

Redes de Petri Orientadas a Objetos

Dalton Dario Serey Guerrero

Tese de Doutorado submetida à Coordenação dos Cursos de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande – Campus I como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação

Angelo Perkusich

Orientador

Jorge C. A. de Figueiredo

Orientador

Campina Grande, Paraíba, Brasil

©Dalton Dario Serey Guerrero, 26 de abril de 2002



G934r Guerrero, Dalton Dario Serey.
Redes de petri orientadas a objetos / Dalton Dario Serey
Guerrero. - Campina Grande, 2002.
169 f.

Tese (Doutorado em Engenharia Elétrica) - Universidade
Federal de Campina Grande, Centro de Ciências e Tecnologia,
2002.

"Orientação: Prof. Dr. Ângelo Perkusich, Prof. Dr. Jorge
César Abrantes de Figueiredo".

Referências.

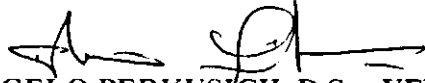
1. Redes de Petri. 2. Redes de Petri Orientadas a
Objetos. 3. Processamento da Informação. 4. Tese -
Engenharia Elétrica. I. Perkusich, Ângelo. II. Figueiredo,
Jorge César Abrantes de. III. Universidade Federal de
Campina Grande - Campina Grande (PB). IV. Título

CDU 004.7(043)

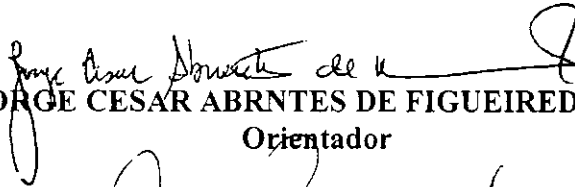
REDES DE PETRI ORIENTADAS A OBJETOS

DALTON DARIO SEREY GUERRERO

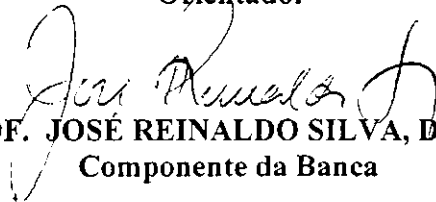
Tese Aprovada em 26.04.2002



PROF. ANGELO PERKUSICH, D.Sc., UFPB
Orientador



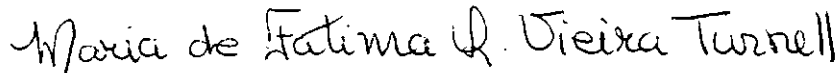
PROF. JORGE CESAR ABRANTES DE FIGUEIREDO, Dr., UFPB
Orientador



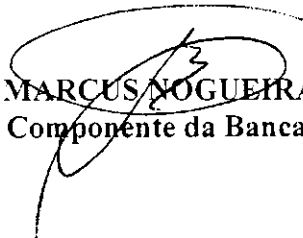
PROF. JOSÉ REINALDO SILVA, Dr., USP
Componente da Banca



PROF. AUGUSTO CESAR ALVES SAMHAIO, Dr., UFPE
Componente da Banca



PROFA. MARIA DE FÁTIMA QUEIROZ VIEIRA TURNELL, Ph.D., UFPB
Componente da Banca



PROF. ANTONIO MARCUS NOGUEIRA LIMA, Dr., UFPB
Componente da Banca

CAMPINA GRANDE - PB
Abril - 2002

UFPA

À parcela do povo brasileiro que, sem saber e sem poder, financia a pesquisa e a educação públicas no Brasil.

Agradecimentos

Este trabalho não teria sido possível sem a contribuição de várias pessoas. Agradeço a meus orientadores pelo apoio e incentivo contínuo durante todo este tempo. Sua participação neste projeto foi bem além da orientação acadêmica. Seu envolvimento e contribuição na compreensão do problema, bem como nas decisões relativas às soluções propostas foi a de verdadeiros colegas de projeto.

Agradeço também a Sandro Alex Damasceno Costa por todas as discussões e debates, especialmente a respeito do modelo conceitual e sobre o formalismo para a descrição de sistemas de objetos. A Érica de Lima Gallindo pela paciência de compreender a notação e aventurar-se a aplicá-la ainda que sem ferramentas específicas para suportar o trabalho. Graças a esse experimento, pude avaliar a notação concreta na prática e melhorá-la em diversos aspectos. Outros alunos de mestrado da COPIN também se envolveram no projeto e desenvolveram outros experimentos de modelagem. Nesse sentido, agradeço aos esforços de Edna Canedo e a José Amâncio Santos por suas contribuições.

Boa parte deste trabalho foi desenvolvido junto ao Grupo de Redes de Petri da Universidade de Århus na Dinamarca. Agradeço a todos os membros do grupo, em especial aos líderes, os professores Kurt Jensen e Søren Christensen pelas excelentes condições de trabalho oferecidas, sem que quaisquer restrições ou condições especiais fossem colocadas. O grupo me proporcionou um ótimo ambiente acadêmico sem o qual este trabalho não seria o mesmo. Agradeço também aos amigos que fiz na Dinamarca que tornaram esse período tão mais agradável. Em especial, a Jorge e Adriana Figueiredo por todo o apoio e pelos ótimos momentos que nossas famílias puderam compartilhar.

À CAPES, pelo financiamento de meus estudos no exterior e de parte deles no Brasil. Ao Departamento de Sistemas e Computação da Universidade Federal de Campina Grande e colegas de trabalho pela oportunidade. À COPELE e seus excelentes funcionários, pelos excelentes serviços prestados durante estes anos.

Finalmente, quero agradecer àqueles que me suportaram em família durante estes anos. Em especial a Verônica Serey, minha esposa, por sua infinita paciência e apoio nos momentos de desânimo, pela disposição para acompanhar-me em todos os momentos e por todo o carinho dedicados. A meus pais e a minha irmã, simplesmente por *tudo* e, em especial, por compreenderem tantas ausências nestes anos.

Resumo

Este trabalho aborda a integração de conceitos da orientação a objetos à modelagem de sistemas por redes de Petri. Em particular, de sistemas distribuídos e concorrentes de software. Argumentamos que a principal causa do insucesso das notações orientadas a objetos de redes de Petri existentes é a *forma* de integração que enfatiza aspectos sintáticos em detrimento dos aspectos semânticos. Defendemos que princípios semânticos devem orientar a composição das estruturas sintáticas e que estas devem se combinar de forma ortogonal. Desta forma, promove-se o desenvolvimento de notações com melhores condições de absorção por projetistas que dominem as linguagens básicas, porque minimiza o impacto de uma sobre a outra, preservando a natureza de cada uma. Além disso, a abordagem maximiza as condições de adaptação de métodos de análise e modelagem existentes porque no caso geral os métodos dependem de propriedades semânticas.

A forma como as notações combinam os conceitos impede que os modelos proporcionem visões complementares e articuladas do sistema. Um dos principais problemas é o desequilíbrio dos conceitos de redes de Petri e da orientação a objetos. As abordagens tendem a focar redes de Petri como o elemento central da notação. Este desequilíbrio tem duas conseqüências. Primeiro, torna necessário modificar os conceitos característicos de redes de Petri para adaptá-los à orientação a objetos. Isto, por sua vez, dificulta a adaptação dos métodos convencionais de análise. Segundo, é necessário adaptar e reinterpretar os conceitos da orientação a objetos para inseri-los na formalização de redes de Petri. Isto tende a descaracterizar o paradigma, o que tem influência sobre a aceitação da notação.

Além dos argumentos que sustentam as observações acima, apresentamos uma estratégia de integração que evita os problemas apontados. A estratégia consiste em equilibrar a ênfase dada aos conceitos de cada um dos paradigmas e compor a notação e seu significado de forma *ortogonal*, enfatizando uma composição semântica. Isto é obtido pela adoção explícita de um modelo conceitual de computação concorrente e distribuída cuja noção fundamental é o objeto. Um formalismo independente de redes de Petri que permite caracterizar a evolução de sistemas de objetos segundo o modelo conceitual é apresentado. Finalmente, propomos nossa notação, usando uma abordagem que enfoca a composição de significados e não das construções sintáticas, preservando as características de cada paradigma.

Abstract

This thesis discusses the integration of object oriented concepts and Petri nets modelling of systems. In particular, distributed and concurrent software systems. We argue that the main cause for the unsuccessful object oriented Petri nets notations is the *approach* used to combine concepts which emphasizes syntactical aspects instead of the semantic ones. We defend that semantic integration principles must guide the combination of syntactical structures in an orthogonal way. This approach promotes the development of notations that are more easily accepted by designers, since it minimizes the influence the paradigms have on each other. The essence of each language is better preserved this way. Also, it maximizes the chances of adapting the existing analysis techniques, because most of them depend on semantical aspects.

Present notations combine concepts in a syntactical way. Such approach does not provide two complementary, articulated views of the models. Instead, it provides a single statically compound view. One of the main reasons is the unbalanced integration of concepts. Because object orientation is often explained informally, formalizations are clearly Petri-net biased. This has two consequences. First, it forces different interpretation for existing concepts of Petri nets, so they can be adapted to object language. This has a severe impact on the chances of adaptation of existing techniques, because they rely heavily on concepts whose interpretations are modified. Second, it forces the adaptation of object concepts, so they fit well within Petri nets. This leads to a loose/weak characterization of the object paradigm and, as a consequence, to a badly accepted notation.

As a result, we present an integration strategy that avoids the problems mentioned above. The strategy consists on balancing the emphasis and the role that each paradigm has on the proposed notation, assuring that the notation and its meaning are defined in a *orthogonal* way, and defining a semantic composition of the involved concepts. In practice, this was achieved through the explicit adoption of a conceptual computing model for concurrent and distributed systems for which the central notion is the object. We also present a (Petri net independent) formalism that allows the modelling of the evolution of concurrent object systems. Finally, we propose an integrated notation, which is based on meaning composition instead of syntactical integration.

Conteúdo

1	Introdução	1
1.1	Motivação e contexto	2
1.2	Declaração do Problema e da Tese	7
1.3	Estrutura deste documento	10
2	Revisão Bibliográfica	13
2.1	Abstração e decomposição em redes de Petri	13
2.2	Orientação a objetos e redes de Petri	18
2.2.1	Principais abordagens	19
2.2.2	Outros trabalhos	24
2.3	Avaliação crítica	25
2.3.1	Sobre os conceitos de redes de Petri	26
2.3.2	Sobre os conceitos de orientação a objetos	27
2.3.3	Outras linguagens formais e OO	30
2.4	Conclusões	31
3	Modelo Conceitual	35
3.1	Características de uma “boa” integração	35
3.2	Projeto da linguagem	39
3.3	O modelo conceitual	40
3.3.1	Objetos	40
3.3.2	Sistemas de objetos	42
3.3.3	Ações e interações	43
3.4	Conclusões	45

4	Sistemas de Objetos	47
4.1	Estruturas	48
4.1.1	Definições elementares	48
4.1.2	Notação e linguagem	50
4.1.3	Estruturas bem-formadas	54
4.1.4	Subestruturas e partições	58
4.1.5	Conclusões	64
4.2	Configurações	64
4.2.1	Ações e estados internos	65
4.2.2	Configurações	67
4.2.3	Notação	68
4.3	Sistemas de Objetos	71
4.3.1	Avaliação do efeito das ações	73
4.3.2	Eventos	74
4.3.3	Ocorrência e alcançabilidade	75
4.3.4	Reduções	76
4.4	Conclusões	77
5	Redes de Petri Orientadas a Objetos	79
5.1	Modelos e sintaxe abstrata	79
5.1.1	Definições básicas	80
5.1.2	Classes	84
5.1.3	Modelos	88
5.1.4	Comentários	88
5.2	Significado dos modelos	89
5.2.1	Comportamento de redes de Petri	90
5.2.2	Semântica de classes	92
5.2.3	Semântica de modelos	94
5.3	Concretização da sintaxe	96
5.3.1	Um exemplo	97
5.3.2	Diagramas de classes	106
5.3.3	Diagramas de comportamento	109

5.3.4	Outros diagramas	112
5.4	Conclusões	112
6	Análise e validação	114
6.1	Introdução	114
6.2	Grafos de alcançabilidade	118
6.3	Condição fundamental de análise	122
6.4	Rede equivalente	124
6.4.1	Algoritmo de construção de uma rede equivalente	125
6.4.2	Exemplo de aplicação do algoritmo	134
6.5	Conclusões	140
7	Conclusão	142
7.1	Contribuições	144
7.2	Trabalhos futuros	146
A	Um estudo de caso: modelagem do protocolo <i>Bouncer</i>	148
A.1	O Bouncer	148
A.2	Diagrama de Classes	150
A.3	Detalhamento das Classes	151
A.3.1	Classe CLIENT	151
A.3.2	Classe BOUNCER	153
A.3.3	Classe HOST	155
A.3.4	Classe GC_SERVICE	156
A.4	Diagrama de Fluxo Mensagens	157
A.5	Conclusões	159

Lista de Figuras

2.1	Mecanismos de decomposição hierárquica de redes de Petri	14
3.1	Visões ortogonais dos aspectos de um modelo	36
3.2	Distribuição dos conceitos da proposta em camadas	39
3.3	Ilustração de um sistema de objetos proativos	42
3.4	Identificadores de objetos e grafo de interconexão	43
4.1	Representação gráfica de uma estrutura	51
4.2	Subestruturas bem-formadas e mal-formadas	54
4.3	Subestruturas não nulas de $E = a + \overline{ab} + b$	59
4.4	Representação gráfica de $\langle a_1 + \overline{ab} + b_1, \{a:b.m, b:a?m\} \rangle$	71
4.5	Comunicação assíncrona entre dois objetos	72
4.6	Sistema produtor/consumidor	73
5.1	Ilustração da mesa de jantar dos filósofos	98
5.2	Diagrama de classes para o problema dos filósofos	99
5.3	Diagramas de comportamento para o problema dos filósofos	101
5.4	Diagrama de objetos para o problema dos filósofos: configuração inicial	102
5.5	Configuração inicial com dois filósofos	104
5.6	Representação do comportamento para a configuração da Fig. 5.5	105
5.7	Relacionamentos entre classes	107
6.1	Representação esquemática da modelagem do ciclo de vida dos objetos	129
6.2	Modelo detalhado do problema dos filósofos	134
6.3	Rede equivalente ao modelo dos filósofos: após etapa 2 do algoritmo	135
6.4	Rede equivalente ao modelo dos filósofos: após etapa 4 do algoritmo	136
6.5	Rede equivalente ao modelo dos filósofos	137

6.6	Versão reduzida da rede da Figura 6.5	139
6.7	Grafo de alcançabilidade da rede equivalente	140
A.1	Cenário Típico	149
A.2	Diagrama de classes	150
A.3	Classe CLIENT	151
A.4	Classe BOUNCER	154
A.5	Classe HOST	156
A.6	Classe GC_SERVICE	157
A.7	Diagrama de fluxo de mensagens	158

Capítulo 1

Introdução

Este trabalho é apresentado como contribuição para o conjunto de saberes a que se denomina *engenharia de software*. Em particular, aborda o tema da modelagem e análise matemática de sistemas distribuídos e concorrentes de software através da família de formalismos a que se denomina *redes de Petri*.

O foco temático é a integração de conceitos da *orientação a objetos* às *redes de Petri*. A idéia da integração não é nova. Na prática, contudo, o uso das chamadas *redes de Petri orientadas a objetos* é inexpressivo. As explicações para o fato são diversas e variam de um caso a outro. Há, porém, um motivo comum. A modelagem matemática de um sistema se justifica pela possibilidade de submeter os modelos a procedimentos automatizados de análise e verificação. Contudo, é justamente neste aspecto que as propostas têm se mostrado inadequadas. Em geral, não há métodos de análise eficientes para as linguagens propostas. Isto, por sua vez, decorre da má escolha ou da distorção de conceitos e de sua expressão inadequada, o que impossibilita o desenvolvimento de técnicas de análise e a construção de ferramentas.

O restante deste capítulo é dedicado a dois propósitos: situar e motivar o trabalho apresentado no contexto dos problemas da engenharia de software; e declarar apropriadamente o problema abordado, apresentando as idéias centrais da solução proposta. Uma seção final delinea o resto deste documento, orientando o leitor para que encontre a informação desejada.

1.1 Motivação e contexto

Engenharia de software Há várias formas de definir o que é *engenharia*. É comum tomá-la como um corpo de conhecimento de base científica acumulado com o propósito de produzir e manter artefatos que atendam às necessidades humanas. Cada uma de suas disciplinas clássicas tem um corpo próprio de conhecimento, formado por *teorias, métodos, técnicas, convenções, ferramentas*. É o conhecimento científico que eleva uma arte humana à condição de engenharia. E é exatamente a natureza científica desse conhecimento que permite antecipar as qualidades e propriedades do que se deseja produzir, que permite prescrever condições de uso e prever o comportamento em situações de limite com conhecido grau de precisão.

Há dois componentes que podemos destacar nesse corpo de conhecimento: a matemática e a ciência. Cada disciplina de engenharia precisa desenvolver e adotar linguagens, convenções e notações necessárias à prática profissional. Também são necessárias para preservar, disseminar e depurar o próprio corpo de conhecimento. É o uso da *matemática* para este fim que confere à engenharia o rigor de expressão e metodologia necessários ao sucesso da prática. Em segundo lugar, há as ciências naturais. São elas que versam sobre o domínio natural sobre os quais atua a engenharia. Em última instância, os limites de atuação da engenharia mecânica, por exemplo, são determinados pelas leis da física e pelas propriedades dos materiais usados. Em suma, são estes dois aspectos que caracterizam uma engenharia, proporcionando-lhes os alicerces: a matemática como principal ferramenta e as ciências naturais como fonte de conhecimento.

Retornemos, diante do exposto, à engenharia de software (ES). Se a atividade a que se propõe consiste em produzir, operar e manter programas e sistemas de computador, então certamente é candidata à condição de engenharia. Além disso, desejamos que adquira características de engenharia. Queremos ser capazes de prever propriedades dos sistemas, de antecipar suas condições limite, de sistematizar os projetos, etc. Também queremos que sua disseminação e aprendizado independam, tanto quanto for possível, de virtudes e aptidões especiais dos indivíduos. Queremos que seja técnica e, como tal, que possa ser aprendida e ensinada. Queremos afastá-la da condição de arte que sempre parece caracterizá-la.

Alcançar esse propósito significa fundamentar a ES cientificamente. Significa adotar a

matemática como ferramenta e a ciência como fonte de conhecimento. Também significa repensar suas técnicas, métodos e processos à luz do conhecimento científico. A questão é: quais são a ciência e a matemática adequadas à ES? Ao contrário das ciências naturais sobre as quais se fundamentam outras engenharias, os objetos de estudo da *ciência da computação* não são fenômenos da natureza ou entidades físicas concretas. Seus objetos são algoritmos, programas e dados. Entidades abstratas que pertencem ao domínio das idéias. Suas listagens em papel, os *bits* armazenados em mídias magnéticas ou óticas não passam de suas representações. Tal como “0”, “1” ou “2” são *representações* numéricas e não números, no sentido conceitual. Por outro lado, a *lógica formal* e a *matemática discreta*, candidatas à matemática da ES, correspondem exatamente à matemática dos conceitos e das abstrações. Esta semelhança entre os conceitos matemáticos e o objeto de estudo da ciência da computação dificulta o entendimento da questão. Os objetos de estudo da ciência da computação têm a mesma natureza que as construções mentais que precisamos usar para caracterizá-los. A ciência e a matemática da ES se confundem (ver [Mai97] sobre este assunto).

As evidências mostram que há relevantes diferenças entre as disciplinas clássicas de engenharia e a ES. E isto motiva intermináveis discussões sobre o tema. Além disso, no atual estado de coisas, as técnicas e métodos científicos à disposição do profissional de software ainda não cobrem todo o espectro de aplicações, dizem alguns autores (veja [Hoa84; Par97] a esse respeito). Logo, concluem, ainda é necessária uma boa dose de *arte* e subjetividade para tratar determinadas classes de sistemas. É certo que esta discussão ainda se estenderá por algum tempo.

Modelagem matemática de software Várias fases precedem a implementação, propriamente dita, em qualquer projeto de engenharia. Em particular, é necessário obter uma descrição razoavelmente completa do que se deseja construir—chamada *especificação*. Entre profissionais e pesquisadores da ES, a importância da fase de especificação é ponto passivo. Ainda assim, apesar da ênfase e da importância que recebe em cursos e livros de ES, é comum que na prática seja posta em segundo plano. Também é freqüentemente confundida com a fase precedente (determinação de requisitos e objetivos) ou com a subsequente (projeto do sistema) [Mey85]. Alguns autores atribuem o problema à forte ênfase dada à atividade de programação em detrimen-

to das atividades de natureza analítica em cursos de graduação relacionados [Par90; Gri96]. Mas, certamente, outro fator que contribui para isso é a semelhança e a confusão que existe entre *especificações, modelos, projetos* e os próprios *artefatos* a que ES se propõe construir—conseqüência do problema discutido anteriormente.

Por outro lado, há uma crescente percepção da necessidade de métodos mais rigorosos do que os usados tradicionalmente. Os chamados *métodos formais* são, de fato, uma coleção de técnicas e métodos matematicamente embasados para a modelagem e análise de sistemas de software. Trata-se exatamente de uma tentativa de fazer valer o que preceitua a experiência adquirida nas demais engenharias: adotar a matemática como principal ferramenta do dia-a-dia¹ do engenheiro de software.

Métodos formais têm sido propostos para todas as fases do desenvolvimento. Contudo, não são propostos como a nova “bala de prata” para atacar o problema da ES. Sabe-se que é pouco provável que um único método possa ser aplicado em todos os contextos e para todos os propósitos (ver [FPB87]). A intenção é desenvolver modelos, notações e técnicas que promovam o rigor no trato, a precisão e não ambigüidade das descrições e o raciocínio matemático (ver [Mey85; Van98] sobre o tema). Indiretamente, ganha também o processo de software. É que o uso de notações e técnicas matemáticas para modelar e analisar sistemas reforça também a possibilidade de considerar especificações como contratos, pois torna viável a verificação de seus cumprimentos (ver [CHJ86]).

“(...) Em essência, métodos matemáticos oferecem as mesmas vantagens para projeto de software que diversas outras disciplinas de engenharia têm explorado—especificamente, análise matemática usando um modelo matematicamente fundamentado. Tais modelos permitem que o projetista antecipe características do comportamento e valide a precisão de um sistema sem que precise confiar completamente em processos exaustivos e não-garantidos de testes.” [Sai96, p. 16]²

O número de casos em que os chamados métodos formais vêm sendo aplicados com sucesso tem se multiplicado nos últimos anos (ver [CW96] para um apanhado geral). Também cresce o número de técnicas e ferramentas de análise científica e matematicamente

¹A expressão *métodos formais* é usada mesmo que o método não seja baseado na lógica formal, de onde o nome é derivado (ver [Par94; Par96]).

²Tradução livre.

embasadas. Em particular, a verificação de modelos (*model-checking*), a prova automática de teoremas e a geração automática de testes têm se mostrado bastante úteis em contextos práticos.

Software concorrente Nos últimos anos temos assistido a um gradual aumento na quantidade e na variedade de estruturas e arquiteturas de máquinas paralelas usadas na prática (máquinas multiprocessadas, redes de processadores, etc). Logo, é natural que novos requisitos de software tenham surgido. Em particular, a distribuição espacial a que têm sido submetidos os novos sistemas exige que aspectos de concorrência e paralelismo sejam adequadamente apreciados durante todos os estágios de seu desenvolvimento.

Concorrência, contudo, não é um novo objeto de estudo para a ciência da computação. Ainda nos anos 60, Carl Adam Petri propôs uma teoria geral da concorrência e de eventos discretos parcialmente independentes. Além disto, diversos outros modelos de concorrência e computação concorrente têm sido propostos em conjunto com bons ferramentais teóricos. Alguns dos trabalhos mais relevantes sobre o assunto são *lógica temporal* [Pnu77], CCS [Mil80], CSP [Hoa85], sistemas de atores [Agh86] e mais recentemente π -cálculo [MPW89; Mil93]. Durante estes anos, porém, a teoria de redes de Petri continuou evoluindo. Extensões e uma vasta teoria de provas e métodos de análise têm sido investigadas e desenvolvidas. Dentre as extensões mais utilizadas destacamos as redes Predicado/Transição [Gen87], e as redes Coloridas [Jen92], como representantes da classe de redes a que se denomina *redes de alto nível*—de especial interesse para a modelagem de software.

Do ponto de vista da ES, contudo, concorrência e distribuição são tópicos de muito menor maturidade. Zave argumenta que modelos concebidos com caráter genérico e unificado para a especificação e modelagem de sistemas concorrentes têm se mostrado inadequados para o tratamento de sistemas realistas [Zav96]. Zave propõe que modelos específicos e comprometidos com classes mais específicas de sistemas sejam desenvolvidos, proporcionando facilidades reais que possam ser efetivamente usadas em contextos práticos. Outras engenharias não despendem esforço na busca por métodos genéricos e aplicáveis a todos os propósitos. Diante destes argumentos, uma estratégia de pesquisa razoável é centrar-se em algumas poucas noções e conceitos que pareçam relevantes, e desenvolver coleções de métodos, modelos, notações e técnicas comprometidos com classes mais específicas de domínios de problemas.

ES e Redes de Petri No foco deste trabalho está a família de formalismos denominada genericamente de *redes de Petri*. Sua aplicação é indicada para a modelagem, especificação e análise de sistemas concorrentes. Da perspectiva de modelos de computação, redes de Petri podem ser vistas como uma classe de autômatos de comportamento intrinsecamente concorrente. Com uma vasta e sólida teoria desenvolvida durante os últimos 40 anos, constituem um formalismo extremamente atrativo para a modelagem e análise de sistemas de software distribuídos e concorrentes. Outros fatores de motivação são a simplicidade dos conceitos que as definem e uma faceta gráfica intuitiva que espelha o comportamento dos sistemas modelados, tornando-a também uma útil ferramenta de comunicação.

Do ponto de vista prático, contudo, há fatores que dificultam sua utilização. O primeiro é o nível extremamente básico com que a definição original permite tratar dados. Esse problema, abordado no final dos anos 70 e início dos anos 80, foi resolvido com a introdução da classe de redes de Petri ditas de *alto nível*. A idéia consiste em definir as redes de tal forma que manipulem diretamente dados complexos em substituição à informação binária manipulada pelas redes nas definições originais, que por oposição são chamadas de redes de *baixo nível* (a partir deste ponto, onde dissermos *redes de Petri*, entenda-se *redes de Petri de alto nível*, exceto onde for explicitamente indicado o contrário).

O segundo fator é que redes de Petri não provêm mecanismos para a estruturação e/ou decomposição dos modelos. Conseqüentemente, mesmo modelos de sistemas complexos têm a forma de uma única rede extensa e plana. Diversos mecanismos de abstração têm sido propostos. Na prática, porém, é utilizado apenas um esquema de “modularização visual”. Através do termo pretendemos que fique claro que se trata apenas de uma forma de desenhar separadamente os trechos da rede. Formalmente, continua-se com uma única rede. Naturalmente, o esquema provê alguma facilidade ao projetista. Contudo, sem as propriedades fundamentais de uma boa modularização. O princípio de *informação escondida* proposto por Parnas, por exemplo, não é respeitado (ver [Par72]). A alteração de um módulo, mesmo em aspectos que deveriam ser puramente locais a ele, pode ter diversas conseqüências sobre o restante do modelo. Isto ocorre porque a separação é meramente visual. O esquema não provê mecanismos efetivos para distribuir responsabilidades, proporcionando uma relação contratual entre os módulos. Além disso, a análise formal do modelo tem que ser efetuada em toda a rede, dado que um único “módulo” não concentra toda a informação necessária para sua análise isolada.

Orientação a objetos Qualquer que seja a solução para o problema comentado, deve introduzir mecanismos de estruturação interna às redes de Petri, minimizando o impacto sobre a “analisabilidade” dos modelos através dos métodos já existentes. A proposta de usar os conceitos de orientação a objetos com esse propósito não é realmente nova³. Contudo, nenhum dos formalismos propostos até o momento tem resolvido o problema a contento. Na maioria das propostas, o formalismo resultante ainda distorce bastante os conceitos de redes de Petri, impedindo a adaptação dos métodos de análise. Em outras, são os conceitos de orientação a objetos que sofrem distorções, reinterpretações ou são simplesmente ignorados. Em muitos casos, a semântica da notação é expressa informalmente ou através de uma ferramenta de software que a implementa antecipadamente, distorcendo a idéia de notação formal. Naqueles em que a semântica é expressa formalmente, é comum que a idéia consista em alterar a regra de disparo, alterando o comportamento básico das redes de Petri.

1.2 Declaração do Problema e da Tese

As equipes de projeto de várias linguagens têm se deparado com o problema da estruturação interna. Diversas abordagens têm sido propostas e experimentadas. Nos últimos anos, contudo, a ampla disseminação dos conceitos da *orientação a objetos* (OO) pelas mais diversas sub-áreas da ciência da computação, e até mesmo em outras engenharias, tem chamado a atenção para este paradigma de decomposição e estruturação. São inúmeras as linguagens e notações que têm sido adaptadas aos preceitos da OO. Naturalmente, há certo modismo no fenômeno, mas não se pode negar-lhe o efeito unificador e popularizador de um conjunto mais ou menos uniforme de conceitos de decomposição e estruturação. Métodos formais não fogem à regra. Diversas linguagens e métodos têm sido expostos à OO, dando origem a novos formalismos e notações em que os preceitos da OO provêm ao mesmo tempo uma forma de estruturação interna e uma disciplina para a produção dos modelos. Além disso, também é bastante atrativa a idéia de que a OO seja um meio de aproximar os métodos formais da prática.

Há, contudo, problemas a resolver. Não é fácil determinar os critérios para validar a integração entre uma linguagem formal e os conceitos de OO. De fato, há diversas formas de

³Durante o congresso de redes de Petri em Torino, Itália em 1995, foi realizado o primeiro *workshop* sobre o tema. Um segundo *workshop* foi realizado no ano seguinte, em Osaka, Japão.

fazê-lo, a depender dos objetivos específicos declarados. Daí a proliferação de uma grande variedade de dialetos e versões para cada linguagem formal a que se integram conceitos de OO. Neste sentido, redes de Petri são um bom exemplo do problema. São diversos os autores que defendem que a OO provê os elementos necessários para a estruturação e decomposição de modelos de redes de Petri. O que se verifica, contudo, são inúmeras notações de redes de Petri ditas baseadas ou orientadas a objetos que, via de regra, são incompatíveis e incomparáveis entre si.

Uma leitura inicial da bibliografia referente às várias tentativas de integrar OO e redes de Petri revela que um dos principais problemas dos formalismos resultantes é a falta de métodos e técnicas de análise. As notações propostas têm características tão diferentes das formas originais de redes de Petri, que tornam proibitiva a utilização das técnicas e dos métodos existentes. A forma como os conceitos de OO são introduzidos altera a dualidade original dos elementos de redes de Petri (compostas exclusivamente de lugares e transições, com propriedades bem específicas), introduzindo novos tipos de elementos que inviabilizam a aplicação da maioria dos métodos de análise—boa parte da teoria se sustenta sobre essa dualidade (ver [Pet86] sobre o assunto). Logo, um fator importante a ser considerado em qualquer tentativa de integração é a necessidade de preservar ao máximo as possibilidades de análise matemática de modelos. São estes os elementos que motivam e definem o problema investigado pelo trabalho aqui apresentado, que procuramos condensar na seguinte declaração.

O problema que motiva o desenvolvimento deste trabalho é a inadequação dos mecanismos de estruturação, decomposição e abstração ora disponíveis para a construção de modelos complexos em redes de Petri. Especificamente, investigamos a forma como os conceitos da orientação a objetos e de redes de Petri têm sido combinados, e as razões que têm impedido que as principais características de cada um desses "mundos" sejam preservadas nas notações derivadas, justificando a integração.

A solução de qualquer problema parte do entendimento de sua origem. Observe que nossa hipótese é que o problema acima tem origem na forma como os dois mundos são integrados. Em geral, a integração é conduzida enfocando-se os conceitos sintáticos da OO (notação, classes, herança, etc). Há um esforço enorme em redefinir redes de Petri

como classes, algumas relações estruturais entre redes como herança, etc. Em seguida, a semântica da notação é expressa através de modificações da regra de disparo. Muitas vezes, isto requer que sejam introduzidas alterações na natureza das redes para adaptá-las à *forma* como a orientação a objetos geralmente se apresenta. Um exemplo recorrente é a introdução de novos tipos de nós com características especiais para a interação entre objetos. Como os conceitos semânticos são deixados em segundo plano, a noção de estado é definida a partir do conceito de marcação, de tal forma que captura o “estado” de cada rede-classe. Além de pouco intuitivas, estas escolhas não facilitam em nada a adaptação de métodos de análise de redes de Petri (esta discussão será aprofundada no Capítulo 2). Diante disto, podemos declarar a idéia central defendida neste trabalho:

A integração dos conceitos de orientação a objetos às redes de Petri, com o propósito de adaptar mecanismos de abstração e estruturação, deve partir da escolha de um modelo semântico/conceitual consistente com ambos os mundos. Somente assim é possível definir uma notação que preserve as características fundamentais tanto da orientação a objetos e das redes de Petri, maximizando as chances de adaptação dos métodos de análise e da disciplina de modelagem da OO.

Boa parte deste documento consiste na organização dos diversos argumentos sobre os quais se sustenta a tese enunciada. Inicialmente, várias notações propostas e outros trabalhos sobre o tema são comparados e analisados. Dessa análise, derivamos as características necessárias a uma notação para que evite os problemas usuais. O corpo principal do documento é dedicado a demonstrar a aplicação da idéia no projeto de uma notação para a modelagem de sistemas distribuídos e concorrentes (orientada a objetos e baseada em redes de Petri, naturalmente). É apresentado o modelo conceitual sobre o qual se articula toda a notação. Cada conceito e noção é justificada em relação à classe de problemas que se pretende abordar: sistemas distribuídos e concorrentes de software. Também é apresentado o modelo semântico que formaliza as noções fundamentais de objetos. Cada aspecto é devidamente formalizado e discutido, antecipando a necessidade que haverá de integrar esse formalismo às redes de Petri. Finalmente, é apresentada a notação, cuja semântica é expressa, compondo o modelo semântico ao comportamento de redes de Petri.

1.3 Estrutura deste documento

O restante deste documento está organizado em 6 capítulos:

Capítulo 2 Revisão Bibliográfica

Na revisão bibliográfica apresentamos, em maiores detalhes, os fatores que motivam a integração dos conceitos da orientação a objetos às redes de Petri. Também revisamos os trabalhos mais relevantes sobre o assunto e avaliamos as principais propostas de integração. O capítulo é finalizado com uma análise crítica das características das abordagens de integração e dos resultados obtidos.

Capítulo 3 Modelo Conceitual

O tema principal do Capítulo 3 é a identificação de um modelo conceitual que servirá de base para a notação que apresentaremos nos demais capítulos. Inicialmente, contudo, identificamos as características de uma *boa* integração. Em seguida, apresentamos um anteprojeto que determina a forma como a integração é elaborada para garantir as *boas* características. Ao final do capítulo, apresentamos o modelo conceitual de computação que fundamenta a proposta de integração.

Capítulo 4 Sistemas de objetos

Neste capítulo, a ênfase é a identificação dos conceitos semânticos envolvidos. Como a caracterização semântica de redes de Petri pode ser considerada convencional, o capítulo é dedicado à formalização semântica do que entendemos de orientação a objetos concorrentes. Uma definição de sistemas de objetos é desenvolvida independentemente de redes de Petri. São enfatizados aspectos de interconexão entre os objetos e o comportamento concorrente.

Capítulo 5 Redes de Petri Orientadas a Objetos

Este capítulo apresenta a notação que integra orientação a objetos e redes de Petri propriamente dita. Redes de Petri são utilizadas para descrever o comportamento dos objetos de um sistema. Aspectos de decomposição do modelo, por outro lado, podem ser abordados através das construções convencionais da orientação a objetos. A semântica da notação é expressa combinando a semântica de redes de Petri à definição de sistemas de objetos desenvolvida no capítulo anterior. Também é apresentada

uma concretização da notação, baseada em redes de Petri coloridas, que permite a utilização prática da integração.

Capítulo 6 Validação e análise de modelos

Neste capítulo, avaliamos a notação proposta, discutindo as possibilidades de adaptar métodos de análise. Demonstramos que ao descrever objetos através da notação proposta, o comportamento efetivo observável é sempre uma restrição do comportamento descrito pela classe a que pertence. Este resultado permite aplicar *diretamente* os métodos de redes de Petri na análise do corpo das classes. Além disso, provê bases formais de suporte à elaboração de classes reutilizáveis de modelos. Também definimos os conceitos fundamentais para suportar a análise de espaços de estados. Finalmente, consideramos uma noção de equivalência semântica entre modelos na notação proposta e modelos expressos em redes de Petri puras.

Capítulo 7 Conclusão

Finalmente, apresentamos nossas conclusões sobre o projeto como um todo, indicando possíveis melhorias que ainda podem ser feitas, bem como problemas enfrentados durante o processo. Finalmente, concluímos o trabalho, enumerando alguns trabalhos que se originaram deste projeto.

Apêndice A Um estudo de caso

No apêndice apresentamos um estudo de caso em que experimentamos a notação em um contexto mais realista. O problema abordado é a modelagem de um sistema distribuído de licenças de software. O modelo é construído a partir de uma especificação informal.

É importante alertar que o texto foi escrito assumindo uma certa familiaridade do leitor com os conceitos de orientação a objetos e com redes de Petri. Os aspectos formais tratados não demandam mais que certa exposição à teoria dos conjuntos e modelos de computação. Não há nenhuma seção em que estes temas sejam especificamente introduzidos neste texto, mas referências são dadas a seguir.

Em seu prestigiado tutorial [Mur89], Murata apresenta uma visão geral da área de estudo de redes de Petri. São apresentadas a motivação, definições básicas e uma visão

panorâmica sobre as principais técnicas de análise. O texto também introduz as redes de Petri de alto nível através das redes Predicado/Transição. Redes de Petri Coloridas (CP-nets), seus respectivos métodos de análise e diversas aplicações industriais são apresentadas por K. Jensen em três volumes sobre o tema [Jen92; Jen94; Jen98].

Orientação a objetos é assunto para o qual não faltam referências. Os conceitos fundamentais da disciplina são tratados por Booch, Rumbaugh e Jacobson em diversos livros (alguns são [RBP⁺91; JCJO92; RBP⁺91; JCJO92]). Os mesmos três autores reuniram seus trabalhos, dando origem à notação denominada de UML (*Unified Modelling Language*) que é apresentada em três volumes [BRJ99; RJB99; JBR99]. A confluência deste tema com concorrência ainda é tema de pesquisa. Alguns trabalhos podem ser encontrados nas coletâneas de artigos sobre o tema em [AFK⁺93] e mais recentemente em [AdC01]. Outros trabalhos com propósitos e idéias semelhantes às usadas aqui foram desenvolvidos para integrar álgebras de processo a outros formalismos e aos conceitos da orientação a objetos (ver [Fis96; Fis97; SSC01; MS01]).

Capítulo 2

Revisão Bibliográfica

Neste capítulo, apresentamos os fatores que motivam a integração da orientação a objetos às redes de Petri. Também revisamos os principais trabalhos sobre o tema e caracterizamos as abordagens de integração mais relevantes. Ao final discutimos as características dessas abordagens e identificamos as razões pelas quais os resultados não são efetivos em permitir a fácil adaptação das técnicas e métodos convencionais utilizados para analisar e construir modelos.

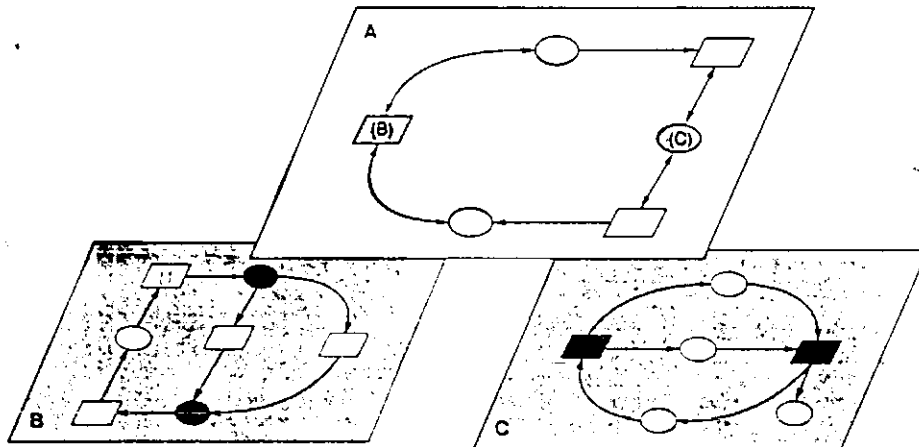
2.1 Abstração e decomposição em redes de Petri

Facilidades de abstração e (de)composição são fundamentais no processo de descrição de sistemas complexos [BG86; Van98]. Portanto, é imperativo que linguagens de especificação ou modelagem de sistemas complexos incluam mecanismos para suportar adequadamente tais facilidades. Entretanto, é exatamente este um dos principais problemas apontados em relação à modelagem por redes de Petri: a precariedade dos mecanismos disponíveis para a abstração e a estruturação dos modelos [Jen92; HJS91; CKR94; Lak97].

Uma forma simples de (de)compor redes de Petri é obtida através de mecanismos primitivos baseados na *fusão* de nós e na adição de arcos. Neste caso, a modelagem consiste em elaborar um conjunto de *redes-parte* e, ao final, compor uma única *rede-todo* que modela o sistema completo através da fusão das estruturas das *redes-parte*. O principal problema deste mecanismo é a fraca noção de abstração que proporciona. Observe que não há regras que controlem como as partes devem ser integradas. Em princípio, a fusão de nós e a adição de arcos é livre de qualquer restrição. Cabe exclusivamente ao projetista determinar

e seguir regras de composição, a fim de minimizar problemas de integração e preservar as características interessantes das partes. Ou seja, o mecanismo não tem propriedades claras que reforcem a relação contratual entre as partes do modelo. Isto, por sua vez, precariza as condições de abstração do projetista.

Redes hierárquicas Para resolver este problema, foi introduzida a noção de *decomposição hierárquica* de redes de Petri. A idéia básica das redes hierárquicas é construir modelos em que determinadas partes da rede sejam abstraídas e representadas por um único nó. Cada nó deste tipo é dito um *nó de substituição*—por substituir toda uma sub-rede. A Figura 2.1 ilustra a idéia. O modelo é expresso em dois níveis de abstração diferentes, representados pelos diferentes planos na figura. A rede *A*, no plano de nível mais alto, é a representação abstrata do sistema. Os nós destacados são nós de substituição que representam as redes *B* e *C*, expressas no segundo nível de abstração. Redes com nós de substituição, como *A*, são ditas *redes abstratas*.



- Figura 2.1: Mecanismos de decomposição hierárquica de redes de Petri

Para formalizar a relação, é necessário que as redes possam ser “encaixadas” nas posições indicadas pelos respectivos nós de substituição. Para isto, os nós vizinhos a um nó de substituição devem ser incluídos na estrutura da rede abstraída—esses nós são ditos *nós de borda*. No exemplo da Figura 2.1, os nós de borda das redes *B* e *C* são destacados dos demais. Observe que os nós de borda devem ser do mesmo tipo dos nós vizinhos do nó de substituição. Logo, os nós de borda de uma rede representada por uma transição de substituição são lugares e vice-versa.

Na prática, a popularidade da ferramenta *Design/CPN* [SCK97] torna as redes H-CPN (*Hierarchical Coloured Petri Nets*), definidas por Huber, Jensen e Shapiro em [HJS90; HJS91; Jen92], a versão mais utilizada de redes hierárquicas. Há, contudo, algumas outras ferramentas que implementam variações da idéia básica.

A abstração em redes hierárquicas A decomposição hierárquica introduz uma forma rudimentar de abstração. De certa forma, os nós de borda podem ser vistos como a especificação de uma *interface contratual* entre as redes. Além disso, o mecanismo pode ser aplicado recorrentemente, o que permite decompor um modelo em um número arbitrário de redes em diversos níveis. Contudo, trata-se ainda de uma forma de “substituição textual” e não de um mecanismo efetivo de abstração em que os detalhes referentes a níveis mais concretos podem ser realmente ignorados no entendimento dos níveis mais abstratos.

A principal razão para isto é que a decomposição hierárquica foi introduzida exclusivamente com o propósito de “*permitir ao projetista construir um modelo, combinando um certo número de redes pequenas em uma única rede maior*” (cf. [Jen92, p. 90]). Logo, o significado de um modelo hierárquico é expresso como o significado da rede obtida pela justaposição das redes-parte. Em última instância, isto é equivalente à composição de redes mediante a fusão de nós. O princípio é compor as descrições propriamente ditas, não seus significados.

Mas também há razões teóricas que tornam essa abordagem pouco atrativa. Relembre que, convencionalmente, transições modelam ações indivisíveis mas não modelam estado. Analogamente, lugares modelam estado mas não ações. Contudo, uma transição de substituição representa todo um subconjunto de eventos do sistema e também pode modelar parte de seu estado. Isto dificulta sua interpretação como uma transição no sentido formal de redes de Petri e, conseqüentemente, também dificulta a interpretação de redes abstratas como redes de Petri.

Este problema pode ser melhor compreendido avaliando o modelo esquemático da Figura 2.1 na página anterior. A questão é: como podemos definir o significado da rede *A* independentemente das redes *B* e *C*, de tal forma que seja efetivamente uma abstração da rede obtida pela substituição dos nós de substituição? Não há solução única nem consensual para esta questão. A abordagem genérica consiste em determinar uma noção de equivalência que possa ser garantida através de uma relação entre os nós de substituição

na rede abstrata e as redes concretas representadas. O problema é que noções fortes de equivalência determinam relações comportamentais que não podem ser garantidas apenas por restrições estruturais. Por oposição, relações que possam ser garantidas dessa forma tendem a determinar noções de equivalência extremamente fracas e que se revelam pouco úteis na prática.

É por estas razões que o significado de redes abstratas em modelos hierárquicos não é definido de forma independente das redes correspondentes a seus nós de substituição. De fato, o comportamento de uma rede hierárquica é expresso em termos da rede resultante da substituição de todos os nós de substituição pelas redes correspondentes [Jen92, p.113-115]. Nem o comportamento das redes abstratas, nem o das partes é definido independentemente. Para efeitos de modelagem, contudo, o projetista assume que o comportamento das partes e das redes abstratas é "semelhante" ao de uma rede de Petri não-hierárquica.

Na prática, há ainda uma restrição adicional. É comum que o uso da decomposição hierárquica se restrinja ao uso de transições de substituição. Algumas dificuldades teóricas foram encontradas para definir lugares de substituição adequadamente (ver [Jen92, p. 119] sobre o tema). A ferramenta *Design/CPN*, por exemplo, suporta apenas transições de substituição e fusão de lugares como mecanismos de integração.

Críticas à decomposição hierárquica A definição da semântica da decomposição tem diversas implicações. Em primeiro lugar, a abstração resultante é essencialmente visual ou estrutural. Isto é, permite a decomposição da descrição, mas não provê uma relação real de abstração que permita (de)compor o significado do modelo e usar essa decomposição para a análise formal. Segundo Cherkasova, a modelagem de sistemas complexos demanda facilidades *reais* de abstração que permitam efetivamente a modelagem em diversos níveis e que, portanto, vão além das proporcionadas por redes hierárquicas [CKR94]. Isto requer que o significado de cada nível de abstração possa ser expresso e avaliado independentemente dos demais, dentro de um mesmo arcabouço semântico. Contudo, devido à fusão dos lugares de borda para obter a rede total, o comportamento de cada módulo pode ser completamente modificado pela rede abstrata em que é usado. Ou seja, o mecanismo não determina uma forma efetiva de encapsulamento (*information hiding*) entre as partes. Logo, não proporciona condições adequadas para a modelagem de módulos efetivamente reutilizáveis.

Em segundo lugar, não é possível definir métodos genéricos para analisar os módulos independentemente. Isto ocorre porque não há uma forma determinada de interpretar os resultados obtidos que leve em consideração como cada módulo pode vir a ser integrado. Embora a integração seja determinada pelos lugares de borda, não há restrições quanto à forma de acesso a esses lugares por parte da rede abstrata. Conseqüentemente, o comportamento efetivo de cada módulo pode ser livremente determinado pela rede abstrata em que é usado. Novamente, cabe apenas ao projetista definir e seguir uma disciplina de modelagem para reduzir problemas de integração e, desta forma, poder analisar independentemente os módulos através dos métodos convencionais de redes de Petri. Logo, podemos dizer que o esquema de decomposição hierárquico não proporciona condições adequadas ao desenvolvimento de métodos composicionais de análise. Evidências destes problemas podem ser observadas pelas ferramentas de análise desenvolvidas para redes hierárquicas. A grande maioria, incluindo as acopladas ao *Design/CPN*, funciona a partir do desdobramento do modelo em uma única rede.

