Dalton Dario Serey Guerrero

SISTEMAS DE REDES DE PETRI MODULARES BASEADOS EM OBJETOS

Fevereiro de 1997

Dalton Dario Serey Guerrero

SISTEMAS DE REDES DE PETRI MODULARES BASEADOS EM OBJETOS

Dissertação apresentada ao Curso de Mestrado em Informática da Universidade Federal da Paraíba como requisito parcial à obtenção do título de Mestre em Informática.

Área de concentração: Sistemas de Software

Orientadores: Prof. Jorge César Abrantes de Figueiredo
Prof. Angelo Perkusich

Campina Grande
Universidade Federal da Paraíba
Fevereiro de 1997

G934s Guerrero, Dalton Dario Serey
Sistemas de rede de petri modulares baseados em objetos / Dalton Dario Serey Guerrero. - Campina Grande, 1997.
105 f. : il.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Software 2. Rede de Petri - 3. Sistemas de Rede 4. Dissertacao I. Figueiredo, Jorge Cesar Abrantes de, Dr. II. Perkusich, Angelo, Dr. III. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.41(043)

SISTEMAS DE REDES DE PETRI MODULARES EM OBJETOS

DALTON DARIO SEREY GUERRERO

DISSERTAÇÃO APROVADA EM 07.03.97

PROF. JORGE CÉSAR ABRANTES DE FIGUEIREDO, Dr.
Presidente

PROF. ANGELO PERKUSICH, D.Sc
Examinador

PROF. JOSÉ REINALDO DA SILVA, Ph.D
Examinador

PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr.
Examinador

PROF. ARTURO HERNANDEZ DOMINGUEZ, Dr.
Examinador

CAMPINA GRANDE - PB

Em memória de meus avôs

Francisco René Serey e *Hector Melchor Guerrero.*

AGRADECIMENTOS

Apresento aqui meus mais profundos agradecimentos a todos que de alguma forma contribuíram com sua presença e força quanto a minha formação. Agradeço a todos os colegas e professores do Curso de Mestrado em Informática por proporcionarem o ótimo ambiente de trabalho e convivência, e pelas eternas conversas durante os cafezinhos.

Agradeço a meus orientadores pela confiança depositada desde o início das atividades e pelo ótimo trabalho de direção sem o qual este trabalho jamais haveria sido possível. Agradeço-lhes ainda pela total disposição à discussão e por proporcionarem sempre um ambiente dinâmico onde o conhecimento prevalece a quaisquer divergências. Agradeço tudo o que aprendi.

Finalmente gostaria de agradecer por tudo isto a minha família. Primordialmente, agradeço a meus pais porque jamais pouparam esforços de qualquer natureza para me proporcionar saúde, educação, conhecimento e carinho. A minha irmã pela compreensão e colaboração nos momentos finais deste trabalho. A minha noiva e amiga por perdoar a minha ausência em tantos momentos e pela incessante motivação e encorajamento. Agradeço porque nunca duvidaram do meu trabalho.

Meus sinceros agradecimentos a todos.

RESUMO

Neste trabalho expomos um modelo formal de descrição, especificação, e modelagem de sistemas complexos de software, baseado em redes de Petri e orientação a objetos. O trabalho é impulsionado pela crescente necessidade de ferramentas com embasamento matemático para o efetivo estabelecimento de uma disciplina de Engenharia de Software. As definições são desenvolvidas de forma gradual de maneira a incluir uma-a-uma as características desejáveis no modelo. Duas aplicações da ferramenta desenvolvida são discutidas como elementos de experimentação e validação do modelo. Permitem, ainda, a detecção de eventuais problemas e/ou efeitos inesperados causados pelo formalismo desenvolvido, abrindo inúmeras possibilidades de estudos futuros para o pleno desenvolvimento da ferramenta.

ABSTRACT

In this work, we introduce a tool which allows the formal description, specification and modeling of complex software systems based on Petri Nets Theory and Object Orientation. Our work is motivated by the growing necessity of mathematical based tools in order to establish an effective Software Engineering Discipline. All definitions related to the tool, are presented through a gradual approach which allows the inclusion of the desired features in a one-per-one basis. Two applications of this tool are discussed with the objective of experimentation and validation. The applicattions, also allow the detection of eventual mistakes and/or unexpected effects caused by the underlying formalism. Thus, opening several research possibilities in order to develop and refine the introduced tool.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

É através do emprego de métodos rigorosos e de um forte embasamento matemático para as linguagens de modelagem que as disciplinas de engenharia já estabelecidas têm permitido a construção e o desenvolvimento de projetos de maior dimensão. O desenvolvimento de sistemas de software é uma atividade essencialmente de engenharia. A concepção, a validação e a disseminação de métodos matemáticos que suportem as atividades de análise e síntese de sistemas complexos de software são metas essenciais para a consolidação de uma disciplina de *Engenharia de Software* [Parnas, 1996; Lutz, 1996; Pressman, 1992].

Os procedimentos matemáticos para desenvolvimento de software, ou os *métodos formais*, como são mais conhecidos, têm sido objeto de estudo por aproximadamente 30 anos. Seu desenvolvimento derivou das pesquisas sobre modelos de computação. A utilização de métodos formais para a construção de software foi proposta inicialmente para a interação entre as fases de definição de requisitos e especificação. Até hoje, diversos autores restringem sua aplicação a apenas essas duas fases do ciclo de vida do software. Entretanto, as vantagens e os benefícios implicados pelo embasamento matemático são por demais evidentes para que não sejam aproveitados em fases subsequentes.

As principais vantagens da utilização de métodos matemáticos são a eliminação de ambigüidades e a possibilidade de estabelecimento de métodos algorítmicos de análise. Dado que a semântica de uma especificação formal é única, pode-se submeter um conjunto de expressões em algum dado formalismo, a processos sistemáticos de verificação de consistência e/ou completude sob certos aspectos. A partir de uma especificação matemática, procedimentos formais de análise podem derivar expressões que denotam características próprias da especificação, como invariantes, por exemplo. Estes argumentos buscam valorizar o uso dos métodos matemáticos como forma de assegurar a *confiabilidade* dos sistemas de software. Requisito de extrema importância quando da construção de sistemas críticos, onde falhas podem levar a perdas

significativas de recursos materiais ou mesmo vidas humanas, como em sistemas médicos, sistemas de controle de tráfego aéreo, sistemas de geo-processamento, aplicações industriais ou aplicações espaciais.

Uma outra argumentação a favor da utilização de práticas matemáticas e rigorosas de desenvolvimento, predica que através do seu uso faz-se possível eliminar erros residuais de especificação e projeto, evitando desta forma, as freqüentes necessidades de retomar atividades relativas a fases anteriores no ciclo de vida. Este argumento, não menos importante que o anterior, demonstra o caráter econômico associado à concepção, ao desenvolvimento e à utilização de métodos matemáticos que provavelmente será determinante na sua disseminação e adoção pela indústria. Boehm conclui com otimismo, em suas previsões, que um total aproximado de $600 bilhões de dólares será gasto no desenvolvimento de sistemas de software no ano 2000 [Boehm, 1987]. Entretanto, é conhecido o fato de que uma parcela bastante significativa do custo de software é relativa às atividades de teste e manutenção, o que vem a corroborar a utilização de práticas e métodos rigorosos de desenvolvimento.

Entretanto, apesar dos argumentos anteriormente expostos, não se tem verificado até hoje a consolidação maciça de nenhum método em particular entre os desenvolvedores. O maior problema com os métodos concebidos parece estar relacionado com o fato de que engenheiros de software e desenvolvedores não são matemáticos (mesmo em outras disciplinas). As ferramentas, entretanto, foram desenvolvidas para serem usadas por pessoas que detêm interesse no formalismo em si. O desenvolvedor precisa compreender profundamente a matemática e a lógica associadas, a fim de utilizar-se corretamente dos métodos.

Durante as fases de definição de requisitos e especificação, os modelos gerados pelos desenvolvedores devem servir para a comunicação com o usuário que detém o conhecimento sobre o domínio da aplicação. A qualidade da comunicação é de extrema importância para que se garanta que os requisitos especificados para o sistema capturem de fato os requisitos críticos do usuário. E, definitivamente, sistemas representados através de conjuntos de expressões em lógica ou de equações não são adequados para a interação necessária com o usuário. Devido a estas razões, os métodos formais estabelecidos até o presente momento são considerados impraticáveis e somente ajudam a dificultar o trabalho do desenvolvedor!

A fim de atingir o propósito de ver plenamente estabelecida uma disciplina de engenharia de software, é necessário resolver estes problemas e prover melhores ferramentas matemáticas, cuja notação seja aceitável para os desenvolvedores, e cuja estrutura formal permita ainda efetuar procedimentos de análise que extraiam características realmente relevantes. É importante ressaltar, que o formalismo por si só não se justifica. É preciso prover mecanismos de análise e pelo menos diretrizes que guiem a construção de ferramentas de software que as efetuem. Parece ser promissor, se não necessário, comprometer o estabelecimento das novas ferramentas formais a domínios mais restritos, no sentido de promover a sua aplicação efetiva na construção de classes bem-determinadas de sistemas de software reais [Zave, 1996].

1.1 Redes de Petri

Redes de Petri são uma ferramenta com embasamento matemático rigoroso que permitem a modelagem de sistemas cuja natureza é essencialmente concorrente, assíncrona, distribuída, paralela, não-determinística e/ou estocástica [Murata, 1989].

Além de serem um formalismo matemático, as redes de Petri são uma ferramenta de expressão gráfica e simulação. São gráficas por apresentarem uma notação que permite a completa associação entre elementos de natureza léxica e seus correspondentes sintáticos. A faceta gráfica de redes de Petri permite a melhor interação entre diferentes indivíduos, reduzindo problemas de comunicação. Como as definições semânticas também são matemáticas, torna-se possível a simulação das atividades concorrentes e dinâmicas dos sistemas modelados através da utilização de fichas para demarcar estados dos sistemas.

As redes de Petri podem e vêm sendo utilizadas para a descrição e especificação de sistemas de software. Como ferramenta matemática, permitem estabelecer equações de estados, e/ou expressões que compreendam a natureza comportamental dos sistemas, permitindo o rigorismo formal e a precisão na descrição de diagramas de fluxo e de estados, bem como a extração e verificação de características subjacentes. Daí sua larga aplicação na descrição e estudo de sistemas de informação distribuídos e paralelos e em protocolos de comunicação. A simulação do comportamento de sistemas modelados por redes de Petri, permite a visualização da dinâmica do sistema por parte dos

desenvolvedores e usuários do sistema, forma de reforçar a confiabilidade nos requisitos e na especificação.

Embora, pareça resolver em parte o problema relacionado à notação das ferramentas matemáticas, o modelo clássico de redes de Petri carece ainda de diversas características que o tornem apropriado para a modelagem de sistemas complexos reais. Em geral, a descrição de sistemas complexos reais através de redes de Petri clássicas tornam-se extremamente extensas em termos da estrutura de rede. Percebe-se também que diversos sub-módulos do modelo se repetem inúmeras vezes, caracterizando certa redundância de expressão. Além disso, o modelo clássico não favorece a modelagem de aspectos temporais quantitativos dos sistemas, permitindo apenas estabelecer relações de dependência de eventos em termos de expressões temporais qualitativas.

Diversas extensões foram propostas ao modelo clássico com o intuito de sanar as limitações citadas. Uma classe de extensões de redes de Petri agrega a possibilidade de tratamento quantitativo de aspectos temporais dos sistemas. Em geral, as ferramentas permitem a associação de atributos de tempo à semântica do modelo. Assim, torna-se possível especificar além da possibilidade da ocorrência de eventos num sistema, as limitações temporais às quais os eventos são restritos. A classe de redes de Petri temporais é de especial interesse nas atividades de verificação e avaliação de desempenho de sistemas de software [Perkusich, 1994a].

Uma outra classe de extensões de redes de Petri, denominadas redes de Petri de Alto Nível, propõe diversos mecanismos para a simplificação dos modelos obtidos, através do enriquecimento do poder de expressão. A principal contribuição deste enfoque é a integração das redes de Petri à *Teoria dos Tipos de Dados*. Em redes de Petri de Alto Nível, as fichas que denotam o estado do sistema podem representar tipos estruturados de dados, ao invés da simples indicação binária que a presença da ficha representa nas redes Clássicas. Isto permite que os modelos sejam simplificados através da fatoração de subestruturas semelhantes, que agora poderão reconhecer as fichas sobre as quais atuam. As abordagens de redes de Petri de Alto Nível que se destacam são as redes Predicado/Transição (PrT-Nets) [Genrich, 1987] e as redes de Petri Coloridas (CP-Nets) [Jensen, 1987; Jensen, 1992].

Entretanto, apesar da introdução das Redes de Petri de Alto nível, a especificação e a modelagem de sistemas complexos pode ainda ser uma tarefa difícil. A principal limitação remanescente é a falta de mecanismos que permitam especificar a estrutura dos sistemas. É natural que projetistas concebam os sistemas através da focalização de determinados aspectos do problema de cada vez, e que a certo nível de abstração desejem uma visão global das diversas partes do modelo, desta forma, estabelecendo como se dá a relação entre as partes, sem se preocupar com detalhes internos de cada uma. Esta forma de abordar a modelagem de sistemas não é suportada pelas redes de Petri de Alto nível convencionais. Como resultado, especificações e modelos de sistemas complexos elaborados através de redes de Petri de Alto Nível têm aspecto essencialmente "plano" onde toda a estrutura faz parte de um único módulo , dado que a natureza dos paradigmas não permite identificar subestruturas do sistema.

Para tratar o problema, novas extensões de redes de Petri foram propostas. A idéia inicial era permitir a introdução do conceito de hierarquia, como nos casos das redes de Petri Predicado/Transição Hierárquicas e as redes de Petri Coloridas Hierárquicas. Embora nem todos os mecanismos de abstração importantes e desejáveis sejam hierárquicos, a facilidade de modelagem introduzida permitia controlar melhor a complexidade dos sistemas em estudo.

1.2 Redes de Petri e Orientação a Objetos

Nos últimos anos, um novo paradigma emergiu como alternativa às atividades de desenvolvimento de sistemas de software: a *orientação a objetos*.

O paradigma é baseado em conceitos simples como objetos, atributos, classes, membros e relações todo-parte. A principal razão do seu sucesso e crescente uso é que esses conceitos básicos associados ao princípio da informação escondida (*information hidding*) e da abstração [Pressman, 1992] provêem um sólido esquema de estruturação e controle de complexidade para sistemas de software. Como argumento adicional de caráter econômico, a orientação a objetos permite amplamente a reutilização de elementos anteriormente descritos, através dos princípios da herança e da agregação.

Apesar do manifesto poder de expressão, a orientação a objetos carece ainda de bases matemáticas mais sólidas. Diversos grupos de pesquisa vêm estudando e propondo bases formais para o paradigma. Uma linha atual de pesquisa tenta integrar o poder de

estruturação provido pelos conceitos de orientação a objetos ao sólido embasamento matemático de que dispõem os modelos de redes de Petri. Do ponto de vista de redes de Petri, a idéia é capturar os conceitos da orientação a objetos e incorporá-los a fim de criar novas extensões com poder agregado de estruturação de sistemas.

As abordagens usadas para unir os dois modelos são diversas. Bastide revela a existência de duas tendências maiores às quais denomina "objetos dentro de redes de Petri" e "redes de Petri dentro de objetos" [Bastide, 1995]. A primeira diz respeito às pesquisas que procuram permitir a utilização de objetos como fichas em redes de Petri, de forma semelhante à utilizada nas fichas com dados da classe de redes de Petri de Alto Nível. A segunda tendência procura usar redes de Petri para modelar o comportamento interno dos objetos, em resposta a estímulos externos (mensagens). O autor finaliza indicando que há evidências de que abordagens que combinem as duas tendências terão maior possibilidade de incorporar todos os conceitos de forma satisfatória.

Uma das primeiras tentativas de integrar redes de Petri aos conceitos de orientação a objetos resultou nas redes denominadas PROT [Baldassari, 1989] e na ferramenta CASE denominada PROTOB. Estas redes suportam uma metodologia orientada a objetos equivalente à associada a HOOD (*Hierarchical Object Oriented Design*) [Robinson, 1992]. Diversas outras abordagens encontram-se à disposição na literatura, entre elas citamos POT/POP (*Paralell Object Transition* e *Paralell Object Place*) [Engelfriet, 19XX], *SimCon* [Verkoulen, 1993], LOOPN [Lakos, 1991], OBJSA [Battiston, 1993], CO-OPN [Buchs, 1991] e, finalmente, OPN [Lakos, 1995].

Dentre os modelos citados, apenas OPN integra todos os conceitos de orientação a objetos às redes de Petri, incluindo herança, polimorfismo e ligação dinâmica. Os demais modelos incorporam apenas alguns dos princípios deixando outros de lado.

Uma outra abordagem à integração dos conceitos de orientação a objetos e redes de Petri é o modelo G-Net introduzido por Deng [Deng, 1992]. G-Net é um modelo de especificação multi-nível executável baseado em redes de Petri. G-Net é baseado nos conceitos de Instanciação e Abstração de Entidades Lógicas (*LEAI*). O modelo G-Net incorpora algumas noções da orientação a objetos, de forma a tornar possível a construção rápida de protótipos executáveis de especificações de sistemas multi-níveis, tornando exeqüível a especificação incremental.

De um ponto de vista pragmático para a engenharia de software, entretanto, todas as abordagens ainda apresentam problemas. Em geral, ainda são de difícil utilização, e obrigam desenvolvedores a compreender seu formalismo subjacente a fim de que aproveitem os resultados prometidos. Noutros casos, o formalismo associado não facilita o desenvolvimento de métodos de análise, tornando a ferramenta virtualmente estéril. Além disso, o formalismo por si só não se justifica, como já foi dito. É preciso melhorar em termos de notação e desenvolver efetivamente métodos algorítmicos de análise para os modelos já estabelecidos, bem como ferramentas de software que auxiliem a modelagem, simulação e análise.

Conseqüentemente, a integração entre redes de Petri e orientação a objetos é ainda objeto de estudo. É necessário além disso, validar a utilidade dos métodos através da sua aplicação a domínios complexos na construção efetiva de sistemas de software reais.

1.3 Proposta da Dissertação

O trabalho desenvolvido durante esta pesquisa procurou aprofundar o conhecimento consolidado a respeito da integração entre as abordagens de redes de Petri e orientação a objetos para análise e síntese de sistemas distribuídos de software.

Diversos estudos a respeito do assunto foram pesquisados. Algumas abordagens de redes de Petri de Alto Nível - G-Nets [Deng, 1992; Perkusic, 1994] e redes de Petri Coloridas [Jensen, 1992], bem como suas respectivas definições formais e metodologias de modelagem. Em seguida, procuramos estabelecer diretrizes de integração dos modelos, de forma a utilizar a abordagem de G-Nets, quanto à utilização dos conceitos de orientação a objetos, em redes de Petri Coloridas. O resultado deste casamento é um modelo híbrido de redes de Petri baseado em objetos denominado G-CPN.

Após definido o novo modelo, estudamos as relações existentes entre sistemas G-CPN e as redes de Petri Coloridas puras. O resultado deste estudo foi o desenvolvimento de um algoritmo que permite a transformação de sistemas G-CPN em sistemas CPN semanticamente equivalentes. A importância da existência de uma técnica formal de transformação reside na possibilidade da aplicação imediata dos métodos formais de verificação e análise já consolidados e implementados para redes de Petri Coloridas em sistemas G-CPN. Além disso, esta relação estabelece possíveis pontos de partida para o desenvolvimento de métodos específicos de análise e verificação de sistemas G-CPN.

Finalmente, com o intuito de validar o novo modelo, procuramos investigar a sua aplicação na especificação de dois tipos de sistemas distribuídos. O primeiro deles é um sistema de edição cooperativa de diagramas hierárquicos. Em seguida, aplicamos a nova ferramenta como proposta para a especificação formal de entidades gerenciadas em sistemas de gerência de redes ATM [Soares, 1995].

1.4 Escopo e Relevância desta Pesquisa

A crescente necessidade de ferramentas matemáticas para o desenvolvimento de produtos de software de natureza complexa e concorrente é um fato. Entretanto, não existem ainda, soluções completamente satisfatórias nesse sentido. Embora as abordagens por Redes de Petri tenham sido concebidas a pelo menos 30 anos atrás, e venham sendo colocadas como fortes candidatas a solução, não há ainda produtos de desenvolvimento consolidados no meio acadêmico, e muito menos na indústria. É necessário antes, que as ferramentas matemáticas se estabeleçam e consolidem.

Este trabalho tem sua importância no sentido em que visa fazer progredir o conhecimento sobre especificação formal de sistemas distribuídos, concorrentes e complexos propondo melhorias às abordagens já consolidadas para sua modelagem, guardando assim certa continuidade e compromisso com trabalhos previamente desenvolvidos. Como forma de validar o novo modelo, escolhemos aplicá-lo à especificação de um ambiente cooperativo de edição que poderá perfeitamente ser utilizado para a modelagem de sistemas usando a própria abordagem, com o propósito de encaminhar e elaborar as diretrizes necessárias à construção do ambiente em questão.

Por fim, mas não menos importante, espera-se que este trabalho tenha contribuído com o estudo sobre a especificação/construção de ferramentas de edição cooperativa (*groupwares*) e sobre a utilização de sistemas multi-agentes na especificação/modelagem de sistemas que permitam trabalho cooperativo suportado por computadores - *Computer Supported Cooperative Work* - CSCW.

1.5 Estrutura da Dissertação

O restante deste documento está organizado como segue: o Capítulo 2 apresenta uma revisão dos conceitos básicos de redes de Petri, redes de Petri Coloridas e G-Nets, assim como suas definições formais. O Capítulo 3 apresenta informalmente os módulos G-CPN

a partir do qual sistemas G-CPN são elaborados e uma série de definições formais de redes de Petri intermediárias que visam agregar incrementalmente os conceitos de orientação a objetos presentes em G-Nets às redes CPN, culminando na definição formal de um módulo G-CPN. O Capítulo 4 define Sistemas G-CPN e aborda a relação entre Sistemas G-CPN e sistemas CPN puros, propondo uma técnica de transformação. O Capítulo 5 apresenta a especificação e modelagem de um ambiente cooperativo de edição através de G-CPN, como meio de validação da ferramenta. O Capítulo 6 trata da segunda aplicação do modelo e propõe diretrizes de especificação e modelagem de entidades gerenciadas de Sistemas de Gerência de Redes ATM através de G-CPN. Finalmente, o Capítulo 7 apresenta as conclusões desta dissertação.

2. CONCEITOS BÁSICOS

Neste capítulo apresentamos o quadro de idéias que promoveu as bases para o desenvolvimento do modelo que iremos expor no decorrer desta dissertação. A compreensão dos conceitos dispostos neste capítulo é primordial para o entendimento dos demais capítulos.

Os modelos de redes de Petri tratados neste capítulo serão abordados de forma a dar noções intuitivas de sua estrutura e funcionamento, bem como de suas definições formais. Todavia, não se pretende que o texto sirva como guia completo de consulta sobre os referidos modelos. Para os leitores interessados rcomendamos [Murata, 1989] para uma visão geral sobre a teoria de redes de Petri, [Deng, 1992] para a compreensão de G-Nets e [Jensen, 1992] para um estudo profundo sobre redes Coloridas.

O Capítulo se encontra dividido em três seções. A Seção 2.1 apresenta os conceitos mais básicos a respeito do modelo clássico de redes de Petri. A Seção 2.2 é uma breve introdução a G-Nets e Sistemas de G-Nets. Finalmente, na Seção 2.3 são apresentados os conceitos fundamentais das Redes de Petri Coloridas - CPN.

2.1 Modelo Clássico de Redes de Petri

Redes de Petri são construtos matemáticos amplamente utilizados para "*a descrição e o estudo de sistemas de processamento de informação que se caracterizem como concorrentes, assíncronos, distribuídos, paralelos, não-determinísticos e/ou estocásticos*" [Murata, 1989].

A estrutura de uma rede de Petri é um grafo bipartido, direcionado e com pesos. O termo bipartido implica que os nós do grafo pertencem a duas classes disjuntas, denominadas **lugares** e **transições**, e que os **arcos** ligam sempre elementos de classes distintas. O termo direcionado é usado para denotar que os arcos têm origem e destino especificados. Finalmente, é preciso dizer que os arcos também podem ter pesos associados, o que justifica a qualificação "com pesos". Os pesos dos arcos são valores

inteiros que podem ser interpretados como replicações dos arcos, assim um arco com peso *w* denota a existência de *w* arcos entre os mesmos nós.

A descrição anterior, no entanto, estabelece apenas a natureza sintática de uma rede de Petri. Como o principal objetivo do construto é a formalização de aspectos dinâmicos de sistemas, é preciso que redes de Petri expressem algum estado do sistema. Para definir matematicamente o comportamento de uma rede de Petri novos conceitos precisam ser adicionados. Com esse objetivo, associam-se marcas à estrutura da rede de Petri que denotam o estado em que se encontra o sistema. As marcas são denominadas **fichas** e podem ser associadas apenas aos lugares da rede. Cada lugar pode ser associado a qualquer número inteiro de fichas. Desta forma, o conjunto total de fichas "espalhadas" nos lugares da rede denotam um estado, denominado **marcação.**

Com os conceitos acima expostos podemos enunciar a definição de uma **rede de Petri** como sendo a união de uma estrutura de rede de Petri (um grafo) a uma marcação inicial (um estado). As definições a seguir estabelecem as versões formais das descrições de redes de Petri como mostradas nestes parágrafos.

