Josenildo Costa da Silva

Aquisição de Conhecimentos e Manutenção para uma Sociedade de Agentes Tutores Artificiais

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba - Campus II como parte dos requisitos necessários para obtenção do grau de Mestre.

Evandro de Barros Costa, D.Sc.
(Orientador)
Edilson Ferneda, Dr.
(Orientador)

Área de Concentração: Inteligência Artificial

Campina Grande – PB
Fevereiro de 1999

S586a Silva, Josenildo Costa da
Aquisicao de conhecimentos e manutencao para uma sociedade de agentes tutores artificiais / Josenildo Costa da Silva. - Campina Grande, 1999.
120 f.

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

1. Inteligencia Artificial 2. Aquisicao de Conhecimento 3. Sistemas Tutores Multi-Agente 4. MATHEMA 5. Dissertacao - Informatica I. Costa, Evandro de Barros II. Ferneda, Edilson III. Universidade Federal da Paraiba - Campina Grande (PB) IV. Título

CDU 004.8(043)

AQUISIÇÃO DE CONHECIMENTO E MANUTENÇÃO PARA UMA SOCIEDADE DE AGENTES TUTORES

JOSENILDO COSTA DA SILVA

DISSERTAÇÃO APROVADA EM 25.02.1999

PROF. EDILSON FERNEDA, Dr.
Orientador

PROF. EVANDRO DE BARROS COSTA, D.Sc
Orientador

PROF. GIUSEPPE MONGIOVI, D.Sc
Examinador

PROF. JOSÉ HAMURABI N. DE MEDEIROS, M.Sc
Examinador

PROF. SOFIANI LABIDI, Dr.
Examinador

CAMPINA GRANDE – PB

Este trabalho é dedicado à meus pais
e irmãos.
Dedico-o ainda à minha
querida "Lene".

Resumo

Neste trabalho, abordamos os problemas relacionados com a aquisição de conhecimento para uma sociedade de agentes tutores artificiais, no âmbito do projeto MATHEMA. Neste sentido, propomos um ciclo de aquisição de conhecimento para a construção de agentes tutores inteligentes e apresentamos a especificação de um ambiente de manutenção que dá suporte a este ciclo. Em relação a esse ambiente, definimos sua arquitetura e as funcionalidades de seus elementos. Em particular, definimos os protocolos de interação entre esse ambiente e a sociedade de agentes. Apresentamos ainda um protótipo que implementa a especificação do ambiente de manutenção proposta.

Abstract

In this work we address issues related to the knowledge acquisition for a society of artificial tutoring agents within MATHEMA Project. We present here a Knowledge Acquisition cycle for building tutoring agents and a specification for the maintenance environment that support this cycle. Regarding this environment we have defined its architecture and functionality of its elements. In particular, we solved issues related to interaction protocols between this maintenance environment and the tutoring agents and we developed a prototype implementing this environment.

Agradecimentos

Gostaria de externar meus sinceros agradecimentos, em primeiro lugar, a meus orientadores, pela paciência, presteza e constante estímulo à independência na condução desta pesquisa.

Devo agradecer também aos colegas do Grupo de IA, especialmente a Luciano, pelas discussões sobre MATHEMA, e ao prof. Hamurabi, pelas discussões sobre Aquisição de Conhecimento em geral.

Também agradeço aos demais colegas do mestrado com quem que tive proveitosas discussões sobre o meu trabalho (e sobre o futuro da humanidade). Em especial à todos que fizeram, apoiaram ou participaram das sessões do Cine-COPIN. Meu agradecimento também a todos os outros amigos que fiz em Campina Grande, não ligados ao mestrado em informática.

Não poderia deixar de agradecer também aos funcionários da UFPB (DSC e COPIN) pelo atendimento pronto e atencioso. Agradeço especialmente ao pessoal da MINIBLIO (Zeneide, Arnaldo e Joseluce) e às secretárias da COPIN (Vera Lucia e Aninha).

À CAPES, pelo suporte financeiro, sem o qual este trabalho não teria sido possível.

E à Deus por tudo.

Lista de Figuras

Sumário

Capítulo 1

Introdução

Este trabalho é resultado de pesquisa realizada no âmbito do projeto MATHEMA. Este projeto tem como objetivo desenvolver um modelo de Sistemas Tutores Inteligentes baseados em uma arquitetura multi-agente, bem como uma metodologia para a construção dos mesmos. Esta dissertação se ocupa dos aspectos de Aquisição de Conhecimento no processo de construção de STIs.

1.1 Contexto e Abordagem

Os Sistemas Tutores Inteligentes (STI) são sistemas computacionais que têm como objetivo possibilitar o aprendizado de um estudante sobre um determinado domínio de conhecimento. Esta área de estudo é um encontro de várias áreas de pesquisa tais como Inteligência Artificial, Educação e Psicologia [Kea87]. Muitos sistemas tutores já foram e estão sendo desenvolvidos. Como exemplo podemos citar o pioneiro SCHOLAR [Car70], os STI desenvolvidos por W. J. Clancey GUIDON, NEOMYCIN, HERACLES [Cla86], ou ainda o sistema Pitagora [MCCS94], o GEOMETRY Tutor [ACKP95], o CIRCSIM-Tutor [Fre97], o SHIECC [LF98], além de muitos outros. Atualmente esta é uma área de intensos debates e apresenta grandes desafios para quem deseja nela trabalhar.

Esta dissertação situa-se no contexto das pesquisas em STI, sendo parte do projeto

MATHEMA. Este projeto tem como objetivo desenvolver um modelo conceitual para sistemas de aprendizagem por computador baseado em uma arquitetura multi-agente [CP96, Cos97, CPF98]. A longo prazo este projeto deve dar origem não só a uma metodologia bem definida para a criação de sistemas de aprendizagem mas também a uma ferramenta de autoria para dar suporte a esta metodologia. Um dos conceitos centrais do modelo MATHEMA é a organização do domínio, como veremos no capítulo 3, que dá origem à sociedade de agentes tutores. Aspectos relacionados com a coordenação da sociedade de agentes e comunicação entre os agentes foram tratados em [Jat98]. Já os problemas relacionados com a adaptação do sistema de acordo com as necessidades de um dado aprendiz são tratadas em [Cou99].

Neste trabalho procuramos abordar os problemas relacionados com a aquisição de conhecimento do domínio e criação dos agentes suportada por uma ferramenta [CdS99]. Deste modo, podemos situar esta dissertação na sub-área da Inteligência Artificial relacionada com Sistemas Baseados em Conhecimento (SBC), mais particularmente a Aquisição de Conhecimento (AC).

1.2 Motivação e Objetivos

MATHEMA propõe uma forma de organização dos conhecimentos do domínio no momento da construção do STI. A motivação principal deste trabalho é dar seqüência ao trabalho inicialmente delineado em [Cos97]. A pesquisa desenvolvida neste trabalho, busca antes de tudo, investigar o método de aquisição de conhecimento baseado em modelos, quanto a sua adequação para a construção de sistemas tutores inteligentes.

Assim, nossos objetivos principais neste trabalho são: (1) detalhar o processo de organização, formalização e implementação das bases de conhecimento dos agentes de um STI-MATHEMA; (2) tratar o problema da manutenção da sociedade de tais agentes (criação, retirada e alteração de agentes).

Quanto ao ponto (1), apresentamos uma estrutura para a base de conhecimento do agente MATHEMA e definimos os passos do ciclo de aquisição de conhecimento para a criação de tais agentes. Para isso utilizamos uma abordagem de aquisição baseada em modelos.

Quanto ao ponto (2), propusemos a arquitetura de uma ferramenta de auxílio à modelagem de conhecimento e manutenção da sociedade agentes. Definimos também a linguagem de interação entre a sociedade de agentes e a ferramenta de manutenção.

1.3 Organização da Dissertação

Esta dissertação está organizada da forma apresentada a seguir. No capítulo 2, apresentamos uma introdução à Aquisição de Conhecimento onde discutimos as idéias básicas desta área, apresentamos um resumo de algumas metodologias e mostramos a descrição de algumas ferramentas. Encerramos apresentando a metodologia KADS, que será utilizada como metodologia de base para este trabalho.

No capítulo 3, apresentamos o modelo MATHEMA e todos os conceitos sobre o qual estamos nos baseando neste trabalho, tais como organização do domínio e a arquitetura do agente tutor MATHEMA. No final deste capítulo mostraremos alguns aspectos atuais do MATHEMA, relacionados com a manutenção de conhecimento, que pretendemos tratar neste trabalho.

O ciclo de AC para o MATHEMA é proposto no capítulo 4. Neste capítulo apresentaremos ainda a especificação de uma ferramenta de aquisição de conhecimento para dar suporte a este ciclo.

No capítulo 5 mostramos um protótipo desenvolvido a partir da especificação feita no capítulo 4. Neste capítulo apresentaremos os algoritmos de cada módulo descrito na especificação e discutiremos detalhes relacionados com este protótipo em particular.

Um estudo de caso é apresentado no capítulo 6. Neste capítulo será apresentado o domínio escolhido e seguiremos os passo do ciclo de AC, definido neste trabalho, para construir um agentes tutore MATHEMA para esse domínio.

Encerramos com o capítulo 7, onde tecemos os comentários finais e traçamos os trabalhos futuros.

Capítulo 2

Introdução à Aquisição de Conhecimento

Resumo: A Engenharia de Conhecimento (EC) é a disciplina que se preocupa com os problemas relacionados com a construção de Sistemas Baseados em Conhecimento. Exemplos de tais problemas são: a análise de requisitos, a escolha da forma de representação de conhecimento e a escolha da linguagem de implementação. Dentro da EC podemos destacar um grupo particular de problemas relacionados com os aspectos cognitivos da construção do sistema. Esses problemas são o objeto de estudo da Aquisição de Conhecimento (AC). Neste capítulo, fazemos uma revisão de alguns conceitos, metodologias e ferramentas de AC.

2.1 Introdução

A Engenharia do Conhecimento (EC) é a disciplina voltada para metodologias e técnicas utilizadas na construção de um Sistema Baseado em Conhecimento (SBC)[1]. Os problemas tratados na EC estão relacionados às etapas da criação de um SBC: análise de requisitos, escolha de uma linguagem de programação adequada para imple-

[1]Sistemas Baseados em Conhecimento são sistemas que fazem uso de conhecimento e o representam explicitamente e de modo separado do módulo principal do sistema. Alguns autores consideram os Sistemas Especialistas como um tipo especial de SBC que se destina a tarefas de resolução de problemas em domínios específicos de conhecimento [Fir88, Ste95].

mentação e escolha do formalismo de representação do conhecimento (ver figura 2.1).

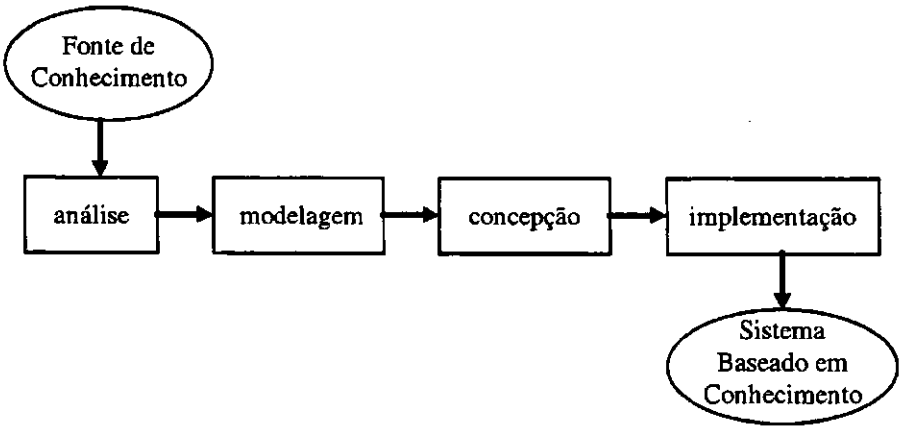


Figura 2.1: Etapas da Engenharia do Conhecimento (copiado de [Lab95])

A *análise de requisitos* do sistema constitui a primeira etapa da construção de um SBC. Seu objetivo é avaliar a viabilidade da construção do sistema. A metodologia CommonKADS, por exemplo, inclui nesta fase inicial uma análise do ambiente onde o futuro sistema irá funcionar, procurando destacar o equipamento computacional já disponível, e capacitação do pessoal que irá trabalhar com o sistema [SWdHA94]. A próxima etapa é a *construção dos modelos do sistema*, incluindo a construção do modelo de conhecimento do sistema. Em particular, para sistemas multi-agentes, a EC inclui a tarefa de definir o modelo dos agentes a ser utilizado no sistema, o modelo de cooperação (que indica como os agentes podem organizar-se para resolver problemas) e um modelo de comunicação (que indica o modo de troca de mensagens entre os agentes do sistema). Em seguida, dá-se a *concepção do sistema*, que origina o projeto lógico. Esse projeto é baseado no modelo de conhecimento do sistema e deve preocupar-se com os métodos de representação do conhecimento, além de incluir outros aspectos mais ligados à engenharia de software, tais como escolha de um paradigma de concepção de sistemas e linguagem de especificação. Um exemplo de paradigma de cocepção é a orientação a objeto. O resultado dessa etapa é a descrição do projeto lógico do sistema. A última etapa é a *implementação do projeto lógico.* Essa etapa envolve a escolha de

uma linguagem de programação de acordo com o paradigma escolhido no momento do projeto lógico do sistema. O resultado dessa etapa é o sistema pronto para ser utilizado.

2.1.1 Abordagens para Aquisição de Conhecimento

Os aspectos cognitivos da EC (por exemplo, escolha da representação do conhecimento ou construção do modelo de conhecimento) são tratados mais especificamente pela disciplina conhecida como Aquisição de Conhecimento (AC). Aqui iremos mostrar uma introdução a essa disciplina. Nosso objetivo é destacar alguns dos conceitos da AC e também dar uma idéia do estado atual sem a preocupação de sermos exaustivos. Inicialmente, devemos fazer uma distinção entre as abordagens da aquisição de conhecimento. As principais abordagens são: Modelagem de Conhecimento (ou Aquisição Baseada em Modelos), Aquisição Semi-Automática e Aquisição Automática. A *Modelagem de Conhecimento* (ou AC baseada em modelos) tem como objetivo modelar formalmente vários aspectos do sistema que se quer construir. Esta abordagem baseia-se na interação entre o engenheiro de conhecimento e o especialista [SWdHA94]. A *Aquisição Semi-Automática* tem a preocupação de desenvolver ferramentas de apoio ao especialista na tarefa de AC [Ale94]. A *Aquisição Automática*, por sua vez, tem como objetivo fazer com que a máquina aprenda por sua própria experiência, através do histórico de problemas já resolvidos ou através de instrução recebida de um especialista [MCM83]. A abordagem adotada neste trabalho será a Modelagem de Conhecimento, principalmente pela facilidade que ela provê para revisão e manutenção do sistema. A AC baseada em modelos nos permite estruturar bem o conhecimento, além de garantir a verificação formal do modelo de conhecimento criado [vHSW97, SWB93]. A abordagem por modelos ainda nos permite inserir no SBC métodos de raciocínios heterogêneos[2] além de facilitar a tarefa de explicação [Cla86, BC95]. Portanto, neste trabalho, sempre que nos referirmos a Aquisição de Conhecimento estaremos nos reportando à Modelagem de Conhecimento. Da mesma forma, qualquer referência neste trabalho à Aquisição Automática de Conhecimento será feita explicitamente.

[2] O SBC pode ter tantos métodos de resolução diferentes quantos forem necessários.

2.1.2 Modelagem de Conhecimento

Nos trabalhos de K. Kausse e de Brian Gaines [Kau93, Gai93] encontramos uma boa visão da história da evolução da AC. A forma de representação de conhecimento dos primeiros SBCs era, em geral, regras de produção[3] e "frames"[4]. A experiência, contudo, mostrou a limitação destas formas de representação para exprimir o conhecimento. Os novos sistemas a cada dia exigiam mais e mais estruturação do conhecimento o que ia além da capacidade de representação por regras ou frames [Mus92].

Idéias iniciais A Hipótese do Nível de Conhecimento [New82] diz que um agente inteligente pode ser descrito por meio de seus objetivos e habilidades, levando-se em conta o *princípio da racionalidade*. Tal princípio preconiza que se um agente tem um objetivo e tem habilidade para atingi-lo, ele o fará. A Hipótese do Nível de Conhecimento nos permite discursar sobre os nossos sistemas de modo independente de linguagem de implementação, nos dando maior liberdade para desenvolver, avaliar e comparar sistemas. O estudo sobre o mecanismo de inferência do sistema MYCIN[5] foi um marco na direção de se expressar os métodos de resolução de problemas de um sistema em um nível mais abstrato [Cla85]. Outros estudos foram feitos em diversos sistemas e percebeu-se que os métodos de resolução utilizados nos diversos SBCs estudados compartilhavam muitas características. Tais estudos serviram de base para a idéia de Tarefas Genéricas (Generic Tasks) [Cha86, AGKS92] onde tarefas de resolução de problemas são reutilizáveis na construção de novos SBCs.

Reusabilidade As idéias acima marcaram o início da busca por reusabilidade em SBC. Esse esforço de pesquisa parte do pressuposto de que para cada classe de pro-

[3]Regras de Produção é um modo de codificação onde o conhecimento é representado em regras no formato: se Condição então Ação. Esta forma de representação foi proposta por Newell e Simon em 1972 (veja cap. 10 de [Fir88]).

[4]"frames"são estruturas formadas por atributos, conectadas em forma de uma rede e que são utilizadas para representação de conhecimento. Esta forma de representação de conhecimento foi proposta Marvin Minsky em 1975 (ver cap. 9 de [Fir88]).

[5]MYCIN é um dos primeiros sistemas especialistas construídos. Ele era capaz de diagnosticar casos de infecção a partir de um conjunto de dados sobre um paciente [Sho76].

blemas (por exemplo diagnóstico) é possível encontrar e descrever *métodos genéricos* (independentes do domínio) para resolução dos problemas dessa classe. Um dos primeiros passos nessa direção foi a proposta de uma taxonomia de problemas [Cla85] que dividia os problemas em classes pré-definidas entre elas diagnóstico, planejamento, monitoramento e classificação. Essa taxonomia ainda é utilizada, com pequenas modificações, por exemplo, na metodologia CommonKADS (ver seção 2.5). A partir de então, vários *métodos genéricos* foram propostos para as várias classes de problemas. Um dos resultados dessas pesquisas foi o desenvolvimento de linguagens para expressar os *métodos genéricos*. Outro resultado foi uma grande quantidade de *métodos genéricos* catalogados pelos diversos grupos de pesquisa (ver [Mus92]). O inconveniente da idéia de *métodos genéricos* é a dificuldade de se descrever *a priori* todos os tipos de problema em uma grande biblioteca. Mas metodologias tais como o CommonKADS e Protégé-II (veja a seção 2.5) oferecem linguagem de construção de métodos genéricos para o caso onde não se encontra um método compatível e se tem que adaptar ou construir um totalmente novo.

AC como Modelagem Atualmente é amplamente aceito na comunidade de AC que a AC é uma atividade de modelagem [SWdHA94]. A AC envolve etapas de identificação dos conhecimentos do domínio e construção de um modelo conceitual do sistema usando uma dada metodologia. Se possível é utilizado como ponto de partida um modelo já catalogado como método genérico [ABB+92]. Ao ser encontrado um método genérico compatível com o sistema a ser modelado deve-se tentar adaptá-lo (se necessário) ao domínio específico para o qual o SBC está sendo construído. O modelo conceitual resultante simplifica a atividade de coleta de dados na medida em que especifica e delimita o tipo de conhecimento necessário para se resolver o problema em questão [vHSW97]. Quando o engenheiro não encontra um *método genérico* compatível, deve construi-lo, usando para tanto, a liguagem oferecida pela metodologia de modelagem de conhecimento de sua escolha.

2.2 Etapas da AC

A AC é constituída por pelo menos duas grandes etapas: elicitação e modelagem. Na etapa de elicitação o engenheiro de conhecimento usa técnicas para definir e esboçar uma primeira estrutura do domínio. O resultado dessa etapa é chamado *Ontologia de Domínio*, constituída do dicionário de termos (o vocabulário do especialista identificado no seu discurso) e a relação entre os conceitos do domínio. Para essa etapa, existem várias técnicas já amplamente utilizadas pela comunidade de AC, como p. ex., Redes de Repertório [Kel55] [BB87] e Análise de Protocolos [RW89]. O estudo destas técnicas está fora do escopo deste trabalho. O leitor interessado em uma análise mais aprofundada de técnicas de elicitação deve consultar os trabalhos de Rose Dieng [Die90], J. Diederich e M. Linster [DL89], Nathalie Aussenac-Gilles et al. [AGKS92], ou ainda o trabalho de John Kingston [Kin94]. Na etapa de modelagem, o engenheiro procura construir o modelo conceitual (ou modelo de conhecimento) do sistema. Esse modelo conceitual serve tanto para guiar o processo de concepção (*design*) e implementação do sistema, como para organizar os dados a serem coletados (instanciação do modelo conceitual). Serve também como meio de comunicação entre o engenheiro e o especialista [SWdHA94]. Outra utilidade do modelo conceitual é permitir uma verificação prévia da consistência do conhecimento adquirido. Tal verificação é possível graças ao fato de que o modelo conceitual é semi-formal, o que possibilita a detecção de algumas inconsistências [GS94, BB98]. O especialista também pode ajudar a verificar inconsistências no modelo conceitual, uma vez que o modelo é escrito usando o vocabulário do seu domínio de especialidade [BC95].

2.3 Fundamentos da Modelagem do Conhecimento

G. van Heijst et al., no seu artigo [vHSW97], apresenta os princípios que fundamentam as metodologias de AC atuais. Aqui procuramos resumi-los:

Limitação por Papéis. Esse é um mecanismo conceitual que permite organizar o conhecimento através de restrição do modo como os elementos do conhecimento de um determinado tipo podem ser utilizados no raciocínio.

Tipos de Conhecimento. Na literatura de AC os tipos (categorias) de conhecimentos que compõem uma base de conhecimento são resumidos abaixo [PETM92, SWA+94, SG95](veja figura 2.2):

- *Tarefa.* É uma descrição dos objetivos que precisam ser atingidos pela resolução de problema.
- *Método de Resolução de Problemas* (MRP). É a descrição de uma estratégia para se alcançar o objetivo estabelecido em uma tarefa. Os passos de resolução podem ser outras tarefas (subtarefas), quando forem passos complexos demais para serem representados atomicamente. Os passos atômicos são as inferências. Um MRP apresenta um conjunto de suposições e requerimentos para ser aplicado. Por exemplo o MRP "diagnóstico", para tarefas de diagnóstico médico, supõe que se tenha uma representação causal entre sintomas e doenças. Se o domínio não atender aos requerimentos de um MRP deve-se procurar (ou desenvolver) outro.
- *Instância de Tarefa.* Uma tarefa descreve um objetivo. Um MRP diz os passos para se atingir um objetivo. Mas note que uma tarefa pode ser satisfeita por vários MRP diferentes. Por exemplo se tomarmos como exemplo a tarefa "resolver equação do segundo grau", podemos listar vários MRPs que podem satisfazê-la: "resolução por substituição", "resolução pela fórmula de Bháskara", etc. Quando decidimos que uma tarefa vai ser realizada por um determinado MRP[6] representamos isto em uma *instância de tarefa*, que nada mais é que uma estrutura que apresenta um objetivo juntamente com uma estratégia escolhida (possivelmente dentre muitas) para atingi-lo.
- *Inferência.* Descreve os passos primitivos no processo de resolução de problemas. Descrevem a natureza das entradas e saídas e o conhecimento necessário para se derivar uma saída a partir das entradas. Em outras palavras, uma inferência é uma especificação das relações entre as suas entradas e sua saída, sem apresentar "como" isto é feito. O "como" é uma decisão

[6]Os cirtérios que guiam a escolha de um MRP para a realização de uma tarefa são as suposições e requisitos anotados na definição do MRP

de implementação e varia dependendo do poderio que o ambiente computacional (linguagens, equipamentos, bibliotecas de programas, ferramentas de análise, etc.) oferece ao projetista. Juntas, as inferências formam um modelo funcional do sistema. Pode-se dizer que as inferências são invocadas pelas tarefas.

- *Ontologia.* Descreve os termos utilizados (conceitos) pelo domínio e relação entre esses termos. Estes termos são referenciados pelas inferências.
- *Conhecimento de Domínio.* É formado pelas instâncias dos termos descritos na ontologia. Este conhecimento é provido pelo especialista no domínio após o engenheiro de conhecimento ter definido a ontologia da aplicação. Caso haja uma ferramenta adequada, não há necessidade de interferência do engenheiro de conhecimento durante este processo.

Reusabilidade. Atualmente, há uma ênfase na reutilização de componentes de conhecimento entre diferentes domínios. As bibliotecas, providas pelas diferentes metodologias, são de grande auxílio, embora cada metodologia tenha uma nomenclatura diferente. Geralmente, uma biblioteca de modelos é o ponto de partida para a construção de um novo SBC. Geralmente as metodologias de modelagem de conhecimento apresentam bibliotecas de modelos. Além do mais as metodologias apresentam linguagens para construção de novos modelos, o que permite a adição de novos item à biblioteca. Isso é particularmente útil quando não encontramos em uma biblioteca um modelo adequado ao sistema que desejamos modelar.

Uso de modelos estruturais. Geralmente os componentes de conhecimento são reutilizados em forma de modelos estruturais. Um modelo estrutural define apenas uma parte de um modelo conceitual. O engenheiro tem que preencher as outras partes a fim de obter um modelo conceitual final do sistema. Pode-se encontrar modelos estruturais para métodos de resolução de problemas e para ontologias (veja por exemplo [TSD+92, VL93, FFR97]).

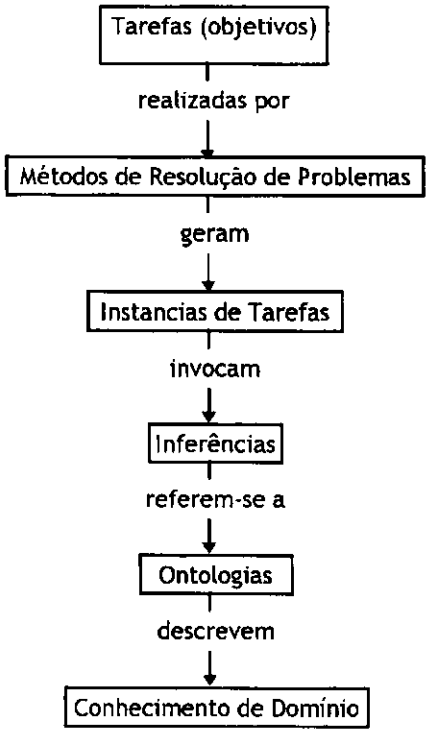


Figura 2.2: Categorias de Conhecimento (baseado em [vHSW97])

2.4 O processo de Modelagem

O processo de modelagem consiste em quatro atividades [vHSW97]:

Construção de um Modelo de Tarefas para a Aplicação A atividade inicial na construção de um SBC é a análise das tarefas. É nessa fase que se identificam as tarefas do SBC e os métodos de resolução capazes de realizá-las. O modelo de tarefas é composto das tarefas e dos métodos identificados na análise.

Seleção e Configuração de uma Ontologia para a Aplicação Corresponde à construção da ontologia específica para a aplicação. Caso exista uma biblioteca de ontologias o processo pode ser facilitado.

