UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB

CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

ESPECIFICAÇÃO FORMAL DE UM SISTEMA DE APOIO À DESCOBERTA

Germana Menezes da Nóbrega

Campina Grande - PB
Novembro, 1998

Germana Menezes da Nóbrega

ESPECIFICAÇÃO FORMAL DE UM SISTEMA DE APOIO À DESCOBERTA

Dissertação de Mestrado apresentada ao Curso de Pós-Graduação em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, como parte dos requisitos para a obtenção do grau de Mestre em Informática.

Orientador: Edilson Ferneda, Dr.

Linha de Pesquisa: **Teoria da Computação e Inteligência Artificial**
Área de concentração: **Ciência da Computação**

Campina Grande - PB
Novembro, 1998

N754e Nobrega, Germana Menezes da
Especificacao formal de um sistema de apoio a descoberta / Germana Menezes da Nobrega. - Campina Grande, 1998.
142 f.

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

1. Inteligencia Artificial 2. Agentes Racionais 3. Descoberta Cientifica 4. Dissertacao I. Ferneda, Edilson II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.8(043)

ESPECIFICAÇÃO FORMAL DE UM SISTEMA DE APOIO À DESCOBERTA

GERMANA MENEZES DA NÓBREGA

DISSERTAÇÃO APROVADA EM 17.11.98

PROF. EDILSON FERNEDA, Dr.
Presidente

PROF. ARTURO HERNÁNDEZ DOMÍNGUEZ, Dr.
Examinador

PROF. GEBER LISBOA RAMALHO, Dr.
Examinador

CAMPINA GRANDE - PB

Dedicatória

A meu avô, Carlos Ferreira da Nóbrega (*in memorian*), exemplo de grande ser humano que carrego no coração.

A meus pais, cujo amor incondicional extrapola todos os significados das palavras *maternidade* e *paternidade*.

A meu esposo, Geovany, pelo seu amor completo demonstrado a cada dia da nossa vida.

Agradecimentos

A Deus, a força maior que por instante algum deixou de mostrar sua presença, através de cada um daqueles que estimo.

A meu irmão, Christus, cuja compreensão demonstrada sobretudo nos últimos momentos da realização deste trabalho, não poderei jamais agradecer com palavras.

A minha irmã, Juliana, cuja credibilidade, por vezes chegando a superar minha própria auto-confiança, me encorajou muito mais nesta caminhada do que possa imaginar...

A Edilson, verdadeiramente mais que um orientador, um amigo.

Àqueles amigos que nunca, nunca me faltaram e que me confiaram por vezes suas alegrias e angústias.

Aos colegas mestrandos, com os quais tive o prazer de compartilhar momentos bons e difíceis nesta trajetória.

A Vera, Aninha, Zeneide, Manuella, Humberto, Lilian e Alberto, que sempre atenderam amavelmente às minhas muitas solicitações.

À CAPES, pelo suporte financeiro para a realização deste trabalho.

Resumo

No contexto de sistemas baseados em agentes, um agente racional pode ser visto como um sistema que aparenta ao usuário ser capaz de raciocinar, manipulando interações, crenças e conhecimento incompleto, impreciso e errôneo. O agente racional SAID é concebido dentro de tais princípios e é caracterizado pela capacidade produzir conhecimento passível de revisão e pela capacidade de dar explicações sobre suas decisões. Este trabalho apresenta uma especificação formal para um sistema de apoio à descoberta baseado no agente racional SAID. Isto representa um ponto de partida no sentido de uma implementação que propiciará, por um lado, o experimento de algoritmos de aprendizagem a partir de exemplos em vários domínios e, por outro lado, uma ferramenta de apoio à geração e validação de teorias científicas. Em um espectro mais amplo, pode-se dizer que pesquisas que visam a generalização de conhecimento são convenientes, dado o crescente volume de informação que manipula a sociedade contemporânea. Nesse sentido, este trabalho vai ao encontro dos atuais esforços realizados no âmbito de disciplinas como Aquisição de Conhecimento e Aprendizagem de Máquina.

Abstract

In the context of agent based systems, a rational agent may be seen as a system appearing to the user as being capable of reasoning, handling interactions, beliefs, as well as incomplete, imprecise and erroneous knowledge. The rational agent SAID is conceived under those ideas, and is caractherized by its skill of producing revisable knowledge, and also by its skill of explaining its decisions. This work introduces a formal specification for a system for aiding discovery based upon rational agent SAID. Such a specification means a starting point to an implementation that will provide, from one hand, experiments of learning from examples algorithms in a variety of domains, and, from other hand, an aiding tool for generation and validation of scientific theories. From a widespread point of view, one should say that research in the direction of knowledge generalization is suitable because the large amount of information people handle nowadays is increasing. In such a sense, this work meets actual investigations both in Knowledge Acquisition and Machine Learning field.

Sumário

Lista de Figuras

Introdução

Motivação

A quantidade crescente de dados e de informações que o homem contemporâneo vem manipulando gerou a necessidade de se criarem técnicas e ferramentas computacionais para extrair conhecimento de bases de dados de tamanho considerável. Em domínios científicos, o volume de informação também cresce na medida que novas descobertas são realizadas. Ferramentas computacionais que realizem generalizações de conhecimento em domínios que disponham de tais bases de dados são de grande relevância para que se criem novas teorias científicas.

A possibilidade do esboço de uma teoria normativa da descoberta e descobertas científicas históricas motivaram o desenvolvimento, dentro da Inteligência Artificial (IA), de sistemas capazes de simular tais descobertas. O desenvolvimento destes sistemas de descoberta contribuiu fundamentalmente para que se chegasse às bases da modelagem computacional da descoberta científica.

Hoje, técnicas desenvolvidas por pesquisadores em Aprendizagem de Máquina, Aquisição de Conhecimento e Sistemas Baseados em Conhecimento realizam com sucesso generalizações em grandes bases de dados e vêm sendo aplicadas em áreas relativamente novas como *Data Mining* e *Knowledge Discovery in Databases* (KDD). Aplicações em KDD têm sido desenvolvidas para Astronomia, Biologia, Finanças, Marketing, Medicina e muitos outros domínios do conhecimento.

No entanto, sistemas que resolvam problemas em cooperação com especialistas no domínio devem merecer atenção das pesquisas em IA neste momento, dada a impossibilidade de se embutir uma teoria completa em um sistema que trabalha sobre um domínio em constante evolução. De fato, a concepção de tais sistemas de apoio à descoberta tem seu ponto de partida nos já consagrados sistemas de descoberta.

Em IA, pode-se pensar na concepção de um sistema de apoio à descoberta na fronteira entre Aquisição de Conhecimento e Aprendizagem de Máquina, na medida que tal sistema verifica princípios como: a fase de obtenção de dados, a abstração a partir de dados dentro de um modelo

conceitual e a particularização deste modelo. Nessa fronteira, reencontra-se também a distinção entre *conhecimentos profundos*, aqueles que se justificam teoricamente e que se comunicam nos escritos científicos, e *conhecimentos situacionais*, aqueles geralmente chamados empíricos e que intervêm na organização dos conhecimentos especialistas e para os quais é desejada a evolução [WIELINGA, 1990]. Um sistema de apoio à descoberta assim concebido tem por função assistir a produção de conhecimentos, tirando partido ao mesmo tempo do conhecimento teórico sobre o domínio e de um conjunto de dados incompletos, imprecisos e errôneos. Em Aquisição de Conhecimento, o processo de descoberta tem por objetivo associar essas duas formas de conhecimento, pois ela trata da formulação dos conhecimentos passíveis de progressão e de revisão.

Objetivos

No contexto deste trabalho, um agente racional é visto como um sistema que aparenta ao usuário ser capaz de raciocínio como resultado da manipulação de interações, de crenças e de conhecimento incompleto, impreciso e errôneo. Um tal agente foi proposto por J. Sallantin: o agente racional SAID[1] [SALLANTIN, 1991A; 1991B]. Este agente combina Aprendizagem de Máquina e Aquisição de Conhecimento para gerar seu próprio conhecimento, fazê-lo evoluir e explicar suas decisões.

Busca-se, como objetivo principal deste trabalho, prover uma especificação formal de um sistema de apoio à descoberta a partir dos elementos que compõem o agente racional SAID. Tal especificação deverá servir de ponto de partida para uma futura implementação que propiciaria, por um lado, o experimento de técnicas de aprendizagem e, por outro lado, sua utilização como uma ferramenta de elicitação de conhecimento. A disponibilização dessa ferramenta permitirá a um especialista em um certo domínio intervir na organização e evolução do conhecimento deste domínio, através de mecanismos de crítica.

Na busca por subsídios que permitam, ao mesmo tempo, construir a especificação formal que se propõe e identificar limitações e perspectivas de trabalhos futuros afins, destacam-se como objetivos deste trabalho ainda o estudo da Descoberta Científica e seu enfoque no contexto da Ciência da Computação, bem como o estudo dos agentes racionais.

[1] SAID: Acrônimo para *Seulement Abduire, Induire et Déduire* (Somente Abduzir, Induzir e Deduzir).

Organização do Trabalho

No Capítulo 1, são apresentados os conceitos que fundamentam a Descoberta Científica e mostrados exemplos de alguns sistemas que obtiveram êxito ao simular descobertas científicas históricas, bem como os passos para o desenvolvimento de tais sistemas. São apresentadas também as novas tendências no sentido de Ambientes de Descoberta e a relevância de Sistemas de Apoio à Descoberta.

No Capítulo 2, discute-se a noção de agentes no contexto da Inteligência Artificial, apresentando-se um breve histórico desde a introdução do termo na Ciência da Computação até sua utilização em sistemas aprendizes.

No Capítulo 3, são apresentados os elementos que compõem do agente racional SAID, um sistema inteligente concebido para ter a capacidade de construir seu próprio conhecimento através da interação com um agente humano e dar explicações sobre suas decisões.

No Capítulo 4, apresenta-se uma modelagem orientada a objeto para um Sistema de Apoio à Descoberta, concebida a partir dos elementos que compõem o agente racional SAID.

No Capítulo 5, refina-se a modelagem proposta no Capítulo 4 através de uma especificação formal, utilizando um tipo particular de redes de Petri.

Finalmente, nas conclusões, são confrontados os objetivos desejados e os resultados alcançados. Vislumbram-se ainda as perspectivas de trabalhos futuros.

Capítulo 1

Descoberta Científica e Sistemas de Descoberta

1.1 Fundamentos para a Descoberta Científica

Desde os primórdios da civilização, o homem busca explicações sobre sua natureza e sobre a natureza dos fenômenos que o cercam. Assim nasceu a Filosofia e a Ciência ocidentais. Foi apenas no início do século XVII, porém, que se propôs articular aquilo que chamamos hoje o método da ciência moderna [CHALMERS, 1993], a partir das idéias de F. Bacon. Para Bacon [BACON, 1984], o conhecimento seria alcançado através da coleta de fatos com observação organizada e derivando teorias a partir daí. Sua concepção empirista de ciência, embora ainda ingênua [OLIVA, 1990], ia contra uma longa tradição escolástica, iniciada com Platão. Diversos pensadores modificaram e aperfeiçoaram as idéias de Bacon, entre os quais J. S. Mill [MILL, 1963], que trabalhou na sistematização do processo de geração de hipóteses.

Entre os diversos filósofos deste século que se preocuparam com a Filosofia da Ciência, destaca-se K. Popper. Popper, em sua Lógica da Descoberta Científica [POPPER, 1993], afirma que uma teoria científica é obtida pela invenção de hipóteses, conjecturas e até mesmo por «adivinhações». Segundo ele, uma teoria científica descreve coisas existentes em um mundo real e que, para a Ciência, não importa como esta é obtida mas sim como se verifica o seu grau de verossimilhança, isto é, de correspondência com o real. Para que uma teoria seja considerada científica, ela deve satisfazer o princípio da refutabilidade: ela deve arriscar-se a fazer predições passíveis de falsificação. Uma teoria é falsificável quando existe um conjunto potencial de falsificadores que possa ser submetido a testes no mundo real. Deste modo, uma teoria científica deve ser definida de forma que possa predizer algo sobre o mundo real.

Com essas idéias sobre a demarcação da Ciência, Popper marcou a Lógica da Ciência ao mostrar que não seria necessário se ter como princípio científico a lógica da indução; a Ciência teria como fundamento uma Lógica das Conjecturas e Refutações. Popper [POPPER 1975] afirma que a Ciência progride através da lógica situacional de resolução de problemas. O que ele chama

lógica situacional de problemas, ou *análise situacional*, está relacionado a uma explicação conjectural ou experimental de alguma ação humana que recorra à situação em que o agente se encontra.

Da mesma forma que Popper marcou o conhecimento na Ciência com o seu princípio da refutabilidade como demarcação do conhecimento científico, I. Lakatos marcou o conhecimento na Matemática. Ele formulou sua Lógica da Descoberta Matemática baseada também no princípio da refutabilidade. Segundo ele, seu objetivo ao formular a sua Lógica das Provas e Refutações baseia-se na constatação de que:

> "[...] *a matemática não formal, semi-empírica, não progride mediante monótono aumento do número de teoremas indubitavelmente estabelecidos, mas mediante incessante aperfeiçoamento de opiniões por especulação e crítica, pela lógica das provas e refutações.*" [LAKATOS, 1976]

A Lógica das Provas e Refutações parte do princípio de que o conhecimento na Matemática informal é hipotético, conjectural, e desenvolvido por meio de especulações e críticas. O seu objetivo é estudar a construção de uma prova e determinar o seu domínio de validade pela análise de exemplos e contra-exemplos dessa prova. Uma prova é um conjunto de lemas determinados a partir da análise de uma conjectura e suas sub-conjecturas. A análise de uma prova é realizada pela apresentação de exemplos e contra-exemplos. É desta forma que lemas são modificados e lemas são descobertos, em função das críticas ou refutações a que as provas são submetidas.

Pode-se notar que o método das provas e refutações exige uma abordagem interativa. Este método se interessa tanto pela elaboração de uma prova da conjectura como pela crítica dessa prova.

Essa discussão, mais especificamente sobre as idéias de K. Popper e I. Lakatos, foi desenvolvida no âmbito do Grupo de Inteligência Artificial da UFPB e é apresentada de forma aprofundada por M. E. de Souza e Silva [SILVA, 1998].

1.2 Inteligência Artificial e Descoberta Científica

Já no século XIX, os filósofos da ciência haviam começado a mostrar-se cada vez mais reticentes em relação à possibilidade de uma teoria para a descoberta e voltaram-se cada vez mais para o processo de verificação da validade de um conhecimento científico [LANGLEY, 1992]. Como resultado, até pouco tempo, a filosofia da ciência do século XX voltou-se quase que exclusivamente para o contexto da confirmação e quase nenhuma atenção era dada ao contexto da descoberta. Este era um pensamento baseado na idéia de que não existia um método lógico ou teoria

da descoberta científica e que a descoberta era um produto da criatividade. A tarefa de construir uma teoria da descoberta foi então deixada a cargo dos psicólogos.

N. R. Hanson foi um dos primeiros filósofos contemporâneos a retomar a questão de uma teoria da descoberta [HANSON, 1958]. Na mesma época que Hanson, A. Newell, J.C. Shaw e H. A. Simon propuseram que a descoberta poderia ser modelada como um processo de resolução de problemas [NEWELL, 1962]. De fato, o esboço de uma teoria normativa para a descoberta tem sido mostrado através de uma gama de sistemas computacionais de descoberta. Tal esboço aparece sob a forma de normas que, se seguidas, proporcionam a um cientista maiores chances de realizar descobertas. Parte-se do princípio de que as descobertas científicas raramente surgem de pesquisas aleatórias ou por tentativa e erro. Pesquisas aleatórias seriam inviabilizadas pela extensão dos espaços de busca. Nesse contexto, uma noção importante é a de *racionalidade*. Para um cientista, tal noção consiste em utilizar o melhor meio disponível para limitar a sua busca a proporções manipuláveis. É esta noção de racionalidade que é relevante para o processo criativo e para a resolução de problemas de maneira geral e que é de interesse para uma teoria normativa para a descoberta.

1.2.1 A Descoberta como Estratégia da Aprendizagem de Máquina

A aprendizagem abrange uma grande diversidade de fenômenos. O processo de aprendizagem inclui a aquisição de novos conhecimentos, o desenvolvimento de habilidades motoras e cognitivas através de instrução ou prática, a organização dos novos conhecimentos em representações genéricas e a descoberta de novos fatos e teorias pela observação e experimentação [CARBONELL, 1983]. Desde o início da era da computação, pesquisadores vêm tentando dotar os computadores de tais capacidades. Resolver este problema tem sido um dos objetivos da Inteligência Artificial (IA). O estudo e modelagem computacional do processo de aprendizagem constitui o tema da disciplina chamada *Aprendizagem de Máquina.*

Em Aprendizagem de Máquina, várias estratégias são estudadas. Segundo o montante de inferência que um sistema computacional aprendiz é capaz de realizar sobre alguma informação recebida, essas estratégias podem ser classificadas em:

- *Aprendizagem por Memorização.* Nenhuma inferência ou transformação sobre o conhecimento é requerida por parte do aprendiz. A aprendizagem se dá pela memorização dos dados que o sistema recebe;
- *Aprendizagem por Instrução.* Esse tipo de aprendizagem requer que o conhecimento ad-

quirido de uma fonte de informação seja transformado em uma representação interna mais diretamente manipulável pelo sistema aprendiz. A informação deve ainda ser integrada com conhecimentos previamente adquiridos. Alguma inferência é realizada pelo aprendiz, mas uma larga fração da responsabilidade é ainda do instrutor. Essa responsabilidade inclui a introdução e organização do conhecimento inicial do aprendiz;

- *Aprendizagem por Analogia.* Uma pessoa que nunca dirigiu um pequeno caminhão mas já dirigiu automóveis é capaz de transformar essa habilidade para realizar a nova tarefa (ainda que com imperfeição). Assim, um sistema que aprende por analogia poderia ser utilizado para converter um programa existente em um outro que realizasse uma tarefa semelhante à original. A aprendizagem por analogia requer mais inferência da parte do aprendiz do que as aprendizagens por memorização e por instrução. Um fato ou habilidade análogos ao que se quer aprender é um parâmetro bastante importante a ser retido pelo sistema aprendiz. Este fato ou habilidade deve ser transformado, aplicado à nova situação e armazenado para uso futuro;

- *Aprendizagem por Exemplos.* Dado um conjunto de exemplos e de contra-exemplos de um conceito, o aprendiz induz uma descrição do conceito que descreve os exemplos e não os contra-exemplos. Este tipo de aprendizagem tem sido bastante explorado na IA. O montante de inferência realizada pelo aprendiz é consideravelmente maior que no caso de aprendizagem por instrução (nenhum conceito genérico é fornecido pelo instrutor) e razoavelmente maior que no caso da aprendizagem por analogia (nenhum conceito similar é fornecido ao aprendiz). A fonte de informação de um sistema utilizando esse tipo de aprendizagem pode ser um instrutor, o próprio aprendiz ou o ambiente externo. Apesar de o instrutor fazer também parte do ambiente externo, espera-se que o aprendizado se dê de forma direcionada, enquanto que os exemplos vindos da observação de um ambiente externo podem ser completamente aleatórios;

- *Aprendizagem por Observação e Descoberta.* Esta é uma forma bastante genérica da aprendizagem indutiva que inclui sistemas de descoberta, tarefas de formação de teorias, criação de critérios de classificação para formar hierarquias taxonômicas e tarefas similares sem a intervenção de um instrutor. Esta forma de aprendizagem não supervisionada requer do aprendiz a realização de mais inferência do que em todas as outras abordagens discutidas até aqui. Nenhum conjunto de instâncias de um conceito é fornecido ao aprendiz e nenhum agente externo pode classificar as instâncias internamente geradas em positivas ou negativas em relação a um conceito. Na literatura encontram-se muitos exemplos de sistemas desenvolvidos sobre a abordagem da

Aprendizagem por Observação e Descoberta. Na sua maioria, tais sistemas simulam descobertas científicas históricas, como será mostrado a seguir neste capítulo.

Descreve-se a seguir, em linhas gerais, os conceitos que fundamentam a Descoberta Científica. Em seguida são mostrados exemplos de alguns sistemas de descoberta. Resume-se então o desenvolvimento de sistemas de descoberta, retomando os passos desse desenvolvimento. Finalmente discute-se a possibilidade de novas tendências no sentido de Ambientes de Descoberta e Sistemas de Apoio à Descoberta.

1.2.2 A Descoberta como Estratégia de Resolução de Problema

Os mecanismos da descoberta científica podem ser vistos como mecanismos genéricos de resolução de problemas [LANGLEY, 1992]. Esta idéia pode ser verificada através da construção de sistemas computacionais capazes de realizar descobertas científicas não triviais e que empregam métodos utilizados pelos humanos na resolução de problemas, particularmente, o método da busca heurística seletiva. Como mencionado, a idéia da descoberta científica como resolução de problema não é nova, mas tem sido discutida desde o início da década de 60 [NEWELL, 1962; SIMON, 1966; BUCHANAN, 1966].

Tomando a descoberta científica como caso especial da resolução de problemas humana, é natural que se recorra ao processo de resolução de problema como descrito pela psicologia cognitiva [LANGLEY, 1992]:

- O cérebro humano é um sistema processador de informações que manipula estruturas simbólicas, recebendo símbolos externos codificados através dos órgãos sensores e enviando símbolos codificados aos órgãos motores. Os pensamentos são realizados através da reorganização e comparação destes símbolos;
- O cérebro resolve um problema criando uma representação simbólica do problema (chamada *espaço do problema*), que é capaz de expressar situações iniciais, intermediárias e finais do problema. Além de aplicar muitos conceitos no processo de resolução, o cérebro utiliza *operadores* contidos na definição do espaço do problema para modificar as estruturas simbólicas que descrevem o problema (realiza uma busca por uma solução no espaço do problema);
- A busca por uma solução não é aleatória, mas seletiva. Ela é guiada no sentido de uma situação alvo (ou estruturas simbólicas que descrevem este alvo) por regras chamadas heurísticas. As heurísticas utilizam a informação extraída das definições do problema e

das situações (ou estados) já exploradas no espaço do problema para identificar caminhos promissores.

Há, contudo, alguns aspectos nos quais a descoberta científica difere de outras instâncias da resolução de problemas:

- a investigação científica é um processo social que geralmente envolve muitos cientistas e se estende por um longo período de tempo, enquanto que resolução de problema humana pode envolver um único indivíduo e por período de tempo consideravelmente menor;
- a investigação científica é caracterizada, muitas vezes, pela falta de objetivos bem definidos.

Apesar dessas diferenças, no entanto, é possível que os processos de resolução de problemas observados em algumas situações sejam aplicáveis aos processos componentes da descoberta científica, se considerarmos que é possível dividir o problema maior em pequenos problemas para, assim, atingir o objetivo de resolvê-lo.

Do exposto, pode-se pensar em um sistema computacional que simule a descoberta científica como um sistema processador de informações capaz de criar representações de problemas e de percorrer uma árvore de estados intermediários, buscando um estado final e utilizando um conjunto de heurísticas para guiar essa busca.

1.2.3 Heurísticas no Processo de Descoberta

As heurísticas são processos seletivos produzidos a partir do conhecimento que se tem de um problema. A seletividade das heurísticas vem das informações do problema em questão e as informações, por sua vez, vêm de sua estrutura [LANGLEY, 1992]. Se a estrutura do problema não apresenta padrões que possam ser incorporados no espaço do problema, não há bases para a construção de processos seletivos. A busca será, neste caso, por tentativa e erro.

As heurísticas podem ser hierarquicamente classificadas de acordo com sua generalidade. Quanto mais *genérica* for a heurística (método fraco), menos conhecimento específico do domínio será necessário para a sua aplicação e maior a possibilidade de sua aplicação em domínios variados. Por outro lado, quanto mais *específica* for uma heurística (método forte), em menos domínios ela poderá ser utilizada. Dessa forma, quanto mais se desce na hierarquia das heurísticas, mais se precisa de conhecimento do domínio do problema para construí-la e, já que heurísticas mais genéricas trabalham a partir de menos informação específica do problema, espera-se que esse

tipo de heurística seja menos seletivo que as heurísticas mais específicas.

Seguem três dos métodos fracos mais observados no processo humano de resolução de problema, sendo que o primeiro é um método mais genérico e os dois últimos são especializações do primeiro:

- *Gerar e testar.* Um operador gera movimentos no espaço do problema e um segundo operador testa a situação após cada movimento para verificar se uma situação alvo foi alcançada. A estrutura do operador que gera os movimentos determina se o método produz uma busca completamente aleatória ou seletiva;
- *Subida de morro.* Esta modificação do método gerar e testar produz um método onde a avaliação realizada pelo segundo operador é utilizada para a geração da situação seguinte. O sucesso deste método depende então da eficácia com a qual a avaliação estima a distância da situação corrente para a situação alvo;
- *Análise meio-fim.* Uma heurística de teste detecta diferenças específicas entre a situação corrente e a situação alvo e invoca operadores específicos para gerar movimentos nos quais não existam tais diferenças.

As heurísticas não são garantia de que se vá atingir uma situação alvo e nem que o caminho mais curto será encontrado. Contudo, o uso de heurísticas tem revelado que elas proporcionam uma *boa chance* de se resolver um problema.

1.2.4 Sistemas de Descoberta

Com o rápido crescimento da quantidade de dados e de conhecimento, surge a necessidade de não somente armazenar, organizar e disponibilizar essa informação mas também de utilizá-la de forma original. A modelagem de um processo de descoberta através de um sistema computacional é um passo importante no sentido de uma normatização para esse processo. Do exposto até aqui, conclui-se que um tal sistema pode ser visto como um sistema de resolução de problemas guiado por heurísticas.

A seguir são apresentados alguns sistemas computacionais de descoberta bastante conhecidos que têm gerado alguns resultados satisfatórios.

AM

O sistema AM (Automatic Mathematician) foi concebido por D. B. Lenat em 1977 para modelar um aspecto da pesquisa em matemática elementar, descobrindo novos conceitos guiado por uma larga gama de heurísticas [LENAT, 1983A]. Os conceitos são representados por estruturas do tipo *frame*. As heurísticas se comunicam via mecanismo de agenda, uma lista global de tarefas a serem realizadas pelo sistema e justificativas que caracterizam as tarefas como plausíveis. Uma tarefa pode levar o AM a definir um novo conceito, a explorar uma propriedade de um conceito (um *slot* de um *frame*) existente, a procurar regularidades em dados, etc. Repetidamente, o sistema seleciona da agenda a tarefa considerada mais plausível e a executa.

Inicialmente, há 115 estruturas correspondendo, cada uma, a um conceito elementar da teoria dos conjuntos, com muitos dos *slots* não preenchidos. A busca sobre essas estruturas é guiada por um conjunto de 255 heurísticas.

O AM expande seu conhecimento básico através da descoberta de conceitos e teoremas. Algumas heurísticas são utilizadas para selecionar os conceitos ou as propriedades de conceitos a serem explorados; outras procuram por informações relacionadas a uma propriedade escolhida; outras ainda identificam relacionamentos entre conceitos já conhecidos para definir novos conceitos e para avaliar o quão interessante é cada um deles.

A partir de conceitos básicos da teoria dos conjuntos, o AM redescobriu conceitos importantes da teoria dos números. Algumas realizações significativas do sistema foram as redescobertas dos números naturais e dos números primos.

Algumas conclusões importantes extraem-se, assim, da experiência com o AM:

(i) um conjunto de heurísticas pode guiar a busca por novos conceitos em um sistema e

(ii) à medida que surgem novos domínios do conhecimento, talvez um conjunto de heurísticas existente já não consiga guiar a busca por novos conceitos nesses domínios e é possível que novas heurísticas sejam necessárias.

EURISKO

Segundo Lenat [LENAT, 1983B], o desempenho do AM era limitado pela natureza estática de suas heurísticas. Os conceitos com os quais o sistema trabalhava, evoluíram em relação aos originais, enquanto que as heurísticas disponíveis para trabalhar com esses conceitos permane-

ceram as mesmas. O sistema EURISKO surgiu como extensão do AM. Sugeriu-se que as heurísticas fossem tratadas como conceitos completos que poderiam ser criados e modificados pelos mesmos processos que atuam sobre os conceitos do domínio do problema. Dessa forma, ao invés de representar conceitos de um domínio, EURISKO representa heurísticas. Essa representação é de tal forma que não se diferencia heurísticas sobre os conceitos de heurísticas sobre heurísticas. Ou seja, os dois níveis de heurísticas têm a mesma representação.

Quando AM e EURISKO foram aplicados a domínios com conceitos maiores e mais complexos que o da teoria dos números, alguns problemas surgiram **[RICH, 1994]**. A sintaxe da linguagem de representação não espelhava tão fielmente a semântica desses domínios e, como resultado, alguns conceitos mal formados começaram a aparecer.

BACON

O sistema AM mostrou como a descoberta poderia acontecer em um ambiente teórico **[RICH, 1994]** (descoberta dirigida por teoria). Mas os cientistas empíricos lidam com dados do mundo e precisam interpretá-los. Estes cientistas elaboram hipóteses e, para validá-las, projetam e executam experiências. Um modelo de descoberta científica dirigida por dados foi proposto e implementado no sistema BACON, um sistema com o propósito de descobrir leis quantitativas. BACON foi desenvolvido na década de 80 por P. Langley e outros **[LANGLEY, 1981]**. O nome surge em homenagem a F. Bacon que, como mostrado, foi um dos primeiros a defender a idéia de uma teoria normativa para a descoberta científica.

O BACON.6, o sexto da série dos sistemas BACON, recebe um conjunto de variáveis dependentes e independentes e gera leis que relacionam essas variáveis entre si. Essas leis são geradas a partir dos dados numéricos, através de heurísticas projetadas para perceber as relações entre estes dados. Adicionalmente, outras heurísticas são utilizadas para minimizar o espaço de busca.

A partir de dados fornecidos, o sistema conseguiu redescobrir algumas leis já bastante conhecidas tais como a *Lei do Gás Ideal*, a *Terceira Lei de Kepler*, a *Lei de Coulomb* e a *Lei de Ohm*. **[LANGLEY, 1992]**.

GLAUBER

GLAUBER **[LANGLEY, 1992]** é um sistema para descoberta de leis qualitativas em Química. O nome homenageia o químico alemão J. R. Glauber, que viveu no século XVII e teve importante

papel no desenvolvimento da teoria dos ácidos e bases.

O sistema é alimentado com um conjunto de informações referentes a reações químicas e com a indicação das substâncias iniciais e resultantes para cada reação. São também fornecidas algumas leis qualitativas e espera-se que o sistema proponha leis mais genéricas. Heurísticas são empregadas para guiar o processo de criação de classes de substâncias através da busca de características comuns nas substâncias. Quando uma classe é criada, o sistema substitui todas as ocorrências das substâncias da classe por uma referência a esta classe. GLAUBER utiliza uma outra heurística para perceber a quantificação das fórmulas derivadas, tendo redescoberto, por exemplo, a *Lei dos Ácidos e Bases* («todo ácido reage com toda base formando um sal»). Graças a esse tipo de heurística evita-se a geração de uma lei incorreta como «todo ácido reage com toda base formando todo sal».

BOOLE

Em 1847, G. Boole descobriu que a Lógica, representada de forma adequada, poderia ser considerada como um ramo da álgebra [LEDESMA, **1997**]. Em outras palavras, todos os resultados conhecidos em lógica poderiam ser obtidos pelo uso de técnicas matemáticas comuns. A descoberta de Boole foi realizada graças à aplicação do Método de Separação de Símbolos, que é uma forma de representar um ramo qualquer da matemática em termos de álgebra. A heurística da Separação de Símbolos tem sido usada pelos matemáticos desde o século XIX e é assim chamada porque os símbolos em um domínio são separados do significado particular que têm nesse domínio e, uma vez que uma porção de conhecimento tenha sido representada por símbolos e passado pela separação de símbolos, seus símbolos comportar-se-ão como se estivessem em qualquer outro ramo da Matemática que também tenha passado pela separação de símbolos.

O BOOLE2 é um sistema que recebe como parâmetro de entrada um conjunto de operações e um conjunto de combinações representando um conhecimento científico. O resultado gerado é um registro das propriedades algébricas do tal conhecimento e uma indicação sobre a possibilidade deste ser passível de sofrer separação de símbolos (*simbolizável*). O programa é capaz de descobrir que a Lógica, a Geometria e um subconjunto do Cálculo Diferencial são simbolizáveis e que a generalização da Geometria gregoriana não o é devido ao seu leque de comutatividade.

É interessante observar que, diferentemente dos programas da família BACON e de GLAUBER, BOOLE2 não realiza descoberta a partir de dados, mas a partir de uma representação abstrata de algum conhecimento científico.

1.2.5 Desenvolvimento de Sistemas de Descoberta

O desenvolvimento de sistemas de descoberta tem sido motivado geralmente por grandes descobertas científicas que entraram para a história. Alguns passos são observados no desenvolvimento de tais sistemas, se vistos como sistemas de resolução de problemas guiados por heurísticas:

(i) escolhe-se um episódio de descoberta na história;

(ii) constrói-se um espaço para o problema;

(iii) escolhe-se uma representação para os dados e

(iv) constrói-se um conjunto de heurísticas, a partir dos fatos registrados na história.

Uma vez realizados todos os passos para a construção do sistema, pode-se iniciar a fase de teste. No caso de descobertas históricas é desejável que o sistema reproduza não apenas o resultado final, mas também que siga os mesmos caminhos registrados no fato histórico sendo simulado. Esta regra é comumente aceita como teste para sistemas que simulam descobertas históricas.

Reconhecido o mérito dos sistemas que simulam descobertas históricas, vale ressaltar que a tendência atual é o desenvolvimento de sistemas que realizem descobertas completamente novas. A partir das bases da descoberta lançadas pelos sistemas de simulação, acredita-se que um grande avanço pode ser conseguido no sentido de novos resultados. De fato, esses resultados já começam a aparecer, como mostra o caso de uma nova lei na física das partículas descoberta com a utilização do sistema PAULI [VALDES-PEREZ, 1996].

1.2.6 Ambientes Integrados de Descoberta

O avanço da ciência é um processo desenvolvido em muitas etapas. É necessário identificar problemas, encontrar uma representação adequada para esses problemas, coletar dados por observação ou experimentos e buscar regularidades e generalizações que descrevam a organização desses dados. Generalizações de leis são desejadas para que se criem teorias melhores e mais compreensíveis. É desejável ainda a predição a partir de tais teorias, assim como sua verificação por novas observações e experimentos [DARDEN, 1997].

Todo o aparato disponível ao cientista para realizar suas tarefas (ferramentas, equipamentos, métodos) é chamado ambiente de descoberta. Com o desenvolvimento de ferramentas em I.A. cada vez mais poderosas surgem novas possibilidades para ambientes de descoberta. O planeja-

mento de experimentos e o descobrimento de regularidades em grandes massas de dados já vêm sendo realizados por programas de descoberta. O conhecimento sistematizado já pode ser transformado em bancos de dados científicos e bases de conhecimento. Todas essas ferramentas computacionais integradas compõem um *ambiente de descoberta apoiado por computador.*

Um ambiente de descoberta computadorizado deve apresentar uma conectividade entre as ferramentas de modo que elas se complementem. Um tal ambiente é proposto por H. Jong e A. Rip [JONG, 1997].

1.3 Discussão

O desenvolvimento de sistemas computacionais que simulam com sucesso descobertas científicas históricas contribuiu fundamentalmente para que se chagasse às bases da modelagem computacional da descoberta científica. Por outro lado, já se fala em ambientes integrados de descoberta que integrem todo o aparato de ferramentas utilizadas por um cientista em seu trabalho. Contudo, alguns passos ainda precisam ser dados antes que se tenha um tal ambiente efetivamente disponível.

Esforços precisam ser dirigidos, nesse momento, para o desenvolvimento de sistemas que resolvam problemas em cooperação com especialistas no domínio. Tais sistemas devem ser vistos como tendo um papel muito importante porque, em domínios não triviais, será virtualmente impossível incorporar ao sistema uma teoria completa que está em constante evolução. Adicionalmente, na medida que o domínio em questão torna-se mais complexo, deve-se atentar para a maneira através da qual o sistema e o especialista no domínio se comunicam e para a forma como o conhecimento relevante do usuário pode ser identificado e construído [SIMON, 1997].

No contexto da realização de sistemas capazes de construir uma teoria em cooperação com um especialista no domínio, itervêm cinco diferentes etapas [SALLANTIN, 1991A]: (*i*) *conceitualização*, que objetiva um acordo entre a apreensão e a formulação do problema em questão; (*ii*) *modelagem*, que precisa através de uma linguagem formal a conceitualização do problema; (*iii*) *cálculo* ou programação, que realiza de forma concreta o que foi estabelecido na modelagem; (*iv*) *experimentação*, que examina os resultados do cálculo e, finalmente, (*v*) *cognição*, que realiza a avaliação da experimentação. Este trabalho vai ao encontro da modelagem de um tal sistema, a qual é proprosta nos Capitulos 4 e 5, tomando como ponto de partida a conceitualização do agente racional SAID, que é apresentado no Capítulo 3. Discute-se no próximo capítulo o impacto da noção de *agente* para a comunidade da I.A.

Capítulo 2

Agentes Racionais

2.1 Introdução

Para a comunidade da computação baseada em agentes, a questão "o que é um agente?" é provavelmente tão embaraçosa quanto é para a comunidade da IA, de modo geral, a questão "o que é inteligência?". Se, por um lado, a adoção do termo *Inteligência* pelos pesquisadores de IA os faz deparar com várias críticas de âmbito filosófico, por outro lado, a busca por um consenso sobre o significado do termo *agente* pode ser frustrada. Essa ausência de consenso, contudo, não tem representado um obstáculo ao avanço da produção científica nem tampouco ao desenvolvimento de aplicações dentro desta área do conhecimento. De fato, ao investigar a literatura a respeito, observam-se divergências que vão desde quais características devem ou não estar associadas ao conceito até a organização dessas características em noções *fraca* e *forte* de agente [WOOLDRIDGE, 1995].

Apesar destas divergências, que na verdade dizem respeito ao conjunto de características que deve-se incorporar ao projeto de um agente, há um objetivo fundamental que se busca com a concepção de um sistema como um agente. É o que se busca esclarecer neste capítulo.

Na seção 2.2, discorre-se sobre o estudo dos agentes dentro da I.A., abordando-se desde a origem do termo "agente" até sua contribuição aos sistemas aprendizes. Na seção 2.3 mostra-se como são encarados hoje pela I.A. os termos "inteligência" e "racionalidade" quando atribuídos a agentes. Na seção 2.4 conclui-se o capítulo, destacando-se o mérito desta abordagem no projeto de sistemas inteligentes.

2.2 Agentes Inteligentes

2.2.1 A Noção de Agente segundo Newell

Na Figura 2.1 são mostrados diferentes níveis nos quais podem ser visualizados os siste-

mas computacionais. Um nível é composto de:

- *medium*, ou o que é processado no nível;
- componentes, que provêm o processamento primitivo;
- leis de composição, que permitem que os componentes sejam montados formando o sistema e
- leis de comportamento, que determinam como o comportamento do sistema depende do comportamento dos componentes e da estrutura do sistema.

Nível	*Medium*
Nível de configuração	Dados
Nível de programa (nível simbólico)	Expressões simbólicas
Nível transferência-registrador	Vetores de bits
Nível lógico	Bits
Nível de circuito	Corrente, voltagem
Nível de dispositivo	Elétrons, campo magnético

Figura 2.1: Níveis de sistemas computacionais associados a seus *mediums*.

A. Newell [NEWELL, 1982] propõe um sexto nível - o nível de conhecimento - disposto "ao lado" do nível de configuração. Segundo sua "Hipótese do Nível de Conhecimento", existe um nível distinto de sistemas computacionais, situado imediatamente acima do nível de símbolo, que é caracterizado pelo *conhecimento* como o *medium* e pelo *princípio da racionalidade* como a lei de comportamento.

Dessa forma, para Newell, *agente* é o sistema no nível de conhecimento. Os componentes no nível de conhecimento são os *objetivos*, as *ações* e um *corpo*. Assim, um agente é composto por um conjunto de ações, um conjunto de objetivos e um corpo. O agente então processa seu conhecimento para determinar ações a realizar. A lei de comportamento (o princípio da racionalidade) pode ser entendida como "ações são selecionadas de maneira a permitir a realização dos objetivos do agente".

A introdução do nível de conhecimento por Newell foi significativamente importante no direcionamento das pesquisas em I.A., sobretudo em sistemas baseados em conhecimento. Através de tal noção, Newell explicitou o que vinha sendo observado como prática comum em I.A. Ou seja, falar de sistemas inteligentes em uma linguagem de "conhecimento" e "intenção".

Nas abordagens de modelagem de sistemas baseados em conhecimento, o desenvolvimento de um sistema pode ser visto como a construção de uma série de modelos relacionados a algum comportamento. Em particular, o modelo no nível de conhecimento[2] é um modelo em termos de conhecimento que racionaliza o comportamento. Estabeleceu-se assim uma prática comum explícita na comunidade de I.A.: o tratamento de sistemas no nível de conhecimento de Newell. Tal prática pode ser verificada observando-se definições de I.A. trazidas por abordagens recentes, a exemplo de:

> "[...] *o ramo da ciência da computação que tem como objetivo a construção de agentes que aparentem comportar-se inteligentemente.*" [RUSSELL, 1995].

2.2.2 Agentes em Inteligência Artificial Distribuída

A I.A. Distribuída (I.A.D.) une a concepção convencional de I.A. com os princípios da computação distribuída. Inicialmente, pensou-se em fazer com que sistemas inteligentes fossem capazes de cooperar na realização de tarefas complexas possibilitando o envolvimento de domínios distintos. A contribuição de cada sistema estava vinculada ao seu domínio do conhecimento. É imediato perceber que tal cooperação entre os sistemas pressupunha a necessidade de dotá-los de uma capacidade de comunicação razoavelmente complexa.

A I.A.D. pode ser vista como o ramo da I.A. que trata dos processos cooperativos de resolução de problemas através de uma sociedade descentralizada de agentes [HUHNS, 1987]. Dentro desta visão, destacam-se três idéias como fundamentais:

(*i*) o termo *sociedade* é geralmente utilizado pela comunidade da I.A.D. para designar um grupo de agentes imersos em um mesmo ambiente;

(*ii*) os processos são ditos *cooperativos* no sentido de que um compartilhamento de informação faz-se necessário para que a sociedade como um todo chegue a uma solução para o problema em questão; e, finalmente,

(*iii*) dizer que a sociedade é *descentralizada* significa dizer que tanto o controle como os dados podem estar (e geralmente estão) lógica e/ou geograficamente distribuídos.

A idéia de entidades que cooperam entre si na busca por uma solução para um problema, ou seja, a própria concepção da I.A.D. como apresentada anteriormente, é provavelmente uma das maiores vantagens de se estudar e aplicar essa abordagem. A metáfora com o comportamento so-

[2] Vale lembrar que sistemas inteligentes, segundo o próprio Newell, não devem ser descritos exclusivamente em termos do nível de conhecimento.

cial humano, dotado de capacidade de comunicação, tem proporcionado aos sistemas desenvolvidos dentro desta concepção algumas vantagens, tais como:

- *modularidade*. A decomposição do problema maior em subproblemas facilita o desenvol-vimento, testes e manutenção de tais sistemas;
- *paralelismo*. As entidades modularizadas ou subsistemas podem operar em paralelo;
- *tolerância a faltas*. O sistema continua operando mesmo se uma das entidades deixa de funcionar por alguma razão;
- *reusabilidade*. Uma entidade independente pode eventualmente tornar-se parte de vários sistemas sem sofrer alterações.

A atração exercida por essas características, comprovada pelo crescente interesse de pesquisadores em áreas como a robótica e a I.A., revela a necessidade de um novo paradigma baseado na noção de agentes (como entidades inteligentes com capacidade de comunicação) para a concepção de sistemas complexos. Nesse sentido, algumas pesquisas já vêm sendo realizadas [**SHOHAM, 1993**].

J. S. Sichman [**SICHMAN, 1992**] divide o universo das pesquisas em I.A.D. em duas abordagens: *Resolução de Problema Distribuída* (R.P.D.) e *Sistemas Multi Agentes* (S.M.A.). Segundo a abordagem R.P.D., agentes são projetados em função da resolução de um problema. Ou seja, dado um problema, a decomposição de tarefas entre os agentes é feita pelo projetista. Para a abordagem S.M.A., a existência dos agentes é independente da existência de outros agentes e de um problema em particular. Neste caso, quando um problema é submetido a uma sociedade (já existente), os agentes decidem entre si sobre a decomposição de tarefas e até mesmo sobre a necessidade de entrada de um novo agente na tal sociedade (característica de *sistema aberto*). Esta abordagem é mais complexa no sentido de que, para a realização da decomposição, os agentes precisam manter uma representação dos demais agentes a fim de determinarem *quem* é capaz de fazer *o que*.

2.2.3 Algumas Características de Agentes Inteligentes

A seguir, são enumeradas algumas características que, ainda que de maneira não consensual, encontram-se na literatura como sendo atribuídas a sistemas projetados como agentes.

Habilidade social

A partir da idéia de descentralização lógica da I.A.D., pode-se imaginar um agente como uma entidade que toma parte em um processo de resolução de problema. Um conjunto de tais en-

tidades (uma sociedade) teria assim um objetivo bem definido: obter uma ou mais soluções para o problema em questão. Assumindo que cada agente da sociedade seja capaz de resolver apenas parte do tal problema, é desejável que esses indivíduos interajam via algum tipo de linguagem de comunicação para que, de forma cooperativa, busquem uma solução [GENESERETH, 1994]. Percebe-se portanto que, do ponto de vista da I.A.D., sistemas inteligentes resolvem problemas respeitando um comportamento social, donde sobressaem *ações* (realização de tarefas que resolvem o problema ou parte dele) e *interações* (comunicações entre os indivíduos visando a resolução de problemas) [HUHNS, 1987]. Em contraste, a noção de inteligência na IA clássica aparece associada a um comportamento individual (introspectivo) e as atenções ao se construir um sistema dentro dessa abordagem são principalmente voltadas à representação do conhecimento e aos métodos de inferência a ela associados.

É importante notar que esta caracterização inicial de agentes em muito se assemelha à concepção do paradigma da orientação a objeto, onde entidades também são dotadas de capacidade de comunicação. Mas a noção de agentes inteligentes comunicates pressupõe um certo nível de complexidade adicional. Em se tratando do paradigma orientação a objeto, quando um método de uma classe qualquer é invocado, salvo problemas com o meio físico, ele será certamente executado. No contexto de agentes, pressupõe-se algo mais do que uma simples chamada a um método seguida de sua execução: assim como para os seres humanos, espera-se de um agente um tipo de comunicação mais complexo, associado provavelmente à característica de *não-benevolência*, apresentada mais adiante.

Reatividade

> "*Um agente é algo que pode ser visto como percebendo seu ambiente através de sensores e agindo nesse ambiente através de atuadores.*" [RUSSELL, 1995]

A definição acima é bastante flexível. Esta flexibilidade pode ser observada pela aplicação da definição, por exemplo, aos seguintes três contextos diferentes de agente:

(i) como um programa de computador abrindo um arquivo para leitura no ambiente do sistema operacional UNIX ou

(ii) como um robô pegando um objeto no ambiente de uma linha de montagem.

O que deve ficar claro é que, ao projetar-se um agente artificial[3] baseado nessa definição é preciso impor restrições sobre cada um dos termos *percebendo*, *ambiente*, *sensores*, *agindo* e *atuadores* e,

[3] Daqui por diante chamaremos de agente artificial um agente não humano: um agente de software e/ou hardware.

mais ainda, cuidar para que essas restrições sejam razoavelmente consistentes entre si. Observe-se que, segundo a definição, um agente é algo que, antes de mais nada, percebe seu ambiente e age sobre ele. O princípio da reatividade para um agente pressupõe *percepção* e *ação*. Este princípio pode ser enunciado como sendo a capacidade de agir sobre o ambiente mediante as mudanças nele percebidas [WOOLDRIDGE, 1995].

Pró-atividade

Os agentes não apenas agem em resposta às mudanças do ambiente no qual estão imersos, mas também devem ser capazes de exibir um comportamento segundo seus objetivos através da *tomada de iniciativa* [WOOLDRIDGE, 1995].

Autonomia

Uma definição de agente, segundo Ferber [FERBER, 1991], é a seguinte:

> "*Um agente é uma entidade real ou virtual que está imersa em um* ***ambiente*** *no qual pode realizar algumas ações, que está apta a* ***perceber*** *e* ***representar*** *parcialmente este ambiente, que está apta a* ***comunicar-se*** *com outros os agentes e que possui um* ***comportamento autônomo*** *que é conseqüência de suas observações, seu conhecimento e suas interações com os outros agentes.*"

Observa-se nesta definição que a autonomia aparece embutida no conceito de agente.

Considere-se agora o que dizem S. Russell e P. Norvig sobre a autonomia de um sistema:

> "*Um sistema é autônomo no sentido de que seu comportamento é determinado por sua própria experiência.*" [RUSSELL, 1995]

Segundo os autores, falta autonomia a um sistema que trabalha apenas de acordo com o conhecimento que lhe foi inicialmente embutido. Um sistema dotado de um conhecimento inicial, de capacidade de perceber e de aprender é capaz de fazer evoluir seu conhecimento e agir segundo este conhecimento. Um tal sistema é dito ser autônomo. Esta definição de autonomia é adotada no contexto deste trabalho.

Não-benevolência

Um agente é *não-benevolente* quando, mesmo se solicitado por outros agentes da sociedade, não necessariamente coopera. Vale ressaltar que esta propriedade não se aplica deliberadamente. Ou seja, a recusa em realizar determinada tarefa acontece quando o agente detecta um conflito entre seus objetivos (*objetivos locais*) e os objetivos da sociedade (*objetivos globais*). Fica a

cargo do agente decidir se deve ou não cooperar ao ser requisitado [SICHMAN, 1996].

A relação entre a não-benevolência e o grau de complexidade das comunicações entre agentes pode agora ser melhor entendida. A linguagem utilizada para prover esta comunicação deve ser capaz de suportar não só uma solicitação de execução de tarefa como também toda uma "negociação" que eventualmente se instaura após tal solicitação. De fato, pode-se imaginar diferentes níveis de complexidade envolvendo a comunicação entre agentes [COSTA, 1997]. Em sistemas relativamente menos complexos, a comunicação pode se dar de maneira mais elementar. É o caso em que os agentes são sempre benevolentes, uma vez que se pode presupor a ausência de conflitos entre objetivos globais e locais.

Sinceridade

O princípio da sinceridade pressupõe que um agente jamais fornecerá "conscientemente" informações incorretas a um outro agente [SICHMAN, 1996]. Ou seja, ele sempre propagará as informações nas quais acredita. Em alguns casos, contudo, pode haver uma justificativa plausível para que um agente tente "enganar" um outro.

Auto-conhecimento

Segundo esse princípio, os agentes devem ter uma completa e correta representação de si mesmos: seus objetivos, suas capacidades, etc. A respeito dos demais, por outro lado, agentes devem ter *crenças*, que, por sua vez, podem ser incompletas ou incorretas [SICHMAN, 1996].

Consistência

Uma propriedade importante com a qual um projetista de um sistema como um agente deve preocupar-se é com a consistência. Segundo essa propriedade, um agente jamais terá crenças contraditórias a respeito dos demais agentes [SICHMAN, 1996]. A manutenção da consistência é o processo através do qual o agente, a partir do momento em que detecta uma inconsistência, reorganiza suas crenças.

2.2.4 Agentes Reativos e Agentes Cognitivos

Uma vez apresentadas algumas características aplicáveis ao projeto de um agente, mostra-se nesta seção como algumas destas características são atribuídas a dois tipos de agentes que são extremos segundo a classificação em *agentes reativos* e *agentes cognitivos* [SICHMAN, 1992]. O

projeto de um agente pode enquadra-se totalmente em um ou outro extremo, ou ainda combinar características dos dois tipos de agentes.