Modelagem de sistemas distribuídos A forma como os componentes de um sistema distribuído se ligam uns aos outros para efeitos de interação é denominada de *topologia de interconexão* (ver [Hew77; Agh86; AFK⁺93]). Em alguns casos, a topologia é estática, ou seja, os componentes e suas ligações permanecem constantes no decorrer de todo o funcionamento do sistema. No caso geral, contudo, a topologia de um sistema pode ser alterada dinamicamente durante seu funcionamento. Embora o conceito seja antigo, na prática, têm se revelado extremamente atuais, devido à demanda crescente de sistemas dessa natureza (especialmente sistemas baseados na *web*). Contudo, as redes hierárquicas e seus mecanismos de abstração não suportam a modelagem direta de tal classe de sistemas.

O problema está relacionado ao conceito de módulo adotado em redes hierárquicas. Devido à forma estática como deve ser integrado ao restante de um modelo, uma rede-módulo não provê os elementos necessários para modelar componentes de sistemas com topologias dinâmicas. Questões como a criação e a destruição de componentes dinamicamente têm que ser tratadas através de codificações e convenções explicitamente introduzidas e controladas pelo projetista. Além disso, pelas razões já discutidas, o conceito de nó de borda também não é adequado para modelar a interface contratual de um componente desta natureza. Em princípio, componentes não necessariamente compartilham estado. Além

disso, a forma de interação depende do meio de comunicação disponível—ora síncrono, ora assíncrono.

Em suma, embora seja desejável utilizar redes de Petri e os métodos de análise já consolidados para modelar e analisar essa classe de sistemas, os elementos de linguagem proporcionados por redes hierárquicas não têm se mostrado satisfatórios (ver [CKR94; Lak95]).

Orientação a objetos É principalmente com o propósito de compensar as deficiências indicadas que diversos pesquisadores têm proposto integrar conceitos da orientação a objetos às redes de Petri. De certo modo, podemos dizer que a idéia é *natural*. Afinal, a orientação a objetos tem sido proposta exatamente para corrigir deficiências semelhantes em outras classes de linguagens: para enriquecer os conceitos de modularização e abstração; para enriquecer a linguagem, permitindo a efetiva reutilização de modelos; para introduzir uma certa disciplina de descrição, etc. Em nosso caso específico, há ainda outra razão: o modelo computacional básico da orientação a objetos é bastante adequado para a modelagem de sistemas distribuídos com topologias dinâmicas de interconexão. O conceito de objeto pode ser facilmente “adaptado” para modelar os componentes de um sistema (sobre orientação a objetos e concorrência, ver [AWY93] e mais recentemente [AdC01]). Na seção seguinte, identificamos e comentamos os principais trabalhos e resultados sobre o tema.

2.2 Orientação a objetos e redes de Petri

Diversos trabalhos procuram integrar conceitos da orientação a objetos às redes de Petri. Contudo, devido à diversidade de motivações e propósitos específicos envolvidos, nem sempre é possível comparar esses trabalhos diretamente. Nesta seção, descrevemos e avaliamos os principais trabalhos que influenciaram este projeto. Inicialmente, apresentamos as abordagens com motivações e propósitos semelhantes aos nossos. Também apresentamos alguns trabalhos que, apesar dos diferentes propósitos, são relevantes no contexto deste trabalho. Finalmente, apresentamos um breve comentário sobre esforços de integração de outras classes de linguagens formais de especificação e dos conceitos da orientação a objetos.

2.2.1 Principais abordagens

Cinco propostas de integração são apresentadas nesta seção. Podemos dizer que cada um dos trabalhos tem o propósito de desenvolver extensões de redes de Petri em que a modularização seja baseada nos conceitos da orientação a objetos. Apesar de compartilharem o propósito, os resultados obtidos diferem bastante uns dos outros.

OPN Certamente, a principal proposta de integração entre redes de Petri e orientação a objetos é o trabalho desenvolvido por C. Lakos. O formalismo proposto, denominado OPN (*Object Petri Nets*), é um dos trabalhos mais divulgados sobre o tema. A motivação e os propósitos são bastante similares aos nossos, como pode ser visto por sua própria descrição:

*“O propósito de OPNs é proporcionar uma integração completa da orientação a objetos ao formalismo de redes de Petri e dessa forma aproveitar os benefícios complementares desses dois paradigmas. (...) Redes de Petri têm uma representação gráfica natural que contribui para o entendimento das especificações, além de uma variedade de técnicas de análise (...) A orientação a objetos, por outro lado, tem se tornado extremamente popular por proporcionar facilidades de estruturação que enfatizam o encapsulamento e promovem a reutilização de software. Isto ataca um aspecto tradicionalmente deficiente em redes de Petri, precisamente o suporte inadequado à composicionalidade.” [Lak37, p. 1]*¹

Lakos define OPN como *uma extensão ao formalismo de CPN que suporta os conceitos de classe, herança, polimorfismo e ligação dinâmica* [Lak95]. Segundo o autor, a característica mais importante de OPN é a adoção de uma hierarquia única de classes para integrar tipos de redes e tipos de fichas [Lak37]. Esta característica, ainda segundo o autor, é o fator chave para suportar a modelagem direta de sistemas com múltiplos níveis de atividade.

Cada classe OPN equivale a uma página em modelos hierárquicos. Lugares são vistos como campos de dados e podem comportar valores simples ou multi-conjuntos de valores como na definição convencional de redes de alto nível. Além de transições, cada classe pode ter também *funções* que permitem a avaliação de condições sobre o estado das redes através de arcos somente de leitura. Isto significa que a avaliação de funções não restringe as condições de ocorrência do restante da rede.

¹Tradução livre.

É possível relacionar classes OPN através de um mecanismo de herança. Ao estabelecer tal relação, a subclasse pode herdar a descrição dos métodos (transições e funções) e atributos definidos na superclasse. É possível controlar quais métodos devem ser eliminados e redefinidos em uma relação de herança. Por convenção, a rede deve ser “copiada” graficamente da superclasse para a subclasse para efetuar quaisquer alterações. Segundo o autor, isto facilita a compreensão da subclasse [Lak37, p. 6].

Os conceitos de hierarquias também são mantidos através de nós de substituição, aqui denominados de *super-lugares* e *super-transições*. A principal diferença é que as sub-redes são consideradas classes cujo acesso é restrito a arcos incidentes sobre nós do tipo complementar. Ou seja, super-transições são definidas com *arcos de borda* incidentes sobre lugares e vice-versa.

A linguagem LOOPN++ é uma alternativa textual para a descrição de sistemas em OPN enquanto não são desenvolvidas ferramentas gráficas apropriadas. A maior parte das aplicações têm sido demonstradas na modelagem de protocolos de redes [LK91; LK95].

CO-OPN/2 Originalmente, o formalismo denominado CO-OPN (*Concurrent Object-Oriented Petri nets*) foi introduzido por Buchs e Guelfi com o propósito de “*oferecer uma estrutura adequada para a especificação e o projeto de sistemas concorrentes de grande porte*” [BG91]. A extensão denominada CO-OPN/2 incorpora suporte adequado aos conceitos de classificação, herança e tipos [BBG01]. Formalmente, são definidos como uma extensão com características de orientação a objetos para redes algébricas definidas por Vautherin e Reisig [Vau90; Rei91].

Objetos em CO-OPN são vistos como entidades que agregam uma estrutura de dados e comportamento. A primeira é especificada através de um formalismo algébrico para tipos abstratos de dados. E redes de Petri são utilizadas para definir o comportamento concorrente, tanto dos objetos isoladamente quanto do sistema como um todo. Expressões de sincronização permitem restringir a ocorrência de eventos dentro de condições desejadas de comportamento. A especificação através de CO-OPN consiste em três etapas:

1. definição dos tipos de dados sobre os quais os objetos operam através de um formalismo algébrico (PLUSS em CO-OPN [Bid87]);
2. definição do conjunto de objetos e suas respectivas redes algébricas;

3. definição de uma série de relações entre os objetos e seus eventos.

O comportamento de cada objeto é definido por uma rede algébrica estendida com transições parametrizadas que representam os métodos do objeto, que por sua vez agem como transformadores sobre os dados descritos. A interação entre objetos é restrita a sincronizações entre as transições parametrizadas.

Uma relação de equivalência sobre o conjunto de objetos é definida a partir de uma relação de equivalência observacional entre especificações algébricas e outra de equivalência baseada em bissimulação entre redes. Desta forma, os autores argumentam que promovem a especificação através de refinamentos sucessivos pela substituição de componentes por outros equivalentes. Esta relação é a base para a definição do conceito de especialização (ou subtipificação) de CO-OPN/2. Contudo, a relação é comportamental e, portanto, não pode ser garantida por relações estruturais sobre as classes, embora possa ser verificada. Por esta razão, os autores incluem um mecanismo sintático de herança que facilita a descrição de classes a partir de outras.

Objetos Cooperativos ou Redes Cooperativas O formalismo proposto por Sibertin-Blanc baseia-se na separação entre objetos ativos e passivos de um sistema. Desta forma, objetos passivos, modelados através de fichas, podem ser manipulados por redes de Petri. Por outro lado, o comportamento de objetos ativos pode ser modelado através de redes de Petri. A interação entre as redes correspondentes aos objetos ativos é baseada no protocolo cliente-servidor, daí porque são denominadas *redes cooperativas* [SB93; SB94]. Redes clientes podem requisitar a execução de serviços em redes servidoras, que são compostas por um conjunto de redes de serviços. O significado de um modelo é obtido através da construção de uma única rede composta a partir das redes que modelam os objetos.

Embora redes clientes e servidoras sejam definidas separadamente, os papéis podem ser assumidos simultaneamente por qualquer rede do sistema, permitindo que alguns serviços façam uso de outros serviços em outras redes. Um serviço ocorre mediante a transferência de fichas de uma rede cliente para uma rede servidora. Redes cliente têm necessariamente duas transições associadas à interação com a rede servidora. Uma das transições modela a requisição do serviço, e a outra modela a recepção do resultado. De forma correspondente, as redes servidoras têm duas transições para a interação com os clientes. Uma transição

modela a aceitação da requisição do serviço e a outra modela a entrega dos resultados computados. Para cada transição relacionada com a interação, acrescenta-se um lugar correspondente para obter uma rede complemento. Desta forma, redes cliente têm um lugar-de-requisição, e um lugar-de-recepção (*request-place* e *accept-place*). Redes servidoras têm um lugar-de-aceitação e um lugar-de-entrega (*accept-place* e *provide-place*). O modelo integrado cliente-servidor é obtido através da fusão dos lugares de requisição aos lugares de aceitação e dos lugares de recepção aos lugares de entrega para cada serviço.

O autor define as propriedades de “honestidade” e “discrição” que devem ser satisfeitas pelas redes cliente e servidora para garantir certas propriedades da comunicação no modelo integrado da rede. A rede cliente deve ser “honesto” em só buscar resultados se antes houver feito a requisição do respectivo serviço. Além disto deve ser “discreto” em necessariamente buscar os resultados após ter feito uma requisição. Do outro lado, a rede servidora também deve agir “honestamente” e só deve aceitar a requisição de um serviço se realmente puder processá-lo. Também deve ser “discreto” e só deve entregar resultados de serviços que realmente foram requisitados. A honestidade e a discrição de redes clientes podem ser garantidas através de restrições puramente estruturais. Para redes servidoras, entretanto, o autor apresenta condições comportamentais que devem ser verificadas para caracterizar sua honestidade e discrição.

G-Nets e G-CPN A classe de redes denominada *G-Nets*, originalmente proposta por Deng e Chang, é definida pelos autores como “*uma abordagem formal para a especificação, modelagem e prototipagem rápida de sistemas distribuídos*” [DC91; DC92; DCdFP93; PdF97]. Um *Sistema G-Net* é constituído por um conjunto de módulos (também denominados *G-Nets*) que encapsulam operações e atributos. Cada módulo é caracterizado pela declaração explícita de uma interface, denominada *GSP (Generic Switching Place)* e uma estrutura interna, denominada *IS (Internal Structure)*, que detalha o funcionamento dos métodos.

Na definição original, o modelo *G-Net* é baseado em redes Predicado/Transição (*Pr/T Nets*) modificadas. A principal modificação consiste em associar uma nova semântica a alguns lugares da estrutura, denominados *ISPs (Instantiating Switching Places)*. Tais lugares correspondem a abstrações da operação de outros módulos do sistema. Assim, fichas depositadas em um *ISP* são retiradas e “transportadas” para o módulo correspondente a fim

de que o serviço invocado seja executado. O término do serviço em um módulo invocado é verificado quando fichas forem depositadas em lugares especiais denominados *lugares-alvo*. Após o término do serviço as fichas colocadas em lugares-alvo são transportadas de volta para o módulo que invocou o serviço.

A abordagem denominada G-CPN (ver [Gue97; GdFP97; GdFP01]) é baseada nos mesmos princípios que originaram as G-Nets. A principal diferença é que em G-CPN são utilizadas redes coloridas para modelar a estrutura interna dos módulos. Em G-CPN um método pode ser invocado simultaneamente (concorrentemente) por vários módulos. Também foram introduzidos os conceitos de *contexto* e *ambiente de interação*. Um contexto corresponde a uma instância de um módulo criada para atender requisições concorrentes de uma invocação. Logo, para cada invocação um novo contexto é criado. O ambiente mantém a relação de dependência entre os contextos ativos do sistema e é responsável pelo transporte de fichas entre os módulos do sistema.

Uma diferença adicional entre G-Nets e G-CPN é que atributos são explicitamente modelados por lugares nas redes G-CPN. Isto permite utilizar redes coloridas sem modificações como estruturas internas dos módulos. Além disso, simplifica o procedimento de obtenção de uma rede colorida equivalente.

OBJSA e CLOWN As redes OBJSA, propostas por Battiston, de Cindio e Mauri [BdCM88; BCM96], proporcionam um mecanismo modular para a construção de redes baseadas na superposição de autômatos e na fusão de transições de *componentes* OBJSA. O enfoque é a agregação de aspectos de concorrência e sincronismo (controle) a técnicas algébricas de descrição de tipos de dados. O modelo é baseado na linguagem OBJ2 (ver [Gog84]) à qual são incluídas descrições de ciclos de vidas de objetos baseadas em superposição de autômatos finitos. A idéia é descrever cada comportamento isolado como um autômato e integrar as descrições através da superposição dos autômatos de acordo com rótulos associados aos eventos, obtendo redes de Petri.

As redes OBJSA determinam as condições mínimas necessárias para a descrição da semântica de linguagens de especificação de sistemas concorrentes orientadas a objetos. Nenhum aspecto de classificação, herança ou tipos é diretamente suportado. Essas questões são tratadas através do formalismo denominado CLOWN para a especificação de sistemas, cuja semântica é dada em termos de redes OBJSA a partir da combinação das redes SA

(*superposed automata*).

2.2.2 Outros trabalhos

Há dois outros tipos de abordagens que destacamos. A primeira propõe e investiga o uso de redes de Petri como formalismo para caracterizar e analisar a semântica de linguagens de programação e especificação concorrentes orientadas a objetos². A segunda procura usar linguagens orientadas a objetos para a descrição e manipulação de tipos de dados em redes de alto nível. Apesar das diferenças de propósitos, alguns desses trabalhos identificam relações relevantes entre conceitos da orientação a objetos e de redes de Petri que interessam do nosso ponto de vista.

POTs & POPs Os modelos POTs & POPs (*Parallel Object-based Transition System e Parallel Object-based Programs*) foram introduzidos por Engelfriet, Leih e Rozenberg [ELR90] como esquemas para denotar a semântica formal de sistemas de computação paralelos baseados em objetos. POTs são sistemas de transição paralelos baseados em objetos. POTs podem ser vistos como redes de Petri (potencialmente infinitas) baseadas em objetos que incorporam conceitos como identidade, ciclo de vida, memória privada e referências a objetos. POPs são usados para elaborar descrições finitárias de baseados em objetos, cuja semântica é dada através de POTs.

THORN As redes THORN (*Timed Hierarchical Object-Related Nets*) foi proposta como uma linguagem que “*deve oferecer modelagem simples, abstração e reusabilidade associadas com a possibilidade de execução eficiente da simulação em computadores*” [SSW95]³. Em redes THORN, fichas são objetos instanciados de classes C++. Os lugares podem ser estruturados como multi-conjuntos, pilhas, filas ou ainda filas com prioridades. Assim, objetos são manipulados através de operações válidas para essas estruturas de dados. Cada lugar é associado a uma classe, de forma que apenas objetos dessa classe podem ser contidos. Os arcos de THORNs podem ser arcos de teste ou inibidores. Transições também podem

²Apesar da separação, algumas das abordagens apresentadas na seção anterior também são propostas pelos autores como formalismos adequados à caracterização da semântica de linguagens concorrentes. É o caso de OBJSA e CLOWN, por exemplo.

³Tradução livre

ser inscritas com guardas (denominadas condições de ativação), blocos de ação e funções de tempo em C++, que são verificadas e/ou executadas durante o disparo da transição.

Um dos principais problemas é a imprecisão das definições formais do modelo. Os autores argumentam que tal fato é devido ao uso de tipos nas fichas e ao forte compromisso com a “eficiência da simulação distribuída” dos modelos através de uma ferramenta de software. Apesar disto, o uso de uma linguagem de programação eficiente e amplamente usada, garante a facilidade de assimilação do modelo por parte de projetistas e programadores. É claro que em compensação perde-se em abstração e distância dos aspectos de solução envolvidos numa modelagem conceitual do sistema.

Outro experimento relacionado, embora não tenha tido continuidade, é a tentativa do grupo de redes de Petri da universidade de Aarhus de utilizar a linguagem orientada a objetos *BETA* para inscrever redes coloridas e para declarar os tipos de dados [CT93].

Ainda outros trabalhos As descrições das abordagens apresentadas consistem em uma revisão e atualização de um relatório de pesquisa sobre o tema [Gue98]. Naturalmente, há diversos outros trabalhos que não foram incluídos aqui. Contudo, consideramos que os aqui apresentados formam uma amostra adequada e relevante das principais idéias e abordagens utilizadas. Recentemente, foi lançada uma coleção de diversos trabalhos sobre o tema em que cada uma das propostas é detalhadamente descrita (ver [AdC01]).

2.3 Avaliação crítica

Qualquer avaliação depende da escolha de um ponto de vista. Em nosso caso, esse ponto de vista é determinado pelo problema de pesquisa declarado (ver página 8). Isto é, interessamos avaliar a forma como a integração é feita e como isso afeta as características e as propriedades da orientação a objetos e de redes de Petri no formalismo resultante. A partir dessa perspectiva, fazemos nossa análise considerando três aspectos: o tratamento dado aos conceitos de redes de Petri, aos conceitos de orientação a objetos e à forma como a integração é efetivamente obtida. Este último ponto pode ser entendido como avaliação da caracterização da semântica do formalismo.

2.3.1 Sobre os conceitos de redes de Petri

Um dos fatores que dificultam a comparação entre as abordagens de integração é a adoção de diferentes formalismos de redes de Petri. Do ponto de vista dos grupos de pesquisa, o objetivo não é apenas determinar uma abordagem de integração. Trata-se também de obter uma notação concreta que possa ser utilizada em contextos de interesse. Devido a isto, não são adicionados apenas conceitos da orientação a objetos, mas também diversas outras extensões e facilidades. Se, por um lado, isto efetivamente melhora a expressividade da notação obtida, por outro dificulta a avaliação das propriedades específicas da integração proposta.

Extensões Das abordagens apresentadas na seção anterior, não há duas que compartilhem uma mesma definição básica de redes de Petri. Redes OPN usam variações de redes coloridas; CO-OPN e CO-OPN/2 são definidas a partir de redes algébricas; redes cooperativas usam uma definição própria de redes de alto nível; G-Nets usam redes Predicado/Transição e G-CPN, redes coloridas; e OBJSA usa redes algébricas. Embora todas partam de redes de alto nível, as diferenças entre as versões concretas apenas dificultam a comparação dos aspectos que efetivamente interessam. Por exemplo, redes OPN são definidas usando diversas extensões não usadas em outras abordagens: arcos de teste, arcos inibidores, políticas de acesso às fichas depositadas nos lugares (*FIFO*, *LIFO*, etc). Embora sejam extremamente convenientes para a modelagem, estas características não podem ser consideradas consequência da forma de integrar os conceitos de orientação a objetos às redes de Petri. E, portanto, não podem ser interpretadas como indicações da melhor qualidade de uma abordagem em relação à outra.

De fato, algumas extensões podem até mesmo introduzir problemas. Muitos métodos de análise para redes de alto nível são adaptações de métodos para redes de baixo nível. Em vários casos, o método só pode ser aplicado se a rede de alto nível admite uma equivalente em baixo nível. Logo, ao introduzir extensões é necessário considerar como as possibilidades de análise são afetadas no caso geral.

Interação e abstração Os mecanismos de interação entre os módulos usados em diversas propostas (por exemplo OPN, G-Nets, G-CPN, Objetos Cooperativos) têm problemas semelhantes aos descritos para redes hierárquicas (ver página 15). Em OPN, os nós podem

ser super-lugares ou super-transições. Como o mecanismo é o mesmo adotado para os nós de substituição em redes hierárquicas, os mesmos problemas podem ser observados. Embora os lugares denominados ISPs em G-Nets sejam propostos como abstrações de outras redes do modelo, o mecanismo utilizado consiste em “transportar” fichas entre as redes que interagem. A semântica convencional da rede é modificada e o projetista deve considerar que as marcações de ISPs podem ser alteradas “espontaneamente”. Ou seja, que mesmo sem a ocorrência de nenhuma transição local, o estado do lugar (e da rede, portanto) pode ser alterado. Em G-CPN, os lugares ISP são substituídos explicitamente por lugares de entrada e lugares de saída. A separação permite distinguir quais lugares podem perder fichas e quais podem recebê-las. Redes cooperativas e CO-OPN usam esquemas semelhantes. Embora isto facilite o controle do projetista sobre cada rede, ainda é necessário alterar a semântica convencional. Além disso, este estilo de abstração pode ser dita “horizontal” ao invés de “vertical”, como é proposto para a modelagem de vários níveis de abstração. Ou seja, permite que os detalhes de partes do sistema sejam abstraídos do ponto de vista de outra parte. Mas não proporciona meios adequados para abstrair o próprio comportamento de um único módulo.

A solução para o problema consiste em encontrar mecanismos que não quebrem o *princípio da dualidade* (ver [Lak97]). Diversas técnicas de análise são definidas a partir desse princípio. Ele determina que deve haver apenas dois tipos de elementos em uma rede, digamos lugares e transições, e que os dois devem modelar o mundo dividindo-o dicotomicamente em componentes estáticos e dinâmicos. Desta forma, lugares devem ser exclusivamente depósitos de informação ou condições. E transições, apenas descrições de atividades e de mudanças de estado. Logo, um mecanismo é adequado se permite que um nó abstraia uma sub-rede, preservando sua definição convencional.

2.3.2 Sobre os conceitos de orientação a objetos

Há uma grande preocupação por parte dos autores em garantir que as características da orientação a objetos sejam realizadas por suas propostas. Um dos principais problemas, contudo, é que dificilmente dois autores concordam quanto a quais são essas características e quais são suas definições.

Objetos Um dos conceitos centrais na orientação a objetos é o *objeto*. Embora sua definição seja considerada ponto passivo entre os autores, é comum que seja abordado de formas diferentes, o que revela que, de fato, o conceito é interpretado de diversas formas.

O principal problema é o desbalanceamento da relação *modelo-instância*. Conceitualmente, classes são descrições de objetos—modelos, portanto. Objetos, por outro lado, são *instâncias* de classes. Isto é, entidades criadas a partir de um “molde”—a classe. A maioria das propostas, contudo, falha em estabelecer essa relação adequadamente. É comum, que o conceito de classe seja uma reinterpretação do conceito de módulo. Logo, cada classe é vista como uma sub-rede de uma rede maior que modela todo o sistema e que deve ser obtida combinando as redes-classe. Neste caso, é comum que o conceito de objeto sequer seja formalmente definido.

Em outros casos, OPN por exemplo, as fichas contidas nos lugares correspondem a instâncias de outras redes. Há dois problemas com esta abordagem. Primeiro, é necessário garantir que toda ficha em um determinado nível de abstração representa um objeto diferente⁴. Isto dificulta a expressão de múltiplas condições sobre um objeto através de vários lugares na rede—um mesmo objeto (ficha) não pode estar contido em dois lugares simultaneamente. Segundo, é necessário considerar que as transições apenas “movem” as fichas-objetos de um lugar para outro. Ou seja, todas as transições, exceto as que criam e/ou eliminam objetos, devem preservar as fichas-objeto de uma rede.

Referências e identidade Na abordagem denominada Objetos Cooperativos, Sibertin-Blanc permite a utilização tanto de fichas-objetos quanto de fichas-referência. A principal vantagem é que uma marcação de uma classe pode conter várias referências para o mesmo objeto. Lakos observa que isto introduz a possibilidade de manter referências inválidas no modelo. Contudo, é importante observar que este problema só existe realmente se as referências inválidas correspondem a objetos em níveis diferentes de abstração.

O tratamento dado ao conceito de *identidade* também é revelador. Em G-CPN, o conceito de *identificador de contextos* se assemelha ao de identidade. Contudo, dificilmente pode ser entendido como a identidade de um objeto dado que um identificador é criado para cada invocação de um serviço. De fato, o que ocorre é que o conceito de objeto não corresponde ao de contexto. E, portanto, o conceito de módulo G-Net e G-CPN não

⁴Neste caso, a marcação de um lugar não pode ser um multi-conjunto de fichas.

correspondem ao conceito de classe.

Redes OBJSA, por outro lado, não tratam os conceitos de identidade, classe ou instância. Contudo, redes OBJSA são propostas apenas para caracterizar a semântica de modelos na linguagem CLOWN. E como tal, permitem modelar tais conceitos com certa simplicidade. Contudo, é importante não perdermos de vista nosso propósito: queremos avaliar as extensões definidas quanto à qualidade com que os conceitos são incorporados e preservados. Ou seja, embora a forma como as redes OBJSA incorporam o conceito de identidade seja suficiente quanto ao propósito para que foram definidas, não podemos dizer que oferecem suporte efetivo para os conceitos de identidade, de objeto e de classe.

Ligações Um conceito bastante negligenciado nas várias propostas é o de *ligações* entre objetos. Em algumas abordagens, as ligações são determinadas de forma estática e não podem ser alteradas durante o funcionamento do sistema. Em nenhuma abordagem o conceito de ligação é explicitamente definido e, conseqüentemente, não há mecanismos explícitos que permitam seu controle.

Nas abordagens em que as ligações correspondem a fichas-referências o controle é feito mediante a inserção ou remoção das fichas nos lugares da rede. Nenhuma das formas, contudo, é adequada para o tratamento explícito de ligações necessário em modelos de sistemas com topologias dinâmicas de interconexão. Isto certamente dificulta o desenvolvimento de métodos para a análise da evolução da topologia do sistema.

Associação, agregação e especialização O conceito chave na maioria das linguagens orientadas a objetos para a descrição de sistemas é a *classe*. Contudo, é pela combinação de classes através de "operações" adequadamente definidas que podemos construir modelos de sistemas complexos. Os principais mecanismos são a *associação*, a *agregação* e a *especialização*. Embora a maioria das abordagens proponha alguma variação do conceito de classe, nenhuma propõe soluções para os três mecanismos acima.

Devido a sua identificação com as linguagens orientadas a objetos, geralmente alguma forma de especialização é proposta a título de *herança*. Em algumas abordagens são definidas relações de ordem entre as classes a fim de determinar hierarquias válidas de herança—CO-OPN e CLOWN, por exemplo. Isto permite verificar se uma relação de herança entre duas classes preserva ou não o princípio da substitutabilidade. O problema é

que, em geral, esta relação não pode ser garantida apenas por relações estruturais. Logo, trata-se de um método de verificação mais do que de um mecanismo de linguagem. Outras abordagens tratam a herança em sua forma estrutural apenas. Isto é, em detrimento do enfraquecimento da relação de tipo, as classes podem ser relacionadas com o propósito principal de “reutilizar” sua descrição. Além disto, em geral, a relação permite o uso de objetos da subclasse em contextos definidos para a superclasse, embora não permita garantir propriedades comportamentais desse uso.

Apesar de sua importância fundamental na modelagem orientada a objetos, a associação entre classes é geralmente negligenciada na grande maioria das abordagens. A idéia é que associações determinem conjuntos de ligações entre objetos das mesmas classes. O mesmo ocorre com a agregação. Embora possa ser interpretada como uma variação da associação, a agregação é um mecanismo fundamental de abstração para a modelagem de sistemas complexos.

2.3.3 Outras linguagens formais e OO

Integrar conceitos da orientação a objetos a linguagens formais não é propósito exclusivo da comunidade de redes de Petri. Diversos outros grupos têm proposto integrações de outras linguagens formais e OO. Além disso, todos os trabalhos se enquadram nas novas tendências de pesquisa em métodos formais, em que se busca a integração de métodos existentes. O principal fator motivador é o reconhecimento de que diferentes métodos enfocam diferentes aspectos dos sistemas e que combiná-los adequadamente pode ser mais eficaz do que tentar buscar um único método capaz de abordar todos os aspectos relevantes de um sistema [CW96; Fis96].

Dentre os formalismos propostos, é importante destacar CSP-Z [Fis96], Object-Z [DRS95] e CSP-OZ [Fis97]. O primeiro combina CSP [Hoa78] e Z [Spi90]. O segundo integra conceitos da orientação a objetos a Z e, finalmente, o terceiro pode ser visto como a integração dos dois primeiros. O aspecto mais relevante quanto à forma da integração é a ênfase em conservar aspectos relevantes das linguagens originais. A semântica de CSP-Z, por exemplo, é expressa usando o modelo convencionalmente usado com CSP, o modelo de falhas e divergências [BR85]. Uma consequência desta escolha é que a semântica dos modelos em CSP-Z pode ser obtida pela composição da semântica das partes em CSP e

das partes em Z. Além disso, a definição é simplificada, dado que a semântica original de CSP é mantida, sendo necessário definir apenas a semântica da parte expressa em Z. A mesma abordagem é usada para definir a semântica de CSP-OZ em termos da de Object-Z e CSP.

Abordar a integração dessa forma também tem impactos positivos sobre a sintaxe e as mudanças de notação necessárias. No exemplo específico, a semântica escolhida para CSP-Z, a torna um superconjunto de CSP—todas as leis algébricas de CSP são preservadas. Isto permite que a sintaxe e os modelos em CSP-Z sejam intuitivamente semelhantes aos de CSP e aos de Z. Na prática, isto significa que os usuários acostumados com CSP e Z terão que aprender muito pouco para utilizar a nova linguagem. Há ainda outra vantagem: a relação semântica entre a nova linguagem e as originais provê sólidos fundamentos para aproveitar resultados das linguagens originais para a nova. Como consequência imediata, também torna possível reutilizar ferramentas desenvolvidas. Essa idéia tem sido explorada, por exemplo, para a definição de estratégias de análise e para a construção de ferramentas de suporte à análise, como verificadores de modelos (*model-checkers*) e provadores de teoremas. Alguns exemplos de trabalhos em que abordagens deste tipo são usadas com as linguagens mencionadas podem ser encontrados em [FW99; SSC01; MS01].

2.4 Conclusões

A primeira conclusão que derivamos da discussão acima é que não há *uma classe de redes de Petri orientadas a objetos* no mesmo sentido em que há uma classe de *redes de alto nível*. Embora haja diferentes variações de redes de alto nível (redes Predicado/Transição, redes coloridas, redes algébricas, etc), em geral é possível tomar qualquer uma delas como representativa da classe. Isto ocorre porque todas são definidas a partir de um mesmo princípio: equipar as fichas com informação de alto nível. Contudo, não podemos dizer que o mesmo ocorra com as chamadas *redes orientadas a objetos*.

A idéia subjacente de todas as propostas é uma só: integrar conceitos da orientação a objetos às redes de Petri. Contudo, o princípio usado difere de uma para outra. Em várias, mas não em todas, a abordagem consiste em promover as fichas da rede à condição de objeto. Destas, algumas modelam os objetos através de redes e outras através de outros

formalismos. Mesmo entre as que modelam os objetos através de redes há diferenças relevantes. Em algumas, a ênfase é dada à modelagem em múltiplos níveis de abstração. Logo, as fichas-objetos são vistas como representações abstratas de objetos cujo comportamento detalhado é dado por sua rede. Em outros casos, a ênfase é dada à decomposição do modelo em várias partes, dentro de um mesmo nível de abstração. Neste caso, fichas-objetos representam objetos que coexistem no sistema, mas no mesmo nível de abstração. Portanto, concluímos que não há *uma* classe de redes orientadas a objetos que permita abordar o desenvolvimento de métodos de análise de uma perspectiva genérica.

Uma segunda conclusão é que há um certo desequilíbrio entre os conceitos de redes de Petri e os da orientação a objetos nas abordagens propostas. Isto fica evidente pelas definições formais. Em geral, o conceito de classe é formalizado como um módulo de rede. O conceito de objeto, por outro lado, é definido de forma obscura. Isto ocorre porque o comportamento do modelo obtido é dado em termos da composição das redes-classe em uma única rede global. A definição de estado do sistema é identificada com a noção de marcação da rede global. Em compensação, perde-se a noção de objeto em sua forma independente. Mesmo no caso das redes OPN, em que a ênfase é a modelagem em múltiplos níveis de abstração, os objetos correspondem às fichas na marcação da rede de mais alto nível. Este desequilíbrio impede a avaliação dos modelos resultantes sob a perspectiva da orientação a objetos.

A terceira e mais importante conclusão é que as possibilidades de análise formal do modelo são reduzidas à análise da rede global obtida pela composição das redes-classe. Se a modelagem do sistema segue uma abordagem orientada a objetos, é natural que as questões de análise (ou *queries*) sejam expressas em termos dos objetos e das configurações por eles alcançadas no decorrer do funcionamento do sistema. Contudo, devido ao desequilíbrio acima mencionado, não há paralelo direto entre proposições dessa natureza e os conceitos semânticos utilizados para caracterizar o comportamento dos modelos. Logo, é improvável que métodos de análise direta possam ser desenvolvidos. As expressões têm que ser interpretadas em relação aos conceitos semânticos baseados em marcações de redes de Petri. Logo, sempre será necessário "traduzir" as questões e "interpretar" os resultados obtidos. Isto é decorrência de uma abordagem estrutural ou sintática de integração em que o método é sempre norteado pelo objetivo de construir uma única rede a partir das redes-classe.

Além das dificuldades para desenvolver métodos de análise apontadas acima, a forma de relacionamento entre as redes-classes também dificulta a análise modular. Na maioria dos casos, as redes podem ser combinadas através de mecanismos que equivalem à fusão de lugares entre as redes. Isto significa que a análise completa de uma rede isoladamente deve considerar todas as possíveis formas de utilização da rede por outras redes, porque o espaço de estados não é previsível considerando apenas a semântica de redes de Petri. Isto inibe o desenvolvimento de classes efetivamente reutilizáveis, dado que suas propriedades comportamentais são afetadas pela forma em que são utilizadas. Relações mais estritas entre o comportamento descrito pela rede isoladamente e o obtido em qualquer forma de utilização são desejáveis para garantir as possibilidades de desenvolvimento de métodos composicionais de análise.

Considerações finais Neste capítulo fizemos um levantamento geral sobre o atual estado da arte em relação às abordagens usadas para integrar conceitos de orientação a objetos às redes de Petri. Dissemos e apresentamos razões pelas quais consideramos que as abordagens propostas são baseadas em formas *estruturais* ou *sintáticas* de integração, o que minimiza as condições de adaptação dos métodos existentes. Evidencia disso são os poucos resultados existentes, até o presente momento, de técnicas específicas de análise para redes de Petri orientadas a objetos—isto também é devido ao fato de que não há uma classe genérica de redes de Petri orientadas a objetos.

Finalmente, concluímos que somente através de uma abordagem semântica de combinar os conceitos é possível encontrar uma forma genérica e efetiva de integração que maximize as condições de adaptação dos métodos existentes.

O ponto de vista adotado no decorrer da exposição e na discussão final é consistente com o problema apresentado na introdução deste documento (ver página 8). No capítulo seguinte, abordamos o problema de especificar uma forma adequada de integração semântica em que as condições de adaptação dos métodos convencionais de redes de Petri e da orientação a objetos sejam maximizadas.

Falhar na solução dada aos fundamentos teóricos da combinação (de métodos formais) é desperdiçar as verdadeiras vantagens do formalismo. Em química, uma distinção é feita entre uma mistura e um composto. Em uma mistura, os ingredientes apenas se juntam. Em um composto, os ingredientes se unem

quimicamente. O mesmo deve ocorrer com a combinação de diferentes métodos formais. Se o significado de uma combinação não for adequadamente explicado, então o resultado é meramente uma mistura: nada mais pode ser deduzido da descrição casada do que delas separadamente. Se o significado da combinação é explicado, então o resultado é bem mais poderoso. Torna-se possível ter duas visões da especificação de um sistema além de permitir operar e refinar uma das visões e compreender as conseqüências na outra. [CW96, p. 14]⁵

⁵Tradução livre.

Capítulo 3

Modelo Conceitual

No capítulo anterior identificamos diversas razões pelas quais os métodos convencionalmente utilizados com redes de Petri e orientação a objetos não podem ser facilmente adaptados para os formalismos propostos. Neste capítulo, apresentamos a especificação de uma proposta de abordagem em que tais deficiências são compensadas.

3.1 Características de uma “boa” integração

Metas A principal justificativa para integrar dois paradigmas de modelagem é que um complementa o outro em relação aos aspectos em que são insuficientes/deficientes. Embora esteja implícito, isto revela que são as características positivas que nos atraem em cada um dos paradigmas e que motivam a integração. Há, portanto, duas metas que devem ser consideradas ao definir uma abordagem. Primeiro, é necessário que as deficiências sejam efetivamente eliminadas. E, segundo, é necessário preservar as características positivas originais.

Aparentemente, a observação acima é óbvia. Contudo, podemos dizer que é em relação a essas metas que a maioria das abordagens propostas falha. A principal razão para isto é que há certa tendência de focar apenas as deficiências a serem compensadas e colocar em segundo plano as boas características originais dos paradigmas. O resultado é que o formalismo resultante elimina deficiências em detrimento da perda de características importantes. Uma evidência disto é a dificuldade de adaptar os métodos de análise convencionais de redes de Petri em decorrência das modificações introduzidas.

Ao declará-las explicitamente, evidenciamos que há estratégias de integração em que

as duas metas se podem contrapor. Isto é particularmente verdadeiro para abordagens de integração orientadas estruturalmente. Não é difícil perceber que as boas e as más características podem ser propriedades semânticas dos elementos de descrição. Em muitos casos, essas características são, do ponto de vista estrutural, virtualmente inseparáveis umas das outras. Logo, é inevitável que em relação a determinados aspectos, as duas metas sejam inconsistentes.

Integração ortogonal A estratégia que propomos consiste em enfatizar que cada um dos paradigmas modela aspectos complementares ao outro. Desta forma, as características podem ser consideradas de diferentes perspectivas, evitando possíveis inconsistências. A Figura 3.1 ilustra a idéia. O formalismo proposto deve permitir modelar sistemas em que duas visões *ortogonais*, referentes a cada um dos paradigmas, se complementam. Desta forma, alguns aspectos do modelo são abordados da visão de redes de Petri e outros da visão da orientação a objetos. Por serem “ortogonais”, em cada visão do modelo são projetados apenas os aspectos referentes a um dos paradigmas. A integração consiste em articular adequadamente os elementos de cada paradigma para descrever um modelo único.

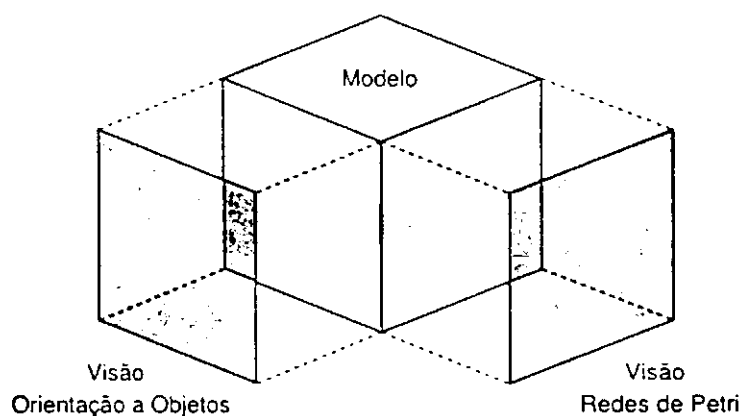


Figura 3.1: Visões ortogonais dos aspectos de um modelo

Características de uma boa integração Com base nos critérios e conclusões apresentados na discussão do capítulo anterior e na estratégia acima descrita, identificamos algumas características fundamentais para obter uma *boa* integração.

Genérica É importante que a abordagem seja definida a partir de formas genéricas e representativas de redes de Petri e de orientação a objetos. A idéia é isolar os conceitos

chaves característicos de cada um dos paradigmas e garantir que a integração seja caracterizada exclusivamente a partir deles. Desta forma, minimizamos a influência de extensões, facilidades sintáticas e pequenas variações semânticas sobre o resultado. Conseqüentemente, garantimos que as propriedades observadas no formalismo resultante são derivadas exclusivamente dos conceitos chaves originais.

Para obter uma integração genérica, é necessário identificar e expressar, de forma não ambígua, todos os *conceitos chaves característicos* de redes de Petri e da orientação a objetos. Por serem definidas formalmente, isto é bastante simples de obter para redes de Petri. Contudo, o mesmo não ocorre com os conceitos da orientação a objetos. Este fator tende a desequilibrar a integração. É devido a isto que diversas propostas abordam a integração incorporando os conceitos da orientação a objetos a uma formalização inicialmente centrada em redes de Petri. O resultado é que os modelos dificilmente podem ser interpretados de uma perspectiva da orientação a objetos porque os conceitos semânticos são essencialmente os de redes de Petri. Por esta razão, dedicamos especial atenção à formalização desses conceitos no Capítulo 4.

Aderência Dizemos que a integração é *aderente* aos paradigmas originais de redes de Petri e da orientação a objetos se preserva todos os conceitos chaves característicos. Há dois aspectos da aderência que devem ser considerados: a aderência estrutural e a aderência semântica. O primeiro se refere à preservação dos elementos estruturais e construções sintáticas originais. O segundo, se refere a preservar o significado de cada um dos conceitos. Em uma integração aderente, os conceitos que a compõem podem ser usados para modelar os mesmos aspectos, para os quais são usados nos paradigmas originais. Conseqüentemente, a aderência facilita a compreensão e a aceitação da integração.

Naturalmente, é necessário introduzir regras para regular e articular o uso dos conceitos no formalismo proposto. Contudo, é desejável que sejam minimamente restritivas, de modo que, dentro de limites específicos definidos nas visões, a modelagem seja regulada exclusivamente pelas regras originais.

Semântica conservativa A semântica da notação proposta deve ser expressa como uma composição da semântica original dos dois paradigmas. Isto é, deve ser possível obter o significado de um modelo, combinando o significado dos aspectos modelados por

conceitos de redes de Petri ao significado dos aspectos modelados por conceitos da orientação a objetos.

Logo, para obter uma semântica conservativa, não é conveniente expressá-la em termos de alterações estruturais e mudanças gradativas da regra de comportamento original de redes de Petri. Uma forma mais adequada de abordar o problema é atribuir a cada visão do modelo um significado independente e combinar esses significados através de uma regra explícita de composição semântica. É importante observar que é desejável definir essa regra de forma que o significado de cada visão seja *consistente* em relação ao significado de todo o modelo.

Estas características proporcionam condições mais adequadas para obter uma composição *ortogonal* dos conceitos originais de redes de Petri e da orientação a objetos. Para obter uma abordagem genérica, somos forçados a isolar os conceitos realmente relevantes de cada paradigma e, conseqüentemente, de cada visão. Isto simplifica a avaliação da abordagem e provê melhores condições para a adaptação e o desenvolvimento de métodos genéricos de análise. Além disso, facilita a avaliação da introdução de extensões, facilidades sintáticas e pequenas variações semânticas.

A aderência nos garante que a integração preserva a forma como cada um dos paradigmas é utilizado originalmente, evitando distorções. Logo, em cada visão um dos paradigmas é utilizado da forma mais próxima possível da original.

— Finalmente, a terceira característica nos obriga a definir uma relação semântica entre as visões ortogonais e o modelo propriamente dito. Desta forma, estabelece fundamentos mais adequados para a adaptação de métodos existentes. Se uma visão do modelo pode ser efetivamente interpretada usando a semântica original de redes de Petri, então certamente podemos aplicar os métodos convencionais de análise à visão. E se o significado da visão é consistente em relação ao significado do modelo completo, então os resultados da análise também podem ser considerados *consistentes*¹. Argumento semelhante vale para a adaptação de métodos e técnicas da orientação a objetos.

Observe ainda que se a integração é ao mesmo tempo genérica, aderente, e propõe uma semântica conservativa, então todas as propriedades do formalismo resultante são derivadas diretamente dos conceitos característicos dos paradigmas originais ou da regra

¹A noção exata de consistência é determinada pela regra de composição dos significados.

de composição. Em particular, toda a diferença semântica em relação aos paradigmas originais deve ser introduzida pela regra de composição semântica.

3.2 Projeto da linguagem

Uma forma de expressar os conceitos que caracterizam a forma de integração aqui proposta é classificando-os. A Figura 3.2 ilustra os quatro níveis (ou camadas) em que separamos os conceitos. Como é convencional neste tipo de representação, os níveis inferiores servem de fundamento para os níveis mais acima. Vejamos em que consiste cada camada.

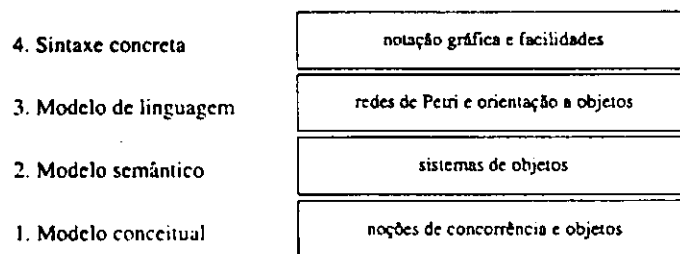


Figura 3.2: Distribuição dos conceitos da proposta em camadas

1. *Modelo conceitual.* Consiste em um conjunto de conceitos e noções fundamentais que caracterizam a leitura que fazemos dos sistemas de interesse, isto é, de sistemas distribuídos e concorrentes de software. São as *metáforas* propostas no modelo conceitual que determinam o papel de cada conceito da orientação a objetos e das redes de Petri realizados pelas demais camadas.
2. *Modelo semântico.* Consiste na escolha de uma representação formal adequada para os aspectos semânticos caracterizados no modelo conceitual. A idéia consiste em determinar uma representação formal para os sistemas de interesse, em que enfatizamos as noções semânticas. Como já dissemos, o modelo semântico deve ser consistente com a caracterização da semântica original de redes de Petri e da orientação a objetos.
3. *Modelo de linguagem.* Define as construções utilizadas para descrever os sistemas de objetos caracterizados pelo modelo semântico. É nesta camada que definimos o uso propriamente dito das redes de Petri e das principais construções sintáticas da orientação a objetos. O significado das descrições será expresso em termos de elementos do modelo semântico.

4. *Sintaxe concreta.* Apenas na última camada definimos precisamente as construções sintáticas que suportam efetivamente a elaboração de modelos. Na prática, caracterizar esta camada consiste em determinar a sintaxe concreta das linguagens utilizadas para inscrever os modelos bem como as convenções gráficas utilizadas.

A seção seguinte apresenta em detalhes o modelo conceitual adotado neste trabalho. O modelo semântico é apresentado no Capítulo 4. Finalmente, o modelo de linguagem e a sintaxe concreta são apresentados no Capítulo 5.

3.3 O modelo conceitual

O modelo conceitual de computação adotado neste trabalho é caracterizado a partir das noções semânticas da orientação a objetos em que enfatizamos aspectos de distribuição e comportamento concorrente.

Todos os conceitos são definidos a partir da noção de *objeto*. A idéia é permitir a representação de sistemas distribuídos e concorrentes através de coleções de objetos.

3.3.1 Objetos

Um aspecto característico de sistemas distribuídos de software é a *localidade*. Isto é, o sistema é caracterizado por pontos de concentração da computação espacialmente distribuídos—cada ponto de concentração tem posição única dentro do espaço. No modelo apresentado, representamos os pontos de concentração através do conceito de *objeto*. Definimos um objeto como uma unidade computacional autônoma, com ciclo de vida definido, que encapsula estado, comportamento e atividade próprios.

O *ciclo de vida* de um objeto é determinado por dois momentos: a criação e a destruição. Entre esses dois momentos o objeto é dito *vivo* e modela um ponto de processamento no sistema representado. Portanto, admite-se que a coleção de objetos que caracteriza um sistema de objetos em um dado instante é definida dinamicamente pelo próprio funcionamento do sistema.