Mapeamento do Modelo de Tarefa na Ontologia da Aplicação A ontologia define os principais conceitos do domínio. Nessa fase define-se a associação entre

os elementos da ontologia e os papéis (estabelecidos no modelo de tarefas) por eles assumidos. Por exemplo, em diagnóstico médico, instâncias do conceito doença geralmente irão assumir o papel de hipótese.

Instanciação da ontologia da Aplicação O conhecimento de domínio descreve as instâncias dos conceitos da ontologia da aplicação. Este conhecimento é apresentado pelo especialista. A separação entre conhecimento de aplicação e ontologia permite que esta seja reutilizada em outros domínios.

2.5 Metodologias de Construção de SBC

Há várias metodologias disponíveis ao engenheiro de conhecimento. Algumas dessas metodologias preocupam-se exclusivamente com a AC, outras envolvem todo o processo de construção de SBC. Uma metodologia de AC geralmente apresenta um conjunto de construtores básicos, modelos e regras indicando como e quando construir tais modelos. Dada a dificuldade de construção e gerenciamento de uma variedade de modelos utilizados em uma certa metodologia de AC dirigida por modelos, é importante que uma tal metodologia seja suportada por ferramentas automatizadas, o que facilita bastante a tarefa para o engenheiro de conhecimento. As primeiras metodologias para modelagem do conhecimento surgiram em meados dos anos 80 com Clancey [Cla85], Chandrasekaran [Cha86] e McDermott [McD88] baseadas na Hipótese do Nível do Conhecimento (veja seção 2.1.2). M. Linster, em [Lin93], apresenta uma visão geral da quantidade e variedade de metodologias para engenharia e aquisição de conhecimento desenvolvidas desde então. Como as metodologias de modelagem de conhecimento têm em comum várias idéias fundamentais (veja seção 2.3), discutiremos aqui somente algumas das metodologias mais conhecidas.

Classificação Heurística. A primeira proposição de modelo conceitual foi feita por W. Clancey [Cla85] através da análise do sistema MYCIN. Ele descreveu o mecanismo de diagnóstico, chamado por ele de Classificação Heurística. Uma vez que essa descrição é independente de domínio, ela é considerada um modelo conceitual para o

problema de diagnóstico médico [BC95]. A Classificação Heurística não é, entretanto, uma metodologia de AC; é, antes, uma descrição de um MRP particular (para diagnóstico médico). A sua importância consiste em ser o primeiro passo na direção da AC orientada a construção de modelos.

Tarefas Genéricas. Essa metodologia apresenta a idéia de Tarefas Genéricas (que deu nome à metodologia). As Tarefas Genéricas são descrições abstratas do processo de resolução de problemas [Cha86]. Uma Tarefa Genérica descreve parâmetros de entrada/saída, métodos apropriados para execução da tarefa, e conhecimentos necessários para execução da tarefa. Essa metodologia ainda apresenta operações de construção de tarefas a partir de tarefas já definidas, como por exemplo, composição de duas ou mais tarefas, ou especialização de uma tarefa. Construir o modelo conceitual nessa abordagem é visto como a utilização dessas tarefas (e operações sobre tarefas) de modo a se construir o método de resolução de problema desejado [AH93].

Método Limitado por Papéis. Semelhante à abordagem de Chandrasekaran, essa abordagem procura prover blocos básicos de construção para métodos de resolução de problema. Cada bloco possui uma descrição abstrata do modo de resolução de um tipo de problema e assim permite determinar qual o papel dos conhecimentos em uso [McD88]. Esse é o motivo do nome Limitado por Papéis. Essa abordagem também conta com uma biblioteca de blocos básicos para serem utilizados na construção do modelo conceitual. Exemplos de métodos estudados por essa metodologia são propôr-e-rever, cobrir-e-diferenciar, e refinamento-de-plano-estrutural.

Componentes de Perícia. Segundo essa abordagem o modelo conceitual deve ser formado por três componentes: modelo, tarefas, e método [Ste90]. Nesta abordagem *modelo* indica o conhecimento necessário para se atingir um objetivo, *tarefa* representa um objetivo a ser atendido e um *método* indica como usar os conhecimentos para atender uma dada tarefa. Essa abordagem não fornece biblioteca de blocos básicos pré-definidos, mas somente mostra como identificá-lo e estruturá-los na construção de MRPs para a aplicação sendo desenvolvida.

KADS. É uma metodologia de construção de SBC desenvolvida no âmbito do projeto KADS-I [BW89], do programa ESPRIT da Comunidade Européia [SWB93]. Essa metodologia envolve todas as etapas da construção de um SBC. Segundo essa metodologia, a construção do SBC passa pela construção de vários modelos independentes, que capturam diferentes características do sistema e do seu ambiente. O KADS-II[7] é uma continuação do projeto KADS-I, e que gerou a metodologia CommonKADS [SWA+94, SWdHA94]. Nesta versão do KADS a linguagem de modelagem conceitual foi modificada e foi introduzida a noção de componentes de granularidade mais fina na biblioteca provida pela metodologia. A filosofia do CommonKADS é também ligeiramente mais ambiciosa que o KADS. O CommonKADS pretende ser uma metodologia de integração de metodologias orientadas a modelo existentes atualmente [VBB93].

Protégé. Seguindo a idéia geral da aquisição orientada a modelos, a metodologia Protégé[8] oferece também um conjunto de construtores básicos e operações para se construir o modelo do conhecimento [Mus92]. Também oferece uma biblioteca de modelos genéricos. Esta metodologia é bastante utilizada, principalmente nos EUA, e várias ferramentas já foram desenvolvidas por este grupo de pesquisa [TKM+89, MFCS87, Mus89, PETM92, TSD+92].

2.6 Ferramentas

As ferramentas são parte importante de uma metodologia de Aquisição de Conhecimento porque automatizam o processo de aquisição diminuindo o tempo de desenvolvimento de SBCs, além de permitir uma melhor documentação e manutenção desses SBCs. As ferramentas variam de metodologia para metodologia em propósito e em capacidade de auxílio. Aqui veremos algumas classes de ferramentas de aquisição de conhecimento com alguns exemplo de cada classe. Para uma discussão mais profunda sobre ferramentas o leitor pode consultar [vHSW97] ou ainda [Gai93].

[7] http://www.swi.psy.uva.nl/projects/CommonKADS/home.html
[8] http://smi-web.stanford.edu/projects/protege/

Shells & Ferramentas para Manutenção de Bases de Regras. As primeiras ferramentas de AC foram criadas para dar apoio aos sistemas baseados em regras e "frames". Muitas eram somente editores de regras, mas com o tempo, foram desenvolvidas ferramentas que incorporavam outras funcionalidades, tais como capacidade de explicação e verificação de consistência da base de regras. Um exemplo dessas ferramentas mais elaboradas é o TEIREISIAS [Dav76]. Já os "shells" são ferramentas geralmente compostas de um interpretador de regras e uma base vazia. Os *shells* são úteis para o desenvolvimento rápido de sistemas relativamente simples, já que uma das limitações mais sérias dos SBC com uso de regras é o gerenciamento da base quando o número de regras aumenta muito. Um exemplo dessas ferramentas é o EMYCIN, composto de um interpretador de regras MYCIN e uma base vazia, a ser preenchida pelo usuário [vMSBP81]. Um exemplo mais atual de shell é o ExpertSINTA, do Grupo de Sistemas Inteligentes Aplicados, Laboratório de IA, UFCE [GSI97].

Ferramentas para Elicitação. Para elicitação temos as ferramentas que auxiliam o engenheiro a montar um primeiro esboço dos termos do domínio. As ferramentas para essa etapa ajudam principalmente a criar a hierarquia de conceitos do domínio e o dicionário de termos. Exemplos são AQUINAS [BB87], KRITON [DRM87] e TOPKAT[9] [Kin94].

Ferramentas de MRP fixo. Essas ferramentas foram desenvolvidas para servirem de apoio a métodos específicos de resolução de problema. Foram desenvolvidas principalmente como resultado das pesquisas sobre métodos de resolução de problemas independente de domínio [Cha86, Cla85, McD88]. Assim, dado um domínio, o engenheiro escolhe um método de resolução adequado e usa uma ferramenta para instanciar o conhecimento dependente de domínio obtendo-se assim um sistema particular ao domínio. O papel que o conhecimento tem é determinado pela estratégia de resolução de problemas adotado. Desse modo, a tarefa da ferramenta é guiar o engenheiro no preenchimento de tais papéis [GM96]. As ferramentas de AC baseada em um método fixo são de grande auxílio para a classe de problemas a que se propõem mas tem algumas

[9]http://www.aiai.ed.ac.uk/ jkk/topkat.html

limitações, sendo as principais [TSD+92]:

- *O motor de inferência não é facilmente extensível.* Caso se queira fazer uma modificação no método, é necessário modificar o código e recompilar a ferramenta;
- *atributos dependentes do domínio podem não se encaixar nos papéis do método.* Quando o método genérico codificado na ferramenta e o domínio apresentam ligeiras diferenças torna-se difícil, e até inviável, a utilização da ferramenta.

Exemplos de tais ferramentas são SALT [MM89] para o método Propôr-e-Rever[10], MOLE [Esh88] para o método Cobrir-e-Diferenciar e PROTÉGÉ [TSD+92] para Refinamento-de-Plano-Estrutural.

Ferramentas de Edição de Modelos. Visando superar as limitações das ferramentas para métodos fixos, foram desenvolvidas as ferramentas que ajudam o engenheiro a criar métodos de resolução e instanciá-los, tornando a ferramenta independente de método. Em geral, o engenheiro dispõe de uma biblioteca de construtores básicos para que ele crie um método de resolução de problema através de um editor de métodos. Em seguida o engenheiro pode partir para uma seção de aquisição orientado pelo método construído, o que resulta em uma forma de aquisição bem mais adaptada ao domínio que as normalmente disponíveis em ferramentas para métodos genéricos. Exemplos são o Protégé-II [PETM92] e o CoKACE [CD96].

Ambientes Integrados. São ambientes onde se pode encontrar um grande número de ferramentas que, juntas, auxiliam o engenheiro nas atividades da EC. Geralmente esses ambientes possuem ferramentas para elicitação, modelagem, e projeto do SBC. Exemplos são o KEW [AH93] e o Shelley [AWT92]. Algumas dessas ferramentas ainda ajudam na implementação[11], como é o caso do VITAL [DMW93].

[10]Propôr-e-Rever e Cobrir-e-Diferenciar e Refinamento-de-Plano-Estrutural são métodos de resolução de problemas estudados pela metodologia Métodos Limitados por Papéis

[11]Na realidade o Protégé-II, mesmo não sendo um ambiente integrado, também gera um código CLIPS automaticamente, pronto para ser executado pelo interpretador CLIPS. Isto pode ser considerado implementação.

2.7 Construção de SBC usando KADS

Na seção 2.5 foi mostrado um grande número de idéias fundamentais da AC. As metodologias atuais baseiam-se em tais idéias, diferindo entre si algumas vezes por pequenos detalhes. Assim, dentre as opções de metodologias de AC baseada em modelos, escolhemos a metodologia KADS[12] como base para o desenvolvimento deste trabalho. Na seção 2.10 discutimos algumas razões que nos levaram a escolher o KADS. Nesta seção apresentaremos os princípios básicos do processo de AC e implementação de SBC usando o KADS[13]. KADS considera a construção de um SBC como uma atividade de modelagem [SWdHA94]. Para isso apresenta sete modelos que devem ser desenvolvidos (ou instanciados) para um sistema a ser construído. Cada modelo, captura um aspecto diferente do sistema. Os modelos propostos pelo KADS são:

Modelo de Organização. MO é um meio de analisar o ambiente organizacional onde o SBC irá ser instalado.

Modelo de Tarefas. MT é uma descrição em nível genérico das tarefas que serão realizadas pelo SBC na organização onde ele será instalado.

Modelo de Agente. MA é o modelo que descreve as capacidades e as características particulares dos agentes. Note que os agentes são executores de tarefas, e podem ser humanos, programas de computador ou qualquer outra entidade capaz de executar as tarefas descritas no MT.

Modelo de Comunicação. MC descreve os detalhes da troca de informações entre os diferentes agentes envolvidos na execução das tarefas descritas no MT.

Modelo de Perícia. MP é o modelo onde o Conhecimento Especializado é declarado. Também é referido como Modelo de Conhecimento. É o ponto principal

[12]Usaremos os termos KADS e CommonKADS como sinônimos para a metodologia de AC CommonKADS.

[13]Algumas metodologias traduzem automaticamente o modelo de conhecimento para alguma linguagem de representação de conhecimento, por exemplo o Protégé-II gera regras de produção para o sistema CLIPS. No KADS a implementação do sistema é responsabilidade do projetista onde informações adicionais relativas a detalhes de implementação são acrescentadas ao modelo de conhecimento.

da metodologia KADS. É formado por três níveis: o nível de domínio, o nível de tarefas e o nível de inferência. O KADS fornece também uma biblioteca de construtores básicos para o desenvolvimento deste modelo. O engenheiro de conhecimento pode ainda definir novos elementos, caso não haja algo na biblioteca que seja aplicável ao problema em estudo. Isso pode ser feito através do uso da linguagem de modelagem conceitual do KADS, o CML (Conceptual Modeling Language [SWA+94]).

Modelo de Implementação. MI é o modelo onde são descritos os módulos que precisam ser implementados, os mecanismos computacionais necessários e o modo de implementação da comunicação entre os módulos implementados.

Uma das maiores crítica ao KADS é sua complexidade, que pode parecer desnecessária para sistemas de pequeno porte. Mas a idéia é que estes modelos sejam vistos como um *arsenal* e não como uma grande arma, isto é, deve-se usar os modelos que nos parecerem necessários na construção de um sistema. Um pequeno sistema pode desprezar a análise organizacional (que origina o MO), mas para um grande sistema esta análise pode ser muito útil. Cabe ao projetista de SBC escolher quais são os modelos relevantes para o seu projeto. Neste trabalho não precisamos de todos os modelos do KADS. Por exemplo não estaremos utilizando o MO uma vez que a análise organizacional é importante para grandes sistemas[14]. A decomposição de tarefas não é feita a priori, dependendo do problema que é apresentado para o agente, assim não utilizamos o MT. O MA não é utilizado porque o MATHEMA tem sua própria definição de agente, que possui as características de tutor. O agentes KADS é mais genérico. O MC também não é utilizado neste trabalho pois os agentes MATHEMA tem um modelo de comunicação embutido, representado pelo conhecimento social do agente juntamente com um conjunto de protocolos de interação (cooperação e licitação) entre os agentes. Quanto ao MI, este não será utilizado por motivos de simplificação. Mas ele é extremamente útil principalmente como documentação do sistema. Assim nos resta, o Modelo de Perícia (MP), ou Modelo Conceitual. Este é o mais importante sob

[14]Talvez seja útil a idéia de análise organizacional para o STI final, mas não para a construção do pequeno módulo de resolução de problemas do agente.

o ponto de vista do funcionamento final do SBC. E é sobre este modelo que daremos enfoque na próxima seção.

2.8 Modelo de Perícia

O MP tem como objetivo capturar os aspectos essenciais para a resolução de problemas do domínio. A definição detalhada do MP pode ser encontrada no artigo [SWA+94]. O MP é dividido em vários níveis, onde cada nível contém um tipo particular de conhecimento. Os níveis do MP são:

- *Nível Domínio*: conhecimento sobre o esquema do domínio e instâncias do domínio.
- *Nível de Inferências*: conhecimento dos passos básicos no processo de resolução de problemas.
- *Nível de Tarefas*: conhecimento de mais alto nível no processo de resolução de problemas.
- *Nível de Resolução de Problemas*: conhecimento estratégico sobre o processo de criação das tarefas utilizadas na resolução de um problema. É um tipo de conhecimento que pode permitir maior flexibilidade ao SBC [AWS94]. Entretanto, não estamos utilizando este conhecimento no nosso agente MATHEMA[15].

As próximas seções detalham os níveis do MP. A sintaxe e comentários detalhados sobre a linguagem de construção do MP são encontrados no artigo [SWA+94]. A equipe que desenvolveu o KADS a chama de Conceptual Modeling Language[16] (CML).

2.8.1 Nível de Domínio

O conhecimento de domínio é formado por três partes:

[15]Incluir este tipo de conhecimento é uma investigação interessante e digna de atenção, mas está fora do escopo deste trabalho.

[16]Linguagem de Modelagem Conceitual

1. *Ontologia*: contém as definições do esquema do conhecimento do domínio. É através da ontologia que são definidos os conceitos do domínio e as relações entre eles.

2. *Modelos do Domínio*: denota partições da base de conhecimento que contém expressões que utilizam definições da ontologia. O *modelo de domínio* pode ser compreendido como uma coleção de instâncias da ontologia.

3. *Mapeamentos*: consiste das descrições de como uma ontologia é mapeada em uma outra.

Definições de Ontologia

A ontologia é constituída das definições dos elementos básicos da estrutura do conhecimento do domínio. O KADS define as seguintes primitivas para a construção da ontologia:

Conceito. Representa um objeto real ou abstrato. Exemplos de conceito são: *carro*, *equação*, a nota musical *fá*, etc. Podemos ver um conceito como algo passível de receber um identificador e ser coletado junto com outros objetos em uma mesma classe. Um conceito é definido por um *nome* associado a um conjunto de *propriedades*. Cada propriedade tem um conjunto de valores associado. Conceito é representado graficamente por um retângulo (fig. 2.3).

Atributo. Associa um conceito a um conjunto de valores. Tem um *nome* e um *conjunto-de-valores*. Pode ser visto como um conceito com apenas uma propriedade "conjunto-de-valores". Um conceito é representado por um retângulo com a indicação do seu conjunto de valores ao lado. (fig. 2.4)

Expressão. A forma geral de uma expressão é:

$$< operando >< operador >< valor >$$

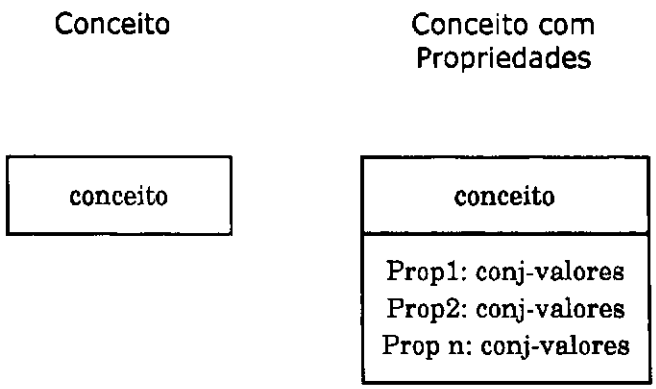


Figura 2.3: Representação gráfica de um conceito em KADS

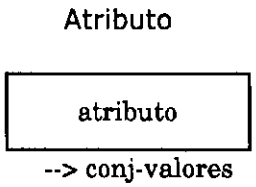


Figura 2.4: Representação gráfica de um atributo

onde, *operando* é um atributo ou uma propriedade de conceito, *operador* é um operador de comparação ou de pertinência de conjuntos e *valor* é um subconjunto do conjunto-de-valores do operador (isto é, atributo, propriedade de conceito). É representada por uma oval (fig. 2.5).

Estrutura. Este construtor tem o objetivo de descrever objetos cuja estrutura interna não se deseja descrever em detalhes em um determinado momento da análise (geralmente nos estágios iniciais da construção do MP). É representada graficamente como um retângulo, exatamente como um *conceito*.

Relação. Este construtor modela o relacionamento entre dois ou mais objetos definido na ontologia. É representado graficamente como um losango (fig. 2.6).

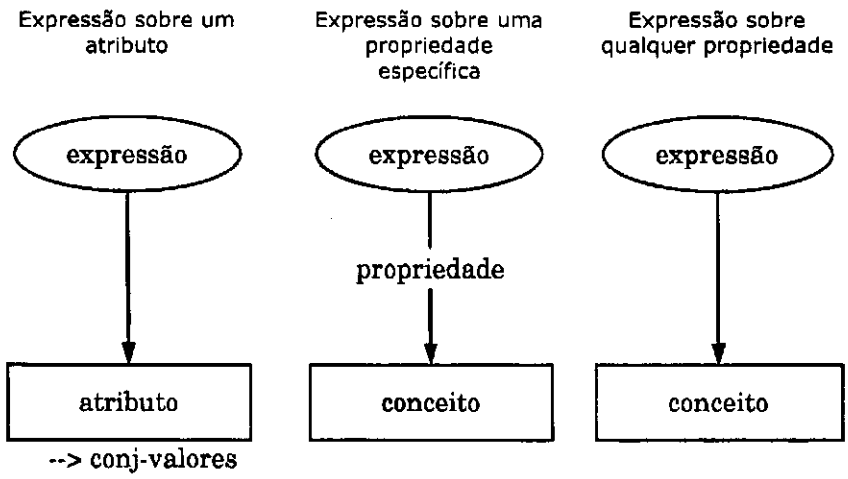


Figura 2.5: Representação gráfica de uma expressão

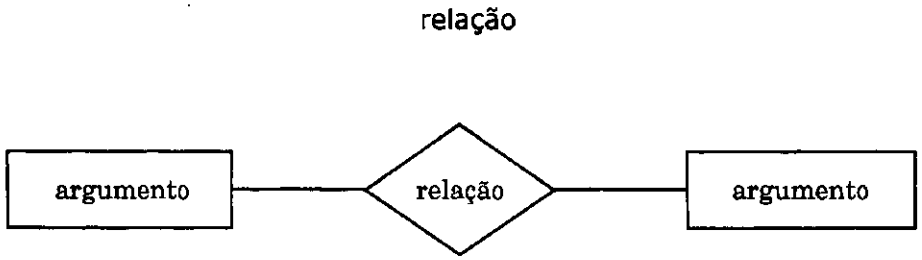


Figura 2.6: Representação gráfica de uma relação

Modelo de Domínio

É uma coleção de expressões sobre o domínio que representam uma instância da ontologia. No KADS, o modelo de domínio é constituído de *instâncias* (para conceitos, expressões e estruturas) e *tuplas* (para instâncias de relações).

Mapeamentos de Ontologia

É utilizado de modo bastante informal. Geralmente é uma descrição textual de como os elementos de uma ontologia são mapeados nos elementos de uma outra ontologia.

Isso facilita o trabalho de reutilização de ontologias[17].

2.8.2 Nível de Inferências

O conceito genérico de inferência já foi apresentado na seção 2.3. A utilização de inferências é um dos pontos cruciais na construção de modelos de conhecimento. O KADS apresenta uma taxonomia básica de inferência, mas o engenheiro de conhecimento tem liberdade para especificar novas inferências sempre que necessário [Abe93]. Sua definição no KADS consiste de um *nome*, a especificação do *tipo-de-operação* e a definição dos conhecimentos que assumem *papéis* de entrada e saída (papeis de conhecimento). A cada papel de conhecimento é indicado qual o elemento da ontologia para o qual ele é mapeado. Para os papéis estáticos (não modificados pela inferência) devemos indicar qual o modelo de domínio acessado. As inferências são representadas graficamente como ovais e os seus papéis como retângulos, como visto na figura 2.7.

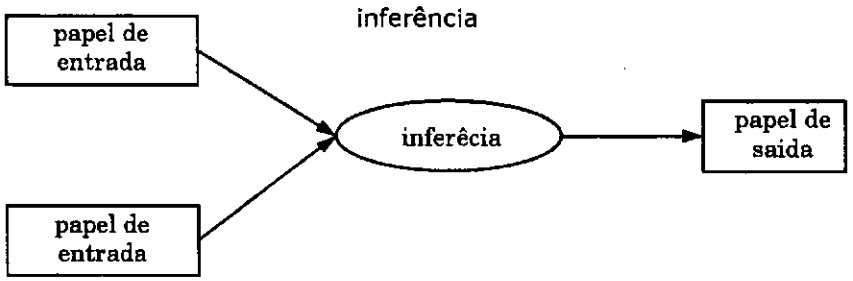


Figura 2.7: Representação gráfica de uma inferência.

2.8.3 Nível de Tarefas

Conforme apresentado por Schreiber et al., [SWA+94], a definição de uma tarefa KADS consiste de duas partes: *Definição da Tarefa* e *Corpo da Tarefa*. A *definição* da tarefa

[17]Na versão de KADS utilizada aqui, não há definições mais rigorosas para mapeamentos. Definir mais formalmente estes mapeamentos pode ser um ponto de pesquisa interessante por facilitar a reutilização de ontologias.

corresponde ao conceito de tarefa visto na seção 2.3. Já o *corpo* corresponde ao conceito de método de resolução visto também na seção 2.3. A *definição da tarefa* define o objetivo da tarefa e consiste de:

Objetivo. Um texto descrevendo o objetivo geral a ser alcançado pela tarefa.

Entrada/Saída. Definição dos papéis de conhecimento manipulados pela tarefa. Os papéis de tarefas não são instanciados diretamente no domínio e sim em papéis dinâmicos das inferências. Estes últimos são instanciados no conhecimento do domínio, via mapeamento definido para inferências.

Especificação. Especificação da dependência entre os papéis de conhecimento manipulados pela tarefa.

O *corpo da tarefa* descreve como o objetivo da tarefa pode ser alcançado. É uma descrição procedural das atividades a serem realizadas para se satisfazer a tarefa. Schreiber et al. [SWA+94] definem três tipos de tarefas de acordo com o seu corpo:

Compostas. São decompostas em subtarefas.

Primitivas. São formadas apenas por inferências.

De Transferência. São as tarefas que interagem com o usuário. O corpo de uma tarefa de transferência não é definido no modelo de perícia, mas no modelo de comunicação.

O *corpo da tarefa* consiste das seguintes partes:

Tipo da Tarefa. Composta ou Primitiva.

Decomposição. Subtarefas em que a tarefa se decompõe.

Método de Resolução de Problema. É o nome do método de resolução de problemas que foi utilizado para se chegar a esta decomposição.

Papéis Adicionais. Papéis adicionais induzidos pela decomposição.

Estrutura de Controle. Descrição do controle sobre as subtarefas para a realização da tarefa.

Suposições. Suposições adicionais devido a decomposição, por exemplo, restrições a certos tipos de estrutura de conhecimento.

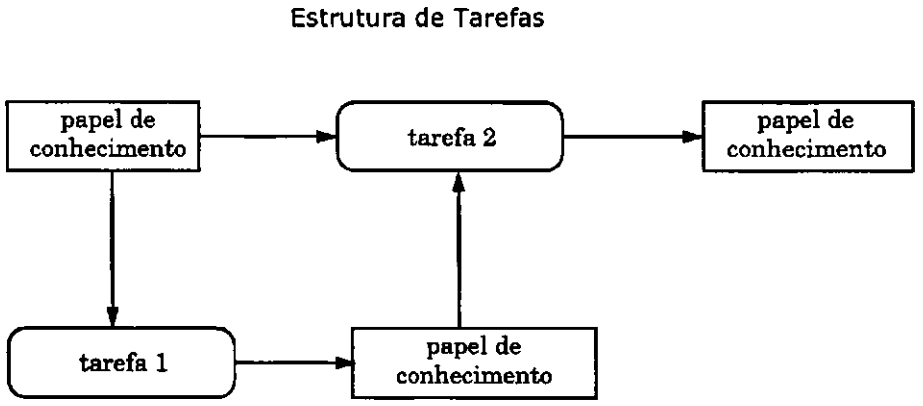


Figura 2.8: Estrutura de Tarefas utilizadas no Modelo de Perícia

As tarefas são representadas graficamente por retângulos de cantos arredondados. Os papéis, por retângulos normais. Geralmente, utiliza-se a representação gráfica das tarefas em uma forma de *estrutura de tarefas* denotando a relação entre os papéis e as tarefas (fig. 2.8). Note contudo que tal estrutura de tarefas não representa o fluxo da informação durante execução do sistema.

