Alessandro Araújo Jatobá

Um Arcabouço Cooperativo para uma Sociedade de Agentes Tutores

Dissertação 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 em Informática.

Orientadores:

Angelo Perkusich - DSc.
Evandro de Barros Costa - DSc.

Campina Grande, Paraíba, Brasil
Alessandro Araújo Jatobá, maio de 1998

J39a Jatobá, Alessandro Araújo.
Um arcabouço cooperativo para uma sociedade de agentes tutores / Alessandro Araújo Jatobá. - Campina Grande, 1998.
88 f.

Dissertação (Mestrado em Informática) - Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, 1998.
"Orientação : Prof. Dr. Ângelo Perkusich, Prof. Dr. Evandro de Barros Costa".
Referências.

1. Inteligência Artificial. 2. Sociedade de Agentes Tutores Inteligentes. 3. Inteligência Artificial Distribuída. 4. Dissertação - Informática. I. Perkusich, Ângelo. II. Costa, Evandro de Barros. III. Universidade Federal da Paraíba - Campina Grande (PB). IV. Título

CDU 004.8(043)

UM ARCABOUÇO COOPERATIVO PARA UMA SOCIEDADE DE AGENTES TUTORES

ALESSANDRO ARAÚJO JATOBÁ

DISSERTAÇÃO APROVADA EM 26.05.1998

PROF. ANGELO PERKUSICH, Dr.
Orientador

PROF. EVANDRO DE BARROS COSTA, D.Sc
Orientador

PROF. JAIME SIMÃO SICHMAN, Dr.
Examinador

PROF. JACQUES PHILIPPE SAUVÉ, Ph.D
Examinador

CAMPINA GRANDE - PB

Agradecimentos

Se fosse possível agradecer a apenas uma pessoa, se me fosse dado espaço para somente um nome, esse nome seria o de Juliana. Durante o desenvolvimento deste trabalho Juliana foi meu alicerce, o seu incentivo incansável e incondicional foi indispensável para a sua conclusão.

A Angelo, que me ajudou como se estivesse me alfabetizando, sua paciência é imensurável, obrigado pela amizade que acabamos estabelecendo e pelo conhecimento transmitido. Agradeço a Evandro pelo crédito dado no início e pela experiência repassada durante o desenvolvimento do trabalho.

A Light-Infocon, pela oportunidade, espaço e liberdade, sempre que precisei fui atendido. A todos os companheiros de trabalho, pela agradável convivência. Reservo espaço aqui para um agradecimento especial a Marcos Sebastian, pelo seu apoio e amizade.

Resumo

Neste trabalho apresentamos a concepção e implementação de um arcabouço cooperativo para uma sociedade de agentes tutores inteligentes. Esta sociedade está inserida no contexto de um ambiente interativo de ensino-aprendizagem denominado MATHEMA e nas pesquisas na área de Inteligência Artificial Distribuída. O desenvolvimento de um tal arcabouço passa pelo estabelecimento de um modelo de organização, cooperação e comunicação para a sociedade e um modelo para os agentes tutores. Além disso, este trabalho contribui para o projeto da arquitetura dos agentes tutores. De modo a viabilizar o comportamento cooperativo define-se uma linguagem de interação e os protocolos que devem ser utilizados. Detalha-se ainda a implementação de um arcabouço de *software* que promove o desenvolvimento de agentes tutores no contexto do MATHEMA.

Abstract

This work presents the design and development of a cooperative framework for a society of intelligent tutoring agents. This society is within the scope of an interactive learning environment named MATHEMA and in the research in the field of Distributed Artificial Intelligence. The development of this framework starts with the definition of models for organization, cooperation, and communication for this society, and a model for the tutoring agents. Furthermore, this work contributes to the definition of the architecture of the tutoring agents. In order to promote the cooperative behavior, an interaction language as well as interaction protocols are defined. Also, some implementation issues related to this framework are detailed.

Sumário

Lista de Figuras

Lista de Tabelas

1

Introdução

Este trabalho se situa na área da Inteligência Artificial Distribuída (IAD). A IAD é uma sub-área da Inteligência Artificial que propõe oferecer um novo paradigma para solucionar os problemas que atualmente são associados com a construção de sistemas grandes, complexos e ricos em conhecimento. A IAD defende que tais problemas devam ser decompostos e trabalhados por entidades (agentes) que se comunicam e cooperam entre si num regime de controle descentralizado **[Jen94]**.

Principalmente na sua primeira década de existência, a ênfase das pesquisas de IAD foi na elaboração de estudos para apontar os problemas chave da área. Estes estudos demonstraram uma variedade de técnicas novas; portanto, desta forma, esta fase possuiu um caráter mais conceitual. Entretanto, nos últimos anos de atividades, houve na IAD um deslocamento do foco para outras direções **[Gas92]**, sendo que uma delas prioriza um enfoque na concretização de soluções aplicáveis a problemas do mundo real.

Há inúmeros domínios em que as técnicas e conceitos de IAD estão sendo empregados, dentre eles podemos citar: processamento de linguagem natural, robótica, manufatura, controle de tráfego aéreo, *design* e sistemas de informação organizacional **[Jen94, KJ98]**.

O presente trabalho está inserido no projeto MATHEMA **[CLF95, CP96, CPJ96, CL97, CP97a, CP97b, CPJ97, Cos97]**. Este projeto, que emprega conceitos da área de IAD, tem como objetivo básico a concepção e especificação de um modelo para um ambiente interativo de aprendizagem. Este ambiente visa essencialmente tratar das relações pedagógicas e interativas entre um aprendiz humano e um conjunto de tutores artificiais. A visão de um sistema tutor como um conjunto de tutores originou-se

basicamente de uma reflexão sobre os problemas ligados à definição do modelo do domínio da aplicação, do modelo do aprendiz e do modelo pedagógico, ou seja, da complexidade do conhecimento envolvido na concepção destes sistemas.

Para um domínio da aplicação, concebeu-se uma visão particionada do conhecimento em entidades num modelo multidimensional **[Cos97]**. Neste contexto, cada entidade seria de responsabilidade de um tutor. Desta forma, foi natural utilizar uma abordagem no contexto da IAD e considerar a organização dos tutores numa sociedade de agentes artificiais. Mais especificamente, utiliza-se uma abordagem multiagentes **[AS97],** na qual os agentes tutores cooperam objetivando viabilizar o processo de ensino-aprendizagem. Tal abordagem oferece uma possibilidade concreta para o gerenciamento da complexidade do desenvolvimento de um tal sistema (modularização, reusabilidade, escalabilidade, etc **[Jen94]**).

A partir da necessidade de desenvolver esta sociedade, surgiram questões clássicas de IAD, notadamente as relativas à viabilização de um comportamento cooperativo entre esses agentes. Os resultados iniciais referentes a estas questões foram divulgados em **[CP96]**, **[CPJ96]** e, posteriormente, em **[CPJ97]**, as mesmas questões foram novamente tratadas, desta vez com mais detalhes e formalidades.

Na presente dissertação, tais questões e outras oriundas dos seus desdobramentos são abordadas detalhadamente. Nesse sentido, os quatro objetivos deste trabalho são:

(i) contribuir para o aprimoramento dos modelos de cooperação e comunicação para a sociedade de agentes tutores, como introduzido em **[Cos97]**;

(ii) com base nos modelos estabelecidos em (i), contribuir para um projeto da arquitetura para os agentes tutores que descreva a configuração e a interação de alto nível dos seus componentes;

(iii) detalhar a linguagem de interação e os protocolos que devem ser utilizados pelos agentes tutores na viabilização do comportamento cooperativo;

(iv) implementar um arcabouço de *software* que promova o desenvolvimento de agentes tutores no contexto do MATHEMA.

A realização dos objetivos mencionados envolveu inicialmente um estudo de IAD e um envolvimento com o projeto MATHEMA. Vale salientar que para alcançar os objetivos (ii) e (iii), a estratégia adotada baseou-se num processo cíclico, iniciado em (ii), que incluia definições, avaliações e refinamentos. No caso da implementação, foi necessário avaliar os requisitos do sistema e a viabilidade de algumas soluções (ferramentas, bibliotecas).

O restante da dissertação está organizado da seguinte forma. O capítulo 2 apresenta noções fundamentais da área de Inteligência Artificial Distribuída ressaltando os aspectos mais relevantes no contexto deste trabalho: as suas principais áreas, os principais alvos de pesquisa, alguns termos e conceitos particulares. O Capítulo 3 apresenta o projeto MATHEMA e discute o modelo do agente definido para os agentes tutores do ambiente. O Capítulo 4 detalha os elementos e os comportamentos da arquitetura desenvolvida a partir do modelo do agente. O Capítulo 5 descreve os protocolos utilizados no processo interativo da sociedade bem como a sintaxe da linguagem utilizada na comunicação entre os agentes. O Capítulo 6 trata da implementação desenvolvida, ressaltando as necessidades operacionais levantadas e alguns detalhes das soluções adotadas. O resultado apresentado neste capítulo é um protótipo de um arcabouço de *software* que objetiva viabilizar o comportamento social dos agentes tutores no ambiente MATHEMA e, com isso, diminuir o esforço empregado no desenvolvimento de sociedades. O Capítulo 7 apresenta as conclusões obtidas do trabalho realizado e aponta os trabalhos futuros, sugerindo alguns possíveis encaminhamentos.