Agentes Reativos

Considere-se um exemplo clássico para esta categoria de agentes: uma colônia de formigas. Embora uma formiga tomada isoladamente não apresente um comportamento que se possa classificar como "inteligente", a colônia, como um todo, pode ser considerada uma entidade inteligente. A idéia principal em torno da noção de agentes reativos é que a "inteligência" está associada a uma entidade formada por indivíduos, cujo comportamento particular não é, necessariamente, tido como inteligente. Esta abordagem pode ser comparada à abordagem conexionista. Porém, uma rede neuronal, uma vez treinada, não é capaz de mudar seu comportamento, enquanto que agentes reativos podem fazê-lo em função das alterações percebidas no ambiente.

O comportamento de agentes reativos é da forma estímulo-resposta (princípio da *reatividade* mostrado na seção 2.2.2). Estes agentes não retêm um histórico de suas experiências passadas, como também não planejam as ações a serem realizadas no futuro. Não há um modelo de comunicação de alto nível, nem tampouco uma representação explícita do ambiente e dos demais membros da sociedade. Em geral, este tipo de sociedade contém um grande número de membros.

Agentes Cognitivos

O comportamento de uma sociedade de agentes cognitivos é baseado na organização social humana. Ou seja, pode-se pensar na característica *habilidade social* (seção 2.2.2) com um protocolo de comunicação de alto nível associado. Este tipo de abordagem requer ainda que os agentes representem explicitamente o ambiente, bem como os demais indivíduos da sociedade. Cada agente é capaz de raciocinar sobre as ações realizadas no passado e planejar tomadas de decisões futuras. Uma sociedade de agentes cognitivos tem geralmente um pequeno número de membros.

2.2.5 Agentes Inteligentes e Aprendizagem

Pode-se imaginar agentes cuja capacidade de resolução de problemas tenha sido inteiramente introduzida por seu projetista. Em várias situações esta pode não ser necessariamente a melhor abordagem. No caso em que o projetista tem um conhecimento incompleto do ambiente onde o agente deve operar, a aprendizagem é a única maneira do agente adquirir o conhecimento necessário para realizar suas tarefas [**RUSSELL, 1995**]. Percebe-se assim que a aprendizagem provê autonomia, no sentido de que o comportamento do agente é determinado por sua própria experiência.

Trabalhos como o de J. P. Müller [MÜLLER, 1987] mostram a viabilidade da construção de sistemas autônomos que:

(i) se são capazes de interpretar estruturas simbólicas;

(ii) se são conscientes de suas limitações;

(iii) se agem de acordo lógico com suas crenças e

(iv) se são capazes de adaptar suas ações às mudanças nos seus conhecimentos,

serão, então, capazes de melhorar sua representação do mundo externo e de melhor interagir com este mundo. Tais trabalhos mostram que essa capacidade de construir e fazer evoluir a representação do mundo desses sistemas pode ser acrescentada à capacidade de aprender de um agente.

No Capítulo 1 mostrou-se uma visão geral do domínio da Aprendizagem de Máquina e de suas estratégias, que vão desde memorização de experiências até a criação de teorias científicas. Na Figura 2.2 é apresentado um modelo genérico de agente aprendiz, dividido em quatro componentes conceituais [RUSSELL 1995]:

- *Componente de Execução.* Responsável por selecionar uma ação a ser realizada. Em agentes "não-aprendizes", este é o único componente. Ou seja, este componente recebe as percepções e decide que ação realizar de acordo com essas percepções.
- *Componente de Aprendizagem.* Responsável por incrementar o conhecimento do agente. Este componente recebe conhecimento do Componente de Execução e um retorno do crítico indicando a pertinência das ações do agente. Dessa forma ele é capaz de determinar alterações no Componente de Execução, de forma que possa agir "melhor" no futuro.
- *Crítico.* O papel deste componente é informar o Componente de Aprendizagem sobre o desempenho do agente, através de uma medida de desempenho. Isto é necessário pois as percepções nada mostram sobre o sucesso do agente. Por exemplo, um sistema computacional que joga xadrez pode perceber que realizou um cheque-mate, mas precisa de uma medida de desempenho que lhe diga que esta é uma "boa" ação. É importante que esta medida venha de fora do agente, evitando assim uma auto-avaliação, sujeita a ser tendenciosa.

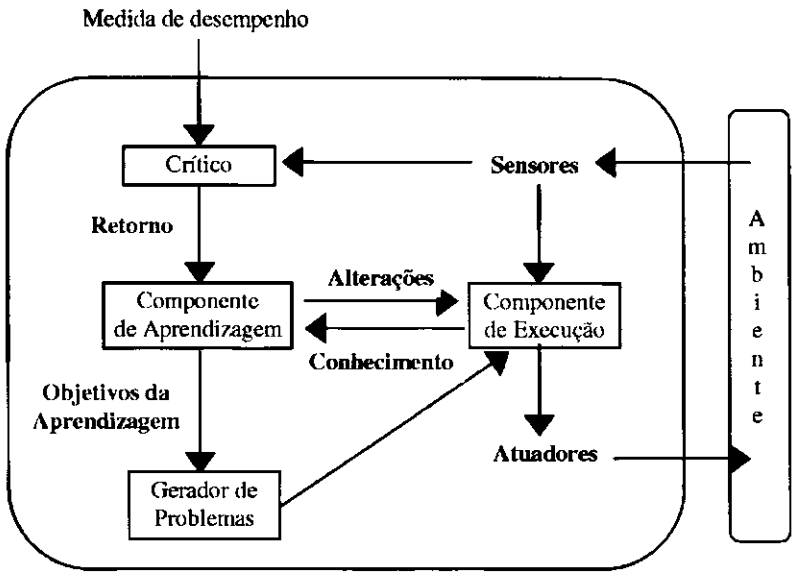


Figura 2.2.: Um modelo genérico de agente aprendiz [**RUSSELL 1995**].

- *Gerador de Problemas.* É o responsável por sugerir ações que confrontem o agente com novas experiências. Sem este componente o agente continuaria realizando as "melhores" ações, segundo o que ele já sabe. Mas a possibilidade de ele falhar mediante alguma percepção aumenta na medida que ele continua operando. A proposição de novas experiências pode levar o agente a detectar a fragilidade de seu conhecimento, através da sequência ação-crítica. Assim, o retorno do Crítico invoca o Componente de Aprendizagem na tentativa de incrementar o conhecimento do agente.

A aprendizagem pode, portanto, ser vista como o resultado da interação entre o agente e o mundo e como resultado da observação do processo de tomada de decisão do agente.

Retomando o caso de um sistema computacional jogador de xadrez, pode-se supor que ele esteja sendo ensinado por um professor da seguinte maneira: o agente dispõe de uma variável global, chamada *exemplos*, que armazena uma lista de pares do tipo *<percepção,ação>*. Uma percepção pode ser determinada pela configuração das peças sobre o tabuleiro, e uma ação pelo melhor movimento a realizar. Quando o agente se depara com uma situação que se identifica com uma das percepções conhecidas, ele escolhe a ação correspondente e a realiza. Em caso contrário, ele pode chamar um método de aprendizagem que, a partir dos exemplos, gere uma hipótese *h* que possa ser utilizada para escolher a ação a ser tomada.

De fato, muitas variações desse esquema podem ser utilizadas na construção de um agente aprendiz. Por exemplo, um raciocínio natural seria que, sempre que recebesse um novo par *<percepção,ação>* a ser acrescentado aos exemplos, o agente simplesmente tentasse alterar sua

hipótese corrente, ao invés de aplicar o método de aprendizagem ao novo conjunto de pares. Trabalhos como o de H. Bezerra [BEZERRA, 1995], porém, mostram a inconveniência de tal abordagem para métodos indutivos.

2.3 Inteligência e Racionalidade

As teorias sobre *racionalidade* foram inicialmente discutidas por H. Simon em economia [SIMON, 1955]. Em I.A. tais teorias definem as formas de interação que um agente mantém com seu ambiente [RUSSELL, 1997]. Como discutido ao longo deste capítulo, um agente pode ser visto como uma entidade que, entre outras coisas, percebe e age e, agir *racionalmente* significa agir de maneira a atingir objetivos de acordo com um conjunto de crenças. Observe-se a consistência entre este pensamento e o princípio da racionalidade abordado por Newell, como mostrado na seção 2.2.1. Assim, a I.A. pode ser encarada como a disciplina que se ocupa do estudo e da construção de agentes racionais [RUSSELL, 1995] e nesse contexto, a expressão *agente racional* pode ser empregada em lugar de *agente inteligente*.

Ainda para Russell, "um agente racional é aquele que faz a coisa certa". Este pensamento, no entanto, requer que se estabeleça um significado para "a coisa certa" no âmbito de interesse. Uma primeira sugestão é tomar como ação correta aquela que conduz o agente ao sucesso, dirigindo assim o foco do problema para a decisão de *como* e *quando* julgar este sucesso do agente. O *como* pode ser resolvido através de uma medida de desempenho que, tal como mostrada para os agentes aprendizes, é estabelecida por uma entidade externa ao agente para medir o sucesso de suas ações. Ressalta-se novamente a importância da avaliação das ações do agente por uma entidade externa pelo fato de que, caso esta avaliação ficasse a cargo do próprio agente, este poderia estar "tentado" a sempre julgar como bem sucedidas suas ações. Obviamente, não há como estabelecer uma medida de desempenho universal que seja adequada para a avaliação de qualquer agente, ficando esta medida dependente de vários fatores relacionados ao projeto do agente. A definição de *quando* avaliar a performance de um agente é também importante, devendo-se evitar julgamentos antecipados ou tardios. Dessa forma, a noção de racionalidade de um agente, em um dado instante, depende do seguinte:

(i) de uma medida de desempenho que defina seu grau de sucesso;

(ii) de tudo que o agente percebeu até o instante (sua seqüência de percepção);

(iii) do que o agente sabe sobre o ambiente e

(iv) das ações que o agente é capaz de realizar.

O agente racional *ideal* é definido como sendo aquele que, para cada seqüência de percepção, realiza a ação que maximiza sua medida de desempenho, de acordo com o seu conhecimento embutido.

Em [MÜLLER, 1987], encontra-se uma definição de agente racional como sendo "... um agente que busca efetuar as ações as mais pertinentes possíveis para atingir seus objetivos em função do conhecimento que ele possui de suas ações e do ambiente". A noção de pertinência de uma ação é considerada como sendo função:

(i) da contribuição da ação à satisfação de seus objetivos e

(ii) de um julgamento sobre a efetividade da ação do ponto de vista de seu custo, de suas conseqüências futuras, do custo do raciocínio sobre suas conseqüências possíveis, etc.

M. Wooldridge e N. R. Jennings [WOOLDRIDGE, 1995] colocam a racionalidade como sendo a suposição de que um agente irá agir de forma a atingir seus objetivos, sendo este comportamento limitado pelas crenças do agente.

Do exposto, considera-se neste trabalho um agente racional como sendo um sistema que:

(i) é capaz de construir e fazer evoluir seus próprios conhecimentos em um certo domínio,

(ii) é autônomo no sentido de que suas ações são determinadas pelo seu conhecimento passível de evolução,

(iii) procura agir de forma a atingir seus objetivos.

2.4 Discussão

> "*Um agente é uma entidade cujo estado é visto como consistindo de componentes mentais tais como crenças, capacidades, escolhas e compromissos.* [...] *O que torna qualquer componente de hardware ou software um agente é o fato de que alguém escolheu analisá-lo e controlá-lo nesses termos mentais.*" [SHOHAM, 1993]

> "*A noção de agente deve ser considerada como uma ferramenta para analisar sistemas, não uma caracterização absoluta que divide o mundo em agentes e não agentes.*" [RUSSELL, 1995].

Considere-se, como apresentado por Russell e Norvig [RUSSELL, 1995], um relógio como um agente simples. Em uma primeira observação, nota-se que o relógio faz a coisa certa: move os ponteiros de maneira apropriada. Mas se o seu proprietário resolve viajar do Brasil para a França, a

coisa certa seria que o relógio adiantasse cinco horas, mas observa-se que isso não acontece: falta ao relógio um equipamento de percepção, ou seja, não importa o que aconteça no ambiente, ele continua a operar da mesma maneira. Considere-se agora que o fabricante do relógio soubesse, ainda na fábrica, a data exata da viagem de seu futuro proprietário e adicionasse um dispositivo para adiantar o relógio na tal data. O relógio então faria a coisa certa, só que, neste caso, o mérito seria do fabricante, não do relógio. O que falta ao relógio, então, além da capacidade de perceber as alterações no ambiente? É necessário que ele possa, além de perceber, agir de forma coerente com suas percepções. É necessário ainda que suas ações sejam regidas não apenas pelo conhecimento que lhe foi embutido na sua construção, mas regidas por um conhecimento que ele próprio possa gerar. Falta-lhe autonomia.

Pode-se concluir que a noção de agentes é de grande relevância para o projeto de sistemas computacionais, pois permite conceber e analisar tais sistemas utilizando-se uma linguagem geralmente empregada para se falar de seres humanos. Cabe a cada projetista, no entanto, dentro de uma ampla gama de características como as que foram apresentadas neste capítulo, explorar no projeto de seu(s) agente(s) aquelas que mais lhe tenham a oferecer. No próximo capítulo, apresenta-se o agente racional SAID no qual baseia-se o sistema de apoio à descoberta proposto nesta dissertação.

Capítulo 3

O Agente Racional SAID

3.1 Introdução

Pelo exposto no Capítulo 2, um agente racional pode ser visto como um sistema que aparenta ao usuário ser capaz de raciocinar, manipulando interações, crenças e conhecimento incompleto, impreciso e errôneo. Neste capítulo, apresenta-se a organização do agente racional SAID[4], um sistema inteligente concebido para ter a capacidade de construir seu próprio conhecimento através da interação com um agente humano e explicar suas decisões. O agente racional SAID pode ser visto como um sistema de aquisição de conhecimento que integra:

(i) a Teoria Semi-Empírica (TSE) [SALLANTIN, 1991A, 1991B], que é um sistema de representação de conhecimento que provê os meios necessários à construção de raciocínios diante de conhecimentos incompletos e passíveis de erros;

(ii) mecanismos de aprendizagem, que provêm os meios necessários para a geração de conhecimento [VILLAREAL, 1989; LIQUIÈRE, 1990; FERNEDA, 1992B; CARVALHO, 1998]; e

(iii) o protocolo de aprendizagem MOSCA [REITZ, 1992], que provê os elementos necessários para o controle por diálogo entre os diversos agentes envolvidos na geração e evolução de conhecimento.

No todo, esta abordagem dá origem a um ambiente de aquisição de conhecimento cujos mecanismos de aprendizagem geram conhecimento passível de crítica.

Na Figura 3.1, mostra-se o agente racional SAID na condição de sistema de apoio à descoberta científica ao evidenciar a utilização dos mecanismos de inferência (abdução, indução e dedução) como metodologia empregada na construção de seu conhecimento. O agente pode ser visto como um sistema popperiano [SALLANTIN, 1997], já que bascia-se no princípio da refutabilidade para validar seu conhecimento. A descoberta, neste trabalho, é vista aqui como "... aquilo que não é ainda, em um instante determinado, apreendido pela modelagem em curso" [FERNEDA 1992A].

---

[4] SAID: Acrônimo para *Seulement Abduire, Induire et Déduire* (Somente Abduzir, Induzir e Deduzir).

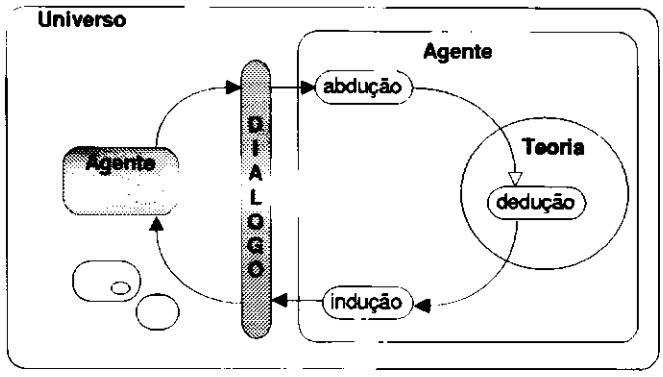


Figura 3.1: Agente Racional SAID.

3.2 Teorias Semi-Empíricas

As Teorias Semi-Empíricas (TSE) podem ser vistas, do ponto de vista da Inteligência Artificial, como uma forma de representação de conhecimento e, do ponto de vista da filosofia, como um sistema de raciocínio [SALLANTIN, 1991A]. Essa formulação de natureza filosófica é construída como princípio de modelos matemáticos e dos métodos que exploram esses modelos. As TSE definem como o conhecimento é formulado, experimentado e difundido, objetivando a construção de raciocínios a partir de conhecimentos, não supostamente exatos, mas capazes de evoluir através de uma interação com um usuário. Esta situação, que caracteriza as TSE, vai de encontro às formas de representação de conhecimento que permitem raciocínio apenas a partir de conhecimentos supostamente exatos e estáveis. Daí extrai-se também, segundo Sallantin, a denominação de *Semi Empírica*, já que, no sentido lakatosiano [LAKATOS, 1976], uma TSE não é nem completamente axiomática (não demanda conhecimento inicial), nem completamente empírica (é passível de evolução gradativa através da análise iterativa e interativa dos dados de entrada).

O modelo da lógica da descoberta científica adotado pelas TSE objetiva o estudo da prova (ou refutação) de uma conjectura para determinar o que intervém no exame da validade (dado o seu poder preditivo) da prova (ou refutação) e ainda o que intervém no exame da relevância (dado o seu poder explanatório) da prova (ou refutação). As TSE são uma forma de representação de conhecimento que expressam uma conjectura através de relacionamentos entre as sentenças de uma linguagem.

O agente racional responsável pela descoberta do conhecimento em um certo domínio deverá formular apenas conhecimento que possa ser criticado por outro agente: o especialista no domínio. Dessa forma, para conceber o agente racional, é necessário que se chegue a um acordo sobre as heurísticas que definem as formas do conhecimento. Estas formas permitirão a construção

de um sistema capaz de elaborar uma TSE, já que as formas é que determinam como as sentenças da linguagem serão interpretadas.

Na Figura 3.2 é apresentada uma taxonomia dos termos empregados para exprimir o conhecimento nas TSE. Esta taxonomia é baseada no trabalho de T. Addis [ADDIS, 1988], que retoma o modelo de conhecimento de C.S. Peirce [PEIRCE, 1958] e foi complementada por Sallantin [SALLANTIN, 1991A] com os termos utilizados na Lógica das Provas e Refutações. A taxonomia inclui:

- *Dados*, que representam o conhecimento de um agente racional. Este conhecimento é representado por expressões a serem geradas e que são passíveis de evolução. O conhecimento é expresso de uma das três formas:

  (i) *fatos*, que são sentenças para as quais um grau de validade pode ser atribuído;

  (ii) *hipóteses*, que são sentenças para as quais um grau de relevância pode ser atribuído e

  (iii) *heurísticas*, que definem as formas que os fatos e as hipóteses podem assumir;

- *Mecanismos* para a geração de conhecimento (*abdução*, que descobre novos fatos), para a organização deste conhecimento (*indução*, que sugere novas hipóteses e novas heurísticas) e para a propagação de restrições sobre este conhecimento (*dedução*, que determina as conseqüências lógicas do conhecimento);

- *Métodos*, relacionados às interações com um agente externo que desempenha o papel de criticar ou propor uma sentença a ser provada. Estes métodos examinam a adequação de conhecimentos tais como ser um *lema*, ser uma *objeção*, ser uma *prova*, etc.

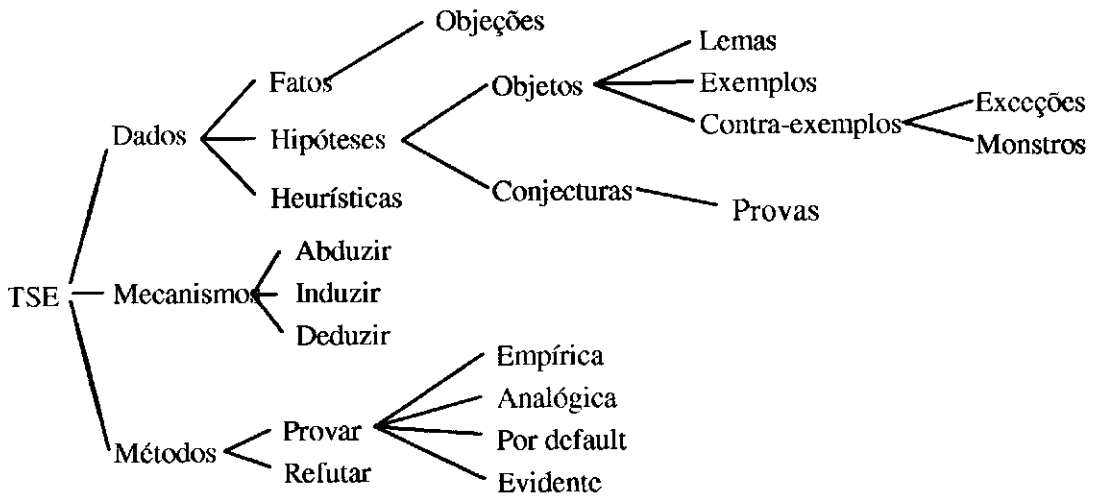


Figura 3.2: Termos que intervêm na formalização e na evolução do conhecimento pelas TSE.

A seguir são apresentadas definições sucintas para os demais termos utilizados na taxonomia das TSE. Definições mais detalhadas podem ser encontradas em [SALLANTIN, 1991A]. A formalização de alguns desses conceitos é mostrada na seção 3.3 a seguir.

(i) *Objetos* são enunciados de uma restrição particular que satisfaz um conjunto de fatos;

(ii) *Conjecturas* são enunciados de uma restrição particular que satisfaz um conjunto de objetos;

(iii) *Lemas* são enunciados conjuntivos que expressam fatos concomitantes; ou disjuntivos que expressam fatos excludentes;

(iv) *Prova* é uma decomposição de conjecturas em um conjunto de lemas; *Refutações* são enunciados de contra-exemplos;

(v) *Exemplos* são objetos que satisfazem a conjectura e os lemas da prova;

(vi) *Contra-exemplos* são objetos que não satisfazem a conjectura ou a prova;

(vii) *Objeções* são enunciados de diferenças entre exemplos que satisfazem a conjectura e objetos já criticados por essa conjectura;

(viii) *Exceções* são objeções aceitas por uma conjectura mas não por sua prova;

(ix) *Monstros* são objeções aceitas por uma prova mas não por sua conjectura;

(x) *Provas evidentes* são provas que usam argumentos construídos com dados iniciais;

(xi) *Provas por default* são provas dadas como argumentos alternativos na ausência de uma prova evidente ou construída pelo aprendiz;

(xii) *Provas empíricas* são provas que utilizam lemas obtidos por raciocínio empírico; e

(xiii) *Provas analógicas* são provas que utilizam lemas obtidos por raciocínio analógico.

Por exemplo, se a *conjectura* diz respeito a «ser um pássaro» e a *prova* é «voar», a avestruz é uma *exceção*, pois verifica a conjectura, ou seja, é de fato um objeto do conceito, mas não verifica a prova; a mosca é um *contra-exemplo*, uma vez que verifica a prova mas não a conjectura, e cachorro é um *monstro*, visto que não verifica nem a conjectura nem a prova.

Um conhecimento nas TSE é avaliado segundo os critérios de *validade* e *pertinência*. A validade é uma avaliação dos fatos produzidos pelas hipóteses. A validade dos fatos pode ser diretamente mensurada, por exemplo, através da determinação da freqüência da verificação de um fato por uma classe de objetos. A pertinência é uma avaliação da maneira pela qual as hipóteses produzem os fatos, refletindo o número de resultados provados, aceitos pelo pesquisador. As

variações das medidas associadas aos critérios de validade e pertinência, em função da complexidade da teoria são chamadas, respectivamente, de *interesse* e *simplicidade*.

Diversos trabalhos instanciam os conceitos da TSE: M. Py [PY 1992] aplicou-os na concepção de um agente racional que raciocina por analogia sobre problemas da Química Orgânica; M. Villareal, [VILLAREAL, 1989], M. Pingand [PINGAND, 1991], G. Gracy, [GRACY, 1993] e E. Mephu N'guifo [MEPHU, 1993] aplicaram tais conceitos em problemas do Genoma; e E. Ferneda [FERNEDA, 1992B] concebeu um agente racional em Geometria Euclidiana Plana.

3.3 Esquema Mental

Conforme foi mostrado, TSE é uma teoria que contém seu próprio mecanismo de evolução. Este mecanismo é baseado na idéia de que uma linguagem é um conjunto ordenado de valores verdade. Newell [NEWELL 1982] mostrou que o conhecimento pode ser entendido independentemente de uma linguagem de representação particular. Ou seja, é possível abstrair o conhecimento de sua representação. Para chegar a uma abstração, um agente racional deve ser capaz de associar objetos a sentenças e a outros objetos.

Uma Teoria Semi-Empírica compõe a maneira pela qual um agente racional representa seu conhecimento e, através da noção de *esquema mental* [AUBERT, 1990; SALLANTIN, 1991A], permite que o conhecimento do aprendiz evolua aprendendo e também revisando o que já aprendeu. Tal esquema mental possibilita a definição de objetos como interpretações de um conjunto de sentenças, assim como a definição de conjecturas como interpretações dos conjuntos de objetos.

Um esquema mental é uma tripla (L, C, Δ), onde:

- L é um conjunto de sentenças ordenado pela relação de ordem parcial $\leq_L$;
- C é um conjunto de valores de crenças ordenado pela relação de ordem parcial $\leq_C$, formando assim um reticulado e tendo um máximo (1) e um mínimo (?).
- Δ: $L \times L \rightarrow C$ é uma função de crença, definida como: se s_1 e s_2 são duas sentenças e $c = \Delta(s_1,s_2)$, então a sentença s_1 explica (com a crença c) a sentença s_2 no esquema mental.

Por exemplo, os enunciados s_1 = «ser um bom tenista» e s_2 = «morar bem» aparecem geralmente correlacionados. Isso pode ser enunciado por $\Delta(s_1,s_2) = \Delta(s_2,s_1)$. É claro que isso não significa que para se resolver o problema de moradia bastaria ensinar as pessoas a jogar tênis.

Uma *interpretação* sobre L é uma função crescente I: $L \rightarrow C$, tal que $(\forall s_1,s_2 \in L)(s_1 \leq_L s_2 \Rightarrow I(s_1) \leq_L I(s_2))$.

Por exemplo, o enunciado «ser um bom tenista» é sempre menos credível do que «ser um tenista».

Um *objeto mental* é uma interpretação O sobre L tal que $(\exists s \in L)(\forall x \in L)(O(x) = \Delta(s,x) \wedge O(s) = 1)$. Ou seja, uma interpretação particular onde uma sentença s_1 explica (com crença 1) todas as outras sentenças de L com crença igual a 1.

Por exemplo, «melhor tenista brasileiro atual» e «campeão de Roland Garros de 1997» são dois enunciados que designam um mesmo objeto mental.

O conjunto de objetos mentais é obtido a partir de um esquema mental.

Observações:

- A sentença s denota o objeto mental O. O_s é o objeto O cujo nome é a sentença s.
- Um mesmo objeto mental pode ser denotado por várias sentenças. Assim, $O_{s_1} = O_{s_2}$ significa que $(\forall x \in L)(O_{s_1}(x) = O_{s_2}(x))$.
- O conjunto de todos os objetos mentais do esquema mental é denotado Ω.
- $\{x \in L \mid \Delta(x,x) = 1 \wedge \forall s_1,s_2 \in L, s_1 \leq_L s_2 \Rightarrow \Delta(x,s_1) \leq_C \Delta(x,s_2)\}$ é o conjunto de todas as sentenças de L que denotam um objeto mental.
- Uma vez que cada objeto em Ω pode ser denotado por uma sentença em L e L é parcialmente ordenada, então o conjunto Ω é parcialmente ordenado. Portanto, um esquema mental utilizando Ω como linguagem pode ser construído.

Um *conceito mental* é uma interpretação K sobre Ω tal que $(\exists O_1 \in \Omega)(\forall O_2 \in \Omega)(K(O_1) = \Gamma(O_1,O_2))$. Ou seja, $\Gamma(O_1,O_2) = \Gamma(O_x,O_\alpha) = \Delta(x,\alpha)$, onde α denota o objeto O_2 e x denota o objeto O_1 : $(\exists \alpha \in L)(\forall O_x \in \Omega)(K(O_x) = O_x(\alpha) = \Delta(x, \alpha))$.

Observações:

- A sentença α denota o conceito mental K. K_α é um conceito K sse $K(O) = 1$.
- Um mesmo conceito mental pode ser associado a várias sentenças. Consequentemente,

$K_\alpha = K_\beta$ significa que $(\forall x \in L)(K_\alpha(x) = K_\beta(x))$.

- $\{x \in L \mid \Delta(x,x) = 1 \wedge \forall s_1,s_2 \in L,\ s_1 \leq_L s_2 \Rightarrow \Delta(s_1,x) \leq_C (s_2,x)\}$ é o conjunto de todas as sentenças de L que denotam um conceito mental.
- Um objeto mental O é um *exemplo* de um conceito mental K sse $K(O) = 1$.
- Um objeto mental O é uma *objeção* para o conceito mental K ssc $K(O) \neq 1$.
- Um *lema* é um objeto O_e, onde e também denota o conceito mental K_e.
- Um *fato mental* é uma sentença f tal que $(\exists O \in \Omega)(O(f) \neq ?)$.

3.4 Protocolo de Aprendizagem MOSCA

Teorias sobre aprendizagem a partir de exemplos [BOUCHERON, 1988] definem um ambiente de aprendizagem mínimo composto de um *aprendiz* comunicando-se com um *oráculo*. O protocolo que rege o diálogo, do ponto de vista de resolução de problemas, pode ser resumido como: o oráculo envia problemas resolvidos ao aprendiz e este último, ao receber tais problemas, reúne-os em uma *amostra*. Dada uma amostra, o aprendiz procura, em um *espaço de hipóteses*, a melhor *hipótese* em *adequação* à amostra. Hipótese, neste contexto, significa um procedimento de resolução de problema. O conceito de adequação é modelado como um critério a ser satisfeito, um critério de aprendizagem, que estabelece o que significa uma melhor hipótese em uma dada amostra. A hipótese selecionada é chamada *hipótese aprendida* (ou *conjectura*, segundo a TSE). Um espaço de hipóteses representa todo o conhecimento que um aprendiz pode obter e, em um dado instante, o seu conhecimento é um ponto nesse espaço de hipóteses. Assim, pode-se ter como parâmetros para o problema da aprendizagem:

(i) uma amostra contendo problemas resolvidos;

(ii) um espaço de hipóteses;

(iii) um critério de aprendizagem e

(iv) uma estratégia para percorrer o espaço de hipóteses.

Dois tipos de ruído aparecem durante a busca por uma hipótese aprendida:

(i) um problema resolvido pode ter sido erroneamente descrito e

(ii) a linguagem de descrição pode não ser suficientemente refinada, de forma a permitir a possibilidade de dois problemas distintos com descrições idênticas, terem, no entanto, soluções distintas.

Na prática, este ambiente mínimo pode ser implementado de forma que o sistema faça o papel de aprendiz e o especialista faça o papel de oráculo. Para iniciar o processo de aprendizagem, uma amostra deve ser construída.

De posse de uma amostra e uma vez elaborada uma hipótese aprendida, o aprendiz pode usar esta hipótese para resolver problemas propostos pelo especialista. Assim, de uma etapa de *aquisição*, passa-se a uma etapa de *exploração* do conhecimento. Nesta última etapa, o especialista é considerado pelo aprendiz como um *cliente*. A Figura 3.3 mostra o ambiente de aprendizagem mínimo composto por um aprendiz, um oráculo e um cliente.

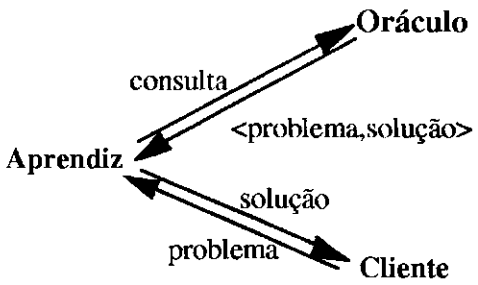


Figura 3.3.: Ambiente de aprendizagem mínimo.

Geralmente, a análise de uma solução produzida por um aprendiz não permite que o especialista julgue uma hipótese aprendida. Particularmente, quando o especialista está em uma situação de descoberta, ele não sabe caracterizar o que faz de uma hipótese uma boa hipótese. Por outro lado, raramente uma hipótese aprendida é diretamente passível de exploração, quase sempre devido à sua complexidade ou interpretação. Desta forma, é conveniente a introdução de um mecanismo de argumentação no aprendiz. Ou seja, a partir de uma solicitação do especialista, o aprendiz irá argumentar seus resultados [**REITZ, 1992**]. Desde que a argumentação seja apropriadamente construída para sua exploração pelo especialista, este estará indiretamente julgando a hipótese aprendida ao analisar as argumentações do aprendiz. Criticando uma argumentação, o especialista pressiona o aprendiz a revisar sua hipótese aprendida.

Do exposto, distinguem-se cinco papéis essenciais:

- um *oráculo* produz problemas resolvidos cujas soluções são irrefutáveis, formando uma amostra;
- um *aprendiz* obtém uma hipótese segundo uma relação de adequação com a amostra e é, ainda, capaz de argumentar suas soluções;
- um *cliente* propõe problemas ao aprendiz, que os resolve (no melhor caso);

- um *mestre* recebe argumentações do aprendiz e responde com uma crítica;
- uma *sonda* envia ao aprendiz problemas resolvidos (com soluções refutáveis) objetivando forçar o aprendiz a argumentar.

O esquema da Figura 3.4 resume o protocolo de aprendizagem MOSCA. A seguir, apresentam-se as especificações informais deste protocolo.

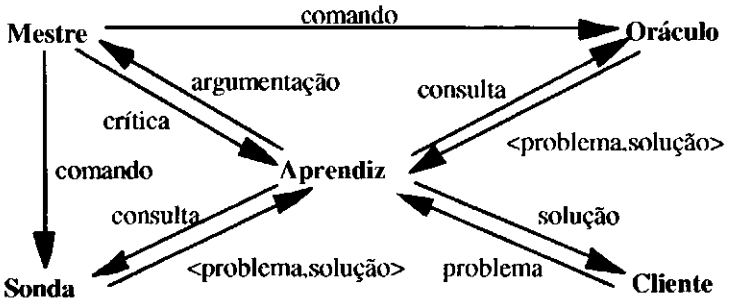


Figura 3.4.: Protocolo de aprendizagem MOSCA do ponto de vista de resolução de problemas.

O aprendiz recebe do oráculo problemas resolvidos. Esses problemas são armazenados e constituem a amostra. Cada transformação na amostra leva o aprendiz a revisar sua hipótese aprendida. Essa hipótese é extraída de um espaço de hipóteses e satisfaz a um critério de aprendizagem.

O aprendiz pode solicitar ao oráculo mais problemas resolvidos. Esta solicitação pode se dar através do envio de um sinal, quando a escolha do problema é deixada a critério do oráculo, ou compreende o enunciado do problema para o qual o aprendiz deseja a solução. Esta última forma de solicitação deve ser empregada com restrição, caso contrário, o aprendiz passará a sempre interrogar o oráculo ao invés de aprender.

Algumas funcionalidades para o aprendiz já aparecem:

- adicionar um problema resolvido à amostra,
- encontrar uma hipótese adequada à amostra e
- utilizar a hipótese para resolver problemas.

O aprendiz também recebe problemas resolvidos de um outro agente: a sonda. Contudo, a sonda pode enviar problemas com soluções errôneas. Assim, ainda que a hipótese aprendida não seja compatível com o problema resolvido recebido da sonda, o aprendiz poderá mantê-la. Ao receber problemas da sonda, o aprendiz compara a solução recebida com a sua solução (gerada pela aplicação da hipótese aprendida) e então envia uma argumentação ao mestre. Distinguem-se

dois tipos de argumentação: *explicação*, quando as duas soluções coincidem e *objeção*, quando as soluções diferem. A função do aprendiz que produz uma argumentação, dados um problema e uma solução, é chamada *argumentador*. Esta função é dependente da hipótese aprendida: qualquer mudança na hipótese pode modificar a função *argumentador*.

Após o envio de uma argumentação o aprendiz recebe uma crítica. As argumentações criticadas são memorizadas formando um *argumentário*. Distinguem-se dois tipos de críticas: *positiva* e *negativa*. Uma crítica negativa força o aprendiz a encontrar outra argumentação, se possível. Quando não é possível encontrar uma outra argumentação, uma primeira estratégia é o enfraquecimento da hipótese aprendida, excluindo o problema não passível de argumentação do domínio da hipótese. Desta forma, uma vez submetido o mesmo problema novamente ao aprendiz, este responderá com silêncio. Uma segunda estratégia é que o aprendiz faça uma nova solicitação ao oráculo, o que acarretará uma revisão sobre sua hipótese aprendida. Na prática, espera-se que todas as críticas enviadas ao aprendiz sejam positivas, o que caracterizará este aprendiz como pertinente aos olhos do mestre.

Procurou-se definir o problema da determinação de um argumentador como o problema de aprendizagem de uma hipótese: o argumentador é o resultado do processo de busca num espaço de argumentadores, busca esta restringida por um critério que define a adequação do argumentador com o argumentário e uma hipótese aprendida. Ou seja, um argumentário é para o mestre aquilo que a amostra é para o oráculo e o argumentador é para o argumentário aquilo que a hipótese aprendida é para a amostra. A Figura 3.5 resume estas idéias.

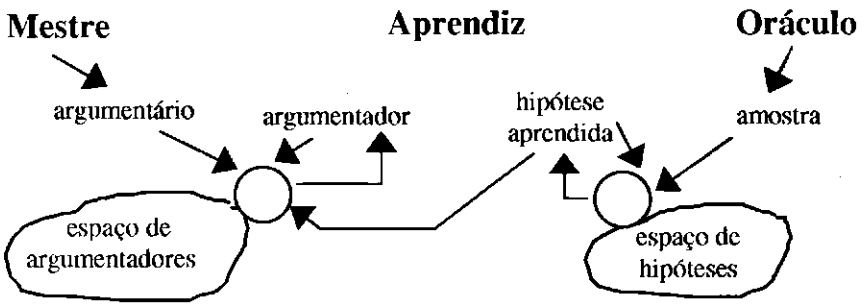


Figura 3.5.: Decomposição de uma aprendizagem em duas etapas: a produção de uma hipótese e de sua argumentação.

Para a argumentação, os parâmetros do problema a ser resolvido são, então, os seguintes:

- um argumentário, composto de argumentações criticadas;
- um espaço de argumentadores;

- um critério de adequação argumentador/argumentário/hipótese aprendida e
- uma estratégia para percorrer o espaço de argumentadores.

As funcionalidades relacionadas à argumentação são:

- adicionar uma argumentação criticada ao argumentário;
- procurar um argumentador e
- utilizar o argumentador para produzir uma argumentação.

Quando o aprendiz enfraquece sua hipótese aprendida (silênciará frente a alguns problemas), o mestre pode forçá-lo a revisar esta hipótese enviando ao oráculo um sinal para que este encaminhe ao aprendiz algum problema resolvido. Sem esta possibilidade, o aprendiz tornar-se-ia mais e mais silencioso a medida em que recebesse críticas negativas do mestre e não teria como modificar este comportamento.

Um cliente submete um problema ao aprendiz, que responde, no melhor caso, com uma solução para o problema proposto. O aprendiz pode argumentar sua solução (explicação), dependendo do tipo de aprendiz que se escolhe: sempre justificar o uso de uma hipótese, justificar soluções para algumas classes de problema, etc.

3.5 Sistema de Crenças

O sistema de crenças considerado baseia-se na noção de bi-reticulado de Ginsberg [GINSBERG, 1990]. Uma crença é um elemento do conjunto C, munido de duas relações de ordenação: $\leq_k$ e $\leq_t$, que interpretam-se, respectivamente, como *menos conhecido que* e *menos verdade que*. Pela relação $\leq_k$, C é um reticulado com *silêncio* como seu mínimo e com $\perp$ (contradição) como seu máximo. Pela relação $\leq_t$, C é um reticulado com *verdadeiro* como máximo e *falso* como mínimo. A crença $\perp$ é tal que *verdadeiro* $\leq_k \perp$ e *falso* $\leq_t \perp$, representando uma contradição.

O conjunto C de crenças considerado é

$$C = C_M \cup C_O \cup C_S \cup C_C \cup C_A,$$

onde:

C_M = {objetado, não-objetado, silêncio},

C_O = {verdade-O, falso-O},

determinar a crença esperada como solução do problema proposto. Em outras palavras, a solução consiste da associação de uma crença à sentença "o objeto em questão é um exemplo do conceito". A Figura 3.7 mostra o protocolo MOSCA do ponto de vista da associação de crenças a objetos.

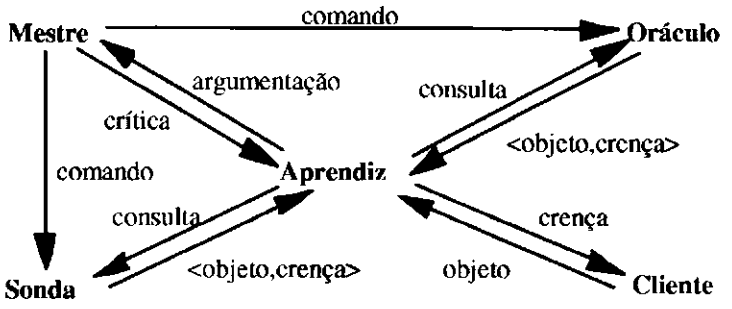


Figura 3.7: Protocolo MOSCA do ponto de vista da associação de crenças a objetos.

Teoricamente, conforme mostrou-se, uma conjectura é extraída de um espaço de conjecturas. Na prática, porém, este espaço não precisa ser construído, mas sim a conjectura, a partir de combinações lógicas de componentes elementares (os fatos). É necessário, então, que o aprendiz disponha de um mecanismo para a construção de tal conjectura.

3.6.1 Princípios de construção de uma conjectura

Um dos objetivos do aprendiz é, então, construir uma conjectura sobre um conceito. O oráculo envia exemplos e contra-exemplos (objetos associados a crenças) deste conceito ao aprendiz para que este último forme sua amostra. Uma vez que os objetos desta amostra são descritos por sentenças que podem ser decompostas, regularidades são extraídas da tal amostra de forma que o aprendiz possa construir regras da forma:

quando o meu objeto satisfizer à regularidade R, sua crença na sentença S será ...

Podem existir muitas regras desse tipo e a hipótese aprendida (a conjectura) será uma composição dessas regras. Cabe lembrar que, no contexto deste trabalho, a extração das regularidades é uma abdução, ao passo que a construção das regras caracteriza uma indução.

3.6.2 Princípios de aplicação de uma conjectura

A aplicação das regras aprendidas é relativamente simples: quando o aprendiz recebe um objeto, a crença a ser associada é determinada em dois passos:

Passo1: primeiro, o aprendiz aplica o conjunto de regras que formam a conjectura, associando

uma crença para cada regra e

Passo 2: a seguir, agrega todos os valores de crença encontrados em uma única crença resultante e a associa ao objeto.

Para a agregação dos valores das várias crenças em uma única, podem-se aplicar, dentre outras, as seguintes abordagens:

(*i*) *lógica clássica*: se o conjunto de valores de crença é {*verdadeiro*, *falso*}, a agregação pode ser feita tanto por conjunção (se uma das regras gera o valor *falso*, a agregação também o será) como por disjunção (se uma das regras gera o valor *verdadeiro*, a agregação também o será).

(*ii*) *lógica majoritária*: dois limites permitirão regular a natureza mais conjuntiva ou mais disjuntiva da agregação das regras aprendidas. O primeiro limite indica o número mínimo de regras que devem ser verdadeiras para que a agregação também o seja. O segundo limite indica o número mínimo de regras que devem ser falsas para que a agregação também seja falsa. Como esses limites não são necessariamente complementares, para alguns casos o resultado não será verdadeiro nem falso, será *silêncio*.

3.6.3 Princípios de construção de uma argumentação

Foram mostrados os princípios de construção e aplicação de uma conjectura. Serão mostrados agora mecanismos que permitem a revisão do conhecimento. Com o protocolo MOSCA, esta revisão pode acontecer de duas maneiras:

- ou novos exemplos (ou contra-exemplos) são produzidos pelo oráculo e enviados ao aprendiz, fazendo com que a conjectura aprendida seja questionada pelo aparecimento de novas regularidades,
- ou o mestre critica argumentações recebidas do aprendiz, o que, indiretamente, questiona a conjectura.

O primeiro caso diz respeito ao estudo da natureza incremental dos mecanismos de detecção de regularidades na amostra. Esse problema clássico da aprendizagem é grande consumidor de recursos de cálculo e portanto é preferível que não seja utilizado como forma de revisão. Adotou-se o caso da argumentação.

Antes, porém, de se examinar o efeito de uma crítica sobre o conhecimento aprendido, mostra-se como uma argumentação pode ser construída. Segundo o protocolo MOSCA, como

mostrado, distinguem-se duas formas de argumentação: a *explicação* e a *objeção*.

Uma explicação é dada pelo aprendiz quando a crença por ele gerada para um objeto recebido é a mesma que a crença enviada pela sonda para o mesmo objeto ou ainda quando o aprendiz justifica uma crença enviada ao cliente que lhe submeteu um objeto. No âmbito das TSE, uma explicação é um conjunto mínimo de regras que justificam o resultado obtido.

Uma objeção é gerada quando a crença produzida pelo aprendiz não coincide com aquela enviada pela sonda, em se tratando do mesmo objeto. Nas TSE, uma objeção **[BARBOUX, 1990]** é formada pelo conjunto de regras suficientes para objetar a crença proposta pela sonda.

3.6.4 Princípios de exploração de uma crítica

Para cada argumentação enviada ao mestre, o aprendiz recebe uma crítica. Como visto anteriormente, críticas distinguem-se entre positivas e negativas. Se recebe uma crítica negativa, o aprendiz deve fornecer outra argumentação, se possível. Caso isso não seja possível, a saída mais simples é o enfraquecimento da conjectura e a exclusão do objeto do domínio desta conjectura. Nesse caso, uma vez novamente submetido o objeto, a resposta do aprendiz será o silêncio.

Quando o aprendiz recebe uma crítica negativa sobre uma argumentação, ele pode classificar as regras aprendidas da seguinte maneira:

- regras que não foram utilizadas em argumentação,
- regras que foram utilizadas em argumentação, negativamente ou positivamente criticadas.

Chama-se *lema* uma regra utilizada em argumentações cujas críticas recebidas tenham sido, a princípio, sempre positivas. Esse tipo de regra tem um papel fundamental, já que mostra que o mestre reconheceu seu poder explicativo. Na verdade, essas regras são termos de uma linguagem que foram admitidas pelas duas partes (mestre e aprendiz), servindo de suporte ao diálogo.

A divisão das regras em lemas e não-lemas sugere duas estratégias para determinar o melhor conjunto de regras a serem retidas para a construção de argumentações:

- se o aprendiz escolhe preferencialmente construir uma argumentação composta de um número máximo possível de lemas, ele correrá o risco de receber uma crítica negativa e colocar em questão a linguagem de comunicação que até então compartilhava com o

mestre;

- o aprendiz pode, alternativamente, escolher também regras que não são lemas para construir sua argumentação. Nesse caso ele objetiva ampliar a sua linguagem de comunicação.

Na prática, o aprendiz pode adotar a segunda estratégia para primeiramente construir uma linguagem de comunicação e, depois então utilizar a primeira estratégia.

Do ponto de vista computacional, pode-se imaginar um contador associado a cada regra, recebendo como valor inicial $-\infty$ (menos infinito). Logo que uma regra é utilizada pela primeira vez seu contador recebe o valor *zero* e é incrementado de um a cada crítica negativa e decrementado de um a cada crítica positiva (devendo ser mantido o valor *zero* como limite mínimo). Um lema é uma regra cujo contador tem valor *zero*. Os contadores são utilizados, então, pelo aprendiz para ordenar as regras na hora de escolhê-las para formar uma argumentação.

A manipulação das argumentações gera alguns problemas para os quais ainda não se tem respostas definitivas:

- quando revisando a conjectura, o que fazer com o argumentário?
- Quando da exaustão das argumentações, quando o mestre sempre critica negativamente, o aprendiz deve reconsiderar a conjectura, sobretudo quando esta mantém uma perfeita relação de adequação com a amostra?
- A argumentação criticada coloca em questão o argumentador ou os argumentos que compõem a argumentação?

3.7 Discussão

No Capítulo 2, mostrou-se um modelo genérico proposto por S. Russell e P. Norvig [RUSSELL 1995] que pode ser utilizado como base para o projeto de um agente aprendiz. Mostrou-se ainda a noção de racionalidade associada à noção de agentes. Neste capítulo, foram apresentados os elementos necessários à construção do Agente Racional SAID na condição de Sistema de Apoio à Descoberta Científica, ao evidenciar a utilização dos mecanismos de inferência (abdução, indução e dedução) como abordagem empregada na construção de seu conhecimento. Um dos elementos componentes de SAID, o protocolo de comunicação MOSCA, provê uma forma de cooperação entre especialista e sistema, através da qual, torna-se possível a construção de teorias passíveis de evolução.

Assim, SAID pode ser visto como um refinamento do modelo genérico de agente aprendiz proposto por S. Russell e P. Norvig (Figura 3.8) e seus componentes podem ser descritos sucintamente como:

- *Componente de Execução*: responsável por aplicar sua conjectura a um objeto enviado pelo cliente e obter uma crença, que será retornada ao cliente.
- *Componente de Aprendizagem*: responsável por gerar uma conjectura (pelos mecanismos de abdução e indução), a partir da amostra obtida do oráculo e fazer evoluir esta conjectura, a partir de críticas recebidas do mestre.
- *Crítico* (mestre): responsável por, através do envio de críticas, avaliar o desempenho do agente. No caso de SAID, o crítico não julga diretamente a ação realizada pelo agente, mas sua justificativa. Dessa forma, além da crença enviada ao cliente, o aprendiz é capaz de produzir também uma explicação.
- *Gerador de Problemas* (sonda): sugere objetos associados a crenças para o Componente de Execução para que produza uma crença, a confronte com aquela recebida e argumente.

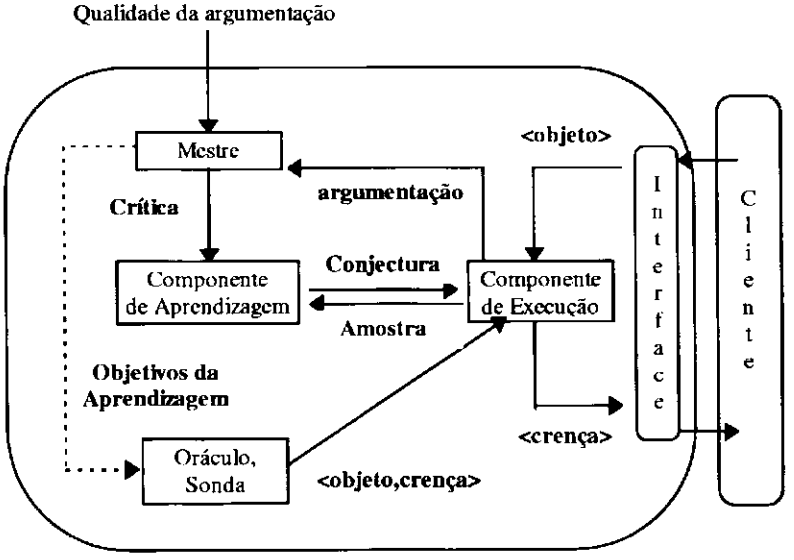


Figura 3.8: Adaptação de SAID ao modelo genérico de Agente Aprendiz de S. Russell e P. Norvig.

Considerando-se este conjunto de componentes, além do que foi mostrado ao longo deste capítulo, apresenta-se, no próximo capítulo, a modelagem de um sistema de apoio à descoberta concebido de acordo com estes componentes.