2.9 Implementação do Modelo de Perícia

O MP descreve o processo de raciocínio do SBC, mas não é um modelo executável do sistema. O KADS apresenta o Modelo de Implementação (MI) destinado a capturar as particularidades da implementação do sistema. O MI contém informações sobre a plataforma utilizada, linguagem de implementação, equipamentos computacionais. Como já foi dito na seção 2.7, optamos por não usar esse modelo. Porém, esse modelo tem, indubitavelmente, um papel importante no desenvolvimento de aplicações. Neste

trabalho, aproveitamos as idéias principais apresentadas por Schreiber [Sch93] sobre a etapa de implementação do Modelo de Perícia no KADS. É o que apresentaremos a seguir.

2.9.1 Projeto com Preservação de Estrutura

O conceito principal apresentado por Schreiber é o Projeto com Preservação de Estrutura, que visa construir uma implementação que tenha a mesma informação e *estrutura* do MP. Em outras palavras, deveria ser possível reconstruirmos o MP a partir da implementação do SBC. Na realidade, para Schreiber, a fase de implementação deveria apenas acrescentar detalhes de implementação ao MP, sem lhe dar outra estruturação. Preservação da informação é a noção chave. Vejamos quais são as informações contidas no Modelo de Perícia:

Perspectiva Funcional. Sob o ponto de vista da análise funcional, o Modelo de Conhecimento contém informação sobre a árvore de decomposição funcional. As tarefas compostas são vistas como funções de mais alto nível, enquanto que as tarefas primitivas são as folhas da árvore. Em outras palavras, a estrutura de tarefas representa a informação sobre a estrutura funcional do SBC.

Perspectiva de Dados. A informação da perspectiva dos dados consiste do Conhecimento de Domínio, que representa a especificação de todo o conhecimento utilizado pelo SBC. Sob esse aspecto, outras informações são os mapeamentos entre os papeis de entrada e saída das tarefas e inferências, bem como a visão de domínio das inferências.

Perspectiva de Controle. Essa informação está presente na estrutura de controle das tarefas.

2.9.2 Arquitetura Genérica de SBC

As principais informações que se deve acrescentar ao MP para se implementar um SBC são: métodos de inferências, funções de acesso ao domínio, funções de memória

de trabalho, e funções de entrada e saída. Discutiremos com mais detalhes cada uma delas.

- *Métodos de Inferência.* Os métodos de inferência são técnicas computacionais que implementam inferências[18]. Podemos pensar nos métodos de inferência como uma biblioteca de programas que realizam aquilo que é especificado em mais alto nível pelas inferências. Schreiber alerta, porém, que o relacionamento entre inferências e métodos de inferências não é um-para-um. Um método pode ser utilizado para realizar várias inferências diferentes[19].

- *Acesso ao Domínio.* São as funções de acesso ao conhecimento de domínio. São também responsáveis por resolver os mapeamentos de papéis de conhecimentos em elementos de domínio. Estas funções garantem que as inferências possam ser descritas de modo independente de domínio.

- *Memória de Trabalho.* É o local onde são guardadas as informações temporárias geradas durante o processo de resolução de problemas. Aqui as principais informações armazenadas são os valores assumidos por papéis de entrada e saída das tarefas.

- *Funções de Interação Homem-Máquina.* Uma implementação de sistema tem que levar em consideração a interação deste sistema com agentes externos. Estas funções provêm essa capacidade.

Schreiber propõe uma arquitetura de SBC genérico que apresenta os módulos necessários para atender às funções descritas acima. Tal arquitetura serve de paradigma de construção de SBC obedecendo ao princípio de Projeto com Preservação de Estrutura. Esta arquitetura é apresentada na figura 2.9. Os componentes desta arquitetura são:

Base de Conhecimento & Funções de Acesso. A base de conhecimento é uma declaração simbólica do conhecimento do domínio (ontologia e instâncias). As

[18]Conforme visto na seção 2.3, as inferências não especificam *como* implementá-las.

[19]Alguns exemplos podem ser encontrados em [Sch93].

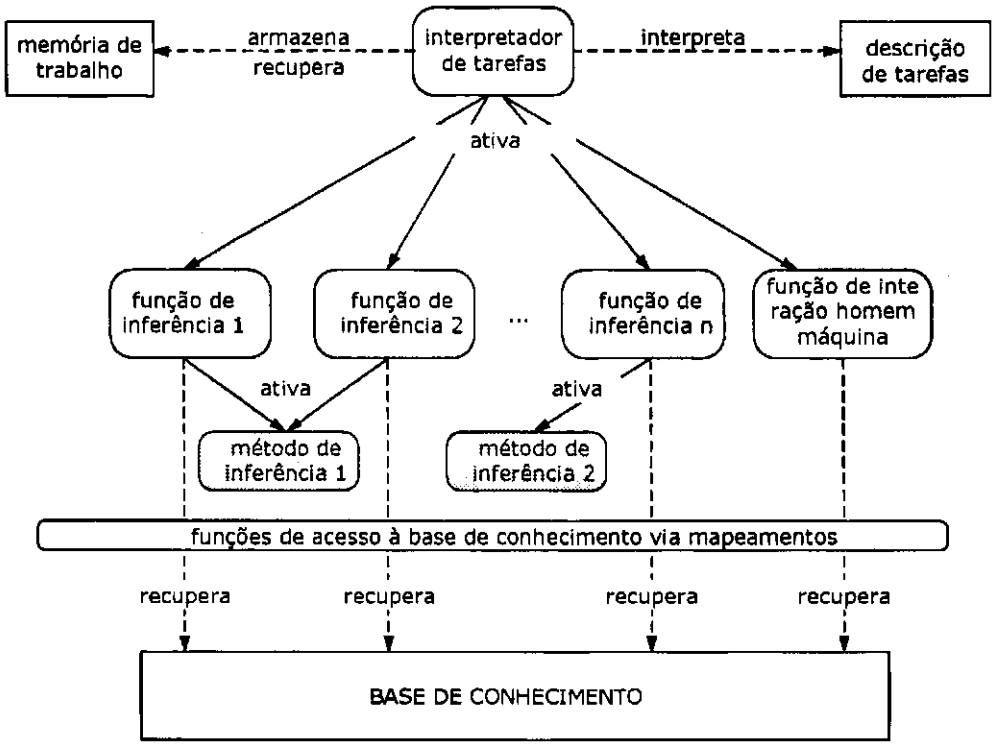
funções de Acesso realizam a tarefa de recuperação e armazenamento de itens da Base de Conhecimento.

Funções & Métodos de Inferência. Funções de inferência são as representações computacionais das inferências. Em termos de implementação, não executamos inferências mas funções de inferência. As funções de inferência contêm as mesmas informações que as inferências, tais como entrada/saída e visão de domínio. Além disso, são elas que ativam os métodos de inferência. Os métodos de inferência, por sua vez, formam uma biblioteca de elementos que podem ser reutilizados na execução de várias inferências.

Interpretador de Tarefas, Descrição de Tarefas & Memória de Trabalho. É a unidade de controle central dessa arquitetura. Usa descrições declarativas das tarefas e armazena (ou recupera) informações temporárias na (ou da) memória de trabalho.

2.10 Discussão

Neste capítulo revisamos alguns dos conceitos básicos de AC e mostramos várias metodologias de AC. É interessante notar que as metodologias iniciais não tinham pouca preocupação com o desenvolvimento da estrutura do domínio [Cha86, Cla85]. Este aspecto só começa a ser tratado a partir da metodologia Componentes de Perícia [Ste90], onde há o conceito de modelo de domínio, que é chamado de ontologia em outras metodologias. Hoje, construção de ontologias é um campo de pesquisas *por si* [FFR97, FdMdR98]. Outro ponto que gostaríamos de destacar é a dificuldade em escolher-se uma metodologia. Atualmente há uma variedade muito grande de metodologias de AC. Neste cenário, onde tantas são as opções para um engenheiro de conhecimento, surge a necessidade de se fazer um estudo comparativo entre as metodologias atuais para guiar uma possível escolha. Uma primeira tentativa nesse sentido foi a criação do grupo Sisyphus por parte da comunidade de AC, que busca realizar esse estudo comparativo [Lin93, vHSW97, BW98]. Tentativas de unificação estão sendo



Legenda: retângulos com cantos arrendodados representam componentes do sistema ativos durante o processo de resolução de problemas; os retangulos com cantos retos são repositórios de dados. Linhas não-tracejadas descrevem operação de controle (ativação); linhas tracejadas descrevem operação de dados (armazenagem/recuperação).

Figura 2.9: Arquitetura Genérica de um SBC proposto pelo KADS

desenvolvidas, como indicado por A. Aamodt [Aam94], entre elas o CommonKADS, desenvolvido na Europa por um grupo de Universidades e Centros de Pesquisas. Esta é uma das razões que nos levaram a optar por usar CommonKADS como metodologia de base para o processo de AC no MATHEMA. A outra razão foi principalmente o fato desta metodologia apresentar boas características nos estudos comparativos citados acima. Isso quer dizer que esta metodologia apresenta uma boa granularidade de seus construtores básicos e uma boa biblioteca de construtores básicos.

Capítulo 3

Arquitetura MATHEMA e o Problema da Manutenção

Resumo: O MATHEMA é um modelo de Sistema Tutor Inteligente (STI) com enfoque multi-agente. Associado a esse modelo o MATHEMA define uma técnica de organização e formalização do domínio sobre o qual vai ser construído o STI. Neste capítulo veremos uma descrição sucinta desse modelo e de sua técnica de organização do conhecimento. Abordamos também o problema da manutenção das bases de conhecimento dos Sistemas Tutores Inteligentes criados segundo o modelo MATHEMA.

Introdução

MATHEMA é um modelo conceitual para sistemas computacionais de ensino-aprendizagem com enfoque multi-agente [Cos97]. Desse modo, define um modelo de agente tutor, bem como os protocolos de interação entre os agentes tutores e os protocolos de interação entre os agentes tutores e um aprendiz humano. Apresentaremos aqui, um resumo do MATHEMA, seus conceitos principais, sua arquitetura geral e seus protocolos. Estaremos nos referindo aos sistemas criados a partir do modelo MATHEMA como Sistemas Tutores Inteligentes MATHEMA (STI-MATHEMA).

Nas seções seguintes mostramos a disciplina de organização de domínio sugerida

pelo MATHEMA (seção 3.1), a arquitetura geral MATHEMA (seção 3.3) e o modelo de Agente MATHEMA (seção 3.4). A seguir, apresentamos a dinâmica de interação entre os agentes MATHEMA (seção 3.5) e encerramos com a seção 3.6 onde apresentamos o problema da manutenção das bases de conhecimento em uma sociedade de agentes MATHEMA.

3.1 Organização de Domínio

MATHEMA propõe uma organização do domínio em duas etapas: construção da visão externa e construção da visão interna. Nas seções seguintes apresentaremos os detalhes da construção de cada uma destas etapas.

3.1.1 Visão Externa

Na primeira etapa, MATHEMA propõe que o domínio seja visto segundo as noções de *contextos*, *profundidades*, e *lateralidades*, mostradas abaixo:

- *contexto:* é um modo de ver o domínio, isto é, uma visão ou abordagem[1].
- *profundidade:* pode ser vista como nível de dificuldade. Em outras palavras, dada uma visão do domínio, procura-se estabelecer quais os graus de dificuldades com que podemos tratar os problemas.
- *lateralidades:* é um conjunto de conhecimento de suporte para um dado par <contexto, profundidade>[2].

A cada par <contexto, profundidade> será associado um sub-domínio. Na figura 3.1 mostramos a organização externa para um domínio representada em forma de uma árvore. Note que o par $<C_1,P_{11}>$ está associado ao subdomínio d_{11}. Como resultado,

[1]Em música, por exemplo, o domínio da harmonia pode ser tratado por um contexto Funcional ou contexto Tradicional [TCdSF95, CTF97].

[2]Um exemplo clássico de lateralidade é o conhecimento de *fatoração*, quando se considera o subdomínio de resolução de equações de segundo grau no conjunto dos reais pelo método de Bháskara.

ao final da organização da visão externa, o domínio estará "dividido" em diversos subdomínios.

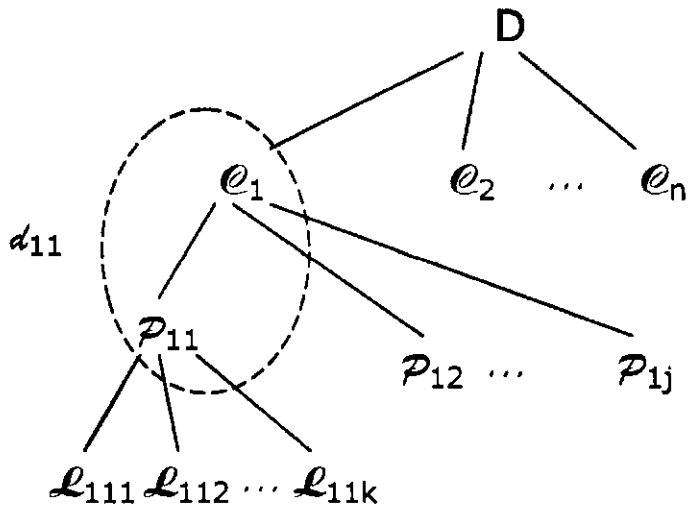


Figura 3.1: Organização do Domínio

3.1.2 Visão Interna do Domínio

A visão interna de um sub-domínio é composta por um conjunto de unidades pedagógicas que formam o *curriculum*, além de uma relação de ordem entre as unidades pedagógicas. Um exemplo de ordem pode ser a ordem de pré-requisitos. Uma *unidade pedagógica* é formada por um conjunto de *problemas* e por um conjunto de *unidades de conhecimento*. A seguir algumas definições.

Um *curriculum* é definido através das seguintes informações (fig. 3.2):

1. *sub-domínio*: descreve o sub-domínio ao qual o *curriculum* está associado;
2. *unidades pedagógicas*: conjunto de *unidades pedagógicas* que formam o curriculum;
3. *ordem pedagógica*: uma relação de ordem entre as unidades pedagógicas do *curriculum*. Um exemplo bem comum é a relação "pré-requisito-de" que, dadas duas unidades pedagógicas, determina qual delas é pré-requisito da outra.

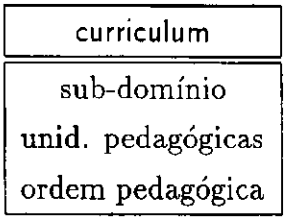


Figura 3.2: Estrutura de um *curriculum*

Uma *unidade pedagógica* é formada por (fig. 3.3):

1. *nome*: uma identificação para unidade pedagógica;
2. *problemas*: conjunto de problemas utilizados por esta unidade pedagógica;
3. *conhecimento*: unidades de conhecimento abordadas por esta unidade pedagógica.

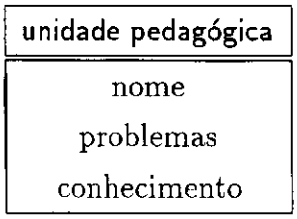


Figura 3.3: Unidade Pedagógica

Definimos um *problema* como uma estrutura composta de (fig. 3.4):

1. *enunciado*: um texto descrevendo o problema, em linguagem natural;
2. *dados*: valores iniciais utilizados no processo de resolução do problema;
3. *conhecimento*: ao se resolver um problema mostra-se que se domina uma certa quantidade de conhecimentos. Aqui são explicitadas quais são as *unidades do conhecimento* necessárias para resolução do problema.
4. *método*: os possíveis métodos utilizados para resolver este problema. Opcional quando se usar solução.

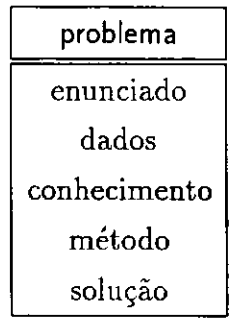


Figura 3.4: Estrutura de um problema

5. *solução*: alguns problemas podem apresentar um exemplo de solução. Opcional quando se usar método.

Os problemas levam à necessidade da definição de *métodos de resolução de problemas*. Podemos aproveitar esta análise para prever o conjunto das habilidades de resolução do agente.

Neste trabalho definimos *unidade de conhecimento* como os elementos utilizados como apoio na resolução de problemas, enumerados em [Cos97]. Estes elementos são: conceitos, resultados, exemplos (instâncias de outras unidades de conhecimento), contra-exemplos, dicas, erros e mal-entendidos.

3.2 Identificação dos agentes a partir da organização do domínio

A partir dessa organização do domínio, o próximo passo é criar uma identificação de agente para cada um dos subdomínios, isto é, para cada par <contexto, profundidade> e também para lateralidades. Como resultado teremos uma coleção de identificação de agentes que será utilizada no processo de criação da Sociedade de Agentes Tutores Artificiais (SATA).

Cada um dos agentes identificados na etapa de organização do domínio desempenhará as atividades tutoriais relativas ao sub-domínio específico ao qual foi associado. Os agentes são criados a partir da modelagem de conhecimento feita para cada subdomínio.

3.3 Arquitetura geral

Fazem parte do modelo MATHEMA, além da SATA, o *Ambiente de Manutenção*, o *Agente de Interface*, a *Fonte de Conhecimentos*, e o *Aprendiz Humano* (ver figura 3.5). Apresentamos a seguir uma descrição de cada um desses componentes.

SATA É a Sociedade de Agentes Tutores Artificiais criada a partir de uma organização do domínio (veja seção 3.2). Tem o objetivo de desenvolver, juntamente com o aprendiz, as seções de estudo, visando atender a demanda do aprendiz.

Agente de Interface. (AI) É o agente que interage diretamente com o aprendiz humano de modo que este não tenha que entrar em contato direto com a SATA. Este agente é que, logo na primeira sessão, ajuda o aprendiz a escolher seu agente supervisor. O agente supervisor é um agente tutor da SATA que vai atuar como "orientador" na interação entre a SATA e o aprendiz, isto é, vai ser encarregado de atender mais diretamente a um dado aluno. O termo Supervisor se refere ao fato de que este agente acompanha de perto as atividades do aluno. O Supervisor é um agente MATHEMA igual aos outros, mas com um "papel especial" em relação a um dado aluno.

Fonte de Conhecimento. (FC) É uma abstração que estamos fazendo para representar os responsáveis pela construção e manutenção da SATA. Em um primeiro momento esse módulo pode representar a equipe de projetistas do sistema. O importante é ressaltar que, por hipótese, essa fonte é responsável por não inserir conhecimentos contraditórios e pela escolha dos agentes da SATA devem sofrer manutenção.

Aprendiz Humano. (AH) É o aluno que participa de sessões de interação com o sistema afim de aprender sobre um determinado domínio.

Ambiente de Manutenção. (AM) Módulo responsável por auxiliar a Fonte de Conhecimento a realizar as operações de manutenção. Essas operações podem ser inclusão, retirada ou alteração de um agente do sistema. Para nós, o Ambiente de Manutenção é uma ferramenta de AC, pois ele pretende auxiliar o projetista

na criação e manutenção da base de conhecimento dos agentes [GM96]. De acordo com as funcionalidades desse ambiente[3] poderemos considerá-lo um ambiente integrado (ver seção 2.6).

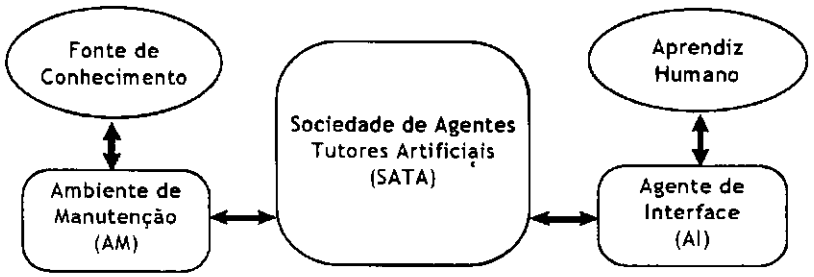


Figura 3.5: Arquitetura Geral do MATHEMA (adaptado de [Cos97])

3.4 Agente MATHEMA

Os agentes tutores artificiais que compõem a SATA são chamados agentes MATHEMA. A seguir apresentaremos o modelo do agente MATHEMA bem como sua arquitetura básica.

3.4.1 Modelo do Agente MATHEMA

Esse modelo é o conjunto de informações gerais sobre o agente que são instanciadas para um agente específico. O Modelo de Agente MATHEMA apresenta as seguintes informações:

- *Sistema Tutor*: Indica um componente especializado em um domínio.
- *Autoconhecimento*: Informações sobre o próprio agente tais como identificação e lista de habilidades do agente.

[3]ver capítulo 4

- *Conhecimento Social*: Lista de habilidades de alguns outros agentes da sociedade.
- *Mundo Externo*: Identificação do Ambiente de Manutenção e do Agente de Interface.

Na figura 3.6 sumarizamos o que foi dito sobre o modelo do agente MATHEMA.

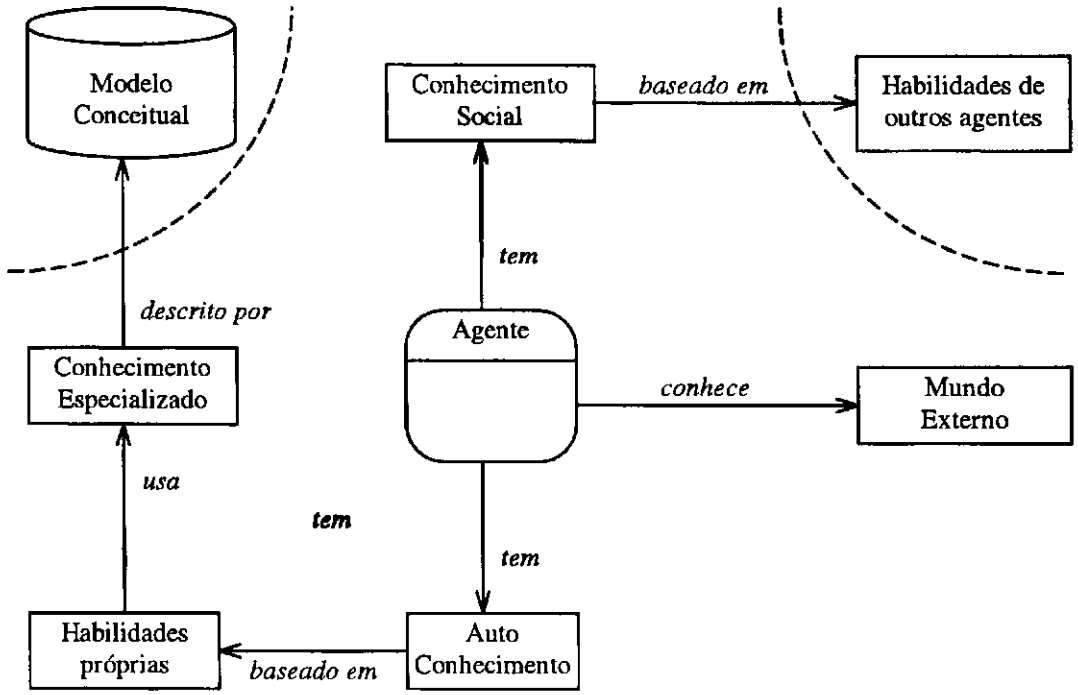


Figura 3.6: Modelo do Agente MATHEMA

3.4.2 Arquitetura do Agente MATHEMA

A arquitetura do agente é composta por três submódulos (ou camadas), a saber, o *Sistema Tutor*, o *Sistema Social*, e o *Sistema de Distribuição* [Cos97].

Na figura 3.7 é mostrada a arquitetura do agente MATHEMA. As setas indicam o fluxo de informações entre os módulos do agente.

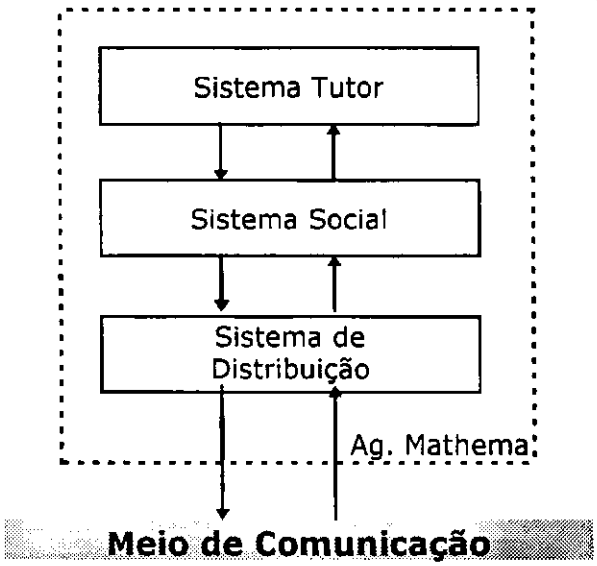


Figura 3.7: Arquitetura do Agente MATHEMA

Sistema Tutor. Módulo responsável pelas atividades tutoriais do sub-domínio ao qual o agente está associado. Desse modo, podemos ver o ST como um mini-STI para um sub-domínio específico.

Sistema Social. Módulo que possui os mecanismos de raciocínio necessário à cooperação entre os agentes. Neste módulo, estão as bases de conhecimento acerca das habilidades que os outros agentes possuem, bem como o conhecimento sobre si mesmo[4].

Sistema de Distribuição. Módulo que cuida do tráfego de mensagens enviadas e recebidas pelo agente [Jat98].

A ênfase maior nesta dissertação, é dada ao Sistema Tutor (ST). O ST é composto de *Raciocinadores*, *Base de Conhecimento*, além do *Mediador*. Cada um desses componentes tem uma função específica mostrada a seguir.

[4]Na realidade preferimos pensar que o agente tem somente uma crença sobre os outros agentes e não conhecimento. Assim, não há "garantias" de que o conhecimento de um agente em relação à outro seja correto.

Raciocinadores. É o componente que oferece as funções pedagógicas, isto é, resolução de problemas, instrução e diagnóstico. É composto pelos seguintes subcomponentes: Especialista (resolução de Problemas e Explicação), Pedagógico (Instrução) e Modelagem do Aprendiz (modelo do estudante).

Bases de Conhecimento. É formada pelas bases específicas para cada Raciocinador.

Mediador. Este subcomponente controla as ações tomadas pelo Sistema Tutor procurando atender as necessidades do aprendiz para que os objetivos do aprendiz sejam atingidos (ver seção 3.5).

A figura 3.8 mostra os módulos do componentes do sistema tutor.

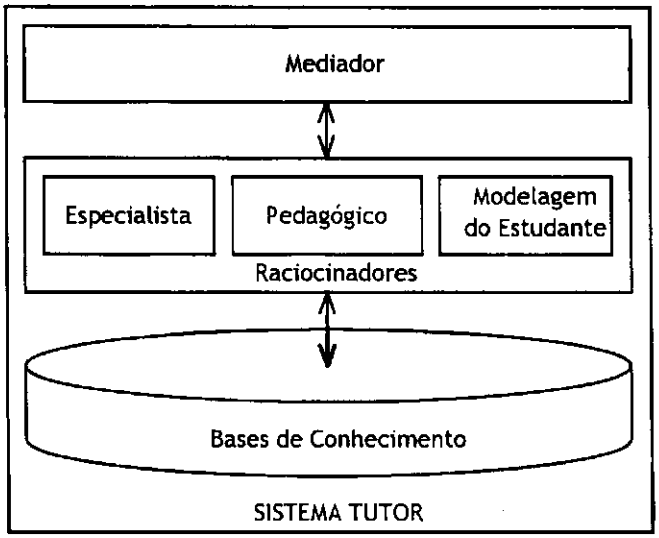


Figura 3.8: Sistema Tutor

Visão Geral. O Sistema Tutor é o subsistema no agente que vai responder as mensagens do Agente de Interface (vindas do Aprendiz). O primeiro módulo do ST a tratar a mensagem é o mediador, que determina qual é o tipo da atividade pedagógica requerida (instrução, resolução, explicação, etc.) e que aciona o Raciocinador. O Raciocinador

aciona o Interpretador e determina uma solução. Em uma situação de manutenção, o Ambiente de Manutenção vai trocar mensagens com o agente. O Sistema Social do agente vai tratar e responder às mensagens recebidas.

