçado via *isp* pelo depósito de uma ficha no lugar RI.

Para o caso de *recuperação regressiva* deve-se considerar quando diferentes redes são invocadas (por diferentes *isp's*), e existir um ponto de sincronização após os resultados serem retornados [101]. Neste caso, se uma falha ocorre após o ponto de sincronização, todos os processos em uma *G-Net* devem ser inibidos pela geração de fichas faltosas que são enviadas para transições com arcos inibidores. Estas fichas são depositadas em

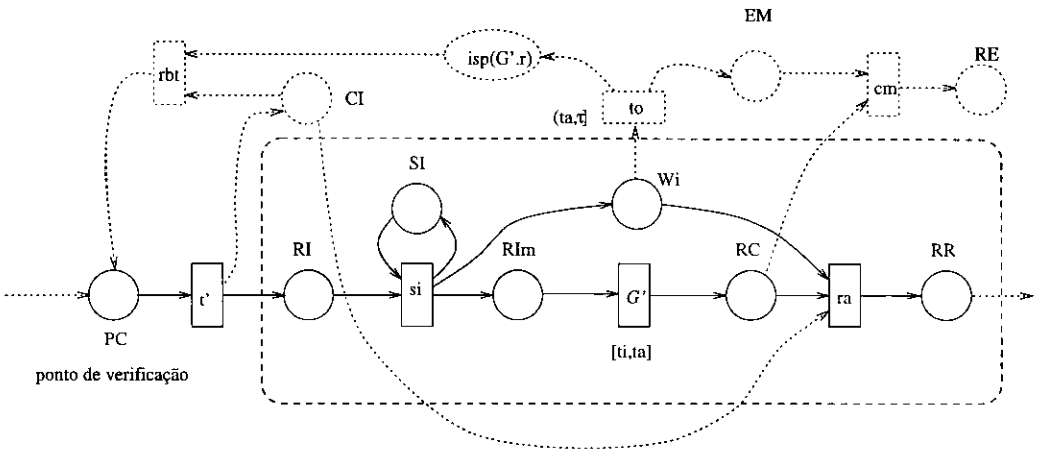

Figura 4.14: Estrutura interna de *isp* tolerante a falhas

lugares conectados com as transições através de arcos inibidores parando assim todas as execuções. Depois que os processos são parados, todos os processos são forçados a regredir, retornando para o ponto de verificação antes da invocação das redes. Neste caso todas as invocações anteriores devem também serem corrigidas e devem voltar para um ponto de verificação. Isto pode ser executado pelos *isp's*, como *isp(G'r)* na Figura 4.14.

Outra possibilidade é usar o esquema de N-versões [4], i.e., executar N versões em paralelo, obter os resultados e, baseado em um mecanismo de decisão de consenso determinar o resultado. Este esquema pode ser considerado como um esquema *recuperação para a frente*. No caso de software é necessário garantir a diversidade das versões [65].

Na Figura 4.15 apresenta-se como o esquema de N-versões pode ser implementado em uma *G-Net*. Na Figura 4.16 apresenta-se a estrutura detalhada dos *isps* envolvidos, *isp(G'1)*, $\cdots$, *isp(G'n)*. A maioria dos lugares e transições tem o mesmo significado como no caso de *recuperação regressiva temporizada*. Neste esquema, o lugar *EMi* não serve apenas de memória de falhas e para o tratamento de mensagens espúrias, como no caso anterior. Ele é também usado para prover a sincronização necessária que é modelada pela transição *rm*, realizando a invocação da rede *GM*, que implementa o algoritmo de decisão. Quando a transição *rm* dispara, o resultado de cada versão é passado para o *isp(GM)* que invocará a rede *GM*. É importante ressaltar que o arco que liga a transição *em* ao lugar *EM* é necessário para garantir que, em caso de *time-out*, uma mensagem espúria seja removida como para o caso de *recuperação regressiva temporizada*

A rede que implementa o algorítmo de decisão define a resposta correta com base

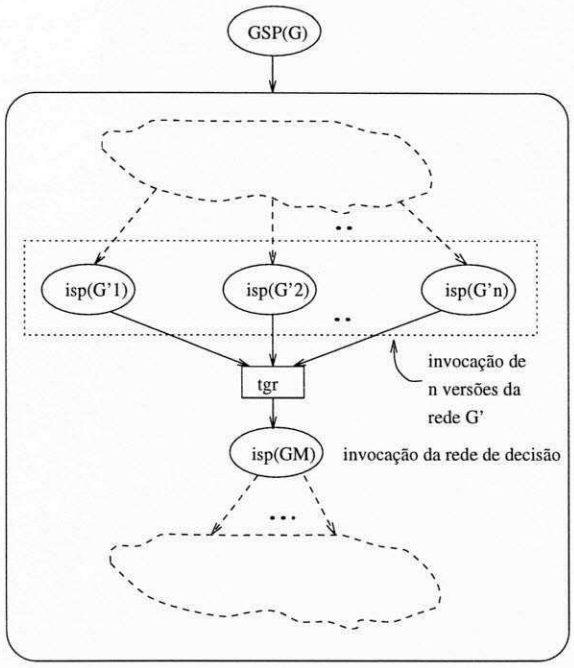

Figura 4.15: Esquema de N-versões implementado por um conjunto de *isp's*

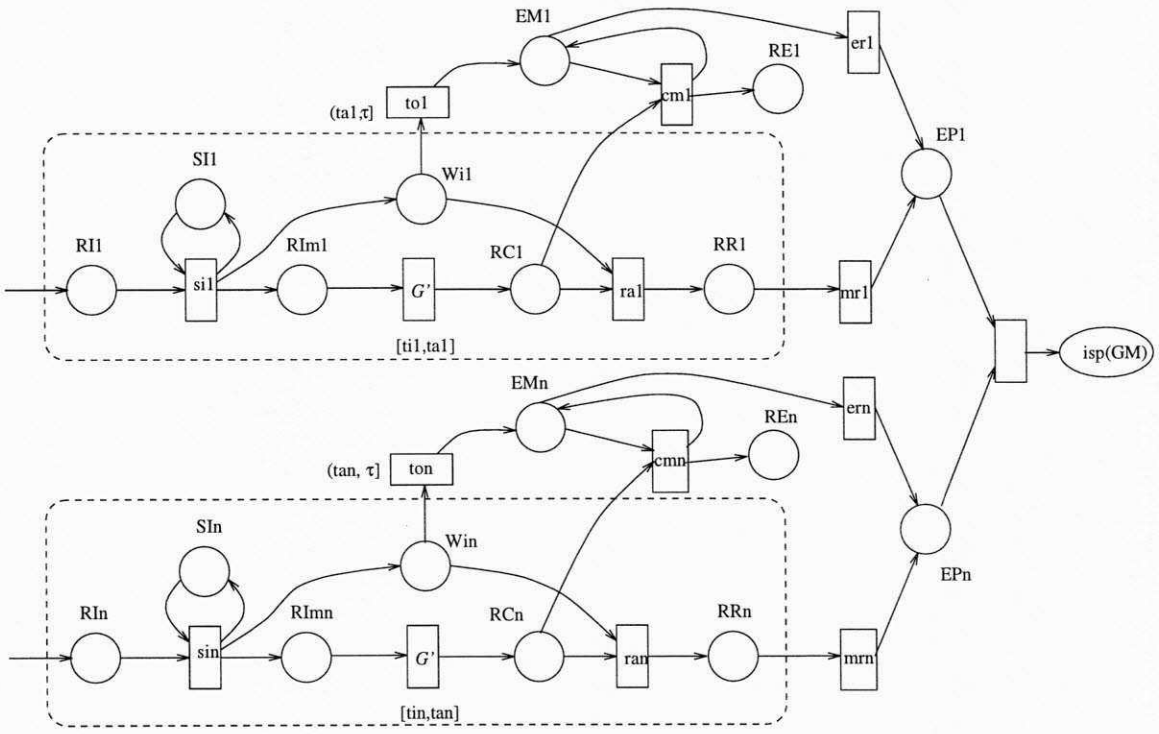

Figura 4.16: Detalhes dos *isps* para o esquema de N-versões

na informação recebida. Se necessário, a rede *GM* pode executar um procedimento de recuperação de falhas em uma versão faltosa através da atualização de seus estados baseada na informação da(s) versão(ões) correta(s).

É desejável que o desempenho do mecanismo de decisão seja otimizado. Esta otimização pode ser obtida pela inclusão de um teste de aceitação, i.e., a discrepância da mensagem retornada, lugar *RRi*, na Figura 4.16, pode ser verificada antes de invocar o mecanismo de decisão. A introdução desta característica é trivial e não será detalhada.

Deve-se enfatizar que a unicidade do campo *tkn.seq* carregado pelas fichas garante que apenas as mensagens espúrias serão removidas do lugar *RC*. Isto porque a transição *em* apenas dispara se uma ficha nos lugares *EM* e *RC* tem o mesmo valor no campo *tkn.seq*.

4.9.2 Introdução de Propriedades de Tolerância a Falhas não Dependentes do Tempo

Assumindo que erros de programadores não podem ser totalmente evitados e que as mudanças no ambiente não podem ser totalmente previsíveis, dois tipos de propriedades de tolerância a falhas não dependentes do tempo devem ser consideradas na concepção de *G-Nets* e sistemas de *G-Nets*: *auto-proteção* e *auto-verificação*.

A inclusão de auto-proteção no projeto de uma *G-Net* baseia-se no procedimento de análise de *sistemas de G-Nets* introduzido em [82]. Esta análise é baseada no paradigma assume-garante [88]. Cada *G-Net* é localmente analizada considerando suposições ambientais. Neste trabalho a introdução de auto-proteção tem como objetivo evitar que ocorram seqüências de invocação errôneas na interação entre *G-Nets*. O procedimento para sistematicamente introduzir auto-proteção é mostrado a seguir. Deve-se observar que o procedimento que será apresentado somente é aplicável para uma seqüencia de invocação de dois métodos. Isto é o método tm_i deve preceder o método tm_j. Deve-se contudo ressaltar que a generalização deste procedimento é trivial.

1. Para cada duas seqüências de invocação faça
   - 1.1 Adicione um lugar à estrutura interna da rede
   - 1.2 Conecte um arco de entrada entre o lugar e a transição no *GSP* decomposto que corresponda ao primeiro método a ser invocado para cada seqüência de invocação
   - 1.3 Conecte um arco de saída entre o lugar e a transição no *GSP* decomposto que corresponda ao método subseqüente a ser invocado
   - 1.4 Adicione uma transição para disparar um bloco de recuperação
   - 1.5 Conecte a transição introduzida através de um arco de entrada ao lugar inicial no *GSP* de modo a remover uma ficha depositada devido a uma seqüência de invocação errônea

1.6 Adicione um bloco de recuperação e um lugar alvo correspondendo ao término do procedimento de recuperação

1.7 Conecte um arco inibidor entre o lugar e a transição que inicializarão o bloco de recuperação

2. Fimfaça

Apresenta-se na Figura 4.17 a aplicação do procedimento para introduzir auto proteção a nível de *GSP* em uma *G-Net*. A suposição sendo considerada é que *o método m2 sempre deve ser invocado antes do método mn*. Observe que a invocação destes métodos corresponde aos disparos das transições *tm2* e *tmn* na Figura 4.17. As linhas pontilhadas na figura correspondem a aplicação do procedimento para introdução de um bloco de recuperação. O lugar *PR*, rotulado com 1 foi introduzido de acordo com o *passo 1.1* do procedimento. Os arcos rotulados 2, 3, 4, 5 e 7 foram introduzidos de acordo com os passos 1.2, 1.3, 1.5, e 1.7. Para introduzir o arco rotulado com 2, *passo 1.2*, considera-se que o método *m2* deve preceder o método *tmn*, e para introduzir o arco 3, *passo 1.3*, cosidera-se que o método *mn* pode ser invocado se e somente se o método *m2* foi previamente invocado. A transição *tr* é introduzida de acordo com o *passo 1.4*. E por fim, o bloco de recuperação e o lugar alvo *GPe* são introduzidos de acordo com o *passo 1.6*. O bloco de recuperação deve ser introduzido, pois uma falha pode ser introduzida se o método *mn* é invocado sem uma prévia invocação do método *m2*. Então, se esta situação ocorrer, a transição *tr* dispara e a ficha erroneamente depositada no lugar *SP* é removida. O bloco de recuperação é executado e uma mensagem indicando que o acesso foi ilegal é enviada para a *G-Net* solicitante através do lugar alvo *GPe*.