Para completar a metáfora introduzida para representar componentes de sistemas distribuídos, cada objeto é considerado *autônomo*. Ou seja, ao contrário da forma convencional da orientação a objetos em que os objetos são entidades reativas, neste modelo conside-

ramos os objetos como entidades *proativas*. Por esta razão dizemos que, além de estado e comportamento, cada objeto encapsula também sua própria *atividade*. As noções de estado e comportamento são semelhantes às convencionais. O estado é caracterizado por um conjunto de atributos e valores instantâneos a eles associados. E o comportamento consiste nas ações que o objeto é capaz de executar e que alteram seu estado.

Localidade e concorrência Observe que o encapsulamento enfatiza que o objeto é a unidade básica de processamento. A decomposição em objetos de um sistema pode ser usada para expressar suas características de localidade e distribuição. Cada objeto é, potencialmente, um ponto de processamento do sistema. Também é possível escolher diferentes níveis de abstração, o que permite relações diferentes entre os objetos de um modelo e as entidades reais representadas. Portanto, é possível representar um único objeto de determinado modelo através de todo um sistema de objetos em um nível diferente de abstração. Dentro de um único nível de abstração, contudo, vale a relação:

objeto \equiv (estado + comportamento + atividade) \equiv ponto de processamento do sistema.

Os objetos são também a única fonte de atividade no modelo conceitual. Ou seja, todos os eventos de um sistema correspondem a ações executadas pelos objetos que o compõem. Portanto, a decomposição em objetos proativos é a fonte primária de atividade concorrente do modelo conceitual. A segunda fonte é o comportamento interno dos objetos. Ou seja, a atividade encapsulada por um objeto é potencialmente concorrente. Portanto, cada objeto pode executar suas ações-internas concorrentemente. Alguns autores utilizam os termos *inter-concorrência* e *intra-concorrência* para distinguir entre estas duas formas de concorrência.

Observe que esta característica do comportamento interno dos objetos reforça a possibilidade de utilizá-los como abstrações de outros sistemas de objetos. Além disso, permite adotar as ações concorrentes como a forma básica de comportamento dos objetos. Desta forma, qualquer linearização das ações é vista como uma restrição a ser imposta sobre o comportamento dos objetos.

A Figura 3.3 ilustra um sistema de objetos composto por quatro objetos. Os objetos são representados por círculos rotulados com *a*, *b*, *c* e *d*. Dentro de cada objeto representamos o estado e a atividade encapsulados. O estado é representado pelas estruturas em forma

de retângulos e a atividade pelas setas—podemos pensar em cada seta como uma linha de execução paralela. Observe que os objetos *b*, *c* e *d* são *intra-concorrentes*.

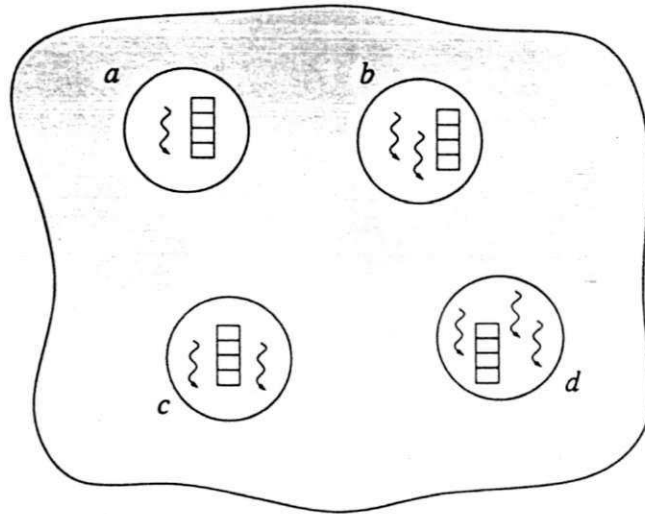


Figura 3.3: Ilustração de um sistema de objetos proativos

3.3.2 Sistemas de objetos

A representação de um sistema real como um sistema de objetos requer que uma coleção de objetos seja considerada. Além disso, é necessário que os objetos sejam capazes de interagir de alguma forma para que possam coordenar adequadamente suas ações. Naturalmente, a interação entre os objetos não se dá arbitrariamente. Cada sistema determina regras diferentes para regular quais e como os objetos podem interagir entre si a fim de servir ao propósito do sistema. No modelo conceitual, contudo, abordamos a questão através de um esquema genérico.

A idéia chave é a *nomeação*: assumimos que cada objeto é identificado unicamente em um sistema. Isto permite indicar os objetos envolvidos em uma interação através de seus identificadores. Além disso, os identificadores são considerados dados e podem ser armazenados pelos objetos.

A Figura 3.4(a) ilustra o mesmo sistema da figura anterior, com a indicação explícita dos identificadores armazenados em cada objeto. O objeto *a*, por exemplo, armazena os identificadores dos objetos *b* e *c*—dizemos que “*a* conhece *b* e *c*” ou que “*a* está ligado aos objetos *b* e *c*”. Isto significa que ações de *a* podem determinar interações com os objetos *b* e *c*. Uma forma alternativa de representar os identificadores armazenados é através de

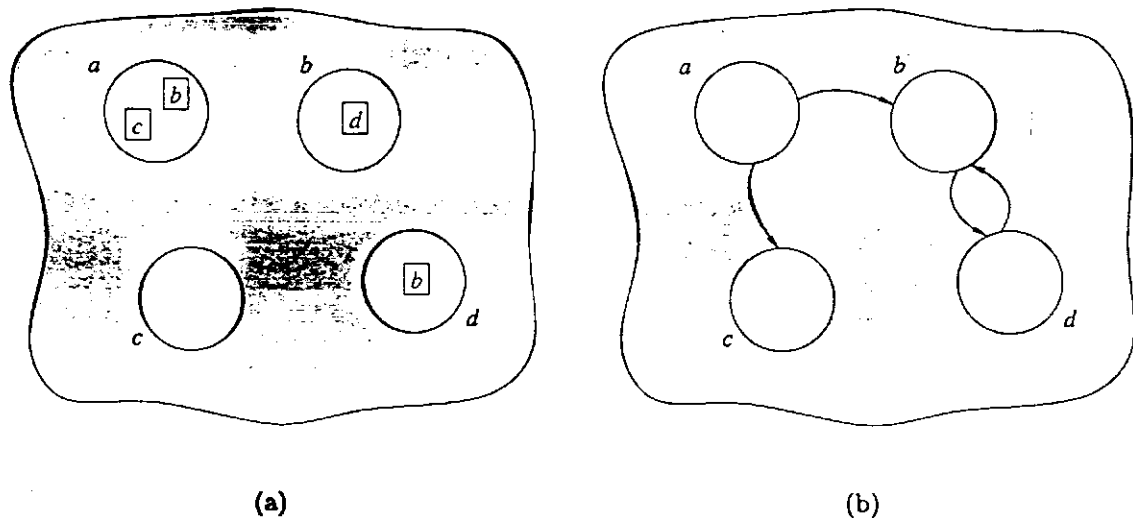


Figura 3.4: Identificadores de objetos e grafo de interconexão

arcos orientados, ligando o objeto que armazena o identificador ao objeto conhecido. A Figura 3.4(b) ilustra a idéia para o mesmo sistema. Os arcos são denominados de ligações entre os objetos e o grafo é dito o *grafo de interconexão*² do sistema. A vantagem desta representação é tornar visualmente evidente a *relação de conhecimento* entre os objetos.

O grafo de interconexão de um sistema não é estático. Trata-se de uma representação instantânea do sistema. As ações dos objetos podem criar e destruir objetos, bem como alterar a relação de conhecimento.

3.3.3 Ações e interações

As ações específicas que um objeto pode executar são dependentes da função que o objeto exerce no sistema. Contudo, para efeitos de representação, podemos classificar as ações quanto aos seus efeitos.

Em primeiro lugar, há as *ações internas*. Uma ação é dita *interna* se tem efeito restrito sobre o estado interno do objeto. Ações internas são utilizadas para descrever processamentos puramente locais nos objetos. Observe que o grafo de interconexão de um sistema não é alterado por ações internas.

As ações mais relevantes do ponto de vista dos sistemas de objetos, contudo, são as que determinam interações. Pela definição dos dicionários, uma interação é uma ação mútua

²Agha e outros autores usam a expressão *topologia de interconexão* com o mesmo significado (ver [Agh86]).

ou recíproca de dois ou mais objetos. Contudo, para nossos propósitos, é mais conveniente caracterizar interações em termos de comunicação entre os objetos. Colocado desta forma, é fácil identificar dois tipos complementares de ações de interação. A primeira, denominada de *entrada de dados*, consiste na leitura de um dado externo ao estado do objeto que a executa. Ao executar uma entrada de dados, o objeto pode incorporar o dado externo a seu estado interno. A ação complementar é denominada *saída de dados*. Uma saída de dados é uma ação através da qual um objeto disponibiliza um dado armazenado para outro objeto.

Observe que há duas ações envolvidas em uma interação completa. De um lado, um objeto disponibiliza um elemento de dado, executando de uma saída de dados. Do outro, um objeto deve recebê-lo, executando uma entrada de dados. Há duas formas diferentes destas ações serem ordenadas no tempo. A primeira possibilidade é a mais intuitiva: primeiro ocorre a saída de dados e, em seguida, a entrada de dados. Neste caso, dizemos que a comunicação é *assíncrona*. Contudo, há situações em que é conveniente assumir que as duas ações ocorrem simultaneamente. Neste caso, dizemos que a comunicação é *síncrona*.

Neste ponto, é conveniente comparar as primitivas de interação adotadas no modelo conceitual introduzido à forma mais convencional de interação da orientação a objetos: a *invocação de métodos*. Quando um objeto ativo do sistema invoca um método de outro objeto, a linha de execução original é interrompida até que a execução do método invocado seja concluída. A invocação pode ser parametrizada o que equivale a passar dados do objeto invocador para o objeto invocado. Também é possível que o método invocado retorne dados para o objeto invocador. Observe, contudo, que este padrão de comportamento não é conveniente quando se trata de objetos concorrentes. Isto obrigaria a interromper a atividade no objeto invocador. Daí porque adotamos primitivas de interação mais genéricas e mais simples. Por outro lado, observe que combinar as primitivas de entrada e saída de dados para caracterizar o padrão convencional de comunicação é extremamente simples.

Devido à comunicação assíncrona, surge o conceito de *mensagem*. Uma mensagem é um elemento de dado disponibilizado por um objeto para outro. Toda mensagem é gerada por uma ação de saída assíncrona do objeto emissor e eliminada por uma ação de entrada assíncrona no objeto receptor—dizemos, respectivamente, que os objetos “enviam” e “recebem” a mensagem. Toda mensagem tem destinatário especificado pelo objeto emissor

e somente ele pode recebê-la.

Criação e destruição de objetos A criação e destruição de objetos no sistema é determinada por ações específicas que têm esses efeitos. A criação segue os moldes tradicionais da orientação a objetos: objetos são livres para criar novos objetos arbitrariamente. A destruição, contudo, é restrita à *auto-destruição*. Novamente, esta restrição é conveniente devido ao comportamento concorrente dos sistemas modelados. A auto-destruição, contudo, não impede a modelagem de sistemas em que um objeto possa eliminar outros. Para isto, basta condicionar a auto-destruição à recepção de estímulos externos de controle.

Criação e remoção de ligações Implicitamente, os objetos também podem executar ações de remoção e de criação de ligações. Recorde que ligações consistem em identificadores armazenados pelos objetos. Logo, a criação de novas ligações é condicionada à recepção de identificadores de objetos através de ações de entradas de dados. Ou seja, se um identificador é recebido, a ligação é "automaticamente" estabelecida. Analogamente, a remoção de ligações consiste em eliminar identificadores do estado interno do objeto.

É importante observar que devido à natureza concorrente dos sistemas, nada impede que *ligações inválidas* sejam mantidas pelos objetos. Isto ocorre quando o objeto para o qual há alguma ligação é destruído.

3.4 Conclusões

Neste capítulo definimos as boas características de uma integração entre redes de Petri e orientação a objetos. A partir dessas características definimos uma estratégia de abordagem que consiste em integrar os dois paradigmas enfocando a composição dos seus significados. Também identificamos quatro camadas de conceitos que devem ser caracterizadas para definir adequadamente a integração.

Finalmente, descrevemos a camada fundamental que determina o chamado *modelo conceitual*. O modelo conceitual foi apresentado como decorrência direta da classe de sistemas que pretendemos abordar através da abordagem de integração de redes de Petri e orientação a objetos introduzida neste documento: sistemas distribuídos e concorrentes de software.

Os conceitos identificados permitem identificar com maior clareza quais aspectos são

melhor abordados através de redes de Petri ou das construções da orientação a objetos. A relação é bastante evidente, devido à escolha da concorrência como a forma fundamental de atividade. Por outro lado, a decomposição e as metáforas adotadas remetem diretamente aos conceitos da orientação a objetos.

É importante observar que o modelo conceitual aqui apresentado é inspirado e semelhante aos adotados por sistemas baseados em atores (*Actor systems* [Agh86]) e por π -cálculo [Mil93]. A principal diferença é que incluímos os dois mecanismos primitivos de interação: síncrona e assíncrona. Outra diferença é que os objetos podem encapsular comportamento concorrente diretamente. Estas escolhas são convenientes do nosso ponto de vista, porque não temos o propósito de identificar um modelo fundamental de computação. Nosso propósito é definir mecanismos que sejam efetivamente úteis para a representação de sistemas reais. Além disso, nossos experimentos anteriores com G-CPN e versões preliminares de RPOO têm nos demonstrado que, embora seja possível modelar ou codificar uma primitiva através da outra, definir toda a linguagem a partir de uma única primitiva não simplifica sua descrição.

Capítulo 4

Sistemas de Objetos

Neste capítulo, nos voltamos para os conceitos semânticos necessários para a integração. Do ponto de vista de redes de Petri, os conceitos utilizados são os convencionais. Por outro lado, como concluímos nos capítulos anteriores, também precisamos identificar os conceitos semânticos necessários para caracterizar a orientação a objetos concorrente de forma independente de redes de Petri. Somente assim podemos definir a integração de forma efetivamente equilibrada e ortogonal.

Portanto, o foco deste capítulo é a caracterização de sistemas de objetos concorrentes. Uma caracterização formal de sistemas de objetos é proposta como modelo semântico para sistemas dinâmicos discretos. São sistemas que evoluem ao longo do tempo pela ocorrência de eventos, possivelmente paralelos. Denominamos de *configuração*, a observação de um sistema de objetos em um instante particular de tempo do seu funcionamento. Configuração é, portanto, o termo usado para o conceito de estado em sistemas de objetos.

É importante lembrar que devemos separar os aspectos internos e os externos dos objetos. Podemos modelar os aspectos internos dos objetos através de redes de Petri e, de fato, fazemos isso no capítulo seguinte. Logo, nosso foco está em caracterizar os aspectos externos aos objetos. Isto nos motiva a dividir o conceito de configuração em duas partes, às quais denominamos *estrutura e dinâmica*.

A estrutura é o conjunto dos elementos que formam a configuração: são os objetos, suas ligações e eventuais mensagens pendentes. A *dinâmica* é, como se pode inferir do termo, a representação das potencialidades dinâmicas da configuração. Logo, é a abstração do estado interno dos objetos da configuração. Ao invés de escolhermos uma representação

explícita para o estado dos objetos, o que nos obrigaria a antecipar sua estrutura interna, a *dinâmica* consiste apenas na enumeração das ações que os objetos estão “aptos” a executar. É a partir da *dinâmica* de uma configuração, que determinamos os eventos que podem ocorrer.

Na Seção 4.1, nosso foco é o conceito de estrutura de uma configuração. Formalizamos esse conceito e apresentamos uma notação e linguagem que nos permitem expressar estruturas de configurações. No início da Seção 4.2, enfocamos o conceito de *dinâmica* e, em seguida, completamos a formalização de configurações.

4.1 Estruturas

4.1.1 Definições elementares

Nas definições a seguir, \mathcal{O} denota um conjunto de objetos e D um domínio de dados, ambos potencialmente infinitos.

Objetos e ligações Uma estrutura (de configuração) consiste em um conjunto finito de objetos, ditos *vivos*, e em um conjunto finito de ligações entre os objetos. Objetos vivos são elementos de \mathcal{O} e ligações são pares não-ordenados de objetos. Em geral, denotamos os conjuntos de objetos e ligações de uma estrutura por O e L , respectivamente. Formalmente, podemos defini-los da seguinte forma:

$$O \subseteq \mathcal{O} \quad (4.1)$$

$$L \subseteq [\mathcal{O}]^2 \quad (4.2)$$

Objetos pertencentes a $\mathcal{O} \setminus O$ são ditos *não-vivos* ou *inexistentes* na estrutura. Para simplificar a notação, denotamos cada ligação $\{a, b\}$ por \overline{ab} (ou, equivalentemente, por \overline{ba}).

Mensagens Além de objetos e ligações, estruturas podem ter *mensagens*. Cada mensagem consiste em um elemento de dado $m \in D$ e dois objetos. Logo, representamos mensagens por triplas da forma:

$$\langle e, m, d \rangle \in \mathcal{O} \times D \times \mathcal{O}.$$

Os elementos e , m e d são denominados, respectivamente, de *emissor*, *conteúdo* e *destinatário* da mensagem. Para simplificar a notação, denotamos mensagens $\langle e, m, d \rangle$ por m_d^e . Em geral, usamos m_i para representar mensagens quaisquer. Neste caso, usamos $[m_i]_e$, $[m_i]_d$ e $[m_i]_m$ para indicar o emissor, o destinatário e o conteúdo de m_i .

Multi-conjuntos Para definir a relação entre mensagens e estruturas usaremos o conceito de multi-conjuntos. Intuitivamente, um multi-conjunto é um conjunto que admite múltiplas cópias de cada elemento. Formalmente, um multi-conjunto é definido como uma função que mapeia o conjunto de elementos a valores inteiros. Seja A o conjunto dos elementos. Um multi-conjunto é uma função $m : A \rightarrow \mathbb{N}$. Dizemos que $m(x)$ é o número de ocorrências do elemento x em m . Às vezes é conveniente denotar multi-conjuntos através de somas formais, do tipo

$$m = \sum_{x \in A} m(x)'x.$$

Neste caso, as parcelas da soma cujo $m(x) = 0$ são opcionais. Por exemplo, seja $A = \{a, b, c\}$. O multi-conjunto contendo duas ocorrências de a e uma ocorrência de c é denotado por $2'a + 1'c$. O multi-conjunto em que $m(x) = 0$ para todo x é dito o multi-conjunto vazio ou nulo e é denotado por 0 . Também é interessante estender a notação convencional de conjuntos para multi-conjuntos. Assim, escrevemos $x \in m$ para indicar que $m(x) > 0$.

Introduzida a notação correspondente, podemos a relação de mensagens com estruturas. A cada estrutura corresponde um *multi-conjunto* finito de mensagens, a que chamamos de *conjunto de mensagens pendentes*, e que denotamos por M . Podemos definir M por:

$$M \subseteq [O \times D \times O]^{ms} \quad (4.3)$$

O uso de multi-conjuntos para definir M é necessário para permitir que mensagens idênticas coexistam em uma mesma estrutura, como preceitua o modelo conceitual. Formalizados os conceitos de conjunto de objetos vivos, ligações e mensagens pendentes podemos reuni-los para definir *estruturas de configurações*.

Definição 4.1 *Sejam O , L e M , respectivamente, conjuntos de objetos vivos, ligações e mensagens pendentes. A tripla $\langle O, L, M \rangle$ é dita uma estrutura, e elementos de O , L e de M são chamados genericamente de elementos da estrutura.*

4.1.2 Notação e linguagem

O objetivo desta seção é introduzir a notação que usamos para expressar e manipular estruturas. Definimos uma representação gráfica que facilitará a compreensão dos elementos de uma configuração—é muito mais fácil, ou intuitivo, captar a estrutura de uma configuração pela sua representação gráfica do que pela sua definição formal. Também introduzimos convenções notacionais e operadores sobre estruturas que nos permitirão expressar e relacionar estruturas sem recorrer à definição formal. O propósito é simplificar a linguagem, evitando, sempre que possível, os detalhes da formalização baseada em tuplas. É importante lembrar que estruturas são apenas um passo em direção à caracterização de sistemas de objetos. A linguagem aqui construída facilitará a definição e o estudo desses sistemas.

Partes de uma estrutura Às vezes é necessário fazer referência aos conjuntos que formam uma estrutura—de objetos, de ligações e mensagens. Como nem sempre dispomos dos nomes desses conjuntos usamos a seguinte notação para expressá-los. Para qualquer estrutura $E = \langle O, L, M \rangle$, escrevemos:

$$\begin{aligned} [E]_O &\stackrel{\text{def}}{=} O && \text{para indicar "os objetos (vivos) em } E,\text{"} \\ [E]_L &\stackrel{\text{def}}{=} L && \text{para indicar "as ligações em } E,\text{"} \\ [E]_M &\stackrel{\text{def}}{=} M && \text{para indicar "as mensagens (pendentes) em } E.\text{"} \end{aligned}$$

Estruturas iguais Uma noção básica e quase óbvia é a de igualdade de estruturas. Definimos que duas estruturas E e E' são iguais se e somente se têm conjuntos iguais de objetos, ligações e de mensagens pendentes. Formalmente:

$$E = E' \stackrel{\text{def}}{\iff} \begin{aligned} [E]_O &= [E']_O && \text{e} \\ [E]_L &= [E']_L && \text{e} \\ [E]_M &= [E']_M \end{aligned}$$

Representação gráfica Podemos representar estruturas de configurações graficamente sem perder informação ou introduzir ambigüidades. Na Figura 4.1 vemos um exemplo: a estrutura E e sua respectiva representação gráfica. Representamos objetos vivos por círculos com bordas cheias—veja objetos a , b e c —e objetos não-vivos, por círculos com bordas tracejadas—objeto d . Arcos entre os objetos representam ligações—veja a ligação \overline{ac} . E,

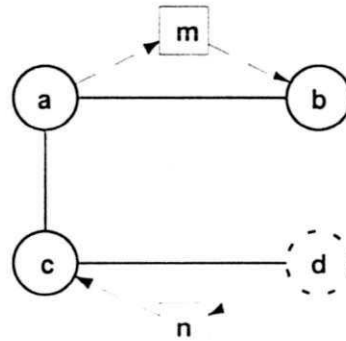
para representar mensagens, usamos retângulos inscritos com o conteúdo da mensagem, e com respectivos objetos emissor e destinatário indicados por setas tracejadas. A representação gráfica torna fácil identificar, por exemplo, que a mensagem m_b^a , tem os objetos a e b como emissor e destinatário, respectivamente

$$E \triangleq \langle O, L, M \rangle$$

$$O = \{a, b, c\}$$

$$L = \{\overline{ab}, \overline{ac}, \overline{cd}\}$$

$$M = \{m_b^a, n_c^d\}$$



(a) Descrição formal

(b) Representação gráfica

Figura 4.1: Representação gráfica de uma estrutura

Estrutura nula Há várias estruturas especiais que merecem destaque e que serão extremamente úteis mais à frente em nosso estudo. Por isto, introduzimos uma notação especial e simplificada para representá-las. A primeira delas é a chamada estrutura *nula* ou *vazia*. Trata-se da estrutura em que não há nenhum objeto, nenhuma ligação e nenhuma mensagem pendente. Naturalmente, consiste em conjuntos vazios de objetos, ligações e mensagens, e será denotada por 0:

$$0 \stackrel{\text{def}}{=} \langle \emptyset, \emptyset, 0 \rangle \quad (4.4)$$

Estruturas elementares Há outras estruturas para as quais é conveniente definirmos notação especial. Estruturas em que há apenas um único objeto ou uma única ligação ou uma única mensagem são denominadas *elementares*. Para elas, usamos a seguinte notação. Sejam $a, b \in \mathcal{O}$ objetos quaisquer e $m \in \mathcal{D}$ um elemento de dado qualquer. Então

escrevemos:

$$\begin{aligned} a &\stackrel{\text{def}}{=} \langle \{a\}, \emptyset, 0 \rangle \\ \overline{ab} &\stackrel{\text{def}}{=} \langle \emptyset, \{\overline{ab}\}, 0 \rangle \\ m_b^a &\stackrel{\text{def}}{=} \langle \emptyset, \emptyset, 1'm_b^a \rangle. \end{aligned}$$

Ou seja, para denotar a estrutura que consiste apenas do objeto a , também escrevemos a . De forma análoga, escrevemos \overline{ab} tanto para denotar a ligação $\{a, b\}$ quanto para a estrutura elementar $\langle \emptyset, \{\overline{ab}\}, 0 \rangle$, que consiste apenas dessa ligação. Naturalmente, alguma ambigüidade é introduzida. Contudo, é sempre fácil identificar pelo contexto a qual dos dois conceitos estamos nos referindo. Caso seja necessário, isso será indicado explicitamente.

Relação de pertinência Para expressar a pertinência ou não de um objeto, ligação ou mensagem a uma estrutura usamos a mesma notação de pertinência de conjuntos. Assim, se E é uma estrutura qualquer e a e b são dois objetos, escrevemos:

$$\begin{aligned} a \in E &\stackrel{\text{def}}{\iff} a \in [E]_O \\ \overline{ab} \in E &\stackrel{\text{def}}{\iff} \overline{ab} \in [E]_L \\ m_b^a \in E &\stackrel{\text{def}}{\iff} [E]_M \geq 1'm_b^a \\ k'm_b^a \in E &\stackrel{\text{def}}{\iff} [E]_M(m_b^a) = k \end{aligned}$$

Adição de estruturas A fim de facilitar a expressão de estruturas, definimos a seguinte operação sobre o conjunto de estruturas. Sejam $E = \langle O, L, M \rangle$ e $E' = \langle O', L', M' \rangle$ duas estruturas quaisquer, a operação $+$ é definida por:

$$E + E' = \langle O \cup O', L \cup L', M + M' \rangle$$

A estrutura algébrica formada pelo conjunto de estruturas, pela operação $+$ definida acima, e pela estrutura nula 0 formam um monóide comutativo. Sejam $E = \langle O, L, M \rangle$, $E' = \langle O', L', M' \rangle$ e $E'' = \langle O'', L'', M'' \rangle$ três estruturas quaisquer. É fácil verificar que:

1. $+$ é associativa

$$\begin{aligned} E + (E' + E'') &= E + \langle O' \cup O'', L' \cup L'', M' + M'' \rangle \\ &= \langle O \cup O' \cup O'', L \cup L' \cup L'', M + M' + M'' \rangle \\ &= \langle O \cup O', L \cup L', M + M' \rangle + E'' \\ &= (E + E') + E'' \end{aligned}$$

2. a estrutura nula 0 é elemento neutro de $+$

$$\begin{aligned} E + 0 &= 0 + E = \langle O \cup \emptyset, L \cup \emptyset, M + 0 \rangle = \langle \emptyset \cup O, \emptyset \cup L, 0 + M \rangle \\ &= \langle O, L, M \rangle \\ &= E \end{aligned}$$

3. $+$ é comutativa

$$\begin{aligned} E + E' &= \langle O \cup O', L \cup L', M + M' \rangle \\ &= \langle O' \cup O, L' \cup L, M' + M \rangle \\ &= E' + E \end{aligned}$$

Devido à associatividade de $+$, podemos escrever expressões compondo mais de duas estruturas sem o uso de parênteses.

Exemplos A linguagem que construímos até este ponto nos permite escrever expressões para representar estruturas e suas propriedades. Embora o principal uso não seja este, é fácil demonstrar¹ que podemos obter qualquer estrutura pela aplicação de $+$ somente a estruturas elementares—ou seja, através da operação $+$, o subconjunto de estruturas elementares gera o conjunto de estruturas. Verifique, por exemplo, que a expressão a seguir representa a estrutura da Figura 4.1:

$$E = a + b + c + \overline{ab} + \overline{ac} + \overline{cd} + m_b^a + n_c^d$$

¹Basta perceber que os conjuntos O , L e M de qualquer estrutura podem ser expressos como a união de conjuntos unitários contendo apenas um objeto ou ligação ou mensagem. Basta, portanto, identificar a partir desses conjuntos as estruturas elementares correspondentes.

Contudo, o uso que fazemos da linguagem está mais relacionado à possibilidade de predicar sobre estruturas e a relacioná-las. Algumas propriedades importantes de estruturas que podem ser facilmente expressas graças à linguagem são:

$$\begin{aligned}\forall a \in E : \quad [E + a]_O &= [E]_O \\ \forall \overline{ab} \in E : \quad [E + \overline{ab}]_L &= [E]_L \\ \forall m : \quad [E + m]_M &= [E]_M + 1'm\end{aligned}$$

Já a fórmula abaixo diz que um objeto pertence a uma estrutura se esta pode ser expressa como a adição da estrutura elementar correspondente a alguma outra estrutura qualquer:

$$\forall a, \forall E : a \in E \iff \exists E' : E = a + E'$$

(Observe que o uso de a para expressar o objeto e a estrutura elementar não cria problemas. O uso do relacional de pertinência indica que se trata do objeto porque só pode relacionar um elemento a uma estrutura. Já o segundo a expressa a estrutura elementar $\langle \{a\}, \emptyset, 0 \rangle$ porque está no contexto de aplicação do operador $+$, definido como lei interna sobre estruturas.)

4.1.3 Estruturas bem-formadas

Nem todas as estruturas que podemos formar são igualmente interessantes para nosso propósito. Considere as três estruturas da Figura 4.2.

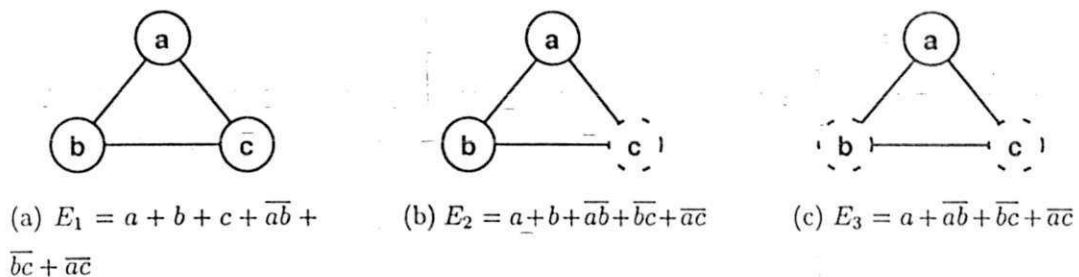


Figura 4.2: Subestruturas bem-formadas e mal-formadas

Não há nada de especial em relação a E_1 . Consiste em três objetos ligados dois a dois, em forma de anel. Embora E_2 seja bastante semelhante a E_1 , há uma particularidade que chama a atenção: as ligações \overline{bc} e \overline{ac} ligam um objeto vivo a um não-vivo (c). Se cada ligação

representa um relacionamento lógico entre dois objetos, então o que significam \overline{bc} e \overline{ac} dado que o objeto c não-existe? Esta condição modela o que chamamos de *ligação inválida*. Trata-se de um modelo adequado para várias situações reais. Por exemplo, se as ligações modelam referências entre objetos do sistema, ligações inválidas modelam adequadamente referências perdidas ou vencidas. Certamente tais situações são indesejáveis em sistemas reais, embora perfeitamente possíveis. Concluimos, portanto, que estruturas como E_2 têm significado adequado.

Analisemos E_3 agora. Neste caso, uma das ligações relaciona dois objetos inexistentes (\overline{bc}). Neste caso, não há significado adequado para estruturas com esta característica. Pela metáfora adotada, deveria ser o modelo de um sistema em que dois objetos inexistentes se encontram relacionados. Relembre que cada estrutura representa um estado de um sistema de objetos. Logo, objetos inexistentes podem pertencer a uma de duas classes: ou são objetos *mortos* que foram vivos em configurações anteriores ou são objetos que nunca existiram e que podem vir a existir em configurações futuras do sistema. Retomemos a estrutura E_3 . Vamos supor que b e c sejam objetos que nunca existiram no passado. Logo, \overline{ab} é a antecipação de um possível relacionamento. Esta interpretação é no mínimo estranha e, portanto, indesejável. Suponha agora que b e c sejam objetos mortos. Então a ligação em E_3 é uma espécie de memória do sistema que registra o fato de que dois objetos mortos foram ligados. Trata-se de outra interpretação estranha e indesejável. Se o sistema deve guardar alguma informação histórica sobre os objetos mortos, isto deve ser feito explicitamente através dos objetos que permanecerem vivos. Afinal, uma configuração é, por definição, a própria representação de estado de um sistema. Só restam duas outras possibilidades, simétricas. Que um dos objetos seja morto e que o outro ainda não tenha sido criado. Pelas mesmas razões anteriores, não há interpretações adequadas.

Este fenômeno é consequência da abordagem que usamos para formar o espaço de estruturas através de um produto cartesiano de três conjuntos. Trata-se de um ponto do produto cartesiano para o qual não há significado adequado. É esta classe de estruturas que queremos banir. A definição abaixo separa o joio do trigo.

Definição 4.2 *Uma estrutura E é dita bem-formada se não tem ligações entre dois objetos inexistentes. Formalmente:*

$$E \text{ é bem formada} \quad \stackrel{\text{def}}{\iff} \quad \forall \overline{ab} \in E : a \in E \text{ ou } b \in E.$$

Caso contrário, a estrutura é dita mal-formada.

Esta definição permite classificar as estruturas E_1 e E_2 descritas no início desta seção como bem-formadas, e E_3 como mal-formada. Interessa-nos, contudo, determinar melhor a natureza das estruturas bem-formadas. Para isso, o conjunto \mathbb{E} definido abaixo será identificado ao conjunto das estruturas bem-formadas. A forma recursiva de definir o conjunto proporciona uma forma de “acessar”, via indução estrutural, cada uma das estruturas bem-formadas.

Definição 4.3 *Definimos \mathbb{E} como o maior conjunto de estruturas obtido pela aplicação recursiva das seguintes regras, onde $a, b \in \mathcal{O}$ são objetos quaisquer, $m \in D$ é um elemento de dado qualquer e E é uma estrutura qualquer de \mathbb{E} :*

1. $0 \in \mathbb{E}$
2. $a + E \in \mathbb{E}$
3. $m_b^a + E \in \mathbb{E}$
4. se $a \in E$, então $\overline{ab} + E \in \mathbb{E}$.

Não é difícil convencer-se e demonstrar que \mathbb{E} corresponde exatamente ao subconjunto das estruturas bem-formadas. Atente para a regra 4: a inclusão de uma estrutura com uma determinada ligação só é possível se um dos objetos (simbolizado por a na regra) já existir na estruturas. Formalizemos esta proposição.

Proposição 4.1 *O conjunto \mathbb{E} definido acima é o próprio conjunto de estruturas bem-formadas. Ou seja, \mathbb{E} contém todas as estruturas bem-formadas e nada mais. Formalmente:*

$$E \in \mathbb{E} \iff E \text{ é bem-formada}$$

Demonstração. Demonstraremos cada sentido da bi-implicação. Em primeiro lugar, demonstraremos que não há estruturas mal-formadas em \mathbb{E} e, em segundo, que toda estrutura bem-formada pertence a \mathbb{E} .

(\Rightarrow) Temos que demonstrar que toda estrutura em \mathbb{E} é bem-formada—ou que não há estruturas mal-formadas em \mathbb{E} . É fácil demonstrar isto por indução estrutural sobre a definição de \mathbb{E} . Em primeiro lugar, observe que 0 é uma estrutura bem-formada (não tem nenhuma

ligação). Isto nos dá a base da indução. Se E é uma estrutura bem-formada (hipótese da indução), então as regras 2, 3 e 4 acima somente nos permitem obter estruturas bem-formadas. Vejamos: $a + E$, onde a é um objeto qualquer, é bem-formada porque incluir um objeto a uma estrutura bem-formada não a pode tornar mal-formada—somente a adição de ligações poderia ter este efeito. Argumento análogo vale para a regra 3. Logo, as regras 2 e 3 só nos permitem obter estruturas bem-formadas. Vejamos agora a regra 4. Ela impõe uma restrição, que exige a existência do objeto na estrutura E usada para obter a nova estrutura. Ora, se $a \in E$, podemos concluir que $a \in \overline{ab} + E$ e, conseqüentemente, esta última é bem-formada. Portanto, também a regra 4 só nos permite obter estruturas bem-formadas. Logo, não há estruturas mal-formadas em \mathbb{E} .

(\Leftarrow) Agora devemos demonstrar que todas as possíveis estruturas bem-formadas pertencem a \mathbb{E} . Usamos redução ao absurdo. Suponha que exista uma estrutura E bem-formada tal que $E \notin \mathbb{E}$. Certamente podemos identificar cada um dos elementos de E e escrever seu conjunto de objetos $[E]_O = \{o_1, \dots, o_{no}\}$, de ligações $[E]_L = \{l_1, \dots, l_{nl}\}$ e de mensagens $[E]_M = \{m_1, \dots, m_{nm}\}$. Isto nos permite expressar E da seguinte forma:

$$E = 0 + \underbrace{o_1 \cdots o_{no}}_{\text{(regra 2)}} + \underbrace{m_1 \cdots m_{nm}}_{\text{(regra 3)}} + \underbrace{l_1 \cdots l_{nl}}_{\text{(regra 4)}}$$

Uma análise cuidadosa da expressão permite concluir que $E \in \mathbb{E}$. Basta observar que a expressão determina a seqüência de aplicações das regras da Definição 4.3 necessárias para obter E , provando, por definição, que pertence a \mathbb{E} . Vejamos: E é obtida a partir da estrutura nula (regra 1). Em seguida, cada um dos objetos é adicionado (regra 2). Depois, cada uma das mensagens (regra 3). E, finalmente, cada ligação é adicionada (regra 4). Observe que a regra 4 pode ser aplicada sem problemas. A condição é satisfeita porque E é, por hipótese, bem-formada. Ou seja, para cada ligação em E , pelo menos um dos objetos relacionados é vivo. Ora, concluir que $E \in \mathbb{E}$ é um absurdo porque por hipótese $E \notin \mathbb{E}$. Logo, por contradição, concluímos que não há estrutura bem-formada fora de \mathbb{E} , o que conclui nossa demonstração. \square

Corolário 1 *Uma estrutura só é bem-formada se admite ser expressa de acordo com a sintaxe abstrata definida pela gramática:*

$$E ::= 0 \mid a + E \mid m_b^a + E \mid E + a + \overline{ab}$$

Demonstração. Basta observar que cada alternativa da sintaxe corresponde respectivamente a uma das regras da Definição 4.3. Toda expressão é iniciada pela estrutura nula—regra 1.

Objetos e mensagens podem ser adicionados livremente—regras 2 e 3. Ao adicionar ligações, contudo, deve ser adicionado um dos objetos relacionados. Isto equivale à condição da regra 4 da definição de \mathbb{E} . \square

Observe que o conjunto das estruturas bem-formadas preserva as propriedades básicas do conjunto de estruturas global. Em particular, também é um monóide comutativo. Vejamos isto formalmente.

Proposição 4.2 *O conjunto \mathbb{E} das estruturas bem-formadas é um monóide comutativo (submonóide do monóide de estruturas).*

Demonstração. Basta observar que \mathbb{E} contém o elemento neutro 0 e preserva as propriedades de associatividade e comutatividade. \square

Corolário 2 *O conjunto \mathbb{E} é fechado em relação a +.*

Demonstração. Por contradição. Suponha que haja duas estruturas bem-formadas $E, E' \in \mathbb{E}$ tais que $E + E' \notin \mathbb{E}$. Para isto é necessário que haja uma ligação $\overline{ab} \in E + E'$ tal que $a \notin E + E'$ e $b \notin E + E'$. Se $\overline{ab} \in E + E'$ então certamente $\overline{ab} \in E$ ou $\overline{ab} \in E'$. O primeiro caso implica que $a \in E$ ou que $b \in E$. O segundo, que $a \in E'$ ou que $b \in E'$. De ambos podemos concluir que $a \in E + E'$ ou $b \in E + E'$, o que seria um absurdo pois indicaria que $E + E'$ é bem-formada, contradizendo nossa hipótese. \square

4.1.4 Subestruturas e partições

Subestruturas Em algumas situações é necessário e útil comparar estruturas. Com este objetivo definimos a relação (parcial) \leq . Dizemos que uma estrutura E é “menor ou igual à” estrutura E' (ou que E é subestrutura de E'), e escrevemos $E \leq E'$, se e somente se as seguintes condições forem satisfeitas:

1. $[E]_O \subseteq [E']_O$, todos os objetos de E são objetos de E' ;
2. $[E]_L \subseteq [E']_L$, todas as ligações de E são de E' ; e
3. $[E]_M \subseteq [E']_M$, todas as mensagens de E são de E' .

Exemplo Considere a estrutura $E = a + \overline{ab} + b$. A estrutura admite sete subestruturas, especificamente: 0, a , b , \overline{ab} , $a + \overline{ab}$, $b + \overline{ab}$ e $a + \overline{ab} + b$ —as subestruturas não nulas estão representadas graficamente na Figura 4.3.

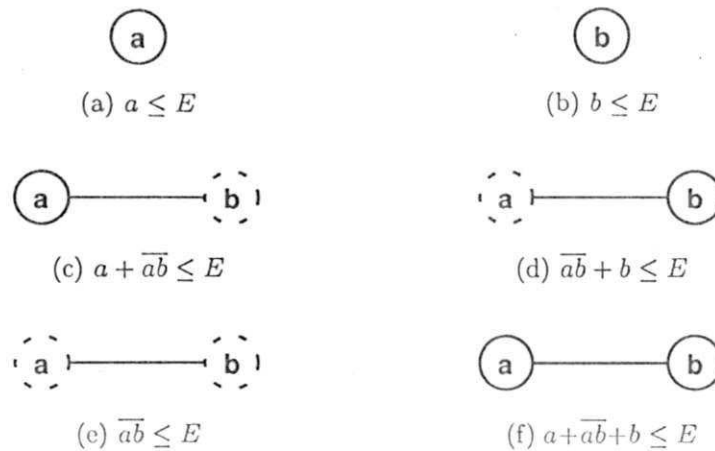


Figura 4.3: Subestruturas não nulas de $E = a + \overline{ab} + b$

Subestruturas próprias A relação \leq , permite relacionar duas estruturas em que a primeira é dita subestrutura da segunda. Contudo, a relação é fraca e permite relacionar estruturas mal-formadas a estruturas bem-formadas, o que nem sempre é interessante. Por exemplo, perceba que $\overline{ab} \leq a + \overline{ab} + b$, embora a subestrutura \overline{ab} não faça sentido isoladamente (não é bem-formada). Para resolver isto, definimos uma relação mais forte entre estruturas que preserva suas características mais importantes. A nova relação consiste em um refinamento da relação \leq , definida apenas sobre estruturas bem-formadas. Vejamos.

Definição 4.4 *Seja E uma estrutura bem-formada e E' uma subestrutura de E ($E' \leq E$). Dizemos que E' é uma subestrutura própria de E , denotado por $E' \preceq E$, se E' é bem-formada e preserva todas as ligações e mensagens pendentes de E "referentes" a seus objetos. Ou seja, dizemos que $E' \preceq E$ se:*

1. a subestrutura preserva todas as ligações de seus objetos:

$$\forall x \in E' : \overline{xy} \in E \Rightarrow \overline{xy} \in E'$$

2. a subestrutura preserva todas as mensagens destinadas a seus objetos:

$$\forall x \in E', \forall k, \forall m : [E]_M(m_x^y) = k \Rightarrow [E']_M(m_x^y) = k$$

3. e, finalmente, se a subestrutura é bem-formada, $E' \in \mathbb{E}$.

Partições Definido o conceito de subestrutura própria, podemos definir *partições*. Antes, porém, é necessário introduzirmos a seguinte notação: as estruturas E_1, \dots, E_n serão ditas *o-disjuntas* se não têm objetos vivos em comum, ou seja se seus conjuntos de objetos são disjuntos. Formalmente:

$$E_1, \dots, E_n \text{ são o-disjuntas} \quad \stackrel{\text{def}}{\iff} \quad \bigcap_{i=1}^n [E_i]_O = \emptyset.$$

Definição 4.5 *Seja E uma estrutura definida pela adição de subestruturas com conjuntos de objetos não vazios², $E = E_1 + \dots + E_n$. Dizemos que o conjunto $\{E_1, \dots, E_n\}$ é uma o-partição de E , ou simplesmente uma partição, se as estruturas E_i são o-disjuntas. Caso $E_i \preceq E$, para todo $i = 1..n$, a partição é dita regular.*

Exemplo Observe que das subestruturas de $a + \overline{ab} + b$ somente $a + \overline{ab}$, $b + \overline{ab}$ e $a + \overline{ab} + b$ são próprias—Figuras 4.3(c), 4.3(d) e 4.3(f). Perceba que toda estrutura bem-formada com conjunto de objetos não vazio é subestrutura própria de si mesma. A seguir, enumeramos todas as partições admitidas por E (as subestruturas que as formam são indicadas pelos parênteses):

$$\begin{aligned} E &= (a) + (\overline{ab} + b) \\ &= (a + \overline{ab}) + (b) \\ &= (a + \overline{ab}) + (b + \overline{ab}) \\ &= (a + \overline{ab} + b) \end{aligned}$$

Em cada linha, E é definida como a soma de subestruturas bem-formadas, diferentes de 0. Contudo, somente as duas últimas são partições regulares—as subestruturas são próprias.

Perceba que qualquer estrutura é partição regular de si mesma—dita partição trivial. Em particular, estruturas com um único objeto só admitem a partição trivial. Observe também que cada partição da estrutura determina uma partição do conjunto de objetos—isto decorre diretamente da definição. O que é mais interessante, embora não seja tão óbvio, é que cada partição do conjunto de objetos induz uma partição regular da estrutura. Os lemas preparam o caminho para esse resultado. Antes, porém, introduzamos a seguinte

²É necessário que $[E_i]_O \neq \emptyset$, para $i = 1..n$ para manter a relação com o conceito de partição de conjuntos.

notação: uma mensagem m_b^a será dita *morta* na estrutura E se seu destinatário não é vivo em E , ou seja, se $b \notin E$.

Lema 1 *Seja $E \in \mathbb{E}$ uma estrutura sem mensagens mortas e $\{O_1, \dots, O_n\}$ uma partição de $[E]_O$. Então $\{E_1, \dots, E_n\}$, com E_i definidos abaixo, é uma partição regular de E :*

$$\begin{aligned} [E_i]_O &= O_i \\ [E_i]_L &= \{ \overline{xy} \mid \overline{xy} \in E \text{ e } x \in O_i \} \\ [E_i]_M(m_x^y) &= \begin{cases} k & \text{se } [E]_M(m_x^y) = k \text{ e } x \in [E_i]_O \\ 0 & \text{nos demais casos.} \end{cases} \end{aligned}$$

Demonstração. Pela Definição 4.5, para provar que $\{E_1, \dots, E_n\}$ é uma partição regular devemos demonstrar que:

1. as estruturas E_i são o-disjuntas e possuem pelo menos um objeto
2. que $E = E_1 + \dots + E_n$;
3. e que $E_i \preceq E$, para todo $i = 1..n$.

Vejamos ponto a ponto:

1. Relembre que $[E_i]_O = O_i$. Se $\{O_1, \dots, O_n\}$ é uma partição de O , então os conjuntos O_i , e conseqüentemente os $[E_i]_O$, são não-vazios e disjuntos.
2. Por definição, $[E]_O = O$. Também sabemos que $O = \bigcup_{i=1}^n O_i$, o que nos leva a concluir que $[E]_O = \bigcup_{i=1}^n [E_i]_O$. Além disso, toda ligação $\overline{xy} \in E$ pertence a algum E_i , desde que $x \in E_i$. Logo, $[E]_L = \bigcup_{i=1}^n O_i [E_i]_L$. Finalmente, toda mensagem $m_b^a \in E$ pertence a algum E_i , desde que $x \in E_i$. Observe que não há ligação ou mensagem que não esteja em algum E_i . No caso das ligações isto ocorre porque E é bem-formada, logo pelo menos um E_i a conterà. No caso da mensagem, o lema exige que a estrutura não tenha mensagens mortas. Logo, o mesmo argumento vale. Ora, se $[E]_O = [E_1 + \dots + E_n]_O$, $[E]_L = [E_1 + \dots + E_n]_L$ e $[E]_M = [E_1 + \dots + E_n]_M$, então concluímos que $E = E_1 + \dots + E_n$.
3. Sabemos, por construção, que $E_i \leq E$. Contudo, devemos provar que cada E_i contém todas as ligações e mensagens referentes a seus respectivos objetos. Ora, isto é evidente pela própria definição de $[E_i]_L$, que é formado por todas as ligações que se refiram a algum objeto de E_i . Argumento semelhante vale para as mensagens. Observe que

a definição de $[E_i]_M$ determina que o multi-conjunto de mensagens de E_i tem igual número de ocorrências que E , das mensagens cujo destinatário pertence a E e nenhuma das demais. Logo, todo E_i é subestrutura própria de E . \square

Lema 2 *A partição $\{E_1, \dots, E_n\}$ é a única partição regular de E que respeita a partição de objetos $\{O_1, \dots, O_n\}$, ou seja, tal que $[E_i]_O = O_i$.*

Demonstração. Por contradição. Suponha que haja outra partição, digamos $\{E'_1, \dots, E'_n\}$, diferente de $\{E_1, \dots, E_n\}$ que respeite a partição de objetos, ou seja, tal que $[E'_i]_O = O_i$. Suponha que ambas as partições estejam ordenadas segundo a partição de objetos, ou seja, para qualquer $i = 1..n$, temos que $[E_k]_O = [E'_k]_O$.

