Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

Contribuições ao Estudo de Redes de Agentes

Autor: Antônio Sérgio Ribeiro Gomes
Orientador: Prof. Dr. Ricardo Ribeiro Gudwin

Banca examinadora:

Prof.: Ricardo Ribeiro Gudwin
DCA/FEEC/UNICAMP

Prof.: Angelo Perkusich
DEE/UFPB

Prof.: Fernando Gomide
DCA/FEEC/UNICAMP

Prof.: Eleri Cardozo
FEEC/UNICAMP

Tese apresentada ao Departamento de Engenharia de Computação e Automação Industrial da Faculdade de Engenharia Elétrica e de Computação da UNICAMP, como parte integrante dos requisitos para obtenção do grau de Mestre em Engenharia Elétrica.

Tese de Mestrado
Campinas - SP - Brasil
Junho, 2000

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

G585c

Gomes, Antônio Sérgio Ribeiro
Contribuições ao estudo de redes de agentes. / Antônio Sérgio Ribeiro Gomes. -- Campinas, SP: [s.n.], 2000.

Orientador: Ricardo Ribeiro Gudwin.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Sistemas inteligentes de controle. 2. Inteligência artificial. 3. Semiótica. I. Gudwin, Ricardo Ribeiro. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

Redes de objetos (ROs) são uma proposta de modelo formal para sistemas dinâmicos a eventos discretos, particularmente útil para a representação e simulação de sistemas inteligentes. Redes de Agentes (RAs) são uma classe de ROs onde utiliza-se uma política autônoma para a geração da assim chamada função de seleção, responsável pela dinâmica da rede. Neste trabalho, apresenta-se uma coletânea de contribuições ao estudo das RAs. Basicamente, estas contribuições podem ser agrupadas em 3 tópicos principais: (a) o aprimoramento do modelo formal para a representação de RAs e a definição de um *framework* lógico-formal para o estudo da dinâmica em RAs, (b) a introdução do conceito de hierarquia em RAs, com a proposta de uma extensão para o suporte a RAs modulares e (c) implementação de um ambiente computacional baseado no paradigma de objetos distribuídos (CORBA) para o desenvolvimento de RAs modulares.

Palavras-Chave: Sistemas inteligentes, Semiótica Computacional, Objetos, Agentes, Sistemas Orientados a Objeto, Inteligência Artificial, Hierarquias.

Abstract

Object networks (ONs) are a formal model proposal to describe discrete event dynamical systems, specially suitable to represent and simulate intelligent systems. Agent networks (ANs) are a class of ONs which uses an autonomous policy to generate the selection function, responsible for determining the network dynamics. In this work, a group of contributions to the study of ANs is presented, and can be summarized in three basic categories, as follows: (a) improvement of the formal model of ANs and a proposal of a formal treatment for the dynamics, (b) introduction of hierarchies in the context of ANs, including a proposal of formal model for modular ANs, and (c) implementation of a computational environment based on the distributed objects paradigm (CORBA) for the development of modular ANs.

Keywords: Intelligent Systems, Computational Semiotics, Objects, Agents, Object-Oriented Systems, Artificial Intelligence, Hierarquies.

À minha família e à memória de meu pai.

Agradecimentos

Foram muitas as pessoas que contribuiram neste trabalho, consolidado a partir de uma longa e árdua jornada. Agradeço em especial ao meu pai Divino, que soube me incentivar nos momentos certos, à minha mãe por sua paciência ao suportar a longa distância e às minhas irmãs Juliana e Tatiane por terem me incentivado e apoiado nos momentos difíceis.

Em especial, agradeço também ao meu orientador, Ricardo R. Gudwin, por ter me aceitado no programa de Mestrado e por ter desempenhado o papel tão crucial que lhe é cabido e à CAPES pela bolsa de estudo concedida.

Por fim, e não menos importante, agradeço encarecidamente a todos os amigos que fiz no laboratório, pela companhia no dia-a-dia e pelas dicas e conselhos importantes que foram de grande valia para mim durante o tempo que convivemos no LCA/FEEC.

Expresso ainda meus sinceros agradecimentos ao Dr. Gerd Doeben-Henisch, pela oportunidade de participar de forma direta no processo de cooperação entre o grupo de pesquisa em Semiótica Computacional e a empresa Knowbotic Systems, sediada em Frankfurt, Alemanha. Sem dúvida, esta experiência marcante causou um impacto enormemente positivo no desenvolvimento deste trabalho, abrindo novas portas até então não vislumbradas.

"Uma teoria pode ser provada por meio de experimentos; mas nenhum caminho leva do experimento até o nascimento de uma nova teoria."

Albert Einstein (1879-1955)

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Definições

Lista de Acrônimos

AAM Ambiente de Agentes Móveis

BDI *Belief-Desire-Intention*

BMSA *Best Matching Search Algorithm*

CORBA *Common Object Request Broker Architecture*

CPN *Coloured Petri Net*

CSP *Constraint Satisfaction Problem*

EG Escopo Gerativo

EH Escopo Habilitante

FS Função de Seleção

HCPN *Hierarquical Coloured Petri Net*

HHPN *Hierarquical High-level Petri Net*

HPN *High-level Petri Net*

IAD Inteligência Artificial Distribuída

IA Inteligência Artificial

JVM *Java Virtual Machine*

MTON *Multi-threaded Object Network*

OMG *Object Management Group*

OPN *Object Petri Net*

ORB *Object Request Broker*

PN *Petri Net*

RAM Rede de Agentes de Modular

RA Rede de Agentes

RDP Resolução Distribuída de Problemas

RMI *Remote Method Invocation*

RO Rede de Objetos

SMA Sistema Multiagentes

SMA Sistema Multiagente

UML *Unified Modeling Language*

Capítulo 1

Introdução

1.1 Prólogo

Há séculos o fenômeno da inteligência humana tem sido o objetivo de intenso estudo no campo das ciências humanas, mais notadamente na Filosofia, com destaque especial para a Semiótica, a ciência de todas as linguagens (Santaella, 1999). Paralelamente, de forma mais contemporânea, o século passado presenciou o nascimento de uma nova ciência também preocupada com este fenômeno, a Inteligência Artificial. A Inteligência Artificial, por se preocupar de forma explícita com o processo de síntese de sistemas inteligentes artificiais requer a elaboração de modelos computacionais que sejam capazes de incorporar a noção de inteligência[1]. Vários modelos foram então propostos na tentativa de caracterizar este fenômeno (Newell, 1982; Albus, 1991; Meystel, 1996).

Na tentativa de estabelecer uma base computacional para o desenvolvimento de tais sistemas foram propostas por Gudwin (Gudwin, 1996) as *redes de objetos*. As redes de objetos (ROs) foram, a princípio, destinadas à representação e processamento de estruturas de conhecimento no paradigma da *Semiótica Computacional* (Gudwin, 1996; Gudwin and Gomide, 1997a,b). Sob este contexto, ROs podem ser aplicadas a uma vasta gama de problemas, como processos de raciocínio, computação com palavras, redes neurais, sistemas nebulosos e algoritmos genéticos (Gudwin, 1996; Gudwin and Gomide, 1997c, 1998a; Guerrero, 2000). Entretanto, apesar de sua concepção ter sido inicialmente voltada para esse contexto, as redes de objetos podem igualmente ser utilizadas para a modelagem e simulação de sistemas a eventos discretos em outros campos de aplicação, tais como automação industrial, protocolos de rede, sistemas computacionais colaborativos, etc., com as mesmas vantagens indicadas no caso dos sistemas inteligentes.

A dinâmica de uma RO, ou seja, o disparo dos objetos e seus respectivos parâmetros, é implementada por meio de um mecanismo chamado *função de seleção*. Este mecanismo é uma descrição genérica para o comportamento de cada um dos objetos da rede. A função de seleção é determinada pelo projetista do sistema. É necessário frisar que essa função tem caráter mandatório na dinâmica do sistema. O fato de ser centralizada, faz com que toda a responsabilidade sobre o comportamento do sistema recaia sobre o projetista do mesmo. Em sistemas mais complexos, essa responsabilidade pode se tornar um fardo por demais pesado, causando dificuldades na caracterização e desenvolvimento do modelo pretendido. Em (Guerrero et al., 1999) foi introduzida uma proposta inicial para tratar o

[1]Em (Russel and Norvig, 1995) apresenta-se uma exposição condensada sobre alguns fundamentos filosóficos para a inteligência, identificando duas correntes principais, as chamadas "noção fraca" e "noção forte".

problema da determinação da função de seleção, tendo-se em conta um mecanismo automático para sua geração. Esse mecanismo dividia a responsabilidade da seleção entre cada objeto isoladamente, de tal forma que sua concepção poderia ser planejada de forma autônoma a cada classe de objeto sendo desenvolvida. Posteriormente em (Guerrero, 2000) essa proposta foi revisada e aperfeiçoada, dando origem às assim chamadas redes de agentes (RAs). As RAs diferenciam-se das ROS exatamente por proverem um mecanismo automático auxiliar para a determinação das funções de seleção. Esse mecanismo é dividido em duas partes. Na primeira, cada objeto do sistema provê um meio de avaliar seu interesse em interagir com outros objetos a disposição. Esta parte é consubstanciada na chamada função de avaliação, que é definida para cada função de transformação (método) de uma classe sendo definida. Essa função de avaliação é utilizada, em seguida, por um algoritmo de alocação generalizado, que gera então, automaticamente, a função de seleção.

1.2 Motivação

O presente trabalho é uma extensão aos conceitos introduzidos por Gudwin (Gudwin, 1996) e posteriormente estudados por Guerrero (Guerrero et al., 1999; Guerrero, 2000), trazendo basicamente três grandes contribuições distintas:

1. Revisão do modelo formal de redes de agentes (Guerrero, 2000), com o intuito de incorporar um enfoque mais efetivo de uma "noção de agência" para as redes de agentes. Nessa revisão, foram incluídas formalizações adicionais para aspectos até então introduzidos apenas informalmente por Guerrero (Guerrero, 2000), sedimentando o caminho para a especificação do modelo formal de redes de agentes modulares. Além disso, apresenta-se uma abordagem formal ao estudo da dinâmica de funcionamento de uma RA, baseado no algoritmo BMSA[2] (Guerrero et al., 1999; Guerrero, 2000). O estudo do comportamento de uma RA propicia uma melhor compreensão da dinâmica do sistema sendo modelado. Uma abordagem formal à determinação da função de seleção é absolutamente necessária à implementação de modelos mais sofisticados.

2. Proposta de um modelo formal para redes de agentes modulares, feita pela introdução de mecanismos de hierarquização na definição da topologia (configuração de lugares e arcos), permitindo a representação de um dado problema em múltiplos níveis de abstração, bem como a capacidade de reutilização de partes.

3. Implementação de um ambiente computacional para o desenvolvimento e simulação de redes de agentes modulares em ambientes distribuídos e heterogêneos. O desenvolvimento de uma plataforma computacional utilizando recursos de objetos distribuídos CORBA[3] busca pavimentar o caminho para uma futura integração com plataformas de escopo mais generalizado.

Para a conveniência do leitor este trabalho segue a seguinte estrutura de capítulos:

Capítulo 2 Este capítulo apresenta um resumo do panorama atual em tecnologia de agentes, enfocando aspectos relevantes no âmbito da tese;

[2] *Best Matching Search Algorithm*
[3] *Common Object Request Broker Architecture*

Capítulo 3 Este capítulo realiza uma extensa revisão do modelo formal de redes de agentes (Guerrero, 2000). Para a conveniência do leitor todas as definições anteriores de relevância no presente trabalho foram incluídas no capítulo;

Capítulo 4 Investiga técnicas de estruturação hierárquica em redes de agentes. Este capítulo introduz ainda um modelo formal para as assim chamadas redes de agentes modulares;

Capítulo 5 Descreve o sistema de desenvolvimento de redes de agentes ONTOOL-2, incluindo um exemplo de aplicação de redes de agentes modulares em um problema de agendamento de tarefas;

Capítulo 6 Apresenta um resumo dos principais pontos abordados no trabalho, enfatizando-se as contribuições apresentadas bem como sugerindo possíveis tópicos para trabalhos futuros;

Apêndice A Apresenta uma rápida introdução a problemas de satisfação de restrições, diretamente relacionados com a formalização proposta para o algoritmo BMSA, apresentada no Capítulo 3;

Apêndice B Introduz alguns conceitos elementares da arquitetura CORBA;

Capítulo 2

Teoria de Agentes

Este capítulo traça um panorama geral da teoria de agentes, desde suas noções intuitivas básicas até os chamados sistemas multiagentes. O enfoque se dá com a intenção de definir um contexto teórico no qual as redes de agentes possam ser apresentadas.

2.1 Introdução

A princípio não existe uma definição universalmente aceita para o que vem a ser um agente. Ainda há muita discussão e controvérsia sobre o assunto, conforme cita Wooldridge (Wooldridge, 1999, p28). Dentre os componentes conceituais presentes na noção de agência a **autonomia** desponta como uma das mais típicas. Mas, apesar disso, a própria definição de autonomia é relativa[1].

Para um desenvolvimento adequado, há a necessidade de pelo menos algumas linhas gerais que delimitem o campo de estudo, caso contrário, o termo pode perder seu significado. A partir de uma análise superficial, observa-se que a definição de agência é determinada, em grande parte, pelo campo de pesquisa em questão. Assim, sob o panorama exposto neste trabalho, torna-se inconveniente uma definição rigidamente fechada para a noção de agência. No decorrer do capítulo serão enfatizados aqueles pontos que apresentem a melhor relação entre custo e benefício para os propósitos desta pesquisa. Uma proposta muito fechada pode limitar o campo de estudo a um grupo muito pequeno de casos e uma muito aberta pode tornar muitos conceitos indiscerníveis.

2.1.1 Organização do Capítulo

A Seção 2.2 apresenta algumas definições fundamentais para a noção de agência, incluindo uma discussão a respeito da diferenciação semântica dos termos agente e objeto. A Seção 2.3 estuda os sistemas multiagentes e suas técnicas de coordenação, a Seção 2.4 caracteriza os agentes móveis e, ao final, faz-se um breve resumo dos tópicos abordados neste capítulo.

[1] Conforme mostrado mais adiante na comparação entre agentes e objetos.

2.2 Agentes

O termo "agente", no conceito descrito aqui, tem suas origens nos primeiros trabalhos em IA[2], quando pesquisadores concentravam-se em construir entidades artificiais que pudessem imitar o comportamento humano. No seu sentido mais restrito, o termo pode ser aplicado a toda uma gama de entidades, incluindo os sistemas de software que se tornaram sinônimos do termo, como robôs autônomos e organismos biológicos (Green et al., 1997).

2.2.1 Noções Básicas

A primeira definição apresentada a seguir é dada por Wooldridge (Wooldridge, 1999, pág.29). Ela possui a vantagem de ser genérica o bastante para permitir uma ampla gama de estudo (veja a Figura 2.1):

> "Um *agente* é um sistema computacional *situado* em um *ambiente*, sendo capaz de agir autonomamente para alcançar suas metas."

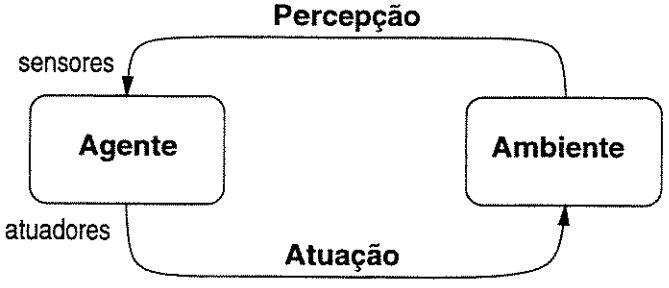


Figura 2.1: Um agente em seu ambiente.

Esta definição representa um ponto de partida na definição de agência, pois as propriedades descritas podem ser identificadas em virtualmente todas as definições de agente comumente encontradas na literatura.

De uma forma análoga ao estudo da Inteligência Artificial, geralmente classificada em abordagem fraca e abordagem forte (Russel and Norvig, 1995, p99), o estudo da agência também apresenta duas vertentes principais. Wooldridge e Jennings (Wooldridge and Jennings, 1995) identificam estes dois níveis possíveis como:

- **agência fraca**: compreende autonomia, habilidade social, reatividade e pró-atividade. Representa características mais genéricas, podendo ser apresentadas por uma gama variada de sistemas de hardware e software. Assim, um *daemon*[3] UNIX pode ser caracterizado como um agente, tal como ocorre com um robô de *software* (*softbot*). Este tipo de abordagem encontra aceitação acentuada principalmente na área de Ciência da Computação.

[2]Inteligência Artificial

[3]Compreendido como um programa executado em segundo plano, sendo projetado para oferecer um determinado tipo de serviço a outros programas.

- **agência forte**: capacidade de apresentar, além de todos os elementos anteriores, a idéia de *estados mentais*. Esta é a abordagem tipicamente aceita pelos pesquisadores no campo da Inteligência Artificial. Encampa a noção de equivalência com os conceitos aplicados a seres humanos, incluindo conhecimento, crença, intenção e obrigação. Alguns pesquisadores vão adiante com a adição de estados emocionais (Padghan and Taylor, 1996).

Outros atributos geralmente considerados no contexto de agência incluem:

- *mobilidade*, a habilidade do agente se mover em seu ambiente;
- *veracidade*, a suposição de que o agente não comunica, voluntariamente, nenhuma informação falsa;
- *benevolência*, a suposição de que agentes não têm objetivos conflitantes, de forma que cada agente sempre tenta cumprir as tarefas solicitadas; e
- *racionalidade*, a suposição de que um agente agirá para cumprir os seus objetivos e não o contrário, na medida em que suas crenças assim o permitam.

Guerrero (Guerrero, 2000, p51) condensa um apanhado de definições de agência a partir de várias fontes diferentes. Cada uma delas é essencialmente vinculada ou à abordagem fraca ou à abordagem forte, dependendo do enfoque de trabalho do pesquisador em questão.

No que se refere ao ambiente no qual o agente está inserido, Russel e Norvig (Russel and Norvig, 1995, p46) sugerem uma extensa classificação: acessível ou inacessível, determinístico ou não-determinístico, episódico ou não-episódico, estático ou não-estático e, finalmente, discreto ou contínuo.

2.2.2 Agente ou Objeto?

Graças ao uso intenso das palavras *objeto* e *agente* em escopos gradativamente crescentes torna-se necessária a diferenciação clara entre estes dois termos. Na visão do autor, existem dois cenários básicos onde tais discussões podem eventualmente acontecer. A primeira se dá no campo computacional, em que os termos em questão possuem uma tendência a seguir preceitos derivados da programação orientada a objeto. A segunda, por outro lado, apresenta uma noção filosófica de objeto, há séculos utilizada para representar entidades caracterizadas por fenômenos do mundo (Smith, 1996).

Visão Computacional

Um objeto é geralmente visto como uma entidade com um conjunto de estados e um conjunto de métodos que podem ser invocados por meio do envio de mensagens (Pascoe, 1990; Stefik and Bobrow, 1990; Selic et al., 1994).

Comparando-se essa visão de objeto com a de um agente, a primeira diferença que se observa refere-se à autonomia. Considere um objeto onde todos os seus estados internos sejam privados[4] e apenas seus próprios métodos possam acessá-los diretamente (embora alguns desses métodos possam ser visíveis a outros objetos). Mesmo que ele exerça controle sobre seus estados ele não tem necessariamente poder sobre as mensagens que recebe, que são geralmente enviadas por outros objetos. Desta forma, ele não consegue ser o único responsável pelo seu comportamento. No entanto, um objeto

[4]Em linguagens como Java e C++ empregando-se a palavra-chave `private`.

pode implementar alguma política interna aos métodos para ter a opção de aceitar ou não uma dada mensagem, mas isto já se distancia do conceito original de objeto.

Uma segunda diferença se refere ao comportamento flexível. Sob o ponto de vista dinâmico, há duas classes distintas (Booch et al., 1997):

> *Um objeto ativo possui seu próprio thread de execução [...]. Objetos ativos são geralmente autônomos, o que significa que podem exibir comportamento sem necessariamente serem estimulados. Objetos passivos, por outro lado, podem sofrer uma mudança de estado apenas quando explicitamente operados por outros objetos.*

Objetos ativos podem possuir um comportamento dinâmico semelhante ao de um agente. A diferença fundamental está no fato de que objetos ativos não apresentam necessariamente um comportamento autônomo flexível tal como um agente tem. Um agente é dito autônomo na medida em que sua capacidade de tomar decisões é baseada nas suas próprias experiências e não nos conhecimentos embutidos pelo projetista (Russel and Norvig, 1995, p35).

Outros autores são mais pragmáticos, enumerando características gerais para as noções de agência e de objeto (Kendall et al., 1995), conforme ilustra a Figura 2.2. Tais diferenciações *ad hoc* são bastante influenciadas pelo domínio dos problemas tratados, não refletindo de forma confiável a fronteira conceitual. Cabe fazer algumas ressalvas em relação aos seguintes pontos citados em (Kendall et al., 1995):

- **herança**: a princípio nada impede que mecanismos de herança estrutural sejam usados na definição de classes de agentes derivados.

- **comportamento**: conforme apresenta Wooldridge, objetos demonstram comportamento sem exercer, de forma implícita, controle sobre ele.

- **concorrência**: apesar do trabalho citado considerar a concorrência um elemento intrínseco dos agentes, torna-se necessário, mais uma vez, reafirmar que objetos ativos podem muito bem suportar um elevado grau de sincronização.

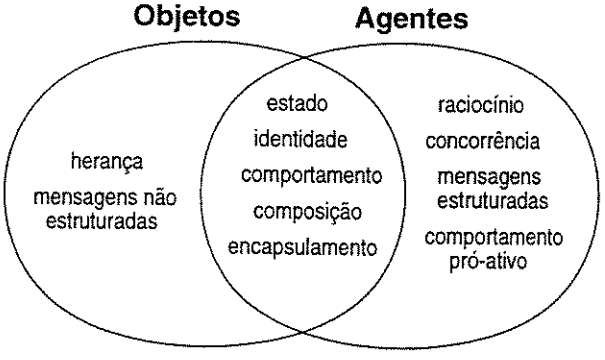


Figura 2.2: Comparativo de propriedades entre agentes e objetos proposto por Kendall et al. (1995)

Caracterizando Objetos em Outros Contextos

Sob o ponto de vista mais geral possível, a palavra *objeto*, datada do século XIV, significa[5]: "algo material que pode ser percebido pelos sentidos, com relação a um pensamento, sentimento ou ação".

Na visão da filosofia semiótica de Peirce, a noção de objeto é muito mais genérica, apresentando duas formas essenciais (Santaella, 1999):

- *objeto dinâmico*, o objeto independente do *signo*[6], em toda a sua plenitude. Pode-se dizer que este objeto é inatingível. Pode ser comparado a um diamante com infinitas faces, em que cada uma delas representa apenas parte deste objeto dinâmico, nunca descrevendo-o por completo. As nossas percepções sempre serão um reflexo da nossa capacidade perceptiva (ou interpretativa), capturando sempre parcialmente informações sobre o objeto dinâmico; e

- *objeto imediato*, o objeto assim como é representado pelo *signo*. Representa apenas uma das possíveis percepções do objeto dinâmico. No exemplo do diamante, descrito anteriormente, um objeto imediato poderia ser comparado a uma das faces do diamante.

2.2.3 Arquiteturas de Agentes

Uma arquitetura (concreta) de agentes é um mapa dos detalhes internos de um agente, suas estruturas de dados, as operações que ele pode realizar sobre estas estruturas e o fluxo de controle entre as estruturas de dados. Müller (Müller, 1996) apresenta uma extensa coletânea de arquiteturas de controle para agentes autônomos, enfocando três linhas principais: agentes reativos, agentes deliberativos e agentes interativos[7]. Em (Sommaruga and Catenazzi, 1995), por exemplo, são apresentadas várias considerações pertinentes à criação de arquiteturas de agentes, no que se refere à estrutura dos agentes, comunicação, tomada de decisões e aspectos computacionais relativos à cooperação entre agentes. Esta subseção apresenta um rápido apanhado de algumas das arquiteturas mais empregadas na especificação de agentes, conforme cita Wooldridge (Wooldridge, 1999).

Arquiteturas Baseadas em Lógica

As decisões são feitas com base em dedução lógica. Estas arquiteturas possuem uma grande desvantagem: a complexidade computacional inerente à prova de teoremas torna questionável sua aplicação em ambientes onde se tenha restrições de tempo (Wooldridge, 1999).

Arquiteturas Reativas

No final dos anos 80 vários pesquisadores começaram a buscar alternativas à Inteligência Artificial simbólica. Tais alternativas, muitas vezes chamadas de *comportamentais*[8] e *situadas*[9] são o que se pode chamar de puramente *reativas*, pois elas simplemente reagem ao ambiente, sem executar nenhum

[5]Do dicionário *Merriam-Webster's*.

[6]Um signo, do ponto de vista semiótico, pode ser caracterizado, de forma bastante simplificada, como qualquer coisa que represente algo para alguém.

[7]Do inglês *interacting agents*.

[8]Criação e combinação de comportamentos.

[9]Agentes situados dentro do ambiente.

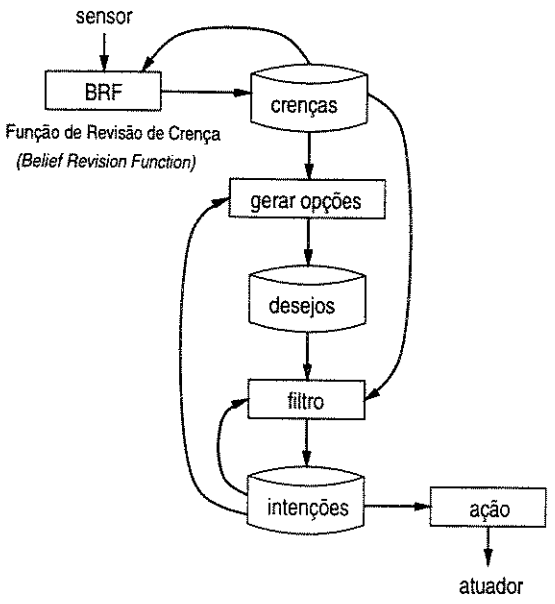


Figura 2.3: Diagrama esquemático de uma arquitetura BDI genérica.

processo de raciocínio. A mais conhecida delas é a *Subsumption Architecture* proposta por Brooks (Brooks, 1990).

Arquiteturas reativas apresentam como vantagens: simplicidade, economia, tratabilidade computacional e robustez contra falhas. No entanto, existem uma série de desvantagens intrinsicamente relacionadas às arquiteturas puramente reativas (Wooldridge, 1999):

- como elas não mantêm um modelo interno do mundo devem conter uma quantidade suficiente de informações para determinar uma ação aceitável;
- é difícil imaginar como uma decisão local pode considerar informações não-locais; e
- não há uma metodologia de projeto bem estabelecida para tais agentes, visto que o comportamente "emerge" de interações com o ambiente.

Arquitetura BDI

Arquiteturas BDI[10] são baseadas em raciocínio prático, no qual o processo de decisão sobre o que fazer assemelha-se ao tipo de racíocínio que seres humanos costumam realizar no dia-a-dia, de acordo com a Figura 2.3.

Um dos principais problemas com as arquiteturas BDI é o processo de ajuste do balanço entre o comprometimento e não-comprometimento das intenções do agente: o processo de deliberação deve ser precisamente ajustado ao seu ambiente garantindo que: (a) em ambientes mais dinâmicos e imprevisíveis ele reconsidere frequentemente suas decisões e (b) em ambientes mais estáticos tenha-se menos reconsideração.

[10] *Belief-Desire-Intention*

O modelo BDI é atrativo por diversas razões (Wooldridge, 1999). Primeiro, ele é intuitivo. Todos nós reconhecemos o processo de decisão do que fazer e como fazer e temos ainda um entendimento informal das noções de crença, desejo e intenção. Segundo, ele apresenta uma decomposição funcional clara que indica quais tipos de subsistemas são necessários para a construção de um agente. A maior dificuldade, como sempre, é saber como implementar estas funções eficientemente.

Arquiteturas em Camadas

Pode-se identificar dois tipos de fluxo de controle em arquiteturas em camadas:

- *horizontalmente em camadas*. Todos os níveis de *software* são conectados à entrada sensorial e ao atuador (Figura 2.4(a)). De fato, cada camada age como um agente sugerindo qual ação executar. De forma a garantir a consistência, geralmente incluem uma função para mediação, responsável por definir qual camada tem o controle do agente a cada instante. Se existem n camadas em uma arquitetura e cada uma delas é capaz de sugerir m ações possíveis então existem m^n interações possíveis a serem consideradas. A Figura 2.5 mostra a arquitetura *TouringMachine*[11] , uma arquitetura em camadas horizontais clássica.

- *verticalmente em camadas*. A entrada e saída são conectadas a no máximo uma camada. Esta arquitetura pode ser divida em arquitetura *unidirecional* (*one pass*) e *bidirecional* (*two pass*). Em arquiteturas de controle "unidirecional" (Figura 2.4 (b)) o controle flui através de cada uma das camadas até a última, produzindo a saída. Em arquiteturas de controle "bidirecional" (Figura 2.4 (c)), a informação flui para cima e o controle para baixo.

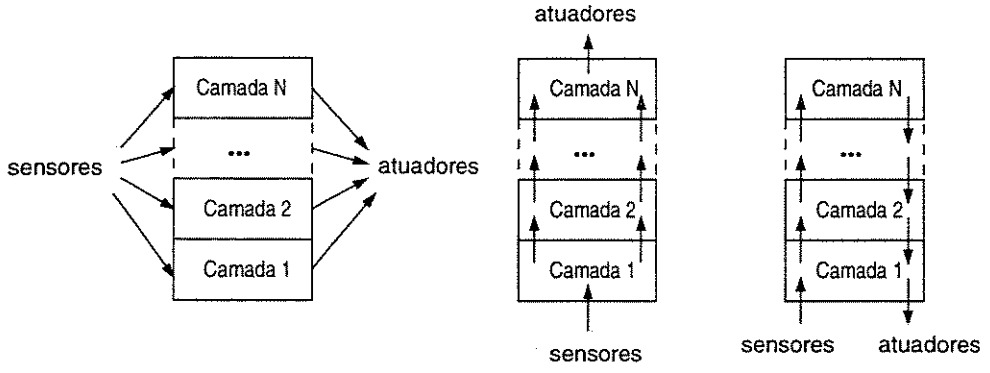


Figura 2.4: Fluxos de informação e controle em arquiteturas de agentes baseados em camadas.

[11] Importante ressaltar que tal nomenclatura não tem relação com a bem conhecida *Máquina de Turing*, desenvolvida pelo matemático A. Turing.

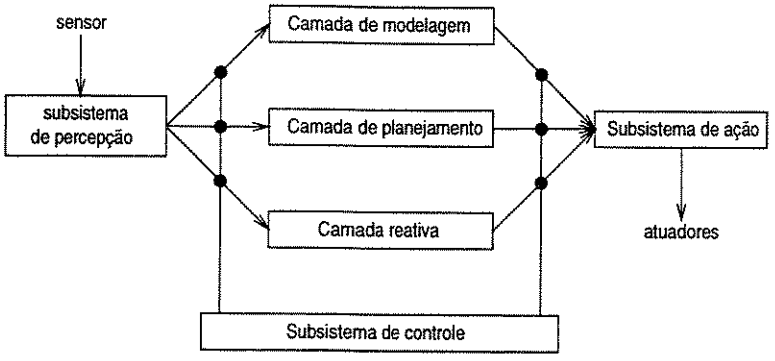


Figura 2.5: *TouringMachines*: uma arquitetura de agentes em camadas horizontais.

2.3 Sistemas Multiagentes

Desde sua criação, a Inteligência Artificial (IA) tem buscado compreender os mecanismos fundamentais que regem o comportamento inteligente. Notadamente nos últimos anos, tem-se dedicado grandes esforços à chamada Inteligência Artificial Distribuída (IAD), um subcampo da IA que trata de uma sociedade de resolvedores de problemas ou agentes que interagem entre si de forma a resolver um problema comum. Tal coleção é comumente denominada de Sistema Multiagente (SMA), ou seja, uma rede de solucionadores de problemas que trabalham coletivamente além das capacidades individuais de cada um.

O interesse crescente em SMAs se deve às vantagens inerentes de tais sistemas, incluindo habilidades em resolver problemas muito grandes para um único agente, resolver problemas inerentemente distribuídos[12], oferer maior velocidade e confiabilidade, oferecer claridade conceitual e simplicidade de projeto (em alguns casos) e tolerar dados e conhecimentos incertos.

2.3.1 Propriedades de um Sistema Multiagente

Em geral, ambientes multiagentes possuem as seguintes características:

i. fornecem uma infra-estrutura especificando protocolos de comunicação e interação.

ii. são tipicamente abertos e não possuem um criador centralizado.

iii. contêm agentes que são autônomos e distribuídos, podendo estes apresentar interesse próprio ou agir de forma cooperativa.

2.3.2 Comunicação Entre Agentes

Agentes se comunicam para alcançar melhores resultados para si próprios ou para a sociedade na qual eles existem. A comunicação permite que agentes coordenem suas ações e comportamento, resultando em sistemas mais coerentes. Existem vários motivos pelos quais a coordenação é essencial em SMAs (Wooldridge, 1999; Green et al., 1997), entre eles pode-se citar:

[12]Para alguns problemas uma abordagem centralizada torna-se tecnicamente impossível, pois sistemas e dados pertencentes a organizações independentes devem ser mantidos privativamente por razões competitivas.

- Nenhum agente possui uma visão global de todo o sistema na medida em que o SMA torna-se mais complexo. Pode ser exemplificado como a seguir: um funcionário de uma grande empresa com grande certeza não é capaz de saber exatamente sobre tudo aquilo que os outros funcionários estão fazendo o tempo todo. Consequentemente, agentes tendem a ter visões parciais do problema. Não apenas é praticamente impossível como às vezes é indesejável permitir que um agente tenha conhecimento sobre todo o sistema, visto que isto implica em um maior número de decisões (e situações) que ele será obrigado a processar[13].
- *alcançar restrições globais.* Sistemas podem apresentar restrições dependentes do comportamento coletivo de todos os agentes. Um SMA para gerenciamento de redes, por exemplo, deve ser capaz de se recuperar após uma falha em um certo período máximo de tempo.
- *agentes possuem habilidades distintas.* Agentes com habilidades e capacidades diferentes devem trabalhar em conjunto para resolver problemas.

Coordenação

Coordenação é a propriedade apresentada por um sistema de agentes ao executar alguma atividade em algum ambiente compartilhado. A grau de coordenação define o quanto ele evita atividades desnecessárias reduzindo a contenção de recursos, travamento (*deadlock*), mantendo condições de consistência de dados. Identificam-se dois tipos básicos de coordenação (Figura 2.6) (Wooldridge, 1999; Green et al., 1997):

i. **Sistema multiagente cooperativo**
Os agentes atuam de forma a maximizar uma meta global do sistema. O enfoque é baseado no desempenho do sistema e não do agente. Esta abordagem pressupõe sociabilidade.

ii. **Sistema multiagente competitivo**
Pela metade da década de 80 começou a surgir uma nova linha de pesquisa dentro da IAD. Alguns pesquisadores começaram a levantar questões sobre *agentes com motivação individual*, geralmente projetados por pesquisadores independentes. Tais agentes são considerados como possuidores de interesse próprio e competitividade (ou não-cooperativo), podendo possuir um comportamento antagônico. Um exemplo de agentes deste tipo seriam os agentes de negócios na Internet.

Rede de Contratos

A técnica de coordenação mais renomada é a *Rede de Contratos* (*Contract Net*) (Smith, 1980), um protocolo de interação para solução de problemas de forma interativa entre agentes. O protocolo de uma Rede de Contratos baseia-se no mesmo mecanismo usado por empresas para regular a troca de bens e serviços entre si, e oferece uma solução para o chamado *problema de conexão*: encontrar o agente adequado para uma dada tarefa. A Figura 2.7 ilustra o processo descrito. Inicialmente, um *contratante* (que geralmente tem um recurso a ser utilizado ou uma tarefa a ser executada) convida outros agentes para a elaboração de propostas. Os agentes dispostos a aceitar a proposta retornam com ofertas. No final, após as devidas considerações, o contratante atribui a tarefa ao agente mais conveniente (geralmente aquele com maior valor pago).

[13]Mesmo o presidente de uma empresa não pode ter o conhecimento de tudo que seus funcionários fazem, mas somente uma visão simplificada de suas atitudes.

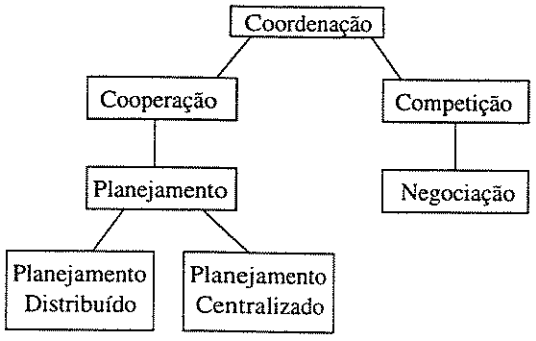


Figura 2.6: Taxonomia de algumas das diferentes maneiras que agentes podem usar para coordenar seu comportamento e atividades.

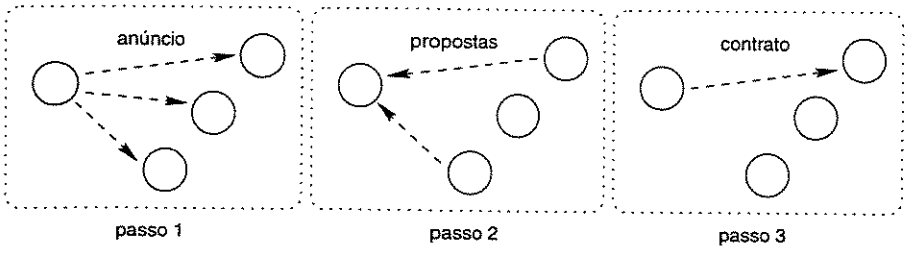


Figura 2.7: Passos básicos no protocolo de interação da Rede de Contratos (Smith, 1980; Wooldridge, 1999).