Capítulo 4

Uma Modelagem para um Sistema de Apoio à Descoberta

4.1 Introdução

No Capítulo 3, foram mostrados os elementos que compõem o Agente Racional SAID, a saber:

(i) as Teorias Semi-Empíricas, através da quais se representa o conhecimento do agente e se permite a evolução desse conhecimento;

(ii) o Esquema Mental, que formaliza tal conhecimento e sua evolução e

(iii) os elementos do protocolo MOSCA, que se comunicam e exibem comportamentos em função dessas comunicações na busca por uma solução para um problema.

Neste capítulo, tais elementos serão utilizados como base para a modelagem de um sistema de apoio à descoberta, o sistema SAID (agora significando *System for Aiding Discovery* - Sistema de Apoio à Descoberta). Tal modelagem é o ponto de partida para a especificação formal que será posteriormente construída.

Este capítulo é organizado da seguinte maneira: na seção 4.2 apresenta-se o sistema através de sua interação com o ambiente. A seguir, na seção 4.3, são definidos os requisitos aos quais, com base no protocolo MOSCA, a especificação proposta deverá atender. A seção 4.4 traz uma breve discussão sobre o paradigma de modelagem adotado. A metodologia adotada, fundamentada sobre este paradigma, é apresentada na seção 4.5. As seções 4.6 e 4.7 trazem, respectivamente, o modelo de objetos e parte do modelo dinâmico, que, como será mostrado, são visões distintas e complementares do sistema, segundo a metodologia que se utiliza. Finalmente, uma discussão conclui o capítulo na seção 4.8.

4.2 SAID e o Ambiente

A Figura 4.1 mostra a interação de SAID com o ambiente. Observa-se nesta figura que um

editor de objetos do domínio interage com o sistema. Este editor deverá prover ao usuário as duas funcionalidades seguintes:

(i) a edição do domínio, que definirá os termos da linguagem utilizados no domínio e a organização desses termos, e

(ii) a edição de objetos no âmbito desse domínio, que permitirá a construção de instâncias de conceitos do domínio, através dos termos da linguagem definidos para o tal domínio.

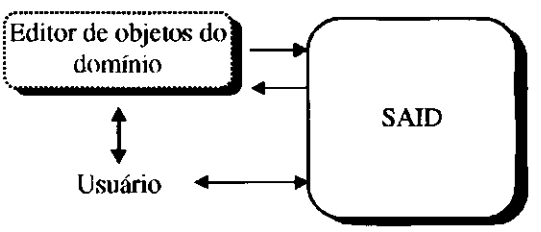


Figura 4.1: Interação Ambiente ↔ SAID.

O modelo de representação de conhecimento adotado pelo editor para representar o domínio e os objetos deve ser compatível com a forma de representação adotada por SAID. Na concepção inicial do sistema SAID, o sistema SAID-Cabri [FERNEDA, 1992A; FERNEDA, 1992B], utilizou-se a linguagem *Objlog* [DUGERDIL, 1988; FERNEDA 1996] como linguagem de representação de conhecimento. Tal forma de representação, que utiliza a linguagem de classes e objetos, restringe os tipos de domínio que SAID é capaz de manipular àqueles passíveis de representação sob forma estruturada. Naquela concepção, o domínio considerado foi a Geometria Euclidiana Plana e utilizou-se como ambiente do domínio o sistema Cabri-géomètre [BAULAC, 1990].

A interação Editor → SAID da Figura 4.1 traduz-se tanto na escolha das estruturas de classe a serem consideradas para a sessão futura em SAID como dos objetos a serem considerados.

A interação SAID → Editor encerra, juntamente com a interação no sentido contrário, a noção de *equitraduzibilidade* [HAMMOUDI, 1996A, 1996B]. Segundo essa noção, pode-se entender tal interação da seguinte maneira: em alguns momentos específicos durante uma sessão no sistema quando este faz referência a determinado objeto, dever-se-ia permitir ao usuário dispor deste objeto através do editor do domínio. A construção do editor, porém, está fora do escopo deste trabalho.

A seguir, são apresentadas as funcionalidades que o usuário deverá ser capaz de realizar em sua interação com SAID (interação Usuário ↔ SAID). Apresentam-se também os comportamentos que o sistema deve exibir em resposta a tais funcionalidades.

4.3 Especificação de Requisitos

Tendo como ponto de partida o protocolo de aprendizagem MOSCA, os elementos que compõem o sistema SAID deverão atender a requisitos tanto de comportamento como de comunicação. Tais requisitos são mostrados a seguir.

4.3.1 Cliente

Comportamento

R1. Selecionar objeto. O Cliente deve ser capaz de escolher um objeto previamente editado (no editor de objetos do domínio) para ser enviado ao Aprendiz.

Comunicação Cliente → Aprendiz

R2. Cliente envia objeto ao Aprendiz. Tendo selecionado um objeto, o Cliente deve ser capaz de enviá-lo ao Aprendiz.

4.3.2 Aprendiz

Comportamento

R3. Construir conjectura. De posse de uma amostra recebida do Oráculo, o Aprendiz deverá construir sua conjectura, utilizando mecanismos de aprendizagem. O aprofundamento destes mecanismos foge à abrangência deste trabalho[5].

R4. Procurar objeto na amostra. Ao receber um objeto do Cliente ou da Sonda, o Aprendiz primeiramente deverá procurar na amostra o objeto recebido na tentativa de encontrar a crença associada a este objeto.

R5. Aplicar conjectura a objeto. Não encontrando um objeto na amostra, o Aprendiz deverá aplicar sua conjectura ao objeto e daí obter uma crença.

R6. Construir argumentação. O Aprendiz deve ser capaz de construir uma argumentação quando aplicar sua conjectura a um objeto que lhe foi submetido. Existe uma situação onde a conjectura não precisa ser aplicada e ainda assim uma argumentação é disponibilizada: quando o Aprendiz conhece um objeto idêntico (o objeto faz parte de sua amostra), está-se em uma situação de *prova por evidência*.

R7. Explorar crítica. Ao receber uma crítica negativa, o Aprendiz deverá gerar uma outra argumentação, se possível. Se não for possível, ele deverá enfraquecer a conjectura.

Uma crítica positiva deverá provocar o fortalecimento da conjectura.

R8. Adicionar um conjunto de problemas resolvidos à amostra. Tendo recebido do Oráculo uma amostra (que pode conter apenas um problema resolvido), o Aprendiz deverá ser capaz de formar/reformular uma amostra. Sempre que receber uma amostra, o Aprendiz deverá questionar o usuário sobre manter ou desprezar a amostra anterior (se houver uma).

Comunicação Aprendiz → Oráculo

R9. Aprendiz solicita amostra. Estando *pronto* para iniciar um processo de aprendizagem, o Aprendiz envia ao Oráculo uma mensagem indicando que precisa de uma amostra. O Aprendiz estará pronto quando o ambiente no qual se dará a aprendizagem tiver sido definido. Esta definição envolve alguns dos requisitos mostrados a seguir em "Comportamento de usuário".

Comunicação Aprendiz → Mestre

R10. Aprendiz envia argumentação ao Mestre. O Aprendiz deve, sempre que associar uma crença a um objeto, tentar disponibilizar uma argumentação ao Mestre.

Comunicação Aprendiz → Sonda

R11. Aprendiz solicita problemas resolvidos à Sonda. Estando *pronto* para iniciar o processo de revisão, o Aprendiz deve enviar uma mensagem à Sonda solicitando um problema resolvido. O Aprendiz estará pronto para iniciar o processo de revisão, tão logo tenha construído sua conjectura a partir da amostra recebida do Oráculo.

Comunicação Aprendiz → Cliente

R12. Aprendiz envia crença ao Cliente. Tendo recebido um objeto do Cliente, o Aprendiz deverá obter uma crença resultante e disponibilizá-la ao Cliente.

4.3.3 Oráculo

Comportamento

R13. Construir amostra. O Oráculo deve ser capaz de selecionar objetos (previamente editados no editor de objetos do domínio) e associá-los às suas crenças (classificando-os como *exemplo* ou *contra-exemplo* do conceito considerado).

Comunicação Oráculo → Aprendiz

R14. Oráculo envia amostra ao Aprendiz. Tendo construído a amostra, o Oráculo deve ser

---

[5] Diversos trabalhos apresentam tais mecanismos [FERNEDA, 1992B; CARVALHO, 1997]. J. N. Carvalho estudou este tema com mais profundidade [CARVALHO, 1998].

capaz de enviá-la ao Aprendiz.

4.3.4 Sonda

Comportamento

R15. Selecionar objeto e crença. A Sonda deve ser capaz de selecionar um objeto (previamente editado no editor de objetos do domínio) e associá-lo a uma crença (indicando se ele é exemplo ou contra exemplo do conceito para o qual a aprendizagem é desejada).

Comunicação Sonda → Aprendiz

R16. Sonda envia problema resolvido ao Aprendiz. Tendo selecionado um objeto e o associado a uma crença, a Sonda deve ser capaz de enviar ao Aprendiz o par *<objeto, crença>*.

4.3.5 Mestre

Comportamento

R17. Construir crítica. A partir de uma argumentação recebida, o Mestre deve ser capaz de construir uma crítica, que consiste da associação de um parâmetro a cada hipótese da argumentação recebida.

Comunicação Mestre → Aprendiz

R18. Mestre envia crítica ao Aprendiz. Tendo construído sua crítica, o Mestre deverá ser capaz de enviá-la ao Aprendiz.

Comunicação Mestre → Oráculo

R19. Mestre envia sinal ao Oráculo. Uma vez detectado o enfraquecimento da conjectura (consequência do envio de uma crítica negativa), o Mestre deve ser capaz de enviar uma mensagem ao Oráculo indicando que este envie novo conjunto de problemas resolvidos ao Aprendiz.

Comunicação Mestre → Sonda

Segundo as especificações do protocolo MOSCA, esta comunicação acontece quando fica a cargo do Mestre determinar o tipo de problema que a Sonda deve enviar ao Aprendiz. Neste trabalho, resolveu-se não modelar esta comunicação, assumindo-se que a Sonda é capaz de escolher um problema adequado a enviar ao Aprendiz.

4.3.6 O usuário

Os papéis previstos para as entidades participantes do protocolo MOSCA foram cobertos pelos requisitos apresentados até aqui. Há, porém, algumas outras tarefas que devem ser realizadas pelo usuário especialista que fornecem informações indispensáveis para o processo de aprendizagem. Outras tarefas devem ainda prover a realização de operações usuais em sistemas computacionais. Esse conjunto de tarefas é apresentado a seguir.

R20. Gravação de ambiente de aprendizagem. A comunicação Cliente → Aprendiz, como foi mostrado, possibilita o envio de um objeto ao Aprendiz para que este associe uma crença ao objeto. Foi também mostrado que esta crença atribui um significado à sentença "objeto é exemplo de conceito". *Conceito* é então representado pela conjectura (Hipótese Aprendida) corrente do Aprendiz. O que ainda não foi mostrado é como será oferecida ao usuário a possibilidade de interromper e retomar posteriormente a aprendizagem de um conceito. Uma das operações que devem prover esta possibilidade é a *gravação* de um ambiente de aprendizagem, que deverá armazenar, entre outras coisas, a amostra e a Hipótese Aprendida (se já existirem) para uso posterior.

R21. Escolha de ambiente de aprendizagem. Através desta operação, o usuário deve ser capaz de selecionar o ambiente de aprendizagem que deseja manipular, seja um ambiente previamente gravado, seja um novo ambiente de aprendizagem. No caso de um ambiente existente, o sistema deve restaurar a situação imediatamente anterior à última operação de gravação realizada pelo usuário, para que este possa continuar o processo de aprendizagem exatamente de onde parou.

R22. Escolha de domínio. A aprendizagem em um ambiente acontece segundo os termos da linguagem e a organização desses termos. Dessa forma, o usuário deve ser capaz de escolher um domínio (previamente editado no editor de domínio).

R23. Filtragem de domínio. A escolha de um domínio sobre o qual a aprendizagem deverá se dar implica que deverão estar disponíveis ao Aprendiz todos os termos da linguagem definidos pelo domínio. Pode acontecer, entretanto, que, para a aprendizagem de um determinado conceito (dentro do domínio) seja interessante apenas um subconjunto de tais termos. Dessa forma, o usuário deve ser capaz de filtrar o domínio escolhido, selecionando apenas os termos da linguagem que intervêm na aprendizagem do conceito desejado.

R24. Definição de parâmetros de aprendizagem. Os mecanismos de aprendizagem (abdução, indução e dedução) são regidos por parâmetros que controlam a geração e aplica-

ção das regras. O usuário deve ser capaz de definir esses parâmetros segundo seu conhecimento do domínio.

4.4 O Paradigma Adotado

A análise e o projeto orientados a objeto são uma forma de se pensar sobre problemas usando-se modelos organizados em torno de conceitos do mundo real [RUMBAUGH, 1991]. A construção fundamental é o *objeto*, que combina estruturas de dados e comportamento numa única entidade. Um sistema computacional organizado como um conjunto de tais objetos que interagem é dito orientado a objeto. A adoção da abordagem da Orientação a Objeto deve-se ao fato de que busca-se modelar um sistema composto de entidades cooperantes.

O desenvolvimento orientado a objeto, se tomado como as duas primeiras fases do ciclo de vida de um software (análise e projeto), é um processo conceitual independente de uma linguagem de programação particular. Assim, a essência do desenvolvimento é a identificação e a organização dos conceitos do domínio da aplicação. Este capítulo vai ao encontro da identificação e organização dos conceitos em torno dos elementos para a construção do sistema SAID, tendo como ponto de partida os componentes do Agente Racional SAID.

Diversas metodologias baseadas no paradigma Orientação a Objeto podem ser encontradas na literatura [HUTT, 1994]. Neste trabalho, adotou-se a metodologia OMT[6] [RUMBAUGH, 1991], que, através dos seus três modelos, proporciona diferentes perspectivas de se visualizar um sistema. Desde sua concepção inicial, OMT vem se consagrando como metodologia de Engenharia de Software de mérito reconhecido pela comunidade científica. Esta metodologia é apresentada na seção 4.5 a seguir.

4.5 A Metodologia OMT

Em [RUMBAUGH, 1991], a palavra *modelo* é vista em duas dimensões: como um *estágio do desenvolvimento* (análise, projeto ou implementação) e como uma visão do sistema (um dos chamados *três modelos*: *objeto*, *dinâmico* ou *funcional*). Rumbaugh considera ainda que o significado desta palavra fica claro segundo o contexto onde ela é utilizada.

Sucintamente, pode-se dizer que a metodologia OMT consiste da construção dos modelos refletindo o domínio da aplicação e da posterior adição de detalhes de implementação a estes modelos durante o projeto.

4.5.1 Estágios do Desenvolvimento

A metodologia OMT envolve os seguintes estágios: *Análise*, *Projeto do Sistema*, *Projeto dos Objetos* e *Implementação*.

Genericamente falando, a análise proporciona a construção de modelos a partir da situação no mundo real. No caso particular deste trabalho, pode-se dizer que a situação que se modela não é um problema do mundo real típico, mas uma situação do mundo computacional. Em outras palavras, a existência do computador é a própria razão de ser do sistema, já que se está modelando uma situação de aprendizagem onde o aprendiz não é um agente humano, mas um agente artificial.

Durante a fase de projeto do sistema, o projetista deve tomar decisões de alto nível sobre a arquitetura geral deste sistema. Assim, acrescentam-se aos resultados da análise as partes necessárias para que o domínio do problema possa ser visto como um sistema computacional.

Para o projeto dos objetos, é construído um modelo baseado nos resultados da análise mas contendo detalhes de implementação. Neste estágio do desenvolvimento, a atenção é voltada para o detalhamento das estruturas de dados e dos algoritmos necessários à implementação de cada classe.

As classes e relacionamentos deverão ser traduzidos em uma linguagem de programação durante a implementação. A programação pode ser considerada uma parte mínima e um tanto mecânica do ciclo de desenvolvimento desde que as decisões importantes tenham sido tratadas durante o projeto do sistema e dos objetos. Neste trabalho, esforços são destinados no sentido de se prover uma especificação que faça valer essa idéia.

4.5.2 Os Modelos

A metodologia OMT utiliza três tipos de modelos para descrever um sistema: o *modelo de objetos*, que descreve os objetos do sistema e seus relacionamentos; o *modelo dinâmico*, que descreve as interações entre os objetos do sistema e o *modelo funcional*, que descreve as transformações nos dados do sistema. Cada modelo pode ser aplicado durante todos os estágios do desenvolvimento, incorporando detalhes de implementação de acordo com o progresso deste desenvolvimento.

O *modelo de objetos* descreve a estrutura estática dos objetos no sistema e seus relacionamentos, através do diagrama de objetos. Um diagrama de objetos é um grafo cujos nós são

[6] do inglês, *Object Modeling Technique*

classes e cujos arcos são relacionamentos entre classes.

O *modelo dinâmico* descreve os aspectos do sistema que mudam com o tempo. Este modelo é usado para especificar e implementar os aspectos de *controle* do sistema. O modelo dinâmico contém *diagramas de estado*, que são grafos cujos nós são estados e cujos arcos são *transições* entre os estados. Estas transições são causadas pelos *eventos*.

O *modelo funcional* descreve as transformações sobre os valores dos dados do sistema. Um modelo funcional tradicional contém diagramas de fluxo de dados (DFDs). Um DFD é um grafo cujos nós são processos e cujos arcos são fluxos de dados.

Neste trabalho, para evitar uma mistura de abordagens (estruturada e Orientada a Objetos), o modelo funcional não foi utilizado.

4.6 Modelo de Objetos

O modelo de objetos descreve a estrutura dos objetos no sistema: suas identificações, seus relacionamentos com outros objetos, seus atributos e suas operações. Objetiva-se com a construção do modelo de objetos capturar os conceitos importantes para a aplicação. Este modelo é representado graficamente através de diagramas de objetos contendo classes de objetos. As classes são organizadas em hierarquias que compartilham estruturas e comportamentos comuns e são associadas a outras classes.

4.6.1 O Domínio do Problema

Uma primeira tentativa de identificação de classes para um modelo de objetos refletindo o domínio do problema gerou o diagrama da Figura 4.2.

Utilizou-se para a construção do diagrama de objetos a ferramenta *Playground*, que implementa a notação de P. Coad. Um resumo desta notação é mostrado no *Anexo A*. As notações de P. Coad e J. Rumbaugh são equivalentes para o caso do Modelo de Objetos e dos cenários no Modelo Dinâmico. Para ilustrar esta equivalência, mostram-se, no *Anexo B*, as classes no diagrama da Figura 4.2, representadas segundo a notação de J. Rumbaugh.

Para P. Coad, *assunto* é um agrupamento de classes do diagrama de objetos [COAD, 1993]. Esta noção será empregada para guiar a exposição do modelo de objetos.

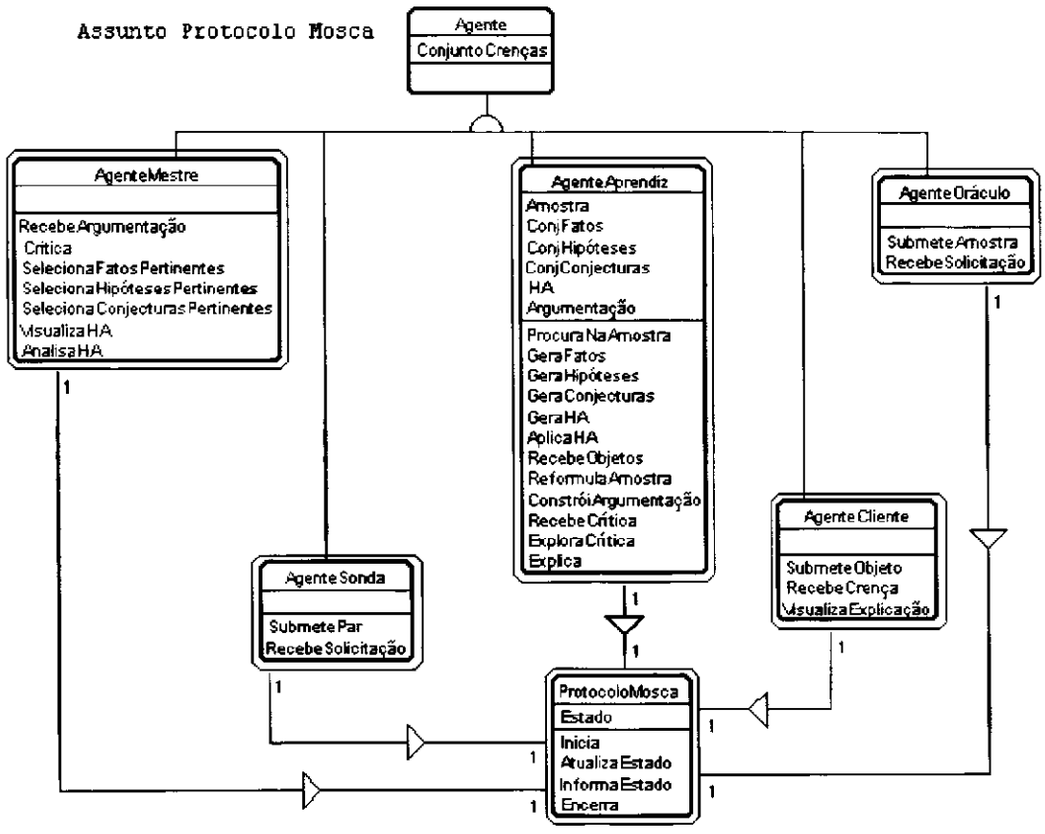


Figura 4.2: Assunto Protocolo Mosca do Modelo de Objetos (Página 1).

4.6.1.1 Assunto Protocolo Mosca

P. Reitz [REITZ, 1992] discute informalmente as noções de *agente* e de *mundo*. Segundo ele: "*Um agente é caracterizado por, pelo menos, uma propriedade: seu comportamento.*" e "*Um mundo é um agrupamento de agentes.*" e ainda: "*Todo comportamento de um agente de um mundo transforma este mundo*". O que se propõe como um primeiro modelo de objetos é um reflexo da estrutura do protocolo MOSCA. O que se toma como *mundo* aqui é o próprio protocolo, enquanto *agente* é cada um dos elementos que compõem esse mundo.

A caracterização para agentes inteligentes apresentada no Capítulo 2 continua válida no contexto mais amplo do Agente Racional SAID. Observa-se ainda na Figura 4.2 que cada classe modelando um agente tem um comportamento que reflete o papel deste agente no mundo do protocolo. O quadro da Figura 4.3 mostra os requisitos aos quais os métodos dessas classes atendem.

Classe	Método	Requisito
AgenteCliente	*SubmeteObjeto*	R1
	RecebeCrença	R12
AgenteAprendiz	*RecebeObjetos*	R2, R14, R16
	ProcuraNaAmostra	R4
	AplicaHA	R5
	ConstróiArgumentação	R6
	ReformulaAmostra	R8
	GeraHA	R3
	RecebeCritica	R18
	ExploraCrítica	R7
AgenteOráculo	*RecebeSolicitação*	R9
	SubmeteAmostra	R13
AgenteSonda	*RecebeSolicitação*	R11
	SubmetePar	R15
AgenteMestre	*RecebeArgumentação*	R10
	Critica	R17
	AnalisaHA	R19

Figura 4.3: Requisitos atendidos pelos métodos das classes do Assunto *Protocolo Mosca* no modelo de objetos.

O atributo *Amostra* aparece na classe *AgenteAprendiz* armazenando a amostra recebida de *AgenteOráculo*. Os atributos *ConjFatos*, *ConjHipóteses*, *ConjConjecturas* e *HA* armazenam o conhecimento gerado pelo Aprendiz no processo de aprendizagem de um conceito. O conjunto de fatos gerado representa as regularidades encontradas na amostra. O Aprendiz gera também um conjunto de hipóteses, a partir do conjunto de fatos, e um conjunto de conjecturas, a partir do conjunto de hipóteses. É gerada ainda uma Hipótese Aprendida a partir do conjunto de conjecturas. *Argumentação* armazena a argumentação do Aprendiz para uma solução por ele gerada. O *Anexo C* exemplifica as noções de *domínio*, *filtro*, *amostra*, *conjunto de fatos*, *conjunto de hipóteses*, *conjunto de conjecturas* e *Hipótese Aprendida*. Esses e todos os demais atributos de classes que compõem esta modelagem serão mostrados como estruturas de dados na especificação apresentada no Capítulo 5.

A classe *Agente* generaliza todos os componentes do protocolo MOSCA e traz o atributo *ConjuntoCrenças*, que armazena, para cada agente, o conjunto de crenças associado, segundo o Sistema de Crenças mostrado no Capítulo 3.

A classe *ProtocoloMosca* aparece no modelo como uma composição de todos os agentes. Tal classe aparece nesta página[7] do diagrama como elo de ligação com outra página que será mostrada a seguir (seção 4.6.2).

[7] A noção de *página* é utilizada por J. Rumbaugh para se referir a uma divisão física de um diagrama de objetos com o intuito de facilitar sua exposição. Uma página de diagrama é geralmente associada a uma

Os métodos *SelecionaFatosPertinentes*, *SelecionaHipótesesPertinentes* e *SelecionaConjecturasPertinentes* da classe *AgenteMestre* não atendem a nenhum requisito dentre os listados na seção 4.3, mas são importantes para o processo de aprendizagem. Estes métodos serão explicados na seção 4.7.

A Figura 4.4 mostra uma segunda página do Modelo de Objetos, contendo os assuntos *Mecanismo* e *Domínio*.

4.6.1.2 Assunto Mecanismo

A classe *Mecanismo* aparece no modelo generalizando as classes *MecanismoAbdução*, *MecanismoIndução* e *MecanismoDedução*. A Figura 4.5 mostra os requisitos atendidos por essas classes. O método *Altera* permite a alteração dos valores dos atributos, atendendo ao requisito 24 da lista da seção 4.3. Através do método *Inicializa*, o sistema atribui valores iniciais a estes atributos. Os atributos de *MecanismoDedução* armazenam os limites apresentados na seção 4.6.2, através dos quais o Aprendiz decide a crença resultante em cada nível de regra (fatos, hipóteses e conjecturas). Os atributos em *MecanismoAbdução* e *MecanismoIndução* são aqueles utilizados pelos mecanismos de aprendizagem brevemente descritos no Capítulo 3. Os requisitos atendidos por estes métodos são mostrados na Figura 4.5.

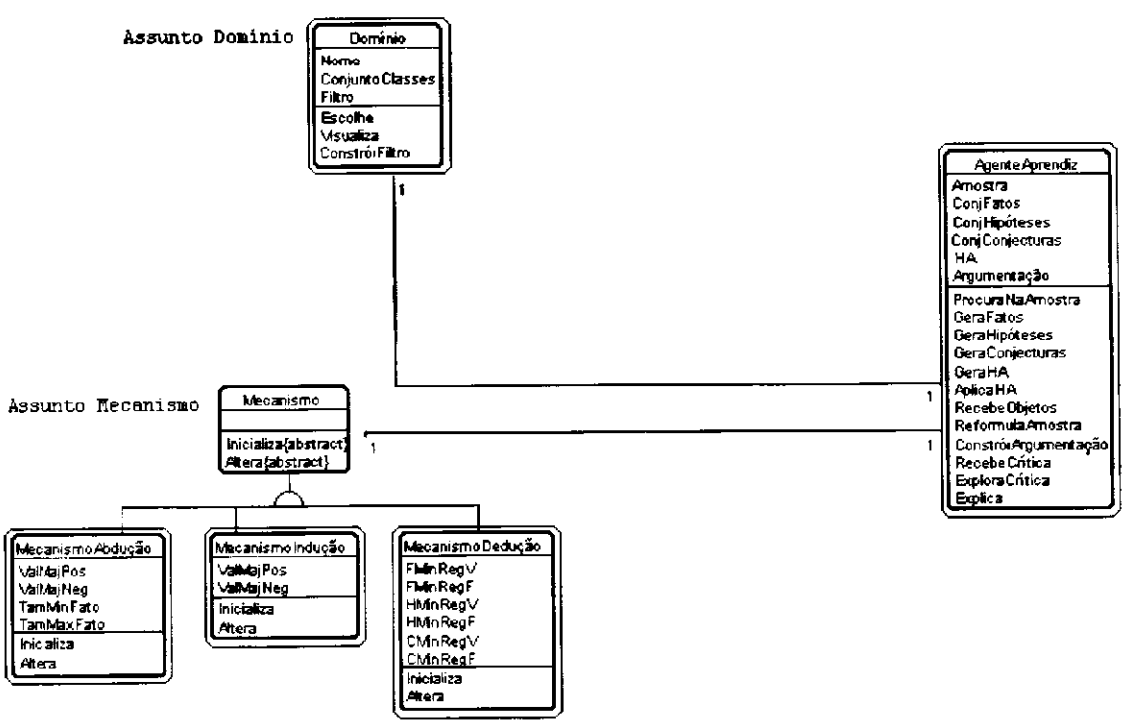


Figura 4.4: Assuntos *Mecanismo* e *Domínio* do Modelo de Objetos (Página 2).

página impressa.

Classe	Método	Requisito
MecanismoAbdução	*Altera*	
MecanismoIndução	*Altera*	R24
MecanismoDedução	*Altera*	

Figura 4.5: Requisitos atendidos pelos métodos das classes do Assunto *Mecanismo* no Modelo de Objetos.

4.6.1.3 Assunto Domínio

A classe *Domínio* armazena no atributo *ConjuntoClasses* as classes utilizadas para descrever o domínio, tais como foram definidas no editor de objetos do domínio. O atributo *Filtro* armazena o conjunto de classes selecionados pelo método *ConstróiFiltro* (se for o caso). O método *Visualiza* provê ao usuário a visualização do domínio selecionado. Os requisitos atendidos pelos métodos da classe *Domínio* são mostrados na Figura 4.6.

Classe	Método	Requisito
Domínio	*Seleciona*	R22
	ConstróiFiltro	R23

Figura 4.6: Requisitos atendidos pelos métodos da classe *Domínio* no Modelo de Objetos.

4.6.2 Interface e Gerenciamento de Dados

Pode-se dizer que os assuntos mostrados na Página 3 do Modelo de Objetos são aqueles comuns a todo sistema computacional. Ou seja, mostradas as classes modelando o domínio do problema, acrescentam-se ao modelo de objetos aquelas classes que provêm métodos para a interface com usuário e manipulação de arquivos. A Figura 4.7 mostra as classes dos assuntos *Arquivo* e *Interface.*

4.6.2.1 Assunto Arquivo

A classe *Arquivo* traz os métodos usuais de manipulação de arquivo. Esses métodos são mostrados no quadro da Figura 4.8, associados aos requisitos aos quais atendem.

4.6.2.2 Assunto Interface

A classe *ProtocoloMosca* traz o atributo *Estado*, que, como será mostrado pelo modelo dinâmico do sistema, retém a evolução das comunicações e comportamentos dos componentes do

protocolo. Esta evolução precisa ser mostrada ao usuário, pois dele também depende tal evolução. Isto é provido pelas classes do Assunto *Interface* (*Interface*, *Titulo*, *Menu*, *MenuArquivo*, *MenuAmbiente*, *MenuAprendizagem*, *MenuRevisão*, *MenuExploração* e *BarraStatus*). Tal estrutura busca refletir as etapas de uma sessão em SAID. As etapas de uma sessão no sistema são apresentadas a seguir na seção 4.7.1. A Figura 4.9 mostra o projeto do menu para SAID.

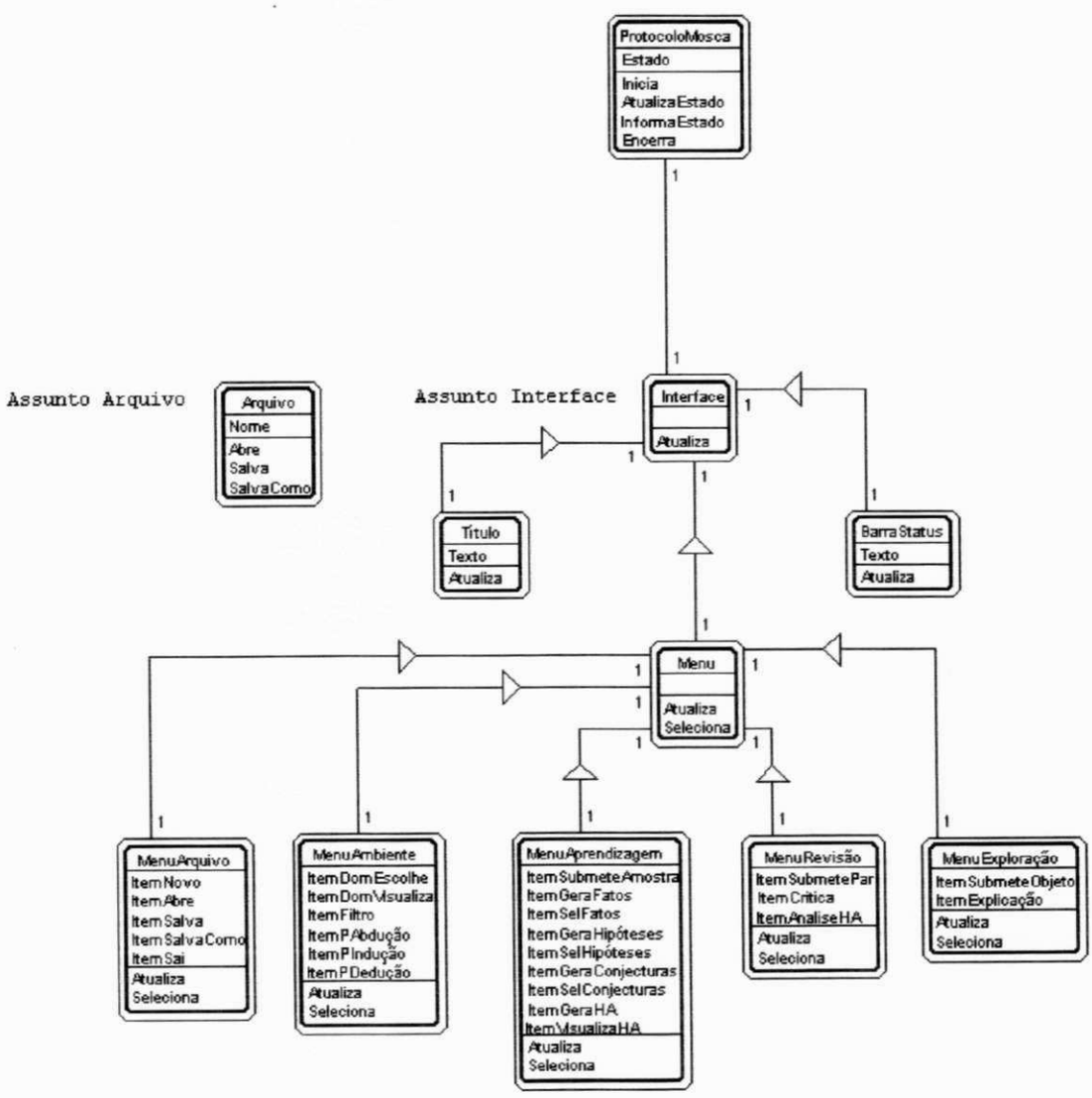


Figura 4.7: Assuntos *Arquivo* e *Interface* do Modelo de Objetos (Página 3).

Classe	Método	Requisito
Arquivo	*Abre*	R21
	Salva	R20
	SalvaComo	R20

Figura 4.8: Requisitos atendidos pelos métodos da classe *Arquivo* no Modelo de Objetos.

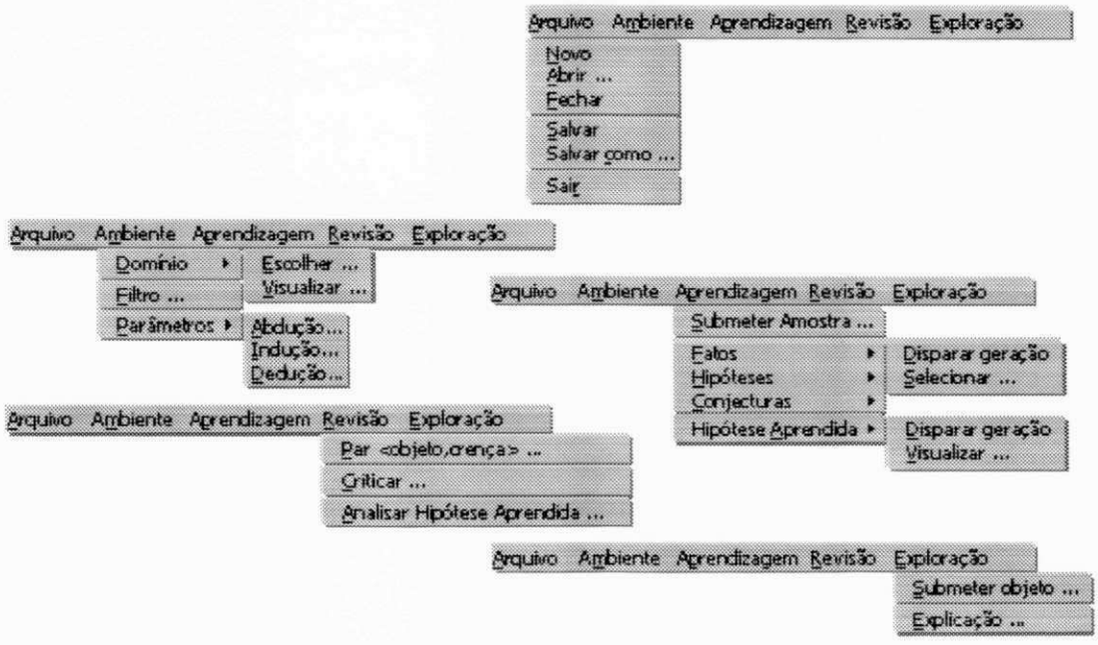


Figura 4.9: Hierarquia de comandos no menu.

O modelo de objetos proposto tem, assim, apenas um diagrama de objetos que, por sua vez, é dividido em três páginas. Na Figura 4.10 é possível visualizar as três páginas deste diagrama como uma única, perdendo-se, porém, a informação dos atributos e métodos de cada classe.

4.7 Modelo Dinâmico

Um sistema pode ser melhor entendido examinando-se primeiro sua estrutura estática, isto é, a estrutura de seus objetos e seus relacionamentos. Uma tal estrutura para SAID foi mostrada na seção anterior. Nesta seção, examinam-se as mudanças nos objetos daquela estrutura ao longo do tempo. Os aspectos de um sistema relacionados com as mudanças ao longo do tempo são representados no *modelo dinâmico*. O *controle* é o aspecto que descreve as sequências de operações que ocorrem em resposta aos estímulos externos, sem considerar como as operações são implementadas.

Esta seção descreve conceitos relativos ao fluxo de controle, interações e sequência de operações em SAID. Os principais conceitos em torno da modelagem dinâmica são os *eventos*, que representam os estímulos externos, e os *estados*, que representam valores dos objetos. O *diagrama de estado* relaciona eventos e estados.

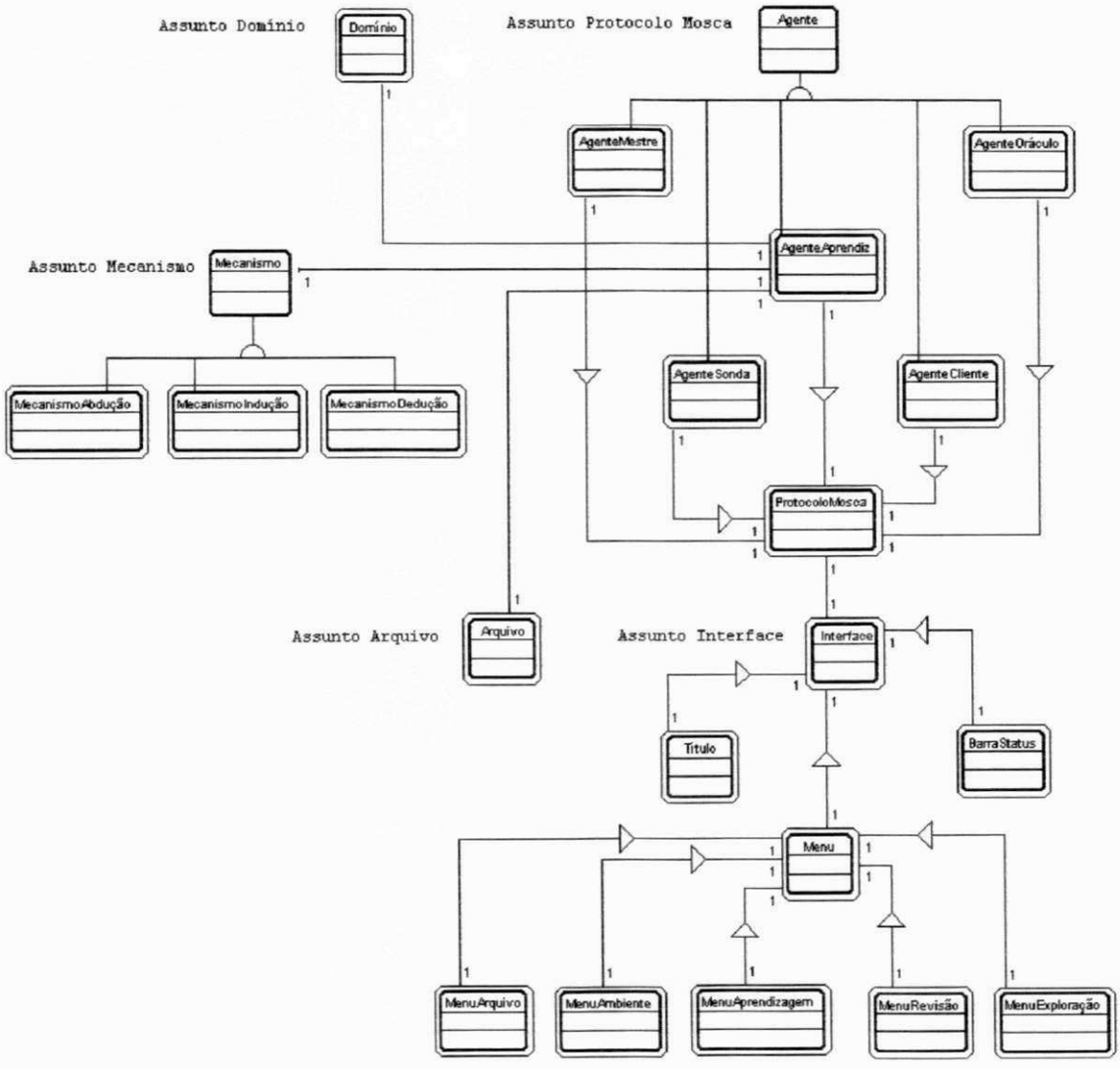


Figura 4.10: Hierarquia de classes completa no Modelo de Objetos.

4.7.1 As Etapas de uma Sessão em SAID

As tarefas que o usuário e o sistema são capazes de realizar ao longo de uma sessão são divididas em *etapas*, a saber: *definição de ambiente*, *aprendizagem*, *revisão* e *exploração*.

4.7.1.1 Definição de Ambiente

De fato, a manipulação do conhecimento em SAID se dá pela realização das etapas de *Aprendizagem*, *Revisão* e *Exploração*. Contudo, o ambiente de suporte a esta manipulação precisa ser previamente definido. Dessa forma, para preparar o ambiente para as etapas seguintes, o usuário deverá realizar as seguintes tarefas:

(i) *escolha de ambiente*, realizada sempre no início de uma sessão;

(ii) *escolha de domínio*, necessária apenas caso o usuário tenha escolhido manipular um novo ambiente;

(iii) *construção de um filtro sobre o domínio*, selecionando apenas parte deste domínio para ser utilizada na aprendizagem e

(iv) *parametrização dos mecanismos de aprendizagem*, possibilitando ao usuário alterar os valores padrão para os parâmetros sugeridos pelo sistema.

4.7.1.2 Aprendizagem

Escolhidos um ambiente e um domínio, SAID está *pronto* para iniciar o processo de aprendizagem. As tarefas que usuário e sistema devem realizar nesta etapa são:

(i) ***construção e envio de amostra;***

(ii) *geração de fatos*;

(iii) ***geração de hipóteses;***

(iv) *geração de conjecturas* e

(v) ***geração de hipótese aprendida.***

Cada uma destas tarefas depende da realização da anterior. À exceção das tarefas em *(i)*, todas as outras são realizadas pelo sistema. De fato, *(ii)*, *(iii)* e *(iv)* são passos necessários para que se possa realizar *(v)*.

Há ainda outras tarefas que o usuário pode realizar durante a etapa de aprendizagem: *(i)* seleção de fatos pertinentes; *(ia)* seleção de hipóteses pertinentes e *(iii)* seleção de conjecturas pertinentes. Por exemplo, o sistema considera como "pertinentes" todos os fatos gerados pelo passo *(ii)*. Esses fatos seriam utilizados como entrada para o passo *(iii)*. Mas ao usuário é dada a possibilidade de escolha de apenas alguns desses fatos (aqueles que *ele* julgar "pertinentes") para serem utilizados pelo passo *(iii)*. O mesmo acontece para as hipóteses e as conjecturas.

4.7.1.3 Revisão

Como resultado da execução com sucesso da tarefa *(v)* da etapa de aprendizagem, tem-se o sistema pronto para iniciar a etapa de revisão, uma vez que este dispõe de uma hipótese aprendida. As tarefas que devem ser realizadas nesta etapa são:

(i) seleção e envio de par <*objeto,crença*>;

(ii) construção e envio de argumentação;

(iii) construção e envio de crítica;

(iv) exploração de crítica e

(v) análise de evolução da Hipótese Aprendida.

Como foi mostrado, uma vez recebido do usuário no papel de Sonda um par *<objeto,crença>*, o Aprendiz deverá gerar uma argumentação. O sistema ficará então pronto para que o usuário no papel de Mestre possa construir e enviar sua crítica sobre a argumentação gerada. As tarefas *(i)* e *(iii)* nesta etapa podem ser repetidas quantas vezes deseje o usuário. No entanto, se uma possibilidade de mudança de etapa é detectada pelo Aprendiz após explorar a crítica recebida, este comunicará o fato ao usuário. No Capítulo 5 será mostrado como o aprendiz detecta tal possibilidade e como a comunica ao Mestre. Uma decisão do usuário, pela análise da evolução da Hipótese Aprendida, poderá provocar o retorno à etapa de aprendizagem ou o avanço à etapa de exploração.

4.7.1.4 Exploração

Uma vez dispondo de uma hipótese aprendida composta de lemas, o sistema está pronto para iniciar a etapa de exploração dessa hipótese. Nesta etapa, o usuário pode realizar as tarefas de seleção e envio de um objeto ao Aprendiz para que este retorne uma crença sobre o objeto submetido, segundo sua Hipótese Aprendida (ou segundo sua amostra). Escolheu-se o tipo de Aprendiz que sempre justifica sua solução, de forma que este deverá estar pronto para explicar sempre que uma resposta for enviada ao Cliente.

4.7.2 Os Cenários refletindo as etapas

Um cenário é uma sequência de eventos que ocorre durante uma execução particular de um sistema. O escopo do cenário pode incluir todos os eventos no sistema ou apenas aqueles eventos gerados por alguns dos objetos do modelo de objetos.

Nesta modelagem, os cenários principais refletem as etapas de uma sessão em SAID mostradas na sessão 4.7.1. Estes cenários são: *Definição de Ambiente*, *Aprendizagem*, *Revisão* e *Exploração*.

4.7.2.1 Cenário Definição de Ambiente

A Figura 4.11 mostra o cenário *Definição de Ambiente*. Este cenário mostra um leque de

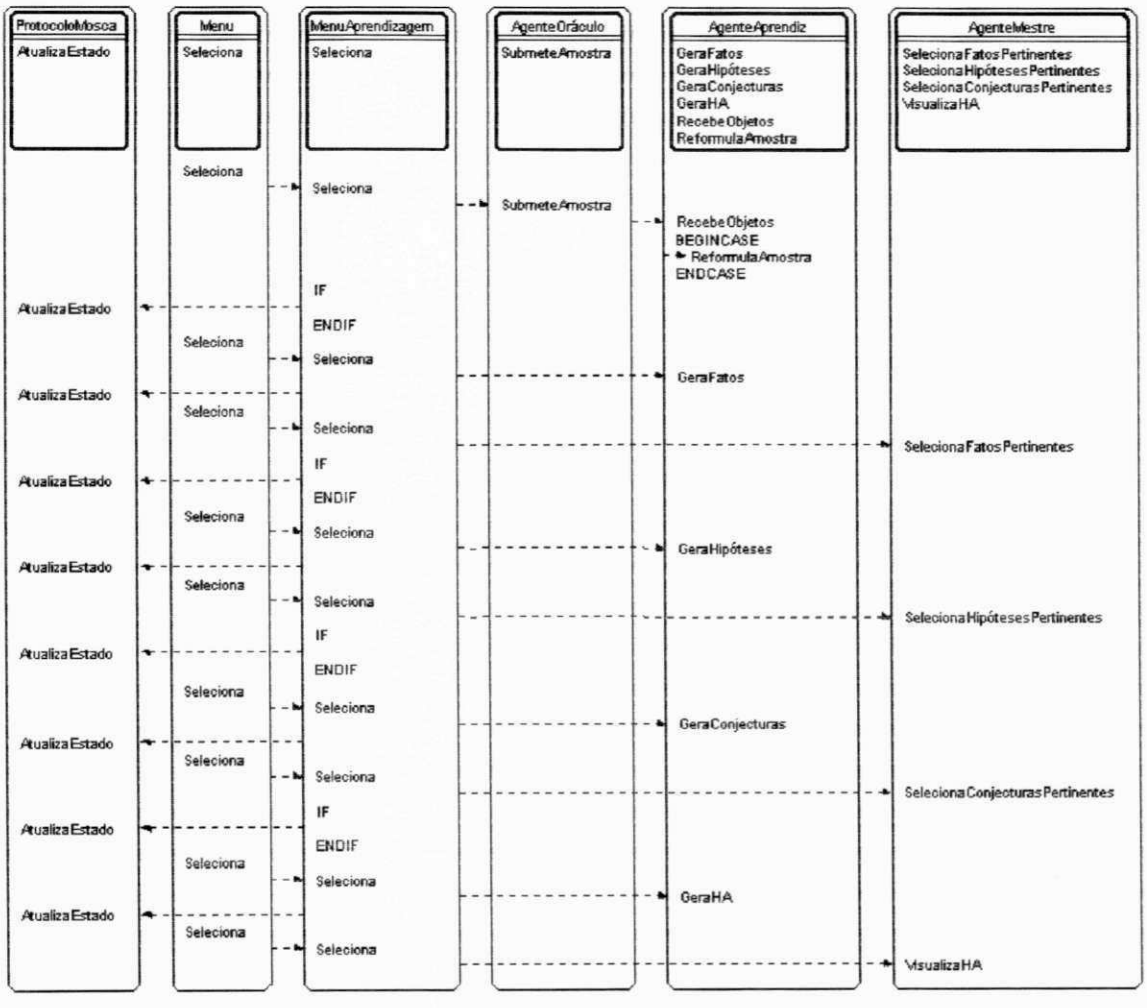


Figura 4.12: Cenário *Aprendizagem*.

4.7.2.3 Cenário Revisão

A Figura 4.13 traz o cenário *Revisão*. Continuando com a reflexão do usuário, poder-se-ia ter a seguinte descrição: "O sistema está pronto para a fase de revisão. Eu seleciono um objeto, indico se ele é exemplo ou contra-exemplo do conceito que a Hipótese Aprendida deverá representar e o envio ao sistema. O sistema me envia uma resposta, concordando ou discordando da minha indicação e me informa que dispõe de uma argumentação para seu resultado. Ele me informa ainda que aguarda minha crítica. Eu construo minha crítica e a envio para ele. Eu repito todo o processo algumas vezes, desde o envio de um objeto até o envio de uma crítica. Ao analisar a Hipótese Aprendida, eu posso decidir passar à etapa de exploração, se todas as hipóteses são lemas, ou ainda pedir que o sistema solicite uma nova amostra ao Oráculo, se alguma hipotese é não-lema."

utilizada pela ferramenta *Playground*.