2

Noções fundamentais em IAD

Neste capítulo, pretende-se apresentar ao leitor uma visão introdutória relativa à Inteligência Artificial Distribuída (IAD), objetivando assim oferecer o suporte necessário ao entendimento de alguns conceitos utilizados no restante desta dissertação. Não se tem aqui a intenção de que o texto sirva para uma leitura independente dos objetivos expostos nesta dissertação.

O presente capítulo é apresentado com o seguinte roteiro. Inicialmente apresentamos as diferenças entre os enfoques da IA Distribuída e da IA Clássica, e discutimos as possibilidades que motivam os investimentos em IA Distribuída. A seguir são apresentados alguns conceitos básicos como: agente, sociedade, ambiente, organização e interação. Aborda-se também as sub-áreas Resolução Distribuída de Problemas (RDP) e Sistemas Multiagentes (SMA), bem como os conceitos de agentes reativos e cognitivos. Comenta-se ainda neste capítulo os problemas clássicos na área. Finalmente são discutidos aspectos relativos ao protocolo de interação, linguagem de interação e visões de programação.

2.1. IA Clássica e a IA Distribuída

Na IA Clássica, o enfoque é na representação do conhecimento e nos métodos de inferência; além disso, a metáfora da inteligência é de origem psicológica e baseia-se no comportamento humano individual. Por sua vez, na IA Distribuída, a ênfase está nas ações e nas interações entre agentes; aqui, a metáfora da inteligência é de natureza sociológica/etológica e baseia-se no comportamento social **[SDB92]**.

Há várias situações onde se pode atribuir um "comportamento inteligente" a uma coleção de entidades: uma empresa competitiva, uma equipe de futebol, uma colônia de formigas. Observe que especificamente neste último caso poucos atribuiriam a propriedade de inteligência a uma formiga isolada.

Como mencionado já na introdução deste trabalho, este tipo de abordagem é promissora para a resolução de problemas grandes e complexos com o envolvimento de conhecimentos de vários domínios distintos. Por outro lado, do ponto de vista computacional, a visão distribuída de um sistema conduz a algumas possibilidades que também motivam os investimentos na área de IAD, dentre elas podemos citar:

- **desenvolvimento e manutenção**: criar e manter um conjunto de módulos independentes é uma solução mais flexível do que um módulo monolítico;

- **incremento de desempenho**: é possível fragmentar um determinado problema para ser realizado paralelamente e, com isso, conseguir uma melhoria significativa de desempenho;

- **robustez e confiabilidade**: um sistema distribuído pode continuar funcionando mesmo que alguma das partes apresente falhas, e desde que ele tenha sido projetado para isso;

Para o leitor interessado em maiores detalhes sobre as questões abordadas nesta seção, bem como sobre questões históricas sobre a abordagem IAD, recomenda-se consultar **[AS97, Jen94, KJ98]**.

2.2. Conceitos Básicos

O objetivo desta seção é introduzir alguns conceitos básicos que são utilizados neste trabalho. Desta forma, introduz-se as definições de agente, sociedade, ambiente, interação e organização.

2.2.1. Agente, Sociedade e Ambiente

O termo agente é largamente empregado (inclusive em outras áreas, como por exemplo: Sistemas Distribuídos e Orientação a Objetos), e há uma ampla variedade de definições formuladas segundo um comprometimento com algum contexto específico. A literatura consultada no tema revela a ausência de uma definição consensual e única a seu respeito. Franklin & Graesser em **[FG96]** organizaram uma coletânea de definições sobre agentes, mostrando como cada autor, entre os relacionados em tal referência, se pronuncia com respeito à questão. Com base nessa coletânea, pode-se perceber a presença de uma série de propriedades que servem para mostrar os pontos comuns entre as visões dos autores, ao mesmo tempo em que se observa os pontos que servem para distinguir tais visões. Algumas destas definições são destacadas a seguir:

> *Um agente é qualquer entidade que pode ser vista como percebendo seu ambiente através de sensores e agindo sobre este ambiente através de efetuadores.*
>
> *Agentes Inteligentes realizam constantemente três funções: percepção das condições dinâmicas no ambiente; ações para afetar as condições no ambiente; e raciocínio para interpretar percepções, resolver problemas, efetivar inferências, e determinar ações.*
>
> *Agentes Inteligentes são entidades de software que executam algum conjunto de operações em favor de um usuário ou um outro programa com algum grau de independência ou autonomia, e em assim fazendo, empregam algum conhecimento ou representação dos objetivos ou desejos dos usuários.*

Apesar das definições anteriores, uma definição muito bem aceita pela comunidade e que se mostra abrangente e adequada aos propósitos deste trabalho, é a proposta por **[FG91]**:

> *Um agente é uma entidade real ou virtual, imersa num ambiente sobre o qual ela é capaz de agir, que dispõe de uma capacidade de percepção e de*

representação parcial deste ambiente, podendo se comunicar com outros agentes e apresentando um comportamento autônomo, consequência de suas observações, de seu conhecimento e das suas interações com outros agentes.

Em **[AS97]** determina-se que agentes são as entidades ativas de um sistema, o conjunto de agentes forma uma sociedade, e as entidades passivas são designadas pelo termo ambiente. Um agente, entre outras atividades, raciocina sobre o ambiente, sobre os outros agentes e decide quais objetivos deve perseguir, quais ações deve tomar. Numa analogia grosseira, um agente corresponde a um processo; a sociedade, a um conjunto de processos; e o ambiente, a um conjunto de entidades passivas, excluindo os processos, com os quais o sistema tem relação (um banco de dados, um arquivo, algum periférico, etc).

2.2.2. Interação

Uma troca de informação (ações, planos, objetivos, hipóteses) que pode ocorrer entre os agentes de uma sociedade é designada pelo termo interação. Novamente, fazendo uma analogia grosseira com sistemas distribuídos, uma interação corresponde a uma comunicação entre dois processos. Alguns trabalhos enfatizam a distinção entre comunicação e interação **[Pop93, AS97, Dem95]**: interação é uma troca de informação, comunicação é uma troca de dados.

Grande parte das abordagens para interações entre agentes em IAD se baseia na teoria de atos da fala (*Speech Act Theory*) **[Sea69]**. Segundo tal teoria, comunicar é agir: a comunicação é uma ação normal que deve ser gerada e processada pelos agentes. Tal teoria sugere uma categorização de primitivas de comunicação. Os atos podem ser classificados conforme três aspectos: **locucionário**: emissão de palavras e sentenças com algum significado; **ilocucionário**: corresponde à intenção da emissão (*request, apologize, assert)*; **perlocucionário**: corresponde ao resultado da emissão (*convince, insult, frighten).* A categorização sugerida pela teoria de atos da fala serve de referencial para a escolha da natureza (primitiva) das mensagens trocadas entre os agentes.

Um aspecto ligado à interação que merece menção e que deve ser considerado durante o projeto de uma sociedade de agentes é o modo de comunicação utilizado como suporte. As duas técnicas existentes são superficialmente bem diferentes, porém, na prática, oferecem a mesma funcionalidade, são elas:

- **Quadro-Negro** (*Blackboard*): a comunicação ocorre indiretamente através de uma estrutura de conhecimento compartilhada. Os agentes podem afixar itens no quadro e ler os que foram postados para ele por outros agentes;

- **Troca de Mensagens**: um agente envia a um ou mais agentes a mensagem diretamente através de um ambiente de comunicação.

2.2.3. Organização

A noção de organização está reclacionada com o modelo de comportamento social dos agentes. Como apresentado em **[Gas92]**, uma organização disponibiliza um arcabouço para atividade e interação através da definição de papéis, comportamentos, e relações de autoridade (controle). Uma possível classificação de tipos organizacionais é a seguinte:

- **Centralizada ou hierárquica**: a autoridade para tomada de decisões de controle é concentrada num único agente ou grupo especializado, em cada nível na hierarquia.

- **Mercado**: O controle é distribuído num ambiente análogo a um mercado, e os agentes interagem via competição por tarefas e recursos através de um mecanismo de controle contratual e de ligação.

- **Comunidade pluralística**: O modelo de como a comunidade científica opera é usado como modelo de referência. As soluções dos problemas são obtidas localmente, e então são comunicadas a outros agentes que podem testar, discutir e refinar as mesmas.

- **Comunidade com regras de comportamento**: Trata-se de um organização plana, composta por agentes especialistas em alguma área particular. As interações são regidas por regras de comportamento, que se refletem em protocolos de interação.

2.3. RDP e SMA

Há duas sub-áreas na IAD: Resolução Distribuída de Problemas (RDP) e Sistemas Multiagentes (SMA). Conforme apresentado em **[Sic95]**, cada uma delas possui motivações científicas e características próprias.