Sistemas Blackboard

A solução de problemas baseado no modelo *Blackboard* é geralmente apresentada seguindo-se a seguinte metáfora (Corkill, 1991) (ela captura várias das características mais importantes dos sistemas *Blackboard*):

> *Imagine um grupo de especialistas humanos sentados próximos a um grande quadro-negro. Os especialistas trabalham cooperativamente para resolver um problema, usando o quadro-negro como local para o desenvolvimento da solução.*
> *O processo de solução começa quando o problema e os dados iniciais são escritos no quadro-negro. Os especialistas prestam atenção ao quadro-negro, buscando uma oportunidade para aplicar suas habilidades no desenvolvimento da solução. Quando um especialista encontra informação suficiente para fazer uma contribuição, ela é então anotada no quadro-negro, com a idéia de habilitar outros especialistas a também usarem suas habilidades. Este processo de adição de contribuições ao quadro-negro continua até que o problema seja resolvido.*

Técnicas de Negociação

Uma idéia comum nas contribuições da IAD à negociação é que agentes usam a negociação para resolução de conflitos, portanto, para coordenação. Embora a negociação seja bastante importante para a modelagem de sistems multiagentes, não há uma definição clara e amplamente aceita do que seja negociação. Na verdade, existem tantas definições de negociação quantos pesquisadores no assunto. Duas definições básicas são dadas a seguir:

> *"Negociação é o processo pelo qual uma decisão conjunto é alcançada por dois ou mais agentes com objetivos individuais."* *(Weiss, 1999, p104)*

> *"... negociação é o processo de comunicação de um grupo de agentes para alcançar um acordo mutualmente aceito sobre algum assunto."* *(Green et al., 1997)*

O objeto do acordo entre os agentes pode ser o preço, questões militares, local de encontro, uma ação conjunta ou sobre um objetivo conjunto. O processo de busca pode envolver o intercâmbio de informações, a relaxação dos objetivo iniciais, concessões mútuas e até mesmo mentiras e ameaças. Concluindo, a noção mais elementar sobre negociação é expressa por um *consenso*. Outros pesquisadores, no entanto, argumentam que a noção de negociação deve ser associada a um processo de raciocínio sobre as crenças, desejos e intenções de outros agentes.

2.4 Agentes Móveis

O conceito de *agente móvel* (Ghezzi and Vigna, 1997; Shulze and Madeira, 1997; Mira da Silva and Rodrigues da Silva, 1997; Green et al., 1997) pode ser definido, com um certo grau de precisão, como:

> Um agente móvel é uma entidade de *software* situada em um ambiente de software. Esta entidade herda algumas das características de um agente (conforme definido antes). Um agente móvel deve conter cada um dos seguintes modelos: (a) um modelo de agente, (b) um modelo de ciclo de vida, (c) um modelo computacional, (d) um modelo de segurança, (e) um modelo de comunicação e ,finalmente, (f) um modelo de navegação.

Para se estabelecer de forma mais clara a definição anterior define-se ainda a idéia de um ambiente de agentes móveis: um *ambiente de agentes móveis* (AAM) é um sistema computacional distribuído por uma rede de computadores heterogênea. Sua tarefa principal é fornecer um ambiente no qual agentes móveis possam executar suas funções. Um AAM implementa a maioria dos modelos que aparecem na definição de agente móvel, podendo ainda prover:

1. suporte a serviços que se relacionam com o próprio AAM,
2. suporte a serviços que pertencem ao ambiente no qual o AAM é construído,
3. suporte a serviços que permitam o acesso a outros AAMs e
4. capacidade de abertura, para acesso a ambientes que não sejam baseados em agentes móveis.

As definições anteriores afirmam sobre os aspectos que dão a essência de um agente móvel e de um AAM, bem como um resumo dos paradigmas utilizados em agentes móveis. Uma comparação dos paradigmas de código móvel pode ser encontrada na Tabela 2.1.

Paradigma	*lado A*	*lado B*
cliente/servidor	-	***know-how*** **recursos** **processador**
avaliação remota	*know-how*	**recursos** **processador**
código sob demanda	**recursos** **processador**	*know-how*
agentes móveis	*know-how* processador	**recursos**

Tabela 2.1: Paradigmas de código móvel. A tabela mostra a localição das capacidades antes de serem executadas. As posições em negrito na tabela indicam onde a computação associada acontece (Ghezzi and Vigna, 1997).

2.5 Resumo

Este capítulo apresentou uma rápida visão de alguns dos elementos da teoria de agentes, no que se refere a:

- noções intuitivas básicas sobre a definição de agente;
- comparação entre os conceitos vigentes para os termos *objeto* e *agente*;
- revisão de algumas das mais utilizadas arquiteturas de agentes;
- introdução a sistemas multiagentes e coordenação entre agentes; e
- noções básicas sobre o paradigma de código móvel no contexto de sistemas multiagentes.

Capítulo 3

Redes de Agentes

Este capítulo apresenta uma revisão do modelo conceitual de *rede agentes* proposto por Guerrero (Guerrero, 2000), e tem como objetivo contribuir com aprimoramentos em diversos aspectos, entre eles: maior identificação com a noção de agência, formalização de alguns conceitos até então definidos informalmente e dinâmica de funcionamento mais completa. Para o desenvolvimento deste trabalho optou-se pela mesclagem dos conceitos anteriores (Gudwin, 1996; Guerrero, 2000) com os novos neste mesmo capítulo, pois julga-se mais adequado ao leitor uma apresentação tão contínua quanto possível dos conceitos envolvidos. Por este motivo serão introduzidos comentários, onde se fizer necessário, que permitam ao leitor identificar as contribuições aqui propostas.

3.1 Introdução

As *redes de objetos* (ROs) foram introduzidas inicialmente por Gudwin (Gudwin, 1996) como uma ferramenta computacional para a representação e processamento de estruturas de conhecimento sob o paradigma da Semiótica Computacional (Gudwin, 1996; Gudwin and Gomide, 1997a,b). Basicamente, uma RO é formada por um agrupamento de *objetos matemáticos*, que são entidades que podem apresentar estados e funções de transformação que atuam nestes estados[1], uma topologia constituída de *lugares* e *arcos* conectados entre si e um mecanismo que rege o comportamento dos objetos. Este comportamento, isto é, quais objetos disparam e com quem eles interagem, é implementado pela assim chamada *função de seleção* (FS), que é definida para todos os objetos da rede. A noção de FS é na verdade a representação genérica do comportamento dos objetos dado um conjunto de restrições[2], de forma que uma RO pode modelar comportamentos adaptativos sofisticados, pois a FS pode ser definida no tempo. Esta generalidade, a princípio, diferencia as ROs das Redes de Petri, onde os elementos ativos (dados pelas transições) são especificadas *a priori* na definição da rede[3].

A construção das funções de seleção para cada um dos objetos é responsabilidade do projetista do modelo. Assim, neste contexto, torna-se importante o estudo de ferramentas auxiliares que permitam

[1]Estes objetos apresentam diversas propriedades inspiradas no paradigma de programação orientada a objetos (Pascoe, 1990; Stefik and Bobrow, 1990; Snyder, 1993)

[2]Estas restrições foram introduzidas em (Gudwin, 1996).

[3]Interessante destacar a grande diversidade de modelos baseados em Redes de Petri (Gerogiannis et al., 1998; Esser, 1997; Lakos, 1995b), baseados em: modificação de semântica, incerteza, temporalidade, estruturação, incorporação de noções de orientação a objeto, etc. O Capítulo 4 comenta rapidamente alguns destes modelos.

uma aplicação mais direta do modelo de ROs em domínios específicos. Um primeiro passo nesta direção foi apresentado em (Guerrero et al., 1999), baseando-se na proposta de um algoritmo[4] para determinação de alguns dos componentes da FS. Posteriormente, em (Guerrero, 2000), esta idéia foi então integrada ao modelo original de rede de objetos, dando origem às *redes de agentes* (RAs). Esta abordagem baseia-se na hipótese fundamental de que todos os objetos são capazes de avaliar de maneira autônoma o grau de utilidade de suas ações potenciais. A partir destas informações o algoritmo BMSA determina quais das propostas iniciais serão efetivamente executadas. A nomenclatura "redes de agentes" reflete o comportamente autônomo de um objeto ativo, pois, neste caso, cada um deles possui um certo nível de autonomia, ditada pelo interesse próprio[5].

Um dos mais importantes exemplos de modelo implementado sob o paradigma de ROs vem a ser um sistema de controle de um veículo autônomo (Gudwin, 1996). Outros modelos baseados em ROs e RAs foram propostos para a simulação de Redes de Petri lugar-transição (PT-net) (Gudwin, 1996), Redes de Petri Coloridas (CPN) (Guerrero, 2000), processos de raciocínio, computação com palavras, redes neurais, sistemas nebulosos e algoritmos genéticos (Gudwin, 1996; Gudwin and Gomide, 1997c, 1998a; Guerrero, 2000).

3.1.1 Organização do Capítulo

A Seção 3.2 inclui as definições preliminares, extraídas em sua maioria de (Gudwin, 1996; Guerrero, 2000). A Seção 3.3 apresenta a revisão conceitual de objetos matemático e a proposta de *agente formal*. A Seções 3.4, 3.5 e 3.6 discutem sistemas de agentes, redes de agentes e aspectos formais referentes à dinâmica de uma RA, respectivamente. A Seção 3.7 contém uma retrospectiva geral dos temas abordados no capítulo.

3.2 Definições Preliminares

Esta seção apresenta as definições matemáticas gerais. A maioria delas foi introduzida em (Gudwin, 1996) e algumas em (Guerrero, 2000)[6]. Como contribuições deste trabalho pode-se citar as definições 3.16, 3.17, 3.18, 3.19 e 3.20. As definições anteriores foram incluídas aqui neste capítulo apenas por motivos didáticos.

Estas definições são feitas para um domínio discreto N, normalmente associado a instantes de tempo ou passos de algoritmos. As extensões para um domínio contínuo são possíveis, mas não serão tratadas aqui. Outra observação é que não se faz uma maior distinção entre os conceitos de funcão e grafo de uma função. Sendo assim, utilizar-se-á as notações $f : A \rightarrow B$ e $f \subseteq A \times B$ em diferentes situações para representar a mesma função f, conforme se deseje ressaltar a característica funcional ou o fato de uma função também ser um conjunto.

DEFINIÇÃO 3.1 (ÊNUPLA)

Sejam $q_1, q_2, \ldots, q_n$ elementos genéricos pertencentes aos conjuntos $Q_1, Q_2, \cdots, Q_n$, respectivamente, e $Q = Q_1 \times Q_2 \times \ldots \times Q_n$. Define-se uma **ênupla** $q \in Q$ como um agrupamento

[4] Denominado *best matching search algorithm* (BMSA).

[5] Baseada na hipótese do *agente benevolente*, que não usa informações incorretas ou maliciosas para um eventual favorecimento.

[6] Neste caso apenas as definições 3.10 e 3.11.

ordenado de elementos na forma:

$$q = (q_1, q_2, \ldots, q_n) \tag{3.1}$$

Observações importantes:

- Qualquer elemento de uma ênupla (ou componente) pode ser referenciado por seu índice de referência (Definição 3.3);
- Uma ênupla com apenas um elemento é o próprio elemento, $q_1 \equiv (q_1)$;
- Uma ênupla formada apenas de elementos simples, isto é, que não sejam outras ênuplas, é chamada *ênupla simples*;
- Componentes de uma ênupla podem também ser ênuplas. Neste caso forma-se uma *ênupla complexa*, como $q = (q_1, (q_{21}, q_{22}), q_3)$; e
- Uma ênupla não é um conjunto.

DEFINIÇÃO 3.2 (ARIDADE DE UMA ÊNUPLA)

Seja uma ênupla $q = (q_1, \ldots, q_n)$. Define-se[7] a **aridade** de uma ênupla q, representada por $\mathrm{Ar}(q)$, como o número de elementos que constituem a ênupla q. Por exemplo, para a ênupla q tem-se $\mathrm{Ar}(q) = n$.

Exemplos de aridade:

- $q = (a, b, c)$, $\mathrm{Ar}(q) = 3$
- $q = (a, (b, c), d)$, $\mathrm{Ar}(q) = 3$
- $q = ((a, b, c), (d, (e, f), g))$, $\mathrm{Ar}(q) = 2$

DEFINIÇÃO 3.3 (ÍNDICE DE REFERÊNCIA)

Um **índice de referência** permite a localização de qualquer elemento de um ênupla. No caso de uma ênupla simples q tem-se $1 \leq i \leq \mathrm{Ar}(q)$. Se q for uma ênupla complexa o próprio índice de referência i poderá ser uma outra ênupla, onde cada elemento i_k de i representa o subíndice dentro da ênupla q_k. Analogamente, cada subíndice pode assumir valores entre 1 e a aridade da ênupla q_k. Veja a figura 3.1 para um exemplo de ênupla complexa.

DEFINIÇÃO 3.4 (FÓRMULA DE INDUÇÃO)

Uma **fórmula de indução** é uma expressão sobre os elementos de uma ênupla $q = (q_1, q_2, \ldots, q_n)$, de acordo com a gramática descrita abaixo:

[7] A Definição 3.20 apresenta uma pequena adição a este conceito.

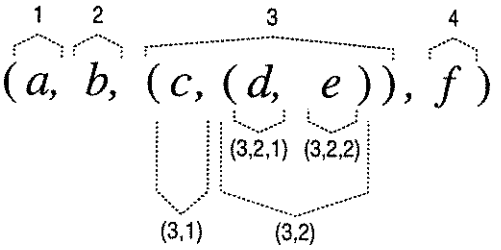


Figura 3.1: Índices de referência em uma ênupla complexa.

```
k ::= [ <EXP> ]
<EXP> ::= [ <EXP>, <EXP> ] |
          <EXP>, <EXP> |
          <IR>
```

onde o símbolo terminal `<IR>` é um índice de referência de q.

DEFINIÇÃO 3.5 (INDUÇÃO DE UMA ÊNUPLA)

Sejam:

- Uma ênupla $q = (q_1, q_2, \ldots, q_n)$, definida em $Q = Q_1 \times Q_2 \times \cdots \times Q_n$, e
- Uma fórmula de indução k.

Define-se a **indução** de q segundo a fórmula de indução k como uma nova ênupla $q_{(k)}$ na qual os componentes de q são reagrupados seguindo-se a fórmula k:

(1) substitui-se os colchetes por parênteses e

(2) os índices de referência i_j pelos respectivos componentes q_{i_j} da ênupla q.

O domínio $Q_{(k)}$ de $q_{(k)}$ também pode ser obtido seguindo-se a fórmula de indução k:

(1) omite-se os colchetes externos da fórmula, substituindo-se os internos por parênteses,

(2) as vírgulas pelo sinal de produto Cartesiano '$\times$' e

(3) os índices i_j pelo subdomínios originais Q_{i_j} de Q.

Exemplos de indução para $q = (a, b, c, d)$, $Q = Q_1 \times Q_2 \times Q_3 \times Q_4$:

- $k_1 = [1, 3, 4, 2]$, $q_{(k_1)} = (a, c, d, b)$, $Q_{(k_1)} = Q_1 \times Q_3 \times Q_4 \times Q_2$
- $k_2 = [4, 1]$, $q_{(k_2)} = (d, a)$, $Q_{(k_2)} = Q_4 \times Q_1$
- $k_3 = [1, [2, 3], 4]$, $q_{(k_3)} = (a, (b, c), d)$, $Q_{(k_3)} = Q_1 \times (Q_2 \times Q_3) \times Q_4$

DEFINIÇÃO 3.6 (SUBÊNUPLA)

Sejam q uma ênupla e k uma fórmula de indução formada apenas por índices unários (números inteiros) sem repetição e apenas um par de colchetes. Uma **subênupla** de q é dada pela indução de q segundo k, representado neste caso por $q_{(k)}$.

DEFINIÇÃO 3.7 (RELAÇÃO)

Sejam:

- n conjuntos $R_1, R_2, \ldots, R_n$, $n > 1$, e
- $R = \{(r_{i1}, \ldots, r_{in})\}$, $i = 1, \ldots, M$, um conjunto de M ênuplas de aridade n, onde $\forall i \in \{1, \ldots, M\}$, $\forall k \in \{1, \ldots, n\}$ tem-se $r_{ik} \in R_k$.

O conjunto R, $R \subseteq R_1 \times \cdots \times R_n$, é dito ser uma **relação** em $R_1 \times \cdots \times R_n$ e o produto Cartesiano $R_1 \times \cdots \times R_n$ é chamado de *universo da relação*.

DEFINIÇÃO 3.8 (PROJEÇÃO DE UMA RELAÇÃO)

Sejam:

- Uma relação n-ária $R = \{r_i\}$, $r_i = (r_{i1}, \ldots, r_{in})$ definida em $R_1 \times \ldots \times R_n$, e
- k uma fórmula de indução formada por índices unários (inteiros) sem repetição, na forma $k = [k_1, \ldots, k_m]$, onde:
  - $1 \leq k_i \leq n$ para $i = 1, \ldots, m$
  - $k_i = k_j \Rightarrow i = j$, para $i = 1, \ldots, m$ e $j = 1, \ldots, m$.

Define-se a **projeção** de R segundo k, $R \downarrow k$, ou alternativamente $R_{(k)}$ – ou ainda $R \downarrow R_{k_1} \times \cdots \times R_{k_m}$ – como a relação obtida pela união de todas as ênuplas $r_{i(k)} = (r_{ik_1}, \ldots, r_{ik_m})$ geradas a partir da indução de cada ênupla R segundo k:

$$R_{(k)} = \bigcup_i r_{i(k)}$$

Observações:

- Os elementos de $R_{(k)}$ podem aparecer em qualquer ordem;
- A aridade das ênuplas de $R_{(k)}$ sempre será igual ou menor do que a aridade das ênuplas de R;
- A cardinalidade de $R_{(k)}$ será sempre menor ou igual à cardinalidade de R.

DEFINIÇÃO 3.9 (PROJEÇÃO LIVRE DE UMA RELAÇÃO)

Sejam:

- Uma relação n-ária $R = \{r_i\}$, $r_i = (r_{i1}, \ldots, r_{in})$ definida em $R_1 \times \ldots \times R_n$, e

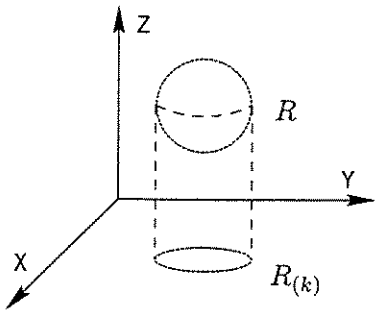


Figura 3.2: Projeção de uma relação $R \subseteq X \times Y \times Z$ no espaço $X \times Y$ dada por uma fórmula de indução $k = [1, 2]$.

- k uma fórmula de indução qualquer.

Define-se a **projeção livre** de R segundo k, $R \downarrow k$, ou alternativamente $R_{(k)}$, como a relação obtida pela união de todas as ênuplas $r_{i(k)}$ obtidas a partir da indução de cada ênupla R segundo k:

$$R_{(k)} = \bigcup_i r_{i(k)}$$

Uma projeção livre de uma relação diferencia-se de uma operação de projeção normal por *não impor restrições* sobre a fórmula de indução aplicada. Uma projeção normal é um caso especial de uma projeção livre, onde somente ênuplas induzidas por fórmulas formadas por índices unários são consideradas, o que implica em ênuplas definidas sobre as dimensões principais do domínio da relação.

DEFINIÇÃO 3.10 (FÓRMULA DE EXTENSÃO)

Uma fórmula de extensão é uma expressão sobre os elementos de uma ênupla $q = (q_1, q_2, \ldots, q_n)$, de acordo com a gramática descrita a seguir:

```
k ::= [ <EXP> ]
<EXP> ::= [ <EXP>, <EXP> ] |
          <EXP>, <EXP> |
          <IR> |
          <CONJ>
```

Onde o símbolo terminal `<IR>` representa um índice de referência em q e `<CONJ>` um conjunto não-vazio qualquer.

Exemplos de fórmulas de extensão:

- $k = [1, 2, 4]$
- $k = [1, X]$

- $k = [1, [3, V_1], V_2]$

DEFINIÇÃO 3.11 (EXTENSÃO DE UMA ÊNUPLA)

Sejam:

- Uma ênupla $q = (q_1, \ldots, q_n)$ definida em $Q_1 \times \cdots \times Q_n$;
- $C = \{C_1, \ldots, C_m\}$, onde cada C_j é um conjunto não-vazio;
- Uma fórmula de extensão k onde qualquer elemento de k que não seja um índice de referência em q seja dado por um C_j tal que $C_j \in C$.

Define-se a **extensão** de q segundo k como um conjunto representado por $q^{(k)}$ formado da seguinte maneira:

- **Passo 1**: seguindo-se a fórmula de extensão k, substitui-se os colchetes por parênteses e os índices de referência pelos respectivos componentes da ênupla q. Obtem-se então a *ênupla geradora* $\hat{\omega}$. Esta ênupla terá como componentes elementos originais de q (que foram substituídos pelos índices de referência) ou conjuntos (que faziam parte de k);
- **Passo 2**: gera-se o conjunto $q^{(k)} = \{\omega_i\}$. Cada ω_i é construído como descrito:
  - Sejam:
    - J o conjunto dos índices de referência unários[8] para a ênupla $\hat{\omega}$;
    - $J' \subseteq J$ o conjunto de índices de referência de $\hat{\omega}$ que especifiquem valores originais de q;
    - $J'' \subseteq J$ o conjunto de índices de referência de $\hat{\omega}$ que especifiquem conjuntos (elementos de C);
  - **Passo 2.1** cria-se uma ênupla w', que segue a mesma estrutura de $\hat{\omega}$, onde cada um de seus componentes w'_j recebe (para todos os valores $j \in J$):

$$\omega'_j = \begin{cases} \hat{\omega}_j & \text{caso } j \in J' \\ \omega \in \hat{\omega}_j & \text{caso } j \in J'' \end{cases}$$

  - **Passo 2.2** obtem-se $\omega_i = \omega'$.
  - **Passo 2.3** repete-se o passo anterior até que não exista nenhuma combinação restante de elementos $w \in \hat{\omega}_j$.

Exemplos:

- $q = (a, b)$, $V = \{\alpha, \beta, \gamma\}$, $k = [1, 2, V]$:
  - Passo 1: $\hat{\omega} = (q_1, q_2, V) \Rightarrow \hat{\omega} = (a, b, V)$
  - Passo 2: $q^{(k)} = \{(a, b, \alpha), (a, b, \beta), (a, b, \gamma)\}$.

[8] que não sejam outra ênupla.

- $q = (a, b, c)$, $E = \{\delta\}$, $X = \{\alpha, \beta\}$, $k = [1, E, X]$:
  - Passo 1: $\hat{\omega} = (q_1, E, X) \Rightarrow \hat{\omega} = (a, E, X)$
  - Passo 2: $q^{(k)} = \{(a, \delta, \alpha), (a, \delta, \beta)\}$.
- $q = (a, b, c)$, $E = \{\delta\}$, $k = [E]$:
  - Passo 1: $\hat{\omega} = (E)$
  - Passo 2: $q^{(k)} = \{(\delta)\}$.

DEFINIÇÃO 3.12 (EXTENSÃO CILÍNDRICA DE UMA RELAÇÃO)

Sejam:

- $R = \{r_i\}$, $r_i = \{r_{i1}, \ldots, r_{in}\}$, uma relação n-ária definida em $R_1 \times \cdots \times R_n$;
- $V = \{V_1, \ldots, V_s\}$ uma coleção de conjuntos não-vazios; e
- uma fórmula de extensão k.

Define-se a **extensão cilíndrica** de R segundo k, $P = R \uparrow k$, ou alternativamente $R^{(k)}$, como a relação obtida pela união de todas as extensões de todas as ênuplas r_i segundo k:

$$R^{(k)} = \bigcup_i r_i^{(k)}$$

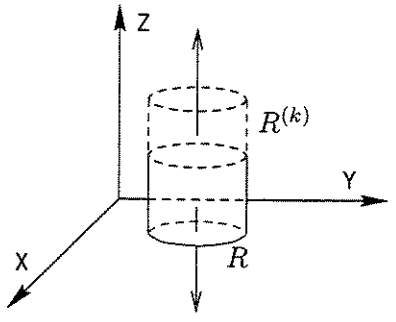


Figura 3.3: Extensão cilíndrica de uma relação $R \subseteq X \times Y$ no espaço $X \times Y \times Z$, dada por uma fórmula de extensão $k = [1, 2, Z]$.

DEFINIÇÃO 3.13 (JUNÇÃO DE RELAÇÕES)

Sejam:

- $R = \{r_i\}$, $r_i = (r_{i1}, \ldots, r_{in})$, uma relação n-ária definida em $R_1 \times \cdots \times R_n$;
- $S = \{s_j\}$, $s_j = (s_{j1}, \ldots, s_{jm})$, uma relação m-ária definida em $S_1 \times \cdots \times S_m$; e

- k_1 e k_2 duas fórmulas de extensão tais que:

$$R_1 \times \cdots \times R_n \uparrow k_1 = P \quad \text{e} \quad S_1 \times \cdots \times S_m \uparrow k_2 = P$$

Define-se a **junção das relações** R e S em P, denotada por $R \star S\big|_P$, como sendo:

$$R \star S\big|_P = (R \uparrow P) \cap (S \uparrow P)$$

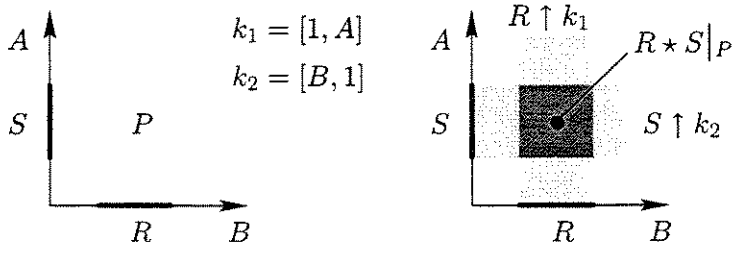


Figura 3.4: Junção das relações $S \subset A$ e $R \subset B$ em $P = A \times B$.

Exemplos de junção de relações:

- Considere:
  $A = \{1, 2\}$, $B = \{a, b, c\}$, $C = \{\alpha, \beta, \gamma\}$
  $R = \{(1, a), (2, c)\} \subset A \times B$
  $S = \{(a, \alpha), (b, \beta)\} \subset B \times C$

- **Ex.1**:
  $k_1 = [1, 2, C]$, $k_2 = [A, 1, 2]$, $P = A \times B \times C$
  $R \star S\big|_{A \times B \times C} = \{(1, a, \alpha)\}$

- **Ex.2**:
  $k_1 = [1, 2, B, C]$, $k_2 = [A, B, 1, 2]$, $P = A \times B \times B \times C$
  $R \star S\big|_{A \times B \times B \times C} = \{(1, a, a, \alpha), (1, a, b, \beta), (2, c, a, \alpha), (2, c, b, \beta)\}$

DEFINIÇÃO 3.14 (VARIÁVEL)

Sejam:

- T um conjunto enumerável e
- $X \subseteq U$ um subconjunto de um universo U.

Define-se uma **variável** x de tipo X em T como uma função que mapeia elementos de T em X:

$$x : T \to X$$

DEFINIÇÃO 3.15 (VARIÁVEL DE CONJUNTO)

Sejam:

- T um conjunto enumerável e
- $X \subseteq U$ um subconjunto de um universo U.

Define-se uma **variável de conjunto** x de tipo X em T como uma função que mapeia elementos de T em 2^X:

$$x : T \rightarrow 2^X$$

Na definição original de uma variável no tempo, dada por $v : \mathbf{N} \rightarrow V$, cada instância temporal $v(n)$ assume valores em V. Este trabalho introduz a noção de variáveis que podem assumir valores *vazios*. Seu significado pode ser analogamente comparado a um container. Tal característica pode ser obtida a partir de uma convenção, na qual, a partir do conjunto V inicial no qual a variável em questão é definida, adiciona-se um elemento $\varepsilon \notin V$. Então define-se uma nova variável $v^\star : \mathbf{N} \rightarrow V \cup \{\varepsilon\}$. As definições a seguir formalizam o processo.

DEFINIÇÃO 3.16 (CONJUNTO $\star$)

Seja um conjunto V não-vazio. Um **conjunto** $V^\star$ é dado pela união de V com um conjunto unitário contendo apenas o símbolo indicador de nulo convencionado por ε:

$$V^\star = V \cup \{\varepsilon\} \tag{3.2}$$

A partir deste ponto em diante, no desenvolvimento deste trabalho, sempre que se escrever um conjunto com o símbolo $\star$ superscrito, tal como $V^\star$, considerar-se-á que exista um conjunto V tal que $V^\star = V \cup \{\varepsilon\}$.

DEFINIÇÃO 3.17 (VARIÁVEL $\star$)

Seja um conjunto $V^\star$ e ε o símbolo indicador de nulo. Uma **variável** $v^\star$ é dada por um mapeamento:

$$v^\star : \mathbf{N} \rightarrow V^\star \tag{3.3}$$

Adicionalmente, para qualquer $n \in \mathbf{N}$ tal que $v^\star(n) = \varepsilon$, diz-se que a variável $v^\star$ é nula (ou vazia) em n.

DEFINIÇÃO 3.18 (FÓRMULA DE INDUÇÃO $\star$)

Uma **fórmula de indução** $\star$ é uma expressão $k^\star$ sobre os elementos de uma ênupla $q = (q_1, q_2, \ldots, q_n)$, de acordo com a gramática descrita abaixo:

```
k*  ::= [ <EXP> ] | [ ]
<EXP> ::= [ <EXP>, <EXP> ] |
          <EXP>, <EXP> |
          <IR>
```

onde o símbolo terminal <IR> é um índice de referência de q.

DEFINIÇÃO 3.19 (FÓRMULA DE INDUÇÃO $\star$ SIMPLES)

Uma **fórmula de indução $\star$ simples** é uma fórmula de indução $\star$ dada por uma expressão $k^\star$ sobre os elementos de uma ênupla $q = (q_1, q_2, \ldots, q_n)$ com as seguinte restrições: (a) deve seguir a gramática restrita dada a seguir e (b) todos os índices de referência devem ser unários e únicos na fórmula, isto é, sem repetição.

```
k*  ::= [ <EXP> ] | [ ]
<EXP> ::= <EXP>, <EXP> |
          <IR>
```

onde o símbolo terminal <IR> é um índice de referência unário e único em q.

DEFINIÇÃO 3.20 (INDUÇÃO $\star$ DE UMA ÊNUPLA)

Seja um símbolo ε convencionado como indicador de nulo. A indução de uma ênupla q por uma fórmula $k^\star$ é dada por:

- se $k^\star = [\,]$ então $q_{(k^\star)} = \varepsilon$,
- caso contrário $q_{(k^\star)}$ segue o processo geral de indução de uma ênupla[9].

Adicionalmente, define-se $\mathrm{Ar}(\varepsilon)$ como zero[10].

3.3 Agente Formal

A partir do conceito de variável (Definição 3.14) pode-se definir o conceito de objeto, que está intimamente ligado ao conceito físico de objeto (Gudwin, 1996).

A Subseção 3.3.1 introduz alguns conceitos intuitivos sobre objetos e redes de objetos, conforme definidos (Gudwin, 1996). A Subseção 3.3.2 propõe uma classificação ontológica preliminar para evitar eventuais conflitos de nomenclatura nos termos empregados. A Subseção 3.3.3 define o agente formal.

3.3.1 Propriedades de Objetos

Conforme descrito, a idéia de agente formal, por derivar diretamente daquela de objeto matemático, também está intimamente ligada ao conceito físico de objeto. Objetos podem representar quaisquer conceitos concretos ou abstratos que possam ser caracterizados por seus atributos e comportamento ao longo do tempo. Gudwin propôs em (Gudwin, 1996) um conjunto de propriedades gerais que podem ser aplicadas a objetos, formando a base de hipóteses sobre as quais as ROs (Gudwin, 1996) e RAs (Guerrero, 2000) foram formuladas:

[9] Conforme a Definição 3.5.

[10] Conforme a noção de aridade apresentada pela Definição 3.2.

- **Cada objeto é único e identificado por seu nome.** Outro aspecto importante é que nomes podem, sem perda de representatividade, ser reutilizados. Se o objeto referenciado por um certo nome é destruído seu nome pode ser atribuído a outro objeto. Mas um objeto **nunca** pode mudar seu nome durante toda sua existência.

- **Cada objeto possui um conjunto de atributos ou partes.** Seguindo a noção de *frame-of-reference*, os atributos são características em diferentes domínios. Além disso, objetos podem conter estruturas, desde subpartes simples até estruturas recursivas. Entretanto, para finalizar a recursão, em algum ponto da cadeia, as partes devem ser somente atributos simples[11].

- **Um objeto pode possuir um conjunto de funções de transformação.** Objetos podem eventualmente conter funções de transformação, utilizadas para manipular seu estado interno. Um objeto com pelo menos uma função de transformação é chamado **objeto ativo**. Caso contrário, **objeto passivo**.

- **Um objeto do sistema pode consumir/gerar outro objeto.** Esses princípios, em conjunto com o conceito de função de transformação, definem o caráter ativo do objeto no sistema. Objetos ativos que consomem e não geram outros objetos são chamados **objetos vertedouro** e objetos que geram e não consomem outros objetos são chamados **objetos fonte**.

- **Objetos podem ser classificados hierarquicamente em função dos seus atributos e funções de transformação.** Pode-se identificar dois tipos de hierarquias: (a) de partes, onde um objeto é formado por partes e subpartes, e (b) de tipo, onde partes de um objeto possuem os mesmos domínios de atributos, funções de transformação, etc.

- **A interação entre objetos se limita ao consumo e geração de novos objetos.** Nenhum mecanismo adicional é necessário para que o sistema evolua no tempo.

Snyder, em (Snyder, 1993), apresenta uma outra possível caracterização para o conceito de objeto. Segundo esta proposta Gudwin (Gudwin, 1996) argumenta que: (a) elas podem ser encapsuladas pela proposta apresentada nesta seção e (b) de certa forma, elas são mais adequadas ao contexto de linguagens de programação, já que muitos dos conceitos são mais genéricos.

3.3.2 Objeto Convencional versus Objeto Matemático

Objetos matemáticos (Gudwin, 1996), quando comparados a objetos convencionais[12], apresentam uma vantagem importante: sua **capacidade de agir autonomamente** em um ambiente compartilhado por vários objetos (Guerrero, 2000). A abordagem adotada na definição original de uma RA (Guerrero, 2000), por seguir à risca os moldes originais lançados em (Gudwin, 1996), não trata exatamente de definir um modelo formal de agente, mas de impor um conjunto de restrições ao modelo de objeto matemático original de forma que este se comportasse como um agente com interesse próprio. Este capítulo trata da obtenção de um modelo de objeto matemático em que se identifique de forma

[11] Este comportamento pode ser constatado em linguagens de programação orientadas a objeto (C++, SmallTalk, Java, etc.). Todas possuem suporte a estruturas recursivas, mas suportam tipos fundamentais para encerrar a recursividade, como valores numéricos, caracteres, ponteiros, lógicos, etc.

[12] Convencional segundo a idéia original de sistemas orientados a objeto (Pascoe, 1990; Stefik and Bobrow, 1990; Snyder, 1993).

mais transparente a noção de agência. Com isto objetiva-se uma integração que propicie uma base formal mais adequada a futuros desenvolvimentos. A implementação desta integração exige uma diferenciação clara e consistente no conjunto de termos adotados.

Uma Ontologia para Objetos Matemáticos

O dicionário *American Heritage* define "ontologia" como "o ramo da metafísica que trata da natureza do ser". O termo foi recentemente adotado pela comunidade de Inteligência Artificial para se referir a um conjunto de conceitos ou termos que podem ser usados para descrever alguma área do conhecimento ou construir uma representação dele. Uma ontologia pode ser de alto nível, consistindo de conceitos que organizam as partes superiores de uma base de conhecimento ou pode ser específica a domínio, tal como uma ontologia de veículos.

Uma ontologia provê uma estrutura básica na qual a base de conhecimento pode ser construída (Chandrasekaran et al., 1999). Uma ontologia contém um conjunto de conceitos e termos que descrevem o domínio em questão, enquanto uma base de conhecimento usa estes termos para representar aquilo que é verdadeiro sobre algum mundo real ou hipotético.

Conforme discutido na Subseção 2.2.2, ainda existe muita discussão sobre a conceituação dos termos "objeto" e "agente". Estando este trabalho intensamente baseado nos significados e relacionamentos entre estes dois termos propõe-se uma pequena classificação ontológica. Assim, sob esta ótica, tem-se a idéia de objeto matemático como elemento geral, englobando ambos os objetos passivos e objetos ativos[13] (Figura 3.5). Estes últimos aqui são tratados, pelos motivos anteriormente descritos, como *agentes formais*.

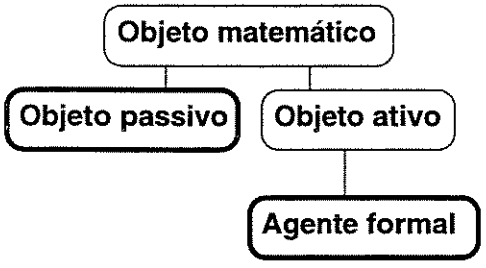


Figura 3.5: Possível visão ontológica da classificação de objeto matemático e agente formal na perspectiva deste trabalho.