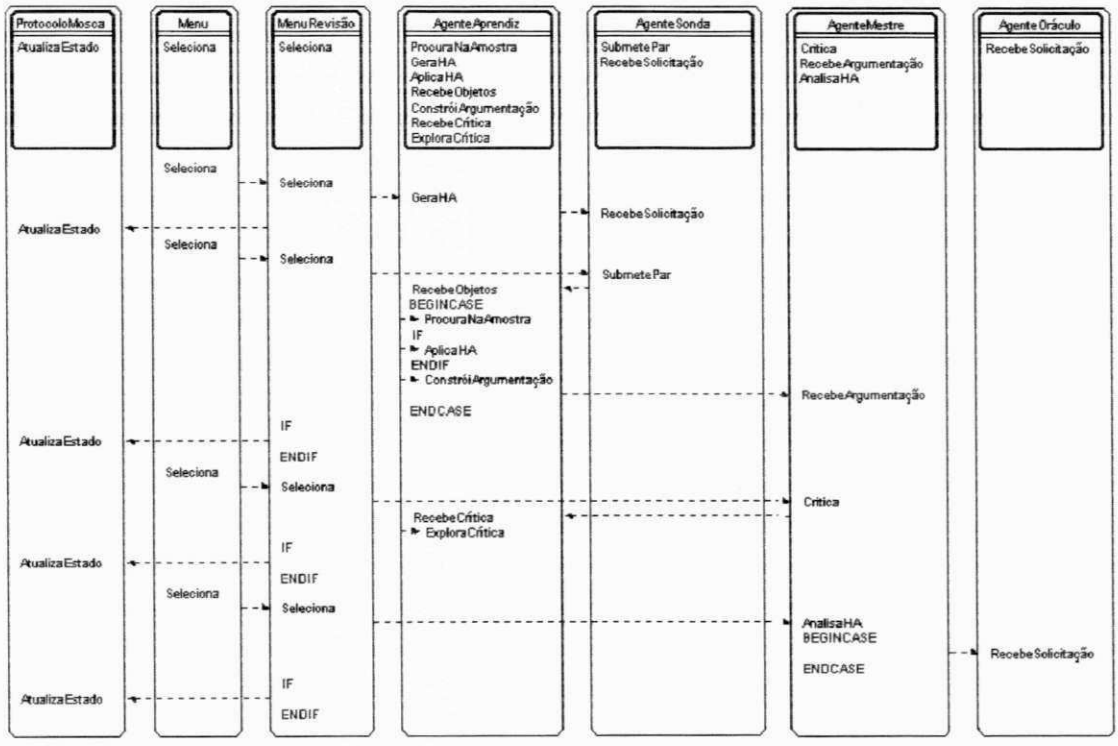


Figura 4.13: Cenário *Revisão*.

4.7.2.4 Cenário Exploração

O cenário *Exploração* é mostrado na Figura 4.14. Recorre-se novamente a uma suposta narrativa do usuário para descrevê-lo: "Eu seleciono um objeto e o envio ao sistema. Ele me retorna uma resposta, indicando sua crença sobre o objeto submetido, de acordo com o conceito que a Hipótese Aprendida representa. Ele me informa ainda que dispõe de uma explicação para sua resposta".

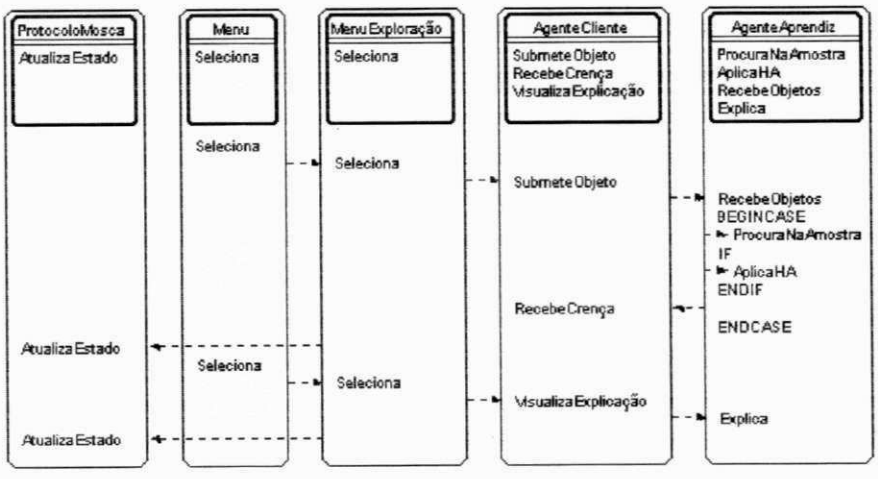


Figura 4.14: Cenário *Exploração*.

4.7.2.5 Um outro Cenário

Observando-se a figura relativa a cada cenário, percebe-se que uma chamada ao método *Atualiza* da classe *ProtocoloMosca* aparece ao final das chamadas ao método *Seleciona*, da classe de menu equivalente à etapa. *Atualiza* é responsável por disparar a sequência de eventos internos responsáveis pela atualização da interface que guiará o usuário ao longo de uma sessão no sistema. Essa sequência de eventos é mostrada no cenário *ProtocoloMosca Atualiza seu Estado* da Figura 4.15.

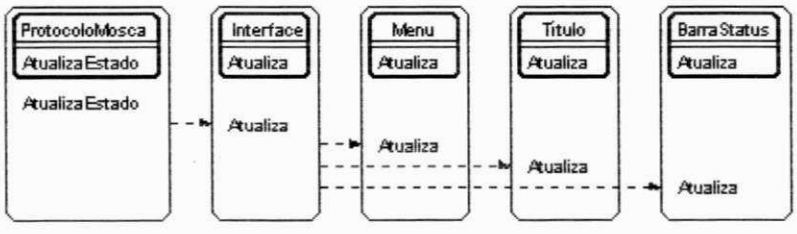


Figura 4.15: Cenário *ProtocoloMosca Atualiza seu Estado*.

4.7.3 Diagrama de Estados

Os *diagramas de estado* são grafos cujos nós são estados e cujos arcos são *transições* entre os estados. Estas transições são causadas pelos *eventos*, que podem ser externos (causados diretamente pelo usuário) ou internos (causados pelo sistema em meio à realização de alguma operação). A seguir são mostrados os diagramas de estados para as classes que modelam os agentes do Protocolo MOSCA.

O diagrama de estados para a classe *AgenteAprendiz* é mostrado na Figura 4.16. O estado inicial é *Inativo*. Quando o método *IniciaAprendizagem* desta classe é invocado, um objeto passa ao estado *ProntoParaReceberAmostra*. Deste estado, o objeto passa a *ProntoParaGerarFatos*, através do evento externo *Receber amostra*, que invoca o método *RecebeObjetos* desta classe. Uma vez *ProntoParaGerarFatos* e recebendo o evento externo *Gerar fatos*, que invoca o método *GeraFatos*, o objeto passa ao estado *ProntoParaGerarHipóteses*. O objeto passa ao estado seguinte, *ProntoParaGerarConjecturas*, ao receber o evento externo *Gerar hipóteses*, que dispara a execução do método *GeraHipoteses*. O evento externo *Gerar conjecturas* invoca o método *GeraConjecturas*, levando o objeto ao estado *ProntoParaGerarHA*. Deste, o objeto passa ao estado *ProntoParaReceberPar*, através do evento externo *Gerar HA*, que invoca o método *GeraHA*. O evento externo aqui chamado *Receber par* é disparado pelo usuário no papel de Sonda, invocando o método *RecebeObjetos* desta classe. O término da execução deste método leva o objeto ao *estado Pron-*

toParaArgumentar. Através do evento externo *Argumentar*, que acontece quando o usuário no papel de Mestre dispara a execução do método *Criticar* (da classe *AgenteMestre*), o objeto passa ao estado *AguardandoCrítica*, pelo término da execução do método *InformaArg* desta classe. O objeto sai deste estado através do evento externo *Receber crítica*, que acontece quando o usuário no papel de Mestre envia a crítica construída. O método *RecebeCritica* é então invocado e, após seu término, o objeto está no estado *AguardandoAceitaçãoHA*. Quando o usuário no papel de Mestre aceita a Hipótese Aprendida com conceito aprendido, a fase de revisão é encerrada. O objeto passa ao estado *AguardandoObjeto* pelo evento externo *Receber objeto*, que representa o envio de um objeto (exemplo ou contra-exemplo do conceito) pelo usuário no papel de Cliente. O objeto pode continuar no mesmo estado se uma explicação é solicitada (evento externo *Explicar*, que invoca o método *Explica* desta classe) ou um novo objeto é submetido (evento externo *Receber objeto*, invocando o método *RecebeObjeto*). Os fluxos de eventos representados pelas setas de baixo para cima indicam as possibilidades de retorno a um estado anterior, através de um dos eventos acima apresentados.

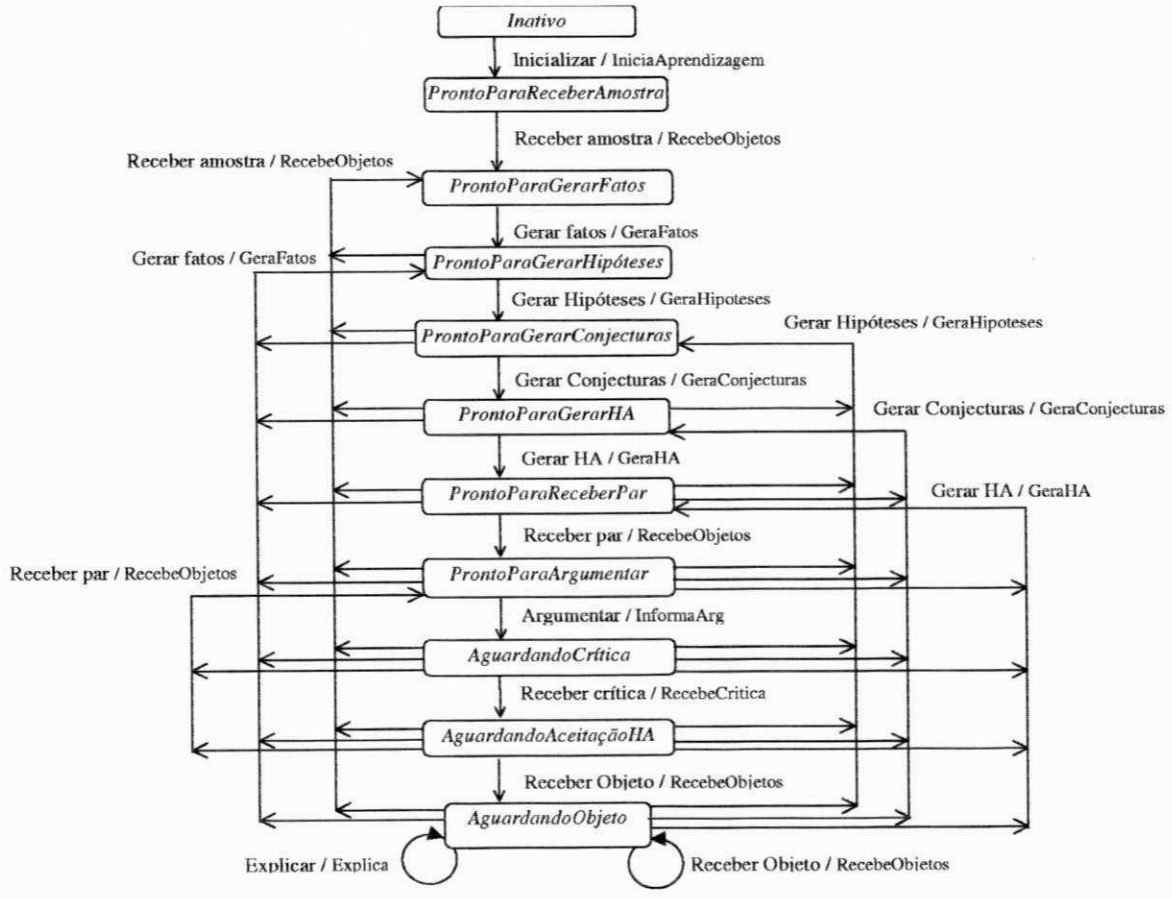


Figura 4.16: Diagrama de estados para a classe *AgenteAprendiz*.

O diagrama de estados para a classe *AgenteOraculo* é mostrado na Figura 4.17. Neste diagrama, o estado inicial é chamado *Inativo*. Deste estado, um objeto passa ao estado *RecebendoSoli-*

citação, através de um evento interno que é a chamada ao método *RecebeSolicitacao* desta classe (que acontece quando o Aprendiz dispõe de um Domínio e está pronto para iniciar a fase de aprendizagem). Através do evento externo *OK*, o usuário determina a passagem deste para o estado *ProntoParaEnviarAmostra*. Daí, o objeto pode voltar ao estado *RecebendoSolicitação* (caso um novo evento *Chegada de Solicitação* ocorra) ou ir para *SubmetendoAmostra* (através do evento externo *Submeter amostra*). De *SubmetendoAmostra*, o objeto volta ao estado *ProntoParaEnviarAmostra*, através dos eventos externos *Enviar* ou *Cancelar*.

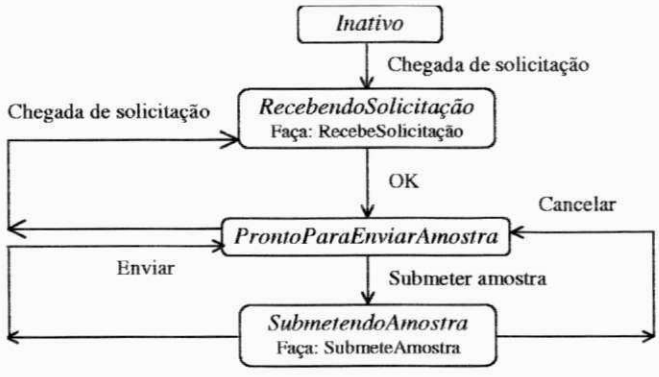


Figura 4.17: Diagrama de estados para a classe *AgenteOraculo*.

O diagrama de estados para a classe *AgenteSonda* é mostrado na Figura 4.18. Este diagrama é semelhante ao diagrama para a classe *AgenteOraculo*. Também neste diagrama, o estado inicial é *Inativo*. Um objeto passa deste para o estado *RecebendoSolicitação* através de um evento interno que é a chamada ao método *RecebeSolicitacao* da classe (quando o Aprendiz dispõe de uma Hipótese Aprendida e está pronto para iniciar a fase de revisão). Disparando o evento *OK*, o usuário determina a passagem deste para o estado *ProntoParaEnviarPar*. Deste estado, objeto pode voltar ao estado *RecebendoSolicitação* (caso um novo evento *Chegada de Solicitação* ocorra) ou ir para *SubmetendoPar* (através do evento externo *Submeter par*). De *SubmetendoPar*, o objeto volta ao estado *ProntoParaEnviarPar*, através dos eventos externos *Enviar* ou *Cancelar*.

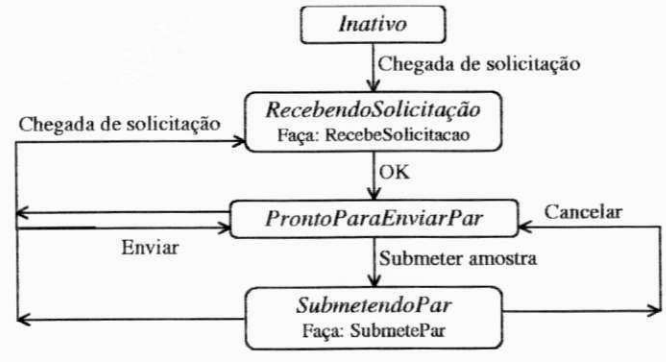


Figura 4.18: Diagrama de estados para a classe *AgenteSonda*.

O diagrama de estados para a classe *AgenteMestre* é mostrado na Figura 4.19. O primeiro estado é chamado *Inicial*. Deste estado, um objeto pode (a partir de um evento externo *Visualizar HA*) disparar a execução do método *VisualizaHA* (se a Hipótese Aprendida estiver disponível). De *Inicial*, o objeto pode ainda passar ao estado *AnalisandoHA*, através do evento externo *Analisar HA*. Nesse estado é executado o método *AnalisaHA*. Se, neste estado, o evento externo *Continuar revisão* acontece, o objeto volta ao estado *Inicial*. Se o evento externo *Aceitar HA* acontece, o objeto passa ao estado *AguardandoArgumentação*. De *AguardandoArgumentação* o objeto pode disparar a execução do método *VisualizaHA* ou passar ao estado *RecebendoArgumentação*, pelo evento interno *Receber argumentação*. Daí, o objeto passa ao estado *ProntoParaCriticar*, através do evento externo *OK*. O objeto pode, a partir do evento externo Visualizar HA, disparar a execução do método *VisualizaHA* e retornar ao mesmo estado, ou, através do evento externo *Criticar*, passar ao estado *Criticando* (onde é executado o método Critica). O objeto pode, através do evento externo *Enviar*, retornar ao estado *AguardandoArgumentação* ou, através do evento externo *Cancelar*, retornar ao estado *ProntoParaCriticar*.

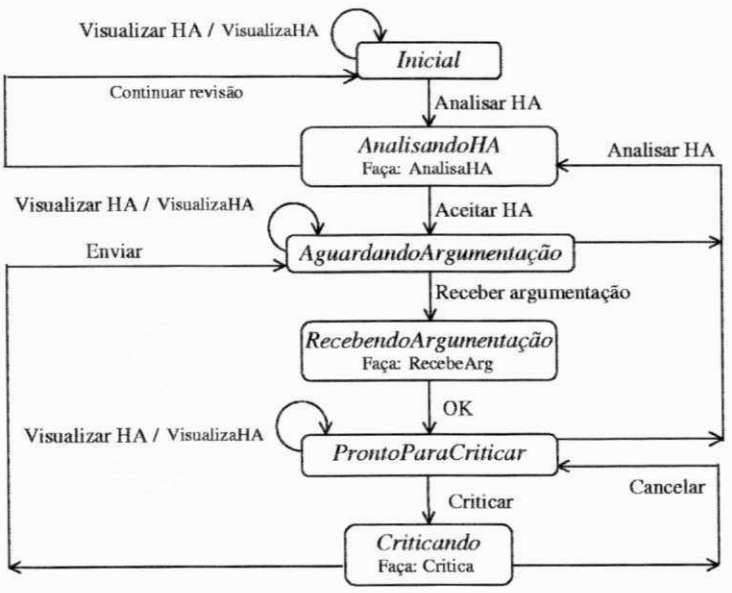


Figura 4.19: Diagrama de estados para a classe *AgenteMestre*.

Os eventos externos que disparam os eventos *SelecionarFatosPertinentes*, *SelecionarHipotesesPertinentes* e *SelecionarConjecturasPertinentes* podem ser realizados a partir de um dos estados *Inicial*, *AguardandoArgumentação* e *ProntoParaCriticar*. A execução destes pode terminar de duas maneira: ou um evento externo *OK* acontece (confirmando a escolha das regras pertinentes) ou um evento externo *Cancelar* acontece (cancelando uma eventual seleção). Se *OK* acontece, o objeto volta ao estado *Inicial*. Se *Cancelar* acontece, o objeto continua no mesmo estado. Estes fluxos não são mostrados no diagrama, pois dificultariam a visualização dos fluxos mais importantes para esta classe, que são aqueles que correspondem ao papel do Mestre no Protocolo Mosca.

O diagrama de estados para a classe *AgenteCliente* é mostrado na Figura 4.20. O estado inicial é *ProntoParaSubmeterObjeto*. Deste estado, o objeto passa a *SubmetendoObjeto*, através do evento externo *SubmeterObjeto*. A saída deste estado é determinada pelos eventos externos *Enviar* ou *Cancelar*. Se o evento é *Cancelar*, o objeto volta ao estado *ProntoParaSubmeterObjeto*. Se o evento é *Enviar*, o objeto passa ao estado *AguardandoCrença*. Através do evento *interno Chegada de crença* (uma chamada ao método *RecebeCrenca* desta classe), o objeto passa ao estado *RecebendoCrença*. O evento externo *OK* leva o objeto ao estado *ProntoParaSolicitarExplicação*. A partir deste estado, o objeto pode retornar a *SubmetendoObjeto* (pelo evento *externo Submeter objeto*) ou passar ao estado *VisualizandoExplicação* (pelo evento externo *Solicitar explicação*). Deste último estado, o objeto é levado ao estado *ExplicaçãoVisualizada* pelo evento externo *OK*. Daí, o objeto pode voltar ao estado *VisualizandoExplicação* (através do evento externo *Solicitar explicação*) ou ao estado *SubmetendoObjeto* (através do evento externo *Submeter objeto*).

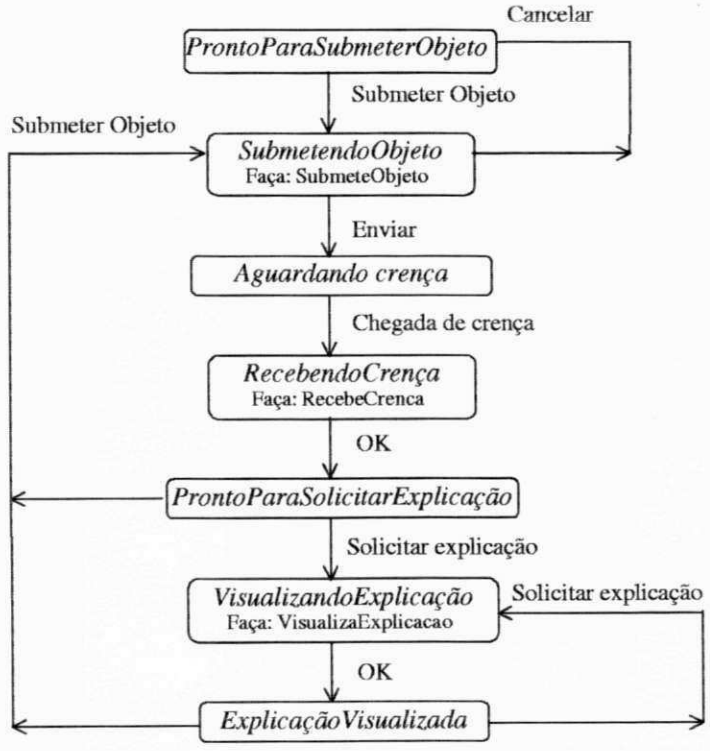


Figura 4.20: Diagrama de estados para a classe *AgenteCliente*.

4.8 Discussão

Neste capítulo foi proposta uma modelagem para um sistema de apoio à descoberta baseado no agente racional SAID. Tal modelagem baseou-se no paradigma da orientação a objetos. Através do modelo de objetos foi apresentada a estrutura estática do sistema, isto é, a estrutura das classes, incluindo seus atributos, métodos, bem como os relacionamentos entre estas classes. O

Capítulo 5

Especificação Formal do Sistema SAID

5.1 Introdução

No Capítulo 4, apresentou-se uma modelagem para um sistema de apoio à descoberta baseado no Agente Racional SAID. Apresenta-se neste capítulo, uma especificação formal que pode ser vista como um refinamento de tal modelagem.

Para construir tal especificação utiliza-se G-CPN, uma ferramenta formal para modelagem e especificação de sistemas de software baseada em Redes de Petri e Orientação a Objeto. O modelo G-CPN é construído sobre os princípios que originaram a estrutura de G-Net em [DENG, 1992;1993] e as Redes de Petri Coloridas [JENSEN, 1987;1992].

A utilização de G-CPN no contexto deste trabalho é de significativa importância pelo seguinte:

(*i*) o modelo G-CPN proporciona a especificação formal de sistemas através do formalismo associado às Redes de Petri;

(*ii*) tal formalismo pode ser alcançado através da construção de modelos gráficos, mais facilmente manipuláveis do que o formalismo matemático direto;

(*iii*) o comportamento do sistema pode ser validado através de simulação, utilizando-se o ambiente Design/CPN [**CPN, 1998**];

(*iv*) além de Redes de Petri, a ferramenta G-CPN é construída sobre os princípios do paradigma da Orientação a Objeto, permitindo o rápido transporte da modelagem proposta no capítulo 4;

(*v*) O formalismo associado à especificação é fundamentalmente importante para que propriedades de uma futura extensão do sistema (discutida nas *Conclusões*) possam ser verificadas.

O capítulo é organizado da seguinte maneira: na seção 5.2 introduz-se a noção de Redes de

Petri. Na seção 5.3, uma introdução aos Sistemas de G-CPNs é apresentada (definições formais e algumas aplicações podem ser encontradas em [GUERRERO, 1997]). Na seção 5.4 apresenta-se, por assunto, os módulos G-CPN construídos a partir da modelagem apresentada no Capítulo 4. Na seção 5.5 encerra-se este capítulo com uma breve discussão sobre os resultados alcançados.

5.2 Noções de Redes de Petri

Uma Rede de Petri pode é *estrutura de rede* mais uma *marcação* inicial. A estrutura pode ser vista como um grafo onde os nós são *lugares* (representados graficamente por círculos) ou *transições* (retângulos) e os arcos representam uma relação de fluxo, indo de um lugar para uma transição ou de uma transição para um lugar. As *fichas* representam os dados sendo manipulados por uma rede. Uma marcação é um conjunto de fichas distribuídas pelos lugares da rede. Uma transição pode ter lugares de *entrada* e de *saída*, representados, respectivamente, pelos lugares no extremo dos arcos que chegam e que saem da transição. Cada arco tem um *peso*, determinando o número de fichas que carrega. Uma transição está *habilitada* se cada lugar de entrada contém pelo menos o número de fichas especificado nos pesos de seus arcos de entrada. Uma transição habilitada pode ou não disparar. Ao disparar, uma transição coloca nos lugares de saída o número de fichas especificado nos arcos de saída. Uma introdução às Redes de Petri pode ser encontrada em [REISIG, 1985].

Um tipo de extensão das Redes de Petri de alto nível são as Redes de Petri Coloridas (Coloured Petri Nets - CP-nets) [JENSEN, 1992]. Através das CP-nets é possível representar tipos de dados complexos, associando a cada ficha um valor de dado chamado *cor* da ficha. O disparo das transições pode causar modificações nas cores das fichas. Há ainda outras noções importantes associadas às Redes de Petri Coloridas, mas estas estão além do interesse deste trabalho.

5.3 Sistemas de G-CPNs

Um módulo G-CPN é o elemento básico a partir do qual sistemas de G-CPNs são construídos. Cada módulo é um objeto com seu próprio estado e comportamento.

Sistemas de G-CPNs são conjuntos de módulos G-CPN concorrentes, cooperantes e fracamente acoplados, cujo propósito principal é a modelagem e especificação formais e executáveis de sistemas de software distribuídos. Do ponto de vista estrutural, um sistema de G-CPNs é a integração de um conjunto de módulos e um ambiente de interação. O ambiente de interação é responsável por manter a semântica correta associada ao sistema. No contexto de modelagem de sistemas

distribuídos, o ambiente pode ser visto como uma abstração de uma rede de comunicação e diz respeito à semântica do envio de mensagens entre os módulos.

Quando um módulo solicita um serviço de um outro módulo, o ambiente de interação é responsável pelo transporte das mensagens de solicitação e de resposta. Dessa forma, do ponto de vista de um módulo, o mecanismo de transferência pode ser abstraído e é suficiente indicar o serviço desejado e o módulo servidor.

O modelo de interação utilizado na comunicação entre módulos G-CPN é essencialmente um protocolo cliente-servidor. Quando a solicitação de execução de um dado serviço é atendida, diz-se que uma *invocação* ocorreu. A partir daí, o módulo aguarda que o ambiente ative o serviço remoto e obtenha a resposta correspondente. Estando a informação necessária no ambiente, um serviço de ativação é gerado pelo depósito dos dados no lugar inicial do método. O método servidor ativado está, assim, apto a realizar as operações solicitadas. A IS (*Internal Structure* – estrutura interna), que diz respeito ao comportamento do método, é sintaticamente equivalente a uma CPN. Sempre que um resultado é obtido, o ambiente o detecta (checando a marcação do lugar predeterminado) e inicia seu transporte de volta para o módulo cliente. Este evento é chamado uma *terminação* de um método. Após a ocorrência de uma terminação em um módulo servidor, o ambiente detecta o módulo cliente correspondente e força a ocorrência de um evento neste módulo, chamado *retorno*. Tal evento deposita a(s) ficha(s) resultante(s) relacionada(s) com o serviço previamente solicitado.

Os eventos associados ao protocolo de interação entre dois dados módulos são então: invocação, ativação, terminação e retorno. Ë importante notar que todos os eventos ocorrem no nível de sistema, mas apenas dois deles ocorrem em cada módulo envolvido. Uma invocação e um retorno ocorrem no módulo cliente, enquanto que uma ativação e uma terminação ocorrem no módulo servidor.

Cada módulo G-CPN é estruturalmente representado por duas subestruturas. A primeira representa a abstração do módulo e é chamada GSP (*Generic Switching Place*) ou simplesmente, a interface; o segundo é a estrutura interna do módulo, chamada IS (*Internal Structure*). A interface determina a visão do módulo do ponto de vista de um outro módulo no sistema. A interface serve também para declarar os atributos e os métodos encapsulados no módulo. Para cada método declarado na interface há dois lugares indicados na IS: o primeiro determina onde as fichas de ativação devem ser colocadas, enquanto que o segundo determina o lugar de terminação.

Para representar invocações de métodos remotos dentro da IS, utiliza-se um lugar especial

chamado ISP (*Instantiating Switching Place*). Este lugar opera de maneira intuitiva: quando uma ficha é nele depositada, uma invocação do método relativo pode ocorrer; se a invocação ocorre, a ficha irá desaparecer; se o método invocado encerra (uma terminação ocorre), um retorno pode ocorrer e, quando o retorno ocorre, uma ficha com o resultado é depositada no ISP, habilitando as transições de saída.

Lugares normais, bem como atributos e lugares de ativação são graficamente representados por círculos. Lugares de terminação são representados por círculos duplos. ISPs são denotados por elipses e uma inscrição interna determina o método e o módulo a ser invocado. Outros elementos (transições, inscrições e arcos) são graficamente representados como em CP-Nets. Uma exceção deve ser observada: dois conjuntos de cores são associados a um ISP, um para as fichas de invocação e outro para as fichas de retorno. Na Figura 5.1 mostram-se tais elementos em um módulo cliente e em um módulo servidor, onde o método *Cliente.MostraMsg* modela a exibição de uma mensagem cujo conteúdo é conhecido pela invocação do método *Servidor.InformaMsg*.

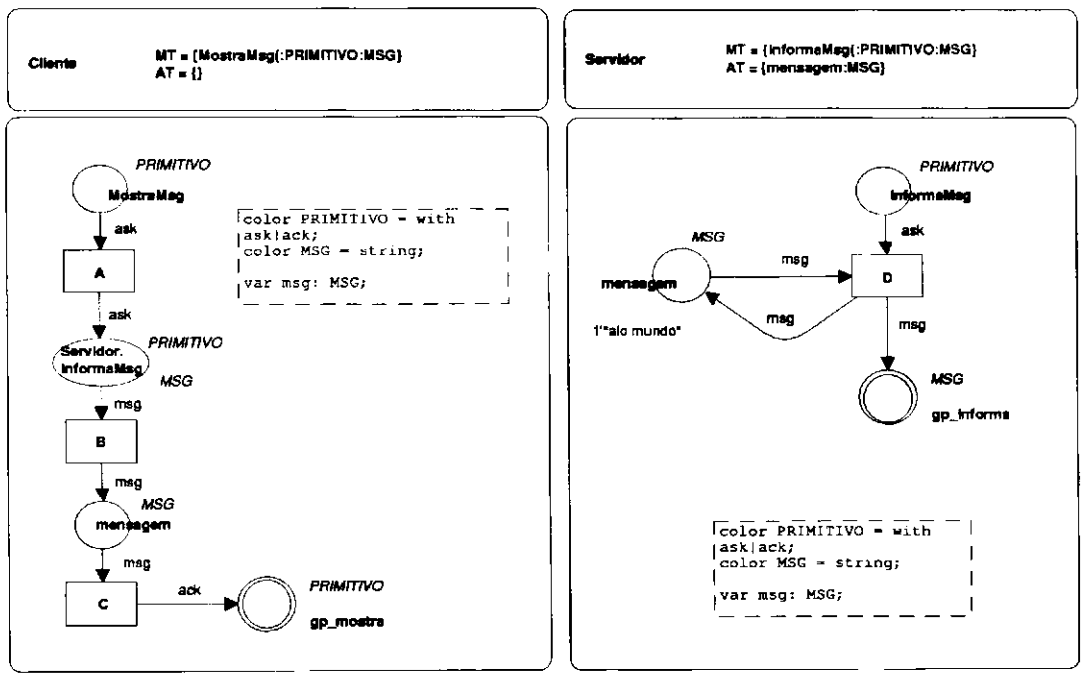


Figura 5.1: Exemplo de módulos cliente e servidor.

Um módulo G-CPN pode atender a qualquer número de solicitações concorrentemente. Isto é possível porque quando uma ativação ocorre, o ambiente cria uma nova instância do módulo servidor para atender à solicitação. Cada instância é tratada por um contexto. Contextos podem ser vistos como um objeto com uma estrutura funcional idêntica do método invocado. Contudo, há uma exceção desta regra. Atributos em módulos G-CPN são associados a lugares na IS. Como um atributo é considerado como parte do estado global de um objeto, seu estado (marcação) é único e

assim é compartilhado por todos os contextos. Dessa forma, se um módulo G-CPN tem atributos, invocações criam contextos copiando a estrutura do módulo, exceto lugares representando atributos. Um contexto é automaticamente destruído quando sua ativação termina.

5.4 SAID como um Sistema de G-CPNs

A seguir, são mostrados os módulos G-CPN que compõem a especificação formal proposta para o sistema SAID. Este módulos podem ser vistos como um refinamento da modelagem proposta no Capítulo 4. Uma observação importante é que, apesar de não sugerir explicitamente um ambiente de programação, faz-se, durante a exposição dos métodos de cada classe, menção a alguns componentes que geralmente fazem parte de linguagens voltadas para o ambiente Windows, tais como botões e caixas de mensagens. A menção destes componentes deve ser entendida como uma maneira de facilitar o entendimento do leitor, não como uma restrição de implementação para a especificação proposta.

5.4.1 Assunto Protocolo MOSCA

As classes neste assunto são: *ProtocoloMosca, AgenteCliente, AgenteOraculo, AgenteSonda, AgenteAprendiz e AgenteMestre*. A seguir, mostram-se os módulos G-CPN para estas classes.

5.4.1.1 G-CPN *ProtocoloMosca*

O módulo *ProtocoloMosca* é mostrado na Figura 5.2. Este módulo traz como métodos *Inicia, AtualizaEstado, InformaEstado* e *Encerra* e como atributos *estado* e *rodando.*

Inicia é o primeiro método invocado assim que a aplicação é lançada. Este método modela a inicialização do sistema, invocando os módulos que necessitam fazer inicialização de seus atributos. A marcação inicial no lugar de ativação do método determina, assim, o início de uma sessão em SAID. Esta marcação é uma ficha *ask*[9] que habilita a transição *PM1*. As fichas depositadas nos lugares de saída de *PM1* representam o seguinte:

- no lugar-atributo *estado*, uma ficha *0*, indicando estado inicial do sistema. Valores numéricos inteiros são utilizados nesta especificação para representar os estados possíveis de SAID por razões de praticidade na manipulação em expressoões utilizando operadores relacionais, como será mostrado ao longo deste capítulo. A tabela da Figura 5.3 mostra o número inteiro associado a cada estado;

[9] As fichas da cor PRIMITIVO, *ask*, *ack* e *nak* representam, respectivamente, uma solicitação, uma confirma-

chamado ISP (*Instantiating Switching Place*). Este lugar opera de maneira intuitiva: quando uma ficha é nele depositada, uma invocação do método relativo pode ocorrer; se a invocação ocorre, a ficha irá desaparecer; se o método invocado encerra (uma terminação ocorre), um retorno pode ocorrer e, quando o retorno ocorre, uma ficha com o resultado é depositada no ISP, habilitando as transições de saída.

Lugares normais, bem como atributos e lugares de ativação são graficamente representados por círculos. Lugares de terminação são representados por círculos duplos. ISPs são denotados por elipses e uma inscrição interna determina o método e o módulo a ser invocado. Outros elementos (transições, inscrições e arcos) são graficamente representados como em CP-Nets. Uma exceção deve ser observada: dois conjuntos de cores são associados a um ISP, um para as fichas de invocação e outro para as fichas de retorno. Na Figura 5.1 mostram-se tais elementos em um módulo cliente e em um módulo servidor, onde o método *Cliente.MostraMsg* modela a exibição de uma mensagem cujo conteúdo é conhecido pela invocação do método *Servidor.InformaMsg*.

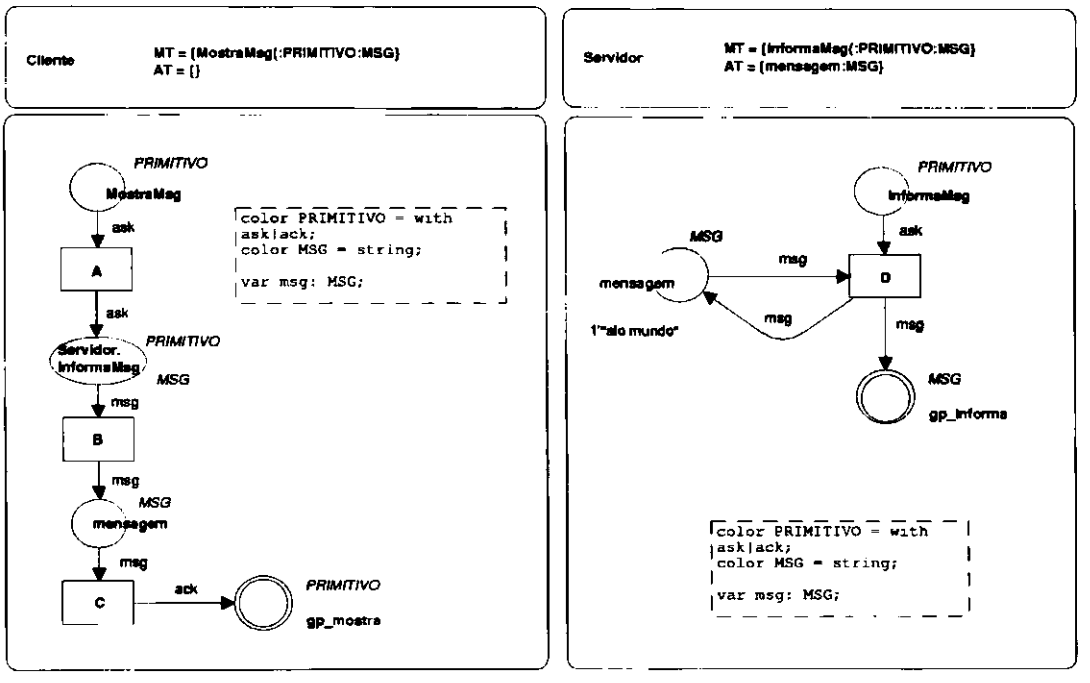


Figura 5.1: Exemplo de módulos cliente e servidor.

Um módulo G-CPN pode atender a qualquer número de solicitações concorrentemente. Isto é possível porque quando uma ativação ocorre, o ambiente cria uma nova instância do módulo servidor para atender à solicitação. Cada instância é tratada por um contexto. Contextos podem ser vistos como um objeto com uma estrutura funcional idêntica do método invocado. Contudo, há uma exceção desta regra. Atributos em módulos G-CPN são associados a lugares na IS. Como um atributo é considerado como parte do estado global de um objeto, seu estado (marcação) é único e

assim é compartilhado por todos os contextos. Dessa forma, se um módulo G-CPN tem atributos, invocações criam contextos copiando a estrutura do módulo, exceto lugares representando atributos. Um contexto é automaticamente destruído quando sua ativação termina.

5.4 SAID como um Sistema de G-CPNs

A seguir, são mostrados os módulos G-CPN que compõem a especificação formal proposta para o sistema SAID. Este módulos podem ser vistos como um refinamento da modelagem proposta no Capítulo 4. Uma observação importante é que, apesar de não sugerir explicitamente um ambiente de programação, faz-se, durante a exposição dos métodos de cada classe, menção a alguns componentes que geralmente fazem parte de linguagens voltadas para o ambiente Windows, tais como botões e caixas de mensagens. A menção destes componentes deve ser entendida como uma maneira de facilitar o entendimento do leitor, não como uma restrição de implementação para a especificação proposta.

5.4.1 Assunto Protocolo MOSCA

As classes neste assunto são: *ProtocoloMosca, AgenteCliente, AgenteOraculo, AgenteSonda, AgenteAprendiz e AgenteMestre*. A seguir, mostram-se os módulos G-CPN para estas classes.

5.4.1.1 G-CPN *ProtocoloMosca*

O módulo *ProtocoloMosca* é mostrado na Figura 5.2. Este módulo traz como métodos *Inicia, AtualizaEstado, InformaEstado* e *Encerra* e como atributos *estado* e *rodando*.

Inicia é o primeiro método invocado assim que a aplicação é lançada. Este método modela a inicialização do sistema, invocando os módulos que necessitam fazer inicialização de seus atributos. A marcação inicial no lugar de ativação do método determina, assim, o início de uma sessão em SAID. Esta marcação é uma ficha *ask*[9] que habilita a transição *PM1*. As fichas depositadas nos lugares de saída de *PM1* representam o seguinte:

- no lugar-atributo *estado*, uma ficha *0*, indicando estado inicial do sistema. Valores numéricos inteiros são utilizados nesta especificação para representar os estados possíveis de SAID por razões de praticidade na manipulação em expressões utilizando operadores relacionais, como será mostrado ao longo deste capítulo. A tabela da Figura 5.3 mostra o número inteiro associado a cada estado;

[9] As fichas da cor PRIMITIVO, *ask*, *ack* e *nak* representam, respectivamente, uma solicitação, uma confirma-

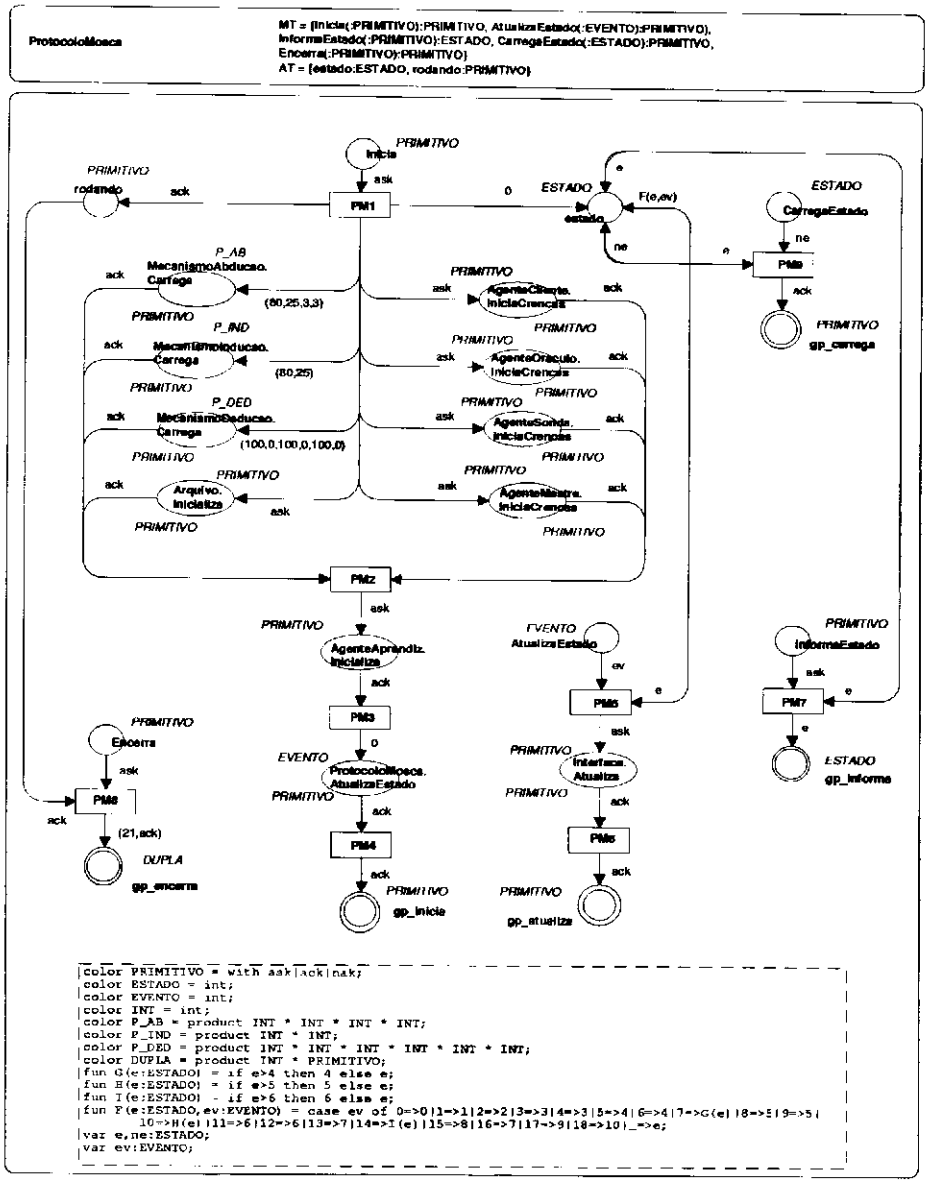


Figura 5.2: Módulo G-CPN *ProtocoloMosca*.

Estado	
Inicial	0
Ambiente escolhido	1
Domínio escolhido	2
Amostra enviada	3
Fatos disponívcis	4
Hipóteses disponíveis	5
Conjecturas disponíveis	6
HÁ disponível	7
Argumentação disponível	8
HA aceita	9
Explicação disponível	10

Figura 5.3: Associação dc estados a valores numéricos inteiros.

ção e uma negação (por exemplo: invocação, retorno bem sucedido e retorno sem sucesso de um método).

- no lugar-atributo *rodando*, uma ficha *ack*. Esta ficha permanecerá neste lugar ao longo de uma sessão e será removida pelo método *Encerra* do módulo;
- nos ISPs *MecanismoAbducao.Carrega*, *MecanismoInducao.Carrega* e *MecanismoDeducao.Carrega*, invocando o método *Carrega* de cada classe com valores iniciais sugeridos;
- nos ISPs *AgenteMestre.IniciaCrencas*, *AgenteSonda.IniciaCrencas*, *AgenteOraculo.IniciaCrencas* e *AgenteCliente.IniciaCrencas*, invocando o método *IniciaCrencas* de cada uma destas classes.

A transição *PM2* estará habilita quando uma ficha de retorno *ack* chegar a cada um dos ISPs de entrada desta transição. Seu disparo invoca o método *Inicializa* do módulo *AgenteAprendiz*. A transição *PM3* é então habilitada pela presença da ficha de retorno *ack* naquele ISP e se dispara invoca o método *AtualizaEstado* do módulo *ProtocoloMosca*, com uma ficha *0* (evento inicial). A chegada de uma ficha de retorno *ack* no ISP habilita a transição *PM4* que, ao disparar, encerra o método *Inicia*, colocando uma ficha *ack* em *gp_inicia*.

AtualizaEstado recebe como entrada um evento (o último ocorrido no sistema) e atualiza o atributo *estado* em função deste evento. O método atualiza então a interface de SAID (invocando o método *Atualiza* da classe *Interface*), de acordo com o novo estado.

InformaEstado modela o envio (para o módulo cliente) do valor corrente do atributo *estado*.

Encerra modela o encerramento de uma sessão em SAID. A transição *PM8*, habilitada pela presença de uma ficha no lugar-atributo *rodando* e outra no lugar de ativação, ao disparar consome a ficha em *rodando* e deposita uma ficha em *gp_encerra*.

Neste e demais módulos no sistema as ativações concorrentes, a exemplo dos ISPs de entrada da transição *PM2*, não requerem necessariamente um processamento paralelo. Além disso, a ordem em que as ativações ocorrem não é relevante no contexto da especificação proposta. A única restrição é que todas as ativações e seus respectivos retornos ocorram antes do disparo da transição de saída, como indicado pela estrutura da rede.

O nó de declaração do módulo *ProtocoloMosca* traz os conjuntos de cores: *PRIMITIVO*, cujas cores *ask*, *ack* e *nak* representam, respectivamente, uma solicitação, uma confirmação e uma negação; *ESTADO*, cuja cor é um inteiro representando o atributo *estado* do objeto; *EVENTO*, também um inteiro representando o último evento ocorrido no sistema; *DUPLA* (explicada na seção

5.4.1.2) e P_AB, P_IND e P_DED, que carregam n-tuplas com os valores inteiros sugeridos para inicialização dos atributos, respectivamente, de: *MecanismoAbducao, MecanismoInducao e MecanismoDeducao*. A função *F* mapeia um evento (o último ocorrido no sistema) em um estado (o estado seguinte, dado o evento ocorrido). *G, H* e *I* são funções de suporte a *F*. Finalmente, às variáveis *e* e *ne* é atribuída a cor *ESTADO* e à variável *ev* a cor *EVENTO*.

5.4.1.2 G-CPN *AgenteCliente*

Na Figura 5.4 mostra-se o módulo *AgenteCliente*. Os métodos que este módulo modela são: *IniciaCrencas, InformaConjCrencas, SubmeteObjeto, RecebeCrenca* e *VisualizaExplicação*. O atributo único é *conj_crencas*.

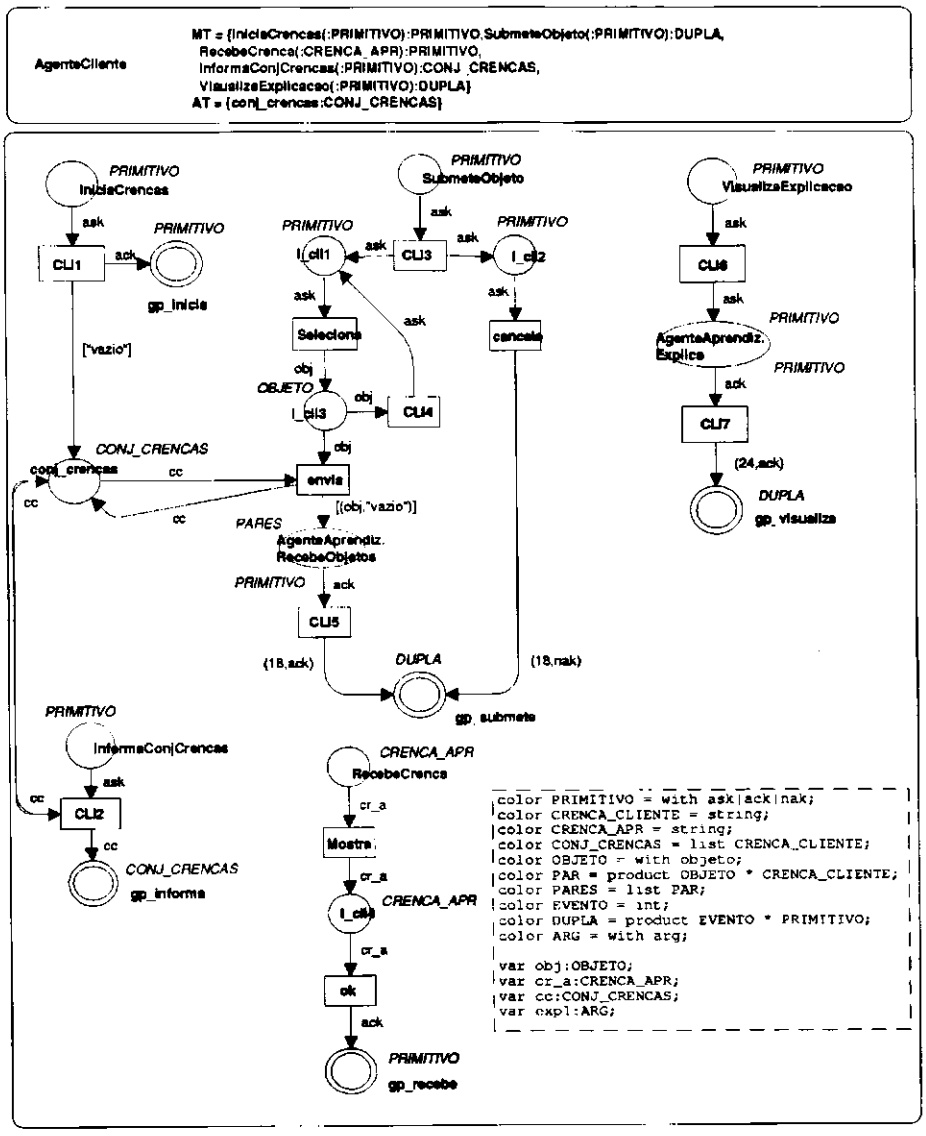


Figura 5.4: Módulo G-CPN *AgenteCliente*.