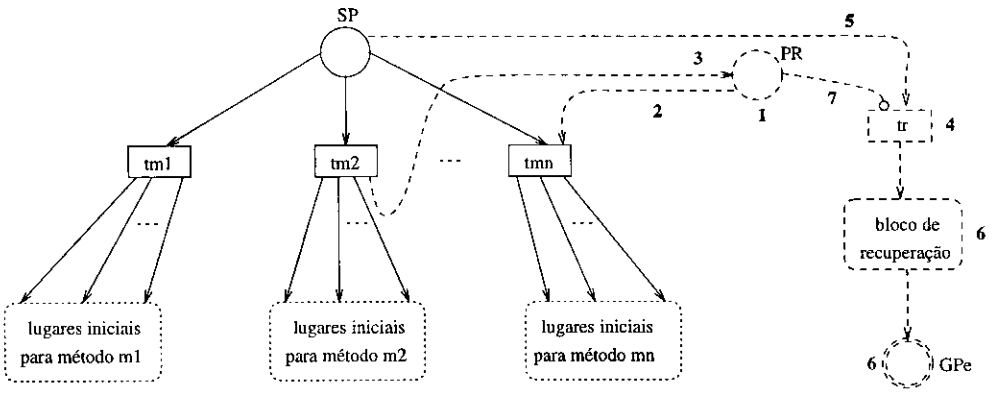

Figura 4.17: Inclusão de auto-proteção a nível de *GSP*

4.9.3 Exemplificação da Aplicação do Algoritmo

Produtor/Consumidor

Na Figura 4.18 apresentamos a aplicação do procedimento para introduzir auto-proteção ao consumidor. A propriedade exportada sendo considerada é: $\mathbf{AG}(ack \rightarrow si)$. De acordo com o procedimento, as linhas pontilhadas representam a introdução de um bloco de recuparação ao consumidor. O lugar PR, rotulado com 1 na figura, é introduzido de acordo com o passo 1.1. Os arcos com rótulos 2, 3, 5 e 7 são introduzidos de acordo com os passos 1.2, 1.3, 1.5, e 1.7. Observe que para o arco com o rótulo 2, passo 1.2, consideramos o lado direito da fórmula, e para o arco 3, passo 1.3, estamos considerando o lado esquerdo da fórmula expressando a consideração sobre o ambiente. A transição tr é introduzida de acordo com o passo 1.4. E o bloco de recuperação e o lugar alvo GPe são introduzidos de acordo com o passo 1.6. Este bloco de recuperação deve ser introduzido pois uma falha pode ser introduzida se o método mc é invocado sem uma prévia invocação do método ms. Portanto, se esta situação ocorrer, a transição tr dispara e a ficha erradamente depositada no lugar SC é removida, um bloco de recuperação é executado e uma menssagem indecando que o acesso foi ilegal e enciada para a rede invocadora através do lugar alvo GPe.

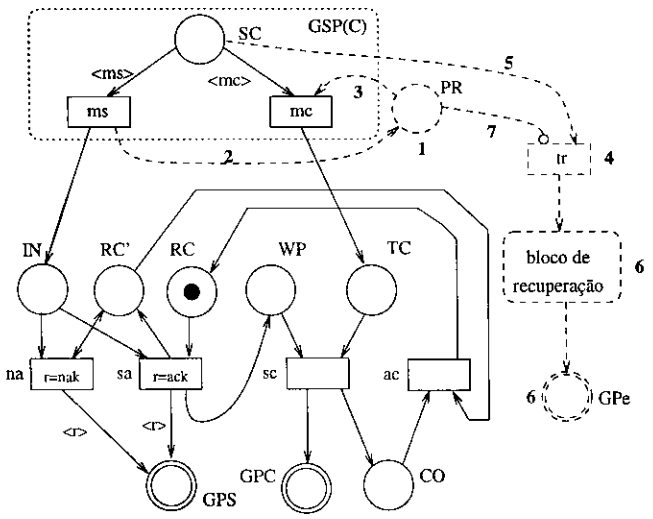

Figura 4.18: Incluindo auto-proteção na concepção do módulo para o consumidor

Capítulo 5

EXPERIMENTAÇÃO

5.1 Introdução

Neste capítulo discutimos aspectos de implementação e resultados experimentais desta tese. Apresentamos a estrutura de um sistema de verificação para *G-Nets*, mostramos ainda nossa contribuição para a implementação deste sistema. O sistema foi implementado em linguagem C, e é portável para diferentes máquinas. A implementação corrente pode ser executada em DECStations e SPARCstations sob UNIX. A interface do sistema foi implementada com base no sistema XWindows XWindows [3, 104, 110].

Detalhamos ainda as funcionalidades dos módulos do sistema de verificação, assim como os procedimentos definidos para verificação. Ilustramos nossa apresentação através da aplicação do sistema desenvolvido para a análise do problema do produtor e consumidor.

5.2 Sistema de Verificação

Antes de detalharmos a implementação propriamente dita, mostramos na Figura 5.1 o diagrama de blocos para o sistema de verificação. A partir desta figura podemos identificar dois diferentes tipos de análise: análise de desempenho e análise comportamental. A parte responsável pela análise de desempenho não é discutida nesta tese, para detalhes refira-se a [30, 42, 43, 82]. A análise comportamental, incluindo a geração da árvore e do grafo de alcaçabilidade serão detalhados a seguir.

Como mostramos na Figura 5.1, o sistema de verificação tem como entrada a especificação de uma *G-Net*, no momento esta especificação é uma descrição em forma de texto como introduzida por Chen em [23, 24]. Esta descrição em forma de texto é carregada pelo gerador de árvore de alcaçabilidade, e é usada para gerar a árvore de alcançabilidade, este módulo é apresentado na Seção 5.2.1. Baseado-se nesta árvore de alcançabilidade e alguma informação adicional, geramos o grafo de alcançabilidade para a rede especificada, veja Seção 5.2.3 para detalhes. O grafo de alcançabilidade é então utilizado para executar verificação de reentrância, verificação da estrutura e comporta-

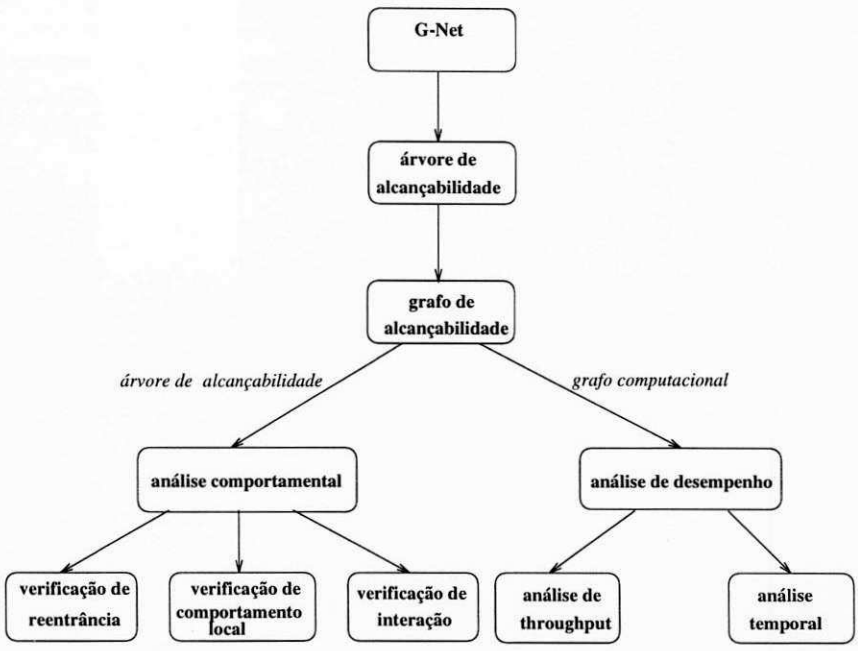

Figura 5.1: Diagrama de blocos do sistema de verificação

mentos locais, e análise de interação. Este módulos são discutidos na Seção 5.2.4. Deve ser enfatizado que nem todos os módulos estão completamente implementados ou tem suas funcionalidades limitadas, portanto a apresentação enfatizará as funcionalidades do módulo ao invés da implementação. Sempre que necessário procedimentos e exemplos serão utilizados para ajudar o leitor a melhor entender o sistema.

5.2.1 Gerador de Árvore de Alcançabilidade

O *Gerador de Árvore de alcançabilidade* (GAA), na verdade uma árvore de alcançabilidade *parcial*, como discutido no Capítulo 4, é o módulo que carrega a descrição em forma de texto para a *G-Net* a ser analisada e gera a árvore de alcançabilidade. Mostramos o diagrama de blocos do módulo para gerar árvore de alcançabilidade na Figura 5.2.

De modo a gerar a árvore de alcançabilidade o GAA necessita das seguintes informações como entrada:

1. A especificação em forma de texto para a *G-Net.*

2. Funções primitivas.

3. Considerações ambientais.

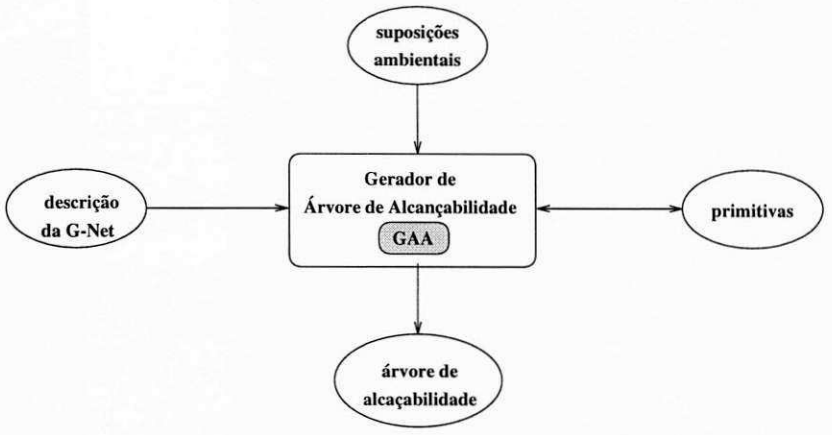

Figura 5.2: Diagrama de blocos do gerador da árvore de alcaçabilidade

Para iniciar a execução do módulo RTG, o usuário pode definir o seguinte comando:

```
Nome_da_Rede,[método],[argumentos],[tempo]
```

onde `Nome_da_Rede` é o nome do arquivo contendo o formato texto, `método` é um nome opcional para um método válido, `argumentos` são argumentos opcionais para o método, e `tempo` é um campo opcional de temporização utilizado somente pelo simulador, mas sem nenhum significado para a geração da árvore de alcançabilidade.

Antes de detalharmos o procedimento para geração da árvore de alcaçabilidade, introduziremos o significado e como cada uma das informações é representada. Com o objetivo de ajudar o leitor nossa discussão será exemplificada utilizando o exemplo do produtor/consumidor.

Descrição de uma G-Net

Assumimos como entrada para o GAA a especificação de uma *G-Net* na forma de um arquivo texto. Este arquivo texto permite que o usuário represente a rede utilizando símbolos ASCII. Além disto, um editor para criar este arquivo é disponível e detalhado em [23]. O formato do arquivo de texto é o seguinte[1]:

[1] A descrição do arquivo texto, assim como outras descrições seguem a implementação original em inglês desta tese.

```
GSP <GSP definition>
&  <symbol & is used as a separator>
Nid <name of the net>
&
m_name1 <name of the method>
mstrategy1 <always goal_state>
minitiator1 <method_id:<parameter_list>,<time>>
&& <end GSP definition>
NP <normal places>
&
Pid1 <identifier of the place>
capacity <capacity of the place>
action_name1 <action associated>
ifgoal<1 if goal place 0 otherwise>
&& <end of normal places>
ISP <instantiated switching places>
&
Isp_id1 <identifier>
Nid1 <name of isp>
m_name <method to invoke>
m_action.before <action before send token>
m_action.after <action after token is received>
ifgoal <1 if goal place 0 otherwise>
&& <end isp list>
TRANSITION <begin transitions>
&
tname1 <name>
&
INPUT <input places list>
&
Pid <id place of input place>
num <number of tokens taken>
ccolor <always white>
bcolor <activate token for arc bcolor>
&
OUTPUT <output places list>
&
Pid <id of output place>
num <number of tokens deposited>
ccolor <always white>
Pid,(field)
Pid,(field)
&& <separates next transition record>
&&& <end of file>
```

Os comentários em cada linha especificam o tipo de linha e o significado dos separadores, p.e. o símbolo &. De modo a a ajudar o leitor a entender este formato mostramos nas Figuras 5.4 e 5.3, respectivamente, a *G-Net* modelando o consumidor e o produtor com os identificadores utilizados no formato texto para a a rede.