É importante ressaltar que este trabalho não se propõe a encerrar a definição de agência e sua formalização da concepção de agente formal. Ele preocupa-se em oferecer um modelo simples o suficiente para permitir o estudo e implementação de arquiteturas para síntese de sistemas inteligentes. Além disso, busca ser flexível o bastante para permitir a continuidade natural em futuros trabalhos.

[13]Segundo (Gudwin, 1996) um objeto é ativo caso tenha a capacidade de alterar seu estado por si próprio, ou seja, se possui pelo menos uma função de transformação, por outro lado, objetos passivos só podem ser alterados por intermédio da ação de um objeto ativo.

3.3.3 Definição de Agente Formal

Matematicamente, um agente formal é semelhante a um objeto matemático (Gudwin, 1996), sendo também descrito por uma ênupla. A diferença está na semântica de cada um dos componentes desta ênupla, que pode ser um sensor, atuador, estado interno ou uma função de transformação, conforme será definido a seguir. Apesar de preservar a base conceitual sobre a qual a formalização de objeto matemático foi construída, grande parte das definições anteriores (Gudwin, 1996; Guerrero, 2000) teve de ser retrabalhada. Assim, de forma análoga à Seção 3.2, esta subseção incluirá, de acordo com a necessidade, definições introduzidas em (Gudwin, 1996) e (Guerrero, 2000), cabendo comentários onde se julgar necessário.

Resumidadamente, as principais contribuições apresentadas nesta subseção consistem em:

- diferenciação dos componentes da ênupla (que descreve o objeto) destinados à entrada e saída de informações (pela incorporação da noção de *sensor* e *atuador*), bem como armazenamento de estados;
- mecanismo de avaliação de utilidade diretamente ligado à função de transformação, inspirado no algoritmo BMSA (Guerrero et al., 1999; Guerrero, 2000);
- tratamento dos estados internos como repositórios de conteúdo, permitindo uma especificação dinâmica mais clara (pela inclusão do indicador de nulo ε); e
- reformulação de alguns conceitos (e exclusão de alguns) anteriores com o objetivo de facilitar futuras extensões.

Seguindo a proposta de *descritor de classe* introduzida inicialmente por Guerrero (Guerrero, 2000) apresenta-se o descritor de classe adaptado à nova proposta.

DEFINIÇÃO 3.21 (DESCRITOR DE CLASSE)

Considere os seguintes tipos de componentes da ênupla que compõem a descrição matemática de um agente formal :

- **sensor** – representa uma *porta de entrada de objetos* pela qual um agente formal pode perceber seu ambiente externo. Cada sensor é capaz de perceber objetos que sejam da *mesma classe* para o qual tenha sido definido;
- **atuador** – serve como *porta de saída para objetos* gerados pelo agente. Cada atuador transporta objetos da mesma classe para o qual foi definido;
- **estado interno** – armazena informações que descrevem uma instância temporal do objeto;
- **função de transformação** – atua sobre os estados internos, podendo eventualmente ter acesso aos sensores e atuadores.

Neste contexto, um **descritor de classe** corresponde a uma ênupla contendo a *especificação genérica* dos domínios das propriedades dos agentes pertencentes a uma classe de

agentes, no seguinte formato:

$$\mathcal{D}^{\mathrm{c}}(C) = (\overbrace{S_1,\ldots,S_{\theta_s}}^{\text{sensores}},\overbrace{A_1,\ldots,A_{\theta_a}}^{\text{atuadores}},\overbrace{E_1,\ldots,E_{\theta_e}}^{\text{estados}},\overbrace{\mathcal{D}^{\mathrm{f}}(f_1),\ldots,\mathcal{D}^{\mathrm{f}}(f_{\theta_f})}^{\text{funções}}) \tag{3.4}$$

Onde:

- $\theta_s \geq 0$, $\theta_e \geq 0$, $\theta_a \geq 0$ e $\theta_f \geq 0$, valendo a restrição $\theta_s+\theta_a+\theta_e+\theta_f > 0$ (não faz sentido definir uma classe com objetos vazios);
- S_i e A_i são conjuntos representando os tipos de objetos que os sensores e atuadores podem tratar, respectivamente. E_i são conjuntos que descrevem o tipo dos estados internos; e
- $\mathcal{D}^{\mathrm{f}}(f_i)$ representa o descritor da função (Definição 3.22) de transformação f_i.

DEFINIÇÃO 3.22 (DESCRITOR DE UMA FUNÇÃO)

Sejam: $\mathcal{D}^{\mathrm{c}}(C)$ um descritor de classe, $k_1^\star$, $k_2^\star$, $k_3^\star$ e $k_4^\star$ quaisquer fórmulas de indução $\star$ simples[14]. Define-se então:

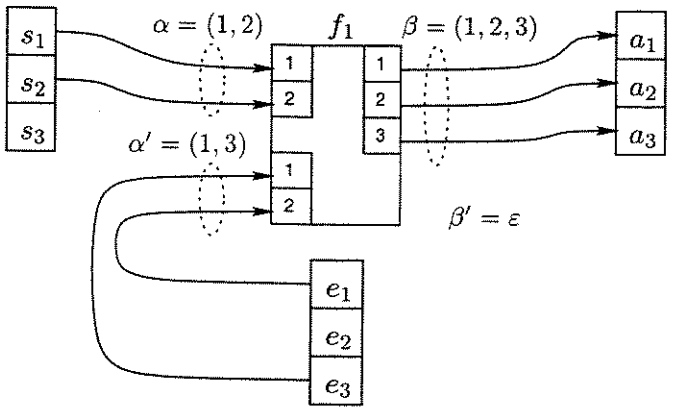
- as **ênuplas descritoras de domínio** como:
  $\alpha = (1,\ldots,\theta_s)_{(k_1^\star)}$, leitura dos sensores;
  $\alpha' = (1,\ldots,\theta_e)_{(k_2^\star)}$, leitura interna.
- as **ênuplas descritoras de contradomínio** como:
  $\beta = (1,\ldots,\theta_s)_{(k_3^\star)}$, escrita nos atuadors;
  $\beta' = (1,\ldots,\theta_e)_{(k_4^\star)}$, escrita interna.

Um **descritor de função** é dado por uma ênupla que descreve o mapeamento de domínio e contradomínio no seguinte formato:

$$\mathcal{D}^{\mathrm{f}}(f_j) = (\alpha,\alpha',\beta,\beta') \tag{3.5}$$

Adicionalmente, todas as ênuplas descritoras não-vazias (diferentes de ε) devem satisfazer as seguintes restrições:

- $\alpha = (\alpha_1,\ldots,\alpha_i,\ldots,\alpha_{\phi_1})$, $1 \leq \phi_1 \leq \theta_s$, tem-se:
  $1 \leq \alpha_i \leq \theta_s$, $i = 1,\ldots,\phi_1$;
- $\alpha' = (\alpha'_1,\ldots,\alpha'_i,\ldots,\alpha'_{\phi_2})$, $1 \leq \phi_2 \leq \theta_e$, tem-se:
  $1 \leq \alpha'_i \leq \theta_e$ para $i = 1,\ldots,\phi_2$;
- $\beta = (\beta_1,\ldots,\beta_i,\ldots,\beta_{\phi_3})$, $1 \leq \phi_3 \leq \theta_a$, tem-se:
  $1 \leq \beta_i \leq \theta_a$ para $i = 1,\ldots,\phi_3$; e
- $\beta' = (\beta'_1,\ldots,\beta'_i,\ldots,\beta'_{\phi_4})$, $1 \leq \phi_4 \leq \theta_e$, tem-se:
  $1 \leq \beta'_i \leq \theta_e$ para $i = 1,\ldots,\phi_4$.



$$\mathcal{D}^{\mathrm{c}}(C) = (S_1, S_2, S_3, A_1, A_2, A_3, E_1, E_2, E_3, \mathcal{D}^{\mathrm{f}}(f_1))$$
$$\mathcal{D}^{\mathrm{f}}(f_1) = ((1,2),(1,3),(1,2,3),\varepsilon)$$

Figura 3.6: Representação gráfica da noção de descritor de função.

DEFINIÇÃO 3.23 (FUNÇÃO DE TRANSFORMAÇÃO)

Seja um descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$. Uma **função de transformação** f_j é definida por um mapeamento na forma:

$$f_j : P_j \rightarrow Q_j \tag{3.6}$$

Para a definição do domínio P_j e do contradomínio Q_j considere:

- α, α', β e β' ênuplas descritoras de domínio e contradomínio, sujeitas às restrições[15]: $\mathrm{Ar}(\alpha) + \mathrm{Ar}(\alpha') > 0$ e $\mathrm{Ar}(\beta) + \mathrm{Ar}(\beta') > 0$;
- $S_1, \ldots, S_{\theta_s}$ os conjuntos que descrevem os tipos dos sensores;
- $A_1, \ldots, A_{\theta_a}$ os conjuntos que descrevem os tipos dos atuadores;
- $E_1^\star, \ldots, E_{\theta_e}^\star$ os conjuntos que descrevem os tipos dos estados internos, admitindo valores nulos (conforme a Definição 3.16);
- $P'_j = \bigtimes_{i=1}^{\mathrm{Ar}(\alpha)} S_{\alpha_i}$ e $P'' = \bigtimes_{i=1}^{\mathrm{Ar}(\alpha')} E^\star_{\alpha'_i}$ como os domínios para as entradas sensoriais e internas, respectivamente; e
- $Q'_j = \bigtimes_{i=1}^{\mathrm{Ar}(\beta)} A_{\beta_i}$ e $Q'' = \bigtimes_{i=1}^{\mathrm{Ar}(\beta')} E^\star_{\beta'_i}$ como os contradomínios para os atuadores e estados internos, respectivamente.

[14] Conforme a Definição 3.19

[15] Esta restrição evita a definição de uma função sem domínio ou contradomínio.

Assim, P_j e Q_j podem ser escritos como[16]:

$$P_j = \begin{cases} P'_j & \alpha \neq \varepsilon \text{ e } \alpha' = \varepsilon \\ P''_j & \alpha = \varepsilon \text{ e } \alpha' \neq \varepsilon \\ P'_j \times P''_j & \alpha \neq \varepsilon \text{ e } \alpha' \neq \varepsilon \end{cases} \quad Q_j = \begin{cases} Q'_j & \beta \neq \varepsilon \text{ e } \beta' = \varepsilon \\ Q''_j & \beta = \varepsilon \text{ e } \beta' \neq \varepsilon \\ Q'_j \times Q''_j & \beta \neq \varepsilon \text{ e } \beta' \neq \varepsilon \end{cases} \tag{3.7}$$

A partir da definição anterior, resulta que, para cada $h \in P_j$, $h' \in P'_j$ e $h'' \in P''_j$, valem as seguintes relações[17] (figura 3.7):

$$\begin{aligned} h' &= h \downarrow P'_j \\ h'' &= h \downarrow P''_j \end{aligned} \tag{3.8}$$

Analogamente, para quaisquer $s \in Q_j$, $s' \in Q'_j$ e $s'' \in Q''_j$:

$$\begin{aligned} s' &= s \downarrow Q'_j \\ s'' &= s \downarrow Q''_j \end{aligned} \tag{3.9}$$

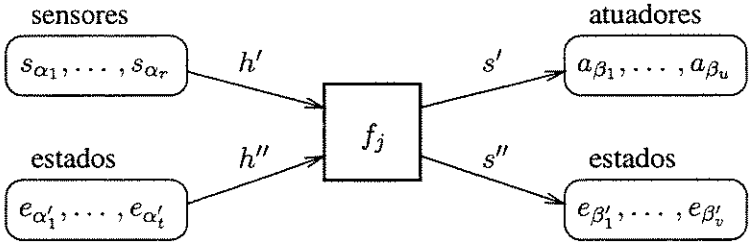


Figura 3.7: Mapeamentos de uma função de transformação, onde $r = \mathsf{Ar}(\alpha)$, $t = \mathsf{Ar}(\alpha')$, $u = \mathsf{Ar}(\beta)$ e $v = \mathsf{Ar}(\beta')$.

DEFINIÇÃO 3.24 (CONCORDÂNCIA COM UM DESCRITOR DE FUNÇÃO)

Uma função f_j é dita concordar com um descritor de função $\mathcal{D}^{\mathsf{f}}(f)$ se seu mapeamento domínio versus contradomínio, dado por P_j e Q_j (conforme construídos na Definição 3.23) apresentarem o mesmo formato de P_j e Q_j definidos segundo $\mathcal{D}^{\mathsf{f}}(f)$.

DEFINIÇÃO 3.25 (CONCORDÂNCIA COM UM DESCRITOR DE CLASSE)

Seja um descritor de classe $\mathcal{D}^{\mathsf{c}}(C)$ e uma ênupla:

$$c = (s_1, \ldots, s_{\theta_s}, a_1, \ldots, a_{\theta_a}, e_1, \ldots, e_{\theta_e}, f_1, \ldots, f_{\theta_f})$$

A ênupla c é dita concordar com o descritor de classe $\mathcal{D}^{\mathsf{c}}(C)$ se todas as condições abaixo forem satisfeitas:

[16] Importante ressaltar que devido às restrições impostas sobre as ênuplas descritoras de domínio e contradomínio os casos $\alpha = \alpha' = \varepsilon$ e $\beta = \beta' = \varepsilon$ nunca podem acontecer.

[17] A operação de projeção "$\downarrow$" é empregada conforme a Definição 3.8.

- $s_i \in S_i$, para $1 \leq i \leq \theta_s$;
- $a_i \in A_i$, para $1 \leq i \leq \theta_a$;
- $e_i \in E_i$, para $1 \leq i \leq \theta_e$; e
- f_i está em concordância com o descritor de função $\mathcal{D}^{\mathrm{f}}(f_i)$, para $1 \leq i \leq \theta_f$.

DEFINIÇÃO 3.26 (CLASSE)

Uma **classe**[18] C é o conjunto de todos os elementos c_i que concordam com o descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$.

DEFINIÇÃO 3.27 (CLASSE PASSIVA E CLASSE ATIVA)

Seja uma classe C obtida por um descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$. C é dita uma **classe ativa** se $\theta_f > 0$, caso contrário ($\theta_f = 0$) uma **classe passiva**[19].

DEFINIÇÃO 3.28 (AGENTE FORMAL)

Seja uma classe C não-vazia. Seja c uma variável cujas instâncias temporais concordam com o descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$. A variável c é então chamada de **agente** da classe C.

Um agente da classe C é definido como uma variável de tipo C em N (a Figura 3.8 mostra alguns exemplos de agentes):

$$\begin{aligned} &c : \mathbf{N} \rightarrow C \\ &c(n) = (s_1 \ldots s_{\theta_s}, a_1 \ldots a_{\theta_a}, e_1 \ldots e_{\theta_e}, f_1 \ldots f_{\theta_f}) \end{aligned} \tag{3.10}$$

Onde $s_i \in S_i$, $a_i \in A_i$, $e_i \in E_i$ e f_i dada pelo descritor $\mathcal{D}^{\mathrm{f}}(f_i)$.

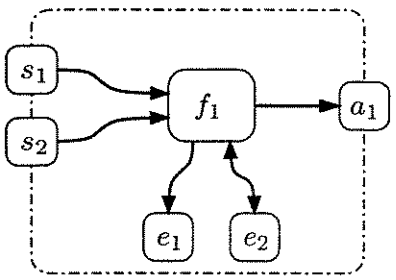
Conforme descrito anteriormente, a garantia da unicidade de um agente não está associada ao conteúdo de seus campos, e sim ao seu nome, que é único no sistema de agentes. Dois objetos da mesma classe com os mesmos valores e nomes diferentes são de fato duas entidades diferentes.

Um descritor de classe é uma forma de se agregar as informações de definição de uma classe a partir de uma meta-representação. A partir dos parâmetros do descritor de classe pode-se identificar os seguintes casos (não necessariamente excludentes):

- $\theta_f = 0$: a classe não possui nenhuma função de transformação, caracterizando um *objeto passivo*;
- $\theta_f > 0$: um *objeto ativo*, ou, alternativamente, um agente formal puro. Neste caso existem outros tipos interessantes:
  - $\theta_s \geq 0, \theta_a = 0$: *agente vertedouro* (ou puramente perceptivos), podendo apenas consumir objetos.
  - $\theta_s = 0, \theta_a \geq 0$: *agente fonte* (ou puramente atuador), podendo apenas gerar novos objetos.

[18] Tal como definido em (Guerrero, 2000)

[19] Conforme apresentado em (Gudwin, 1996).

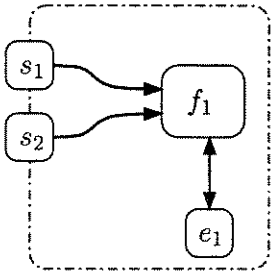


$\mathcal{D}^{\mathrm{c}}(C) = (S_1, S_2, A_1, E_1, E_2, \mathcal{D}(f_1))$

$\mathcal{D}^{\mathrm{f}}(f_1) = ((1,2),(2),(1),(1,2))$

$f_1 : S_1 \times S_2 \times E_2^\star \longrightarrow A_1 \times E_1^\star \times E_2^\star$

(a)



$\mathcal{D}^{\mathrm{c}}(C) = (S_1, S_2, E_1, \mathcal{D}(f_1))$

$\mathcal{D}^{\mathrm{f}}(f_1) = ((1,2),(1),\varepsilon,(1))$

$f_1 : S_1 \times S_2 \times E_2^\star \longrightarrow E_1^\star$

(b)

Figura 3.8: Representação de agentes formais, identificando sensores, atuadores, campos internos e funções de transformação.

- $\theta_e = 0$: o objeto não possui estado interno, comportando-se de forma puramente reativa.

Além do que foi exposto, todas as interfaces definidas em uma classe, sejam elas de entrada ou saída, devem pertencer pelo menos ou a um domínio ou a um contradomínio de pelo menos uma de suas funções de transformação.

DEFINIÇÃO 3.29 (SUBCLASSE)

Sejam:

- Uma classe C cujas instâncias temporais c possuem aridade p;
- Uma classe $\hat{C}$ cujas instâncias temporais $\hat{c}$ possuem aridade r, de forma que $r \geq p$;
- Uma fórmula de extensão (conforme a Definição 3.10) $k = [k_1, k_2, \ldots, k_i, \ldots, k_r]$ onde cada k_i é:
  - ou um número inteiro $k_i \in \{1, 2, \ldots, p\}$ sem repetição, $i \neq j \Rightarrow k_i \neq k_j$ para i e j assumindo qualquer valor onde o índice represente um número, ou
  - um conjunto V_i.

$\hat{C}$ é uma **subclasse** de C se existe k tal que[20] $C^{(k)} = \hat{C}$.

DEFINIÇÃO 3.30 (SUPERCLASSE)

Sejam:

- Uma classe C cujas instâncias temporais c possuem aridade p;
- Uma classe $\hat{C}$ cujas instâncias temporais $\hat{c}$ possuem aridade r, de forma que $r \geq p$;

[20] Aplica-se a extensão de uma ênupla de acordo com a Definição 3.11.

- Uma fórmula de indução $k = [k_1, k_2, \ldots, k_i, \ldots, k_p]$ onde cada k_i é um número inteiro $k_i \in \{1, 2, \ldots, r\}$ sem repetição, ou seja, $i \neq j \Rightarrow k_i \neq k_j$ para $i = 1, \ldots, p$ e $j = 1, \ldots, p$.

C é uma **superclasse** de $\hat{C}$ se existe k tal que[21] $\hat{C}_{(k)} = C$.

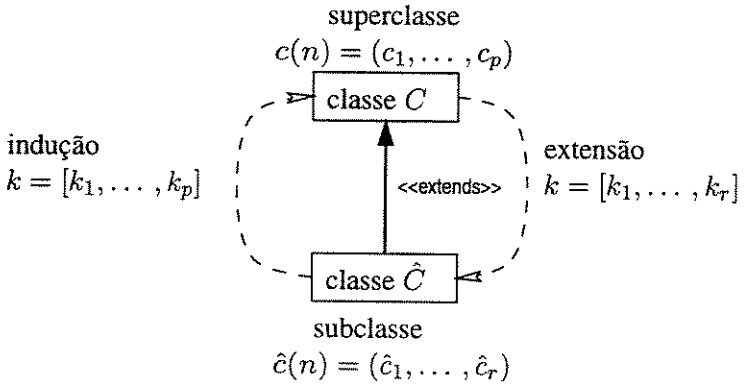


Figura 3.9: Hierarquia de classes: superclasse e subclasse.

As definições anteriores tratam apenas de caracterizar o mecanismo de herança (conforme ilustra a Figura 3.9), sem no entanto descrever um método para implementação.

3.3.4 Aspectos Temporais

As definições nesta subseção tem dependência direta com a noção de tempo. As duas definições a seguir são dadas no mesmo formado apresentado por Gudwin (Gudwin, 1996).

DEFINIÇÃO 3.31 (EXISTÊNCIA DE UM OBJETO)

Um objeto c é dito existir em um instante n se a função que mapeia as *instâncias temporais* de c em C é definida para $n \in \mathbb{N}$.

DEFINIÇÃO 3.32 (GERAÇÃO E CONSUMO DE OBJETOS)

Um agente é dito **gerado** em um instante n se ele não existe em n e existe em $n + 1$. Um objeto é **consumido** em n se ele existe em n e não existe em $n + 1$.

DEFINIÇÃO 3.33 (ESCOPO DE VISIBILIDADE DE UM SENSOR)

Sejam:

- uma classe C descrita por um descritor de classe $\mathcal{D}^c(C)$, tal que $\theta_s > 0$,
- $\mathcal{C}$ o conjunto de todos os objetos,

[21] Aplica-se a indução de uma ênupla de acordo com a Definição 3.5

- um agente c_k da classe C, e
- um sensor s_i definido em c_k, com capacidade perceptiva para elementos da classe S_i.

O **escopo de visibilidade** de s_i no instante n é dado pelo conjunto de todos objetos da classe S_i cujas instâncias temporais sejam *acessíveis* a partir do sensor s_i no instante n, excluindo-se ele próprio. Pode ser representado pela função:

$$\psi_k(i) : \mathbf{N} \rightarrow 2^C \tag{3.11}$$

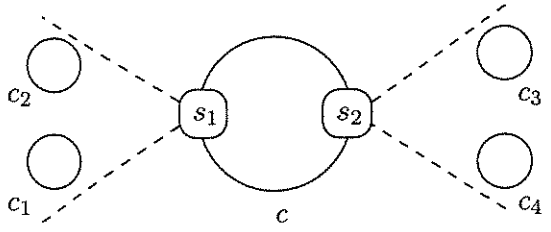


Figura 3.10: Escopo de visibilidade dos sensores s_1 e s_2 do agente c em um instante n: $\psi(1,n) = \{c_1, c_2\}$ e $\psi(2,n) = \{c_3, c_4\}$.

A noção de *visibilidade* é dependente das restrições de escopo do modelo. Em um sistema de agentes genérico o escopo de um sensor pode ser bastante abrangente, visto que não há restrições quanto a quais agentes são visíveis a partir de um observador. Geralmente, a consideração de todas as possibilidades não é desejável, pois a quantidade de combinações de interações possíveis cresce de forma bastante acentuada com o número de objetos, podendo tornar o problema intratável. Nas redes de agentes, a inclusão do conceito de lugares e arcos introduz restrições no escopo de visibilidade dos agentes, permitindo um melhor gerenciamento da complexidade das interações. Note que é possível a existência de dois ou mais sensores com o mesmo escopo de visibilidade. A definição de *escopo habilitante* em agente é derivada da combinação de todos os escopos de visibilidade dos sensores de cada função interna do objeto, conforme descrito adiante.

DEFINIÇÃO 3.34 (ESCOPO DE VISIBILIDADE DE UMA FUNÇÃO)

Sejam:

- um agente c_k da classe C (descrita por $\mathcal{D}^{\mathrm{c}}(C)$) e
- uma função f_j (descrita por $\mathcal{D}^{\mathrm{f}}(f_j)$) definida no agente c_k e α a ênupla descritora de domínio para as entradas sensoriais, onde $\alpha \neq \varepsilon$.

O **escopo de visibilidade de uma função** em um instante n é dado por um conjunto de ênuplas contendo todas as combinações possíveis de objetos acessíveis pelos sensores presentes no domínio de entrada da função f_j no instante n:

$$\Psi_k(j,n) = \bigtimes_{i=1}^{\mathrm{Ar}(\alpha)} \psi_k(\alpha_i, n) \tag{3.12}$$

Além disso, a partir da definição, para todos os elementos $v \in \Psi_k(j,n)$ tem-se que $\mathrm{Ar}(v) = \mathrm{Ar}(\alpha)$. Caso $\alpha = \varepsilon$ tem-se que $\forall n \in \mathbf{N}\ \ \Psi_k(j,n) = \emptyset$.

A partir desta definição pode-se observar que o escopo de visibilidade de uma função compreende apenas os objetos perceptíveis a partir dos sensores, sem incluir as informações presentes nos estados internos.

DEFINIÇÃO 3.35 (ESCOPO HABILITANTE DE UMA FUNÇÃO)

Sejam:

- um agente c_k da classe C, dada pelo descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$;
- uma função f_j definida em c_k, dada pelo descritor de função $\mathcal{D}^{\mathrm{f}}(f_j)$;
- α e α' as ênuplas descritoras de domínio para a função f_j;
- $\mathcal{B} = \{\mathbf{I^E}, \mathbf{I^S}, \mathbf{I^C}\}$ o conjunto de **modos de acesso** possíveis que um agente pode apresentar quando interagindo com os objetos através de seus sensores; e
- $\Psi_k(j,n)$ o escopo de visibilidade da função f_j.

Um **escopo habilitante** para esta função é uma ênupla $H_k(j,n) = (h,b)$ onde:

- $h \in \Psi(j,n)$, apresentando o formato $(h_1, \ldots, h_i, \ldots, h_r)$, sendo $r = \mathrm{Ar}(\alpha)$, satisfazendo a seguinte restrição:

$$\forall i \in \{1,\ldots,r\}\ \forall l \in \{1,\ldots,r\} \quad i \neq l \Rightarrow h_i \neq h_l$$

- $b \in \mathcal{B}^r$ apresenta o formato $(b_1, \ldots, b_i, \ldots, b_r)$, indicando o modo de acesso do agente c_k em relação ao objeto h_i:
  - $\mathbf{I^E}$ (assimilação exclusiva): h_i terá seu conteúdo assimilado[22] por c_k, com a restrição adicional de que nenhum outro agente poderá conter h_i em seu escopo habilitante no mesmo instante n;
  - $\mathbf{I^S}$ (assimilação compartilhada): semelhante ao anterior, mas permite-se que outros agentes também referenciem h_i em seu escopos habilitantes no instante n; e
  - $\mathbf{I^C}$: (consumo): h_i será assimilado por c_k e depois destruído.

O conjunto de todos os escopos habilitantes possíveis $\overline{H}_k(j,n)$ é dado por:

$$\overline{H}_k(j,n) = \Psi(j,n) \times \mathcal{B}^r \tag{3.13}$$

Um escopo habilitante descreve o padrão das interações de um agente em relação às suas percepções (objetos nos escopos de visibilidade de seus sensores). Além disso, uma mesma função pode ter, em um dado instante, diversos *escopos habilitantes possíveis*, dados por $H_k(j,n) \in \overline{H}_k(j,n)$,

[22] A individualidade de h_i enquanto objeto do sistema de agentes é preservada.

mas *apenas um deles* pode ser escolhido. Uma análise preliminar dos tipos interações diretas permite constatar a possibilidade de existência de escopos habilitantes que conflitem entre si[23]. Este aspecto será estudado de forma detalhada na seção sobre os aspectos competitivos da interação entre os agentes.

DEFINIÇÃO 3.36 (FUNÇÃO DE AVALIAÇÃO)

Sejam:

- c_k um agente da classe C, descrita pelo descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$;
- f_j uma função de transformação definida em c;
- α e α' as ênuplas descritoras de domínio para a função f_j;
- P'_j e P''_j conforme a equação 3.7, pág 33;
- $\mathcal{B}$ o conjunto de modos de acesso, conforme descrito na definição anterior;
- $r = \mathrm{Ar}(\alpha)$; e
- $\overline{H}_i(j,n)$ o conjunto de escopos habilitantes para o agente c_i no instante n.

A **função de avaliação**, relativa à função f_j, descreve o *grau de utilidade* de cada combinação de escopo habilitante (pertencente a $\overline{H}_i(j,n)$) e estados internos no instante n, sendo dada por um mapeamento $g_j : G_j \rightarrow \mathbf{R}$, onde G_j é na forma:

$$G_j = \begin{cases} P'_j \times \mathcal{B}^r & \alpha \neq \varepsilon \text{ e } \alpha' = \varepsilon \\ P''_j & \alpha = \varepsilon \text{ e } \alpha' \neq \varepsilon \\ (P'_j \times \mathcal{B}^r) \times P''_j & \alpha \neq \varepsilon \text{ e } \alpha' \neq \varepsilon \end{cases} \tag{3.14}$$

Como requisito fundamental, exige-se que todas as funções f_j, $1 \leq j \leq \theta_f$, apresentem uma respectiva função de avaliação. Adicionalmente, define-se um **limiar de utilidade** $G_0 \in \mathbf{R}$, tal que todos os valores de utilidade inferiores a G_0 não podem habilitar a função f_j.

DEFINIÇÃO 3.37 (ESCOPO GERATIVO DE UMA FUNÇÃO)

Sejam:

- um agente c_k da classe C, dada pelo descritor de classe $\mathcal{D}^{\mathrm{c}}(C)$;
- uma função f_j definida em c_k, dada pelo descritor de função $\mathcal{D}^{\mathrm{f}}(f_j)$;
- β e β' as ênuplas descritoras de contradomínio para a função f_j; e
- Q'_j e Q''_j conforme a equação 3.7, pág 33.

[23] O caso mais simples onde tais conflitos aparecem pode ser visto, por exemplo, quando dois escopos habilitantes especificam o consumo de um mesmo objeto.

Um **escopo gerativo** para esta função é uma ênupla $S_k(j,n) \in Q'_j$ no formato $(s_1, \ldots, s_i, \ldots, s_r)$, $r = \mathrm{Ar}(\beta)$, onde cada s_i representa um objeto criado por c_k no instante n, ou um objeto presente no escopo habilitante da função, que é transportado.

DEFINIÇÃO 3.38 (HABILITAÇÃO DE UMA FUNÇÃO)

Seja um agente c_k da classe C, uma função f_j definida em c_k e $d_k^j(n)$ uma subênupla contendo o valor dos campos internos pertencentes ao domínio de f_j no instante n. A função f_j é dita estar **habilitada** em n se:

- caso 1 – $\alpha \neq \varepsilon$ e $\alpha' \neq \varepsilon$:

$$\exists H_k \in \overline{H}_k(j,n) \quad g_j(H_k, d_k^j(n)) > G_0$$

- caso 2 – $\alpha \neq \varepsilon$ e $\alpha' = \varepsilon$:

$$\exists H_k \in \overline{H}_k(j,n) \quad g_j(H_k) > G_0$$

- caso 3 – $\alpha = \varepsilon$ e $\alpha' \neq \varepsilon$:

$$g_j(d_k^j(n)) > G_0$$

A habilitação é uma condição necessária mas não suficiente para o disparo da função.

DEFINIÇÃO 3.39 (HABILITAÇÃO DE UM AGENTE)

Um agente c está **habilitado** em um instante $n \in \mathbf{N}$ quando pelo menos uma de suas funções de transformação estiver habilitada em n.

Conforme comentado anteriormente, o agente precisa estar habilitado para que dispare, mas o fato de estar habilitado não implica necessariamente que o mesmo dispare.

DEFINIÇÃO 3.40 (DISPARO DE UM AGENTE)

Seja um agente c de uma classe C habilitado em n por escopo habilitante $\hat{H} = H(j,n)$. Seja f_j uma função de c habilitada em n. O disparo de f_j em n corresponde aos seguintes processos (não mutuamente excludentes):

- **regeneração** do conteúdo de todos os campos internos que não fizerem parte do domínio e/ou contradomínio da função f_j, ou seja, todos os e_i para os quais $\nexists k \in \{1, \ldots, \mathrm{Ar}(\alpha')\}$ tal que $\alpha'_k = i$ e, adicionalmente, $\nexists k \in \{1, \ldots, \mathrm{Ar}(\beta')\}$ tal que $\beta'_k = i$. Assim tem-se $e_i(n+1) = e_i(n)$;

- **consumo** de todos os objetos h_i presentes no escopo habilitante $\hat{H} = (h, b)$ para os quais $b_i = \mathbf{I^C}$;

- **geração** de todos os objetos s_i presentes no escopo gerativo $s \in S(f_j, n)$, onde $s = (s_1, \ldots, s_r)$ e $r = \mathrm{Ar}(\beta) + \mathrm{Ar}(\beta')$; e

- **atualização** de todos os estados internos que fizerem parte do contradomínio da função f_j. Assim, sejam $e_1(n), \ldots, e_{\theta_e}(n)$ os estados internos de c e $w = f_j(\hat{H}_1, \ldots, \hat{H}_r, e_{\alpha'_1}, \ldots, e_{\alpha'_p}) \downarrow Q''$, para $r = \mathsf{Ar}(\alpha)$, $p = \mathsf{Ar}(\alpha')$ e Q'' da mesma forma apresentada na Definição 3.23. Então $e_{\alpha_i}(n+1) = w_i$, $i = 1, \ldots, \mathsf{Ar}(\beta')$.

Todos os objetos que não dispararem e não fizerem parte de nenhum escopo habilitante $\hat{H}$ usado no disparo de outro agente possuem todos os seus campos internos regenerados.

3.4 Sistemas de Agentes

Um sistema de agentes (SA) é um conjunto de objetos (passivos ou ativos) que possuem suas instâncias temporais associadas entre si no tempo por um conjunto de propriedades. Um SA é um sistema fechado, no sentido em que qualquer objeto, a qualquer instante, deve pertencer ou ao estado inicial do sistema ou ter sido criado por um agente do sistema. De forma análoga, um objeto qualquer do sistema só pode ser consumido por um agente do sistema. As interações descritas anteriormentes são determinadas pela função de determinação da dinâmica, chamada *função de seleção.*

DEFINIÇÃO 3.41 (FUNÇÃO DE SELEÇÃO)

Sejam:

a) um conjunto de classes C_i, dadas pelos descritores $\mathcal{D}^{\downarrow}(C_i)$;

b) $\mathcal{C} = \{c_i\}$ um conjunto de objetos, onde cada objeto c_i é da classe C_i;

c) $\Theta_i = \{0, \ldots, m_i\}$, onde m_i ($m_i \geq 0$) é o número de funções de transformação definidas no objeto c_i;

d) $\hat{h}_i(n) = (h_i, b_i)$ o escopo habilitante de c_i no instante n; e

e) $\hat{s}_i(n)$ o escopo gerativo de c_i no instante n.

Uma **função de seleção** é um mapeamento γ_i definido para todos os objetos de $\mathcal{C}$ que descreve os participantes dos escopos habilitante e gerativo do objeto c_i. Cada γ_i tem a forma:

$$\gamma_i(n) = \left(\hat{h}(n), \hat{s}(n), \theta_i\right) \tag{3.15}$$

Esta função associa, a cada instante de tempo, para cada objeto c_i, um escopo habilitante $\hat{h}_i = (h, b)$, um escopo gerativo $\hat{s}_i$ e um índice $\theta_i \in \Theta_i$ que define a função de transformação a ser executada. Caso o índice da função de transformação seja 0 convenciona-se que nenhuma função interna será executada. Adicionalmente, se c_i for um objeto passivo $\forall n \in \mathbf{N}$ $\gamma_i(n) = (\varepsilon, \varepsilon, 0)$. Neste caso denomina-se uma *função de seleção vazia.*

Todas funções de seleção estão sujeitas a um conjunto de *regras de consistência*, de forma que os casos descritos abaixo são ditos inconsistentes:

i. Um agente está presente em seu próprio escopo habilitante.

ii. Dois ou mais agentes possuem escopos habilitantes especificando interações com um mesmo objeto e pelo menos um deles apresenta modo de acesso diferente de $\mathbf{I}^{\mathbf{S}}$.

iii. Um agente sendo consumido ou assimilado em um instante n não pode disparar em n.

Uma função de seleção é uma descrição genérica do processo de funcionamento de um agente. Sua determinação pode ser realizada por qualquer método que cumpra as regras de consistência. A Seção 3.6 apresenta uma formalização para a determinação dinâmica, incluindo alguns algoritmos para elaboração semi-automática.

DEFINIÇÃO 3.42 (SISTEMA DE AGENTES)

Um sistema de agentes Ω é um conjunto de pares $\{\omega_i\}$, $\omega_i = (c_i, \gamma_i)$ tal que:

I. para $n = 0$ existe pelo menos um ω_i com um objeto c_i definido.

II. para $n > 0$ todos os agentes que disparam o fazem de acordo com a função seleção $\gamma_i(n)$, interagindo de acordo com os objetos de seu escopo habilitante e gerando os objetos indicados no escopo gerativo a partir do disparo de uma de suas funções.

III. todos os objetos consumidos em n fazem parte do escopo habilitante de um agente do sistema no instante n.

IV. todos os objetos gerados em n fazem parte do escopo gerativo de um agente do sistema no instante n.

A definição de um sistema de agentes é uma especificação de propriedades gerais, garantindo que o sistema tenha um estado inicial não-vazio (pelo menos um objeto) e que todos os agentes gerados e consumidos para qualquer valor de $n > 0$ sejam unicamente em decorrência do disparo de agentes do próprio sistema. As características citadas permitem delimitar a diferença entre um simples conjunto de objetos e um sistema de agentes.

3.5 Redes de Agentes

Uma rede de agentes (RA) é um tipo especial de sistema de agentes com restrições topológicas dadas pela adição de dois tipos de entidades passivas: lugares e arcos. Esta nomenclatura possui inspiração nas Redes de Petri (Murata, 1989), onde fichas são armazenadas em lugares e processados (consumidos e gerados) por transições conectadas por arcos de entrada e saída.