A execução do método *IniciaCrencas* resulta na inicialização do atributo *conj_crencas*. Nesta e em todas as classes modelando os agentes do protocolo MOSCA, este atributo é uma lista de *strings*. No Capítulo 3, mostrou-se que, segundo o sistema de crenças considerado, o conjunto de crenças para *Cliente* é vazio. Por questões de controle, considera-se o string "*vazio*" como elemento único deste conjunto.

O método *InformaConjCrencas* informa o valor do atributo *conj_crencas* do módulo *AgenteCliente* ao módulo chamador.

SubmeteObjeto é o método modelando a escolha e envio de um objeto a *AgenteAprendiz*. O método pode ser visto como uma caixa de diálogo, onde o usuário pode escolher um objeto previamente editado e enviá-lo pressionando um botão *Envia* da caixa. O método modela ainda a troca de um objeto já escolhido por outro, antes do envio. A seleção de um objeto durante a simulação da rede pode ser realizada através de um *binding*[10] para a variável *obj*. Um evento *envia* pode ocorrer uma vez feita a escolha, resultando no envio do objeto a *AgenteAprendiz* (através da invocação do método *RecebeObjetos* da classe *AgenteAprendiz*). Assim que uma ficha *ask* chega ao lugar de ativação do método, a transição *CLI3* é habilitada e, ao disparar, coloca uma ficha no lugar *l_cli2*, habilitando a transição *cancela*. O evento *cancela* pode também ocorrer (o disparo da transição *cancela*), cancelando o método (nenhum objeto será enviado a *AgenteAprendiz*). Nesta especificação, a transição *cancela* aparecerá em todos os métodos modelando caixas de diálogo.

O método *RecebeCrenca* modela a exibição de uma crença recebida (do módulo *AgenteAprendiz*) ao usuário no papel de *Cliente*. A execução deste método pode ser vista como uma caixa de mensagem exibindo a crença recebida ao usuário. O usuário poderá encerrar a caixa, por exemplo, através de um *click* em um botão *OK*, disparando o evento *OK* da rede.

Através do método *VisualizaExplicacao* o usuário no papel de *Cliente* pode visualizar, através das crenças associadas pelo sistema ao aplicar sua Hipótese Aprendida, uma explicação para a crença resultante. A aplicação da Hipótese Aprendida é mostrada na seção 5.4.1.5.

O conjunto de cores *OBJETO* tem, neste módulo, a cor *objeto*. Para efeito de implementação, o conjunto de cores *OBJETO* a ser observado é aquele definido no módulo *AgenteAprendiz*. Neste módulo *AgenteCliente*, tal conjunto de cores é assim definido por questões apenas de simplificação do nó de declaração da rede. A mesma ressalva é válida também para o conjunto de cores *ARG*. O conjunto *PARES* é definido como uma lista de cores da cor *PAR*. Esta última é definida como uma dupla de cores, respectivamente, dos conjuntos *OBJETO* e *CRENCA_CLIENTE*. Uma cor

do conjunto *DUPLA* leva ao modulo cliente a informação de qual método foi invocado (através de *EVENTO*) e se este terminou com sucesso ou foi cancelado (através de *PRIMITIVO*). As variáveis *obj*, *cr_a*, *cc* e *expl* são, respectivamente, dos conjuntos de cores *OBJETO*, *CRENCA_APR*, *CONJ - CRENCAS* e *ARG*.

5.4.1.3 G-CPN *AgenteOraculo*

O módulo G-CPN *AgenteOraculo* é mostrado na Figura 5.5. Os métodos modelados por este módulo são: *IniciaCrencas*, *SubmeteAmostra*, *RecebeSolicitacao* e *InformaConjCrencas*. O atributo único é *conj_crencas*.

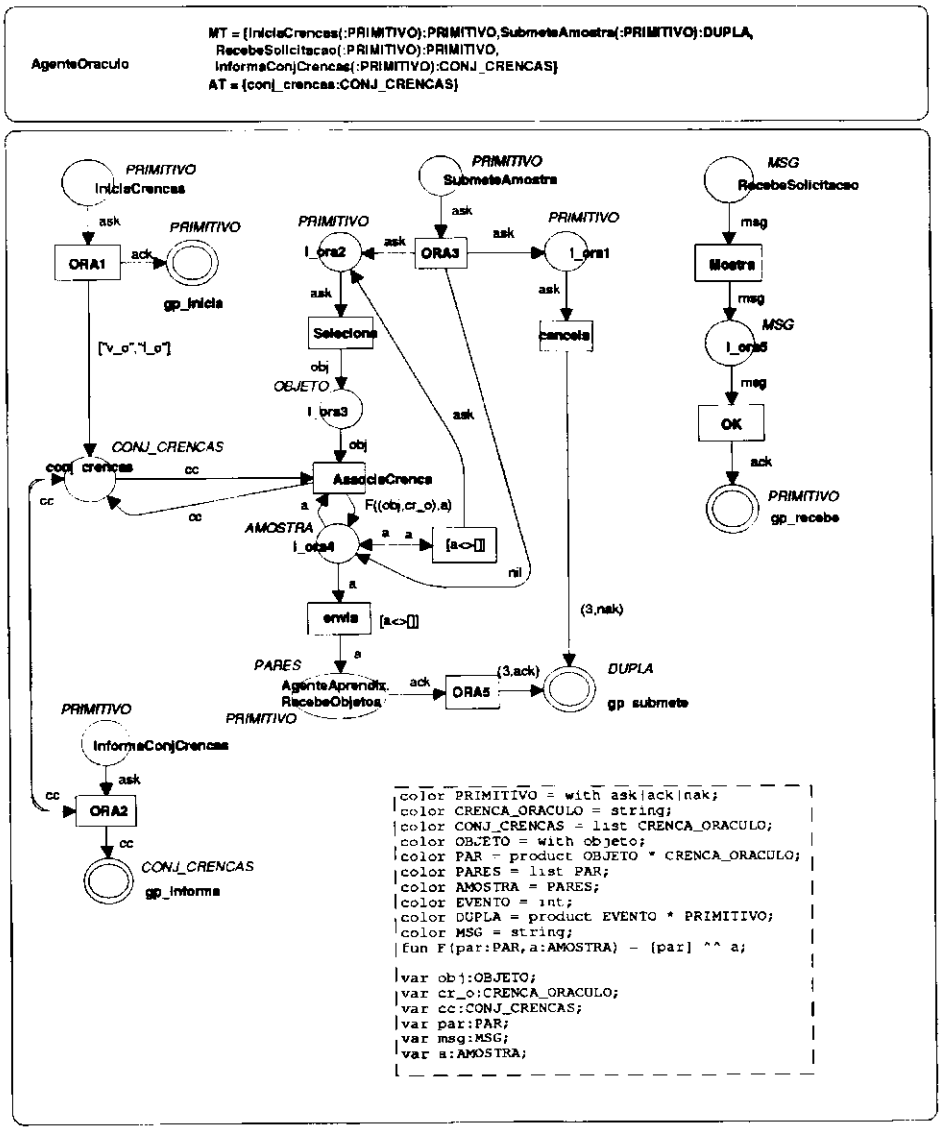


Figura 5.5: G-CPN *AgenteOraculo*.

[10] Através de um *binding* é possível instanciar-se uma variável antes do disparo de uma transição.

Como para o módulo *AgenteCliente*, a execução do método *IniciaCrencas* resulta na inicialização do atributo *conj_crencas*. Segundo o sistema de crenças considerado, este conjunto tem dois elementos, que, nesta especificação, são modelados como dois *strings*: "v_o" e "f_o".

O método *SubmeteAmostra* pode ser visto como uma caixa de diálogo onde o usuário no papel de *Oráculo* seleciona um objeto previamente editado e o associa a uma crença. Vários objetos podem ser escolhidos e associados a crenças, formando uma amostra. A amostra pode então ser enviada ao módulo *AgenteAprendiz* pelo disparo da transição *envia* que invoca o método *RecebeObjetos* da classe *AgenteAprendiz*. Através do disparo da transição *cancela* é também possível encerrar o método.

O método *RecebeSolicitacao* modela a exibição de uma mensagem recebida que deve ser exibida ao usuário no papel de *Oráculo*. Tal método pode ser invocado pelos módulos *AgenteAprendiz* ou *AgenteMestre*. As mensagens que cada módulo envia serão mostradas nas seções 5.4.1.5 e 5.4.1.6, respectivamente. A mensagem deve ficar exibida até que o usuário dê um *click* em um botão *OK* (se o método é implementado como uma caixa de mensagem) para encerrar a exibição, disparando o evento *OK*.

O método *InformaConjCrencas* informa o valor do atributo *conj_crenças* do módulo *AgenteOraculo* àquele que o invoca.

O conjunto de cores *OBJETO* é definido como a cor *objeto* para efeito de simplificação do nó de declaração. Uma implementação do sistema deve considerar este conjunto como definido no nó de declaração do módulo *AgenteAprendiz*.

5.4.1.4 G-CPN *AgenteSonda*

O módulo *AgenteSonda* é mostrado na Figura 5.6. No GSP deste módulo estão declarados os métodos: *IniciaCrencas*, *SubmetePar*, *RecebeSolicitacao* e *InformaConjCrencas*. O atributo único é *conj_crencas*.

Os métodos *IniciaCrencas* e *InformaConjCrencas* para este módulo são semelhantes aos métodos de mesmo nome do módulo *AgenteOraculo*. A exceção é que o conjunto de crenças para este módulo é o conjunto de *strings* "v_s" e f_s".

O método *SubmetePar* modela uma caixa de diálogo onde o usuário no papel de *Sonda* é capaz de selecionar um objeto previamente editado e associá-lo a uma das crenças de *conj_cren-*

cas. Esta seleção pode ser cancelada e uma nova pode ser feita quantas vezes desejar o usuário. Estando seguro de sua escolha, o usuário pode pressionar um botão *enviar* (modelado pela transição *envia*) para que o método *RecebeObjetos* do módulo *AgenteAprendiz* seja invocado e o par selecionado seja recebido por este módulo. Como para todos os métodos modelando caixas de diálogo nesta especificação, o disparo de uma transição *cancela* encerra a execução do método sem que, neste caso, par algum seja enviado ao módulo *AgenteAprendiz*.

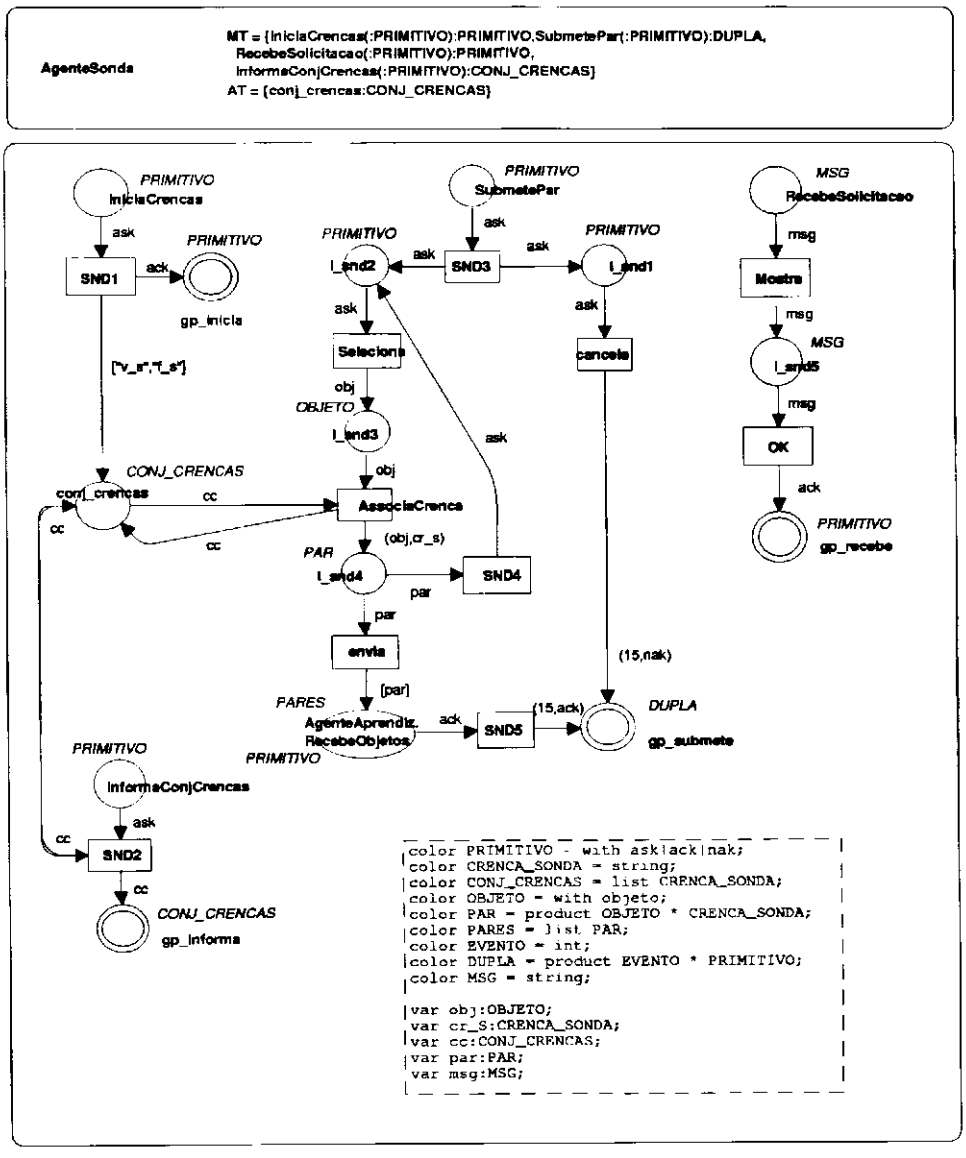


Figura 5.6: G-CPN *AgenteSonda*.

O método *RecebeSolicitacao* modela uma caixa de mensagem. A mensagem a ser exibida é enviada pelo módulo *AgenteAprendiz* e sua exibição é modelada por este método do módulo *AgenteSonda*. A mensagem deverá ser exibida até que o usuário dê um *click* em um botão *OK*.

A respeito do conjunto de cores *OBJETO*, valem as ressalvas feitas sobre este conjunto na

seção 5.4.1.2.

5.4.1.5 G-CPN *AgenteAprendiz*

O módulo *AgenteAprendiz* é composto pelas páginas que são mostradas nas figuras de 5.7 a 5.17. Como este módulo modela uma quantidade significativa de métodos, tornando extenso o GSP do módulo, decidiu-se mostrar a declaração dos métodos e atributos apenas na Figura 5.17.

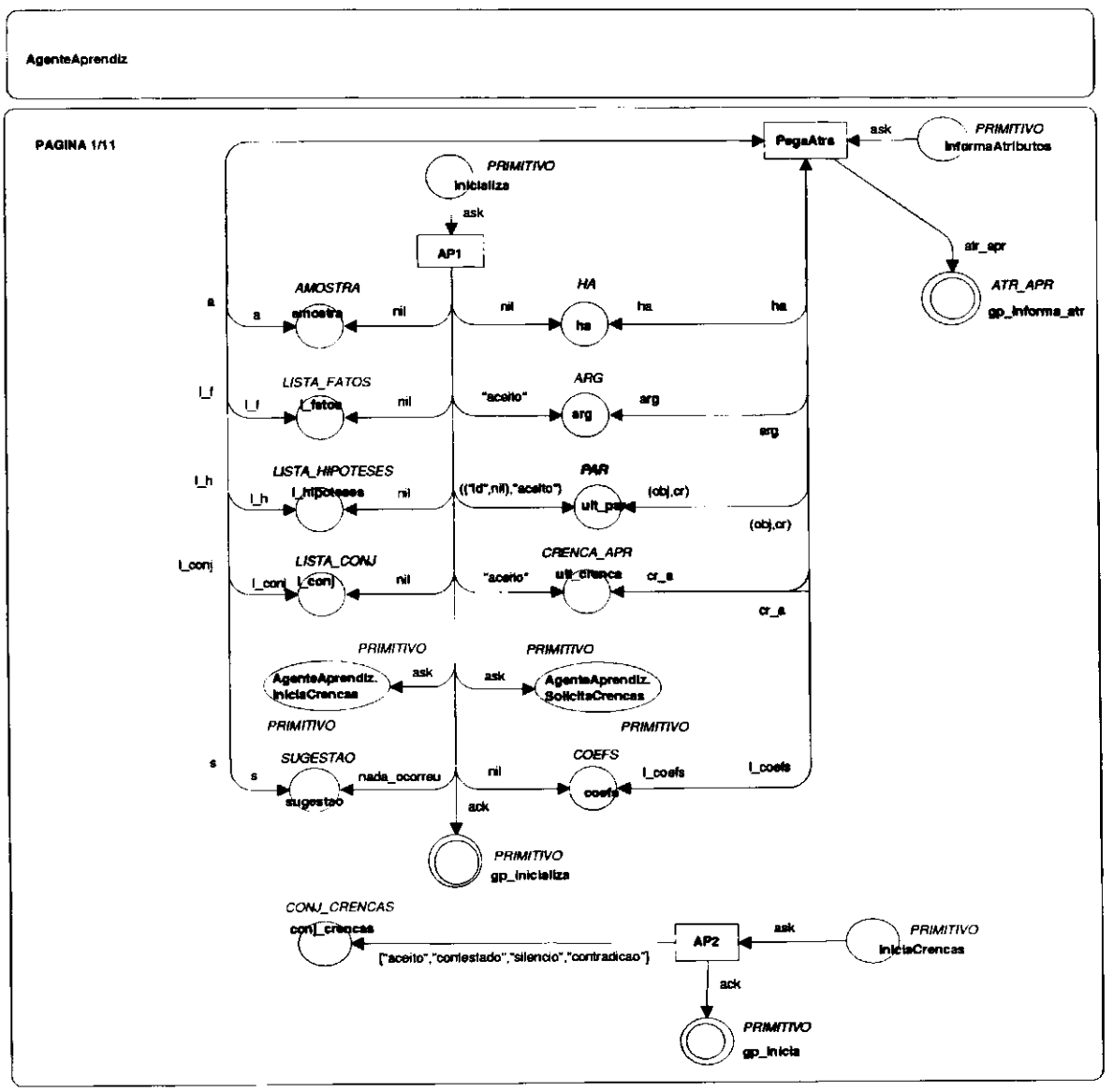


Figura 5.7: G-CPN *AgenteAprendiz* (Página 1).

O método *Inicializa* (Figura 5.7) modela a inicialização dos atributos de *AgenteAprendiz*. Este método é invocado por *ProtocoloMosca*. A definição de cada atributo é mostrada no GSP na Figura 5.17. O método invoca ainda outros dois métodos: (*i*) *IniciaCrencas* (Figura 5.7), que atri-

bui uma lista de *strings* representando o conjunto de crenças do *Aprendiz* (segundo sistema de crenças mostrado no Capítulo 3) ao atributo *conj_crencas* e (*ii*) *SolicitaCrencas* (Figura 5.8), para conhecer o conjunto de crenças do demais agentes do protocolo MOSCA.

InformaAtributos (Figura 5.7) modela o envio ao chamador dos valores nos atributos de *AgenteAprendiz*. Como é mostrado no nó de declaração (Figura 5.17), a ficha de retorno deste método carrega uma n-tupla contendo como elementos os atributos de *AgenteAprendiz*. Este método é invocado, por exemplo, pelo módulo *Arquivo* (seção 5.4.5) para gravar uma determinada situação do sistema.

O método *SolicitaCrencas* (Figura 5.8) invoca os métodos *InformaConjCrencas* de cada um dos módulos *AgenteCliente*, *AgenteOraculo*, *AgenteSonda* e *AgenteMestre* e deposita as fichas carregando o resultado de cada retorno nos lugares-atributo *ccO*, *ccC*, *ccS* e *ccM*. A ficha em cada lugar-atributo é utilizada pelo método *RecebeObjetos*, explicado a seguir.

O método *RecebeObjetos* (Figura 5.8) é invocado por *AgenteCliente*, *AgenteSonda* e *AgenteOraculo* sempre que o usuário, no papel de *Cliente*, *Sonda* ou *Oráculo* enviar um conjunto de objetos ao *Aprendiz* (que é, respectivamente, *vazio*, um *par <objeto,crença>* ou uma *amostra*). *RecebeObjetos* identifica o chamador, verificando a qual dos conjuntos de crenças pertence a crença associada ao objeto recebido, e realiza a(s) operação(ões) relativa(s) a cada caso.

O método *ReformulaAmostra* (Figura 5.9) é invocado pelo método *RecebeObjetos* sempre que um objeto é recebido de *AgenteOraculo*. Se *AgenteAprendiz* não possui amostra alguma, este método atribui o conjunto de pares recebido ao atributo amostra. Caso contrário, uma pergunta é lançada ao usuário para que este decida sobre desprezar o conjunto de pares corrente na amostra ou juntá-lo ao novo conjunto recebido. Ao simular a rede, esta escolha é feita disparando-se uma das transições [juntar="s"] ou [juntar="n"]. A inscrição no arco de saída de cada transição modela a realização da operação associada à opção do usuário.

O método *BuscaCrenca* (Figura 5.9) é invocado pelo método *RecebeObjetos* sempre que um objeto é recebido de *AgenteCliente* ou de *AgenteSonda*. Este método deverá retornar uma crença que poderá ser obtida de duas maneiras distintas: ou o objeto para o qual se deseja uma crença já existe na amostra ou não. *AgenteAprendiz.ProcuraNaAmostra* é invocado para realizar esta verificação. Se o objeto já existe na amostra, a crença retornada é a mesma já associada ao objeto. Se o objeto não existe na amostra, *AgenteAprendiz.AplicaIIA* é invocado para gerar a crença desejada, que é retornada a *RecebeObjetos*.

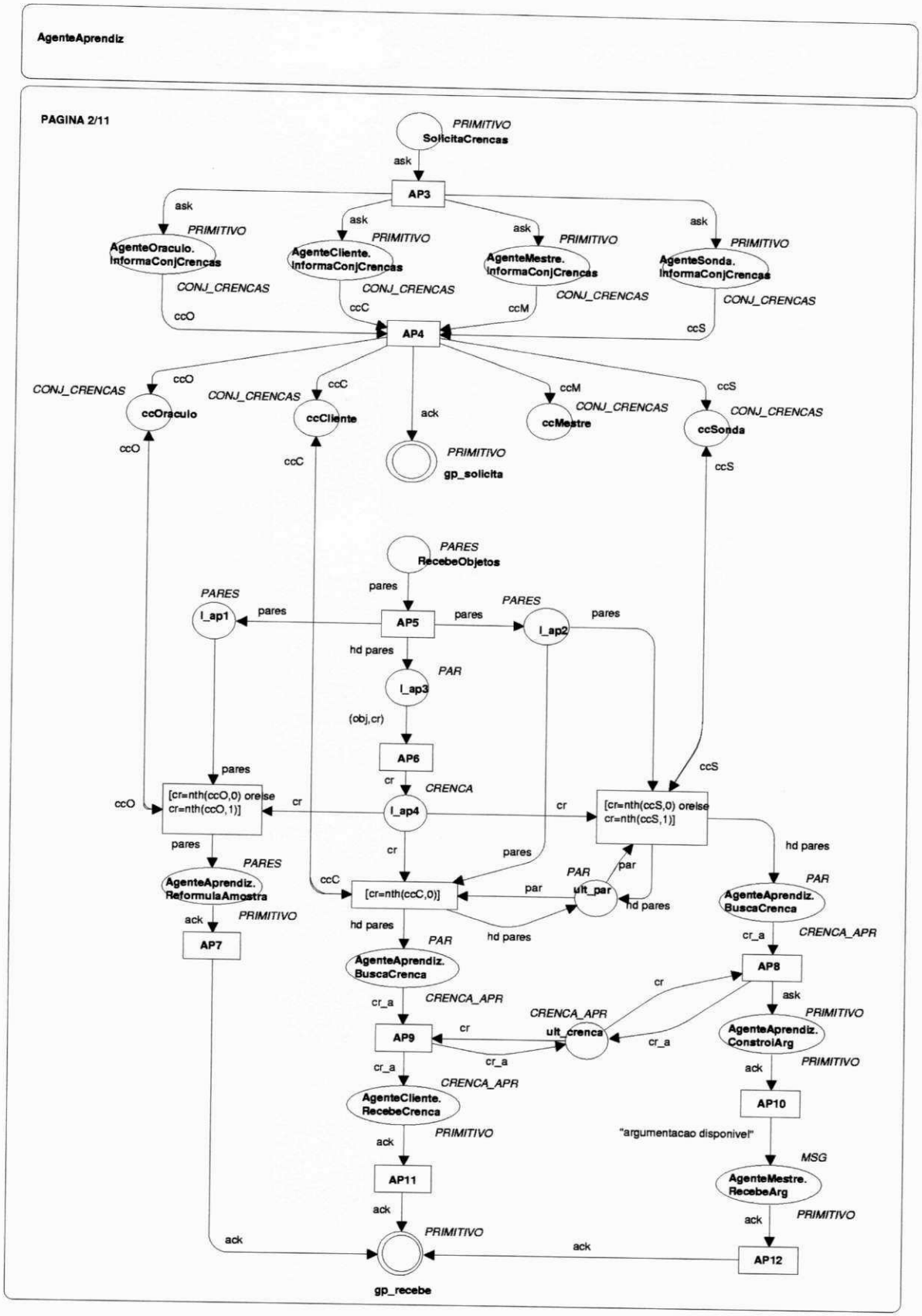


Figura 5.8: G-CPN *AgenteAprendiz* (Página 2).

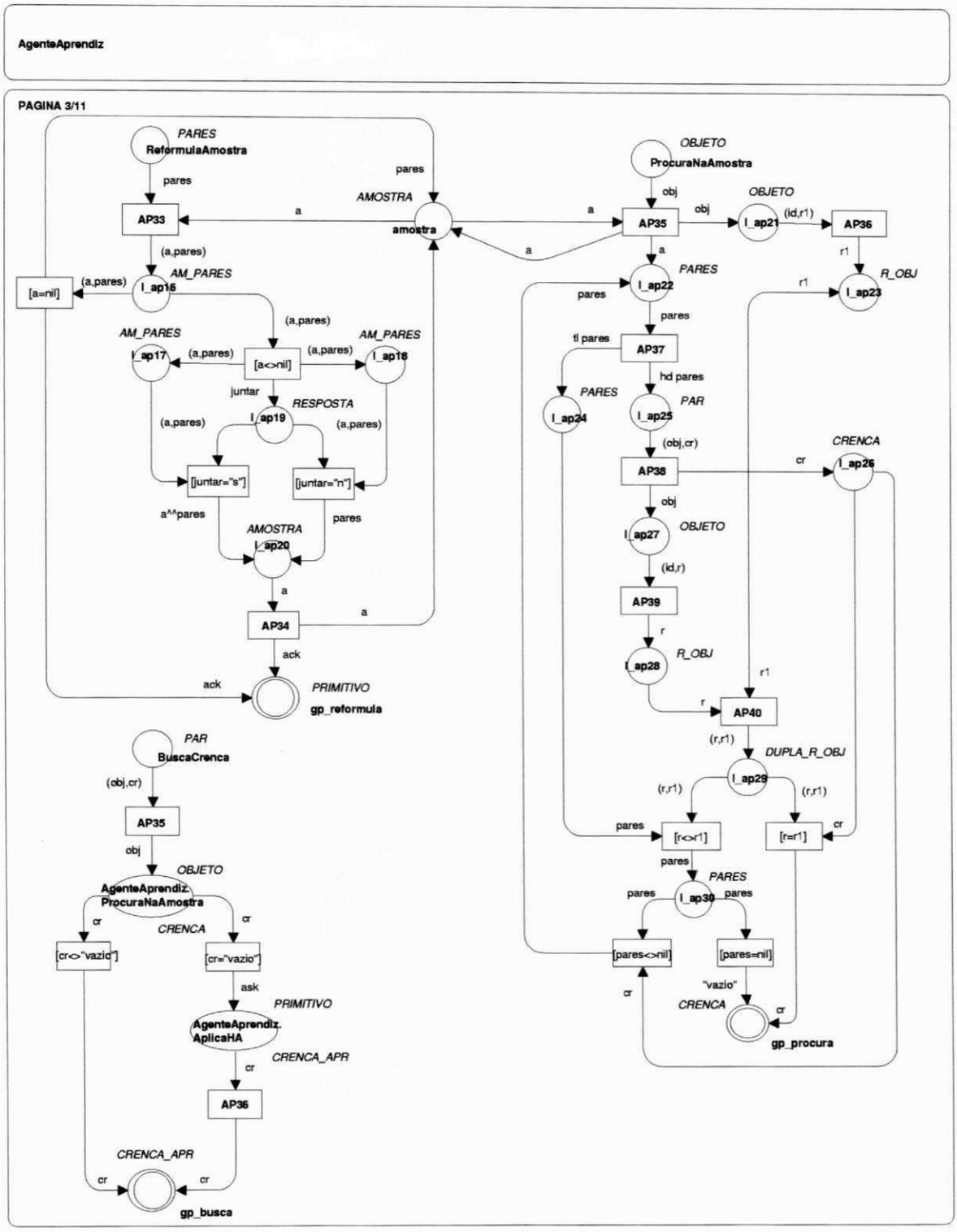


Figura 5.9: G-CPN *AgenteAprendiz* (Página 3).

O método *ProcuraNaAmostra* (Figura 5.9) é invocado pelo método *BuscaCrenca* e recebe como parâmetro de entrada um objeto (um exemplo ou contra-exemplo do conceito), devendo retornar a crença associada a este objeto, se ele fizer parte da amostra, ou o *string* "vazio", indicando que o objeto não foi encontrado na amostra.

Os métodos *GeraFatos*, *GeraHipoteses*, *GeraConjecturas* e *GeraHA* (Figura 5.10) são responsáveis pela geração do conhecimento do *Aprendiz*. Tal geração pode ser vista como consistindo dos passos: (*i*) busca de similaridades entre os objetos da amostra e (*ii*) construção de um objeto generalizante. Estes passos são sucintamente descritos a seguir.

Busca de similaridades na amostra

Este passo deve identificar os elementos estruturais comuns (ditos similaridades ou regularidades) entre os objetos submetidos ao sistema através da amostra. Tal identificação em SAID será provida pelo método S3O (*Similarity Search on Structured Objects*) [CARVALHO, 1997], que é capaz de extrair similaridades a partir de uma amostra composta de exemplos e contra-exemplos de um conceito sendo aprendido, sendo tal amostra composta de objetos estruturados. O método S3O deverá construir dois conjuntos de enunciados majoritariamente válidos: *enunciados de assimilação*, formado por enunciados comumente encontrados nos componentes da amostra e *enunciados de discriminação*, formado por enunciados comumente encontrados nos exemplos e raramente encontrados nos contra-exemplos. Para a construção destes conjuntos, o método S3O percorre dois passos: identificação de enunciados elementares da amostra e construção, a partir desses enunciados elementares, dos enunciados que representam as similaridades. Os enunciados assim obtidos são chamados *fatos*. No módulo *AgenteAprendiz*, a realização deste passo é modelada pela transição *S3OA* (Figura 5.10). Os atributos de *MecanismoAbducao* (parâmetros necessários à execução do passo) entram na transição pelo arco rotulado por *pab*, carregando a ficha de retorno de *MecanismoAbducao.InformaAtributos*. Entra também na transição o filtro contendo classes, atributos e valores previamente selecionados pelo usuário como pertinentes na aprendizagem do conceito em questão. Como entrada ainda da transição, mostra-se na rede a amostra na qual o método buscará regularidades.

Construção de um objeto generalizante

Este passo divide-se em três sub-passos, uma vez que *objeto generalizante* refere-se a três níveis de regras: as *hipóteses*, que são geradas a partir dos fatos; as *conjecturas*, geradas a partir das hipóteses e a *Hipótese Aprendida*, gerada a partir das conjecturas. O processo de geração de cada nível é também provido pelo método S3O e, na rede, é representado por *S3OB*, *S3OC* e *S3OD* (Figura 5.10), respectivamente, para a geração de *hipóteses*, *conjecturas* e *Hipótese Aprendida*. Como entrada para cada transição, tem-se o conjunto de regras gerado pelo passo anterior e os atributos de *MecanismoInducao*.

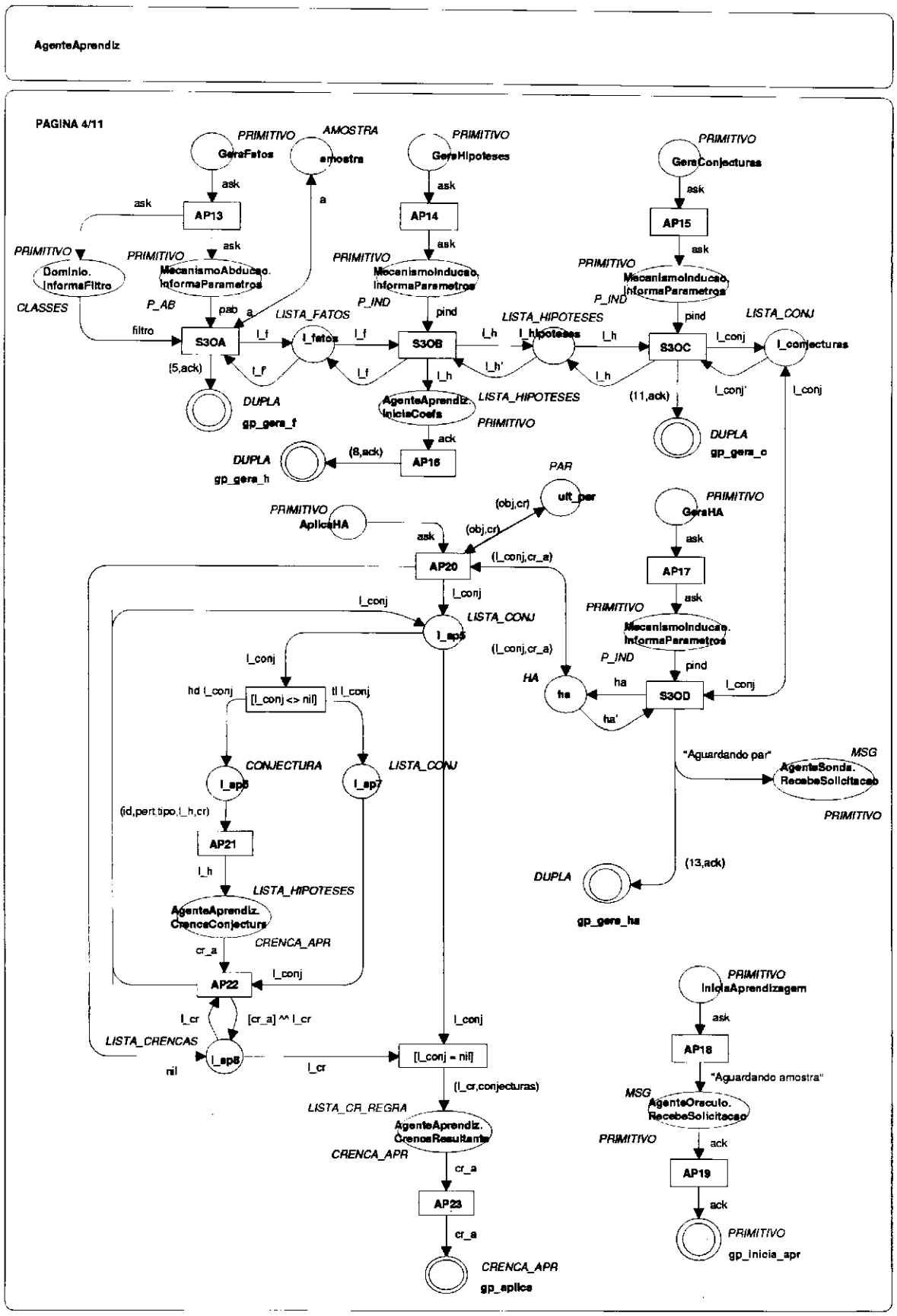


Figura 5.10: G-CPN *AgenteAprendiz* (Página 4).

Observe-se que a transição S3OB ao gerar a lista de hipóteses invoca o método *IniciaCoefs* (que é mostrado na Figura 5.15) para inicializar os coeficientes das hipóteses geradas. Observe-se ainda que a transição *S3OD*, se dispara, invoca *AgenteSonda.RecebeSolicitacao* para sinalizar ao usuário no papel de *Sonda* que o sistema dispõe de uma Hipótese Aprendida e aguarda par <*objeto, crença*> para iniciar a etapa de revisão.

O método *AplicaIIA* (Figura 5.10) modela a aplicação da Hipótese Aprendida gerada, mediante a submissão de um objeto (pelo usuário no papel *Cliente* ou de *Sonda*). Este método deve obter uma crença, armazená-la no lugar-atributo *ult_par* e retorná-la ao chamador (*AgenteAprendiz.BuscaCrenca*). Para obter a crença, o método varre a lista de conjecturas e obtém a crença associada a cada conjectura, através de *AgenteAprendiz.CrencaConjectura* (Figura 5.11) e, uma vez obtidas todas as crenças da lista, *AgenteAprendiz.CrencaResultante* (Figura 5.12) é invocado para, por lógica majoritária, definir a crença geral (ver seção 3.6.2). Analogamente, o método *CrencaHipotese* faz uma varredura para obter cada crença, só que na lista de hipóteses. O método *CrencaFato* (Figura 5.12) é invocado para decidir a crença na lista de nível mais baixo. Uma ressalva é a seguinte: em se tratando de uma implementação utilizando-se a linguagem *Prolog*, a Hipótese Aprendida pode ser vista como uma regra como a mostrada no exemplo do *Anexo C*. Neste caso, uma lista de variáveis deve fazer parte de cada regra, possibilitando a instanciação do objeto para o qual se deseja uma crença.

O método *IniciaAprendizagem* (Figura 5.10) modela o envio de uma mensagem ao usuário no papel de *Oráculo* pela invocação do método *AgenteOraculo.RecebeSolicitacao*. A modelagem deste último método é mostrada na seção 5.4.1.3.

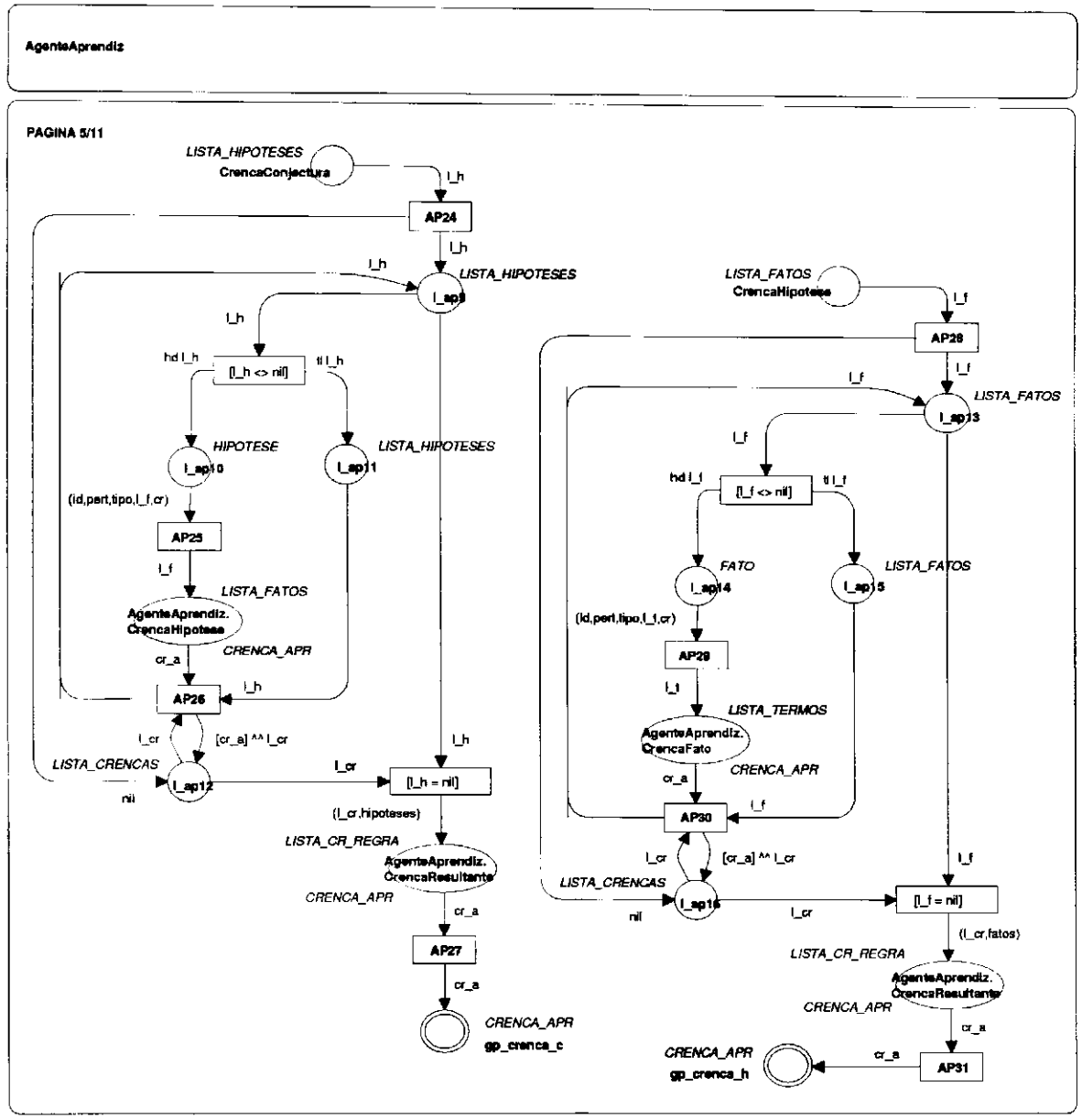


Figura 5.11: G-CPN *AgenteAprendiz* (Página 5).

No mecanismo de construção de argumentação adotado, uma argumentação consiste de um elemento de *ConjCrencas* de *AgenteAprendiz*, indicando se a crença do sistema coincide com a crença do usuário (no papel de *Sonda*) para o mesmo objeto. Dessa forma, o método *ConstroiArg* (Figura 5.13) compara a crença gerada pela última aplicação da Hipótese Aprendida com a crença associada ao último par <objeto,crença> recebido. Observe-se que este último par pode, no entanto, ter sido enviado pelo módulo *AgenteCliente* (caso em que a crença é o *string* "vazio") e a crença que o método associa ao atributo *arg* é "aceito", sinalizando uma argumentação do tipo explicação.

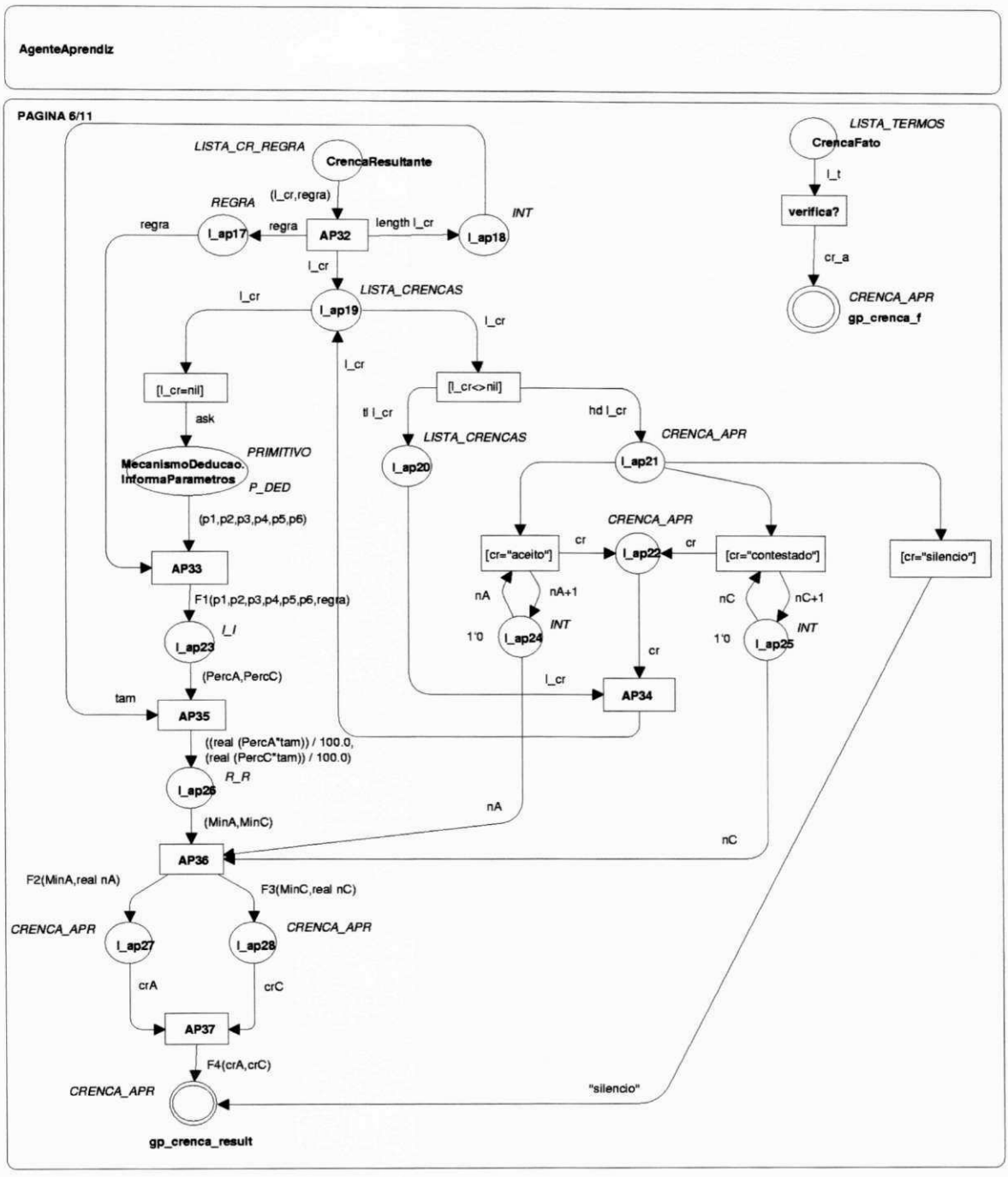


Figura 5.12: G-CPN *AgenteAprendiz* (Página 6).

O método *InformaArg* (Figura 5.13) retorna ao módulo cliente a cor associada ao lugar-atributo *arg*.

O método *RecebeCritica* (Figura 5.14) é invocado pelo método *Critica* de *AgenteMestre*. Este método recebe como parâmetro de entrada uma lista de críticas a hipóteses e invoca *AgenteAprendiz.ExploraCritica*, passando esta lista de críticas como parâmetro.

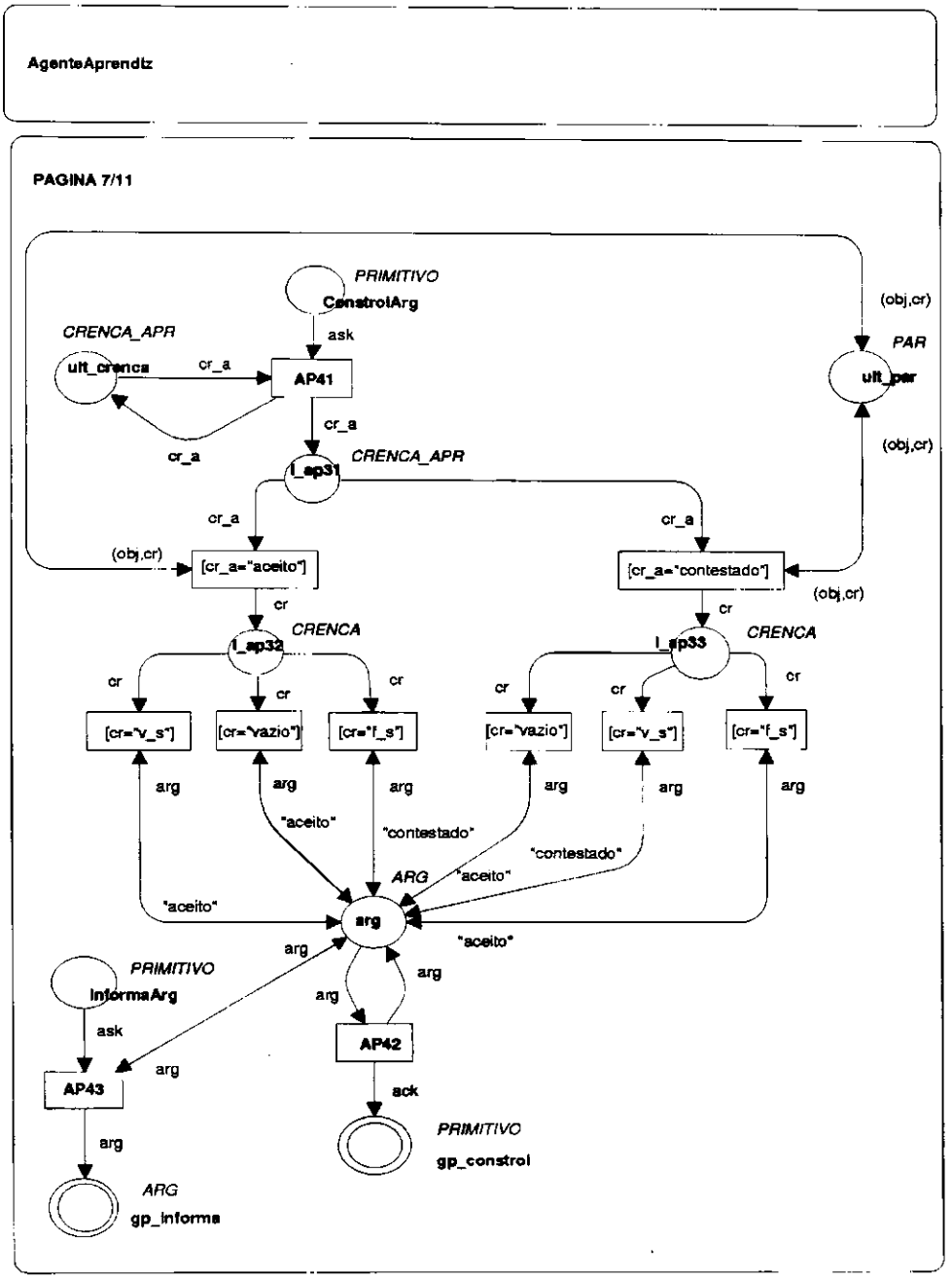


Figura 5.13: G-CPN *AgenteAprendiz* (Página 7).

O método *ExploraCritica* (Figura 5.14) realiza uma varredura na lista de críticas a hipóteses recebida como parâmetro de entrada e, para cada elemento da lista, invoca *AgenteAprendiz.AjustaCoef* para incrementar ou decrementar o coeficiente associado à hipótese, de acordo com a crítica que a hipótese recebeu. Após varrer toda a lista, *ExploraCritica* invoca *AgenteAprendiz.TodasLemas* para verificar a condição de lema de todas as hipóteses. Após o retorno do método, *AgenteAprendiz* armazena no atributo *sugestao* uma sugestão em função deste retorno, através da chamada a *AgenteAprendiz.GuardaSugestao*.