Definição 2.1 - *Estrutura de Rede de Petri*

Uma estrutura de rede de Petri é uma quádrupla <P, T, A, W> em que se verificam:

(i) $P = \{ p_1, p_2, ..., p_n \}$ é um conjunto finito de lugares

(ii) $T = \{ t_1, t_2, ... t_m \}$ é um conjunto finito de transições

$P \cap T = \varnothing$ e $T \cup P \neq \varnothing$

(iii) $A \subseteq P \times T \cup T \times P$ é um conjunto finito de arcos

(iv) $W: A \rightarrow \{ 1, 2, 3, ... \}$ é uma função de peso para os arcos

Definição 2.2 - *Marcação*

Seja N = <P, T, A, W> uma estrutura de Rede de Petri. Uma marcação para a estrutura N é uma função M, tal que:

(i) $M: P \rightarrow \{ 0, 1, 2, 3, ... \}$

Definição 2.3 - *Rede de Petri*

Uma rede de Petri é uma dupla $<S, M_0>$ em que se verificam:

(i) S = <P,T,A,W> é uma estrutura de rede de Petri

(ii) M_0: P → { 0, 1, 2, 3, ... } é a marcação inicial da rede de Petri

É importante citar aqui, que diversos autores preferem definir uma rede de Petri como uma única quíntupla que envolve todos os elementos da estrutura e a marcação inicial. Assim, nada impede que uma rede de Petri PN = <S, M_0> também seja denotada pela quíntupla <P, T, A, W, M_0>, onde S = <P, T, A, W>.

No lado esquerda da Figura 2.1 observa-se um exemplo de estrutura de rede de Petri. Aproveitamos para introduzir a notação gráfica utilizada para os diversos elementos da estrutura: lugares são representados por circunferências; transições por retângulos e os arcos por setas direcionadas. Em geral, escrevem-se os pesos dos arcos ao lado dos mesmos, entretanto, quando o peso for igual à unidade, este não será representado. Ao lado de cada elemento escreve-se sua identificação.

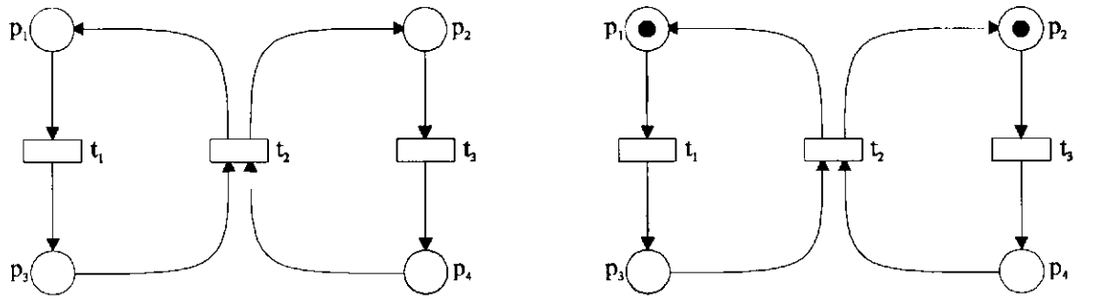

Figura 2.1 - Exemplo de Rede de Petri

No lado direito da Figura 2.1, apresenta-se uma rede de Petri que usa a estrutura comentada anteriormente. Os pequenos círculos pretos são as fichas presentes nos lugares na marcação inicial. Neste caso temos uma ficha no lugar p_1, uma ficha no lugar p_2 e nenhuma nos lugares p_3 e p_4.

Vejamos agora a definição formal subjacente à rede denotada graficamente à qual chamaremos de PN_1.

(i) PN_1 = <S, M_0> e S=<P, T, A, W>

(ii) P = { p_1, p_2, p_3, p_4 }

(iii) T = { t_1, t_2, t_3 }

(iv) A = { (p_1, t_1), (t_1, p_3), (p_3, t_2), (t_2, p_1), (t_2, p_2), (p_4, t_2), (p_2, t_3), (t_3, p_4) }

(v) $W(x,y) = a \Leftrightarrow$

$((x,y),a) \in \{ ((p_1, t_1), 1), ((t_1, p_3), 1), ((p_3, t_2), 1), ((t_2, p_1), 1), ((t_2, p_2), 1),$

$((p_4, t_2),1), ((p_2, t_3),1), ((t_3, p_4),1) \}$

(vi) $M_0(p) = x \Leftrightarrow (p,x) \in \{ (p_1, 1), (p_2, 1), (p_3, 0), (p_4, 0) \}$

Estas definições de redes de Petri não são suficientes para manifestar nenhuma característica sobre a dinâmica dos sistemas que desejamos modelar. Obviamente, é preciso descrever como e sob que circunstâncias a marcação de uma dada rede de Petri é alterada. Somente assim, poderemos encontrar critérios que nos indiquem como ligar elementos do mundo real a seus representantes em um modelo de rede de Petri. Precisamos, portanto, apresentar as definições da semântica associada a redes de Petri. É bastante comum denominar por regra de disparo às definições que estabelecem tal comportamento.

Para determinar como se comporta uma rede de Petri, precisamos dar significado aos elementos que a compõem. Da mesma forma que a dinâmica de um sistema é observada pelo avanço de seu estado no tempo, a dinâmica de uma rede de Petri será denotada pelo avanço de suas marcações. Portanto, uma marcação pode levar a outra. Resta-nos determinar como calcular uma nova marcação a partir de uma dada.

Os eventos capazes de alterar a marcação de uma rede de Petri estão intrinsecamente associados às transições da estrutura da rede. Cada transição determina um evento em potencial e as suas condições de ocorrência, bem como a especificação de como se dá a alteração do estado, caso ocorra. Costuma-se dizer que a transição ocorre.

É interessante introduzir aqui alguma terminologia, a fim de simplificar as descrições mais à frente. Cada transição pode estar ligada a diversos lugares por meio de arcos. Aos arcos que têm origem na própria transição denominamos arcos de saída. Se a transição é o destino de um arco, dizemos que se trata de um arco de entrada. Por extensão, os lugares ligados a uma dada transição através de arcos de saída são ditos lugares de saída. O mesmo vale para definir lugares de entrada. Finalmente, deve-se dizer que termos similares são usados com relação aos lugares. Assim, cada lugar tem seus próprios arcos de saída, arcos de entrada, transições de saída e transições de entrada.

Os arcos de entrada das transições são os elementos determinantes das condições de ocorrência. Cada arco de entrada exprime a necessidade da presença de uma ficha no seu

lugar de origem para que a transição ocorra. Arcos com peso "exigem" a presença de tantas fichas quanto seja o valor de seu peso. Assim, uma transição só pode ocorrer se há fichas suficientes nos seus lugares de entrada. Se uma transição reúne as condições necessárias para a sua ocorrência em uma certa marcação, dizemos que a transição está habilitada ou gatilhada.

Tomemos como exemplo, a rede da Figura 2.1. A transição t_1 encontra-se habilitada na marcação inicial, porque o lugar p_1 tem uma ficha, exatamente como é requerido pelo arco entre p_1 e t_1. Nessa mesma marcação a transição t_3 reúne as condições necessárias e portanto também está habilitada. A transição t_2, entretanto, não está habilitada porque nenhum de seus lugares de entrada tem o número de fichas exigido pelos respectivos arcos.

Os arcos de entrada em conjunto com os arcos de saída são os elementos determinantes das condições após a ocorrência do evento. Assim, a nova marcação após a ocorrência de uma transição será dada pela exclusão de fichas dos lugares de entrada, e pela inclusão de novas fichas nos lugares de saída da transição. A quantidade de fichas a retirar e colocar respectivamente nos lugares de entrada e de saída é determinada pelo peso do arco. Assim, por exemplo, se a transição t_1 ocorre na rede da Figura 2.1, a nova marcação será encontrada retirando-se uma ficha do lugar p_1 e acrescentando-se uma ficha ao lugar p_3. É importante lembrar que a transição é um evento atômico, portanto, não existe um estado intermediário entre a retirada e a colocação de fichas. Outro fator importante, é lembrar o fato de que transições habilitadas podem ou não ocorrer.

Vejamos agora as definições formais associadas aos conceitos apresentados. Nas definições seguintes:

(i) $PN = <S, M_0>$ é uma rede de Petri em que $S = < P, T, A, W >$

(ii) M, M_1 e M_2 são marcações da rede PN

Definição 2.4 - *Transição Habilitada*

Uma transição t é dita habilitada na marcação M se e somente se:

(i) $\forall p \in P: [(p,t) \in A \Rightarrow M(p) \geq W(p,t)]$

Definição 2.5 - *Ocorrência de uma Transição*

Uma transição habilitada $t \in T$ na marcação M_1 pode ocorrer. Se ocorre, então a marcação da rede muda de M_1 para M_2, como definida através da função por casos a seguir:

(i) $\forall p \in P \wedge t \in T$:

(ii)
$$\begin{aligned} M_2(p) &= M_1(p) && \Leftrightarrow && (p,t) \notin A \wedge (t,p) \notin A \\ &= M_1(p) + W(t,p) && \Leftrightarrow && (p,t) \notin A \wedge (t,p) \in A \\ &= M_1(p) - W(p,t) && \Leftrightarrow && (p,t) \in A \wedge (t,p) \notin A \\ &= M_1(p) + W(p,t) - W(t,p) && \Leftrightarrow && (p,t) \in A \wedge (t,p) \in A \end{aligned}$$

A Definição 2.5 estabelece a ocorrência de uma transição habilitada através de uma função definida por casos. Assim, para cada lugar a nova marcação é definida através de uma expressão diferente. Para cada lugar da rede, os casos possíveis são os seguintes: 1) não existe nenhum arco ligando o lugar à transição em ocorrência; 2) existe apenas um arco de entrada para o lugar a partir da transição; 3) existe apenas um arco de saída do lugar para a transição em ocorrência; e 4) existem dois arcos ligando o lugar à transição, um de entrada e outro de saída.

A título de exemplo observemos como é fácil verificar que na rede de Petri da Figura 2.1 a transição t_3 está habilitada porque seus lugares de entrada (apenas p_2) têm pelo menos o número de fichas indicados pelos pesos dos arcos (1 ficha). Portanto a expressão $\forall p \in P$: [$(p,t_3) \in A \Rightarrow M(p) \geq W(p,t_3)$] é verdadeira o que implica em que t_3 está habilitada. Suponha agora, que t_3 ocorra. Neste caso a nova marcação de cada lugar será dada por:

(i) $M_1(p_1) = M_0(p_1) = 1$, pois não existem os arcos (p_1,t_3) nem (t_3,p_1);

(ii) $M_1(p_2) = M_0(p_2) - W(p_2,t_3) = 0$, pois existe o arco (p_2,t_3) e não existe (t_3,p_2);

(iii) $M_1(p_3) = M_0(p_3) = 0$, pois não existem os arcos (p_3,t_3) nem (t_3,p_3);

(iv) $M_1(p_4) = M_0(p_4) + W(t_3,p_4) = 1$, pois existe o arco (t_3,p_4) e não existe (p_4,t_3);

A existência formal de definições sintáticas e semânticas para as redes de Petri garantem a possibilidade do desenvolvimento de métodos sistemáticos de análise e verificação de sistemas. Além disso, graças ao fato de que sua representação gráfica permite mapear de forma única todos os elementos em seus correspondentes formais, não há a necessidade de que o projetista de sistemas utilize o formalismo para descrever seus modelos.

Finalmente, a dinâmica do sistema pode ser percebida sem que haja a necessidade de recorrer às definições de ocorrência, visto que seu comportamento é bastante intuitivo, permitindo que simulações sejam feitas diretamente na representação gráfica da rede.

As redes de Petri, como apresentadas aqui são conhecidas como redes de Petri Lugar/Transição, e sua definição é semelhante à dada em [Murata, 1989]. Entretanto, há autores que definem as redes Lugar/Transição de diferentes formas [Jensen, 1992]. Às vezes são consideradas capacidades para os lugares, que determinam o número máximo de fichas que podem suportar. Outras vezes, as definições formais diferem na abordagem utilizada. Contudo, é fato bastante conhecido que as várias definições se equivalem, e portanto são capazes de especificar as mesmas classes de sistemas.

Nas seções a seguir apresentaremos modelos de redes de Petri que fazem parte da classe denominada Redes de Petri de Alto-Nível. Modelos dessa classe se caracterizam pelo fato de que as fichas não são elementos de natureza unicamente binária. As fichas de redes de Petri de Alto Nível transportam informações. Desta forma, inscrições associadas à estrutura da rede podem identificar a natureza das fichas presentes nos lugares e proporcionar mecanismos mais complexos de disparo. Em decorrência disso, redes de Petri de Alto Nível podem ter maior poder de expressão.

2.2 Sistemas de G-Nets

Nesta seção introduzimos os conceitos básicos relacionados com G-Nets e Sistemas de G-Nets. Sistemas de G-Nets, originalmente propostos em [Deng, 1992; Deng, 1993] são uma abordagem para a especificação, concepção e modelagem formal de sistemas distribuídos de informação. Um Sistema de G-Nets é constituído por um conjunto de G-Nets, que podem ser vistas como um módulo ou objeto que encapsula operações denominadas métodos. Os métodos provêem mecanismos bem-definidos para esconder detalhes de suas implementações, bem como elementos de interface que permitem a sua utilização. Assim, um módulo pode utilizar os serviços providos por outros módulos do sistema, através dos seus elementos de interface.

Cada G-Net é composta por dois elementos: um lugar especial denominado Lugar Especial de Chaveamento (GSP - *Generic Switching Place*) e uma Estrutura Interna (IS - *Internal Structure*). O GSP é um mecanismo de abstração do módulo G-Net. Nele estão especificados os elementos de interface para os métodos. A IS é uma rede de Petri

do tipo Predicado Transição (PrT) [Genrich, 1987; Genrich, 1989; Genrich, 1991] modificada que representa o detalhamento da realização interna dos métodos da G-Net.

A notação gráfica usada para representar G-Nets é mostrada na Figura 2.2. Os diversos elementos que a compõem são descritos a seguir:

1. O GSP é indicado pela elipse no canto superior à esquerda. Nela define-se o nome da G-Net, que será usado por outras G-Nets em caso de invocações de serviços.

2. O retângulo de bordas arredondadas no canto superior à direita é usado para declarar e identificar os métodos e atributos da rede. As inscrições seguem o seguinte esquema:

    - *<nome_do_atributo>* define uma identificação única para um atributo da rede, cujo domínio é determinado por *<tipo>*, que é restrito ao conjunto dos inteiros não negativos;

    - *<nome_do_método>* define um identificador único para um método da G-Net. O método é descrito através do elemento *<descrição>*. O elemento <p1:descrição, ..., pn:descrição> é uma lista de argumentos para o método e *<sp>* dá o nome do lugar inicial do método.

3. A IS é representada dentro do retângulo maior de bordas arredondadas, que determina o seu limite. Dentro da IS encontram-se diversos elementos de natureza diferentes que são descritos a seguir:

    - Circunferências representam lugares normais

    - Elipses representam lugares especiais de chaveamento para instanciação (*ISP - Instantiating Switching Place*). Cada *isp* é usado para prover comunicação inter-módulos. A inscrição isp(G',mi) indica uma invocação do método *mi* da G-Net G'.

    - Retângulos representam transições aos quais pode-se associar uma inscrição. As inscrições podem ser atribuições ou restrições de disparo.

    - Circunferências com borda dupla representam lugares-alvo.

- Arcos direcionados interligam transições e lugares. Cada arco pode ser associado a uma expressão.

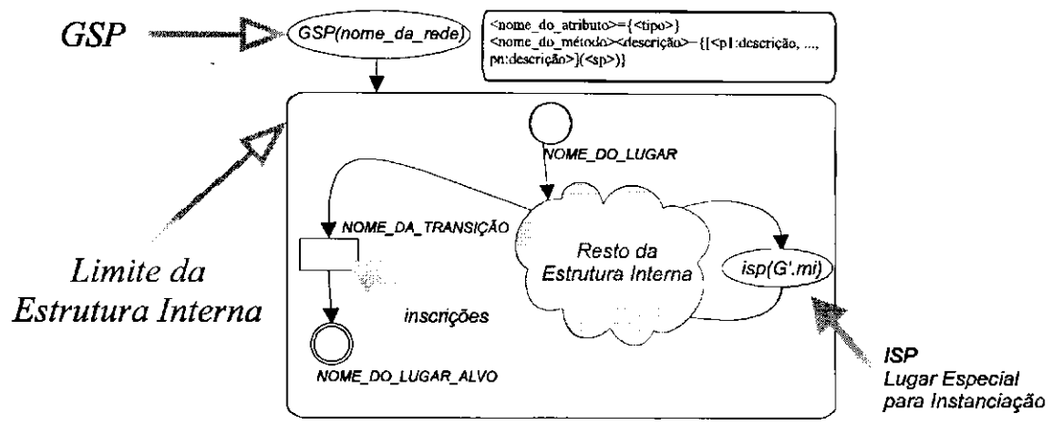

Figura 2.2 - Notação Gráfica usada para G-Nets

Como objetivo de explicar como o comportamento associado a Sistemas de G-Nets, apresentamos um exemplo simples. O Sistema da Figura 2.3 é composto por duas G-Nets identificadas por G1 e G2. Observando a G-Net G1 podemos perceber que apenas um método é provido, denominado *ma*. G1 tem quatro lugares especificados como P1, P2, P3 e P4. O lugar P4, por ter duas bordas é denominado o lugar-alvo da G-Net. O lugar P3 representa um *isp* que invoca o método *mr* de G2. Os demais lugares da estrutura interna de G1 são lugares normais. A transição T1 não apresenta nenhuma inscrição, entretanto T2 tem uma inscrição do tipo atribuição. A G-Net G2 tem apenas um método denominado *mr*. Na sua estrutura interna observam-se os lugares normais P1, P2 e P3, bem como o lugar alvo P4. As inscrições das transições T1 e T2 são restrições de ocorrência, enquanto que em T3 e T4 são atribuições.

O funcionamento do sistema apresentado na Figura 2.3 é bastante simples. Quando o método *ma* é invocado, recebendo um inteiro *a*, uma ficha com esse valor é depositada no lugar P1. A transição T1 pode ocorrer sem nenhuma restrição, gerando fichas de mesmo valor nos lugares P2 e P3. A ficha depositada no lugar P3 dá início à invocação do método *mr* da G-Net G2. Quando o fim da execução do método *mr* for detectado pela presença de uma ficha no seu lugar alvo, essa ficha será devolvida à G-Net invocadora para o lugar P3. A nova ficha no lugar P3, juntamente com a ficha existente no lugar P2 habilitam a transição T2. Quando T2 ocorrer, uma ficha de valor *i*, dado pela

expressão dentro de T2, será depositada no lugar alvo P4, caracterizando o fim do método.

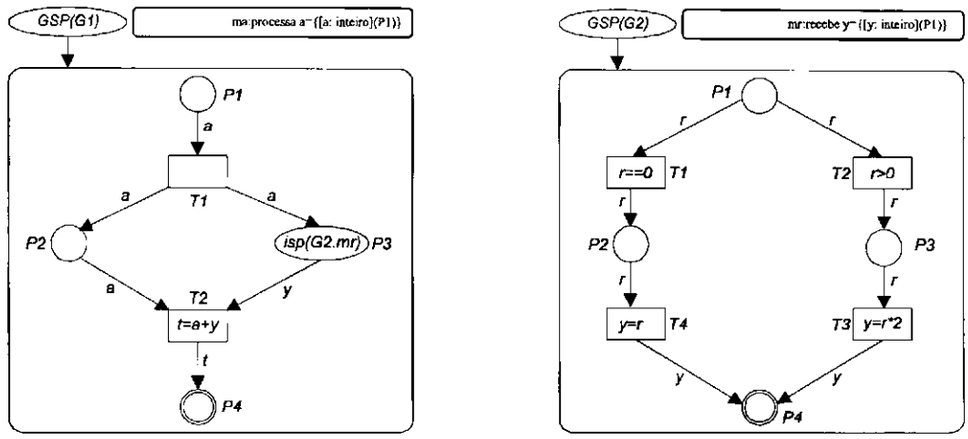

Figura 2.3 - Exemplo de Sistema de G-Nets

No que segue apresentamos algumas das definições formais relacionadas a Sistemas de G-Nets, como descritas em [Perkusich, 1994b]:

Definição 2.6 - *Sistema de G-Nets*

Um sistema de G-Nets (GNS) é uma tripla GNS = <TS,GS,AS>, onde:

(i) TS é uma coleção de fichas geradas dinamicamente durante a execução do sistema;

(ii) GS é um conjunto de G-Nets; e

(iii) AS é um conjunto descentralizado de agentes computacionais concorrentes, executando um sistema de G-Nets.

Definição 2.7 - *G-Net*

Uma G-Net é uma dupla G = <GSP, IS>, onde

(i) GSP é um *Lugar Genérico de Chaveamento* (GSP) que provê uma abstração para a G-Net; e

(ii) IS é uma rede PrT modificada denominada *Estrutura Interna*.

Definição 2.8 - *GSP*

Um lugar genérico de chaveamento GSP é uma estrutura GSP = <NID, MS, AS>, onde:

(i) NID é uma identificação única da G-Net G;

(ii) MS é um conjunto de métodos; e

$\forall mtd_i \in MS$, $mtd_i = (m\_nome_i, m\_argumentos_i, m\_iniciador_i)$, onde

m_nome_i é um nome para o método mtd_i;

$m\_argumentos_i$ é uma tupla de variáveis especificando a ficha de entrada para o método mtd_i; e

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

(iii) AS é um conjunto de atributos, e

$\forall as_i \in AS$, $as_i \in \{0, 1, 2, 3, ...\}$

Definição 2.9 - *IS*

A estrutura interna IS de uma G-Net G é uma estrutura de rede, isto é, um grafo direcionado bipartido definido por $<\Sigma, P, T, I, O>$, onde:

(i) Σ é uma estrutura consistindo de algum tipo de predicado, juntamente com o conjunto de relações e operações definidas para estes predicados;

(ii) P é um conjunto finito e não vazio de lugares, denotados por círculos;

(iii) T é um conjunto de transições, denotadas por retângulos;

(iv) $I: T \rightarrow P^{\infty}$ é denominada função de entrada, definindo inscrições para transições e arcos de entrada;

(v) $O: T \rightarrow P^{\infty}$ é denominada função de saída, definindo inscrições para transições e arcos de saída;

As definições apresentadas caracterizam formalmente a sintaxe de G-Nets e Sistemas de G-Nets. Entretanto, outras definições formalizam o comportamento destas redes, através da definição precisa do que seja uma ficha, bem como, os eventos relacionados com a comunicação inter-módulos. Por se tratar apenas de uma introdução aos conceitos relacionados a G-Nets, não apresentaremos as definições sobre comportamento. O leitor interessado pode referir-se a [Deng, 1992; Deng, 1993; Perkusich, 1994b].

2.3 Redes de Petri Coloridas

Nesta seção expomos os conceitos e as definições formais subjacentes ao modelo de Redes de Petri Coloridas (CPN - *Coloured Petri Nets*). O modelo CPN é uma classe de redes de Petri de Alto Nível aplicável à especificação, projeto, simulação, validação e

implementação de sistemas complexos de software [Jensen, 1992]. Sua ampla utilização deve-se essencialmente à clareza do sólido embasamento matemático, ao enorme leque de métodos formais de análise, verificação e validação e à existência de um grande número de ferramentas de software já plenamente desenvolvidas. Além disso, redes de Petri Coloridas vêm sendo aplicadas aos mais diversos domínios com bastante sucesso [Rasmunssen, 1995].

Uma rede de Petri Colorida (CP-Net) é composta essencialmente por uma estrutura, um conjunto de inscrições e um conjunto de declarações. A estrutura de uma CP-Net é semelhante à estrutura das redes Lugar/Transição. Portanto, é também um grafo dirigido e bipartido. Entretanto, ao invés de pesos inteiros, os arcos são associados a inscrições que determinam dinamicamente quantas e quais fichas devem ser removidas ou adicionadas aos lugares associados, na ocorrência de uma transição. Inscrições também podem ser associadas a transições, e permitem restringir ocorrências de eventos a determinadas condições. O estado inicial de uma CP-Net também é determinado por inscrições associadas aos lugares. Cada inscrição é, em geral, uma expressão construída a partir de constantes, variáveis e operadores previamente definidos. O conjunto de declarações de uma CP-Net serve primordialmente para declarar a natureza dos elementos citados nas diversas inscrições, e por conseguinte, dos objetos manipulados pelo modelo.

As inscrições e declarações de uma CP-Net podem, *a priori*, ser escritas em praticamente qualquer linguagem que tenha sintaxe e semântica bem definidas. Pode-se, por exemplo, usar a notação matemática padrão, embora isso seja de certa forma inconveniente, devido a problemas gerados pelo uso de símbolos. Em geral, CP-Nets vêm sendo utilizadas em associação com uma linguagem denominada CPN-ML, derivada da linguagem funcional *Standard ML*, cuja sintaxe é bastante semelhante à usada por linguagens de programação convencionais.

A Figura 2.4 mostra uma CP-Net simples que modela um sistema produtor consumidor. Observam-se com facilidade os três elementos componentes da rede: a estrutura determinada pelo grafo; as inscrições espalhadas por toda a estrutura; e o nó de declarações no retângulo com bordas tracejadas.

É possível, também, observar-se diversos tipos de inscrições. Ao lado de cada lugar da rede uma inscrição com os caracteres em itálico associa um tipo de dado ao lugar. Observe, por exemplo, o lugar denominado *Buffer* cujo conjunto de cores associado é *MESSAGE*. Marcações iniciais dos lugares são determinadas por inscrições sublinhadas, veja por exemplo os lugares P1 e P3. Lugares não associados a inscrições sublinhadas têm marcação inicial nula. Embora não sejam obrigatórias, as inscrições que denotam os nomes dos elementos são posicionadas dentro dos elementos gráficos. Finalmente, as inscrições dos arcos denotam as fichas que devem ser movimentadas quando da ocorrência de transições.