Ora, se $\{E'_1, \dots, E'_n\} \neq \{E_1, \dots, E_n\}$, então necessariamente, existe algum k para o qual $E_k \neq E'_k$. Isto só é possível se um das alternativas abaixo é verdadeira:

1. ou $[E_k]_L \neq [E'_k]_L$
2. ou $[E_k]_M \neq [E'_k]_M$.

Se a primeira alternativa for verdadeira então há algum $\bar{x}\bar{y} \in E_k$ tal que $\bar{x}\bar{y} \notin E'_k$ (ou vice-versa). Ora, isto nos leva a uma contradição. Se $\bar{x}\bar{y} \in E_k$ e E_k é partição regular de E , então $\bar{x}\bar{y} \in E$. Por outro lado, se $\bar{x}\bar{y} \notin E'_k$ e E'_k também é partição regular de E , então $\bar{x}\bar{y} \notin E$. Logo, a primeira alternativa acima não é possível e necessariamente temos que $[E_k]_L = [E'_k]_L$.

A segunda alternativa nos leva a contradição semelhante. Se for verdadeira, então há alguma mensagem com número de ocorrências diferentes em E_k e E'_k . Ou seja, para alguma mensagem $m_b^a \in E$, $[E_k]_M(m_b^a) \neq [E'_k]_M(m_b^a)$. Se E_k é uma subestrutura própria de E , então o número de ocorrências de uma mensagem destinada a um objeto em E_k é igual ao número de ocorrência da mesma mensagem em E . Como o mesmo vale para E'_k , que também é subestrutura própria de E , temos uma contradição: $[E]_M(m_b^a) = [E_k]_M(m_b^a) \neq [E'_k]_M(m_b^a) = [E]_M(m_b^a)$! Logo, é necessário que $[E_k]_M = [E'_k]_M$.

Ora, então chegamos às seguintes conclusões: $[E_k]_L = [E'_k]_L$, $[E_k]_M = [E'_k]_M$. Como sabemos que $[E_k]_O = [E'_k]_O$, podemos concluir que $E_k = E'_k$, o que é a última contradição. Somos levados a aceitar, portanto, que não há duas partições regulares que respeitem uma mesma partição de objetos para uma dada estrutura sem mensagens mortas. Portanto, a partição dada no Lema 1 é única. \square

Proposição 4.3 *Toda estrutura $E \in \mathbb{E}$ admite pelo menos uma partição regular.*

Demonstração. Seja $E' \leq E$ a maior subestrutura de E sem mensagens mortas, e $E'' = E - E'$ uma estrutura composta apenas pelas mensagens mortas de E . Pelos Lemas 1 e 2, sabemos que E' admite uma única partição regular, digamos $\{E_1, \dots, E_n\}$. Logo, é fácil perceber que $\{E_1 + E'', E_2, \dots, E_n\}$ é uma partição regular de E . \square

Corolário 3 *Seja $E \in \mathbb{E}$ uma estrutura e $\{O_1, \dots, O_n\}$ uma partição de $[E]_O$. Logo, E admite n^m partições regulares, onde m é o número de mensagens mortas em E .*

Demonstração. Seja $\{E_1, E_2, \dots, E_n\}$ a partição regular da estrutura $E' \leq E$, que contém apenas as mensagens mortas de E . Se E' tem m mensagens pendentes, então é possível arranjá-las nas n subestruturas E_i , formando partições regulares de E . Logo, há n^m possíveis arranjos e, conseqüentemente, partições regulares. \square

Corolário 4 *Estruturas sem mensagens mortas admitem uma única partição regular para cada partição $\{O_1, \dots, O_n\}$ de seu conjunto de objetos.*

Demonstração. Conseqüência direta do Corolário 3. \square

Diferença de estruturas Outra operação que será bastante útil mais à frente é a de diferença de estruturas. Ela é definida para todas as estruturas $E = \langle O, L, M \rangle$ e $E' = \langle O', L', M' \rangle$ tais que $E' \leq E$, por:

$$E - E' = \langle O \setminus O', L \setminus L', M - M' \rangle$$

É importante atentar para a relação entre os operadores $+$ e $-$. Relembre que as estruturas formam um monóide quando munidas de $+$ e não um grupo. Isto significa que não há simétricos em relação à adição de elementos. Logo, embora seja fato que

$$E_1 - E_2 = E_3 \Rightarrow E_3 + E_2 = E_1,$$

a inversa não é verdadeira:

$$E_3 + E_2 = E_1 \not\Rightarrow E_1 - E_2 = E_3$$

Um exemplo trivial disso é dado quando as três estruturas são idênticas. Por exemplo, se $E_3 = E_2 = E_1 = a$. Sabemos que $E_3 + E_2 = a = E_1$. Contudo $E_1 - E_2 = 0 \neq E_3$.

Definição 4.6 *Dizemos que duas estruturas E e E' são complementares se são o-partições regulares de $E + E'$. Neste caso, denotamos a soma por $E \oplus E'$.*

4.1.5 Conclusões

Há várias vantagens em encontrar uma forma recursiva de definir \mathbb{E} . A principal delas, contudo, é a confiança que provê sobre construção que mentalizamos. O conjunto assim definido tem propriedades bastante claras e fáceis de verificar indutivamente. Isto só é possível porque dispomos de um mecanismo que nos permite construir argumentos indutivos. As proposições vistas demonstram isto.

Um comentário final: a abordagem que utilizamos para definir as estruturas de configurações de sistemas de objetos consiste em formar um produto cartesiano. Isto nos permite definir seus componentes separadamente. Em compensação, o resultado do produto tem pontos sem significado adequado. A forma de eliminar esses pontos é a questão principal. Ao invés de simplesmente incluímos um axioma, banindo todos os casos inválidos conhecidos, abordamos a questão algebricamente. Ou seja identificamos o conjunto alvo a partir de poucos elementos bem conhecidos e um conjunto de regras simples cujas propriedades podemos verificar facilmente³.

4.2 Configurações

Conceitualmente, uma configuração é a caracterização plena de um sistema de objetos em um instante particular de seu funcionamento. A ênfase é empregada para ressaltar que a forma como um sistema de objetos se *configura* em um dado instante não determina somente os elementos estruturais de que é constituído. Determina também todo o possível comportamento futuro do sistema a partir daquele instante. Formalizamos o primeiro aspecto através do conceito de estrutura. Agora, voltamos nossa atenção para o segundo aspecto de uma configuração, a que chamamos *dinâmica*.

Relembre que, pelo modelo conceitual adotado, todo evento de um sistema corresponde a ações executadas pelos seus objetos. Logo, embora diante de nosso propósito seja importante evitarmos ou retardarmos ao máximo a escolha de uma representação para a estrutura interna dos objetos⁴, é inevitável que a definição dos conceitos de dinâmica e

³“Quando construímos um espaço de estados através de um produto cartesiano, não é de forma alguma certo que teremos bom uso para todos os seus pontos” [Dij76, p. 13]

⁴É importante lembrar que podemos representar o funcionamento interno dos objetos através de redes de Petri. Daí porque evitamos escolher uma representação para a estrutura interna dos objetos. Mais à

configuração explorem alguma noção de representação interna dos estados dos objetos e de seu comportamento.

Ações e eventos Antes de prosseguirmos em nossa apresentação é importante que o leitor atente para a distinção que fazemos entre os termos *ação* e *evento*. Usamos *evento*, como talvez já tenha ficado claro, para expressar uma ocorrência em um sistema de objetos. Um evento “leva” um sistema de objetos de uma configuração para outra. Por outro lado, usamos *ação* para expressar uma ocorrência provocada por e em algum objeto do sistema. Enquanto o termo *evento* tem a acepção de acontecimento sem enfatizar causa ou autoria, *ação* pressupõe a existência de um agente, um causador da ocorrência. Logo, a escolha dos termos evidencia a natureza de *agente* dos objetos. Naturalmente, os dois conceitos se relacionam intrinsecamente e veremos, mais adiante, que todo evento corresponde a pelo menos uma ação.

4.2.1 Ações e estados internos

Um objeto é uma unidade computacional discreta e autônoma que encapsula seu próprio estado, comportamento e atividade (ver modelo conceitual). Logo, podemos caracterizar o comportamento de um objeto pelos seus respectivos estados e ações. Formalmente, faremos isto através de uma relação de transição definida sobre seus estados internos, que rotulamos com as respectivas ações.

Ações Inicialmente definimos o conjunto das ações que podem ser executadas pelos objetos (ou os rótulos que usamos nas transições de estado). As ações elementares correspondem às identificadas no modelo conceitual. A gramática abaixo caracteriza *ações elementares* (categoria *e*) e *ações* (categoria *a*) a partir de objetos *x* e *y*:

$$\begin{aligned} e &::= \tau \mid \text{new } x \mid x?y \mid x.y \mid x!y \mid \tilde{x} \mid \text{end} \\ a &::= e \mid a \circ a \end{aligned} \tag{4.5}$$

Os nomes e a interpretação de cada tipo de ação elementar são apresentados na Tabela 4.1. Ações compostas, ou simplesmente ações, são formadas pela justaposição de ações elementares cuja execução é atômica. Naturalmente, uma ação composta deve ter como frente, os dois formalismos serão integrados, complementado-se.

AÇÃO	NOME
τ	ação local ou interna
new x	criação ou instanciação de objetos
$x?m$	entrada de dados
$x.m$	saída assíncrona de dados
$x!m$	saída síncrona de dados
\tilde{x}	remoção de ligação ou desligamento
end	ação final ou auto-destruição

(Onde x é um objeto e m é um elemento de dado.)

Tabela 4.1: Ações elementares

efeito a combinação dos efeitos das ações elementares de que é formada—veremos isso mais à frente. Alguns exemplos de ações definidas pela categoria a são:

$$a?m \circ \text{end}$$

$$a?m \circ b.m \circ c!n$$

$$a?m \circ b.m \circ b.m$$

Comportamento Não definimos formalmente a estrutura de um estado interno. Contudo, assumimos que a cada objeto x corresponde um conjunto de estados internos que serão representados por x_0, x_1, \dots (por convenção, os estados de um objeto serão sempre representados como uma família x_i , onde x é o objeto a que os estados se referem e $i = 0, 1, 2, \dots$, para uma ordenação arbitrária dos estados—reservamos o índice 0 para indicar o estado inicial do objeto.).

Para representar o comportamento de um objeto definimos uma relação de transição \rightarrow , com elementos da forma $\langle x_i, a \rangle \rightarrow x_j$ (ou, mais simplesmente, $x_i \xrightarrow{a} x_j$). Cada elemento da relação é dito uma *ação potencial* do objeto, cuja interpretação é: “se x_i é o estado interno de x , então ao executar a ação a , x passa para o estado interno x_j ”. A ação a é dita *iminente* ou *habilitada* no estado x_i . Contudo, há que atentar para esta interpretação. O fato de x se encontrar no estado x_i não dá garantias de que possa executar a . Diz apenas que x reúne as condições internas necessárias para tal ação e que, caso a execute, o estado alcançado será x_j .

Pseudo-estado Intuitivamente, a ação end faz um objeto terminar seu ciclo-de-vida. Logo, não há estado alcançado quando esta ação ocorre. Para resolver este problema

técnico, definimos o pseudo-estado denominado “morto”, que será denotado por “■”, para indicar o “estado alcançado” por ações desse tipo em qualquer objeto. Naturalmente, nenhuma ação pode estar habilitada no pseudo-estado “morto”.

Exemplo Considere um objeto capaz de receber uma mensagem de um segundo objeto e repassá-la para um terceiro. Para exemplificar, considere ainda que o objeto se auto-destrói após repassar a mensagem. Se indicarmos o objeto em questão por a , e os outros dois por b e c , respectivamente, podemos representar seu comportamento pela relação \rightarrow , abaixo:

$$a_0 \xrightarrow{b?.m} a_1 \quad a_1 \xrightarrow{c.m} a_2 \quad a_2 \xrightarrow{\text{end}} \blacksquare$$

Esta forma de descrever o comportamento de um objeto pode ser incluída na abordagem de autômatos. Logo, é importante que o leitor perceba as características desta abordagem. Em primeiro lugar, não há nenhum limite quanto ao número de estados de um objeto, ou quanto ao número de ações habilitadas a partir de um estado. Potencialmente, ambos são infinitos. Em segundo lugar, o número de rótulos disponíveis é efetivamente infinito: podemos combinar ações elementares, formando ações complexas, que são “novos” rótulos. E, finalmente, é importante enfatizar que a interpretação efetiva do comportamento de um objeto é dependente de condições externas a ele. Toda ação, exceto ações internas (ou locais), tem pré-condições e efeitos relacionados à estrutura da configuração (tome como exemplo ações de entrada e saída de dados). Mais à frente, ao definirmos precisamente o conceito de evento, tratamos esta questão em mais detalhes.

4.2.2 Configurações

Escolhida uma representação para o comportamento dos objetos, podemos retomar o conceito de configuração. Observe que já dispomos de elementos para definir formalmente o conceito. Se aliarmos a uma estrutura, a caracterização do comportamento de seus objetos e indicarmos o estado em que cada um deles se encontra, teremos caracterizado precisa e formalmente o que entendemos por configuração. Vejamos isto passo-a-passo.

Comportamento de objetos Na seção anterior vimos como podemos representar o comportamento de um objeto através de uma relação de transição \rightarrow . Relembre, contudo, que adotamos uma convenção para representar estados internos: estados do objeto x serão

denotados por x_0, x_1, \dots . Logo, podemos sobrecarregar o uso da relação \rightarrow para caracterizar o comportamento de vários objetos sem que haja qualquer confusão. Por exemplo, a relação abaixo definida caracteriza o comportamento de dois objetos, a e b :

$$a_0 \xrightarrow{b.m} \blacksquare \qquad b_0 \xrightarrow{a.m} \blacksquare$$

Estados internos Uma escolha natural para representar os estados dos objetos de uma configuração é um mapeamento que atribua a cada objeto um de seus estados internos—uma função de atribuição. Porém, também devido à convenção que adotamos para denotar estados internos, esta opção é redundante. É preferível, e mais simples, indicar diretamente o conjunto dos estados internos. Observe que isto não introduz nenhuma ambigüidade porque a própria representação do estado o associa unicamente ao objeto a que se refere. Veja, como exemplo, a representação dos estados iniciais (relembre que reservamos o índice 0 para isto) de uma configuração com três objetos a , b e c nas duas formas propostas (no primeiro caso, σ é uma função de atribuição; no segundo, o conjunto de estados internos):

$$\sigma(x) = \begin{cases} a_0 & \text{se } x = a \\ b_0 & \text{se } x = b \\ c_0 & \text{se } x = c \end{cases} \quad \text{ou, simplesmente,} \quad \sigma = \{a_0, b_0, c_0\}$$

Configurações Como já dissemos, os conceitos introduzidos são suficientes para definirmos formalmente configuração e dinâmica. Uma estrutura, uma declaração de estados internos para os objetos vivos e uma especificação de comportamento de objetos determinam precisamente uma configuração.

Definição 4.7 *Uma configuração de um sistema de objetos consiste em uma estrutura E ; em um conjunto de estados internos σ para os objetos vivos da estrutura; e numa relação \rightarrow que caracteriza o comportamento desses objetos. Logo, denotamos a configuração pela tupla $\langle E, \sigma, \rightarrow \rangle$. O par $\langle \sigma, \rightarrow \rangle$ é dito a dinâmica da configuração.*

4.2.3 Notação

Nem sempre é conveniente expressar e manipular configurações segundo a definição dada. Por isto, adotamos algumas convenções que nos permitem uma notação mais simples e de fácil compreensão.

Em primeiro lugar, assumimos que o comportamento dos objetos de um sistema é pré-determinado e constante. Por pré-determinado entenda que o comportamento de qualquer objeto é dado independentemente do resto do sistema. Por constante, entenda que o comportamento de um objeto é idêntico em todas as configurações em que existir. Isto equivale a assumir que tal comportamento é determinado precisamente, ainda que o processo pelo qual é definido não seja de nosso interesse. Em outras palavras, que fixado um determinado sistema, podemos subentender a relação \rightarrow e, portanto, podemos representar suas configurações por pares $\langle E, \sigma \rangle$.

Em segundo lugar, para enfatizar os aspectos globais de uma configuração, estendemos a notação de estruturas para representá-las. A idéia é “diluir” a representação de estados internos na estrutura. Fazemos isto, substituindo cada objeto nas expressões que representam a estrutura pela representação do estado interno em que se encontra. Formalmente, podemos definir a notação como:

$$E\{x_i/x\} \stackrel{\text{def}}{=} \langle E, \sigma \rangle \quad \text{para todo } x_i \in \sigma$$

Onde $E\{x_i/x\}$ denota a cadeia obtida pela substituição de todos os elementos x por x_i na representação de E como soma formal. Isto não inclui a substituição das “referências” a x em ligações ou mensagens pendentes. Por exemplo, a notação adotada nos permite escrever:

$$a_i + \overline{ab} + b_j \stackrel{\text{def}}{=} \langle a + \overline{ab} + b, \{a_i, b_j\} \rangle.$$

Outra representação Às vezes é conveniente enfatizar as ações iminentes dos objetos em uma configuração, sem expressar toda a dinâmica, ou mesmo os estados dos objetos. Para isto, escrevemos $x:a$ para representar que a ação a está iminente para algum estado x_i do objeto x . Em seguida, podemos definir o conjunto das ações iminentes de uma configuração por:

$$\{ x:a \mid x_i \xrightarrow{a} x_j \quad \forall x_i \in \sigma \}$$

Anexando este conjunto a alguma representação simplificada da configuração, em que omitimos a representação completa da dinâmica, obtemos uma nova representação mais detalhada da configuração. Às vezes usamos até mesmo a estrutura apenas.

Exemplos Um exemplo trivial de configuração é a denominada configuração *vazia* ou *nula*, que formalmente corresponde à configuração $\langle 0, \emptyset \rangle$, mas que representamos simplificada por 0. Vejamos outros exemplos:

$$1. \langle a + \overline{ab} + b, \{a_1, b_1\}, \{a_0 \xrightarrow{b?m} a_1, a_1 \xrightarrow{b.n} a_2, b_0 \xrightarrow{a.m} b_1, b_1 \xrightarrow{a?n} b_2\} \rangle$$

Trata-se da representação completa de uma configuração, seguindo a definição formal. Observe que expressar todos os elementos da definição da configuração apenas dificulta a leitura—embora a configuração seja extremamente simples.

$$2. a_1 + \overline{ab} + b_1$$

Trata-se da mesma configuração do exemplo anterior. Usamos, contudo, a representação simplificada: os estados internos dos objetos estão expressos diretamente na estrutura e não detalhamos o comportamento de cada objeto. Relembre que o comportamento dos objetos é constante ao longo do funcionamento do sistema e, portanto, pode ser expresso separadamente, evitando obscurecer a notação.

$$3. \langle a_1 + \overline{ab} + b_1, \{a:b.n, b:a?n\} \rangle$$

Novamente a mesma configuração. Desta vez a dinâmica se encontra parcialmente expressa, indicando apenas as ações iminentes. Observe que há considerável ganho em relação ao primeiro exemplo, sem que fiquem subentendidos todos os aspectos de comportamento.

Representação gráfica Já vimos como desenhamos estruturas. Naturalmente, não seria conveniente representar graficamente todo o comportamento dos objetos de um sistema. Logo, desenhamos configurações, explorando a mesma idéia que usamos para simplificar a notação (ver exemplos acima), indicando apenas a parte mais relevante da dinâmica: os estados internos e as ações iminentes. Os estados internos dos objetos são indicados nos próprios nós dos objetos, em substituição ao nomes, ou são omitidos quando for conveniente. O conjunto das ações iminentes de cada objeto é representado como uma anotação próximo ao respectivo nó. A Figura 4.4 ilustra a configuração usada nos exemplos acima.

Sub-configuração Outros conceitos que podem ser úteis são as de subconfigurações, partições de uma configuração e configurações complementares. A idéia é seguir o mesmo princípio dos conceitos correspondentes para estruturas.

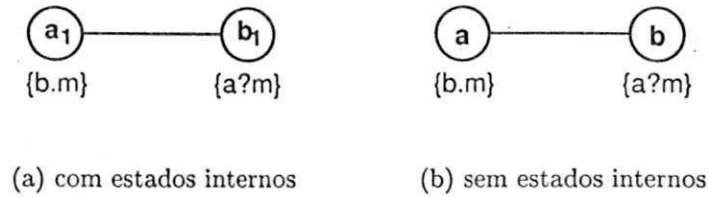


Figura 4.4: Representação gráfica de $\langle a_1 + \overline{ab} + b_1, \{a:b.m, b:a?m\} \rangle$

Também pode ser útil definir operações para manipular configurações diretamente. Pode ser útil ter formas de “adição” e “subtração” de configurações para expressar regras de comportamento.

4.3 Sistemas de Objetos

Antes das definições formais, vejamos alguns exemplos de sistemas de objetos. Embora ainda distante do que poderíamos chamar de sistemas realistas, estes exemplos caracterizam situações típicas de vários sistemas. Além disto, esta seção permitirá antecipar a notação que adotamos para sistemas de objetos.

Exemplo 4.1 *Considere um sistema cuja configuração inicial é*

$$\langle a_0 + \overline{ab} + b_0, \{a:b.m, b:a?m\} \rangle.$$

O sistema tem dois objetos, a e b, ligados um ao outro. O objeto a está apto a enviar uma mensagem para b e, correspondentemente, b está apto a receber essa mensagem—ambos os objetos estão em seus estados iniciais. Observe ainda que a ação iminente em a é assíncrona. Logo, o único evento possível no sistema corresponde a essa ação. Ao ocorrer, o sistema passa à configuração

$$\langle a_1 + \overline{ab} + b_0 + m_b^a, \{b:a?m\} \rangle.$$

Nesta configuração, à estrutura do sistema foi adicionada a mensagem m_b^a , enviada por a para b. Observe também que só a mudou de estado interno, dado que apenas ele executou alguma ação. A nova configuração habilita b a executar sua ação iminente $b:a?m$ para receber a mensagem. A configuração alcançada após este evento é

$$\langle a_1 + \overline{ab} + b_1, \{\} \rangle.$$

Nesta configuração, o sistema retorna a sua estrutura original—sem a mensagem pendente. Contudo, como não há mais ações habilitadas, também não há mais eventos possíveis e o sistema termina sua atividade.

Na Figura 4.5 representamos graficamente o sistema descrito. As setas rotuladas entre as configurações indicam os eventos, caracterizando a transição do sistema entre seus estados. Os rótulos das setas indicam as ações correspondentes ao evento.

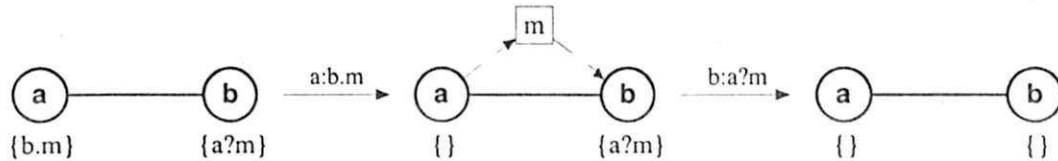


Figura 4.5: Comunicação assíncrona entre dois objetos

Exemplo 4.2 Considere agora o sistema cuja configuração inicial é

$$\langle a_0 + \overline{ab} + b_0 + \overline{ac} + c_0, \{a:b!m, a:c!m, b:a?m, c:a?m\} \rangle.$$

O objeto a está apto a enviar, sincronamente, a mensagem m , tanto para b quanto para c . Correspondentemente, b e c estão aptos a receber a mensagem. Logo, em cada caso podemos obter as seguintes configurações:

$$\langle a_1 + \overline{ab} + b_1 + \overline{ac} + c_0, \{c:a?m\} \rangle \quad \text{ou} \quad \langle a_1 + \overline{ab} + b_0 + \overline{ac} + c_1, \{b:a?m\} \rangle.$$

Observe que em ambos os casos, a sincroniza com um dos objetos, e deixa de estar apto a sincronizar com o outro. Isto evidencia que, internamente, o objeto a estava apto a sincronizar com apenas um dos dois—tínhamos uma situação de conflito

Exemplo 4.3 O sistema representado na Figura 4.6 modela um sistema do tipo produtor-consumidor com buffer ilimitado. Os itens são modelados por mensagens pendentes da forma m_i^a . Logo, o objeto a é o produtor e b , o consumidor. Na configuração inicial, à esquerda, o único evento possível é $a:b.m$. Na segunda configuração alcançada pelo sistema, a ação $a:b.m$ continua iminente, o que indica que o objeto a continua apto a produzir um novo item. Como o sistema tem um número infinito de configurações, as demais estão representadas de forma esquemática. Observe que o sistema acumula itens produzidos na forma de mensagens pendentes na estrutura. Observe também, que a partir da segunda configuração, o objeto b pode consumir os itens produzidos, fazendo o sistema retornar às configurações mais à esquerda

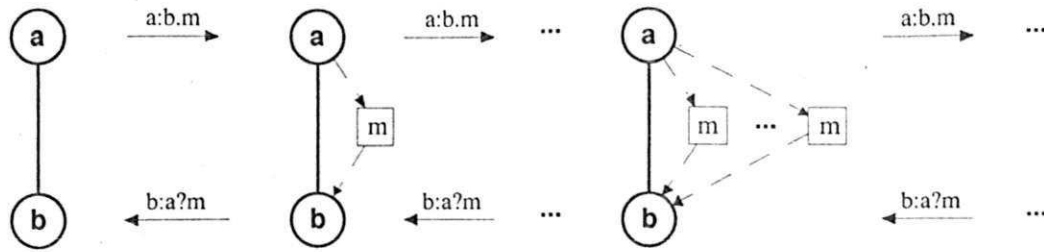


Figura 4.6: Sistema produtor/consumidor

4.3.1 Avaliação do efeito das ações

Aplicação de ações sobre estruturas Escrevemos $\langle E, a \rangle \rightarrow E'$ para indicar que a aplicação da ação a sobre a estrutura E tem o efeito de transformá-la em E' . As regras a seguir formalizam os efeitos da aplicação de uma ação sobre uma estrutura.

Ações internas Conceitualmente, o efeito de uma ação interna é nulo sobre a estrutura. Logo, definimos:

$$\langle E, x:\tau \rangle \rightarrow E$$

Instanciação A criação de objetos ou instanciação não deve apenas criar um objeto, mas também uma ligação entre o objeto criado e o objeto criador. Além disso, esta ação só está definida se o objeto a ser criado é *novo* na estrutura. Por novo devemos entender que além de não ser vivo, também não há referências a ele em termos de ligações ou mensagens pendentes—indicamos que x é novo em E escrevendo $x \notin! E$. Então definimos:

$$\langle E, x:\text{new } y \rangle \rightarrow E + \overline{xy} + y, \quad \text{se } y \notin! E$$

Entrada de dados Relembre que a entrada de dados tanto poderá ser usada para recepção de mensagens (comunicação assíncrona) como para sincronizações. Definimos o efeito de uma entrada de dados como sendo o de eliminação de uma mensagem pendente. Mais à frente veremos como a sincronização é definida. Observe que neste ponto torna-se relevante o fato de que alguns elementos de dados são referências de objetos. Isto implica que o efeito, neste caso, além de consumir a mensagem pendente também cria uma nova ligação:

$$\langle E + m_x^y, x:y?m \rangle \rightarrow E + \overline{xz}, \quad \text{se } m = \bar{z} (\forall z \in \mathcal{O})$$

e

$$\langle E + m_x^y, x:y?m \rangle \rightarrow E, \quad \text{se } m \neq \bar{z} \ (\forall z \in \mathcal{O})$$

Saída de dados Conceitualmente, toda saída produz uma mensagem pendente para o objeto destinatário do dado. Contudo, novamente devemos alertar para o caso em que o dado seja uma referência (observe que definimos igualmente os efeitos de saídas síncronas e assíncronas). Definimos:

$$\langle E, x:y.m \rangle \rightarrow E + m_y^x, \quad \text{se } m \neq \bar{z} \text{ e } \bar{x}\bar{y} \in E$$

$$\langle E, x:y.m \rangle \rightarrow E + m_y^x, \quad \text{se } m = \bar{z} \text{ e } \bar{x}\bar{y}, \bar{x}\bar{z} \in E$$

Desligamento Conceitualmente, um objeto pode se desligar de outro desde que esteja ligado. Logo, podemos definir:

$$\langle E + \bar{x}\bar{y}, x:\tilde{y} \rangle \rightarrow E, \quad \text{se } \bar{x}\bar{y} \notin E$$

Ações finais A auto-destruição, ou ação final, de um objeto faz com que ele seja eliminado da configuração. Naturalmente, esperamos que todas as ligações inválidas envolvendo o objeto a ser eliminado, também sejam eliminadas, a fim de garantir que a configuração alcançada seja bem-formada. Assim, definimos

$$\langle E, x:\text{end} \rangle \rightarrow E \ominus x$$

Ações compostas O efeito de uma ação composta é definido concatenando os efeitos das ações elementares de que formada:

$$\frac{\langle E, x:a_1 \rangle \rightarrow E' \quad \langle E', x:a_2 \rangle \rightarrow E''}{\langle E, x:a_1 \circ a_2 \rangle \rightarrow E''}$$

4.3.2 Eventos

Antes de definirmos *evento*, vamos estender a relação \rightarrow para caracterizar o efeito combinado de um conjunto de ações sobre uma estrutura. Partimos do efeito de um conjunto vazio de ações que, naturalmente, deve ser nulo:

$$\langle E, \emptyset \rangle \rightarrow E$$

A partir dessa base, definimos o efeito de um conjunto de ações como a combinação de seus efeitos isolados. Seja c um conjunto de ações de objetos diferentes de x :

$$\frac{\langle E, c \rangle \rightarrow E' \quad \langle E', x:a \rangle \rightarrow E''}{\langle E, c \cup \{x:a\} \rangle \rightarrow E''}, \quad \nexists x:a' \in c$$

Observe que as regras definidas apenas caracterizam a avaliação do efeito das ações sobre uma estrutura, sem levar em consideração que algumas ações requerem sincronização, isto é, a ocorrência simultânea de ações complementares em outros objetos. É para isto que introduzimos o conceito de *evento* como sendo de um conjunto *completo* de ações. Isto é, que satisfaz a todas as restrições de sincronização.

Definição 4.8 *Um conjunto de ações e é dito um evento se para toda ação elementar $x:y!m$ que ocorra em e , a ação elementar $y:x?m$ também pertence a e . Formalmente⁵:*

$$e \text{ é um evento} \stackrel{\text{def}}{\iff} \forall x, y, m: x:y!m \in e \Rightarrow y:x?m \in e$$

Sobre os estados internos

Também definimos o efeito de um evento sobre os estados internos σ de uma configuração. Analogamente à aplicação sobre estruturas, escrevemos $\langle \sigma, e \rangle \rightarrow \sigma'$. Definimos:

$$\frac{x_i \xrightarrow{a_1} x_j \quad \dots \quad z_i \xrightarrow{a_n} z_j}{\langle \sigma, \{x:a_1, \dots, z:a_n\} \rangle \rightarrow \sigma' \{x_j/x_i\} \{ \dots \} \{z_j/z_i\}}, \quad \text{se } x_i \in \sigma$$

A regra acima caracteriza o efeito de um evento sobre os estados internos dos objetos. A idéia é simples e intuitiva: consiste em trocar os estados dos objetos em que as ações ocorreram pelo estado determinado pelo seu comportamento. Observe que $\langle \sigma, e \rangle \rightarrow \sigma'$ só é definido se a cada ação $x:a \in e$ corresponde algum $x_i \xrightarrow{a} x_j$ no comportamento do objeto.

4.3.3 Ocorrência e alcançabilidade

Para caracterizar o comportamento de um sistema definimos duas relações sobre suas configurações, denominadas de ocorrência e alcançabilidade. A relação de ocorrência, escrita $C \vdash C'$, indica que o sistema pode passar da configuração C à configuração C' pela ocorrência de algum evento. A relação de *alcançabilidade* é definida como o fechamento reflexivo e

⁵Observe que dizemos que a ação elementar $x:a$ pertence ao evento e , escrevendo $x:a \in e$, mesmo que essa ação seja apenas uma das ações que compõem uma ação composta em e .

transitivo de \vdash e será escrita $C \vdash^* C'$ —dizemos que a configuração C' é alcançável a partir de C . Uma única regra é suficiente para caracterizar a relação. Seja e um evento composto de ações dos objetos pertencentes a E :

$$\frac{\langle E, e \rangle \rightarrow E' \quad \langle \sigma, e \rangle \rightarrow \sigma'}{\langle E, \sigma \rangle \vdash \langle E', \sigma' \rangle}, \quad \text{onde } x:a \in e \Rightarrow x \in E$$

A regra acima determina que um sistema na configuração $\langle E, \sigma \rangle$ pode passar à configuração $\langle E', \sigma' \rangle$ desde que: E' seja o resultado da aplicação de um evento e sobre E ; e que σ' consista em σ , efetuando as mudanças de estados referentes às ações dos objetos. Observe que subentendemos que toda ação em e é uma ação possível para algum dos objetos na estrutura E . Para evidenciar o evento envolvido em uma ocorrência rotulamos o símbolo da relação, escrevendo $E \vdash_e E'$.

4.3.4 Reduções

As regras apresentadas permitem avaliar o comportamento de um sistema de objetos. Considere, por exemplo, um sistema cuja configuração inicial seja

$$\langle E_0, \sigma_0 \rangle = a_0 + \bar{a}b + b_0.$$

Relembre que há duas informações aí contidas: que a estrutura inicial é $a + \bar{a}b + b$ e que os estados iniciais dos objetos a e b são a_0 e b_0 , respectivamente. Considere agora que o comportamento dos objetos dado por

$$a_0 \xrightarrow{b.m} a_1 \quad b_0 \xrightarrow{a.m} b_1.$$

O comportamento deste sistema é expresso pela seguinte seqüência de ocorrências:

$$\begin{aligned} a_0 + \bar{a}b + b_0 &\vdash a_1 + \bar{a}b + b_0 + m_b^a \\ &\vdash a_1 + \bar{a}b + b_1 \end{aligned}$$

É fácil verificar, pelas regras, que as ocorrências podem de fato ocorrer. Para isto podemos instanciar as regras, reduzindo a condição à possibilidade de derivar as premissas. A primeira ocorrência acima, por exemplo, só é possível se pudermos identificar o evento e da regra abaixo:

$$\frac{\langle a + \bar{a}b + b, e \rangle \rightarrow a + \bar{a}b + b + m_b^a \quad \langle \{a_0, b_0\}, e \rangle \rightarrow \{a_1, b_0\}}{a_0 + \bar{a}b + b_0 \vdash a_1 + \bar{a}b + b_0 + m_b^a}$$

caracterizar a semântica de redes de Petri em termos de objetos requereu a definição do modelo apresentado neste capítulo. Sua principal característica é a completa separação entre as atividades com efeitos internos e externos dos objetos. Apenas as ações com efeitos externos são parte efetiva da formalização apresentada. Observe que aspectos de fluxo de controle, por exemplo, não são diretamente tratados. Isto é relevante porque, em nossa abordagem, tais aspectos podem e devem ser especificamente modelados por redes de Petri. Esta separação permite escolher o nível de abstração de modelagem e atribuir comportamento concorrente aos objetos de um sistema ou decompô-lo em vários objetos seqüenciais.

Ora, pela segunda premissa, vemos que o evento deve consistir de uma ação do objeto a que o leve do estado a_0 para a_1 . O evento deve ser $e = \{a:b.m\}$, o que se encaixa perfeitamente bem, pois:

$$\langle a + \bar{a}b + b, \{a:b.m\} \rangle \rightarrow a + \bar{a}b + b + m_b^a \quad \text{e} \quad \frac{a_0 \xrightarrow{b.m} a_1}{\langle \{a_0, b_0\}, \{a:b.m\} \rangle \rightarrow \{a_1, b_0\}}$$

O lado esquerdo acima vem diretamente do axioma da saída assíncrona de dados. O lado direito é a instancia da regra que determina o efeito de um evento sobre os estados internos de um sistema.

4.4 Conclusões

O formalismo aqui apresentado tem como propósito servir de base para a formalização da semântica de uma notação orientada a objetos para redes de Petri. Observe que a separação entre os aspectos internos e externos dos objetos permite que a descrição do comportamento de um sistema seja dada através de qualquer outro formalismo que permita derivar o comportamento isolado dos objetos da forma em que o definimos. É importante observar que o formalismo é proposto para descrever diretamente os sistemas de objetos. Na prática, a notação baseada em redes de Petri e orientação a objetos é que terá seu significado expresso em termos dos sistemas de objetos.

Outros modelos semânticos para linguagens orientadas a objetos concorrentes existem. Um dos trabalhos mais antigos e influentes como modelos de concorrência é o modelo de *Actors* [Agh86]. Outro trabalho influente é o relacionado à linguagem POOL (*Parallel Object-Oriented Language*), cuja semântica operacional foi caracterizada formalmente usando sistemas de transição rotulados (*LTSs*) [AdBKR86]. A semântica denotacional de POOL também foi definida usando o conceito de espaços métricos [AdBKR92]. Outro modelo utilizado para caracterizar a semântica de linguagens concorrentes OO é a chamada *lógica de reescrita* (*rewriting logic*) [MOM93], originalmente proposta por Meseguer—um exemplo de sua utilização pode ser encontrado em [Mes90] na caracterização de Maude. Finalmente, os trabalhos mais difundidos são os baseados em álgebras e cálculos de processos. Dentre os mais influentes destacamos CSP [Hoa78; Hoa85] e π -cálculo [Mil93].

Apesar dos vários modelos semânticos disponíveis, as peculiaridades do problema de

Capítulo 5

Redes de Petri Orientadas a Objetos

Neste capítulo, enfocamos os conceitos do nível de modelo de linguagem, definidos no Capítulo 3. Definimos os elementos de uma notação para a modelagem de sistemas de objetos, como caracterizados no modelo conceitual apresentado no Capítulo 3, usando redes de Petri como a linguagem básica para a descrição de comportamento. Inicialmente, formalizamos os conceitos de *classe* e *modelo*, caracterizando as partes de que são compostos. A idéia é apresentar os elementos sintáticos fundamentais usados para descrever sistemas de objetos. Em seguida, definimos o *significado* dessas classes e modelos, determinando os conjuntos de objetos e de sistemas de objetos por eles descritos. Logo, definimos a semântica da notação, usando sistemas de objetos como definidos no capítulo anterior, proporcionando uma composição a nível semântico de redes de Petri e da orientação a objetos. Finalmente, apresentamos uma versão concretizada da notação, que pode ser usada em contextos práticos de modelagem. Nessa versão, construções mais adequadas ao processo de modelagem são introduzidas, bem como algumas facilidades de notação. Um exemplo simples de modelagem, o problema clássico do jantar dos filósofos, é usado para introduzir a notação concreta e demonstrar seu uso.

5.1 Modelos e sintaxe abstrata

Nosso objetivo, nesta seção, é caracterizar formalmente os elementos que compõem um modelo na notação apresentada. É importante observar que as definições aqui apresentadas não determinam uma sintaxe concreta, nem mesmo todas as construções sintáticas efetivamente usadas para elaborar modelos na prática. Apenas identificamos e definimos

um número mínimo de elementos necessários para caracterizar modelos.

5.1.1 Definições básicas

Classes: visões sintática e semântica Um sistema de objetos é caracterizado pelos objetos que o compõem. Logo, uma forma natural de descrevê-lo é modelando seus objetos. A maioria das linguagens orientadas a objetos baseia a modelagem dos objetos no conceito de *classes*. A idéia parte da observação de que é possível *classificar* os objetos de um sistema de acordo com suas características em comum. Classes, portanto, podem ser definidas como *conjuntos de objetos que compartilham as mesmas propriedades*. Neste sentido, o termo pode ser considerado sinônimo de *tipo* (de objetos). É comum dizermos, por exemplo, que “dois objetos são do mesmo tipo se pertencem à mesma classe”.

Contudo, do ponto de vista da construção de modelos e da notação usada para escrevê-las, também chamamos de classe à descrição, propriamente dita, dos objetos. Neste sentido, definimos classe como *uma construção de linguagem usada para descrever um conjunto de objetos do mesmo tipo*. É importante perceber a sutil relação entre os dois conceitos de classe. O primeiro é semântico. Trata de conjuntos de objetos que só existem nos próprios sistemas de objetos. O segundo é sintático. Faz sentido no mundo da descrição dos sistemas de objetos.

Notação Para facilitar a distinção entre os conceitos usados em cada situação, adotamos a seguinte notação e respectivas convenções tipográficas. Em primeiro lugar, os objetos de um sistema são denotados por

$$o_1, o_2, \dots, o_i, \dots$$

A quantidade de objetos em um sistema pode ser arbitrariamente grande—potencialmente infinita, mas é geralmente irrelevante do ponto de vista de modelagem¹. Para garantir modelos finitos, assumimos que é possível classificar os objetos do sistema em um número finito de classes, digamos n , denotadas por

$$C_1, C_2, \dots, C_n.$$

¹O número de objetos em cada configuração do sistema é finito. Entretanto, o total de objetos de um sistema é determinado pela união dos objetos de todas as configurações alcançáveis. Logo, é potencialmente infinito.

Identificadores Conceitualmente, objetos e identificadores são biunivocamente relacionados. Para evidenciar esta relação, denotamos o identificador do objeto o_i por \vec{o}_i . Também assumimos que os identificadores dos objetos da classe C_i pertencem ao conjunto N_i (de nomes). Assumimos que os conjuntos N_i são disjuntos³ e potencialmente infinitos. Devido às condições adotadas, podemos identificar o tipo de um objeto pelo tipo de seu identificador, ou seja,

$$o_i \in C_j \iff \vec{o}_i \in N_j. \quad (5.1)$$

Também definimos o conjunto universo de identificadores, denotado por N , como a união dos conjuntos N_i . Ou seja, se há n classes no modelo, então $N = N_1 \cup N_2 \cup \dots \cup N_n$. Conceitualmente, o conjunto universo de identificadores N corresponde à classe genérica O .

Tipos de dados e Sortes Os objetos de um sistema armazenam e processam dados. Logo, parte relevante da modelagem consiste em descrever os tipos de dados que devem ser manipulados pelos objetos. A modelagem de dados que adotamos consiste em utilizar uma Σ -álgebra $\langle \mathcal{D}, \mathcal{D} \rangle$ —onde \mathcal{D} é um conjunto de domínios (ou de conjuntos de cores, no jargão de redes de Petri coloridas) e \mathcal{D} o respectivo conjunto de operações (sobre especificações algébricas, ver [EM85]).

Os domínios em \mathcal{D} pertencem a dois grupos: os tipos de dados específicos do problema e os tipos formados a partir dos identificadores de objetos. Os domínios correspondentes aos tipos do problema, denotados por D_1, D_2 , etc, caracterizam os dados “de interesse” que devem ser tratados pelo sistema. Os tipos de identificadores são os conjuntos N_i e o conjunto N , definidos anteriormente. O propósito de incluí-los é tornar os identificadores de objetos em dados de primeira classe. Isto permite que objetos armazenem e processem identificadores como outros tipos de dados (relembre que um objeto só pode interagir com outro, se “conhece” seu identificador). Logo, se em um sistema há m tipos específicos de dados e n classes, os domínios em \mathcal{D} são:

$$\mathcal{D} = \{D_1, D_2, \dots, D_m, N_1, N_2, \dots, N_n, N\}.$$

³Para permitirmos a introdução de hierarquias de tipos, devemos relaxar um pouco esta restrição. A disjunção não deve ser requerida para conjuntos de identificadores de classes de uma mesma linha de hierarquia. Por ora, partimos do pressuposto de que há apenas um nível na hierarquia de tipos.

Novamente, é preciso separar os conceitos semânticos daqueles que efetivamente podemos usar na notação. Para anotar o modelo, usamos *sortes*. Isto é, os nomes dos domínios, e não os domínios propriamente ditos. Formalmente, isto corresponde a usar a assinatura Σ da álgebra acima, descrita por:

$$\Sigma = \langle \mathcal{S}, \mathcal{A} \rangle$$

$$\mathcal{S} = \{S_1, S_2, \dots, S_m, C_1, C_2, \dots, C_n, \mathcal{O}\}$$

O conjunto \mathcal{S} consiste em uma sorte para cada um dos domínios em \mathcal{D} . Seus $m + n + 1$ elementos correspondem às sortes (ou nomes) dos m tipos específicos, das n classes de objetos e da classe genérica. Para os tipos específicos do sistema, adotamos as sortes S_i . Por convenção, o índice usado associa cada sorte ao respectivo domínio—ou seja S_i é a sorte de D_i . Para os domínios de identificadores N_i , adotamos os respectivos nomes de classes C_i como sortes. Entretanto, usamos S_i como representante de um elemento qualquer de \mathcal{S} (analogamente, usamos D_i para representar um elemento qualquer de \mathcal{D}).

O uso de sortes permite anotar o modelo com informações sobre os tipos de dados esperados em cada situação. Como identificadores são tratados como tipos de dados, também é possível indicar o tipo de identificador esperado em qualquer situação. Indiretamente, devido a (5.1), isto equivale a indicar a classe a que um objeto referenciado pertence. Ainda assim, em algumas situações é interessante não indicar uma classe. É nessas situações que se justifica o uso da sorte \mathcal{O} .

Finalmente, \mathcal{A} é um conjunto de símbolos operacionais para as operações em \mathcal{D} , com suas respectivas aridades, definidas da forma convencional. Observe que o projetista é livre para definir novos tipos compostos, usando os tipos identificadores. Contudo, não há operações básicas disponíveis para esses tipos. Ainda assim, é permitido ao projetista que defina operações nulárias—isto equivale a definir constantes para identificadores.

Variáveis, termos e equações Além da assinatura, usamos os seguintes conceitos convencionais de especificações algébricas, derivados de Σ : \mathcal{V} e \mathcal{T} denotam, respectivamente, um conjunto de variáveis para Σ , e o conjunto de termos derivados. Ambos são formados pela união disjunta de famílias de variáveis e de termos, indexadas por sortes de Σ . Ou

seja, definimos:

$$\mathcal{V} = \mathcal{V}_{S_1} \cup \dots \cup \mathcal{V}_{S_n} \cup \underbrace{\mathcal{V}_{C_1} \cup \dots \cup \mathcal{V}_{C_n} \cup \mathcal{V}_O}_{\text{variáveis-objeto}}$$

$$\mathcal{T} = \mathcal{T}_{S_1} \cup \dots \cup \mathcal{T}_{S_n} \cup \underbrace{\mathcal{T}_{C_1} \cup \dots \cup \mathcal{T}_{C_n} \cup \mathcal{T}_O}_{\text{termos-objeto}}$$

Dizemos que a variável “ v é do tipo S_i ” se $v \in \mathcal{V}_{S_i}$. Formalmente, isto significa que só valores do domínio adequado podem ser atribuídos a v . Ou seja, se $v \in \mathcal{V}_{S_i}$ e denotamos uma atribuição por $v \mapsto d$, então $d \in D_i$. Analogamente, dizemos que o termo “ t é do tipo S_i ” se $t \in \mathcal{T}_{S_i}$. Formalmente, isto significa que $\llbracket t \rrbracket \in D_i$, onde $\llbracket t \rrbracket$ é a avaliação de t . Dizemos que v é uma variável-objeto se $v \in \mathcal{V}_{C_i}$, para alguma classe C_i . Analogamente, termos $t \in \mathcal{T}_{C_i}$ são chamados de *termos-objeto*.

Também usamos o conceito de equações como pares de termos escritos na forma “ $t_1 = t_2$ ” e sua interpretação convencional. Ao conjunto das equações definidas sobre Σ denotamos \mathcal{E} .

Multi-conjuntos como tipos Estendemos o conceito de “tipo de um termo” para multi-conjuntos de termos. Se m é um multi-conjunto de termos (ver definição de multi-conjuntos na página 49), dizemos que “o tipo de m é S_i ” se $m \in \mathcal{T}_{S_i}^{ms}$.

5.1.2 Classes

Cada classe de um modelo consiste em três partes: *nome*, *corpo* e *inscrições de interação*. O *nome* de uma classe pode ser escolhido arbitrariamente, desde que seja único no modelo. O *corpo* de uma classe é a descrição principal do funcionamento de seus objetos. Em nossa abordagem, usamos redes de Petri com esse propósito. Logo, identificamos o estado interno dos objetos ao conceito de estado de redes, e suas ações aos eventos da rede. Finalmente, as *inscrições de interação* complementam a descrição da classe, indicando condições e efeitos externos ao objeto. Em nossa abordagem, usamos as inscrições de interação nos elementos do corpo da classe.