3.5 Dinâmica de Interação

Na primeira sessão de uso do sistema, o Aprendiz Humano, interagindo com o Agente de Interface, estabelece os objetivos pedagógicos a serem alcançados. A partir disso, o Agente de Interface ajuda o Aprendiz a eleger um agente da SATA para ser o agente supervisor "responsável" pelo aprendiz. O aprendiz e a SATA entrarão em sessões de interação com o objetivo de atingir os objetivos pedagógicos já estabelecidos. Caso necessário, um agente MATHEMA pode solicitar ajuda dos outros para que uma tarefa seja atendida.

Cada agente da SATA sabe quais são suas habilidades e tem uma "idéia" das habilidades dos outros agentes. Assim, caso uma tarefa não seja do conhecimento de um agente, este usará seu *conhecimento social* para solicitar ajuda. Se ele souber a quem pedir ajuda ele entrará em processo de cooperação. Caso contrário fará uma *licitação* [Cos97]. Vejamos mais detalhadamente cada um desses conceitos.

Cooperação. Um agente pode solicitar a ajuda de um outro agente para realização de uma determinada tarefa. O agente solicitado vai atuar em um papel de submissão e o solicitante como mestre[5]. Em um caso de cooperação, diz-se que o agente tem *ignorância parcial* sobre o assunto da cooperação, pois ele não sabe, mas sabe quem sabe.

Licitação. Quando o agente não sabe realizar a tarefa e não conhece quem pode ajudá-lo, diz-se que o agente está em *ignorância total* sobre o assunto. Nesse caso, o agente realiza uma licitação, isto é, anuncia a tarefa na sociedade pedindo ajuda a algum agente que souber resolvê-la. O resultado da licitação é uma lista de candidatos

[5]Esses papéis estão definidos no modelo original MATHEMA [Cos97] e refletem o fato de os agentes MATHEMA possuírem a característica de benevolência, isto é, nunca se recusam a cooperar.

à resolução da tarefa. O agente solicitante escolhe um agente da lista e entra em cooperação com ele.

O Modelo de Cooperação. O modelo de cooperação de um Sistema Multi-Agente descreve as possíveis interações entre os agentes. Geralmente, esse modelo deve ser construído para cada domínio indicando como os agentes podem se organizar em cada situação [Lab95].

No MATHEMA, o modelo de cooperação não é explícito e está embutido na arquitetura do agente, na forma de *conhecimento social.* Assim, todos os agentes da SATA têm a possibilidade de solicitar cooperação com qualquer outro, desde que o solicitante saiba a que agente se dirigir. Podemos dizer então que o modelo de cooperação do MATHEMA é independente do domínio. Desse modo, não temos que construir um modelo de cooperação para modelar a dinâmica de interação dos agentes para o domínio em estudo.

3.6 O problema da Manutenção das Bases de Conhecimento

A SATA é uma sociedade aberta e dinâmica. Isso significa que os agentes da sociedade podem ser incluídos, retirados ou modificados pelos projetistas (no papel de Fonte de Conhecimento). No entanto, no MATHEMA há um forte comprometimento entre a estruturação dos subdomínios e a sociedade de agentes, exigindo que as alterações feitas não fujam ao escopo do domínio escolhido. Esse comprometimento evita também que um novo agente entre na sociedade sem necessidade. Esse comprometimento é tarefa da Fonte de Conhecimento. Aqui apresentaremos a noção de *falta de conhecimento*, que é uma situação em que a SATA não consegue acompanhar o aprendiz. Definiremos essa situação e apresentaremos sua forma de percepção e o procedimento de correção correspondente. Apresentaremos também os problemas relacionados a esta operação no MATHEMA.

Falta de Conhecimento na SATA. Essa situação reflete a incapacidade da SATA em desenvolver uma determinada atividade com o aprendiz. Isto ocorre porque nenhum dos agentes da SATA tem o conhecimento necessário para desenvolver a tarefa em questão. Como os agentes são definidos no momento da partição do domínio (veja seção 3.1), pode parecer estranho que haja tal situação. Não é irreal, entretanto, supormos que erros de modelagem possam acontecer.

Como estamos falando de ambiente multi-agentes, pode ocorrer ainda a falha na comunicação entre os agentes, o que não caracteriza uma falta de conhecimento. A Falta por Falha de Comunicação ou falhas semelhantes estão fora do escopo deste trabalho.

Em resumo, o que nos interessa neste trabalho é como tratar a falta depois que ela ocorreu e não como ela ocorreu.

Percepção da Falta de Conhecimento. Ao aprendiz que esteja utilizando um STI MATHEMA, a Falta de Conhecimento será notificada imediatamente, ou seja, o agente supervisor informará que nenhum agente da sociedade está apto a atender a tarefa dada. No momento em que a Falta ocorrer, esta será notificada em um "histórico" criado pelo agente que acusou a Falta. Deve-se frisar que a Falta é percebida somente depois que uma tarefa não pode ser atendida, o que pode ser constatado por uma licitação na sociedade.

Procedimento de Recuperação. Quando ocorre uma Falta de Conhecimento, deve-se fazer uma operação de manutenção na SATA. Isso significa que a Fonte de Conhecimento deve seguir alguns passos para que a falta seja suprida. De modo informal podemos enumerar os seguintes passos:

- analisar o histórico de interações para indicar qual agente deve sofrer alteração;
- indicar que tipo de alteração o agente identificado deve sofrer;
- verificar se a modificação não altera a estrutura de subdomínios original;
- usar o Ambiente de Manutenção para executar as modificações no agente.

Esta lista está obviamente incompleta e é especificada de modo superficial. O nosso objetivo, neste momento, é dar apenas uma idéia das responsabilidades que tem o Ambiente de Manutenção e a Fonte de Conhecimento. O Ambiente de Manutenção só é utilizado depois que uma série de decisões críticas foram tomadas pelos agentes humanos no papel de Fonte de Conhecimento e tem uma série de funcionalidades para auxílio na manutenção: verificação de coerência nas modificações (local a um agente), verificação sintática, editores para a linguagem do conhecimento utilizada, biblioteca de modelos, etc.

A manutenção é uma atividade que faz parte do ciclo de AC no MATHEMA. Esta atividade pode ser feita sempre que a FC decidir que a SATA deve ser alterada. Isso se dá, por exemplo, em caso de mudanças no domínio sobre o qual o STI foi construído ou sempre que ocorrer uma Falta de Conhecimento. O objetivo ao se efetuar uma modificação é fazer com que a sociedade de agentes evolua de um estado faltoso/desatualizado para um estado sem faltas/atual.

No sentido de permitir que tais operações de manutenção sejam realizadas, nos deparamos com uma série de questões. Uma primeira questão é quanto à arquitetura do Agente. Para que seja possível realizar modificações nos agentes, sua arquitetura tem que ser especificada de tal modo que o agente possa lidar com situações de modificação de sua base de conhecimento.

Outra questão refere-se ao protocolo de manutenção. Dissemos que um agente pode ser modificado, mas para isso um protocolo deve especificar exatamente *quando* e *como* isso deve ser feito. Para a operação de manutenção já foram especificados [Cos97] os protocolos de entrada, saída e manutenção de uma agente da SATA. Entretanto, estes protocolos só determinam as mensagens trocadas entre um agente que vai sair (ou entrar) e o resto da SATA; logo, devem ser detalhados para tratarem da interação entre o Agente a ser modificado e o Ambiente de Manutenção.

3.7 Discussão

A disciplina de organização de domínio, no MATHEMA, é o primeiro passo para a construção de um STI-MATHEMA. A organização consiste em definir os subdomínios e

suas lateralidades, e associar um agente tutor a cada subdomínio (e a cada lateralidade). A seguir estes subdomínios devem passar por um processo de elicitação e formalização. Isso pode ser feito com o auxílio do ambiente de manutenção.

A elicitação não faz parte do escopo desse trabalho. Em linhas gerais a elicitação é a etapa onde os conhecimentos a serem formalizados são primeiramente delineados. Aqui estamos assumindo que a etapa de elicitação já está concluída e podemos prosseguir com a formalização[6].

Neste trabalho, formalizar o conhecimento significa descrevê-lo segundo uma linguagem de descrição do conhecimento provida por uma metodologia. Cada metodologia de AC propõe uma linguagem[7] com características particulares. Portanto, especificaremos o ambiente de manutenção para dar suporte a uma linguagem de descrição de conhecimento[8], visando facilidade de uso para a Fonte do Conhecimento e poder de expressão do formalismo. Preocuparemos-nos ainda com aspectos tais como biblioteca de blocos básicos, e possibilidade de execução das bases construídas.

Um ponto importante a considerar é a relação entre a estrutura do agente e a arquitetura do ambiente de manutenção. A idéia central é que o ambiente de manutenção permita a construção de bases de conhecimento dos agentes. Assim, se os (raciocinadores dos) agentes forem construídos como interpretadores de regras, o ambiente de manutenção tem que prover o auxílio necessário à criação e manutenção de regras. Neste trabalho optamos por desenvolver o módulo especialista do agente como um interpretador de tarefas baseado em modelo de conhecimento, segundo a arquitetura mostrada na seção 2.9.2. Neste caso o ambiente de manutenção tem que prover auxílio à criação de modelos de conhecimento. Com essa abordagem surge a possibilidade de reusabilidade, introduzida pelo uso de bibliotecas de ontologias e modelos genéricos de conhecimento.

---

[6]Mais sobre elicitação pode ser encontrado em [Die90] ou ainda [DL89, AGKS92, Kin94].

[7]Algumas metodologias chegam a oferecer várias linguagens, variando desde informal até formal.

[8]Neste trabalho escolhemos a CML, que é a linguagem de descrição do modelo de perícia KADS. Veja seção 2.8.

Capítulo 4

Uma Ferramenta de Aquisição de Conhecimento para o MATHEMA

Resumo: MATHEMA é um modelo para Sistemas Tutores Inteligentes com enfoque Multi-Agente. Delineamos neste capítulo uma ferramenta de AC para auxiliar na construção destes STI-MATHEMA. Apresentamos aqui os conceitos básicos que dão suporte a uma tal ferramenta, a sua especificação e os protocolos de interação dessa ferramenta com os agentes da SATA.

Introdução

Como visto no capítulo anterior, uma das etapas da construção de um STI-MATHEMA é a construção dos modelos de conhecimentos dos subdomínios. Para tanto, deve-se escolher uma metodologia de AC, que vai indicar como realizar a elicitação e a formalização do conhecimento. Nesta etapa, é de interesse do projetista que a metodologia seja suportada por uma ferramenta automatizada. O nosso objetivo neste capítulo é especificar uma ferramenta para auxiliar na criação do modelo de conhecimento dos agentes, bem como a criação destes agentes. Essa ferramenta no modelo MATHEMA é chamada Ambiente de Manutenção (AM).

Começamos este capítulo apresentando o ciclo de construção de STI-MATHEMA (seção 4.1). Em seguida, revisamos a arquitetura do Agente MATHEMA (seção 4.2)

e apresentamos então a arquitetura do ambiente de manutenção (seção 4.3). Na seção seguinte, mostramos os protocolos de interação entre a SATA e o ambiente de manutenção (seção 4.5) e encerramos com a especificação das mensagens trocadas entre o ambiente de manutenção e um agente em uma seção de manutenção (seção 4.4).

4.1 Ciclo de Aquisição de Conhecimento no MATHEMA

Apresentamos a seguir as etapas do ciclo de AC no MATHEMA.

Organização Externa. Esta etapa corresponde à criação da visão externa do domínio (seção 3.1). A visão externa permite destacar os subdomínios de conhecimento que serão utilizados para se construir um STI-MATHEMA. Cada subdomínio e cada lateralidade dará origem a um agente. Nesta etapa criamos apenas as identificações deste agentes. Estas identificações serão usadas na etapa de criação dos agentes.

Estrutura Pedagógica. Esta etapa corresponde à definição da visão interna de cada sub-domínio (seção 3.1). Nesta etapa construímos os conhecimentos pedagógicos do STI (curriculum, unidades pedagógicas, etc.).

Modelagem Conceitual do Conhecimento dos Subdomínios. Esta etapa tem como objetivo a modelagem do conhecimento especializado do agente. Este conhecimento é utilizado pelo agente nas atividades de resolução de problemas do sub-domínio a que o agente está associado. Como já foi dito, utilizamos a metodologia KADS para realização desta etapa.

Geração da Base de Conhecimento. Aqui definimos base de conhecimento como uma representação em *nível simbólico*[1] de um dado domínio. Gerar essa representação

[1]Oposto a nível de conhecimento.

em nível simbólico, isto é, em uma linguagem de representação do conhecimento, significa implementar o módulo de resolução de problemas do agente (o módulo ESPECIALISTA). Neste trabalho, utilizamos a abordagem de implementação da base conhecimentos sugerida pelo KADS (veja a seção 2.7).

Criação dos Agentes. Finalmente, a ultima etapa é a definição e implementação dos agentes na linguagem que pareça mais adequada ao projetista do STI. Para cada sub-domínio será criado um agente. Suas habilidades devem ser criadas de acordo com os métodos de resolução de problemas identificados na construção da visão interna do subdomínio a que ele está associado.

Manutenção Manutenção é uma etapa presente em qualquer ciclo de vida de sistemas. No caso do MATHEMA, esta etapa consiste em criar novos agentes, retirar da SATA agentes existentes, ou realizar a atualização dos conhecimentos dos agentes.

O ciclo de AC no MATHEMA pode ser visto de forma esquemática na figura 4.1. O ideal seria uma ferramenta que auxiliasse o projetista em todas as etapas do ciclo. Por simplificação decidimos especificar uma ferramenta para auxiliar apenas nas etapas de modelagem do conhecimento, criação da base de conhecimento e criação/manutenção dos agentes[2].

4.2 O Agente MATHEMA Revisado

Conforme visto na seção 3.4, o agente MATHEMA é composto pelos seguintes subsistemas: Sistema Tutor, Sistema Social e Sistema de Distribuição. Vimos ainda que o ST é formado por um componente chamado Mediador, um componente Raciocinadores e suas Bases de Conhecimento.

A partir da arquitetura original, mostrada na figura 3.8, detalhamos o Sistema Tutor do Agente MATHEMA de modo a acomodar os tipos básicos de conhecimento, mostrados na seção 2.3.

[2]Sugere-se, para o futuro, estender a especificação da ferramenta para cobrir as etapas iniciais do ciclo.

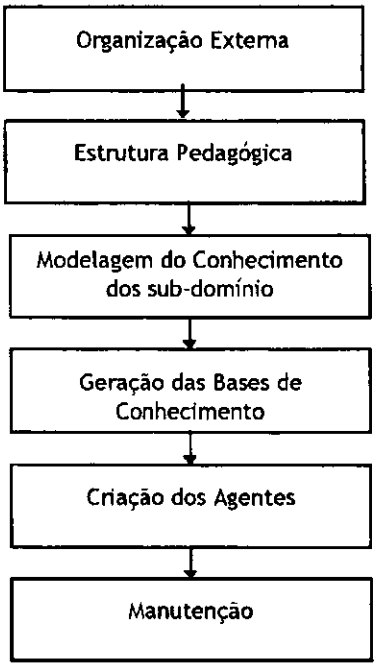


Figura 4.1: Resumo do Ciclo de AC no MATHEMA

Desse modo, definimos a BASE DO ESPECIALISTA formada por três partes:

- *ontologia:* para descrever os termos e relações do domínio. No MATHEMA vai descrever além disso, alguns elementos pedagógicos, tais como os contra-exemplos e os resultados.
- *instâncias de ontologia:* destinado a receber o conhecimento particular do domínio.
- *conhecimento procedural:* tarefas[3] e inferências.

O módulo ESPECIALISTA é um interpretador para a base de conhecimento definida acima. A estrutura deste interpretador está relacionado com o formalismo de representação do conhecimento da base. Por exemplo, a representação interna poderia ser "frames", o padrão KIF, ou ainda cláusulas Prolog. O que importa é que este módulo

[3]Na realidade são instâncias de tarefas, nos termos da seção 2.3.

seja capaz de "executar" a base de conhecimento e derivar uma resposta adequada a uma solicitação. Neste trabalho, definimos o ESPECIALISTA como um interpretador de tarefas, adaptado do interpretador mostrado na seção 2.9.2.

O módulo PEDAGÓGICO não foi estudado neste trabalho. Entretanto, definimos a sua base. A base do PEDAGÓGICO conterá os conhecimentos modelados durante a estruturação da visão interna do subdomínio. Assim, a base do módulo PEDAGÓGICO é constituída de: *curriculum*, *unidades pedagógicas* e *problemas*. As unidades de conhecimentos são definidas na ontologia e armazenados na base do especialista, uma vez que este conhecimento é também necessário para a resolução de problemas[4].

Mediador. Tem a capacidade de decidir sobre o tipo de interação que deve acontecer entre o agente e o Aprendiz. Para isso, ele utiliza conhecimentos sobre o modelo do aprendiz e conhecimentos pedagógicos.

Na figura 4.2 apresentamos a arquitetura detalhada do agente, de acordo com as idéias desta seção.

4.3 Ambiente de Manutenção

O Ambiente de Manutenção (AM) é a ferramenta que dá suporte ao ciclo de aquisição de conhecimento no modelo MATHEMA [CdS99]. Aqui vamos definir a sua arquitetura e os protocolos seguidos durante as seções de manutenção da SATA.

4.3.1 Requisitos para o AM

As ferramentas de AC, de modo geral, devem apresentar as seguintes funcionalidades, conforme Swartout e Gil [SG95]:

- permitir que o usuário modifique conhecimento factual de domínio e de resolução;
- permitir uma AC adequada ao modelo de resolução de problema atual;

[4]Na realidade essa divisão é influenciada pelo fato de não estarmos trabalhando, aqui, com o módulo PEDAGÓGICO. Em termos mais globais, o mais natural é considerar as unidades de conhecimento como uma base separada, compartilhada pelos módulos PEDAGÓGICO e PELO ESPECIALISTA.

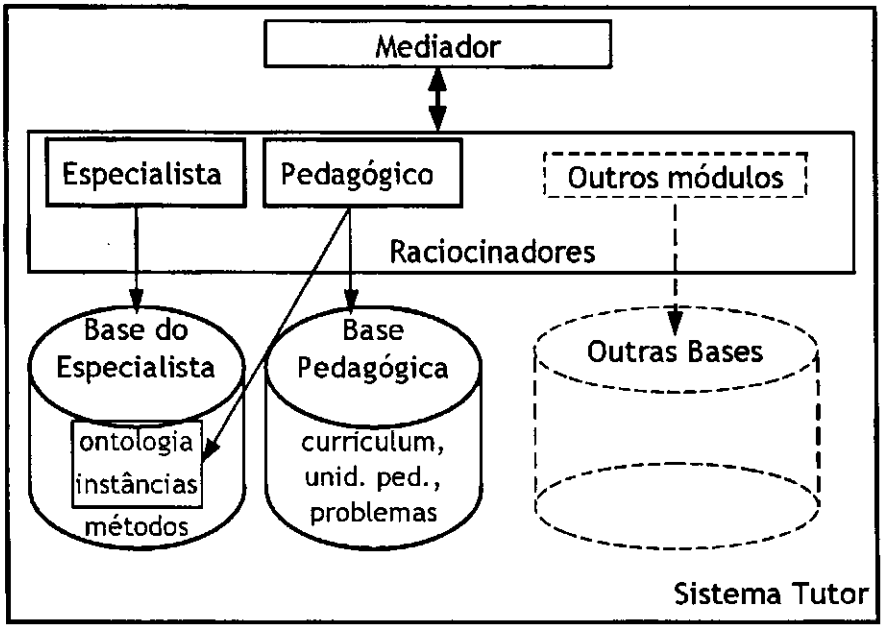


Figura 4.2: Detalhes do agente MATHEMA para a AC

- permitir uma verificação semântica ao invés de simplesmente sintática: checagem de tipo, checagem de elementos não declarados, etc.

Definir uma ferramenta com estas características exige um estudo aprofundado das implicações existentes na manutenção de bases de conhecimento. Modificar bases de conhecimento não é uma atividade trivial, como apontam Gil e Tallis [GT97]. As principais dificuldades são:

- *o conhecimento é armazenado em pedaços separados*: um dos princípios da AC dirigida por modelos é a tipagem do conhecimento. Assim, todos os conhecimentos da base estão separados e classificados por tipo.

- *deve-se manter a compatibilidade de tipos entre métodos e conhecimento de domínio*: cada tarefa tem uma especificação de entrada e saída. Quando estamos fazendo uma manutenção, devemos estar atentos para não introduzirmos um erro de mistura de tipos ao tentar atribuir um tipo incompatível à uma entrada ou saída.

- *dificuldades em se propagar mudanças*: os conhecimentos da base estão fisicamente separados mas estão interligados logicamente. Portanto, quando se for fazer uma modificação, deve-se checar todas as alterações indiretas decorrente das interligações das partes.

Mais especificamente, de acordo com o ciclo de AC apresentado na seção 4.1, a ferramenta de manutenção para o MATHEMA deve ter as seguintes características:

- *capacidade de edição de modelos de conhecimento:* para suportar a etapa de modelagem de conhecimento.
- *capacidade de geração de bases de conhecimento segundo a representação interna de conhecimento dos agentes:* para dar suporte à etapa de criação das bases de conhecimento.
- *capacidade de gerenciamento da SATA:* Para dar suporte às etapas de criação e manutenção dos agentes da SATA. Para isso, deve manter algumas informações sobre os agentes, tais como: dicionário de nomes dos agentes da SATA, relação dos agentes que foram retirados SATA (ou que ainda não entraram) e relação das bases criadas mas ainda não utilizadas para na criação de um agente.

4.3.2 Arquitetura do AM

Para atender os requisitos estabelecidos na seção anterior, o AM deve ser, no mínimo, composto de um Editor de Ontologias, de um Editor de Conhecimento Procedural, de um Editor de Instâncias de Ontologia, de um Verificador de Coerência da Base, de uma Biblioteca de Métodos de Resolução de Problemas, de um Módulo de Comunicação e de um Gerenciador de Agentes da SATA. Na figura 4.3 apresentamos a arquitetura proposta para o AM neste trabalho.

Esta proposta foi influenciada principalmente pelas seguintes ferramentas de AC baseada em modelos (ver seção 2.6):

- PROTÉGÉ II: projetada para permitir a construção de modelos de resolução de problemas e ontologias. Inspiramo-nos na idéia geral da ferramenta.

- EXPECT: faz checagens de várias naturezas na base de conhecimento. O módulo verificador proposto para o AM foi inspirado principalmente neste trabalho.
- CoKACE: permite edição de modelos de perícia do KADS. Nos inspiramos particularmente na sua verificação dos modelos construídos.

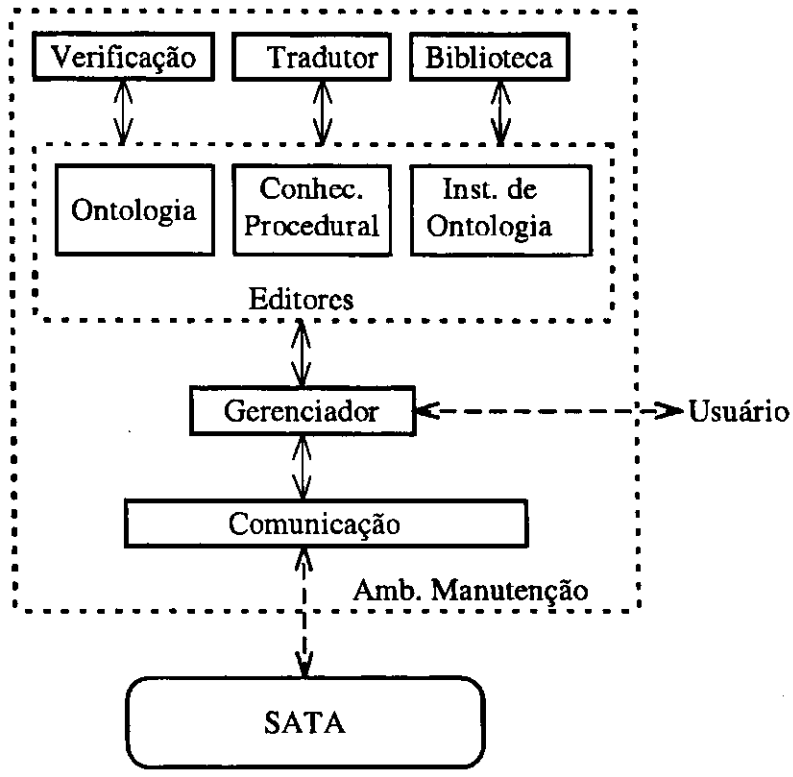


Figura 4.3: Arquitetura do Ambiente de Manutenção

Editores

Editor de Ontologia. Permite a criação, e alteração de ontologias. Esse editor é responsável pela checagem sintática das ontologias criadas.

Editor de Conhecimento Procedural. É a ferramenta que permite a modificação do conhecimento de resolução de problema do agente. Esta ferramenta é usada para criar, alterar e fazer checagem sintática das tarefas e inferências.

Editor de Instancias de Ontologia. Auxilia na instanciação da ontologia. Permite também a navegação pela base de instâncias de conhecimento, isto é, esta ferramenta serve também como um visualizador das instâncias da ontologia.

Biblioteca

Serve de auxilio à construção de novas Ontologias e de novos Métodos de Resolução de Problemas (MRP) adaptados ao domínio buscando facilitar a reusabilidade. A biblioteca pode ser vista como uma coleção de Ontologias e MRPs. Ela deve permitir a recuperação de elementos armazenados e também a inclusão de novos elementos. Além de ontologias e MRPs, outros elementos fazem parte de nossa biblioteca:

- *função genérica:* é uma descrição de um elemento de fluxo de dados. Sua descrição inclui um nome, papéis de entrada e saída, descrição/objetivo, suposições e referência à tipologia. Uma função genérica pode dar origem a uma inferência ou a uma tarefa no modelo conceitual final. Isso só depende da decisão do engenheiro em expandir ou não a função genérica. Uma tal expansão é definida por um método de expansão, definido mais adiante.

- *estrutura de funções:* é um conjunto de funções genéricas relacionadas entre si pelos papéis de entrada e saída.

- *método de expansão:* relação entre funções genéricas e estruturas de funções indicando como uma determinada função genérica pode ser refinada.

Em resumo, a biblioteca é composta de ontologias, MRPs, funções genéricas, estruturas de funções e métodos de expansão. A biblioteca assim definida é inspirada nos resultados obtidos na construção da Biblioteca do CommonKADS [ABB+92], onde é apresentada a linguagem completa para a construção de bibliotecas segundo o CommonKADS.