Na abordagem RDP, os agentes são projetados com o intuito de resolver algum problema inicial preciso e particular. Isto quer dizer que, a priori, os agentes não podem ser utilizados para resolver outros problemas similares. As noções de organização e controle em tal abordagem são geralmente preestabelecidas e concebidas a partir do problema alvo durante a fase de projeto. As habilidades e objetivos dos membros da sociedade são definidos pelo projetista. As tarefas e as possíveis decomposições, em muitos casos, são também totalmente preestabelecidas pelo projetista, e devido a isso, situações de conflito não tendem a surgir. A sociedade possui um caráter estático pois, normalmente, novos agentes não podem ser adicionados na sociedade durante a fase de resolução. A abordagem RDP encontra-se ilustrada na Figura 2.1 e pode ser acompanhada no esquema que segue:

(problema a resolver) ⇒ (projeto dos agentes) ⇒ (resolução do problema) ⇒ (solução)

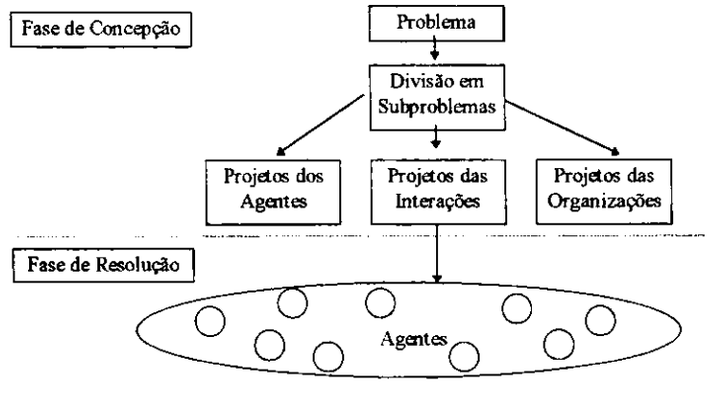


Figura 2.1: A abordagem RDP **[Sic95]**

A abordagem SMA se preocupa com a atividade de agentes autônomos num universo multiagente **[DM90, HS97]**. A principal diferença entre a abordagem RDP e a SMA pode ser explicada através da propriedade de *autonomia*. Autonomia implica que os agentes possuem uma existência própria, sem a necessidade da existência prévia de um problema a ser resolvido. Os projetos das organizações e das interações são desenvolvidos visando uma independência com a aplicação alvo; busca-se então conceber protocolos genéricos que possam ser reutilizados em aplicações similares. A decomposição das tarefas é realizada pelos agentes e não pelo projetista. O controle é implementado de forma descentralizada nos agentes. Um SMA é um sistema aberto, qualquer agente pode entrar e sair da sociedade durante a fase de resolução. A abordagem SMA está ilustrada na Figura 2.2 e pode ser acompanhada no esquema que segue:

(agentes já existentes) ⇒ (problema a resolver) ⇒ (resolução do problema) ⇒ (solução)

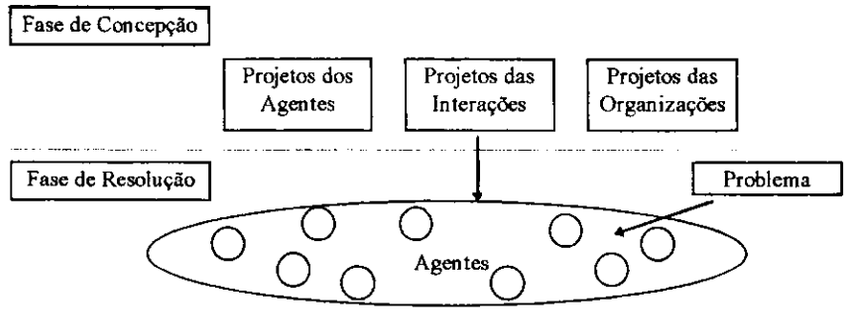


Figura 2.2: A abordagem SMA **[Sic95]**

Três relações possíveis entre as abordagens RDP é SMA são apresentadas em **[DR94]**: (i) RDP é um subconjunto de SMA e, desta forma, RDP pode ser considerado como um caso particular de SMA; (ii) SMA fornece uma base para RDP; (iii) RDP e SMA são abordagens complementares, em determinadas situações é difícil para um observador externo definir a abordagem que foi utilizada como base na construção de determinados sistemas.

2.4. Agentes Reativos e Cognitivos

Na área de SMA, há duas diferentes abordagens distintas que estão relacionadas com a granularidade da sociedade: agentes reativos (muitos agentes não inteligentes) e agentes cognitivos (poucos agentes inteligentes).

Uma sociedade de agentes reativos é baseada em modelos organizacionais biológicos/etológicos. A principal idéia nesta abordagem é que um comportamento complexo inteligente pode surgir de um conjunto de entidades muito simples. O exemplo clássico é um formigueiro. Uma formiga sozinha não possui muita inteligência, porém um formigueiro age como uma entidade inteligente. Um agente reativo responde a estímulos e, com base em **[AS97, FG91]**, as principais características dos agentes e dos sistemas reativos são:

(i) *não há representação explícita do conhecimento*: o conhecimento de um agente é implícito e se manifesta através do seu comportamento;

(ii) *não há representação do ambiente*: o seu comportamento se baseia no que é percebido a cada instante do ambiente;

(iii) *não há memória das ações*: não é mantido um histórico das ações de um agente reativo, uma ação passada não influencia uma ação futura;

(iv) *grande número de membros*: as sociedades de agentes reativos possuem, em geral, um grande número de componentes, na ordem de centenas;

(v) *comunicação indireta*: a comunicação entre os agentes reativos, geralmente, é realizada de modo indireto, através do ambiente;

A abordagem via agentes reativos é muitas vezes considerada similar à abordagem conexionista, como definida em IA, isso basicamente devido à granularidade e à simplicidade dos componentes do sistema.

Por sua vez, uma sociedades de agentes cognitivos é baseada em modelos organizacionais sociais, como grupos, hierarquias e mercados. Segundo **[FG91]**, as principais características dos agentes cognitivos são as seguintes:

(i) *mecanismo de controle deliberativo*: os agentes raciocinam e decidem sobre quais objetivos devem alcançar, que planos seguir e quais ações devem ser executadas num determinado momento;

(ii) *há representação explícita do ambiente*: os agentes possuem uma representação do seu ambiente e dos outros agentes da sociedade;

(iii) *há memória das ações*: os agentes podem raciocinar sobre as ações tomadas no passado e planejar ações a serem tomadas no futuro;

(iv) *pequeno número de membros*: uma sociedade contém tipicamente poucos integrantes, na ordem de uma dezena;

(v) *comunicação direta*: a comunicação entre os agentes é feita de modo direto, através do envio e recebimento de mensagens;

2.5. Problemas Clássicos da IAD

Como introdução, os principais problemas de IAD, tanto em RDP quanto em SMA, podem ser apresentados em cinco áreas, a saber **[BG88]**: (i) descrição, decomposição e alocação de tarefas; (ii) comunicação, interação, linguagens e protocolos; (iii) coordenação, controle e comportamento coerente; (iv) conflito e incerteza; (v) linguagens e ambientes de programação. A seguir, introduzimos as cinco áreas.

(i) **Descrição, Decomposição e Alocação de Tarefas:** Em certos casos, é necessário decompor uma tarefa pois não necessariamente a sociedade ou o próprio agente tem a competência para solucioná-la por inteiro. Decompor uma tarefa significa particioná-la num conjunto de subtarefas no intuito de formar uma estratégia para a sua execução. Com a decomposição pode-se tirar proveito do paralelismo. Já a alocação é um processo de seleção dos agentes

aptos a executar uma determinada tarefa que foi submetida à cooperação. A alocação tende a reduzir o número de negociações num SMA. Como se deve ***descrever*** e ***decompor*** uma tarefa complexa em subtarefas, de modo estático ou dinâmico, como ***alocar*** estas subtarefas e em que ordem elas devem ser executadas ?

(ii) **Comunicação, Interação, Linguagem e Protocolo:** Quais primitivas básicas de comunicação devem ser utilizadas para exprimir os conceitos semânticos envolvidos num processo interativo e como estas primitivas devem ser combinadas em protocolos de interação ?

(iii) **Coordenação, Controle e Comportamento Coerente (Coletivo):** Como se pode garantir um comportamento global coerente numa sociedade de agentes, cada qual com os seus próprios objetivos e habilidades, e como um controle de um tal sistema pode ser projetado ?

(iv) **Conflito e Incerteza:** Quando um agente não possui informação completa sobre o ambiente e sobre os outros agentes da sociedade, como podem ser detectadas e solucionadas as situações de conflito, e como devem ser tratados os dados incertos e incompletos a fim de garantir resultados coerentes ?

(v) **Linguagens e Ambientes de Programação:** Do ponto de vista computacional, quais as linguagens de programação que devem ser utilizadas para o desenvolvimento de tais sistemas, e o que um ambiente de desenvolvimento deve oferecer para possibilitar a implementação e o teste destes sistemas ?