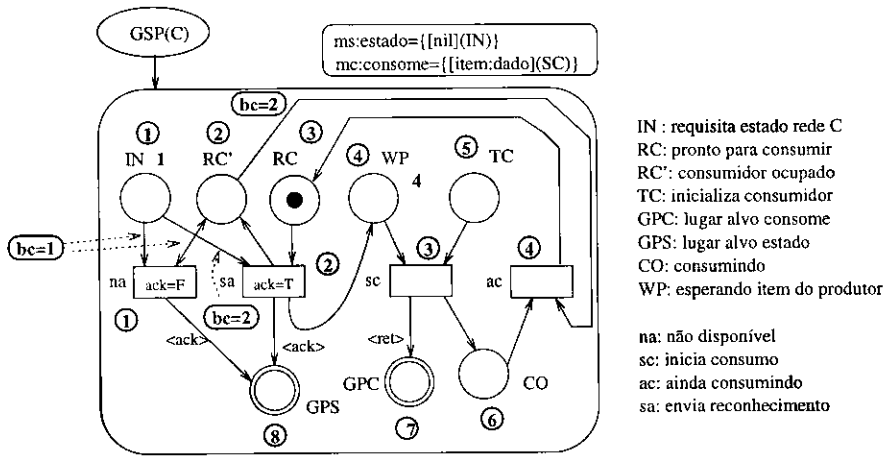

Figura 5.3: *G-Net* modelando o consumidor, com os identificadores utilizados para o formato texto

Para a *G-Net* modelando o consumidor, mostrada na Figura 5.3, o formato texto é o seguinte:

```
GSP             3           OUTPUT      1           &&          &
&               5           &           white       &           INPUT
PROD_Net        p3_prod     2           1           4           &
&               0           1           &&          &           3
m1              &           white       &           INPUT       1
goal_state      4           1           3           &           white
1:2:n:T         5           &&          &           2           1
&&              p4_prod     &           INPUT       1           &&
NP              1           2           &           white       OUTPUT
&               &&          &           2           1           &
1               &           INPUT       1           &&          1
5               1           &           white       OUTPUT      1
p1_prod         &           1           2           &           white
0               INPUT       1           &&          3           3
&               &           white       OUTPUT      1           &&
2               1           2           &           white         &&&
5               1           &&          2           2
p2_prod         white       OUTPUT      1           &&
0               1           &           white       &
&               &&          4           2           5
```

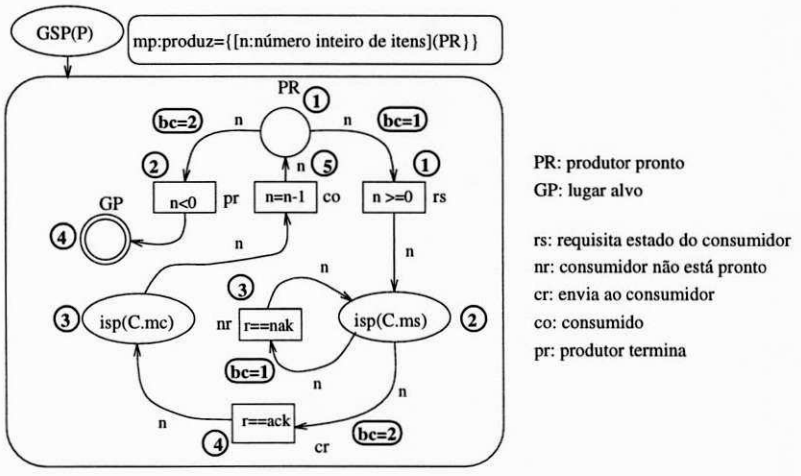

Figura 5.4: *G-Net* modelando o produtor com os identificadores utilizados para o formato texto

Para maiores detalhes a respeito do arquivo em formato texto para *G-Nets* refira-se a [23, 24]. Apesar do fato de que em alguns casos esta descrição em formato texto possui informações que não são relevantes para este trabalho, em outros casos esta descrição sofre com a ausência de informações necessárias, p.e., *nomes* de lugares e transições, entretanto utilizamos o mesmo formato de arquivo para manter compatibilidade com a implementação corrente do simulador para *G-Nets*.

```
/* this function is associated with place PR(1) of G(P)(PROD_Net) */
#include "../G_gen.h" /* header file for primitives */

main(argc, argv)
int argc;
char *argv[];
{
   char token_message[NIDLEN]; /* token message buffer */
   int bb;                     /* branch color */
   char cc[12];                /* temporary buffer */
   char str;                   /* temporary buffer */
   char buf[MAXLINE];          /* temporary buffer */
   int pipefd;                 /* pipe identifier */
   int t = 1;                  /* predefined consumed time */
   int tstr[5];                /* value of field n (number of items) */

/* get the pipe identifier for the returning token */
   pipefd = atoi (argv[2]);
/* get the token message */
   printf (token_message, "%s", argv[1]);
/* this is only for timing purpose */
   get_fieldv(buf, "T", tstr);
/* get the number of items (n) and convert to integer */
   get_fieldv(argv[1], "n", ff1);
   t += atoi (tstr);
   sprintf (tstr, "%d", t);
   if ( strncmp(ff1, "0", 1) == 0)
      bb = 2; /* no more items, set branch color to transition pr(2) */
   else bb = 1; /* still items to produce, branch color is transitions rs(1) */
   strncpy(cc, "white", 12);
   change_field_value (argv[1], "T", tstr);
   sprintf(buf, "%s:%d\#%s", "white", bb, argv[1]);
/* send the resulting token back */
   write (pipefd, buf, sizeof(buf));
}
```

Figura 5.5: Código C para a função associada ao lugar PR (**1**) de $G(P)$

Funções Primitivas

Uma vez que a nossa implementação do módulo para gerar a árvore de alcaçabilidade mantém compatibilidade com o simulador de *G-Net*, devemos prover, escrever e compilar, as funções associadas às primitivas. As primitivas executam a mesma função que as expressões dos arcos e inscrições das transições. Uma função primitiva simples é por exemplo aquela associada ao lugar PR (identificador **1**) para o produtor. Mostramos na Figura 5.5 o código em C para esta função.

A função associada ao lugar é executada quando uma ficha é depositada em PR. Para executar a função o gerador da árvore de alcançabilidade simplesmente executa o *fork* de um processo, e executa a função associada ao lugar. Além disto, um *pipe* é aberto para permitir a comunicação entre o gerador da árvore de alcançabilidade e função associada ao lugar. Quando a função é executa ela recebe a ficha e o identificador do *pipe* pelo entrada padrão. A ficha possui um mensagem associada, a qual entre outras informações possui o valor de n, i.e., o número de itens a produzir. Então a função, verifica o campo n da mensagem. Quando n é maior ou igual a zero a cor de desvio da ficha torna-se

1. Contrariamente, a cor de desvio torna-se 2. Antes de terminar a execução a função reconstrói a ficha com a nova informação resultante da execução da primitiva. Entre outras coisas a ficha carrega a cor de desvio, o valor dos novos parâmetros e a seqüência de propagação. Esta ficha é então retornada ao gerador de árvore de alcaçabilidade através do *pipe* previamente aberto. Então, a função implementa as funcionalidades tanto das expressões associadas aos arcos como das inscrições das transições.

Considerações Ambientais

As considerações ambientais são necessárias para gerar a árvore de alcaçabilidade, pois não podemos decidir localmente quando uma ficha será depositada em um lugar inicial. Portanto devemos prover meios pelos quais seja possível decidir quando uma ficha é depositada em um lugar inicial, isto é, definir a marcação inicial da rede a ser analizada. Além disto, necessitamos de algum meio para definir o comportamento de uma outra rede invocada por um *isp*. Logo, as considerações ambientais são relacionadas ao comportamento dos *isp's* e *GSP's*.

Como dito anteriormente, necessitamos definir o comportamento de um *isp* de modo a permitir a *simulação* do comportamento da rede sendo invocada. Portanto, necessitamos especificar, ou assumir, o comportamento da rede invocada. Na verdade, como discutido no Capítulo 3, este comportamento corresponde ao comportamento externo ou abstraído da rede invocada, o qual deve ser verificado quando analisando-a. Do ponto de vista de implementação definimos os possíveis comportamentos para a rede invocada por meio de uma cadeia de caracteres armazenada em um arquivo. Por exemplo, quando o produtor invoca o consumidor com método estado, o possível comportamento para o consumidor pode ser caracterizado por uma cadeia de caracteres da forma[2]:*nak* ack*. Portanto, podemos definir a árvore de alcançabilidade assumindo que o comportamento do consumidor é *nak ack*. Mesmo considerando que esta cadeia não é única, ela é suficiente para localmente gerar a árvore de alcaçabilidade da *G-Net* para o produtor.

O comportamento do *GSP* da rede deve também ser assumido. Isto pois temos que definir como a rede invocando comportar-se-á. No caso do produtor esta definição é direta, é suficiente especificar que a rede invocando depositará, por exemplo, uma ficha com o campo $n = 1$. No caso do consumidor temos dois problemas para resolver. O primeiro é como definir a marcação inicial, uma vez que ela é não vazia. Segundo, para gerar a árvore de alcançabilidade que representa o comportamento do consumidor,

[2] A * corresponde ao operador de Kleene.

necessitamos de um meio para definir como o ambiente irá comportar-se, e qual será a marcação inicial para a primeira invocação. Observe que a marcação inicial que estamos discutindo não é aquela relacionada aos lugares iniciais, mas sim ao estado inicial da rede. Por exemplo, para o consumidor, o lugar RC possui uma ficha representando que o buffer de requisição possui uma posição disponível. A parte difícil é definir a seqüência de invocação. A seqüência de invocação é codificada em um arquivo de definição que possui o seguinte formato:

Nid, method_1, arguments, time &
Pid tokens @
Nid, method_2, arguments, time @
...
Nid, method_n, arguments, time .

No formato acima Nid é o nome da rede, por exemplo PROD_Net ou CONS_Net, $\mathsf{method}_1 \cdots \mathsf{method}_n$ é o nome dos métodos, por exemplo *mp* para o produtor, e *ms* e *mc* para o consumidor, arguments são os argumentos para o método, por exemplo para o consumidor o valor de n (número de itens a produzir). Por exemplo, no caso do consumidor assumimos o seguinte comportamento para o ambiente.

```
CONS_Net,ms,T:0 &
3 1 @
CONS_Net,mc,T:0 @
CONS_Net,ms,T:0 @
CONS_Net,ms,T:0 .
```

A seguir explicamos o significado de cada uma das linhas das considerações ambientais descritas acima:

- A primeira linha indica que a rede `CONS_Net`, *G-Net* $G(C)$, é invocada com método `ms` sem nenhum parâmetro associado. O campo `T:0` não possui significado para esta implementação, ele foi definido para manter compatibilidade com o simulador implementado, e está relacionado à restrições temporais. O símbolo `&`, informa que segue informação relacionada com a marcação inicial da rede `CONS_Net`, neste caso temos uma ficha no lugar **3** (o lugar RC na rede $G(C)$ mostrada na Figura 5.3). O símbolo `@` informa que a árvore de alcaçabilidade pode então ser gerada para a invocação especificada.

- A terceira linha informa que quando a primeira invocação termina, a rede $G(C)$ será invocada com método consome, `mc`, e o símbolo `@` informa que a definição para esta invocação está completa, e a geração da árvore de alcaçabilidade pode então proceder para esta invocação.

- A quinta linha indica a última invocação, isto é especificado pelo ponto ao fim da linha.

No caso do produtor definimos o seguinte arquivo de definição, o qual informa que a árvore de alcançabilidade é para ser gerada para a rede especificada no arquivo `PROD_Net`, o método invocado é `ms`, com um parâmetro definido, i.e., `n=1`, e esta é a última linha de comando a ser lida (o . ao fim da linha indica isto):

```
PROD_Net,mp,n:1,T:0 .
```

Uma vez que explicamos o significado e o formato das informações necessárias para gerar a árvore de alcançabilidade, introduziremos o procedimento para gera-la.

5.2.2 Procedimento para Gerar a Árvore de Alcançabilidade

O procedimento Gere_Árvore_de_Alcaçabilidade, mostrado na Figura 5.6, lê como entrada as informações anteriormente descritas e gera a árvore de alcançabilidade como saída. O gerador de árvore de alcaçabilidade executa a mesma função que um *jogador de redes de Petri* (*token game player*). Na verdade a implementação atual do procedimento baseia-se no *motor de inferência* do simulador de *G-Nets* [23]. O procedimento lê o arquivo texto para a rede definida, e também lê os nomes dos lugares e transições, os quais são utilizados para apresentar ao usuário a marcação dos lugares e o disparo das transições, como mostraremos a seguir. O procedimento então lê a primeira e segunda linhas do arquivo de definição, estabelece a marcação inicial, inicia o disparo de transições e começa a gerar a árvore de alcaçabilidade. Quando o fim do arquivo de definição é alcançado o procedimento termina sua execução. Devemos enfatizar que a marcação inicial, como por exemplo uma ficha no lugar RC para a rede do consumidor, é definida somente uma vez, mais especificamente, a primeira vez que o procedimento executa o laço entre as linhas 5.1. e 5.4. Após isto, quando a geração da árvore de alcaçabilidade para, a invocação termina, a marcação final é armazenada na memória, uma nova linha de comando é lida do arquivo de definição, se existente, e o procedimento prossegue com a geração da árvore de alcançabilidade. A complexidade de tempo para

este procedimento é exponential ($O(2^n)$) com o número de estados. De qualquer forma, pode-se manter a *G-Net* a ser analisada razoavelmente pequena de modo que aplicação do procedimento, mesmo considerando esta complexidade, é ainda aplicável do ponto de vista prático.