Cada elemento da biblioteca deve ser organizado por uma tipologia para cada categoria. Isso ajudará a indexar os elementos na biblioteca, e conseqüentemente ajudará na sua recuperação [VL93]. É ainda interessante salientar que cada elemento tem associado a si um conjunto de restrições o que pode guiar o processo de criação de um

modelo conceitual, pois a escolha de um elemento pode impor ou excluir a escolha de um outro. Em cada sessão de utilização da *biblioteca* será guardado um caminho de decisões de construção para que se saiba as necessidade e restrições atuais.

Ferramenta de Verificação

Permite que se detectem incoerências no modelo de conhecimento. Um exemplo de erro que deve ser checado é a ausência de definição de conceitos referenciados em um método.

O modelo conceitual, por seu caráter formal (ou semi-formal), é de grande auxilio para a verificação das bases de conhecimento. Isso porque em um modelo de conhecimento a ontologia e as tarefas são definidas explicitamente, permitindo que se detecte um grande número de incoerências. Mais sobre verificação de modelos de conhecimento pode ser encontrado nos trabalhos de Geldof e Slodzian [GS94] ou ainda nos trabalhos de Breuker e Boer [BB98].

Neste trabalho consideramos a seguinte estrutura para o verificador:

- *verificador de métodos:* verifica a definição dos métodos (multi-definição, redundância, etc.) e papéis de conhecimento que eles usam. Um exemplo de erro detectável por esse módulo é: "método não definido".

- *verificador de ontologia:* verifica se existem problemas de definição na ontologia, tais como erros de hierarquia de conceitos, falta de definição de elementos referenciados por outros elementos da ontologia, etc.

- *verificador de domínio:* verifica as instâncias do conhecimento de domínio em relação à ontologia. Dependendo do formalismo de representação do conhecimento do domínio, técnicas de verificação particulares podem ser utilizadas. Regras de Produção, por exemplo, podem ser checadas quanto a redundância, circularidade, conflito, etc.

- *verificador cruzado:* realiza a verificação de incoerências que surgem somente quando consideramos todos as partes do modelo de conhecimento juntas. Es-

te módulo deve, por exemplo, detectar se um método utiliza um elemento da ontologia que não foi definido.

Outros Módulos

Tradutor. A função do tradutor é converter o modelo de conhecimento, descrito em nível de conhecimento, para um formato de nível de símbolos (regras de produção, por exemplo), de modo a se obter uma base de conhecimento pronta para ser utilizada pelo agente MATHEMA.

Gerenciador de Agentes. Realiza as operações de criação, alteração e retirada de agentes da SATA. Ele se comunica com os agentes da SATA através do módulo de comunicação e comunica-se diretamente com os editores. As suas operações serão definidas na seção 4.3.3.

Módulo de Comunicação. É responsável pela troca de mensagens entre o Ambiente de Manutenção e os agentes da SATA. Para gerência a iteração entre a SATA e o AM, este módulo utiliza os protocolos de manutenção (seção 4.5).

Funcionamento Geral do Ambiente de Manutenção

Inicialmente, o AM entra em contato com o agente que vai sofrer manutenção para que ele saia da SATA e forneça uma cópia de sua base de conhecimento. Só então pode-se proceder a alteração. Utilizando-se os editores o usuário pode modificar a base de conhecimento do agente e quando estiver concluído o AM informa ao agente sobre as modificações efetuadas em sua base. Após a confirmação das modificações, o agente pode retornar à SATA.

Os editores permitem que se modifique a base de conhecimento de várias formas. Por exemplo, podemos alterar a definição de um método. Estas alterações podem levar à inconsistência do modelo de conhecimento. Por exemplo, pode-se referenciar um papel de conhecimento não definido. Neste caso o verificador deve indicar ao usuário qual o problema que está tornando o modelo inconsistente[5].

[5]Será interessante que o verificador ajude o usuário a corrigir as inconsistências detectadas, su-

4.3.3 Operações de Manutenção

As operações possíveis no processo de manutenção são divididas em três grupos:

- *operações sobre bases de conhecimento :* criam ou modificam uma base de conhecimento.
- *operação sobre agentes :* criação de novos agentes.
- *operações dos agentes:* operações realizadas pelos agentes internamente em resposta a mensagens de manutenção.

A operação sobre base de conhecimento:

criar-base(B). É uma abstração para a criação de uma base através dos editores do AM.

atualizar(B, B'). Esta operação é a abstração da edição de bases de conhecimento realizada através dos editores do AM. Definimos esta operação como uma modificação da base B para base B'.

Operações sobre agentes:

criar(ag-id, B). Cria um novo agente com a identificação ag-id e com a base B. Se já existe um agente com esta identificação a operação falha. No ato de criação do agente é embutida os endereços de todos os agentes atualmente na SATA. Esta informação inicial possibilita que o agente entre em contato com a SATA[6].

Operações dos agentes:

entrar(ag-id). Esta operação faz com que o agente ag-id inicie um protocolo de entrada na sociedade (veja seção 4.5), isto é, ele apresenta-se aos outros agentes da sociedade. O resultado desta operação é o agente ag-id ser conhecido pelos outros agentes, situação na qual o agente é considerado como parte da sociedade.

gerindo quais as operações necessárias para corrigir o problema. Isso já foi feito por Gil e Tallis no sistema EXPECT [GT97]

[6]Estes endereços podem ser modificados depois quando o agente começar a interagir com a SATA, no caso de entradas ou saída de novos agentes.

sair(ag-id). Esta operação faz com que o agente **ag-id** inicie um protocolo de saída da sociedade (veja seção 4.5). Assim os agentes da sociedade já não contarão com o agente **ag-id** para atividades de cooperação, situação na qual o agente é considerado fora da sociedade[7].

copiar-base(ag-id, B). Esta operação faz com que o agente **ag-id** prepare uma cópia sua base. B é uma variável que receberá a cópia da base.

substituir-base(ag-id, B). Essa operação faz com que o agente assuma a base nova que lhe é passada como parâmetro. A base atual pode ser guardada pelo AM para fins de documentação.

4.4 Mensagens de Manutenção

Nesta seção, descreveremos as mensagens trocadas entre o ambiente de manutenção (AM), fonte de conhecimento (FC) e a SATA[8].

As mensagens aqui definidas referem-se às *operações de manutenção* definidas na seção 4.3.3. Os agentes podem manter diversas conversações simultâneas, com vários outros agentes, assim, as mensagens têm uma identificação da conversação **conv-id** que é utilizada para o gerenciamento correto das conversações.

comando(conv-id, emiss, recep, operação). Utilizada quando emissor **emiss** requer que o receptor **recep** execute a **operação** indicada.

solicitação(conv-id, emiss, recep, operação). Utilizada quando o emissor **emiss** deseja que uma determinada **operação** seja efetuada pelo receptor **recep**. Do modo como definimos uma *solicitação* tem uma semântica mais fraca que um *comando*, pois um comando não pode ser recusado enquanto que uma solicitação tem um caráter de negociação, isto é, pode ser recusada.

[7]Um agente que esteja fora da SATA não responderá a nenhuma mensagem que não venha do AM.

[8]As mensagens apresentadas aqui, podem ser reescritas como uma extensão às mensagens definidas no KQML (uma vez que a definição do KQML é aberta à extensões).

confirmação(conv-id, emiss, recep, operação). O emissor emiss confirma o sucesso da execução de uma dada operação.

resposta(conv-id, emiss, recep, operação, resp). Resposta do emissor emiss a uma solicitação anteriormente feita pelo receptor recep.

falha(conv-id, emiss, recep, operação). Mensagem enviada pelo emissor emiss indicando que a operação não foi executada com sucesso.

informação(conv-id, emiss, recep, conteudo). É utilizada para passar informações de propósito geral entre o emissor emiss e recep. Aqui conteudo pode ser qualquer informação útil para o receptor, e que, de acordo com o estado da conversação, já está sendo esperada.

4.5 Protocolos de Manutenção

Na seção 4.3.3 apresentamos as operações de manutenção e na seção 4.4 apresentamos os tipos de mensagens trocadas entre os agentes da SATA, ambiente de manutenção (AM) e fonte de conhecimento (FC). Mostraremos aqui os protocolos que guiam as seções de manutenção, isto é, as regras de conversação entre a FC, o AM e um agente que esteja em processo de manutenção.

Alguns protocolos já foram definidos por Costa [Cos97]. São eles:

Entrada. Para que um agente entre na SATA, ele deve enviar uma mensagem de *apresentação*[9] para todos os agentes da SATA. Isso faz com que os agentes da SATA também se apresentem para o novo agente. Após o ciclo de apresentações o novo agente é considerado um membro da SATA.

Saída. O agente que sai da SATA deve enviar uma mensagem de *aviso de saida* para todos os agentes da SATA, para que os outros agentes não mais o requisitem. Ele será considerado fora da sociedade quando os agentes não possuírem mais o seu endereço.

[9]As mensagems de "apresentação", "aviso de saida" foram definidas no trabalho original [Cos97], por isso não as apresentamos na seção 4.4 as mensagens.

Manutenção. Foi apenas esboçado da seguinte maneira: o agente faz uma saída seguida de uma entrada. O novo protocolo chamado Atualização por Edição pretende acrescentar um nível de detalhamento maior a este protocolo.

Os novos protocolos, definidos neste trabalho, são mostrados a seguir:

Protocolo de Criação. Utilizado para a criação de novos agentes e é descrito por:

1. FC cria nova base através da operação criar-base(B).
2. FC envia comando para AM para a criação de um agente que utilize a base B.
3. AM cria novo agente ag-id com a base B.
4. AM envia solicitação ao agente ag-id para que ele entre na SATA.
5. agente ag-id entra na SATA via protocolo de entrada.
6. ag-id envia para AM uma mensagem de confirmação ou falha sobre sua entrada na SATA.
7. AM envia para FC uma mensagem de confirmação ou falha sobre a operação de criação de agente ag-id e está encerrado o protocolo.

Protocolo de Retirada. Utilizado quando for necessário retirar um agente da SATA. É descrito a seguir:

1. FC envia comando para AM. Este comando diz que o agente ag-id deve sair da SATA.
2. AM envia ao agente ag-id uma solicitação para execução do comando enviado pela FC.
3. agente ag-id envia uma resposta à solicitação feita. Esta resposta pode ser uma pedido par aque o AM aguarde, caso o agente esteja envolvido em alguma atividade. Neste caso a conversação é suspensa[10] e quando o

[10] A conversação é congelada e o AM ficará esperando uma resposta afirmativa ou negativa. Ressaltamos aqui que a conversação fica congelada mas o AM continua as outras conversações normalmente.

agente estiver pronto envia uma outra mensagem de resposta (afirmativa ou negativa). a conversação continua.

4. Caso a resposta do agente ag-id seja afirmativa, ele sai da SATA via protocolo de saída e envia uma confirmação para AM e este por sua vez envia uma mensagem de confirmação para FC e está encerrado o protocolo.
5. Caso seja negativa a resposta do agente ag-id o AM envia uma mensagem de falha para FC e está encerrado o protocolo.

Protocolo de Atualização Utilizado nas situações onde se precisa modificar a base de um agente. É descrito a seguir:

1. FC envia um comando de atualização da SATA para o AM indicando que o agente ag-id deverá será modificado.
2. AM solicita ao agente ag-id uma operação de cópia de sua base para uma variavel qualquer B.
3. agente ag-id envia para AM informação cujo conteúdo é B (a cópia de sua base) e inicia um protocolo de saída da SATA.
4. AM envia informação cujo conteúdo é a B (a cópia da base do agente ag-id) para FC.
5. FC atualiza a base com a operação *atualizar(B, B')*.
6. FC envia para AM informação cujo conteúdo é B'.
7. AM aguarda[11] uma mensagem do agente ag-id confirmando que ele saiu da SATA. AM envia solicitação ao agente ag-id para a substituição da sua base atual pela nova base B' recém atualizada.
8. agente ag-id responde à solicitação.
9. AM envia para agente ag-id informação cujo conteúdo é B'.
10. agente ag-id substitui base com a operação *substituir-base(B,B')*.
11. agente ag-id envia confirmação sobre o sucesso da substituição da base.

[11]Na prática não pode-se esperar indefinidamente. Deve haver um tempo limite para esta espera.

12. AM solicita ao agente ag-id que entre na SATA novamente. O agente ag-id atualiza seu autoconhecimento o agente inicia um protocolo de entrada na sociedade.

13. agente ag-id envia para o AM confirmação ou falha sobre sua entrada na SATA . AM envia para FC confirmação ou falha sobre a atualização da SATA. Está encerrado o protocolo.

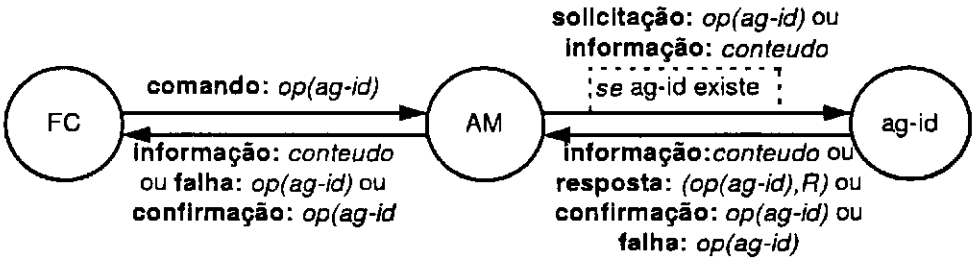


Figura 4.4: Visão geral dos protocolos de manutenção

Na figura 4.4 apresentamos um resumo dos protocolos de manutenção apresentados até aqui[12]. Na figura, é visto que o AM sempre se dirige ao agente através de uma *solicitação*, quando se trata de uma operação de manutenção. Isto permite ao agente recusar operações de manutenção enquanto estiver envolvido em alguma atividade. Em resumo, mostramos nesta figura, quais são as possíveis mensagens que cada entidade envolvida na manutenção de um agente pode receber ou enviar.

[12]Os círculos na figura 4.4 representam as entidades que se comunicam: FC, AM e um agente ag-id qualquer. As setas indicam a direção da passagem de mensagem e as mensagem são escritas junto às setas no seguinte formato: em negrito o tipo da mensagem (**confirmação** ou **solicitação** por exemplo), seguido de dois pontos, e em itálico os argumentos da mensagem (*op(ag-id)* por exemplo).

4.6 Discussão

Apresentamos aqui um esboço de como será definida e suportada a AC no MATHEMA. A nossa definição de AC apresenta os passos necessários desde a organização do domínio em contextos e profundidades, até a definição dos agentes e criação de suas bases de conhecimento. Em relação ao suporte apresentamos a especificação de uma ferramenta que deve auxiliar na tarefa de AC no MATHEMA.

Escolhemos a AC baseada em modelos para a modelagem conceitual do conhecimento. Essa abordagem nos permite estruturar o conhecimento de modo a garantir sua verificação formal. A abordagem por modelos ainda nos permite inserir no Sistema Tutor métodos de raciocínio variados[13], além de facilitar a tarefa de explicação.

O nosso objetivo geral é garantir que os agentes possam ser reestruturados sempre que seja necessário. Neste trabalho, isso é feito tomando-se como ponto de partida os modelos de conhecimento.

[13]Cada agente pode ter tantos métodos de resolução diferentes quantos forem necessários

Capítulo 5

Implementação do Ambiente de Manutenção

Resumo: Neste capítulo apresentamos um protótipo desenvolvido de acordo com a especificação do Ambiente de Manutenção (capítulo 4). Discutiremos o funcionamento geral cada módulo implementado e futuros melhoramentos. Ao final de cada seção fazemos um resumo do módulo em forma de entrada, saída pré e pós-condições.

5.1 Comentários Gerais sobre a Implementação

No capítulo 4 apresentamos a arquitetura do AM, para uma ferramenta de auxilio à AC no MATHEMA. A partir disso, desenvolvemos um protótipo cujo principal objetivo é testar os algoritmos de tradução e servir de base para o desenvolvimento de uma ferramenta mais robusta. Portanto, não estamos, neste momento, preocupados com os aspectos relacionados à performance.

Escolhemos a linguagem LPA-Prolog, por motivos pragmáticos. É uma linguagem de programação que permite o desenvolvimento rápido, evitando preocupações com aspectos que não estão no foco de nossa atenção agora, como por exemplo, interface.

Devido ao esforço significativo necessário para o desenvolvimento completo do AM, optou-se pela implementação apenas dos módulos tidos como primordiais. Dessa forma,

implementamos os seguintes módulos:

- *Editor de Ontologia & Editor de Conhecimento Procedural:* estes dois módulos estão reunidos em um só módulo chamado *editor*.
- *Verificador:* implementado com simplificações.
- *Tradutor:* implementação completa baseada no algoritmo a ser apresentado na seção 5.2.3.
- *Gerenciador de Agentes:* implementado com simplificações.

Convenções de Extensões de Arquivos. Nas próximas seções, ao discutirmos a implementação do protótipo para o AM, usaremos algumas convenções para os tipos de arquivos. Estas convenções são apresentadas abaixo:

Arquivo .AGT É um arquivo gerado automaticamente pelo GERENCIADOR DE AGENTES e descreve um agente MATHEMA.

Arquivo .BC É um arquivo gerado pelo interpretador como resultado da tradução do modelo de perícia.

Arquivo .DIN Neste arquivo são armazenadas as declarações de inferência.

Arquivo .FIN Contém as funções de inferência. É gerado automaticamente a partir de um arquivo DIN. Este arquivo precisa ser editado pelo usuário depois de gerado, para que se indique, para cada função de inferência, quais os métodos de inferência a serem invocados.

Arquivo .INS Neste arquivo ficam armazenadas as instâncias de ontologia. É gerado automaticamente a partir de um arquivo BC.

Arquivo .LOG Um arquivo LOG contém informações de tradução usadas pelo verificador.

Arquivo .CML Contém o modelo de perícia (escrito em CML, seção 2.8). Este arquivo é criado automaticamente a partir de um arquivo BC.

Arquivo .ONT Contém a ontologia. Este arquivo é gerado automaticamente a partir de um arquivo BC.

Arquivo .MAP Contém informações sobre os mapeamentos entre os termos usados nas inferências e os termos de domínio, declarados na ontologia. Este arquivo é criado pelo EDITOR DE MAPEAMENTOS.

Arquivo .MET É uma biblioteca de "procedimentos"escritos em Prolog e que dão suporte às funções de inferência.

Arquivo .TRF Contém as declarações de tarefas. Gerado automaticamente a partir de um arquivo BC.

Arquivo .VER Um arquivo VER, ou "ver", contém os relatório de uma sessão de verificação. É criado pelo módulo VERIFICADOR.

Comentários sobre a Sintaxe do Modelo de Perícia. Neste trabalho a sintaxe da CML[1] como apresentada originalmente em [SWA+94] para construir Modelos de Perícia (MP). Isto implica que os MP devem usar os termos reservados da CML em inglês.

Devemos ainda salientar que modificamos a definição dos papéis de conhecimento nas tarefas, onde acrescentamos a informação de *tipo do papel*. O *tipo do papel* pode ser: conjunto, lista, ou elemento. Isto indicará ao tradutor como o papel de conhecimento deve ser manipulado pela memória de trabalho do interpretador de tarefas (fig. 2.9).

Outra modificação, mais sutil, foi a adoção da extensão da sintaxe CML para estrutura de controle, não definida originalmente no artigo de Schreiber et al. [SWA+94]. Esta extensão foi desenvolvida pelo mesmo grupo só que ainda não foi publicada em artigo, entretanto está disponível na página do projeto KADS na internet[2].

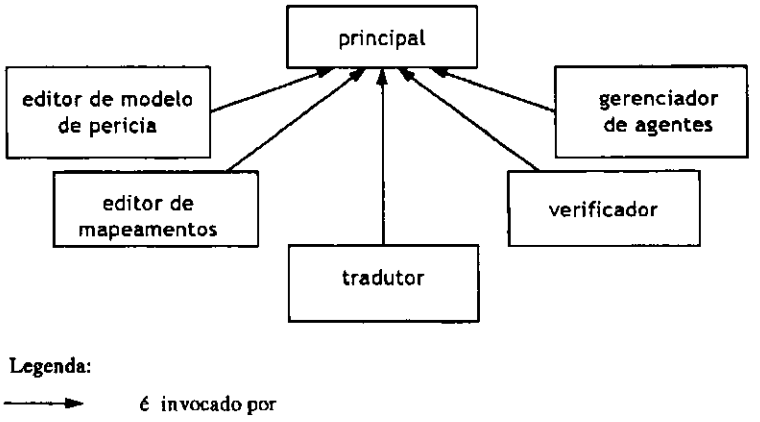


Figura 5.1: Arquitetura do protótipo para o AM

5.2 Módulos Implementados

Na figura 5.1 apresentamos os módulos implementados e a relação de ativação entre eles. Note que os módulos não são organizados em uma hierarquia de dependência, sendo interligados apenas pelo módulo principal.

O fluxo de informações entre os módulos é mostrado na figura 5.2. Na realidade essa figura mostra o compartilhamento de arquivos, que é o principal modo de troca de informações no nosso protótipo. Os arquivos em destaque, cercados por uma linha tracejada, representam os arquivos finais que formam um agente criado pelo AM (veja seção 5.3).

5.2.1 O Módulo Principal

Este módulo tem como objetivo servir de interface para os diversos submódulos: tradutor, editor de modelos de perícia, editor de mapeamentos, e verificador. O MÓDULO PRINCIPAL é portanto, uma interface gráfica[3] que dá acesso às funcionalidades do AM.

Na janela principal do AM (fig. A.1, apêndice A) encontramos um campo deno-

[1]Conceptual Modeling Language, vista na seção 2.8.

[2]http://www.swi.psy.uva.nl/projects/kads22/index.html

[3]O protótipo foi desenvolvido na plataforma Windows/95 usando LPA-Prolog.

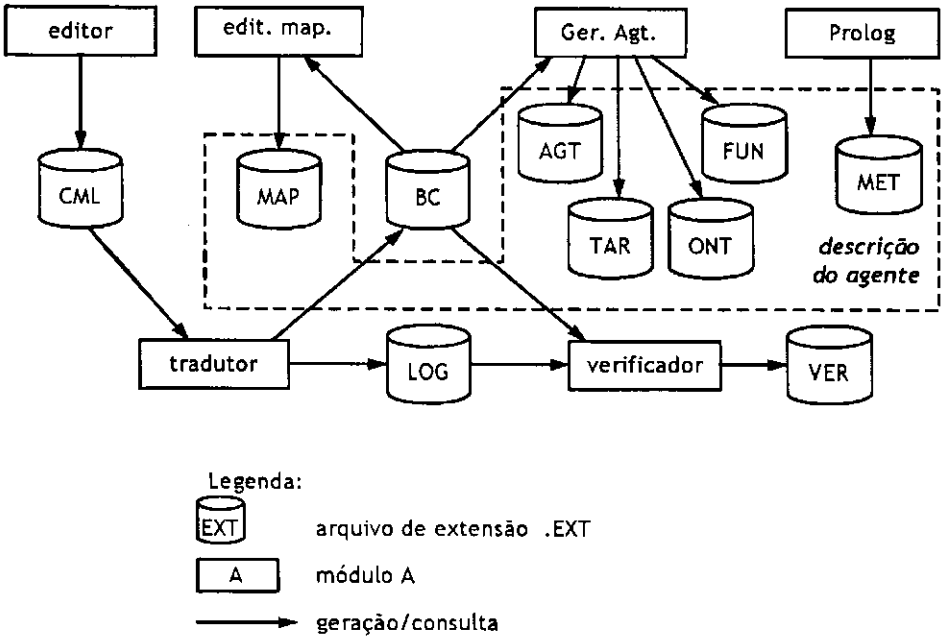


Figura 5.2: Relação entre os arquivos e módulos no protótipo AM

minado *arquivo a ser usado*, onde deve ser informado o nome de um arquivo CML qualquer. Se o arquivo já existe ele será aberto, senão um novo arquivo será criado com o nome informado. Todos os outros módulos funcionam com base no nome do arquivo de trabalho atual[4].

Os demais componentes da janela principal são os botões que iniciam os diversos módulos, o botão *sobre*, que abre uma janela de informação sobre o protótipo AM. Há ainda, uma linha de aviso na base da janela principal, para mostrar ao usuário o estado atual do AM, que pode ser: *pronto*, *processando...* ou um aviso de que *a última operação provocou um erro.*

5.2.2 Editor

A partir deste módulo pode-se criar e modificar modelos de perícia. Este módulo foi implementado como um editor de textos com as funções de copiar e colar texto, salvar

[4]Assim, por exemplo, se quisermos fazer a verificação do arquivo `meu-arquivo.CML`, o verificador, com base neste nome, vai procurar pelos os arquivos `meu-arquivo.BC` e `meu-arquivo.LOG`

arquivos, etc. Veja a figura A.3, apêndice A)

O EDITOR apresenta ainda um submódulo chamado *assistente de sintaxe*. Trata-se de um sistema de substituição de texto, baseado em um banco de padrões de texto que chamamos de Pseudo-Gramática. A Pseudo-Gramática contém uma representação da gramática CML em formato BNF[5]. Para se usar o assistente deve-se selecionar uma palavra reservada do CML cuja sintaxe precisamos de ajuda. Se existir na Pseudo-Gramática uma entrada para a palavra selecionada então este será substituída pelo texto indicado na Pseudo-Gramática. Caso contrário nada acontece. O *assistente de sintaxe* visa poupar o trabalho de consulta à gramática durante o processo de edição de um modelo de conhecimento[6].

Em resumo, o Editor toma como entrada um arquivo CML e tem como saída também um arquivo CML. Não há pré-condição. Se não existir o arquivo indicado é criado um com o nome dado. A pós-condição é a existência de um arquivo CML pronto para ser traduzido.

5.2.3 Tradutor

O TRADUTOR cria a base de conhecimento a partir de um Modelo Perícia (MP). A base de conhecimento é formada pelos módulos dependentes do domínio apresentados na figura 2.9.

Geração Semi-Automática da Base de Conhecimento

Analisando a figura 2.9 (pág. 30) podemos notar que ela apresenta vários elementos que não são dependentes do domínio: interpretador, memória de trabalho, funções de acesso (à base de conhecimento) e métodos de inferências. Estes elementos podem ser vistos como uma "máquina genérica"de inferência. As partes dependentes do domínio

[5]Bakus-Naur Form.

[6]Uma outra opção para implementação do assistente seria apresentar ao usuário janelas de formulários quando ele quiser editar um elemento novo no modelo de conhecimento. Essa é uma boa abordagem de implementação, mas por ser mais complicada que a que adotamos, foi colocada em segundo plano e pode ser usada em uma versão posterior do protótipo.

são: descrição de tarefas, funções de inferência, novos métodos de inferência[7] e base de conhecimento. Os elementos dependentes do domínio podem ser vistos como uma "grande base de conhecimento"(declarações de tarefas inclusive).

Uma vez implementadas a "máquina genérica", construir outro SBC é apenas uma questão de implementar a "grande base de conhecimento". A construção desta "grande base de conhecimentos"é completamente baseada no MP, pois é no MP que estão descritas as tarefas, as inferências, a os conhecimentos do domínio (ontologia e modelo de domínio).

Schreiber apresenta uma implementação para a "máquina genérica". No entanto, não apresenta um algoritmo para a criação automática dos módulos dependentes do domínio a partir do MP. Em vez disso ele mostra o formato em que estes módulos devem ser representados. A partir disso desenvolvemos um algoritmo para criação da "grande base"[8].