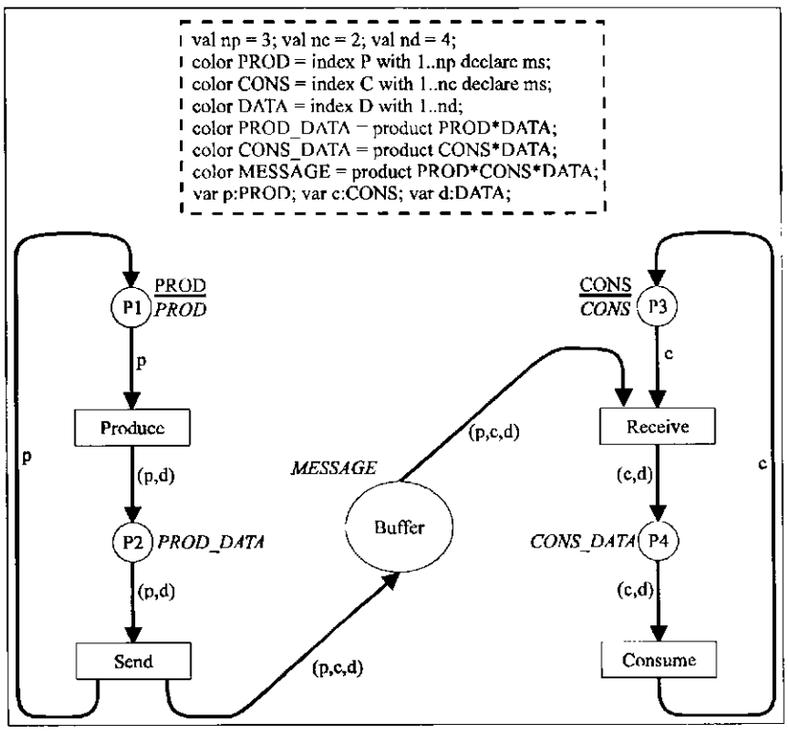

Figura 2.4 - Exemplo de Rede de Petri Colorida

Embora não estejam presentes no exemplo utilizado, existe outro tipo de inscrição em CP-Nets, denominada guarda. As guardas são associadas às transições, e têm a função de restringir as condições de sua ocorrência. Devido à sua função as guardas devem ser expressões de natureza booleana, isto é, ao serem avaliadas devem resultar exclusivamente em valores do conjunto verdade. As guardas são representadas graficamente por inscrições entre colchetes ao lado das transições.

Por se tratar de uma rede de Petri de Alto Nível, as fichas presentes em CP-Nets sempre "transportam" algum valor. Cada valor deve pertencer ao domínio de algum tipo de dado bem definido nas declarações. Por razões históricas, em CPN usa-se a expressão *conjunto de cores* (*colour set*) em substituição a tipo de dados e, por conseqüência, cada valor é denominado de cor (*colour*). Desta forma, cada lugar na estrutura interna é associado a um conjunto de cores, que indica o tipo de fichas que o lugar pode conter. A linguagem CPN-ML dispõe de mecanismos que permitem a definição de conjuntos de cores relativamente complexos, além de oferecer diversos conjuntos previamente definidos.

Seis conjuntos de cores são declarados na rede da Figura 2.4. O conjunto PROD, formado por três elementos (p1, p2 e p3), representa três processos produtores. O conjunto CONS, definido de forma semelhante representa dois processos consumidores (c1 e c2). O conjunto DATA define quatro elementos de dados que os processos produtores podem enviar aos consumidores (d1, d2, d3 e d4). Os conjuntos PROD_DATA e CONS_DATA são definidos através de produtos cartesianos entre dois conjuntos. Uma elemento de PROD_DATA, digamos (p,d), representa que o produtor p gerou o dado d e está pronto para ser enviado. De forma análoga, um elemento de CONS_DATA, digamos (c,d), representa que o dado d está disponível para ser consumido por c. Finalmente, o conjunto de cores MESSAGE caracteriza uma mensagem enviada pelo produtor para algum consumidor. Elementos do conjunto MESSAGE são triplas que identificam a origem da mensagem, o destinatário e o dado propriamente dito.

Um conceito fundamental para o estabelecimento das CP-Nets é a idéia de multi-conjuntos (*multi-sets*). Multi-conjuntos são construtos semelhantes a conjuntos, exceto pelo fato de que podem conter múltiplas aparições de um mesmo elemento. Todo multi-conjunto é definido sobre um conjunto a partir do qual os elementos são tomados. A título de exemplo, os multi-conjuntos a seguir foram criados sobre o conjunto dos números naturais: {0,2,3,7}, {0,1,1,3}, {0,1,1,2,3,3,7}, {}.

Uma marcação para uma CP-Net é uma distribuição de fichas nos seus lugares. A marcação de cada lugar é definida como um multi-conjunto sobre o conjunto de cores associado ao lugar. Na rede da Figura 2.4 os lugares P1 e P2 têm inscrições de marcação inicial iguais a "PROD" e "CONS", respectivamente. Esses termos são iguais aos nomes

dos conjuntos de cores associados, o que implica (em CPN-ML) que os lugares serão inicializados com o multi-conjunto contendo exatamente uma aparição de cada elemento do referido conjunto de cores.

É comum usar a expressão *variáveis da transição* para nos referirmos ao conjunto de variáveis presentes nas inscrições dos arcos e na guarda da referida transição. Outro conceito de extrema importância em redes de Petri Coloridas é o de ligação (*binding*) de uma transição. Uma ligação de uma transição pode ser vista como a substituição de cada variável da transição por uma cor (valor). É requerido, entretanto, que as cores pertençam aos conjuntos de cores apropriados e que impliquem na avaliação da guarda como verdadeira.

Estabelecidos os conceitos de marcação e ligação podemos definir como se comportam as redes de Petri Coloridas. Em cada marcação, a ocorrência de uma transição sob uma determinada ligação é dita habilitada se todos os seus lugares de entrada tiverem fichas suficientes para satisfazer às expressões dos arcos. Cada expressão deve ser devidamente avaliada segundo as substituições determinadas pela ligação, a fim de determinar quantas e quais fichas são requeridas nos lugares de entrada. Caso a transição ocorra, então são retiradas fichas dos lugares de entrada e depositadas novas fichas nos lugares de saída. A quantidade de fichas é determinada também pela avaliação das expressões dos arcos segundo as substituições implicadas pela ligação.

Vejamos um exemplo. Considere a transição **Produce** e os lugares P1 e P2, na Figura 2.4. A marcação inicial do lugar P1 é o multi-conjunto {p1, p2, p3} e a do lugar P2 é o multi-conjunto vazio. Uma ligação, como já foi dito, é uma série de substituições válidas para todas as variáveis da transição, neste caso **p** e **d**. Um exemplo de ligação que habilita a transição **produce** é {**p**=p3, **d**=d4}. Se a transição ocorre, deve-se retirar uma ficha de cor p3 do lugar P1 e adicionar uma ficha de cor (p3,d4) ao lugar P2. Portanto, a nova marcação do lugar P1 é o multi-conjunto {p1, p2} e a do lugar P2 é o multi-conjunto {(p3,d4)}. Observe, que nesta nova marcação, a transição **produce** não está mais habilitada para a ligação {**p**=p3, **d**=d4} usada anteriormente, embora, outras ligações possam habilitá-la. Outra observação interessante é que a nova marcação da rede habilita, sob as ligações apropriadas, duas transições: **produce** e **send**.

No que segue, apresentamos as definições formais dos diversos conceitos relacionados a redes de Petri Coloridas.

Definição 2.10 - *Multi-Conjunto*

Seja C um conjunto não vazio. Um multi-conjunto *m* sobre o conjunto C é uma função total que mapeia elementos de C no conjunto dos inteiros não-negativos:

(i) $m: C \rightarrow \{0, 1, 2, ...\}$

Logo, para cada elemento *c* pertencente ao conjunto C existe um único valor *m*(c), que é denominado o **coeficiente** ou o **número de aparições** de *c* em *m*. Quando o coeficiente de um elemento $c \in C$ é diferente de zero dizemos que *c* pertence ao multi-conjunto *m*. A pertinência de um elemento *c* ao multi-conjunto *m* é denotada por $c \in m$. Para denotar o conjunto de todos os multi-conjuntos que podem ser formados sobre C usaremos a expressão C_{MS}.

Um multi-conjunto *m* também pode ser representado através da soma formal:

$$m \equiv \sum_{c \in C} m(c)`c$$

Perceba que na representação através de soma formal usamos um operador específico entre o coeficiente e o elemento (`). Além disso, por convenção omitiremos os termos cujos coeficientes sejam iguais a zero. Por fim, também podemos representar multi-conjuntos pela enumeração de cada aparição dos elementos do conjunto gerador, de forma análoga a conjuntos.

Vejamos um exemplo: considere o conjunto C = {x, y, z}. Então, podemos definir os multi-conjuntos m_1, m_2 e m_3 pertencentes a C_{MS} e representá-los por:

(i) $m_1(x) = 1$; $m_1(y) = 2$; $m_1(z) = 1$

(ii) $m_2(x) = 2$; $m_2(y) = 0$; $m_2(z) = 1$

(iii) $m_3(x) = 1$; $m_3(y) = 2$; $m_3(z) = 3$

ou por:

(i) $m_1 = 1`x + 2`y + 1`z = \{x, y, y, z\}$

(ii) $m_2 = 2`x + 1`z = \{x, x, z\}$

(iii) $m_3 = 1`x + 2`y + 3`z = \{x, y, y, z, z, z\}$

É possível definir diversas operações sobre multi-conjuntos. Em [Jensen, 1992] encontram-se as definições formais para as operações de adição, produto por um escalar, comparação entre multi-conjuntos e cardinalidade. Além disso, são mostradas diversas propriedades das operações assim definidas.

Mostramos a seguir a definição formal associada à sintaxe de CP-Nets.

Definição 2.11 - *Rede de Petri Colorida (não-hierárquica)*

Uma rede de Petri Colorida (CP-Net) é uma tupla CPN = (Σ, P, T, A, N, C, G, E, I) que satisfaz aos seguintes requisitos:

(i) Σ é um conjunto finito de tipos não vazios, denominados **conjuntos de cores** (*colour sets*)

(ii) P é um conjunto finito de **lugares**

(iii) T é um conjunto finito de **transições**

(iv) A é um conjunto finito de **arcos**

$P \cap T = P \cap A = T \cap A = \varnothing$

(v) N é uma função **nó**

$N: A \rightarrow P \times T \cup T \times P$

(vi) C é uma função de **cores**

$C: P \rightarrow \Sigma$

(vii) G é uma função de **guardas**

$G: T \rightarrow$ EXPRESSIONS tal que:

$\forall t \in T: [Type(G(t))=BOOLEAN \wedge Type(Var(G(t))) \subseteq \Sigma]$

(viii) E é uma função de **expressões** de arcos

$E: A \rightarrow$ *EXPRESSÕES*, tal que

$\forall a \in A: [Type(E(a))=C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$

onde p(a) é o lugar de N(a).

(ix) I é uma função de **inicialização**

$\forall p \in P: [Type(I(p))=C(p)_{MS} \wedge I(p)$ é fechada]

A Definição 2.11 é dependente da existência de uma linguagem que determine as expressões necessárias. Por outro lado, essa dependência permite que quase qualquer linguagem seja utilizada, desde que satisfaça a certas restrições. É necessário, por exemplo, que a linguagem tenha sintaxe e semântica bem definidas a ponto de que se

possa estabelecer formalmente as funções Type() e Var() usadas na definição. A semântica associada a esses elementos é definida informalmente em [Jensen, 1992].

Apresentamos agora as definições formais associada a uma ficha em CP-Nets. Para que identificar unicamente uma ficha, é preciso saber o lugar no qual está e o valor que transporta. Essa é a idéia por trás da formalização.

Definição 2.12 - *Elemento de Ficha e Marcação*

Um elemento de ficha é um par (p,c) onde $p \in P$ e $c \in C(p)$. O conjunto de todos os elementos de ficha é denotado por TE. Uma marcação é um multi-conjunto sobre TE.

Definição 2.13 - *Ligação (binding)*

Uma ligação de uma transição t é uma função b definida sobre Var(t), tal que:

(i) $\forall v \in Var(t)$: $b(v) \in Type(v)$

(ii) G(t)<b>

(iii) B(t) denotará o conjunto de todas as ligações para a transição t.

A definição formal de ligação é bastante intuitiva, e reflete exatamente o que foi explicado na introdução informal. A notação usada na linha (ii) denota a avaliação da expressão da guarda G(t) sob a ligação b.

Definição 2.14 - *Elemento de Ligação (binding element) e Passo (step)*

Um elemento de ligação é um par (t,b) onde $t \in T$ e $b \in B(t)$. O conjunto de todos os elementos de ligação é denotado por BE. Um passo é um multi-conjunto sobre BE.

O conceito de passo não havia sido comentado anteriormente. Um passo é intuitivamente um multi-conjunto de possibilidades de ocorrências de transições. Desta forma, redes CP-Nets apresentam a vantagem de capturar a essência da idéia de concorrência. Em outros modelos, o termo concorrência significa apenas que os eventos podem ocorrer uns após os outros em qualquer ordem, mas não simultaneamente. Com a introdução do conceito de passo, o significado de concorrência passa a ser o mais intuitivo, representando que diversas ações podem ocorrer em um evento único.

Definição 2.15 - *Passo habilitado (enabled step)*

Um passo Y é dito habilitado na marcação M se e somente se:

$$\forall p \in P: \sum_{(t,b)\in Y} E(p,t)<b> \leq M(p)$$

Definição 2.16 - *Ocorrência de um Passo habilitado*

Quando um passo Y está habilitado na marcação M_1, então ele pode ocorrer. Se ocorre a rede muda da marcacão M_1 para a marcação M_2, definida por:

$$\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t)<b>) + \sum_{(t,b)\in Y} E(t,p)<b>$$

Como exemplo, voltemos à rede da Figura 2.4. A marcação inicial da rede pode ser formalizada por:

(i) M_0 = 1`<P1,**p1**> + 1`<P1,**p2**> + 1`<P1,**p3**> + 1`<P3,**c1**> + 1`<P3,**c2**> ou

(ii) M_0 = {<P1,**p1**>, <P1,**p2**>, <P1,**p3**>, <P3,**c1**>, <P3,**c2**>}

A ocorrência da transição **produce** sob a ligação {**p**=p3, **d**=d4} na marcação M_0 poderia ser vista como a ocorrência do passo Y_0 definido por:

(i) Y_0 = 1`<**produce**, {**p**=p3, **d**=d4}>

Com a sua ocorrência a nova marcação da rede será dada por M_1:

(i) M_1 = 1`<P1,**p1**> + 1`<P1,**p2**> + 1`<P2,**p3**> + 1`<P3,**c1**> + 1`<P3,**c2**> ou

(ii) M_1 = {<P1,**p1**>, <P1,**p2**>, <P2,(**p3,d4**)>, <P3,**c1**>, <P3,**c2**>}

Nesse caso um passo habilitado seria Y_1:

Y_1 = 1`<**produce**, {**p**=p1, **d**=d2}> + 1`<**send**, {**p**=p3, d=**d4**, c=**c1**}>

Diversas outras definições formais são necessárias para estabelecer fortemente as redes de Petri Coloridas. Entretanto, devido ao caráter introdutório que este capítulo tem, elas não serão apresentadas aqui. Para o leitor interessado, a formalização completa e as explicações apropriadas encontram-se em [Jensen, 1992].

3. MÓDULOS G-CPN

O objetivo principal deste capítulo é introduzir os conceitos e as definições relativas à estrutura G-CPN. A metodologia adotada é a de apresentar inicialmente uma descrição informal de Sistemas G-CPN acompanhada de um exemplo simples e em seguida uma seqüência de definições formais que evoluem de forma incremental a partir da definição consolidada de redes CPN [Jensen, 1992] em direção à estrutura proposta para Módulos G-CPN.

À medida em que avançamos nas definições, incorporamos conceitos da estrutura G-Net e alguns novos a fim de simplificar a definição formal. É importante destacar que as definições foram elaboradas objetivando abstrair ao máximo a natureza do modelo, sem deixar-nos influenciar por possíveis questões de implementação de um *Sistema G-CPN*.

G-CPN é uma estrutura de redes de Petri baseada em objetos para a especificação e modelagem de sistemas distribuídos de software. O Modelo G-CPN foi elaborado partindo dos princípios que originaram a estrutura G-Net, introduzida em [Deng, 1992; Deng 1993] e nas redes de Petri de alto nível denominadas CPN [Jensen, 1987; Jensen, 1992]. Dizemos que *G-CPN* é baseada em objetos, porque apenas alguns dos princípios da orientação a objetos (OO) foram adotados a fim de permitir estruturar os sistemas modelados.

O modelo G-CPN permite o desenvolvimento incremental e não-monotônico da especificação de um sistema de software, favorecendo as condições de manutenção e de reusabilidade. Isto ocorre pela utilização dos conceitos de módulos fracamente acoplados, interfaces bem-definidas e informação escondida, que resultam no fato de que um módulo só pode ter acesso a informações e métodos de outros módulos no sistema através de um mecanismo de abstração. Dado que é altamente improvável que as primeiras tentativas de estabelecer soluções para sistemas complexos resultem imediatamente corretas [Booch, 1991], a possibilidade de efetuar alterações, de permitir o "ir e vir" na especificação, aproveitando ao máximo sub-partes consideradas corretas, é essencial em ferramentas de análise e síntese.

A formalização de G-CPN foi construída de forma a preservar aspectos gerais da natureza sintática das redes CPN [Jensen, 1992], por exemplo, a mesma linguagem funcional é utilizada para as inscrições das redes: CPN-ML [DesignCPN]. Isto significa, que a certo nível, a modelagem em G-CPN se assemelha à modelagem em CPN. Muito embora, o comportamento das redes G-CPN seja diferente, implicando em definições semânticas diferentes. Do ponto de vista intuitivo, o comportamento associado a um módulo G-CPN é perfeitamente dedutível para desenvolvedores habituados às redes CPN.

3.1 Introdução Informal aos Sistemas G-CPN

Sistemas G-CPN são conjuntos de módulos G-CPN concorrentes, cooperantes e fracamente acoplados cujo objetivo comum é a modelagem/especificação formal e executável de sistemas distribuídos de software.

É importante estabelecer a diferença entre um *Sistema G-CPN* e um *Módulo G-CPN* ou simplesmente uma *G-CPN*. Estruturalmente um *Sistema G-CPN* é a integração de um conjunto de módulos e um ambiente de interação (ver Figura 3.1). Cada *Módulo G-CPN*, ou cada *G-CPN* define um objeto instanciável do sistema. O *ambiente de interação* é o elemento responsável pela semântica associada ao sistema. Em se tratando da modelagem de sistemas distribuídos, o ambiente pode ser visto como a abstração da rede de comunicação, sendo o responsável pela semântica de transferência de mensagens entre os módulos.

Quando um módulo requisita um serviço de outro módulo, o ambiente de interação é responsável pelo transporte das mensagens de requisição e de resposta. Desta forma, do ponto de vista do módulo, a transferência pode ser abstraída, bastando indicar qual o serviço desejado e o identificador do módulo que o oferece. Uma outra característica relevante de módulos G-CPN, é o fato de poderem efetuar chamadas recursivas.

O modelo de interação entre módulos G-CPN é tipicamente cliente-servidor [Soares, 1995; Tanembaum, 1991; Tanembaum, 1995; Simon, 1996]. Quando a requisição da execução de um serviço é atendida pelo ambiente, dizemos que uma *invocação* ocorreu. A partir daí, o módulo apenas espera que o ambiente faça efetivamente a ativação do serviço remoto, e passa a esperar a chegada de resposta. O ambiente, de posse das informações necessárias, efetua uma *ativação* do método invocado no módulo servidor

através da passagem dos dados necessários ao método. O método servidor, agora ativo, atende ao pedido. A estrutura interna responsável por atender o pedido, é sintaticamente uma rede CPN. Quando for obtido algum resultado, o ambiente detecta a disponibilidade desses resultados (fichas em algum lugar pré-determinado) e os retoma a fim de retransmití-los ao chamador, este evento é denominado *terminação*. Após a ocorrência da *terminação* no módulo servidor, o ambiente detecta o módulo cliente correspondente e devolve corretamente os dados, forçando uma ocorrência do último evento associado à interação, denominado *retorno*.

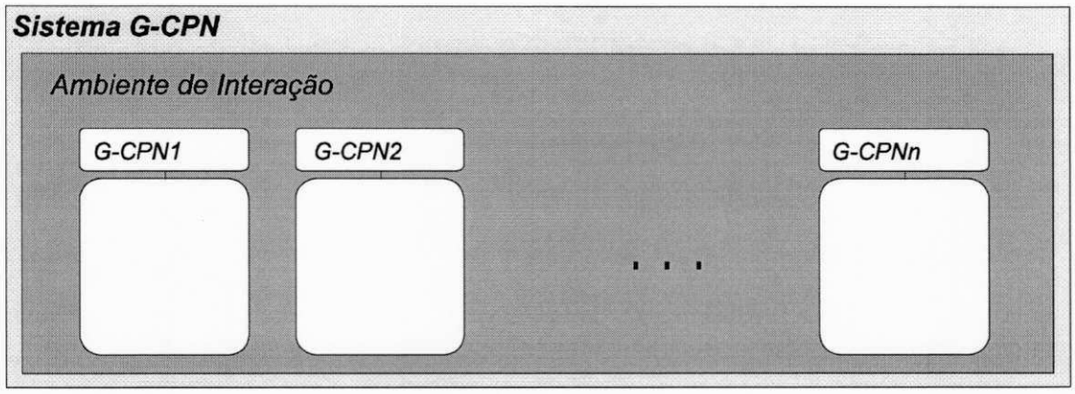

Figura 3.1 - Sistema G-CPN

Os quatro eventos associados à interação entre dois módulos, mostrados na Figura 3.2, são: *invocação*, *ativação*, *terminação* e *retorno*. É importante ressaltar que todos os eventos ocorrem a nível de ambiente ou Sistema G-CPN, entretanto apenas dois deles ocorrem em cada um dos módulos envolvidos. Uma *invocação* e um *retorno* ocorrem no módulo cliente, enquanto que uma *ativação* e uma *terminação* ocorrem no módulo servidor.

Cada módulo G-CPN é estruturalmente representado por duas subestruturas: a primeira representa a abstração do módulo e é denominada GSP (*Generic Switching Place*) ou simplesmente interface; a segunda é a estrutura interna do módulo - IS (*Internal Structure*). A interface determina a visão do módulo para o resto do sistema. Nela estão declarados os atributos, e métodos encapsulados. Para cada método declarado na interface, existem dois lugares diferenciados na IS: o primeiro determina onde serão colocadas as fichas de *ativação* do método; e o segundo especifica qual o lugar de *terminação*. A estrutura interna é (sintaticamente) uma rede de Petri Colorida.

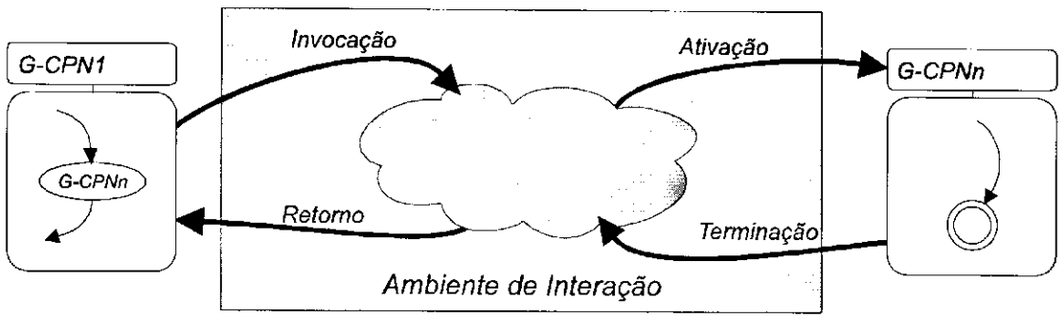

Figura 3.2 - Eventos ocorridos durante a Interação

Para representar invocações de métodos remotos na IS, utilizamos um lugar especial denominado ISP (*Instantiating Switching Place*). Seu funcionamento é intuitivo: quando uma ficha é depositada, uma *invocação* pode ocorrer, implicando no desaparecimento da ficha; em seguida, dependendo do funcionamento do método remoto, deverá ocorrer um *retorno*, e uma outra ficha com novos dados será depositada no ISP, permitindo a habilitação das transições de saída.

Os lugares normais, bem como atributos e lugares que indicam ativação de métodos são representados graficamente por círculos. Os lugares de terminação são representados por círculos de borda dupla. ISPs são denotados por elipses e uma inscrição interna indicando o método e a G-CPN a invocar. Os demais elementos (transições e inscrições) seguem a mesma notação usada em CPN. Uma exceção deve ser notada: dois conjuntos de cores são associados a cada ISP, um para fichas de invocação e outro para fichas de terminação. Veja a Figura 3.3.

Um único módulo G-CPN pode atender a qualquer número de pedidos de serviços concorrentemente (ver definição formal). Isso é possível, porque ao ocorrer uma ativação, uma nova instância do módulo, criada automaticamente, será responsabilizada para atender o pedido. Cada instância é tratada pelo que denominamos de *contexto*. Um contexto pode ser visto como um novo objeto cuja estrutura funcional é uma cópia fiel da estrutura interna do módulo invocado. A única exceção é quanto aos lugares que representam atributos, que não serão copiados, mas sim compartilhados. O contexto é automaticamente destruído quando não restarem mais fichas relativas àquela invocação na rede.

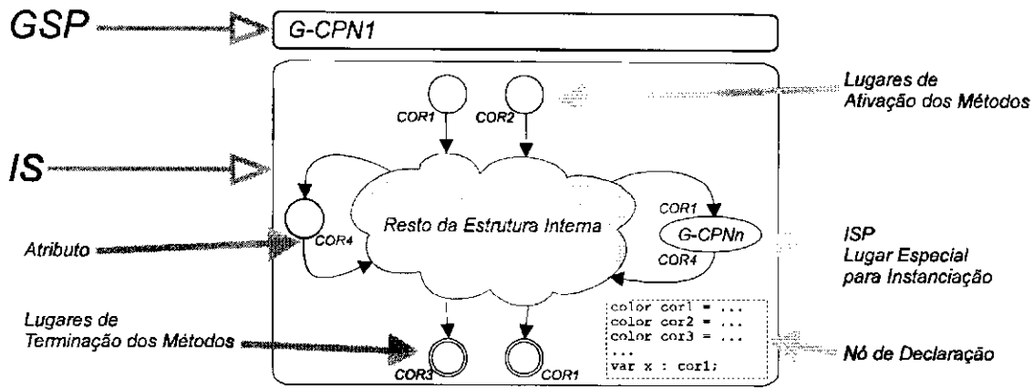

Figura 3.3 - Convenções Léxicas para Módulos G-CPN

Os atributos em módulos G-CPN são associados a lugares na estrutura interna. Como atributos são considerados parte do estado global do objeto, o seu estado (marcação) é único e portanto, compartilhado por todos os contextos do mesmo módulo G-CPN. Conseqüentemente, os diversos contextos ativos concorrem pela utilização dos atributos (fichas nos lugares que os representam).