Antes de definirmos o corpo de uma classe e suas inscrições de interação, definimos os elementos estruturais de redes de Petri. Como já dissemos, adotamos diretamente a definição de redes como o *corpo* de uma classe. Daí porque as definições comportamentais são deixadas para a seção de caracterização da semântica da notação.

Redes de Petri Aqui definimos *estruturas de redes de Petri* e em seguida *redes de Petri (Coloridas)*. Há pouca diferença entre a definição apresentada e a adotada por K. Jensen em [Jen92]. Como nosso propósito é usar as redes como linguagem para descrever o comportamento dos objetos, enfatizamos o uso de conceitos sintáticos na definição. Ao invés de mapearmos lugares diretamente aos domínios de dados, por exemplo, os mapeamos às sortes. Além disso, as expressões são explicitamente tomadas como os termos da álgebra.

Definição 5.1 *Uma estrutura de rede de Petri é uma tupla $\langle P, T, F \rangle$, em que P é um conjunto finito de lugares, T é um conjunto finito de transições e $F \subseteq (P \times T) \cup (T \times P)$ é um conjunto de arcos, com P e T disjuntos.*

A definição a seguir vincula a uma estrutura de rede de Petri todos os tipos de inscrições que caracterizam uma rede Colorida: declarações de tipos dos lugares, expressões de arcos, guardas nas transições e expressões de inicialização de marcação. As expressões são construídas a partir dos termos da assinatura Σ .

Definição 5.2 *Sejam uma assinatura Σ , uma estrutura de rede de Petri $N = \langle P, T, F \rangle$ e funções $C : P \rightarrow \mathcal{S}$, $G : T \rightarrow 2^{\mathcal{E}}$, $E : F \rightarrow \mathcal{T}^{ms}$, e $I : P \rightarrow \mathcal{T}^{ms}$ denominadas, respectivamente, de função de cores, de guardas, de expressões de arcos, e de inicialização. Dizemos que $\langle N, C, G, E, I \rangle$ é uma rede de Petri colorida definida sobre Σ desde que:*

$$\forall f \in F : E(f) \text{ seja do tipo } C(p), \text{ onde } f = \langle p, t \rangle \text{ ou } f = \langle t, p \rangle \text{ e}$$

$$\forall p \in P : I(p) \text{ é uma expressão fechada (não tem variáveis) e é do tipo } C(p).$$

A função de cores mapeia os lugares da rede a sortes. Indiretamente, determina o tipo de dados correspondente a cada lugar. A função de guardas mapeia transições às equações que devem ser satisfeitas para garantir a habilitação. A função de expressão de arcos permite indicar o “peso” de cada arco. Finalmente, a função de inicialização associa uma expressão fechada a cada lugar que determina sua marcação inicial.

Inscrições de interação As possibilidades de interação determinadas pelo modelo conceitual escolhido são: entradas de dados, saídas de dados (síncronas e assíncronas), instanciação de objetos e eliminação de ligações (relembre que a criação de ligações ocorre “automaticamente” sempre que um identificador é recebido). Há ainda a ação final ou auto-

destruição que, embora não seja uma interação propriamente dita, tem influência direta sobre o estado global do sistema.

As *inscrições de interação* permitem determinar como um objeto interage com outros objetos. Formalmente, consistem em descrições de interação, vinculadas ao corpo de uma classe. Como o corpo de uma classe é uma rede de Petri, as inscrições são vinculadas diretamente aos elementos da rede. As definições seguintes formalizam o conceito.

Definição 5.3 *Sejam t um termo, s um termo-objeto, \mathbb{C} um nome de classe e v uma variável-objeto. A seguinte sintaxe abstrata caracteriza as inscrições de interação*

$$I ::= s.t \mid s!t \mid s?t \mid v:\mathbb{C} \mid \tilde{s} \mid \text{end}$$

$$R ::= \epsilon \mid I \circ R$$

O conjunto de todas as inscrições de interação é denotado por R .

A categoria sintática I define os seis tipos de inscrições de interação. Inscrições nas formas “ $s.t$ ”, “ $s!t$ ” e “ $s?t$ ” são chamadas, respectivamente, de *saídas assíncronas de dados* (ou *emissões de mensagens*), *saídas síncronas de dados* e *entradas de dados*. Em todos os casos, s e t são termos que, ao serem avaliados, determinam, respectivamente, o identificador do objeto e os dados envolvidos na interação. Inscrições na forma “ $v : \mathbb{C}$ ” são chamadas de *instanciações*. A variável-objeto v é ligada ao identificador do objeto instanciado, logo, deve ser do tipo \mathbb{C} . Inscrições da forma “ \tilde{s} ” são chamadas de *desligamentos*. O termo-objeto s determina o identificador do objeto cuja ligação deve ser eliminada. E, finalmente, “end” é chamada de inscrição de *finalização*.

A categoria sintática R define as *inscrições de interação* propriamente ditas. Cada inscrição consiste em uma seqüência de inscrições de interação. Resta apenas vincular inscrições ao corpo de uma classe. A definição a seguir formaliza o conceito sintático de classe, compondo os elementos descritos até aqui.

Definição 5.4 *Seja \mathbb{C}_i um nome de classe, K uma rede de Petri e $I : T \rightarrow R$ uma função que mapeia transições a inscrições de interação. Dizemos que $\langle \mathbb{C}_i, K, I \rangle$ é uma classe de nome \mathbb{C}_i e corpo K . Em geral, usamos \mathbb{C}_i como representativo da classe, subentendendo corpo e inscrições de interação.*

Observe que a função I mapeia transições a inscrições de interação da categoria sintática R . A inscrição vazia ($i = \epsilon$) representa formalmente a ausência de inscrição de interação na

transição. Dizemos que transições sem inscrições de interação, tais que $I(t) = \epsilon$, modelam ações internas.

Associações Embora não seja evidente, a definição acima permite estabelecer associações entre classes. Observe que para estabelecer uma associação direta entre as classes \mathbb{C}_i e \mathbb{C}_j , basta que os objetos de uma possam armazenar identificadores dos objetos da outra. Para isto, é suficiente que o tipo de algum lugar do corpo da primeira seja de identificadores da segunda. Por exemplo, se o lugar p do corpo de \mathbb{C}_i é do tipo \mathbb{C}_j , ou seja, se $C(p) = \mathbb{C}_j$, então há uma associação direta entre as duas classes—neste caso, dizemos que a associação é navegável a partir de \mathbb{C}_i .

Na prática, esta forma de declarar associações não é a mais adequada. Na versão concreta da notação, associações são especificadas através das construções convencionais da orientação a objetos. Contudo, é requerido que as classes realizem as associações corretamente (ver Seção 5.3.2 na página 107).

Classes referenciadas A observação acima evidencia que, em geral, uma classe depende de outras. Mais precisamente: que o funcionamento dos objetos de uma classe depende da existência de objetos de outras classes. Para indicar essa dependência entre as classes, introduzimos a seguinte notação: escrevemos $\text{clsref}(\mathbb{C}_i)$ para denotar o conjunto das classes referenciadas no corpo de \mathbb{C}_i . Formalmente, o conjunto é definido por

$$\text{clsref}(\mathbb{C}_i) = \{ \mathbb{C}_j \mid C(p) = \mathbb{C}_j \quad \forall p \in K \}, \quad (5.2)$$

onde K é o corpo de \mathbb{C}_i .

Esta definição específica torna-se inadequada se os identificadores puderem ser armazenados em estruturas de dados mais complexas. Por exemplo, o tipo efetivo de um lugar pode consistir em um par ordenado em que um dos elementos é um identificador. Neste caso, o tipo do lugar não é explicitamente \mathbb{C}_j . Na modelagem de dados apresentada, contudo, não definimos os mecanismos disponíveis para definir os tipos de dados. Logo, em tais situações, não há elementos suficientes para definir $\text{clsref}(\mathbb{C}_i)$ com precisão. A especificação, contudo, é suficientemente clara para que possamos usar a notação sem qualquer preocupação: $\text{clsref}(\mathbb{C}_i)$ contém *todas* as classes que de alguma forma são referenciadas em \mathbb{C}_i .

5.1.3 Modelos

O modelo de um sistema é composto por duas partes relativamente independentes: a *modelagem de dados* e a *modelagem de objetos*. Nesta seção, focalizamos nossa atenção na modelagem de objetos. O modelo de dados, considerado “pronto”, é representado simplesmente por Σ .

Definição 5.5 *Um modelo (de objetos) é um conjunto não vazio de classes.*

A definição nos permite ver cada classe como um modelo. Dizemos que “a classe C_i modela os objetos da classe C_i ” ou mais diretamente que “ C_i modela a classe C_i ”. Neste caso, dizemos que objetos de C_i são *instâncias* de C_i .

Observe, contudo, que se a classe depende de outras classes, ou seja, se $\text{clsref}(C_i) \neq \emptyset$, então é necessário complementar o modelo com as classes referenciadas para obter um modelo completo. As definições seguintes formalizam a idéia.

Definição 5.6 *Seja M um modelo. O conjunto das classes referenciadas em M , denotado por $\text{clsref}(M)$, é a união dos $\text{clsref}(C_i)$ de cada classe C_i pertencente a M , ou seja,*

$$\text{clsref}(M) = \bigcup_{C_i \in M} \text{clsref}(C_i).$$

Definição 5.7 *Um modelo M é dito completo se contém todas as classes nele referenciadas, ou seja, se $\text{clsref}(M) \subseteq M$. Caso contrário, M é dito incompleto.*

Podemos concluir que a modelagem de objetos de um sistema consiste em construir um *modelo completo* de seus objetos, de acordo com as definições anteriores.

5.1.4 Comentários

Observe que usamos os nomes das classes de objetos como nomes dos respectivos domínios de identificadores. Isto difere do que normalmente esperamos em relação ao uso do nome de um tipo. Se declaramos uma variável v como sendo do tipo S_i , esperamos que seus possíveis valores sejam elementos do respectivo domínio, isto é, de D_i . Neste caso, porém, ao declararmos uma variável do tipo C_i , seus possíveis valores não são objetos da classe C_i . São elementos de N_i , identificadores de objetos da classe C_i .

É importante observar que, dos conceitos introduzidos apenas os nomes de classe C_i podem ser efetivamente usados nas descrições. As classes C_i e os objetos propriamente ditos

são entidades do mundo semântico, ou seja, são parte do significado, dos modelos e não podem fazer parte da descrição propriamente dita. Eles são introduzidos aqui como parte da meta-linguagem e, mais à frente, serão usados para definir o significado das descrições.

Agregação e especialização Outros dois mecanismos relevantes da orientação a objetos são a *agregação* e a *especialização*. Do ponto de vista formal, consideramos agregação uma variação da associação.

Considero estes mecanismos como essencialmente sintáticos. E, formalmente, podemos pensar neles como operações sobre classes de objetos que geram novas classes. Desta forma, podemos considerar um pré-passo antes de interpretar o modelo: gerar todas as classes concretas necessárias para instanciar os objetos do modelo. Há ainda a questão da coexistências de objetos de um tipo e de um subtipo. Neste caso, ambas as classes são concretizadas, e caso a relação deva ser percebida devemos relaxar a exigência de disjunção entre os conjuntos de nomes.

Podemos ver a declaração das associações como restrições sobre os modelos. Por exemplo, se uma associação $a : C_i[s] \times C_j[t]$ for declarada, devemos entender que se trata de um compromisso assumido de que a classe C_i pode usar a variável t para referir-se a objetos da classe C_j dentro de seu corpo e vice-versa. Isto tem mais objetivo prático do que de caracterização dos elementos de um modelo.

5.2 Significado dos modelos

Semântica Interpretar uma descrição é apreender, capturar, identificar seu *significado*. E o significado de uma descrição é a própria entidade descrita. Adotamos o conceito de classe com o propósito de descrever os objetos que compõem *sistemas de objetos* (como definidos no Capítulo 4). Logo, podemos dizer que *o significado de uma classe é um conjunto de objetos*, e que interpretá-la significa determiná-los. De forma análoga, *o significado de um modelo é o conjunto de sistemas de objetos que descreve*.

Embora as observações acima pareçam óbvias, é tomando-as literalmente que definimos a *semântica* de classes e modelos. Ou seja, formalmente, a semântica de uma classe C_i é definida como um conjunto de objetos, que denotamos por $\llbracket C_i \rrbracket$. E o problema de

determiná-la consiste em derivar os objetos desse conjunto a partir da descrição da classe⁴.

Recorde que, no capítulo anterior, caracterizamos o comportamento dos objetos através da relação \rightarrow definida sobre seus possíveis estados. Os elementos da relação, chamados *ações potenciais*, têm a forma $x_i \xrightarrow{a} x_j$, onde a é a ação executada pelo objeto para passar do estado x_i ao estado x_j . Essa representação foi adotada por sua relação direta com a representação do comportamento de redes de Petri. Ou seja, a escolha tem o propósito de tornar simples a interpretação das redes de Petri como o comportamento dos objetos.

5.2.1 Comportamento de redes de Petri

O conceito de estado usado em redes de Petri é denominado *marcação*. Informalmente, marcações são distribuições de *fichas* pelos lugares da rede. Cada ficha é um elemento do *conjunto de cores* (domínio de dados) associado ao lugar a que pertence. Mais adiante, formalizamos a “adoção” de marcações como estados de objetos.

Definição 5.8 *Uma ficha é um par $\langle p, d \rangle \in P \times D_i$, tal que $C(p) = S_i$. Denotamos por F o conjunto das fichas de uma rede. Uma marcação é um multi-conjunto sobre F . O conjunto das marcações de uma rede é denotado por M .*

A representação de marcações como multi-conjuntos permite certa flexibilidade na notação. Uma variação interessante é interpretar cada marcação m como uma função sobre os lugares da rede. Neste caso, $m(p)$ é a marcação do lugar p . Portanto, se $m(p) \in D_i^{m_s}$, então $C(p) = D_i$.

A marcação inicial $m_0 \in M$ das redes (que modela o estado inicial dos objetos da classe) é determinada pela função de inicialização I . A marcação inicial de cada lugar é determinada avaliando as respectivas expressões de inicialização. Seja $\llbracket e \rrbracket$ o resultado da avaliação da expressão e , então a marcação inicial de uma rede é definida por

$$m_0(p) = \llbracket I(p) \rrbracket \quad \text{para cada } p \in P. \quad (5.3)$$

Disparos A noção básica de comportamento de redes de Petri é a *ocorrência* (ou o *disparo*) de uma transição. As condições em que uma transição pode disparar são determinadas

⁴No início da seção anterior, introduzimos C_i para descrever o conjunto de objetos denotado por C_i . Logo, $\llbracket C_i \rrbracket = C_i$.

pelas expressões de seus arcos de entrada e pelas condições especificadas nas guardas. Além disso, as expressões dos arcos (tanto os de entrada como os de saída) também determinam as fichas a eliminar ou adicionar aos lugares caso a transição ocorra. Ou seja, todos os detalhes de uma ocorrência dependem da avaliação das diversas expressões (de termos) usadas. Portanto, obtemos diferentes *modos* de ocorrência da transição se escolhermos diferentes atribuições para as variáveis envolvidas. A definição seguinte formaliza o conceito⁵.

Definição 5.9 *Um modo de uma transição t é uma atribuição de valores para as variáveis de \mathcal{V} . Se denotamos a atribuição por a , então escrevemos t^a para denotar a transição t no modo a . (Sempre que for conveniente e possível, subentendemos a e escrevemos apenas t .)*

Esta definição nos permite simplificar a linguagem e a notação sem introduzir ambigüidades. A idéia é subentender o modo e enfatizar a transição. Nas definições seguintes, por exemplo, definimos a habilitação e a ocorrência de uma transição t no modo a . A partir daí, freqüentemente falamos da habilitação e da ocorrência de uma transição t sem mencionar o modo explicitamente.

Definição 5.10 *Sejam t uma transição, m uma marcação e a um modo de t . Dizemos que t^a está habilitada na marcação m , e denotamos isso por $m[t^a]$, se $\llbracket e \rrbracket_a$ é verdade para toda equação $e \in G(t)$ e se*

$$\llbracket E(p, t) \rrbracket_a \leq m(p) \quad \text{para todo } p \in P.$$

A definição estabelece duas condições para a habilitação de t^a . Primeiro, toda equação usada como guarda, ou seja, todo $e \in G(t)$, tem que ser avaliada como verdadeira (considerando a atribuição a). E, segundo, a marcação de cada lugar de entrada tem que ter mais fichas do que o resultado da avaliação da expressão do arco que o liga à transição. De posse da condição de habilitação, definimos a marcação alcançada após a ocorrência da transição.

Definição 5.11 *Sejam t^a e m tais que $m[t^a]$. Escrevemos $m[t^a]m'$ para denotar que m' é a marcação alcançada após a ocorrência de t^a na marcação m . Definimos m' por*

$$m'(p) = m(p) - \llbracket E(p, t) \rrbracket_a + \llbracket E(t, p) \rrbracket_a \quad \text{para todo } p \in P. \quad (5.4)$$

⁵Em [Jen92], o conceito é denominado *binding element*. Preferimos o termo *modo* como é usado em outras redes de alto nível.

Se $m[t^a]m'$, dizemos que m' é diretamente alcançável a partir de m . Chamamos cada $m[t^a]m'$ de disparo ou de ocorrência.

Observe que a marcação de cada lugar após a ocorrência de t^a , determinada por (5.4), consiste em remover as fichas indicadas pelos arcos de saída do lugar e adicionar as indicadas pelos arcos de entrada à marcação original do lugar. Portanto, assumimos que $[[E(p, t)]_a = 0$ e que $[[E(t, p)]_a = 0$ caso não existam os arcos $\langle p, t \rangle$ e $\langle t, p \rangle$, respectivamente. Conseqüentemente, $m'(p) = m(p)$ para todo lugar p desligado de t .

É conveniente estender a notação de disparos para seqüências de disparos. Para isto, denotamos por $m[s]m'$, onde $s = t_1 t_2 \dots t_n$, que a seqüência de transições s é disparável a partir da marcação m e que, caso ocorra, leva à marcação m' . Ou seja, $m[s]m'$ se existem marcações intermediárias m_1, m_2, \dots, m_{n-1} , tais que

$$m[t_1]m_1, \quad m_1[t_2]m_2, \quad \dots, \quad m_{n-2}[t_{n-1}]m_{n-1}, \quad m_{n-1}[t_n]m'$$

Isto motiva a definição da noção de alcançabilidade como o fechamento reflexivo e transitivo sobre a alcançabilidade direta, definida acima.

Definição 5.12 Dizemos que a marcação m' é alcançável a partir de m se $m = m'$ ou se existe uma seqüência s não vazia de transições tais que $m[s]m'$. Ao conjunto de todas as marcações alcançáveis a partir de m denotamos por $[m]$. Se R é uma rede com marcação inicial m_0 , então escrevemos $[R]$ para denotar $[m_0]$.

5.2.2 Semântica de classes

Não é difícil perceber a semelhança entre as representações de disparos de transições de redes de Petri e de ações potenciais dos objetos. Ambas relacionam dois estados pela ocorrência de um evento. De fato, as ações potenciais foram intencionalmente definidas dessa forma para simplificar a interpretação de redes de Petri como modelos do comportamento de objetos (ver Seção 4.2.1 na página 65). Logo, caracterizamos os objetos, tomando marcações como representações de estados e determinando as ações potenciais a partir de disparos.

Definição 5.13 Seja uma classe C_i com corpo K . Os estados dos objetos da classe C_i são marcações alcançáveis $m \in [K]$ do corpo da classe.

Observe que devido à forma de modelar associações que adotamos (ver comentário na página 87), cada marcação determina tanto o estado interno do objeto quanto suas respectivas ligações. De fato, cada ligação é um identificador armazenado pelo objeto, portanto, é parte de seu estado. Contudo, é conveniente, em muitas situações, considerar apenas as ligações e abstrair o (resto do) estado interno do objeto. Para isto, usamos a notação $\text{objref}(o_i, m)$ para indicar as ligações do objeto o_i na marcação m .

As ações potenciais correspondentes a cada transição do corpo de uma classe são expressas, no modelo, através das inscrições de interação. Logo, para identificar as ações, devemos definir como as inscrições devem ser interpretadas.

Definição 5.14 *Seja i uma inscrição de interação. Dizemos que $\llbracket i \rrbracket_a$ é a ação resultante da avaliação de i sob a atribuição de variáveis a , definida recursivamente por*

$$\llbracket i \rrbracket_a = \begin{cases} \tau, & \text{se } i = \epsilon \\ \text{new} \llbracket v \rrbracket_a, & \text{se } i = v:\mathbb{C} \\ \text{end}, & \text{se } i = \text{end} \\ \widetilde{\llbracket s \rrbracket}_a, & \text{se } i = \tilde{s} \\ \llbracket s \rrbracket_a \cdot \llbracket t \rrbracket_a, & \text{se } i = s.t \\ \llbracket s \rrbracket_a ! \llbracket t \rrbracket_a, & \text{se } i = s!t \\ \llbracket s \rrbracket_a ? \llbracket t \rrbracket_a, & \text{se } i = s?t \\ \llbracket i_1 \rrbracket_a \circ \llbracket i_2 \rrbracket_a, & \text{se } i = i_1 \circ i_2 \end{cases}$$

onde $\llbracket s \rrbracket_a$ e $\llbracket t \rrbracket_a$ são as avaliações (convencionais) dos termos s e t que compõem a inscrição i . No último caso, i é formado por duas inscrições i_1 e i_2 .

Esta definição é a interpretação direta, ou seja, orientada pela sintaxe, das inscrições de interação. Observe que cada caso mapeia uma das formas de inscrição a uma ação (ações são definidas por (4.5) na página 65 e inscrições na Definição 5.3 na página 86). Os termos que compõem as inscrições são avaliados para produzir um elemento de dado ou um identificador de objeto correspondente à ação. Na definição seguinte, usamos a interpretação de inscrições para mapear disparos em ações potenciais e, em seguida, definimos o significado de uma classe.

Definição 5.15 *Seja uma classe \mathbb{C}_i com corpo K . O comportamento de seus objetos, denotado pela relação \rightarrow_i , consiste no conjunto de ações potenciais definidas por*

$$m \xrightarrow{U(t)_a} m' \quad \text{para cada disparo } m[t_a]m' \text{ em } K.$$

Definição 5.16 *O significado de uma classe \mathbb{C}_i com corpo K é o conjunto dos objetos com estados definidos por $[K]$ e comportamento determinado por \rightarrow_i , ou seja,*

$$[[\mathbb{C}_i]] = \{ o_j \mid o_j \text{ tem estados } [K] \text{ e comportamento } \rightarrow_i \}.$$

5.2.3 Semântica de modelos

Conceitualmente, modelos descrevem sistemas de objetos. Portanto, seria natural definir o significado de um modelo como um conjunto de sistemas de objetos. Contudo, é mais simples representar um sistema de objetos por sua configuração inicial. Logo, formalmente, é mais conveniente definir o significado de um modelo como um conjunto de configurações—onde cada uma determina um sistema de objetos.

Configurações Relembre que configurações são representações instantâneas de sistemas de objetos. A estrutura de uma configuração determina os objetos vivos, suas ligações e eventuais mensagens pendentes. Além disso, cada configuração também determina o comportamento dos objetos, bem como seus possíveis estados e o estado específico em que cada um deles se encontra (ver Definição 4.7 na página 68).

Relembre também que configurações são construídas a partir de um conjunto de objetos e de um conjunto de dados. Naturalmente, para definir as configurações correspondentes a um modelo, os objetos devem ser instâncias das classes e os tipos de dados devem ser elementos dos domínios que fazem parte do modelo. O comportamento \rightarrow e os estados possíveis dos objetos são indiretamente definidos pelas classes a que pertencem, como definidos na seção anterior. Resta apenas representar o estado instantâneo em que cada objeto se encontra na configuração, que é dado por σ , de tal modo que $\sigma(o_i) \in [K_j]$, onde K_j é o corpo da classe \mathbb{C}_j à qual o_i pertence.

Definição 5.17 *Seja $C = \langle E, \sigma \rangle$ uma configuração. Dizemos que C é uma instância de M se:*

1. todo objeto $o_i \in E$ é instância de alguma classe de M , ou seja,

$$\forall o_i \in E : o_i \in \llbracket \mathbb{C}_j \rrbracket \text{ para algum } \mathbb{C}_j \in M$$

2. toda ligação $\overline{o_i o_j} \in E$ é consistente com o estado do objeto de origem, ou seja,

$$\forall \overline{o_i o_j} \in E : o_j \in \text{objref}(o_i, \sigma(o_i)).$$

3. toda mensagem $d_{o_j}^{o_i} \in E$ é de um domínio do modelo, ou seja,

$$\forall m_{o_j}^{o_i} \in E : m \in D_k \text{ para algum } D_k \in \mathcal{D}$$

4. e σ mapeia cada objeto a um de seus estados válidos, ou seja,

$$o_i \in \llbracket \mathbb{C}_j \rrbracket \Rightarrow \sigma(o_i) \in [K_j], \quad \text{onde } K_j \text{ é o corpo de } \mathbb{C}_j$$

Se C é instância de M , então também dizemos que o sistema de objetos com configuração inicial C é instância de M . O significado de um modelo M , denotado por $\llbracket M \rrbracket$, é conjunto de suas instâncias.

Em tese, se as ligações correspondem aos identificadores armazenados pelos objetos, então poderiam ser omitidas da representação da configuração de um sistema. Isto é, poderíamos representar a estrutura de um sistema apenas pelos objetos existentes e as mensagens pendentes. As ligações poderiam ser “calculadas” a partir dos estados dos objetos. Contudo, as ligações são a parte relevante dos estados dos objetos, do ponto de vista do estado global do sistema. De fato, podemos ver uma estrutura como uma abstração do estado global em que todos os detalhes dos estados dos objetos são eliminados, exceto as ligações.

Observe que, pela definição, apenas a configuração inicial é caracterizada como instância de M . Nada é dito diretamente sobre as demais configurações alcançáveis. Contudo, a definição é suficiente para garantir que todas as configurações alcançáveis são instâncias de M .

Proposição 5.1 *Sejam M um modelo completo e $C \in \llbracket M \rrbracket$. Afirma-se: toda configuração C' alcançável a partir de C é instância de M , ou seja,*

$$C \vdash^* C' \Rightarrow C' \in \llbracket M \rrbracket, \quad \text{para todo } C'.$$

Demonstração. Perceba que um modelo completo contém todas as classes referenciadas internamente no modelo. Isto é, $\text{clsref}(M) \subseteq M$ (ver Definição 5.7 na página 88). Portanto, todos os objetos do sistema, mesmo os criados dinamicamente, são instâncias das classes em M . Argumento semelhante vale para as ligações e para mensagens. Somente ligações de tipos previstos no modelo podem ser criadas. Isto é, aquelas para as quais existe a associação adequada entre classes. Além disso, todas as inscrições de interação de um modelo, incluindo as de envio assíncrono de mensagens, são escritas a partir dos termos da álgebra Σ . Logo, todas as mensagens que podem ser produzidas pertencem aos domínios do modelo.

5.3 Concretização da sintaxe

O principal propósito da definição formal apresentada na Seção 5.1 é caracterizar precisamente o que são os modelos na notação que propomos. Na prática, contudo, não é viável construir modelos diretamente a partir dessas definições. Elementos de descrição mais adequados são fundamentais para a elaboração de modelos de sistemas reais. É a escolha e introdução desses elementos de descrição na notação que chamamos de *concretização da sintaxe*. Por oposição, as definições formais apresentadas anteriormente são ditas a *sintaxe abstrata* da notação.

Nesta seção, nosso objetivo é apresentar os elementos que compõem uma versão concreta da notação. É importante observar, contudo, que há diversos aspectos envolvidos no projeto de uma sintaxe concreta para uma linguagem. Esses aspectos vão desde os mais objetivos, como a escolha de construções básicas para os elementos da sintaxe abstrata, passando por outros de caráter mais prático, como a introdução de facilidades sintáticas e a introdução de extensões às construções básicas; até aspectos mais subjetivos, que incluem questões de usabilidade e simplicidade das construções, etc. Todos, porém, envolvem decisões que vão além da pura caracterização formal de modelos e que, portanto, estão além do escopo deste trabalho. Portanto, a notação que apresentamos se justifica pela necessidade imediata de elaborar e expressar modelos. A comunicação entre os participantes do projeto, a experimentação da notação e o desenvolvimento de alguns métodos relacionados dependem da sintaxe concreta para elaborar e expressar modelos. Mesmo a demonstração dos conceitos da sintaxe abstrata é facilitada pelo uso da sintaxe aqui apresentada.

Visão geral A disciplina de modelagem e a organização dos produtos que constituem um modelo estão intrinsecamente relacionados. Podemos dizer que a orientação a objetos também determina uma certa disciplina de modelagem. Logo, é natural que a organização interna dos modelos, na notação proposta, seja feita nos moldes da orientação a objetos.

Logo, a notação concreta que apresentamos se organiza de forma semelhante à usada pela maioria das notações OO: como conjuntos de diagramas. Cada diagrama determina uma visão de algum aspecto dos sistemas modelados. Os principais diagramas usados são: diagramas de classes, diagramas de comportamento e diagramas de objetos.

Diagramas de classe constituem “declarações” das entidades existentes no modelo e de seus relacionamentos. É nos diagramas de classe que usamos os mecanismos de associação, agregação e especialização para descrever classes a partir de outras e para relacioná-las. Os diagramas de comportamento são construídos sobre a base estabelecida pelos diagramas de classes. Têm a função de determinar o comportamento do sistema. Portanto, consistem basicamente em redes de Petri. Finalmente, os diagramas de objetos permitem descrever sistemas de objetos em um instante qualquer. Geralmente, são usados para determinar o estado inicial das instâncias de um modelos.

A notação apresentada usa a forma mais disseminada de redes de Petri coloridas como corpo das classes. Isto foi feito com o propósito de usar o suporte ferramental disponível para essa classe de redes, como base para a edição e construção dos modelos na notação que propomos. A ferramenta usada para editar os modelos apresentados é o *Design/CPN* (ver [SCK97]).

5.3.1 Um exemplo

Para introduzir a notação, iniciamos apresentando um exemplo simples de modelo. Isto proporciona um contato inicial com os principais elementos envolvidos e uma visão de conjunto que facilita a compreensão dos detalhes que são apresentados nas seções seguintes. O problema abordado é o clássico jantar dos filósofos, concebido por Dijkstra [Dij71]. A descrição precisa do que o modelo deve caracterizar é dada em seguida.

Descrição do Problema Ao redor de uma mesa redonda, sentam-se cinco filósofos para jantar. Sobre a mesa são dispostos cinco garfos, de tal forma que cada filósofo tenha dois a seu alcance: um a sua esquerda e outro a sua direita. Cada filósofo só é capaz de fazer duas

coisas: ou pensa ou come. Ao comer, usa os dois garfos ao seu alcance. Portanto, cada garfo é compartilhado por dois filósofos da mesa e enquanto um come, seus dois vizinhos devem estar pensando. A Figura 5.1 ilustra a cena: ao centro a mesa com os cinco garfos dispostos; ao redor da mesa os filósofos.

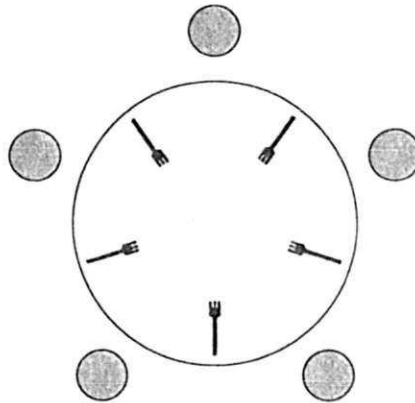


Figura 5.1: Ilustração da mesa de jantar dos filósofos

Desejamos construir uma solução distribuída para este problema. Filósofos e garfos devem ser entidades independentes que executam em nós independentes. Nosso problema consiste em construir um modelo para essa solução. Em um caso real, poderíamos dizer que há dois propósitos envolvidos: o modelo deve nos permitir analisar a solução proposta; e, eventualmente, deve servir de especificação para a fase de realização do sistema. Em nosso exemplo, o modelo tem o simples propósito de servir como demonstração dos principais elementos de modelagem envolvidos.

Diagramas de classes Os objetos existentes em um sistema, bem como seus relacionamentos, são descritos através de *diagramas de classes*. A Figura 5.2 ilustra o diagrama de classes para o problema em questão. Cada classe é representada por um retângulo com bordas arredondadas com o nome correspondente inscrito ao centro. A escolha dos nomes das classes é arbitrária mas, naturalmente, deve refletir as entidades envolvidas no problema. Os nomes das classes do modelo são *Filosofo* e *Garfo*.

Os arcos no diagrama denotam associações entre as classes. Cada associação determina possíveis ligações entre objetos das classes associadas. A inscrição e a seta que decoram cada associação determinam seu *sentido de navegação*. Isto é, determinam a partir de qual objeto da associação deve ser possível acessar o outro objeto. Formalmente, isto equivale

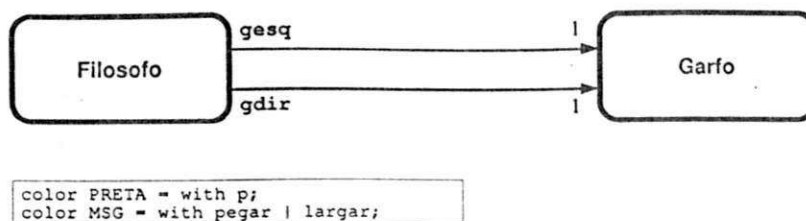


Figura 5.2: Diagrama de classes para o problema dos filósofos

a determinar qual objeto armazena o identificador do outro. Os números 1 ao lado das associações determinam sua multiplicidade. Logo, as associações do modelo determinam que cada filósofo está associado a exatamente dois garfos, cujos seletores são `gesq` e `gdir`. Naturalmente, os seletores correspondem, respectivamente, ao garfo à esquerda e ao garfo à direita de cada filósofo.

O último elemento envolvido no diagrama de classes é um nó de declaração. Um nó de declaração consiste em uma caixa de texto em que são descritos tipos de dados em CPN-ML. O nó de declaração está no escopo de todas as classes envolvidas no diagrama. Logo, os conjuntos de cores denominados `PRETA` e `MSG` são visíveis em ambas as classes do modelo.

Interpretação do diagrama de classes Em um diagrama de classes declaramos nomes de classes, associações e seletores. Nomes de classes fazem parte da sintaxe abstrata e já foram discutidos nas seções anteriores. Associações, contudo, merecem atenção especial. Formalmente, uma associação entre duas classes consiste em um conjunto de pares de objetos. Logo, se **Filosofo** é o conjunto dos objetos descritos pela classe `Filosofo` e **Garfo** é o conjunto dos objetos descritos pela classe `Garfo`, ou seja,

$$\mathbf{Filosofo} = \llbracket \text{Filosofo} \rrbracket \quad \text{e} \quad \mathbf{Garfo} = \llbracket \text{Garfo} \rrbracket,$$

então cada associação declarada entre essas duas classes é um conjunto de pares $\langle f, g \rangle$ tais que $f \in \mathbf{Filosofo}$ e $g \in \mathbf{Garfo}$, ou seja, é uma relação r que satisfaz

$$r \subseteq \mathbf{Filosofo} \times \mathbf{Garfo}.$$

Observe, contudo, que não há, na sintaxe abstrata, um equivalente direto para associações. De fato, as associações do diagrama de classes devem ser interpretadas como declarações de associações. A direção de navegação e os seletores servem exatamente para

complementar as declarações, indicando as classes que as devem realizar. A origem da direção de navegação (da seta) determina a classe em que a associação deve ser realizada. E o seletor declara a variável usada para escrever expressões de navegação. Logo, formalmente, cada seletor é uma variável-objeto.

Os seletores *gesq* e *gdir*, do exemplo, são variáveis-objeto do tipo *Garfo* que podem ser usadas no corpo da classe *Filosofo* para fazer referência aos garfos da direita e da esquerda correspondentes a cada objeto filósofo⁶. Ou seja, a avaliação de *gesq* em objetos da classe *Filosofo* produz o identificador do garfo *g* tal que $\langle f, g \rangle \in r$, onde *r* é a relação correspondente à associação que liga filósofos aos seus garfos da esquerda. Logo, se a relação em determinado instante inclui $\langle f_1, g_1 \rangle$, então

$$\llbracket f_1.\text{gesq} \rrbracket = g_1$$

Finalmente, é necessário relembrar que ao declararmos as classes de um modelo, estamos indiretamente declarando tipos identificadores para os objetos. Logo, através do diagrama de classes da Figura 5.2 na página anterior, declaramos implicitamente dois tipos de identificadores de objetos: identificadores para filósofos e para garfos.

Diagramas de comportamento Os *diagramas de comportamento* do modelo atribuem um corpo a cada classe presente nos diagramas de classe. O corpo de cada classe é uma rede de Petri Colorida com inscrições de interação. A Figura 5.3 mostra o diagrama de comportamento para o exemplo, em que são detalhados os corpos das classes *Filosofo* e *Garfo*—subfiguras (a) e (b), respectivamente.

A modelagem dos objetos é feita focalizando o comportamento de cada classe separadamente. Avaliemos, por exemplo, o funcionamento da classe *Filosofo*. O corpo da classe consiste numa rede com dois lugares—*pensando* e *comendo*—e duas transições—*comer* e *pensar*. As inscrições de marcação inicial do corpo determinam o estado inicial dos objetos da classe. Logo, filósofos têm estado inicial igual a

$$m_0 = 1'(p, \text{pensando}),$$

ou seja, uma ficha *p* no lugar *pensando*, o que modela que inicialmente cada filósofo está pensando. A partir do estado inicial m_0 , apenas a ação correspondente à transição *comer*

⁶Relembre que variáveis-objeto são do tipo *identificador* da classe correspondente. Logo, *gesq* e *gdir* são ligadas a identificadores de garfos.

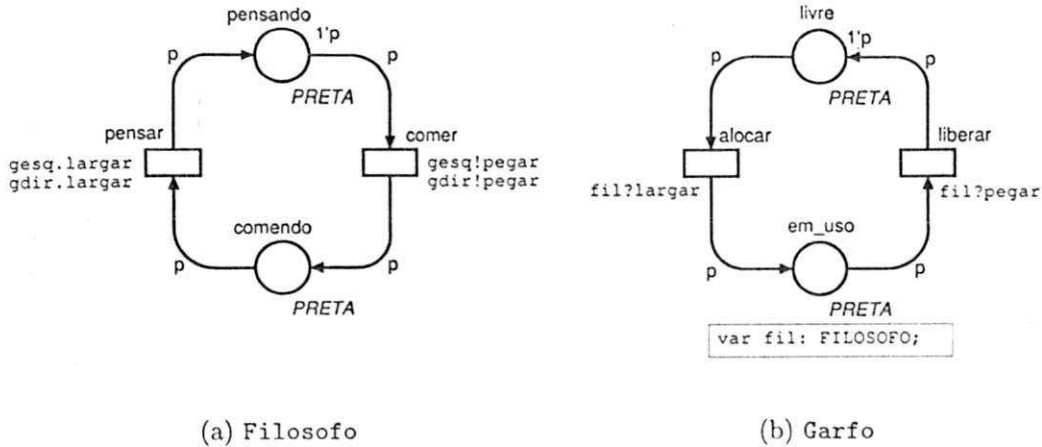


Figura 5.3: Diagramas de comportamento para o problema dos filósofos

está habilitada. Ao ocorrer, a ação leva o filósofo ao estado

$$m_1 = 1'(p, \text{comendo}),$$

que modela o estado em que o filósofo está comendo. No estado m_1 , apenas a transição pensar está habilitada. Ao ocorrer, o filósofo volta a seu estado inicial m_0 .

O comportamento da classe Garfo é semelhante. Inicialmente cada garfo está livre no estado m_0 . Nesse estado, apenas a transição alocar pode ocorrer, o que leva o garfo ao estado m_1 que indica que está em uso. A partir de m_1 , apenas a transição liberar pode ocorrer, fazendo o garfo retornar a seu estado inicial. Os estados m_0 e m_1 dos garfos são:

$$m_0 = 1'(p, \text{livre}) \quad \text{e} \quad m_1 = 1'(p, \text{em_uso}).$$

Observe ainda que a variável f declarada na classe Garfo é do tipo Filosofo. De fato, a variável assume valores do tipo identificador de objetos da classe Filosofo.

Diagramas de objetos Os diagramas até aqui apresentados descrevem somente os tipos de objetos do sistema e seus possíveis relacionamentos. Em nenhum deles há indicações sobre o número de filósofos e garfos que compõem o sistema ou sobre a forma em que se ligam. Essa é a função de *diagramas de objetos* ou de *instâncias*.

Um diagrama de objetos descreve um conjunto de objetos e suas ligações em um dado instante. Formalmente, é a representação gráfica de uma instância do modelo—uma configuração. Naturalmente, há um número infinito de instâncias do modelo. Contudo, na maioria dos casos a análise parte de uma *configuração inicial*, tomada como referência.

A configuração inicial para o jantar dos filósofos, mostrada na Figura 5.4, consiste em cinco instâncias da classe **Filosofo**, com identificadores f_1, \dots, f_5 ; e cinco instâncias da classe **Garfo**, com identificadores g_1, \dots, g_5 . O estado de cada objeto é o estado inicial correspondente à classe a que pertence. Ou seja, todos os filósofos estão pensando e todos os garfos estão livres. Observe ainda que para cada objeto indicamos a qual associação cada ligação pertence. Por exemplo, os garfos à esquerda e à direita do filósofo f_5 são, respectivamente, g_5 e g_1 .

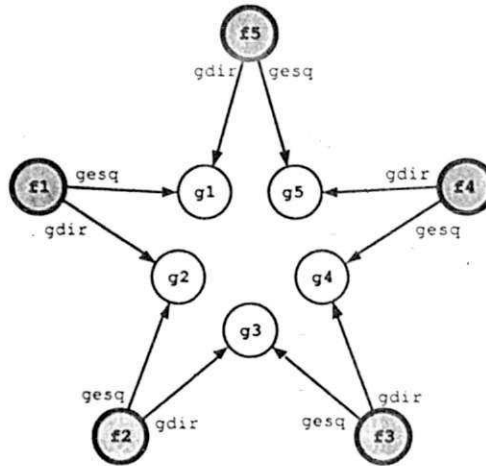


Figura 5.4: Diagrama de objetos para o problema dos filósofos: configuração inicial

Formalmente, os identificadores usados para nomear os objetos em um diagrama de objetos são constantes que representam nomes do sistema. Por convenção, contudo, adotamos a seguinte regra para “gerar” as constantes identificadoras de objetos: usamos o menor prefixo do nome da classe que a distingue unicamente no modelo (f para a classe **Filosofo** e g para **Garfo**) e usamos índices inteiros para distinguir entre as instâncias.

Funcionamento Observe que na análise inicial das classes ignoramos completamente as inscrições de interação. Contudo, podemos complementar o entendimento do funcionamento das classes de objetos, avaliando as ações modeladas pelas inscrições de interação. Por exemplo, a transição *comer* da classe **Filosofo** tem inscrições de interação $gesq!pegar$ e $gdir!pegar$. Formalmente, a transição modela a seguinte ação potencial de objetos da classe **Filosofo**

$$m_0 \xrightarrow{g_1!pegar \circ g_2!pegar} m_1, \quad \text{onde } g_1 = \llbracket gesq \rrbracket \text{ e } g_2 = \llbracket gdir \rrbracket.$$

A sintaxe indica sincronização. Relembre que, pelas regras de ocorrência de sistemas de objetos, uma ação de sincronização só pode ocorrer, ou só caracteriza um *evento*, se a ação complementar ocorrer simultaneamente (ver Definição 4.8 na página 75). Para isso, os objetos g_1 e g_2 indicados acima devem estar aptos a executar as ações complementares. Uma breve análise no corpo da classe *Garfo*, nos permite perceber que a ação complementar é modelada pela transição *alocar*. A ação potencial modelada por essa transição é

$$m_0 \xrightarrow{f?Pegar} m_1.$$

Logo, o evento completo que modela um filósofo f deixando de pensar e passando a comer consiste em três ações: $f.comer$, $g_1.alocar$ e $g_2.alocar$, onde vale a relação entre f , g_1 e g_2 como indicados anteriormente. Isto é completamente consistente com o problema descrito, em que um filósofo só pode passar a comer se tiver pego os dois garfos correspondentes.

De forma análoga, podemos entender os demais eventos do modelo. As inscrições da transição *pensar* da classe *Filosofo* são $gesq.largar$ e $gdir.largar$. Portanto, a transição modela a ação potencial

$$m_1 \xrightarrow{g_1.largar \circ g_2.largar} m_0, \quad \text{onde } g_1 = \llbracket gesq \rrbracket \text{ e } g_2 = \llbracket gdir \rrbracket.$$

Ou seja, ao passar a pensar o filósofo “larga” os dois garfos que usava para comer. Desta vez, contudo, a interação consiste em saídas *assíncronas*. Ou seja, uma mensagem contendo o dado *largar* é produzida para cada um dos garfos. A ação complementar da interação é modelada pela transição *liberar* da classe *Garfo*, mas pode ocorrer assincronamente. Logo, a ação de envio das mensagens, $f.pensar$, pode ocorrer independentemente das respectivas recepções nos garfos— $g_1.liberar$ e $g_2.liberar$.

Espaço de estados Uma forma gráfica de representar o comportamento de um sistema é através de seu *espaço de estados*. A idéia é construir um grafo em que os nós representam estados do sistema e cada arco o evento que o faz passar de um estado a outro. Logo, para representar o comportamento de um modelo podemos construir um grafo onde cada nó seja uma configuração e cada arco um evento do sistema (grafos de alcançabilidade são definidos formalmente na Seção 6.2 na página 118).

Para reduzir o exemplo, considere a configuração inicial mostrada na Figura 5.5. Esta configuração consiste em apenas dois filósofos e dois garfos em uma mesa.

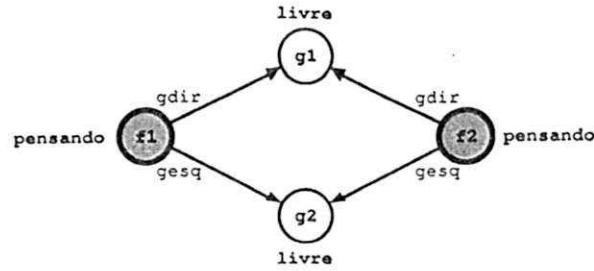


Figura 5.5: Configuração inicial com dois filósofos

Para simplificar as inscrições do grafo, introduzimos a seguinte notação para representar as ações dos objetos. Observe que o par $\langle f_i, m_j \rangle$ representa o objeto f_i em seu estado m_j . Definimos:

$$\begin{aligned} f_1.\text{comer} &\stackrel{\text{def}}{=} \langle f_1, m_0 \rangle \xrightarrow{g_1!\text{pegar} \circ g_2!\text{pegar}} \langle f_1, m_1 \rangle \\ f_1.\text{pensar} &\stackrel{\text{def}}{=} \langle f_1, m_1 \rangle \xrightarrow{g_1.\text{largar} \circ g_2.\text{largar}} \langle f_1, m_0 \rangle \\ f_2.\text{comer} &\stackrel{\text{def}}{=} \langle f_2, m_0 \rangle \xrightarrow{g_2!\text{pegar} \circ g_1!\text{pegar}} \langle f_2, m_1 \rangle \\ f_2.\text{pensar} &\stackrel{\text{def}}{=} \langle f_2, m_1 \rangle \xrightarrow{g_2.\text{largar} \circ g_1.\text{largar}} \langle f_2, m_0 \rangle \end{aligned}$$

Usamos a mesma notação para indicar ações dos garfos:

$$\begin{aligned} g_i.\text{alocar} &\stackrel{\text{def}}{=} \langle g_i, m_0 \rangle \xrightarrow{f?\text{pegar}} \langle g_i, m_1 \rangle \\ g_i.\text{liberar} &\stackrel{\text{def}}{=} \langle g_i, m_1 \rangle \xrightarrow{f?\text{largar}} \langle g_i, m_0 \rangle \end{aligned}$$

Pela análise feita anteriormente, na configuração inicial há apenas dois eventos possíveis:

$$f_1.\text{comer} \circ g_1.\text{alocar} \circ g_2.\text{alocar} \quad \text{ou} \quad f_2.\text{comer} \circ g_2.\text{alocar} \circ g_1.\text{alocar}.$$

Finalmente, podemos representar o espaço de estados do sistema, calculando todas as configurações alcançáveis e os eventos que levam a elas. A Figura 5.6 é a representação gráfica do espaço de estados do exemplo. A configuração inicial encontra-se na parte superior da figura. O estado interno de cada objeto em cada configuração é representado pela inscrição: p indica que o filósofo está pensando, c que está comendo, l indica que o garfo está livre e u que está em uso. Os pequenos nós quadrados ligados aos garfos em algumas configurações representam mensagens pendentes que transportam a mensagem largar.