Cabe ainda ressaltar que o algoritmo não produz automaticamente os métodos de inferência (módulos mais escuros na figura 2.9), que devem ser desenvolvidos à parte e adicionados ao sistema conforme a necessidade. Por conta disto nossa forma de tradução não pode ser considerada automática, mas semi-automática.

A representação O interpretador exige uma representação em Cláusulas de Horn[9]. Desse modo devemos traduzir cada elemento em CML para um conjunto de cláusulas, criando tantas clausulas quantos forem seus atributos. Como exemplo, consideremos a definição de tarefa do KADS. Uma tarefa possui os atributos nome, entrada, saída, e especificação (entre outros), como mostramos na figura 5.3.

A representação em cláusula para a tarefa mostrada na figura 5.3 é representada por cinco cláusulas, como mostrada na figura 5.4.

Convenções. Neste algoritmo usamos as seguintes convenções:

[7]Se não houver métodos na biblioteca de métodos que sejam adequados para o problema atual.

[8]O algoritmo que desenvolvemos é portanto particular para esse formato.

[9]Isto é altamente influenciado pelo fato de que a implementação dos módulos genéricos terem sido implementados em Prolog.

```
tarefa nome-da-tarefa;
    entrada: nome-papel-entrada;
    saida: nome-papel-saida ;
    espec:
        < passo 1, passo 2, passo n >.
end tarefa ;
```

Figura 5.3: Exemplo de definição de tarefa

```
tarefa(nome-da-tarefa).
entrada(nome-da-tarefa, nome-papel-entrada).
saida(nome-da-tarefa, nome-papel-saida).
espec(nome-tar, (passo1, ..., passo n)).
```

Figura 5.4: Exemplo de representação em cláusulas para um exemplo de tarefa

- *construtor:* são os construtores básicos de um determinado nível de conhecimento no MP. Por exemplo, conceito e tupla são construtores do nível de domínio, visão-de-domíno é um construtor do nível de inferência e assim por diante.

- *nome:* é um nome definido pelo engenheiro do conhecimento ao construir o MP. Assim, no algoritmo, citamos nome-construtor para dizer nome particular de um construtor.

- *atributo:* usado no algoritmo para denotar os outros atributos dos construtores exceto o nome.

- *id:* é um identificador de instância ou tupla.

- *valor-tupla:* valor encontrado em uma instância ou tupla.

- *tipo-papel:* é um dos seguintes: conjunto, lista, ou elemento. Indica se o papel usado na tarefa é um conjunto de informação, uma lista ordenada, ou elemento simples.

- *função:* pode ser: tarefa ou inferência.
- *entrada & saida:* são as entradas e saídas das funções.

O algoritmo de tradução semi-automática do MP para o formato de cláusulas é mostrado a seguir (algoritmo 1).

Este algoritmo usa a tabela 5.5 para fazer a tradução da estrutura de controle (no corpo da tarefa). Esta tradução é intimamente relacionada com os comandos que o interpretador de tarefas reconhece. Os comandos reconhecidos pelo interpretador de tarefas implementado por Schreiber são mostrados na tabela 5.5. Na primeira coluna está o comando de acordo com a sintaxe do MP e na segunda coluna está a sua representação em cláusula de acordo com o exigido pelo interpretador.

comando na sintaxe do MP	*traduzir para*
repetir Acao ate Condicao	repetir(Acao, ate(Condicao)).
se Condicao entao Acao	se(Condicao, Acao).
se Condicao entao Acao1 senao Acao2	se(Condicao, Acao1, Acao2).
para-cada a $\in$ A faca Acao	paratodo(operacao(membroA, a'), Acao) onde a' é uma variável que substitui "a".
teste se A vazio	operacao-dados(vazio, A) onde A é lista ou conjunto.
atribuição a:=a+b	operacao-dados(adiciona, a, b).
atribuição a:=b+c	operacao-dados(armazena, b', b), operacao-dados(adiciona, b', c), operacao-dados(armazena, a, b'), onde b' é uma variável temporária.

Figura 5.5: Tabela de tradução da estrutura de controle das tarefas

Algoritmo 1: Geração de Base de Conhecimento

Dados : Especificação CML
Resultados : Representação em clausulas
Método :
para cada *construtor da ontologia* **faça**
criar cláusula na forma `construtor(nome-construtor)`
fim
para cada *atributo de um construtor da ontologia* **faça**
criar cláusula na forma `atributo(nome-elemento, valor)`
fim
para cada *construtor instância ou tupla no modelo de domínio* **faça**
criar uma cláusula na forma `construtor(id, valor-tupla)`
fim
para cada *definição de entrada e saída de tarefa* **faça**
criar cláusula na forma `tipo-de-dado(nome-papel, tipo-papel)`
fim
para cada *definição de tarefa* **faça**
criar cláusula `estrutura-de-controle(nome-tarefa, corpo)`, onde corpo é gerado da seguinte forma:
para cada *chamada de função (na estrutura de controle)* **faça**
criar cláusula `exec-funcao(nome, entrada, saida)`
fim
para cada *passo descrito na estrutura de controle* **faça**
usar a tabela da figura 5.5
fim
fim
para cada *definição de inferência* **faça**
criar uma cláusula `inference-function(nome-inferencia, entrada,saida)`
fim

Implementação do Algoritmo

O TRADUTOR é uma implementação do algoritmo apresentado na seção 5.2.3. Neste protótipo, o TRADUTOR é um analisador sintático para CML que realiza tradução orientada pela sintaxe. Um analisador sintático é fácil de se desenvolver em Prolog. Outra opção, fugindo do PROLOG, seria usar um gerador de analisador sintático tal como o YACC[10] para linguagem C ou o ANTLR[11] para JAVA. A tarefa do tradutor é, então, analisar o arquivo CML e executar as diretivas de tradução para cada parte sintaticamente correta. Estas diretivas seguem à risca o algoritmo 1.

Durante a tradução um arquivo de LOG é gerado. Este arquivo contém informações para o VERIFICADOR, mas esta não é uma imposição da especificação feita no capítulo 4 e sim uma particularidade de nossa implementação. O arquivo LOG deve conter uma tabela com as funções[12] declaradas.

Ao final da tradução, caso um erro sintático tenha sido encontrado, um relatório resumido da tradução é apresentado ao usuário.

Em resumo. O TRADUTOR tem como entrada um arquivo CML e tem como saída um arquivo BC e um arquivo LOG. A pré-condição é que exista o arquivo CML indicado como entrada. A pós-condição é existir os arquivos BC, LOG e um relatório em caso de erro sintático, indicando o trecho onde provavelmente ocorreu o erro.

5.2.4 Verificador

O VERIFICADOR já foi discutido na seção 4.3.2. Por simplificação, a nossa implementação do verificador inclui apenas a verificação do Conhecimento Procedural. Assim, o nosso protótipo só realiza as seguintes checagens[13]:

- verificação de falta de declaração de funções.

[10]Yacc-Yet Another Compiler-Compiler, escrito por S.C. Johnson of Bell Telephone Laboratories, Murray Mill, N.J.-USA.

[11]ANTLR foi desenvolvido por Terence Parr, no MageLang Institute (http://www.MageLang.com).

[12]Função é um nome genérico para algo que pode ser uma Tarefa ou uma Inferência.

[13]Outros aspectos podem ser acrescentados (em versões posteriores) tais como a verificação de papeis de conhecimentos usados por tarefas ou inferências mas não declarados na ontologia.

- verificação de múltiplas declarações de uma mesma função.

O algoritmo básico para o verificador é dividido em duas partes, uma para cada item citado acima.

Para verificação de falta de declaração de função, o nosso algoritmo usa o arquivo BC gerado pelo TRADUTOR e uma tabela de funções declaradas encontrada no arquivo LOG (algoritmo 2). O verificador procura no arquivo BC por referências às funções que não estão na tabela de funções declaradas. A partir disso o VERIFICADOR gera uma tabela de funções não declaradas[14].

Algoritmo 2: Identificação de funções não definidas

Dados : Um arquivo BC e um arquivo LOG
Resultados : Lista de funções não definidas
Método :
para cada *chamada de função em BC* **faça**
    procurar função na tabela de funções definida no arquivo LOG;
    se *não encontrar função na tabela* **então**
        adiciona entrada à lista de funções não-definidas;
    fim
fim

Para verificação de múltiplas declarações para uma função, o VERIFICADOR usa apenas a tabela de funções declaradas, encontrada no arquivo LOG. Esta tabela contem o nome e o tipo da função (tarefa ou inferência). O algoritmo considera multideclarada uma função cujo nome aparece mais de uma vez na tabela (algoritmo 3). Isso significa que não podemos declarar uma tarefa e uma inferência com um mesmo nome. Isso é uma restrição imposta pela CML, que não faz diferenciação ao chamada de tarefas e inferências[15]. A saída do VERIFICADOR é um arquivo VER, que contém um relatório da sessão de verificação.

[14]Uma melhoria aqui, será tentar achar nomes de funções declaradas que se assemelhem com os nomes das funções não declaradas para sugerir ao usuário a correção.

[15]Esta restrição no entanto não é uma desvantagem. Na realidade, nós não encontramos na prática, nenhum problema relacionado com isso.

Algoritmo 3: Identificação de funções multidefinidas

Dados : Um arquivo LOG
Resultados : Lista de funções multidefinidas
Método :
para cada *entrada da tabela de funções em LOG* **faça**
  se *existir outra entrada com o mesmo nome na tabela* **então**
    adiciona entrada à lista de funções multidefinidas;
  fim
fim

Em resumo. O VERIFICADOR toma como entrada um arquivo LOG e um arquivo BC e tem como saída um arquivo VER. A pré-condição é existir os arquivos BC e LOG gerados a partir de um mesmo CML. A pós-condição é existir um arquivo VER, onde apresenta as funções multideclaradas e as não declaradas.

5.2.5 Editor de Mapeamento

As entradas, saídas e conhecimentos que as inferências acessam, são definidos de modo independente de domínio. O EDITOR DE MAPEAMENTOS serve para que se indique o mapeamento entre os termos usados no nível de inferência e o conhecimento do domínio particular que se está trabalhando.

O funcionamento do EDITOR DE MAPEAMENTOS é simples. A partir de um arquivo BC, o Editor busca informações sobre as entradas e saídas e conhecimento usado pelas inferências. Cada uma dessas entradas e saídas gera uma janela de diálogo com o usuário para que ele informe qual o termo do domínio específico que está relacionado com aquele termo da inferência.

As informações requeridas pelo EDITOR DE MAPEAMENTOS são Tipo e Termo da Ontologia:

- Tipo do termo de entrada/saida encontrada na inferência. Pode ser *entidade* indicando que se trata de um conceito, *relação* para relações, *expressão* para expressões, ou *def*, quando se quer definir constantes.

- Nome do termo. Deve ser o nome de um termo (ou lista de termos) definido na ontologia.

Em resumo. O EDITOR DE MAPEAMENTOS tem como entrada um arquivo BC e como saída um arquivo MAP. A pré-condição é que o arquivo BC exista e já tenha sido verificado com sucesso. A pós-condição é a existência de um arquivo MAP e também de um arquivo BC prontos para serem usados na criação de um novo agente[16].

5.2.6 Gerenciador de Agentes

O GERENCIADOR DE AGENTES, é o módulo responsável pelas operações de criação, alteração, entrada e saída de agentes da SATA. É esse módulo que cuida para que cada modelo de conhecimento pertença a somente um agente e cada agente só tenha um modelo de conhecimento.

O algoritmo do GERENCIADOR DE AGENTES segue os protocolos definidos na seção 4.5. Para maior simplificação apresentamos o algoritmo do GERENCIADOR DE AGENTES em duas partes: uma para criação de agentes e outra para manutenção de agentes (algoritmo 5).

Para a criação de um agente usamos o algoritmo 4. Uma entrada para a criação do agente é uma identificação, que será checada em uma tabela de agentes existentes para evitar conflito com agentes já criados. Caso ainda não exista um agente com aquela identificação, então o GERENCIADOR DE AGENTES vai gerar as informações necessárias para criar um agente, de acordo com o modelo de agentes MATHEMA (visto na seção 3.4.1). Estas informações são:

- *habilidades:* as habilidades são criadas a partir dos métodos descritos no BC.
- *sistema tutor:* o GERENCIADOR DE AGENTES assume que o sistema tutor é descrito em um arquivo BC de nome igual à identificação do agente.

[16] O arquivo BC pode também ser usado para um agente já existente, por meio de uma operação de substituição da base de conhecimento (ver seção 4.3.3)

- *mundo externo:* o mundo externo é uma tupla formada pelas identificações do AM e do Agente de interface. Esta tupla é gerada automaticamente pelo GERENCIADOR DE AGENTES). Estas identificações serão usadas pelo agente para enviar mensagens para estas duas entidades.

Algoritmo 4: Criação de agente

Dados : Uma identificação de agente **ag-id** e nomes dos arquivos BC e MAP

Resultados : Um novo agente com identificação **ag-id** descrito pelos arquivos AGT, DIN, FIN, TRF, ONT e INS

Método :

se *ainda não existir um agente com a identificação* ***ag-id*** **então**

gera os arquivos ONT, INS, DIN, FIN e TRF a partir do arquivo BC fornecido;

gera instância do modelo do agente **ag-id** no arquivo AGT ;

senão

informa que não é já existe agente com a identificação **ag-id**

fim

Para que um agente entre na SATA (ou saia dela), usamos o algoritmo 5. Em caso de alteração esclarecemos que o agente não envia uma cópia de sua base, como especificado na seção 4.5. Optamos nesta implementação por usar apenas uma referência à base que o agente usa. Assim não temos realmente que copiar a base do agente, mas apenas saber qual o nome do arquivo BC que ele usa[17]. O GERENCIADOR DE AGENTES então passa a ter a tarefa de controlar o acesso a arquivos BC definidos no sistema, para que não se modifique acidentalmente a base de conhecimento de um agente que ainda esteja na SATA.

Em resumo. O GERENCIADOR DE AGENTES tem como entrada os arquivos BC, MAP e uma identificação de agente. Tem como saída um arquivo AGT ou uma ação

[17]Em uma implementação onde os agentes sejam realmente distribuídos e o gerenciador de agentes não "veja" todos as bases de conhecimentos essa abordagem não pode ser aplicada.

Algoritmo 5: Manutenção de agente

Dados : Uma identificação de agente **ag-id** e uma operação de manutenção **op**
Resultados : SATA modificada
Método :
se *existir agente* **ag-id** **então**
  envia solicitação da operação **op** ao agente **ag-id** ;
  se *se agente* **ag-id** *responder afirmativamente* **então**
    inicia um protocolo de acordo com a operação **op** ;
  senão
    informa que agente **ag-id** não pode entrar em manutenção no momento
  fim
senão
  informa que a o agente **ag-id** não existe
fim

sobre um agente já existente. A pré-condição é que a identificação do agente ainda não esteja sendo usada (em caso de novo agente) e que o BC seja um arquivo já verificado e seja mapeado para o arquivo MAP. A pós-condição é que não exista mais de um agente com a mesma identificação na SATA.

5.3 Considerações sobre o Agente

Como visto na seção 5.2.6, o módulo gerenciador de agentes cria os arquivos AGT, DIN, FIN, ONT, INS e TRF a partir de um arquivo BC. Esses arquivos (exceto BC), juntos, descrevem um agente.

O arquivo FUN é gerado de modo incompleto. Para que ele se torne completo é necessário que o editemos para acrescentar detalhes tais como: métodos de inferência utilizados e conhecimento de domínio acessado[18]

[18]Editar o arquivo FIN exige conhecimentos sobre o funcionamento do interpretador de tarefa e conhecimentos sobre o Prolog.

Nesta implementação, o arquivo MET (métodos de inferências) é na realidade uma biblioteca de predicados Prolog.

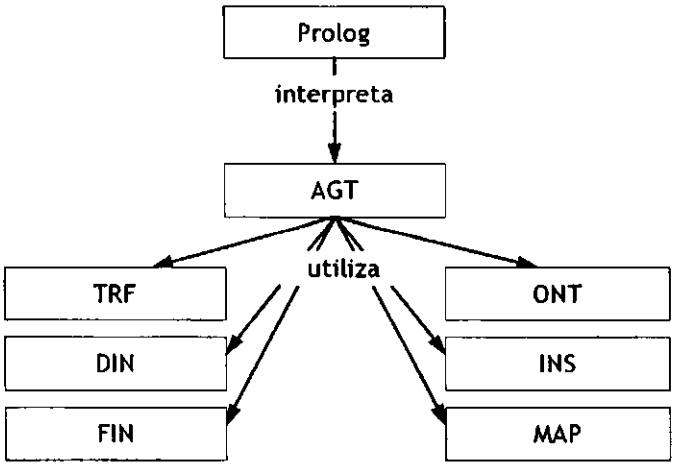


Figura 5.6: Relação entre os arquivos de um agente

O módulo principal de uma agente é seu arquivo AGT. Este arquivo AGT, é diretamente interpretável pelo Prolog. Os arquivos são carregados pelo módulo AGT durante a execução. Isso é ilustrado na figura 5.6.

Os arquivos gerados a partir da tradução do BC, representa os módulos dependentes do domínio na arquitetura apresentada na figura 5.7 (declarações de tarefas, ontologia e instancias, funções de inferência, declaração de funções). Vale relembrar que os módulos genéricos, são incluídos no agente (arquivo AGT) automaticamente no momento de sua criação.

A estratégia de criação de agentes aqui está muito ligada à linguagem de implementação que escolhemos (o Prolog). Por isso, caso se escolha outra linguagem para desenvolvimento de versões posteriores do Ambiente de Manutenção, detalhes técnicos da criação de agentes pode mudar significativamente. O importante é manter a idéia central delineada aqui, que é o uso dos protocolos de manutenção, mostrados na seção 4.5 para guiar o processo de criação e manutenção dos agentes.

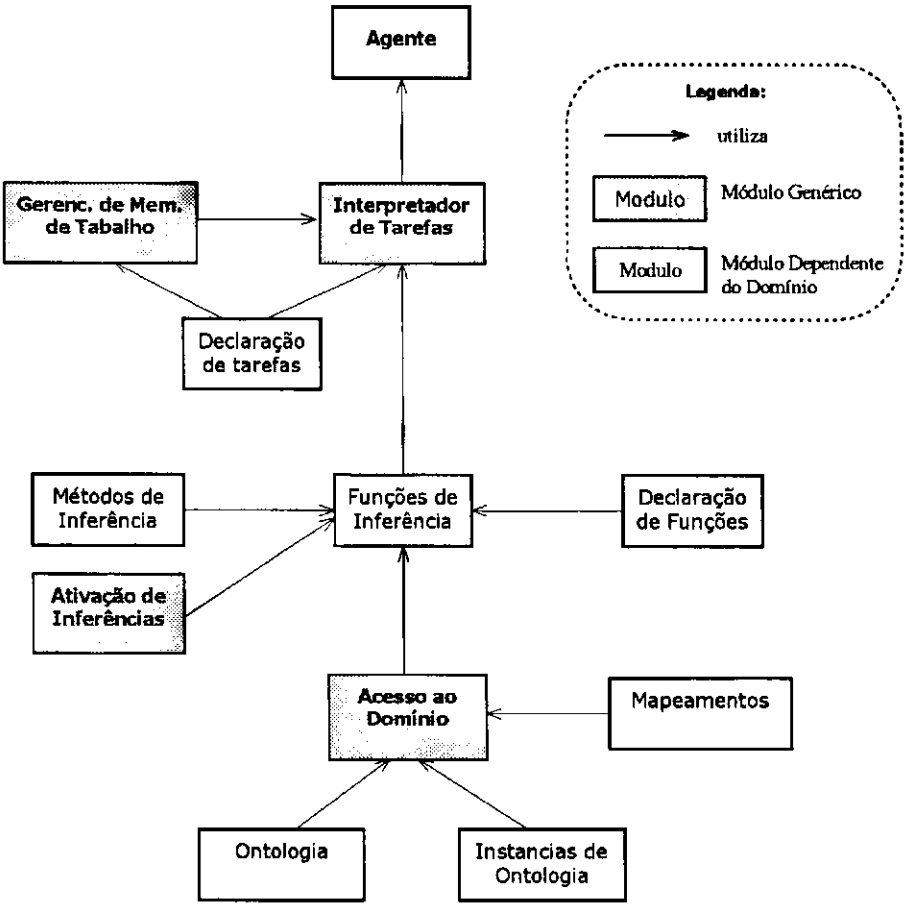


Figura 5.7: Agente e Interpretador (adaptado de [SWB93])

5.4 Discussão

Usar o Prolog é apenas uma dentre as possibilidades de implementação para o AM. Por isso optamos por apresentar os algoritmos em pseudo-código para servirem de guia para uma outra implementação. Esta é também a razão para termos feito um resumo em termos de pré e pós condições, ao final da descrição de cada elemento implementado.

No capítulo 6 um exemplo de uso do AM é mostrado.

Capítulo 6

Aplicação & Resultados

Resumo: Neste capítulo, apresentamos um exemplo de aplicação do Ciclo de Aquisição de Conhecimento para a construção de um STI MATHEMA. O domínio escolhido foi a resolução de equações algébricas. Mostramos aqui os resultados obtidos no processo de modelagem do conhecimento e na modelagem de um agente MATHEMA. Mostramos ainda o uso do Ambiente de Manutenção no processo de modelagem e criação da SATA.

Introdução

Neste capítulo, mostramos os passos para a construção de um Agente Tutor (AT) de acordo com o Ciclo de AC definido na seção 4.1 e com o suporte da ferramenta apresentada no capítulo 5.

Para desenvolvermos um exemplo de construção de agente, escolhemos o domínio das equações do segundo grau. Nossa escolha se deve ao fato de que se pode achar farto material sobre o assunto. Para o estudo que queremos fazer consideramos a organização de domínio proposta por Costa [Cos97]. As seções seguintes apresentam uma análise do domínio sob o ponto de vista da resolução de problemas.

6.1 Apresentação do Domínio

Paulo Bucchi [Buc92], em seu livro de matemática do segundo grau, apresenta as seguintes seções no capítulo sobre funções do segundo grau:

1. *definição:* definição de função do segundo grau.
2. *gráfico:* estudo sobre o gráfico de funções do segundo grau.
3. *raízes:* o conceito de raízes e métodos para calcular raízes de funções do segundo grau.
4. *vértice da parábola:* estudo sobre a Imagem da função do segundo grau.
5. *construção do gráfico:* técnicas de construção do gráfico da função do segundo grau.
6. *sinal:* estudo do sinal da função do segundo grau.
7. *inequações:* estudo de inequações do segundo grau.

Uma *equação do segundo grau* é uma equação algébrica cujo maior expoente é 2. Tais equações podem ser reduzidas para a forma $ax^2 + bx + c = 0$ onde a, b e c são constantes reais, com $a \neq 0$ e x uma variável qualquer. Esta forma é dita **forma canônica** da equação. O problema de se resolver uma equação é determinar quais são os valores de sua variável que tornam a equação verdadeira. Estes valores são também chamados de *raízes* da equação. A resolução de uma equação do segundo grau pode ser vista como uma tarefa de transformar a equação inicial para uma forma canônica equivalente. Há contudo os métodos para casos especiais (b=0 ou c=0), que geralmente são mais simples de se executar.

No domínio de equações, um conceito chave é *equação do segundo grau.* O estágio fundamental é o aluno aprender a reconhecer uma equação deste tipo. Além deste, muitos outros conceitos são necessários (regras de simplificação, fórmula de Bháskara, etc.) para o estudo desse tema. A seguir, apresentamos um conjunto de conceitos-chave, para o estudo de equações do segundo grau (também baseado no livro de Paulo Bucchi).

Por exemplo, tomemos as primeiras unidades pedagógicas:

- *definição:* nesta seção, aprende-se a **forma canônica** de uma equação do segundo grau.
- *gráfico:* nesta seção, Bucchi apresenta a noção de **gráfico da função** e mostra o conceito de **concavidade do gráfico.**
- *raízes:* nesta seção, apresenta-se os conceitos de **zeros da função**, cálculo do *discriminante* e mostra-se como reconhecer **a quantidade de raízes reais** e **como calcular as raízes** de uma equação do segundo grau.
- *vértice:* apresenta os conceitos de **vértice da parábola**, cálculo das **coordenadas do vértice** e apresenta ainda o conceito de **conjunto imagem** da função do segundo grau.

6.2 Organização do Domínio

Na seção 6.1, apresentamos o domínio das equações do segundo grau. Nossa preocupação maior foi destacar os problemas deste domínio e estratégias gerais para resolvê-los. Nesta seção, iniciamos o Ciclo de AC MATHEMA para o domínio escolhido. Para tanto, consideramos um domínio bastante genérico das equações do segundo grau, segundo vários contextos e várias profundidades.

O primeiro passo do Ciclo de AC MATHEMA é a organização do domínio. A organização é feita em duas etapas:

1. *criação da visão externa:* construção da árvore de contextos, profundidades e lateralidades. Nesta etapa, identificam-se os subdomínios que darão origem à SATA.
2. *criação da visão interna:* construção do *curriculum*, problemas e identificação do conhecimento de apoio e suporte. Esta etapa serve como fonte do material usado para a modelagem de conhecimento dos agentes.

Visão Externa do Domínio. Um estudo deste domínio já foi iniciado por Costa [Cos97]. Na figura 6.1 é mostrada a organização do domínio da Álgebra segundo os contextos dos Métodos Canônicos (C_1) e o contexto dos Métodos Numéricos (C_2). Na figura vemos que o contexto dos Métodos Canônicos possui a profundidades dos Números Reais (P_{11}) e profundidade dos Números Complexos (P_{12}). Na parte inferior da arvore, temos algumas das lateralidades para a profundidade P_{11} (L_{111}, L_{112} ... L_{11n}).

Agora, consideremos o contexto C_1 e a profundidade P_{11}. Esse par contexto-profundidade indica um subdomínio que indicaremos por d_{11} e chamaremos de *Subdomínio do Métodos Canônicos para o Contexto dos Números Reais.*

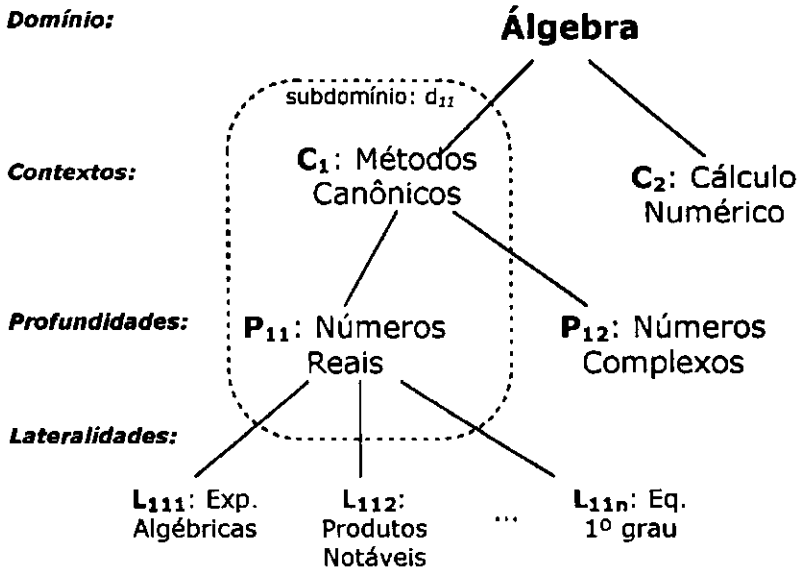


Figura 6.1: Organização do Domínio da Álgebra segundo Contextos, Profundidades e Lateralidades

Nesta etapa, devemos ainda criar uma identificação de agente para cada subdomínio. Por exemplo, criamos a identificação `EqCanonica` para o agente associado ao subdomínio d_{11}.