Esta forma de definir o modelo G-CPN permite que durante a modelagem, um módulo G-CPN represente ou um objeto propriamente dito, com características intrinsecamente concorrentes, ou uma classe, que a cada invocação constrói objetos idênticos. Neste caso, os atributos declarados devem representar atributos de classe!

3.2 Um exemplo de Sistema G-CPN

Nesta seção abordaremos o clássico problema produtor-consumidor como exemplo da aplicação de Sistemas G-CPN. A nossa abordagem é baseada na solução proposta em [Perkusic,1994b] através de G-Nets.

O problema produtor-consumidor, bastante difundido na literatura sobre comunicação inter-processos e sobre sistemas distribuídos, consiste na sincronização entre processos produtores e processos consumidores. Cada processo produtor gera itens que devem ser enviados a um consumidor através do uso de mensagens ou espaço compartilhado, daí porque é também conhecido como o problema do *buffer* limitado (*bounded buffer*).

Através do modelo G-CPN, na Figura 3.4 mostramos os módulos que modelam o comportamento de processos produtores e consumidores.

Observando a interface do módulo produtor pode-se verificar que apenas um método existe. O lugar de mesmo nome na estrutura interna determina o lugar de ativação, onde a ficha de inicialização será depositada. O lugar com borda dupla indica a terminação do método. O método deve ser acionado a fim de que o objeto produza um certo número de itens e os envie para consumo. O módulo produtor não possui nenhum atributo.

O funcionamento do método é simples: ao ser ativado, a ficha de inicialização que transporta um número inteiro indica quantos itens devem ser produzidos. Se o valor dessa ficha for **0**, apenas a transição A pode ocorrer, o que finaliza a atividade do método. Caso o valor da ficha seja positivo, digamos **n**, a transição B dispara e coloca fichas nos lugares de saída com valor **n** no lugar W1 e valor **ask** (ver definições de cores no nó de declaração) no ISP, invocando o método *C.check_state*. O método *check_state* do objeto C deve retornar uma indicação do estado do consumidor. Dado que a cor de retorno do ISP é *STATE*, apenas três respostas são possíveis: **free** indicando que o consumidor estava livre, e que agora espera pelo envio de um item para consumo; **wait** indicando que o consumidor está em espera pelo envio de algum ítem de outro produtor; e **consuming** indicando que o consumidor está em processo de consumo e portanto não pode atender a novos pedidos. Caso a resposta seja diferente de **free** o produtor entra em loop através do disparo da transição E que repetidamente envia **ask**'s para o método *C.check_state*. Quando a resposta obtida for **free**, a transição C ocorre, implicando na produção de um **item**, que será enviado para o método *C.consume* do consumidor, e no depósito de uma nova ficha de valor **n** no lugar W2. Espera-se que o método *consume* retorne apenas um valor: **ack** indicando o consumo e a liberação do consumidor. Após isso, a transição D ocorre, colocando uma nova ficha de valor **n-1** no lugar *produce*, repetindo o ciclo até que não haja mais itens por produzir.

O módulo consumidor por sua vez tem dois métodos declarados na interface: *check_state* e *consume*. Apenas um atributo é utilizado no módulo consumidor e indica o estado do único consumidor no sistema. Esta forma de definir o sistema implica que o módulo produtor G-CPN modela uma classe de produtores, enquanto o módulo consumidor modela um objeto específico. Isto caracteriza alternativas de modelagem disponíveis para o desenvolvedor de sistemas.

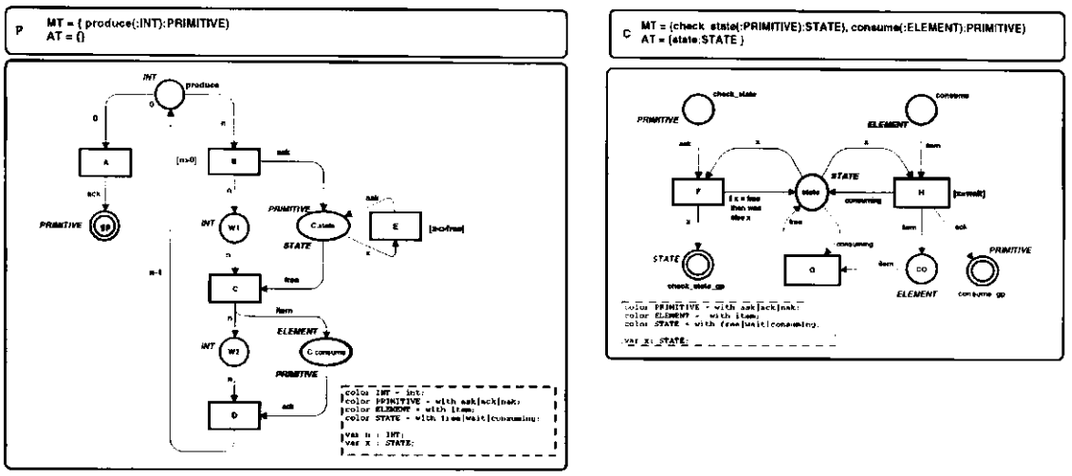

Figura 3.4 - Sistema G-CPN produtor-consumidor

O método *check_state* funciona como esperado pelo objeto produtor: ao chegar uma ficha no lugar de ativação, que deve conter um **ask**, habilita-se a transição F. Quando F ocorre, o estado do consumidor é ligado à variável *x*. Caso *x* seja igual a **free** o novo estado do consumidor passa a ser **wait**. Caso *x* seja diferente de **free** o estado será mantido inalterado. Em qualquer caso, o estado retornado (ficha depositada no lugar de terminação) será *x*, igual ao estado anterior do consumidor.

A presença de uma ficha de cor *ELEMENT* no lugar de ativação do método *consume* habilita a ocorrência da transição H, desde que o estado do consumidor seja **wait** o que é assegurado pela guarda da transição. Se a transição H ocorre, então três fichas são colocadas nos lugares de saída: o estado do consumidor é estabelecido como **consuming**, uma confirmação **ack** liberando o produtor é depositada no lugar de terminação, e o processo de consumo propriamente dito é iniciado através do deposito de **item** no lugar CO. Ao término do consumo, a transição G libera novamente o estado do consumidor, mudando seu estado de **consuming** para **free**. O depósito de uma ficha no lugar de terminação antes da finalização da ação do método permite obter maior proveito da natureza concorrente do sistema, tornando possível ao produtor voltar a produzir enquanto o consumidor exerce seu papel.

Embora trate-se de um exemplo simples, a especificação e a modelagem formais deste sistema através de G-CPN e a sua transformação em um sistema CPN puro, através da técnica abordada no Capítulo 4 desta dissertação, permitiram-nos a verificação de diversas características desejadas, assim como a extração de novos conhecimentos sobre

o sistema. Além disso, diversos erros semânticos foram corrigidos no decorrer do desenvolvimento da modelagem através da interação com a ferramenta CPN. Como exemplo, podemos citar o fato de que a necessidade de estabelecer uma marcação inicial para o atributo STATE no módulo consumidor (o que não foi deixado explícito) só foi percebida na primeira tentativa de simulação, quando nada funcionou como esperado! Além disso, avaliamos o grafo de estados do sistema para diversas situações com vários produtores concorrentes. O que se observou foi a inexistência de bloqueios e o fato de que a marcação final é sempre a mesma: o estado do consumidor volta sempre a **free** e nenhuma ficha é acumulada noutros lugares!

3.3 Definições Formais para Módulos G-CPN

Nesta seção apresentamos as definições formais que estabelecem G-CPN, através de uma seqüência de formalismos intermediários que gradativamente avançam em direção a G-CPN. Partimos da definição formal de CPN até chegarmos às definições formais dos Módulos G-CPN introduzidos na Seção 3.1.

É importante ressaltar que após consolidado e verificado, o formalismo não precisa ser preocupação por parte de usuários, analistas, projetistas e/ou desenvolvedores de sistemas. A existência das definições formais dá garantia de semântica única aos sistemas modelados, além de permitir a existência de técnicas automáticas de análise, verificação e possivelmente de geração automática de código para prototipagem.

3.3.1 Módulo G-CPN Isolado Não-concorrente

O primeiro passo na direção de G-CPN é elaborarmos as definições formais para um módulo de redes de Petri Coloridas que permita encapsular métodos e atributos. Este primeiro módulo pode ser visto como uma forma de encapsular toda uma rede de Petri Colorida em um único objeto capaz de receber pedidos de serviços.

O objeto será definido por dois elementos: uma *interface* e uma *estrutura interna*. A interface é o elemento reponsável pela interação com o usuário, nela são declarados os métodos e os atributos encapsulados. O objeto encerra toda a especificação funcional em uma única rede, sua estrutura interna, sem que lhe seja permitido interagir com outros objetos. A interface deve prover meios de acesso aos resultados quando estes forem

obtidos. Além disso, o módulo só precisa prover um único serviço a um único usuário por vez.

SINTAXE

A estrutura interna do módulo especifica o comportamento mediante ativações de seus métodos. Sintaticamente a IS é uma rede de Petri Colorida que atende a certas restrições. A interface por sua vez, provê uma visão abstrata de todo o módulo, além de mecanismos de entrada e saída de dados. Na interface são declarados os conjuntos de métodos e atributos, bem como funções que mapeiam esses elementos em seus correspondentes na estrutura interna. Cada atributo é associado a um lugar e todo método está associado a dois lugares: um de ativação e um de terminação. Esta abordagem permite que a IS permaneça estruturalmente igual a uma CPN.

Estabelecendo uma comparação com a teoria dos *tipos abstratos de dados* (álgebra de múltiplos domínios ou polisortida, mais especificamente) o conjunto de atributos modela o domínio de interesse do *tipo*, os métodos modelam as operações que são permitidas sobre os domínios e a interface caracteriza a visão abstrata, mediante a qual não é necessário considerar detalhes internos de implementação.

O *módulo*, portanto, permite ser visualizado sob dois pontos de vista diferentes: uma visão interna (IS) na qual percebe-se a lógica de implementação e os detalhes de modelagem dos atributos; e uma visão externa (GSP), cujo objetivo é considerar apenas os aspectos de utilização dos serviços do objeto. É importante ressaltar que os requisitos estabelecidos não descrevem ainda a necessidade de prover mecanismos de comunicação inter-módulos nem de atividade concorrente intra-módulos, tratando-se portanto de um módulo de operação isolada e não-concorrente.

A seguir damos a definição formal correspondente à estrutura sintática do módulo descrito. É importante ressaltar que consideramos a existência de uma linguagem funcional formalmente completa para a construção de inscrições em todos os módulos descritos a seguir. Nesta dissertação, em especial, utilizaremos a linguagem CPN-ML utilizada nas redes Coloridas e descrita em [Jensen, 1992] e em [DesignCPN].

Definição 3.1 - *Sintaxe de um Módulo G-CPN Isolado e Não-Concorrente*

Um ***Módulo G-CPN Isolado Não-concorrente*** é uma dupla <GSP, IS> onde GSP, chamada de interface é a tupla <MT, AT, AP, GP> e IS, chamada de estrutura interna é a tupla <Σ, P, T, A, N, C, G, E, I> que satisfazem aos seguintes critérios:

Quanto à estrutura interna - IS:

(i) Σ é um conjunto finito de tipos não vazios denominados ***conjuntos de cores***

(ii) P é um conjunto finito de ***lugares***

(iii) T é um conjunto finito de ***transições***

(iv) A é um conjunto finito de ***arcos***

tais que: $P \cap T = P \cap A = T \cap A = \varnothing$

(v) N é uma ***função de nós***, tal que:

$N: A \rightarrow P \times T \cup T \times P$

(vi) C é uma ***função de cores***, tal que:

$C: P \rightarrow \Sigma$

(vii) G é uma ***função de guardas***, tal que:

$G: T \rightarrow$ *EXPRESSÕES*, desde que

$\forall t \in T$: [Type(G(t))=BOOLEAN $\wedge$ Type(Var(G(t)))$\subseteq\Sigma$]

(viii) E é uma ***função de expressões de arcos***, tal que:

$E: A \rightarrow$ *EXPRESSÕES*, desde que

$\forall a \in A$: [Type(E(a))=$C(p(a))_{MS}$ $\wedge$ Type(Var(E(a)))$\subseteq\Sigma$]

onde p(a) é o lugar de N(a).

(ix) I é uma ***função de inicialização***, tal que:

$I: P \rightarrow$ *EXPRESSÕES*, desde que:

$\forall p \in P$: [Type(I(p))=$C(p)_{MS}$ $\wedge$ I(p) é fechada]

Quanto à interface - GSP:

(x) MT é um conjunto finito e não-vazio de ***métodos***

(xi) AT é conjunto finito de ***atributos***, tal que:

$AT \subseteq P$

(xii) AP é uma ***função de ativação de métodos***, tal que:

$AP: MT \rightarrow P$

onde $\forall m \in MT$: [$AP(m)=p_1 \Rightarrow I(p_1)=0$]

(xiii) GP é uma ***função de terminação de métodos***, tal que:

GP: MT $\rightarrow$ P

onde $\forall m \in$ MT: [AP(m)=$p_l \Rightarrow$ I(p_l)=*0*]

Observações:

Na Definição 3.1, as linhas de (i) até (ix) que determinam a estrutura interna do módulo são idênticas à definição da estrutura de uma rede de Petri Colorida Não-hierárquica, incluindo as mesmas restrições. As linhas restantes de (x) a (xii) agregam os conceitos de encapsulação, métodos e atributos.

(x) O conjunto MT descreve os métodos disponíveis no módulo. Um módulo deve oferecer pelo menos um serviço, e cada serviço deve ser unicamente identificável.

(xi) O conjunto AT descreve os atributos encapsulados pelo módulo. Em termos formais, cada atributo é um dos lugares da estrutura interna. É como se apenas apontássemos para alguns lugares da rede para identificá-los como atributos. O tipo de um atributo é dado pela aplicação da função de cores definida em (vi).

(xii) e (xiii) As funções AP e GP servem para mapear cada método a dois lugares na estrutura interna: um lugar de ativação e um lugar de terminação. As restrições indicam que esses lugares não devem ter marcações iniciais.

É importante observar que embora formalmente *métodos* e *atributos* sejam elementos pertencentes à interface do módulo, a ***implementação de um método*** e a ***representação de um atributo*** são partes da estrutura interna. No decorrer deste documento, permitimo-nos a utilização dos termos ***método*** e ***atributo*** quando na realidade intencionamos dizer ***implementação do método*** e ***representação do atributo***. Quando for necessário, ou quando não for possível extrair a informação do contexto, deixaremos explícito a que conceito nos referimos.

SEMÂNTICA

O propósito do estabelecimento do *Módulo G-CPN Isolado Não-concorrente* é preservar o comportamento associado às redes CPN a nível interno e proporcionar uma visão externa para toda a rede. Conseqüentemente, as definições semânticas ou relativas ao comportamento da *IS* de um módulo *G-CPN Isolado Não-concorrente* (***ligação***,

elemento de ficha, ***elemento de ligação***, ***marcação***, ***passo***, ***passo habilitado*** e ***ocorrência***) são, na sua totalidade, idênticas às definições dos conceitos de mesmo nome para redes CPN Não-hierárquicas. Por conseguinte, basta que nos concentremos na elucidação do comportamento relativo à interface do módulo.

A interface de um *Módulo G-CPN isolado Não-concorrente* serve essencialmente como porta de comunicação usuário-objeto. Visto que o objetivo maior deste trabalho é a construção de uma ferramenta para a modelagem de sistemas reais, é preciso prover mecanismos de acesso aos módulos que permitam a ativação dos seus métodos com a respectiva passagem de parâmetros, bem como a recepção de resultados gerados. Assim, dois eventos serão definidos formalmente: ativação e terminação.

A formalização leva em consideração que o termo *não-concorrente* expressa que esta classe de módulos não precisa suportar invocações concorrentes de métodos. Portanto, consideramos que a interface oferece um ponto único de acesso aos métodos, e que ocorrida uma ativação, novas ativações só ocorrerão depois do término da atividade gerada pela primeira.

Definição 3.2 - *Ativação de um Método*

Seja G um módulo *G-CPN Isolado Não-concorrente* e $m \in MT$ um método de G, então a ativação do método *m* pode ocorrer. Se ocorre, a marcação do módulo G muda de M_1 para M_2 dada por:

(i) $M_2 = M_1 + <AP(m),c>$

Onde:

$c \in C(AP(m))$.

A Definição 3.2 estabelece o que é uma ativação do ponto de vista de um módulo: é o surgimento "espontâneo" de uma ficha de cor apropriada no lugar de ativação de algum dos métodos do módulo. O termo espontâneo foi utilizado para enfatizar o fato de que do ponto de vista interno, o módulo nada mais sabe sobre eventos externos.

Definição 3.3 - *Terminação*

Sejam G um módulo G-CPN *Isolado Não-concorrente*, *m* um método de G e um *elemento de ficha* <p,c>. A terminação do método *m* está habilitada pelo elemento de ficha <p,c> se e somente se:

(i) p=GP(m) e c∈C(p)

Definição 3.4 - *Ocorrência de uma Terminação*

Se a terminação do método *m* está habilitada pelo elemento de ficha <p,c>, então ela pode ocorrer. Se ocorre, a marcação do módulo G muda de M_1 para M_2 dada por:

(i) $M_2 = M_1 - <p,c>$

A Definição 3.3 determina que a condição de habilitação da *terminação* de um método, é a existência de uma ficha no respectivo lugar de terminação. A Definição 3.4 estabelece qual é a marcação após a ocorrência do evento. A definição da nova marcação reflete a idéia intuitiva de que a ocorrência do evento faz a ficha "desaparecer" do lugar de terminação. Novamente, o termo é usado entre aspas para denotar que do ponto de vista interno, o módulo nada sabe sobre eventos externos.

3.3.2 Módulo G-CPN Isolado

Embora capture um nível de expressão acima das redes CPN puras, o modelo definido na Seção 3.3.1 ainda não oferece facilidades para a especificação de sistemas cujos módulos tenham comportamento intrinsecamente concorrente. Portanto, tal módulo não permite modelar classes de objetos por não especificar como se dá a criação de novas instâncias, nem pode prover serviços a múltiplos usuários concorrentemente.

A fim de modelar procedimentos que possam ser invocados concorrentemente, o projetista precisaria administrar explicitamente os mecanismos de controle de concorrência na *IS* do módulo. Para remediar este problema propomos nesta seção um novo tipo de módulo G-CPN que permite a invocação concorrente de seus métodos, sem que para isso o projetista do sistema precise dedicar esforços especiais.

É preciso fazer aqui, uma observação importante: embora, através deste novo módulo seja possível modelar métodos que permitem invocações *concorrentes*, nada implica que as propriedades dinâmicas relativas a cada uma das *ativações* sejam absolutamente independentes. Isto ocorre porque, embora as *ativações* disponham de estados particulares (marcações próprias), eventualmente estes estados só são alcançáveis mediante a obtenção do direito de acesso a atributos, que por natureza são elementos compartilhados na estrutura interna.

SINTAXE

A estrutura sintática de um *Módulo G-CPN Isolado Concorrente* é idêntica à estrutura sintática do módulo apresentado na seção anterior. Isto se dá porque as diferenças são exclusivamente comportamentais, bastando dar uma nova interpretação aos elementos sintáticos que compõem um módulo. Por conseguinte, passaremos diretamente à definição de uma nova semântica associada, sem nos determos na transcrição das definições sintáticas.

SEMÂNTICA

Do ponto de vista de cada ativação, o comportamento deste novo tipo de módulo deve ser o mais parecido possível ao do módulo *não-concorrente*. Ou seja, as novas definições para a *invocação* e a *ativação* de um método devem ser elaboradas de forma a garantir essa semelhança. Para cada usuário, o objeto deve aparentar comportamento semelhante a um módulo *não-concorrente*, exceto por eventuais "efeitos" colaterais causados pelo compartilhamento de alguns recursos (atributos).

Com esse objetivo, introduzimos o conceito de *contexto*[1] de ativação. A idéia é que cada ativação de método, seja associada a um *contexto* sob o qual a rede deve operar. Assim, se todas as fichas forem "marcadas" com o contexto a que pertencem, o agente responsável pela execução poderá distinguir as fichas durante a simulação e conseqüentemente poderá gerenciar várias *ativações* concorrentemente sem que haja perigo de "misturar" as fichas de cada uma.

O principal problema introduzido por esta abordagem, como já foi citado, é o tratamento diferenciado que se deve dar aos *atributos*. A solução adotada foi considerar um contexto especial que engloba todos os demais contextos de ativações. Assim, qualquer ativação poderá acessar a marcação dos lugares que representem *atributos* na estrutura interna do módulo.

---

[1]O termo foi escolhido a partir do conceito de *contexto*, que permite a um único processador operar em diversos processos "simultaneamente".

Formalmente, essa idéia será efetuada pela inclusão de um novo elemento na tupla que caracteriza uma ficha. Essa alteração implica em outras mais profundas, entretanto, é importante deixar claro qual é o ponto de partida para essas transformações. A seguir apresentamos a nova definição formal de *elemento de ficha*. As definições relativas aos demais conceitos semânticos serão apenas comentadas a fim de evitar sua repetição na seção seguinte onde suas versões definitivas serão apresentadas.

Definição 3.5 - *Elemento de Ficha e Conjunto de Todos os Elementos de Fichas*

Seja O um conjunto infinito de contextos. Um *elemento de ficha* é uma tripla <p,c,o>, na qual se verifica que:

(i) $p \in P$ é um lugar;

(ii) $c \in C(p)$ é uma cor pertencente ao conjunto de cores do lugar p;

(iii) $o \in O$ é um contexto;

A Definição 3.5 define *elemento de ficha* como uma tripla. Desta forma, uma ficha em um módulo G-CPN Isolado pode ser unicamente identificada pela cor que transporta, pelo lugar em que está e pelo contexto ao qual pertence. O conjunto O, também introduzido aqui, caracteriza a natureza dos contextos.

Esta definição implica em mudanças nas demais definições semânticas. Isso ocorre, por introduzir um caráter dual a todos os conceitos que se baseiam na definição de *elemento de ficha*. Uma marcação, por exemplo, pode ser encarada sob dois pontos de vista: o do módulo e o do contexto. Cada módulo tem sua própria marcação: um elemento formal (um *multi-set*) indicador do seu estado. Entretanto, pode-se falar também no estado de cada ativação, cuja formalização se dará através do novo conceito de sub-marcação de contextos. Uma sub-marcação é o *multi-set* de todos os *elementos de ficha* relativos a um único contexto pertencentes à marcação do módulo.

Um passo também pode ser encarado sob os dois pontos de vista: do módulo ou de um contexto. Assim, introduzimos o conceito de sub-passo de contexto, que nada mais é que o *multi-set* de todos os *elementos de ligação* relativos a um único contexto pertencentes a um passo do módulo, ou simplesmente a parte de um passo relativa a um contexto. Outro importante conceito decorrente da introdução dos contextos é o de sub-passo habilitado, definido de forma similar à de passo habilitado em redes CPN puras.

Estes novos conceitos, além de permitirem predicar mais facilmente sobre contextos isoladamente, facilitam o estabelecimento de outras definições formais. Por exemplo, um passo será dito habilitado para um módulo G-CPN numa dada marcação, unicamente se para todos os contextos ativos, os sub-passos forem habilitados nas respectivas sub-marcações e se há fichas suficientes nos lugares correspondentes aos atributos. Para simplicidade do formalismo, ocorrências só são definidas para passos do módulo e não para sub-passos. Finalmente, eventos de ativação e terminação, permanecem com suas definições praticamente inalteradas.

A classe de módulos G-CPN proposta nesta seção permite encapsular uma rede CPN e estabelecer uma interface bem-definida de acesso às informações encapsuladas através de serviços denominados métodos. Os serviços podem ser ativados concorrentemente sem que haja maiores problemas. Além disso, devido ao uso de um modelo de computação estritamente concorrente (redes de Petri), cada ativação ou contexto de método pode dar origem a diversas sub-atividades concorrentes a fim de executar o serviço.

3.3.3 Módulo G-CPN

A fim de completarmos o modelo G-CPN como proposto informalmente na Seção 3.1, faz-se necessário acrescentar algum mecanismo de interação entre módulos. Isto será feito através da introdução de um novo elemento sintático e novos eventos que permitem que um módulo utilize serviços externos. O funcionamento do elemento sintático necessário, denominado ISP, bem como os eventos a ele associados (invocações e retornos) foram informalmente expostos na Seção 3.1.

Esta seção apresenta a versão final da formalização para um Módulo G-CPN. A fim de tornar esta seção independente das demais, todas as definições necessárias foram aqui dispostas, apesar de eventuais semelhanças entre estas e as de módulos anteriormente descritos. Finalmente, esta formalização e a nomenclatura associada será usada como base para a definição formal de Sistemas G-CPN na seção seguinte.

SINTAXE

Com o objetivo de preservar a definição sintática da estrutura interna inalterada, será necessário incluir os novos elementos sintáticos na interface do módulo, e associá-los a elementos normais da estrutura interna. Embora, nenhum novo elemento seja adicionado

à IS, algumas restrições serão impostas aos elementos que os representem, a fim de que se possa inferir o comportamento adequado às invocações de serviços remotos.

Por se tratar de um elemento de comunicação, o ISP deve ser declarado no GSP do módulo. Na estrutura interna, cada ISP será devidamente decomposto em dois lugares. O primeiro lugar representa a parte superior do ISP na qual são depositadas as fichas indicando intencionalidade de invocação de serviços remotos. O segundo lugar representa a parte inferior do ISP na qual são depositadas as fichas de retorno do serviço invocado. Uma função presente no GSP permite interligar a declaração do elemento invocador aos seus respectivos lugares na IS. Outra função na interface mapeia cada ISP a um serviço prestado externamente.