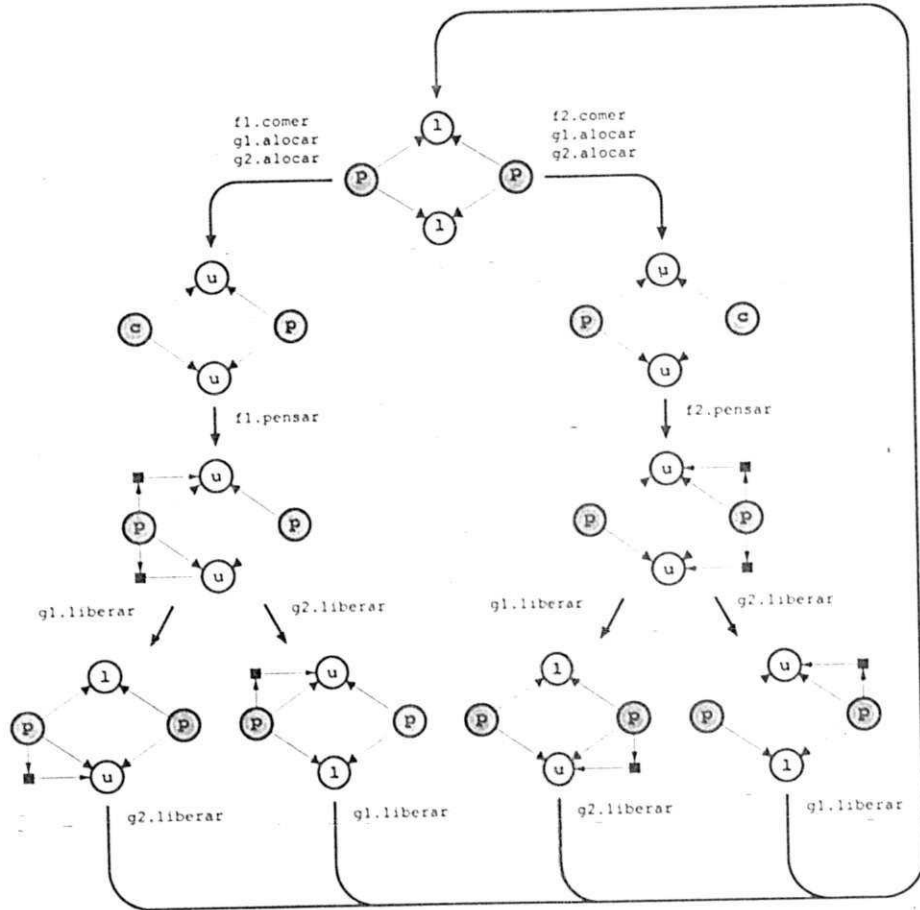


Figura 5.6: Representação do comportamento para a configuração da Fig. 5.5

Comentários Devido à simplicidade do problema, nem todos os elementos disponíveis na notação foram utilizados neste exemplo. Em particular, podemos dizer que a *topologia de interconexão* do sistema é estaticamente definida. Ou seja, nenhuma ligação entre os objetos é modificada no decorrer do funcionamento do sistema e nenhum objeto é criado ou eliminado da configuração. Apesar disso, o exemplo cumpre seu papel de introduzir os elementos da notação. Nas seções seguintes definimos mais rigorosamente cada um dos elementos da notação, incluindo os que não foram usados neste exemplo.

5.3.2 Diagramas de classes

Cada diagrama de classes do modelo representa uma visão sobre as classes envolvidas. A notação propriamente dita é a usada na maioria das linguagens de modelagem OO⁷. Representamos classes por retângulos e relacionamentos por arcos entre as classes. Cada classe e cada relacionamento é decorado com elementos que os detalham. Além das classes e relacionamentos, cada diagrama de classes pode conter um nó de declaração de tipos.

Nós de declaração Na sintaxe abstrata, a modelagem de dados é um aspecto quase que isolado do restante do modelo. Na prática, contudo, essa abordagem não é conveniente. De fato, é praticamente impossível antecipar todos os tipos de dados que serão usados em um modelo, antes de construí-lo. Além disso, é comum que tipos de dados necessários em algumas partes do modelo não sejam necessários em outras. Isto motiva a modelagem dos tipos de dados ao longo das partes do modelo.

Para controlar a declaração de tipos, usamos o conceito de *escopo*. Cada nó de declaração⁸ é válido em um subconjunto das classes do modelo. A esse conjunto de classes chamamos *escopo do nó*. Por extensão, definimos o escopo de um tipo de dados como sendo o escopo do nó em que foi definido. Naturalmente, o escopo do nó de declaração de um diagrama de classes é o conjunto das classes no próprio diagrama.

Escopos podem se sobrepor—uma mesma classe pode pertencer ao escopo de mais de um nó de declaração. Contudo, os conjuntos de nomes de escopos com sobreposição devem ser disjuntos. Ou seja, é um erro declarar dois tipos com o mesmo nome em escopos que se

⁷Devido à disseminação da notação denominada UML [RJB99], preferimos desenhar os modelos seguindo as orientações dessa notação.

⁸As declarações de tipos em CPN-ML são agrupadas em unidades denominadas *nós de declaração*.

sobrepõem.

(Observe que os tipos identificadores e o tipo genérico de identificadores são sempre globais. Isto é, seus escopos incluem todas as classes do modelo. Observe ainda que o nome do tipo global é Objeto—isto corresponde ao nome da classe genérica \textcircled{O} da sintaxe abstrata.

Relacionamentos Há três tipos de relacionamentos entre classes: associações, agregações e especializações. A Figura 5.7 ilustra como cada um dos relacionamentos é representado graficamente.

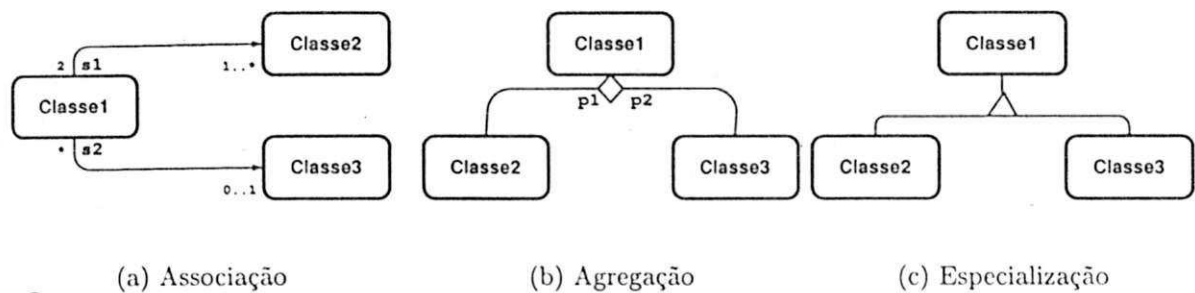


Figura 5.7: Relacionamentos entre classes

Associações Associações descrevem possíveis ligações entre objetos de duas classes. Graficamente, associações são representadas por arcos entre as classes. Os elementos usados para decorar associações determinam: a direção de navegação, o seletor e a multiplicidade. A Figura 5.7(a) ilustra a notação usada em casos típicos de associação.

A *direção de navegação*, indicada pela seta em uma das extremidades do arco, determina a classe que deve realizar a associação: a do lado da origem da seta (na página 110 definimos o que significa *realizar* uma associação). O *seletor* declara uma variável-objeto do tipo da classe associada que pode ser usado no corpo da classe para “navegar” através da associação. Em outras notações OO, os seletores são denominados de *papéis*. Neste caso, a inscrição é colocada do lado oposto da associação.

A *multiplicidade* de uma associação especifica a quantidade de ligações permitidas para cada tipo de objeto participante da associação. A quantidade de ligações de cada lado é denotada por faixas de inteiros no formato

<valor mínimo>..<valor máximo>,

ou simplesmente como um único valor. Também é possível usar * para indicar que a cardinalidade não é restrita. Se a cardinalidade de qualquer um dos lados de uma associação não for explicitamente indicada, então tem valor *.*.

Considere, por exemplo, as associações entre as classes *Filosofo* e *Garfo* do diagrama da Figura 5.2. As decorações das associações determinam que devem ser realizadas pela classe *Filosofo* e que os seletores são *gesq* e *gdir* para cada associação. Além disso, determinam que cada filósofo está ligado a exatamente um garfo esquerdo e a um garfo direito. O outro lado da multiplicidade está indeterminado, logo cada garfo pode estar ligado a qualquer número de filósofos. Uma forma mais estrita poderia indicar que a cada garfo correspondem exatamente dois filósofos, um à esquerda e outro à direita.

Observe que as decorações das associações nos diagramas de classes devem ser entendidas como especificações para as realizações determinadas pelos corpos das classes.

Agregações Através da agregação é possível descrever uma classe de objetos, a partir da combinação de outros. Também é chamada relação *parte-todo* ou *composição*. Em nossa abordagem consideramos a agregação uma operação sobre as descrições de classe (ver explicação sobre a realização da agregação página 110). Por exemplo, a classe *Classe1* da Figura 5.7(b) é descrita a partir de dois objetos, um do tipo *Classe2* e outro do tipo *Classe2*. A idéia é que a descrição da *Classe1* possa usar esses dois objetos como se fossem parte da própria classe. As partes podem ser usadas pela *Classe1* através de inscrições de interação, usando os seletores indicados.

Conceitualmente, a agregação pode ser interpretada como uma associação com semântica mais estrita. Isto motiva a realização da agregação de forma semelhante à da associação. Ou-seja, em alguns casos a agregação pode ser realizada usando associações. Neste caso, contudo, a classe deve explicitamente criar e manipular adequadamente os objetos que compõem o agregado.

Especializações A especialização consiste em descrever uma classe mais específica a partir de outra mais genérica—a relação inversa da especialização é a *generalização*. Dizemos que a classe mais especializada, dita a *subclasse*, *herda* as características da genérica, dita a *superclasse*. A especialização também é definida como uma operação sobre classes. Graficamente, ilustramos a especialização como mostra a Figura 5.7(c). Dizemos que as

classes `Classe2` e `Classe3` especializam a classe `Classe1`. Na prática, isto significa que o corpo das classes especializadas é descrito a partir do corpo da superclasse, usando a palavra-chave `super` para escrever expressões de uso (ver explicação a seguir). Novamente, as expressões de uso têm exatamente a mesma sintaxe que as inscrições de interação. As subclasses podem introduzir novas propriedades específicas.

5.3.3 Diagramas de comportamento

Os diagramas de comportamento correspondem aos corpos das classes expressos como redes de Petri coloridas com inscrições de interação. Graficamente, cada rede é representada da forma convencional. Isto é, como um grafo bipartido, cujos nós são transições e lugares. As transições são desenhadas como retângulos e os lugares como círculos. Inscrições de interação são escritas como rótulos das transições. Além do corpo propriamente dito, a cada classe corresponde um *nó de declaração* de tipos.

Nó de declaração O nó de declaração de uma classe permite declarar tipos estritamente locais à classe. Isto é conveniente para esconder detalhes da realização da classe do restante do modelo. Qualquer tipo pode ser declarado em um nó de declaração local, exceto tipos de dados usados em interações com outros objetos. Nesse caso, é obrigatória a declaração dos tipos em nós com escopo que inclua todas as classes que os usem.

Sintaxe das inscrições de interação A sintaxe usada para escrever inscrições de interação é bastante similar à usada na versão abstrata da notação (ver Definição 5.3 na página 86). A principal diferença é que usamos expressões em CPN-ML para construí-las. A BNF abaixo descreve a sintaxe das inscrições:

$$\begin{aligned} \langle \text{inscrição} \rangle ::= & \langle \text{entrada de dados} \rangle \mid \langle \text{saída de dados} \rangle \\ & \mid \langle \text{instanciação} \rangle \mid \langle \text{auto-destruição} \rangle \\ & \mid \langle \text{inscrição} \rangle^* \end{aligned}$$

$$\langle \text{entrada de dados} \rangle ::= \langle \text{expressão-objeto} \rangle ? \langle \text{expressão} \rangle$$

$$\langle \text{saída de dados} \rangle ::= \langle \text{expressão-objeto} \rangle . \langle \text{expressão} \rangle \mid \langle \text{expressão-objeto} \rangle ! \langle \text{expressão} \rangle$$

$$\langle \text{instanciação} \rangle ::= \langle \text{variável-objeto} \rangle : \text{New } \langle \text{classe} \rangle$$

$$\langle \text{auto-destruição} \rangle ::= \square$$

Os conjuntos sintáticos $\langle \textit{expressão-objeto} \rangle$, $\langle \textit{expressão} \rangle$, $\langle \textit{variável-objeto} \rangle$ e $\langle \textit{classe} \rangle$ correspondem a expressões, nomes de variáveis e nomes de classes declarados em CPN-ML. Evidentemente, $\langle \textit{expressão-objeto} \rangle$ deve ser uma expressão que ao ser avaliada resulte em um objeto; $\langle \textit{expressão} \rangle$ pode ser de um tipo qualquer; $\langle \textit{variável-objeto} \rangle$ é uma variável cujo tipo corresponde a identificadores de objetos; e $\langle \textit{classe} \rangle$ é o nome de uma classe do modelo e do tipo de identificadores correspondentes. O terminal \square é usado para marcar transições finais—na prática, evitamos o uso de \square em inscrições complexas.

É importante observar que os tipos identificadores de objeto são considerados globais. Logo, em qualquer classe podemos “usar” objetos de outra classe.

Corpo de uma classe O corpo de cada classe é uma rede que realiza as especificações determinadas pelos diagramas de classes. Dizemos que uma classe *realiza* uma associação se suas instâncias podem armazenar ligações para os objetos da classe associada. Há duas formas de fazer isto: ou o corpo da classe contém lugares que podem armazenar identificadores dos objetos; ou a classe determina *ligações estáticas* para objetos da classe associada. O primeiro caso equivale a garantir que $C_j \in \text{clsref}(C_i)$ —ver Definição 5.6 na página 88. O segundo consiste em usar constantes de identificadores nas expressões da classe. Neste caso, embora os identificadores não façam parte do estado do objeto propriamente dito, a ligação existe e é estática.

Contudo, observe que, pela definição dada, o corpo da classe *Filosofo* não realiza as associações especificadas no diagrama de classes (ver Figuras 5.3 na página 101 e 5.2 na página 99): não há nenhum lugar que armazene os identificadores dos garfos de um filósofo, nem há ligações estáticas. Isto ocorre porque permitimos uma simplificação (*syntactic sugar*) para a realização de associações cuja multiplicidade seja igual a 1.

Observe que se a multiplicidade de uma associação é 1 em um dos lados, o seletor do lado oposto só pode assumir um único valor em cada instante. Neste caso, e se o valor permanece inalterado no decorrer do ciclo-de-vida do objeto, subentende-se o armazenamento do identificador determinado a partir da configuração inicial. (De fato, é possível estender este *padrão de projeto* e definir macros para manter e alterar as ligações de cada objeto, independentemente do restante do corpo da classe.)

Realização de agregação e especialização Há várias formas de realizar a especialização e a agregação. Contudo, em nossa abordagem as consideramos operações puramente sintáticas. Isto é, operações sobre classes, definidas com o propósito de facilitar a obtenção de novas classes a partir das já definidas.

Suponha que a classe \mathbb{C}_i seja descrita como um agregado de objetos das classes $\mathbb{C}_j, \dots, \mathbb{C}_k$. Podemos denotar a agregação através da soma formal

$$\mathbb{C}_i = K_i(s, \dots, t) + s:\mathbb{C}_j + \dots + t:\mathbb{C}_k,$$

onde o termo $K_i(s, \dots, t)$ é o corpo específico da classe \mathbb{C}_i e os termos da forma $s:\mathbb{C}_j$ denotam as partes do agregado. Conceitualmente, a expressão indica que cada objeto da classe \mathbb{C}_i é constituído por objetos-parte denotados por s, \dots, t cujos tipos são, respectivamente, $\mathbb{C}_j, \dots, \mathbb{C}_k$. Além disso, determina que o corpo da classe agregada é expresso em função dessas partes. Daí porque é denotado por $K_i(s, \dots, t)$.

Do ponto de vista de uso, a construção funciona como se de fato houvesse instâncias das classes $\mathbb{C}_j, \dots, \mathbb{C}_k$ dentro de cada objeto da classe \mathbb{C}_i . Logo, o uso das partes é descrito usando as inscrições de interação já descritas. Formalmente, contudo, o que ocorre é que o corpo da classe agregada é complementado com os corpos das partes—podemos pensar nisto como a união disjunta das redes. E as expressões de uso apenas determinam como as partes são coordenadas pelo corpo K_i . É importante observar ainda que as partes são efetivamente encapsuladas pelo agregado. Logo, só podem ser usados pelo corpo específico da classe agregada e não diretamente por outros objetos.

Lógica semelhante é usada para expressar o significado da especialização. A principal diferença é que o corpo herdado pela superclasse define uma parte acessível da subclasse. Suponha que \mathbb{C}_i é subclasse de \mathbb{C}_j , então podemos denotar a operação por

$$\mathbb{C}_i = K_i(\text{super}) \odot \text{super}:\mathbb{C}_j.$$

Por convenção, o seletor usado na especialização é denominado *super* e não precisa ser declarado explicitamente. Os termos da expressão são semelhantes aos usados para a agregação. A principal diferença é o significado da operação, denotada por \odot . Neste caso, o corpo da classe \mathbb{C}_j é herdado por \mathbb{C}_i . Isto significa que além de complementar o corpo da subclasse, o corpo da superclasse é parte diretamente acessível. Conseqüentemente, todo o comportamento é efetivamente herdado. Também é possível determinar diferenças no comportamento através do corpo específico $K_i(\text{super})$.

5.3.4 Outros diagramas

Diagramas de objetos Diagramas de objetos ou de instâncias são usados para determinar estados específicos em que um sistema (instância do modelo) se encontra em um dado instante de tempo. Formalmente, diagramas de objetos correspondem a configurações. Graficamente, são representados como grafos, cujos nós representam objetos e cujos arcos representam ligações.

Em geral, os diagramas de objetos são decorados com informações adicionais sobre a configuração. Ainda assim, algumas informações são abstraídas. Geralmente é importante indicar, por exemplo, o tipo de cada objeto—a classe a que pertence, mas não os estados internos. Tome como exemplo as configurações do exemplo do jantar dos filósofos na Figura 5.6 na página 105. Em outras situações, é conveniente usar *diagramas parciais*, isto é, diagramas em que nem todos os objetos são explicitamente representados. Isto permite evidenciar os aspectos relevantes da configuração.

Um dos principais usos de diagramas de objetos é a caracterização do estado inicial de instâncias do modelo. Isto é comum quando instâncias específicas do modelo devem ser analisadas. Para indicar configurações iniciais, contudo, é necessário indicar todos os objetos com os respectivos estados.

Outros diagramas Os diagramas apresentados são suficientes para caracterizar modelos e instâncias de modelos. Durante o processo de modelagem, contudo, é comum que outros diagramas auxiliares sejam usados. Nos experimentos realizados com a notação apresentada, diagramas de seqüências de mensagens e diagramas de colaboração, por exemplo, se mostraram extremamente úteis para expressar cenários de comunicação entre objetos. Logo, contribuem para a descrição de protocolos. Do ponto de vista formal, é perfeitamente viável tomá-los como complemento da especificação do modelo. Desta forma, podem ser usados para validação, através da verificação de consistência entre as informações.

5.4 Conclusões

A notação apresentada neste capítulo determina uma abordagem orientada a objetos para a construção de modelos de redes Petri. O comportamento dos objetos é efetivamente expresso através de redes de Petri. E o comportamento global dos sistemas é expresso como

a coordenação dessas várias redes, através das regras que caracterizam o comportamento dos objetos. A composição é feita através do conceito de *interação*, derivado do modelo conceitual de objetos.

A notação foi apresentada em dois níveis: primeiro definimos o que chamamos de sintaxe abstrata e em seguida sua concretização (ver Seção 3.2). Podemos dizer que as Seções 5.1 e 5.2 definem formalmente o que entendemos por *redes de Petri orientadas a objetos* e que a Seção 5.3 define uma notação concreta e algumas técnicas para a *realização* de modelos OO em termos do formalismo definido.

No Apêndice A apresentamos um exemplo mais complexo de modelagem em que um sistema distribuído de software é abordado através da notação aqui introduzida.

Capítulo 6

Análise e validação

No capítulo anterior definimos uma forma orientada a objetos de redes de Petri. Neste, avaliamos a integração proposta, a partir do mesmo ponto de vista usado na avaliação das demais propostas de integração no Capítulo 2. Logo, o que está em questão é *se e como* a abordagem proposta permite adaptar os métodos, técnicas e benefícios originais de redes de Petri e da orientação a objetos. Para suportar a argumentação, definimos os conceitos fundamentais para o desenvolvimento de métodos de análise baseados na exploração de espaços de estados.

6.1 Introdução

Um dos principais problemas das propostas de integração discutidas no Capítulo 2 é a dificuldade de adaptar as técnicas de análise e modelagem originais de cada um dos dois mundos. Clarke e Wing indicam dois aspectos importantes a considerar ao integrar diferentes métodos:

1. encontrar um *estilo* adequado; e
2. encontrar um *significado* adequado.

O *estilo* se refere aos aspectos gerais que dão cara ao método. Em grande parte, esses elementos determinam o uso e o sucesso do método na prática. A integração deve considerar os estilos originais de cada método e encontrar uma forma adequada de combiná-los, preservando o que há de mais relevante em cada um.

O *significado* se refere à clara caracterização da semântica formal dos modelos. Ao integrar métodos, é necessário expressar a semântica, combinando os significados de cada método. Assim, as duas visões de uma especificação são combinadas em uma única, permitindo compreender como refinamentos e modificações em uma das visões afetam a outra (ver [CW96]).

No caso de redes de Petri e orientação a objetos, as propostas têm falhado em encontrar *estilo* e *significado* adequados. As diferenças existentes entre os formalismos que caracterizam a orientação a objetos é uma das causas. Em particular, nenhuma das propostas parte da caracterização formal da orientação a objetos. As implicações são evidentes tanto no *estilo* quanto no *significado* das propostas. A integração que propomos procura intencionalmente evitar este problema. A idéia é identificar os elementos fundamentais que caracterizam o estilo e o significado de cada um dos métodos integrados e combiná-los de forma que sejam preservados da melhor forma possível.

Orientação a objetos A notação proposta preserva os elementos fundamentais que caracterizam o estilo da orientação a objetos. A notação é conservativa em relação à orientação a objetos. Como na grande maioria das notações e métodos orientados a objetos, o conceito fundamental na notação que propomos é a *classe*. Toda a modelagem gira em torno deste conceito e dos mecanismos para relacioná-las: associações, agregações e especializações.

Conseqüência direta disto é a facilidade para adaptar os métodos e outros elementos estilísticos da orientação a objetos para a notação proposta. Um exemplo é dado pela chamada *concretização da sintaxe*, apresentada no capítulo anterior. Observe que o artefato conhecido como *diagrama de classes* não é parte da definição formal da notação. O artefato foi adotado na notação concreta como forma de documentar a especificação das classes que devem ser elaboradas. Isto só é possível devido à adoção adequada dos conceitos da orientação a objetos. O mesmo pode ser feito em relação a outros artefatos.

Devido à ampla disseminação, é relativamente fácil identificar os elementos que compõem o estilo da orientação a objetos. O mesmo, contudo, não é verdade em relação ao seu significado—em especial porque tratamos de orientação a objetos concorrente. Este problema é decorrente do uso restrito de notações formais para expressar o significado de modelos OO. Na notação proposta, resolvemos este problema adotando explicitamente um

modelo conceitual no Capítulo 3 e sua respectiva formalização em termos de sistemas de objetos no Capítulo 4.

Certamente há outros modelos formais de objetos que poderíamos ter adotado. Preferimos adotar um modelo específico, contudo, para melhor tratarmos das particularidades na integração com redes de Petri. Observe, por exemplo, que sistemas de objetos apenas complementam redes de Petri nos aspectos de interação entre objetos, mas não podem ser considerados um modelo completo de computação ao estilo de π -cálculo. Outra particularidade relevante é a escolha das primitivas de interação sem retorno explícito de dados. Em muitas notações é comum expressar o comportamento de um objeto em termos de métodos que “retornam” resultados ao objeto invocador. Uma decisão determinante de toda a notação foi a escolha de primitivas que apenas estimulam o comportamento de outros objetos mas que não necessariamente retornam dados. Esta escolha é decorrente da natureza concorrente dos sistemas para os quais a notação é proposta. Apesar, o padrão de comunicação baseado em retorno é facilmente modelado combinando as primitivas disponíveis.

Redes de Petri Do ponto de vista de redes de Petri, a integração proposta também é conservativa. Redes de Petri são usadas para descrever as classes de objetos do modelo. Um aspecto importante é que as redes são usadas sem *nenhuma* modificação estrutural. Em particular, na versão concreta da notação usamos redes de Petri Coloridas exatamente como são definidas para a ferramenta Design/CPN.

O significado das redes também é preservado em relação à definição convencional. Ao considerarmos cada classe independentemente, a rede determina o exato comportamento da classe de objetos. Em outras abordagens, a rede-classe não podia ser analisada de forma independente do restante do modelo (das outras redes). A análise independente não permitia observar/antecipar todo o comportamento possível dos objetos da classe. Na formalização apresentada isso não ocorre. O espaço de estados induzido pela corpo da classe corresponde ao espaço de estados total de cada objeto instanciado da classe. Esta propriedade é chave para a elaboração de classes efetivamente reutilizáveis.

Semântica global Foi a partir desta observação que definimos a forma como as redes são usadas em nossa proposta e como seu comportamento é combinado para determinar o comportamento dos sistemas de objetos. Na maioria das outras abordagens o conceito

de estado do sistema é definido como a generalização do conceito de marcação. Da forma análoga, o comportamento é diretamente expresso por modificações da regra de disparo. Esta abordagem, contudo, desequilibra a relação entre a semântica de redes de Petri e a da notação proposta. Em nossa abordagem, as redes são elementos formais independentes que expressam o comportamento dos objetos de um sistema de objetos. Desta forma, preservamos a semântica original de redes de Petri. Logo, o projetista acostumado a redes de Petri poderá usar a mesma lógica para modelar objetos.

Para expressar a semântica global dos modelos, usamos as regras de comportamento de sistemas de objetos. Em nenhum momento é necessário compor explicitamente as estruturas das redes das classes. Ao invés disso, *compomos o comportamento expresso nas redes*, através das regras de comportamento de sistemas de objetos. É esta mudança na forma de expressar a semântica da integração que consideramos fundamental para garantir a possibilidade de analisar os modelos. As regras apenas restringem o comportamento total dos objetos. Logo, podemos concluir que o espaço de estados efetivo de cada objeto é sempre uma restrição do espaço de estados descrito pela classe. Este resultado, que mais adiante será formalizado, provê uma importante propriedade que nos permite analisar uma classe de objetos independentemente do contexto em que serão usados. É esta a característica fundamental que provê a base para criação de classes efetivamente reutilizáveis.

Análise e validação Pelo exposto acima, fica evidente que podemos analisar e validar cada classe de objetos de um modelo usando todas as técnicas convencionais de redes de Petri. Além disso, a integração semântica provê a base para a adaptação e construção de métodos de análise para modelos completos.

Em particular, interessam-nos as técnicas de análise baseadas na exploração do espaço de estados. No restante deste capítulo apresentamos as noções fundamentais que permitem a análise de espaço de estados dos modelos, bem como alguns resultados que os sustentam. Em particular, uma técnica bastante útil em algumas situações é a tradução. A idéia consiste em definir uma noção de equivalência entre modelos na notação proposta e redes de Petri puras. Em seguida, nos voltamos para a obtenção de uma rede equivalente a partir de um modelo dado. Embora não seja nosso objetivo explorar esse resultado na prática, ele serve também para demonstrar a equivalência de expressibilidade entre a notação proposta e redes de Petri convencionais.

6.2 Grafos de alcançabilidade

Informalmente, o espaço de estados de um sistema consiste nos possíveis estados que pode alcançar. Uma forma simples para representar espaços de estados é usar grafos. A idéia é que cada vértice do grafo represente um estado e que arco, o evento que faz o sistema passar de um estado ao outro. Tradicionalmente, na comunidade de redes de Petri, tais grafos são denominados *grafos de alcançabilidade* ou de *ocorrência*. A seguir apresentamos as definições básicas necessárias para o desenvolvimento do tema no que se refere à notação apresentada.

Definição 6.1 *Seja R um conjunto de rótulos. Um grafo orientado R -rotulado consiste em um conjunto V de vértices (ou nós) e um conjunto $A \subseteq V \times R \times V$ de arcos rotulados. Denotamos o grafo por $\langle V, A, R \rangle$ ou simplesmente $\langle V, A \rangle$, se o conjunto de rótulos é subentendido.*

Sistemas de objetos Podemos representar o comportamento de sistemas de objetos através de grafos de alcançabilidade. Observe, contudo, que é necessário um grafo para cada sistema de objetos. Não faz sentido definir grafos de alcançabilidade para modelos, mas sim para suas instâncias. Neste caso, vértices correspondem às configurações alcançáveis do sistema e os arcos aos eventos. A Figura 5.6 na página 105 pode ser considerado o grafo de alcançabilidade de uma instância do modelo do jantar dos filósofos, desde que entendamos que cada configuração alcançada é um vértice.

Definição 6.2 *Seja C_0 a configuração inicial de um sistema de objetos e E seu conjunto de eventos. Seu grafo de alcançabilidade, denotado por $G(C_0)$, é um grafo E -rotulado $\langle V, A \rangle$, onde*

$$V = \{ C_i \mid C_0 \vdash^* C_i \}$$

$$A = \{ \langle C_1, e, C_2 \rangle \in V \times E \times V \mid C_1 \vdash_e C_2 \}.$$

A definição acima é bastante intuitiva. Um sistema de objetos é representado por sua configuração inicial C_0 . O conjunto de vértices de $G(C_0)$ é definido como o conjunto das configurações C_i alcançáveis a partir de C_0 , ou seja, tais que $C_0 \vdash^* C_i$. Os arcos são definidos entre cada par de configurações C_1 e C_2 tais que C_2 seja imediatamente alcançável a partir de C_1 . Neste caso, o evento e é usado para rotular o arco.

Perceba que caminhos no grafo determinam seqüências de ocorrências no sistema—e vice-versa. É exatamente através da exploração dos caminhos do grafo de alcançabilidade que analisamos os sistemas de interesse. Diversas técnicas de análise e verificação são usadas para verificar as propriedades dos sistemas a partir do grafo de alcançabilidade. Uma das mais usadas e estudadas no momento é a conhecida por *verificação de modelos*¹. A idéia consiste em verificar se propriedades expressas em alguma linguagem de especificação (lógica temporal, por exemplo) são ou não válidas no grafo de alcançabilidade. Para funcionar bem, a aplicação desta técnica requer que os sistemas sejam finitos. Contudo, observe que no caso geral o grafo pode ter um número infinito de vértices—isto ocorre quando o número de configurações alcançáveis do sistema não é finito. Neste caso, fazem-se necessárias técnicas avançadas para representar de forma finita os espaços de estados de natureza infinita. O estudo de tais técnicas, bem como das estratégias específicas de verificação das propriedades, contudo, estão além do escopo deste trabalho. Aqui nos restringimos à definição dos espaços de estados e às relações entre os espaços dos sistemas e os especificados pelas classes.

Formalmente, se $G(C_0) = \langle V, A \rangle$, então pela Definição 6.2 na página anterior, valem as seguintes relações:

$$C_0 \vdash C_1 \iff C_1 \in V \quad (6.1)$$

$$C_1 \vdash_e C_2 \iff \langle C_1, e, C_2 \rangle \in A. \quad (6.2)$$

Comportamento efetivo de um objeto Às vezes é conveniente estudar o comportamento de um único objeto dentro de um sistema de objetos. Neste caso, convém representar seu comportamento através de um grafo de alcançabilidade. Para isto, usamos os estados alcançados pelo objeto como vértices e as ações que executa para rotular os arcos. (Relembre que, como foi dito no Capítulo 4, escrevemos o_{m_i} para representar o objeto o no estado m_i e $o_{m_i} \in C$, para indicar que o é vivo na configuração C e que m_i é seu estado interno.)

Definição 6.3 *Sejam o um objeto de um sistema com configuração inicial C_0 e R o conjunto das ações de o . O grafo de alcançabilidade de o é o grafo R -rotulado $G(o) = \langle V, A \rangle$*

¹Tradução de *Model checking*.

onde

$$V = \{ m_i \mid o_{m_i} \in C_j \text{ e } C_0 \vdash^* C_j \} \quad (6.3)$$

$$A = \{ \langle m_1, a, m_2 \rangle \in V \times R \times V \mid C_1 \vdash_e C_2, \quad o_{m_1} \in C_1, \quad o_{m_2} \in C_2, \quad o:a \in e \}. \quad (6.4)$$

Dizemos que $G(o)$ representa o comportamento efetivo de o no sistema determinado por C_0 .

Informalmente, a definição de $G(o)$ acima consiste em extrair de $G(C_0)$ a parte referente ao objeto o . Logo, se dispomos do grafo de alcançabilidade do sistema, digamos $G(C_0) = \langle V', A' \rangle$, então por (6.2) acima podemos reescrever (6.3) e (6.4) da seguinte forma equivalente:

$$V = \{ m_i \mid o_{m_i} \in C_j \text{ e } C_j \in V' \}$$

$$A = \{ \langle m_1, a, m_2 \rangle \in V \times R \times V \mid \langle C_1, e, C_2 \rangle \in A', \quad o_{m_1} \in C_1, \quad o_{m_2} \in C_2, \quad o:a \in e \}.$$

Dizemos que o grafo de alcançabilidade $G(o)$ representa o *comportamento efetivo* de o , porque seus vértices correspondem aos estados efetivamente alcançados pelo objeto durante seu ciclo-de-vida em um determinado sistema. Observe que o comportamento efetivo é determinado pelo *contexto* em que o objeto é usado—suas ligações e os demais objetos do sistema. Isto evidencia que objetos de uma mesma classe podem ter comportamentos efetivos diferentes.

Comportamento potencial Para complementar o conceito de comportamento efetivo, adotamos a idéia de comportamento *potencial* de um objeto. Informalmente, o comportamento potencial de um objeto corresponde a seu comportamento sem considerar um contexto específico. Formalmente, o comportamento *potencial* é determinado pela classe a que pertence. Na prática, propriedades sobre o comportamento dos objetos devem ser provadas sobre o comportamento potencial para garantir que serão válidas em todas as situações. Provar propriedades de um objeto sobre seu comportamento efetivo pode ser útil, contudo, por questões de custo. O grafo de alcançabilidade do comportamento potencial ou simplesmente o *grafo de alcançabilidade da classe* é determinado pelo comportamento expresso pelo seu corpo—a rede de Petri. A definição a seguir formaliza a representação do comportamento de uma rede de Petri através de um grafo de alcançabilidade.

Definição 6.4 *Seja N uma rede de Petri com (modos de) transições T' . O grafo de alcançabilidade de N , denotado por $G(N)$, é o grafo T' -rotulado $\langle V, A \rangle$, onde*

1. $V = [N]$
2. $A = \{ \langle m_1, t^a, m_2 \rangle \in V \times T' \times V \mid m_1[t^a]m_2 \text{ é um disparo de } N \}$.

A definição acima caracteriza o comportamento potencial dos objetos de uma classe. Se K é o corpo da classe \mathbb{C}_i , o comportamento descrito por \mathbb{C}_i é representado pelo grafo de alcançabilidade $G(K)$, como definido acima. Também dizemos que $G(K)$ é o comportamento potencial das instâncias de \mathbb{C}_i .

Comportamento interno e observável Dizemos que o grafo $G(K)$ definido acima é uma representação *interna* do comportamento dos objetos de \mathbb{C}_i porque seus arcos são rotulados com os nomes das próprias transições do corpo da classe. Por oposição, se usamos ações como rótulos, dizemos que o grafo representa o comportamento *observável* do objeto. A definição do comportamento efetivo de um objeto, por exemplo, usa ações como rótulos, logo $G(o)$ é uma representação do comportamento *observável* de o (ver Definição 6.3). Para efeitos de comparação, é relevante que ambas as descrições sejam rotuladas da mesma forma. A definição a seguir determina como obter uma representação do comportamento observável descrito por uma classe a partir de $G(K)$.

Definição 6.5 *Seja $G(K) = \langle V, A \rangle$ o grafo de alcançabilidade do corpo de \mathbb{C}_i e R o respectivo conjunto de ações. O comportamento observável descrito por \mathbb{C}_i é representado pelo grafo R -rotulado $G(\mathbb{C}_i) = \langle V', A' \rangle$, onde*

$$V' = [\mathbb{C}_i] = [K] = V$$

$$A' = \{ \langle m_1, \llbracket I(t) \rrbracket_a, m_2 \rangle \in V \times R \times V \mid \langle m_1, t^a, m_2 \rangle \in A \}.$$

Não é difícil perceber que o comportamento observável é uma abstração do comportamento interno. Os vértices de ambos os grafos são exatamente os mesmos. A diferença é que nem todo arco de $G(K)$ tem um correspondente em $G(\mathbb{C}_i)$. Isto é possível porque duas transições diferentes que causam a mesma mudança de estado em um objeto podem ter a mesma observação do ponto de vista externo em $G(\mathbb{C}_i)$.

6.3 Condição fundamental de análise

Diversas propriedades de um objeto podem ser verificadas diretamente na representação de seu comportamento efetivo—obtido a partir do contexto determinado pelo sistema em que é usado. Apesar disso, freqüentemente é conveniente generalizar certas propriedades, verificando que independem do contexto em que o objeto venha a ser usado. Isto é particularmente útil, como já dissemos, para o desenvolvimento de objetos reutilizáveis. Também é relevante durante o processo de especificação como forma contratual de especificações, permitindo diminuir o acoplamento e as dependências entre diferentes partes de um modelo. Neste caso, a abordagem de análise consiste em avaliar o comportamento descrito pela classe, verificando propriedades de suas instâncias. Os resultados dessa análise, contudo, só são válidos se a seguinte condição for verdadeira: que o comportamento efetivo de um objeto em qualquer sistema corresponde a uma restrição do comportamento potencial descrito por sua classe.

Embora a condição pareça natural, garanti-la tem sido um dos principais problemas observados na prática em relação à integração de redes de Petri e orientação a objetos. Relembre que, como discutido no Capítulo 2, diversas abordagens usam a fusão de nós como o mecanismo básico de interconexão entre as redes das classes. Isto é, a modelagem de um sistema a partir das classes consiste em determinar como os nós das redes devem ser fundidos. Em algumas, também é possível interconectar as redes pela adição de arcos (de redes de Petri) entre lugares e transições das redes das classes. Em ambos os casos, contudo, o resultado é o mesmo: as adições à estrutura podem alterar o comportamento previsto pelo projetista de forma incontrolável. Isto ocorre porque o espaço de estados de uma rede de Petri tanto pode ser reduzido ou aumentado pela livre adição de elementos a sua estrutura. Uma forma de amenizar o problema é garantir, por construção, que a interação ocorre dentro de padrões aceitáveis. Esta abordagem é usada, por exemplo, por Sibertin-Blanc ao definir as propriedades de *honestidade* e *discrição* das redes Cooperativas (ver Capítulo 2). O problema é que cabe ao projetista do modelo garantir que tenham as propriedades.

Interação por contrato Em nossa abordagem, a idéia central é que o projetista deve especificar como os *objetos* devem interagir e não como as *redes-classes* devem ser inter-

conectadas. A diferença conceitual permite fugir do esquema convencional, deixando de usar a fusão de nós como o mecanismo para “ligar” as redes que compõem o sistema. Além disso, a abordagem também preserva a regra de *esconder a informação* da orientação a objetos. A cada classe corresponde um *contrato* que determina as possíveis interações de seus objetos em termos dos estímulos que são capazes de receber e produzir. Logo, não há integração explícita das redes. Não cabe ao projetista dizer como as redes devem ser integradas. Cabe determinar como as interações ocorrem entre os objetos modelados. Poderíamos dizer que isto equivale a usar a definição do significado das inscrições de interação e das regras de interação para garantir que as redes sejam adequadamente interconectadas.

O cumprimento do contrato consiste em mapear os estímulos externos ao comportamento descrito pela rede de Petri. Os elementos da rede usados para isto são ditos *pontos de interação*. Observe que na notação proposta, apenas transições podem ser usadas como pontos de interação. A especificação da interação propriamente dita é feita através de *inscrições de interação*. Em princípio, qualquer elemento da rede poderia ser usado como ponto de interação. Contudo, isto equivale a repetir o erro das outras abordagens. Na prática, o uso de um lugar como ponto de interação significa que a porção de estado modelada pelo lugar está sendo “compartilhada” com outros objetos². Além de quebrar frontalmente o princípio de encapsulamento, o uso de lugares como pontos de interação dificulta a análise do comportamento da classe. Observe que para definir o comportamento potencial dos objetos a partir da rede, seria necessário considerar que a marcação de tais lugares pode ser alterada “espontaneamente” (possivelmente para valores imprevisíveis).

Por outro lado, é importante observar que restringir os pontos de interação às transições não significa diminuir o poder de expressão³. Objetos que permitam o compartilhamento de lugares podem ser descritos desde que transições adequadas sejam adicionadas, para modelar as formas de acesso. A diferença é que o projetista da classe é obrigado a prover a especificação da interação. O uso de transições como pontos de interação favorece a percepção do projetista da classe de que tem total controle do estado da rede e de que toda interação é mediada por um contrato—a inscrição de interação. Outra vantagem deste esquema é a total separação entre as redes das classes. É perfeitamente possível, por

²Isto equivale a permitir ligar arcos externos ou a definir fusões com lugares externos.

³Isto pode ser demonstrado por construção. Basta perceber que é possível reduzir todo tipo de interação imaginável ao proposto.

exemplo, modificar o corpo de uma classe sem modificar os contratos de utilização.

A proposição a seguir, formaliza a propriedade mencionada acima, comparando os grafos de alcançabilidade que representam o comportamento efetivo dos objetos ao comportamento descrito por suas classes.

Proposição 6.1 *Seja $o \in \llbracket \mathbb{C}_i \rrbracket$ uma instância de \mathbb{C}_i . O comportamento efetivo de o é uma restrição do comportamento descrito por \mathbb{C}_i , ou seja, $G(o)$ é subgrafo de $G(\mathbb{C}_i)$.*

Demonstração. Conseqüência direta das definições de $G(o)$ e $G(\mathbb{C}_i)$ e das regras que definem a relação \vdash . Sejam $G(o) = \langle V_1, A_1 \rangle$ e $G(\mathbb{C}_i) = \langle V_2, A_2 \rangle$. Todo vértice m_i de V_1 é um estado alcançável de o . Como os estados de um objeto são marcações do corpo de sua classe, então $m_i \in \llbracket \mathbb{C}_i \rrbracket = V_2$. Portanto, $V_1 \subseteq V_2$. Os arcos de A_1 correspondem às ações executadas pelo objeto durante o funcionamento do sistema. As ações potenciais do objeto são descritas pelas inscrições de interação do corpo de sua classe. Logo, por definição, para todo $\langle m_1, a, m_2 \rangle \in A_1$ há pelo menos uma transição t tal que $\llbracket I(t) \rrbracket = a$ (em algum modo), e, portanto, $\langle m_1, a, m_2 \rangle \subseteq A_2$, logo $A_1 \subseteq A_2$.

6.4 Rede equivalente

Na seção anterior definimos grafos de alcançabilidade como forma de representação do comportamento de redes de Petri, de objetos e de sistemas objetos. Na prática, gerar e representar grafos de alcançabilidade é um problema por si só que requer suporte computacional adequado. Para fins de experimentação, contudo, uma estratégia viável é usar ferramentas disponíveis para gerar espaços de estados de redes de Petri.

Uma forma bastante direta é construir uma rede de Petri cujo grafo de alcançabilidade seja “semelhante” ao grafo de alcançabilidade do sistema de interesse. Formalmente, partimos da definição de uma relação de equivalência entre sistemas de objetos e redes de Petri. Dizemos que uma rede de Petri e um sistema de objetos são equivalentes se seus grafos de alcançabilidade são isomórficos.

Definição 6.6 *Seja N uma rede de Petri e C_0 a configuração inicial de um sistema. Dizemos que a rede e o sistema expressam comportamentos equivalentes, denotado por $N \equiv C_0$, se seus grafos de alcançabilidade $G(N)$ e $G(C_0)$ são isomórficos.*

Do ponto de vista prático, a relação acima nos permite reduzir o problema de gerar o espaço de estados de um sistema de objetos à geração do grafo de alcançabilidade de uma rede equivalente. Desta forma, enquanto não houver ferramentas específicas para a notação proposta, podemos construir o espaço de estados de um sistema. O problema, portanto, é reduzido a obter uma rede equivalente a um dado sistema de objetos.

Uma forma de abordarmos o problema é focar redes equivalentes como “modelos” do sistema de objetos. Logo, uma forma simples de obter uma rede é identificar um método sistemático de construir esse “modelo”.

Em primeiro lugar, comparemos a natureza dos grafos de alcançabilidade de redes de Petri e de sistemas de objetos. Cada vértice de $G(N)$ é uma marcação. Por outro lado, vértices de $G(C_0)$ são configurações do sistema de objetos. Logo, é intuitivo que podemos construir uma rede equivalente em que cada marcação é a codificação de uma configuração do sistema de objetos. De forma análoga, devemos observar os arcos de cada grafo. No primeiro, os arcos são transições. No segundo, eventos. Logo, a rede equivalente deve modelar cada evento do sistema de objetos como uma transição.

Em segundo lugar, devemos recordar que o modelo do sistema de objetos consiste exatamente em um conjunto de redes de Petri. Logo, é intuitivo que uma rede equivalente pode ser construída a partir dessas redes. Naturalmente, a forma de composição deve ser derivada das regras de interação entre os objetos.

6.4.1 Algoritmo de construção de uma rede equivalente

Nesta seção apresentamos um algoritmo para a construção de uma rede de Petri equivalente a um dado modelo de sistemas de objetos. Para tornar a abordagem mais genérica, a saída do algoritmo é a estrutura da rede equivalente sem a marcação inicial. A marcação inicial deve ser adicionada à estrutura para refletir a configuração inicial de interesse.

O algoritmo é apresentado em pseudo-código. Em diversas linhas o código é suficientemente preciso para ser implementado diretamente. Em alguns casos, contudo, optamos por usar linguagem natural, com objetivo de evitar o detalhamento de operações que não adicionariam elementos relevantes ao algoritmo e que dependem das estruturas de dados utilizadas na implementação. Mesmo nesses casos, a especificação é suficientemente precisa e caracteriza as operações abstratas.

Organizamos o algoritmo em etapas. A idéia é que cada etapa seja suficientemente coesa para que possa ser avaliada e analisada independentemente das demais.

As convenções usadas são as mesmas adotadas no restante do texto. Apesar disso, é interessante registrá-las aqui. A entrada do algoritmo é o modelo denotado por M que consiste em n classes, especificamente $\mathbb{C}_1, \mathbb{C}_2, \dots, \mathbb{C}_n$. O corpo de cada classe é denotado por K_i , com $i = 1, \dots, n$. Os índices identificam a que classe cada elemento pertence. Assim, P_1 é o conjunto de lugares do corpo K_1 da classe \mathbb{C}_1 , enquanto que E_3 é a função de expressões de arcos do corpo K_3 da classe \mathbb{C}_3 , e assim por diante.

A rede que é construída pelo algoritmo, e que é sua saída, é denotada por K (sem nenhum índice). Da mesma forma, seus componentes são denotados da forma convencional, sem índices. Assim, T denota o conjunto de transições da rede equivalente K . Outras notações serão introduzidas ao longo da exposição do algoritmo.

Estratégia A idéia por trás do algoritmo é compor K a partir dos corpos das classes. Na fase inicial, as redes K_i são preparadas para a fusão nos aspectos em que podem ser tratadas independentemente. Isso inclui:

1. condições para que cada rede K_i possa modelar múltiplas instâncias da classe simultaneamente; e
2. complementação da rede para que modele explicitamente todo o ciclo de vida dos objetos, incluindo instanciação e auto-destruição.

A segunda fase consiste em interligar as redes K_i , complementado a rede K com os elementos e restrições necessários para satisfazer as regras de interação entre objetos. Isso inclui:

1. adição de lugares e arcos para modelagem de mensagens pendentes e do efeito das ações de saída assíncronas;
2. caracterização do efeito das ações de entrada, considerando tanto o caso de sincronizações quanto o de entradas assíncronas;
3. caracterização do efeito das ações de instanciação.

Para simplificar a notação, assumimos que o modelo M pode ser destruído pelo algoritmo. Isto nos permite tratar os elementos de M como objetos que pode ser diretamente manipulados e adicionados à rede K . Em geral, operações com elementos das redes são expressos com a notação de teoria de conjuntos. Nos casos em que as operações têm características especiais, usamos linguagem natural.