É importante salientar que alguns dos problemas citados são alvos de investigação também em outras áreas como a IA, a lógica, e a computação distribuída.

2.6. Protocolo de Interação

Um protocolo serve para estruturar as trocas de mensagens entre agentes. Protocolos são concebidos com um caráter genérico para possibilitar a utilização em outras

aplicações **[AS97]**. De fato, um protocolo restringe as diferentes interações que um agente pode ter com os outros membros da sociedade considerando um problema a ser solucionado **[Dem95]**.

Um protocolo clássico bastante utilizado em IAD é o protocolo de **Rede Contratual** — também conhecido como protocolo de licitação. No presente trabalho, licitação é considerada apenas uma das etapas do protocolo de **Rede Contratual**. Pela definição em **[Smi92]**, trata-se de um protocolo de alto nível, baseado numa estrutura de mercado, desenvolvido para especificar a comunicação e o controle para os nós num resolvedor distribuído de problemas. Porém, independente disto, devido a sua adequação, tal protocolo é bastante empregado também em SMAs.

Basicamente, o protocolo é baseado no estabelecimento de um contrato, que após um processo de negociação (licitação) cria um vínculo de cooperação entre dois agentes: onde um é o cooperador e o outro é o beneficiado. Isto acontece através de um processo de seleção mútua baseado numa transferência bilateral de informação. A Figura 2.3 ilustra este protocolo, o qual é detalhado a seguir.

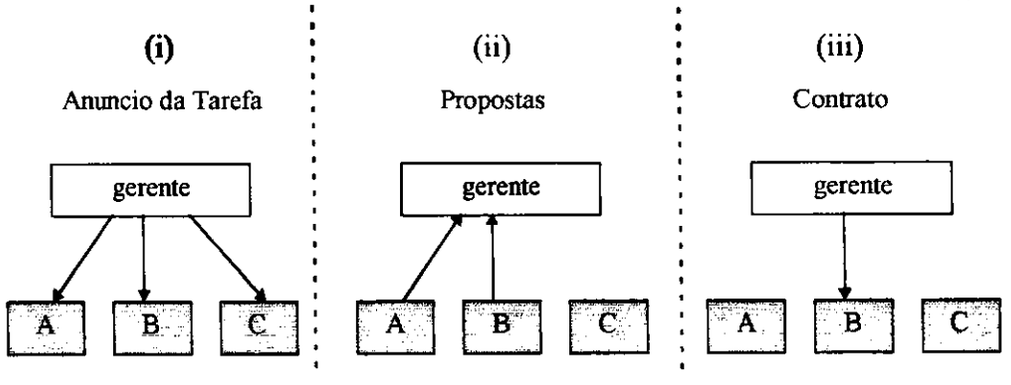


Figura 2.3: Protocolo de Rede Contratual

(i) um agente, denotado na Figura 2.3 como *gerente*, precisa de cooperação e propõe uma tarefa para a sociedade;

(ii) os agentes subordinados que se vêem na situação de poder cooperar oferecem propostas que são coletadas pelo gerente;

(iii) o gerente escolhe a melhor proposta para a tarefa em questão para então estabelecer um contrato com o escolhido.

Há inúmeros outros protocolos de interação mencionados na literatura de IAD: *Everbody's Request Protocol*; *Request-until-Satisfaction Protocol*; *Berthet's Doubt, Instruction and Introduction Protocols*; *Sian's Cooperative Learning Protocol*; *MOSCA Protocol*, entretanto uma discussão destes protocolos está fora do escopo desta dissertação. O leitor interessado pode consultar Demazeau e Berthet et alli **[Dem95, BDB93]** para obter informações sobre estes outros protocolos de interação.

2.7. Linguagem de Interação

Uma linguagem de interação é utilizada para estruturar as interações dos agentes da sociedade. Uma linguagem de interação define restrições formais, um vocabulário comum (sintaxe) e uma semântica associada para as interações **[DBK94]**. A estruturação facilita a interpretação da interação e permite um ganho de eficiência nos processos de inferência sobre ela. Uma linguagem de interação usualmente possibilita expressar informações relativas a três níveis: comunicação, tratamento social e domínio de aplicação.

Há atualmente algumas linguagens já desenvolvidas que se destinam apenas à interação entre agentes. O exemplo mais conhecido é o resultado de um projeto denominado "*knowledge sharing initiative*" **[Fin94]**. Tal projeto é dividido em duas partes: uma linguagem para manipulação e consulta de conhecimento (KQML) e um formato para intercâmbio de conhecimento (KIF). KQML define performativas baseadas em atos de fala, KIF é uma representação em ASCII de uma versão estendida de lógica de primeira ordem e serve para expressar o conteúdo da mensagem.

2.8. Programação de Agentes

Com relação ao paradigma de agentes, como pode ser visto em **[AS97]**, há duas visões de programação bem distintas; são elas:

- **ponto pequeno** (em inglês: *programming-in-the-small*): os paradigmas atuais de linguagens de programação — procedural, funcional, lógico, objetos — não são adequados. A noção de agentes é suficiente para gerar um novo paradigma **[Sho93]** e, devido a isso, deve-se desenvolver linguagens particulares (compiladores/interpretadores) que implementem as necessidades computacionais de tal paradigma, por exemplo: definição e tratamento de estados mentais (crenças, objetivos, planos, engajamentos) e definição de primitivas que se ajustem à teoria de atos de fala **[Sea69]**. Como exemplo desta visão podemos citar as linguagens AGENT0 **[Sho93]** e PLACA **[Tho93].**

- **ponto grande** (em inglês: *programming-in-the-large*): a maior contribuição do paradigma de agentes consiste numa prática de modelagem e integração de sistemas heterogêneos e abertos. Assim sendo, não é necessário impor uma linguagem específica; deve-se sim, propiciar às diversas linguagens, independente do paradigma de programação, a possibilidade de implementar as construções computacionais necessárias através de extensões — por exemplo, bibliotecas. Como exemplo podemos citar o ambiente MASENV **[Dem95]**.

3

Modelo do Agente

Neste capítulo descreve-se o modelo de agente tutor adotado no âmbito do projeto MATHEMA, baseando-se em **[COS97, CPJ96, CPJ97].** Inicialmente, faz-se uma apresentação sucinta do ambiente MATHEMA a fim de contextualizar mais especificamente o presente trabalho. A seguir, apresentamos o modelo do agente tutor considerando os seguintes aspectos: organização, controle, cooperação e comunicação. É importante observar que neste trabalho o termo "modelo do agente" é uma abstração de um dado agente tutor, diferentemente de outros autores que consideram o termo "modelo do agente" como uma descrição estrutural que caracteriza as capacidades de um outro agente da sociedade **[Gas92]**. No modelo do agente tutor, detalham-se aspectos sintáticos e semânticos, sintáticos de modo formal e semânticos informalmente. Introduzimos no final do capítulo a arquitetura que foi projetada para tais agentes, com base no modelo definido.

3.1. O ambiente MATHEMA

O MATHEMA é um modelo de ambiente interativo de aprendizagem, concebido para lidar com uma relação pedagógica envolvendo, essencialmente, um aprendiz humano e um sistema tutor multiagentes (representado por uma sociedade de agentes tutores artificiais). Entretanto, em sua arquitetura figuram outras entidades que, juntamente com essas duas, são discutidas mais adiante nesta seção. A idéia central no MATHEMA é envolver o aprendiz numa situação de resolução de problemas e daí o sistema encarrega-se de prover o suporte necessário para tornar essa atividade produtiva.

Portanto, o que se busca na relação aprendiz *versus* sistema tutor é uma qualidade no processo de interação no sentido de que o aprendiz tire um maior proveito em termos de aprendizado. Nesse sentido, faz-se, por exemplo, uma série de exigências sobre o tutor no tocante ao seu conhecimento e seus mecanismos de manipulação deste conhecimento. Um tal conhecimento normalmente diz respeito a questões, tais como: *o que ensinar* (conhecimento especializado sobre o domínio de aplicação - modelo do domínio), *a quem ensinar* (conhecimento que o sistema possui sobre o aluno - modelo do aluno) e *como ensinar* (conhecimento sobre estratégias de ensino - modelo pedagógico).

Daí, no âmbito do MATHEMA, procurou-se impor critérios de qualidade [CP97a, CP97b] sobre os conhecimentos envolvidos num sistema tutor, chegando-se a um modelo multidimensional para o conhecimento do domínio. Com este modelo, buscou-se um compromisso entre riqueza e estruturação do conhecimento apropriada ao domínio, resultando na disponibilidade de múltiplas visões sobre este domínio [Cos97]. A idéia básica é submeter um dado domínio, disciplinado pelas múltiplas visões, a um particionamento em diferentes subdomínios, buscando um corpo de conhecimento que lhe seja subordinado no momento de sua operacionalização. Esta busca é, portanto, orientada de maneira a alcançar um conhecimento com especialidades distribuídas em três dimensões, a saber: contexto, profundidade e lateralidade, como ilustrado na Figura 3.1.