Definição 3.6 - *Sintaxe de um Módulo G-CPN*

Um ***Módulo G-CPN*** é uma dupla <GSP, IS> onde GSP, chamado de *interface* é a tupla <MT, AT, ISP, AP, GP, UP, LP, OBJ> e IS, chamado de *estrutura interna* é a tupla <Σ, P, T, A, N, C, G, E, I> que satisfazem aos seguintes critérios:

Quanto à estrutura interna - IS:

(i) Σ é um conjunto finito de tipos não vazios denominados ***conjuntos de cores***

(ii) P é um conjunto finito de ***lugares***

(iii) T é um conjunto finito de ***transições***

(iv) A é um conjunto finito de ***arcos***

tais que: $P \cap T = P \cap A = T \cap A = \emptyset$

(v) N é uma ***função de nós***, tal que:

$N: A \rightarrow P \times T \cup T \times P$

(vi) C é uma ***função de cores***, tal que:

$C: P \rightarrow \Sigma$

(vii) G é uma ***função de guardas***, tal que:

$G: T \rightarrow$ *EXPRESSÕES*, desde que

$\forall t \in T: [Type(G(t))=BOOLEAN \wedge Type(Var(G(t))) \subseteq \Sigma]$

(viii) E é uma ***função de expressões de arcos***, tal que:

$E: A \rightarrow$ *EXPRESSÕES*, desde que

$\forall a \in A: [Type(E(a))=C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$

onde p(a) é o lugar de N(a).

(ix) I é uma ***função de inicialização***, tal que:

I: P $\rightarrow$ *EXPRESSÕES*, desde que:

$\forall p \in P$: [Type(I(p))=$C(p)_{MS}$ $\wedge$ I(p) é fechada]

Quanto à interface - GSP:

(x) MT é um conjunto finito e não-vazio de ***métodos***

(xi) AT é um conjunto finito de ***atributos***, tal que:

$AT \subseteq P$

(xii) ISP é um conjunto finito de ***invocadores***

(xiii) AP é uma ***função de ativação de métodos***, tal que:

AP: MT $\rightarrow$ P

onde $\forall m \in$ MT: [AP(m)=$p_1 \Rightarrow$ I(p_1)=*0*]

(xiv) GP é uma ***função de terminação de métodos***, tal que:

GP: MT $\rightarrow$ P

onde $\forall m \in$ MT: [GP(m)=$p_1 \Rightarrow$ I(p_1)=*0*]

(xv) UP é uma função de ***lugar superior*** de invocadores

UP: ISP $\rightarrow$ P

$\forall isp \in$ ISP: [UP(isp) = p1 $\Rightarrow$ p1$^\bullet = \varnothing \wedge$ I(p1) = 0]

(xvi) LP é uma função de ***lugar inferior*** de invocadores

LP: ISP $\rightarrow$ P

$\forall isp \in$ ISP: [LP(isp) = p1 $\Rightarrow$ $^\bullet$p1 = $\varnothing \wedge$ I(p2) = 0].

(xvii) OBJ é uma função ***objetivo*** dos invocadores

OBJ: ISP $\rightarrow$ S

onde S é o conjunto dos serviços invocáveis.

Observações:

(i) a (x) Definição da estrutura interna de um módulo G-CPN. Perceba que é idêntica à definição sintática de redes coloridas não-hierárquicas (ver [Jensen, 1992]).

(x) e (xi) O conjunto de atributos define os tipos de dados sobre os quais o módulo G-CPN atua. O conjunto de métodos declara e identifica os serviços permitidos através da interface.

(xii) O conjunto de invocadores contém um elemento identificador para cada ponto na estrutura interna que deva ser considerado sob a semântica de um ISP.

(xiii) e (xiv) As funções de ativação e terminação de métodos associam a cada método dois lugares: o primeiro indica onde devem ser colocadas as fichas de ativação e o segundo indica de onde devem ser retiradas as fichas de terminação. Além disso, os conjunto de cores associados aos lugares de ativação e terminação definem indiretamente os tipos de dados recebidos e retornados pelos métodos. Exige-se que estes lugares não tenham marcação inicial nenhuma.

(xv) e (xvi) As funções de lugar superior e lugar inferior de invocadores associam dois lugares a cada invocador declarado no GSP. A presença de fichas no lugar superior caracteriza as habilitações de invocação. O lugar inferior é onde devem ser colocadas as fichas relativas a retornos. Lugares superiores não podem ter transições de saída, e lugares inferiores não podem ter transições de entrada. De forma análoga aos lugares de ativação e terminação, também determinam as cores dos dados enviados e recebidos. Exige-se que tenham marcação inicial nula.

(xvii) Finalmente a função objetivo leva cada elemento de invocação - ISP da estrutura interna no identificador do serviço que deve ser invocado. O conjunto S é descrito apenas informalmente a fim de que represente, a nível de módulo, um conjunto de serviços. Na definição de Sistemas G-CPN, o conjunto S será adequadamente formalizado.

A título de exemplo apresentamos a seguir a versão formal da sintaxe do módulo G-CPN produtor (PROD) do Sistema G-CPN Produtor-Consumidor apresentado na Figura 3.4:

(i) PROD = <GSP,IS>

(ii) GSP = <MT,AT,ISPAP,GP,UP,LP,OBJ>

(iii) IS = <Σ, P, T, A, N, C, G, E, I>

(iv) Σ = {INT, PRIMITIVE, ELEMENT, STATE}

(v) P = {PRODUCE, W1, W2, UP1, LP1, UP2, LP2, GP}

(vi) T = {A,B,C,D,E}

(vii) A = {produce_a, produce_b, a_gp, b_w1, b_up1, w1_c, lp1_c, lp1_e, e_up1, c_w2, c_up2, w2_d, lp2_d, d_produce}

(viii) N = {(produce_a, (PRODUCE, A)), (produce_b, (PRODUCE, B)), (a_gp, (A, GP)), (b_w1, (B, W1)), (b_up1, (B, UP1)), (w1_c, (W1, C)), (lp1_c, (LP1, C)), (lp1_e, (LP1, E)), (e_up1, (E, UP1)), (c_w2, (C, W2)), (c_up2, (C, UP2)), (w2_d, (W2, D)), (lp2_d, (LP2, D)), (d_produce, (D, PRODUCE))}

(ix) C = {(PRODUCE, INT), (W1, INT), (W2, INT), (UP1, PRIMITIVE), (LP1, STATE), (UP2, ELEMENT), (LP2, PRIMITIVE), (GP, PRIMITIVE)}

(x) G = {(A, ''), (B, 'n>0'), (C, ''), (D, ''), (E, 'x<>free')}

(xi) E = {(produce_a, '0'), (produce_b, 'n'), (a_gp, 'ack'), (b_w1, 'n'), (b_up1, 'ask'), (w1_c, 'n'), (lp1_c, 'free'), (lp1_e, 'x'), (e_up1, 'ask'), (c_w2, 'n'), (c_up2, 'item'), (w2_d, 'n'), (lp2_d, 'ack'), (d_produce, 'n-1')}

(xii) I = {(PRODUCE, ''), (W1, ''), (W2, ''), (UP1, ''), (LP1, ''), (UP2, ''), (LP2, ''), (GP, '')}

(xiii) MT = {produce}

(xiv) AT = { }

(xv) ISP = {isp1, isp2}

(xvi) AP = {(produce, PRODUCE)}

(xvii) GP = {(produce, GP)}

(xviii) UP = {(isp1, UP1), (isp2, UP2)}

(xix) LP = {(isp1, LP1), (isp2, LP2)}

(xx) OBJ = {(isp1, 'C.check_state'), (isp2, 'C.consume')}

SEMÂNTICA

As definições a seguir referem-se explicitamente a módulos G-CPN apresentados formalmente pela Definição 3.6. No que segue, as seguintes convenções serão utilizadas:

(i) G = <GSP,ISP> é qualquer módulo G-CPN

(ii) GSP = <MT, AT, ISP, AP, GP, UP, LP, OBJ> é a interface de G

(iii) IS = <Σ, P, T, A, N, C, G, E, I> é a estrutura interna de G

(iv) ISP = { isp_1,isp_2, ..., isp_i, ..., isp_n } é o conjunto de serviços de G

(v) M, M_1 e M_2 são marcações alcancáveis de G

(vi) Var(t) é o conjunto de variáveis associadas à transição t

(vii) Type(v) é o conjunto de cores da variável v

(viii) O é o conjunto de contextos.

Definição 3.7 - *Elemento de Ficha e Conjunto de Todos os Elementos de Ficha*

Seja O um conjunto infinito de contextos. Um *elemento de ficha* é uma tripla <p,c,o>, na qual se verifica que:

(i) $p \in P$ é um lugar;

(ii) $c \in C(p)$ é uma cor pertencente ao conjunto de cores do lugar p;

(iii) $o \in O$ é um contexto;

A Definição 3.7 estabelece que um elemento de ficha é determinado por um lugar na estrutura interna do módulo, uma cor pertencente ao conjunto de cores do lugar, e um contexto ao qual a ficha pertence. Denotaremos o conjunto de todos os *elementos de ficha* por TE.

Definição 3.8 - *Marcação e Sub-marcação de um Contexto*

Uma marcação é um *multi-set* sobre TE. Se M é a marcação de G, então o *multi-set* M^o definido como segue é a sub-marcação relativa ao contexto o:

(i) $\forall <p,c,o_1> \in M: M^o(<p,c,o_1>) = M(<p,c,o>)$ se $o_1 = o$

(ii) $= 0$ se $o_1 \neq o$

A Definição 3.8 estabelece uma marcação como sendo um *multi-set* sobre o conjunto de *elementos de ficha*. A definição de sub-marcação é construída usando a notação de função para *multi-sets*. A sub-marcação relativa a um dado contexto é dada pela marcação do módulo menos os *elementos de ficha* cujo contexto não é o referido. Em termos intuitivos, a sub-marcação de um contexto "ignora" fichas dos demais contextos.

Definição 3.9 - *Ligação*

Uma *ligação* de uma transição t é uma função b definida sobre Var(t), tal que:

(i) $\forall v \in Var(t): b(v) \in Type(v)$

(ii) $G(t)<b>$

Devido a sua independência da definição de *elemento de ficha*, a definição de *ligação* permanece inalterada em relação à definição de mesmo nome para redes CPN. O conjunto de todos as *ligações* para a transição t será denotado por B(t).

Definição 3.10 - *Elemento de Ligação e Conjunto dos Elementos de Ligação*

Seja O um conjunto infinito de contextos. Um *elemento de ligação* é uma tripla <t,b,o>, na qual se verifica que:

(i) $t \in T$ é uma transição;

(ii) $b \in B(t)$ é uma *ligação* pertencente ao *conjunto de ligações* da transição t;

(iii) $o \in O$ é um contexto;

A Definição 3.10 estabelece um *elemento de ligação* como sendo a conjunção entre uma transição e uma ligação específica daquela transição. Dado um *elemento de ligação*, pode-se identificar de forma única, que substituições em que transição ele denota. O conjunto de todos os *elementos de ligação* será denotado por BE.

Definição 3.11 - *Passo*

Um passo é um *multi-set* finito sobre BE. Um *multi-set* Y^o é o sub-passo relativo ao contexto *o* ou simplesmente o passo do contexto *o* se e somente se:

(i) $\forall <t,b,o_1> \in Y: Y^o(<t,b,o_1>) = Y(<t,b,o_1>)$ se $o_1 = o$

(ii) $= 0$ se $o_1 \neq o$

As definições de *elemento de ligação* e passo são bastante semelhantes às de *elemento de ficha* e marcação. A tupla de um *elemento de ligação* também transporta o valor do contexto ao qual é associado, e um sub-passo é definido de forma semelhante à usada para definir uma sub-marcação. Um sub-passo relativo a um dado contexto é dado pelo passo em questão menos os *elementos de ligação* de outros contextos. Intuitivamente, um sub-passo de um contexto "ignora" os *elementos de ligação* dos demais contextos.

Definição 3.12 - *Sub-passo habilitado*

Um sub-passo Y^o é dito habilitado se e somente se:

(i) $\forall p \in P: \sum_{(t,b,o) \in Y^o} E(p,t)<b> \leq M^o(p)$

O conceito de sub-passo habilitado, já introduzido para o módulo apresentado na seção anterior, tem sua utilidade no sentido de simplificar a definição seguinte. Intuitivamente, um sub-passo está habilitado se os lugares de entrada relativos a todos os *elementos de*

ligação têm fichas suficientes, do contexto em questão, para satisfazer as avaliações das expressões dos arcos.

Definição 3.13 - *Passo habilitado*

Um passo Y é dito habilitado na marcação M se e somente se:

(i) $\forall o \in O: \forall M^o \subseteq M: Y^o$ está habilitado em M^o

(ii) $\forall p \in AT: \sum_{(t,b,o) \in Y} E(p,t)<b> \leq M(p)$

Esta definição permite detectar as pré-condições para uma ocorrência de passo num módulo G-CPN. Um passo pode ocorrer apenas se (i) todos os sub-passos estão habilitados e (ii) os lugares que representam atributos têm fichas suficientes para satisfazer simultaneamente as expressões relativas a todos os contextos simultaneamente.

A partir da Definição 3.14 até a Definição 3.18 apresentamos a formalização dos eventos propriamente ditos em módulos G-CPN.

Definição 3.14 - *Ocorrência de um Passo*

Um passo Y habilitado na marcação M_1 pode ocorrer. Se ocorre, então a marcação do módulo muda de M_1 para M_2 dada por:

(i) $\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b,o) \in Y} E(p,t)<b>) + \sum_{(t,b,o) \in Y} E(t,p)<b>$

Definição 3.15 - *Ativação*

A ativação do método m pode ocorrer num módulo cuja marcação é M_1. Se ocorre, então a marcação do módulo muda de M_1 para M_2 dada por:

(i) $M_2 = M_1 + <p,c,o>$

(ii) Onde:

$p \in AP(m)$;

$c \in C(AP(m))$;

$o \in O$.

A ocorrência da ativação de um método provoca uma única alteração no estado do objeto: acresce uma ficha, denominada ficha de inicialização, no lugar de ativação do

método ativado. É interessante perceber que do ponto de vista do módulo, a ativação de um de seus métodos é um evento sempre habilitado.

Definição 3.16 - *Terminação*

A terminação do método m é habilitada na marcação M_1 pelo *elemento de ficha* <p,c,o> se e somente se p=GP(m). Neste caso, a terminação pode ocorrer, se ocorre, a marcação do módulo muda de M_1 para M_2 definida por:

(i) $M_2 = M_1 - <p,c,o>$

Definição 3.17 - *Invocação*

A invocação de $isp_i \in ISP$ é habilitada na marcação M_1 pelo *elemento de ficha* <p,c,o> se e somente se $p=UP(isp_i)$. Neste caso, a invocação pode ocorrer, se ocorre, a marcação do módulo muda de M_1 para M_2 definida por:

(i) $M_2 = M_1 - <p,c,o>$

A única conseqüência da ocorrência de uma terminação ou uma invocação num módulo G-CPN, é a eliminação da ficha que habilita o evento.

Definição 3.18 - *Retorno*

Um retorno de $isp_i \in ISP$ pode ocorrer num módulo cuja marcação é M_1. Se ocorre, então a marcação do módulo muda de M_1 para M_2 dada por:

(i) $M_2 = M_1 + <p,c,o>$

(ii) Onde:

$p \in LP(isp_i)$;

$c \in C(LP(m))$;

$o \in O$.

Perceba que a ocorrência de um retorno, em termos formais, é um evento sempre habilitado dentro de um módulo G-CPN. Portanto, do ponto de vista formal, retornos podem ocorrer, mesmo sem ocorrências prévias de invocações. Embora isto não pareça desejável do ponto de vista intuitivo, é interessante deixar a formalização da relação entre os dois eventos para uma instância hierárquica superior. Assim, qualquer que seja o sistema usuário do módulo G-CPN, ele será o responsável por garantir a seqüência adequada dos eventos externos.

Neste capítulo apresentamos as definições formais relativas à estrutura e ao comportamento de um Módulo G-CPN. O módulo assim definido permite a descrição executável de objetos de software em termos de atributos e métodos encapsulados. O acesso às informações se dá exclusivamente através de uma interface denominada GSP que permite a invocação dos métodos. Dentro da estrutura interna, denominda IS, o formalismo permite descrever métodos invocáveis concorrentemente sem que o desenvolvedor ou o projetista precise dedicar esforços especiais para o controle. Além disso, a própria natureza das Redes de Petri permite a descrição de métodos que desempenhem suas funções através de procedimentos intrinsecamente concorrentes. A fim de oferecer maior facilidade em termos de descrição de controle de concorrência, os atributos são elementos formalizados como lugares de acesso comum na estrutura interna, garantindo que apenas uma linha de execução terá acesso instantâneo. Além disso, através de um elemento de invocação, denominado ISP é possível que descrições internas dos métodos invoquem serviços de elementos externos ao módulo. Tudo isto é possível sem fazer nenhuma suposição sobre o sistema usuário do módulo, exceto que ele manterá a seqüência apropriada de eventos de comunicação.

4. SISTEMAS G-CPN E SUA RELAÇÃO COM REDES CPN NÃO-HIERÁRQUICAS

O principal objetivo deste capítulo é a apresentação da formalização de Sistemas G-CPN, bem como suas relações com redes CPN não-hierárquicas.

O capítulo está dividido em três Seções. A Seção 4.1 apresenta as definições sintáticas relativas a Sistemas G-CPN. A Seção 4.2 apresenta as definições semânticas relacionadas ao comportamento de Sistemas G-CPN. Finalmente na Seção 4.3 apresentamos um breve estudo sobre as relações que existem entre redes CPN e Sistemas G-CPN.

4.1 Sintaxe de Sistemas G-CPN

Se nos basearmos na definição intuitiva, um Sistema G-CPN poderia ser formalizado simplesmente como um conjunto de Módulos G-CPN. Entretanto, isso pode levar a diversos problemas de integração entre os módulos e conseqüentemente dificuldades bem maiores no estabelecimento das definições semânticas. Por exemplo, considere o fato de que cada módulo precisa "visualizar" os serviços providos pelos outros módulos. Isto é conseguido através da *externalização* dos elementos de acesso aos métodos de cada módulo a fim de formar um novo conjunto de serviços visível para todos os módulos igualmente.

Ainda assim, um Sistema G-CPN será definido como uma estrutura que integra um conjunto de Módulos G-CPN (ver Capítulo 3) num sistema com um objetivo em comum. Entretanto, é preciso esclarecer precisamente como se forma essa estrutura que integra além de módulos G-CPN, alguns componentes adicionais. Além disso, não é qualquer conjunto de módulos G-CPN que pode formar um Sistema G-CPN. É preciso estabelecer critérios que permitam restringir a natureza dos módulos e suas relações para que determinem um Sistema G-CPN.

Assim, por exemplo, dois módulos que não usem nenhum de seus serviços mutuamente não apresentarão nenhum problema de integração. Entretanto, se queremos integrar os

módulos A e B nos quais A invoca algum método do módulo B, então é preciso garantir que a passagem de parâmetros seja possível. Além disso, embora tenha ficado implícito, é necessário que cada método seja identificável de forma única em todo o sistema.

A definição sintática dada a seguir foi elaborada considerando as observações anteriores. O Sistema deve ser, portanto, um ambiente comum aos módulos, onde cada objeto (módulo G-CPN) pode "ver" os demais objetos e seus serviços. A estrutura formal de um Sistema G-CPN, portanto, deve prover além do conjunto de módulos, um conjunto de identificadores de serviços que cada módulo pode invocar na sua estrutura interna. Além disso a estrutura inclui o conjunto total de domínios sobre o qual o sistema opera.

Nas definições que se seguem, utilizamos o operador ponto "." já consolidado pelas linguagens orientadas a objetos, para fazer referencias a elementos pertencentes aos módulos G-CPN do sistema. Assim, o conjunto de conjuntos de cores dos módulos G_j e G_i, por exemplo, serão indicados por "$G_j.\Sigma$" e "$G_i.\Sigma$" ao invés de "Σ_j" e "Σ_i".

Definição 4.1 - *Sintaxe de um Sistema G-CPN*

Um Sistema G-CPN é a tripla $<\Sigma, S, GCPN>$, onde

(i) Σ é um conjunto finito de domínios não-vazios denominados **conjuntos de cores**

(ii) GCPN é um conjunto finito não-vazio de **módulos G-CPN**
que satisfaz às seguintes restrições:

$GCPN = \{ G_1, G_2, \ldots G_i, \ldots G_n \}$

$G_1.P \cap G_2.P \cap \ldots \cap G_i.P \cap \ldots \cap G_n.P = \varnothing$

$G_1.T \cap G_2.T \cap \ldots \cap G_i.T \cap \ldots \cap G_n.T = \varnothing$

$G_1.S \cap G_2.S \cap \ldots \cap G_i.S \cap \ldots \cap G_n.S = \varnothing$

$\forall k\ 1 \leq k \leq n$: $G_k.\Sigma \subseteq \Sigma$;

(iii) S é o **conjunto de serviços** do sistema tal que:

$S = G_1.S \cup G_2.S \cup \ldots \cup G_i.S \cup \ldots \cup G_n.S$

$\forall isp \in G_i.ISP$: $\forall s \in G_j.S$:

$G_i.OBJ(isp)=G_j.s \Rightarrow G_i.C(UP(isp))=G_j.C(AP(s)) \wedge G_i.C(LP(isp))=G_j.C(GP(s))$

A definição acima estabelece que um sistema G-CPN é formado por um conjunto de módulos G-CPN cujos métodos são unidos para formar o conjunto de serviços do sistema. É imposto, ainda, que cada módulo defina um subconjunto do conjunto de

domínios do sistema como seu próprio conjuntos de conjuntos de cores, e finalmente é requerido que todo par invocador-serviço tenha em comum os conjuntos de cores de seus lugares correspondentes.

A formalização do sistema mostrado na Figura 3.4 é dada a seguir como exemplo de um Sistema G-CPN descrito usando a definição formal:

Seja o sistema Produtor-Consumidor, denotado por SPC, a tripla <Σ, S, GCPN>, onde

(i) Σ = { INT, PRIMITIVE, ELEMENT, STATE }

(ii) GCPN = { P, C } onde:

(* Definições formais dos módulos G-CPN propriamente ditos *)

P = <P.GSP, P.IS>

...

C = <C.GSP, C.IS>

...

(* *)

(iii) S = P.S ∪ C.S = { P.produce, C.check_state, C.consume }

Dado que a estrutura SPC acima descrita atende aos critérios estabelecidos pela Definição 4.1, concluímos que SPC é um Sistema G-CPN.

4.2 Semântica de Sistemas G-CPN

Com o propósito de formalizar o comportamento de Sistemas G-CPN, faz-se necessário introduzir novos conceitos que permitam definir o que é o estado de um sistema. Nesse sentido, estabeleceremos os conceitos de ambiente, estado, e eventos, além de alguns conceitos auxiliares.

Informalmente podemos definir o ambiente como sendo a parte do estado do sistema que não se encontra representada nas marcações dos módulos. O ambiente é formalizado como uma estrutura de conjuntos de relacionamentos cujo objetivo é manter informações sobre pendências de seqüências de eventos não concluídos. Nesses termos, é fácil perceber que estamos nos referindo aos eventos de interação entre módulos: invocações, ativações, terminações e retornos.

As relações esperadas (ou desejadas) entre os eventos são as seguintes:

- para toda invocação ocorrida em algum módulo, espera-se que ocorra uma ativação em outro módulo do sistema;
- uma terminação de um método pode ocorrer apenas após uma ativação do mesmo método;
- toda terminação ocorrida em algum módulo, deve implicar na ocorrência de um retorno em outro módulo;
- o retorno de um método só deve ocorrer após uma invocação.

Naturalmente, além dos critérios já mencionados espera-se que o sistema associe corretamente os contextos interagentes.

A fim de formalizar o ambiente, três conjuntos serão estabelecidos. O primeiro é denominado conjunto de ativações pendentes (PA). Cada ativação pendente pode ser interpretada como um pedido de ativação pronto para ser efetivado. A ativação pendente deve guardar informações sobre o serviço que deve ser ativado, o contexto sob o qual será atendida e os dados de cor da ficha de inicialização a ser usada.

Definição 4.2 - *Ativações Pendentes*

Uma ativação pendente é uma tripla <s,d,o> onde:

(i) $s \in S$ é um serviço do sistema

(ii) $d \in D$ é uma cor do conjunto união de todas as cores de todos os domínios

$D = \{ c \mid c \in C \wedge C \in G_i.\Sigma \wedge G_i \in G \}$

(iii) $o \in O$ é um elemento do conjunto de contextos.

O segundo conjunto é denominado conjunto de retornos pendentes e tem o significado equivalente ao de ativações pendentes já comentado e definido. Entretanto, cada retorno pendente é completamente caracterizado por apenas dois elementos: a cor da ficha de retorno e o contexto cuja terminação gerou o retorno pendente.

Definição 4.3 - *Retornos Pendentes*

Um retorno pendente é uma dupla <d,o> onde:

(i) $d \in D$ é uma cor do conjunto união de todas as cores de todos os domínios

$D = \{ c \mid c \in C \wedge C \in G_i.\Sigma \wedge G_i \in G \}$

(ii) $o \in O$ é um elemento do conjunto de contextos.

Entretanto, um retorno pendente não detém informação suficiente para permitir um retorno. A fim de que se possa gerar um retorno efetivamente é preciso saber quais são o contexto e o invocador responsáveis pela invocação. Nesse sentido, definimos o elemento denominado associação de contextos, que será utilizada para exprimir a relação entre o contexto cliente e o contexto servidor.

Definição 4.4 - *Associação de Contextos*

Uma associação de contextos é a tripla $<o_1,o_2,isp>$ onde:

(i) $o_1,o_2 \in O$ são elementos do conjunto de contextos;

(ii) isp é um invocador pertencente a um dos módulos do sistema

Finalmente definimos o conceito de um ambiente em G-CPN. Um ambiente é uma estrutura que integra um conjunto de ativações pendentes, um conjunto de retornos pendentes e um conjunto de associações de contextos.