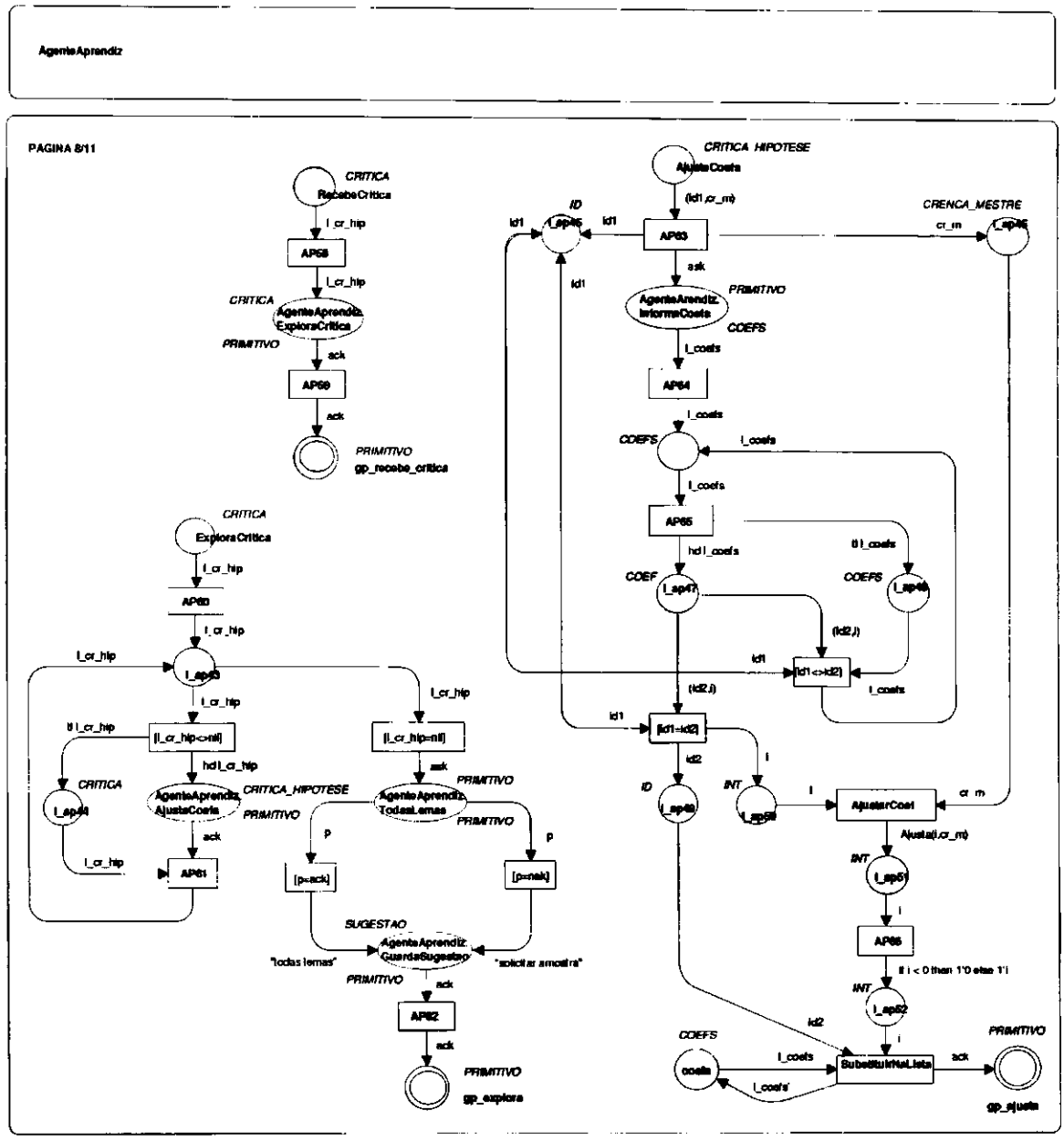


Figura 5.14: G-CPN *AgenteAprendiz* (Página 8).

O método *AjustaCoef* (Figura 5.14) recebe como parâmetro de entrada uma dupla contendo um identificador de hipótese (um *string*) e um elemento de *ConjCrencas* de *AgenteMestre*. A tarefa associada a este método é o ajuste do coeficiente da hipótese cujo identificador é o primeiro elemento da dupla recebida. O coeficiente nunca deve ser inferior a zero, ainda que uma crítica positiva tenha sido atribuída a uma hipótese que já é lema (a qual já tem seu coeficiente igual a zero).

O método *IniciaCoefs* (Figura 5.15) cria uma lista onde cada elemento é uma dupla contendo um identificador de hipótese (um *string*) e um coeficiente (um número inteiro). Este coeficiente atribui a condição de lema à hipótese associada. Assume-se, neste caso que, uma vez gera-

da uma Hipótese Aprendida, todas as suas hipóteses são lemas. Ou seja, o conhecimento adquirido é válido. Dessa forma, tenta-se, através da fase de revisão, fornecer a *AgenteAprendiz* algum exemplo ou contra-exemplo do conceito que refute esta validade. Isto será refletido no sistema pelo surgimento de críticas negativas às hipóteses, de forma que elas percam sua condição de lema (após a exploração da crítica por *AgenteAprendiz*).

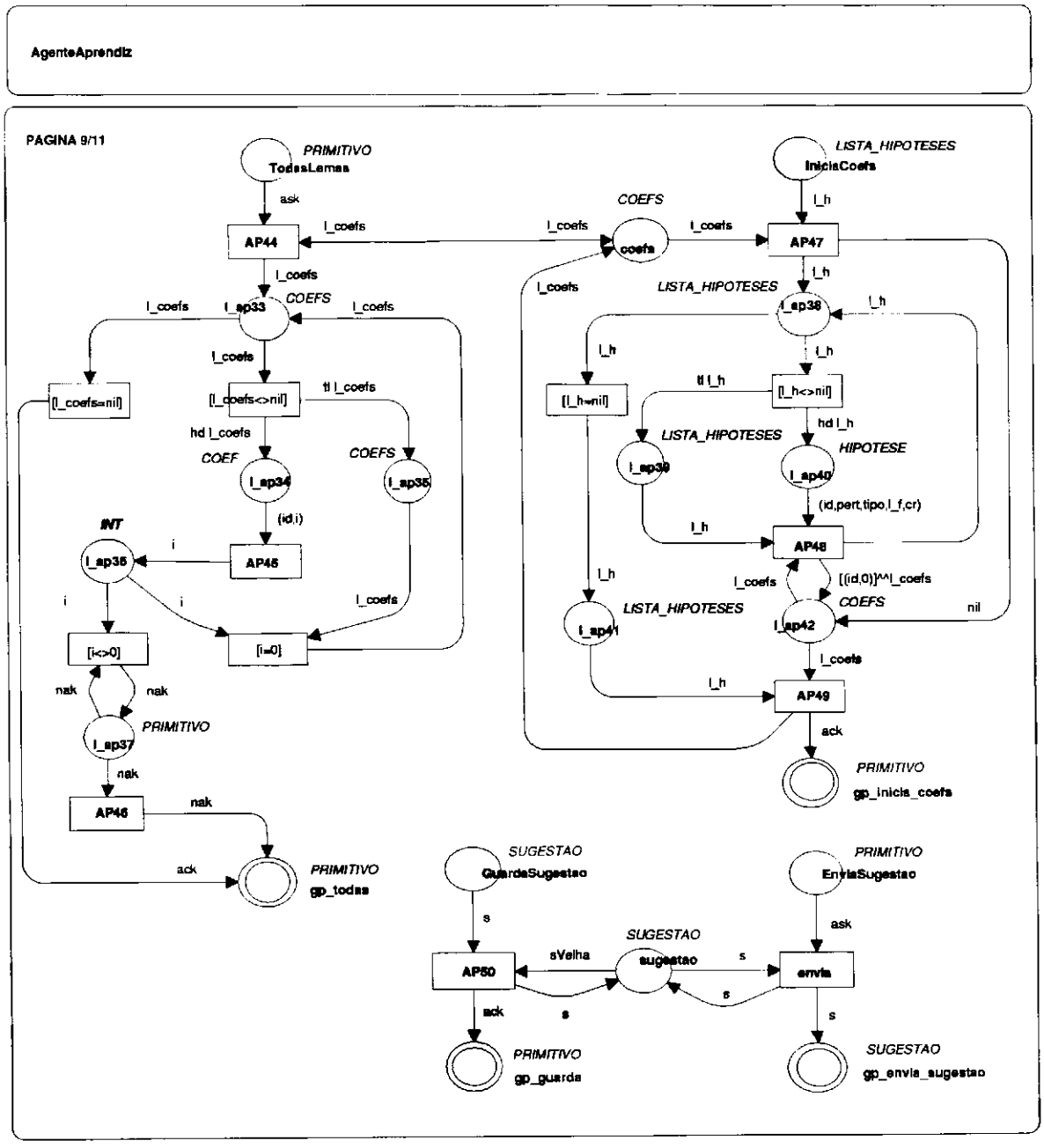


Figura 5.15: G-CPN *AgenteAprendiz* (Página 9).

O método *TodasLemas* (Figura 5.15) realiza uma verredura na lista de coeficientes das hipóteses para verificar se todas elas estão na condição de lema. Ou seja, se todas têm coeficientes

iguais a zero. Em caso afirmativo, uma ficha com a cor *ack* será depositada no lugar de terminação do método. Caso contrário, uma ficha com a cor *nak* é depositada neste lugar.

O método *GuardaSugestao* (Figura 5.15) é invocado por *AgenteAprendiz.ExploraCritica*, recebendo como parâmetro de entrada um *string* contendo uma sugestão de *AgenteAprendiz* à decisão do usuário no papel de *Mestre* sobre a fase de revisão. Este *string* é armazenado no atributo *sugestao*.

EnviaSugestao (Figura 5.15) é invocado pelo módulo *AgenteMestre* na execução do método *AnalisaHA* (seção 5.4.1.6). Este método retorna a *AgenteMestre* a cor armazenada no atributo *sugestao*.

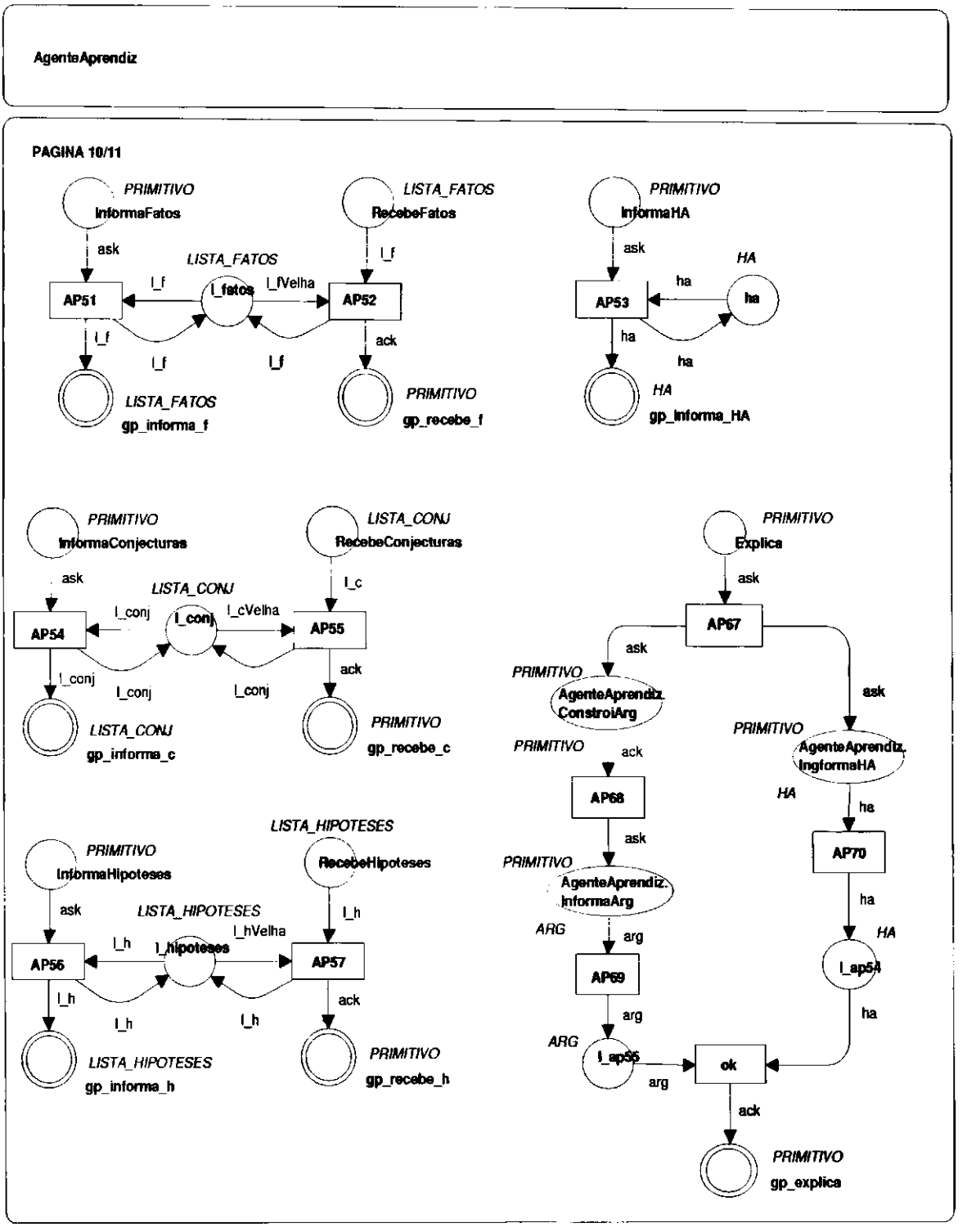


Figura 5.16: G-CPN *AgenteAprendiz* (Página 10).

Os métodos *InformaFatos*, *InformaHipoteses* e *InformaConjecturas* (Figura 5.16) levam ao módulo cliente, respectivamente, uma ficha com a cor armazenada nos atributos *l_fatos*, *l_hipoteses* e *l_conj*. O módulo *AgenteMestre* invoca estes métodos para recuperar as listas de regras.

Os métodos *RecebeFatos*, *RecebeHipoteses* e *RecebeConjecturas* (Figura 5.16) recebem como parâmetro de entrada uma ficha carregando uma nova lista de, respectivamente, fatos, hipóteses e conjecturas. Estes métodos são também invocados pelo módulo *AgenteMestre* em seus métodos de seleção de regras pertinentes para devolver a lista de regras possivelmente modificada ao módulo *AgenteAprendiz*.

A interface e o nó de declaração do módulo *AgenteAprendiz* são mostrados na Figura 5.17. Vale ressaltar que o conjunto de cores *CLASSES* é definido como a cor *classes* apenas por efeito de simplificação. Uma implementação do sistema proposto deveria consider tal conjunto como o definido no nó de declaração do módulo *Dominio* (seção 5.4.3).

5.4.1.6 G-CPN *AgenteMestre*

O módulo *AgenteMestre* é composto por seis páginas, mostradas nas Figuras 5.18 a 5.23. Os métodos que este módulo modelam são: *IniciaCrencas*, *InformaConjCrencas*, *SelFatos*, *SelHipoteses*, *SelConjecturas*, *VisualizaIIA*, *RecebeArgumentacao*, *Critica* e *AnalisaIIA*. O atributo único é *conj_crencas*. O nó de declaração deste módulo é mostrado na Figura 5.23.

Os métodos *IniciaCrencas* e *InformaConjCrencas* (Figura 5.18) são estrutura e funcionalmente idênticos aos métodos de mesmo nome do módulo *AgenteCliente* (seção 5.4.1.2), exceto pela lista de *strings* que inicializa o atributo *conj_crencas*, que aqui é ["objetado", "n_objetado", "silencio"].

AgenteAprendiz

MT = (Inicializa(:PRIMITIVO):PRIMITIVO, InformaAtributos(:PRIMITIVO):ATR_APR, IniciaCrencas(:PRIMITIVO):PRIMITIVO, SolicitaCrencas(:PRIMITIVO):PRIMITIVO, RecebeObjetos(:PARES):PRIMITIVO, ReformulaAmostra(:PARES):PRIMITIVO, BuscaCrenca(:PAR):CRENCA_APR, ProcuraNaAmostra(:OBJETO):CRENCA, GeraFatos(:PRIMITIVO):DUPLA, GeraHipoteses(:PRIMITIVO):DUPLA, GeraConjecturas(:PRIMITIVO):DUPLA, GeraHA(:PRIMITIVO):DUPLA, AplicaHA(:PRIMITIVO):CRENCA_APR, IniciaAprendizagem(:PRIMITIVO):PRIMITIVO, CrencaConjectura(:LISTA_HIPOTESES):CRENCA_APR, CrencaHipotese(:LISTA_FATOS):CRENCA_APR, CrencaFato(:LISTA_TERMOS):CRENCA_APR, CrencaResultante(:LISTA_CR_REGRA):CRENCA_APR, ControlArg(:PRIMITIVO):PRIMITIVO, InformaArg(:PRIMITIVO):ARG, RecebeCritica(:CRITICA):PRIMITIVO, ExploraCritica(:CRITICA):PRIMITIVO, AjustaCoefs(:CRITICA_HIPOTESE):PRIMITIVO, TodasLemas(:PRIMITIVO):PRIMITIVO, IniciaCoefs(:LISTA_HIPOTESES):PRIMITIVO, GuardaSugestao(:SUGESTAO):PRIMITIVO, EnviaSugestao(:PRIMITIVO):SUGESAO, InformaFatos(:PRIMITIVO):LISTA_FATOS, RecebeFatos(:LISTA_FATOS):PRIMITIVO, InformaHipoteses(:PRIMITIVO):LISTA_HIPOTESES, RecebeHipoteses(:LISTA_HIPOTESES):PRIMITIVO, InformaConjecturas(:PRIMITIVO):LISTA_CONJ, RecebeConjecturas(:LISTA_CONJ):PRIMITIVO, InformaHA(:PRIMITIVO):HA, Explica(:PRIMITIVO):PRIMITIVO)

AT = [amostra:AMOSTRA, l_fatos:LISTA_FATOS, l_hipoteses:LISTA_HIPOTESES, l_conj:LISTA_CONJ, sugestao:SUGESTAO, ha:HA, arg:ARG, ult_par:PAR, ult_crenca:CRENCA_APR, coefs:COEFS, conj_crencas:CONJ_CRENCAS]

PAGINA 11/11

```
color PRIMITIVO = with ask|ack|nak;
color INT = int;
color ID = INT;
color STRING = string;
color NOME_ATR = STRING;
color NOME_TERM = STRING;
color LISTA_NOME_TERM = list NOME_TERM;
color ATR_TERM = NOME_ATR * LISTA_NOME_TERM;
color NOME_CLASSE = STRING;
color NOME_INSTANCIA = STRING;
color NOME_REF = NOME_CLASSE + NOME_INSTANCIA;
color LISTA_NOME_REF = list NOME_REF;
color ATR_REF = product NOME_ATR * LISTA_NOME_REF;
color ATRIBUTO = ATR_REF + ATR_TERM;
color LISTA_ATRIUTOS = list ATRIBUTO;
color OBJETO = product ID * LISTA_ATRIBUTOS;
color CRENCA = STRING;
color PAR = product OBJETO * CRENCA;
color PARES = list PAR;
color AMOSTRA = PARES;
color PERT = with p|n_p;
color TIPO = with ass|disc;
color TERMO = STRING;
color LISTA_TERMOS = list TERMO;
color FATO = product ID * PERT * TIPO * LISTA_TERMOS * CRENCA_APR;
color LISTA_FATOS = list FATO;
color HIPOTESE = product ID * PERT * TIPO * LISTA_FATOS * CRENCA_APR;
color LISTA_HIPOTESES = list HIPOTESE;
color CONJECTURA = product ID * PERT * TIPO * LISTA_HIPOTESES * CRENCA_APR;
color LISTA_CONJ = list CONJECTURA;
color SUGESTAO = with nada_ocorreu|solicitar_amostra|todas_lemas;
color CRENCA_APR = CRENCA;
color HA = product LISTA_CONJ * CRENCA_APR;
color ARG = CRENCA_APR;
color COEF = product ID * INT;
color COEFS = list COEF;
color CONJ_CRENCAS = list CRENCA;
color ATR_APR = product AMOSTRA * LISTA_FATOS * LISTA_HIPOTESES * LISTA_CONJ * SUGESTAO * HA * ARG * PAR * CRENCA_APR * COEFS;
color MSG = STRING;
color AM_PARES = product AMOSTRA * PARES;
color RESPOSTA = with s|n;
color R_OBJ = LISTA_ATRIBUTOS;
color DUPLA_R_OBJ = product R_OBJ * R_OBJ;
color CLASSES = with classes;
color P_AB = product INT * INT * INT * INT;
color P_IND = product INT * INT;
color P_DED = product INT * INT * INT * INT * INT * INT;
color EVENTO = INT;
color DUPLA = product EVENTO * PRIMITIVO;
color LISTA_CRENCAS = list CRENCA;
color REGRA = with fatos|hipoteses|conjecturas;
color LISTA_CR_REGRA = product LISTA_CRENCAS * REGRA;
color I_I = product INT * INT;
color REAL = real;
color R_R = product REAL * REAL;
color CRENCA_MESTRE = CRENCA;
color ID_HIPOTESE = ID;
color CRITICA_HIPOTESE = product ID_HIPOTESE * CRENCA_MESTRE;
color CRITICA = list CRITICA_HIPOTESE;

fun F1(fa,fc,ha,hc,ca,cc:INT,regra:REGRA) = case regra of fatos=>(fa,fc)|hipoteses=>(ha,hc)|conjecturas=>(ca,cc);
fun F2(a:REAL,b:REAL) = if a>=b then 1`"aceito" else 1`"silencio";
fun F3(a:REAL,b:REAL) = if a>=b then 1`"contestado" else 1`"silencio";
fun F4(cr1,cr2) = if cr1="aceito" andalso cr2="contestado" then 1`"silencio" else if cr1="aceito" then 1`"aceito" else 1`"contestado";
fun ajusta(i:INT,cr_m:STRING) = if cr_m="objetado" then 1`(i+1) else if cr_m="nao_objetado" then 1`(i-1) else 1`i;

var p:PRIMITIVO;
var p1,p2,p3,p4,p5,p6:INT;
var tam,MinA,MinC,nA,nC,i,id1,id2:INT;
var obj:OBJETO;
var cr:CRENCA;
var par:PAR;
var pares:PARES;
var a:AMOSTRA;
var pert:PERT;
var tipo:TIPO;
var l_t:LISTA_TERMOS;
var l_f,l_fVelha:LISTA_FATOS;
var l_h,l_hVelha:LISTA_HIPOTESES;
var l_conj,l_cVelha:LISTA_CONJ;
var s,sVelha:SUGESTAO;
var cr_a,crA,crC: CRENCA_APR;
var arg:ARG;
var l_coefs,l_coefsVelha:COEFS;
var ccO,ccC,ccM,ccS:CONJ_CRENCAS;
var atr_apr:ATR_APR;
var juntar:RESPOSTA;
var r,r1:R_OBJ;
var filtro:CLASSES;
var pab:P_AB;
var pind:P_IND;
var pded:P_DED;
var l_cr:LISTA_CRENCAS;
var regra:REGRA;
var PercA,PercC:INT;
var cr_m:CRENCA_MESTRE;
var l_cr_hip:CRITICA;
```

Figura 5.17: G-CPN *AgenteAprendiz* (Página 11).

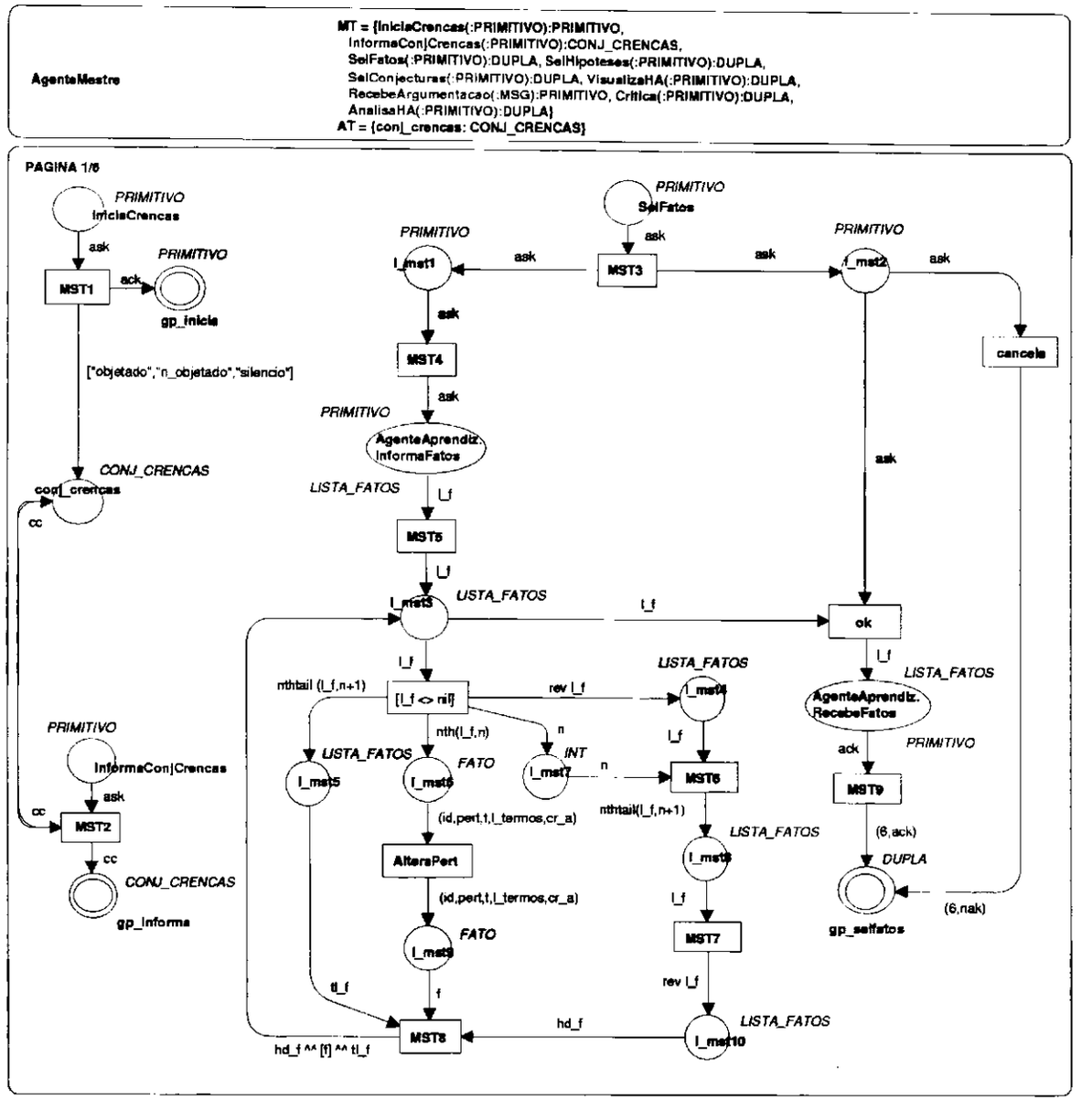


Figura 5.18: G-CPN *AgenteMestre* (Página 1).

O método *SelFatos* (Figura 5.18) pode ser visto como uma caixa de diálogo onde o usuário dispõe da lista de fatos gerados pelo sistema. Cada fato da lista é modelado como uma lista de elementos, dentre os quais, seu identificador e um indicador de pertinência, que pode receber um dos valores *p* ou *n_p* (indicando, respectivamente, fato "pertinente" e fato "não pertinente"). Ao gerar os fatos, o sistema atribui a este indicador o valor *p*. Na rede, quando a ficha carregando a lista de fatos encontra-se disponível no lugar *l_mst3*, a transição rotulada pela guarda [l_f<>nil] é habilitada. Ou seja, se a lista é não-vazia, um n-ésimo componente pode ser selecionado (ao simular a rede, um *binding* instancia *n*). Esta seleção pode ser implementada como um *click* de mouse sobre uma caixa de lista. Na rede, uma vez selecionado um fato, a transição *AlteraPert* é habilitada e, através de *binding*, é possível instanciar a variável *pert* para alterar o indicador de pertinência do

fato selecionado. Uma ficha carregando o fato com indicador de pertinência possivelmente alterado é depositada no lugar *l_mst9* após o disparo de *AlteraPert*. A chegada de ficha em *l_mst9* habilita, juntamente com as fichas nos lugares *l_mst5* e *l_mst10*, a transição *MST8*. Uma vez que a estrutura associada ao conjunto de fatos nesta rede é uma lista e que um componente desta lista foi selecionado e possivelmente alterado, é necessário que esta alteração seja repassada à lista. Assim, junta-se tal fato à cabeça e ao resto da lista original. É o que é feito na inscrição do arco de saída de *MST8*, que leva a lista atualizada novamente ao lugar *l_mst3* (observe-se que em linguagens de programação que permitem o acesso direto a um componente de uma lista, a alteração é imediata). Deste ponto, o processo de seleção de fato e alteração de indicador de pertinência pode ser repetido. A transição *OK* pode ser disparada desde que haja ficha em *l_mst3* e seu disparo devolve ao módulo *AgenteAprendiz* a lista de fatos possivelmente alterada, através da invocação do método *RecebeFatos* do módulo.

O método *SelHipoteses* (Figura 5.19) pode ser visto como uma caixa de diálogo com funcionalidade idêntica àquela modelando o método *SelFatos*, exceto que a lista manipulada neste caso é a de hipóteses e, portanto, é o indicador de pertinência de uma hipótese que é alterado aqui.

O método *VisualizaIIA* (Figura 5.19) modela também uma caixa de diálogo. Através desta caixa, o usuário no papel de *Mestre* pode visualizar a *Hipótese Aprendida* que, ao final da fase de aprendizagem, é gerada pelo sistema. O método *InformaIIA* do módulo *AgenteAprendiz* é invocado para que a estrutura contendo a *Hipótese Aprendida* possa ser retida no que se modelou como o lugar *l_mst11* nesta rede. A forma de visualização pode ser implementada através de componentes distintos de interface para cada nível de regra que compõe a *Hipótese Aprendida* (por exemplo, um componente para conjecturas, um para hipóteses e um para fatos). O usuário pode fechar a caixa (encerrando o método) pressionando um botão *OK*, cujo evento é aqui modelado pela transição de mesmo nome, que é habilitada desde que a ficha carregando a *Hipótese Aprendida* esteja disponível no lugar l_*mst11*.

A chamada ao método *RecebeArgumentacao* (Figura 5.20) modelaria o envio de uma argumentação do *Aprendiz* ao usuário no papel de *Mestre*. Contudo, ao invés da argumentação propriamente dita, este método recebe como parâmetro apenas uma mensagem indicando a disponibilidade de uma argumentação. Esta mensagem é modelada por um *string* e a execução do método pode ser associada à exibição de uma caixa de mensagem mostrando a mensagem recebida até que o usuário no papel de *Mestre* solicite seu encerramento, pressionando um botão *OK* (cujo evento é aqui associado ao disparo da transição *OK*).

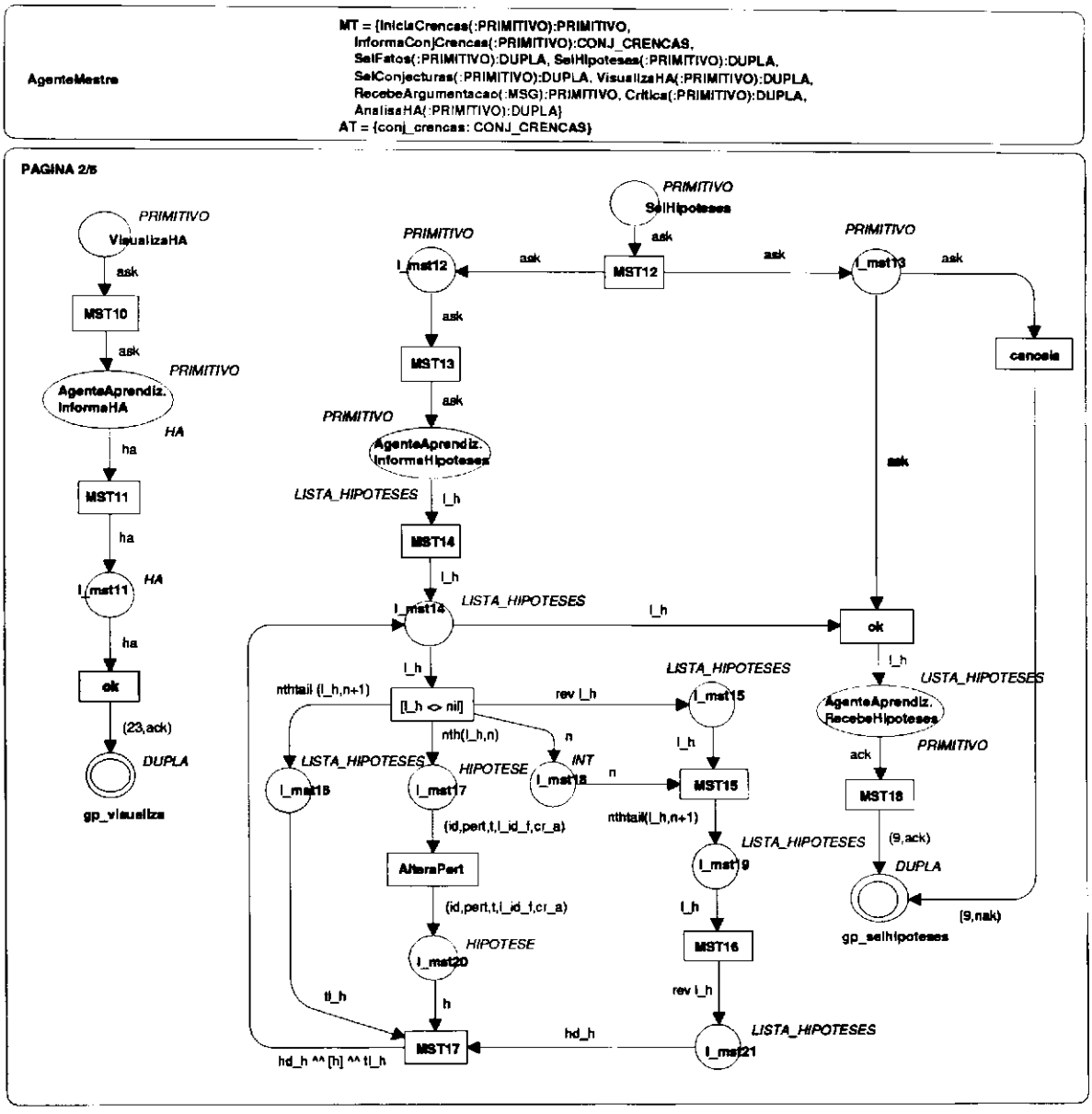


Figura 5.19: G-CPN *AgenteMestre* (Página 2).

A rede modelando o método *SelConjecturas* (Figura 5.20) é idêntica àquela modelando *SelFatos*. Porém, neste caso, a estrutura disponível para alteração é uma lista com as conjecturas geradas pelo sistema. O funcionamento desta rede é idêntico ao daquela modelando o método *SelFatos*, considerando-se a substituição da estrutura *FATO* pela estrutura *CONJECTURA*.

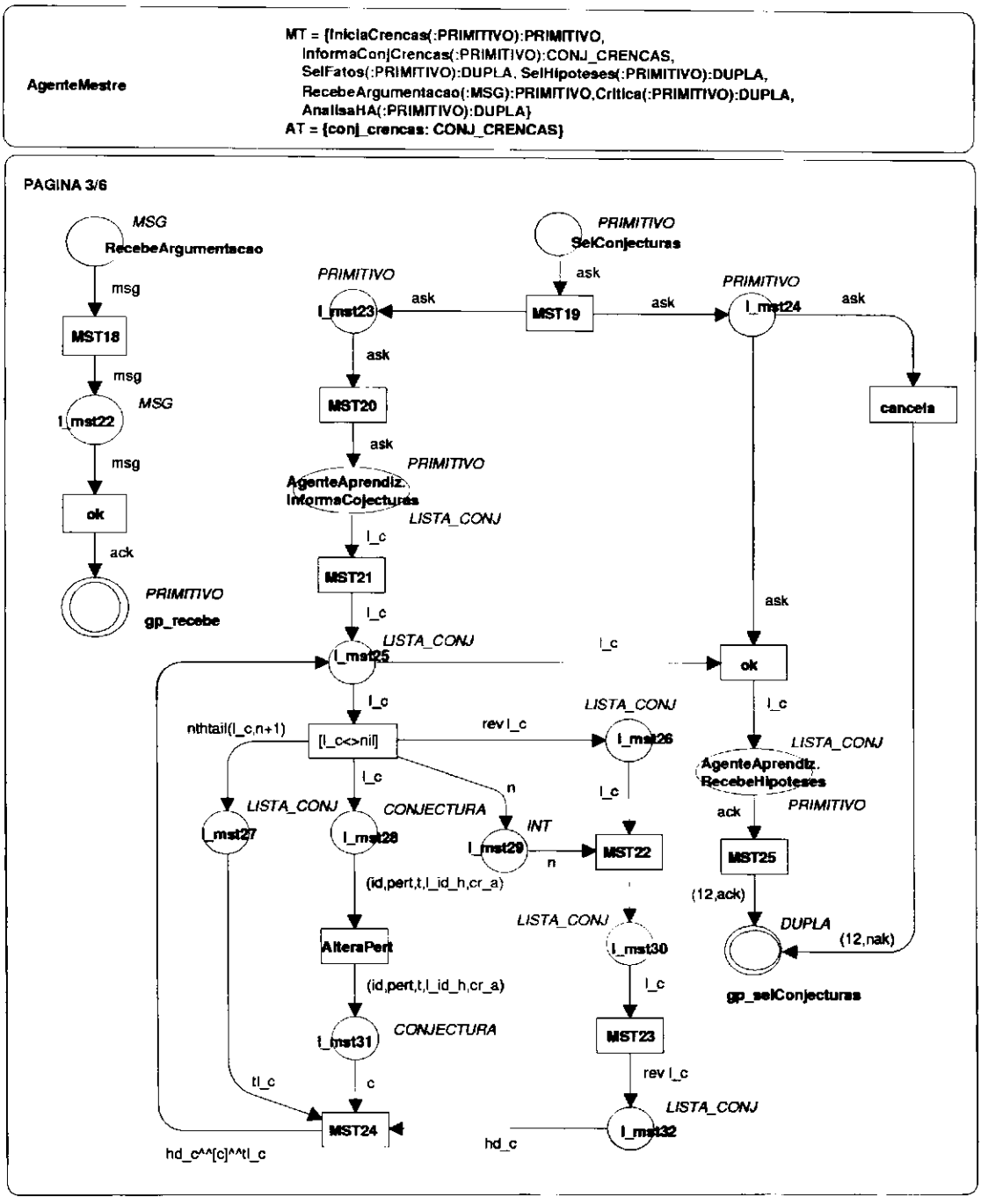


Figura 5.20: G-CPN *AgenteMestre* (Página 3).

O método *Critica* (Figura 5.21) modela uma caixa de diálogo na qual o usuário no papel de *Mestre* constrói sua crítica a uma argumentação recebida. A crença recebida como argumentação indica a posição do *Aprendiz* mediante a crença atribuída pelo usuário no papel de *Sonda* ao objeto enviado. A crença atribuída a cada regra na aplicação da *Hipótese Aprendida* ao objeto justifica a posição do *Aprendiz*. Assim, o usuário observa a *Hipótese Aprendida* e pode emitir sua crítica a cada hipótese apresentada pelo sistema. Estas hipóteses aparecem na rede, no lugar *hipóteses*, habilitando a transição *SelHip*. O disparo desta transição modela a escolha de uma hipótese (índice

do elemento na lista). A ficha carregando o elemento escolhido é depositada no lugar *hip_selecionada*. Uma vez selecionada uma hipótese, é possível criticar esta hipótese e também visualizar os fatos que a compõem. Ao criticar (disparar transição *criticar hip*), uma crença do conjunto de crenças do *Mestre* (previamente selecionada) é atribuída à hipótese (através de seu identificador), formando uma lista de críticas. A transição *MostraFatos* deve exibir no lugar *fatos* todos os fatos da lista geral (do módulo *AgenteAprendiz*, que está disponível no lugar *l_mst43*) cujos identificadores aparecem na variável *l_id_f* da hipótese selecionada.

O método *AnalisaHA* (Figura 5.22) modela uma caixa de diálogo na qual o usuário no papel de *Mestre* avalia o processo de revisão e pode decidir se continua ou não este processo. Nesta caixa, são mostrados ao usuário: a *Hipótese Aprendida* (que está sendo aplicada durante a revisão) e os coeficientes associados a cada uma das hipóteses, associando a cada uma a condição de lema ou não-lema. Além disso, o sistema sugere ao usuário, baseado na condição das hipóteses, uma decisão. O usuário pode cancelar a execução do método (pelo disparo da transição *cancela*) ou pode acatar a sugestão do sistema (pelo disparo da transição *OK*). Esta sugestão pode ser: (*i*) se todas as hipóteses são lemas, é possível passar à fase de exploração ou (*ii*) se alguma hipótese é não-lema, deve ser solicitada uma nova amostra ao oráculo. De fato, o envio de uma nova amostra representa uma volta ao primeiro passo da fase de aprendizagem e esta decisão pode ser adiada pelo usuário no papel de *Mestre*. Alternativamente, o usuário poderia retornar a etapas intermediárias da fase de aprendizagem, como a reavaliação de regras pertinentes. O disparo de cada transição associada a uma sugestão poderia ser provido em uma implementação por *click* sobre um botão, cujo rótulo seja o nome do evento associado.

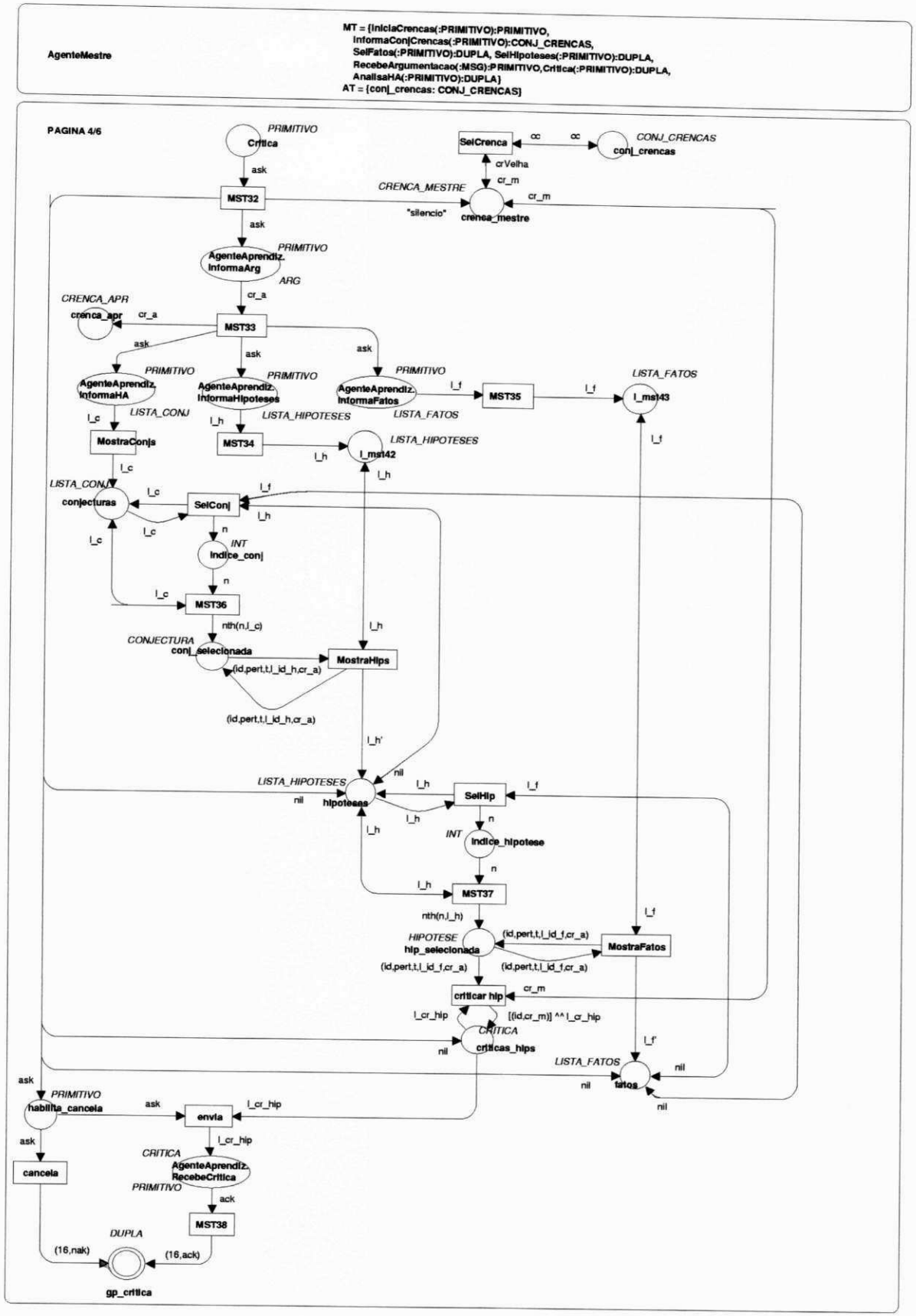


Figura 5.21: G-CPN *AgenteMestre* (Página 4).

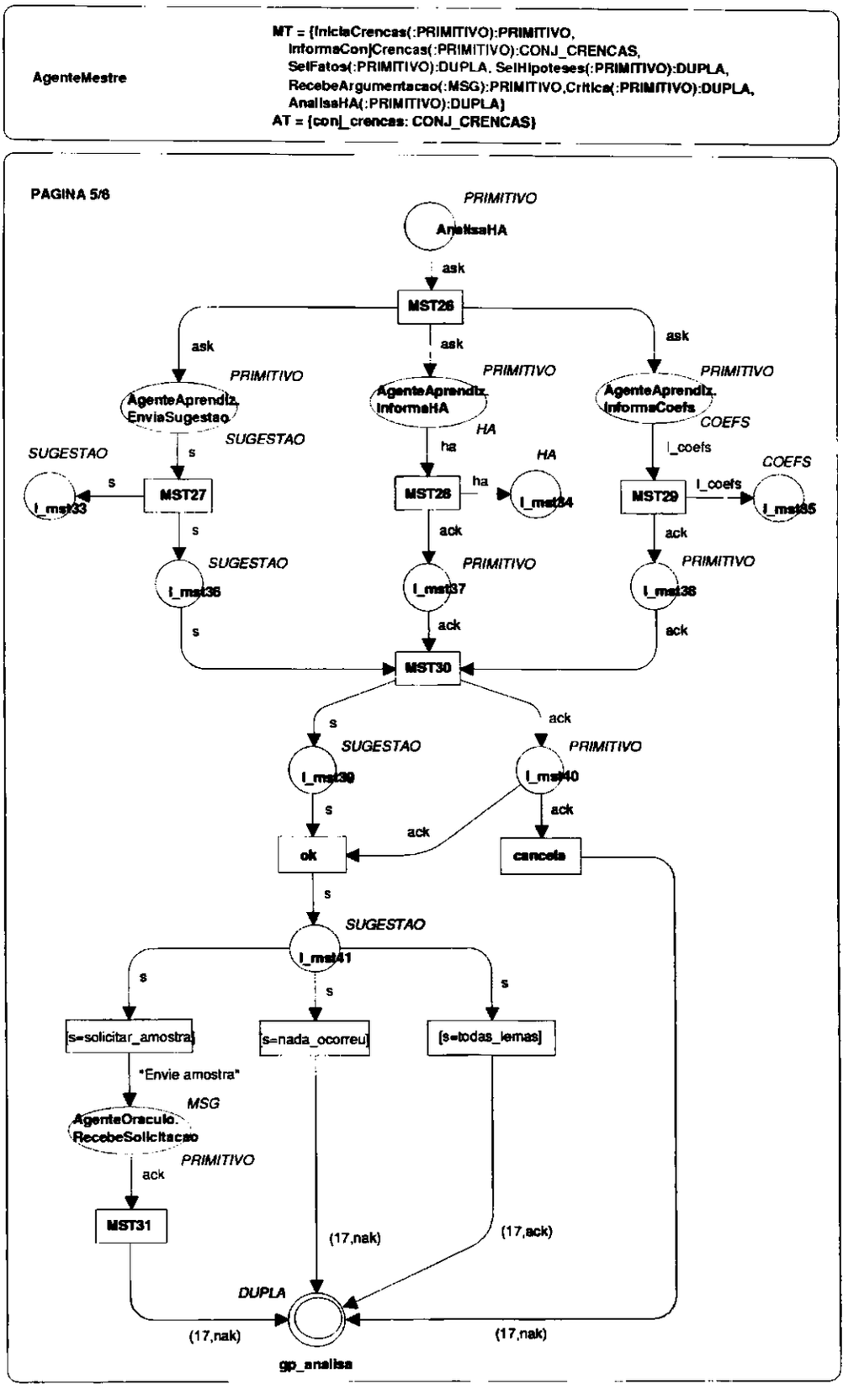


Figura 5.22: G-CPN *AgenteMestre* (Página 5).

AgenteMestre

MT = {iniciaCrencas(:PRIMITIVO):PRIMITIVO,
informaConjCrencas(:PRIMITIVO):CONJ_CRENCAS,
selFatos(:PRIMITIVO):DUPLA, selHipoteses(:PRIMITIVO):DUPLA,
recebeArgumentacao(:MSG):PRIMITIVO,critica(:PRIMITIVO):DUPLA,
analisaHA(:PRIMITIVO):DUPLA}
AT = {conj_crencas: CONJ_CRENCAS}

PAGINA 6/6

```
color PRIMITIVO = with ask|ack|nak;
color STRING = string;
color CRENCA_MESTRE = STRING;
color CRENCA_APR = STRING;
color CONJ_CRENCAS = list CRENCA_MESTRE;
color INT = int;
color ID = INT;
color PERT = with p|n_p;
color TIPO = with ass|disc;
color TERMO = STRING;
color LISTA_TERMOS = list TERMO;
color LISTA_ID_FATO = list ID;
color LISTA ID_HIPOTESE = list ID;
color LISTA_ID_CONJ = list ID;
color FATO = product ID * PERT * TIPO * LISTA_TERMOS * CRENCA_APR;
color HIPOTESE = product ID * PERT * TIPO * LISTA_ID_FATO * CRENCA_APR;
color CONJECTURA = product ID * PERT * TIPO * LISTA_ID_HIPOTESE * CRENCA_APR;
color LISTA_FATOS = list FATO;
color LISTA_HIPOTESES = list HIPOTESE;
color LISTA_CONJ = list CONJECTURA;
color HA = product LISTA_ID_CONJ * CRENCA_APR;
color MSG = STRING;
color ARG = CRENCA_APR ;
color CRITICA = list CRITICA_HIPOTESE;
color SUGESTAO = with nada_ocorreu|solicitar_amostra|todas_lemas;
color COEF = product ID * INT;
color COEFS = list COEF;
color EVENTO = INT;
color DUPLA = product EVENTO * PRIMITIVO;

var cc:CONJ_CRENCAS;
var l_f:LISTA_FATOS;
var l_h:LISTA_HIPOTESES;
var l_c:LISTA_CONJ;
var n:INT;
var id:ID;
var pert:PERT;
var t:TIPO;
var cr_a:CRENCA_APR;
var cr_m,crVelha:CRENCA_MESTRE;
var l_termos:LISTA_TERMOS;
var hd_f,tl_f:FATO;
var hd_h,tl_h:HIPOTESE;
var hd_c,tl_c:CONJECTURA;
var f:FATO;
var h:HIPOTESE;
var c:CONJECTURA;
var ha:HA;
var msg:MSG;
var l_id_f:LISTA_ID_FATO;
var l_id_h:LISTA_ID_HIPOTESE;
var l_id_c:LISTA_ID_CONJ;
var l_cr_hip:CRENCAS_HIPOTESES;
var l_coefs:LISTA_COEFS;
var s:SUGESTAO;
```

Figura 5.23: G-CPN *AgenteMestre* (Página 6).

5.4.2 Assunto Mecanismo

Os módulos neste assunto são: *MecanismoAbducao*, *MecanismoInducao* e *MecanismoDeducao*.