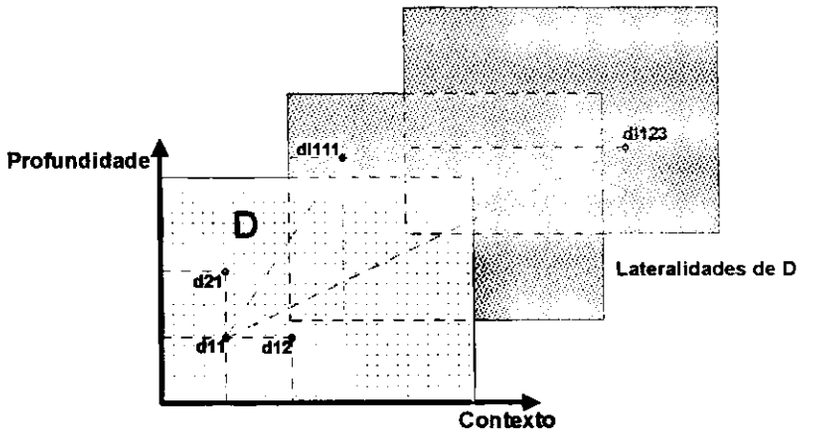


Figura 3.1: Visão Multidimensional do Conhecimento do Domínio

O contexto diz respeito às várias interpretações que podem ser dadas a um domínio de conhecimento a ser ensinado, por exemplo num domínio da álgebra, mais especificamente com equação do segundo grau, pode-se assumir dois contextos: através de métodos algébricos (fórmula de Báskara, e método da redução), ou através de métodos de resolução do cálculo numérico.

A profundidade é definida com base num contexto particular, significando uma visão do domínio em vários níveis, assumindo-se que um determinado objeto de conhecimento pode ser ensinado em níveis distintos. No mesmo exemplo anterior, fixando-se o primeiro contexto, podemos considerar dois domínios: dos números reais e dos números complexos.

Por fim, a lateralidade trata do conhecimento fortemente relacionado ao objeto de conhecimento em questão, visto num contexto e profundidade fixados. Se adotarmos o contexto de métodos algébricos no domínio dos números reais, os seguintes conhecimentos laterais podem servir de suporte: expressões algébricas, operações com polinômios, produtos notáveis/fatoração, frações algébricas. Em [Cos97], exemplos de outros domínios de conhecimento numa visão multidimensional são apresentados.

Analisando-se as características do modelo para o conhecimento do domínio, constatou-se uma certa complexidade a ser administrada e então, tornou-se praticamente contingente a adoção de uma abordagem de agentes, resultando-se na definição de um sistema tutor multiagentes.

Por fim, devido a certas considerações feitas em [COS97], o ambiente MATHEMA passou a ser definido segundo a arquitetura mostrada na Figura 3.2. A seguir apresentamos a definição sintática da arquitetura do MATHEMA.

DEFINIÇÃO 3.1: A arquitetura do MATHEMA, denotada por *Arq_Mat*, é definida por uma 6-upla, como segue: $Arq\_Mat = \langle AH, SATA, SEH, AI, AM, ME \rangle$, onde:

(i) *AH* denota um Aprendiz Humano;

(ii) *SATA* denota uma Sociedade de Agentes Tutores Artificiais;

(iii) *SEH* é uma Sociedade de Especialistas Humanos (funcionando como fonte de conhecimento através de operações de manutenção sobre *SATA*);

(iv) *AI* denota o Agente de Interface que é encarregado da interação entre o aprendiz e a *SATA*;

(v) *AM* denota o Agente de Manutenção que é responsável pela interface entre *SEH* e *SATA*;

(vi) *ME* denota um Motivador Externo representando entidades humanas externas que podem motivar o aprendiz a trabalhar no MATHEMA.

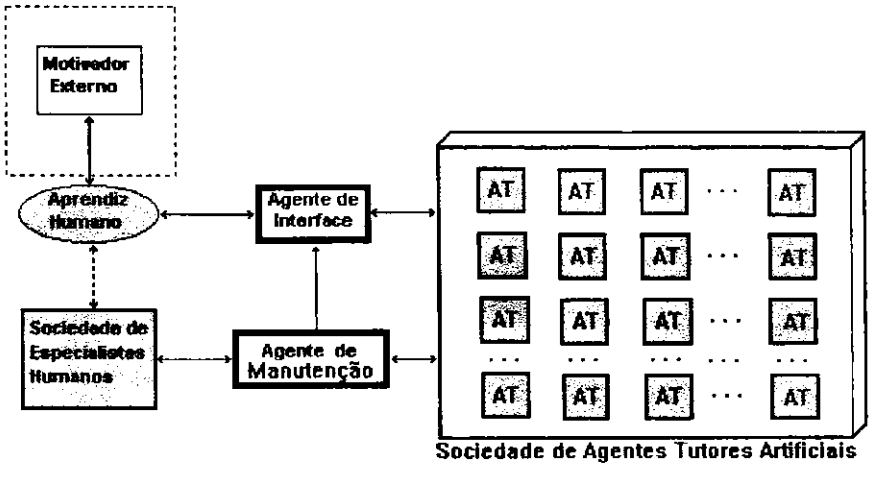


Figura 3.2: Arquitetura do MATHEMA

A arquitetura do MATHEMA, mostrada na Figura 3.2, ilustra esses seus seis componentes, sendo cada um deles descritos abaixo:

(i) *Aprendiz Humano*: entidade interessada em aprender algo sobre um dado domínio, trabalhado no ambiente MATHEMA. Ele desempenhará essencialmente o papel de um agente ativo, quando envolvido em atividades baseadas em resolução de problemas num dado domínio de aplicação, sob a assistência especializada de um agente tutor (imerso numa sociedade de agentes tutores cooperantes).

(ii) *Sociedade de Agentes Tutores Artificiais*: coleção de agentes que podem cooperar entre si a fim de promover a aprendizagem de um dado aprendiz em atividade de resolução de problema. Cada um desses agentes é especializado em subdomínios relacionados a um dado domínio de conhecimento (domínio alvo). Essa idéia foi inicialmente inspirada nas reflexões contidas em "Sociedade da Mente", de Marvin Minsky **[Min85]**. Segundo Minsky, a inteligência emerge da combinação de agentes mentais, cada um responsável por um pequeno processo.

(iii) *Sociedade de Especialistas Humanos*: fonte de conhecimento externa ao sistema computacional definido para *SATA*. Ela desempenha, em certos momentos, o papel de uma entidade oracular, servindo de suporte para a *SATA*. Dessa Sociedade é requerida a manutenção da *SATA* (inclusão, exclusão de agentes, bem como alterações no conhecimento dos agentes) e a assistência dos aprendizes, em caso de falhas da *SATA*. A *SEH* é, portanto, responsável pela incrementabilidade nas capacidades da *SATA*. Ademais, a *SEH*, mesmo sem ser requisitada por *SATA*, pode analisar o desenvolvimento das interações entre o aprendiz e a *SATA*, avaliando o desempenho de seus agentes tutores e, sempre que necessário, promovendo-lhe melhorias.

(iv) *Agente de Interface*: representa o elo de ligação entre o *Aprendiz Humano* e a *SATA*. Ele é responsável por desempenhar, primeiramente, o papel de comunicação do agente tutor com o Aprendiz e vice-versa. Além disso, é responsável particularmente por ajudar o aprendiz a escolher um agente dentre os agentes *AT*s em *SATA*, para agir como seu ***agente supervisor*** (metáfora para um orientador acadêmico num ambiente de pesquisa e pós-graduação). Desta interação, identifica-se uma lista ordenada de agentes pertinentes aos interesses do aprendiz, onde o mais adequado é escolhido para ser o tal supervisor. É admitido que o *AI* sabe da existência dos agentes e suas capacidades em *SATA*.

(v) *Agente de Manutenção*: representa um elo de ligação (*interface*) entre a *SEH* e a *SATA*, encarregando-se de prover a interação entre elas. Para isso, oferece os meios necessários para a percepção, comunicação e manutenção da *SATA*.

(vi) *Motivador Externo*: entidades humanas externas que desempenham o papel de quem motiva o aprendiz a trabalhar no MATHEMA. Alguns motivadores podem ser um professor dele (numa situação onde por exemplo está bem definido o trilogismo aluno, professor e computador), seus colegas, etc. Num trabalho futuro sobre o MATHEMA, algumas dessas entidades poderão ser unidas ao Aprendiz para constituir a idéia de grupo cooperativo.

3.2. O Modelo do Agente

Um agente tutor no MATHEMA é baseado num modelo que descrevemos nesta seção. Um modelo é uma especificação para um agente tutor qualquer na SATA composto por uma definição sintática (elementos) e uma definição semântica (comportamental). Inicialmente detalhamos quais são os comportamentos comuns a cada agente da sociedade, isto é, como os agentes se organizam, como é o controle da sociedade, como eles cooperam entre si e se comunicam. A seguir discutimos quais são os elementos, variáveis de um agente para outro, até chegar numa definição sintática formal para o modelo.