3.5.1 Lugares, Arcos e Portas

A restrições topológicas dizem respeito a dois pontos principais, diretamente relacionados aos escopos habilitante e gerativo:

- *escopo de visibilidade dos sensores*: agentes só apresentam interação direta (para assimilação e/ou consumo) com objetos localizados em lugares conectados por arcos de entrada; e

- *liberação de objetos gerados*: objetos gerados por um agente são liberados em lugares conectados por um arco de saída.

DEFINIÇÃO 3.43 (LUGAR)

Seja uma classe C. Um **lugar** π é definido como uma entidade passiva que agrupa zero ou mais objetos (ativos ou passivos) da mesma classe C. Adicionalmente, se C for uma classe ativa[24], diz-se que π é um **lugar ativo**, caso contrário, um **lugar passivo** (figura 3.11).

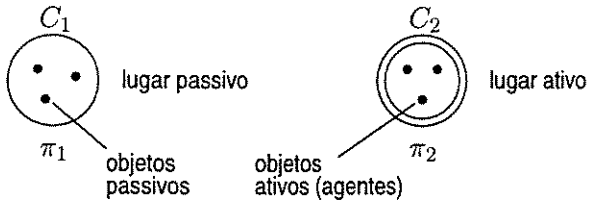


Figura 3.11: Lugares passivos e ativos.

O modelo de arco descrito neste trabalho, introduzido inicialmente em (Guerrero et al., 1999) e posteriormente em (Guerrero, 2000), possui grande similaridade com aquele descrito nas redes de objetos (Gudwin, 1996). No entanto, o modelo atual apresenta a noção de *portas*, permitindo uma formalização mais completa para a descrição topológica de uma rede de agentes.

Dado que uma porta permite a entrada e saída de objetos de um lugar, não faz sentido definir uma porta que não esteja associada a nenhum lugar. Um arco é definido, a grosso modo, como uma conexão entre uma porta de saída (ligada ao lugar de origem) e uma porta de entrada (ligada ao lugar de destino), conforme ilustrado na figura 3.12.

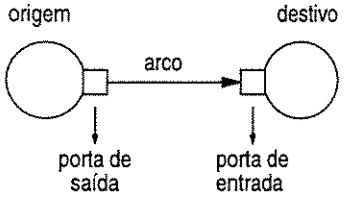


Figura 3.12: Portas de entrada e saída.

Portas são classificadas em duas características ortogonais (figura 3.13). A primeira delas – e a mais simples – se refere à direção do fluxo de informações, seja entrando ou saindo do lugar (figura 3.12). A segunda indica como os objetos são efetivamente inseridos ou removidos do lugar. A notação gráfica para a representação de arcos adotada neste trabalho – e originalmente descrita em (Guerrero et al., 1999) – apresenta pequenas modificações visuais dependendo da classificação das portas que formam o arco, sendo indicadas adiante.

Quanto ao comportamento interno de uma porta existem duas possibilidades distintas:

[24]Conforme Definição 3.27.

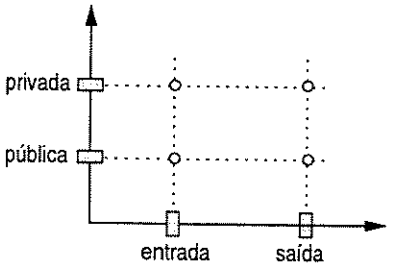


Figura 3.13: Características ortogonais de uma porta.

- **porta privada**: é conectada diretamente aos sensores ou atuadores de agentes presentes no lugar. Objetos que entram (por uma porta de entrada privada) são *percepções*[25] dos agentes localizados no lugar. Objetos que saem (por uma porta de saída privada) são *ações*[26] liberadas pelos atuadores dos agentes do lugar. Todas as portas privadas de um lugar só são consistentes quando associadas ou a um sensor ou a um atuador, de forma que sua quantidade é, dessa forma, limitada pela definição da classe associada ao lugar. Conclui-se, naturalmente, que este tipo de porta só existe em lugares ativos nos quais as classes tenham pelo menos um sensor e/ou atuador.

- **porta pública**: objetos que entram por uma porta pública são *inseridos* no lugar, ficando lá até serem removidos. Objetos que saem por uma porta pública são *removidos* do lugar. Qualquer lugar, seja ativo ou passivo, pode conter quantas portas públicas se desejar.

A partir da conceituação superficial de um arco e da caracterização dos tipos de portas deriva-se uma classificação mais detalhada sobre os tipos de arcos possíveis. Apenas duas combinações são consideradas válidas:

- saída pública para entrada privada (figura 3.14(a)) e
- saída privada para entrada pública (figura 3.14(b)).

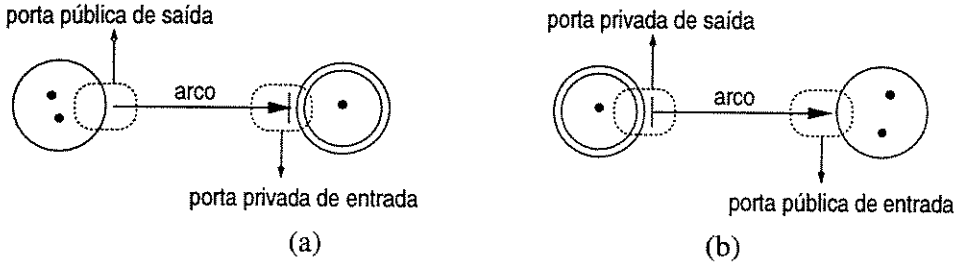


Figura 3.14: Tipos de arcos estudados neste trabalho.

As outras combinações de portas são:

[25] Quaisquer objetos sendo consumidos ou assimilados.
[26] Quaisquer objetos gerados

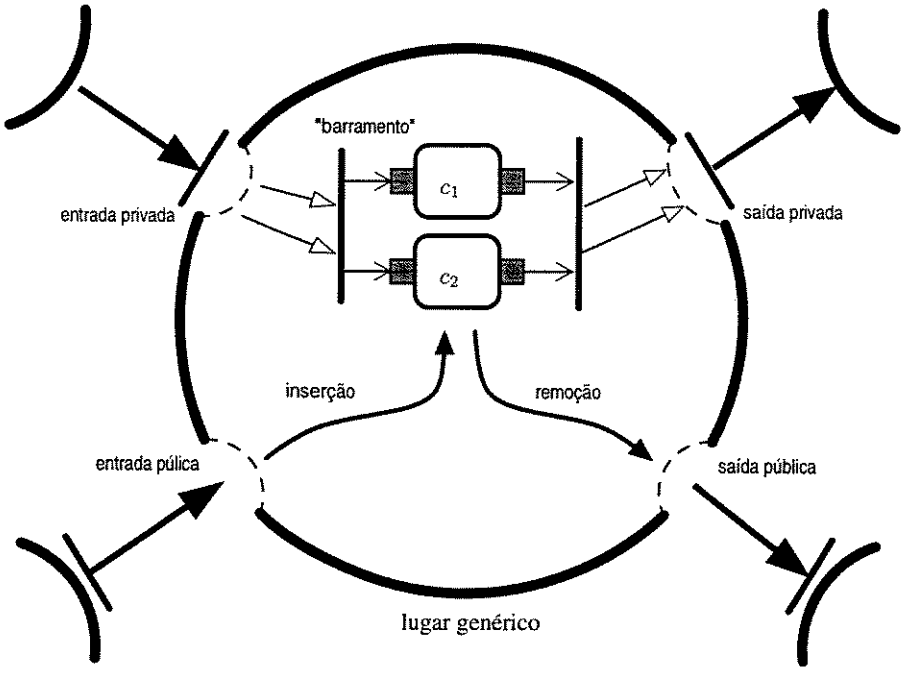


Figura 3.15: Estrutura interna de um lugar genérico, ilustrando os quatro tipos de portas.

- um arco *privado-privado* representa uma condição de corrida, onde o atuador de um agente está diretamente ligado ao sensor de outro agente, exigindo um mecanismo explícito de sincronização.
- um arco *público-público* representa apenas uma conexão inerte entre dois lugares.

A diferença entre arcos público-privado e privado-público pode ser melhor compreendida a partir da figura 3.15. Ela ilustra todas as quatro combinações possíveis de portas.

DEFINIÇÃO 3.44 (ARCO PÚBLICO-PRIVADO)

Sejam:

- π_o um lugar de origem que possui como classe associada C_o;
- π_d um lugar de destino ativo (não necessariamente distinto de π_o) que tenha uma classe associada C_d, descrita por $\mathcal{D}^c(C_d)$. Esta classe satisfaz as seguintes restrições:
  - $\mathcal{D}^c(C_d)$ possui pelo menos um sensor, $\theta_s > 0$, para objetos do tipo S_i, $1 \leq i \leq \theta_s$; e
  - o tipo do sensor é consistente com o tipo do lugar de origem, $S_i = C_o$.

Um **arco público-privado** $a_{\pi_o,\pi_d,i}$ é uma entidade que permite que agentes presentes em π_d possam interagir com quaisquer objetos presentes em π_o usando para isso seu i-ésimo sensor.

DEFINIÇÃO 3.45 (ARCO PRIVADO-PÚBLICO)

Sejam:

- π_d um lugar de destino qualquer com uma classe associada C_d;
- π_o um lugar de origem ativo (não necessariamente distinto de π_d) que tenha uma classe associada C_o, descrita por $\mathcal{D}^c(C_o)$. Esta classe satisfaz as seguintes restrições:
  - $\mathcal{D}^c(C_o)$ possui pelo menos um atuador, $\theta_a > 0$, para objetos do tipo A_i, $1 \leq i \leq \theta_a$; e
  - o tipo do atuador é consistente com o tipo do lugar de destino, $A_i = C_d$.

Um **arco privado-público** a_{π_o,i,π_d} é uma entidade topológica que permite que agentes presentes em π_o possam gerar objetos em π_d usando para isso o seu i-ésimo atuador.

DEFINIÇÃO 3.46 (REDE DE AGENTES)

Sejam:

- $\Sigma = \{C_i\}$ um conjunto de classes. Cada C_i é descrita por um descritor de classe $\mathcal{D}^c(C_i)$;
- $\mathcal{C} = \{c_i\}$ um conjunto de objetos em que cada c_i é de uma classe $C_j \in \Sigma$;
- $\Pi = \{\pi_i\}$ um conjunto de lugares;
- $\Xi : \Pi \rightarrow \Sigma$ é uma função de mapeamento de classes que associa uma classe de Σ a cada um dos lugares de Π;
- $\xi : \mathbf{N} \times \mathcal{C} \rightarrow \Pi$ é uma função de localização, que associa para cada objeto c_i, em um instante n, um lugar π_j. A função ξ segue a seguinte restrição: seja um objeto $c_i \in \mathcal{C}$ de classe $C_j \in \Sigma$, então $\Xi(\xi(n, c_i)) = C_j$;
- $\mathcal{A} = \{a_i\}$ um conjunto de arcos. $\mathcal{A}$ pode ser descrito pela união de dois conjuntos disjuntos $\mathcal{A}'$ e $\mathcal{A}''$ tais que $\mathcal{A}' \cap \mathcal{A}'' = \emptyset$ e $\mathcal{A}' \cup \mathcal{A}'' = \mathcal{A}$. $\mathcal{A}'$ descreve os arcos do tipo público-privado e $\mathcal{A}''$ descreve os arcos do tipo privado-público.
- $\eta' : \mathcal{A}' \rightarrow \Pi \times \Pi \times \tau'$ é uma função de nó para arcos tipo público-privado, onde $\tau' = \{1, \ldots, \theta_s\}$. θ_s expressa o número de sensores da classe associada ao lugar de destino;
- $\eta'' : \mathcal{A}'' \rightarrow \Pi \times \Pi \times \tau''$ é uma função de nó para arcos tipo privado-público, onde $\tau'' = \{1, \ldots, \theta_a\}$. θ_a expressa o número de atuadores da classe associada ao lugar de origem;
- $F : \Pi \rightarrow 2^{\Pi}$, uma função de mapeamento de lugares fontes. É composta na forma:

$$F(\pi) = \bigcup_{t \in \tau'} F'(\pi, t) \cup F''(\pi)$$

onde $\tau' = \{1, \ldots, \theta_s\}$, θ_s representa o número de sensores na classe associada ao lugar de destino. F' e F'' são as *funções de mapeamento de lugares fontes* via arcos público-privado e privado-público, respectivamente, podendo ser determinadas na forma (conforme Figura 3.16):

$$\begin{aligned} F'(\pi, t) &= \pi_j \in \Pi \mid \exists a_k \in \mathcal{A}' \;\; \eta'(a_k) = (\pi_j, \pi, t) \\ F''(\pi) &= \{\pi_j \in \Pi \mid \exists a_k \in \mathcal{A}'' \;\; \eta''(a_k) \downarrow \Pi \times \Pi = (\pi_j, \pi)\} \end{aligned} \tag{3.16}$$

- $V : \Pi \rightarrow 2^{\Pi}$, uma função de mapeamento de lugares vertedouros. É composta na forma:

$$V(\pi) = V'(\pi) \cup \bigcup_{t \in \tau'} V''(\pi, t)$$

onde $\tau' = \{1, \ldots, \theta_a\}$, θ_a representa o número de atuadores na classe associada ao lugar de origem. V' e V'' são as *funções de mapeamento de lugares vertedouros* via arcos público-privado e privado-público, respectivamente, podendo ser determinadas na forma (conforme Figura 3.16):

$$\begin{aligned} V'(\pi) &= \{\pi_j \in \Pi \mid \exists a_k \in \mathcal{A}' \;\; \eta'(a_k) \downarrow \Pi \times \Pi = (\pi, \pi_j)\} \\ V''(\pi, t) &= \pi_j \in \Pi \mid \exists a_k \in \mathcal{A}'' \;\; \eta''(a_k) = (\pi, \pi_j, t) \end{aligned} \tag{3.17}$$

- $\Gamma = \{\gamma_i(n)\}$ o conjunto das funções de seleção. Cada $\gamma_i(n)$ é a função de seleção para o objeto c_i, apresentando as mesmas restrições da definição 3.41, na página 41.

Define-se uma **Rede de Agentes** $\mathcal{R}$ como uma ênupla na seguinte forma:

$$\mathcal{R} = (\Sigma, \mathcal{C}, \Pi, \Xi, \xi, \mathcal{A}, \eta', \eta'', \Gamma)$$

As seguintes restrições devem ser cumpridas:

- **(Restrição 1)** $\mathcal{R}$ deve ser caracterizada como um sistema de agentes, ou seja, $\mathcal{C} \times \Gamma$ deve estar de acordo com a Definição 3.42;
- **(Restrição 2) - restrição topológica nos escopos habilitantes**
  Para cada um dos agentes c_i com escopo habilitante não-vazio considere:
  - $H_i(\delta, n) = (h, b)$, $1 \leq \delta \leq \theta_f$, é o escopo habilitante do agente c_i em n. δ indica a função na qual H_i está definido[27];
  - $h = (h_1, \ldots, h_k, \ldots, h_r)$, onde $r = \mathrm{Ar}(\alpha)$[28], a parcela do escopo habilitante referente aos sensores.

  A seguinte proposição deve ser satisfeita:

$$\forall k \in \{1, \ldots, \theta_s\} \;\; F'(\xi(c_i, n), k) = \xi(h_k, n) \tag{3.18}$$

[27] Em outras palavras, δ especifica a função escolhida para o disparo de c_i.
[28] Referente ao domínio de f_δ

- **(Restrição 3) - restrição topológica nos escopos gerativos**
  Para cada um dos agentes c_i com escopo gerativo não-vazio considere:
  - $S_i(\delta, n)$, $1 \leq \delta \leq \theta_f$, é o escopo gerativo do agente c_i em n. δ indica a função na qual S_i está definido em n;
  - $s = (s_1, \ldots, s_k, \ldots, s_r)$, onde $r = \mathrm{Ar}(\beta)$[29], a parcela do escopo gerativo referente aos atuadores.

  A seguinte proposição deve ser satisfeita:

$$\forall k \in \{1, \ldots, \theta_a\} \;\; V''(\xi(c_i, n), k) = \xi(s_k, n+1) \tag{3.19}$$

Uma característica geralmente desejável em redes de redes é a *computabilidade*. Gudwin apresenta (Gudwin, 1996) uma discussão sobre esse assunto. Dado que o modelo apresentando é uma generalização do modelo original proposto valem as mesmas regras de análise, de forma que estas não são abordadas neste trabalho.

DEFINIÇÃO 3.47 (NÚCLEO DE UMA REDE DE AGENTES)

Define-se o núcleo de uma rede agentes como uma rede de agentes na forma:

$$\mathcal{R}_0 = (\Sigma, C^0, \Pi, \Xi, \xi^0, \mathcal{A}, \eta', \eta'', \Gamma)$$

Onde:

- Σ, Π e Ξ são conforme a definição de rede de agentes,
- $C^0 = \{c_i'\}$ é um conjunto de objetos definidos para $n = 0$,
- ξ^0 é uma função de localização definida apenas para $n = 0$ e
- Γ é um conjunto de funções de seleção computáveis, determinadas a partir de um algoritmo γ'.

Analogamente a uma rede de objetos (Gudwin, 1996), a partir do núcleo de uma rede de agentes, novas redes de agentes são calculadas. Cada etapa do algoritmo corresponde à geração de um novo elemento na sequência de redes de agentes. Gudwin apresentou uma proposta de algoritmo genérico para o cálculo de redes de objeto (Gudwin, 1996). Tal algoritmo pode ser aplicada igualmente às redes de agente na forma proposta.

3.6 Dinâmica de uma Rede de Agentes

A dinâmica de uma rede de agentes é determinada pela função de seleção[30], que indica quais escopos habilitantes (EHs), escopos gerativos (EGs) e funções de transformação serão usados. Estes

[29] Referente ao contradomínio de f_δ

[30] Definição 3.41, pág. 41.

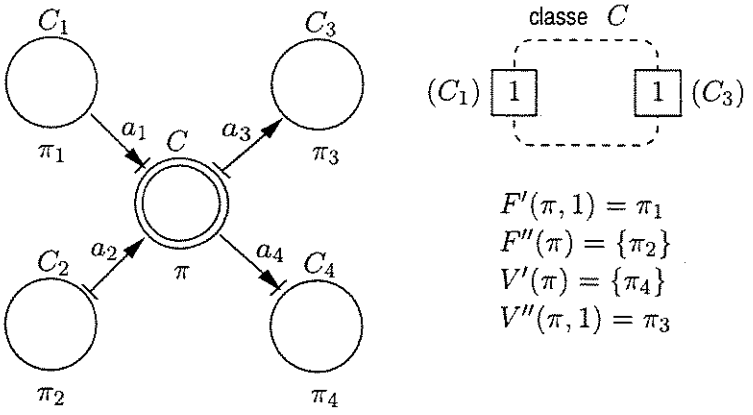


Figura 3.16: Exemplo de construção das funções F, F', V' e V''.

componentes apresentam uma relação cronológica entre si, visto que o escopo gerativo (objetos gerados) depende diretamente da função disparada e do escopo habilitante (objetos assimilados). A princípio, estes três componentes podem sempre ser determinados pelo projetista responsável pela criação do modelo, de acordo com o domínio do problema em questão. No entanto, guardadas as devidas proporções, a definição de mecanismos automáticos (ou semi-automáticos) capazes de simplicar o trabalho geralmente é de grande utilidade.

A partir da Definição 3.35, observa-se que, para uma mesma função de transformação de um dado agente, podem existir várias possibilidades diferentes de escolha para o EH. De forma geral, para uma função f_j de um agente c_k, no instante n, tem-se que $H_k(j,n) \in \overline{H}_k(j,n)$. O conteúdo de $\overline{H}_k(j,n)$ depende do escopo de visibilidade da função f_j. Apesar de ter uma determinação direta, o conjunto de EHs possíveis, $\overline{H}_k(j,n)$, pode apresentar conflitos com outros escopos habilitantes, sejam estes do mesmo agente ou não. O problema inicial pode ser colocado da seguinte forma: *como escolher os agentes, suas respectivas funções a serem disparadas e seus respectivos EHs para todos os valores de* n? A solução deste problema pode ser obtida de inúmeras formas diferentes, desde que se leve em consideração que:

i. existem 3 tipos básicos de interação entre um agente e um objeto, conforme descrito na Definição 3.35, pág. 38.

ii. todas as funções de seleção de todos os agentes, para qualquer instante de tempo, seguem as restrições de um sistema de agentes, apresentadas na Definição 3.41, pág.41.

3.6.1 Determinação das Ações dos Agentes

Conforme indicado anteriormente, podem existir vários mecanismos possíveis de determinação dinâmica. Esta subseção introduz um *framework* para a especificação de algoritmos de solução nos moldes de um problema de satisfação de restrições (CSP)[31] (Kumar, 1992; Hower, 1994; Russel and Norvig, 1995; Weiss, 1999). O Apêndice A contém um pequeno resumo sobre a formalização e

[31] Do inglês *constraint satisfaction problem*.

solução de CSPs, fornecendo ao leitor uma visão geral das noções básicas relacionadas a este tipo de problema.

Apresenta-se a seguir uma formalização para o processo BMSA (Guerrero et al., 1999; Guerrero, 2000) e mais adiante algumas implicações de relevância prática. Esta formalização é derivada em parte daquela utilizada em CSPs, mas leva em consideração algumas características intrínsecas do problema tratado[32].

Definição Formal para as Ações

Como hipótese inicial, todos os agentes em uma RA são capazes de determinar o grau de utilidade de cada uma de suas funções para cada uma das possíveis combinações de interações (ou escopos habilitantes)[33]. Cada uma dessas possíveis combinações denomina-se *ação*.

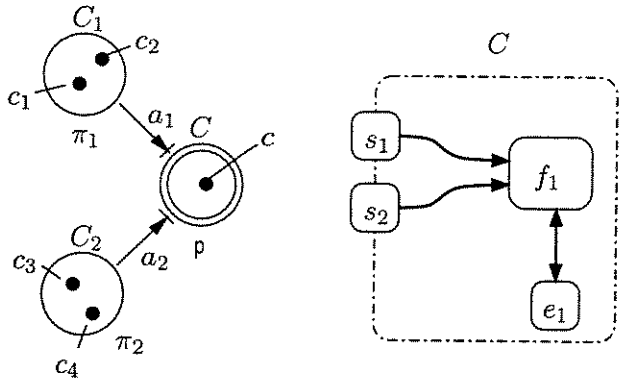


Figura 3.17: Exemplo de geração de ações em uma rede de agentes. Neste caso o agente c pode propor até quatro ações que especifiquem interações com outros objetos, visto que o conjunto das possibilidades é dado por $\{c_1, c_2\} \times \{c_3, c_4\}$.

DEFINIÇÃO 3.48 (AÇÃO DE UM AGENTE)

Sejam (no contexto de uma rede de agentes[34] $\mathcal{R}$):

- $\mathcal{C}$ o conjunto de objetos de $\mathcal{R}$;
- $\mathcal{B}$ o conjunto de modos de acesso, conforme especificado na Definição 3.35;
- $\Theta = \{1, \ldots, \theta\}$, onde θ é o número máximo de funções definidas nas classes de $\mathcal{R}$.

O **espaço de ações** dos agentes em $\mathcal{C}$ é representado por $\mathbb{A}$, definido como:

$$\mathbb{A} = \mathcal{C} \times 2^{\mathcal{C} \times \mathcal{B}} \times \mathbf{R} \times \Theta \tag{3.20}$$

[32] O número de variáveis presentes na solução pode variar, permitindo uma maior flexibilidade.
[33] A partir das definições 3.23 (vide Figura 3.17) e 3.36.
[34] Conforme a Definição 3.46.

Assim, uma **ação de um agente** é definida como uma ênupla em $\mathbb{A}$, na forma:

$$a = (c_r, I_r, \rho, \theta) \tag{3.21}$$

onde cada um dos componentes da ênupla a representa:

- $c_r \in \mathcal{C}$: agente (requerente) que planeja executar a ação;
- $I_r = \{(c_i, b_i)\} \subset 2^{\mathcal{C}\times\mathcal{B}}$: objetos com os quais r deseja interagir. Para cada um destes objetos especifica-se o *modo de acesso*, que indica o tipo de interação que c_r deseja sobre c_i. I_r pode ser vazio, implicando que o agente c_r, a partir da execução de a, não deseja interagir com nenhum outro objeto do ambiente.
- $\rho \in \mathbf{R}$: utilidade definida pelo requerente c_r.
- $\theta \in \Theta$: número da função de transformação do agente c_r a ser disparada.

Tal como indicado na Definição 3.3, os componentes internos da ênupla a podem ser indicados pelos respectivos índices de referência. No entando, por conveniência de representação, as seguintes funções podem ser usadas como uma alternativa:

- req : $\mathbb{A} \rightarrow \mathcal{C}$, agente requerente c_r;
- intrcs : $\mathbb{A} \rightarrow 2^{\mathcal{C}\times\mathcal{B}}$, interações I_r;
- util : $\mathbb{A} \rightarrow \mathbf{R}$, utilidade ρ; e
- func : $\mathbb{A} \rightarrow \Theta_r$, função a ser disparada θ.

Esta definição para o predicado consistent evita os seguintes casos:

a) uma ação especifica mais de um tipo de interação com um mesmo objeto, e

b) uma ação indica que o requerente deseja interagir consigo próprio (recursividade).

DEFINIÇÃO 3.49 (PREDICADO DE COEXISTÊNCIA)

Sejam: $A \subseteq \mathbb{A}$ o conjunto de ações possíveis (propostas), $\mathcal{B}$ o conjunto de tipos de interações disponíveis (modos de acesso) e $\mathcal{C}$ o conjunto de objetos existentes. O **predicado de coexistência** é definido como:

$$\begin{aligned}
&\forall a_i \in A\ \forall a_j \in A\ \forall b_i \in \mathcal{B}\ \forall b_j \in \mathcal{B} \quad \mathsf{coexist}(a_i, a_j) \iff \\
&\Big\{\big[\mathsf{req}(a_i) \neq \mathsf{req}(a_j)\big] \wedge \\
&\neg\big[\exists c_k \in \mathcal{C} \quad (b_i \neq \mathbf{I^S} \vee b_j \neq \mathbf{I^S}) \wedge \\
&\qquad (c_k, b_i) \in \mathsf{intrcs}(a_i) \wedge (c_k, b_j) \in \mathsf{intrcs}(a_j)\big] \wedge \\
&\big[(\mathsf{req}(a_i), b_i) \notin \mathsf{intrcs}(a_j) \wedge (\mathsf{req}(a_j), b_j) \notin \mathsf{intrcs}(a_i)\big]\Big\}
\end{aligned} \tag{3.22}$$

Esta definição evita os seguintes tipos de conflitos:

a) duas ações propondo interações com um mesmo objeto (tais ações só podem coexistir se ambas especificarem $\mathbf{I}^{\mathbf{S}}$ como seu modo de acesso),

b) uma das ações especifica uma interação com o requerente da outra e

c) duas ações definem o mesmo requerente.

Formalização do Problema

Seja $\mathcal{R}$ uma rede de agentes e $\mathbb{A}$ o universo de ações de agentes em $\mathcal{R}$. Define-se $A = \{a_1, a_2, \ldots, a_{\mathsf{NA}}\}$ como o subconjunto de $\mathbb{A}$ em um determinado instante n[35]. Admite-se que algumas ações de A podem conflitar com outras, de forma que não podem ser executadas no mesmo instante n. Esta restrição binária[36] é representada pelo predicado $\mathsf{coexist}(a_i, a_j)$, $a_i \in \mathbb{A}$ e $a_j \in \mathbb{A}$. Adicionalmente, considera-se que:

i. $\forall a_i \in \mathbb{A}, a_j \in \mathbb{A} \quad \mathsf{coexist}(a_i, a_j) \iff \mathsf{coexist}(a_j, a_i)$

ii. $\forall a \in \mathbb{A} \quad \mathsf{coexist}(a, a)$

O objetivo do algoritmo é então encontrar uma solução para o problema, ou seja, um subconjunto de ações definidas em A tal que todas as restrições existentes sejam satisfeitas.

DEFINIÇÃO 3.50 (CONJUNTO SOLUÇÃO)

Um conjunto $A' \subseteq A$, $A \subseteq \mathbb{A}$, é chamado de **solução de** A se não existirem quaisquer inconcistências dentro do conjunto A:

$$\forall A' \subseteq A \quad \mathsf{sol}(A', A) \iff \left[\forall a_i \in A' \; \forall a_j \in A' \quad \mathsf{coexist}(a_i, a_j)\right] \tag{3.23}$$

No campo de estudo da Inteligência Artificial Distribuída e sistemas multiagentes existem diversos critérios de avaliação de protocolos de negociação. Entre eles pode-se citar (Weiss, 1999):

- **bem-estar social**: soma dos saldos de todos os agentes em uma dada solução.
- **eficiência de Pareto**: critério de avaliação de soluções que considera uma perspectiva global. Uma solução x é Pareto-eficiente (ou Pareto-ótima) se não existe nenhuma solução x' na qual pelo menos um agente é melhor em x' do que em x e nenhum agente é pior em x' do que em x. O critério de bem-estar é um subconjunto do critério de otimalidade de Pareto.

Sob este contexto, introduz-se a noção de *solução-limite*, um tipo de solução mais restrito do que aquela apresentada na Definição 3.50.

DEFINIÇÃO 3.51 (SOLUÇÃO-LIMITE)

Sejam: $A \subseteq \mathbb{A}$ um conjunto de ações e $A' \subseteq A$ um subconjunto. A' é chamado de **solução-limite** de A se A' é uma solução e, adicionalmente, não resta nenhuma ação em A – que

[35] Em geral A depende do interesse próprio dos agentes em disparar, variando ao longo do tempo.

[36] Pois pode ser reduzida a uma restrição entre duas ações.

ainda não esteja em A' – que coexista com todas as ações de A'.

$$\begin{aligned} &\forall A' \subseteq A \quad \mathsf{limitSol}(A', A) \iff \\ &\left\{\mathsf{sol}(A', A) \wedge \forall a_i \in A' \left[\neg\exists a_j \in (A - A') \; \mathsf{coexist}(a_i, a_j)\right]\right\} \end{aligned} \tag{3.24}$$

Uma solução-limite corresponde a um subconjunto de A que atingiu o sua cardinalidade máxima, não restando nenhuma outra ação adicional que possa ser incluída sem que se viole alguma restrição. O estudo do comportamento de soluções-limite no que se refere à otimalidade de Pareto é deixada como trabalho futuro. Porém, pode-se estimar que as propriedades apresentadas por uma solução-limite são *mais fracas* do que aquelas apresentadas por uma Pareto ótima, pois dependendo da heurística, é possível a geração de soluções não necessariamente ótimas (no sentido de Pareto).

Note que, dependendo das ações de A e das restrições impostas por elas, é possível que se tenha mais de uma solução-limite. Um dos casos mais simples em que este comportamento é observado ocorre, por exemplo, quando $A = \{a_1, a_2\}$ e $\neg\mathsf{coexist}(a_1, a_2)$. Neste caso existem duas soluções-limite: $A' = \{a_1\}$ e $A' = \{a_2\}$. É possível generalizar a situação anterior se $\exists a_i, a_j \in A \; a_i \neq a_j \wedge \neg\mathsf{coexist}(a_i, a_j)$. Sempre é possível definir pelo menos duas soluções limite: uma que inclui a_i (sem a_j) e outra que inclui a_j (sem a_i). Adicionalmente, existem outras propriedades de interesse prático no que se refere às soluções-limite, sendo discutidas a seguir.

DEFINIÇÃO 3.52 (CONJUNTO DE AÇÕES INDEPENDENTES)

Sejam $A_1 \subseteq \mathbb{A}$ e $A_2 \subseteq \mathbb{A}$ conjuntos de ações. Dois conjuntos de ações A_1 e A_2 são considerados independentes se forem disjuntos, não-vazios e todas as ações de um sempre coexistirem com as ações do outro.

$$\begin{aligned} &\forall A_1 \subset A \; \forall A_2 \subset A \quad \mathsf{indep}(A_1, A_2) \iff \\ &\left\{A_1 \cap A_2 = \emptyset \wedge A_1 \neq \emptyset \wedge A_2 \neq \emptyset \; \wedge \right. \\ &\left.\left[\forall a_1 \in A_1 \; \forall a_2 \in A_2 \quad \mathsf{coexist}(a_1, a_2)\right]\right\} \end{aligned} \tag{3.25}$$

TEOREMA 3.1 (SOLUÇÃO INDEPENDENTE)

Sejam: $A_1 \subseteq \mathbb{A}$ e $A_2 \subseteq \mathbb{A}$. Se A'_1 e A'_2 são soluções-limite de dois conjuntos de ações independentes A_1 e A_2, respectivamente, a união de A'_1 e A'_2 é uma solução-limite da união de A_1 e A_2:

$$\begin{aligned} &\forall A_1 \subset \mathbb{A} \; \forall A_2 \subset \mathbb{A} \quad \mathsf{indep}(A_1, A_2) \Longrightarrow \\ &\left[\forall A'_1 \subset A_1 \; \forall A'_2 \subset A_2 \right. \\ &\quad \mathsf{limitSol}(A'_1, A_1) \wedge \mathsf{limitSol}(A'_2, A_2) \Rightarrow \\ &\left.\quad \mathsf{limitSol}(A'_1 \cup A'_2, A_1 \cup A_2)\right] \end{aligned} \tag{3.26}$$

[Prova] (Teorema 3.26)
Sejam $A = A_1 \cup A_2$ e $A' = A'_1 \cup A'_2$. Qualquer ação de A que ainda não estiver em A' e for adicionada a A' ou conflita com A'_1 (por que veio de A_1) ou conflita com A'_2 (por que veio de A_2). Portanto, A' é uma solução-limite de A. ■

O teorema anterior possui grande importância prática, pois permite que o processo de solução seja particionado em várias buscas independentes, mesmo em nós computacionais com acoplamento fraco.

3.6.2 Best Matching Search Algorithm (BMSA)

Este algoritmo foi proposto inicialmente (Guerrero et al., 1999) como um mecanismo capaz de determinar parte das funções de seleção de uma rede de objetos[37] pela propagação de restrições sobre o conjunto de ações, de acordo com o grau de utilidade de cada uma delas. O algoritmo foi derivado do modelo de controle com adaptação de contexto introduzido por Gudwin e Gomide (Gudwin and Gomide, 1998b). A versão apresentada funciona escolhendo uma ação (por intermédio da função selectAction, codificada no Algoritmo 3.1), a cada passo, e removendo qualquer outra ainda não selecionada que não possa coexistir com ela (propagação de restrições). Interessantemente, devido ao critério de escolha da próxima ação, baseado no grau de utilidade, não há a necessidade direta de *backtracking* – uma vez escolhida a ação ela não é mais descartada, já que a solução pode ser sempre alcançada empregando-se unicamente a propagação de restrições.

Algoritmo 3.1.

```
1  funct selectAction(A : 2^𝔸) : 𝔸 ≡
2    ⌈p_max := max_{a'∈A} util(a');
3     A_max := {a ∈ A | util(a) = p_max};
4     if A_max = {a}
5       then a_sel = a;
6       else a_sel = randomElement(A_max);
7     fi
8     return(a_sel);
9    ⌋
```

Algoritmo 3.2.

```
1  funct limitSolve(A : 2^𝔸) : 2^𝔸 ≡
2    ⌈A' := ∅;
3     A'' := A;
4     while A'' ≠ ∅ do
5           a' := selectAction(A'');
6           A' := A' ∪ {a'};
7           A'' := A'' − {∀a ∈ A''|a = a' ∨ ¬coexist(a, a')};
8     od
9     return(A');
10   ⌋
```

O algoritmo 3.2 é computável, pois:

a) A função *selectAction* é computável.

b) A única condição de permanência no laço para a função *limitSolve* é a da linha 4. Note que A'' tem, garantidamente, pelo menos um elemento removido a cada iteração (linha 7), pois a função *selectAction* sempre retorna uma ação. Dado que A'' é finito o laço em algum momento será interrompido. Adicionalmente, o conjunto A' retornado sempre é uma solução-limite de A.

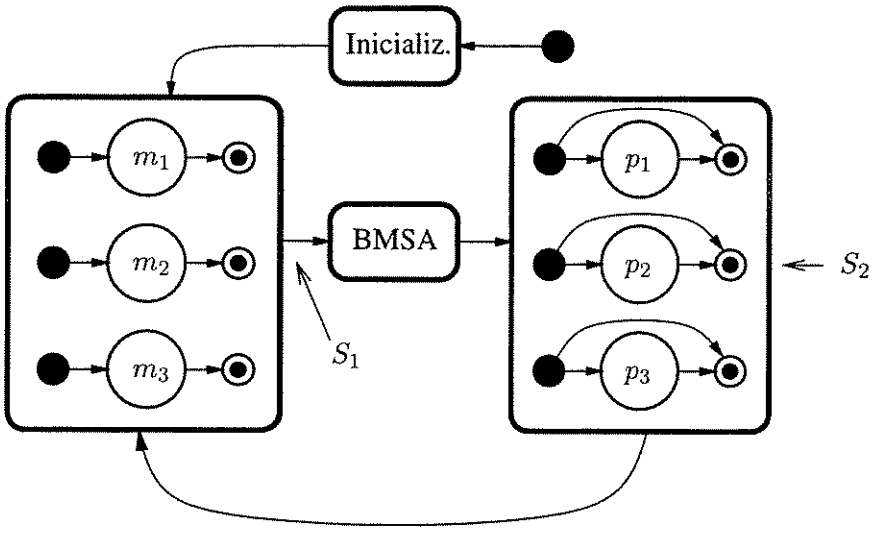


Figura 3.18: Estados de uma RA com três agentes empregando o BMSA. m_1, m_2 e m_3 indicam a fase de elaboração das propostas de ações. p_1, p_2 e p_3 indicam a fase de execução das ações efetivadas. S_1 e S_2 localizam os pontos de sincronização.