Visão Interna do Subdomínio Por simplificação, apresentamos apenas a visão interna do subdomínio d_{11}. Para esta etapa utilizamos a discussão apresentada na seção 6.1.

Consideremos estas seções como as *unidades pedagógicas* de um curso sobre equações do segundo grau. Temos ainda que considerar uma *ordem* entre estas unidades, como por exemplo, qual unidade é pré-requisito de outra. Isto nos levará a um diagrama de pré-requisitos, como visto na figura 6.2. Nesse diagrama, os elementos de mais baixo nível são pré-requisitos para os de mais alto nível. Já os elementos de um mesmo nível não apresentam nenhuma relação entre si. Desse modo, a unidade *gráfico* e a unidade *raízes* são independentes entre si, mas ambas têm como pré-requisito a unidade *definição*. A unidade *vértice* tem como pré-requisitos a unidade *gráfico* e a unidade *zeros*. As demais unidades seguem uma ordem mais bem comportada: *vértice* é pré-requisito para *sinal*, que é pré-requisito para *construção do gráfico*.

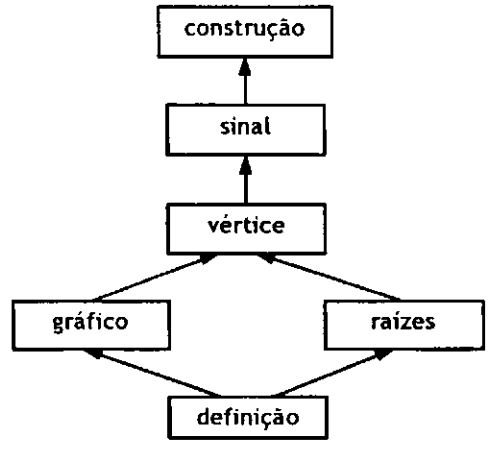


Figura 6.2: Diagrama de pré-requisitos para d_{11}

Assim, podemos propor um *curriculum* para d_{11} de acordo com o formato de *curriculum* definido na seção 3.1.2. A figura 6.3 apresenta um *curriculum* construído a partir das unidades pedagógicas apresentadas acima.

Falta-nos ainda, completar as unidades pedagógicas. Como visto na seção 3.1.2 as unidades pedagógicas são formadas por unidades de conhecimento. Para construir nos-

curriculum	
subdomínio:	d_{11}.
unidades pedagógicas:	definição, gráfico, zeros, vértice, sinal, construção do gráfico.
ordem pedagógica:	{definição} → {gráfico, raízes} → {vértice} → {sinal} → {construção}.

Figura 6.3: *curriculum* de d_{11}

sas unidades pedagógicas, utilizaremos os conceitos principais do domínio apresentado na seção 6.1. Na figura 6.4 mostramos a unidade pedagógica sobre raízes da equação e na figura 6.5 exemplos de problemas.

unidade pedagógica	
nome:	raízes da equação
conhecimento:	zeros, discriminante, quantidade de raízes, calculo de raízes.
problemas:	lista de problemas da unidade.

Figura 6.4: Unidade Pedagógica Raízes

6.3 Modelagem do Conhecimento dos Subdomínios

Na etapa inicial, foi feita a associação entre os subdomínios e os agentes. Nesta etapa, faremos a modelagem do subdomínio de conhecimento associado ao agente. Nesta dissertação, foi dada ênfase na modelagem do conhecimento de resolução de problemas. Mas, nesta etapa, de um modo mais geral, deve-se fazer a aquisição de todo o conhecimento necessário pelo agente tutor, isto é, conhecimento de estratégias de resolução de problemas, problemas, conceitos, exemplos, estratégias pedagógicas, etc.

Nosso ponto de partida será a identificação e a descrição dos métodos de resolução de problemas, já que resolução é uma característica fundamental do agente tutor MATHEMA. Ao identificarmos os problemas, em uma etapa anterior, indicou-se o método

problema 1	
enunciado:	Determine os zeros da função $f : \mathbf{R} \rightarrow \mathbf{R}$ definida por $f(x) = x^2 - 4x + 4$.
dados:	$a = 1, b = -4, c = 4$
conhecimento:	discriminante, quantidade de raízes, fórmula de Bháskara
método:	determinar-raízes-eq-canonica
solução:	<nenhum exemplo>
problema 2	
enunciado:	Determine as raízes da função $f(x) = x^2 + 3x + 5$
dados:	a=1, b=3, e c=5.
conhecimento:	discriminante, quantidade de raízes, fórmula de Bháskara
método:	determinar-raízes-eq-canonica
solução:	<nenhum exemplo>

Figura 6.5: Exemplos de problemas

de resolução utilizado. Agora, deve-se detalhar estes métodos.

Nesta etapa, a escolha de uma metodologia de modelagem, suportada por uma ferramenta com suporte a biblioteca de métodos genéricos, pode ser um fator de aceleração no processo de construção de STI-MATHEMA.

Como visto na seção 2.10, utilizaremos o KADS para descrever os métodos de resolução de problemas. Ou, nos termos do KADS, devemos construir o modelo de perícia para os agentes. Isso requer:

1. identificação das tarefas do método
2. identificação da inferências do método
3. identificação da ontologia e suas instâncias

6.3.1 Identificando o Método de Resolução

Os problemas indicam métodos de resolução. Por exemplo, na figura 6.5 há indicação para o método *determinar-raízes-eq-canonica*. Nesta seção, identificaremos e descreveremos este método.

O *primeiro passo* na resolução de uma equação do segundo grau é verificar se ela está na forma canônica. Se não estiver na forma canônica, é necessário que se faça uma simplificação de expressões para que transformemos a equação, da forma dada, para a forma canônica.

Tomando como exemplos as equações abaixo, podemos ver que a equação 6.1 está na forma canônica. Por outro lado, as equações 6.3 e 6.4 precisam de um certo tratamento (simplificação) para ficarem na forma canônica.

$$5x^2 + 3x + 7 = 0 \tag{6.1}$$

$$y^2 - 7y + 1 = 0 \tag{6.2}$$

$$x^2 + x = 5 \tag{6.3}$$

$$(x + 1) * (x + 2) = 7 + x \tag{6.4}$$

Desse modo, o *segundo passo* é simplificar a equação, dependendo da sua forma inicial. A simplificação, no entanto, pode ser uma tarefa muito complexa. Por exemplo, simplificar a equação 6.4 é bem mais complicado que simplificar a equação 6.3. Deixamos então os detalhes da simplificação para adiante. Por enquanto, vamos continuar a estabelecer os passos gerais para achar as raízes de uma equação.

O *terceiro passo* é calcular o discriminante da equação. O discriminante, indicado por Δ, é dado pela fórmula:

$$\Delta = b^2 - 4ac$$

O cálculo do discriminante nos permite determinar quantas raízes reais a equação pode ter. Este é o *quarto passo*. Caso $\Delta > 0$, a equação tem duas raízes reais. Se

$\Delta = 0$, então a equação só apresenta uma raíz, ou mais rigorosamente, apresenta duas raízes reais iguais. E, por fim, se $\Delta < 0$, então a equação não apresenta raízes reais (mas nós não estamos interessados aqui em raízes complexas).

O quinto e *último passo* consiste em calcular as raízes. Como percebemos, este passo depende da conclusão que chegamos no passo anterior. Se a equação tem duas raízes, x' e x'', dadas pelas fórmulas:

$$x' = \frac{-b + \sqrt{\Delta}}{2a}$$

$$x'' = \frac{-b - \sqrt{\Delta}}{2a}$$

Se a equação tem duas raízes reais iguais, então x' e x'' são dados por:

$$x' = x'' = -\frac{b}{2a}$$

E finalmente, se a equação não apresenta raízes reais, não temos nada a calcular.

Resumindo, temos os seguintes passos para resolver uma equação do segundo grau:

1. verificar se equação está na forma canônica;
2. simplificar se necessário;
3. calcular o Δ;
4. verificar quantas raízes reais existem;
5. calcular as raízes aplicando a fórmula adequada de acordo com o número de raízes, ou dizer que as raízes são complexas.

Nas seções seguintes, veremos alguns dos passos um pouco mais detalhadamente. Os passos não detalhados já estão bastante claros pelo que já foi dito.

Verificar se a equação é canônica

Verificar se a forma de uma equação é canônica consiste em verificar se a equação apresenta todos os termos de uma equação canônica. Os termos são $\pm ax^2$, $\pm bx$, e $\pm c$ e o zero. Por exemplo as equações 6.1, 6.2 e 6.5 (mesmo apresentada de forma não convencional) estão na forma canônica.

$$0 = z - 12z^2 - 3 \tag{6.5}$$

Uma estratégia para reconhecermos se uma equação é canônica consiste em simplesmente procurar pelos termos canônicos, sem se importar com a ordem deles. A única observação é que os termos e o zero estejam em lados opostos da igualdade. Não importa qual lado.

Em resumo, os passos para verificar se a equação é canônica são:

1. procurar pelos termos canônicos;
2. verificar se o zero está isolado em um dos lados da igualdade.

Simplificar

Simplificar é uma tarefa de transformação. A equação inicial é (ou deseja-se que seja) transformada em uma forma mais simples, de preferência a forma canônica. Simplificar pode ser visto como uma busca em um espaço de estados, onde cada estado é uma equação e onde o estado a ser alcançado é a equação que está na forma canônica. Para passar de um estado a outro, devemos utilizar diversas regras de simplificação tais como: o termo $x * x$ é simplificado para x^2. Pode-se usar também propriedades matemáticas como a distributividade, associatividade, eliminação de parênteses e o fato de que podemos somar (subtrair, dividir, etc.) uma mesma quantidade aos dois lados de uma igualdade. Devemos utilizar com cautela estas propriedades, no entanto, tendo sempre em mente que o objetivo é tornar a equação mais simples inteligível.

Dependendo do grau de conhecimento, pode-se não chegar à forma canônica de uma dada equação. Por exemplo, caso não se saiba como simplificar o termo

$$\frac{x^5}{x * x^2}$$

muito provavelmente não encontraremos a forma canônica da equação 6.6.

$$\frac{x^5}{x * x^2} + (x - 12) = 3 \tag{6.6}$$

Simplificar não é uma tarefa trivial. Achamos mesmo que é a parte crucial do processo de resolução de equações do segundo grau. Mas para aprendermos a resolver equações do segundo grau, não necessariamente devemos saber simplificar. Podemos ensinar alguém a resolver equações sem ensiná-lo a simplificar. Jamais poderá se dizer tal pessoa não sabe resolver uma equação do segundo grau. O único inconveniente é que ele só saberá resolver equações na forma canônica.

6.3.2 Identificação das Tarefas e Inferências

Baseado na seção anterior, usaremos a CML[1] para descrever "mais formalmente" o método de resolução do problema de *determinar as raízes* de uma equação do segundo grau em termos das seguintes tarefas:

tarefa resolver-equação-do-segundo-grau.
entrada: uma equação do segundo grau.
saida: as raízes da equação ou a indicação de que não existem raízes reais.
objetivo: calcular as raízes de uma equação do segundo grau.
subtarefas:
verificar-se-canônica,
simplificar,
calcular Δ,
verificar-quantidade-de-raizes,
calcular-raizes.

tarefa verificar-se-canônica.
entrada: uma equação do segundo grau.
saida: sim ou não.

[1]Veja seção 2.8.

objetivo: decidir se uma equação está na forma canônica ou não.

tarefa simplificar.

entrada: uma equação do segundo grau.

saida: uma equação na forma canônica.

objetivo: transformar a equação de entrada para a forma canônica.

Não vamos aqui apresentar todas as tarefas. Basta saber que cada subtarefa da tarefa principal deve ser expandida em outras subtarefas ou em um conjunto de inferências. Como já foi dito, as inferências são os passos básicos do processo de raciocínio. Determinar o conjunto de inferências depende bastante da tecnologia computacional disponível ao engenheiro do conhecimento.

6.3.3 Ontologia

Estando terminada a fase de identificação das tarefas e inferências deve-se construir a ontologia. Para isso, tomam-se como base os conceitos, ou idéias, principais do discurso do especialista.

A idéia de *equação* é primordial no estudo de resolução de equações do segundo grau. A partir das seções anteriores, muitas outras idéias surgem (termos canônicos, regras de simplificação, fórmula de Bháskara) e também têm um papel importante no processo de resolução de equações. Nesta seção, visamos definir o conjunto de idéias, e relações entre estas, utilizadas na resolução de equações do segundo grau. Apresentamos a seguir uma lista de definições:

Equação do segundo grau. Fórmula matemática da forma $P(x) = 0$, onde $P(x)$ é um polinômio de grau 2.

Equação Canônica. *Equação* cujo polinômio $P(x)$ está na forma:

$$ax^2 + bx + c = 0$$

Raízes. Valores de x que tornam $P(x) = 0$.

Regras de Simplificação. Regras que permitem transformar termos de uma equação em termos mais simples.

Constantes. Constantes do polinômio $P(x)$ de uma equação canônica.

Fórmula de Bháskara. Usada para calcular as raízes de uma equação canônica. A fórmula de Bháskara é dada por:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

Discriminante. Utilizado para se decidir quantas raízes tem a equação. É dado pela fórmula:

$$\Delta = b^2 - 4ac$$

Estas definições são o principio da construção de nossa ontologia. Por exemplo poderíamos começar por definir o conceito equação.

conceito *equação.*
parte 1: expressão.
parte 2: Zero.
separador: sinal de igualdade '='.
fim conceito

Nesta definição declaramos que uma equação tem duas partes, sendo a primeira uma expressão e a segunda um zero, separado por um sinal de igualdade. Baseado nas idéia do domínio apresentadas acima, devemos continuar declarando os elementos da ontologia.

6.3.4 Instâncias de Ontologia

Este é o conhecimento específico do domínio formado pelas instâncias dos conceitos e relações previamente definidos na Ontologia. Para o agente `EqCanonicaReal` as

instâncias de ontologia são constituídas principalmente das regras (para simplificação de expressões) e pela fórmula de cálculo de raízes (fórmula de Bháskara). Um exemplo de regra é:

$$x * x \Rightarrow x^2$$

A fórmula de Bháskara é dada pela seguinte formula:

$$x = \frac{-b \pm \sqrt{\Delta}}{2a}$$

6.4 Criação das Bases de Conhecimento

Como visto na seção 3.4, a base de conhecimento do Agente Tutor MATHEMA é constituída de partes específicas para cada módulo do *raciocinador*. Assim, temos três grandes bases:

- *base do especialista:* armazena o conhecimento de resolução de problemas, isto é, ontologia e instâncias.
- *base do tutor:* armazena o conhecimento pedagógico, isto é, *curriculum*, problemas e unidades pedagógicas.
- *modelo do aprendiz:* armazena informações sobre o aprendiz com o objetivo de melhor adaptar o processo de aprendizagem de acordo com as características individuais de cada aluno.

Nessa etapa, construiremos a base de conhecimento dos agentes a partir dos modelos de perícia de cada subdomínio. Para isso, deve-se traduzir o modelo de perícia, utilizando o AM, para se obter uma base de conhecimento (arquivo BC). Como mostrado na seção 5.3 essa é uma das partes que vão formar os agentes.

6.5 Criação dos Agentes

A criação dos agentes é feita com o uso do AM, como visto na seção 5.2.6. O GERENCIADOR DE AGENTES, a partir da base de conhecimento (arquivo BC), vai gerar

todos os outros módulos necessários para o funcionamento do interpretador de tarefas. Devemos ainda editar os detalhes do arquivo FUN (veja seção 5.2.6).

O resultado final pode ser visto na figura 6.6, onde apresentamos a organização final dos módulos criados. Os módulos sem cor, representam os módulos gerados a partir do MP, isto é, dependentes de domínio. O módulo com cantos arredondados é o módulo de métodos de inferências (arquivo MET), que representa uma biblioteca em Prolog para dar suporte à execução do interpretador de tarefas. No topo da figura vê-se o módulo agente, criado automaticamente a partir das informações contidas no MP. Na figura, esse módulo aparece em cor branca, indicando que é um módulo dependente do domínio. Isto se deve ao fato de que, apesar de o módulo principal do agente ser criado genericamente (tendo por exemplo a capacidade de cooperar ou interagir com o ambiente de manutenção), há informações tais como a identificação e lista de habilidades que são dependentes do domínio e, de fato, são extraídas do MP no momento da criação do agente. Os demais módulos representam a estrutura do interpretador de tarefas do agente. Note que nessa figura representamos apenas o sistema tutor e sua relação com o módulo principal do agente.

6.6 Modelos dos Agentes

Para cada agente identificado durante a organização da visão externa do domínio é criada uma instância do modelo de agentes MATHEMA (ver seção 3.4.1). Uma instância do modelo de agente MATHEMA apresenta as características particulares de um agente. No entanto, usando o AM, não precisamos nos preocupar com os modelos de agentes, uma vez que a criação dos agentes é feita automaticamente a partir do MP do subdomínio. O modelo de agente funciona então, como uma documentação para o projetista do STI-MATHEMA.

Para fins ilustrativos, apresentamos na figura 6.7 a instância de modelo de agente criado a partir do domínio d_{11}. As suas habilidades são mostradas na figura 6.8.

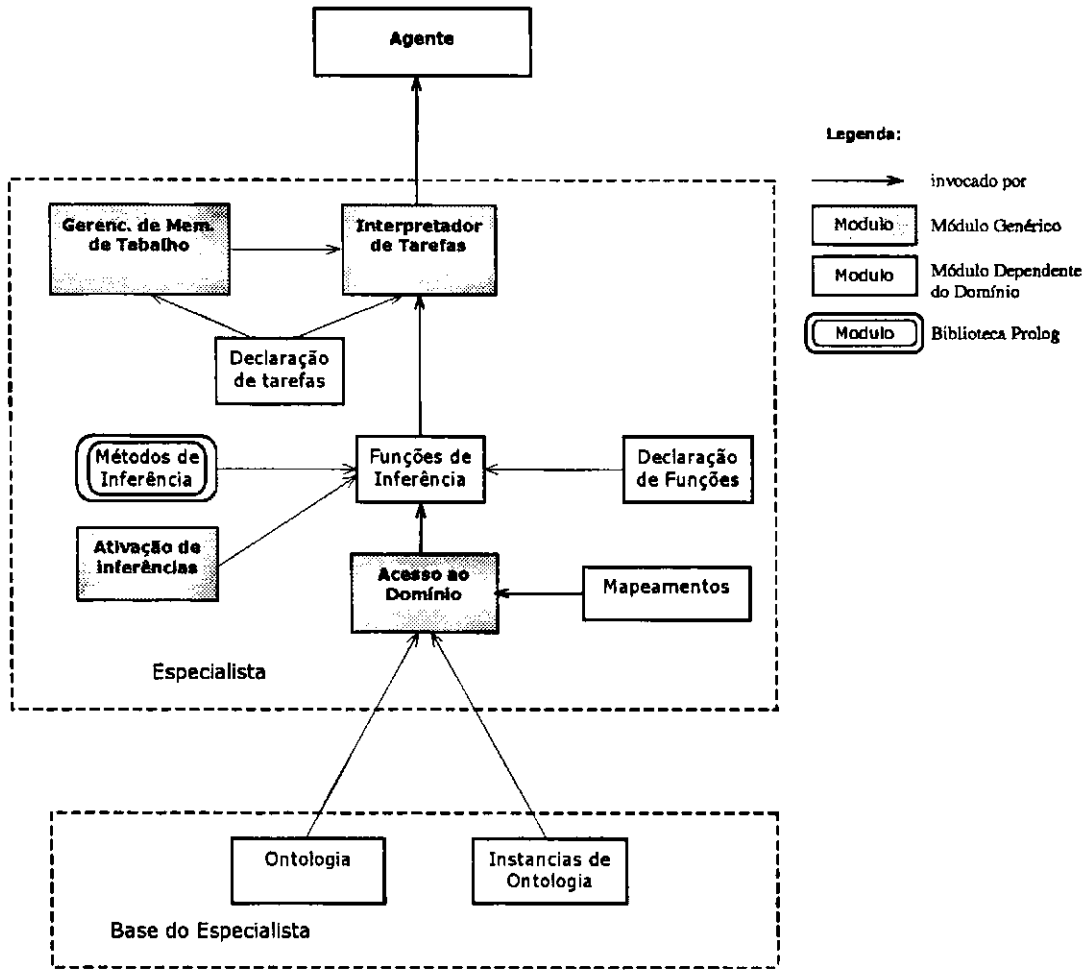


Figura 6.6: Organização dos módulos do Agente MATHEMA

modelo de agente		
autoconhecimento:	identificação	d_{11}
	apelidos:	`EqCanonica`
	lista de habilidades:	habilidade 1
		habilidade 2
		habilidade n
sistema tutor:	nome do modelo de perícia	
conhecimento social:	<nenhum>	
mundo externo:	<gerado automaticamente pelo AM.>	

Figura 6.7: Instância de Modelo de Agente para o agente `EqCanonica`

habilidades do agente EqCanonica		
habilidade 1		
tipo:	resolução	
identificação:	Bháskara	
representações:	entrada:	(num a, num b, num c)
	saída:	(num raízes, num x1, num x2)
habilidade 2		
tipo:	resolução	
identificação:	discriminante	
representações:	entrada:	(num a, num b, num c)
	saída:	(num discriminante)

Figura 6.8: Habilidades do agente `EqCanonica`

6.7 Resumo da utilização do AM

No protótipo construído para o AM, só utilizamos a ferramenta para as etapas de Modelagem do Conhecimento, Geração das Bases de Conhecimento e Criação dos Agentes. As etapas de Organização da Visão Externa e Estruturação Pedagógica (visão interna) não são suportadas pelo AM. Um resumo da utilização do AM é mostrado abaixo:

1. *criação dos modelos de perícia:* os modelos de perícia são criados na etapa de modelagem de conhecimento. Nesta etapa o AM é utilizado para:
   - criação e modificação dos modelos de perícia;
   - verificação dos modelos de perícia construídos;
   - criação semi-automática das bases de conhecimento;
2. *criação dos agentes:* o AM é utilizado para gerar o código Prolog para os Agentes Tutores de cada subdomínio.
3. *manutenção:* O AM é utilizado para a modificação da SATA em geral, isto é, criação de novos agentes, retirada ou atualização da base de conhecimento de agentes existentes.

6.8 Discussão

Apesar de não termos implementado um verificador com muitas funcionalidades, na prática, ele nos alertou para um grande número de erros introduzidos durante o processo de construção de modelos de perícia. Com um verificador de modelos de perícia, pode-se garantir que a base de conhecimento gerada será coerente. Entretanto, não se pode garantir que o comportamento do agente vai ser de acordo com o comportamento desejado, já que isso é indecidível.

O primeiro ponto a ser revisto em nosso protótipo, rumo a uma implementação robusta é a escolha de uma linguagem de programação mais portável, tal como C++ ou JAVA. No estágio atual, entanto, o protótipo já mostrou pontos importantes, tais

como, a ajuda provida pelo assistente de sintaxe e o papel do verificador semântico. A nossa implementação mostrou também o bom funcionamento do tradutor.

Capítulo 7

Conclusão e Trabalhos Futuros

Este trabalho é uma contribuição para o desenvolvimento do projeto MATHEMA. Acreditamos ter atingido os objetivos estabelecidos na introdução desta dissertação. É claro, este é apenas uma pequena parte de um grande projeto, que precisa ser continuado para que o MATHEMA venha a se tornar uma verdadeira metodologia de construção de Sistemas Tutores Inteligentes. Teceremos agora, alguns comentários sobre as contribuições desta dissertação e quais as direções futuras para as pesquisas do projeto MATHEMA.

7.1 Principais Contribuições

A definição inicial do MATHEMA [Cos97] preocupou-se com os aspectos gerais para a construção de um modelo genérico para STI em uma abordagem multi-agentes. Além de uma arquitetura geral para STI, MATHEMA propõe uma arquitetura para os agentes que compõem a sociedade de agentes tutores, além de delinear alguns protocolos de interação entre esses agentes.

Esta dissertação pretendeu complementar o trabalho original definido em [Cos97], em seus aspectos de Aquisição e Manutenção de Conhecimento. Podemos apontar como uma contribuição o detalhamento do Ciclo de Aquisição de Conhecimento MATHEMA e o detalhamento da arquitetura do Agente MATHEMA para esses aspectos de Aquisição de Conhecimento.

Outra contribuição é a especificação do Ambiente de Manutenção, juntamente com os Protocolos e Mensagens de Manutenção. Essa ferramenta tem o papel de auxiliar a construção dos agentes da SATA e a alteração de suas bases de conhecimento. Essa ferramenta pode ser usada não somente para criar agentes tutores, mas o seu editor de Modelos Conceituais pode ser usado, por exemplo, para criar modelos de conhecimento que serão usados em outros sistemas baseados em conhecimento, não necessariamente sistemas tutores.

7.2 Conclusões gerais

A arquitetura do Agente MATHEMA apresenta um elemento chamado ESPECIALISTA responsável pela resolução de problemas no agente tutor. O seu propósito é fornecer soluções para os problemas do domínio. Escolhemos para o desenvolvimento desse módulo, uma abordagem de aquisição de conhecimento baseada em modelos. Essa abordagem é hoje amplamente aceita, pela comunidade de AC. Tal abordagem proporciona o desenvolvimento de sistemas com maior poder de explicação e também com maior capacidade de manutenção dos sistemas construídos [BC95].

Outro ponto importante é o conhecimento pedagógico. Não enfatizamos, nesta dissertação, este tipo de conhecimento. Podemos porém, construi-lo, utilizando-se uma abordagem semelhante àquela usada na construção do módulo ESPECIALISTA. Para tanto, termos tais como *curriculum*, *unidade pedagógica*, *unidade de conhecimento*, usados na Visão Interna, devem ser formalizados em uma Linguagem Pedagógica. Em outras palavras, uma vez que seja definida a linguagem formal para a Visão Interna de um subdomínio (identificado na primeira etapa do Ciclo de AC do MATHEMA), podemos utilizar um editor para a construção do conhecimento Pedagógico, ferramentas de Verificação e uma Biblioteca de Conhecimento Pedagógico.

7.3 Trabalhos Relacionados

O Laboratório de Sistemas Inteligentes[1] (ISL) da Universidade de Michigan (EUA), vem investigando a construção de STI a partir de SBC desenvolvidos segundo a metodologia Generic Tasks [ESS98]. As principais diferenças do nosso trabalho em relação ao trabalho desse grupo são:

- estamos utilizando a metodologia KADS, que é uma metodologia de granularidade mais fina que Generic Tasks.
- estamos trabalhando com a noção de Sociedade de Agentes Tutores Inteligentes enquanto que o está mais preocupado com os problemas de construção de Sistema Tutor monolítico.

Outra pesquisa tem sido no sentido de se definir uma metodologia para a construção de STI. Mizoguchi, Sinitsa e Ikeda [MS96] apresentam resultados interessantes neste sentido. Os resultados dessas pesquisas ajudarão na definição de ferramentas de autoria para STI. O objetivo do MATHEMA a longo prazo é também desenvolver uma metodologia de construção de STI e ferramentas de autoria para STI-MATHEMA [CdS99].

7.4 Trabalhos futuros

MATHEMA deve agora caminhar na direção de definir uma linguagem unificada para a construção de suas bases de conhecimento. Isso tornará possível a criação de uma ferramenta mais ampla e portanto mais útil para quem queira construir um STI-MATHEMA.