3.2.1. Organização

Para um agente tutor envolver outros agentes numa resolução cooperativa de uma dada tarefa, este lança mão da sua estratégia para formação de coalizão. Uma tal estratégia está relacionada com a noção de organização de grupo de agentes dentro da *SATA*. A organização social de um sistema multiagentes é a maneira pela qual um grupo é formado, em um dado momento, a fim de resolver uma certa tarefa conjuntamente.

O modelo de organização adotado no presente trabalho para os agentes tutores é baseado num modelo de agentes cognitivos **[SDB92]**, no sentido de que ele é inspirado em um tipo de organização social (a exemplo de um departamento acadêmico),

caracterizado por meio de suas interações sociais. Este modelo se inclui na categoria ascendente (*bottom-up*) apresentada por Conte e Castelfranchi, em 1992 [apud **AS97**], segundo a qual os agentes não têm necessariamente um objetivo comum a atingir. Mais especificamente, trata-se de um modelo baseado na complementaridade tal como mostrado por Conte e Sichman, em 1995 [apud **AS97**], onde se considera a possibilidade da existência de capacidades complementares entre os agentes, permitindo, desse modo, que um agente possa alcançar seus objetivos pedindo auxílio aos demais.

O modelo de organização de grupo na *SATA* foi definido para ser dinâmico. Isso impõe que o processo de formação de uma coalizão não esteja preestabelecido, mas ocorra dinamicamente em função de uma demanda particular de um agente.

3.2.2. Controle

O controle das atividades cooperativas entre os agentes na *SATA* é de natureza essencialmente distribuída. A idéia básica do controle pode ser resumida assim: inicialmente o controle fica sob a responsabilidade do *agente supervisor* que interage diretamente com o *Aprendiz*. Esse agente controla as interações com os agentes aos quais ele pediu cooperação. No momento em que um desses agentes precisa de cooperação, ele identifica os agentes com os quais pode cooperar e, nesse caso, passa a ser o controlador dessa cooperação. Esse processo continua de tal maneira que quem pede cooperação passa a ser o controlador dessa nova cooperação. Em suma, o controle não está sob a responsabilidade de um agente específico, mas todos podem potencialmente exercê-lo. Portanto, pode-se dizer que o controle, no âmbito da sociedade de agentes, é distribuído, porém cada agente possui localmente um mecanismo de controle.

3.2.3. Cooperação

Como já foi discutido anteriormente, um agente nem sempre consegue resolver por si só uma tarefa ou mesmo uma parte dela. Quando isto acontece, ele busca cooperação com outros agentes na sociedade a fim de resolver aquilo que ele não conseguiu solucionar. Daí surge a idéia de cooperação, que diz respeito à possibilidade de um agente ter que

envolver outros agentes para resolver uma tarefa sua, que ele não é capaz de realizar total ou parcialmente.

A maneira pela qual os agentes se unem para resolver tarefas encontra-se definida num modelo de cooperação. Um modelo de cooperação define os protocolos e as interações que podem ocorrer no momento da realização de uma atividade cooperativa. A definição das interações é decorrente dos protocolos utilizados na viabilização do processo cooperativo.

O modelo de cooperação adotado no presente trabalho é híbrido, pois se utiliza tanto de um protocolo Mestre-Escravo, quanto de um protocolo baseado em Licitação. O primeiro concerne a situação na qual um agente dispõe de uma tarefa a ser resolvida e, tendo identificado algum agente com capacidade para executá-la, ele simplesmente envia a tarefa para este. Quando o agente requisitado conclui a tarefa, ele retorna os resultados ao agente que o requisitou. Já o segundo modelo tem a ver com a seguinte situação: se um agente tiver uma tarefa a ser resolvida e não conhece ninguém habilitado a resolvê-la, o agente envia um anúncio de tarefa para toda sociedade ou para uma parte dela. Os agentes que se acharem em condições de executá-la, enviam uma proposta de volta. Caso mais de um agente responda favoravelmente, o anunciante escolhe um dentre os proponentes e utiliza o protocolo Mestre-Escravo. Caso apenas um agente se proponha, o anunciante utiliza o mesmo modelo do caso anterior, diretamente. No caso mais extremado, quando nenhum agente dá um retorno positivo, chega-se numa situação de impossibilidade de cooperação com a sociedade, surgindo a necessidade de se recorrer ao mundo externo, neste caso a *SEH*.

É importante salientar que na SATA uma dada cooperação pode resultar em outra, a partir do agente que está cooperando, e assim sucessivamente. Deste modo, caracteriza-se que o comportamento cooperativo é passível de recursão.

3.2.4. Comunicação

A interação entre os agentes tutores é viabilizada por algum mecanismo que promova a comunicação entre eles. Nesse sentido, a comunicação é aqui colocada como um suporte

para o desenvolvimento das atividades interativas. Há quatro necessidades básicas a serem consideradas no modelo de comunicação utilizado pelos agentes tutores na *SATA*:

(i) um meio de comunicação, no qual as mensagens possam trafegar;

(ii) uma linguagem de comunicação comum que assegure um entendimento mútuo entre o transmissor e o receptor de uma mensagem;

(iii) um modo assíncrono de comunicação;

(iv) tipos de endereçamento que contemplem as necessidades do sistema. Tais necessidades dependem do modelo de cooperação descrito na seção anterior (observe a ilustração na Tabela 3.1);

Tipo	Destino	Representação Gráfica
Direto	para um agente conhecido	
Global	para todos os agentes da sociedade	
Complementar	para os agentes pertinentes ao complemento de um subconjunto de agentes da sociedade	

Tabela 3.1: Tipos de endereçamentos

Mais adiante, no Capítulo 5, estes requisitos juntamente com a definição do modelo de comunicação são detalhados.

3.2.5. Definição Formal

Cada agente tutor possui um conjunto de habilidades utilizadas localmente na execução de tarefas. A habilidade diz respeito a um método para resolver uma determinada classe de tarefas. Um método corresponde a uma das três categorias de atividades pedagógicas: resolução de problema, diagnóstico e instrução. Uma tarefa, por sua vez, é composta pela informação da habilidade que pode executá-la e pelos recursos necessários a sua execução. Desta forma podemos definir uma habilidade como segue.

DEFINIÇÃO 3.2: Uma habilidade de um agente tutor AT é definida por $Hab = \langle Tipo, Identificações, Representação \rangle$, onde:

(i) *Tipo* refere-se às tarefas pedagógicas, tais como: ***Resolução, Diagnóstico*** e ***Instrução***.

(ii) *Identificações* é um conjunto definido por $Identificações = \langle Ident_1, Ident_2, \ldots, Ident_l \rangle$, onde cada $Ident_k \in Identificações$ objetiva atribuir um nome para a habilidade *Hab*.

(iii) *Representação* define um formato de entrada e saída pelo qual a habilidade pode ser acessada, e é dada por $Rep = \langle Entrada, Saída \rangle$.

Com respeito à propriedade de habilidade social atribuída aos agentes tutores em questão, isto é, a possibilidade deles interagirem entre si, há a noção de autoconhecimento e conhecimento social. Isso permite a um agente buscar outros agentes para cooperar sobre alguma tarefa, com base em tal conhecimento.

Cada agente tutor é uma entidade especializada em algum domínio específico, tendo o conhecimento necessário para realizar tarefas pedagógicas nesse domínio. Esses agentes desempenham papéis que incluem principalmente os de tutores inteligentes cooperativos, no momento de suas interações com um aprendiz. Desta forma define-se sintaticamente o autoconhecimento, o conhecimento social e o modelo do agente como segue.

DEFINIÇÃO 3.3: Seja AC o autoconhecimento de um agente tutor AT, definido por $AC = \langle AgenteId, LH \rangle$, onde: *AgenteId* é um identificador único para um agente e LH é uma lista indicando um conjunto finito de suas habilidades, sendo dada por $LH = \langle Hab_1, Hab_2, \ldots, Hab_n \rangle$.

DEFINIÇÃO 3.4: Seja CS o conhecimento social de agente tutor AT, definido por $CS = \langle AC_1, AC_2, \ldots, AC_n \rangle$, onde cada $AC_K \in CS$ é um conhecimento sobre os outros agentes em $SATA$. Observe que no CS não se inclui o seu autoconhecimento, representado por AC_i.

DEFINIÇÃO 3.5: Seja *MAG* o **modelo de um agente tutor** *AT*, definido por *MAG* = ⟨*ST, CS, AC*⟩, onde *ST* é um sistema tutor especializado num domínio, *CS* é o conhecimento social, *AC* é o autoconhecimento.

3.3. A Arquitetura de um Agente

A arquitetura concebida para um agente tutor na SATA é fruto direto do modelo de agente definido na seção anterior. Além do modelo, algumas outras arquiteturas de agente, desenvolvidas em outros trabalhos, foram utilizadas como base conceitual **[OMR93, SDB92, JW92, Oli93]**. O objetivo desta seção é então apresentar a arquitetura de um agente tutor, a qual é organizada em sistemas. Inicialmente, apresentamos estes sistemas e como eles interagem e, por fim, introduzimos rapidamente os componentes da arquitetura.