Procedimento Gera_Árvore_de_Alcaçabilidade(Nome_da_Rede)

```
1.      Rede ← Lê_Arquivo_Texto(Nome_da_Rede)
2.      Inicializa_Espaço_de_Trabalho()
3.      Transições ← Lê_Nomes_de_Transições(Nome_da_Rede)
4.      Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
5.      Enquanto not_eof(Arquivo_de_Definição) Faça
5.1.      Enquanto marcação inicial não definida Faça
5.1.1.      marcação ← Lê_Definição(Nome_da_Rede)
5.1.2.      Define_Marcação(marcação)
5.2.      FimEnquanto
5.3.      Enquanto transição a disparar é verdadeiro Faça
5.3.1.      Dispara_Transição()
5.3.2.      Armazena_Marcação()
5.4       FimEnquanto
6.      FimEnquanto
```

Figura 5.6: Procedimento para gerar a árvore de alcançabilidade

A interface para o sistema de análise para *G-Nets* foi implementado utilizando o ambiente XWindows, existe também uma versão que pode ser executada sob o Unix sem o ambiente XWindows. O ambiente é inicializado pelo console do usuário, mostrado no canto superior esquerdo da cópia da tela na Figura 5.7 (a janela `console`). O usuário define o nome da rede, e outras informações, tais como o método e os argumentos são lidos do arquivo de definição, o qual foi introduzido acima. Após o usuário definir o nome da rede, neste caso `CONS_Net`, o sistema inicializa outras janelas, denominadas: `Show Net ASCII Description`, `Reachability Tree Generation`, `Reachability Graph Generation`, e `Properties Analysis`, como mostrado no lado direito da Figura 5.7. Nesta Figura mostramos a execução do módulo que apresenta o arquivo formato texto para a rede. Devemos dizer que o gráfico mostrado na janela `CONS_Net` é criado a priori. Ele não é gerado com base na descrição em formato texto para a rede.

Na Figura 5.8 mostramos a cópia da tela após o usuário ter selecionado a opção `Create` na janela `Reachability Tree Generation` (no canto inferior esquerdo da janela). Mostramos o resultado do procedimento para gerar a árvore de alcançabilidade para o consumidor (*G-Net* $G(C)$). Nesta janela mostramos os passos executados durante a geração da árvore de alcançabilidade, incluindo a seqüência de disparo das transições.

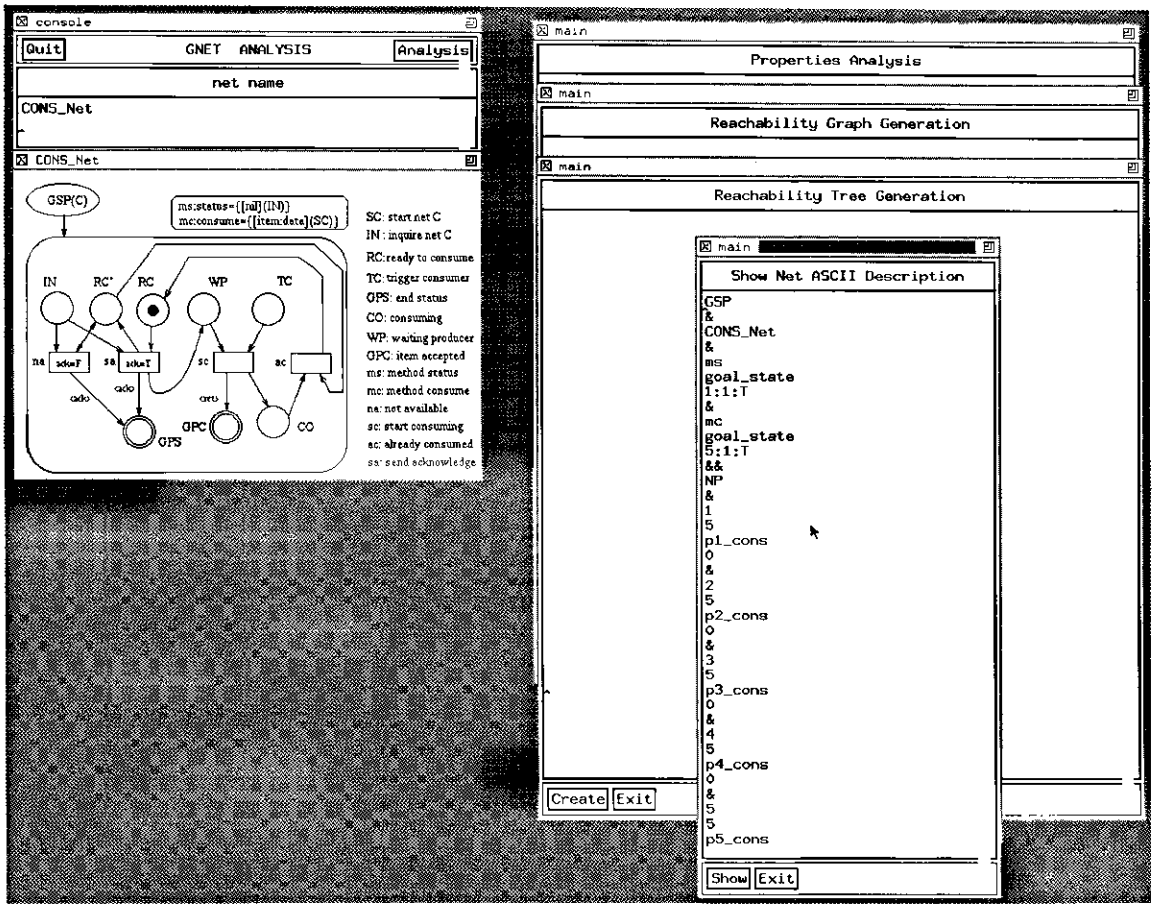

Figura 5.7: Cópia da tela de execução mostrando a descrição ASCII para o consumidor

A seqüência de disparo das transições é armazenada também, pois ela é necessária para identificar as transições pertencentes a cada método, as quais são necessárias para verificação de ausência de conflito. Na parte inferior da janela mostramos o formato texto para a árvore de alcançabilidade. Esta árvore de alcaçabilidade é armazenada em um arquivo para ser utilizada como entrada para o gerador grafo de alcançabilidade. Na verdade o formato do arquivo não é como o mostrado na figura, para armazenar a árvore de alcançabilidade utilizamos uma versão codificada, como a usada para codificar a especificação da rede. Os nomes dos elementos notacionais são obtidos a partir de dois arquivos separados que serão detalhados a partir de dois arquivos separados, os quais serão itroduzidos quando discutirmos o gerador grafo de alcançabilidade.

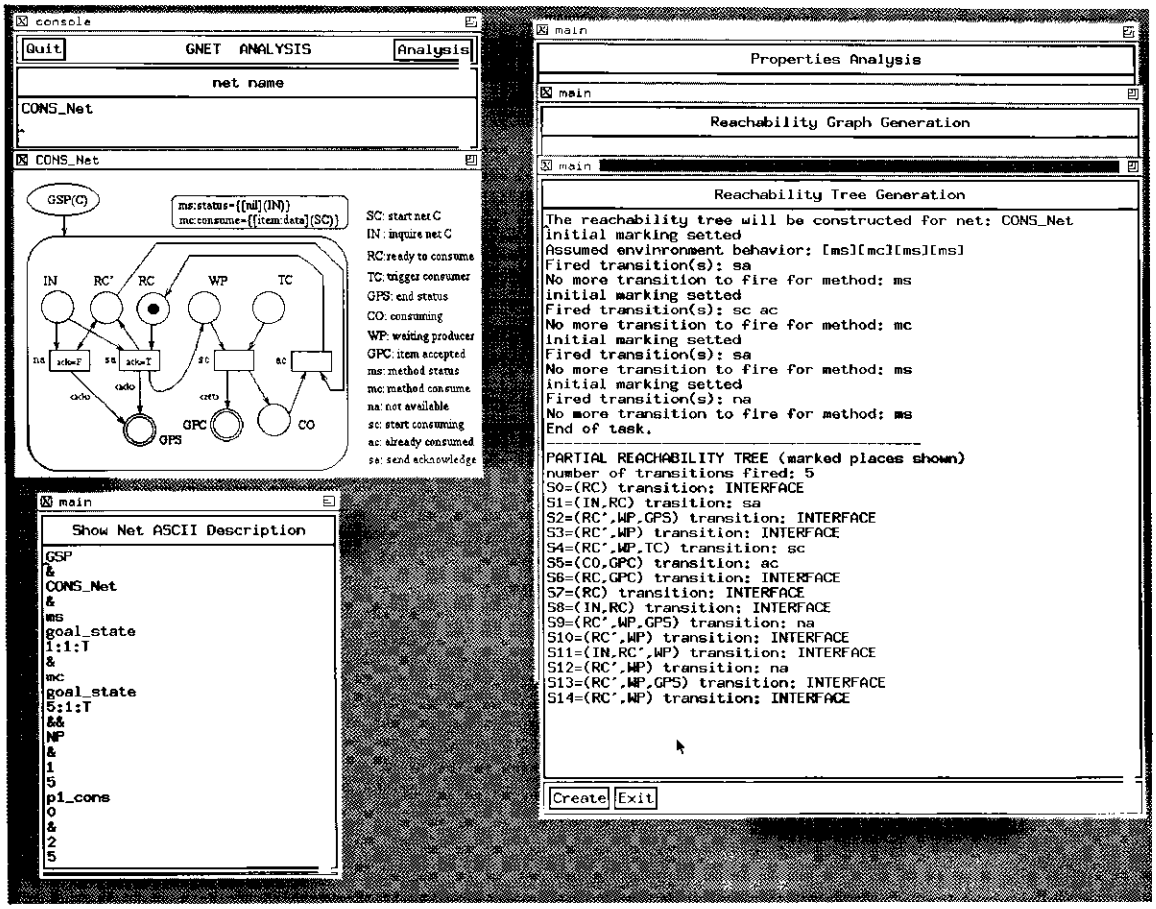

Figura 5.8: Cópia da tela mostrando a construção da árvore de alcançabilidade para o consumidor

5.2.3 Gerador do Grafo de Alcançabilidade

Para gerar o grafo de alcançabilidade, na verdade um parcial, necessitamos da árvore de alcançabilidade como entrada. Além disto necessitamos de informação sobre os lugares e transições, como será discutido a seguir. Na Figura 5.9 mostramos o diagrama de blocos para o módulo que gera o grafo de alcançabilidade.

Mostramos na Figura 5.2.3 o procedimento para gerar o grafo de alcançabilidade. Este procedimento gera também informação sobre os lugares e transições relacionados à àrvore de alcaçabilidade. Definimos então uma lista vazia para armazenar os nós do grafo de alcançabilidade. Cada $\text{Nó}_i \in$ Nós corresponde a uma única marcação na árvore de alcançabilidade. Entre os passos 7. e 8. geramos a lista de Nós, e entre os passos 8. e 9. geramos a lista de Arcos para cada Nó. Ao fim do procedimento, o grafo de

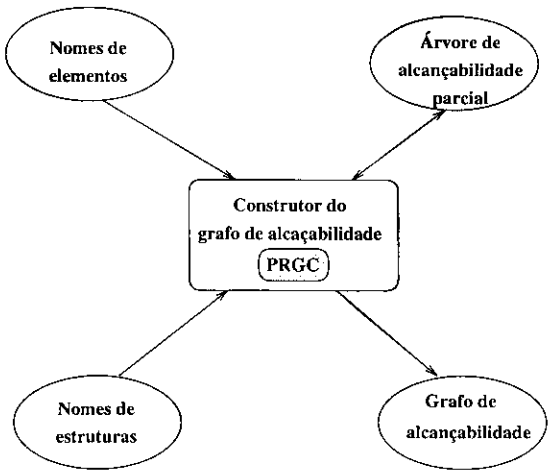

Figura 5.9: Diagrama de blocos para o gerador do grafo de alcançabilidade

alcançabilidade *RG(Nós,Arcos)* gerado é armazenado em um arquivo para ser utilizado para análise.

Na Figura 5.11 mostramos a cópia de tela para o procedimento de geração do grafo de alcançabilidade após o usuário ter selecionado a opção `Create` (canto inferior esquerdo da janela `Reachability Graph Generation`). Primeiramente apresentamos a lista de transições com seus nomes e os intervalos de *habilitação* e *atrazo* – estes dois intervalos são utilizados somente para análise de desempenho, portanto não discutiremos seus significados. Após a lista de transições apresentamos a lista de lugares, cada um dos quais com suas informações associadas. Com o propósito de análise definimos quatro diferentes tipos de lugares.