Uma das principais vantagens do BMSA é a sua simplicidade e velocidade, pois conforme descrito, não há a necessidade de *backtrack* (tal como explicado na página 101). Por outro lado, apresenta o inconveniente de exigir a disponibilidade simultânea de todas as ações em um mesmo ponto para a busca da solução, inviabilizando uma implementação paralela direta. A figura 3.18 mostra um caso típico de uma RA com 3 agentes em diagrama de estados (Booch et al., 1997). Existem dois pontos de sincronização[38], implicando em uma queda drástica na performance e tempo de resposta, visto que o tempo de execução de uma iteração é dado pelo tempo de execução do agente mais lento.

3.6.3 Propriedades Invariantes

Uma RA pode apresentar dois tipos básicos de propriedades: (a) aquelas que dependem unicamente da topologia (configuração de lugares e arcos) e (b) aquelas que dependem do comportamento dos agentes (critérios de utilidade, funções de transformação, sequência de disparo dos agentes, etc.). As propriedades do tipo (a) são chamadas de *propriedades invariantes*[39], pois uma vez determinadas podem ser aplicadas para todos os instantes de tempo da rede.

Uma das propriedades invariantes mais importantes em uma RA (com implicação direta na execução do algoritmo BMSA) vem a ser a identificação de regiões de independência – regiões onde todas as ações são sempre independentes. Assim, de acordo com o Teorema 3.1, o processo de busca indicado pelo Algoritmo 3.2 pode ser executado separadamente em cada uma destas regiões para a

[37]Conforme comentado na introdução deste capítulo, esta determinação engloba apenas os escopos habilitantes e função de transformação.

[38]Sincronização entre *threads* distintos (Tanenbaum, 1992). O processamento só pode prosseguir após todas as linhas de execução atingirem o ponto de *unificação*.

[39]Com referência à noção de invariantes em Redes de Petri (Murata, 1989).

obtenção da solução total.

Dependendo da configuração de lugares e arcos é possível encontrar conjuntos de lugares nos quais todas as propostas de ações apresentadas por agentes localizados em lugares de um destes conjuntos sempre sejam independentes em relação às ações de outros agentes localizados em lugares de outro conjunto. Esta propriedade é definida formalmente a seguir.

DEFINIÇÃO 3.53 (INDEPENDÊNCIA INVARIANTE)

Sejam:

- $\mathcal{R}$ uma rede de agentes, conforme a Definição 3.46;
- a definição formal de ação, conforme a Definição 3.48;
- coexist o predicado de coexistência, conforme a Definição 3.49; e
- $\overline{A} = \{A_0, A_1, \ldots A_n, \ldots\}$, onde cada $A_n \subseteq \mathbb{A}$ é o conjunto de ações propostas no instante $n \in \mathbf{N}$.

A propriedade de **independência invariante** é definida na forma:

$$\begin{aligned}
&\forall \Pi_a \subset \Pi \ \forall \Pi_b \subset \Pi \quad \mathsf{indepInv}(\Pi_a, \Pi_b) \iff \\
&\quad \Big[\forall n \in \mathbf{N} \ \ \forall a_i \in A_n \ \ \forall a_j \in A_n \\
&\qquad \Big(\xi\big(\mathsf{req}(a_i), n\big) \in \Pi_a \wedge \xi\big(\mathsf{req}(a_j), n\big) \in \Pi_b\Big) \Rightarrow \\
&\qquad \mathsf{coexist}(a_i, a_j)\Big]
\end{aligned} \tag{3.27}$$

TEOREMA 3.2 (CONDIÇÃO SUFICIENTE)

Sejam $\Pi_a \subset \Pi$ e $\Pi_b \subseteq \Pi$ (onde $\Pi_a \cap \Pi_b = \emptyset$) conjuntos de lugares. Uma condição suficiente para que Π_a e Π_b sejam invariavelmente independentes é que não existam arcos tipo público-privado entre nenhum lugar $\pi_i \in \Pi_a$ e $\pi_j \in \Pi_b$.

[Prova] (Teorema 3.2)
Sejam:

- $\mathcal{R}$ uma rede de agentes no instante n;
- $\Pi_a \subset \Pi$ e $\Pi_b \subset \Pi$, $\Pi_a \cap \Pi_b = \emptyset$, conjuntos de lugares. Como não existem arcos tipo público-privado entre lugares de Π_a e Π_b (conforme a hipótese do teorema), Π_a e Π_b apresentam a seguinte propriedade:

$$\begin{aligned}
&\forall \pi_a \in \Pi_a \ \forall \pi_b \in \Pi_b \\
&\quad \big(F'(\pi_a) \cap \Pi_b = V'(\pi_a) \cap \Pi_b = \emptyset\big) \wedge \\
&\quad \big(F'(\pi_b) \cap \Pi_a = V'(\pi_b) \cap \Pi_a = \emptyset\big)
\end{aligned}$$

- c_a e c_b dois agentes definidos em $\mathcal{R}$, tais que $\xi(c_a, n) \in \Pi_a$ e $\xi(c_b, n) \in \Pi_b$;

- $a_i \in \mathbb{A}$ e $a_j \in \mathbb{A}$ duas ações quaisquer dos agentes c_a e c_b[40], respectivamente, no instante n.

De acordo com a Definição 3.49, o predicado coexist é verdadeiro se *nenhuma* das três condições a seguir (apenas reescritas aqui por conveniência) for verdadeira:

a) duas ações propondo interações com um mesmo objeto. Tais ações só podem coexistir se ambas especificarem $\mathbf{I}^{\mathbf{S}}$ como seus modos de acesso;

b) uma das ações especificando uma interação com o requerente da outra; e

c) duas ações definem com o mesmo requerente.

De acordo com a restrição apresentada pela Equação 3.18 (pág. 47), todos os objetos presentes no escopo habilitante de um agente devem obrigatoriamente estar localizados em lugares fontes[41] conectados por arcos tipo *público-privado*. Isto implica que: c_a **não pode** acessar objetos definidos em lugares de Π_b e vice-versa.

Desta forma os itens (a) e (b) não podem acontecer e o item (c) também é *falso*, pois pela própria definição $c_a \neq c_b$. Portanto, Π_a e Π_b apresentam **independência invariante**. ■

Determinação das Regiões de Independência

Uma forma direta de determinação de regiões independentes em uma rede de agentes pode ser feita de acordo com os seguintes passos:

1. comece com o grafo formado pela configuração de lugares e arcos dadas por Π e $\mathcal{A}$.
2. remova todos os arcos privado-público.
3. separe todos os subgrafos desconexos resultantes, gerando para cada um deles um respectivo conjunto de lugares (dados pelos nós do subgrafo) $\Pi_i = \{\pi_{i_1}, \pi_{i_2}, \dots\}$. Qualquer combinação de dois conjuntos Π_i e Π_j satisfaz o critério de independência invariante.

Este processo está ilustrado na Figura 3.19, onde a rede inicial (3.19a) tem os arcos tipo privado-público (com linha tracejada) marcados. As regiões de independência são então mostradas na Figura 3.19b.

3.7 Resumo

Este capítulo apresentou uma revisão geral do modelo de redes de agentes introduzido por Guerrero (Guerrero, 2000). As várias modificações introduzidas, além da adição de novos conceitos, foram propostas com os seguintes objetivos principais:

- propor uma ontologia sobre objetos matemáticos (Gudwin, 1996) aplicável à teoria de agentes;
- oferecer maior proximidade de representação entre objetos matemáticos e a noção de agentes comumente empregada na literatura (Wooldridge and Jennings, 1995; Russel and Norvig, 1995; Green et al., 1997; Wooldridge, 1999);

[40] Em outras palavras $\mathsf{req}(a_i) = c_a$ e $\mathsf{req}(a_j) = c_b$.

[41] Fonte em relação ao lugar onde o agente se encontra.

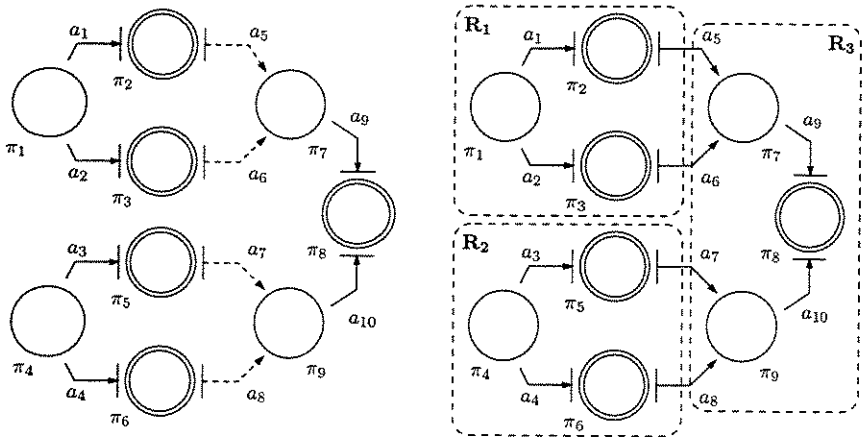


Figura 3.19: Determinação de regiões invariavelmente independentes: (a) marcação dos arcos privado-público e (b) regiões independentes.

- formalizar pontos definidos anteriormente somente do ponto de vista informal. O modelo proposto em (Guerrero et al., 1999) apresentou apenas de maneira informal a descrição da topologia de uma rede de agentes, bem como as noções de portas públicas e privadas; e
- definir um *framework* para o estudo da dinâmica de funcionamento de uma rede de agentes em termos de um problema de satisfação de restrições (Kumar, 1992; Hower, 1994; Russel and Norvig, 1995; Weiss, 1999), bastante difundido na literatura de Inteligência Artificial. Ainda neste tópico, apresentou-se uma análise mais formal para o algoritmo de determinação de dinâmica BMSA (Guerrero et al., 1999; Guerrero, 2000). Os resultados obtidos permitiram a identificação de propriedades invariantes das RAs, de grande relevância prática.

Como propostas de estudos futuros, vislumbra-se:

- extensão do modelo de redes de agentes para operação em tempo contínuo (mudando o espaço temporal de $\mathbf{N}$ para $\mathbf{R}$). Istou tornaria possível a aplicação de redes de agentes na modelagem de sistemas em tempo real;
- estudo mais aprofundado da teoria de Redes de Petri (Murata, 1989; Gerogiannis et al., 1998), enfocando a adaptação de métodos de análise formal em redes de agentes; e
- estudo de outros mecanismos de determinação de dinâmica entre os agentes, envolvendo noções de mercado, estabilidade, etc., bem como estudos adicionais para a operação em ambientes computacionais com acoplamento fracos.

Pode-se dizer que o estudo em ROs (e suas extensões) está ainda em fase de sedimentação no que se refere aos aspectos formais de análise.

Capítulo 4

Redes de Agentes Modulares

Este capítulo propõe uma importante extensão conceitual ao modelo de redes de agentes (RAs) introduzido inicialmente por Guerrero (Guerrero, 2000) e extensamente revisado no Capítulo 3[1]. Sua principal contribuição consiste na construção e simulação de modelos hierárquicos em termos de RAs.

4.1 Introdução

Para uma compreensão adequada da importância de modelos hierárquicos faz-se necessária a introdução da noção de sistema heterogêneo.

Um sistema *heterogêneo* é uma entidade complexa contituída de subsistemas com características diferentes. Geralmente tais sistemas são altamente concorrentes, de forma que as interações entre estes subsistemas podem ser extremamente complexas e de natureza bastante diferente. Os recentes avanços obtidos em eficiência e domínio da complexidade durante a modelagem e projeto de sistemas heterogêneos têm acontecido, basicamente, em função da introdução de novas linguagens e metodologias que trazem níveis adicionais de abstração. Tudo isto habilita os projetistas a tratar a complexidade continuamente crescente em níveis de detalhe apropriados. Sistemas heterogêneos frequentemente mostram um comportamento complexo que não pode ser completamente ou facilmente descrito em um simples formalismo. Assim, é desejável a existência de outros formalismos adequados a estes tipos de problemas (Esser, 1997). A simulação de sistemas complexos pode ser favorecida por uma representação em múltiplos níveis de abstração.

Existem certas aplicações nas quais seria particularmente interessante uma modelagem com vários níveis de atividade. Um simples exemplo disto pode ser um cruzamento de trânsito, onde os carros que se movem através da interseção podem ser considerados tanto como objetos de dados (como um objeto passivo representado apenas por seu estado) ou como um agente com suas próprias atividades internas, tais como consumo de petróleo, falha mecânica, etc.

Mecanismos de estruturação hierárquica facilitam um processo de projeto incremental *bottom-up* ou *top-down*, abstração de conceitos em resoluções adequadas, reutilização de componentes da rede, encapsulamento (informações detalhadas podem ser escondidas de forma adequadamente estruturada) e, finalmente, permitem uma separação clara dos componentes do sistema (Selic et al., 1994).

Conforme será discutido na próxima seção, mecanismos de estruturação hierárquica podem ser

[1]Mais especificamente indicado na Definição 3.46.

apresentados em níveis distintos, tratados por diversas ferramentas de modelagem especialmente adaptadas a este tipo de problema. Dentre estas ferramentas destacam-se as Redes de Petri (*Petri Nets* – PNs), por apresentarem uma extensa e bem elaborada base formal (Murata, 1989) e inúmeros subtipos derivados (Gerogiannis et al., 1998), cada qual abrangendo um tipo específico de extensão (algumas delas serão discutidas na próxima seção). Neste contexto, muitas das noções propostas para especificação das redes de agentes modulares têm inspiração direta em modelos de PNs.

4.1.1 Organização do Capítulo

A Seção 4.2 traça um rápido panorama das abordagens de introdução de mecanismos hierárquicos em Redes de Petri, além de definir o escopo de trabalho deste capítulo. A Seção 4.3 apresenta o modelo formal de rede de agentes modulares. A Seção 4.4 apresenta uma discussão referente a estudos futuros no âmbito das redes hierárquicas baseadas em superobjetos e a Seção 4.5 um resumo dos tópicos tratados neste capítulo.

4.2 Hierarquias

A noção de hierarquia trata da construção de estruturas capazes de atribuir níveis de resolução e/ou abstração ao modelo de um sistema. O primeiro subtópico a seguir traça um rápido panorama das propostas de inclusão de mecanismos de estruturação hierárquica em Redes de Petri. Ao final, são discutidas formas de introdução de estruturação hierárquica em RAs tendo como base as noções obtidas a partir das PNs.

4.2.1 Hierarquias em Redes de Petri

A maior restrição prática ao uso dos modelos de Redes de Petri de baixo nível, como as redes condição-evento (*condition-event* – CE-net) e lugar-transição (*place-transition* – PT-net), é que mesmo a modelagem de sistemas com complexidade média, em geral, requer redes bastantes extensas para sua representação, dificultando sua construção e análise (Gerogiannis et al., 1998). Por estas razões foram propostas as Redes de Petri de alto nível (*high-level* PNs – HPNs), capazes de representar de forma mais concisa e gerenciável sistemas de considerável complexidade, além de modelar o fluxo de dados e controle, condições/ações específicas de disparo, etc. De forma mais abrangente, as extensões mais comumente encontradas na literatura são (Gerogiannis et al., 1998):

1. extensões baseadas em fichas individuais (HPNs puras),
2. redes de alto nível com semântica modificada,
3. extensões baseadas em mecanismos de estruturação,
4. extensões que representam informações incertas (nebulosas) (Pedrycz, 1999; Cardoso et al., 1999) e
5. abordagens baseadas na integração com outros métodos de especificação.

Entre os modelos que apresentam estas abordagens destacam-se as do tipo indicado no item (1): Redes de Petri Coloridas (*Coloured Petri Net* – CPN) (Jensen, 1990), as redes predicado/transição (PrT-nets) (Genrich and Lautenbach, 1981) e as redes de ficha individual (ITNs) (Reisig, 1985, 1992).

Dependendo da complexidade, no que se refere às dimensões e heterogeneidade, *um modelo é melhor representado em termos de múltiplos níveis de abstração*, visto que mesmo uma rede de grandes dimensões ainda representa apenas uma única visão do sistema em questão. Isto motivou o desenvolvimento de mecanismos de estruturação e, em particular, a introdução de hierarquias (Redes de Petri hierárquicas de alto nível – HHPNs). HHPNs podem ser tratadas como uma subclasse específica das HPNs em que lugares e transições são substituídos por redes mais especializadas. Interessante dizer que mesmo em tais modelos a análise é efetuada no modelo executável plano.

Um exemplo típico de HHPNs é a CPN hierárquica (HCPN) (Huber et al., 1990; Gerogiannis et al., 1998). Uma HCPN consiste em um número de subredes hierarquicamente inter-relacionadas, chamadas *páginas*, que representam uma substituição para transições. Uma transição hierárquica pode ser substituída por uma página de forma a dar uma descrição mais detalhada da sequência interna de disparo da transição. Assim, o modelo pode ser descrito como um conjunto de subredes relacionadas (desenhadas em páginas distintas).

4.2.2 Redes de Petri Baseadas no Paradigma de Orientação a Objetos

Desde a década passada tem havido grande interesse na aplicação de tecnologias orientadas a objeto no domínio das PNs. Em alguns casos, a preocupação principal é fornecer uma base formal para linguagens orientadas a objeto ou tentar combinar tipos de dados abstratos com PNs, enquanto outras tentam aplicar as técnicas de orientação diretamente (Lakos and Keen, 1991; Lakos, 1995b,a; Valk, 1995; Maier and Moldt, 1997; Esser, 1997).

Lakos, em (Lakos, 1995b), apresenta um modelo detalhado onde o modelo CPN é gradualmente aprimorado até a elaboração das Redes de Petri a Objeto (*Object Petri Net* – OPN). Segundo ele, OPNs são bastante adequadas a problemas com diversos níveis de atividade, sistemas operacionais orientados a objeto e metodologias de *design* orientado a objetos. O formalismo das OPNs suporta uma ampla integração com os conceitos de orientação a objetos, como herança, polimorfismo e ligação dinâmica. Este modelo permite o tratamento de fichas como subredes independentes, oferecendo, desta forma, um nível bastante elevado de abstração. Entre os diversos modelos de PNs baseados no paradigma de orientação a objetos pode-se citar a linguagem textual LOOPN (Lakos and Keen, 1991), o modelo precursor das OPNs.

4.2.3 Hierarquias em Redes de Agentes

As PNs, como uma forte inspiração para redes de objetos (Gudwin, 1996), oferecem uma gama ampla de conceitos e termos pertinentes ao assunto tratado neste capítulo. Assim, há a preocupação com a reutilização de termos e conceitos equivalentes sempre que possível.

No contexo de RAs (Definição 3.46), constituída de lugares, arcos e objetos, existem duas possibilidades de introdução de mecanismos de estruturação hierárquica:

I. substituição de uma subrede por um **superlugar**. A semântica de uma subrede é – de certa forma – adaptada à semântica de um lugar (Definição 3.43), possuindo semelhança direta com o modelo utilizado nas redes HCPN descritas anteriormente. A composição, como será apresentada na

seção seguinte, é feita pela conexão de portas privadas de entrada e saída, empregando-se a noção de *fusão de lugares* (Christensen and Petrucci, 1992). Os componentes da hierarquia têm um *acoplamento forte*, visto que a organização das conexões é definida diretamente na configuração de lugares e arcos (definições 3.45 e 3.44).

II. substituição de uma subrede por um **superobjeto**. A semântica de uma subrede é adaptada à semântica de um objeto, possuindo semelhança com os modelos PN derivados do paradigma de orientação a objetos (Saleh et al., 1999; Pascoe, 1990; Wegner, 1986). Pode-se dizer que neste caso o acoplamento é *fraco*, pois a conexão das portas segue o mesmo princípio dos sensores e atuadores em um agente formal (Definição 3.28).

Este trabalho aborda um modelo de redes de agentes modular pela inclusão de superlugares. A seção a seguir trata de definir um modelo formal para a construção de tais estruturas.

4.3 Redes de Agentes Modulares

De forma bastante geral, a estrutura de uma rede de agentes modular (RAM) é constituída por um agrupamento de instâncias de páginas de subrede conectadas entre si. Sob este ponto de vista, uma página pode ser encarada como um tipo de dado abstrato (*abstract data type*). Em outras palavras, uma página é uma especificação genérica de uma coleção de lugares, arcos, interfaces de subrede e superlugares que pode ser instanciada.

Os lugares e arcos em uma RAM seguem a definição original apresentada no Capítulo 3. Dois outros elementos são adicionados: superlugares e interfaces (de subrede). Um *superlugar* é uma entidade topológica que representa toda uma subrede (descrita por uma outra página)[2]. Uma *interface*, por sua vez, apresenta um comportamento semelhante àquele associado a um lugar convencional, sendo empregado para comunicação com as *páginas-pai*[3]. Esta comunicação é baseada em um mecanismo chamado *fusão de lugares*, tipicamente empregado em HCPNs para implementar a composição de estruturas hierárquicas (Gerogiannis et al., 1998; Lakos, 1995b; Christensen and Petrucci, 1992). Dizer que dois ou mais lugares apresentam uma relação de fusão equivale à afirmação de que eles *são de fato o mesmo lugar* – todas as operações aplicadas a um deles se refletem nos outros, bem como as propriedades (objetos localizados em seu interior, portas públicas e privadas, etc.).

Convém ressaltar, mais uma vez, a distinção entre três conceitos importantes que serão constantemente empregados no decorrer das definições a seguir. Apesar de serem diferentes, podem representar fonte de confusão, caso sejam mal interpretados. São eles: página, superlugar e instância de página (vide Figura 4.1). Uma *página* nada mais é do que uma especificação genérica que indica uma certa configuração de lugares, arcos, interfaces e superlugares. Uma página pode incluir (ou de certa forma herdar) a especificação de outra página pela inclusão de um superlugar desta outra página. Constata-se, desta forma, que um superlugar representa a incorporação de uma página à definição de outra. Assim, um *superlugar* é uma referência a uma página, em um formato similar a um lugar convencional. É possível comparar uma página a uma classe em linguagens orientadas a objeto.

Por outro lado, para que seja possível simular uma rede de agentes modular (formada por uma coleção de páginas inter-relacionadas, etc.) é necessário criar-se uma *árvore de instanciação*, que des-

[2]Em linhas gerais se assemelha à idéia de superlugar apresentada por Lakos (Lakos, 1995b) no contexto de Redes de Petri Coloridas Modulares (MCPN).

[3]Páginas que contêm superlugares do tipo da página que define a interface.

creve uma coleção de *instâncias de página*, onde cada uma destas instâncias recebe um identificador único. Esta identificação se justifica pois uma mesma página pode ser instanciada múltiplas vezes dentro de uma mesma rede. Adota-se uma convenção na qual cada instância de página recebe um identificador *oid* (*object identifier*)[4]. Como requisito básico, considera-se que o sistema que implementa o processo de simulação seja capaz de atribuir um $oid \in OID$ único a todas as instâncias de páginas no momento de sua criação.

Dentre o conjunto de definições apresentadas adiante, destacam-se as definições de *página* e de *rede de agentes modular*. Definições adicionais são introduzidas de forma a permitir uma especificação mais ordenada. Convém ressaltar que estas definições são intimamente dependentes entre si. Mais do que isso, a definição de página é recursiva, pois uma página pode referenciar outras páginas. Dessa forma, na própria definição de página, encontra-se entre as premissas, um conjunto de páginas auxiliares, entendidas aqui como uma referência recursiva à própria noção de página.

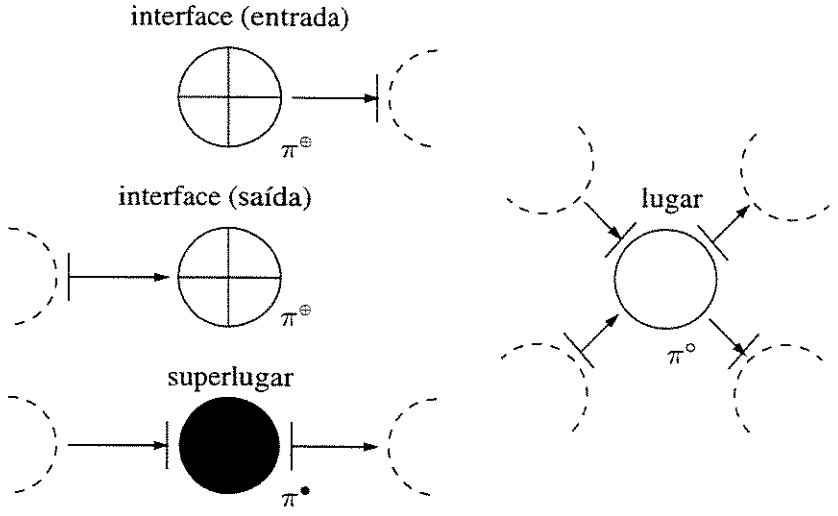


Figura 4.1: Entidades topológicas possíveis em uma página, com destaque para os tipos de portas possíveis em cada uma delas: interfaces (apenas portas públicas de entrada ou saída, não ambas), superlugares (apenas portas privadas de entrada e/ou saída) e lugares (tipicamente aceitam qualquer tipo de porta).

DEFINIÇÃO 4.1 (PÁGINA)

Sejam (no contexto de um conjunto de páginas):

- $\Sigma = \{C_i\}$ um conjunto de classes, onde cada C_i é descrita por um $\mathcal{D}^{\mathsf{c}}(C_i)$;
- $\Phi = \{\mathcal{S}_i\}$ um conjunto de páginas auxiliares referenciadas;
- $OID = \{oid_0, oid_1, \ldots, oid_{\mathsf{NID}}\} \subset \mathbf{OID}$, $\mathsf{NID} \geq 0$, um conjunto de identificadores globais para as instâncias desta página;

[4]Seguindo o mesmo raciocínio adotado em (Lakos, 1995b).

- $\Pi^{\circ} = \{\pi_1^{\circ}, \ldots, \pi_{\theta^{\circ}}^{\circ}\}$, $\theta^{\circ} \geq 0$, um conjunto de lugares;
- $\Pi^{\oplus} = \{\pi_1^{\oplus}, \ldots, \pi_{\theta^{\oplus}}^{\oplus}\}$, $\theta^{\oplus} \geq 0$, um conjunto de interfaces de subrede, tais que $\Pi^{\circ} \cap \Pi^{\oplus} = \emptyset$;
- $\Pi^{\bullet} = \{\pi_i^{\bullet} \ldots, \pi_{\theta^{\bullet}}^{\bullet}\}$, $\theta^{\bullet} \geq 0$, um conjunto de superlugares, tais que $\Pi^{\circ} \cap \Pi^{\bullet} = \emptyset$ e $\Pi^{\oplus} \cap \Pi^{\bullet} = \emptyset$;
- $\Pi = \Pi^{\circ} \cup \Pi^{\oplus} \cup \Pi^{\bullet} = \{\pi_i\}$, $\Pi^{\circ\oplus} = \Pi^{\circ} \cup \Pi^{\oplus} = \{\pi_i^{\circ\oplus}\}$ e $\Pi^{\circ\bullet} = \Pi^{\circ} \cup \Pi^{\bullet} = \{\pi_i^{\circ\bullet}\}$;
- $P = \{1, 2, \ldots\}$ um conjunto de números inteiros usados para identificação de portas privadas;
- $IE : \{1, \ldots, \theta^{\oplus}\} \rightarrow \Pi^{\oplus}$ uma função de mapeamento de interfaces usadas como entrada;
- $IS : \{1, \ldots, \theta^{\oplus}\} \rightarrow \Pi^{\oplus}$ uma função de mapeamento de interfaces usadas como saída;
- $\Xi : \Pi^{\circ\oplus} \rightarrow \Sigma$ uma função que atribui uma classe a cada lugar e interface de subrede;
- $\Xi' : \Pi^{\bullet} \rightarrow \Phi$ uma função que atribui uma página a cada superlugar;
- $\mathcal{A} = \{a_i\}$ um conjunto de arcos tais que $\Pi \cap \mathcal{A} = \emptyset$. $\mathcal{A}$ pode ser descrito pela união de dois conjuntos disjuntos $\mathcal{A}'$ e $\mathcal{A}''$ tais que $\mathcal{A}' \cap \mathcal{A}'' = \emptyset$ e $\mathcal{A}' \cup \mathcal{A}'' = \mathcal{A}$. $\mathcal{A}'$ descreve os arcos do tipo público-privado e $\mathcal{A}''$ descreve os arcos do tipo privado-público.
- $\eta' : A' \rightarrow \Pi^{\circ\oplus} \times (\Pi^{\circ\bullet} \times P)$ uma função de mapeamento de nós para arcos tipo público-privado, dada na forma $\eta'(a) = (\pi_i^{\circ\oplus}, (\pi_j^{\circ\bullet}, p))$, onde p, o índice da porta privada de entrada no lugar destino, está sujeito a duas restrições:
  - **(R1)** $\forall p \in P \quad p \in \{1, 2, \ldots, p_{\mathsf{max}}\} \subset P$. p_{max} é dado por[5]:

$$p_{\mathsf{max}} = \begin{cases} \theta_s \text{ da classe } \Xi(\pi_j^{\circ\bullet}) & \text{caso } \pi_j^{\circ\bullet} \in \Pi^{\circ} \\ \mathsf{Card}\left(\Pi^{\oplus}_{\Xi'(\pi_j^{\circ\bullet})}\right) & \text{caso } \pi_j^{\circ\bullet} \in \Pi^{\bullet} \end{cases}$$

  - **(R2)** Sejam $a_i \in \mathcal{A}'$, $a_j \in \mathcal{A}'$, $\eta'(a_i) = (\pi_i^{\circ\oplus}, (\pi_i^{\circ\bullet}, p_i))$, $\eta'(a_j) = (\pi_j^{\circ\oplus}, (\pi_j^{\circ\bullet}, p_j))$. Então a seguinte condição deve ser satisfeita: $(a_i \neq a_j \wedge \pi_i^{\circ\bullet} = \pi_j^{\circ\bullet}) \Rightarrow p_i \neq p_j$.
- $\eta'' : A'' \rightarrow (\Pi^{\circ\bullet} \times P) \times \Pi^{\circ\ominus}$ uma função de mapeamento de nós para arcos tipo privado-público, dada na forma $\eta''(a) = ((\pi_i^{\circ\bullet}, p), \pi_j^{\circ\oplus})$, onde p, o índice da porta privada de saída no lugar origem, está sujeito a duas restrições:
  - **(R1)** $p \in \{1, 2, \ldots, p_{\mathsf{max}}\} \subset P$. p_{max} é dado por:

$$p_{\mathsf{max}} = \begin{cases} \theta_s \text{ da classe } \Xi(\pi_i^{\circ\bullet}) & \text{caso } \pi_i^{\circ\bullet} \in \Pi^{\circ} \\ \mathsf{Card}\left(\Pi^{\oplus}_{\Xi'(\pi_i^{\circ\bullet})}\right) & \text{caso } \pi_i^{\circ\bullet} \in \Pi^{\bullet} \end{cases}$$

[5] $\Pi^{\oplus}_{\Xi'(\pi_j^{\circ\bullet})}$ corresponde ao conjunto de interfaces de subrede definidas na página descrita por $\Xi'(\pi_j^{\circ\bullet})$.

- **(R2)** Sejam $a_i \in \mathcal{A}''$, $a_j \in \mathcal{A}''$, $\eta''(a_i) = \left((\pi_i^{\circ\bullet}, p_i), \pi_i^{\circ\oplus}\right)$, $\eta''(a_j) = \left((\pi_j^{\circ\bullet}, p_j), \pi_j^{\circ\oplus}\right)$. Então a seguinte condição deve ser satisfeita: $(a_i \neq a_j \wedge \pi_i^{\circ\bullet} = \pi_j^{\circ\bullet}) \Rightarrow p_i \neq p_j$.

Uma **página de subrede** $\mathcal{S}$, definida no contexto de um conjunto de páginas Φ e de classes Σ, é então definida como uma ênupla que descreve uma configuração de lugares, arcos, interfaces e superlugares, no formato:

$$\mathcal{S} = (OID, \Pi^{\circ}, \Pi^{\ominus}, \Pi^{\bullet}, \mathcal{A}, \eta', \eta'', \Xi, \Xi', IE, IS) \tag{4.1}$$

Uma página pode incluir superlugares de outras páginas, formando uma *hierarquia de páginas*. A única restrição que esta hierarquia deve satisfazer é que o grafo formado, no qual os nós são representados pelas páginas e os arcos pelas relações de inclusão (criadas pelos superlugares), devem formar obrigatoriamente um *reticulado*. Esta restrição aplica-se a hierarquias de classes em geral nas linguagens orientadas a objeto e evita que uma classe possa ser, mesmo que indiretamente, uma superclasse de si mesma (Stefik and Bobrow, 1990).

Introduz-se neste trabalho uma notação para a representação de hierarquias de páginas em um formato semelhante ao adotato nos *diagramas de classes* em UML (*Unified Modeling Language*) (Booch et al., 1997). Este formato é denominado **diagrama de páginas**, tendo seus componentes ilustrados na Figura 4.2. Este tipo de representação é interessante, pois permite uma representação conceitual que abstrai a noção de instância, que será aplicada mais adiante.

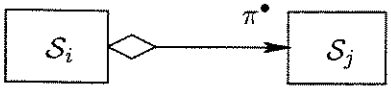


Figura 4.2: Notação empregada para diagramas de páginas. Neste caso a página $\mathcal{S}_i$ contém um superlugar $\pi^{\bullet}$ do tipo da página $\mathcal{S}_j$. Adicionalmente, $\mathcal{S}_i$ é dita uma *página-pai* de $\mathcal{S}_j$ ou, alternativamente, $\mathcal{S}_j$ uma *página-filha* de $\mathcal{S}_i$.

As duas definições apresentadas a seguir foram adaptadas a partir das funções F', F'', V', V'' construídas no corpo da definição de uma rede de agentes (Definição 3.46). Ambas se encontram fora da definição anterior por uma questão de conveniência.

DEFINIÇÃO 4.2 (FUNÇÕES LUGARES-FONTE DE UMA PÁGINA)

Sejam:

- $\mathcal{S}$ uma página;
- $P = \{1, 2, \dots\}$ o conjunto de identificadores de portas privadas;
- $\Pi^{\circ\bullet} = \Pi^{\circ} \cup \Pi^{\bullet}$ o conjunto de lugares e superlugares de $\mathcal{S}$;
- $\Pi^{\circ\oplus} = \Pi^{\circ} \cup \Pi^{\ominus}$ o conjunto de lugares e interfaces de $\mathcal{S}$;

As **funções de mapeamento de lugares-fonte** via arcos público-privado e privado-público de uma página são construídas, respectivamente, na forma:

$$F' : \Pi^{\circ\bullet} \times P \rightarrow \Pi^{\circ\ominus}$$
$$F'(\pi^{\circ\bullet}, p) = \pi_i^{\circ\ominus} \in \Pi^{\circ\ominus} \mid \exists a_k \in \mathcal{A}' \quad \eta'(a_k) = \left(\pi_i^{\circ\oplus}, (\pi^{\circ\bullet}, p)\right)$$

$$F'' : \Pi^{\circ\oplus} \rightarrow 2^{\Pi^{\circ\bullet}}$$
$$F''(\pi^{\circ\oplus}) = \left\{ \pi_i^{\circ\bullet} \in \Pi^{\circ\bullet} \mid \exists a_k \in \mathcal{A}'',\ \exists p \in P \quad \eta''(a_k) = (\pi_i^{\circ\bullet}, (\pi^{\circ\oplus}, p)) \right\}$$

DEFINIÇÃO 4.3 (FUNÇÕES LUGARES-VERTEDOURO DE UMA PÁGINA)

Sejam:

- $\mathcal{S}$ uma página;
- $P = \{1, 2, \dots\}$ o conjunto de identificadores de portas privadas;
- $\Pi^{\circ\bullet} = \Pi^{\circ} \cup \Pi^{\bullet}$ o conjunto de lugares e superlugares de $\mathcal{S}$;
- $\Pi^{\circ\oplus} = \Pi^{\circ} \cup \Pi^{\oplus}$ o conjunto de lugares e interfaces de $\mathcal{S}$;

As **funções de mapeamento de lugares-vertedouro** via arcos público-privado e privado-público de uma página são construídas, respectivamente, na forma:

$$V' : \Pi^{\circ\oplus} \rightarrow 2^{\Pi^{\circ\bullet}}$$
$$V'(\pi^{\circ\oplus}) = \left\{ \pi_i^{\circ\bullet} \in \Pi^{\circ\bullet} \mid \exists a_k \in \mathcal{A}',\ \exists p \in P \quad \eta'(a_k) = \left((\pi^{\circ\bullet}, p), \pi_i^{\circ\oplus}\right) \right\}$$

$$V'' : \Pi^{\circ\bullet} \times P \rightarrow \Pi^{\circ\oplus}$$
$$V''(\pi^{\circ\bullet}, p) = \pi_j^{\circ\oplus} \in \Pi^{\circ\oplus} \mid \exists a_k \in \mathcal{A}'' \quad \eta''(a_k) = \left((\pi^{\circ\bullet}, p), \pi_j^{\circ\oplus}\right)$$

As duas definições dadas a seguir são auxiliares, e servem de base para a formalização final de uma RAM no que se refere ao estabelecimento das duas restrições topológicas fundamentais[6]. Deste ponto em diante convém ressaltar a distinção que será feita entre lugares, interfaces e superlugares, definidos em uma página $\mathcal{S}$, e os respectivos elementos definidos em uma *instância de página* $\mathcal{S}$. O primeiro é uma especificação abstrata, o segundo é a instância do outro, sendo sempre representado pela associação de dois elementos distintos: (a) o lugar, superlugar ou interface definida na página e (b) o identificador da instância da página. Por exemplo, seja uma página $\mathcal{S}$ na qual se define um lugar π_1°. Considere duas instâncias desta página $\mathcal{S}$ referenciadas por identificadores oid_1 e oid_2. As respectivas instâncias do lugar π_1° são referenciadas por (π_1°, oid_1) e (π_1°, oid_2).