3.3.1. Sistemas da Arquitetura de um Agente

O modelo conceitual de arquitetura elaborado para um agente tutor define sua estrutura através de uma composição hierárquica de três sistemas principais, a saber: **tutor**, **social** e **de distribuição**. Cada um desses sistemas está exibido na Figura 3.3, sendo descrito logo em seguida. Os três sistemas são conceitualmente independentes e guardam uma interdependência funcional.

(i) **Sistema Tutor:** este sistema é responsável pela interação direta com o aprendiz humano, cabendo a ele a execução das atividades tutoriais nos termos tratados no âmbito do ambiente MATHEMA. Isoladamente, este sistema pode ser visto como um sistema tutor inteligente. É nesse sistema onde estão os conhecimentos que o agente possui para resolver os problemas no domínio de aplicação.

(ii) **Sistema Social:** este sistema é responsável pela viabilização do comportamento cooperativo entre os agentes tutores. Ele é composto por bases de conhecimento e mecanismos de raciocínio necessários para realizar tal comportamento.

(iii) **Sistema de Distribuição:** este sistema é responsável pela manipulação das mensagens enviadas e recebidas pelo agente tutor, através do meio de comunicação. Além disso, também é de sua responsabilidade a função de gerenciamento interno da distribuição das mensagens no agente tutor.

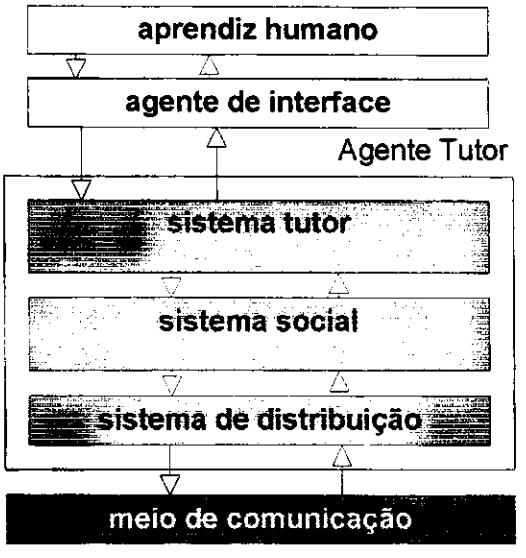


Figura 3.3: Sistemas da Arquitetura de um Agente Tutor

Para dar uma idéia de como esses sistemas interagem numa situação de cooperação, a dinâmica colocada em termos sucintos acontece do seguinte modo:

(i) Em meio a uma interação com o Aprendiz, o Sistema Tutor esbarra em uma de suas limitações quando da tentativa de resolver uma dada tarefa e, assim, evidencia a sua necessidade de pedir cooperação a outros agentes. Nesse momento, ele efetua uma requisição de cooperação ao Sistema Social, tomando como parâmetro uma determinada tarefa proveniente da sua interação com o aprendiz.

(ii) O Sistema Social, no intuito de efetivar a cooperação, faz um tratamento apropriado sobre esta tarefa, objetivando identificar e selecionar agentes habilitados a ajudá-lo a resolvê-la. De posse dessa seleção e concordância dos agentes escolhidos, o agente através do seu Sistema Social utiliza-se do Sistema de Distribuição como mediador, para interagir com tais agentes. Em suma, a interação entre esses dois sistemas ocorre da seguinte maneira: de

cima para baixo, há um pedidos de envio de mensagem; de baixo para cima, há notificações de recebimento de mensagem.

(iii) Após o retorno da execução da tarefa, o processo de cooperação finalmente chega ao fim com o Sistema Social retornando o resultado da execução da tarefa para o Sistema Tutor.

3.3.2. Componentes da Arquitetura de um Agente

Cada sistema da arquitetura é composto por módulos, como exibido na Figura 3.4, introduzidos a seguir. Falaremos aqui com mais detalhes do Sistema Tutor com o objetivo de fornecer uma base conceitual para o leitor, pois este sistema está fora do escopo e não será mais discutido na dissertação. Maiores detalhes sobre o Sistema Tutor podem ser encontrados em **[COS97]**.

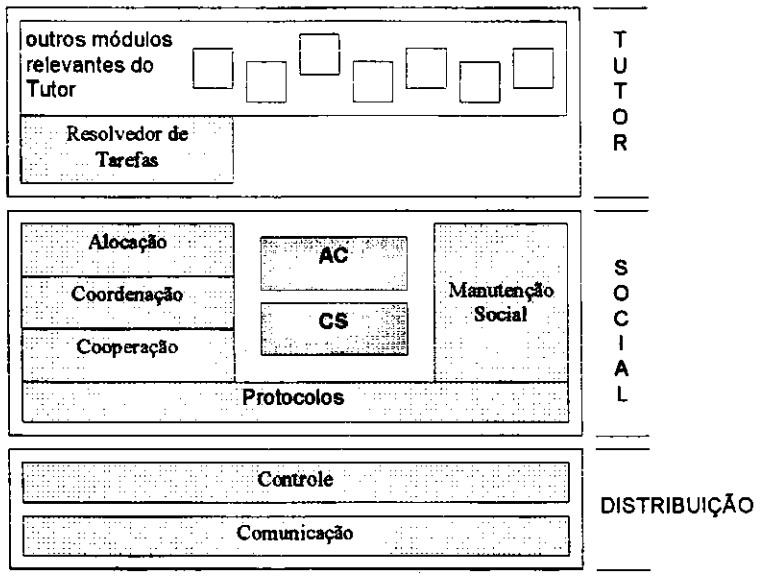


Figura 3.4: Componentes da Arquitetura de um Agente Tutor

Com relação à Figura 3.4, convém ressaltar que os módulos destacados com um tom mais escuro dizem respeito às bases de conhecimento de apoio à atividade cooperativa. Já os outros módulos representam mecanismos operacionais.

O Sistema de Distribuição é definido por dois módulos: o *Controle* é responsável pela distribuição interna das interações e o módulo de *Comunicação* é responsável pela

distribuição e coleta externa das mensagens, fazendo a mediação entre o Sistema de Distribuição e o ambiente de comunicação.

O Sistema Social é definido por seis módulos: *CS* (*Conhecimento Social*) é a estrutura na qual o agente representa explicitamente o conhecimento sobre as habilidades dos outros agentes da sociedade; *AC* (*AutoConhecimento*) é a estrutura de conhecimento de suas próprias habilidades, representando o que ele é capaz de fazer; *Alocação* é o módulo responsável pela seleção de agentes para cooperação; *Coordenação* é o sincronizador das atividades cooperativas; *Cooperação* é responsável pela viabilização e esgotamento de todas as possibilidades de cooperação; *Protocolos* é o módulo que gerencia os protocolos de interação; *Manutenção Social* é responsável pelas situações de manutenção da sociedade (entrada, saída e atualização de um agente).

O Sistema Tutor pode ser visto como um Sistema Baseado em Conhecimento (SBC). Nele há um conjunto de raciocinadores sobre as bases de conhecimento disponíveis. Além disso, existe uma entidade *Mediadora* em dois níveis: um responsável pelo monitoramento do processo de interação, incluindo funções de avaliação e de tomada de decisões pedagógicas e o outro dispõe de meta-conhecimento sobre o domínio, sendo assim responsável pelas ações reflexivas do agente. Detalhes sobre este sistema estão fora do escopo desta dissertação e podem ser encontrados em **[Cos97]**. Neste trabalho, entretanto, interessa-nos apenas perceber este SBC através de uma entidade aqui denominada *Resolvedor de Tarefas*.

O *Resolvedor de Tarefas* é dotado de mecanismos de decomposição para as tarefas pedagógicas incluindo: resolução de problemas, diagnóstico e instrução. Os agentes tutores possuem suas especialidades em resolver tarefas em domínios bem específicos. Cabe a este módulo identificar se uma determinada tarefa é de competência do agente tutor. Caso não seja, o Resolvedor de Tarefas tenta decompô-la antes de submetê-la ao Sistema Social.

Decomposição de tarefas é um método muito utilizado na IA. A decomposição no âmbito do MATHEMA depende do domínio de ensino sendo tratado. Há duas motivações para considerá-la no escopo deste trabalho:

4

Sistemas Social e de Distribuição

Neste capítulo detalhamos o Sistema Social e de Distribuição da arquitetura dos agentes tutores do MATHEMA. Para isso, enfoca-se individualmente cada um dos componentes dos sistemas e, logo a seguir, discutem-se os comportamentos emergentes na interação desses componentes.

4.1. Componentes do Sistema Social

No capítulo anterior introduzimos de modo genérico os componentes (módulos) que compõem o Sistema Social. A Figura 4.1 corresponde somente ao Sistema Social da arquitetura de um agente MATHEMA apresentado na sua totalidade na Figura 3.3. Nesta seção, detalhamos cada um desses componentes.