Lugares iniciais: as informações associadas aos lugares inicias de uma *G-Net* são: seu nome, por exemplo na Figura 5.11, o lugar *IN* possui nome *IN*, seu tipo é *INITIAL*, e o método associado é *ms*, e a transição externa associada é *ms*. Na Figura 5.11 a informação associada ao lugar *IN* apresenta o seguinte formato:
`IN [Name:IN][Type:INITIAL][Method:ms][E_Trasition:ms]`

Lugares normais: para os lugares normais definimos seus nomes, por exemplo para o lugar *RC* na Figura 5.11 temos o seguinte formato:
`RC [Name:RC][Type:NORMAL]`

Lugares alvo: as informações associadas aos lugares alvo são: seu nome, por exemplo na Figura 5.11, o lugar alvo *GPS* possui nome *GPS*, seu tipo é *GOAL*, *proposição de terminação* (`C_Proposition`) é *mc* – isto é, é o lugar alvo para o método *mc*, e

Procedimento Gere_Grafo_de_Alcaçabilidade(Nome_da_Rede)

```
1.      Árvore ← Lê_Árvore_de_Alcaçabilidade(Nome_da_Rede)
2.      Transições ← Lê_Nomes_de_Transições(Nome_da_Rede)
3.      Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
4.      Nós ← ∅;
5.      Nós_0 ← Marcação(0,Árvore)
        /* gera lista de vértices (diferentes marcações ∈ Árvore) */
6.      ParaTodo Marcação ∈ Árvore Faça
6.1.       Se (Macação(i,Árvore) ∉ Nós)
6.1.1.        Nós_i ← Marcação(i,Árvore)
6.2.       FimSe
7.      FimParaTodo
8.      /* conecta marcações alcaçáveis através de arcos */
8.1.    ParaTodo Nós Faça
           Arcos_i ← ∅
8.2.       /* encontre todos os sucessores do nó inicial Nós_i */
8.2.1.     ParaTodo Nós_i Faça
              If Nós_j é o sucessor de Nós_i
8.2.1.1.      /* concatene o sucessor do nó corrente á lista de arcos */
8.3.             Arcos_i ← Nós_j
8.4.          FimSe
           FimParaTodo
8.5.       /* ligue uma linha do grafo de alcançabilidade à lista de arcos */
9.         Liga Arcos _i à Nós_i
        FimParaTodo
```

Figura 5.10: Procedimento para Gerar o Grafo de Alcançabilidade

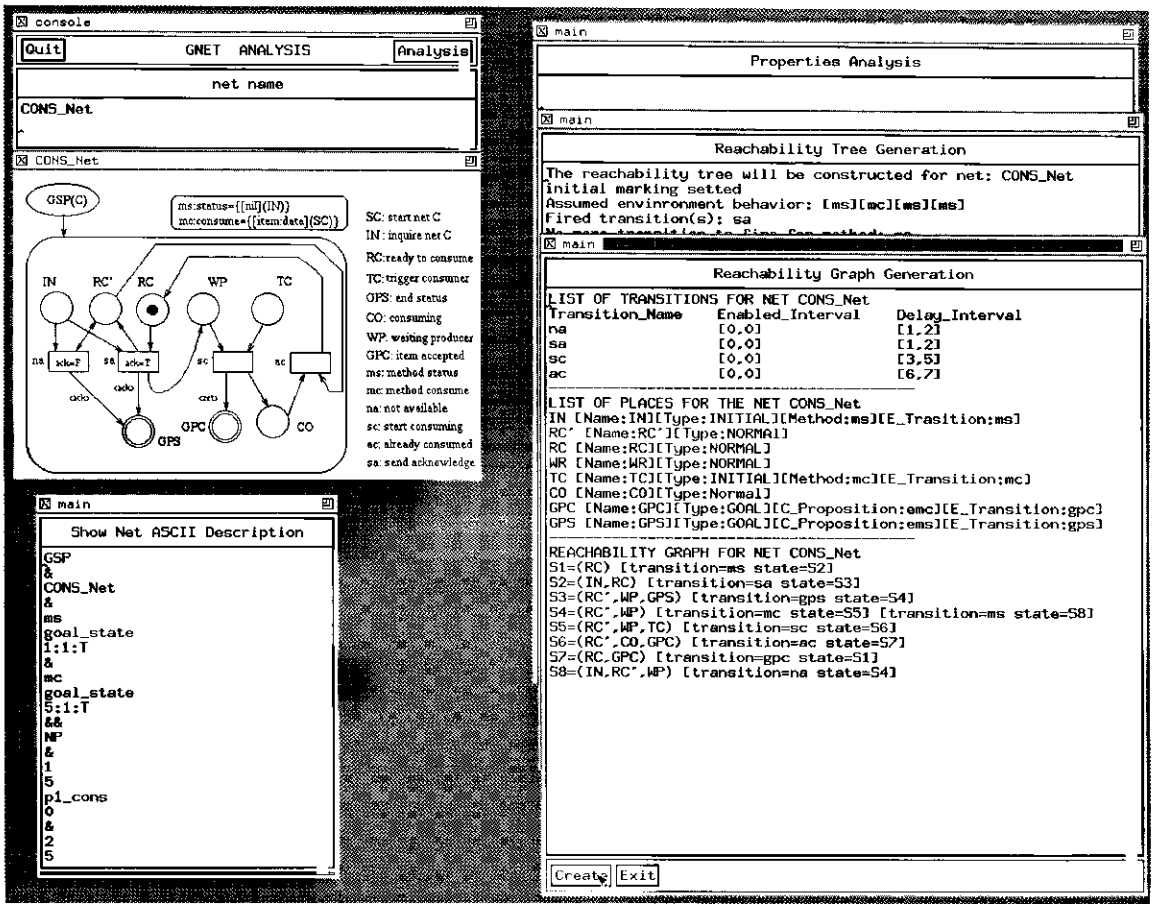

Figura 5.11: Cópia da tela mostrando a geração do grafo de alcaçabilidade para a rede do consumidor

a transição externa (E_Transition) é *gpc*. O formato apresentado na Figura 5.11 é:

`GPC [Name:GPC][Type:GOAL][C_Proposition:emc][E_Transition:gpc]`

Lugares de chaveamento para instânciação: a informação associada a estes tipos de lugares é: nome, tipo é *ISP*, a *proposição de invocação* (I_P), a *proposição de terminação* (C_P), e o *intervalo de atrazo* (D_I) – D_I somente é utilizada para análise de desempenho e não será discutido. O formato mostrado a informação associada ao *isp* é (este exemplo é para o produtor *G-Net*):

`isp(C.mc) [Name:isp(Cmc)][Type:ISP][I_P:si][C_P:ra][D_I:1,4]`

5.2.4 Análise de Estrutura e Comportamento

A análise de estrutura tem por objetivo verificar as propriedades de reentrância e ausência de conflito entre métodos. A análise de comportamento tem por objetivo verificar se as propriedades locais são ou não satisfeitas. Para ambos os casos a verificação tem por base os conceitos introduzidos no Capítulo 4. O grafo de alcançabilidade e as suposições ambientais são utilizadas para executar a análise estrutural e comportamental.

Na Figura 5.12 mostramos o diagrama de blocos do analizador de estrutura e comportamento (AEC). Com base no grafo de alcançabilidade, as suposições ambientais e as informações associadas aos lugares e transições, o AEC executa a verificação de acordo com os seguintes passos:

1. *Verificação de reentrância:* as propriedades de reentância como alcaçabilidade de lugares alvo e reinicialização são verificadas.

2. *Verificação de ausência de conflito:* o objetivo é verificar se o conjunto de transições pertencendo a um método definido para uma rede é livre de conflito em relação aos outros métodos ou não.

3. *Verificação do comportamento local:* verificamos as propriedades locais da rede.

4. *Verificação de Interação:* o objetivo é verificar se a abstração para a rede sendo analizada verifica ou não as suposições ambientais impostas.

De fato no passo 4. devemos verificar se a rede garante o comportamento assumido para a rede sendo analizada. Este comportamento faz parte do ambiente de outras redes. A seguir discutimos cada um dos passos de verificação acima introduzidos.

Verificação de Reentrância

O procedimento que realiza a verificação de reentrância é apresentado na Figura 5.13. Após ler o grafo de alcançabilidade, nomes de lugares e transições, o procedimento computa o fechamento transitivo (*transitive closure*) RG* do grafo de alcançabilidade RG. Para obter RG* aplicamos a técnica de quadrado interativo (*interactive squaring*). Utilizando RG* podemos então verificar se um lugar alvo é sempre alcançável e se a rede é reinicializável, por exemplo. Verificamos ainda se a marcação para os lugares iniciais é vazia antes do correspondente método ser invocado. Podemos então verificar se as propriedades de reentrância estão ou não sendo violadas.

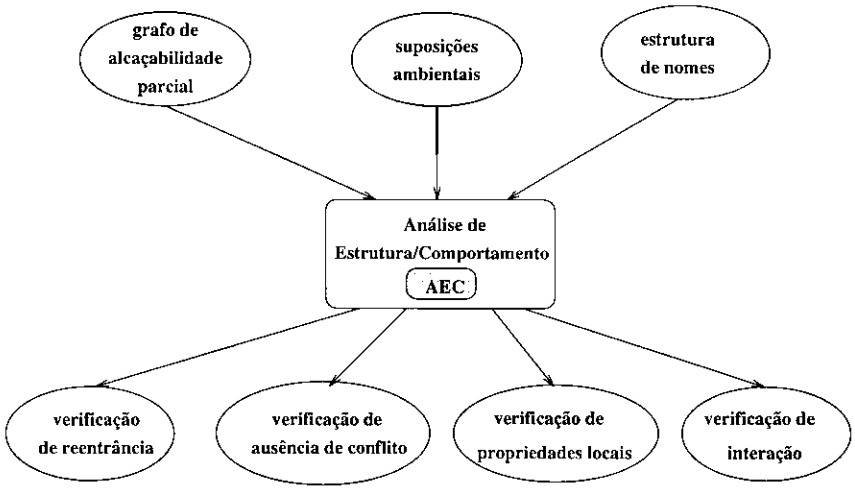

Figura 5.12: Diagrama de blocos do analizador de estrutura e comportamento

Procedimento Verifica_Propriedades_de_Reentrância(Nome_da_Rede)

1. RG ← Lê_Grafo_de_Alcaçabilidade(Nome_da_Rede)
2. Transições ← Lê_Nomes_de_TransiçõesNome_da_Rede)
3. Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
4. RG* ← Computa_Fechamento_Transitivo(RG)
5. **ParaTodo** método$_i$ ∈ métodos **Faça**
   /* verifique se a marcação dos lugares iniciais é vazia */
5.1. **ParaTodo** lugares_iniciais ∈ métodos **Faça**
5.1.1. Vazio(lugar_inicial$_i$)
5.2. **FimParaTodo**
   /* verifica alcaçabilidade e unicidade dos lugares alvos */
5.3. **ParaTodo** lugares_iniciais ∈ métodos **Faça**
5.3.1. Caminho(lugar_inicial$_i$,lugar_alvo$_i$,RG,RG*)
5.3.2. Caminho_Único(lugares_iniciais,lugares_alvo,RG,RG*)
5.4. **FimParaTodo**
   /* verifica se a rede é reinicializável */
5.5. **ParaTodo** lugares_iniciais ∈ métodos **Faça**
5.5.1. Caminho(lugar_inicial$_i$,lugar_inicial$_i$,RG,RG*)
5.6. **FimParaTodo**
6. **FimParaTodo**

Figura 5.13: Procediemento para verificar as propriedades de reentrância

Verificação de Ausência de Conflito

Discutiremos informalmente o procedimento para verificar a ausência de conflito dinâmico. Para verificar a ausência de conflito entre conjuntos de transições pertencendo a todos os métodos definidos para uma *G-Net* sendo analizada, primeiramente são identificados todos os lugares que são entradas para transições pertencendo a cada método. Para identificar estes lugares geramos a matriz de incidência para a *G-Net*. A partir desta matriz de incidência podemos identificar quais são os lugares que podem levar a um potencial conflito estático em conjunto. Quando estes lugares são identificados, verificamos se as transições para as quais estes lugares são entrada pertencem a diferentes métodos. Em caso afirmativo estas transições estão em conflito estático em conjunto. Após isto, como discutimos no Capítulo 4, devemos verificar duas condições para caracterizar o conflito dinâmico:

1. Se os lugares podem ser marcados para a mesma marcação de habilitação para todos os conjuntos de transições.

2. Se o disparo de uma destas transições disabilita a execução de outros métodos que possuem transições em conflito estático por conjunto.

Uma vez definidos os lugares que quando marcados podem levar a um conflito, pode-se analizar o grafo de alcançabilidade para verificar se a condição 2 acima é satisfeita. Em caso afirmativo dizemos que um dos métodos eventualmente estará impossibilitado de progredir, e o conjunto de transição é dito estar em conflito dinâmico por conjunto. Contrariamente, os conjuntos de transições estão livres de conflito dinâmico por conjunto.