Definição 4.5 - *Ambiente*

Um ambiente é uma tripla <PA, PR, CA> onde:

(i) PA é um conjunto finito de ativações pendentes;

(ii) PR é um conjunto finito de retornos pendentes;

(iii) CA é um conjunto finito de associações de contextos;

Informalmente, podemos dizer que o estado de um sistema, é dado pelo conjunto das marcações dos módulos do sistema, o conjunto de invocações pendentes, o conjunto de retornos pendentes, e o conjunto de associações de contextos. Para simplificar, podemos dizer que um estado de um sistema G-CPN é plenamente caracterizado pelas marcações dos módulos e um ambiente. Além disso, acrescemos ao estado do sistema um conjunto de identificadores de contextos não utilizados. O conjunto de contextos não utilizados será útil para gerar identificadores de contextos quando ativações ocorrerem. Vejamos como a definição de estado de um sistema é formalizado:

Definição 4.6 - *Estado de um Sistema G-CPN*

Um estado do Sistema G-CPN é uma tripla <ENV, MM, NC>, onde:

(i) ENV é um ambiente;

(ii) MM é uma função de marcações de módulos

$MM: GCPN \rightarrow G_1.M \cup G_2.M \cup ... \cup G_i.M \cup ... \cup G_n.M$

$\forall j, 1 \leq j \leq n: MM(G_j) = G_j.M$

(iii) NC é um conjunto de contextos denominado *novos contextos*

$NC \subseteq O$

Definidos os conceitos básicos, podemos passar, então, às definições de eventos que refletem alterações no sistema.

Os eventos que podem ocorrer em um sistema G-CPN podem ser classificados em duas classes: os que só alteram as marcações dos módulos sem alterar o ambiente; e os que alteram as marcações dos módulos e também alteram o ambiente. Os eventos do primeiro tipo são denominados passos. Cada passo do sistema é um conjunto formado por um passo de cada módulo do sistema. Os eventos que alteram o ambiente do sistema são os relacionados aos mecanismos de comunicação inter-módulos: invocações, ativações, terminações e retornos.

Em termos formais, não definimos simplesmente um passo do sistema. A definição seguinte estabelece apenas o que é um passo habilitado. Perceba que um passo ao ocorrer altera exclusivamente as marcações de cada módulo, deixando o restante do sistema inalterado.

Definição 4.7 - *Passo num Sistema G-CPN*

Um passo YS habilitado de um Sistema G-CPN é um conjunto formado por um passo habilitado de cada módulo do sistema. Um passo habilitado pode ocorrer. Se ocorre, então o sistema muda do estado E_1 para o estado E_2 determinado por:

(i) Seja $E_1 = \{ ENV_1, \{ G_1.M_1, G_2.M_1, ... , G_i.M_1, ... , G_n.M_1 \}, NC_1 \}$

(ii) e o passo habilitado $YS = \{ G_1.Y, G_2.Y, ... , G_i.Y, ... , G_n.Y \}$

(iii) Se YS ocorre então o novo estado do sistema será

(iv) $E_2 = \{ ENV_1, \{ G_1.M_2, G_2.M_2, ... , G_i.M_2, ... , G_n.M_2 \}, NC_1 \}$

(v) onde:

$\forall j$, $1 \leq j \leq n$: $G_j.M_2$ é a marcação do módulo G_j após a ocorrência de $G_j.Y$

As definições dos eventos relacionados à comunicação entre os módulos G-CPN são estabelecidas com base nos eventos correspondentes nos módulos. Entretanto, a ocorrência desses eventos deve determinar como são modificadas as marcações dos módulos, bem como o estado do ambiente: ativações e retornos pendentes e as associações de contextos.

Do ponto de vista dos módulos, as invocações, ativações, terminações e retornos alteram as marcações de forma idêntica aos eventos correspondentes para Módulos G-CPN. A diferença destas definições é que elas incluem as condições em que ocorrem, bem como determinam o estado a que levam o sistema.

Definição 4.8 - *Invocação num Sistema G-CPN*

Uma invocação $isp_j \in G_i.ISP$ é dita habilitada no sistema G-CPN no estado

(i) $E = \{ ENV, \{ G_1.M, G_2.M, \ldots, G_i.M, \ldots, G_n.M \}, NC \}$

se e somente se existe uma ficha <p,c,o> que habilite a invocação de isp_j em G_i no estado $G_i.M$. Neste caso, diz-se que a ficha <p,c,o> habilita a invocação isp_j no sistema.

Definição 4.9 - *Ocorrência de uma Invocação num Sistema G-CPN*

A invocação do $isp_j \in G_i.ISP$ habilitado no sistema para o estado E_1 pode ocorrer. Se ocorre, então o sistema muda do estado E_1 para o estado E_2, dado por:

(i) Seja $E_1 = \{ <PA_1, PR_1, CA_1>, \{ G_1.M_1, G_2.M_1, \ldots, G_i.M_1, \ldots, G_n.M_1 \}, NC_1 \}$ e

(ii) <p,c,o> o elemento de ficha que habilita a invocação

(iii) $E_2 = \{ <PA_2, PR_1, CA_2>, \{ G_1.M_1, G_2.M_1, \ldots, G_i.M_2, \ldots, G_n.M_1 \}, NC_2 \}$

Onde:

$PA_2 = PA_1 \cup \{<G_i.OBJ(isp_j),c,nc>\}$

$nc \in NC_1$

$CA_2 = CA_1 \cup \{ o, nc \}$

$G_i.M_2 = G_i.M_1 - <p,c,o>$

$NC_2 = NC_2 - \{nc\}$

Definição 4.10 - *Ativação num Sistema G-CPN*

A ativação do método $G_i.m \in G_i.MT$ é dita habilitada no sistema G-CPN no estado E = { <PA, PR, CA>, MM, NC } se e somente se existe uma ativação pendente <m,d,o>∈PA, onde d∈C(AP(m)).

Definição 4.11 - *Ocorrência de uma Ativação num Sistema G-CPN*

Se a ativação do método $G_i.m$ está habilitada no sistema G-CPN no estado E1, então a ativação pode ocorrer. Se ocorre, o sistema muda do estado E_1 para o estado E_2, dado por:

(i) Seja E1 = { $<PA_1, PR_1, CA_1>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_1, \ldots, G_n.M_1$ }, NC_1 }

(ii) Seja <s,d,o> a ativação pendente que habilita a ativação do método

(iii) E2 = { $<PA_2, PR_1, CA_1>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_2, \ldots, G_n.M_1$ }, NC_1 }

Onde:

$PA_2 = PA_1 - <p,c,o>$

$Gi.M_2 = Gi.M_1 + <p,c,o>$

Definição 4.12 - *Terminação num Sistema G-CPN*

Uma terminação do método $G_i.m \in G_i.MT$ é dita habilitada no sistema G-CPN no estado

(i) E = { ENV, { $G_1.M, G_2.M, \ldots, G_i.M, \ldots, G_n.M$ }, NC }

se e somente se existe uma ficha <p,c,o> que habilite a terminação de m em G_i no estado $G_i.M$. Neste caso, diz-se que a ficha <p,c,o> habilita a terminação de $G_i.m$ no sistema.

Definição 4.13 - *Ocorrência de uma Terminação num Sistema G-CPN*

A terminação do método $G_i.m$ habilitado no sistema para o estado E_1 pode ocorrer. Se ocorre, então o sistema muda do estado E_1 para o estado E_2, dado por:

(i) Seja E1 = { $<PA_1, PR_1, CA_1>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_1, \ldots, G_n.M_1$ }, NC_1 }

(ii) e <p,c,o> o elemento de ficha que habilita a terminação

(iii) E2 = { $<PA_1, PR_2, CA_1>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_2, \ldots, G_n.M_1$ }, NC_1 }

Onde:

$PR_2 = PR_1 \cup \{<c,o>\}$

$G_i.M_2 = G_i.M_1 - <p,c,o>$

Definição 4.14 - *Retorno num Sistema G-CPN*

O retorno do invocador $G_i.isp_j \in G_i.ISP$ é dito habilitado no sistema G-CPN no estado E = { <PA, PR, CA>, MM, NC } se e somente se existe um retorno pendente $<d,o> \in PR$, e uma associação de contextos $<o_1, o_2> \in CA$, tais que $d \in G_i.C(G_i.LP(G_i.isp_j))$ e $o_1=o_2$.

Definição 4.15 - *Ocorrência de um Retorno num Sistema G-CPN*

Se o retorno do invocador $G_i.isp_j$ está habilitado no sistema G-CPN no estado E_1, então o retorno pode ocorrer. Se ocorre, o sistema muda do estado E_1 para o estado E_2, dado por:

(i) Seja E_1 = { $<PA_1, PR_1, CA_1>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_1, \ldots, G_n.M_1$ }, NC_1 }

(ii) Sejam $<d,o_2>$ e $<o_1, o_2>$ o retorno pendente e a associação de contextos que habilitam o retorno do invocador $G_i.isp_j$

(iii) E2 = { $<PA_1, PR_2, CA_2>$, { $G_1.M_1, G_2.M_1, \ldots, G_i.M_2, \ldots, G_n.M_1$ }, NC_1 }

Onde:

$PR_2 = PR_1 - <d,o_2>$

$CA_2 = CA_1 - <o_1,o_2>$

$G_i.M_2 = G_i.M_1 + <G_i.LP(G_i.isp_j),d,o_1>$

4.3 Relação entre CPN e G-CPN

Nesta Seção investigamos a relação existente entre Sistemas G-CPN e Redes CPN. O propósito deste estudo é óbvio: capturar os conceitos e os poderosos métodos de análise matemática já bem consolidados das redes CPN formando uma base para sua adaptação para Sistemas G-CPN. Além disso, todo o arsenal de ferramentas de software disponíveis para redes CPN torna-se útil para Sistemas G-CPN.

Embora, seja extremamente importante proceder com o máximo de formalismo ao comparar e apontar semelhanças entre diferentes modelos de computação, trataremos o assunto de um ponto de vista intuitivo.

Na realidade, apresentaremos uma técnica sistemática de obtenção de uma rede CPN pura a partir de um Sistema G-CPN. A existência desta técnica de transformação é de extrema importância para simular o comportamento de Sistemas G-CPN, dado que ainda não existem ferramentas de software que permitam a edição e simulação dos sistemas.

Apresentaremos um conjunto de argumentos que leva a perceber que a técnica apresentada para construir uma rede CPN a partir de um Sistema G-CPN é, além de computável, preservadora das características comportamentais do sistema. Desta forma, estabelecemos uma definição informal para a equivalência entre os modelos. Os termos "equivalente" e "equivalência" serão usados no suceder desta seção, com esse sentido.

Intuitivamente percebemos que o modelo G-CPN não agrega qualquer poder computacional às redes CPN. Portanto, deve ser possível construir uma rede CPN que se comporte de forma equivalente a qualquer sistema G-CPN. Começaremos a busca pela rede equivalente a um Módulo G-CPN.

Claramente a estrutura interna do módulo poderá ser simulada por uma rede CPN bastante parecida. A diferença principal reside no fato de que as fichas de G-CPN têm na sua estrutura um elemento a mais: o contexto. A solução desse problema é bastante simples: fazer com que todas as fichas da CP-Net equivalente transportem explicitamente o valor do contexto a que pertencem. Assim, os conjuntos de cores também devem ser alterados. Vejamos como exemplo a definição dos conjuntos de cores para o exemplo do módulo produtor do sistema apresentado na Figura 3.4:

O nó de declaração define as seguintes cores:

(i) `color INT = int;`

(ii) `color PRIMITIVE = with ask | ack | nak;`

(iii) `color ELEMENT = with item;`

(iv) `color STATE = with free | wait | consuming;`

Em CPN poderíamos definir:

(i) `color CONTEXT = (* algum conjunto de contextos *);`

(ii) `color INT = int;`

(iii) `color INT_ = product INT*CONTEXT;`

(iv) `color PRIMITIVE = with ask | ack | nak;`

(v) `color PRIMITIVE_ = product PRIMITIVE*CONTEXT;`

(vi) `color ELEMENT = with item;`

(vii) `color ELEMENT_ = product ELEMENT*CONTEXT;`

(viii) `color STATE = with free | wait | consuming;`

(ix) `color STATE_ = product STATE*CONTEXT;`

O conjunto de cores definido pela linha (i) é usada para definir em CPN-ML a natureza do conjunto de contextos. As linhas (ii), (iv), (vii) e (viii) são idênticas às usadas no sistema G-CPN. As linhas (iii), (iv), (v) e (vi) são as versões "contextualizadas" que serão usadas na CP-Net.

Os lugares teriam como conjuntos de cores as novas cores definidas, as cores originais são mantidas na definição para auxiliar na construção das cores auxiliares que permitirão a completa simulação. Desta forma, a dinâmica da rede CPN pode manter as fichas referentes a diversas invocações concorrentes simultaneamente sem confundí-las, permitindo que haja equivalência entre passos de módulos com passos da rede. É claro que variáveis, funções e inscrições também devem ser re-escritas de acordo. Finalmente, como os ISP's são na realidade mapeados em dois lugares na estrutura interna, não há maiores problemas para efetuar sua transformação.

Passemos agora à busca da CP-Net equivalente a um sistema como um todo. Dado que todos os módulos do sistema podem ser transformados como explicado anteriormente, o problema remanescente se reduz a encontrar uma representação para a estrutura do ambiente.

A solução é criar um lugar para cada componente do ambiente. Assim cada um dos conjuntos de ativações pendentes, retornos pendentes e associações de contextos será representado por um lugar específico na estrutura. Novamente, toda a complexidade do problema se concentra na determinação dos conjuntos de cores associados a esses lugares.

Como mencionado na formalização, uma ativação pendente é uma estrutura formada por um serviço, uma cor e um contexto. A cor é a mensagem propriamente dita. E como uma ativação pode ter sido gerada por qualquer invocador no sistema, a cor pode pertencer a qualquer conjunto de cores do sistema. Em termos formais, denotamos esse conjunto de cores por D. Entretanto, a versão em CPN-ML desse conjunto será chamada de SIGMA.

Ao invés de unir todos os conjuntos de cores do sistema, o conjunto de cores SIGMA será definido como a união dos conjuntos de cores associados a lugares de ativação dos métodos. Para representar os serviços presentes no sistema também usaremos um conjunto de cores. O conjunto de serviços é um um conjunto enumerado simples. Dessa

forma podemos definir um conjunto de cores especificamente para o lugar que representará ativações pendentes. Veja o exemplo a seguir para o sistema produtor-consumidor:

(i) `color CONTEXT = (*algum conjunto de contextos *)`

(ii) `(* demais definições *)`

(iii)
```
color SIGMA = union primitive:PRIMITIVE +
element:ELEMENT + int:INT;
```

(iv)
```
color SERVICES = with P_produce | C_check_state |
C_consume;
```

(v) `color PA = product SERVICES*SIGMA*CONTEXT;`

Os termos *primitive*, *element* e *int* incluídos antes dos nomes dos conjuntos de cores são exigidos pela sintaxe de CPN-ML [Jensen, 1992]. Perceba que na definição deste conjunto, não é preciso utilizar as novas versões dos conjuntos de cores, por se tratar apenas de uma cor ue transporta uma mensagem.

Um esquema de transformação semelhante é usado para representar os conjuntos de retornos pendentes e de associações de contextos do ambiente.

Resta-nos agora associar os eventos de invocação, ativação, terminação e retornos ao sistema. Isso pode ser feito agregando uma transição para cada tipo de evento, de forma que seus arcos de entrada verifiquem a existência de condições que habilitem o evento, e seus arcos de saída efetuem as mudanças necessárias nos módulos e no ambiente. A fim de simplificar, utilizaremos uma transição para cada lugar de comunicação. Isto é, agregaremos transições específicas para gerar os eventos associados a cada lugar superior, inferior, de ativação e de terminação do sistema.

As transições associadas aos lugares superiores dos invocadores representam a ocorrência de invocações, as transições associadas a lugares inferiores dos invocadores representam retornos. As transições de invocação devem detectar fichas nos lugares superiores e caso ocorram devem retirar tais fichas e gerar novas fichas nos lugares que representam as ativações pendentes, e as associações de contextos. As transições de retorno detectam fichas no lugar correspondente aos retornos pendentes e associações de contextos e geram fichas nos lugares inferiores dos invocadores.

Para modelar os eventos de ativações e terminações procede-se de forma análoga.

O último elemento de que ainda não falamos é o conjunto de contextos que também faz parte do estado do sistema. O conjunto de contextos será modelado em CPN simplesmente através do conjunto dos naturais não negativos. Além disso, é necessária a existência de um lugar na estrutura de equivalência que denote o conjunto de contextos não utilizados. Entretanto, como não podemos colocar um conjunto infinito de fichas em um lugar, utilizaremos um esquema bastante simples. O lugar manterá apenas um valor de contexto não-utilizado, e as inscrições dos arcos de saída e entrada devem garantir o depósito de uma ficha que caracterize um novo contexto a cada evento.

A título de exemplo, vejamos na Figura 4.1 a rede CPN equivalente, nos termos aqui especificados, ao sistema produtor consumidor modelado através de G-CPN.

Como foi mencionado no início desta seção, a abordagem utilizada aqui não se baseia em fatos formais. Entretanto, devido à falta de ferramentas de software desenvolvidas, a semântica associada aos Sistemas G-CPN é a obtida pela sua transformação em uma CP-Net. O maior problema relacionado com esta técnica é que do ponto de vista formal seria preciso proceder da seguinte maneira:

(i) Estabelecer critérios de equivalência entre uma rede CPN e um Sistema G-CPN: em geral o que se deseja é que dois modelos sejam considerados equivalentes se apresentam o mesmo comportamento.

(ii) Estabelecer uma técnica de obtenção de uma rede CPN a partir de um dado Sistema G-CPN: a técnica deve ser algorítimica para que seja realmente interessante.

(iii) Provar que a aplicação da técnica a qualquer Sistema G-CPN resulta numa rede CPN que é equivalente segundo os critérios previamente estabelecidos.

Uma outra forma de estabelecer formalmente a relação entre os modelos, seria definir seu comportamento (definições semântica) em termos de redes CPN. Isto seria perfeitamente válido, dado que as redes CPN são construtos essencialmente matemáticos. Entretanto, isto tornaria a semântica do novo modelo dependente da semântica de CPN.

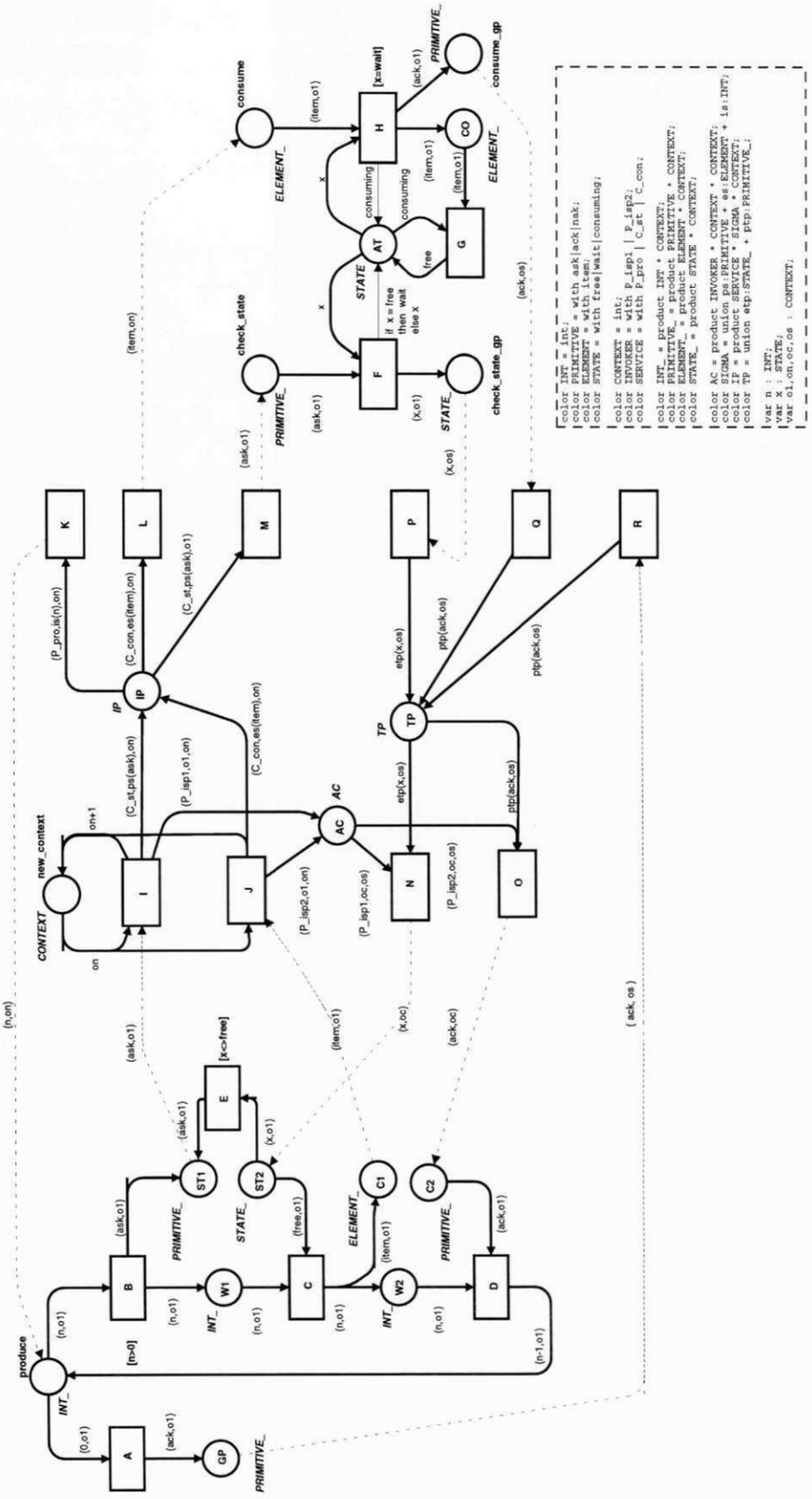

Figura 4.1 - CP-Net equivalente ao Sistema G-CPN Produtor/Consumidor

5. UM AMBIENTE COOPERATIVO DE EDIÇÃO

A fim de experimentar e validar o novo modelo de descrição e especificação de sistemas distribuídos de software torna-se indispensável aplicá-lo aos mais diversos domínios de problemas reais que encontrem soluções através de sistemas distribuídos.

Um outro problema relacionado à consolidação de qualquer paradigma matemático de modelagem, é a falta de ferramentas de software para sua experimentação. É preciso, portanto, desenvolver um ambiente apropriado para especificar, projetar, simular, validar e auxiliar a implementação de sistemas de software através da ferramenta proposta. Um interessante problema se apresenta de imediato nesta fase de trabalho: deve-se proceder a especificação das próprias ferramentas de software através do formalismo proposto? Parece-nos interessante, a este ponto do desenvolvimento do modelo, aplicar esforços na descrição e especificação de sistemas que devem realmente ser construídos e servirão de apoio ao futuro desenvolvimento do próprio modelo.

Como forma de experimentação e validação do Modelo G-CPN no desenvolvimento de sistemas de software distribuídos e relativamente complexos, apresentamos neste Capítulo o desenvolvimento da descrição e especificação de um subconjunto de componentes de um ambiente cooperativo de edição de diagramas hierárquicos de acordo com o proposto como caso de estudo para o *Segundo Workshop em Programação Orientada a Objetos e Modelos de Concorrência*. Devido ao fato de que Sistemas G-CPN podem ser representados através de diagramas hierárquicos, o desenvolvimento deste ambiente é base para a construção de ferramentas de software relativas ao próprio modelo.

O restante do Capítulo encontra-se estruturado do seguinte modo: na Seção 5.1 introduzimos o problema da edição cooperativa, bem como alguns conceitos importantes associados. A Seção 5.2 descrevemos uma arquitetura baseada em agentes como proposta de solução. Na Seção 5.3 apresentamos o procedimento de modelagem de alguns dos componentes identificados através de G-CPN. Na Seção 5.4 finalizamos o Capítulo 4 expondo resultados e conclusões obtidas durante o processo de modelagem.

5.1 Introdução

Um aspecto importante, cuja presença vem sendo requerida por membros de diversas equipes de projeto em ambientes de modelagem e desenvolvimento computacionais, nos últimos anos, é a capacidade de suporte ao desenvolvimento cooperativo e concorrente em ferramentas auxiliadas por computador [Dewan, 1993].

Em geral, os ambientes computacionais de projeto e desenvolvimento permitem a sua utilização por apenas um usuário em um dado instante. Claramente, é um problema para as equipes de projeto munidas de infra-estruturas baseadas em redes de comunicação de computadores. O problema é que esses ambientes não fazem uso otimizado dos recursos computacionais disponíveis. A solução geralmente adotada nesses casos, é a partição do projeto em diversos sub-projetos a serem trabalhados separada e simultaneamente pelos diversos membros da equipe. O que ocorre nesse caso, é que a re-integração do projeto torna-se extremamente complexa se cada membro não estiver completa e constantemente integrado ao trabalho de todos os demais membros. Decorrendo na constante necessidade de reuniões com esse objetivo. Isso se dá pela inexistência de ambientes de desenvolvimento computacional adequado ao trabalho cooperativo e concorrente.

O problema mencionado é agravado quando se trata de ferramentas de auxílio ao desenvolvimento de sistemas complexos de software. A construção de sistemas de software de grande porte não é suportada apropriadamente pelas ferramentas atuais. O espantoso crescimento dos já consideráveis gastos na gerência de projetos de sistemas de software podem ser consideravelmente reduzidos se ambientes que suportem o desenvolvimento cooperativo concorrente estiverem disponíveis [Forte, 1992].

Duas áreas de concentração de pesquisa percebem-se no tocante ao assunto. A disciplina denominada CSCW (*Computer Supported Cooperative Work*) [Dewan, 1993] integra esforços na compreensão das atividades de cooperação e concorrência de grupos de indivíduos interagindo através de recursos de computação. O trabalho é voltado para a natureza do comportamento dos indivíduos diante do grupo, bem como na busca por padrões de comportamento social que possam ser úteis na adequação dos recursos de computação. Os pesquisadores envolvidos não são necessariamente cientistas da computação, a comunidade de CSCW reúne tamém psicólogos, sociólogos, lingüístas entre outros. A segunda área de pesquisa, de caráter tecnológico, concentra-se na busca

e aplicação de métodos e técnicas de desenvolvimento baseados em tecnologias de computação e comunicação para a construção de sistemas que suportem grupos ou equipes de pessoas interagindo a fim de realizar um trabalho comum. Utiliza-se o termo *groupware* [Ellis, 1991] para denominar a tecnologia e os produtos desenvolvidos, embora o nome se confunda com a própria comunidade.