DEFINIÇÃO 4.4 (FUNÇÕES LUGARES-FONTE GLOBAIS)

Sejam:

- $\Phi = \{\mathcal{S}_i\}$, $i = 0, 1, \dots$, um conjunto de páginas;
- $OID = \{oid_0, oid_1, \dots, oid_{\mathsf{NID}}\} \subset \mathbf{OID}$, NID ≥ 0, o conjunto de identificadores globais para as instâncias de páginas;

[6] Introduzidas inicialmente em (Gudwin, 1996) e reescritas nas equações 3.18 e 3.19.

- ID : $\mathbf{OID} \rightarrow \Phi$ a função de identificação de instâncias de páginas, que associa uma página de subrede a cada identificador de instância;
- $P = \{1, 2, \dots\}$ o conjunto de identificadores de portas privadas;
- $\overline{\Pi}^{\circ\bullet} = \overline{\Pi}^{\circ} \cup \overline{\Pi}^{\bullet}$ o conjunto de todos os lugares e superlugares das páginas de Φ;
- $\overline{\Pi}^{\circ\ominus} = \overline{\Pi}^{\circ} \cup \overline{\Pi}^{\ominus}$ o conjunto de todos os lugares e interfaces das páginas de Φ;
- $F'_{ID(oid)}(\pi, p)$ a função lugar-fonte via arcos tipo público-privado definida na página da instância identificada por oid; e
- $F''_{ID(oid)}(\pi)$ a função lugares-fonte via arcos tipo privado-público definida na página da instância identificada por oid.

As **funções lugares-fonte globais** via arcos público-privado e privado-público de uma página são construídas, respectivamente, na forma:

$$\overline{F}' : \overline{\Pi}^{\circ\bullet} \times \mathbf{OID} \times P \rightarrow \overline{\Pi}^{\circ\ominus} \times \mathbf{OID}$$
$$\overline{F}'(\pi^{\circ\bullet}, oid, p) = \left(F'_{ID(oid)}(\pi^{\circ\bullet}, p), oid\right)$$

$$\overline{F}'' : \overline{\Pi}^{\circ\ominus} \times \mathbf{OID} \rightarrow 2^{\overline{\Pi}^{\circ\bullet} \times \mathbf{OID}}$$
$$\overline{F}''(\pi^{\circ\oplus}, oid) = F''_{ID(oid)}(\pi^{\circ\ominus}) \times \{oid\}$$

DEFINIÇÃO 4.5 (FUNÇÕES LUGARES-VERTEDOURO GLOBAIS)

Sejam:

- $\Phi = \{\mathcal{S}_i\}$, $i = 0, 1, \dots$, um conjunto de páginas;
- $OID = \{oid_0, oid_1, \dots, oid_{\mathsf{NID}}\} \subset \mathbf{OID}$, NID ≥ 0, o conjunto de identificadores globais para as instâncias de páginas;
- ID : $\mathbf{OID} \rightarrow \Phi$ a função de identificação de instâncias de páginas, que associa uma página de subrede a cada identificador de instância;
- $P = \{1, 2, \dots\}$ o conjunto de identificadores de portas privadas;
- $\overline{\Pi}^{\circ\bullet} = \overline{\Pi}^{\circ} \cup \overline{\Pi}^{\bullet}$ o conjunto de todos os lugares e superlugares das páginas de Φ;
- $\overline{\Pi}^{\circ\ominus} = \overline{\Pi}^{\circ} \cup \overline{\Pi}^{\ominus}$ o conjunto de todos os lugares e interfaces das páginas de Φ;
- $V'_{ID(oid)}(\pi)$ a função lugares-vertedouro via arcos tipo público-privado definida na página da instância identificada por oid; e
- $V''_{ID(oid)}(\pi, p)$ a função lugar-vertedouro via arcos tipo privado-público definida na página da instância identificada por oid.

As **funções lugares-vertedouro globais** via arcos público-privado e privado-público de uma página são construídas, respectivamente, na forma:

$$\overline{V}' : \overline{\Pi}^{\circ\oplus} \times \mathbf{OID} \rightarrow 2^{\overline{\Pi}^{\circ\bullet} \times \mathbf{OID}}$$
$$\overline{V}'(\pi^{\circ\oplus}, oid) = V'_{ID(oid)}(\pi^{\circ\oplus}) \times \{oid\}$$

$$\overline{V}'' : \overline{\Pi}^{\circ\bullet} \times \mathbf{OID} \times P \rightarrow \overline{\Pi}^{\circ\oplus} \times \mathbf{OID}$$
$$\overline{V}''(\pi^{\circ\bullet}, oid, p) = \left(V''_{ID(oid)}(\pi^{\circ\bullet}, p), oid\right)$$

Conforme introduzido no início desta seção, a noção de *fusão de lugares* é bastante útil na composição de redes hierárquicas, por permitir a comunicação entre os níveis da hierarquia. Nas redes de agentes modulares, todas as instâncias de interface são na verdade referências para instâncias de lugares definidos em outra instância de página (a página-pai). Um objeto enviado para uma instância de interface é de fato enviado para um lugar na hierarquia superior.

DEFINIÇÃO 4.6 (FUNÇÃO DE FUSÃO DE LUGARES)

Sejam:

- $\Phi = \{\mathcal{S}_i\}$, $i = 0, 1, \ldots$, um conjunto de páginas;
- $OID = \{oid_0, oid_1, \ldots, oid_{\mathsf{NID}}\} \subset \mathbf{OID}$, NID ≥ 0, o conjunto de identificadores globais para as instâncias de páginas;
- $\overline{\Pi}^{\circ}$ o conjunto de todos os lugares das páginas de Φ;
- $\overline{\Pi}^{\oplus}$ o conjunto de todas as interfaces das páginas de Φ;
- $\overline{\Pi}^{\circ\oplus} = \overline{\Pi}^{\circ} \cup \overline{\Pi}^{\oplus}$ o conjunto de todos os lugares e interfaces das páginas de Φ;

Uma **função de fusão de lugares** associa uma interface $\pi^{\oplus}$ em uma instância de página a um lugar π° em uma outra instância de página, sendo definida como uma função no formato:

$$\widetilde{F} : \overline{\Pi}^{\oplus} \times \mathbf{OID} \rightarrow \overline{\Pi}^{\circ} \times \mathbf{OID}$$

Adicionalmente, por conveniência, define-se a **função de fusão de lugares generalizada** baseada na definição de $\widetilde{F}$:

$$\widetilde{F}_{\mathrm{g}} : \overline{\Pi}^{\oplus\circ} \times \mathbf{OID} \rightarrow \overline{\Pi}^{\circ} \times \mathbf{OID}$$
$$\widetilde{F}_{\mathrm{g}}(\pi, oid) = \begin{cases} (\pi, oid) & \text{caso } \pi \in \overline{\Pi}^{\circ} \\ \widetilde{F}(\pi, oid) & \text{nos demais casos} \end{cases}$$

A atribuição de identificadores únicos às instâncias de uma coleção de páginas e a definição da função de fusão podem ser especificadas de inúmeras formas. Apresenta-se a seguir um possível algoritmo para a solução deste problema. É interessante observar que a quantidade de instâncias de páginas depende unicamente das inter-relações entre as próprias páginas, ditadas pela distribuição dos superlugares. Isto pode ser observado de forma mais clara no diagrama de páginas (Figura 4.2).

Como hipótese inicial, admite-se a existência de uma *página raiz*, uma página com duas propriedades básicas: (a) nenhuma outra página pode incluir um superlugar do tipo desta página e (b) esta página não pode conter nenhuma interface (a rede de agentes, como um todo, ainda é um sistema fechado). A página raiz sempre é a primeira a receber um identificador e, adicionalmente, só pode ter uma única instância em toda a rede.

O algoritmo proposto funciona de forma construtiva, iniciando-se a partir da página raiz, referenciada por $\mathcal{S}_0$, e percorrendo uma *árvore de instanciação* numa busca em profundidade. Cada um dos superlugares de $\mathcal{S}_0$ tem a respectiva página instanciada de forma recursiva, até que se chegue a um superlugar cuja página não tenha mais outros superlugares[7] (veja as figuras 4.3 e 4.4). Durante a execução três componentes são gradativamente gerados: (a) o conjunto de identificadores de instâncias de páginas $OID \subset \mathbf{OID}$, (b) uma função de identificação de tipo de instância $ID : \mathbf{OID} \rightarrow \Phi$ e (b) a função de fusão de lugares $\widetilde{F}$. Note que estas duas funções são tratadas aqui como conjuntos, $f : A \rightarrow B$ na forma $f \subseteq A \times B$. O ponto de entrada do programa é codificado no Algoritmo 4.1, que faz uso de duas outras funções, implementadas pelos algoritmos 4.2 e 4.3. Para a operação do programa admite-se que:

- a expressão a seguir é satisfeita: $\forall \mathcal{S}_i \in \Phi \; \forall \mathcal{S}_j \in \Phi \quad \Pi_{\mathcal{S}_i} \neq \Pi_{\mathcal{S}_j} \Rightarrow \Pi_{\mathcal{S}_i} \cap \Pi_{\mathcal{S}_j} = \emptyset$; e
- exitem um conjunto de classes Σ, um conjunto de páginas Φ e uma função $getNextOID$[8].

Adicionalmente, adota-se uma notação em que os componentes da ênupla que formam uma página (Equação 4.1) e as funções F',F'',V' e V'' podem ser referenciados em relação a uma página específica pela inclusão do nome da página em subscrito. Por exemplo, $\Pi^{\circ}_{\mathcal{S}_i}$ indica o conjunto de lugares definidos na página $\mathcal{S}_i$.

Algoritmo 4.1.

```
1  OID := {oid_0};
2  ID := {(oid_0, S_0)};
3  F̃ := ∅;
4  newInstance(OID, ID, S_0, oid_0, F̃);
```

Algoritmo 4.2.

```
1  proc newInstance(OID : 2^OID, ID : 2^(OID×Φ), S : Φ,
2      oid : OID, F̃ : (Π̄^⊕ × OID) × (Π̄^∘ × OID)) ≡
3    foreach π• ∈ Π•_S
4    do
5      oid' := getNextOID(OID);
6      OID := OID ∪ {oid'};
7      ID := ID ∪ {(oid', Ξ'_S(π•))};
8      makeFusion(S, π•, oid, oid', F̃);
9      newInstance(OID, ID, Ξ'_S(π•), oid', F̃);
10   od
```

[7]Note que, conforme descrito anteriormente, uma hierarquia de páginas deve formar obrigatoriamente um reticulado. Pois, caso contrário, o processo de instanciação entra em recursão infinita.

[8]A função $getNextOID : 2^{\mathbf{OID}} \rightarrow \mathbf{OID}$ abstrai a geração de um novo indentificador a partir do conhecimento dos outros, garantido assim a unicidade. Por exemplo, para $\mathbf{OID} \equiv \mathbf{N}$, o exemplo mais simples poderia ser uma função que retornasse $(\max_i oid_i) + 1$.

Algoritmo 4.3.

```
1  proc makeFusion(S : Φ, π• : Π̄•, oid : OID,
2      oid' : OID, F̃ : (Π̄⊕ × OID) × (Π̄° × OID)) ≡
3    S' := Ξ_S(π•);
4    P' := {p' ∈ P | ∃π°⊕ ∈ Π°⊕_S   π°⊕ = F'_S(π•, p)};
5    foreach p ∈ P'
6    do
7      π°⊕ := F'_S(π•, p);
8      if π°⊕ ∈ Π°⊕
9        then π := F̃(π°⊖, oid);
10       else π := π°⊕;
11     fi
12     ins1 := (IE_S'(p), oid');
13     ins2 := (π, oid);
14     F̃ := F̃ ∪ {(ins1, ins2)};
15   od
16   P'' := {p'' ∈ P | ∃π°⊕ ∈ Π°⊕_S   π°⊕ = V''_S(π•, p)};
17   foreach p ∈ P''
18   do
19     π°⊖ := V''_S(π•, p);
20     if π°⊕ ∈ Π°⊖
21       then π := F̃(π°⊖, oid);
22       else π := π°⊖;
23     fi
24     ins1 := (IS_S'(p), oid');
25     ins2 := (π, oid);
26     F̃ := F̃ ∪ {(ins1, ins2)};
27   od
```

DEFINIÇÃO 4.7 (REDE DE AGENTES MODULAR)

Sejam:

- $OID = \{oid_0, oid_1, \ldots, oid_{\text{NID}}\} \subset \mathbf{OID}$, NID ≥ 0, o conjunto de identificadores globais para as instâncias de páginas. oid_0 é o *identificador da página raiz*;
- $\Sigma = \{C_i\}$ um conjunto de classes. Cada C_i é descrita por um descritor de classe $\mathcal{D}^c(C_i)$;
- $\Phi = \{\mathcal{S}_i\}$, $i \geq 0$, um conjunto de páginas. $\mathcal{S}_0$ é chamada a *página raiz*. Adicionalmente, todas as páginas em Φ devem satisfazer a seguinte restrição: $\forall \mathcal{S}_i \in \Phi \; \forall \mathcal{S}_j \in \Phi \quad \Pi_{\mathcal{S}_i} \neq \Pi_{\mathcal{S}_j} \Rightarrow \Pi_{\mathcal{S}_i} \cap \Pi_{\mathcal{S}_j} = \emptyset$;

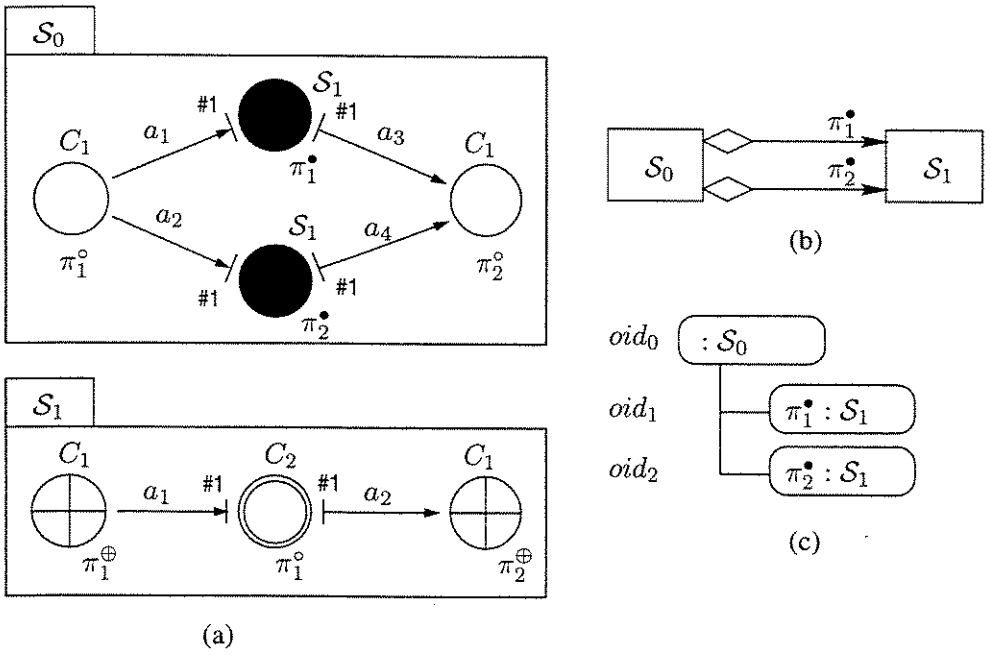


Figura 4.3: Coleção de páginas inter-relacionadas de acordo com o diagrama de páginas em (b). A parte (c) apresenta a árvore de instanciação.

- ID : **OID** $\rightarrow \Phi$ a função de identificação de instâncias de páginas, que associa uma página de subrede a cada identificador de instância;
- $\mathcal{C}_{(oid_i)}$ o conjunto de objetos localizados na instância de página identificada por oid_i;
- $\mathcal{C}$ um conjunto de objetos localizados em todas as instâncias de página:

$$\mathcal{C} = \bigcup_{oid_i \in OID} \mathcal{C}_{(oid_i)}$$

- $\overline{\Pi}^\circ$ o conjunto de todos os lugares definidos dentro das páginas de Φ;
- $\xi : \mathcal{C} \times \mathbf{N} \rightarrow \overline{\Pi}^\circ \times \mathbf{OID}$ uma função de localização de objetos, que associa um lugar em uma instância de página a cada objeto;
- $\Gamma = \{\gamma_i(n)\}$ o conjunto das funções de seleção. Cada $\gamma_i(n)$ é a função de seleção para o objeto c_i, apresentando as mesmas restrições da definição 3.41, na página 41;
- $\mathcal{C}^0 = \{c_i^0\}$ um conjunto de *objetos iniciais*;
- $\xi^0 : \mathcal{C}^0 \rightarrow \overline{\Pi}^\circ \times \mathbf{OID}$ uma função de *localização inicial* que associa um lugar π°, em uma instância de página identificada por oid_i, a cada objeto c_i^0; e
- $\widetilde{F}_g$ uma função de fusão de lugares generalizada.

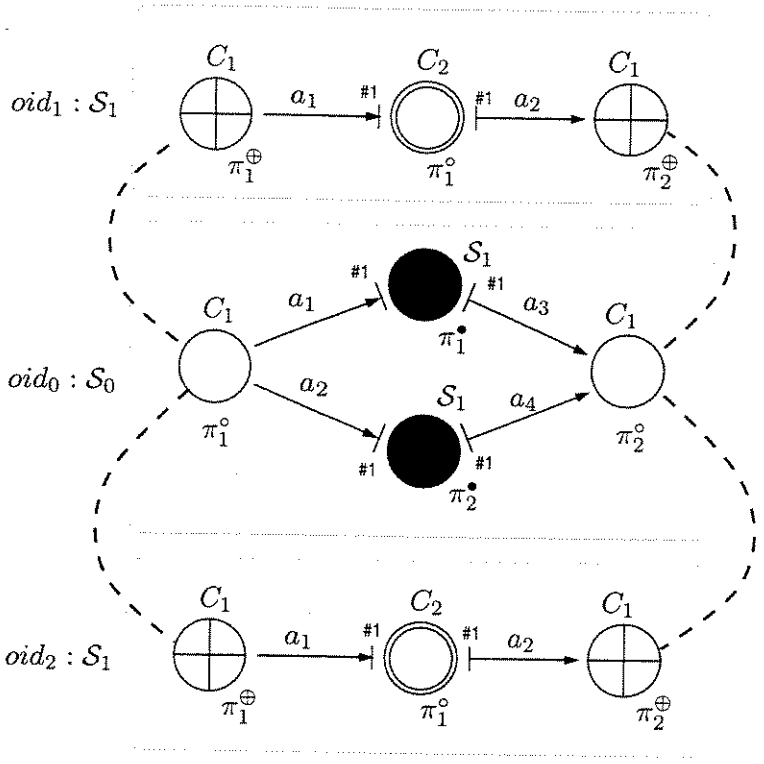


Figura 4.4: Processo de instanciação da coleção de páginas apresentadas na Figura 4.3. As linhas tracejadas representam as relações de fusão.

Uma rede de agentes modular é dada por uma ênupla no seguinte formato:

$$\mathcal{R} = (OID, \Sigma, \Phi, \mathcal{C}, \xi, \Gamma, \mathcal{C}^0, \xi^0, \mathcal{S}_0, oid_0) \tag{4.2}$$

De forma análoga às redes de agentes (Definição 3.46), as seguintes restrições devem ser cumpridas:

- **(Restrição 1)** $\mathcal{R}$ deve constituir, por meio de $\mathcal{C}$ e Γ, um sistema de agentes (Definição 3.42);
- **(Restrição 2) - restrição topológica nos escopos habilitantes**
  Para cada um dos agentes c_i com escopo habilitante não-vazio considere:
  - $H_i(\delta, n) = (h, b)$, $1 \leq \delta \leq \theta_f$, é o escopo habilitante do agente c_i em n. δ indica a função na qual H_i está definido[9];
  - θ_s o número de sensores e α a ênupla descritora de domínio para os sensores definidos na classe do agente c_i (vide Definição 3.21);

[9]Em outras palavras, δ especifica a função escolhida para o disparo de c_i.

- $h = (h_1, \ldots, h_k, \ldots, h_r)$, onde $r = \mathsf{Ar}(\alpha)$[10], a parcela do escopo habilitante referente aos sensores; e
- $(\pi_i^\circ, oid_i) = \xi(c_i, n)$.

A seguinte proposição deve ser satisfeita:

$$\forall k \in \{1, \ldots, \theta_s\} \;\; \widetilde{F}_{\mathrm{g}}\Big(\overline{F}'\,(\pi_i^\circ, oid_i, k)\Big) = \xi(h_k, n) \tag{4.3}$$

- **(Restrição 3) - restrição topológica nos escopos gerativos**
  Para cada um dos agentes c_i com escopo gerativo não-vazio considere:
  - $S_i(\delta, n)$, $1 \leq \delta \leq \theta_f$, é o escopo gerativo do agente c_i em n. δ indica a função na qual S_i está definido em n;
  - θ_a o número de atuadores e β a ênupla descritora de contradomínio para os atuadores definidos na classe do agente c_i (vide Definição 3.21);
  - $s = (s_1, \ldots, s_k, \ldots, s_r)$, onde $r = \mathsf{Ar}(\beta)$[11], a parcela do escopo gerativo referente aos atuadores; e
  - $(\pi_i^\circ, oid_i) = \xi(c_i, n)$.

  A seguinte proposição deve ser satisfeita:

$$\forall k \in \{1, \ldots, \theta_a\} \;\; \widetilde{F}_{\mathrm{g}}\Big(\overline{V}''\,(\pi_i^\circ, oid_i, k)\Big) = \xi(s_k, n+1) \tag{4.4}$$

- **(Restrição 4) - regra de instanciação de páginas**
  OID, ID e $\widetilde{F}_{\mathrm{g}}$ são definidos de acordo com um *algoritmo de instanciação* que siga os inter-relacionamentos entre as páginas (Algoritmo 4.1).

4.3.1 Exemplo

Esta subseção apresenta um exemplo de rede de agentes modular com o objetivo de ilustrar os conceitos introduzidos na seção anterior.

O modelo (Figura 4.5) é composto por três páginas $\mathcal{S}_0$, $\mathcal{S}_1$ e $\mathcal{S}_2$ inter-relacionadas de acordo com a diagrama de páginas em anexo. Sejam dadas as classes $\Sigma = \{C_1, C_2, C_3, C_4\}$:

- **Página** $\mathcal{S}_0$:
  $\Pi^\circ = \{\pi_{01}^\circ, \pi_{02}^\circ\}$
  $\Pi^\ominus = \{\emptyset\}$
  $\Pi^\bullet = \{\pi_{01}^\bullet, \pi_{02}^\bullet\}$
  $\Xi = \{(\pi_{01}^\circ, C_2), (\pi_{02}^\circ, C_3), \}$
  $\Xi' = \{(\pi_{01}^\bullet, \mathcal{S}_1), (\pi_{02}^\bullet, \mathcal{S}_2)\}$
  $\mathcal{A}' = \{a_{01}, a_{02}\}$ e $\mathcal{A}'' = \{a_{03}, a_{04}\}$
  $\eta'(a_{01}) = (\pi_{01}^\circ, (\pi_{01}^\bullet, 1))$

[10] Referente ao domínio de f_δ
[11] Referente ao contradomínio de f_δ

$\eta'(a_{02}) = \left(\pi^{\circ}_{01}, (\pi^{\bullet}_{02}, 1)\right)$
$\eta''(a_{03}) = \left((\pi^{\bullet}_{01}, 1), \pi^{\circ}_{02}\right)$
$\eta''(a_{04}) = \left((\pi^{\bullet}_{02}, 1), \pi^{\circ}_{02}\right)$
$IE = \emptyset$
$IS = \emptyset$

- **Página** $\mathcal{S}_1$:
  $\Pi^{\circ} = \{\pi^{\circ}_{11}\}$
  $\Pi^{\oplus} = \{\pi^{\oplus}_{11}, \pi^{\oplus}_{12}\}$
  $\Pi^{\bullet} = \emptyset$
  $\Xi = \left\{(\pi^{\circ}_{11}, C_2), (\pi^{\oplus}_{11}, C_1), (\pi^{\oplus}_{12}, C_3)\right\}$
  $\Xi' = \emptyset$
  $\mathcal{A}' = \{a_{11}\}$ e $\mathcal{A}'' = \{a_{12}\}$
  $\eta'(a_{11}) = \left(\pi^{\oplus}_{11}, (\pi^{\circ}_{11}, 1)\right)$
  $\eta''(a_{12}) = \left((\pi^{\circ}_{11}, 1), \pi^{\oplus}_{12}\right)$
  $IE = \left\{(1, \pi^{\oplus}_{11})\right\}$
  $IS = \left\{(1, \pi^{\oplus}_{12})\right\}$

- **Página** $\mathcal{S}_2$:
  $\Pi^{\circ} = \{\pi^{\circ}_{21}, \pi^{\circ}_{22}\}$
  $\Pi^{\oplus} = \{\pi^{\oplus}_{21}, \pi^{\oplus}_{22}\}$
  $\Pi^{\bullet} = \{\pi^{\bullet}_{21}\}$
  $\Xi = \left\{(\pi^{\circ}_{21}, C_2), (\pi^{\circ}_{22}, C_4), (\pi^{\oplus}_{21}, C_1), (\pi^{\oplus}_{22}, C_3)\right\}$
  $\Xi' = \left\{(\pi^{\bullet}_{21}, \mathcal{S}_1)\right\}$
  $\mathcal{A}' = \{a_{21}, a_{22}, a_{23}\}$ e $\mathcal{A}'' = \{a_{24}, a_{25}, a_{26}\}$
  $\eta'(a_{21}) = \left(\pi^{\oplus}_{21}, (\pi^{\circ}_{21}, 1)\right)$
  $\eta'(a_{22}) = \left(\pi^{\oplus}_{21}, (\pi^{\bullet}_{21}, 1)\right)$
  $\eta'(a_{23}) = \left(\pi^{\circ}_{21}, (\pi^{\circ}_{22}, 1)\right)$
  $\eta''(a_{24}) = \left((\pi^{\circ}_{21}, 1), \pi^{\oplus}_{22}\right)$
  $\eta''(a_{25}) = \left((\pi^{\circ}_{22}, 1), \pi^{\circ}_{21}\right)$
  $\eta''(a_{26}) = \left((\pi^{\bullet}_{21}, 1), \pi^{\oplus}_{22}\right)$

  $IE = \left\{(1, \pi^{\oplus}_{21})\right\}$
  $IS = \left\{(1, \pi^{\oplus}_{22})\right\}$

A partir da execução do algoritmo de instanciação tem-se $OID = \{oid_0, oid_1, oid_2, oid_3\}$, resultando na seguinte função de fusão:

$\widetilde{F}(\pi^{\oplus}_{11}, oid_1) = (\pi^{\circ}_{01}, oid_0)$
$\widetilde{F}(\pi^{\oplus}_{12}, oid_1) = (\pi^{\circ}_{02}, oid_0)$
$\widetilde{F}(\pi^{\oplus}_{21}, oid_2) = (\pi^{\circ}_{01}, oid_0)$
$\widetilde{F}(\pi^{\oplus}_{22}, oid_2) = (\pi^{\circ}_{02}, oid_0)$
$\widetilde{F}(\pi^{\oplus}_{11}, oid_3) = (\pi^{\circ}_{01}, oid_0)$
$\widetilde{F}(\pi^{\oplus}_{12}, oid_3) = (\pi^{\circ}_{02}, oid_0)$

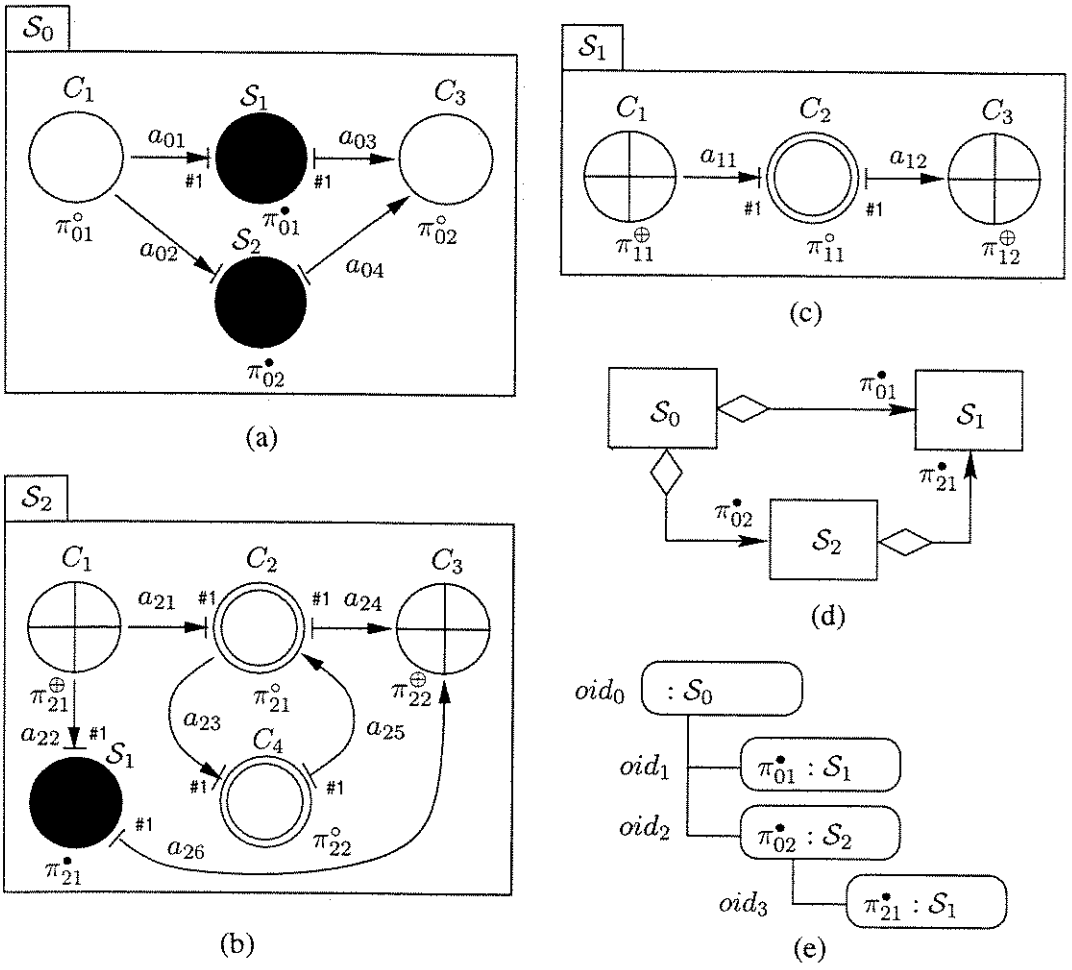


Figura 4.5: Rede exemplo mostrando uma RAM formada por três páginas (a), (b) e (c) inter-relacionadas entre si de acordo com o diagrama de páginas em (d). A parte (e) apresenta a árvore de instanciação.

4.4 Redes de Agentes Baseadas em Superobjetos

Esta seção discute algumas idéias gerais referentes à construção de redes de agentes hierárquicas baseadas em superobjetos. O intuito principal é o de oferecer subsídios a trabalhos futuros.

Embora as redes de agentes modulares ofereçam um maior conforto na modelagem de sistemas heterogêneos e de maior complexidade, elas não estendem o poder de representação das redes de agentes. Além disso, toda uma rede de agentes modular ainda se comporta como um único *sistema de agentes*[12].

Na definição de um superlugar (como a abstração de uma referência a uma página), implementou-se um mecanismo de identificação para a criação de instâncias de página capazes de se comunicarem por meio da fusão de suas interfaces com os lugares definidos nas páginas-pai. Este tipo de composição

[12]Definição 3.42.

é de certa forma restrita por estar fortemente ligada à configuração existente entre as páginas, conforme pode ser observado pelo diagramas de páginas. Esta característica pode ser efetivamente identificada no algoritmo de instanciação, onde a função de fusão depende apenas da configuração de páginas, não podendo ser modificada durante a simulação da rede.

De forma análoga, esta mesma situação foi identificada nas Redes de Petri Hierárquicas (Gerogiannis et al., 1998). O argumento apresentado foi que a estruturação, aqui chamada de estática, ou fortemente acoplada, não acrescenta características de modelagem que permitam uma representação genuinamente multiresolucional em múltiplos e distintos níveis de atividade (Lakos, 1995b).

Uma abordagem que pode ser sugerida para resolver este problema, no domínio das redes de agentes[13], é permitir que a função de fusão possa ser ajustada dinamicamente com o tempo. De maneira ainda mais geral, uma instância de página poderia ser adaptada para que se comportasse, de certa forma, como um objeto matemático, apresentando as seguintes características, tipicamente encontradas nos objetos matemáticos (e mais notadamente nos agentes formais):

- **mobilidade**, podendo se locomover pela rede de acordo com a configuração de lugares e arcos;
- **ciclo de vida**, sendo criados e destruídos dinamicamente durante o processamento da rede, além de poderem apresentar um nível de atividade próprio;
- **capacidade de perceber e atuar no meio**, de forma semelhante a um agente formal. Onde a interação com o meio aconteceria através do mecanismo de fusão aplicado às suas interfaces de entrada (neste caso funcionando como sensores) e suas interfaces de saída (funcionando com atuadores).

Outro ponto de interesse a ser considerado é que, neste cenário, cada uma das redes em operação dentro destes superobjetos poderia se comportar como um sistema de agentes independente. Isto reforça a idéia de múltiplos níveis de atividade e se adequa de forma apropriada à noção compartilhada por vários pesquisadores identificados com a síntese de sistemas inteligentes autônomos em múltiplos níveis de resolução (Albus, 1991; Meystel, 1996).

4.5 Resumo

Este capítulo apresentou uma proposta de estruturação hierárquica de redes de agentes (conforme Definição 3.46) pela incorporação da noção de superlugares. Uma parcela dos conceitos adotados teve inspiração nas Redes de Petri, mais notadamente, nos mecanismos de estruturação descritos por Lakos (Lakos, 1995b), baseando-se na definição de *páginas* de subrede como estruturas de dados abstratas. No que se refere à comparação do modelo proposto com os disponíveis na literatura pode-se constatar que as facilidades oferecidas são bastante semelhantes àquelas nas Redes de Petri Colorida Hierárquicas, mais notadamente em relação às Redes de Petri Coloridas Modulares (*Modular Coloured Petri Nets* – MCPNs) (Lakos, 1995b).

Outro ponto importante neste capítulo foi a extensão das propriedades fundamentais (restrições de escopos habilitantes e escopos gerativos) relacionadas às redes de agentes no Capítulo 3 (equações 3.18 e 3.19) para as redes de agentes modulares com o uso da noção de *fusão de lugares*.

[13] E de certa forma em Redes de Petri, conforme apresenta Lakos na sua proposta de Redes de Petri a Objeto (Lakos, 1995b).

Com a incorporação destes novos recursos ampliou-se o poder de abstração das redes de agentes, sendo possível a criação de modelos em múltiplos níveis de abstração. Além disso, os novos mecanismos de estruturação permitem a reutilização de páginas, oferecendo maior conforto e aplicabilidade na manipulação de modelos complexos.

Como ferramenta adicional de modelagem propôs-se um diagrama gráfico para a representação das relações entre páginas, chamado *diagrama de páginas*, inspirado nos diagramas de classe (Booch et al., 1997). A partir deste diagrama é possível criar diretamente a árvore de instanciação, que descreve todas as instâncias de páginas em uma rede.

No que se refere a estudos futuros, foram discutidos alguns tópicos de grande relevância para investigação posterior de redes de agentes hierárquicas, baseadas na construção de superobjetos, ou seja, objetos que podem conter toda uma página de rede. Julga-se que a base formal oferecida neste capítulo seja um passo intermediário necessário na obtenção de um modelo verdadeiramente multiresolucional.

Capítulo 5

Ambiente Computacional

Este capítulo descreve um ambiente computacional para o desenvolvimento e simulação de redes de agentes modulares, conforme as definições apresentadas no Capítulo anterior.

5.1 Introdução

Redes de objetos (ROs) e redes de agentes (RAs) constituem uma ferramenta formal para a modelagem de sistemas a eventos discretos (Guerrero, 2000), principalmente aqueles que são caracterizados por sua complexidade e sofisticação, tais como os sistemas de computação flexível (Gudwin and Gomide, 1997c) e computação com palavras (Gudwin and Gomide, 1998a), dentre outros. Devido à quase ausência de métodos formais de análise de ROs[1] os modelos têm de ser necessariamente simulados para a obtenção de conclusões. Esta situação de dificuldade de análise é, dentro de certos limites, semelhante àquela observada nas Redes de Petri (PN). PNs possuem uma extensa base de métodos formais de análise (Murata, 1989). No entanto, em modelos de complexidade elevada, comumente encontrados em aplicações práticas, a única solução viável em muitos casos é a simulação computacional direta (Murata, 1989; Gerogiannis et al., 1998).

5.1.1 Organização do Capítulo

A Seção 5.2 apresenta as motivações para o presente estudo, além de traçar um rápido histórico dos trabalhos anteriores. A Seção 5.3 apresenta o novo ambiente integrado de desenvolvimento de redes de agentes ONTOOL-2, e a Seção 5.4 introduz as noções básicas de programação de classes para o ONTOOL-2. A Seção 5.5 apresenta um exemplo de aplicação de redes de agentes na solução de problemas de gerenciamento de recursos. Ao final, na Seção 5.6, é apresentado um resumo dos principais tópicos abordados neste capítulo.