Exemplificaremos o procedimento para verificar se um conjunto de transições está livre de conflito dinâmico utilizando os conjuntos de transições para os métodos definidos para a *G-Net* do consumidor. A partir da matriz de incidência identificamos que o lugar *RC'* é entrada para a transição *na*, pertencente ao método *ms*, e é também entrada para a transição *ac*, pertencente ao método *mc*. Portanto, estas duas transições estão em conflito estático. Supondo que temos fichas nos lugares *RC'*, *IN* e *CO* dois casos podem ocorrer. Primeiro, o disparo da transição *ac* desabilitará o disparo da transição *na*, mas a transição *sa* pode ainda disparar, e a invocação do método *ms* pode proceder. Contrariamente, se a transição *na* dispara uma ficha é depositada no lugar *RC'* e a transição *ac* estará ainda habilitada e poderá disparar, logo o método *mc* pode proceder. Portanto estes dois conjuntos de transições estão livres de conflito por conjunto.

Verificação de Propriedades Locais

Apresentamos na Figura 5.14 o procedimento para verificar as propriedades locais de uma *G-Net*. O procedimento utiliza a árvore de alcaçabilidade para verificar as propriedades locais. Utilizam-se consultas utilizando lógica temporal para formular questões relevantes sobre as propriedades da rede sendo analizada. Este procedimento não está ainda completamente implementado na atual versão do sistema. A parte mais difícil é interpretar as fórmulas em lógica temporal, de modo que estas possam ser utilizadas como consultas ao grafo de alcançabilidade. Discutimos este problema e alguns outros requisitos para implementar este procedimento quando intoduzirmos o procedimento para verificar os compromissos da rede, uma vez que os dois procedimentos são bastante similares.

Procedimento Verifica_Propriedades_Locais(Nome_da_Rede)

1. Transições ← Lê_Nomes_de_Transições(Nome_da_Rede)
2. Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
3. RG ← Lê_Grafo_de_Alcaçabilidade_Abstraído(Nome_da_Rede)
4. Compromissos ← Lê_Compromissos(Nome_da_Rede)
5. RG* ← Compute_Fechamento_Transitivo(RG)
6. **ParaTodo** Propriedades$_i$ ∈ Propriedades **Faça**
6.1. Verifica_Propriedades$_i$(RG,RG*,Lugares)
7. **FimParaTodo**

Figura 5.14: Procedimento para verificar as propriedades locais de uma $\widehat{\text{G}}$-Net

Abstração

Para extrair o comportamento abstraído para cada método de uma rede, temos que primeiramente identificar os lugares de interface para cada método. Uma vez que estes lugares de interface forem identificados, podemos então gerar o grafo de alcaçabilidade abstraído para cada método. Para executar esta tarefa utilizamos o fechamento transitivo da matriz de incidência para o grafo de alcançabilidade, o mesmo utilizado para verificar as propriedades de reentrância. Com base nesta matriz de fechamento transitivo podemos identificar os caminhos entre cada lugar de interface. Obviamente, pelo menos sabemos que os lugares alvo são alcançaveis para os métodos, uma vez que assumimos que a análise de reentrância foi completada com sucesso. Na Figura 5.15 apresentamos este procedimento.

Procedimento Extrai_Abstração(Nome_da_Rede)

```
1.          RG ← Lê_Grafo_de_Alcaçabilidade(Nome_da_Rede)
2.          Transições ← Lê_Nomes_de_TransiçõesNome_da_Rede)
3.          Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
4.          RG* ← Computa_Fechamento_Transitivo(RG)
            /* extrai grafo de alcaçabilidade abstraído para todos os métodos */
5.          ParaTodo método_i ∈ métodos Faça
               /* gere lista de nós para o método_i */
5.1.           Nós_i ← ∅;
5.2.           Nós_{m_i,0} ← Lugar_Inicial_{m_i}
5.3.           ParaTodo Lugares_de_Interface ∈ método_i Faça
5.3.1.            Nós_{m_i,j} ← Lugares_de_Interface_j
5.4.           FimParaTodo
               /* conecta lugares de interface alcançáveis por arcos */
5.5.           ParaTodo Nós_{m_i} Faça
5.5.1.            Arco_j ← ∅
                  /* encontra todos os sucessores do nós inicial Nós_{m_i,j} */
5.5.2.            ParaTodo Nós_{m_i,j} Faça
5.5.2.1.             Se Nós_{m_i,k} é sucessor de Nó_j
                                 /* adiciona sucessor do nó à lista de arcos */
5.5.2.1.1.                       Arcos_j ← Nós_{m_i,k}
5.5.3.               FimSe
5.6.              FimParaTodo
                  /* conecte linha do grafo de alcançabilidade à lista de nós */
5.7.              Liga Arcos_j a Nós_{m_i}
5.8.           FimParaTodo
6.          FimParaTodo
```

Figura 5.15: Procedimento para extrair a abstração de uma *G-Net*

5.2.5 Procedimento para Verificar os Compromissos de uma G-Net

Apresentamos agora o procedimeto para verificar se as suposições feitas a respeito do comportamento de uma *G-Net* são respeitados (garantidos). Na verdade verificamos se a interação entre as redes está correta. Como discutimos no Capítulo 4, enquanto uma rede garante as suposições feitas a respeito de seu comportamento, estamos habilitados a garantir que o sistema interage corretamente. O usuário deve prover cada uma das suposições a ser verificada em um arquivo separado. Como discutido no Capítulo 4 estas suposições são expressas em lógica temporal. Este procedimento não está implementado até este ponto. Para implementar este procedimento é necessário primeiro interpretar estas fórmulas em lógica temporal, traduzi-las, de modo que elas possam ser utilizadas como consultas ao comportamento abstraído (grafo) da rede. Verificamos se estas consultas são ou não válidas pela análise do fechamento transitivo do comportamento abstraído da rede. Na Figura 5.16 mostramos o esqueleto deste procedimento.

Procedimento Verifica_Compromissos(Nome_da_Rede)

1. Transições ← Lê_Nomes_de_TransiçõesNome_da_Rede)
2. Lugares ← Lê_Nomes_de_Lugares(Nome_da_Rede)
3. ARG ← Lê_Grafo_de_Alcaçabilidade_Abstraído(Nome_da_Rede)
4. Compromissos ← Lê_Compromissos(Nome_da_Rede)
5. ARG* ← Computa_Fechamento_Transitivo(ARG)
6. **ParaTodo** Compromisso$_i$ ∈ Compromissos **Faça**
6.1. Verifique_Compromissos$_i$(ARG,ARG*,Lugares)
7. **FimParaTodo**

Figura 5.16: Procedimento para verificar os compromissos (suposições feitas sobre o comportamento de uma *G-Net*)

Na Figura 5.17 apresentamos uma cópia de tela para a execução do procedimento de análise. Note que nem todos os procedimentos de análise estão completamente implementados. Mostramos nesta figura o fechamento transitivo para o grafo de alcaçabilidade. Com base neste fechamento transitivo executamos análise de reentrância e verificação de ausência de conflito para cada método. Ao fim da janela apresentamos os lugares pertencentes à interface da rede

Para verificar o compromisso da rede com as suposições ambientais, aplicamos o procedimento Verifica_Suposições(Nome_da_Rede), como introduzido anteriormente. Este procedimento, até este momento, é executado manualmente.

Aplicamos o sistema descrito para auxiliar na verificação da seção bidirecional para o sistema de controle de trens como introduzido no Capítulo 3. Mesmo considerando

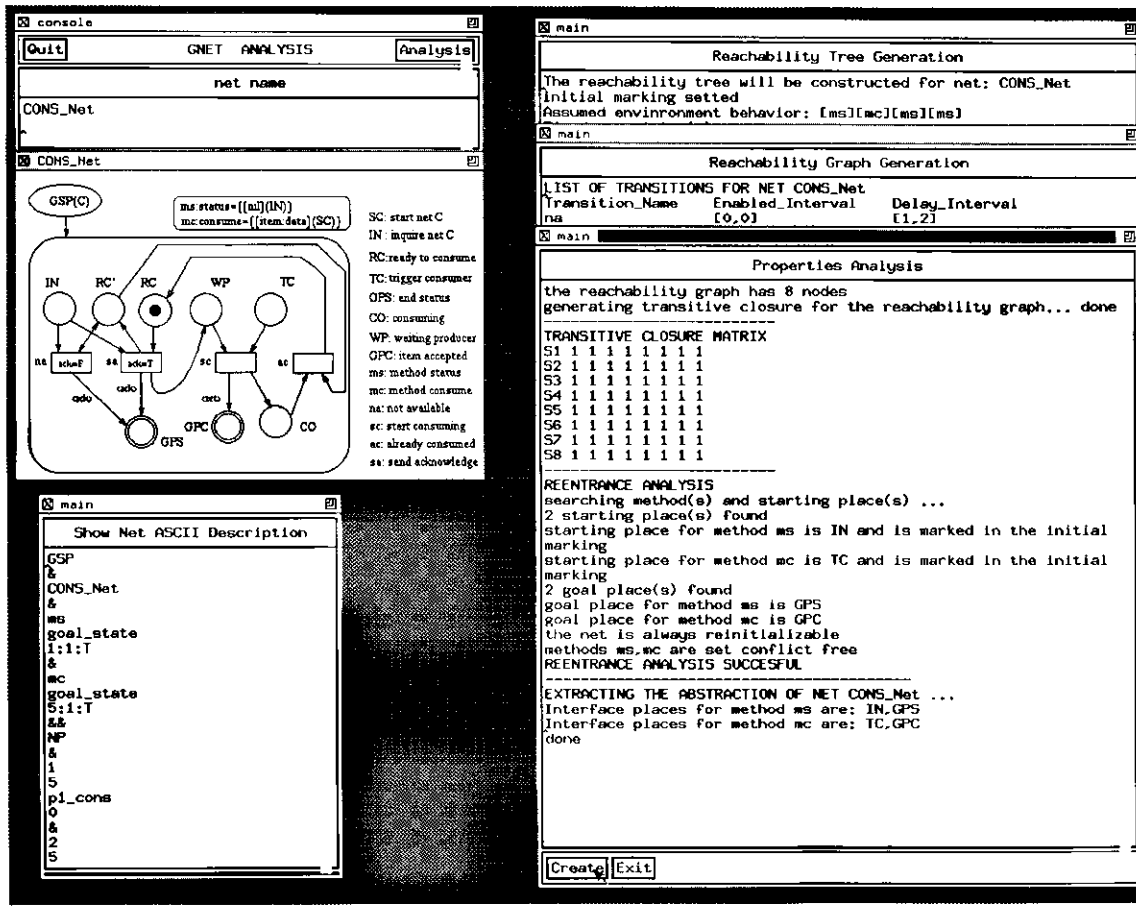

Figura 5.17: Cópia da tela mostrando a análise para a rede do consumidor

que ainda parte da verificação tem que ser executada manualmente, pois alguns procedimentos não estão completamente implementados, o sistema que introduzimos neste capítulo simplificou consideravelmente a tarefa de verificação.

Capítulo 6

DISCUSSÃO E CONCLUSÃO

6.1 Sumário

Nos capítulos precedentes introduzimos uma metodologia modular para a análise de uma classe de redes de Petri de alto-nível denominada *G-Nets*. Discutimos a motivação para uma abordagem modular como a que foi introduzida: a necessidade de evitar o problema da explosão de estados quando analizando sistemas complexos. O problema tem motivado diferentes pesquisas na área de redes de Petri, parte das quais discutimos nesta tese.

Formalizamos os conceitos de *G-Nets* de modo a possibilitar a aplicação de uma metodologia modular de análise e verificação. Definimos também propriedades estruturais e comportamentais para *G-Nets*. Quando estas propriedades estruturais e comportamentias são satisfeitas, é possível aplicar uma abordagem de verificação comportamental/lógica para evitar o problema da explosão de estados.

A introdução do conceito de reentrância, assim como a definição de propriedades relacionadas com este conceito, e a aplicação deles a *G-Nets* provê ao projetista uma ferramenta muito bem estruturada para a concepção de sistemas complexos. Na verdade, com a introdução do conceito de reentrância, juntamente com uma metodologia de decomposição apropriada para os elementos de interface de uma *G-Net*, formados pelo *GSP*, *isp* e *lugares alvo*, tornou disponível uma forma bem estruturada e definida de compor *G-Nets* em um sistema de *G-Nets*, possibilitando a aplicação de uma metodologia de análise modular.

Com o desdobramento dos lugares de interface podemos então aplicar uma metodologia modular de análise baseada no paradigma assume/garante. Esta abordagem mostrou que é possível adotar com sucesso uma abordagem modular para a análise e verificação de sistemas complexos durante a fase de concepção. O comportamento externo de uma *G-Net* pode ser extraído de uma maneira direta. Isto é possível principalmente devido as metodologias de decomposição e desdobramento. Com base nesta abstração, é possível definir quais propriedades locais de uma *G-Net* serão parte do ambiente de outras *G-Nets* em um sistema de *G-Nets*. Mais ainda, com base nesta visão

abstrata é possível verificar se as suposições a respeito do ambiente de uma *G-Net* são ou não satisfeitas.