A despeito do que foi citado, é importante mencionar o fato de que ainda há diversas controvérsias a respeito da definição precisa dos termos *CSCW* e *Groupware*, por se tratar de uma área extremamente nova. Contudo, é fácil perceber que as duas áreas têm muito em comum, e portanto, se influenciam mutuamente. O termo utilizado para denotar os sistemas de software que incorporem características de cooperação de grupo variam: ora *sistemas de CSCW* (*CSCW Systems*) [Navarro, 1993] ora *groupwares* [Simon, 1996].

Diversos trabalhos vêm procurando classificar Sistemas de CSCW ou *groupwares*. Em [Navarro, 1993] são descritas duas características principais que podem ser usadas como critérios para classificação: a forma de interação suportada e a natureza geográfica do sistema. Em [Williams, 1994] foram identificadas quatro grandes classes de groupwares: salas de encontro (*Meeting Rooms*), conferência por computador (*Computer Conferencing*), sistemas baseados em mensagens (*Message-based Systems*) e sistemas de co-autoria (*Co-authoring*). Segundo esta classificação, um sistema de edição cooperativa pode ser caracterizado como um sistema CSCW de co-autoria.

Dentre as questões relevantes quanto ao desenvolvimento de sistemas cooperativos, o controle de concorrência tem especial importância. A falta de cuidado ao estabelecer os mecanismos pelo qual será feito, pode levar a incosistências no documento que está sendo editado, devido à perda ou ao tratamento fora de ordem de mensagens. Além disso, inconsistências podem levar o usuário do sistema a um modelo mental falso daquilo em que está trabalhando. Em [Greenberg, 1994] atenta-se para o fato de que ambientes cooperativos de edição não têm a mesma natureza dos sistemas de software distribuídos convencionais devido à presença forte do fator humano. Ressalta-se, ainda, a importância da escolha correta dos mecanismos de controle de concorrência e seus efeitos na interface do usuário. A escolha apropriada, segundo o autor, não deve basear-se exclusivamente em questões de caráter técnico ou de performance, mas acima de tudo em aspectos comportamentais humanos.

Uma outra questão de elevada importância é a granularidade do elemento editado. Diversos e diferentes problemas emergem ao extremar-se o nível de granularidade em qualquer sentido, especialmente em se trantando de mecanismos de controle de concorrência através de travas (*locks*). Grãos grandes implicam em um número menor de requisições de travas, em compensação, menor nível de concorrência será permitido. Grãos pequenos implicam exatamente no oposto. As opções são escolher um valor intermediário fixo apropriado à natureza do problema ou permitir diversos níveis de granularidade durante as atividades. Obviamente, não há uma solução única para o problema. A escolha apropriada depende fortemente do domínio do problema do caso.

5.2 O Ambiente Cooperativo de Edição

5.2.1 Descrição do Ambiente e Seus Componentes

Nesta seção nós apresentamos a especificação informal ou os requisitos usados como diretrizes na construção deste ambiente cooperativo. Os requisitos da especificação que serão apresentados em seguida foram baseados em um estudo de caso proposto por Bastide et al. para o *2nd workshop on Object-Oriented Programming and Models of Concurrency*. Diversas modificações foram efetuadas com o objetivo de atingir o nosso principal objetivo: a especificação de um ambiente cooperativo de edição de Sistemas G-CPN.

Basicamente, o sistema deve prover os seguintes requisitos: usuários cadastrados devem poder entrar ou deixar uma sessão de edição quando desejarem; membros que estejam participando de uma sessão devem ser visíveis para todos bem como seus privilégios sobre os elementos; os elementos podem estar livres ou pertencer a um usuário. Por se tratar de um sistema de co-autoria de sistemas complexos, os requisitos relaxam a exigência por características WYSIWIS (*What You See Is What I See*), permitindo uma visão particularizada do diagrama a cada usuário.

As permissões e privilégios sobre os elementos devem ser suficientes para permitir que elementos sejam: compartilhados ou restritos, para permitir o controle sobre os direitos de alteração de seus atributos; visíveis ou invisíveis, para permitir o controle de acesso de leitura sobre partes do sistema; e livres ou possuídos por algum usuário, para

determinar os direitos de alteração do estado do elemento, bem como o direito de eliminação.

Nossa abordagem será baseada na definição de um sistema multi-agentes. Diferentes agentes devem interagir através de uma rede de comunicação para prover um serviço de cooperação para uma certa classe de aplicações. Cada agente é definido como um conjunto de objetos de software, e a comunicação entre eles deve ser garantida por um protocolo de alto nível bem definido.

O sistema é composto por um agente coordenador, um agente gerente e uma classe de agentes usuários. A classe de agentes usuários é o conjunto de agentes responsáveis por prover um ponto de acesso ao ambiente de cooperação. Cada agente usuário pode ser considerado como um editor acrescido de características cooperativas. O agente gerente permite a inclusão e eliminação de usuários no sistema. Além disso, o gerente deve possuir permissões para a resolução de eventuais situações de conflito. O agente coordenador, é um agente responsável pela manutenção da consistência e a correta interação entre os usuários.

O Agente Usuário dever pode efetuar as seguintes ações: entrar em uma sessão de edição; sair de uma sessão de edição; criar elementos; alterar características de elementos sobre os quais detenha permissão; requisitar permissões sobre elementos; dispensar permissões; e eliminar elementos por ele possuídos.

O Agente Gerente deve poder efetuar as seguintes ações: registrar novos usuários; remover usuários registrados; modificar quaisquer permissões sobre os elementos do diagrama hierárquico.

Por não se tratar de um agente de interação humana, o Coordenador provê serviços aos demais agentes do sistema. O Coordenador deve deter todas as informações do diagrama hierárquico em edição, bem como informações sobre o andamento da sessão. Entre os serviços providos pelo Coordenador devem constar: criação, acesso, alteração e eliminação de elementos; notificações sobre mudanças ocorridas em elementos do diagrama; notificações sobre entrada e saída de usuários nas sessões; informação sobre as ações de cada usuário.

5.2.2 Arquitetura dos Agentes

Cada um dos agentes mencionados é internamente concebido como um sub-sistema multi-camadas, onde cada camada provê um serviço refinado para a sua camada superior.

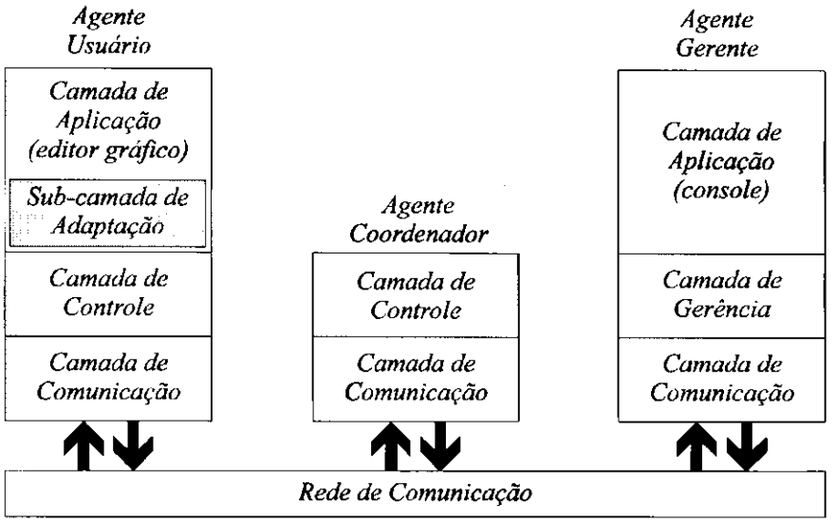

Figura 5.1 - Arquitetura proposta para o Ambiente

Para desenvolver o ambiente cooperativo definimos a arquitetura mostrada na Figura 5.1. A camada superior, dita camada de aplicação, é um editor gráfico de diagramas hierárquicos (p. ex: Sistemas G-CPN) ou um console do sistema a ser usado apenas pelo gerente. A camada de aplicação apresenta uma sub-camada denominada camada de adaptação, responsável pelo mapeamento entre a representação léxica usada pela aplicação e a representação em termos de elementos do diagrama hierárquico. A presença desta sub-camada na arquitetura, permite que a aplicação propriamente dita, seja voltada para a natureza dos objetos em termos do domínio do problema. A segunda camada é denominada camada de controle e está relacionada com o correto tratamento de todos os tipos de objetos em uma sessão de edição. A camada de controle é também chamada de camada de cooperação, dado que ela executa a base necessária de CSCW. No agente gerente a camada de cooperação é denominada de camada de gerência, por apresentar serviços diferenciados. A terceira camada da arquitetura é a camada de comunicação que é responsável pelo correto tratamento de mensagens trocadas entre os usuários, através do usos de serviços da rede de comunicação e do sistema operacional.

Esta abordagem permite o uso de um editor gráfico simples sobre a pilha de serviços cooperativos. Tudo o que o editor gráfico (camadas de aplicação) deve fazer é garantir que a camada de controle seja questionada/informada sobre quaisquer ações efetuadas sobre os elementos gráficos. Logo, se um novo editor deve ser desenvolvido, com características adicionais, poucas modificações são necessárias nas camadas de comunicação e controle.

Camada de Aplicação

Como dito anteriormente, no caso do editor cooperativo de diagramas hierárquicos, a camada de aplicação é um editor gráfico com a habilidade de acessar serviços oferecidos pela camada de controle. O editor gráfico deve oferecer funcionalidades semelhantes à de um editor *stand-alone*, embora seu comportamento seja orientado pela camada de controle.

A função da camada de aplicação do agente gerente é prover mecanismos de controle sobre os usuários pertencentes ao grupo de edição. O console do sistema é a aplicação que acessa os serviços oferecidos pela camada de gerência. Inclusões e exclusões de usuários de um grupo de edição e alterações de permissões forçadas, devem ser efetuadas pelo gerente através do console. Um serviço opcional interessante que poderia ser incluído no agente gerente seria a manutenção de informações sobre critérios para solução de conflitos.

O editor gráfico deve ser implementado sem considerar travas prévias (*locks*) sobre elementos. Neste caso, ao executar qualquer ação sobre os elementos gráficos é necessário apenas requisitar à camada de controle a visualização sobre o referido elemento. Se o usuário tem a permissão necessária, a camada de controle poderá requerer o elemento explicitamente com o nível de acesso desejado. Neste caso, fica a critério da aplicação o uso de um tratamento otimista e efetuar ações sem ter acesso restrito ao elemento, ou um tratamento pessimista, requisitando e garantindo uma visão com acesso restrito a fim de garantir a consistência das alterações. As características de um objeto devem estabelecer também a possibilidade de que o objeto seja apenas visualizado sem o direito de alterações. Isso permite que usuários vejam, embora sem interferir, as ações de outros usuários.

Camada de Controle

Quando operações de acesso aos elementos devem ser executadas pelo editor em uma sessão de edição cooperativa, a camada de controle deve prover o serviço à aplicação. Portanto, a camada de controle é a representação abstrata local do sistema como um todo em cada *site* de operação.

A camada de controle mantém informações atualizadas sobre usuários e elementos da sessão. Baseada nestas informações, a camada mantém uma visão do estado global do sistema e atende às requisições de serviços da aplicação. A camada de cooperação é capaz de resolver completamente os serviços para os quais os dados da visão local forem suficientes ou se as informações só devem refletir o estado local. Entretanto, se as informações não se encontram disponíveis ou se for preciso comunicar outros indivíduos do grupo a respeito de alguma mudança local, a camada de cooperação requisita os serviços oferecidos pelo agente coordenador.

Camada de Comunicação

A camada de comunicação de todos os agentes provê serviços simples de comunicação. Nenhum serviço especial de comunicação de grupo deve ser requerido a fim de simplificar a sua construção. Além disso, devido à particularização das visões do diagrama hierárquico por parte dos usuários, as mensagens nem precisam ser enviadas a todos os participantes da sessão, o que implica na diminuição do número de mensagens enviadas e conseqüentemente em economia de banda passante.

5.3 Modelagem G-CPN do Ambiente

Nesta Seção procederemos a especificação e a modelagem do ambiente cooperativo descrito na Seção 5.2 através de G-CPN.

5.3.1 Considerações Iniciais

Dado que a idéia inicial é especificar um Sistema G-CPN que modele o ambiente como um todo, nada mais razoável que determinar que cada agente será representado no sistema por um conjunto de módulos. Entretanto, é preciso determinar aqui, a que nível

de detalhamento a especificação se dará, e como cada camada será representada no modelo final a ser obtido.

Dado que nenhum serviço adicional de comunicação de grupo ou de ordenação de mensagens será requerido, a utilização dos mecanismos de invocação de métodos e de comunicação inter-módulos providos pelo Modelo G-CPN será suficiente para caracterizar e simular de forma abstrata os serviços de entrega de mensagens da camada de comunicação.

Esta forma de modelar a camada de comunicação a torna "invisível" no modelo, o que é interessante de certo ponto de vista. Contudo, se houvesse interesse em determinar com maior precisão as entidades de comunicação e seu funcionamento, essa solução seria inadequada. Uma solução mais interessante, neste caso, seria incluir módulos G-CPN que modelassem especificamente entidades presentes na camada de comunicação. Em decorrência, entretanto, seria necessário garantir que quando as entidades de camadas superiores desejassem se reportar a entidades remotas, elas na realidade requisitassem serviços de entidades de comunicação locais.

No presente documento, o interesse maior está nas atividades de cooperação e concorrência relativas à camada de cooperação do ambiente. Portanto, a camada de comunicação será abstraída como já foi indicado, permitindo que os diversos módulos remotos enviarão mensagens reportando-se diretamente a eles.

A camada de aplicação, como já foi dito, é um aplicativo voltado para o domínio do problema, capaz de interagir da forma apropriada com a camada de cooperação através da interface aplicação-controle. Não faz parte do nosso interesse, modelar uma aplicação em si, estando portanto fora de nosso escopo.

É através da interface aplicação-controle que pedidos de serviços são feitos e respostas são devolvidas à aplicação. Em termos de G-CPN, a interface será representada por um conjunto de serviços de algum módulo da camada de controle.

Portanto, dadas as considerações acima expostas e que o agente coordenador não apresenta camada de aplicação, concentraremos os esforços de modelagem nos seguintes componentes: a interface aplicação-controle e a camada de controle dos agentes coordenador e usuário.

5.3.2 Formalizando os Componentes

Devido ao fato de que G-CPN é essencialmente um modelo matemático, o ato de modelar sistemas através de G-CPN é essencialmente formalização. Conseqüentemente, é preciso pensar nas estruturas de representação dos diversos componentes. No que segue, apresentamos a estrutura formal de representação do principal objeto do ambiente cooperativo de edição: o diagrama hierárquico.

Um diagrama hierárquico - DH é um conjunto de elementos que se relacionam hierarquicamente. Cada elemento é uma estrutura com atributos e valores próprios que determinam suas particularidades ou características. As relações hierárquicas entre os objetos podem ter semântica diferente de acordo com o domínio do problema para o qual se usa o DH.

Assim, por exemplo, um DH pode ser a representação um texto hierarquizado. Onde os elementos são seções, subseções, parágrafos, palavras, caracteres, figuras, etc. As relações entre esses elementos devem determinar que o texto como um todo é um conjunto de seções que por sua vez são conjuntos de subseções e parágrafos e assim por diante.

Neste trabalho, escolhemos representar diagramas hierárquicos através de árvores. O nó raiz da árvore representa o DH como um todo. Os nós um nível abaixo da raiz, representam os elementos de que é composto o DH, e assim por diante. Por ser um elemento do DH, cada nó da árvore tem seu próprio conjunto de atributos. Além disso, cada elemento deve ser unicamente identificável no DH. Com o objetivo de registrar as relações de hierarquia, cada elemento guarda o identificador do elemento imediatamente acima na relação hierárquica denominado elemento "pai". Nesta abordagem, cada elemento contém também um conjunto de identificadores imediatamente abaixo na hierarquia denominados elementos "filhos", embora isso não seja necessário e inclua redundância na representação, entretanto, simplifica diversas operações no DH.

Portanto, um elemento é uma estrutura formada pelos seguintes componentes:

(i) ***element_id***: é um identificador de valor único para cada elemento no diagrama hierárquico. É representado em G-CPN através de um número inteiro. O elemento raiz do sistema tem identficador igual a 1 (um). O identificador de número 0 (zero) é de uso reservado.

(ii) ***parent_id***: é o identificador do elemento pai na hierarquia. O elemento raiz do diagrama indica como pai o elemento de número 0.

(iii) ***state***: determina o estado do elemento quanto à sua utilização. É representado por uma estrutura que indica entre outras coisas o proprietário do elemento e o nível de acesso permitido aos demais usuários.

(iv) ***attributes***: é um conjunto de atributos que caracterizam o elemento dentro do domínio do problema. Por se tratar de um elemento de caráter semântico associado à camada de aplicação, representaremos os atributos como um inteiro.

(v) ***sons_list***: é o conjunto de elementos imediatamente abaixo na hierarquia. A representação será feita através de uma lista de identificadores.

O estado do elemento (*state*) é a estrutura:

(i) ***owner_id***: é o identificador do usuário que detém todos os direitos de proprietário do elemento.

(ii) ***permission_list***: é uma lista de identificadores de usuários de caráter restritivo, que determina os usuários com acesso ao elemento. A lista vazia indica que o objeto é permitido a todos os usuários. Se a lista não é vazia, apenas os usuários explicitamente indicados têm acesso.

(iii) ***attributes_access***: indica o nível de acesso aos atributos concedido aos usuários determinados pela lista de permissão. As possibilidades são: *none* para indicar que os atributos não devem ser sequer visualizados pelos usuários; *read_only* para indicar que os usuários têm apenas acesso de leitura; e *read_write* para indicar a permissão de acesso de leitura e escrita dos atributos.

(iv) ***sons_access***: indica o nível de acesso aos filhos do elemento concedido aos usuários determinados pela lista de permissão. As possibilidades são: *none* para indicar que os sub-objetos não são visíveis; *read_only* indica que os usuários não podem criar novos elementos filhos, embora possam ver os já existentes; e *read_write* indica a permissão para a criação de novos elementos filhos.

Em termos de CPN-ML, a definição da estrutura de um elemento fica associada ao estabelecimento dos seguintes conjuntos de cores:

(i) `color ID = int;`

(ii) `color ID_LIST = list ID;`

(iii) `color ACCESS = with none | read_only | read_write;`

(iv) `color STATE = product ID*ID_LIST*ACCESS*ACCESS;`

(v) `color ATTRIBUTES = (* Depende da aplicação *)`

(vi) `color ELEMENT = product ID*ID*STATE*ATTRIBUTES*ID_LIST;`

Com o elemento de um diagrama hierárquico formalmente definido, resta apenas dizer que DH é um diagrama hierárquico se é um conjunto finito de elementos que satisfaz às seguintes condições:

(i) o elemento de identificador igual a 1 pertence a DH;

(ii) quaisquer que sejam os elementos F e P (exceto F igual a 1):
se F pertence a DH e P é pai de F, então P também pertence a DH.

Com base nas definições anteriores, definir formalmente o que é a visão de um usuário é bastante simples: dado um diagrama hierárquico DH, uma visão do DH é um sub-conjunto de elementos de DH.

A definição acima, entretanto, encerra um grave problema para implementação como sistema distribuído. Garantir que um dado elemento da visão é válido a cada instante, pode ser algo extremamente complexo, especialmente em se tratando de um sistema com alto nível de concorrência. Conseqüentemente usaremos uma versão relaxada de visão, que associa a todo elemento uma validade. A validade de um elemento será garantida através da idéia de inspeções. Um usuário pode ou não inspecionar um elemento do DH. Para inspecionar, ele deve solicitar ao agente coordenador que o notifique de todas as mudanças ocorridas no dado elemento. Os elementos não inspecionados, entretanto, permanecem na visão marcados como inválidos. Esta forma de definir visões, permite diminuir bastante o total de mensagens geradas para manter a consistência dos *sites*.

Finalmente, a camada de cooperação do agente usuário, representada pelo módulo USER deve apresentar em sua interface pontos de acesso para a camada de aplicação suficientes para fornecer todos os serviços necessários. Neste modelo, usaremos os serviços dos módulos G-CPN para a modelagem de recepção de mensagens de entidades remotas. Assim, para cada classe de mensagens a receber, o módulo deve apresentar um método apropriado para o seu tratamento. Os serviços identificados para o módulo USER são:

PONTOS DE ACESSO PARA A CAMADA DE APLICAÇÃO

(i) **login**: permite a entrada de algum usuário à sessão;

(ii) **logout**: permite a saída do usuário da sessão;

(iii) **create_element**: solicita a criação de um novo elemento;

(iv) **delete_element**: solicita a eliminação de elemento;

(v) **update_element_state**: requisita a modificação do estado de um elemento;

(vi) **update_element_attributes**: requisita a modificação dos atributos de um elemento;

(vii) **read_element**: requisita a leitura de algum elemento.

MENSAGENS RECEBIDAS

(i) **user_joined**: indica a entrada de um novo usuário na sessão;

(ii) **user_left**: indica que algum dos usuários deixou a sessão;

(iii) **inspector_joined**: indica que algum usuário inspeciona algum objeto da visão;

(iv) **inspector_left**: indica que algum usuário deixou de inspecionar algum objeto;

(v) **element_created**: indica a criação de algum elemento;

(vi) **element_deleted**: indica a eliminação de algum elemento;

(vii) **element_updated**: indica a modificação de algum elemento.

O agente coordenador será modelado por três módulos G-CPN: ELEMENT, SESSION e GROUP. O módulo ELEMENT será responsável pelo correto tratamento de elementos do DH. É no módulo ELEMENT que residem as implementações dos serviços de criação, modificação e deleção de elementos propriamente ditos. Além disso, uma visão completa do DH deve se fazer presente no módulo ELEMENT. A Figura 5.2 apresenta uma versão simplificada do módulo ELEMENT que encerra apenas o método **create**, cuja função é requisitar do agente coordenador a criação de um elemento no DH.

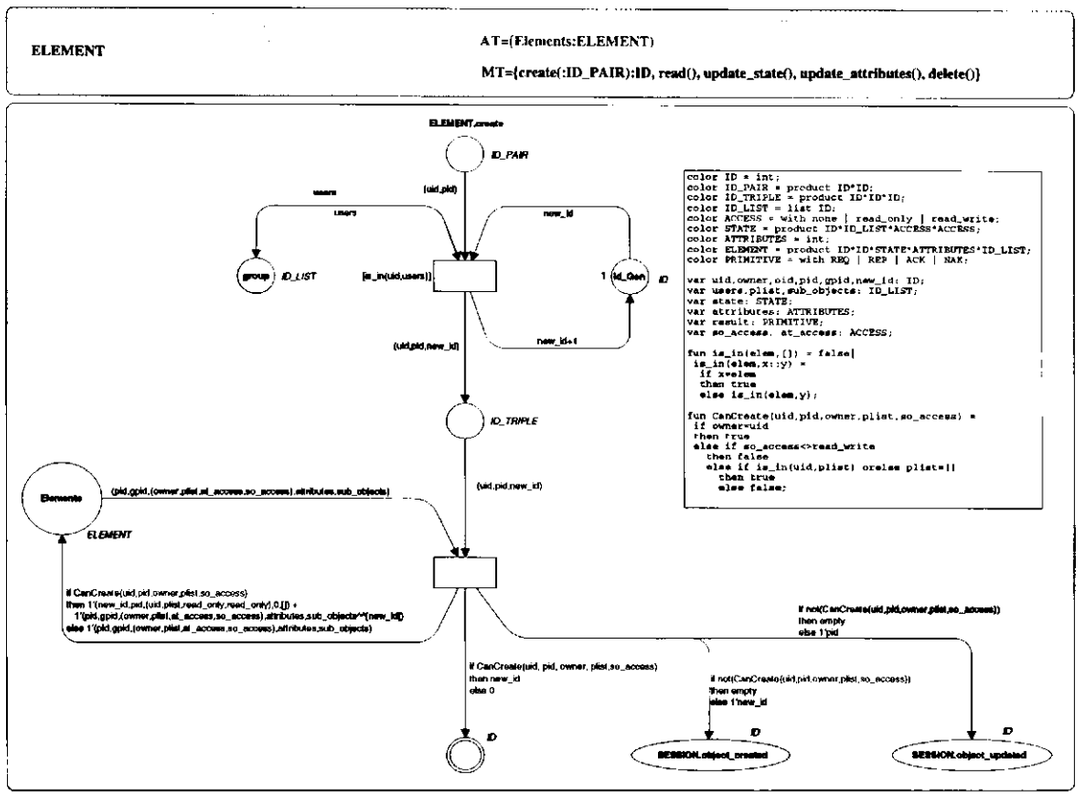

Figura 5.2 - Método **create** da G-CPN ELEMENT

O funcionamento do método *create* do módulo ELEMENT é bastante simples. O módulo ELEMENT tem um atributo denominado DH. No lugar correspondente ao atributo DH, na estrutura interna, serão colocados os elementos pertencentes ao diagrama hierárquico em edição. Ao ser invocado, o método *create* recebe um par de identificadores. O primeiro especifica o usuário que fez a requisição e o segundo determina o elemento pai no qual o novo elemento deve ser criado. A primeira transição simplesmente gera um identificador para o potencial novo objeto. Quando a segunda transição ocorrer, o elemento pai será retirado do DH. Se a criação não for possível, o que é verificado através da função *can_create*, o elemento pai é devolvido e nenhum elemento adicional é colocado. Nesse caso o método retorna o identificador 0 para indicar a impossibilidade de criar o novo elemento. Se a função *can_create* for avaliada como verdadeira, então o elemento pai será devolvido ao DH com mais um identificador na lista de elementos filhos e o novo elemento será inicializado. Além disso, o novo identificador é retornado e duas mensagens são geradas: *element_created* e *element_updated*. A primeira se refere ao novo elemento e a segunda ao elemento pai,

que sofreu alterações no conjunto de elementos filhos. Perceba que essas mensagens são enviadas para o módulo SESSION que também pertence ao agente coordenador.