5.2 Histórico

Na fase inicial da pesquisa em ROs havia grande dificuldade na implementação efetiva dos modelos de simulação dada a inexistência de bibliotecas de software genéricas, pois todos os problemas

[1] Com exceção à análise de independência invariante introduzida por este trabalho no final do Capítulo 3, bem como os métodos, de certa forma rudimentares, introduzidos em (Gudwin, 1996).

empregavam implementações específicas, bastante particulares ao próprio modelo[2] e ao ambiente de programação. O alto nível de abstração das rede de objetos atua como um fator que pode elevar consideravelmente a complexidade das implementações.

Em (Gudwin, 1996) foi descrito um sistema de controle de um veículo autônomo, constando como o primeiro sistema efetivamente implementado em redes de objetos[3]. Este sistema apresentava uma grande quantidade de lugares e classes diferentes, bem como uma quantidade relativamente grande de recursos. Mas uma questão foi colocada: poderia o programa desenvolvido neste projeto, aliado às suas funcionalidades, ser reutilizado em outros experimentos? A reutilização de programas específicos como este levaria à necessidade de revisão de detalhes particulares do programa, como estruturas de dados usadas na representação do modelo, definição da função de seleção, linguagem de programação, detalhes específicos ao sistema operacional, hardware empregado, etc. A partir desta motivação, iniciou-se o projeto ONTOOL (Guerrero et al., 1999; Guerrero, 2000), um ambiente genérico para o desenvolvimento de redes de objetos (mais especificamente redes de agentes). Em (Guerrero, 2000) foram apresentados diversos exemplos de RAs para modelagem de sistemas evolutivos, sistemas de inferência fuzzy e sincronização de tarefas. O ambiente ONTOOL-1 representou um avanço no estudo das ROs e RAs por permitir a criação e simulação de modelos relativamente sofisticados sem a necessidade de extensa codificação e verificação. Este sistema apresentava, resumidamente, as seguintes características:

- composto por três módulos principais implementados em linguagem Java: *desktop* de edição e controle, servidor MTON (*Multi-Threaded Object Network*) e servidor de *plugs* (discutidos logo adiante). A integração entre estes módulos empregava uma pequena coleção de protocos proprietários e Java RMI (*Remote Method Invocation*);
- capacidade de geração e compilação de código fonte Java específico à implementação de cada modelo, de acordo com a especificação fornecida pelo usuário;
- linguagem textual para a especificação de redes de agentes ONSL (*Object Network Specification Language*);
- edição gráfica interativa para os componentes da rede;
- sistema de depuração incorporado;
- servidor de *plugs*[4], permitindo a conexão com objetos Java remotos via RMI; e
- biblioteca de simulação MTON (*Multi-Threaded Object Network*), com os seguintes recursos: (a) atribuição de um *thread* de execução a cada lugar da rede, (b) suporte a *classes externas*[5], (c) algoritmo de determinação de função de seleção BMSA (*Best Matching Search Algorithm*)[6] e (d) implementação das classes diretamente em linguagem Java.

[2]No que se refere ao modelo a complexidade está na determinação da função de seleção e, em menor grau, nas funções de transformação.

[3]Este sistema foi implementado em linguaguem C e compilado em um sistema operacional UNIX.

[4]*Plugs*, no contexto do sistema ONTOOL-1, correspondem a objetos Java remotos que operam para entrada e saída de dados.

[5]Discutidas adiante.

[6]Discutido na Subseção 3.6.2.

O sistema ONTOOL-1 empregava um modelo cliente/servidor para a interligação de seus três módulos (veja a Figura 5.1). Desta forma, os módulos do sistema podiam ser distribuídos em máquinas diferentes pela rede.

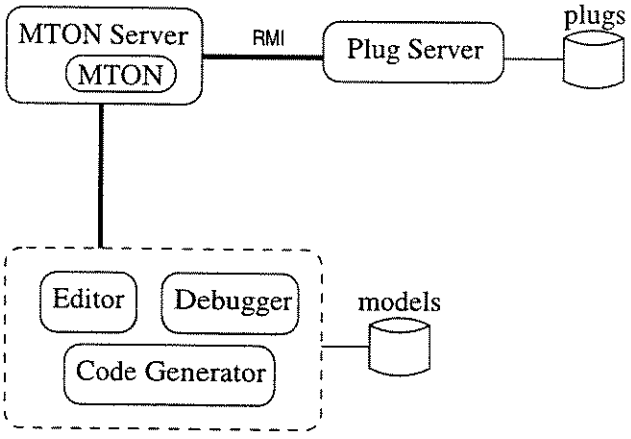


Figura 5.1: Sistema ONTOOL-1

Com a evolução natural dos estudos em redes de agentes e a tentativa de se estender os campos possíveis de aplicação, originaram-se novos requisitos para o sistema ONTOOL. A arquitetura composta em três componentes era limitada por dificultar a aplicação em alguns problemas complexos (com mais de um núcleo de simulação), dificultar o gerenciamento dos módulos (devido aos protocolos de comunicação específicos) e expansão das funcionalidades (suporte às redes de agentes modulares). Como proposta para preencher estes requisitos, este trabalho introduz uma nova versão do ambiente de desenvolvimento de redes de agentes, buscando avanços em diversos aspectos, desde a interface gráfica até o suporte a rede.

5.3 O Ambiente ONTOOL-2

O sistema ONTOOL-2 introduz uma série de novas características. Entre elas pode-se enumerar:

- Interface gráfica redesenhada, implementada sob o paradigma *model-view-controller*. Permite a edição visual e simultânea de vários componentes, inclusive páginas de subrede. Proporciona uma melhor manuseabilidade ao usuário (vide Figura 5.2).

  As motivações para os aprimoramentos na interface gráfica desta versão se justificam por facilitar a manipulação dos modelos pelo usuário. As mais importantes são a capacidade de edição de propriedades de múltiplos elementos simultaneamente, como arcos, lugares, superlugares, código da implementação da classe, etc. Adicionalmente, inclui-se um editor de classes no formato gráfico, de acordo com a representação de agente formal estudada no Capítulo 3 (Figura 5.3). Outra característica importante é a adição de mecanismos de ajuda sensíveis ao contexto (Figura 5.4).

- Gerador de código com suporte automático à geração de adaptadores (discutidos mais adiante)

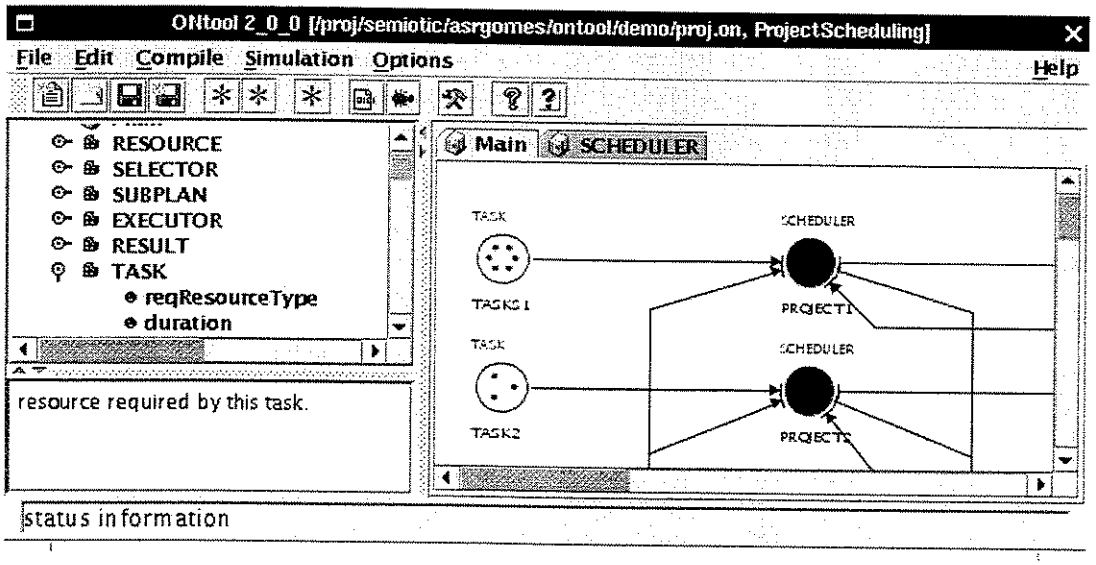
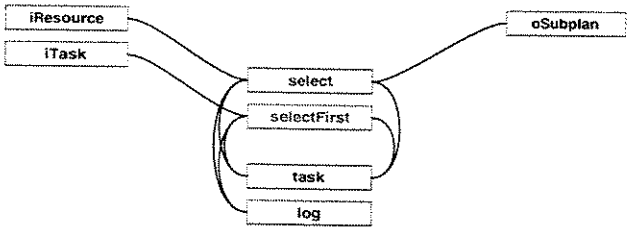
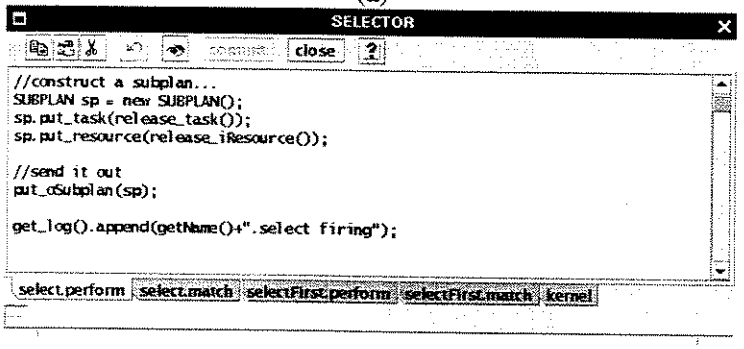


Figura 5.2: Janela principal do sistema ONTOOL2, onde podem ser identificadas 3 partes principais. Na parte superior-esquerda localiza o *navegador*, na inferior-esquerda a área de edição de *documentação sensível ao contexto* e na direita a área de edição da *topologia das páginas*.



(a)



(b)

Figura 5.3: Subsistema de edição de classes. A figura (a) mostra a visualização gráfica de uma classe, onde podem ser identificados os sensores (`iResource` e `iTask`), atuador (`oSubplan`), estados internos (`task` e `log`) e funções (`select` e `selectFirst`). A figura (b) mostra a janela de edição do código Java da implementação da função `select`.

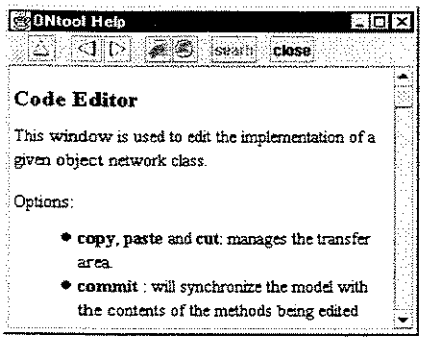


Figura 5.4: Janela de ajuda sensível ao contexto, neste caso relativa à janela de edição de código.

para classes externas via introspecção[7], bem como um submódulo para geração automática de documentação em formato HTML[8]. Além disso, suporta de forma transparente a importação de interfaces IDL (*Interface Definition Language*) previamente compiladas para Java.

A Figura 5.5 mostra, de forma simplificada, o ciclo de desenvolvimento típico dentro do ambiente ONTOOL. Inicialmente, o usuário elabora o modelo computacional a partir de um modelo conceitual prévio, sendo depois compilado. Neste estágio o código que efetivamente implementa o modelo é gerado na forma de uma coleção de classes Java específicas ao modelo fornecido e à biblioteca de simulação MTON. Uma característica importante nesta arquitetura é a existência de dois tipos de classes:

- *Classes internas.* Uma classe interna é a versão computacional da classe descrita na Definição 3.26, contendo sensores, atuadores, funções de transformação, etc.
- *Classes externas.* Uma classe externa, por outro lado, representa uma classe Java externa fornecida pelo usuário como, por exemplo, uma classe para cálculos matriciais. Este tipo de classe é suportado pela criação de adaptadores (figuras 5.6 e 5.7), classes especiais, geradas automaticamente, responsáveis por ajustar a classe externa aos requisitos do simulador. O gerador de código suporta ainda dois subtipos de classes externas. Aquelas chamadas *locais*, ou seja, uma implementação típica de uma classe Java compilada em *bytecode* na forma de um arquivo com extensão `.class`. O outro tipo corresponde a uma interface Java resultante da compilação prévia de uma interface IDL. Objetos de classes externas remotas podem ser, desta forma, conectados de forma transparente ao núcleo de simulação.

A existência desta diferenciação pode ser justificada de duas formas: (a) possibilidade de reuso de bibliotecas previamente disponíveis dentro do ambiente de simulação e (b) exigência de se encerrar a recursividade na estrutura dos objetos das classes internas[9].

[7]Processo de inspeção da estrutura de uma classe pela aplicação do recurso de *reflexão* Java. O intuito é o de determinar seus parâmetros (membros de dados, métodos, interfaces, etc.).

[8]Este recurso foi inspirado no programa gerador de documentação `javadoc` desenvolvido pela Sun Microsystems para a documentação de classes Java.

[9]Conforme comentado na Subseção 3.3.1 (pág. 27), cada objeto possui um conjunto de atributos ou partes. Mas, para finalizar a recursão, em algum ponto da cadeia, as partes devem ser somente atributos simples.

O ambiente, em seu estágio atual, atua apenas na parte específica da implementação do modelo, sendo de pouca ajuda no aprimoramento do modelo conceitual em si.

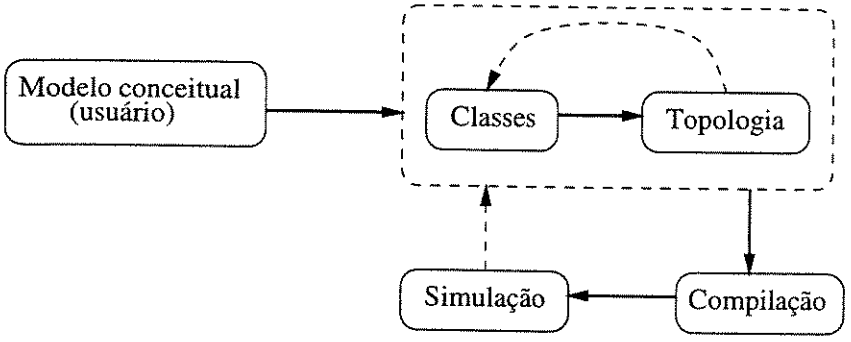


Figura 5.5: Ciclo de desenvolvimento de modelos no ambiente ONTOOL-2.

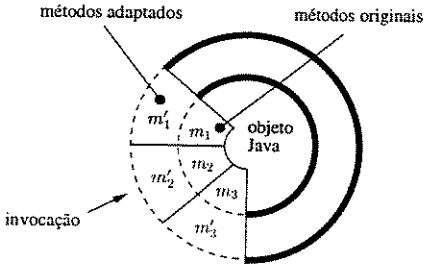


Figura 5.6: Visão conceitual de um adaptor de classe.

- Implementação dos módulos do sistema como objetos CORBA[10], permitindo uma maior flexibilidade em futuras implementações.

  Como descreve Saleh (Saleh et al., 1999), a adoção de tecnologias orientadas a objeto é uma tendência que tem se seguido na indústria de software, enfatizando a importância da reusabilidade, manutenção e flexibilidade no processo de desenvolvimento de software, elevando a qualidade e diminuindo os custos. A integração da arquitetura CORBA com linguagens orientadas a objeto, como Java, facilita a introdução de ambientes distribuídos independentes de plataforma. Tipicamente, aplicações distribuídas são divididas em três camadas: (a) nível de apresentação (GUI), (b) funcionalidade & conectividade e (c) nível de dados. Os componentes contrutivos destas aplicações possuem uma nítida separação entre *interface* e *implementação*.

  Toda a organização dos módulos principais que compõem o sistema foi também modificada para refletir a adoção do paradigma de objetos baseado na arquitetura CORBA. O novo modelo pode ser visto na Figura 5.8. Objetos externos remotos podem ser utilizados como mecanismos de entrada e saída de dados. Considere, por exemplo, o cenário mostrado na figura citada anteriormente. O objeto externo remoto em **W2** pode representar um sistema de captura de dados

[10] O Apêndice B apresenta uma introdução rápida aos conceitos fundamentais.

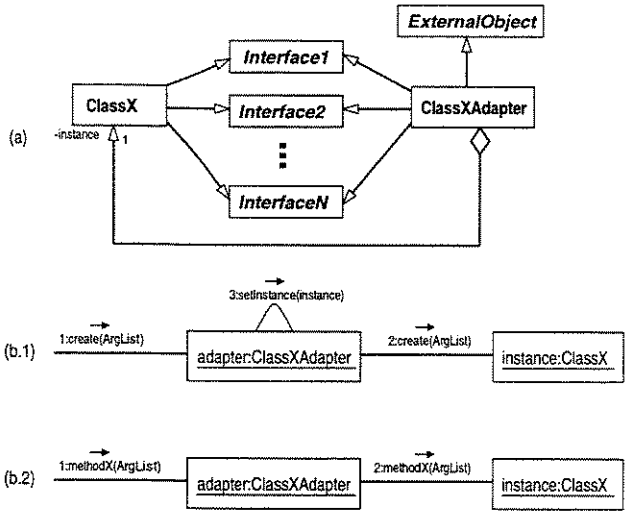


Figura 5.7: Diagramas UML indicando a construção e funcionamento de um adaptador: (a) diagrama de classes, onde a classe Java externa `ClassX` – que implementa as interfaces `Interface1`, `Interface2`, ..., `InterfaceN` – tem como adaptador a classe `ClassXAdapter`, (b.1) diagrama de colaboração para o construtor de um adapter e (b.2) repasse da invocação de um método pelo adaptador (comportamento proxy).

de telemetria (como temperatura, pressão, etc.) e o objeto em **W3** uma estação de visualização para os dados obtidos a partir da simulação da rede de agentes sendo processada em **W1**.

- Suporte a edição e simulação de RAMs. A plataforma de simulação possui suporte a simulações passo-a-passo e/ou ininterruptas[11], que não existia no ONTOOL-1 (de forma similar a um *daemon* UNIX);

- Compatibilidade total com a licensa GPL[12] *(GNU Public Licence)* (vide Figura 5.9).

5.4 Programação ONTOOL-2 Básica

Esta seção tem como objetivo oferecer algumas noções básicas sobre o mapeamento do modelo de agente formal para a implementação computacional, destacando os detalhes mais relevantes[13].

Inicialmente, convém ressaltar que o modelo computacional de agente formal adotado no ambiente ONTOOL-2 deriva diretamente da biblioteca de simulação MTON, amplamente descrita em (Guerrero et al., 1999) e mais profundamente em (Guerrero, 2000). A biblioteca MTON, em sua versão atual, suporta todo o formalismo presente no modelo de agente formal e rede de agentes descritos neste trabalho. Assim, para o leitor interessado numa abordagem mais detalhada, sugere-se uma

[11] Muito mais adequado a sistemas embutidos.

[12] Para maiores informações consulte a *homepage* `http://www.gnu.org`

[13] Assim, esta seção não pode ser caracterizada como um tutorial.

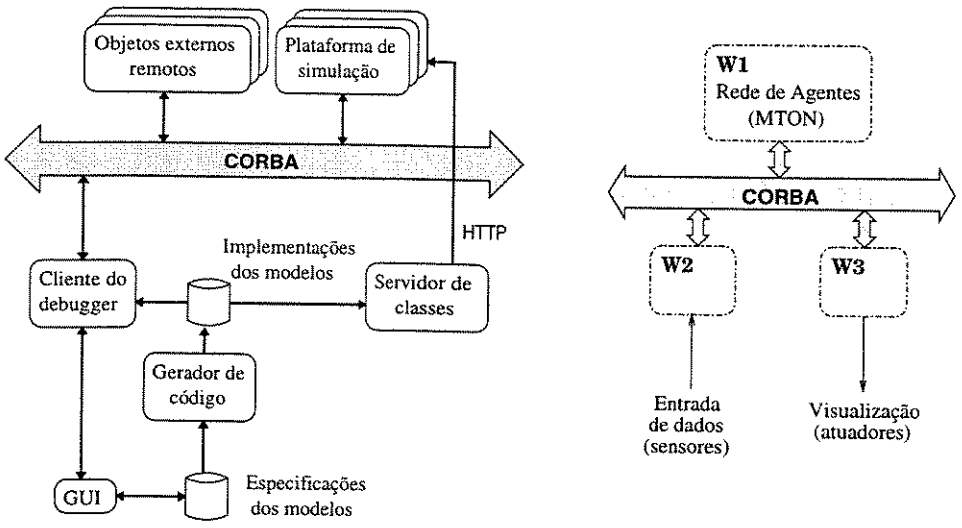


Figura 5.8: Componentes da plataforma ONTOOL-2 (equerda)) e cenário típico de aplicação (direita), onde a RAM sendo executada na estação **W1** tem dois objetos externos remotos em **W2** e **W3**, para entrada e saída de dados, respectivamente.

consulta preliminar ao Capítulo 4 de (Guerrero, 2000), visto que quase todas as funcionalidades oferecidas pela biblioteca MTON são ainda suportadas pela nova versão. As diferenças mais marcantes se referem à facilidade de geração automática de adaptadores para classes externas sem a necessidade de intervenção do usuário, pequenas otimizações e suporte a classes externas remotas (o mecanismo de *plugs* foi descontinuado com a incorporação de objetos CORBA).

5.4.1 Classes Internas

Conforme comentado anteriormente neste capítulo, O ONTOOL reconhece dois tipos distintos de classes, as classes internas e as classes externas. Uma classe interna no ONTOOL é formada por um conjunto elementos descritos como:

a) **estados internos** – armazenam o estado do agente;

b) **sensores** – permitem a entrada de informações a partir do ambiente do agente;

c) **atuadores** – permitem que o agente atue no meio, através da liberação de objetos;

d) **funções de transformação** – representa a componente ativa do agente, podendo obter dados dos sensores e alterar os estados internos, bem como atuar no meio pelos atuadores. Cada uma destas funções é implementada por duas subfunções: (1) uma função de transformação propriamente dita (Definição 3.23, página 32) e (2) uma função de avaliação (Definição 3.36, página 39); e, finalmente,

e) **uma função de inicialização (sempre uma para cada classe)** – especifica o código de inicialização dos objetos (ativos e passivos) no instante inicial da rede (Definição 3.47, página 48).

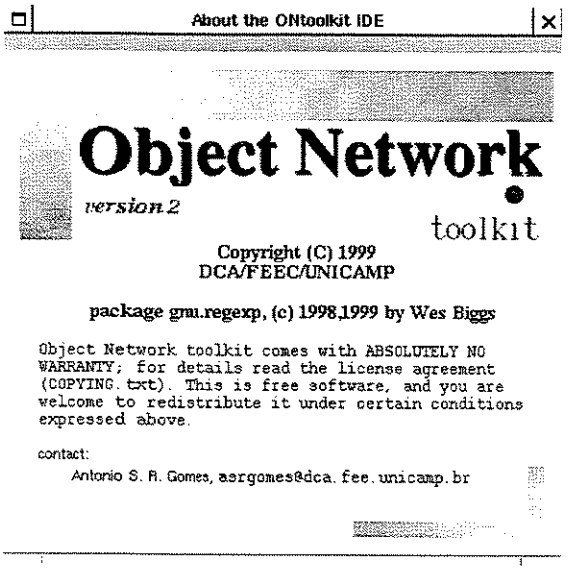


Figura 5.9: Janela de identificação do sistema, ressaltando a licença GPL (*GNU Public License*).

Esta função contém o código que estabelece o estado inicial dos objetos definidos no núcleo da rede, ou seja, o conteúdo dos estados internos dos objetos de $\mathcal{C}^0$.

Os códigos das funções de transformação (as duas subpartes) e da função de inicialização são escritos em linguagem Java, a partir do próprio ambiente de edição de redes de agentes no ONTOOL-2[14]. Este código pode, de acordo com as convenções dadas a seguir, acessar de forma ordenada os sensores, atuadores e estados internos do agente. Mas antes desta explicação é necessário observar de forma atenta o ciclo de vida básico de um agente formal, conforme mostra a Figura 5.10. Após a criação, o agente entra no laço percepção-atuação, onde no estado de *matching* cada uma das componentes de avaliação do agente é executada para determinação do *grau de utilidade* de cada uma das possíveis combinações de objetos percpetíveis através dos sensores[15]. Após esta etapa, executa-se o algoritmo BMSA[16] para a solução dos conflitos de interesses, de forma que, ao final, os agentes vencedores entram no estado de execução (*perform*). Note que um agente pode executar no máximo uma função de transformação ao mesmo tempo. Durante a etapa de avaliação o agente não pode alterar nenhum de seus estados internos nem objetos acessíveis pelos sensores, visto que múltiplos agentes podem estar realizando acessos concorrentemente[17]. Já na fase de execução o agente pode, de acordo com as condições impostas para o disparo[18], alterar seu estado.

[14] Veja a Figura 5.3(b), como exemplo do editor de código.

[15] A idéia descrita aqui é explicada graficamente na Subseção 3.6.1, Figura 3.17.

[16] A versão atual da biblioteca MTON segue o padrão descrito na Seção 3.6.2.

[17] Esta restrição é, em alguns casos, verificada automaticamente pela biblioteca MTON. No entanto, esta propriedade não pode se garantida para classes externas, visto que seu código – implementado pelo usuário – pode a adotar, a princípio, políticas onde seja possível alterar o estado mesmo durante a fase de *matching*. Como critério de projeto, deve-se evitar tais construções.

[18] Dependendo do modo de acesso, conforme apresentado na Definição 3.35, página 38.

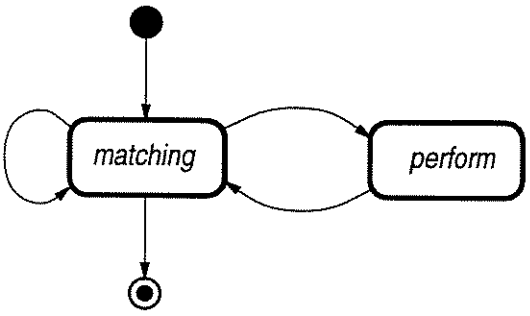


Figura 5.10: Estados possíveis de um agente formal

O objetivo do código de avaliação é fornecer propostas de ações (conforme Definição 3.48, página 50) para o módulo BMSA, ou seja, um conjunto de objetos e seus respectivos modos de acesso, bem como um valor numérico real que expresse o grau de utilidade das interações propostas. Para tanto, o código deve retornar uma instância da classe Java `MatchResult`, que contém dois métodos principais:

- `public void set(float utility)`
  Define o valor de utilidade para a ação sendo proposta. Caso não seja especificada, o módulo BMSA considera que a ação deve ser descartada (a ação não deve ser executada).
- `public void setAccessMode(NetObject obj, AccessMode am)`
  Define o tipo de interação sobre o objeto `obj`. Os modos de acesso possíveis para `am` são dados por:
  - `AccessMode.CONSUME`: `obj` deve ser consumido (de forma exclusiva).
  - `AccessMode.SHARED`: `obj` deve ser acessado de forma compartilhada.
  - `AccessMode.EXCLUSIVE`: `obj` deve ser acessado de forma exclusiva.

  `obj` é uma referência a um objeto perceptível por um sensor do agente. Pode ser computada dinamicamente pelo método `get_NomeDoSensor` (definido automaticamente pelo gerador de código).

Um detalhe importante é que o código de avaliação deve ser mantido sempre o mais simples possível (no que se refere ao esforço computacional), visto que ele é automaticamente processado para todas as combinações possíveis de escopo de visibilidade da função (conforme Definição 3.34, página 37). Um exemplo simplificado de uma função de avaliação para a função `move` da classe `TRAN` (Figura 5.11) com acesso a um sensor chamado `iFicha` pode ser visto no Código 5.4.1.

Neste caso a definição da utilidade foi mantida a mais simples possível. Mas note que a mesma pode ser computada levando-se em conta quaisquer parâmetros disponíveis durante a execução do trecho de código.

Para a parte de execução (ou *perform*) da função existem outras operações primitivas possíveis além da já citada `get_Nome`. Todas elas são geradas automaticamente pelo gerador de código, de forma que o usuário não precisa tratar de nenhuma conversão de tipo. De forma geral, estas primitivas podem ser descritas como:

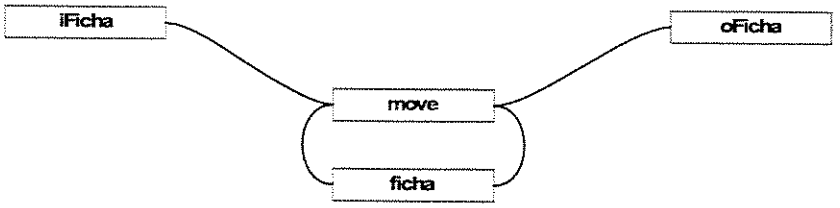


Figura 5.11: Classe TRAN.

```
1  \\cria instância (sempre da mesma forma)
2  MatchResult m = new MatchResult();
3
4  \\utilidade fixa 1.5 (mas pode ser qualquer valor!)
5  m.set(1.5);
6
7  \\usa referência ao objeto no sensor 'iFicha'
8  \\objeto deve ser consumido
9  m.setAccessMode(get_iFicha(), AccessMode.CONSUME);
10
11 \\retorna instância (sempre da mesma forma)
12 return m;
```

Código 5.4.1: Código para definição da função de avaliação.

- `public ClasseON get_SensorOuEstado()`
  obtém uma referência ao sensor ou estado indicado.

- `public ClasseON release_SensorOuEstado()`
  desconecta (ou remove) o objeto contido no estado ou sensor indicado e obtém um referência. Note que, se esta referência for perdida (ou intencionalmente descartada) o objeto será automaticamente removido pelo *Garbage Collector* Java. Para que seja executada sobre um sensor exige-se que a função tenha especificado um modo de acesso `AccessMode.CONSUME` durante a fase de geração de propostas.

- `public ClasseON clone_SensorOuEstado()`
  cria uma cópia (se o objeto permitir) e obtém uma referência para esta cópia.

- `public void put_EstadoOuAtuador(ClasseON obj)`
  conecta (ou insere) um objeto em um estado interno (que deve estar vazio) ou em um atuador (sendo, desta forma, inserido em um lugar da rede). Para tanto a função deve apresentar, em sua especificação, permissão de escrita no estado ou atuador desejado. Importante frisar que ou um objeto está livre ou está vinculado a um (e no máximo um) sensor/estado/atuador. Neste caso, não faz sentido aplicar a primitiva `put` em dois campos/atuadores simultaneamente (condição inválida), visto que um objeto pode estar conectado a no máximo um campo interno do agente.

A listagem do Código 5.4.2 ilustra o exemplo de uma função que move o objeto ficha do sensor para o atuador e sempre armazena uma cópia.

```
1  \\limpa o campo 'ficha', senão teremos um erro!
2  if (get_ficha() != null)
3      release_ficha();
4
5  \\armazena uma cópia do objeto selecionado pelo
6  \\sensor 'iFicha' no estado interno 'ficha'
7  put_ficha(clone_iFicha())
8
9  \\remove o objeto selecionado pelo sensor 'iFicha' e
10 \\o reinsere na rede através do atuador 'oFicha'
11 put_oFicha(release_oFicha());
```

Código 5.4.2: Implementação da parte *perform* para a função `move`.

5.4.2 Classes Externas

Classes externas correspondem a classes Java fornecidas pelo usuário e manipuladas como objetos passivos dentro do mecanismo de simulação do MTON. Quaisquer classes Java podem ser importadas pelo gerador de código via geração de adaptadores[19]. Como exemplo, considere a classe Java para geração de números aleatórios `java.util.Random`, importada pelo simulador com um apelido `RND`. Neste caso, o gerador de código criará um classe chamada `RND.Adapter` como uma subclasse da classe original, incluindo todos os respectivos construtores. Além disso, esta classe criada possui suporte específico à biblioteca MTON. Uma instância do gerador de números aleatórios pode ser então criada da forma:

```
RND.Adapter rnd = new RND.Adapter();
```

Adicionalmente, é permitido repassar uma interface Java derivada de um compilador IDL para o gerador de código. Neste caso, será criado um adaptador especial para o suporte adequado à instância remota. Como exemplo, considere uma interface IDL chamada `ext.Log`. Após a compilação (com a geração dos *stubs* e *skeletons*) gera-se uma interface Java `ext.Log`. Esta interface é então fornecida ao gerador de código com o apelido `LOG`. Supondo a existência de uma implementação desta interface, chamada `ontool.ext.LogImpl`, uma instância pode ser criada a partir do código das funções usando-se a seguinte construção:

```
LOG.Adapter log = LOG.createInstance(``ontool.ext.LogImpl'');
```

Note que as variáveis `rnd` e `log` podem ser manipuladas diretamente pelas operações primitivas apresentadas na subseção anterior.

5.5 Exemplo

Esta seção apresenta um exemplo de aplicação que emprega o ambiente ONTOOL-2 na criação de um modelo baseado em redes de agentes modulares.

[19] mesmo classes Java declaradas com o modificador `final`.

O exemplo tratado pode ser classificado como um problema clássico de agendamento de tarefas. Existem inúmeros métodos de solução para problemas deste tipo na literatura (Zweben and Fox, 1994), cada qual com vantagens e desvantagens específicas, de acordo com os detalhes do problema em si (estrutura dos recursos, restrições do domínio, etc.). Na abordagem apresentada optou-se por manter o modelo o mais simples possível, com a finalidade exclusiva de demonstrar dois pontos principais: (a) a capacidade de modularização e (b) o processo de competição entre os agentes baseado no algoritmo BMSA. Note que muitos recursos disponíveis não foram empregados, como mobilidade, capacidade de criação e destruição de agentes durante a execução, adaptabilidade (Gudwin, 1996), etc.

O problema pode ser descrito da seguinte forma: considere um conjunto de projetos $\{Pr_1, Pr_2, \ldots, Pr_n\}$ e um conjunto de recursos $\{r_1, r_2, \ldots, r_m\}$. Cada Pr_i contém uma sequência de tarefas $t_1, t_2, \ldots, t_{p_i}$ interdependentes, de forma que t_{i+1} só pode ser executada depois de t_i. Cada t_i (Figura 5.15) apresenta quatro componentes básicos: (1) sequência (ordem cronológica dentro do mesmo projeto), (2) tipo de recurso (Figura 5.16) requerido, (3) prioridade e (4) duração de tempo para a execução da tarefa. Admite-se ainda que os recursos podem ser reutilizados.

Na implementação realizada no ONTOOL construiu-se um modelo com 5 projetos, cada qual com um conjunto de tarefas distintas. Todos os projetos competem pelo mesmo conjunto de recursos. A rede possui duas páginas (veja Figura 5.12, 5.13 e Figura 5.14). O processo de ordenação é feito pela página `SCHEDULER`, mais especificamente, pelo agente da classe `SELECTOR` (Figura 5.17). Esta classe possui duas funções de transformação. A primeira escolhe a próxima tarefa do projeto que deve ser executada (de acordo com sua sequência) e a segunda compete para alocar o recurso requerido para a execução da tarefa escolhida pela primeira e gerar um subplano (Figura 5.18). Após este passo, a tarefa é executada (Figura 5.19) gerando um resultado (Figura 5.20).

O processo de escolha das ações a serem tomadas é, de certa forma, simples. Basicamente, a parte do modelo que seleciona as opções está presente na função `select`, definida na classe `SELECTOR`. Recordando as definições da Capítulo 3, agentes possuem funções de transformação e de avaliação. O mesmo ocorre com esta função. A listagem do Código 5.5.1 é a implementação da parte de avaliação da função `select`. Seu papel é gerar propostas de ações, constando os objetos (o produto cartesiano do conjunto de recursos com o conjunto de tarefas) e valor de utilidade.

Caso uma das ações propostas se efetive, a função de transformação do respectivo agente é então aplicada. Neste caso, o recurso e a tarefa são usados na construção de um subplano, que depois é executado pelo agente no lugar `P3` (Código 5.5.2).

Os dados da simulação são representados pela lista de tarefas de cada projeto e dos recursos disponíveis. A Figura 5.21 mostra o estado final de simulação conforme obtido pelo depurador. Note que cada recurso é representado por uma cor diferente, sendo usado no máximo por uma tarefa ao mesmo tempo.

Pontos interessantes que podem ser observados a partir do exemplo:

- A rede de agentes modular apresentada pode oferecer dois níveis de abstração distintos. O primeiro no que se refere à visão abrangente dos projetos e a segunda em relação ao processo interno de cada projeto. É importante salientar que desta forma, a modularização, representada pela página `SCHEDULER`, permite que o projetista modifique toda a estrutura interna da página sem que se mude a página principal.

- No exemplo tratado a preocupação principal foi mostrar que RAMs podem ser adequadamente empregadas como ferramentas de modelagem com grande apelo visual e poder computacional.

```
1  MatchResult match = new MatchResult();
2
3  TASK task = get_task();
4  RESOURCE ires = get_iResource();
5
6  \\se a próxima tarefa já foi escolhida...
7  if (task != null){
8      \\se o recurso necessário está disponível...
9      if (ires.get_type().equals(task.get_reqResourceType())){
10         \\a utilidade é então dada pela prioridade
11         match.set((double)task.get_prio().getValue());
12         match.setAccessMode(ires, AccessMode.ASSIMILATION);
13     }
14 }
15 return match;
```

Código 5.5.1: Implementação da subfunção de avaliação (*matching*) para a função `select` da classe SELECTOR.

```
1  \\cria um novo subplano
2  SUBPLAN sp = new SUBPLAN();
3
4  \\define o estado interno do subplano
5  sp.put_task(release_task());
6  sp.put_resource(release_iResource());
7
8  \\libera o subplano para execução
9  put_oSubplan(sp);
```

Código 5.5.2: Implementação da subfunção de execução (*perform*) da função `select` da classe SELECTOR.

Não houve a preocupação com a otimalidade da solução, dado que estudos futuros podem ser aplicados na elaboração de uma nova estrutura mais eficiente para a página SCHEDULER.