Etapa 1 Interpretadas independentemente contexto e da semântica de RPOO, cada rede K_i modela o comportamento de *uma única* instância de \mathbb{C}_i . É intuitivo que podemos usar K_i para modelar o comportamento de todas as instâncias da classe \mathbb{C}_i . Para isto, contudo, devemos fazer algumas alterações em cada rede.

Poderíamos pensar em replicar a estrutura para modelar múltiplas instâncias. Contudo, o mesmo efeito pode ser obtido usando a abordagem convencional de redes de alto nível: usar a estrutura uma única vez e codificar a informação referente aos objetos individuais nas fichas manipuladas pela rede. Para isto, é necessário fazer com que as fichas manipuladas pela rede identifiquem a instância específica a que se referem. Para isto, os conjuntos de cores dos lugares e as expressões dos arcos devem ser alterados correspondentemente.

A solução é simples: para cada tipo $D_j \in \mathcal{D}$ adicionamos o tipo $D_j \times N_i$ ao modelo (relembre que N_i é o domínio de identificadores de objetos da classe \mathbb{C}_i). Do ponto de vista sintático, isto corresponde a incluir as sortes referentes a esses tipos em Σ . Relembre que as redes K_i são definidas sobre a mesma assinatura Σ do modelo M e que, por convenção, \mathbb{S}_j é a sorte correspondente ao tipo D_j . Usamos $\mathbb{S}_{j,i}$ para denotar a sorte correspondente ao tipo $D_j \times N_i$. Abaixo a primeira etapa do algoritmo:

REDE-EQUIVALENTE(M)

- 1: **for** $\mathbb{C}_i \in M$ **do**
- 2: **for** $\mathbb{S}_j \in \text{Img}(C_i)$ **do**
- 3: $S \leftarrow S \cup \{\mathbb{S}_{j,i}\}$ (correspondendo a $D_j \times N_i$)

O significado das linhas acima é bastante claro. A única explicação necessária é em relação a $\text{Img}(C_i)$. A idéia é que a linha 3 seja executada para toda sorte \mathbb{S}_j que seja usada como inscrição do conjunto de cores de algum lugar da rede. Ou seja, tal que exista algum lugar $p \in P_i$ tal que $C_i(p) = \mathbb{S}_j$. Observe que \mathbb{S}_j pode ser qualquer sorte do modelo, incluindo o próprio \mathbb{C}_i .

Etapa 2 A segunda etapa do algoritmo consiste em fazer as mudanças nos conjuntos de cores e nas expressões de arcos, segundo a idéia apresentada. Observe que, pela notação escolhida, basta mudar cada anotação \mathbb{S}_j por $\mathbb{S}_{j,i}$ e cada expressão de arco da forma e pelo par $\langle e, o_i \rangle$, onde o_i é uma nova variável que deve ser adicionada ao modelo. Como a mesma variável o_i é usada em todas as expressões de arcos da subrede K_i , a ocorrência de qualquer transição corresponderá a uma ação da instância cujo identificador for ligado a o_i . O código correspondente é:

```

4: for  $\mathbb{C}_i \in M$  do
5:   for  $p \in P_i$  do
6:     if  $C_i(p) = \mathbb{S}_j$  then  $C_i(p) \leftarrow \mathbb{S}_{j,i}$ 
7:    $\mathcal{V}_{\mathbb{C}_i} \leftarrow \mathcal{V}_{\mathbb{C}_i} \uplus \{o_i\}$ 
8:   for  $f \in F_i$  do
9:      $E_i(f) \leftarrow \langle E_i(f), o_i \rangle$ 

```

As linhas 5 e 6 alteram as inscrições de conjuntos de cores dos lugares. A linha 7 adiciona as variáveis-objeto o_i . E as linhas 8 e 9 alteram as inscrições de arcos.

Etapa 3 A terceira etapa do algoritmo complementa cada rede K_i , com os elementos necessários para elementos necessários para modelar o ciclo de vida completo de cada objeto. Até este ponto, as alterações feitas em K_i não caracterizam precisamente o ciclo de vida dos objetos. Por exemplo, a semântica adotada determina que um objeto deve ser eliminado da configuração após a ocorrência de uma ação de auto-destruição. Contudo, isto não é respeitado pelas transições correspondentes nas redes K_i . Quando uma transição correspondente a uma ação de finalização ocorre, o objeto simplesmente não “morre”. Além da auto-destruição, é necessário introduzir a instanciação de objetos no modelo.

Uma solução simples é mostrada de forma esquemática na Figura 6.1. A idéia é compor K_i com uma rede que modela o ciclo de vida dos objetos—à esquerda na figura. A forma de compor as redes é ilustrada à direita da figura. A rede consiste na modelagem direta do ciclo de vida dos objetos: cada objeto pode ser instanciado (transição *cria*); enquanto estiver vivo (lugar *vivos*) pode executar ações (transição *acao*); se a ação for de auto-destruição, o objeto é eliminado do sistema (transição *fim*).

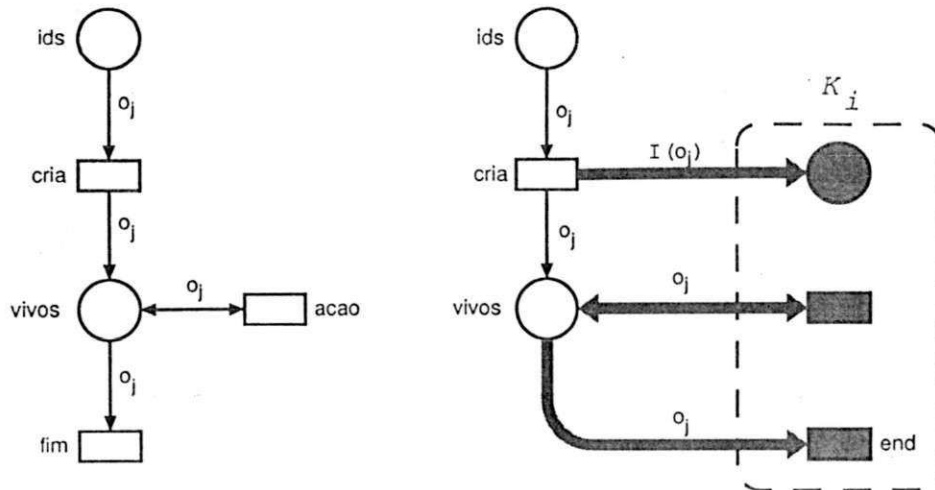


Figura 6.1: Representação esquemática da modelagem do ciclo de vida dos objetos

Para compor essa rede a K_i basta considerarmos os efeitos desejados. A transição *cria* deve instanciar um novo objeto. Isto consiste em marcar os lugares da rede K_i com as fichas correspondentes ao estado inicial do objeto. Enquanto o objeto estiver vivo, ele deve poder executar ações. Este efeito é obtido ligando todas as transições de K_i ao lugar *vivos*, através de arcos de teste. Se a transição modelar ações de auto-destruição o arco deve ser apenas de saída de *vivos*. Assim, após sua ocorrência, nenhuma outra transição referente ao objeto poderá ocorrer.

O código correspondente às transformações descritas acima é apresentado a seguir. Observe que as inscrições de auto-destruição “end” são eliminadas depois que a semântica correspondente é incorporada às redes.

```

10: for  $C_i \in M$  do
11:    $T_i \leftarrow T_i \uplus \{ \text{cria}_i \mid C_i \in M \}$ 
12:    $P_i \leftarrow P_i \uplus \{ \text{ids}_i, \text{vivos}_i \mid C_i \in M \}$ 
13:    $C_i(\text{ids}_i) \leftarrow C_i$ 
14:    $C_i(\text{vivos}_i) \leftarrow C_i$ 
15:    $F_i \leftarrow F_i \uplus \{ (\text{ids}_i, \text{cria}_i), (\text{cria}_i, \text{vivos}_i) \}$ 
16:    $E_i((\text{ids}_i, \text{cria}_i)) \leftarrow o_i$ 
17:    $E_i((\text{cria}_i, \text{vivos}_i)) \leftarrow o_i$ 
18:   for  $p \in P_i \setminus \{ \text{ids}_i, \text{vivos}_i \}$  do
19:      $F_i \leftarrow F_i \uplus \{ (\text{cria}_i, p) \}$ 

```

```

20:      $E_i((\text{cria}_i, p)) \leftarrow \langle I_i(p), o_i \rangle$ 
21:   for  $t \in T_i \setminus \{\text{cria}_i\}$  do
22:      $F_i \leftarrow F_i \uplus \{ (\text{vivos}_i, t) \}$ 
23:      $E_i((\text{vivos}_i, t)) \leftarrow o_i$ 
24:     if  $I(t)$  tem inscrição de auto-destruição "end"
25:       then remove a inscrição "end" de  $I(t)$ 
26:     else  $F_i \leftarrow F_i \uplus \{ (t, \text{vivos}_i) \}$ 
27:      $E_i((t, \text{vivos}_i)) \leftarrow o_i$ 

```

Etapa 4 Até este ponto, o algoritmo tratou todos os aspectos que podiam ser tratados em cada rede K_i independentemente. A partir deste ponto, os efeitos externos das ações dos objetos passam a ser considerados. Inicialmente, contudo, definimos explicitamente que a rede K consiste na união (disjunta) das redes K_i .

```

28:  $K \leftarrow K_1 \uplus K_2 \uplus \dots \uplus K_n$ 

```

Relembre que dissemos que o algoritmo "destrói" o modelo original. Isto significa que a partir deste ponto, se alterarmos qualquer elemento de K_i , estamos indiretamente modificando K . Na prática, a linha acima pode consistir em atribuir a cada componente de K a união disjunta dos componentes dos K_i .

Etapa 5 Nesta etapa, o algoritmo resolve inscrições de saídas assíncronas. Isto é feito adicionando lugares para comportar mensagens pendentes e ligando as transições que modelam saídas assíncronas a esses lugares para produzir as mensagens pendentes. O lugar que armazena mensagens do tipo S_k enviadas por objetos da classe C_i para objetos da classe C_j é denotado por $mp_j^i[k]$.

```

29: for  $C_i \in M$  do
30:   for  $t \in T_i$  do
31:     if  $I(t)$  tem inscrição de saída assíncrona " $o.m$ ", onde  $o \in \mathcal{T}_{C_i}$  e  $m \in \mathcal{T}_{S_k}$  then
32:        $P \leftarrow P \cup \{ mp_j^i[k] \}$ 
33:        $S \leftarrow S \cup \{ S_{i,k,j} \}$  (correspondendo a  $N_i \times D_k \times N_j$ )
34:        $C(mp_j^i[k]) \leftarrow S_{i,k,j}$ 
35:        $F \leftarrow F \cup \{ (t, mp_j^i[k]) \}$ 

```

- 36: $E(\langle t, mp_j^i[k] \rangle) \leftarrow E(\langle t, mp_j^i[k] \rangle) + \langle o_i, m, o \rangle$
 37: remove a inscrição "o.m" de $I(t)$

Etapa 5 Esta etapa complementa a anterior, resolvendo inscrições de entradas de dados. Observe, contudo, que as entradas de dados não especificam se a entrada é síncrona ou assíncrona. Por esta razão, o algoritmo deve substituir cada transição que modela uma entrada por duas: uma que resolve entradas assíncronas e outra para entradas síncronas. Somente as primeiras são ligadas aos lugares que armazenam mensagens pendentes. As que modelam entradas síncronas são "marcadas" para posterior tratamento.

- 38: for $\mathbb{C}_i \in M$ do
 39: while há $t \in T_i$, tal que $I(t)$ tem inscrição "o?m", onde $o \in \mathcal{T}_{\mathbb{C}_i}$ e $m \in \mathcal{T}_{S_k}$ do
 40: $P \leftarrow P \cup \{ mp_i^j[k] \}$
 41: $\mathcal{S} \leftarrow \mathcal{S} \cup \{ \mathcal{S}_{j,k,i} \}$ (correspondendo a $N_j \times D_k \times N_i$)
 42: $C(mp_i^j[k]) \leftarrow \mathcal{S}_{j,k,i}$
 43: $t_a \leftarrow$ cópia de t
 44: $t_s \leftarrow$ cópia de t
 45: remove t e adiciona t_a e t_s a T
 46: remove a inscrição "o?m" de $I(t_a)$
 47: $F \leftarrow F \cup \{ \langle mp_i^j[k], t_a \rangle \}$
 48: $E(\langle mp_i^j[k], t_a \rangle) \leftarrow \langle o, m, o_i \rangle$
 49: substitui a inscrição "o?m" de $I(t_s)$ por "o?!m"

Observe que, para simplificar a exposição do algoritmo, assumimos a existência de operações abstratas para manipular a estrutura da rede: a cópia, a eliminação e a adição de transições usadas nas linhas acima incluem a manipulação dos arcos incidentes e dos demais elementos que os decoram (expressões de arcos, guardas e inscrições de interação). Mais adiante, também usaremos também fusões de transição com essa semântica.

Etapa 6 Observe que as transições identificadas até este ponto do algoritmo modelam *ações potenciais* independentes dos objetos e seus respectivos efeitos. Cada rede K_i tem exatamente uma transição para cada ação potencial mais uma transição que modela a instanciação de novos objetos. A semântica de sistemas de objetos, contudo, define o

conceito de evento como um conjunto de ações em que todas as restrições de sincronização são respeitadas. A partir desta etapa, o algoritmo se resume a identificar eventos a partir das ações. Para evidenciar essa estratégia, o conjunto das transições é renomeado para A , indicando que se refere a ações. Desta forma, o conjunto T das transições de K , que devem modelar eventos, é reinicializado como o conjunto vazio. Em seguida, cada etapa avalia as ações e suas inscrições de interação a fim de identificar novos eventos, incluindo-os em T .

50: for $C_i \in M$ do

51: $A_i \leftarrow T_i$

52: $T_i \leftarrow \emptyset$

Etapa 7 Os eventos mais fáceis de identificar são os correspondentes a ações internas dos objetos, modelados por transições em A sem inscrições de interação. Também é importante observar que nenhum outro evento mínimo inclui tais ações. Logo, após incluir os eventos, podemos remover as ações do conjunto A .

53: for $t \in A \setminus \{ \text{cria}_i \mid i = 1, \dots, n \}$ do

54: if $I(t)$ não tem inscrição ($I(t) = \emptyset$) then

55: adiciona t a T

56: remove t de A

Etapa 8 Nesta etapa, o algoritmo resolve ações de instanciação. Observe que a semântica definida para essas ações determina que a ocorrência de uma ação de instanciação garante a criação do novo objeto e liga a variável da inscrição ao identificador do objeto instanciado. Este efeito pode ser obtido pela fusão das transições que modelam ações de instanciação (que têm inscrições do tipo " $v:C_j$ ") às transições cria_j . Naturalmente, isto é feito inscrição por inscrição, a fim de garantir que o efeito total seja obtido, em caso de criação simultânea de mais de um objeto.

57: for $C_j \in M$ do

58: while há $t \in A_i$ com inscrição " $v:C_j$ " $\in I(t)$ onde $v \in \mathcal{V}_{C_j}$ do

59: $t \leftarrow$ fusão de t com cópia de cria_j (união disjunta de variáveis)

60: $G(t_e) \leftarrow G(t_e) \cup \{ "v = o_j" \}$ (ou a variável usada)

61: remove a inscrição " $v:C_j$ " de $I(t)$

```

62:         if  $t$  não tem mais inscrições, ou seja  $I(t) = \emptyset$ 
63:             then adiciona  $t$  a  $T$ 
64:             else adiciona  $t$  a  $A_i$ 
65:  $A \leftarrow A \setminus \{ \text{cria}_i \mid i = 1, \dots, n \}$ 

```

A linha 57 acima funde a transição cria_j à transição t . Observe o comentário após a linha, indicando que a fusão deve ocorrer com o cuidado necessário de garantir que a união das variáveis deve ser disjunta. Na prática, isso significa que pode ser necessário adicionar e/ou renomear variáveis existentes. A ligação entre a variável v usada pelo projetista da classe e a variável o_j usada na transição cria_j é garantida através da adição da guarda (linha 58).

Etapa 9 Esta etapa do algoritmo resolve sincronizações. O processo é semelhante ao usado na etapa anterior. O algoritmo busca as ações complementares de cada ação de saída síncrona. Se uma ação é encontrada, uma nova transição é criada fundindo as transições originais. Se a nova transição não tiver mais restrições de interação, isto é, se não tiver mais inscrições, a transição modela um evento do sistema e é adicionada a T . Caso contrário é adicionada a A , para que em iterações seguintes do algoritmo a restrição possa ser resolvida. Depois que todas as possibilidades de sincronização são resolvidas para uma dada ação de saída, a transição original é removida de A .

```

66: while há  $t_1 \in A_i \subset A$  com inscrição " $o_1!m_1 \in I(t_1)$ ", onde  $o_1 \in \mathcal{T}_C$  e  $m_1 \in \mathcal{T}_{S_k}$  do
67:     for  $t_2 \in A_j$  do
68:         if  $t_2$  tem inscrição " $o_2?m_2 \in I(t_2)$ " onde  $o_2 \in \mathcal{T}_C$  e  $m_2 \in \mathcal{T}_{S_k}$  then
69:              $t_{1,2} \leftarrow$  fusão de cópias de  $t_1$  e  $t_2$  (união disjunta de variáveis)
70:              $G(t_{1,2}) \leftarrow G(t_{1,2}) \cup \{ "m_1 = m_2", "o_i = o_2", "o_j = o_1" \}$  (variáveis usadas)
71:             remove as inscrições " $o_1!m_1$ " e " $o_2?m_2$ " de  $I(t_{1,2})$ 
72:             if  $t_{1,2}$  não tem mais inscrições, ou seja  $I(t_{1,2}) = \emptyset$ 
73:                 then adiciona  $t_{1,2}$  a  $T$ 
74:                 else adiciona  $t_{1,2}$  a  $A_i$ 
75:             remove  $t_1$  de  $A_i$ 
76: return  $K$ 

```


6.4.2 Exemplo de aplicação do algoritmo

Para ilustrar o algoritmo descrito, demonstraremos sua aplicação ao modelo dos filósofos apresentado na Seção 5.3.1. Antes de aplicar o algoritmo é necessário obter o modelo detalhado do problema dos filósofos. Relembre que no modelo apresentado foram usadas algumas facilidades sintáticas para simplificar os diagramas. O modelo detalhado é mostrado na Figura 6.2. Nesse modelo, as associações são modeladas explicitamente por lugares que armazenam identificadores de objetos. Além disso, como o algoritmo foi construído a partir da definição formal, foi necessário integrar os nós de declaração das classes em um único nó global.

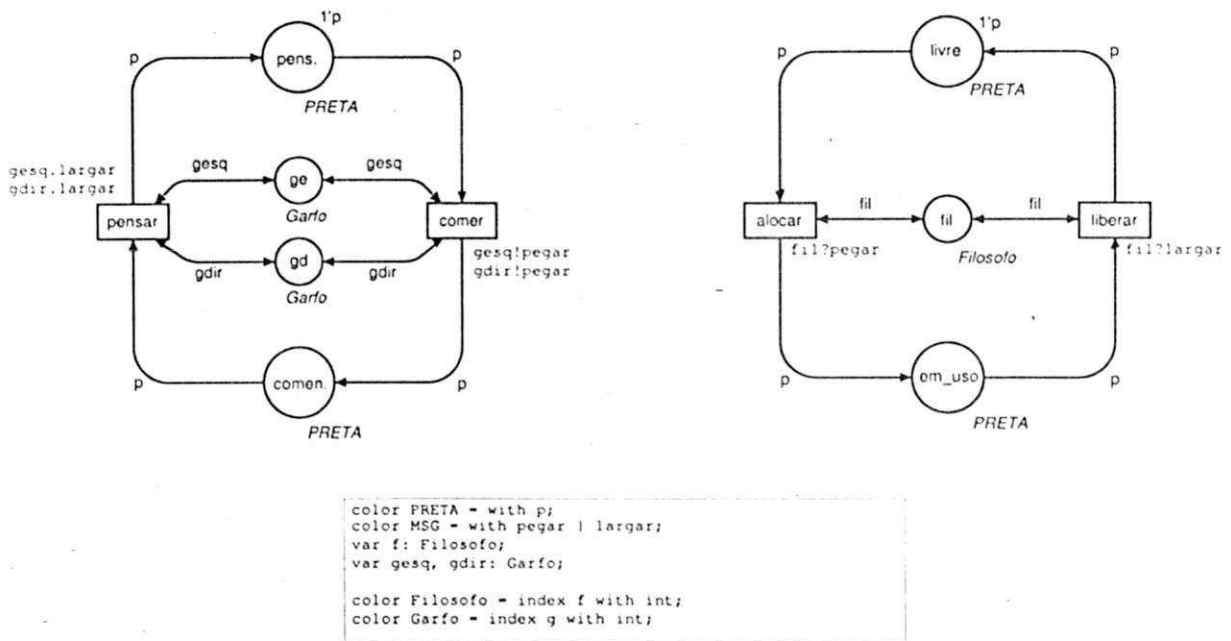


Figura 6.2: Modelo detalhado do problema dos filósofos

As etapas 1 e 2 do algoritmo preparam as redes do modelo para suportar múltiplas instâncias. Isso é feito equipando as fichas da rede com o identificador do objeto que está efetivamente executando as ações. A Figura 6.3 ilustra a rede após essas etapas. Observe que as mudanças correspondem à adição de tipos (por exemplo, os tipos `PRETA_Filosofo` e `PRETA_Garfo`) e das variáveis que serão usadas para identificar a instância a que se refere cada ação (variáveis `f1` e `g1`).

As etapas 3 e 4 do algoritmo completam as redes com a caracterização do ciclo de vida dos objetos. A rede resultante é ilustrada na Figura 6.4. Observe que no caso específico deste exemplo não há ações de criação ou de auto-destruição. Para tornar a figura

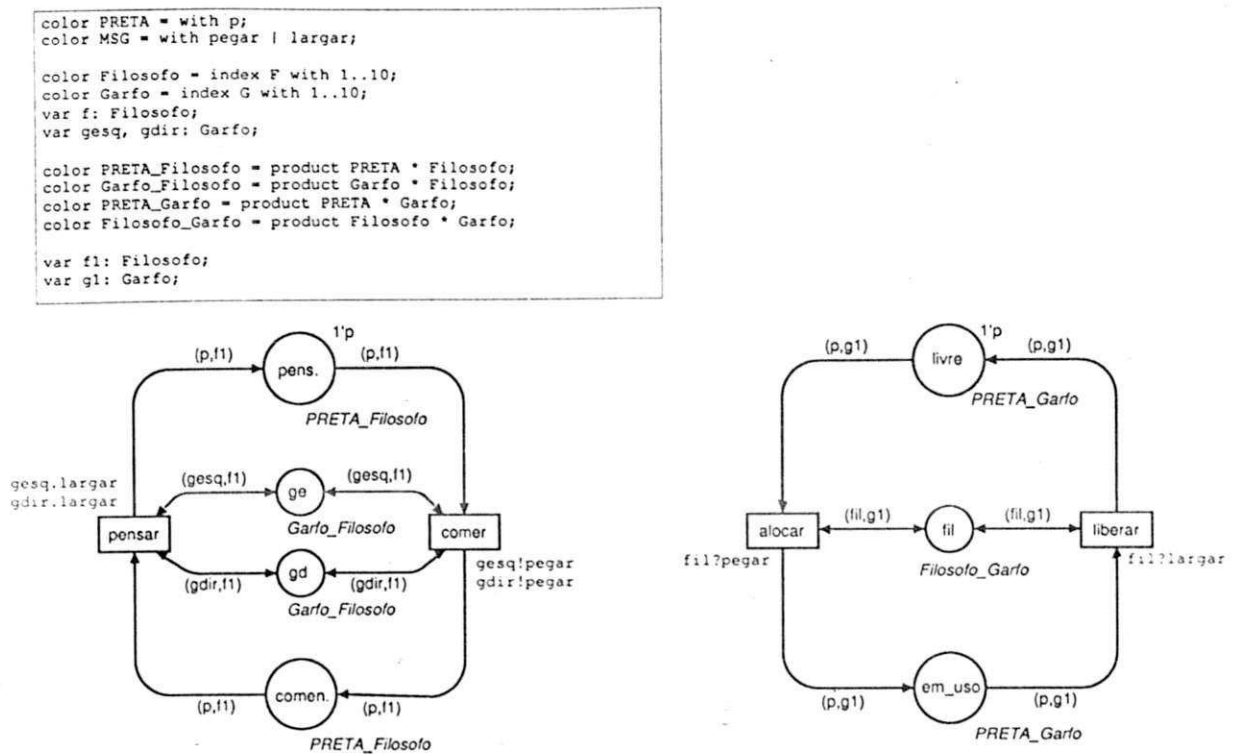


Figura 6.3: Rede equivalente ao modelo dos filósofos: após etapa 2 do algoritmo

mais legível, os arcos referentes ao ciclo de vida são diferenciados graficamente por linhas tracejadas em cinza.

As etapas de 5 a 9 integram as várias redes, resolvendo as inscrições de interação. Ao final do algoritmo, apenas 4 transições foram deixadas na rede equivalente, mostrada na Figura 6.5. Ou seja, apenas 4 eventos foram identificados a partir das ações especificadas pelas classes. Observe que as transições de instanciação *cria_F* e *cria_G* adicionadas na etapa 3 são eliminadas na etapa 8. Isto ocorre porque essas transições servem apenas como “moldes” para resolver inscrições de instanciação. Como no modelo não há ações de instanciação, tais transições não tiveram maior influência—observe que por essa razão os lugares *ids_F* e *ids_G* ficaram isolados do resto da rede.

Observe ainda que a comunicação assíncrona especificada para as ações *pensar* dos filósofos e *liberar* dos garfos foi resolvida ligando essas transições ao lugar denominado *mp* que modela mensagens pendentes do tipo *MSG*. Já a comunicação síncrona especificada para as ações *comer* e *alocar* foi resolvida pela fusão das duas transições—veja a transição *comer + alocar*. De fato, trata-se de duas fusões: primeiro o algoritmo funde *comer* com *alocar* para resolver a inscrição *gesq!pegar*. Em seguida, essa transição é novamente

```

color PRETA = with p;
color MSG = with pegar | largar;

color Filosofo = index F with 1..10;
color Garfo = index G with 1..10;
var f: Filosofo;
var gesq, gdir: Garfo;

color PRETA_Filosofo = product PRETA * Filosofo;
color Garfo_Filosofo = product Garfo * Filosofo;
color PRETA_Garfo = product PRETA * Garfo;
color Filosofo_Garfo = product Filosofo * Garfo;

var f1: Filosofo;
var g1: Garfo;

```

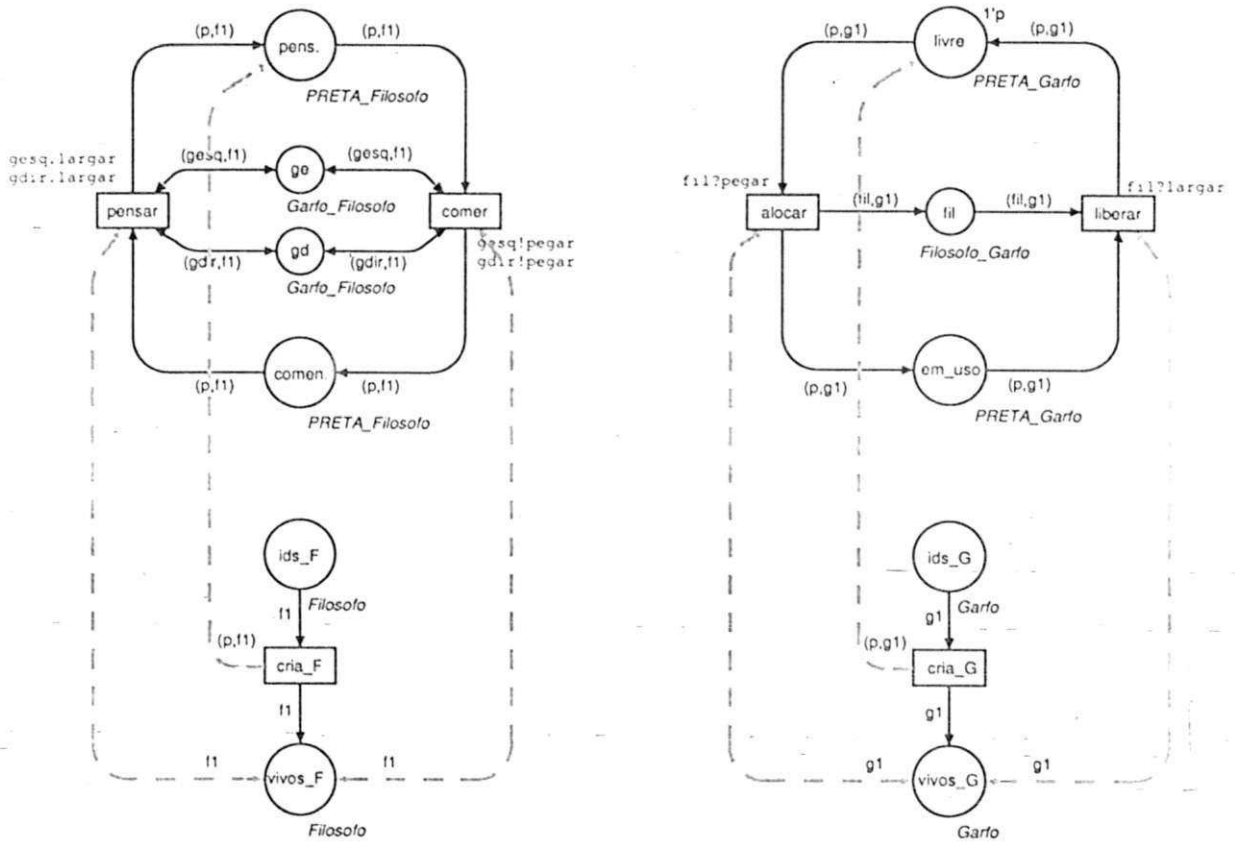
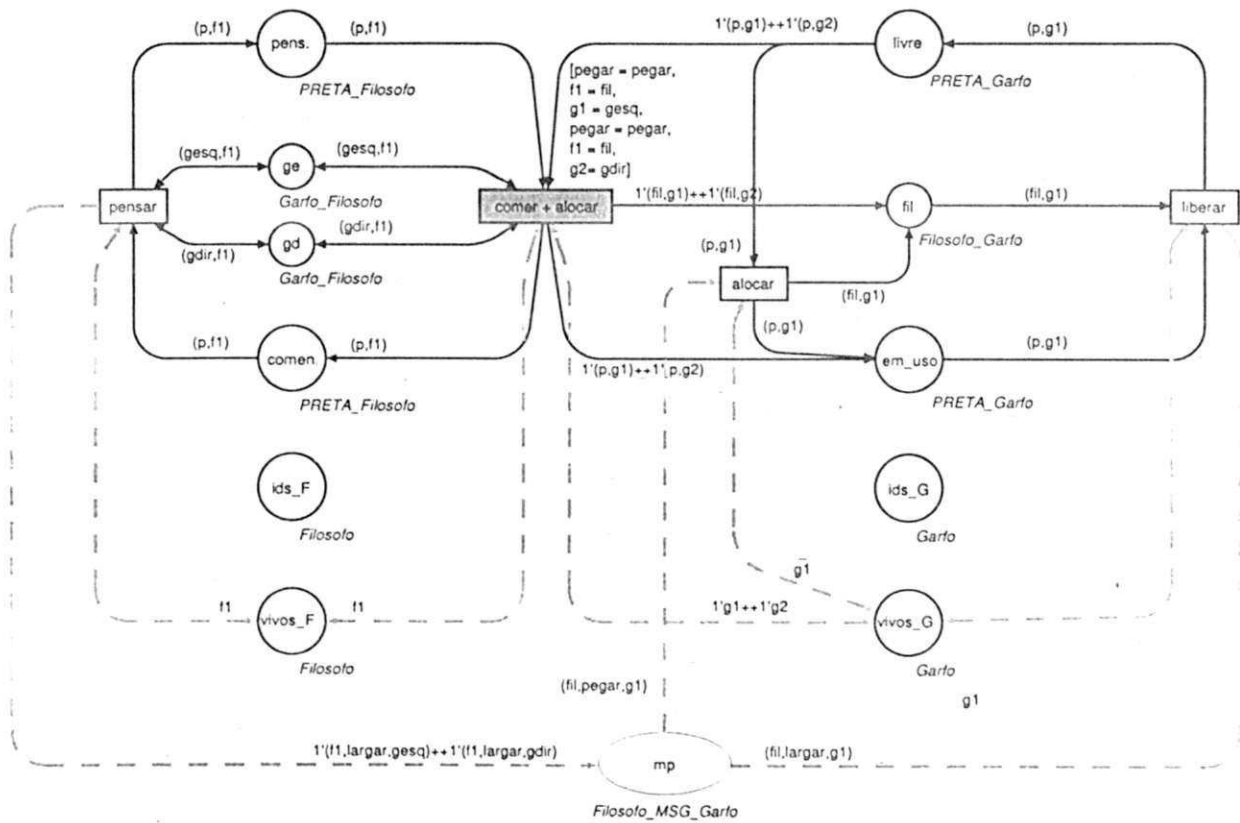


Figura 6.4: Rede equivalente ao modelo dos filósofos: após etapa 4 do algoritmo



```

color PRETA = with p;
color MSG = with pegar l largar;

color Filosofo = index F with 1..10;
color Garfo = index G with 1..10;

var fil: Filosofo;
var gesq, gdir: Garfo;

color PRETA_Filosofo = product PRETA * Filosofo;
color Garfo_Filosofo = product Garfo * Filosofo;
color PRETA_Garfo = product PRETA * Garfo;
color Filosofo_Garfo = product Filosofo * Garfo;
color Filosofo_MSG_Garfo = product Filosofo * MSG * Garfo;

var f1: Filosofo;
var g1, g2: Garfo;

```

Figura 6.5: Rede equivalente ao modelo dos filósofos

fundida com `alocar` para resolver a inscrição `gdir!pegar`. Isso pode ser observado pelas inscrições dos arcos e pela guarda da transição que parece duplicada em alguns aspectos—a variável `g2` foi incluída para garantir a união disjunta das variáveis na linha 69 do código.

Finalmente, observe a transição `alocar`. Essa transição é a versão assíncrona da ação. Ela foi incluída porque o algoritmo não verifica se há uma ação complementar que torne a transição útil na rede. De fato, a transição é morta, dado que nenhuma mensagem pendente com conteúdo `pegar` é gerada—basta ver os arcos de entrada de `mp`. De forma análoga, os lugares `vivos_F` e `vivos_G` também são inúteis na rede, dado que não há ações de instanciação ou destruição. Para facilitar o entendimento da rede equivalente, é conveniente considerar a versão reduzida ilustrada na Figura 6.6 em que os elementos aqui comentados foram eliminados.

Grafos de alcançabilidade A rede equivalente foi obtida através da aplicação do algoritmo apresentado. Embora a aplicação tenha sido feita manualmente, a edição e diversas operações básicas foram suportadas pela ferramenta *Design/CPN*. A rede obtida, portanto, nos permite utilizar o módulo de geração de grafos de ocorrência do *Design/CPN* para obter grafos de alcançabilidade de sistemas de objetos. A título de exemplo, marcamos a rede com a marcação equivalente à configuração inicial ilustrada na Figura 5.5 na página 104. Observe que a marcação inicial deve modelar o estado de cada objeto do sistema. Logo, a marcação inicial é:

$$m'_0(p) = \begin{cases} 1'(p, F(1)) + 1'(p, F(2)), & \text{se } p = \text{pensando} \\ 1'(p, G(1)) + 1'(p, G(2)), & \text{se } p = \text{livre} \\ 1'(G(2), F(1)) + 1'(G(1), F(2)), & \text{se } p = \text{ge} \\ 1'(G(1), F(1)) + 1'(G(2), F(2)), & \text{se } p = \text{gd} \\ 0, & \text{nos demais casos.} \end{cases}$$

Esta marcação é para a rede reduzida. Para a rede completa, contudo, basta adicionar os identificadores dos objetos vivos nos lugares `vivos_F` e `vivos_G`. Observe que os identificadores de filósofos e garfos são denotados por $F(i)$ e $G(i)$, respectivamente.

O grafo de alcançabilidade obtido através da ferramenta para a marcação m_0 é mostrado na Figura 6.7. Embora o detalhamento de cada marcação não seja mostrado, a topologia

se. Estas condições promovem efetivamente os princípios de abstração, encapsulamento e modularidade, garantindo condições reais para o desenvolvimento de modelos formais reutilizáveis.

Um resultado particularmente prático apresentado neste capítulo é o algoritmo para a obtenção de uma rede equivalente a um dado modelo. Na prática, esse algoritmo permite o uso imediato da notação para a construção de modelos em um estilo orientado a objetos, mesmo sem que haja ferramentas específicas. A notação pode ser vista como uma macro-linguagem para a construção de uma rede de Petri. O espaço de estados do modelo pode ser gerado e analisado da forma convencional, embora a interpretação dos resultados deva ser cuidadosa para que seja adaptada ao estilo OO. Do ponto de vista formal, contudo, o algoritmo oferece um objeto de estudo que permite a compreensão da relação semântica dos dois modelos, servindo de base para a adaptação de métodos de análise existentes.

Capítulo 7

Conclusão

Ao longo deste trabalho, discutimos a integração de conceitos da orientação a objetos e de redes de Petri. Inicialmente, discutimos e avaliamos as principais abordagens de integração existentes. Em seguida, identificamos os principais problemas e as características de uma solução adequada. Os demais capítulos desta tese apresentam uma nova formalização e notação de redes de Petri orientadas a objetos, desenvolvida segundo as características inicialmente identificadas. O significado dos modelos na notação proposta é obtido pela composição dos significados das duas visões do sistema.

Diferente das abordagens existentes, a notação foi desenvolvida com o objetivo de minimizar o impacto de uma visão sobre a outra em relação às sintaxes. O resultado é uma notação em que as duas visões são independentes quanto ao estilo e sintaxe. Apenas o significado é efetivamente composto. A independência é tal que permite modularizar a construção de um modelo e delegar responsabilidades a indivíduos da equipe que não necessariamente sejam especialistas em ambas as notações. É possível, por exemplo, requerer a modelagem de uma classe de objetos usando como contrato uma especificação que dá liberdade ao projetista quanto à forma de modelagem interna através de redes de Petri. O formalismo permite que tanto a especificação quanto o cumprimento do contrato independam das redes de Petri. Este forte encapsulamento permite que o modelo interno de cada classe seja uma rede de Petri convencional com estrutura e significados também convencionais, maximizando as condições de aceitação da notação. A integração efetiva só ocorre no nível semântico.

O modelo semântico utilizado, denominado de sistemas de objetos, não é um modelo

computacional completo. De fato, fluxo e computação são tratados de uma forma suficientemente abstrata para permitir o uso de qualquer modelo computacional completo em substituição. A ênfase no formalismo é a caracterização da evolução topológica dos sistemas tipicamente distribuídos e concorrentes. O formalismo permite a distinção das ações que afetam a topologia e das ações internas dos objetos que têm efeito apenas sobre seus estados internos. Esta separação é a chave para articular a organização orientada a objetos do modelo e as redes de Petri.

Um modelo na notação proposta é organizado da forma convencional da orientação a objetos (classes, associações, hierarquias de herança, agregações, pacotes, etc). Por outro lado, o comportamento dos objetos, incluindo fluxo e computação, é expresso através de redes de Petri. O significado dos modelos, contudo, é obtido pela articulação dos significados das redes, segundo as regras impostas no modelo. A articulação entre as diversas redes, entre os diferentes níveis de abstração envolvidos no modelo, bem como a especificação de contratos entre a organização macro do modelo e as redes de Petri é especificada através de contratos formais que cada classe deve respeitar. O formalismo proposto permite que esses contratos expressem o comportamento esperado observável de cada classe—geralmente enfatizando mais as ações que afetam a topologia do sistema do que os efeitos internos esperados. O aspecto mais importante a observar é que mesmo considerando as estritas restrições contratuais a que fica obrigado o projetista de uma dada classe ou parte do modelo, há total liberdade para o projetista da classe quanto à forma de estruturar a rede de Petri. O cumprimento do contrato é proporcionado pelo projetista de uma rede através de inscrições de interação com que a rede deve ser *decorada*. O termo *decorada* é especialmente verdadeiro, dado que tais inscrições em nada afetam a estrutura ou a interpretação da rede de Petri quando considerada isoladamente.

A propriedade acima enfatizada como forma de promover o encapsulamento no modelo também tem um importante papel na possibilidade de adaptar métodos de análise existentes. No Capítulo 6 demonstramos que a semântica adotada proporciona uma forte relação entre o comportamento descrito e o comportamento efetivamente observável de um objeto em qualquer contexto de uso que satisfaça as restrições contratuais: o comportamento efetivo de qualquer objeto descrito através da notação apresentada é necessariamente uma restrição do comportamento especificado. Esta relação permite que diversas questões de análise sejam abordadas de forma genérica para toda uma classes de objetos, sem que seja

necessário investigar cada contexto em que o objeto pode ser usado. É importante observar que esta propriedade não é observada nas demais notações existentes¹.

Em resumo, a notação proposta permite efetivamente a modelagem através da articulação ortogonal de duas visões, como proposto no Capítulo 3. Além disso, a caracterização semântica adotada e a relação entre o comportamento efetivo dos objetos e seu comportamento especificado garantem a existência de condições mais adequadas para a adaptação dos métodos originais de análise de redes de petri sem que seja necessário renunciar às facilidades oferecidas pela orientação a objetos.

7.1 Contribuições

Apesar da ênfase dada ao longo do trabalho, é importante ressaltarmos que não consideramos que a principal contribuição seja a notação propriamente dita, mas sim a abordagem usada para defini-la. A maioria das outras abordagens foi elaborada a partir do propósito de *definir uma extensão orientada a objetos para redes de Petri*. Contudo, vimos que esta forma de atacar o problema reduz as possibilidades de integração. Neste trabalho, definimos como propósito *a elaboração de uma notação orientada a objetos e baseada em redes de Petri para a modelagem de sistemas distribuídos e concorrentes de software*. A sutil diferença de ponto de vista foi determinante para a solução encontrada. Foi a partir da declaração de propósitos que percebemos que o verdadeiro foco não são as facilidades sintáticas da orientação a objetos ou as de redes de Petri, mas sim a classe de sistemas de interesse.

Definido o propósito da integração, voltamos nossa atenção para a forma de execução. Em primeiro lugar, reconhecemos a importância da escolha criteriosa de conceitos e noções que determinam o modelo computacional subjacente à notação. Esse conjunto de conceitos e princípios caracterizam a natureza dos sistemas de interesse—é o que chamamos de *modelo conceitual* da notação.

Com base no modelo conceitual, a avaliação do propósito de cada elemento da notação é simplificada. A regra é evidente: se não há propósito claro para um elemento na notação,

¹Algumas abordagens conseguem propriedades semelhantes, à custa de obrigações estruturais e comportamentais (nem sempre verificáveis) a que os projetistas das classes ficam obrigados—ver objetos cooperativos na página 21.

então não deve ser incluído. De fato, após a declaração do modelo conceitual a notação experimental que estava em desenvolvimento foi radicalmente simplificada. Contudo, ainda faltava determinar o segundo nível da estrutura sobre a qual a notação foi definida: o modelo semântico. O modelo conceitual determina apenas um conjunto de diretrizes e noções que podiam ou não ser seguidas, dado que não é expresso formalmente. Até esse ponto o significado dos modelos era expresso em termos de uma generalização do conceito de marcação e da regra de disparo, de forma semelhante à utilizada nas outras abordagens.

Dois fatores principais motivaram a definição formal de sistemas de objetos: a dificuldade de adaptação dos métodos de análise existentes a partir das noções semânticas até então utilizadas e a necessidade de melhor comunicação entre os participantes do projeto. Tornava-se necessária uma representação formal, compacta e próxima aos conceitos definidos no modelo conceitual. O desenvolvimento do formalismo de sistemas de objetos nos permitiu expressar os aspectos relevantes do comportamento de objetos de forma *ortogonal* ao comportamento interno descrito pelas redes de Petri. Isto é, de tal forma que uma caracterização não interferisse com a outra.

Devido à separação da semântica do modelo em camadas, podemos alterar facilmente a notação obtida. Os aspectos puramente notacionais podem ser completamente modificados sem alterar em nada as definições básicas—chamadas ao longo do trabalho de *sintaxe abstrata*. Da mesma forma, é possível alterar as redes de Petri e a notação abstrata sem alterar o modelo conceitual ou o comportamento de sistemas de objetos. Podemos, por exemplo, facilmente investigar variações da notação em que pequenas mudanças no comportamento dos sistemas de objetos sejam necessárias. Por exemplo, para obter uma notação em que não haja a comunicação síncrona entre objetos, basta eliminar as regras que determinam ocorrências síncronas da definição de ocorrência de sistemas de objetos e as respectivas inscrições.

A efetiva ortogonalidade das duas visões tem sido colocada à prova em diversos experimentos de modelagem que têm sido desenvolvidos (um deles é apresentado no Apêndice A). Na prática, o aprendizado da notação tem se mostrado bastante fácil para projetistas fluentes nos paradigmas originais. Atribuímos essa facilidade à forte independência das duas visões. Além disso, a modelagem decomposta em objetos permite a efetiva separação de responsabilidades pelos módulos do modelo como é preconizado pela orientação a objetos. Também é conveniente a separação da realização de uma classe e a caracterização de sua

interface contratual em termos das mensagens que suas instâncias devem ser capazes de atender. Pela mesma razão, a realização através de uma rede de Petri efetivamente independente das demais do modelo permite ao projetista o controle total da rede construída.

Problemas enfrentados Uma má estratégia de pesquisa foi a decisão de não dedicar esforços em duas atividades que teriam sido extremamente importantes para melhorar e antecipar os resultados: a aplicação das notações experimentais em casos de estudo mais realistas e o desenvolvimento de pequenas ferramentas para experimentar e validar os conceitos e idéias produzidas ao longo do processo. Apesar disso, algumas tentativas foram feitas para automatizar alguns processos. O resultado disso é uma versão textual simplificada para modelos e algumas ferramentas de verificação sintática.

7.2 Trabalhos futuros

Um desdobramento natural para os trabalhos é a aplicação da notação a casos de maior porte do que os tratados durante o projeto. Também é fundamental a construção de ferramentas de apoio à modelagem e validação dos modelos. Nesse sentido, há duas possibilidades que estão sendo exploradas: a construção de ferramentas específicas que validem os conceitos e idéias definidas e a elaboração de bibliotecas de suporte à edição e análise de modelos sobre a ferramenta Design/CPN. Há dois projetos no momento sendo desenvolvidos por nosso grupo para o suporte a modelos orientados a objetos sobre o Design/CPN. Um tem o objetivo de elaborar um algoritmo de geração de uma rede pura equivalente para permitir a geração do espaço de estados. O outro tem o objetivo de prover um simulador interativo para sistemas de objetos.

Outro trabalho relevante que deve ser realizado é a completa integração do conceito de tipos à definição. Neste sentido, um dos desafios é integrar as hierarquias de tipos de objetos e tipos de dados e determinar a relação entre tipos e classes de objetos. Também há, nesse sentido, um trabalho em desenvolvimento em nosso grupo com esse objetivo.

A aplicação prática da notação a diversos casos de modelagem também deve motivar trabalhos de comparação entre a decomposição orientada a objetos proposta à decomposição hierárquica convencional usada com redes de Petri. Além disso, a construção de modelos é sempre justificada pela possibilidade de analisar os sistemas modelados. Logo,

é natural que à medida que novos modelos sejam desenvolvidos, as técnicas e ferramentas de análise e validação sejam gradualmente adaptadas e desenvolvidas:

Nesse sentido, um dos caminhos mais promissores é o desenvolvimento de ferramentas de verificação de modelos (*model-checkers*). Por um lado há as propriedades de interesse convencionalmente investigadas em sistemas concorrentes: vivacidade, limitação, etc. Isto inclui a *deadlocks*, *livelocks* e problemas convencionais de alcançabilidade. Do outro, a separação explícita do estado do sistema em três níveis (topologia das configurações + controle e fluxo da computação nos objetos + dados) certamente motivará a especificação de propriedades verificáveis nesses três diferentes níveis. Isto requer que as linguagens de especificação permitam formular e articular propriedades nesses três níveis. Além disso, os métodos de análise podem explorar essa separação como uma forma de decompor o problema.