5.4.2.1 G-CPN *MecanismoAbducao*

O módulo *MecanismoAbducao* é mostrado na Figura 5.24. Este módulo modela os méto-

dos *Carrega*, *Altera* e *InformaParametros*. Tais métodos operam sobre os atributos: *ValMajPos*, *ValMajNeg*, *TamMaxFato* e *TamMinFato*. Os atributos deste módulo são parte dos parâmetros utilizados pelo método S3O para a identificação de regularidades na amostra (os fatos), brevemente discutido na seção 5.4.1.5. Os atributos *ValMajPos* e *ValMajNeg* são, respectivamente, os limites de validade majoritária positiva e negativa e são números inteiros representando valores percentuais. Um exemplo é mostrado no *Anexo C*. Os atributos *TamMaxFato* e *TamMinFato* são números inteiros que o método *GeraFatos* de *AgenteAprendiz* utiliza como limites para o tamanho[11] dos fatos que gera.

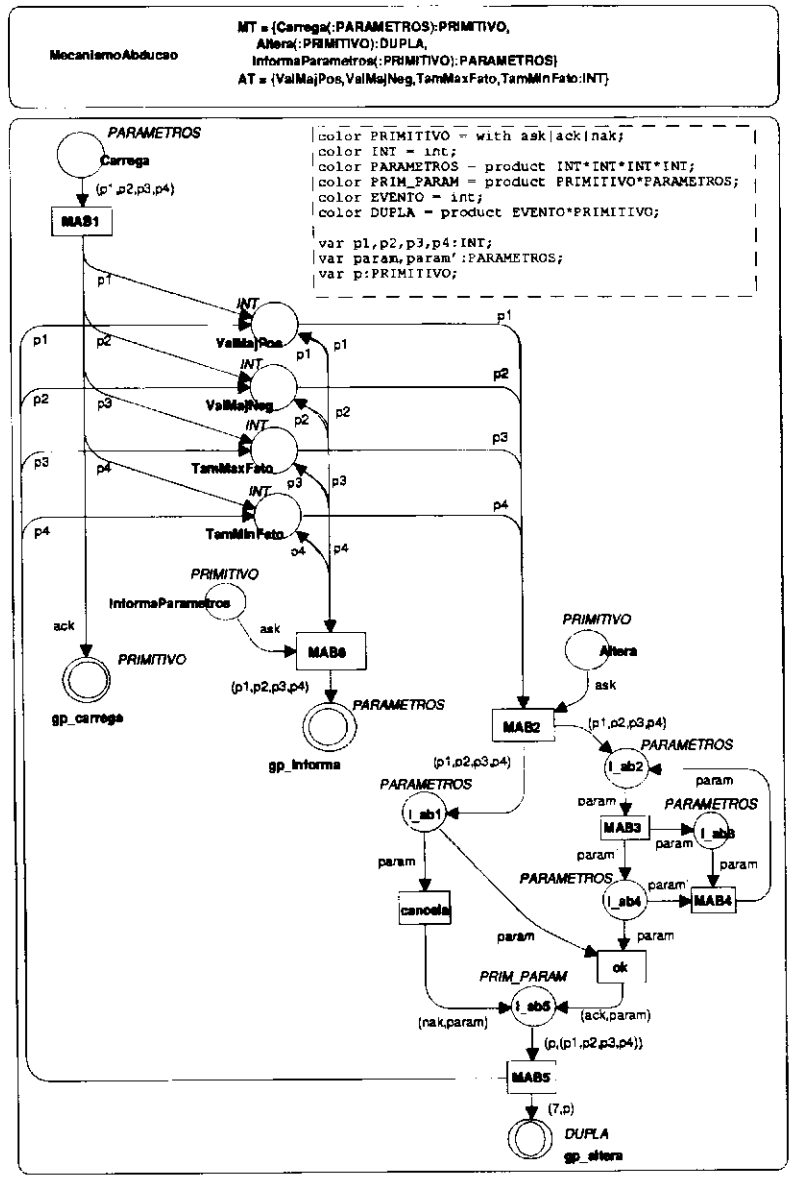


Figura 5.24: G-CPN *MecanismoAbducao*.

O método *Carrega* é invocado por *ProtocoloMosca* para atribuir valores iniciais aos atri-

[11] O *tamanho* de um fato é dado pelo número de *fatos elementares* que o compõem.

butos do módulo e também pelo método *CarregaEstruturas* (do módulo *Arquivo*), ao abrir um ambiente de aprendizagem.

Através da execução do método *Altera*, o usuário pode atribuir novos valores aos atributos do módulo.

O método *InformaAtributos* é invocado pelo módulo *Arquivo* quando o usuário solicita a gravação do ambiente de aprendizagem.

5.4.2.2 G-CPN *MecanismoInducao*

O módulo *MecanismoInducao* é mostrado na Figura 5.25. Este módulo é estrutura e funcionalmente idêntico ao módulo *MecanismoAbducao*, modelando também os métodos *Carrega*, *Altera* e *InformaParametros*. Os atributos são: *ValMajPos* e *ValMajNeg*. Tais atributos são também utilizados pelo módulo *AgenteAprendiz* na geração dos três níveis de objetos generalizantes (hipóteses, conjecturas e Hipótese Aprendida). Os métodos operam de maneira idêntica aos métodos de mesmo nome no módulo *MecanismoAbducao* (seção 5.4.2.1).

5.4.2.3 G-CPN *MecanismoDeducao*

O módulo *MecanismoDeducao* é mostrado na Figura 5.26. Os métodos modelados por este módulo operam de maneira idêntica aos métodos de mesmo nome do módulo *MecanismoAbducao* (seção 5.4.2.1). Os atributos são números inteiros e representam percentuais que são utilizados pelo módulo *AgenteAprendiz* para aplicação da *Hipótese Aprendida*, seguindo os princípios mostrados na seção 3.6.2.

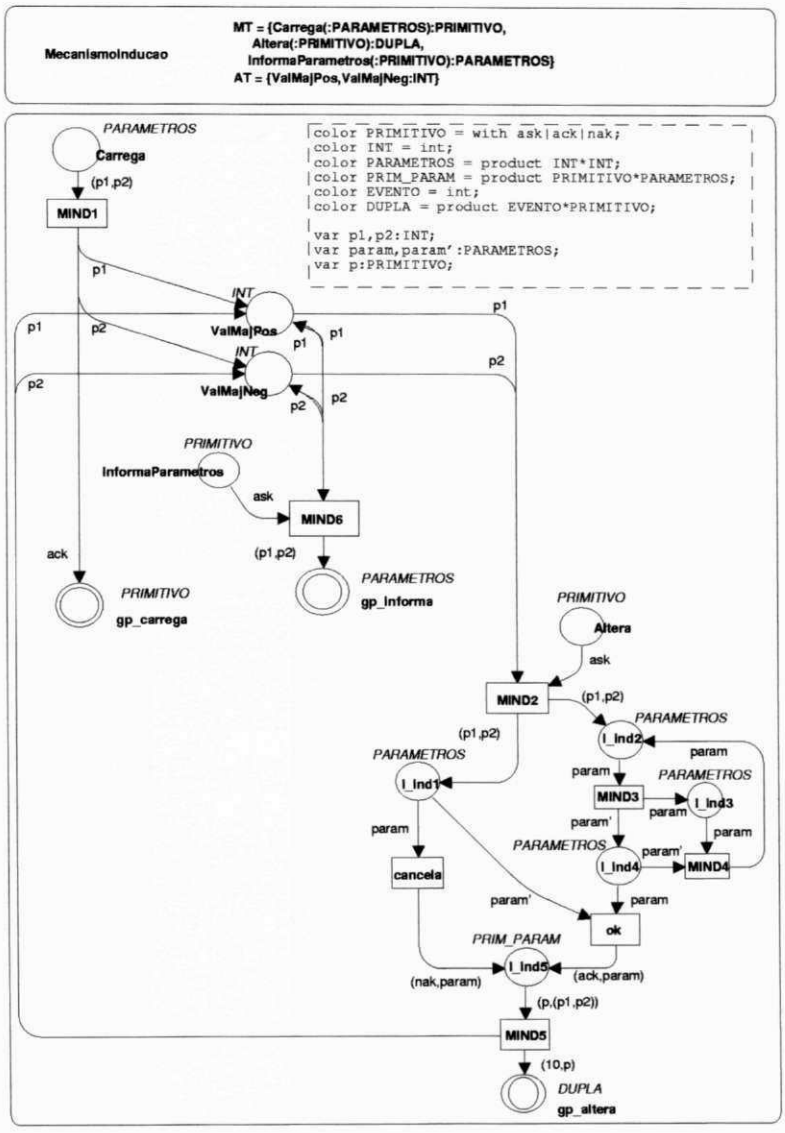


Figura 5.25: G-CPN *MecanismoInducao*.

5.4.3 Assunto Domínio

Neste assunto está o módulo *Dominio*. Tal módulo é mostrado na Figura 5.27. A interface do módulo declara os seguintes métodos: *Escolhe*, *Visualiza*, *ConstroiFiltro*, *InformaAtributos*, *CarregaAtributos* e *InformaFiltro*. Os atributos são *nome* (um *string* contendo o nome do domínio), *l_classes* e *filtro*, contendo respectivamente os conjuntos de classes que compõem o domínio e o filtro sobre este domínio.

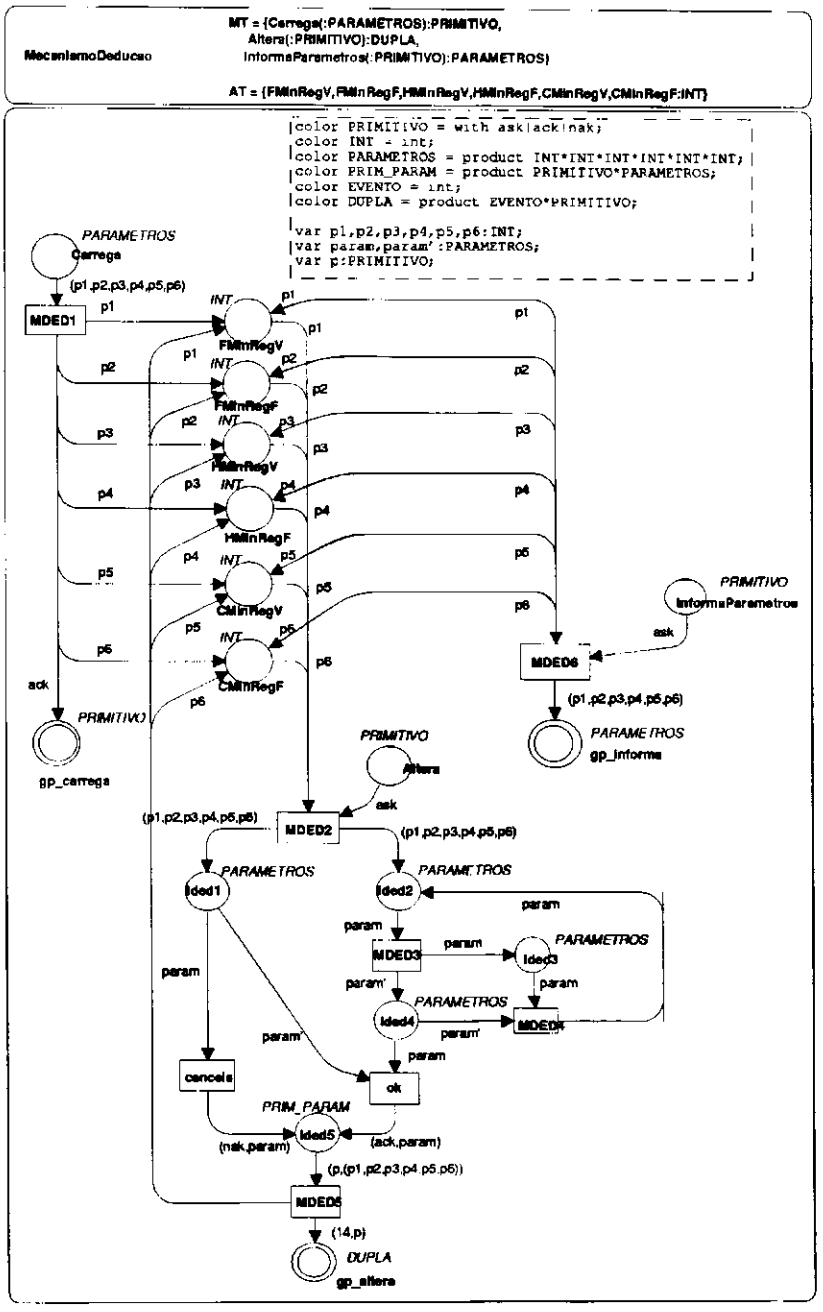


Figura 5.26: G-CPN *MecanismoDeducao*.

O método *Escolhe* modela uma caixa de diálogo através da qual o usuário pode selecionar um domínio previamente construído. O método deve realizar a leitura de um arquivo contendo um conjunto de classes que serão manipuladas pelo sistema durante a etapa de aprendizagem.

Através da execução do método *Visualiza*, que pode ser implementado como uma caixa de diálogo, o usuário pode visualizar o conjunto de classes do domínio previamente selecionado. Nesta caixa, a visualização de todos os atributos de cada classe deve ser provida.

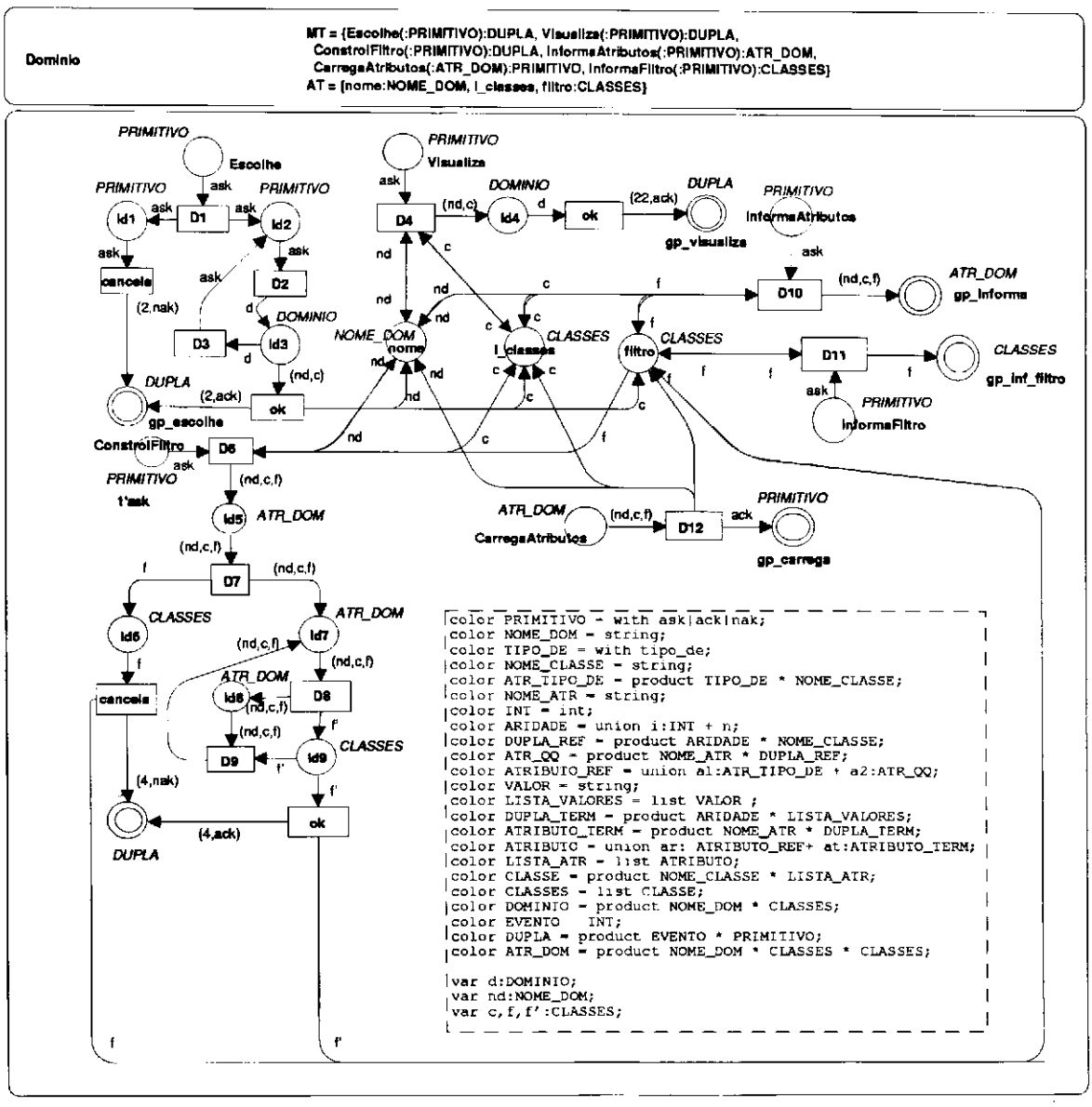


Figura 5.27: G-CPN *Dominio*.

A execução do método *ConstroiFiltro*, que também pode ser implementado como uma caixa de diálogo, possibilita ao usuário a seleção de classes, atributos e valores que este julgar particularmente interessantes para a aprendizagem de um determinado conceito do domínio.

O método *InformaFiltro* é invocado por *AgenteAprendiz.GeraFatos* para conhecer o conjunto de classes, atributos e valores pertinentes para a geração da lista de fatos.

Os métodos *InformaAtributos* e *CarregaAtributos* são invocados, respectivamente, pelo módulo *Arquivo*, durante a execução dos métodos *Salva* e *Abre*, como mostrado na seção 5.4.5.

5.4.4 Assunto Interface

Neste assunto, estão os módulos G-CPN *Interface*, *Titulo*, *Menu*, *MenuArquivo*, *MenuAmbiente*, *MenuAprendizagem*, *MenuRevisao*, *MenuExploracao* e *BarraStatus*.

Na Figura 5.28 é mostrado o módulo *Interface*. Este módulo modela apenas um método: *Atualiza*. Tal método é invocado pelo módulo *ProtocoloMosca* sempre que um evento externo ocorre no sistema. O método único invoca o módulo *ProtocoloMosca* para conhecer o estado corrente e atualizar a interface do sistema de acordo com tal estado. Optou-se por atualizar primeiramente o menu, modelado pelo módulo *Menu* e em seguida, os demais componentes da interface, modelados pelos módulos *Título* e *BarraStatus*.

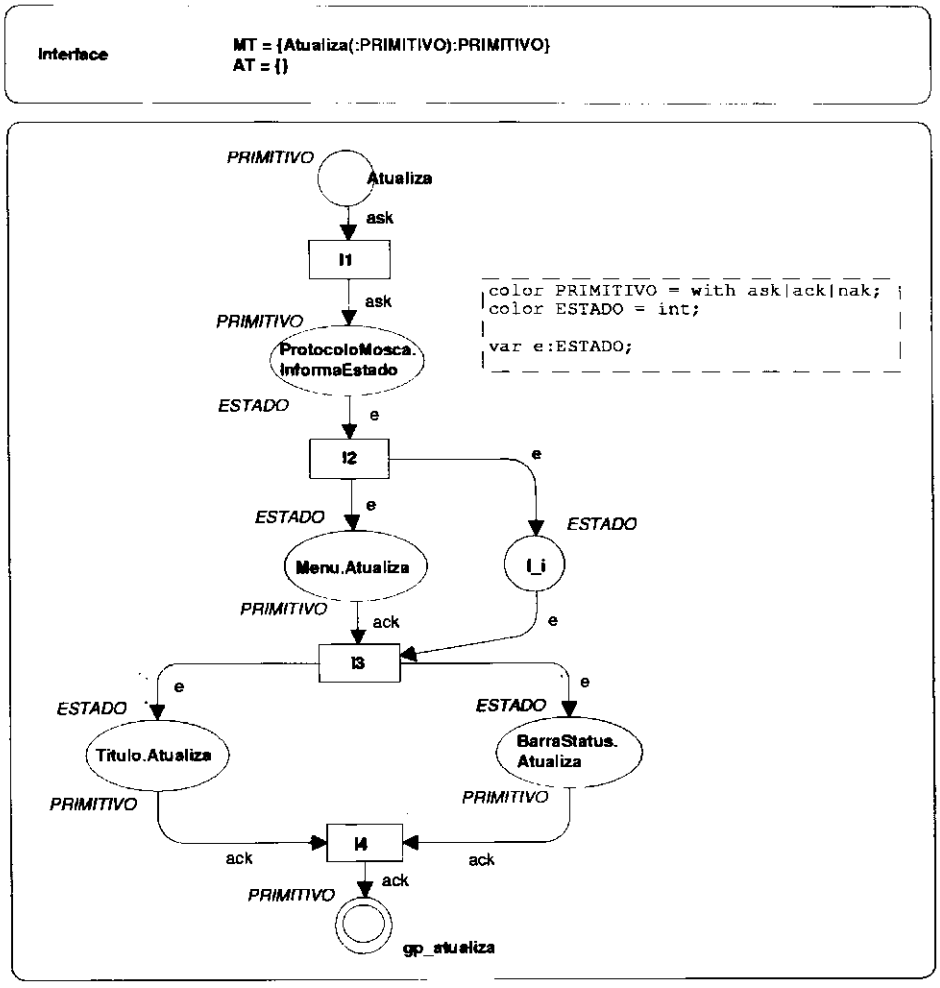


Figura 5.28: G-CPN *Interface*.

Na Figura 5.29 mostra-se o módulo *Título*. Tal módulo modela o método *Atualiza*. O atributo único é *texto*, que é um *string* e deve reter o nome do arquivo sendo manipulado na sessão. O método *Atualiza* atribui a *texto* a cor "SAID" quando o sistema está em estado inicial e atribui o nome do arquivo sendo manipulado, em caso contrário. Ou seja, uma vez ocorridos os eventos

correspondentes à abertura de arquivo ou escolha de novo ambiente de aprendizagem, o texto no título da aplicação é o nome do arquivo sendo manipulado.

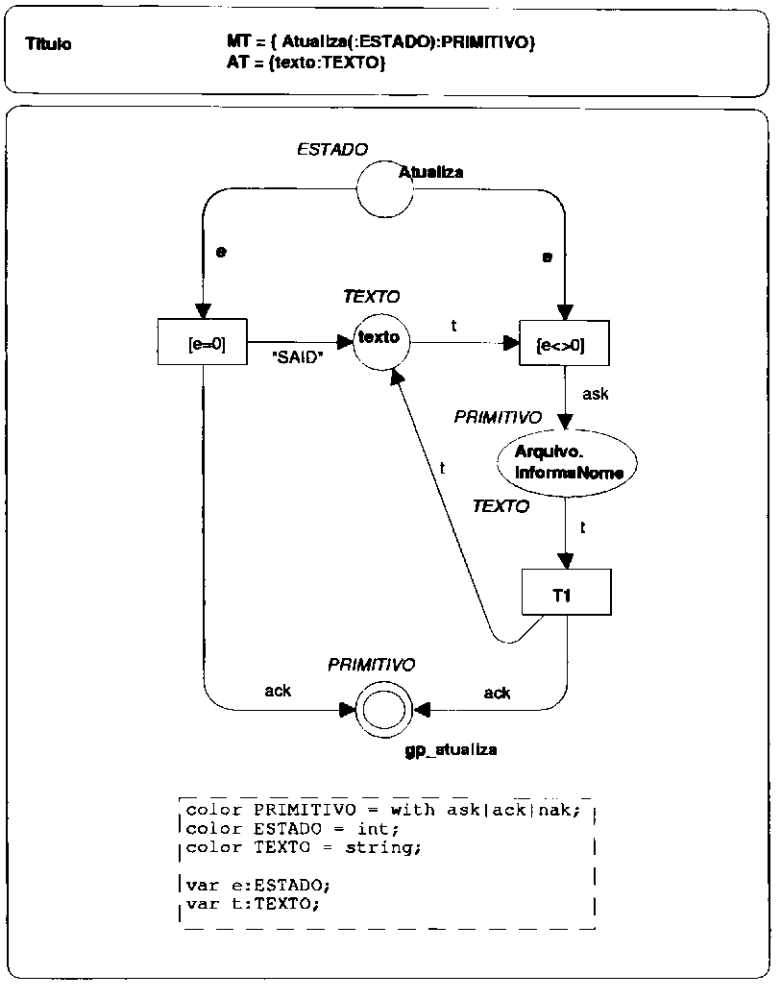


Figura 5.29: G-CPN *Titulo*.

O módulo *Menu* é mostrado na Figura 5.30. A interface do módulo declara dois métodos: *Atualiza* e *Seleciona*. A atualização modelada por *Atualiza* é realizada segundo o estado corrente do sistema (mantido por *ProtocoloMosca*): todos os módulos modelando os submenus (conhecidos como menus *pull-down*) são invocados para atualizarem seus atributos (que correspondem aos itens de cada submenu). A seleção modelada por *Seleciona* pode ser vista intuitivamente como: um item *i* (representado por um número inteiro) de menu é escolhido e o módulo modelando o sub-menu que o contém como atributo é invocado para realizar a tarefa associada à seleção deste item de menu.

O módulo *BarraStatus* é mostrado na Figura 5.31. Tal módulo modela o método *Atualiza*. Como para o módulo *Titulo*, o atributo único é *texto*, um *string* que deve reter uma mensagem refletindo o estado em que o sistema se encontra. A funcionalidade do método único deste módulo

é então exibir tal mensagem a fim de guiar o usuário ao longo de uma sessão em SAID. O método *Atualiza* atribui a cor "Aguardando escolha de ambiente" a *texto* quando o sistema está em estado inicial e atribui uma cor em função do estado em caso contrário, como mostrado no nó de declaração por F(e:ESTADO).

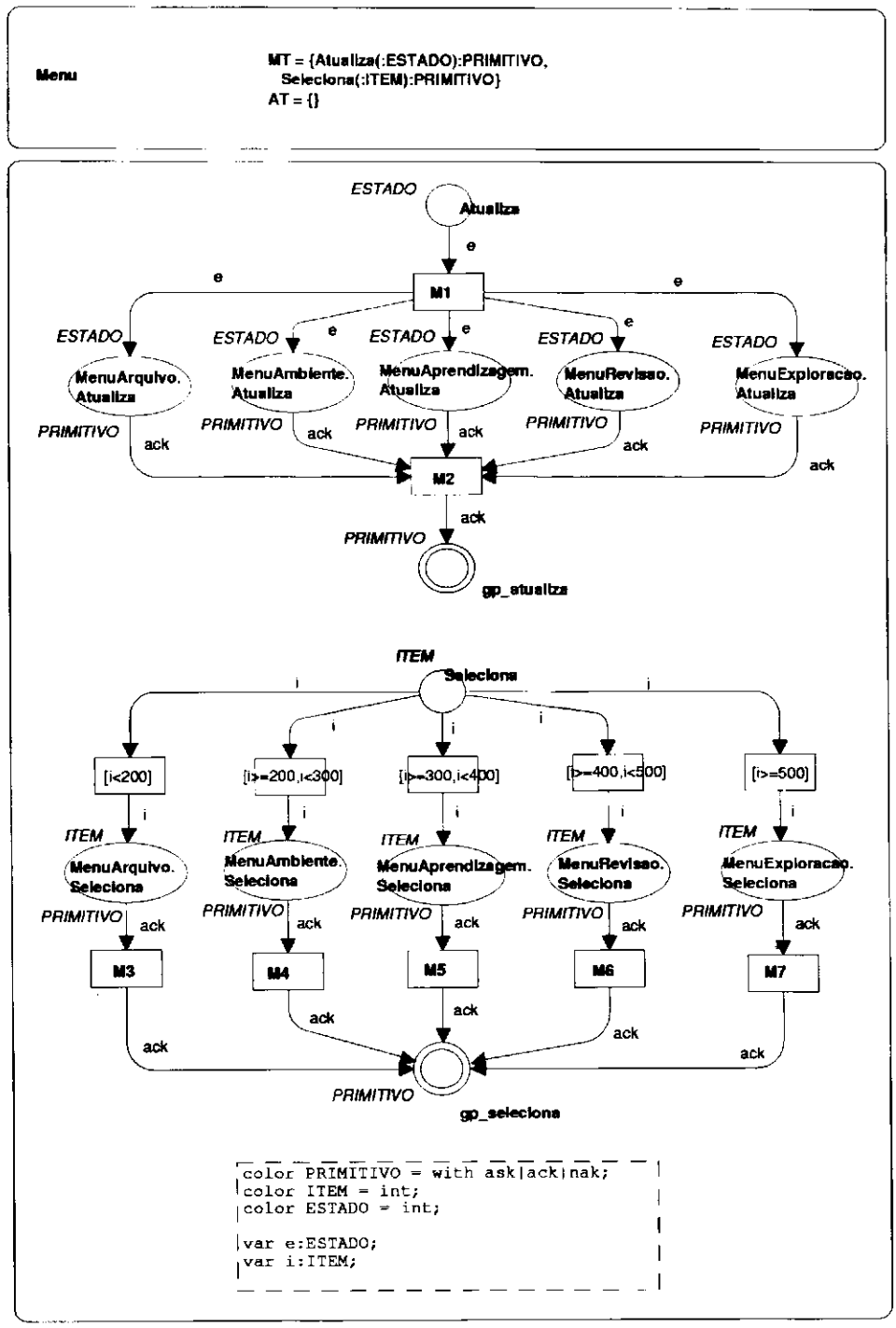


Figura 5.30: G-CPN *Menu*.

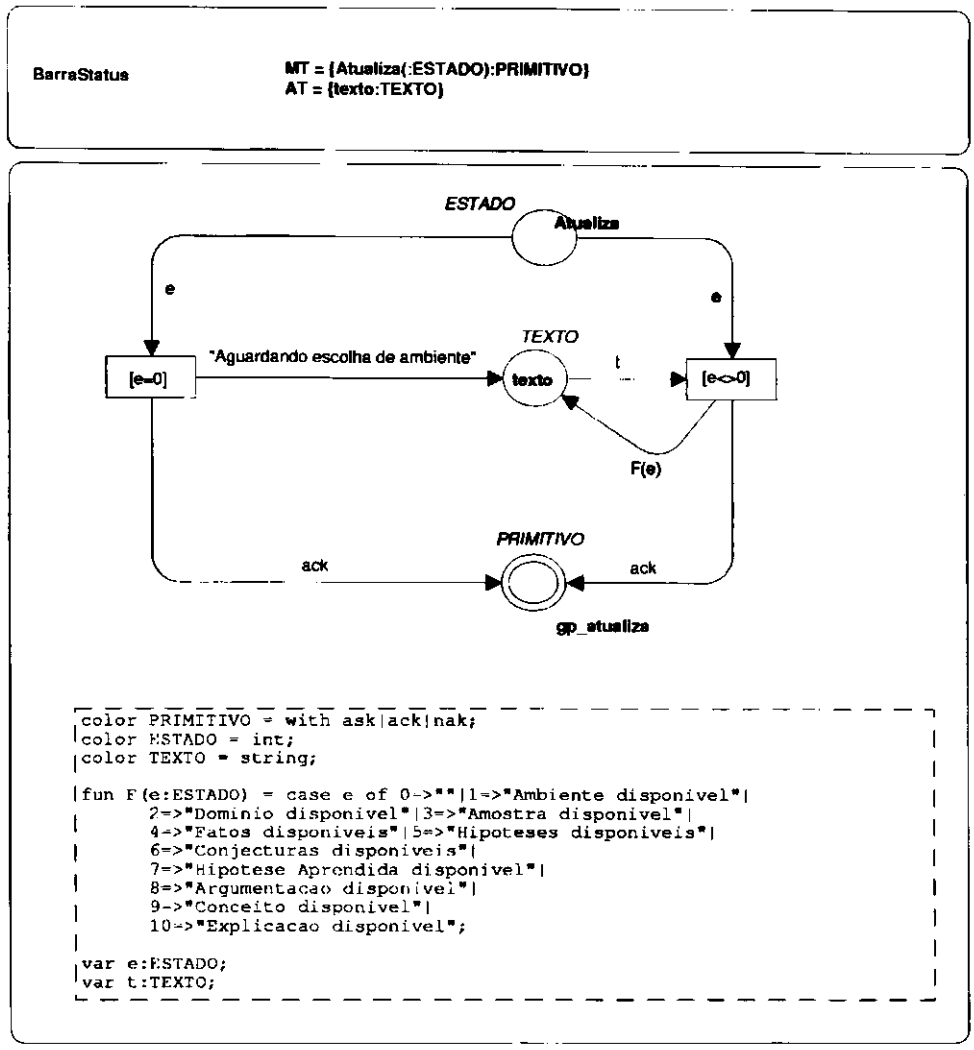


Figura 5.31: G-CPN *BarraStatus*.

Na Figura 5.32 mostra-se o módulo *MenuArquivo*. Os métodos que este módulo modelam são: *Atualiza* e *Seleciona*. Os atributos são *Novo, Abre, Salva, SalvaC* e *Sai*. A atualização modelada por *Atualiza* atribui a cada atributo uma cor *hab* ou *desab* (respectivamente, habilitar ou desabilitar), se o estado corrente do sistema determinar a habilitação ou não do item de menu que o atributo modela. A seleção modelada por *Seleciona* pode ser vista como: caso o item (modelado como um número inteiro) recebido na invocação do método esteja habilitado, a transição correspondente é habilitada e deve invocar o método associado ao evento desejado. Após o retorno do método invocado, *ProtocoloMosca* é invocado para atualizar seu estado, caso o método tenha sido executado com sucesso. Tal informação é trazida pela ficha contendo uma dupla com cores dos conjuntos *EVENTO* e *PRIMITIVO*. A primeira cor carrega o número inteiro associado ao evento e a segunda carrega *ack* ou *nak* (respectivamente, método encerrado com sucesso e método encerrado sem sucesso).

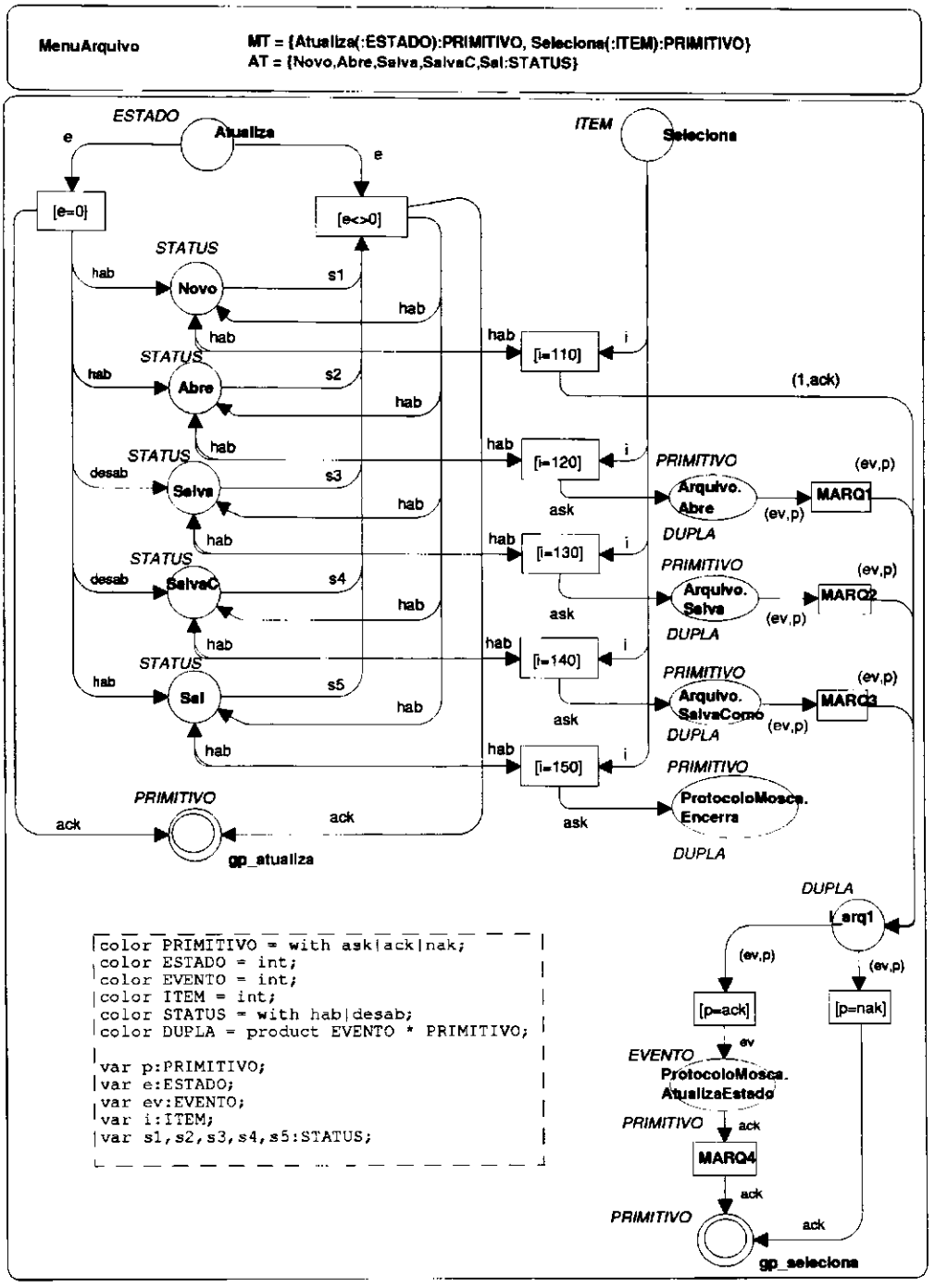


Figura 5.32: G-CPN *MenuArquivo*.

Os demais módulos modelando o menu têm métodos estrutura e funcionalmente idênticos ao módulo *MenuArquivo*, à exceção dos itens de menu e das funções de habilitação de cada item. Nas figuras de 5.33 a 5.36 são apresentados respectivamente os módulos *MenuAmbiente*, *MenuAprendizagem*, *MenuRevisao* e *MenuExploracao*.

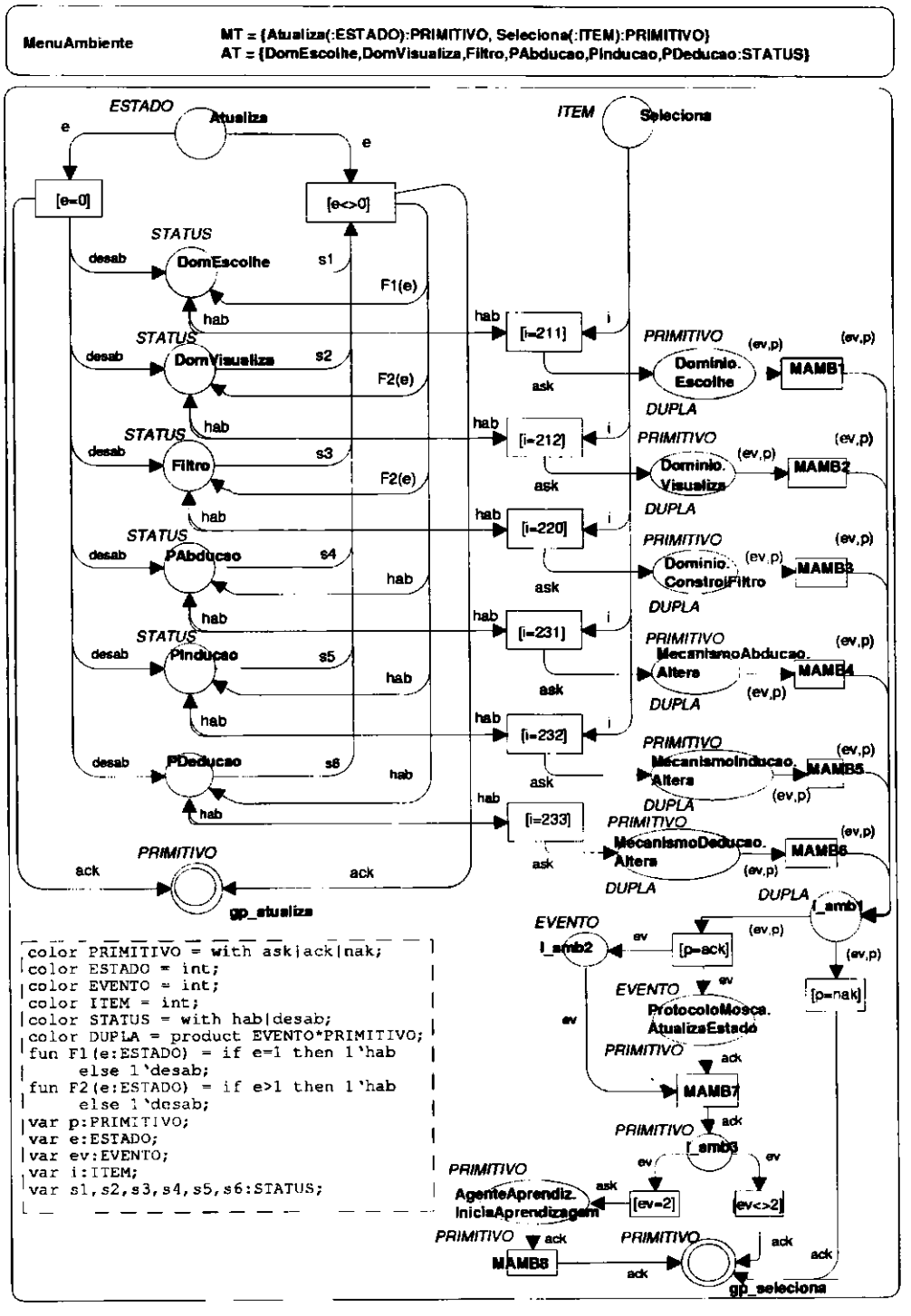


Figura 5.33: G-CPN *MenuAmbiente*.

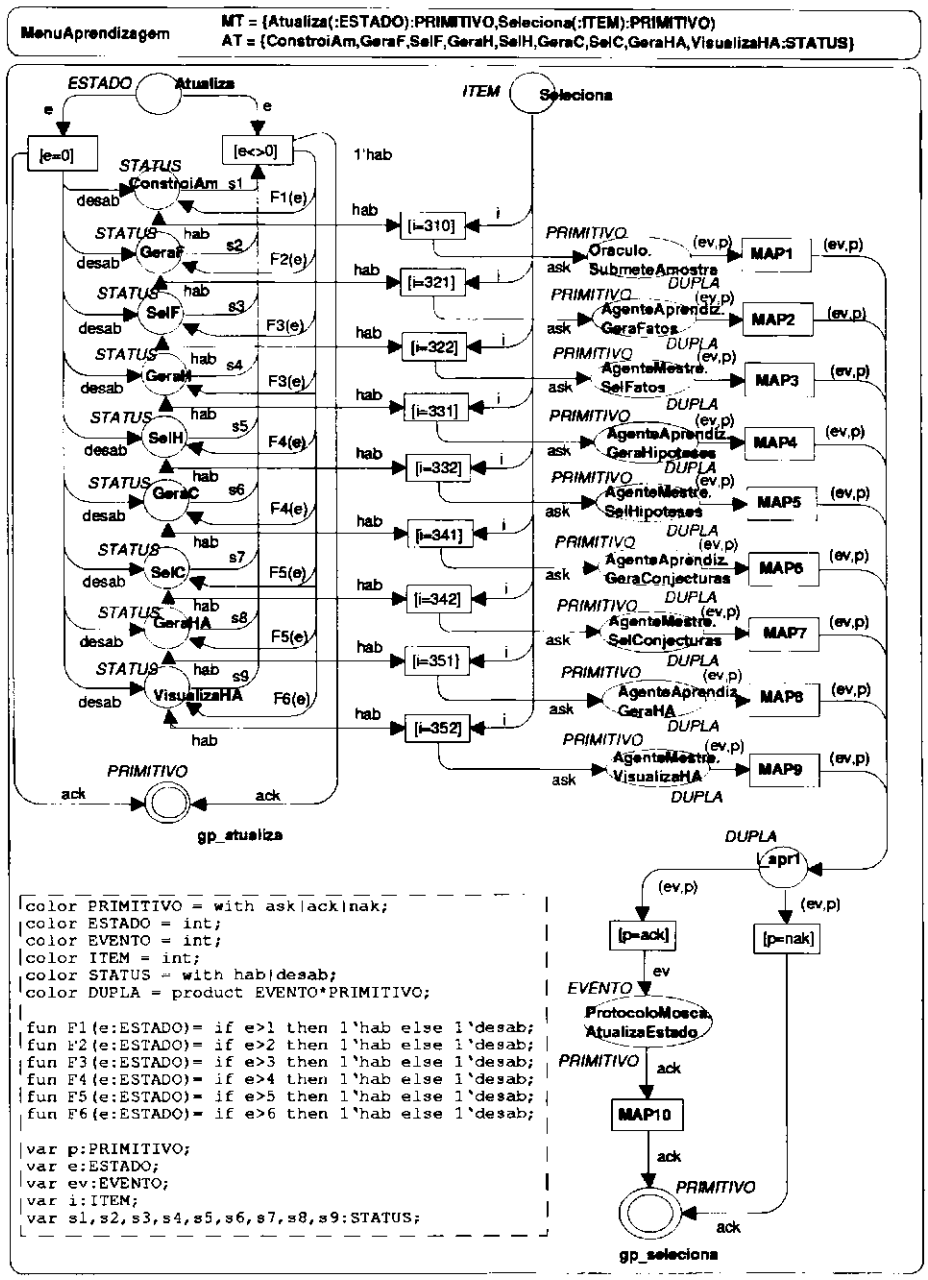


Figura 5.34: G-CPN *MenuAprendizagem*.

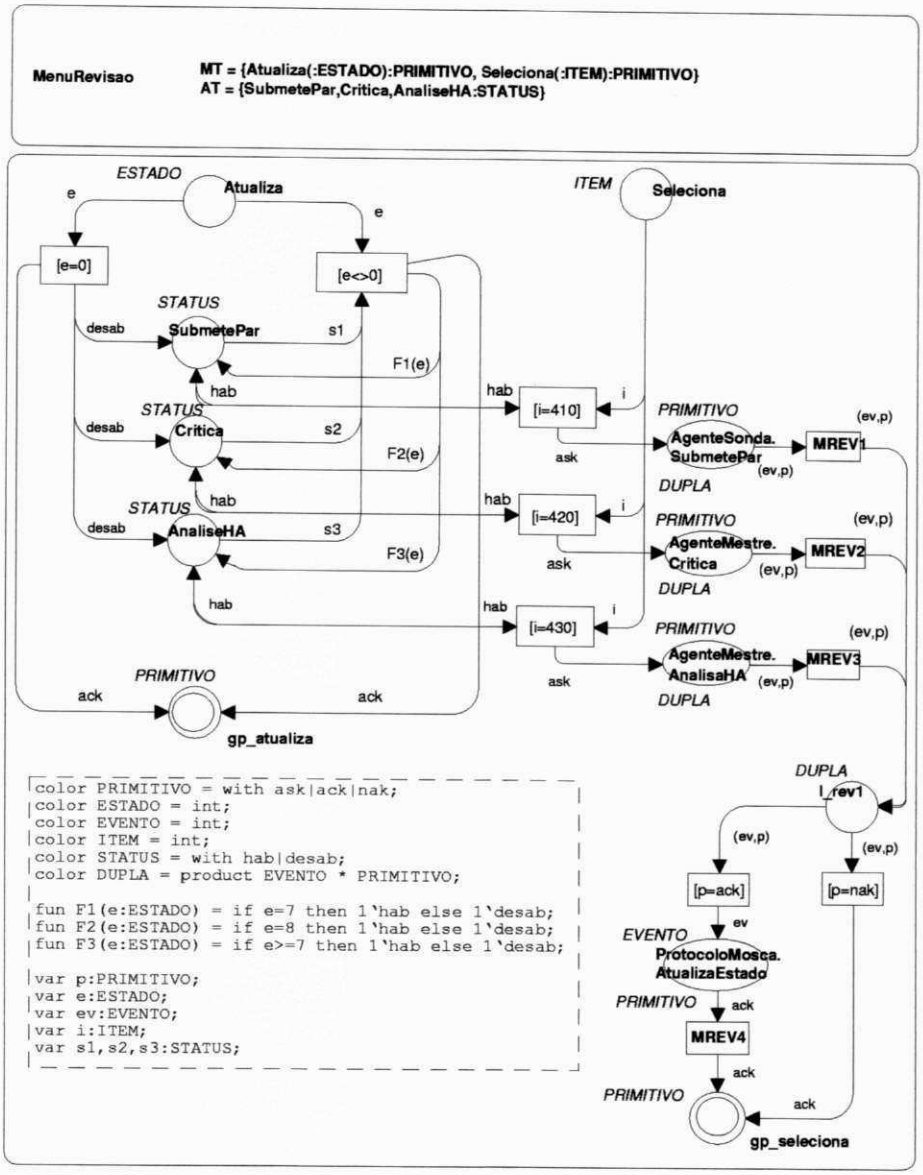


Figura 5.35: G-CPN *MenuRevisao*.

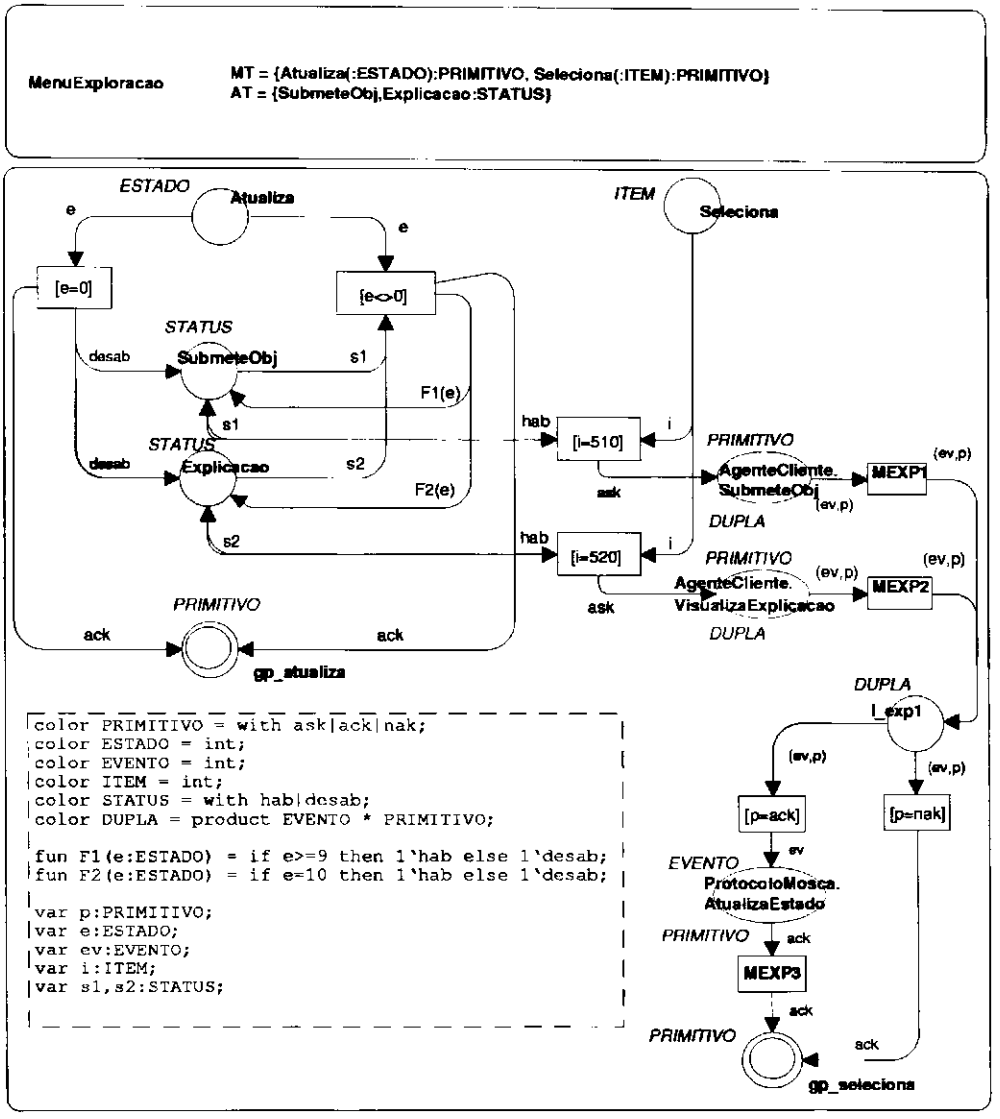


Figura 5.36: G-CPN *MenuExploracao*.