5.6 Resumo

Este capítulo apresentou um ambiente computacional para o desenvolvimento de redes de agentes baseado no projeto ONTOOL (Guerrero et al., 1999; Guerrero, 2000). Este sistema foi desenvolvido para suportar novos requisitos, como escalabilidade, interface gráfica com usuário mais amigável e produtiva, além de suporte ao novo modelo de redes de agentes modulares.

Ao final, apresentou-se um exemplo de aplicação de redes de agentes modulares dentro do ambiente ONTOOL-2 para a solução de uma problema de agendamento.

Para os trabalhos futuros, os seguintes pontos podem ser sugeridos:

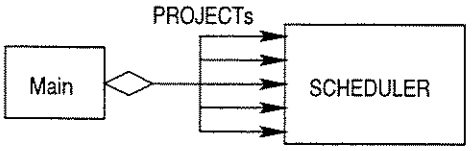


Figura 5.12: Diagrama de páginas para o problema de agendamento.

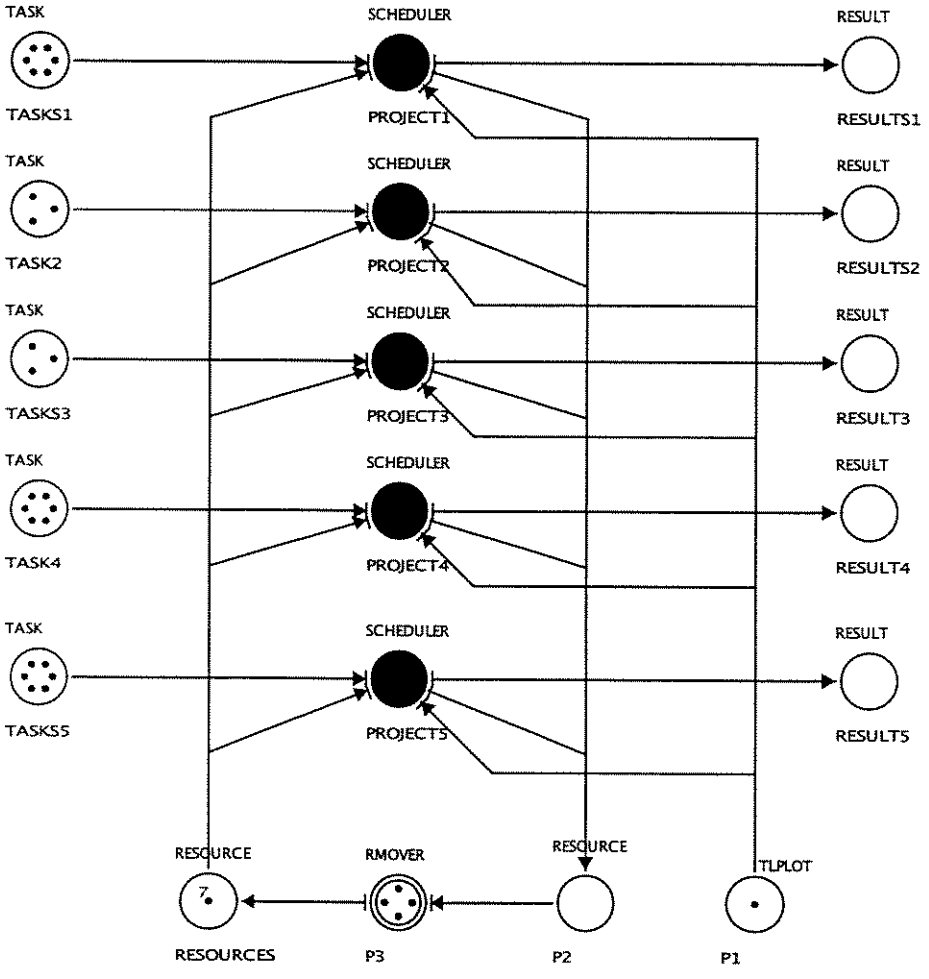


Figura 5.13: Página-raiz `Main`.

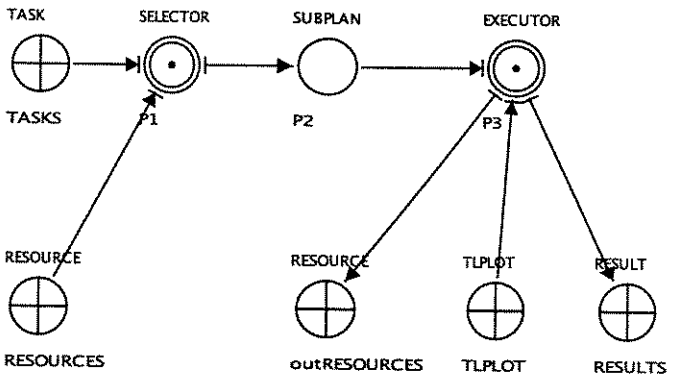


Figura 5.14: Página SCHEDULER.

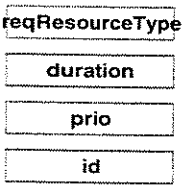


Figura 5.15: Classe TASK.

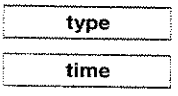


Figura 5.16: Classe RESOURCE.

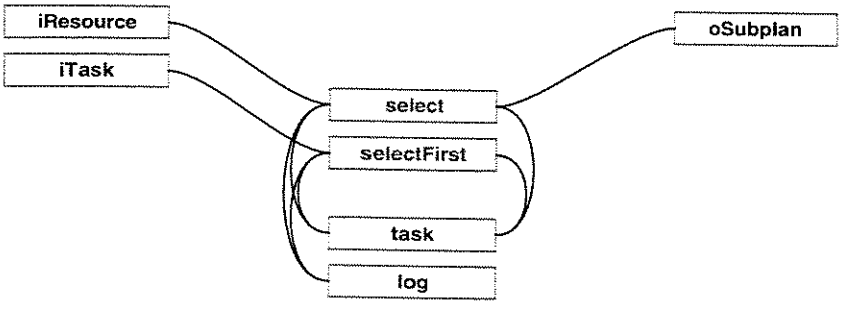


Figura 5.17: Classe SELECTOR.

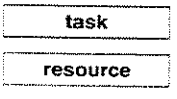


Figura 5.18: Classe SUBPLAN.

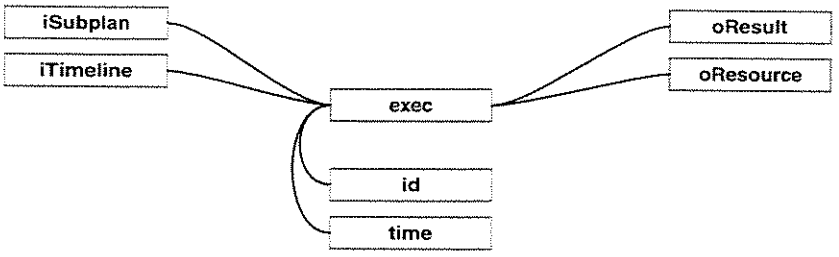


Figura 5.19: Classe EXECUTOR.

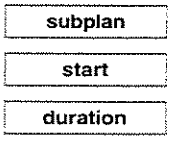


Figura 5.20: Classe RESULT.

- suporte à modelagem conceitual (pelo menos um certo nível) em redes de agentes como uma extensão ao diagrama de páginas introduzido no capítulo anterior, além da adaptação de outros conceitos de modelagem visual, em especial aqueles oferecidos pela linguagem UML (Booch et al., 1997);
- estudo mais elaborado sobre a arquitetura CORBA, enfatizando o uso dos serviços disponíveis (mais notadamente os serviços de eventos, persistência, segurança e *trader*) na construção de uma arquitetura de serviços para a simulação de redes de agentes de forma completamente distribuída, isto é, cada lugar da rede podendo estar localizado em uma máquina diferente;
- implementação de novos paradigmas de interface com usuário no ONTOOL (Janecek et al., 1999) com atenção aos aspectos semióticos desta interação (de Souza et al., 1999); e
- realização de uma avaliação do desempenho na execução da simulação da rede de agentes.

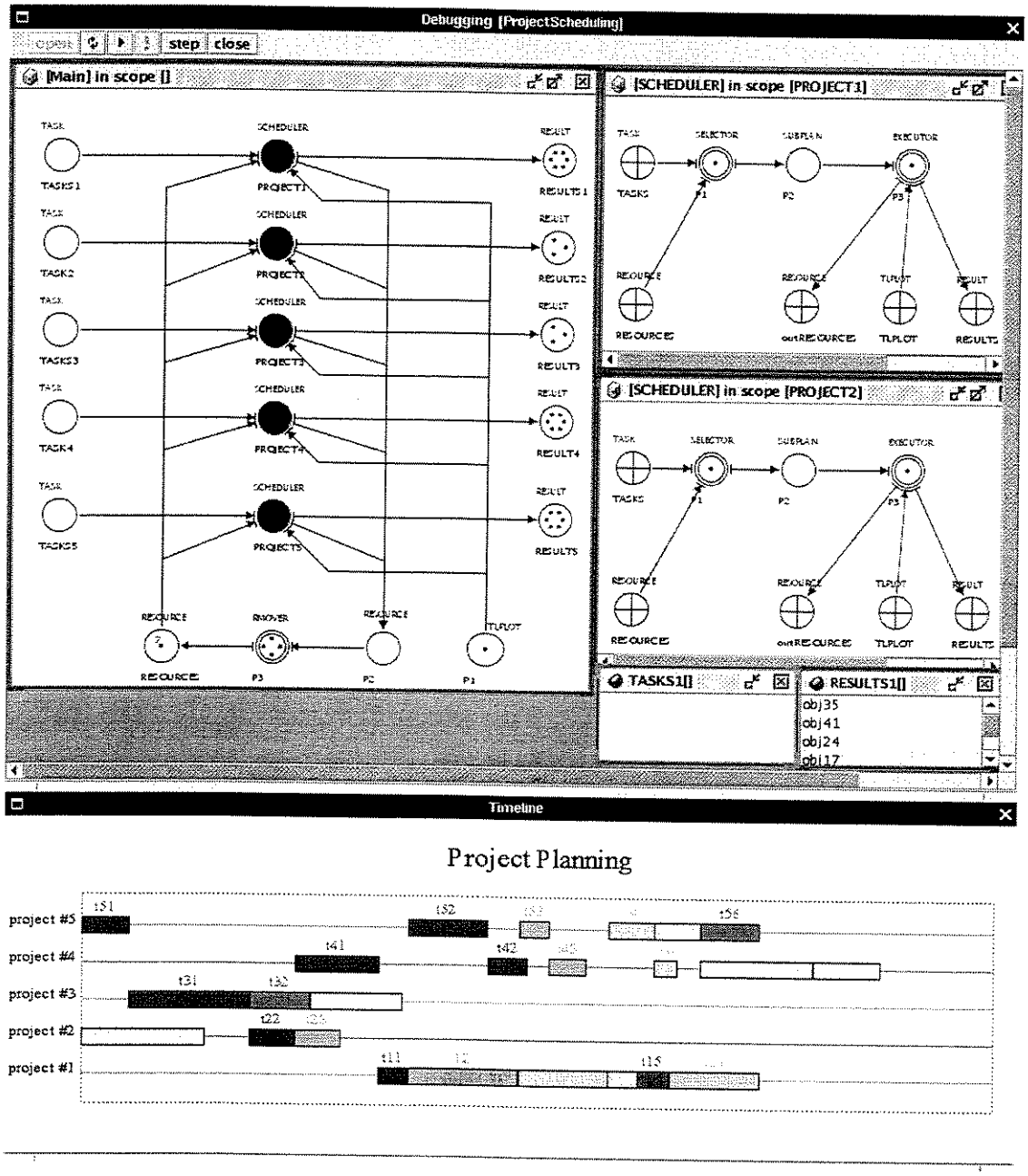


Figura 5.21: Na parte de cima a aparece a tela de depuração, onde podem ser vistas as instâncias de páginas. Na parte de baixo está o objeto externo remoto para visualização.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho apresentou-se uma série de contribuições ao estudo formal e ao uso computacional de redes de agentes. Dentre as mais significativas, podemos destacar:

- Reformulação do modelo de redes de agentes introduzido em (Guerrero, 2000) com o intuito de incorporar definitivamente a noção de agência ao próprio modelo formal. Este esforço agrega a definição formal de pontos até então introduzidos somente de maneira informal (principalmente no que se refere à especificação da topologia de uma rede de agentes);

- Desenvolvimento de um *framework* formal (na verdade, uma proposta inicial) para o estudo da dinâmica de funcionamento das redes de agentes em termos do algoritmo BMSA (Guerrero et al., 1999; Guerrero, 2000), além da identificação de uma propriedade invariante com grande potencial de utilização na paralelização do processo de busca do algoritmo BMSA;

- Extensão do modelo formal de redes de agentes para a modelagem de sistemas com múltiplos níveis hierárquicos, com a introdução de técnicas de estruturação hierárquica baseadas na noção de superlugares e uma coleção de propostas (ainda de caráter informal) para a criação de redes de agentes com superobjetos;

- Implementação de um ambiente computacional independente de plataforma para o desenvolvimento de modelos em redes de agentes modulares.

A seguir, serão discutidas cada uma das contribuições sob uma perspectiva mais ampla, enfatizando ainda os possíveis trabalhos que podem ser realizados em investigações futuras.

A reformulação do modelo proposto por Guerrero (Guerrero, 2000) permite uma especificação mais completa do funcionamento interno do agente, delimitando de forma clara as noções de sensores, atuadores e estados internos. Esta reformulação se mostra determinante, principalmente quanto à especificação formal para os elementos que descrevem a topologia da rede, sendo essencial para a formulação do modelo de redes de agentes modulares. Ao mesmo tempo, detecta-se a necessidade de um estudo mais aprofundado das relações entre este modelo com o de Redes de Petri (principalmente em relação a seus derivados hieráquicos, orientados a objetos, etc.). Imagina-se que este estudo é de cabal importância, pois, ao mesmo tempo que estes modelos comungam de diversas similaridades, também apresentam algumas diferenças conceituais bem demarcadas. Uma comparação analítica entre o modelo de redes de agentes com modelos de redes de Petri estendidas (principalmente as orientadas a objeto), poderá auxiliar na determinação dos aperfeiçoamentos futuros a serem incorporados.

Acredita-se que ambos modelos têm um campo de possibilidades extremamente vasto (para ser descrito aqui) e a noção atual de redes de objetos (e agentes) tem com certeza muito a ganhar com esta interação. Neste mesmo contexto, sugere-se como outra abordagem a idéia de *atores* (Agha, 1997; Agha and Kim, 1999)

A proposta de um *framework* lógico-formal para o estudo da dinâmica de funcionamento de uma rede de agentes, permite um estudo mais aprofundado do funcionamento do algoritmo BMSA, quando aplicado a uma classe de problemas de satisfação de restrições, um tipo de problema bastante estudado na literatura (vide Apêndice A). Como resultado importante desta proposta identifica-se a possibilidade de introdução de métodos formais de análise do funcionamento dinâmico da rede, métodos estes praticamente inexistentes no domínio das redes de agentes. No futuro, pode-se vislumbrar a possibilidade de substituir-se o algoritmo BMSA por outros tipos de algoritmos, generalizando o mecanismo automático de determinação das funções de seleção. Neste caso, poderia-se especular sobre a utilização de outras técnicas de coordenação de sistemas multiagentes, como o ContractNet (Smith, 1980), por exemplo, regulando a dinâmica das redes de agentes. Outro ponto de grande importância na evolução do modelo de redes de agentes compreende o tratamento da dinâmica do tempo contínuo, ou seja, definido não em $\mathbf{N}$, mas em $\mathbf{R}$. Isto poderia permitir a incorporação da noção de tempo-real (Selic et al., 1994), possibilitando a aplicação deste modelo em uma gama maior de problemas com este tipo de restrição.

As técnicas de estruturação hierárquica em redes de agentes que foram propostas no Capítulo 4 elevam consideravelmente o conforto na representação de sistemas heterogêneos e complexos, por permitirem o tratamento do problema em múltiplos níveis de abstração. Além disso, oferecem a capacidade de reutilização de componentes da rede, por meio da criação de bibliotecas. Desta forma, não somente pode-se alterar componentes de uma rede de agentes, como re-aproveitá-los em futuras redes. No que se refere à construção de redes de agentes baseadas em superobjetos, restam ainda diversas questões a serem tratadas, como o sincronismo (ou assincronismo) na comunicação entre os componentes das estruturas envolvidas.

De forma análoga ao estudo em Redes de Petri Coloridas, a existência de ferramentas computacionais específicas desempenha um papel fundamental na validação dos modelos. Neste sentido, o ambiente computacional de desenvolvimento de redes de agentes ONTOOL-2 oferece um campo de testes ao pesquisador, sem que este necessite tratar explicitamente todos os detalhes das implementações. Os novos recursos apresentados pelo ONTOOL-2 compreendem um avanço em relação à sua versão anterior em vários pontos, mas de forma mais importante na interação com o usuário, simulação de redes de agentes modulares e integração (ainda bastante inicial) com a tecnologia de objetos distribuídos CORBA. Pelo seu caráter de protótipo, faltam estudos mais aprofundados em diversos aspectos, destacando-se o estabelecimento de uma arquitetura distribuída solidamente fundamentada nos recursos e serviços oferecidos pelo CORBA, análise de desempenho e integração com ferramentas para a modelagem conceitual, tais como os diagramas UML (Booch et al., 1997) de *use cases*, por exemplo. Diversas outras modificações podem ser propostas, por exemplo, no sentido de tornar a interface com o usuário mais amigável. Na versão atual, a definição das classes precede a definição da topologia. Entretanto, já detectou-se que muitas vezes seria mais adequado dar liberdade ao usuário na definição da topologia da rede, adaptando automaticamente a definição das classes de modo a torná-las compatíveis com a topologia, ao contrário do que é feito hoje, quando obriga-se o usuário a efetuar alterações nas classes de modo a permitir alterações na topologia. Essa "inversão" se deve principalmente a questões envolvendo como o ser humano efetua um "design", ou seja, o processo criativo pelo

qual uma rede de agentes é construída. Esse tema, por si só, compreende todo um estudo na área de interface com o usuário (de Souza et al., 1999).

Ainda com relação à plataforma computacional, a criação de bibliotecas de classes para objetos e agentes de utilidade geral poderão a vir tornar o ONTOOL-2 uma ferramenta muito adequada para a integração e consolidação de sistemas complexos. Imaginamos um cenário onde objetos básicos para a construção de redes neurais, algoritmos evolutivos e processamento de regras e conhecimento fuzzy, poderão abrir uma grande frente de trabalho na criação de sistemas híbridos entre estas tecnologias. Facilidades como acesso a entrada de dados e visualizadores de estados internos, permitem também o uso do ONTOOL-2 como uma ferramenta muito útil na análise do comportamento de tais sistemas, permitindo uma prototipagem rápida e eficiente de tais sistemas.

Outra característica muito peculiar, é que, apesar de matematicamente uma rede de agentes ser um sistema fechado, com a inclusão dos conceitos de objetos externos remotos, é possível proceder-se à entrada de dados *on-line*, bem como obter-se a visualização de dados ou outras formas de saída dinâmica da rede. Isso de certa forma converte uma rede de agentes em um sistema aberto. Ao início dos trabalhos de tese, procurou-se encontrar uma maneira de descrever matematicamente o comportamento destes objetos externos. Entretanto, a formalização desta característica provou ser de uma complexidade insuspeita. Aparentemente, para dar cabo a esta empreitada, será necessário envolver e formalizar alguns conceitos semióticos de difícil assimilação, como por exemplo as idéias de terceiridade, transuação e mediação em Peirce (Santaella, 1999). Com isso, remete-se esta tarefa também para os trabalhos futuros.

De uma maneira geral, depreende-se que além das diversas contribuições adicionadas por esta tese, um longo caminho ainda está pela frente, sendo que um grande potencial para trabalhos futuros se abre no horizonte.

Apêndice A

Problemas de Satisfação de Restrições

Uma grande quantidade de problemas em Inteligência Artificial e em outras áreas da Ciência da Computação podem ser vistos como casos especiais de um problema de satisfação de restrições (CSP)[1]. Alguns exemplos típicos são: visão computacional, agendamento de tarefas, raciocínio temporal, problemas de grafos, etc. (Kumar, 1992; Hower, 1994; Russel and Norvig, 1995; Weiss, 1999).

Um CSP é caracterizado por um problema que tenta encontrar um conjunto consistente de atribuições de variáveis discretas finitas (Kumar, 1992; Hower, 1994; Russel and Norvig, 1995; Weiss, 1999). Formalmente, um CSP consiste em n variáveis $x_1, x_2, \ldots, x_n$ cujos valores são tomados em universos discretos e finitos dados por $D_1, D_2, \ldots, D_n$, respectivamente, além de um conjunto de restrições sobre seus valores. Uma restrição é definida por um predicado, isto é, a restrição $p_k(x_{k_1}, x_{k_2}, \ldots, x_{k_j})$ é um predicado definido no produto Cartesiano $D_{k_1} \times D_{k_2} \times \cdots \times D_{k_j}$. Este predicado é verdadeiro se, e somente se, os valores atribuídos a estas variáveis satisfazem a restrição. A solução de um CSP equivale a um conjunto de valores para as variáveis tal que todas as restrições sejam satisfeitas. Problemas deste tipo são geralmente do tipo NP-completo, tornando inevitável um processo de tentativa e erro durante a busca.

Há na literatura uma grande quantidade de algoritmos diferentes para a solução de problemas com restrições. Os mais comumente descritos compreendem (Kumar, 1992):

- **Paradigma gerar e testar (GT)**[2]. Neste paradigma, cada combinação possível de variáveis é sistematicamente testada para verificar se ela satisfaz as restrições. A primeira combinação a satisfazer as restrições é a solução. O número de combinações consideradas por este método é equivalente ao produto cartesiano de todos os domínios de variáveis.

- ***Backtracking* (BT)**. Neste método as variáveis são instanciadas sequencialmente. Tão logo todas as variáveis relevantes para o problema sejam instanciadas a validade das restrições é verificada. Se uma instanciação parcial violar algumas das restrições, um *backtracking* é realizado para a mais recente variável instanciada que ainda oferecer alternativas. Este técnica equivale a uma busca *depth-first* (Russel and Norvig, 1995) no espaço potencial de soluções. O algoritmo *backtracking* simples, apesar de apresentar melhores resultados que o GT, apresenta complexidade exponencial, e sofre do problema de *trashing*, quando a busca em diferentes partes do domínio falha repetidamente pelos mesmos motivos.

[1] Do inglês *constraint satisfaction problem*.
[2] Do inglês *generate and test*.

- **Propagação de restrições** (CP)[3]. As restrições entre variáveis diferentes são propagadas gerando um problema mais simples. Em alguns casos (dependendo do problema e do grau de propagação empregado) o CSP resultante pode ser tão simples que a solução pode ser encontrada sem busca.

Experimentos e análises de diversos pesquisadores mostram que a ordem em que as variáveis são instanciadas pode ter um impacto substancial na complexidade da busca *backtrack*. As abordagens comumente encontradas na literatura de CSP para obter um menor esforço computacional são (Kumar, 1992):

a) Variáveis com o menor número possível de alternativas restantes devem ser instanciadas primeiro.

b) Variáveis que participem de um grande número de restrições devem ser instanciadas primeiro.

c) Dada uma variável a deve ser instanciada que apresente um conjunto de valores possíveis, deve-se escolher aquele maximize o número de opções disponíveis para atribuições futuras.

d) Outra heurística é preferir um valor (dentre os disponíveis) que resulte num CSP o mais simples possível. Note que nesse caso é necessário que se estime a dificuldade de solução do CSP.

[3]Do inglês *constraint propagation*.

Apêndice B

CORBA (Common Object Request Broker Architecture)

O *Object Management Group* (OMG), um consórcio de mais de 700 empresas produtoras, revendedoras e usuários introduziu a *Object Management Architecture* (OMA), uma visão de alto nível de um sistema de computação completamente distribuído. A OMA consiste de quatro componentes, que podem ser aproximadamente categorizados em dois grupos principais: componentes orientados ao sistema (*Object Request Broker* e os serviços), componentes orientados a aplicação (objetos de aplicação e facilidades) (Saleh et al., 1999).

O componente mais importante desta arquitetura é o *Object Request Broker* (ORB). Ele permite que objetos interajam de forma independente de plataforma em um ambiente distribuído heterogêneo. ORBs cuidam da localização e ativação dos servidores, *marshalling* das requisições e respostas, tratam da concorrência e condições relativas a exceções. O padrão adotado pela OMG é chamado CORBA, que especifica um sistema que provê interoperabilidade entre objetos em um ambiente distribuído heterogêneo. Os serviços usados para o emprego de objetos distribuídos são chamados *CORBAservices*. Serviços para uso em aplicações distribuídas são chamados *CORBAfacilities*. Aplicações desenvolvidas com esses serviços são chamadas *ApplicationObjects*.

O modelo de objetos introduzido pelo CORBA é descrito por um grupo de objetos cooperantes que residem na mesma máquina ou em vários ao mesmo tempo. Os objetos no lado cliente podem invocar métodos nos objetos do lado servidor (*object implementation* é o nome CORBA usado para se referir aos objetos no lado servidor) usando sua *referência de objeto* (*object ID*). Qualquer requisição é interceptada e executada pelo ORB, que é responsável pela localização do objeto remoto e entrega da requisição. O objeto servidor processa a requisição e envia o resultado de volta através do ORB.

Neste modelo, o objeto cliente não precisa conhecer a implementação do objeto servidor. O servidor oferece apenas uma interface que serve de canal de comunicação para os serviços oferecidos. Consequentemente, existe uma separação nítida entre interface e implementação. Uma característica bastante importante é que tais implementações podem ser escritas em qualquer linguagem para a qual exista um mapeamento da interface para seu código específico.

Para atingir esta interoperabilidade, a arquitetura CORBA introduz uma linguagem neutra, declarativa e fortemente tipada, chamada *Interface Definition Language* (IDL). A IDL possui uma aparência semelhante àquela do Java e do C++, mas pode ser mapeada para uma grande quantidade de linguagens diferentes. Todas as interfaces de objetos que são exportadas são escritas em IDL. Posteriormen-

te, emprega-se um compilador especial para traduzir a especificação neutra da IDL para a linguagem desejada.

Uma aplicação cliente/servidor DOC (*Distributed Object Computing*) pode ser desenvolvida empregando-se os seguintes passos:

1. A interface do objeto servidor é escrita em IDL.
2. O código da interface IDL é compilado (mapeado) para a linguagem escolhida para a implementação (tais compiladores existem para Java, C++, Ada, Smaltalk, etc.). O Compilador produz dois tipos de classes: *classes stub* são ligadas ao código do objeto cliente e as *classes esqueleto* são ligadas ao código do objeto servidor.
3. A implementação dos métodos definidos na interface IDL é escrita na linguagem para o qual a interface IDL foi mapeada.
4. Os códigos das aplicações cliente e servidor devem ser compilados juntos, usando os compiladores nativos.

Após realizar os passos anteriores, os objetos cliente e servidor estão prontos para chamar os métodos um do outro.

Referências Bibliográficas

Agha, G. A. (1997). *IFIP Transactions*, chapter Formal Methods for Open Object-based Distributed Systems – IFIP Transactions. Chapman & Hall.

Agha, G. A. and Kim, W. (1999). Actors: A unifiying model for parallel and distributed computing. *The Journal of Systems and Architecture*, (45):1263–1277.

Albus, J. S. (1991). Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):473–509.

Booch, G., Rumbaugh, J., and Jacobson, I. (1997). *The Unified Modeling Language User Guide*. Addison Wesley.

Brooks, R. A. (1990). *Designing Autonomous Agents*, chapter Elephants Don't Play Chess. The MIT Press, Cambrige, MA.

Cardoso, J., Valette, R., and Dubois, D. (1999). Possibilistic petri nets. *IEEE Transactions on Systems, Man and Cybernetics*, 29(5):573–582.

Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What are ontologies and why do we need them? *IEEE Intelligent Systems*, pages 22–26.

Christensen, S. and Petrucci, L. (1992). Towards a modular analysis of coloured petri nets. In Jensen, K., editor, *Proceedings of Application and Theory of Petri Nets 1992*, volume 616 of *LNCS*, pages 113–133, Berlin,. Springer.

Corkill, D. D. (1991). Blackboard systems. *AI Expert*, 6(9):40–47.

de Souza, C. S., Leite, J. C., Prates, R. O., and Barbosa, S. D. J. (1999). Projeto de interfaces de usuário: Perspectivas cognitivas e semióticas. In *Anais do XIX Congresso Nacional da Sociedade Brasileira de Computação*, pages 425–476.

Esser, R. (1997). Applying an object-oriented petri net language to heterogeneus systems design. In *Petri Nets in Systems Engineering: modelling, verification, and validation*, Hamburg, Germany.

Genrich, H. and Lautenbach, K. (1981). System modelling with high-level petri nets. *Theoret. Comp. Sci*, 13:109–136.

Gerogiannis, V. C., Kameas, A. D., and Pintelas, P. E. (1998). Comparative study and categorization of high-level petri nets. *The Journal of Systems and Software*, (43):133–160.

Ghezzi, C. and Vigna, G. (1997). Mobile code paradigms and technologies: A case study. In Rothermel, K. and Popescu-Zeletin, R., editors, *Lecture Notes in Computer Science - Mobile Agents*, number 1219, pages 39–49, Berlin, Germany. Springer-Verlag. First International Workshop MA'97.

Green, S., Hurst, L., Nangle, B., Cunningham, D. P., Somers, F., and Evans, D. R. (1997). Software agents: A review. Internet link (01-jun-2000): `http://www.cs.tcd.ie/research_groups/aig/iag/pubreview.ps.gz`.

Gudwin, R. (1996). *Contribuições ao Estudo Matemático de Sistemas Inteligentes*. PhD thesis, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil.

Gudwin, R. and Gomide, F. (1997a). Computational semiotics: An approach for the study of intelligent systems - part I: Foundations. Technical Report DCA97-09, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil.

Gudwin, R. and Gomide, F. (1997b). Computational semiotics: An approach for the study of intelligent systems - part II: Foundations. Technical Report DCA97-09, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil.

Gudwin, R. and Gomide, F. (1998a). *Information/Intelligent Systems*, chapter Object Network: A Computational Framework to Compute with Words. Springer-Verlag.

Gudwin, R. R. and Gomide, F. A. C. (1997c). A computational semiotic approach for soft computing. In *Proceedings of the IEEE SMC'97 - IEEE International Conference on Systems, Man and Cybernetics*. IEEE. Orlando, FL, USA.

Gudwin, R. R. and Gomide, F. A. C. (1998b). A fuzzy elevator group controller with linear context adaptation. In *Proceedings of FUZZY-IEEE98, WCCI'98 - IEEE World Congress on Computational Intelligence*, pages 481–486, Anchorage, Alaska, USA. IEEE.

Guerrero, J., Gomes, A., and Gudwin, R. (1999). A computational tool to model intelligent systems. In *IV Simpósio Brasileiro de Automação Industrial*, pages 227–232.

Guerrero, J. A. S. (2000). Rede de agentes: Uma ferramenta para o projeto de sistemas inteligentes. Master's thesis, Universidade Estadual de Campinas, Campinas, São Paulo, Brasil.

Hower, W. (1994). Constraint satisfaction - algorithms and complexity analysis. In *Constraint Processing*, pages 103–108, Amsterdan, The Netherlands. Workshop notes of the Workshop W6 at the eleventh European Conference on Artificial Intelligence (ECAI'94).

Huber, P., Jensen, K., and Shapiro, R. (1990). Hierarchies in coloured petri nets. In Rozenberg, G., editor, *Advances in Petri Nets. Lecture Notes in Computer Science*, volume 483, pages 313–341, Berlin-Verlag. Springer.

Janecek, P., Ratzer, A. V., and Mackay, W. E. (1999). Redesigning Design/CPN: Integrating interaction and petri nets in use. Technical Report PB-541, University of Aarhus. CPN'99 Workshop.

Jensen, K. (1990). Coloured petri nets: A high level language for system design and analysis. In Rozenberg, G., editor, *Advances in Petri Nets. Lecture Notes in Computer Science*, volume 483, pages 342–416, Berlin. Springer.

Kendall, E., Malkoun, M., and Jiang, C. (1995). A methodology for developing agent based systems. In Chengqi Zhang, D. L., editor, *Lecture Notes in Artificial Intelligence*, number 1087, pages 85–99, Canberra, ACT, Australia. Springer-Verlag. First Australian Workshop on DAI.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *The AI Magazine*, 13(1):32–44.

Lakos, C. (1995a). The object orientation of object petri nets. In *16th International Conference on Applications of the Theory of Petri Nets – 1st Workshop on Object Oriented Programming and Models of Concurrency*, Turin, Italy. web link (01-jun-2000): `ftp://ftp.dsi.unimi.it/DSI/chizzoni/pub/papers/ws95/lakos.ps.gz`.

Lakos, C. and Keen, C. (1991). Simulation with object-oriented petri nets. In *Proceedings 6th Software Engineering Conference*, pages 203–216, Sydney.

Lakos, C. A. (1995b). From coloured petri nets to object petri nets. In *Proceeding of the 16th International Conference on Application and Theory of Petri Nets*, pages 278–297, Turin.

Maier, C. and Moldt, D. (1997). Object coloured petri nets - a formal technique for object oriented modelling. In *Proceedings of the Workshop on Petri Nets in System Engineering (PNSE'97)*, pages 11–19, Hamburg, Germany.

Meystel, A. M. (1996). Intelligent systems: A semiotic perspective. *International Journal of Intelligent Control and Systems*, 1(1):31–57.

Mira da Silva, M. and Rodrigues da Silva, A. (1997). Insisting on persistent mobile agents. In Rothermel, K. and Popescu-Zeletin, R., editors, *Lecture Notes in Computer Science - Mobile Agents*, number 1219, pages 174–185, Berlin, Germany. Springer-Verlag. First International Workshop MA'97.

Müller, J. (1996). Control architectures for autonomous and interacting agents. In Cavedon, L., Rao, A., and Wobcke, W., editors, *Lecture Notes in Artificial Intelligence - Intelligent and Agent Systems*, number 1209, pages 1–26, Cairns, ACT, Australia. Springer-Verlag. Based on a Workshop at PRICAI'96.

Murata, T. (1989). Petri net: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580.

Newell, A. (1982). The "knowledge level- artificial intelligence. *Artificial Intelligence*, (18):87–127.

Padghan, L. and Taylor, G. (1996). A system for modelling agents having emotions and personality. In Cavedon, L., Rao, A., and Wobcke, W., editors, *Lecture Notes in Artificial Intelligence - Intelligent and Agent Systems*, number 1209, pages 59–71, Cairns, ACT, Australia. Springer-Verlag. Based on a Workshop at PRICAI'96.

Pascoe, G. A. (1990). Elements of object-oriented programming. In Peterson, G. E., editor, *Object-Oriented Computing*, volume 1, pages 15–20. IEEE Computer Society Press.

Pedrycz, W. (1999). Generalized petri nets as pattern classifiers. *Pattern Recogniion Letters*, (20):1489–1498.

Reisig, W. (1985). *Petri Nets: An Introduction. EATCS Monographs on Theoretical Computer Science.* Springer, New York.

Reisig, W. (1992). *A Primer in Petri Net Design.* Springer, New York.

Russel, S. J. and Norvig, P. (1995). *Artifitial Intelligence: A Modern Approach.* Prentice Hall.

Saleh, K., Probert, R., and Khanafer, H. (1999). The distributed computing object paradigm: concepts and applications. *The Journal of Systems and Software*, (47):125–131.

Santaella, L. (1999). *O que é Semiótica.* Editora Brasiliense, 15a edition.

Selic, B., Gullekson, G., and Ward, P. T. (1994). *Real-Time Object-Oriented Modeling.* John Wiley and Sons, Inc.

Shulze, B. and Madeira, E. (1997). Contracting and moving agents in distributed applications based on a service-oriented architecture. In Rothermel, K. and Popescu-Zeletin, R., editors, *Lecture Notes in Computer Science - Mobile Agents*, number 1219, pages 74–85, Berlin, Germany. Springer-Verlag. First International Workshop MA'97.

Smith, B. C. (1996). *On the Origin of Objects.* MIT Press.

Smith, R. G. (1980). The contract net protocol: high level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113.

Snyder, A. (1993). The essence of objects : Concepts and terms. *IEEE Software*, 21(3):31–42.

Sommaruga, L. and Catenazzi, N. (1995). From practice to theory in designing autonomous agents. In Chengqi Zhang, D. L., editor, *Lecture Notes in Artificial Intelligence*, number 1087, pages 130–143, Canberra, ACT, Australia. Springer-Verlag. First Australian Workshop on DAI.

Stefik, M. and Bobrow, D. G. (1990). Object-oriented programming: Themes and variations. In Peterson, G. E., editor, *Object-Oriented Computing*, volume 1, pages 182–204. IEEE Computer Society Press.

Tanenbaum, A. S. (1992). *Modern Operating Systems.* Prentice Hall.

Valk, R. (1995). Petri nets as dynamical objects. In *16th International Conference on Applications of the Theory of Petri Nets – 1st Workshop on Object Oriented Programming and Models of Concurrency*, Turin, Italy. web link (01-jun-2000): `ftp://ftp.dsi.unimi.it/DSI/chizzoni/pub/papers/ws95/valk.ps.gz`.

Wegner, P. (1986). Classification in object-oriented systems. *ACM SIGPLAN Notices*, 21(10):173–173.

Weiss, G., editor (1999). *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence.* The MIT Press.

Wooldridge, M. (1999). *Multiagent Systems*, chapter Chapter 1: Intelligent Agents. The MIT press.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge Engineering Review*.

Zweben, M. and Fox, M. S. (1994). *Intelligent Scheduling*. Morgan Kaufmann.

Índice Remissivo

Índice Remissivo de Citações