Tomando como base esta dissertação, o próximo passo na direção apontada é estender a Linguagem de Modelagem Conceitual para englobar uma Linguagem Pedagógica (da qual falamos na seção anterior).

[1] Do original em inglês, Intelligent Systems Laboratory

Outros trabalhos que indicamos para a pesquisa MATHEMA relativos à AC, são resumidos a seguir:

- Investigar mais profundamente o processo de Verificação e Validação dos Modelos de Conhecimento. A pesquisa nesta área trará contribuições não só para o MATHEMA como para toda a comunidade de AC.

- Desenvolver um editor gráfico para construção dos Modelos de Conhecimento. O atual editor, pode servir de base para a construção de um tal editor gráfico.

- Adotar um formalismo de representação que seja reconhecido internacionalmente na comunidade científica, como o KIF [GF92]; e neste mesmo sentido, adotar uma linguagem de comunicação para os agentes, tal como o KQML [LF93].

Apêndice A

Ambiente de Manutenção v.02

Neste apêndice mostramos as janelas de interface do protótipo desenvolvido para o ambiente de mantutenção. Atualmente o protótipo está na versão 0.2.

A.1 Módulo Principal

Na tela principal do Ambiente de Manutenção, encontram-se os controles[1] que dão acesso às funcionalidades do Ambiente de Manutenção. A tela principal é vista na figura A.1 (pág. 108). Os controles da tela principal são:

- *arquivo a ser usado:* este controle indica o nome do arquivo de trabalho atual.
- *procurar:* o botão procurar dá acesso a uma caixa de diálogo que permite navegar em uma árvore de diretórios em busca de um arquivo.
- *sobre:* informações sobre o protótipo (data de criação, contatos, etc).
- *editar:* dá acesso ao editor de modelos de conhecimento.
- *traduzir:* inicia a tradução de um arquivo indicado no controle *arquivo a ser usado.*

[1] Um controle em uma janela de interface é um botão, um campo de dados, uma barra de rolagem, etc.

- *verificar:* inicia a verificação de um arquivo indicado no controle *arquivo a ser usado.*

- *mapeamentos:* dá acesso a uma caixa de diálogos que permite a edição dos mapeamentos entre termos de inferência e termos do domínio.

- *gerenciador de agentes:* é o último botão da tela principal. Dá acesso a uma caixa de diálogo que permite realizar as operações sobre os agentes (criação, alteração, etc.).

- *linha de aviso:* o último controle da tela principal é uma linha de aviso que informa ao usuário se a ferramenta está ocupada ou pronta para processar comandos, ou ainda se a última operação provocou um erro.

A.2 Módulo de Edição

O módulo de edição permite a criação e modificação de modelos de domínio utilizando-se a linguagem CML. Para se começar uma seção de edição deve-se: (i) escolher um arquivo digitando seu nome no controle *arquivo a ser usado* ou utilizando o botão *procurar*, (ii) pressionar o botão *editor* na tela principal do Ambiente de Manutenção. Ao iniciar-se uma sessão de edição, a tela inicial do Ambiente de Manutenção é disponibilizada ao usuário a área de edição, como vista na figura A.3 (pág. 109).

A.3 Gerenciador de Agentes

O gerenciador de agentes é o módulo que permite a realização das operações de manutenção sobre os agentes da SATA. Para acessar esta módulo deve-se antes editar um arquivo, traduzi-lo, verifica-lo e editar os seus mapeamentos (entre termos de infernçias e termos de ontologia, veja seo 5.2.5). Só então é que se pode invocar este módulo. A interface deste módulo é apresentada na figura A.2 (pág. 108), onde encontramos um botão para cada operação de manutenção:

- *criação:* permite a criação de agentes e inserção destes na SATA.

- *retirada* permite a retirada de agentes da SATA.
- *alteração:* permite iniciar uma seção de alteração dos conhecimentos de um agente.

Nesta janela observamos ainda o controle denominado *agente ID*, que é a identificação do agente que irá sofrer a operação de manutenção. Na parte inferior da janela, há ainda um controle que indica o estado atual do gerenciador de agentes.

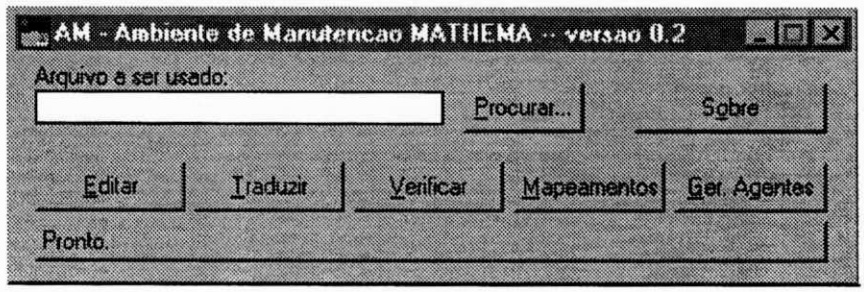


Figura A.1: Tela principal do Ambiente de Manutenção

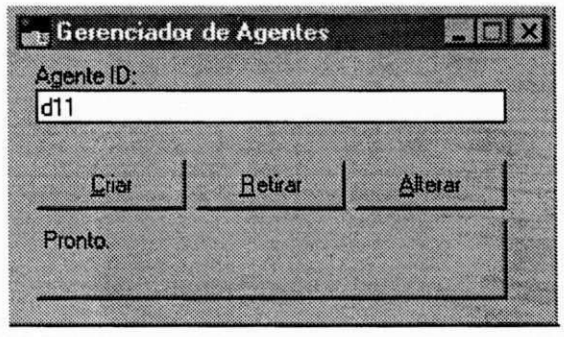


Figura A.2: Caixa de Diálogo do Genrenciador de Agentes

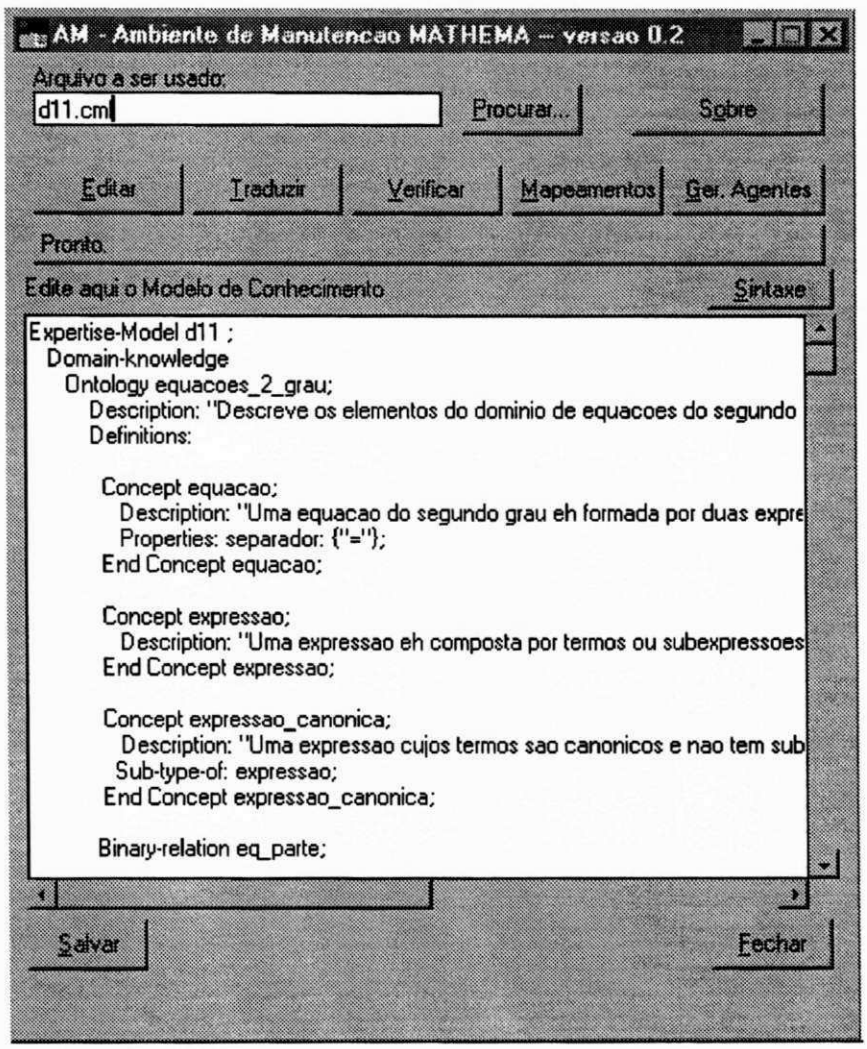


Figura A.3: Tela do Ambiente de Manutenção durante uma seção de edição

Referências Bibliográficas

[Aam94] Agnar Aamodt. Knowledge acquisition and learning by experience: The role of case specific knowledge. Also available at http://www.ifi.ntnu/grupper/ai/pubs.html, 1994.

[ABB+92] Agnar Aamodt, Bert Bredeweg, Joost Breuker, Cuno Duursma, Christiane Löckenhoff, Klas Orsvarn, Jan Top André Valente, and Walter Van de Velde. The CommonKADS library. Relatório Técnico KADS-II/T1.3/VUB/TR/005/1.0, ESPRIT Project P5248 KADS-II, May 1992.

[Abe93] Manfred Aben. CommonKADS inferences. Relatório de Técnico KADS-II/M2/TR/Uva/041/1.0, University of Amsterdam, Amsterdan, Holanda, 1993.

[ACKP95] J. Anderson, A. Corbett, K. Koedinger, and Pelletier. Cognitive tutors: lessons learned. *The Journal of the Learning Sciences*, 4(2):167–207, 1995.

[AGKS92] Nathalie Aussenac-Gilles, Jean-Paul Krivine, and Jean Sallantin. Editorial: L'acquisition des connaissances pour les systèmes à base de connaissances. *Revue d'Intelligence Artificielle*, 6(1-2/1992):7–18, 1992. Special Issue: Acquisition des Conaissances.

[AH93] D. Allemang and Gertjan Van Heijst. Generic tasks in KEW. In J.-G. Ganascia N. Aussenac, G. Boy, B. Gaines, M. Linster and Y. Kodratoff, editors, *Proceedings of the 7th European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW '93)*, volume 723 of *LNAI*, pages 138–158, Toulouse and Caylus, France, September 1993. Springer.

[Ale94] Cláudio Reginaldo Alexandre. *Aquisição de Conhecimentos Contemplando os Aspectos Sintático, Semântico, de Generalização e Custo.* Dissertação de mestrado, Universidade Federal da Paraíba, Campina Grande, PB, 1994.

[AWS94] J. M. Akkermans, Bob J. Wielinga, and A. Th. Schreiber. Steps in constructing Problem-Solving Methods. In Brian R. Gaines and M. A. Musen, editors, *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge Based Systems Workshop (KAW '94)*, volume 2, pages 29–1–29–21, Alberta, Canadá, January 1994. SRGD Publications.

[AWT92] Anjo Anjewierden, Jan Wielemaker, and Catherine Toussant. Shelley: computer-aided knowledge engineering. *Knowledge Acquisition*, 4(1), march 1992. Special Issue: The KADS approach to knowledge engineering.

[BB87] J. Boose and J. Bradshaw. Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. *International Journal of Man-Machine Studies*, 26:3–28, 1987.

[BB98] Joost Breuker and Alexander Boer. So you want to validate your PSMs? In *Proceedings of Eleventh Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Voyager Inn, Banff, Alberta, Canada, 18th to 23rd April 1998. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html.

[BC95] Sofie Billet-Coat. *Apport a l'Acquisition Intercative de Connassainces Contextuelles.* Tese de doutorado, Université de Montpellier II, Decembre 1995.

[Buc92] Paulo Bucci. *Matemática: volume único.* Moderna, São Paulo, primeira edition, 1992.

[BW89] Joost Breuker and Bob Wielinga. Models of expertise in knowledge acquisition. In G. Guida and C. Tasso, editors, *Topics in Expert System Design: methodology and tools*, pages 265–295. North-Holland, 1989.

[BW98] Frances M. T. Brazier and Nick J. E. Wijngaards. A purpose driven method for the comparison of modeling frameworks. In *Proceedings of Eleventh Workshop on Knowledge Acquisition, Modeling and Management (KAW'98)*, Voyager Inn, Banff, Alberta, Canada, 18th to 23rd April 1998. http://ksi.cpsc.ucalgary.ca/KAW/KAW98/KAW98Proc.html.

[Car70] J. R. Carbonell. AI in CAI: Artificial intelligence approach to computer assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):190–202, 1970.

[CD96] Olivier Corby and Rose Dieng. CoKACE: A Centaur-based environment for commonKADS conceptual modeling language. In W. Wahlster, editor, *Proceedings of 12th European Conference on Artificial Inteligence (ECAI'96)*. John Willey & Sons, 1996.

[CdS99] Evandro de Barros Costa and Josenildo Costa da Silva. Towards an authoring systems for a society of tutoring agents. In *Proceedings of IA-Ed'99*, Le Mans, France, july 1999.

[Cha86] B. Chandrasekaran. Generic Tasks in knowledge-based reasoning: high-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, fall 1986.

[Cla85] W. Clancey. Heuristic classification. *Artificial Intelligence*, 27:289–350, 1985. Also in Knowledge-based Problem Solving, J. S. Kowalik (ed), Prentice-Hall, Inc., 1985.

[Cla86] Willian J. Clancey. From GUIDON to NEOMYCIN and HERACLES in twenty short lessons: ORN final report 1979–1985. *AI Magazine*, 7(3), august 1986.

[Cos97] Evandro de Barros Costa. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multi-Agentes.* Tese de doutorado, Universidade Federal da Paraíba, COPELE, dezembro 1997.

[Cou99] Luciano Reis Coutinho. *A Modelagem do Estudante em um Ambiente Adaptativo de Aprendizagem Baseada em Resolução de Problemas.* Dissertação de mestrado, Universidade Federal da Paraíba, COPIN, 1999.

[CP96] Evandro de Barros Costa and Angelo Perkusich. Modelling the cooperative interactions in a teaching/learning situation. In C. Frasson, G. Gauthier, and A. Lesgold, editors, *Proceedings of Third International Conference on Intelligent Tutoring Systems (ITS'96)*, volume 1086 of *Lecture Notes in Computer Science*, pages 168–176, Montreal, Canada, june 1996. Springer.

[CPF98] Evandro de Barros Costa, Angelo Perkusich, and Edilson Ferneda. From a tridimensional view of domain knowledge to multi-agent tutoring system. In F.M. de Oliveira, editor, *Proceedings of 14th Brazilian Symposium on Artificial Intelligence*, volume 1515 of *Lecture Notes in Artificial Intelligence*, pages 61–72, Porto Alegre, Brazil, November 1998. Springer.

[CTF97] Evandro de Barros Costa, Luciênio de M. Teixeira, and Edilson Ferneda. Um sistema tutor inteligente multi-agentes em harmonia musical. In *Anais do VIII Simp sio Brasileiro de Inform tica na Educa o - SBIE'97*, volume I, S o Jos dos Campos (SP), novembro 1997.

[Dav76] R. Davis. *Aplications of meta-level knowledge to the construction, maintenance and use of large knowledge bases.* PhD thesis, Stanford University, 1976.

[Die90] Rose Dieng. Méthodes et outils d'acquisition des connaissances. Rapport de Recherche 1319, INRIA, Sofia-Antipolis, 1990.

[DL89] J. Diederich and M. Linster. Knowledge-based knowledge acquisition. In

G. Guida and C. Tasso, editors, *Topics in Expert System Design methodology and tools*, pages 323–350. North-Holland, New York, 1989.

[DMW93] John Domingue, Enrico Motta, and Stuart Watt. The emerging VITAL workbench. In *Proceedings of EKAW'93*, volume 723 of *Lecture Notes in Artificial Intelligence*, pages 320–339, Toulouse, France, september 1993. Springer-Verlag.

[DRM87] J. Diederich, I. Ruhmann, and M. May. Kriton: A knowledge acquisition tool for expert systems. *Intenational Journal for Man-Machine Studies*, 26(1):29–40, 1987.

[Esh88] L. Eshelman. MOLE: a knowledge-acquisition tool for cover-and-differentiate systems. In S. Marcus, editor, *Atomating Knowledge Acquisition for Expert Systems*, pages 225–256. Kluwer Academic, 1988.

[ESS98] Eman El-Sheikh and Jon Sticklen. A framework for developing intelligent tutoring systems incorporationg reusability. In *Proceedings of 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 1415 of *Lecture Notes in Artificial Intelligence*, Benicassim, Catellon, Spain, 1998. Springer-Verlag.

[FdMdR98] Ricardo de A. Falbo, Crediné Silva de Menezes, and Ana Regina C. da Rocha. A systematic approach for bilding ontologies. In *Proceedings of the 6th Ibero-American Conference on AI on Progress in Artificial Intelligence (IBERAMIA-98)*, volume 1484 of *LNAI*, pages 349–360. Springer, 5th to 9th October 1998.

[FFR97] R. Fikes, A. Farquhar, and J. Rice. Tools for assembling modular ontologies in ontolingua. Technical Report KSL-97-03, Knowledge Systems Laboratory, Stanford University, California, USA, April 1997. Also available at ftp://ftp-ksl.stanford.edu/pub/KSL_Reports/KSL-97-03.ps.

[Fir88] Morris W. Firebaugh. *Artificial Intelligence: A knowledge based approach.* PSW-Kent Publishing Co, Boston, 1988.

[Fre97] Reva Freedman. Degrees of mixed-initiative interaction in an intelligent tutoring system. In *Proceedings of AAAI 1997 Spring Symposium on Computational Models for Mixed-Initiative Interaction*, 1997. Also avaliable at http://www.csam.iit.edu/čirsim/documents/rfmis97.ps.

[Gai93] Brian R. Gaines. Modeling and extending expertise. In J.-G. Ganascia N. Aussenac, G. Boy, B. Gaines, M. Linster and Y. Kodratoff, editors, *Proceedings of 7th European Workshop on Knowledge Acquisition for Knoledge-Based Systems EKAW'93*, volume 723 of *Lecture Notes in Artificial Intelligence*, pages 1–22, Toulouse, France, september 1993. Springer-Verlag.

[GF92] Michael R. Genesereth and Richard E. Fikes. Knowledge interchange format (version 3.0): reference manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, California (USA), june 1992.

[GM96] Yolanda Gil and Eric Melz. Explicit representations of problem-solving stratigies to support knowledge acquisition. In *Proceedings of Thirteen National Conference on Artificial Inteligence (AAAI'96)*, Portland, OR, 4th to 8th august 1996.

[GS94] Sabine Geldof and Aurelien Slodzian. From verification to modeling guidelines. In Luc Steels, Guus Schreiber, and Walter Van de Velde, editors, *8th European Knowledge Acquisition Workshop (EKAW'94), A Future for Knowledge Acquisition*, volume 867 of *Lecture Notes in Artificial Intelligence*, pages 226–243, Hoegaarden, Belgium, september 1994. Springer-Verlag.

[GSI97] GSIA. *Manual do ExpertSINTA*. Grupo de Sistemas Inteligentes Aplicados. Lab. de IA, Univ. Federal do Ceará, 1997.

[GT97] Yolanda Gil and Marcelo Tallis. A script-based approach to modifying knowledge bases. In *Proceedings of the Fouteenth National Conference on Artificial Intelligence (AAAI-97)*, Providence, RI, 27th to 31th july 1997.

[Jat98] Alessandro Jatobá. *Um Arcabouço para Coopera o em uma Sociedade de Agentes Tutores Artificiais.* Dissertação de mestrado, Universidade Federal da Paraíba, COPIN, maio 1998.

[Kau93] Karine Kausse. Heuristic control knowledge. In J.-G. Ganascia N. Aussenac, G. Boy, B. Gaines, M. Linster and Y. Kodratoff, editors, *Proceedings of the 7th European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW '93)*, volume 723 of *LNAI*, pages 183–199, Toulouse and Caylus, France, September 1993. Springer-Verlag.

[Kea87] Greg Kearsley, editor. *Artificial Intelligence and Instruction.* Addison-Wesley Publishing Company, CA (USA), 1987.

[Kel55] G. A. Kelly. *The psichology of Personal Constructs.* New York, 1955.

[Kin94] John Kingston. Linking knowledge acquisition with commonkads knowledge representation. In *BCS SGES Expert Systems'94 Conference*, St John's College, Cambridge, december 1994. Also as Technical Report no. AIAI-TR-156 of Artificial Intelligence Aplication Institute. Available at http://www.aiai.ed.ac.uk/jkk/kadspubs.html.

[Lab95] Sofiane Labidi. *Ingénierie da la Connaissance dans le Cadre de Projets Multi-Experts: Méthode, Techniques et Outil.* Thèse de doctorat, L'Université de Nice–Sophia Antipolis, juin 1995.

[LF93] Yannis Labrou and Tim Finin. A proposal for a new kqml specification. Technical Report TR CS-97-03, Computer Science and Eletrical Engineering Department (CSEE), University of Maryland Baltimore County (UMBC), 1993.

[LF98] Sofiane Labidi and Jeane S. Ferreira. Technology-assisted instruction applied to cooperative learning: the SHIECC project. In *Proceedings of the IEEE International Conference Frontiers in Education (FIE'98)*, Tempe, Arizona, 4th to 7th november 1998.

[Lin93] Marc Linster. A review of Sisyphus 91 & 92: models of problem solving knowledge. In J.-G. Ganascia N. Aussenac, G. Boy, B. Gaines, M. Linster and Y. Kodratoff, editors, *Proceedings of the 7th European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW '93)*, volume 723 of *LNAI*, pages 159–182, Toulouse and Caylus, France, September 1993. Springer-Verlag.

[MCCS94] V. Maniezzo, A. Carbonaro, G. Casadei, and P. Salomoni. A co-operative teaching environment for euclidean geometry. Technical Report 94/1, Cesena Laboratory for Computer Science, Cesena Italy, 1994.

[McD88] J. McDermot. Preliminary steps toward a taxonomy of problem-solving methods. In S. Marcus, editor, *Atomating Knowledge Acquisition for Expert Systems*, pages 225–256. Kluwer Academic, 1988.

[MCM83] R. Mikalsky, J. Carbonell, and T. Michell. *Machine Learning: an artificial intelligence approach.* Morgan Kaufmann, Palo Alto, CA, 1983.

[MFCS87] M. A. Musen, L. M. Fagan, D. M. Combs, and E. H. Shortlife. Use of a domain model to drive an interactive knowledge-editing tool. *International Journal of Machine Studies*, 26:105–121, 1987.

[MM89] S. Marcus and J. McDermott. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence*, 39(1):1–37, may 1989.

[MS96] R. Mizoguchi and K. Sinitsa. Task ontology design for intelligent educational/training systems. In *Proceedings of Workshop on Architectures and Methods for Designing Cost-Effective and Reusable ITSs (ITS'96)*, pages 1–21, Montreal, 1996.

[Mus89] Mark A. Musen. Automated support for building and extending expert models. *Machine Learning*, 4:347–376, 1989.

[Mus92] Mark A. Musen. Overcoming the limitations of role-limiting methods. *Knowledge Acquisition*, 4(2):165–170, 1992. Special Issue: Knowledge acquisition for therapy-planning tasks.

[New82] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[PETM92] A. R. Puerta, J. W. Egar, S. W. Tu, and M. A. Musen. A multiple method knowledge acquisition shell for the automatic generation of knowledge acquisition tool. *Knowledge Acquisition*, 4(2):171–196, june 1992.

[RW89] Emile M. Roth and David D. Woods. Cognitive task analisys: an approach to knowledge acquisition for intelligent system design. In G. Guida and C. Tasso, editors, *Topics in Expert System Design*, volume 5 of *Studies in Computer Science and Artificial Intelligence*, pages 233–264. North-Holand, 1989.

[Sch93] A. Th. (Guss) Schreiber. *Pragmatics of the Knowledge Level.* PhD thesis, University of Amsterdam, Department of Social Science Informatics, 1993.

[SG95] Bil Swartout and Yolanda Gil. EXPECT: Explicit representations for flexible acquisition. In *Proceedings Of the Ninth Knowledge Acquisition for Knowledge Based Systems Workshop (KAW'95)*, Banff, Canadá, 26th february to 3rd march 1995.

[Sho76] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York, 1976.

[Ste90] L. Steels. Components of expertise. *The AI Magazine*, 11(2), 1990.

[Ste95] Mark Stefik. *Introduction to Knowledge Systems.* Morgan Kaufmann Publishers, San Francisco, CA, 1995.

[SWA+94] Guss Schreiber, Bob Wielinga, Hans Akkermans, Walter Van de Velde, and Anjo Anjewierden. CML: The CommonKADS Conceptual Modeling

Language. In Luc Steels, Guus Schreiber, and Walter Van de Velde, editors, *Proceedings of 8th. European Knowledge Acquisition Workshop (EKAW'94)*, volume 867 of *Lecture Notes in Artificial Intelligence*, pages 1–25, Hoargaarden, Belgium, September 1994. Springer-Verlag.

[SWB93] Guus Schreiber, Bob Wielinga, and Joost Breuker. *KADS: A principled approach to knowledge based system development.* Academic Press, London, 1993.

[SWdHA94] G. Schreiber, Bob Wielinga, Robert de Hoog, and H. Akkermans. CommonKADS: a comprehensive methodology for KBS development. *IEEE Expert*, pages 28–37, december 1994.

[TCdSF95] Lucienio de M. Teixeira, Evandro de B. Costa, Carlos A. Peres da Silva, and Edilson Ferneda. Aquisição de conhecimento em harmonia: um ambiente de aprendizagem. In *Anais do II Simpósio Brasileiro de Computação e Música - SBCM'95, XV Congresso da Sociedade Brasileira de Computação*, pages 290–296, Canela (RS), agosto 1995.

[TKM+89] S. W. Tu, M. G. Kahn, M. A. Musen, et al. Episodic monitoring of time-oriented data for heuristic skeletal-plan refinement. *Communications of the ACM*, 32:1439–1455, 1989.

[TSD+92] S. W. Tu, Y. Shahar, J. Dawes, J. Winkles, et al. A problem-solving model for episodic skeletal-plan refinement. *Knowledge Acquisition*, 4(2):197–216, june 1992.

[VBB93] A. Valente, J. Breuker, and B. Bredeweg. Integrating modeling approaches in the CommonKADS library. In A. Sloman, D. Hogg, G. Humphreys, A. Ramsay, and D. Partridge, editors, *Prospects for Artifical Intelligence, Proceedings of the AISB'93*, pages 121–130, Amsterdam, 1993. IOS Press.

[vHSW97] G. van Heijst, A. Th. Schreiber, and Bob J. Wielinga. Using explicit ontologies in KBS development. *Journal of Human-*

Computer Studies, 42((2/3)):183–292, 1997. Also available at http://www.swi.psy.uva.nl/usr/Schreiber/home.html.

[VL93] André Valente and Christiane Löckenhoff. Organization as guidance: a library of assessment models. In J.-G. Ganascia N. Aussenac, G. Boy, B. Gaines, M. Linster and Y. Kodratoff, editors, *Proceedings of the 7th European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW '93)*, volume 723 of *LNAI*, pages 243–262, Toulouse and Caylus, France, September 1993. Springer-Verlag.

[vMSBP81] W. van Melle, A. C. Scott, J. S. Bennett, and M. Peairs. The EMYCIN manual. Technical Report HPP-81-16, Computer Science Department, Stanford University, Stanford, 1981.

"Se este é o melhor dos mundos possíveis, o que será dos outros?".

Voltaire In *Cândido.*