A função *CanCreate* determina se o usuário tem permissão ou não para efetuar a criação. A permissão é dada se ocorrer uma das seguintes condições: 1) o usuário é o proprietário do objeto; 2) se o usuário não é o dono do objeto, então ele deve fazer parte da lista de permissões (ou a lista deve ser vazia) e o nível de acesso aos elementos filhos deve ser *read_write*.

Na Figura 5.3 apresentamos o agente coordenador com o método **create_element**, complementar do método **create** do agente usuário. Na estrutura interna do módulo USER, o método *create_element* faz uso do serviço oferecido pelo módulo ELEMENT. O módulo USER tem dois atributos bastante importantes denominados *vision* e *session_group*. Quando o método *create_element* é invocado, um par de identificadores são enviados. O primeiro indica o usuário que faz a requisição e o segundo o objeto pai no qual se deseja a criação do elemento. A primeira transição só ocorre se o usuário estiver presente na sessão, o que é verificado pela existência do seu identificador na lista existente no lugar *session_group* através da guarda. Se ocorre, então a validade do elemento pai é verificada na visão do usuário. Se a visão indica que o elemento não está sendo inspecionado, portanto não é válido, a criação será negada e o método finalizado retornando-se o identificador 0. Se a visão indica que o elemento está sendo inspecionado, uma mensagem é enviada para o módulo ELEMENT no agente coordenador requisitando a criação do elemento. Em seguida a segunda transição só ocorrerá quando ocorrer um retorno do invocador anterior e quando a visão indicar que o elemento pai já contém o novo elemento na sua lista de elementos filhos ou se o identificador retornado foi 0, o que indica a não criação do objeto.

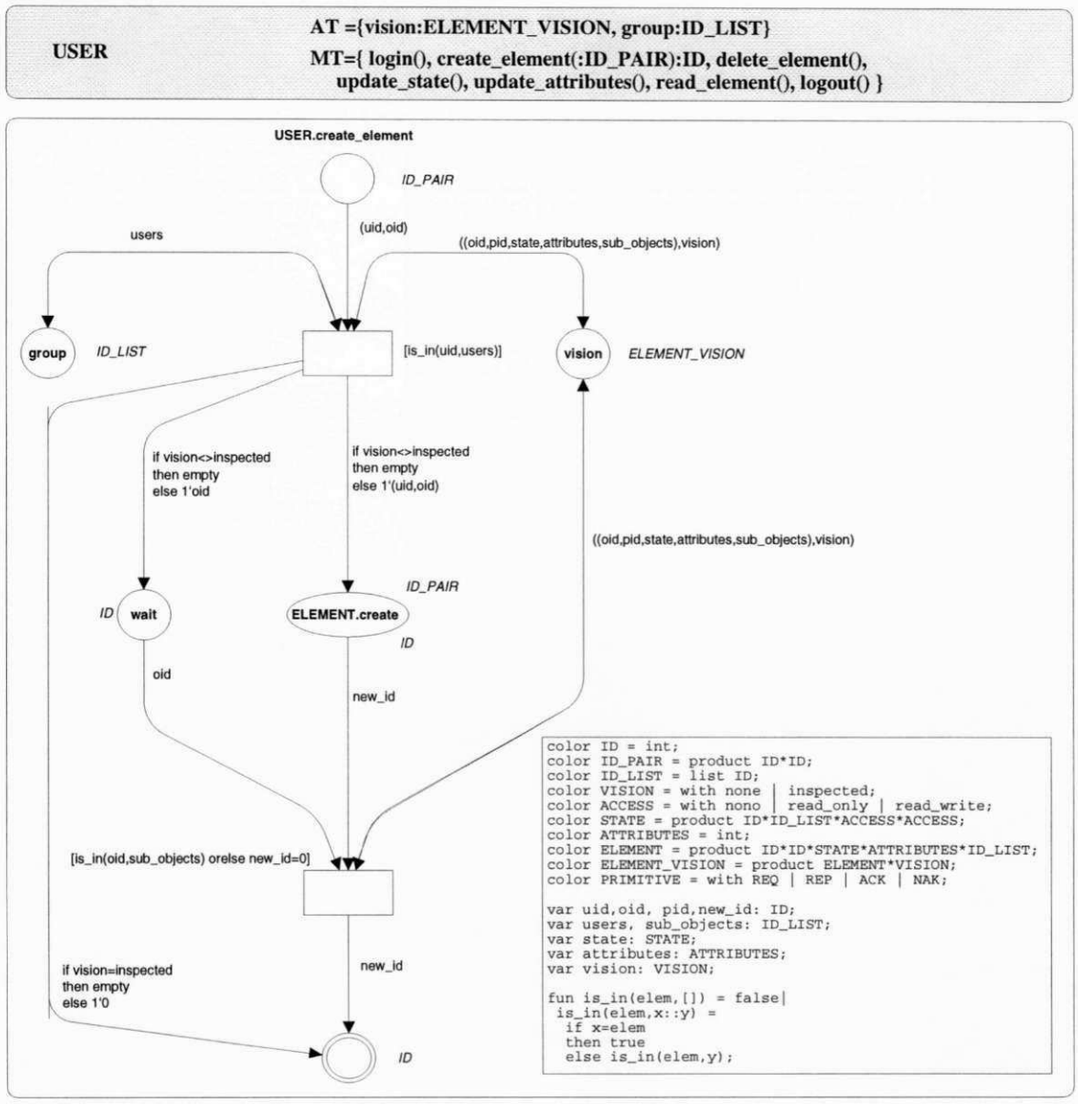

Figura 5.3 - Método **create_element** da G-CPN **user**

Pode parecer estranho que a segunda transição deva esperar que o elemento pai sofra alterações (inclusão do novo elemento na lista de filhos) sem que essa ação sobre a visão seja feita explicitamente pelo método. Na realidade, a alteração é feita em decorrência do método, entretanto, isso se dá por meio de outro método que tem acesso ao lugar *vision*. Esse outro método é justamente o responsável pela recepção e tratamento das mensagens vindas do coordenador que indicam a alteração de um elemento. Dado que o elemento pai está sendo inspecionado pelo usuário, então ele receberá todas as mensagens de alteração do elemento, incluindo uma por ele mesmo solicitada. A verificação da alteração antes do fim do método, garante que a camada de aplicação só

receberá um "aviso" de finalização do método de criação quando a sua própria visão também esteja atualizada.

5.4 Conclusões

O ambiente cooperativo de edição como especificado neste capítulo carece ainda de estudo e trabalho mais profundo. Entretanto, percebe-se que por ser um método matematicamente rigoroso e devido à possibilidade de submeter o sistema a procedimentos sistemáticos de análise, as partes componentes do sistema podem ser verificadas em tempo de especificação, sem que haja necessidade de se especificar o sistema completamente antes de poder efetuar alguns procedimentos de análise e verificação. Além disso, à medida em que desenvolvemos elementos adicionais, é bastante simples submeter o todo (parcial) a métodos de verificação de consistência. O que garante que os elementos produzidos se integram de forma adequada à parte do sistema já desenvolvida.

O processo de modelagem dos módulos do ambiente cooperativo nos fez perceber diversas características do modelo G-CPN. A sua aplicação à especificação e modelagem de sistemas distribuídos tem seus pontos positivos e negativos. É intuitivamente simples especificar componentes se eles caracterizam objetos únicos do sistema, como foi o caso do objeto ELEMENT, embora ele encerre a parte mais complexa de todo o sistema: o controle do diagrama hierárquico. Quanto à sua aplicação à modelagem de classes de objetos, diversos problemas emergem. A falta de possibilidade de estabelecer claramente a diferença entre módulos G-CPN que representam objetos e módulos que representam classes é um problema. Por exemplo, é impossível diferenciar formalmente métodos de classe e métodos de instância. O mesmo se aplica a atributos.

Quanto à modelagem da troca de mensagens entre módulos e à encapsulação que G-CPN promove, não percebemos nehum problema, vindo inclusive a simplificar a especificação do sistema. Além disso, a principal vantagen de modelar um sistema desta natureza através de G-CPN ao invés de usar CPN é a proximidade que se pode perceber entre o produto da especificação e uma possível implementação. Isto tem conotação especial, se na implementação forem utilizadas linguagens de programação orientadas a objetos.

6. ESPECIFICAÇÃO E MODELAGEM DE ENTIDADES GERENCIADAS DE SISTEMAS DE GERÊNCIA DE REDES ATM

6.1 Introdução

A arquitetura de um Sistema de Gerenciamento de Redes (SGR) pode ser organizada em torno de funções e blocos de funções, descrevendo as atividades executadas por um SGR e suas responsabilidades gerenciais. Qualquer par de blocos de funções trocando informações entre si é denominado de ponto de referência. Os pontos de referência definem as fronteiras entre blocos de funções. Além disto, os pontos de referência definem os conjuntos de protocolos e mensagens que são utilizadas para possibilitar a troca de informação entre funções do SGR.

Em um SGR podemos definir três elementos-chave: o sistema de gerência, o sistema gerenciado e os recursos gerenciados. Um gerente, em um sistema de gerência, pode interagir com o sistema gerenciado através de agentes e objetos gerenciados. Objetos gerenciados são abstrações de recursos reais cujo propósito é permitir a gerência do sistema. Os objetos gerenciados são armazenados em um banco de dados de gerência, organizados em torno de uma entidade denominada Base de Gerenciamendo da Informação (BGR, *Management Information Base* (*MIB*)) [Simon, 1996]. Além disso, neste contexto, um gerente pode ser um humano, um processo computacional, ou um sistema de hardware responsável em executar as atividades de gerência.

Considerando o crescente número e complexidade de recursos em uma rede de computadores, é cada vez mais importante a modelagem formal dos objetos gerenciados, a fim de que se possa submeter os modelos propostos a métodos rigorosos de análise. Neste capítulo apresentamos uma proposta de formalização da Interface M4 para Entidades de Objetos Gerenciáveis (EOG, *Object Managed Entities* (*OME*)) informalmente introduzidas em [Af-nm58], através do modelo G-CPN apresentado nos Capítulos 3 e 4. Estas EOGs constituem parte de uma BGR em um SGR. A Interface M4

trata com o Gerenciamento da Informação (GI, *Management Information(MI)*), relacionada com o SGR, considerando diferentes requisitos da visão de rede, detalhes podem ser encontrados na documentação do *ATM Forum* [Af-nm58].

Nas seções seguintes introduziremos os conceitos básicos relacionados à Interface M4 e seu relacionamento com a Interface Q3 de ATM [Soares, 1995]. Apresentamos ainda a análise dos modelos da entidades gerenciadas formalizadas através de G-CPN e simuladas na ferramenta CPN-Design.

6.2 Visão de Rede da Interface M4

O documento AF-NM-0058.000 do *ATM Forum* [Af-nm20] especifica os requisitos funcionais para uma interface capaz de gerenciar Elementos de Rede ATM (ERs, *ATM Network Elements (NEs)*), denominada Interface M4. A função desta interface pode melhor ser compreendida observando o documento de referência de arquitetuta do *ATM Forum* (*The ATM Forum Management Interface Reference Architecture*), como mostrado na Figura 6.1.

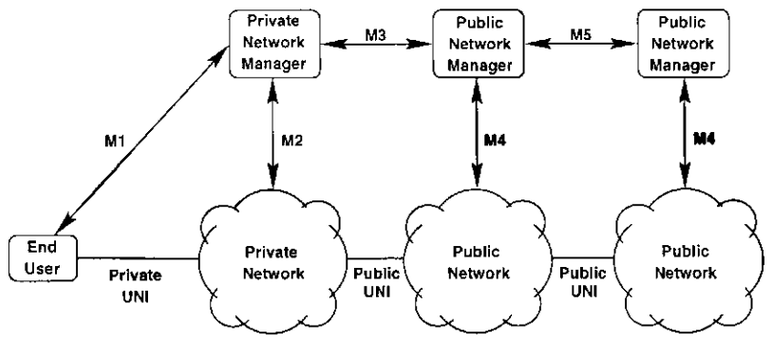

Figura 6.1 - Arquitetura de referência para a interface de gerência do Forum ATM

A Interface M4 define duas visões: a Visão de Elemento de Rede (NE) e a Visão de Rede (VR). A Visão de Elemento de Rede engloba o gerenciamento de ERs ATM. A Visão de Rede engloba o gerenciamento de ERs agregados como uma ou mais sub-redes. A arquitetura lógica é mostrada na Figura 6.2.

Na Figura 6.2 é apresentado como o gerenciamento de rede utiliza a visão de rede para comunicar-se com a função subordinada de gerência de rede. Por completude esta visão incorpora uma visão de elemento de rede entre uma função de gerenciamento de sub-rede e função de gerenciamento do elemento. Um Sistema de Gerenciamento de Rede

pode utilizar entidades gerenciadas na visão de rede, na visão de elemento de rede, ou ambas. Neste documento discutimos a visão de rede M4. Além disso, é possível relacionar a visão de rede com a visão de elemento de rede. Do ponto de vista de arquitetura é possível oferecer somente uma visão de elemento de rede ou uma visão dos serviços de gerenciamento de rede. O documento AF-NM-0058.000 [Af-nm58] aborda somente a funcionalidade da interface para gerenciar a rede, não provendo requisitos dos sistemas de gerenciamento.

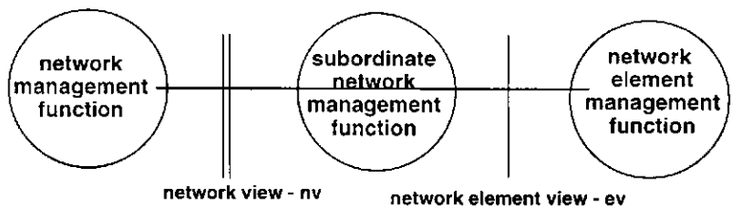

Figura 6.2 - Visão de Rede

6.2.1 Arquitetura de Rede e Elemento de Rede

No nível de arquitetura de rede o SGR possui a habilidade para gerenciar entidades de um único elemento de rede (ER) para o Sistema de Gerenciamento de Sub-Rede (SubSRG) (*Subnetwork Management System*(*SubNMS*)), neste caso a interface M4 entre o ambiente do SGR e o SubSGR apresenta também uma visão de elemento de rede ATM. Nesta arquitetura, o SGR possui a habilidade para ver e gerenciar a rede ATM executando operações na sub-rede como um todo ou desempenhando operações em elementos de rede ATM selecionados. Na Figura 6.3 ilustra-se estas relações.

Utilizando a terminologia TMN (*Telecommunication Managent Network*) a interface M4 é uma Interface *TMN Q3* (Esta terminologia pode ser encontrada na Recomendação M.3010 da ITU-T, *International Telecommunication Union*). Esta recomendação refere-se a Funções de Operação de Sistema (*Operations Systems Functions*(*OSF*)) a qual mapeia os níveis de Gerenciamento de Rede e Gerenciamento de Elemento de Rede entre um SGR público (ou Sistema de Operação TMN, e e um Elemento de Rede TMN, um Sistema de Gerenciamento de Elemento, ou outros SGRs.

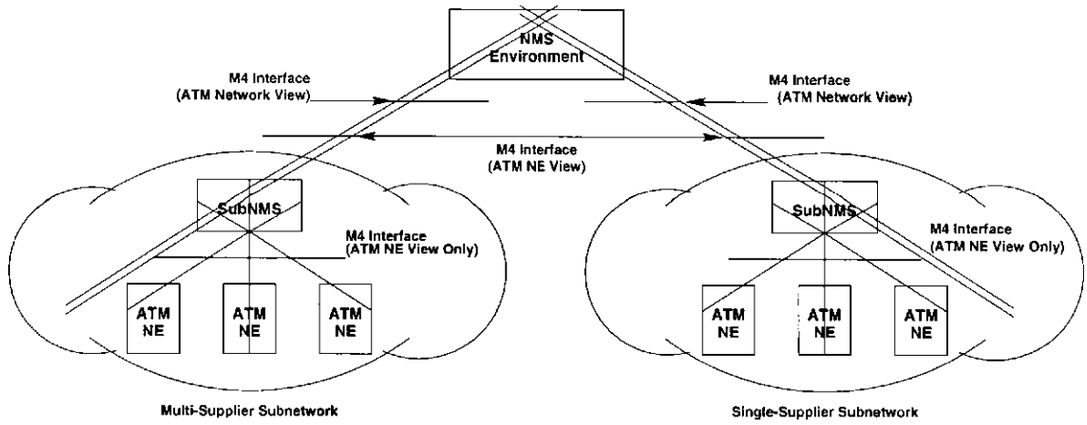

Figura 6.3 - Arquitetura de rede e elemento de rede

A recomendação M3.100 da ITU-T, definindo um modelo genérico de informação constituído de três visões:

- A visão de elemento de rede considera a informação requerida para diferenciar um elemento de rede. Inclui informações necessárias para gerenciar os elementos de rede e seus aspectos físicos.

- A visão de rede considera a informação representando a rede física e logicamente. Considera como as entidades de elementos de rede são relacionadas, topologicamente interconectadas, e configuradas para prover e manter a conectividade fim-a-fim.

- A visão de serviço considera como aspectos de visão de rede (tais como um caminho fim-a-fim) são utilizados para prover um serviço de rede (p.e. disponibilidade, custo, etc), e como estes requisitos são alcaçados através do uso da rede, e todas as informações relacionadas ao usuário.

6.3 Modelagem das Entidades Gerenciadas

No documento AF-NM-0058.000 [Af-nm58], as definições de todas as entidades gerenciadas são apresentadas de maneira informal. Um pequeno parágrafo descreve cada entidade e declara seu propósito no sistema, e apresenta-se uma figura para mostrar o relacionamento de entidade em questão com as outras. Quatro aspectos são então considerados: atributos, notificações, relacionamentos e operações.

Atributos são certas propriedades ou características que distinguem objetos gerenciados. Eles são descritos por uma lista simples que declara o tipo (domínio) e qualificadores leitura/escrita ou somente leitura. Além disso uma curta explicação de cada atributo é apresentada.

Notificações são mensagens (primitivas assíncronas de comunicação) enviadas pelas entidades gerenciadas ao processo gerente. Notificações são necessárias para permitir que o SGR seja informado sobre eventos que ocorrem no sistema. Duas classes de eventos são definidas em [Black, 1995]. Uma relacionada aos eventos gerados por estímulos internos (*internal stimuli*) e outro pelos eventos externos (*external stimuli*). Estímulos internos são eventos ou modificações de estado causadas por condições locais à entidade gerenciada. São exemplos de estímulos internos todas as falhas ocorridas nos equipamentos e nos recursos.

Estímulos externos são eventos oriundos da recepção de uma mensagem indicando uma nova situação em uma outra entidade ou a solicitação de um determinado serviço. Em alguns casos, estímulos externos são gerados pelo próprio SGR de forma direta ou indireta. Como um exemplo, o gerente pode decidir que em um determinado instante algum enlace deve ser desabilitado, e envia uma mensagem ao agente correspondente. Quando do recebimento da mensagem, a entidade gerenciada altera seu estado administrativo e envia uma notificação ao sistema de gerenciamento. Entretanto, mensagens da mesma natureza poderiam ser enviadas à entidade gerenciada por uma outra entidade gerenciada do sistema. Neste caso, a notificação é ainda mais importante, pois ela informa o processo gerente sobre o novo estado, pelo qual ela não requisitou informação.

Cada entidade gerenciada pode ser relacionada a outras entidades de diversas formas. Uma sub-rede, por exemplo, pode ser particionada em diversas outras sub-redes; um enlace topológico é determinado por pontos de terminação de dois enlaces topológicos. Então, é necessário determinar com precisão a natureza, a cardinalidade e a semântica de todos os relacionamentos entre entidades gerenciadas. No documento AF-NM-0058.000 [Af-nm58], relacionamentos são expressos no que é denominado de uma linguagem semi-formal baseada em álgebra relacional. De fato, tem-se uma versão escrita da abrodagem entidade/relacionamento. A forma como cada implementação de um

protocolo específico da Base de Dados de Gerência (BDG) tratará este relacionamento é deixado em aberto.

Existe uma lista de operações para cada entidade gerenciada. Cada operação é descrita por meio de parâmetros de entrada, parâmetros de saída, condições de erro e seu comportamento. Os parâmetros de entrada e saída são descritos de forma similar a ASN.1. Uma lista de condições de erro é declarada com sua semântica informalmente. Finalmente o comportamento é descrito de maneira completamente informal.

6.3.1 Considerações

Apesar de que diversas operações são listadas para cada entidade gerenciada, muitas operações auxiliares (e necessárias) não são descritas. Ainda mais, operações sobre atributos não são apresentadas. Então, muitas outras operações devem ser implementadas durante o desenvolvimento de um protocolo de entidade gerenciada da BDG sem nenhuma recomendação. Não é nem sequer necessário manter uma correspondência um-a-um entre cada elemento descrito na BDG lógica e na implementação. Certamente, isto leva a interpretações errôneas e consequentemente a uma falta de compatibilidade entre diferentes fabricantes, desta forma desconsiderando o principal objetivo do padrão.

A especificação formal de um subconjunto de entidades gerenciadas definidas na visão de rede da interface M4 [Af-nm58] é desenvolvida com o objetivo de aplicar e experimentar o modelo G-CPN. Devido a isso, a especificação aqui apresentada não é completa e nem pretende ser. Ainda mais, muitos detalhes dos modelos desenvolvidos foram omitidos de modo a simplificar os exemplos e as explanações.

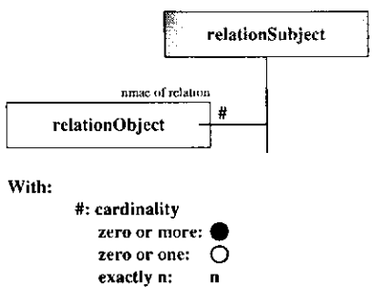

Figura 6.4 - Descrição gráfica informal das entidades gerenciadas

Nossa abordagem provê uma breve descrição informal de cada entidade gerenciada (similar àquelas encontrada em [Af-nm58]), e imediatamente depois , uma explanação de como o modelo G-CPN pode ser obtido. Representações gráficas de partes do modelo (veja a Figura 6.4) são alternadas com explanações. Por fim, o módulo completo é apresentado.

6.3.2 Modelagem das Entidades Gerenciadas

Nesta seção mostramos como modelar as entidades gerenciadas utilizando um exemplo simples. Nosso objetivo é modelar a entidade gerenciada *vcSubnetwork* com todos seus atributos e relacionamentos e com algumas de suas notificações e operações.

A idéia e representar cada classe de entidade gerenciada no sistema por meio de um módulo G-CPN, e cada instância de uma entidade gerenciada por meio de um contexto de G-CPN no módulo apropriado. Desta forma, para criar uma nova instância de uma entidade gerenciada, tudo que o sistema deve fazer é invocar o método correto.

Atributos são modelados pela linguagem CPN-ML. Também, os relacionamentos são modelado como atributos. Definimos um conjunto de cores para cada entidade gerenciada do sistema. Diversos outros conjuntos de cores são definidos de modo a permitir a criação de variáveis utilizadas nas expressões dos arcos.

O envio de uma notificação é modelado pela invocação de um método. Então, o processo gerente, o qual não é modelado aqui, deve oferecer em sua interface uma operação de notificação para permitir a recepção de notificações. Finalmente operações são expressas diretamente na estrutura interna de um módulo. Cada operação sobre atributos ou sobre os relacionamentos é modelada especificamente por um método.

Entidade Gerenciada vcSubnetwork

A entidade gerenciada *vcSubnetwork* representa o elemento topológico conceitual utilizado para tratar informações características (células ATM em uma sub-rede ATM). O nome *vcSubnetwork* significa canal bidirecional de sub-rede, isto ocorre pois esta entidade gerenciada é especializada por nível.

Esta entidade gerenciada é definida para ser criada automaticamente pela entidade gerenciada de rede. Entretanto, um método é necessário para possibilitar a recepção de uma mensagem de criação de uma rede. Esta entidade não pode ser destruída.

Os atributos da entidade *vcSubnetwork* são:

- **Identificador de sub-rede (*Subnetwork ID*)**: este é um atributo somente de leitura. Todas as identificações são modeladas em G-CPN por um conjunto de cor denominado ID.

- **Identificação de Sinal (*Signal Identification*)**: este atributo representa o formato específico que o recurso transporta. No caso desta entidade o atributo é fixado para o nível VC no momento da criação da entidade. Em G-CPN, especificaremos um conjunto de cores específico que contém duas cores: VC e VP.

- **Rótulo de Usuário (*User Label*)**: este é um atributo de leitura/escrita. Permite a identificação da organização do gerente.

Existem somente três eventos que causam a geração de notificações nesta entidade gerenciada. A notificação *UserLabelChange* é utilizada para informar ao gerente sobre as modificações no atributo referente ao rótulo do usuário. A notificação *vcSubnetworkCreated* é utilizada para informar sobre a criação de uma instância desta entidade gerenciada. Finalmente *vcSubnetworkDeletion* não é utilizada aqui, uma vez que a destruição não é suportada neste contexto. Entretanto sua semântica é óbvia.

Relacionamentos são modelados de um modo semelhante aos atributos. Todos os conjuntos de cores para relacionamentos são colocados juntos em um único produto de conjunto de cores. Entretanto, nenhum lugar especial na estrutura interna é utilizado para manter estes relacionamentos. O mesmo lugar (lugar de atributo no contexto de G-CPN) mantém todos os atributos e relacionamentos de cada instância de entidade gerenciada. Então, este lugar possui uma entrada (ficha) para cada *vcSubnetwork* no sistema.

Os relacionamentos da entidade *vcSubnetwork* são com as entidades gerenciadas *vcSubnetworkTPs*, *vcSubnetworkConnection*, *vcSubnetwork*, *vcTopologicalLink*, e *vcTopologicalLinkTP*. Todos os relacionamentos são explicados por diagramas entidade-relacionamento, como mostrado na Figura 6.5.