Apresentamos também como o projetista pode sistematicamente introduzir tolerância a falhas na concepção de uma *G-Net*. A metodologia que introduzimos permite uma clara definição de pontos críticos na concepção de uma *G-Net* e como introduzir propriedades de tolerância a falhas de forma sistemática. Enfatizamos a introdução de auto-proteção na concepção de uma *G-Net* decomposta. O procedimento para introdução de auto-proteção utiliza as suposições sobre o ambiente, considerando que elas podem eventualamente não serem respeitadas. Além disto, mostramos como implementar diversidade de concepção para aumentar a confiabilidade de um sistema concebido utilizando-se a metodologia de *G-Nets*. Outras metodologias de tolerância a falhas podem também ser aplicadas para aumentar a confiabilidade de um sistema de *G-Nets*, como discutido em Perkusich et al [82], mas estas metodologias são principalmente dependentes do tempo, e estão fora do escopo deste trabalho.

A aplicação dos conceitos aos exemplos mostrados nesta tese, o problema produtor/consumidor, o protocolo cliente/servidor e o sistema de controle de veículos, mostrou a aplicabiliade teórica e prática da metodologia desenvolvida. Mais ainda, o sistema de controle de veículos mostrou que a metodologia é suficientemente poderosa para ser aplicada em sistemas complexos reais.

Considerando o aspecto prático, definimos um sistema de verificação para auxiliar o projetista no processo de verificação, de modo que é possível verificar as propriedades do sistema de forma automática. Apesar de o sistema de verificação nao estar em um estágio completo, todos os algorítmos necessários para análise foram implementados e testados. De fato, o que deve ainda ser melhorado é a interface homem-máquina.

Devemos dizer que a ênfase desta tese é nos aspectos de engenharia da metodologia. Detalhes teóricos foram introduzidos com o objetivo de fornecer uma base teórica para as alegações. Mais ainda, acreditamos que o casamento entre métodos formais e uma abordagem de engenharia é a melhor forma de encararmos as dificuldades associadas com a concepção e verificação de sistemas complexos de software. As vantagens deste casamento de conveniência são evidentes. De um lado é possível compartilhar os aspectos metodológicos inerentes a uma abordagem de engenharia para a concepção e verificação de sistemas complexos. E por outro lado, métodos formais poderosos são então disponíveis ao projetista, de modo que, mesmo em fases inicias da concepção é possível verificar se o sistema possui ou não as propriedades especificadas.

6.2 Trabalhos Futuros

Apesar do fato desta tese ser o mais auto-contida possível, existem alguns aspectos que não foram discutidos. Por exemplo, a composição definida no Capítulo 4 está restrita para os casos de multiplos chamadores e hierarquica. Seria interessante investigar outros casos como por exemplo múltiplos clientes. Neste caso a definição das propriedades de ambiente seria muito mais complexa. Considerando a abordagem por *G-Nets*, cremos que seria necessário definir uma operação de união para os chamadores, de forma que fosse possível definir as propriedades do ambiente de forma precisa. De qualquer forma, o mecanismo de composição introduzido possui expressividade suficiente para tratar com muitos tipos diferentes de sistemas, assim como o sistemas de controle de veículos mostrado nesta tese.

Outra possibilidade seria a investigação da aplicação de lógica temporal com ênfase no aspecto axiomatico. Acreditamos que esta direção é interessante. O maior problema é definir um sistema de lógica temporal capaz de tratar com predicados. Apesar de que algumas pesquisas foram apresentadas neste sentido cremos que a maior dificuldade reside no desenvolvimento de sistemas de prova automáticos ou semi-automáticos, para este tipo de lógica.

A investigação de como tratar com a criação dinâmica de processos é um outro aspecto que deve ser considerado em pesquisas futuras. Uma possível solução é a aplicação do mesmo protocolo de comunicação que definimos para a comunicação entre *G-Nets* considerando que o chamador e as redes de serviço são dinâmicamente criadas. Mais ainda, temos que considerar que a comunicação é dinamicamente estabelecida.

Considerando aspectos de implementação, a maior desvantagem da estratégia para construir a árvore de alcançabilidade é o uso de *fork* de processo para executar primitivas associadas aos lugares. Uma melhor solução seria utilizar um *servidor de primitivas*, em um ambiente de rede. Para executar uma primitiva, o gerador de árvore de alcançabilidade ao invés de executar um *fork* de processo, envia uma mensagem ao servidor de primitivas, o qual então executa a primitiva e envia o resultado. Esta abordagem pode também ser utilizada no simulador. No caso do simulador, um servidor the primitivas poderia ser um *um servidor baseado em conhecimento*, com capacidades para selecionar entre o domínio de conhecimento específico, uma primitiva que satisfize-se as funcionalidades requeridas. Obviamente, este tipo de servidor deveria ser capaz de derivar conhecimento sobre alguma base de conhecimento existente, desta forma novas primitivas poderiam ser geradas quando possível. Se o conhecimento na base de conhecimento de domínio não fosse suficiente, o servidor poderia perguntar ao usuário como

a nova primitiva poderia ser gerada.

Devemos ainda enfatizar que vislumbramos a aplicação dos resultados obtidos nesta tese em outras áreas como banco de dados e sistemas flexíveis de manufatura [85].

BIBLIOGRAFIA

[1] R.J. Abbott. Resourceful system for fault tolerance, reliability, and safety. *ACM Computing Surveys*, 22(1):35–68, March 1990.

[2] P. Alache, K. Benzakour, F. Dollé, P. Gillet, P. Rodrigues, and R. Valette. PSI: A petri net based simulator for flexible manufacturing systems. In G. Rozenberg, editor, *Advances in Petri Nets 1984*, volume 188 of *Lecture Notes in Computer Science*, pages 1–14. Springer Verlag, 1984.

[3] Annonymous. *Programming with the HP X Widgets*, November 1988.

[4] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-11(12):1491–1506, December 1985.

[5] A. Avizienis and J.C. Laprie. Dependability computing: From concepts to design diversity. *Proceeding of IEEE*, 74(5):67–80, May 1986.

[6] P. Azema, F. Vernadat, and J.C. Lloret. Requirement analysis for communications protocols. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407, pages 286—-293. Springer-Verlag, lecture notes in computer science edition, 1989.

[7] M. Baldassari, G. Bruno, V. Russi, and R. Zompi. PROTOB a hierarchical object-oriented CASE tool for distributed systems. In C. Ghezzi and J.A. McDermid, editors, *ESEC'89*, volume 387 of *Lecture Notes in Computer Science*, pages 425–445. Springer-Verlag, 1989.

[8] T. de C. Barros. A petri net modeling based technique for manufacture automation. Master's thesis, University of Pernambuco, Recife, Brasil (in Portuguese), 1990.

[9] E. Battiston, F. De Cindio, and G. Mauri. OBJSA nets: a class of high-level nets having objects as domains. In G. Rozenberg, editor, *Advances on Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Science*, pages 20–43. Springer-Verlag, 1988.

[10] F. Belli and K.-E. Grosspeitsch. Specification of fault-tolerant systems issues by predicate/transitions nets and regular expressions-approach and case study. *IEEE Transactions on Software Engineering*, 17(6):513–526, June 1991.

[11] L. Bernardinello and F. De Cindio. A survey of basic models and modular net classes. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351. Springer-Verlag, 1992.

[12] G. Booch. *Object Oriented Design with Applications.* The Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1991.

[13] G. Booch. *Object Oriented Design: With Applications.* Menlo Park, CA: Benjamin/Cummings, 1994.

[14] G.W. Brams. *Réseaux de Petri: Théorie et Pratique.* (colective name). Masson, 1983, (In French).

[15] W. Brauer, R. Gold, and W. Vogler. A survey of behaviour and equivalence preserving refinements of petri nets. In G. Rozemberg, editor, *Advances on Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 1–43. Springer-Verlag, 1991.

[16] M. Broy and T. Streicher. Modular functional modelling of petri nets with individual tokens. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 70–88. Springer-Verlag, 1992.

[17] J.R. Burch, E. Clarke, K.L. Macmillan, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, Los Alamitos, CA, 1990. IEEE Computer Society Press.

[18] J. Castro and J. Kramer. Constructing distributed system specification: A temporal-causal approach. In *Proceedings of Congress of Brazilian Computer Society, XVII SEMISH*, pages 1–15, Vitória, Brazil, 1990.

[19] J. Castro and J. Kramer. Temporal-causal system specification. In *Proceedings of IEEE International Conference on Computer Systems and Software Engineering, Compeuro90*, pages 210–217, 1990.

[20] V. Chandra and M.R. Verma. A fail-safe interlocking system for raylways. *IEEE Design & Test of Computers*, 8(1):58–66, March 1991.

[21] S.K. Chang, A. Perkusich, J.C.A. de Figueiredo, B. Yu, and M.J. Ehrenberger. The design of real-time distributed information systems with object-oriented and fault-tolerant characteristics. In *Proc. of The Fifth International Conference on Software Engineering and Knowledge Engineering*, San Francisco, California, June 1993.

[22] G. Chehaibar. Use of reentrant nets in modular analysis of colored nets. In G. Rozemberg, editor, *Advances in Petri Nets 1991*, volume 524 of *Lecture Notes in Computer Science*, pages 58–77. Springer-Verlag, 1991.

[23] T-C. Chen. *DS Simulator by Using G-Nets.* University of Pittsburgh, Department of Computer Science, Pittsburgh, PA, December 1992.

[24] T.C. Chen, Y. Deng, and S.K. Chang. A simulator for distributed systems using g-nets. In *Proceedings of 1992 Pittsburgh Simulation Conference*, Pittsburgh, PA, USA, May 1992.

[25] S. Christenssen and L. Petrucci. Towards aa modular analysis of coloured petri nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992, 13th International Conferece*, volume 616 of *Lecture Notes in Computer Science*, pages 113–133. Springer-Verlag, Sheffield, UK, 1992.

[26] E.M. Clarke, E.A. Emerson, and Sistla. A.P. Automatic verification of finite state concurrent systems using temporal logic specification: A practical approach. In *10th Annual ACM Symp. on Principles of Programming Languages*, Austin, Texas, May 1983.

[27] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 2(8):244–263, July 1986.

[28] E.M. Clarke and C.-L Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the Second Annual Symposium on Logic in Computer Science*, June 1986.

[29] W. Damm, G. Döhmen, V. Gerstner, and B. Josko. Modular verification of petri nets: The temporal logic approach. In *Stepwise Refinement of Distributed Systems*, volume 430 of *Lecture Notes in Computer Science*, pages 180–207. Springer-Verlag, 1989.

[30] J.C.A. de Figueiredo. Fuzzy time petri nets. Technical report, Department of Computer Science, University of Pittsburgh, 1992.

[31] J.C.A. de Figueiredo and A. Perkusich. Análise temporal baseada em redes de petri para sistemas de software, in portuguese. In *Aceito para publicação nos Anais do XIX Seminário Integrado de Software e Hardware, SEMISH 94*, August 1994.

[32] J.C.A. de Figueiredo, A. Perkusich, and S.K. Chang. Timing analisys of real-time software systems using fuzzy time petri nets. In *Proc. of The Sixth International Conference on Software Engineering and Knowledge Engineering*, pages 243–253, Riga, Latvia, June 1994.

[33] J.C.A. de Figueiredo, A. Perkusich, and M.E. Morais. Tolerância a falhas em sistemas de software utilizando uma abordagem por redes de petri, in portuguese. In *Anais do V Simpósio de Computadores Tolerantes a Falhas*, October 1993.

[34] R. de Lemos, A. Saeed, and T. Anderson. Analysis of timeliness in safety-critical systems. In J. Vytopil, editor, *Formal Techniques in Real-Time Fault-Tolerant Systems*, volume 571 of *Lecture Notes in Computer Science*, pages 571–592. Springer-Verlag, 1992.

[35] Y. Deng. *A Unified Framework for the Modeling, Prototyping and Design of Distributed Information Systems.* PhD thesis, Department of Computer Science, University of Pittsburgh, 1992.

[36] Y. Deng and S.K. Chang. A framework for the modeling and prototyping of distributed information systems. *International Journal of Software Engineering and Knowledge Engineering*, 2(3):203–226, September 1991.

[37] Y. Deng and S.K. Chang. Unifying multi-paradigms in software system design. In *Proc. of the 4th Int. Conf. on Software Engineering and Knowledge Engineering*, Capri, Italy, June 1992.

[38] Y. Deng, S.K. Chang, J.C.A. de Figueiredo, and A. Perkusich. Integrating software engineering methods and petri nets for the specification and prototyping of complex software systems. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 206 – 223. Springer-Verlag, Chicago, USA, June 1993.