5.4.5 Assunto Arquivo

O módulo G-CPN *Arquivo* é o único neste assunto. Este módulo é composto por duas páginas (Figura 5.37 e Figura 5.38). A interface do módulo declara os métodos: *Inicializa*, *InformaNome*, *Abre*, *Salva*, *SalvaComo*, *GravaEstruturas* e *CarregaEstruturas*. O atributo único declarado na interface do módulo é *nome*.

O método *Inicializa* (Figura 5.37) atribui a cor inicial "semnome" ao lugar-atributo *nome*.

O método *InformaNome* (Figura 5.37) envia ao módulo chamador o conteúdo do atributo *nome* (a cor no lugar-atributo *nome*).

O método *Abre* (Figura 5.37) modela uma caixa de diálogo que possibilita ao usuário a escolha de um arquivo contendo estruturas de dados referentes a um ambiente de aprendizagem previamente manipulado. Uma vez feita a escolha do nome do arquivo, o usuário pode pressionar um botão *OK* (disparando um evento representado pela transição de mesmo nome), invocando o método *CarregaEstruturas* deste mesmo módulo. É possível encerrar a exibição da caixa sem que nenhum arquivo seja aberto, se um botão *cancela* é pressionado (a transição *cancela* é escolhida para disparar).

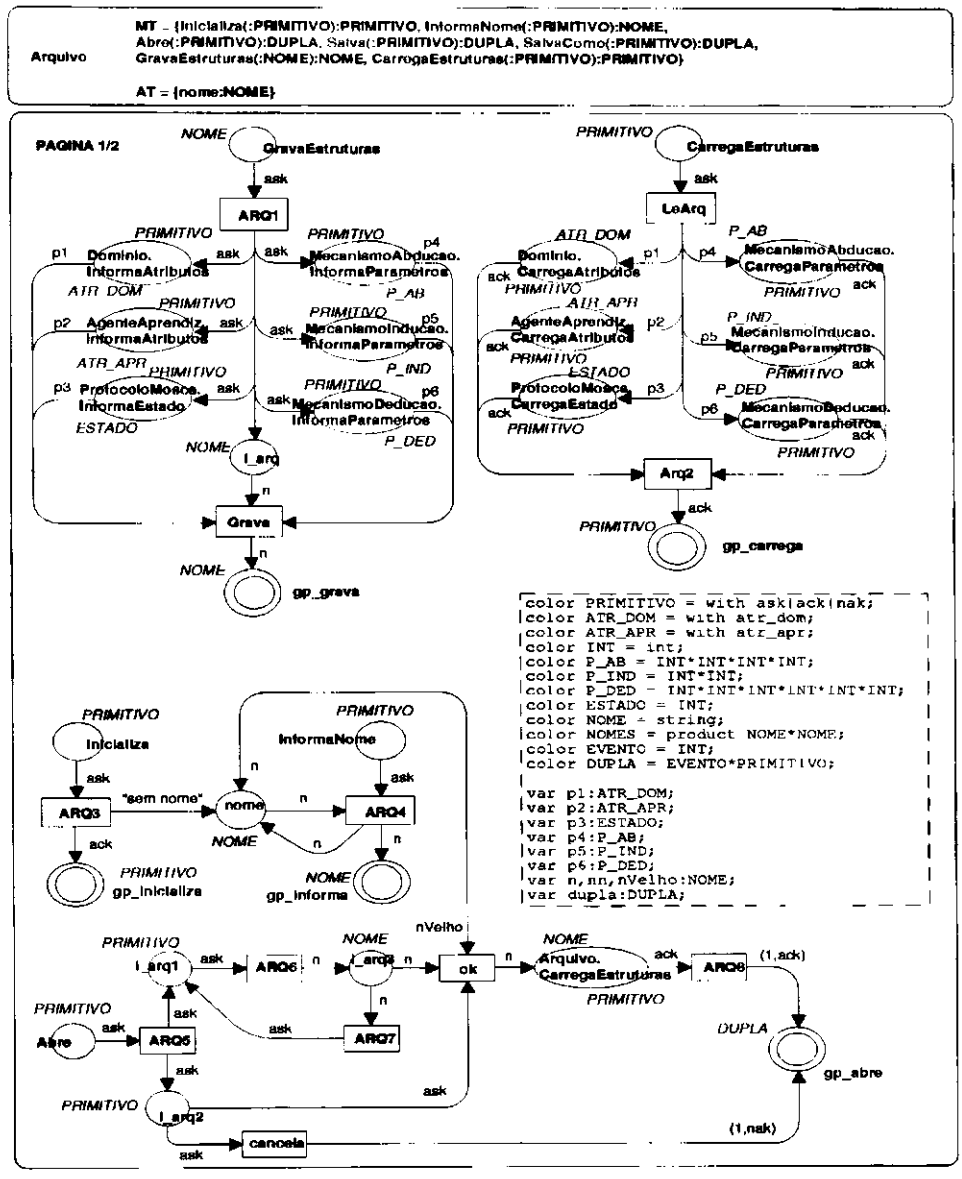


Figura 5.37: G-CPN *Arquivo* (Página 1).

CarregaEstruturas (Figura 5.37) é o método que realiza a leitura de um arquivo escolhido para ser manipulado pelo usuário e preenche as estruturas de dados no sistema que retêm o ambiente: (i) no módulo *Dominio*, o filtro e o conjunto de classes; (ii) no módulo *AgenteAprendiz*, a amostra, as regras (fatos, hipóteses e conjecturas), a argumentação, o último par <objeto, crença> e

a última crença recebida; (iii) no módulo *ProtocoloMosca*, o estado; e (iv) nos módulos *MecanismoAbducao*, *MecanismoInducao* e *MecanismoDeducao*, os parâmetros de aprendizagem.

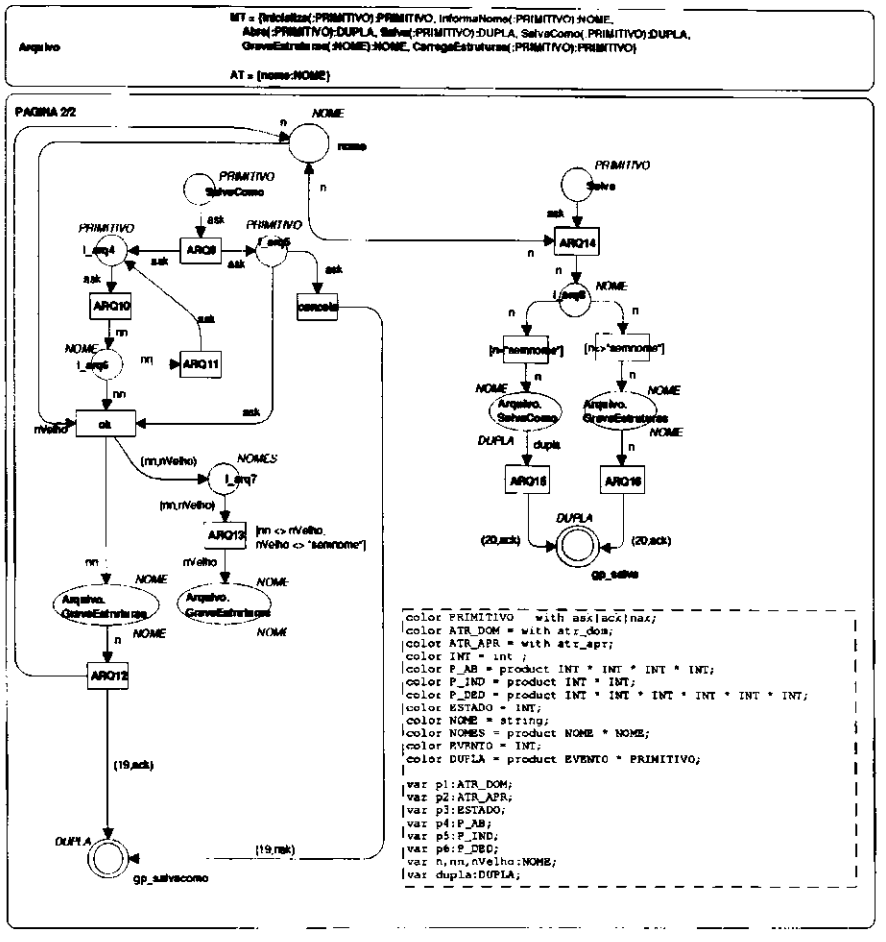


Figura 5.38: G-CPN *Arquivo* (Página 2).

GravaEstruturas (Figura 5.37) é o método que solicita de cada módulo as estruturas de dados mencionadas acima e as armazena em um arquivo, cujo nome é recebido como parâmetro na invocação do método. Para efeito de implementação, os conjuntos de cores modelando as estruturas em cada módulo devem ser observados nos respectivos nós de declaração, já que, neste módulo *Arquivo*, a especificação destas estruturas é simplificada (por exemplo, o conjunto de cores *ATR_DOM* é definido como a cor *atr_dom*).

O método *Salva* (Figura 5.38) invoca, de acordo com o nome de arquivo no lugar-atributo *nome*, o método *SalvaComo* (caso a cor no lugar-atributo *nome* seja "semnome") ou o método *GravaEstruturas* (caso a cor no lugar-atributo *nome* seja diferente de "semnome").

O método *SalvaComo* (Figura 5.38) modela uma caixa de diálogo através da qual um nome de arquivo é escolhido pelo usuário para armazenar o ambiente de aprendizagem corrente. Se o

ambiente corrente já está associado a um arquivo, as estruturas correntes são gravadas e ficam associadas ao novo nome de arquivo escolhido. Se o ambiente não está ainda associado a um arquivo, as estruturas são também gravadas e ficam associadas ao nome de arquivo escolhido.

5.5 Discussão

Neste capítulo, apresentou-se um modelo formal para um sistema de apoio à descoberta concebido a partir dos componentes do Agente Racional SAID. O ponto de partida que permitiu a construção do modelo como um conjunto de módulos cooperantes, refletindo o domínio do problema, foi o Protocolo de Aprendizagem MOSCA.

O modelo G-CPN do sistema SAID foi validado através de simulação, utilizando-se a ferramenta Design/CPN. Como foi mostrado, a IS de um módulo G-CPN é uma CP-net, salvo exceção dos ISPs. Tal exceção, porém, não representou obstáculo à simulação, dada a possibilidade de sua substituição pelo conjunto de componentes "lugar - transição - lugar".

Uma outra possibilidade de análise seria através dos grafos de ocorrência. Um grafo de ocorrência mostra (em seus nodos) todas as marcações possíveis a partir de uma determinada marcação e todos os possíveis disparos (em seus arcos) a partir de cada marcação. A possibilidade de análise por grafos de ocorrência tornou-se, porém, impraticável pela extensão do modelo, pois a explosão de estados dificultaria a verificação de propriedades importantes do sistema, como a *vivacidade*[12]. A extensão do modelo, por outro lado, foi inevitável, uma vez que se buscava alcançar um nível de detalhamento que possibilitasse uma implementação imediata do sistema. Alternativamente, utilizou-se a técnica de prototipagem para confrontar os resultados da modelagem com os requisitos pré-estabelecidos.

A seguir, nas conclusões, faz-se uma avaliação geral dos resultados alcançados com o trabalho, bem como o impacto da utilização de Redes de Petri, particularmente G-CPN, como ferramenta para construção de um modelo formal mediante as extensões que se propõem para este trabalho.

[12] Uma Rede de Petri é viva se, qualquer que seja a marcação atingida a partir de uma marcação inicial, é possível disparar qualquer transição da rede, mediante alguma sequência de disparos. A vivacidade em uma Rede de Petri garante a ausência de bloqueios.

Conclusão e Trabalhos Futuros

Como mostrado no Capítulo 1, a modelagem computacional do processo descoberta teve seus primeiros passos nos resultados alcançados por sistemas capazes de simular, com sucesso, descobertas científicas históricas. Alguns domínios da Ciência, no entanto, são bastante complexos, dado o crescente número de descobertas que vêm sendo realizadas. Dessa forma, a manipulação do montante de informação que se acumula nesses domínios (por exemplo, a medicina) poderia gerar teorias novas e melhores se apoiada por ferramentas computacionais que, em cooperação com especialistas, sistematizasse o processo de descoberta.

A concepção do Agente Racional SAID, baseada sobre a teoria Semi-Empírica e também sobre o Protocolo de Aprendizagem MOSCA possibilita uma abordagem interativa para a construção através de modelagem de teorias a partir do método lakatosiano de provas e refutações. Dessa forma, buscou-se neste trabalho, modelar e especificar formalmente um sistema dentro destes princípios.

No sistema proposto, no entanto, alguns dos princípios para geração e evolução do conhecimento foram especificados de maneira restrita: o mecanismo de argumentação, por exemplo, ignorou o critério de escolha de um conjunto mínimo de hipóteses para justificar a aplicação de uma Hipótese Aprendida. Tais limitações, consequentemente, foram refletidas nos mecanismos de crítica e de exploração de crítica.

Perspectivas Futuras

Do exposto, as perspectivas de trabalhos futuros que se propõem como extensão deste trabalho podem ser consideradas em duas vertentes, a saber: refinamento da especificação proposta e incorporação de características de sistemas multi-agentes.

Dentro da primeira vertente, os seguintes trabalhos poderiam ser desenvolvidos:

- Concepção e desenvolvimento de um Editor do Domíno. Como foi mostrado no Capítulo 4, além do usuário, a especificação proposta requer a interação do sistema com um

editor de objetos do domínio. Apesar da possibilidade de se simular as saídas deste editor que são utilizadas por SAID, a disponibilização de um tal editor propiciaria ao usuário desfrutar de uma interação bidirecional entre SAID e o editor. Ou seja, daquilo que sugere a noção de *equitraduzibilidade* (mostrada no Capítulo 3). A disponibilização dessa noção, proporcionada pelo editor, poderia evitar que problemas (como o da incompletude da linguagem de representação do domínio) fossem detectados mais cedo, através da visão mais "amigável" que o usuário especialista teria sobre os objetos de seu domínio. Uma dificuldade relevante na concepção de um editor genérico é a imensa variedade de domínios passíveis de representação estruturada que existem em diversos domínios da Ciência, o que é um obstáculo à disponibilização gráfica de objetos nesses domínios.

- Implementação de SAID e análise de resultados: A implementação a partir da especificação proposta é o mais imediato passo a ser dado no âmbito do Grupo de Inteligência Artificial da UFPB. Para a experimentação de vários aspectos do sistema, vislumbram-se dois domínios do conhecimento: a Geometria Euclidiana Plana [**FERNEDA, 1992A**] e a Música [**FERNEDA, 1994**]. No experimento do sistema mediante esses dois domínios, alguns aspectos de SAID poderão ser analisados e aperfeiçoados do ponto de vista computacional, sobretudo no que diz respeito à fase de revisão do conhecimento, como as modelagens dos mecanismos de crítica e do procedimento de construção de argumentação e de exploração de crítica.

Dentro da segunda vertente, o seguinte trabalho poderia ser desenvolvido:

- Especificação de SAID como um sistema multi-agente. A especificação proposta para SAID como um sistema de apoio à descoberta contempla algumas das características apresentadas no Capítulo 2, no contexto de agentes. Pode-se dizer que, pelo exposto naquele capítulo, especificou-se um agente racional que, segundo uma outra perspectiva, é um agente aprendiz. Um sistema multi-agente de apoio à descoberta poderia, no entanto, ser concebido, a partir desta concepção inicial. Uma vez abraçada a idéia de que a investigação científica é um processo social, que geralmente envolve muitos cientistas e pode se estender por um longo período de tempo, o processo de descoberta através de um sistema SAID multi-agente propiciaria a participação de diversos especialistas, de forma que tempo e espaço não representassem obstáculos ao processo de descoberta. Dessa forma, uma especificação poderia ser construída abstraindo-se o nível de detalhes inerente à implementação de determinados métodos e a ênfase seria

dada ao controle da interação dos componentes do Protocolo Mosca, desta feita, concebidos como entidades de software independentes. O poder da ferramenta G-CPN poderia ser melhor explorado por um tal trabalho.

Referências Bibliográficas

[ADDIS, 1988] T. R. Addis. "Knowledge organisation for abdution". Proceedings of the *Interdisciplinary Information Technology Conference*, Bradford University (Inglaterra), 1988.

[AUBERT, 1990] J. P. Aubert., C. Barboux, J. Quinqueton, J. Sallantin. "Langage d'un Topos et Machine à raisonner". *Colloque en l'honneur de Jean-Claude Simon - Informatique: nouveaux concepts scientifiques* (238-255), AFCET, Paris (França), 1990.

[BACON, 1984] F. Bacon. *Novum Organum*. Coleção Os pensadores, 3ª Edição, Editora Abril, São Paulo (SP), 1984.

[BARBOUX, 1990] C. Barboux. "Contrôle par objetion d'une théorie incomplète". *These de Doutorado*, LIRMM, *Université de Sciences et Techniques du Languedoc* - Montpellier II, Montpellier (França), 1990.

[BATES, 1994] J. Bates. "The role of emotion in believable agents". *Communications of the ACM*, 37 (122-125), 1994. (Citado [WOOLDRIDGE, 1995]).

[BAULAC, 1990] Y. Baulac. "Un micro-monde de géomètrie, Cabri-géomètre". *Tese de Doutorado*, LSD2, *Université Joseph Fourier*, Grenoble (França), 1990.

[BEZERRA, 1995] H. C. F. Bezerra. "Um Estudo Comparativo entre Algorítmos Indutivos Incrementais". *Dissertação de Mestrado*, COPIN, UFPB, Campina Grande (PB), 1995.

[BOUCHERON, 1988] S. Boucheron. "Apprentissage et Calculs". *Tese de Doutorado*, LIRMM, *Université de Sciences et Techniques du Languedoc* - Montpellier II, Montpellier (França), 1988.

[BUCHANAN, 1966] B. G. Buchanan. *Logics of Scientific Discovery*. AI Memo 47, Computer Science Department, Stanford University, 1966. (Citado em [LANGLEY, 1992])

[BUCHANAN, 1978] B. G. Buchanan and T. M. Mitchell. "Model Directed learning of production rules". In D. A. Waterman and F. Hayes-Roth (eds.), *Pattern-Directed Inference Systems*, Academic New York (EUA), 1978. (Citado em [LANGLEY, 1992])

[CARBONELL, 1983] J. G. Carbonell, R. S. Michalski, T. M. Mitchell. "An Overview of Machine Learning". In: J. G. Carbonell, R. S. Michalski and T. M. Mitchell (Eds.), *Machine Learning An Artificial Intelligence Approach*, Morgan Kaufmann, Palo Alto (USA), 1983.

[CARVALHO, 1997] J. N. Carvalho, G. M. Nóbrega, E. Ferneda "Um método de aprendizagem a partir de objetos complexos". *Anais do I Encontro Nacional de Inteligência Artificial - ENIA'97*, SBC, Brasília (DF), agosto 1997.

[CARVALHO, 1998] J. N. Carvalho (1998) "Um método de busca de similaridades em

objetos estruturados", *Dissertação de Mestrado*. COPIN/CCT/UFPB. (em preparação)

[CHALMERS, 1993] A. F. Chalmers. *O que é ciência afinal?*, Editora Brasiliense, São Paulo (SP), 1993.

[COAD, 1993] P. Coad, E. Yourdon. "Projeto Baseado em Objetos". Editora Campus Ltda. Rio de Janeiro (RJ), 1993.

[COSTA, 1997] E. B. Costa "Um Modelo de Ambiente Interativo de Ensino e Aprendizagem Baseado numa Arquitetura Multi-Agentes". *Tese de Doutorado em Engenharia Elétrica*, COPELE, CCT, UFPB, dezembro 1997.

[CPN, 1998] *Design/CPN - Computer Tool for Coloured Petri Nets*, CPN Group at the University of Aarhus, Dinamarca, 1998. (www.daimi.au.dk/designCPN)

[DARDEN, 1997] L. Darden. "Recent Work in Computational Scientific Discovery". Proceedings of the *Nineteeth Annual Conference of the Cognitive Science Society*, Michael Shafto and Pat Langley (Eds.). Mahwah, New Jersey: Lawrence Erlbaum, 1997.

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

[DENG, 1993] Y. Deng, S. K. Chang, J. C. A. Figueiredo, A. Perkusich. "Integrating software engeneering methods and petri nets for specification and prototyping of complex software systems". *Lecture Notes in Cumputer Science*, Vol. 691: Application and Theory of Petri Nets. Springer-Verlag (206-223), 1993.

[DUGERDIL, 1988] P. Dugerdil. "Contribuition a l'étude de la représentation des connaissances fondée sur les objets: le langage OBJLOG" Tese de Doutorado, GRTC, *Université de Doit, d'Economie et des Sciences d'Aix-Marseille III*, Marselha (França), 1988.

[FERBER, 1991] J. Ferber, L. Gasser. "Intelligence Artificielle Distribuée". In *International Workshop on Expert Systems & Their Applications*, 11. Avignon (France) 1991. (Citado em [SICHMAN, 1992]).

[FERNEDA 1992A] E. Ferneda, M. Py, P. Reitz, J. Sallantin. "L'agent rationnel SAID: une application en géométric", *Proceedings of the First European Colloquium on Cognitive Science*, pp. 175-192, Orsay (França), junho 1992.

[FERNEDA 1992B] E. Ferneda "Conception d'un agent rationnel et examen de son raisonnement en géométrie", *Tese de Doutorado*, LIRMM, *Université de Sciences et Techniques du Languedoc* - Montpellier II, Montpellier (França), 1992.

[FERNEDA 1994] E. Ferneda, C. A. P. Silva, L. M. Teixeira, H. M. Silva "A system for aiding discovery in musical analysis", *Proceedings of the I Brazilian Symposium on Computer Music, XIV Congress of Brazilian Computer Society*, Caxambu, 1994.

[FERNEDA 1995A] E. Ferneda, M. E. Souza e Silva, H. M. Silva. "A system for aiding discovery: elements for specification", *Lecture Notes in Artificial Intelligence*, Vol. 991: Advances in Artificial Intelligence. pp. 264-

273, Springer, Berlin (Alemanha), outubro 1995.

[FERNEDA 1995B] E. Ferneda, M. E. Souza e Silva, H. M. Silva, C. A. V. Gonçalves. "Elements for designing a system for aiding discovery", *Proceedings of the XV International Conference of the Chilean Computer Science Society*, pp. 210-221, Arica (Chile), novembro 1995.

[FERNEDA 1996] E. Ferneda, S. Hammoudi. "A integração entre programação em lógica e orientação a objeto como paradigma de representação de conhecimento. O exemplo de Objlog", *Anais da VI Semana de Informática da UFBA - SEMINFO'96*, pp. 49-65, UFBA, Salvador (BA), maio 1996.

[GENESERETH, 1994] M. R. Genesereth, S. P. Ketchpel. "Software Agents". *Communications of the ACM*, 37 (48-53), 1994.

[GINSBERG, 1990] M. L. Ginsberg. "Bilattices and modal operators". *Journal of Logic and Computation*, nº 1, 1990.

[GRACY, 1993] J. Gracy "Un environnement d'apprentissage appliqué à la modélisation des protéines". *Tese de Doutorado*, Univertsité des Sciences et Techniques du Languedoc (Montpellier II), Montpellier (França), 1993.

[GUERRERO, 1997] D. D. S. Guerrero, A. Perkusich, J. C. A. de Figueiredo. "Modeling a cooperative environment based on na object-based modular Petri-Net". *Proceedings of the IX International Conference on Software Engeneering and Knowledge Engennering*, Madri (Espanha), 1997.

[HAMMOUDI, 1996A] S. Hammoudi, E. Ferneda, E. Ferneda. "Uma linguagem de especificação para a definição de tipos virtuais", *Anais da VI Semana de Informática da UFBA - SEMINFO'96*, pp. 249-264, UFBA, Salvador (BA), maio 1996.

[HAMMOUDI, 1996B] S. Hammoudi, E. Ferneda, E. Ferneda. "Relações entre tipos de objetos e o conceito de ponto de vista: definição e especificação", *Anais da VI Semana de Informática da UFBA - SEMINFO'96*, pp. 217-235, UFBA, Salvador (BA), maio 1996.

[HANSON, 1958] N. R. Hanson. *Patterns of Discovery*, Cambridge University Press, 1958.

[HUHNS, 1987] M. N. Huhns *Distributed Artificial Intelligence* (Foreword). Morgan Kaufmann Publishers, Inc., Los Altos (EUA), 1987.

[HUTT, 1994] A.T.F. Hutt, *Object Analysis and Design; A Description of Methods*, John Wiley & Sons, 1994.

[JENSEN, 1987] K. Jensen. "Coloured Petri Nets". In: W. Brauer, W. Reisig, G. Rozenberg (Eds.), *Lecture Notes in Computer Science*, Vol. 254: Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986 Part I, , Springer-Verlag, 1987.

[JENSEN, 1992] K. Jensen. "Coloured Petri Nets: Basic Concepts, Analysis, Methods and Practical Use". *EACTS - Monographs on Theoretical Computer Science*. Springer-Verlag, 1992.

[JONG, 1997] H. Jong, A. Rip. "The Computer Revolution in Science: steps towards the realization of computer-supported discovery environments". *Artificial Inteligence*, vol. 91, n. 2, pp. 225-256, abril 1997.

[LAKATOS, 1976] I. Lakatos. *A lógica do Descobrimento Matemático: Provas e Refutações*, Zahar, Rio de Janeiro (RJ), 1976.

[LANGLEY, 1978] P. Langley. "BACON1: A general discovery system". *Proceedings of the Second National Conference of the Canadian Society for Computational Studies*, 1978. (Citado em [LANGLEY, 1992])

[LANGLEY, 1981] P. Langley, G. L. Bradshaw, H. A. Simon. "BACON5: The discovery of conservation laws". *Proceedings IJCAI-81*, 1981. (Citado em [LANGLEY, 1992])

[LANGLEY, 1992] P. Langley et al. *Scientific Discovery Computational Explorations of the Creative Process*. The MIT Press, 1992.

[LEDESMA, 1997] L. Ledesma, A. Pérez, D. Borrajo, L. M. Laita. "A computational Approach to George Boole's discovery of Mathematical Logic". *Artificial Inteligence* 91(281-307), 1997.

[LENAT, 1983A] D. B. Lenat. "The Role of Heuristics in Learning by Discovery: Three Case Studies". In: J. G. Carbonell, R. S. Michalski and T. M. Mitchell (Eds.), *Machine Learning An Artificial Intelligence Approach*, Morgan Kaufmann, Palo Alto (USA), 1983.

[LENAT, 1983B] D. B. Lenat. "Theory formation by heuristic search - The nature of heiristics II: Background and examples". *Artificial Inteligence* 21(1-2), 1983. (Citado em [RICH, 1994]).

[LIQUIERE, 1990] M. Liquière "Apprentissage à partir d'objets structurés. Conception et réalisation", *Tese de Doutorado*, LIRMM, Université de Sciences et Techniques du Languedoc - Montpellier II, Montpellier (França), 1990.

[MEPHU, 1993] E. Mephu-Nguifo "Concevoir une abstraction à partir de ressemblances", *Tese de Doutorado*, Univertsité des Sciences et Techniques du Languedoc (Montpellier II), Montpellier (França), 1993.

[MILL, 1963] J. S. Mill "A System of Logic". In *Collected Works*, vols. VII e VIII, University of Toronto Press and Routledge & Kegan Paul, 1963.

[MÜLLER, 1987] J. P. Müller. "*Contribution à l'étude d'un agent rationnel: specification en logique intensionnelle et implantation*", *Tese de Doutorado*, Institut National Polytechnique de Grenoble, Grenoble (França), 1987.

[NEWELL, 1962] A. Newell, J. C. Shaw and H.A Simon. "The process of creative thinking". In H. E. Gruber et al (eds.), *Contemporary Approaches To Creative Thinking*, Atherton, New York (EUA), 1962. (Citado em [LANGLEY, 1992]).

[NEWELL, 1982] A. Newell. "The knowledge level". *Artificial Intelligence*, vol. 4, nº 4 (283-307), 1982.

[OLIVA, 1990] A. Oliva. "A hegemonia da concepção empirista de ciência a partir do Novum Organon de F. Bacon", In: A. Oliva (org.) *Epistemologia: a cientificidade em questão*, Papirus Editora, Campinas, (SP), 1990.

[PEIRCE, 1958] *C.S. Peirce: Selected Writings*. Dove Publications. Inc., 1958.

[PINGAND, 1991] P. Pingand "Étude d'un environnement permettant l'acquisition de connaissance par apprentissage", *Tese de Doutorado*, Univertsité des Sciences et Techniques du Languedoc (Montpellier II), Montpellier (França), 1990.

[POPPER, 1975] K. Popper. *Conhecimento Objetivo*, EDUSP (São Paulo) / Editora Itatiaia

Ltda, Belo Horizonte, 1975.

[POPPER, 1993] K. Popper *A lógica da pesquisa científica*, Cultrix, São Paulo, 1993.

[PY, 1992] M. Py "Un agent rationnel pour raisonner par analogie", *Tese de Doutorado*, LIRMM, Université de Sciences et Techniques du Languedoc - Montpellier II, Montpellier (França), novembro 1992.

[REISIG, 1985] W. Reisig. *Petri Nets. An Introduction.* EATCS Monographs on Theoretical Computer Science, Vol. 4, Springer-Verlag, 1985.

[REITZ, 1992] P. Reitz. "Contribuition a l'étude des environnements d'apprentissage. Conceptualisation, Spécification et Prototypage" These de Doutorado, LIRMM, *Université de Sciences et Techniques du Languedoc - Montpellier II*, Montpellier (França), 1992.

[RICH, 1994] E. Rich and K. Nnight. *Inteligência artificial.* Makron Books do Brasil Editora Ltda, São Paulo, 1994.

[RUMBAUGH, 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. Object Modeling and Design, Prentice-Hall Intern., Inc., Englewood Cliffs (EUA), 1991.

[RUSSEL, 1995] S. Russell, P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall Intern., Inc., Englewood Cliffs (EUA), 1995.

[RUSSEL, 1997] S. Russell. *Rationality and intelligence. Artificial Intelligence*, 94 (57-77), 1997.

[SALLANTIN, 1991A] J. Sallantin, J. Quinqueton, C. Barboux, J.-P. Aubert. "Théories semi-empiriques: éléments de formalisation", *Revue d'intelligence artifielle*, 5(1):69-92, Paris (França), 1991.

[SALLANTIN, 1991B] J. Sallantin, J.-J. Szczeciniarz, C. Barboux, M.-S. Lagrange, M. Renaud "Théories semi-empiriques: conceptualisation et illustrations", *Revue d'Intelligence Artificielle*, 5(1):9-67, Paris (França), 1991.

[SALLANTIN, 1997] J. Sallantin, *Les Agents Intelligents: essai sur la rationalité des calculs*, Editions Hermes, Paris (França), 1997.

[SHOHAM, 1993] Y. Shoham. "Agent-oriented programming". *Artificial Intelligence*, 60 (51-92), 1993.

[SICHMAN, 1992] J. S. Sichman, Y. Demazeau, O. Boissier. "When can knowledge-based systems be called agents?". *Anais do Simpósio Brasileiro de Inteligência Artificial* (172-185). Sociedade Brasileira de Computação, Rio de Janeiro (RJ), 1992.

[SICHMAN, 1996] J. S. Sichman, Y. Demazeau. "A Model for Decision Phase of Autonomous Belief Revision in Open Multi-Agent Systems". *Journal of the Brazilian Computer Society.* Number 1 Vol. 3 (40-50), julho 1996.

[SILVA, 1998] M. E. S. Silva. "De Agentes Racionais a Agentes Semióticos: Um Estudo sobre a Aplicação da Semiótica na Concepção de Sistemas Inteligentes". *Dissertação de Mestrado.* COPIN/CCT/UFPB, maio 1998.

[SIMON, 1955] H. A. Simon. "A behavioral Model of rational choice", *Quart. J Economics* 69 (99-118), 1955.

[SIMON, 1966] H. A. Simon. "Scientific discovery and the psichology of problem solving". In R. Colodny (ed.), *Mind and Cosmos*, University of Pitsburgh

Press, 1966. (Citado em [LANGLEY, 1992]).

[SIMON, 1997] H. A. Simon, R. E. Valdés-Pérez and D. H. Sleeman. "Scientific discovery and simplicity of method" (Editorial). *Artificial Inteligence*, vol. 91, n. 2, pp. 177-181, abril 1997.

[VALDES-PEREZ, 1996] R. E. Valdés-Pérez. "A New Theorem in Particle Physics Enabled by Machine Discovery". Carnegie Mellon University, Pittsburgh (USA), 1996.

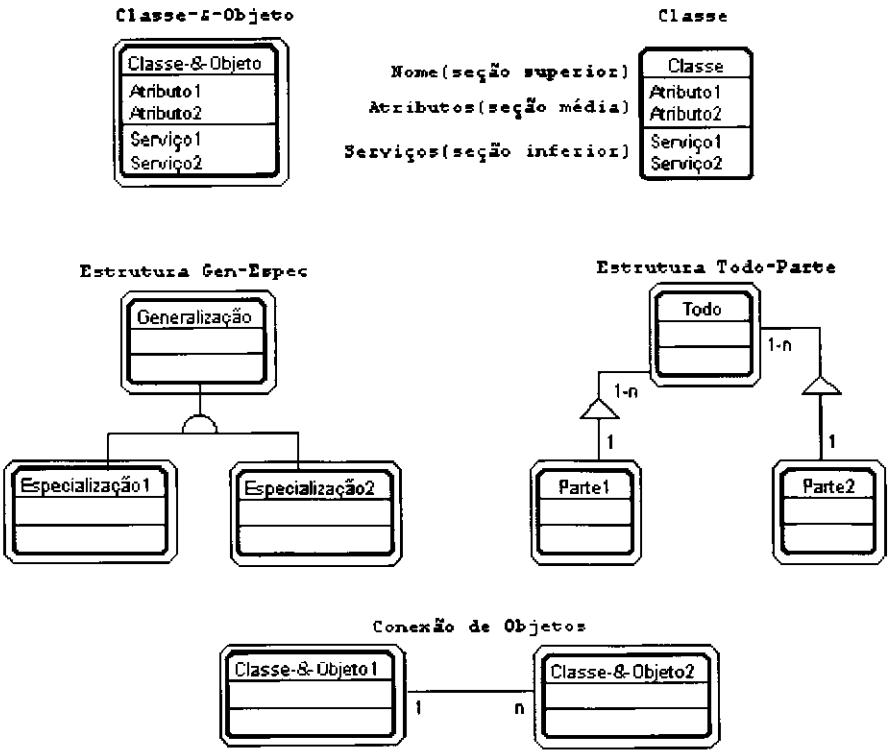
[VILLAREAL, 1989] M. J. Villareal-Fernandez "Construction par apprentissage de modèles d'objets complexes", *Tese de doutorado*, Université de Montpellier II, Montpellier (França), abril 1989.

[WIELINGA, 1990] B. J. Wielinga, J. Boose, B. Gaines, G. Schreiber, and M. W. van Someren (eds), "Current Trends in Knowledge Acquisition", pp. 313-338, Amsterdam (Holanda), 1990.

[WOOLDRIDGE, 1995] M. Woodridge, R. N. Jennings "Intelligent Agents: Theory and Practice", *Knowledge Engineering Review*, 10(2), junho 1995. (Arquivo Internet disponível em: http://www.doc.mmu.ac.uk/STAFF/M.Wooldridge/ker95/ker95-html.html)

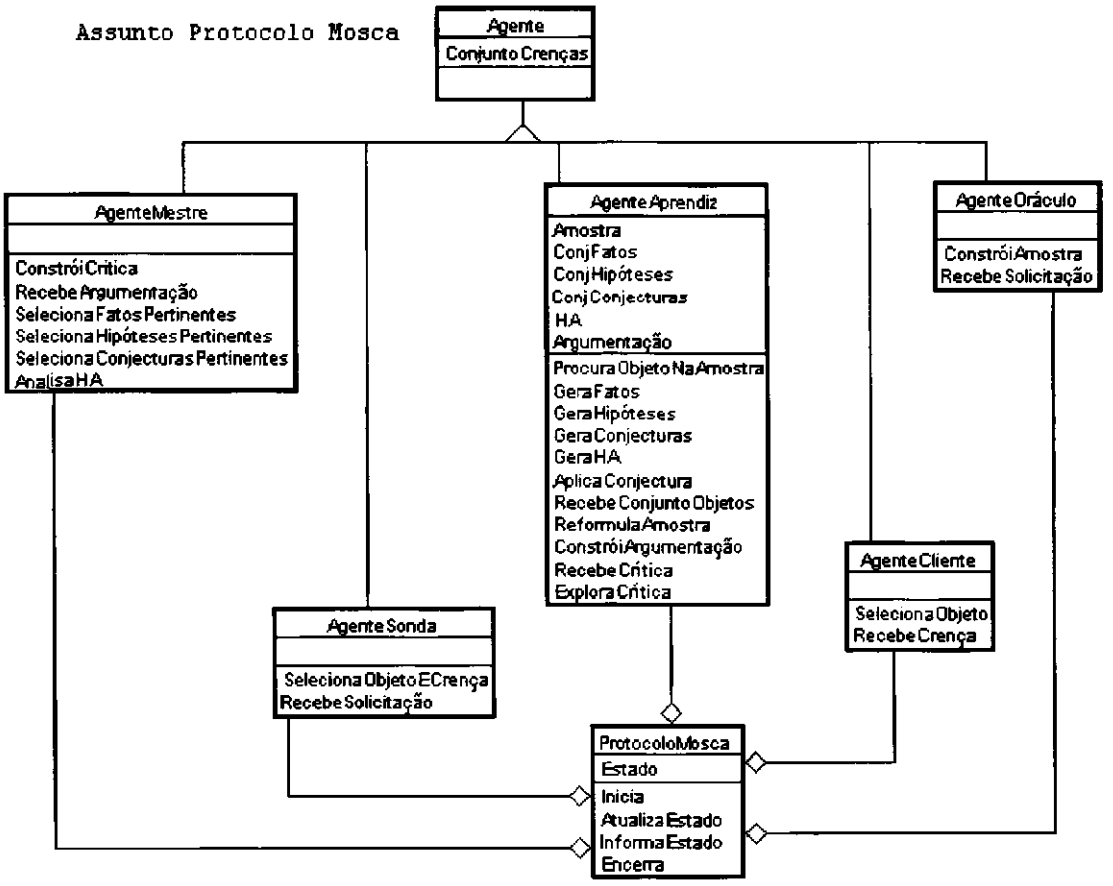
Anexo A

Resumo da Notação de P. Coad



Anexo B

Exemplo de Diagrama de Objetos e Cenário utilizando notação de J. Rumbaugh

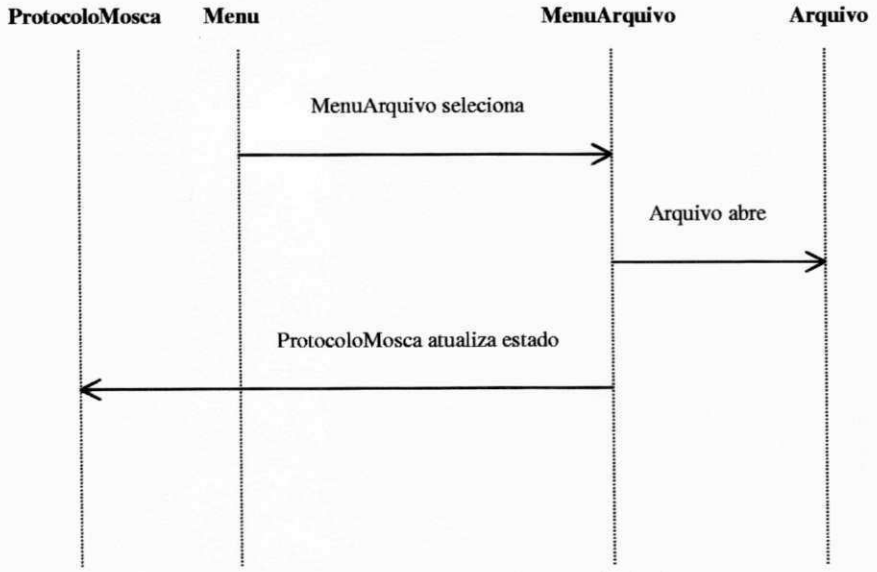


Assunto *Protocolo Mosca* do Modelo de Objetos segundo notação de J. Rumbaugh

Menu seleciona
MenuArquivo seleciona
ProtocoloMosca atualiza estado
Menu seleciona
MenuAmbiente seleciona
Domínio escolhe
ProtocoloMosca atualiza estado
AgenteAprendiz inicia aprendizagem
AgenteOráculo recebe solicitação

Menu seleciona
MenuArquivo seleciona
Arquivo abre
ProtocoloMosca atualiza estado

Cenário *Definição de Ambiente* segundo notação de J. Rumbaugh

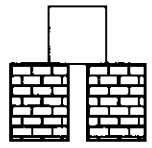


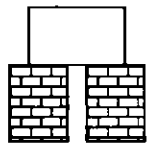
Fluxo de eventos no cenário *Definição de Ambiente*

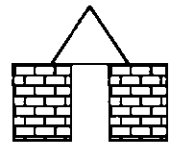
Anexo C

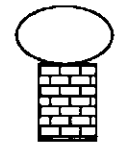
Exemplo de construção de uma conjectura a partir de uma amostra

Considere-se o seguinte conjunto de objetos:


Exemplo1


Exemplo2


Exemplo3


C_Exemplo1

Estes objetos dizem respeito ao *conceito* "arco" do *domínio* do mundo dos blocos. Uma *representação* para tais objetos é composta pela definição de suas classes:

classe: *arco*
<tipo_de [objeto]>
<composto_de [n.forma]>

classe: *forma*
<tipo_de [objeto]>
<cor [1.{preto,branco}]>
<status [1.{de_pe,deitado}]>
<sobre [n.*forma*]>

classe: *estavel*
<tipo_de [*forma*]>

classe: *instavel*
<tipo_de [*forma*]>

classe: *quadrado*
<tipo_de [estavel]>

classe: *retangulo*
<tipo_de [estavel]>

classe: *oval*
<tipo_de [instavel]>

classe: *bloco*
<tipo_de [estavel]>

classe: *triangulo*
<tipo_de [estavel]>

e pelas instâncias representando tais objetos:

instancia: exemplo1
<eh_um [arco]>
<composto_de [quadrado1,bloco1,bloco2]>

instancia: quadrado1
<eh_um [quadrado]>
<cor [branco]>
<status [deitado]>
<sobre [bloco1,bloco2]>

instancia: exemplo3
<eh_um [arco]>
<composto_de [triangulo1,bloco5,bloco6]>

instancia: triangulo1
<eh_um [triangulo]>
<cor [branco]>
<status [deitado]>
<sobre [bloco5,bloco6]>

instancia: bloco1
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>
<esquerda_de [bloco2]>

instancia: bloco2
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>

instancia: exemplo2
<eh_um [arco]>
<composto_de [retangulo1,bloco3,bloco4]>

instancia: retangulo1
<eh_um [retangulo]>
<cor [branco]>
<status [deitado]>
<sobre [bloco3,bloco4]>

instancia: bloco3
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>
<esquerda_de [bloco4]>

instancia: bloco4
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>

instancia: bloco5
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>
<esquerda_de [bloco6]>

instancia: bloco6
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>

instancia: c_exemplo1
<eh_um [arco]>
<composto_de [oval1,bloco7]>

instancia: oval1
<eh_um [oval]>
<cor [branco]>
<status [deitado]>
<sobre [bloco7]>

instancia: bloco7
<eh_um [bloco]>
<cor [preto]>
<status [de_pe]>

Inicialmente, define-se um *filtro* para a não consideração de algumas classes, alguns atributos dentro de uma classe ou mesmo alguns valores para certos atributos. No exemplo, elimina-se através de um filtro apenas o atributo *cor* da classe *forma*, dispensável para a compreensão do conceito em questão.

A cada um desses objetos é associada uma crença em relação ao conceito em questão. Assim, tem-se uma *amostra* formada por três exemplos (Exemplo1, Exemplo2 e Exemplo3) e um contra-exemplo (C_Exemplo1) do conceito "arco".

Entre os diversos *fatos elementares* presentes nos objetos da amostra, a saber:

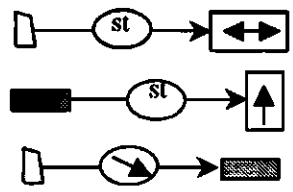
<quadrado, status, deitado>
<triangulo, status, deitado>
<bloco, status, de_pe>
<retangulo, sobre, objeto>
<bloco, esquerda_de, objeto>
<retangulo, status, deitado>
<oval, status, deitado>
<quadrado, sobre, objeto>
<triangulo, sobre, objeto>

são retidos os seguintes *fatos elementares de assimilação* (com suas respectivas representações gráficas), considerando apenas aqueles que aparecem em pelo menos 75% dos objetos da amostra (parâmetro de aprendizagem para a assimilação):

FE^1_a = <forma, status, deitado>

FE^2_a = <bloco, status, de_pe>

FE^3_a = <forma, sobre, bloco>



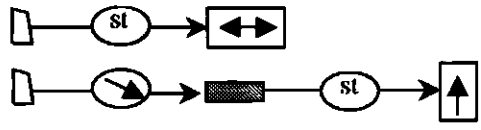
e o seguinte *fato elementar de discriminação*, considerando apenas aqueles que aparecem em pelo menos 75% dos exemplos do conceito e em no máximo em 20% dos contra-exemplos (parâmetro de aprendizagem para a discriminação):

FE^1_d = <bloco, esquerda_de, bloco>

Considerando ainda os mesmos valores dos parâmetros de aprendizagem e considerando o valor 2 como número máximo de fatos elementares a compor os *fatos*, são encontrados os seguintes *fatos de assimilação*:

$F^1_a = FE^1_a$

$F^2_a = FE^3_a \blacklozenge FE^2_a$

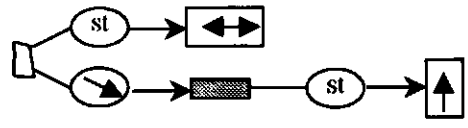


e de discriminação:

$F^1_d = FE^1_d$

A partir desses fatos, o sistema poderia propor as seguintes *hipóteses* válidas de *assimilação*:

$H^1_a = F^1_a \blacklozenge F^2_a$

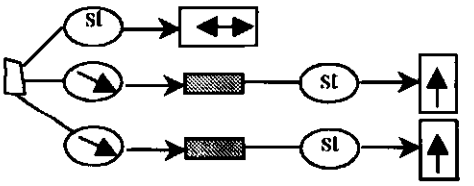


e de discriminação:

$H^1_d = F^1_d$

Entre as diversas conjecturas passíveis de serem propostas pelo sistema, uma seria:

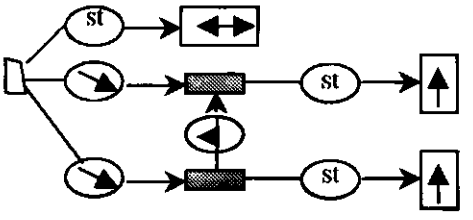
$C^1_a = H^1_a \blacklozenge H^1_a$



$C^1_d = H^1_d$

A *hipótese aprendida* seria, portanto:

$C_{HA}{}^1 = C^1_a \blacklozenge C^1_d$

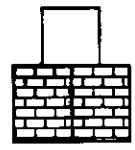


Em Prolog, esta conjectura corresponderia ao seguinte conjunto de regras **[FERNEDA, 1992]**:

```
regra(<<conjectura, hipotese_aprendida>, 0>, [X0], _crenca) :-
    componente(X0, [arco(X0), estavel(X1), bloco(X2, X3)]),
    regra(<<conjectura,assim>,1>, [X0,X1,X2], _crenca1),
    regra(<<conjectura,discr>,1>, [X0,X1,X3,X2], _crenca2),
    decidir_regra(<<conjectura,hipotese_aprendida>,0>,[_crenca1,_crenca2],_crenca).

regra(<<conjectura, assim>, 1>, [X0,X1,X2,X3], _crenca) :-
    regra(<<hipotese,assim>,1>, [X0,X1,X2], _crenca1),
    regra(<<hipotese,assim>,1>, [X0,X1,X3], _crenca2),
    decidir_regra(<<conjectura,assim>,1>,[_crenca1,_crenca2],_crenca).

regra(<<conjectura, discr>, 1>, [X0,X1,X2,X3], _crenca) :-
    regra(<<hipotese,discr>,1>, [X0,X1,X3,X2], _crenca1),
    decidir_regra(<<conjectura,discr>,1>,[_crenca1],_crenca).

regra(<<hipotese,assim>, 1>, [X0,X1,X2], _crenca) :-
    regra(<<fato,assim>,1>, [X0,X1], _crenca1),
    regra(<<fato,assim>,2>, [X0,X1,X2], _crenca2),
    decidir_regra(<<hipotese,assim>,1>,[_crenca1,_crenca2],_crenca).

regra(<<hipotese,discr>, 1>, [X0,X1,X2], _crenca) :-
    regra(<<fato,discr>,1>, [X0,X1,X2], _crenca1),
    decidir_regra(<<hipotese,discr>,1>,[_crenca1],_crenca).

regra(<<fato,assim>, 1>, [X0,X1], _crenca) :-
    regra(<<fato_elementar,assim>,0>, [X0,X1], _crenca1),
    regra(<<fato_elementar,assim>,1>, [X1], _crenca2),
    decidir_regra(<<fato,assim>,1>,[_crenca1,_crenca2],_crenca).

regra(<<fato,assim>, 2>, [X0,X1,X2], _crenca) :-
    regra(<<fato_elementar,assim>,0>, [X0,X1], _crenca1),
    regra(<<fato_elementar,assim>,0>, [X0,X2], _crenca2),
    regra(<<fato_elementar,assim>,3>, [X1,X2], _crenca3),
    regra(<<fato_elementar,assim>,2>, [X2], _crenca4),
    decidir_regra(<<fato,assim>,2>,[_crenca1,_crenca2,crenca3,crenca4],_crenca).
```

```
regra(<<fato,discr>, 1>, [X0,X1,X2], _crenca) :-
    regra(<<fato_elementar,discr>,0>, [X0,X1], _crenca1),
    regra(<<fato_elementar,discr>,0>, [X0,X2], _crenca2),
    regra(<<fato_elementar,discr>,1>, [X1,X2], _crenca3),
    decidir_regra(<<fato,discr>,1>,[_crenca1,_crenca2,crenca3],_crenca).

regra(<<fato_elementar,_>, 0>, [_obj1,_obj2], _crenca) :-
    verificar_fato_elementar(<_obj1,composto_de,_obj2>,_crenca1),
    decidir_regra(<<fato_elementar,_>,0>,[_crenca1],_crenca).

regra(<<fato_elementar,assim>, 1>, [_obj], _crenca) :-
    verificar_fato_elementar(<_obj,status,"deitado">,_crenca1),
    decidir_regra(<<fato_elementar,assim>,1>,[_crenca1],_crenca).

regra(<<fato_elementar,assim>, 2>, [_obj], _crenca) :-
    verificar_fato_elementar(<_obj,status,"de_pe">,_crenca1),
    decidir_regra(<<fato_elementar,assim>,2>,[_crenca1],_crenca).

regra(<<fato_elementar,assim>, 3>, [_obj1,_obj2], _crenca) :-
    verificar_fato_elementar(<_obj1,sobre,_obj2>,_crenca1),
    decidir_regra(<<fato_elementar,assim>,3>,[_crenca1],_crenca).

regra(<<fato_elementar,discr>, 1>, [_obj1,_obj2], _crenca) :-
    verificar_fato_elementar(<_obj1,a_esquerda_de,_obj2>,_crenca1),
    decidir_regra(<<fato_elementar,discr>,1>,[_crenca1],_crenca).
```

A avaliação, porém, de um objeto como o seguinte:



o classificaria como um exemplo do conceito "arco". Mais que isso, pela linguagem apresentada não haveria como distinguí-lo do objeto *Exemplo1* apresentado anteriormente. Essa constatação deveria causar uma reavaliação da linguagem de representação, incluindo-se, por exemplo, um atributo para representar a distância (*junto* ou *separado*) existente entre dois blocos, além de uma nova execução do processo de aprendizagem.