Finalmente, há a continuação do desenvolvimento da notação propriamente dita. Nesse sentido há diversas possibilidades. Em primeiro lugar, é necessário investigar melhor as possibilidades de concretização da sintaxe. Isto é relevante porque na prática é a notação concreta que é utilizada. Facilidades reais incluem melhorar a notação das inscrições de interação para aproximá-las das notações mais usadas para objetos e incluir facilidades sintáticas para facilitar a expressão. Um segundo ponto de interesse é o modelo semântico. Diversas escolhas se mostraram fracas. As ligações bidirecionais, por exemplo, certamente serão substituídas por unidirecionais. Esta forma mais simples enfatiza a idéia de que referências são armazenadas por um único objeto. Além disso, duas ligações unidirecionais podem modelar facilmente as ligações bidirecionais. Outro aspecto que deve ser revisto é a escolha das ações fundamentais. Ações de auto-destruição, por exemplo, podem ser facilmente codificadas. Logo, é importante avaliar como sua eliminação do conjunto de ações pode afetar as regras e a implementação de ferramentas.

Apêndice A

Um estudo de caso: modelagem do protocolo *Bouncer*

Neste apêndice, apresentamos um modelo desenvolvido com o objetivo de experimentar e avaliar o uso prático da notação introduzida. O sistema modelado é um gerenciador distribuído de licenças de software denominado *bouncer*. Os requisitos de modelagem foram obtidos da especificação original do sistema e dos respectivos protocolos, registrados em [BBF97; BBFpBaM97].

A notação utilizada nos modelos corresponde à definida na Seção 5.3. Algumas convenções gráficas foram introduzidas especificamente para o modelo. É o caso, por exemplo, do uso de arcos inibidores. Observe que modificações desta natureza podem ser introduzidas alterando apenas os conceitos no nível de sintaxe concreta (ver Seção 3.2 na página 39).

A.1 O Bouncer

O *bouncer* é um sistema distribuído para o controle do uso de licenças de software em ambientes de redes locais. O sistema é realizado através de uma interface de aplicação que disponibiliza um conjunto de primitivas de configuração e uso para que as aplicações se beneficiem dos serviços [BBF97].

O serviço *bouncer* é estruturado da seguinte forma: as aplicações clientes emitem requisições e liberações de licenças a servidores locais, usando um protocolo cliente-servidor. Em cada máquina da rede em que há clientes, há também um servidor em funcionamento—os

servidores são denominados *bouncers*. Para garantir a consistência entre os dados armazenados, os diversos servidores se comunicam através de um protocolo ponto-a-ponto.

Para minimizar a manutenção, os bouncers não devem funcionar como *daemons*. Eles são executados sob demanda pelo primeiro cliente do serviço de licenças em cada máquina. Logo, no estado inicial do sistema, em que não há licenças em uso, não há nenhum bouncer em funcionamento.

Para atender a requisições, os bouncers consultam dados locais sobre as licenças concedidas. Logo, a primeira ação de um bouncer ao ser iniciado é notificar os demais bouncers sobre sua entrada no sistema e atualizar seus dados sobre as licenças concedidas. A partir daí, o atendimento às requisições e liberações de licenças deve garantir a consistência dos dados entre os bouncers. Para simplificar a solução deste problema, o bouncer é projetado para funcionar sobre um subsistema de comunicação de grupo com garantias de atomicidade e ordenação das mensagens. Devido à organização como grupo, os bouncers mantêm-se consistentes porque efetuam as mesmas alterações nos dados e na mesma ordem.

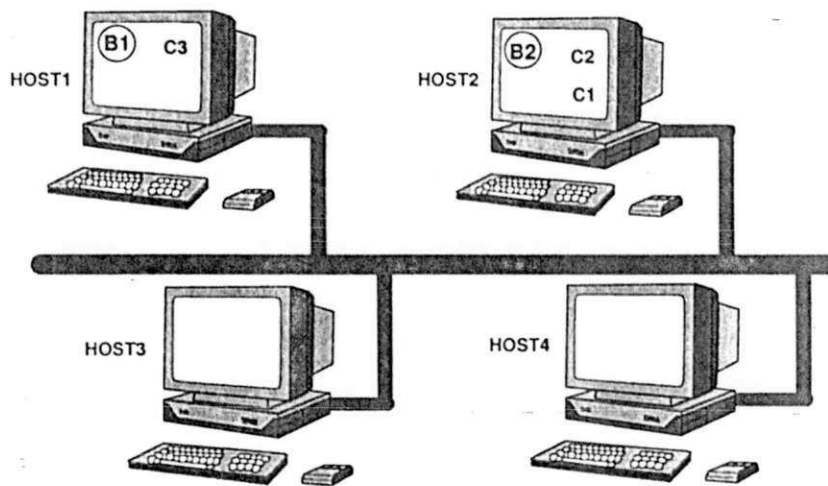


Figura A.1: Cenário Típico

A Figura A.1 ilustra um cenário simples de uso do sistema. A infra-estrutura consiste em uma rede local com 4 máquinas, identificadas por HOST1, HOST2, HOST3 e HOST4. Os círculos em cada máquina representam processos em execução—os processos C1, C2 e C3 são aplicações clientes e B1 e B2 são os servidores bouncer. Observe que em ambas as máquinas onde há uma aplicação cliente, há um bouncer executando. Analogamente, em HOST3 e HOST4 não há bouncers executando, porque não existem clientes locais.

A.2 Diagrama de Classes

O diagrama de classes do sistema é apresentado na Figura A.2. Quatro classes foram identificadas e nomeadas CLIENT, HOST, BOUNCER e GC_SERVICE.

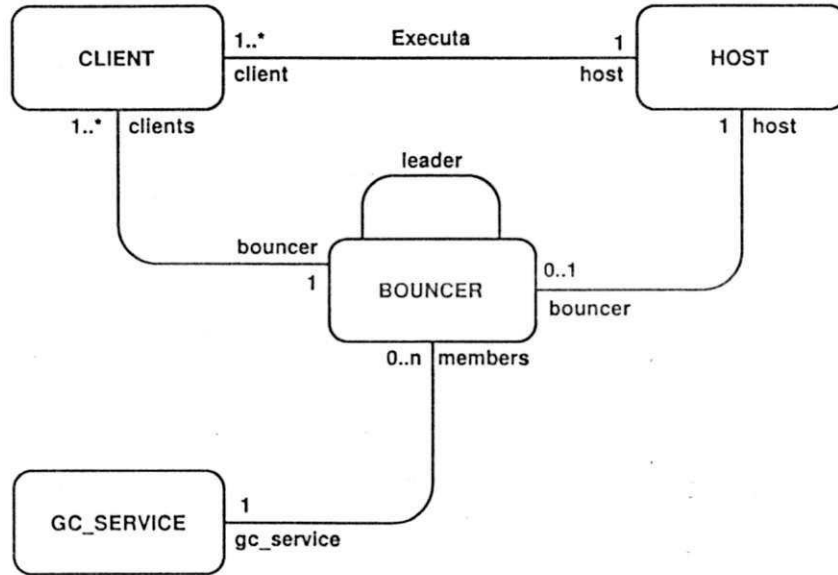


Figura A.2: Diagrama de classes

Claramente, as classes CLIENT e BOUNCER modelam as entidades ativas do sistema bouncer: os processos clientes e servidores. A classe HOST, que modela as máquinas da rede, foi considerada explicitamente no modelo por duas razões:

1. para enfatizar a localidade de execução.

Há vários bouncers no sistema, mas cada cliente só deve se comunicar com o bouncer que executa no mesmo host. Modelar os hosts como objetos facilita a representação dessas relações e restrições.

2. porque as instâncias da classe HOST têm funcionalidade própria relevante no sistema.

Há todo um conjunto de comportamentos ou funcionalidades que se não forem atribuídas aos objetos dessa classe, devem ser forjadas em outros objetos. Este é o caso, por exemplo, da manutenção do estado dos processos e da associação de um cliente a seu respectivo servidor bouncer.

Finalmente, a classe GC_SERVICE é utilizada para modelar o subsistema de comunicação de grupo necessário para o funcionamento do serviço.

A.3 Detalhamento das Classes

A.3.1 Classe CLIENT

O corpo da classe CLIENT é mostrado na Figura A.3 e consiste numa rede com cinco lugares, identificados pelos nomes `Init_request_process`, `Wait_init_bouncer`, `Bouncer_running`, `Waiting_license` e `License_in_use`, além das sete transições identificadas de T1 a T7. Relembre que a marcação inicial da rede determina o estado inicial dos objetos da classe. Logo, o estado inicial dos objetos que representam aplicações clientes é modelado pela marcação contendo uma única ficha com valor `p` no lugar `Init_request_process`. Esta marcação representa o cliente no estado inicial de seu funcionamento, antes mesmo de identificar o servidor bouncer correspondente.

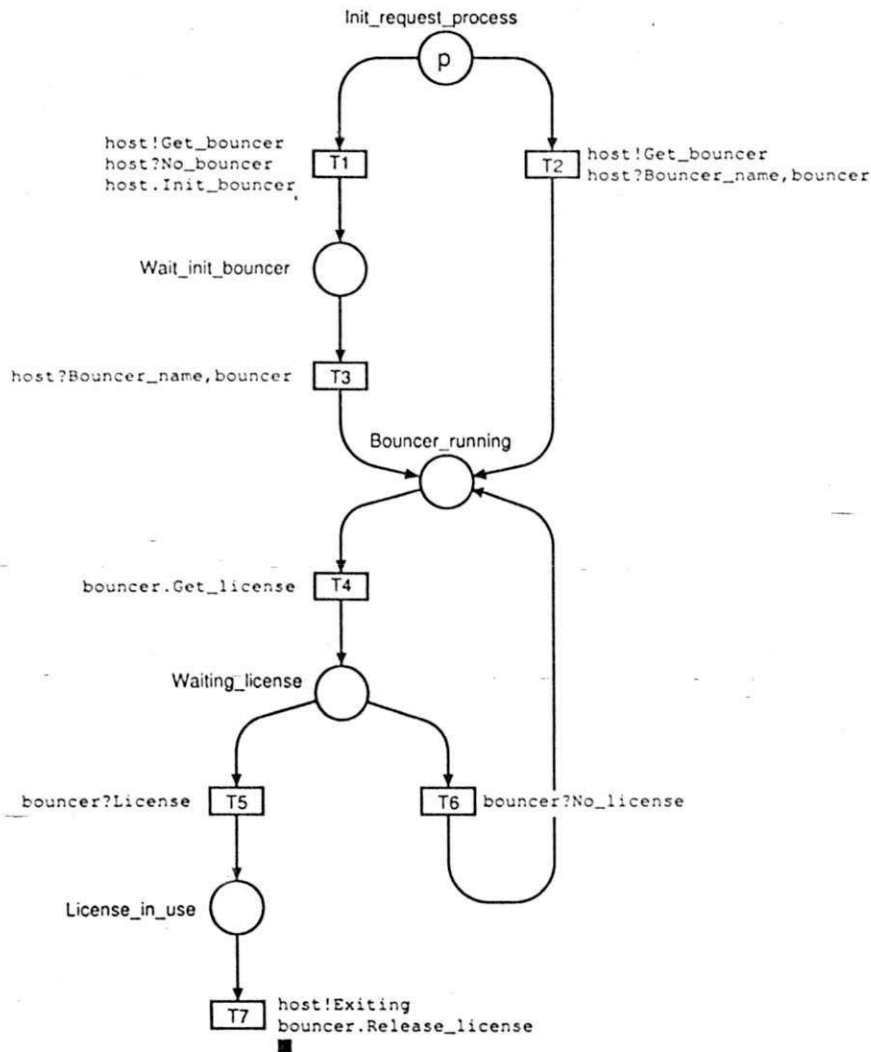


Figura A.3: Classe CLIENT

A partir da marcação inicial, apenas as transições T1 e T2 estão localmente habilitadas. A transição T2 ocorre apenas se já há um bouncer em funcionamento na máquina em que o cliente está executando. Isto é indicado pelas inscrições `host!Get_bouncer` e `host?Bouncer_name, bouncer`. Observe que a transição deve ocorrer sincronamente com a recepção da mensagem `Get_bouncer` pelo objeto indicado por `host`. A segunda inscrição obriga a fazer uma entrada de dados correspondente à indicação da referência do bouncer correspondente. Como as duas inscrições foram colocadas em uma única transição, é necessário que o objeto `host` responda à requisição de forma síncrona. Isto é, ele deve receber a mensagem e retornar o resultado em uma única ação (confira transição T4 na Figura A.5 na página 156).

Por outro lado, a transição T1 ocorre se não há um servidor bouncer em funcionamento na máquina em que o cliente está executando. Isto é indicado pelas inscrições de interação `host!Get_bouncer, host?No_bouncer` e `host.Init_bouncer`. A primeira inscrição é idêntica à primeira de T2 e modela a procura pelo servidor na máquina `host`. A segunda inscrição, contudo, indica que a máquina respondeu com a mensagem `No_bouncer`, o que indica que não há servidor disponível. A terceira inscrição representa um pedido de inicialização do servidor à máquina. Observe que neste caso, optamos por uma requisição assíncrona, o que obriga o cliente a esperar até a recepção da mensagem `Bouncer_name` (ver lugar `Wait_init_bouncer` e transição T3).

Os dois caminhos de execução levam a uma marcação em que uma ficha é depositada no lugar `Bouncer_running`. Observe que a partir desse ponto, o cliente já conhece o servidor bouncer correspondente e pode solicitar a licença para continuar sua execução normal. A solicitação da licença é modelada pela transição T4. Neste caso, optamos por modelar a comunicação de forma assíncrona. Logo, o lugar `Waiting_license` modela o estado de espera correspondente. A partir desse ponto há duas alternativas de execução. A transição T6 modela a possibilidade de negação da licença. Neste caso, o nosso modelo de cliente volta a tentar a solicitação. Naturalmente, na prática, isto deve obedecer a condições mais específicas que são irrelevantes no modelo. A transição T5 modela a concessão da licença ao cliente. Neste caso, a utilização propriamente dita é abstraída pelo lugar `License_in_use`. Ao término do uso, a licença deve ser liberada através do envio da mensagem `Release_license` ao servidor. O pequeno quadrado preto ("■") indica que a transição é de auto-destruição, o que termina o ciclo de vida do cliente.

A.3.2 Classe BOUNCER

A classe BOUNCER modela os servidores bouncer. O corpo da classe é mostrado na Figura A.4. Para facilitar a explicação, seccionamos a figura em seis partes que representam funcionalidades diferentes de um bouncer. As partes foram identificadas de (a) a (f).

A inicialização dos servidores bouncer é mostrada na Figura A.4(a). Ao ser iniciado, cada bouncer segue um procedimento cujo objetivo é atualizar o seu estado de acordo com o dos demais bouncers da rede. Primeiro, o bouncer solicita sua entrada no grupo de comunicação, enviando a mensagem `Join_group` ao serviço de comunicação. Ao entrar no grupo, o bouncer recebe a identificação do líder de grupo, a quem irá requisitar a tabela de dados, enviando a mensagem `Get_table`. Após a inicialização, o bouncer vai para seu estado de espera e despacho de eventos. Isto é feito, depositando uma ficha nos lugares `Ready_gLicense`, `Ready_rLicense`, `Ready_table`, `Ready_release` e `Ready_request`.

Após a inicialização, o bouncer está apto a atender às seguintes requisições concorrentemente:

1. Solicitações de licenças.

Ao receber uma solicitação de licença vinda de seus clientes locais, o bouncer deve repassar o pedido para o grupo—ver transição T3 da Figura A.4(b). Como isto é feito através do serviço de comunicação de grupo, todos os bouncers, incluindo ele próprio, recebem a mensagem `License_request`.

Todos os bouncers, portanto, processam a requisição da maneira indicada pela rede na seção Figura A.4(f). Isto faz com que cada bouncer consulte e atualize sua tabela caso verifique que há licenças disponíveis. Além disso, o bouncer correspondente ao cliente que a requisitou, envia a mensagem `License` ao cliente. Se não há licenças disponíveis, o cliente recebe a mensagem `No_license`.

2. Liberações de licenças.

Como vimos, ao finalizar sua execução, o cliente envia uma mensagem `Release_license` ao bouncer liberando a licença usada. O bouncer executa procedimento semelhante ao anterior. Isto é, repassa a indicação da liberação para todo o grupo—ver Figura A.4(c).

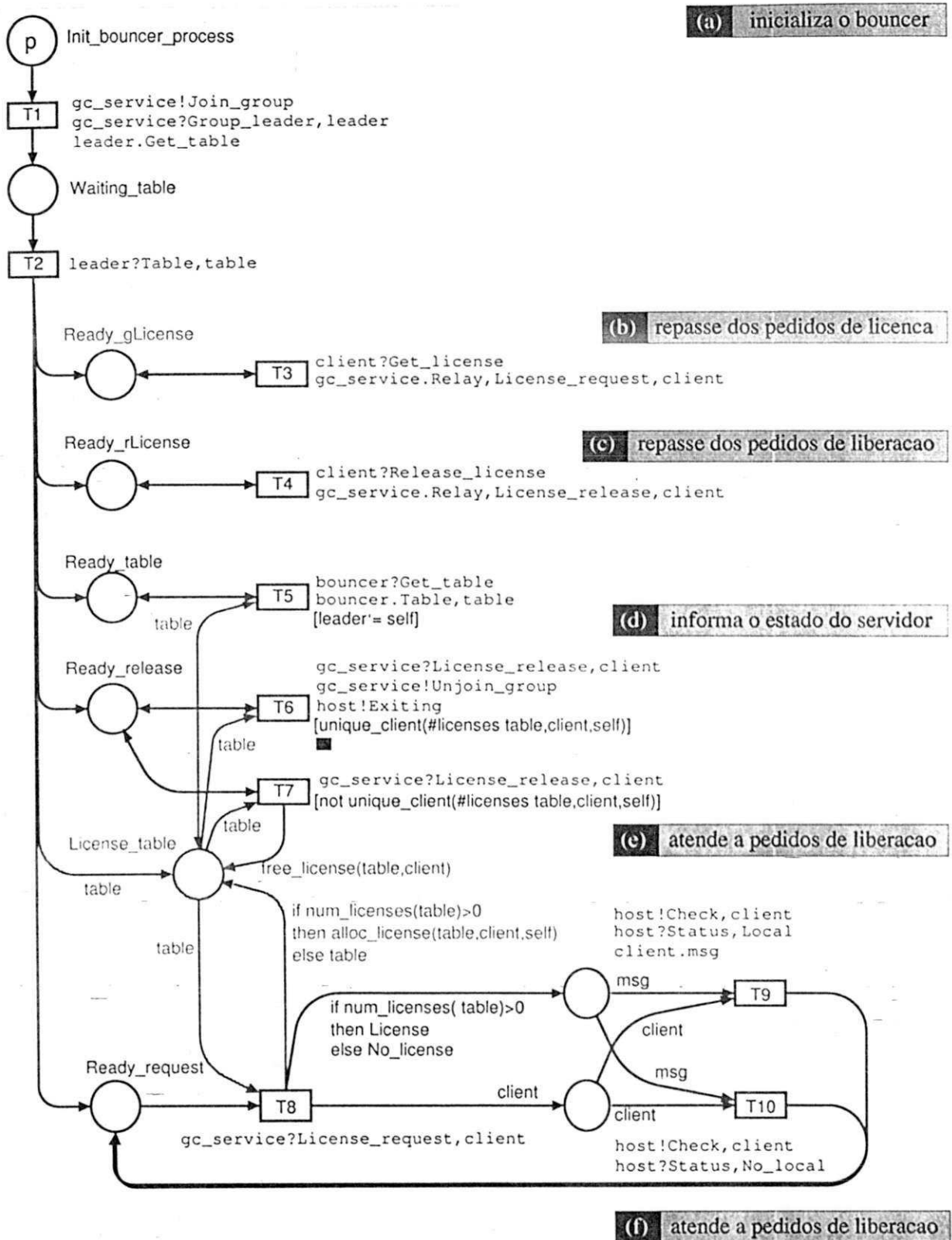


Figura A.4: Classe BOUNCER

O tratamento da liberação é mostrada na Figura A.4(e). Observe que há duas possibilidades, representadas pelas transições T6 e T7. A primeira é executada se o cliente que liberou a licença é o último cliente local do bouncer. Neste caso, o servidor deve deixar o grupo e terminar seu ciclo de vida. Observe que para garantir que a auto-destruição é sincronizada com a atualização da informação na máquina (inscrição `host!Exiting`). A segunda alternativa é executada se o cliente não seja o último. Neste caso, o único efeito é a alteração da tabela.

3. Requisição da tabela de dados (para o líder).

Um bouncer também pode receber a mensagem `Get_table`, desde que seja o líder do grupo bouncer. Neste caso, ele retorna a cópia da tabela para o bouncer que a requisitou. Isto é modelado pela transição T5 na Figura A.4(d).

A.3.3 Classe HOST

Objetos da classe `HOST` representam a infra-estrutura abaixo do nível de processos em que o sistema efetivamente funciona. Objetos desta classe devem atender a pedidos de bouncers e de clientes—relembre que cada processo cliente ou bouncer se comunica exclusivamente com o *host* em que está executando. Uma breve análise das transições T2 e T3, destacadas na Figura A.5, permitem entender como são tratadas as mensagens vindas dos bouncers.

Um bouncer interage com o *host* em que executa, apenas para verificar se um cliente é local a ele. Neste caso, o *host* ao receber uma mensagem `Check`, verifica se este cliente existe em sua base local (`LocalClients`). Caso exista, uma mensagem `Local` é enviada ao bouncer—ver transição T3. Caso contrário, o *host* retorna uma mensagem `No_local` para indicar que o cliente não é local—ver transição T2.

Os clientes também podem fazer pedidos ao *host* em que executam. Como explicado na Seção A.3.1, ao iniciar sua execução, o cliente envia ao *host* uma mensagem `Get_bouncer` requisitando a identificação do servidor bouncer local. Ao receber esta mensagem, o *host* verifica em `LocalBouncer` se já há um servidor bouncer local. Em caso afirmativo, envia de volta ao cliente a identificação do servidor. Caso contrário, retorna a mensagem `No_bouncer`. Ao ser notificado que não há servidor local em funcionamento, o cliente requisita ao *host* a inicialização de um servidor através da mensagem `Init_bouncer`. Ao receber esta solicitação, o *host* cria um novo objeto bouncer e retorna sua identificação ao

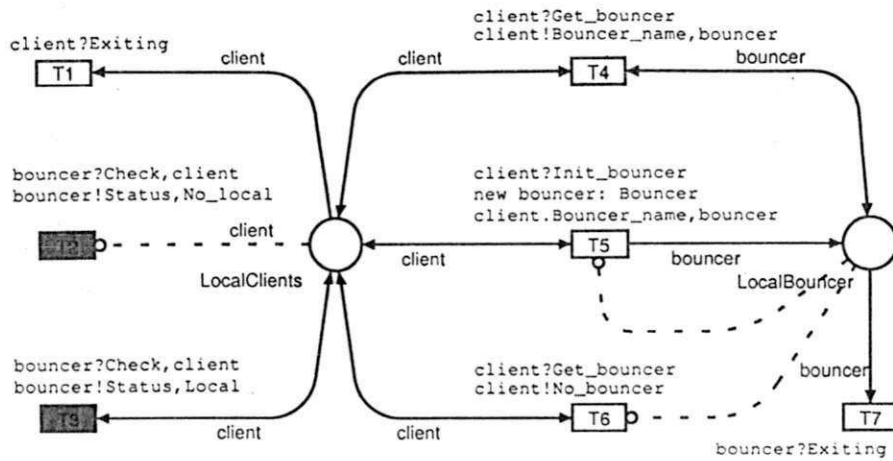


Figura A.5: Classe HOST

cliente que a requisitou—veja as inscrições da transição T5 da Figura A.5. Naturalmente, este é um modelo abstrato do que efetivamente ocorre. Contudo, este nível de detalhe é suficiente para o foco de interesse: o protocolo bouncer.

A.3.4 Classe GC_SERVICE

Um grupo é uma abstração usada para identificar um conjunto de processos que devem obedecer certas regras de comunicação. Como há várias formas possíveis de implementação, e por ser um pressuposto para o serviço bouncer, modelamos esta funcionalidade da forma mais abstrata possível. Logo, qualquer instância deste modelo deve conter um único objeto da classe GC_SERVICE para representar todo o serviço de comunicação de grupo.

O serviço de comunicação provê mecanismos para a entrada e saída de membros do grupo, além de mecanismos para o repasse de mensagens. O corpo da classe é mostrado na Figura A.6.

As transições T1 e T2 modelam o recebimento da mensagem Relay que indica um pedido de repasse de mensagem para os membros do grupo. O repasse é feito de forma extremamente simples pelas inscrições `members(group)!License_request, client` ou `members(group)!License_release, client`. Observe que, neste caso, os objetos para os quais as mensagens devem ser passadas são determinados *dinamicamente* pela avaliação de `members(group)`. Isto significa que o número exato de objetos que devem receber a mensagem não é conhecido a priori. Observe ainda que as transições somente ocorrem se for possível sincronizar com todos esses objetos. É isto que garante a característica de

atomicidade e ordenação da entrega de mensagens.

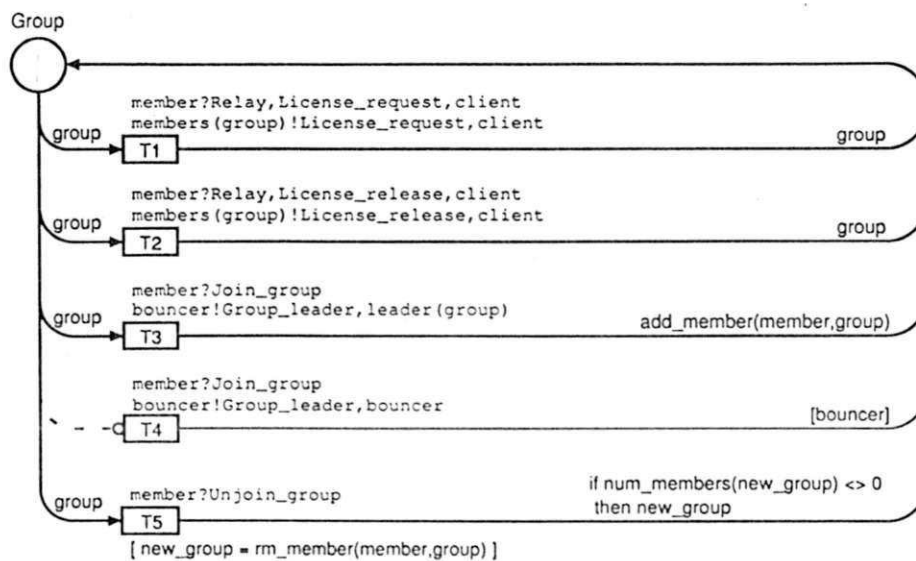


Figura A.6: Classe GC_SERVICE

As transições T3, T4 e T5 modelam entradas e saídas do grupo. As mensagens tratadas são Join_group e Unjoin_Group. Quando um membro do grupo termina sua execução, deve enviar a mensagem Unjoin_group para o serviço de comunicação. Ao receber esta mensagem, o membro será removido do grupo.

A.4 Diagrama de Fluxo Mensagens

O diagrama de fluxo de mensagens foi elaborado para facilitar a identificação das mensagens trocadas entre objetos do sistema e sua relação com o protocolo especificado. O diagrama é mostrado na Figura A.7. Observe que o diagrama permite identificar claramente as mensagens trocadas entre os objetos do modelo. Na parte destacada da figura, podemos identificar as mensagens que podem ser trocadas por objetos das classes CLIENT e BOUNCER. Um objeto cliente pode se comunicar com um objeto bouncer solicitando licenças, ou solicitando a liberação de licenças—mensagens Get_license e Release_license, respectivamente. Um objeto bouncer, por sua vez, interage com um objeto cliente enviando as mensagens No_license para indicar que o pedido de licença foi negado e License, para conceder a licença.

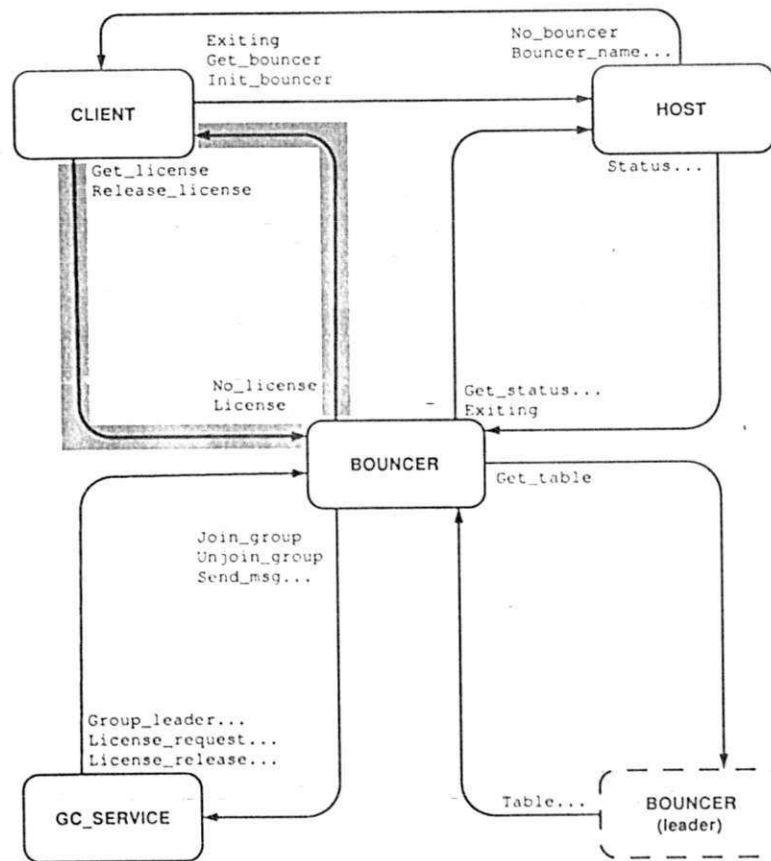


Figura A.7: Diagrama de fluxo de mensagens

A.5 Conclusões

O experimento de modelagem apresentado nos permitiu avaliar a notação introduzida de diversos aspectos. A sintaxe concreta, apresentada na Seção 5.3, foi consolidada através do experimento aqui apresentado. Todas as modificações, contudo, foram feitas de forma que apenas conceitos do nível de sintaxe concreta fossem alterados.

A experiência foi executada por duas pessoas: o próprio autor e outro participante. O outro participante tinha conhecimento prévio de modelagem com redes de Petri coloridas hierárquicas, pouco contato com modelagem orientada objetos e nenhum conhecimento prévio da notação específica utilizada. Apesar disto, não mais do que 10% do tempo dedicado ao projeto foi gasto com o aprendizado e explicação da notação. Devido à aderência aos conceitos originais de redes coloridas, a maior parte do tempo gasto com aprendizado foi dedicada à compreensão do modelo conceitual de objetos concorrentes. Uma vez compreendido o modelo conceitual, a compreensão da notação e de sua semântica foi bastante simples.

A maior parte do tempo dedicado ao projeto foi gasto na compreensão da especificação original do sistema e dos protocolos. Diversas inconsistências, omissões e ruídos foram encontrados na especificação original e revistas com um dos autores do sistema, com quem os modelos obtidos foram validados.

A falta de ferramentas específicas para a notação desenvolvida é um dos fatores que mais dificulta sua aplicação. Apesar disso, devido a sua aderência a redes coloridas, utilizamos a ferramenta Design/CPN [SCK97] para construir e verificar os corpos das classes. É importante observar que a análise local das classes poderia ser feita através dos métodos convencionais de análise. A análise de configurações, contudo, depende ainda da construção de suporte adequado.

Bibliografia

- [AdBKR86] P. America, J. de Bakker, J. N. Kok, and J. Rutten, *Operational semantics of a parallel object-oriented language*, Proceedings of POPL'86 (St. Petersburg Beach, Florida), 1986, pp. 194–208.
- [AdBKR92] ———, *Denotational semantics of a parallel object-oriented language*, Ten Years of Concurrency Semantics: Selected Papers of the Amsterdam Concurrency Group (J. W. de Bakker and J. J. M. Rutten, eds.), World Scientific, Singapore, 1992, pp. 218–205.
- [AdC01] Gul Agha and Fiorella de Cindio (eds.), *Advances in petri nets, special issue on object-oriented concurrent programming*, Lecture Notes in Computer Science, Springer-Verlag, 2001.
- [AFK⁺93] Gul Agha, Svend Frolund, Wooyoung Kim, Rajendra Panwar, Anna Patterson, and Daniel Sturman, *Abstraction and modularity mechanisms for concurrent computing*, Research Directions in Concurrent Object-Oriented Programming (Gul Agha, Peter Wegner, and Akinori Yonezawa, eds.), MIT Press, Cambridge, Massachusetts, 1993.
- [Agh86] Gul A. Agha, *Actors: A model of concurrent computation in distributed systems*, MIT Press, Cambridge, Massachusetts, 1986.
- [AWY93] Gul Agha, Peter Wegner, and Akinori Yonezawa (eds.), *Research directions in concurrent object-oriented programming*, MIT Press, Cambridge, Massachusetts, 1993.
- [BBF97] Tárício Rodrigues Bezerra, Francisco Vilar Brasileiro, and Walfredo Costa Cirne Filho, *Bouncer: Um serviço distribuído e tolerante a faltas para*

controle de licenças de software, VII Simpósio de Computadores Tolerantes a Falhas (Campina Grande, PB – Brasil) (Marcos Borges, ed.), Sociedade Brasileira de Computação, July 1997, pp. 221–235.

- [BBFpBaM97] Francisco Vilar Brasileiro, Tércio Rodrigues Bezerra, Walfredo Costa Cirne Filho, and J. Antão Beltrão Moura, *On the design of bouncer: A robust and flexible license management service for avoiding illegal use of software*, XI SBES, Simpósio Brasileiro de Engenharia de Software (Fortaleza, CE – Brasil) (Marcos Borges, ed.), Sociedade Brasileira de Computação, October 1997, pp. 15–29.
- [BBG01] Olivier Biberstein, Didier Buchs, and Nicolas Guelfi, *Object-oriented nets with algebraic specifications: The CO-OPN/2 formalism*, Advances on Petri Nets: Object Orientation and Models of Concurrency, Lecture Notes in Computer Science (2001), 73–130.
- [BCM96] E. Battiston, F. De Cindio, and G. Mauri, *Modular algebraic nets to specify concurrent systems*, IEEE Transactions on Software Engineering **10** (1996), no. 22.
- [BdCM88] E. Battiston, F. de Cindio, and G. Mauri, *OBJSA nets: a class of high-level nets having objects as domains*, Advances on Petri Nets 1988 (G. Rozenberg, ed.), Lecture Notes in Computer Science, vol. 340, Springer-Verlag, 1988, pp. 20–43.
- [BG86] Robert Balzer and Neil Goldman, *Principles of good software specification and their implications for specification languages*, Software Specification Techniques (N. Gehani and A. D. McGettrick, eds.), International Computer Science Series, Addison-Wesley, 1986, Previously published in Proceedings of IEEE Conference on Specifications of Reliable Software, 1979, pp. 25–39.
- [BG91] D. Buchs and N. Guelfi, *CO-OPN: A concurrent object oriented petri net approach*, Proceedings of the 12th International Conference on the Application and Theory of Petri Nets, Lecture Notes in Computer Science, Springer Verlag, Aarhus, Denmark, 1991.

- [Bid87] M. Bidoit, *The stratified loose approach: a generalization of initial and loose semantics*, Recent Trends in Data Type Specification, selected papers of 5th Workshop on Specification of Abstract Data Types, Lecture Notes in Computer Science, vol. 322, Springer-Verlag, 1987, pp. 1–22.
- [BR85] S. D. Brookes and A. W. Roscoe, *An improved failures model for communication processes*, Seminar on Semantics of Concurrency (S. D. Brookes, ed.), Lecture Notes in Computer Science, vol. 197, Springer-Verlag, 1985, pp. 281–305.
- [BRJ99] Grady Booch, James Rumbaugh, and Ivar Jacobson, *The unified modeling language user guide*, Object Technology Series, Addison-Wesley, 1999.
- [CHJ86] B. Cohen, W. T. Harwood, and M. I. Jackson, *The specification of complex systems*, Addison-Wesley, 1986.
- [CKR94] L. Cherkasova, V. Kotov, and T. Rokicki, *On net modelling of industrial size concurrent systems*, Proceedings of 15th International Conference on the Application and Theory of Petri Nets (Zaragoza), 1994.
- [CT93] S. Christensen and J. Toksvig, *DesignBeta V2.0.1 - BETA code segments in CP-nets*, Tech. Report 5, Computer Science Department, Aarhus University, Aarhus, Denmark, 1993.
- [CW96] Edmund M. Clarke and Jeannette M. Wing, *Formal methods: State of the art and future directions*, ACM Computing Surveys (1996), Special Issue: ACM Strategic Directions in Computing Research.
- [DC91] Y. Deng and S.K. Chang, *A framework for the modeling and prototyping of distributed information systems*, International Journal of Software Engineering and Knowledge Engineering 2 (1991), no. 3, 203–226.
- [DC92] ———, *Unifying multi-paradigms in software system design*, Proc. of the 4th Int. Conf. on Software Engineering and Knowledge Engineering (Capri, Italy), June 1992.

- [DCdFP93] Y. Deng, S.K. Chang, J.C.A. de Figueiredo, and A. Perkusich, *Integrating software engineering methods and petri nets for the specification and prototyping of complex software systems*, Application and Theory of Petri Nets 1993 (M. Ajmone Marsan, ed.), Lecture Notes in Computer Science, vol. 691, Springer-Verlag, Chicago, USA, June 1993, pp. 206 – 223.
- [Dij71] E. W. Dijkstra, *Hierarchical ordering of sequential processes*, Acta Informatica 1 (1971), 115–138.
- [Dij76] Edsger W. Dijkstra, *A discipline of programming*, Prentice-Hall Series in Automatic Computation, Prentice-Hall, 1976.
- [DRS95] R. Duke, G. Rose, and G. Smith, *Object-Z: A specification language advocated for the description of standards*, Computer Standards and Interfaces 17 (1995), 511–533.
- [ELR90] J. Engelfriet, G. Leih, and G. Rozenberg, *Net-based description of parallel object-based systems, or POTs and POPs*, Foundations of Object-Oriented Languages (J.W. de Bakker, W.P. de Roever, and G. Rozenberg, eds.), Lecture Notes in Computer Science, Springer-Verlag, 1990, pp. 229–273.
- [EM85] H. Ehrig and B. Mahr, *Fundamentals of algebraic specification 1 — equations and initial semantics*, EATCS Monographs on Theoretical Computer Science, vol. 6, Springer-Verlag, Berlin, 1985.
- [Fis96] Clemens Fischer, *Combining csp and z*, University of Oldenburg, 1996.
- [Fis97] C. Fischer, *CSP-OZ: a combination of Object-Z and CSP*, Proc. 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS) (Canterbury, UK) (H. Bowman and J. Derrick, eds.), Chapman and Hall, London, 1997, pp. 423–438.
- [FPB87] Jr. Frederick P. Brooks, *No silver bullet: Essence and accidents of software engineering*, IEEE Computer 20 (1987), no. 4, 10–19.
- [FW99] Clemens Fischer and H. Wehtheim, *Model-checking csp-oz specifications with fdr*, Proceedings of the 1st International Conference on Integrated

- Formal Methods (IFM) (K. Araki, A. Galloway, and K. Taguchi, eds.), 1999, pp. 315–334.
- [GdFP97] D.D.S. Guerrero, J.C.A. de Figueiredo, and A. Perkusich, *Object-based high-level petri nets as a formal approach to distributed information systems*, Proc. of IEEE Int. Conf. on Systems Man and Cybernetics (Orlando, USA), October 1997, Invited Paper, pp. 3383–3388.
- [GdFP01] Dalton D. S. Guerrero, J. C. A. de Figueiredo, and A. Perkusich, *An object-based modular cpn approach: its application to the specification of a cooperative editing environment*, Advances on Petri Nets: Object Orientation and Models of Concurrency, Lecture Notes in Computer Science (2001), 338–354.
- [Gen87] H.J. Genrich, *Predicate/Transition nets*, Petri Nets: Central Models and Their Properties (W. Brauer, W. Reisig, and G. Rozemberg, eds.), Lecture Notes in Computer Science, vol. 254, Springer-Verlag, 1987, pp. 207–247.
- [Gog84] J. A. Goguen, *Parameterized programming*, IEEE Transactions on Software Engineering SE-10 (1984), no. 5.
- [Gri96] David Gries, *The need for education in useful formal logic*, IEEE Computer (1996), 29–30, In An Invitation to Formal Methods.
- [Gue97] Dalton Dario Serey Guerrero, *Sistemas de redes de petri modulares baseadas em objetos*, Dissertacao de mestrado, COPIN - Universidade Federal da Paraiba, 1997.
- [Gue98] Dalton D. S. Guerrero, *Orientação a objetos e modelos de redes de petri*, Tech. report, Coordenação de Pós-graduação em Engenharia Elétrica - COPELE/UFPB, Campina Grande, PB, April 1998.
- [Hew77] Carl Hewitt, *Viewing control structures as patterns of passing messages*, Journal of Artificial Intelligence 8 (1977), no. 3, 323–364.
- [HJS90] P. Huber, K. Jensen, and R.M. Shapiro, *Hierarchies in coloured petri nets*,

- Advances in Petri Nets (G. Rozenberg, ed.), Lecture Notes in Computer Science, no. 483, Springer-Verlag, 1990.
- [HJS91] ———, *Hierarchies in coloured petri nets*, High-Level Petri Nets: Theory and Application (Jensen. K. and G. Rozenberg, eds.), Springer-Verlag, 1991, pp. 313–341.
- [Hoa78] C. A. R. Hoare, *Communicating sequential processes*, Communications of the ACM **21** (1978), no. 8, 666–677.
- [Hoa84] C.A.R. Hoare, *Programming: Sorcery or science?*, IEEE Software **4** (1984), no. 4, 6–16.
- [Hoa85] ———, *Communicating sequential processes*, Prentice-Hall, 1985.
- [JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The unified modeling language*, Object Technology Series, Addison-Wesley, 1999.
- [JCJO92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object oriented software engineering: A use case driven approach*, Addison-Wesley, 1992.
- [Jen92] Kurt Jensen, *Coloured petri nets: Basic concepts, analysis, methods and practical use. volume 1*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1992.
- [Jen94] ———, *Coloured petri nets: Basic concepts, analysis methods and practical use. volume 2*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1994.
- [Jen98] ———, *Coloured petri nets: Basic concepts, analysis, methods and practical use. volume 3*, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1998.
- [Lak37] Charles Lakos, *Object oriented modelling with object petri nets*, Advances on Petri Nets: Object Orientation and Models of Concurrency, Lecture Notes in Computer Science (1–37), 2001.

- [Lak95] ———, *From Coloured Petri nets to object Petri nets*, Proceedings of the 15th International Conference on the Application and Theory of Petri Nets, Lecture Notes in Computer Science, Springer Verlag, Turin, Italy, 1995, pp. 278–297.
- [Lak97] C. Lakos, *On the abstraction of coloured petri nets*, Application and Theory of Petri Nets (Toulouse, France) (Pierre Azéma and Gianfranco Balbo, eds.), Lecture Notes in Computer Science, vol. 1248, Springer-Verlag, June 1997, pp. 42–61.
- [LK91] C.A. Lakos and C.D. Keen, *Modelling layered protocols in LOOPN*, Proceeding of Fourth International Workshop on Petri Nets and Performance Models (Melbourne, Australia), 1991.
- [LK95] C. Lakos and C. Keen, *LOOPN++ a new language for object-oriented petri nets*, Proceedings of Modelling and Simulation - European Simulation Conference (Barcelona, Spain), 1995, pp. 369–374.
- [Mai97] T. S. M. Maibaum, *What we teach software engineers in the university: Do we take engineering seriously?*, Software Engineering Notes 22 (1997), no. 6, 40–50, Also published as LNCS 1301.
- [Mes90] J. Meseguer, *A logical theory of concurrent objects*, OOPSLA/ECOOP'90 (Ottawa, Canada) (Norman Meyrowitz, ed.), ACM Sigplan Notices, vol. 25, ACM Press, October 1990, pp. 101–115.
- [Mey85] Bertand Meyer, *On formalism in specifications*, IEEE Software January (1985), no. 1, 6–26.
- [Mil80] Robin Milner, *A calculus of communicating systems*, Lecture Notes in Computer Science, vol. 92, Springer-Verlag, Berlin, Heidelberg - Germany, 1980.
- [Mil93] ———, *The polyadic π -calculus: a tutorial*, Logic and Algebra of Specification, ASI Series, vol. 94, Springer-Verlag, 1993.
- [MOM93] N. Martí-Oliet and J. Meseguer, *Rewriting logic as a logical and semantic framework*, Technical Report SRI-CSL-93-05, SRI International, 1993.

- [MPW89] R. Milner, J. Parrow, and D. Walker, *A calculus of mobile processes*, Tech. Report ECS-LFCS-89-85 and -86, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, Edinburgh University, 1989.
- [MS01] Alexandre Mota and Augusto Sampaio, *Model-checking csp-z: strategy, tool support and industrial application*, Science of Computer Programming **40** (2001), no. 1, 59–96.
- [Mur89] T. Murata, *Petri nets: Properties, analysis and applications*, Proc. of the IEEE **77** (1989), no. 4, 541–580.
- [Par72] David L. Parnas, *On the criteria to be used in decomposing systems into modules*, Communications of the ACM **14** (1972), no. 1, 221–227.
- [Par90] ———, *Education for computing professionals*, IEEE Computer **23** (1990), no. 1, 17–22.
- [Par94] ———, *Mathematical descriptions and specifications of software*, Proceedings of IFIP World Congress, vol. 1, August 1994, pp. 354–359.
- [Par96] ———, *Mathematical methods: what we need and don't need*, IEE Computer (1996), 28–29, In An Invitation to Formal Methods.
- [Par97] ———, *Software engineering: An unconsummated marriage*, Software Engineering Notes **22** (1997), no. 6, 1–3, Also published as LNCS 1301.
- [PdF97] A. Perkusich and J.C.A. de Figueiredo, *G-nets: A petri net based approach for logical and timing analysis of complex software systems*, Journal of Systems and Software **39** (1997), no. 1, 39–59.
- [Pet86] C. A. Petri, *"Forgotten Topics" of net theory*, Petri Nets: Applications and Relationships to Other Models of Concurrency (G. Goos, J. Hartmanis, W. Brauer, W. Reisig, and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 255, Springer-Verlag, 1986, pp. 500–514.
- [Pnu77] A. Pnueli, *The temporal logic of programs*, 18th IEEE Annual Symposium on the Foundations of Computer Science (Providence), 1977, pp. 46 – 57.

- [RBP⁺91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-oriented modelling and design*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.
- [Rei91] W. Reisig, *Petri nets with algebraic specifications*, Theoretical Computer Science, no. 81, Springer-Verlag, 1991.
- [RJB99] James Rumbaugh, Ivar Jacobson, and Grady Booch, *The unified modeling language reference manual*, Object Technology Series, Addison-Wesley, 1999.
- [Sai96] Hossein Saiedian, *An invitation to formal methods*, IEEE Computer (1996), 16–17.
- [SB93] C. Sibertin-Blanc, *A client-server protocol for the composition of petri nets*, Application and Theory of Petri Nets 1993 (M. Ajmone Marsan, ed.), Lecture Notes in Computer Science, vol. 691, Springer-Verlag, Chicago, USA, June 1993, pp. 377–396.
- [SB94] ———, *Cooperative nets*, Proceedings of 15th International Conference on the Application and Theory of Petri Nets, Lecture Notes in Computer Science, vol. 815, Springer-Verlag, Zaragoza, Spain, 1994, pp. 471–490.
- [SCK97] Jens Bæk Jørgensen Søren Christensen and Lars Michael Kristensen, *Design/CPN — a computer tool for coloured petri nets*, Tech. Report PB-511, DAIMI, University of Aarhus, Denmark, February 1997.
- [Spi90] J. M. Spivey, *The z notation: A reference manual*, 2nd. ed., Prentice-Hall, 1990.
- [SSC01] Adnan Sherif, Augusto Sampaio, and Sérgio Cavalcante, *An integrated approach to specification and validation of real-time systems*, FME 2001: Formal Methods for Increasing Software Productivity (P. Zave J.N. Oliveira, ed.), Lecture Notes in Computer Science, vol. 2021, Springer-Verlag, March 2001, pp. 278–299.

- [SSW95] Stefan Schof, Michael Sonnenschein, and Ralf Wieting, *Efficient simulation of THOR nets*, Proceedings of the 16th International Conference on Application and Theory of Petri Nets (Turin, Italy) (G. de Michelis and M. Diaz, eds.), Lecture Notes in Computer Science, Springer-Verlag, June 1995, pp. 412–431.
- [Van98] S. Alagar Vangalur, *Specification of software systems*, Graduate Texts in Computer Science, Springer-Verlag, New York, 1998.
- [Vau90] Jacques Vautherin, *Parallel systems specifications with coloured petri nets and algebraic specifications*, Advances in Petri Nets (G. Rozenberg, ed.), Lecture Notes in Computer Science, vol. 266, Springer-Verlag, 1990, pp. 293–308.
- [Zav96] Pammela Zave, *Formal methods are research, not development*, IEEE Computer (1996), 26–27, In An Invitation to Formal Methods.