[39] M. Diaz and G. Guidacci da Silveiva. Specification and validation of protocols by temporal logics and nets. In R.E.A. Mason, editor, *Proceedings of the IFIP 9th World Computer Comgress*, pages 47–52, Amsterdan,, 1983. North Holland.

[40] E.A. Emerson and J. Srinivasan. Branching time temporal logic. In J.W. Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in COmputer Science*. Springer-Verlag, 1988.

[41] R. Fairley. *Software Engineering Concepts*. MacGraw-Hill, New York, NJ, 1985.

[42] J.C. A. de Figueiredo. Fuzzy time petri nets. Ph.D. Thesis Proposal, Universidade Federal da Paraıba, COPELE/DEE/UFPB (also available in Portuguese), June 1993.

[43] J.C.A. de Figueiredo. *Fuzzy Time Petri Nets*. PhD thesis, Curso de Pòs Graduação em Engenharia Elétrica, Universidade Federal da Paraíba, Campina Grande, PB, August 1994.

[44] S. Fisher, A. Scholtz, and D. Taubner. Verification in process algebra of the distributed control of track vehicles – a case study. In G.V. Bochman and D.K. Probst, editors, *CAV'92, Computer Aided Verification, 4th Int. Workshop*, volume 663 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1992.

[45] H.J. Genrich. Predicate/Transition nets. In W. Brauer, W. Reisig, and G. Rozemberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 207–247. Springer-Verlag, 1987.

[46] H.J. Genrich. Equivalence transformations of PrT-Nets. In G. Rozenberg, editor, *Advances in Petri Nets 1989*, volume 424 of *Lecture Notes in Computer Science*, pages 179–208. Springer-Verlag, 1989.

[47] H.J. Genrich. Predicate/Transition nets. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets: Theory and Application.* Springer-Verlag, 1991.

[48] H.J. Genrich and K. Lautenbach. System modeling with high level-petri nets. *Theoretical Computer Science*, 13:109–136, 1981.

[49] H.J. Genrich and E. Stankiewicz-Wiechno. A dictionary of some basic notations od net theory. In W. Brauer, editor, *Net Theory and Applications, Proc . of the First Advanced Course General Net Theory of Systems and Process*, volume 84 of *Lecture Notes in Computer Science*, pages 519–535. Springer-Verlag, 1980.

[50] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A unified high-level petri net formalism for time-critical systems. *IEEE Transactions on Software Engineering*, 17(2):160–171, February 1991.

[51] A. Giordana and L. Saitta. Modeling production rules by means of predicate transition networks. In *Information Sciences*, volume 35, pages 1–41. Elsevier Science Publishing, 1985.

[52] A. Giordana and L. Saitta. Modelling production rules by means of predicate transition networks. *Information Sciences*, 35:1–41, 35.

[53] O. Grumberg and D.E. Long. Model checking and modular verification. In K. Baeten, editor, *CONCUR'91*, volume 458 of *Lecture Notes in Computer Science*, pages 250–265. Springer-Verlag, 1991.

[54] O. Grumberg and D.E. Long. Model checking and modular verification. personal communication, June 1992.

[55] X. He and J. A. Lee. A methodology for constructing predicate transition nets specifications. *Software–Practice and Experience*, 21(8):845–875, August 1991.

[56] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Reachability trees for high-level petri nets. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets.* Springer-Verlag, 1991.

[57] P. Huber, K. Jensen, and R.M. Shapiro. Hierarchies in coloured petri nets. In Jensen. K. and G. Rozenberg, editors, *High-Level Petri Nets: Theory and Application*, pages 313–341. Springer-Verlag, 1991.

[58] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object Oriented Software Engineering: A Use Case Driven Approach.* Addison-Wesley, 1992.

[59] K. Jensen. Coloured petri nets: A high level language for system design and analysis. In *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science.* Springer-Verlag, 1990.

[60] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use.* EACTS – Monographs on Theoretical Computer Science. Springer-Verlag, 1992.

[61] B.W. Johnson. Design and analysis of fault-tolerant systems for industrial applications. In W. Goke and H. Sorensen, editors, *Informatik-Fachberichte, Fault-Tolerant Computing Systems*, volume 214, pages 57–73. Springer-Verlag, 1989.

[62] B. Josko. MCTL: An extension of CTL for modular verification of concurrent systems. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logics in Specification*, volume 398 of *Lecture Notes in Computer Science*, pages 165–187. Springer-Verlag, 1987.

[63] B. Josko. Verifying the correctness of aadl modules using model checking. In *Stepwise Refinement of Distributed Systems*, volume 430 of *Lecture Notes in Computer Science*, pages 386–400. Springer-Verlag, 1989.

[64] B. Josko. A context dependent equivalence relation between kripke structures. In E.M. Clarke and R.P. Kurshan, editors, *2nd International Conference on Computer Aided Verification CAV'90*, volume 531 of *Lecture Notes in Computer Science*, pages 204–213. Springer-Verlag, 1991.

[65] J.P.J. Kelly, T.I. McVittie, and W.I. Yamamoto. Implementing design diversity to achieve fault-tolerance. *IEEE Software*, 8(3):61–71, July 1991.

[66] K.G. Larsen and B. Thomsen. Partial specification and compositional verification. *Theoretical Computer Science*, 88:15–32, 1991.

[67] K.H. Lee and J. Favrel. Hierarchical reduction method for analysis and decomposition of petri nets. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(2):272–280, March 1985.

[68] N.G. Levenson. Software safety: Why, what, and how. *ACM Computing Surveys*, 18(2):125–163, June 1986.

[69] N.G. Leveson and J.L. Stolzy. Analyzing safety and fault tolerance using time petri nets. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science, Proceedings of TAPSOFT'85*, volume 186, pages 339–355. Springer-Verlag, 1985.

[70] N.G. Leveson and J.L. Stolzy. Safety analysis using petri nets. *IEEE Transactions on Software Engineering*, SE-13(3):386–397, March 1987.

[71] M. Lindqvist. Parameterized reachability trees for predicate/transition nets. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets*, pages 351–372. Springer-Verlag, 1991.

[72] Luqi. Software Evolution Through Rapid Prototyping. *IEEE Transactions on Computers*, pages 13–25, May 1989.

[73] K.L. MacMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, May 1992.

[74] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, New York, NJ, 1992.

[75] G. Memmi and J. Vautherin. Analysing nets by the invariant method. In G. Rozenberg, editor, *High-Level Petri Nets: Theory and Application*, pages 247–336. Springer-Verlag, 1991.

[76] P.M. Merlin and D.J. Farber. Recoverability of communication protocols - implications of a theoretical study. *IEEE Transactions on Communication*, COM-24(9):1036–1043, September 1976.

[77] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, April 1989.

[78] T. Murata and D. Zhang. A predicate-transition net model for parallel interpretation of logic programs. *IEEE Transactions on Software Engineering*, 14(4):481–497, April 1988.

[79] H. Oswald, R. Esser, and R. Mattmann. An environment for specifying and executing hierarchical petri nets. In *Proceedings of the 12th International Conference on Software Engineering*, pages 164–172, 1990.

[80] A. Perkusich, T.C. Barros, J.C.A de Figueiredo, and M.L.B. Perkusich. A petri net based approach for knowledge base construction for fault analysis and control of discrete time systems. In *IEEE Industrial Electronics Society Conference IECON'91*, pages 1631–1636, Kobe, Japan, November 1991.

[81] A. Perkusich, T.C. Barros, M.L.B. Perkusich, D.S. Barbalho, and J.C.A. de Figueiredo. Knowledge based systems application to implement petri net models of discrete time systems. In *IFIP Working Conference on Dependability of Artificial Inteligence Systems*, Viena, Austria, May 1991.

[82] A. Perkusich, J.C.A. de Figueiredo, and S.K Chang. Embedding fault-tolerant properties in the design of complex systems. *Journal of Systems and Software*, 2(25):23–37, 1994.

[83] A. Perkusich, J.C.A. de Figueiredo, and M.E. Morais. Análise e verificação de sistemas baseados em objetos utilizando uma abordagem por redes de petri, in portuguese. In *Anais do XIX Seminário Integrado de Software e Hardware, SEMISH 93*, September 1993.

[84] A. Perkusich, J.C.A. de Figueiredo, and M.E. Morais. Projeto de sistemas em tempo real distribuídos com característica baseada em objetos e tolerância a falhas, in portuguese, in portuguese. In *Anais do XIX Seminário Integrado de Software e Hardware, SEMISH 93*, September 1993.

[85] M.L.B Perkusich, A. Perkusich, and U. Schiell. Modelo orientado a objetos para modelagem de sistemas e bancos de dados em tempo-real, in portuguese. In *Aceito para publicação nos Anais do IX Simpósio Brasileiro de Banco de Dados, IX SBDD*, August 1994.

[86] G. Peterka and T. Murata. Proof procedure and answer extraction in petri net model of logic programs. *IEEE Transaction on Software Engineering*, 15(2):209–217, February 1989.

[87] J.L. Peterson. *Petri Net Theory and Modeling of Systems.* Prentice-Hall, 1981.

[88] A. Pnueli. In transition from global to modular temporal reasoning about programs. In K.R. Apt, editor, *Logics and Models of Concurrent Systems, NATO ASI series, Series F, Computer and Systems Sciences*, volume 13. Springer-Verlag, 1984.

[89] L Pomello, G. Rozenberg, and C. Simone. A survey of equivalence notions for net based systems. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 410–472. Springer-Verlag, 1992.

[90] M. Rauhamaa. A comparative study of methods for efficient reachability analysis. Technical Report 14, Helsinki University of Technology, Espoo, Finland, September 1990.

[91] W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag, 1985.

[92] W Reisig. Petri nets with individual tokens. *Theoretical Computer Science*, 41:185–213, 1985.

[93] W. Reisig. Place/transition systems. In W Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and their Properties, Proc. of 2nd Advanced Course on Petri Nets*, volume 254 of *Lecture Notes in Computer Science*, pages 117–141. Springer-Verlag, 1987.

[94] W. Reisig. Temporal logic and causality in concurrent systems. In *International Conference on Concurency, CONCUR'88*, volume 335 of *Lecture Notes in Computer Science*, pages 121–139. Springer-Verlag, 1988.

[95] W. Reisig. Towards a temporal logic for causality and choice in distributed systems. In J.W. de Bakker, W.-P Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 603–627. Springer-Verlag, 1988.

[96] W. Reisig. Combining petri nets and other formal methods. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 24–44. Springer-Verlag, 1992.

[97] D.A. Rennels. Fault tolerant computing-concepts and examples. *IEEE Transactions on Computers*, C-33(12):1116–1129, December 1984.

[98] P. Robinson, editor. *Object-oriented Design.* Chapman & Hall, 1992.

[99] G. Rozenberg and P.S. Thiagarajan. Petri-nets: Basic notions, structure, behaviour. In J.W. de Bakker, W.-P. Roever, and G. Rozenberg, editors, *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science.* Springer-Verlag, 1986.

[100] S.M. Shatz. *Development of Distributed Software.* Macmillan Publishing Company, New York, 1993.

[101] Y. Shieh, D. Ghosal, P.R. Chintamaneni, and K. Tripathi. Application of petri net models for the evaluation of fault-tolerant techniques in distributed systems. Technical Report Series CS-TR-2128, University of Maryland, College Park, Maryland, 20742, USA, October 1988.

[102] Ye Souissi and G. Memmi. Composition of nets via a communication medium. In G. Rozemberg, editor, *Advances on Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 455–470. Springer-Verlag, 1991.

[103] I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of petri nets. *Journal of Computer and System Sciences*, 27:61–76, 1983.

[104] UNIX System Laboratories, Inc., editor. *Open Look, Graphical User Interface: Programmer's Guide.* Unix Press, 1992.

[105] P.S. Thiagarajan. Elementary net systems. In W Brauer, W. Reisig, and G. Rozenberg, editors, *Petri-Nets: Central Models and their Properties, Proc. of and Advance Course on Petri Nets*, volume 254 of *Lecturte Notes in Computer Science*, pages 26–59. Springer-Verlag, 1987.

[106] H. Tuominen. Logic in petri net analysis. Technical Report 5, Helsinky University of Technology, Espoo, Finland, January 1988.

[107] N. Uchihira and S. Honiden. Verification and synthesis of concurrent programs using petri nets and temporal logic. Technical Report TM-0900, Institute for New Generation Computer Technology, ICOT, Tokey, Japan, July 1990.

[108] R. Valette. Analysis of petri nets by stepwise refinements. *Journal of Computer and Systems Sciences*, 18:35–46, 1979.

[109] R. Valette, M. Corvousier, and C. Desclaux. Putting petri nets to work for controlling flexible manufacturing systems. In *Proc. of IEEE International Symposyum on Circuits and Systems, ISCAS 85*, Kyoto, Japan, 1985.

[110] D.A. Young. *The X Window System Programming and Applications with Xt.* Prentice Hall, New Jersey, 1990.