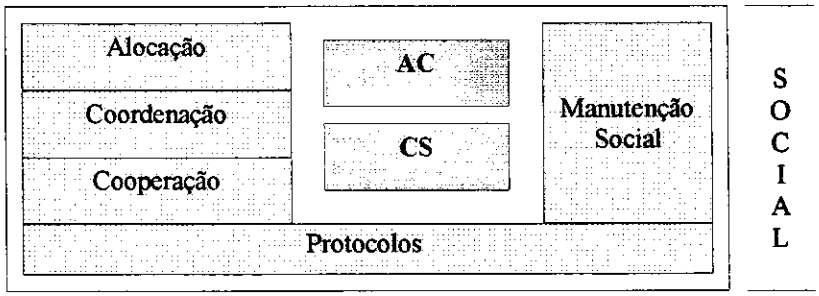


Figura 4.1: Componentes da Arquitetura no Sistema Social

Autoconhecimento (*AC*)

Estrutura representando o conhecimento que o agente possui sobre as suas próprias habilidades. Esta estrutura é utilizada quando um agente necessita decidir se ele possui as habilidades necessárias para resolver uma determinada tarefa pedagógica.

Conhecimento Social (*CS*)

Estrutura na qual o agente mantém o conhecimento sobre as habilidades dos outros agentes tutores da sociedade. Esta estrutura é usada quando o conhecimento do agente é insuficiente, ou mesmo ausente, para resolver uma dada tarefa. Assim, o agente usa o *CS* visando identificar na sociedade agentes que possam potencialmente engajar-se numa cooperação. Como visto no Capitulo 3, o *CS* é definido pelo conjunto de *AC*s dos outros agentes tutores. Devido ao dinamismo inerente à sociedade, esta estrutura pode se alterar no decorrer das interações. O CS pode estar inconsistente: incorreto, descrevendo habilidades inexistentes; ou incompleto, faltando a descrição de habilidades existentes.

Alocação

Módulo responsável pelo processo de seleção dos agentes identificados como aptos a resolver uma determinada tarefa que foi submetida pelo Sistema Tutor para cooperação. Seu funcionamento básico é o seguinte.

Como uma tarefa pode ser apresentada para cooperação de maneira decomposta, o módulo *Alocação* necessita selecionar agentes para cada uma das subtarefas envolvidas. Dada uma estrutura representando uma tarefa, que quando decomposta organiza-se numa lista $\langle t_1, t_2, \ldots, t_n \rangle$ oriunda do *Resolvedor de Tarefas* (cada t_i é uma subtarefa obtida após uma possível decomposição), o módulo *Alocação* opera sobre ela e produz como resultado uma lista de pares $\langle t_1, LA_1 \rangle \langle t_2, LA_2 \rangle \ldots \langle t_n, LA_n \rangle$ formada por cada uma das subtarefas envolvidas e uma correspondente lista de agentes aptos a executá-la. Para efeitos de ilustração, na Figura 4.2 apresenta-se um esquema de alocações feitas para as subtarefas obtidas na decomposição de uma determinada tarefa *T*.

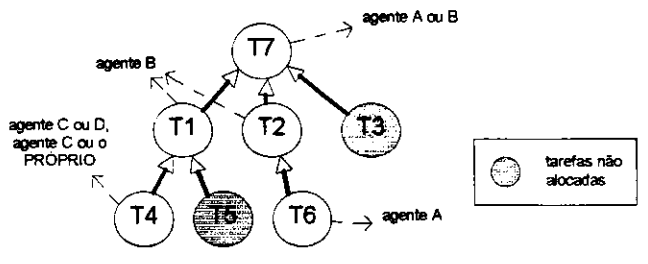


Figura 4.2: Representação gráfica das alocações para uma tarefa decomposta

Os círculos preenchidos representam as subtarefas não alocadas (T_3 e T_5). As setas tracejadas que saem dos círculos não preenchidos indicam os agentes alocados para a subtarefa em questão. Observe que para as subtarefas T_4 e T_7, mais de um agente foi alocado. É importante salientar que em T_4 é possível executar a subtarefa internamente pelo próprio agente tutor, o qual é priorizado. De acordo com a Figura 4.2, o processo de alocação pode selecionar um agente, um conjunto deles, ou até mesmo nenhum. Cada situação conduz a um tratamento particular, que é abordado ainda neste capítulo.

Coordenação

Módulo responsável por promover a execução da tarefa decomposta completamente, assegurando a coerência da sua realização. O módulo opera sobre a estrutura da decomposição da tarefa enviada pelo módulo de *Alocação* e, a partir da análise das dependências existentes entre as subtarefas, sincroniza (através de paralelização ou serialização) os pedidos de cooperação. É importante salientar que, além disso, o módulo deve garantir a distribuição e coleta correta dos recursos utilizados durante a realização de cada tarefa.

Na Figura 4.2, pode-se observar que a tarefa T_7 só pode ser executada após a realização das subtarefas T_1, T_2 e T_3. Da mesma forma, a subtarefa T_1 só pode ser disparada após a execução de T_4 e T_5; enquanto T_2 só após T_6. Pode-se observar também que as subtarefas T_3, T_4, T_5 e T_6 podem e devem ser realizadas de forma concorrente, pois são mutuamente independentes. A paralelização destes pedidos é possível devido à natureza das subtarefas, efetivamente realizável devido à natureza distribuída do sistema, e compensadora pois diminui o tempo de realização da cooperação sobre a tarefa T como um todo.

O módulo de *Coordenação* é funcionalmente justificado devido à necessidade de tratar possíveis pedidos de cooperação a partir de tarefas decompostas. Caso não haja decomposição, a função do módulo é trivial.

Cooperação

Módulo responsável por promover a execução das subtarefas através do encaminhamento delas para algum agente da sociedade. Há a possibilidade de um agente encaminhar uma subtarefa para si próprio. Quando isso acontece, através de uma chamada direta do módulo de *Cooperação* para o Sistema Tutor, diz-se que houve uma cooperação local. Considerando que a execução de uma subtarefa não é assegurada, cabe a esse módulo exaurir todas as possibilidades de cooperação.

Para o módulo de *Cooperação* é passado, pelo módulo de *Coordenação*, um par identificando uma tarefa e uma lista de agentes a ela associada. Dependendo do conteúdo desta lista, o módulo de *Cooperação* se depara com as três situações seguintes:

(i) **Nenhum agente foi selecionado (lista vazia):** Neste caso deve-se realizar um processo de negociação com a sociedade.

(ii) **Um único agente foi selecionado:** Deve-se contactar diretamente o agente selecionado para cooperar. Em caso de insucesso nesta cooperação, procede-se como na situação (i).

(iii) **Mais de um agente foi selecionado:** Chega-se num conflito e para resolvê-lo deve-se eleger um dentre os agentes selecionados para se proceder como na situação (ii); em caso de insucesso deve-se continuar elegendo agentes para contactá-los diretamente até não mais existir agentes aptos. Quando esta situação extrema acontecer, procede-se como na situação (i).

Para melhorar o desempenho, e no intuito de evitar possíveis ciclos no algoritmo envolvido, quando for necessário realizar uma negociação após tentativas fracassadas de cooperação direta (caso extremo da situação (iii)), é vantajoso negociar apenas com aqueles agentes ainda não excluídos. Nestes casos, ao invés de realizar uma difusão, é necessário apenas uma difusão seletiva (uso do endereçamento complementar, definido no Capítulo 3) para não mobilizar a sociedade toda. Esta situação é definida neste trabalho como negociação reduzida.

Tanto no processo de negociação quanto no contato direto de um agente por cooperação, o mecanismo utilizado para efetivar tais situações pelo módulo de *Cooperação* é a ativação de protocolos. Para cada situação há um protocolo adequado: com a negociação é ativado um protocolo de licitação; com o contato direto, o protocolo utilizado é o mestre-escravo. Observe que nas situações descritas acima, vários protocolos podem ser ativados durante o esgotamento de uma cooperação sobre uma determinada tarefa. Mais adiante discutiremos com mais detalhes os algoritmos utilizados no módulo de *Cooperação*.

Manutenção Social (MS)

Um dos requisitos básicos para garantir um funcionamento apropriado do módulo de *Alocação* é que todos os agentes tutores da sociedade estejam sempre com os seus *CS*s atualizados. Para manter a consistência do *CS*, cada agente tutor deve executar protocolos de manutenção. Estes protocolos estão relacionados as situações de entrada, saída e atualização de agentes na sociedade. O módulo *MS* é responsável pela inicialização desses protocolos. Estes protocolos bem como os ativados pelo módulo de cooperação ainda serão considerados mais adiante no Capítulo 5.

Protocolos

Como visto no Capítulo 2, um protocolo serve para estruturar as trocas de mensagens através da restrição das diferentes interações que um agente pode ter com os outros membros da sociedade considerando uma determinada situação de interação social, por exemplo a solução de um problema.

O módulo *Protocolos* é o responsável pela criação, ativação, destruição e execução dos diálogos. Veremos mais adiante que é possível a execução concorrente de protocolos no Sistema Social. Devido a isso foi necessária a utilização do conceito de diálogos. Um protocolo define um comportamento interativo, já um diálogo é uma entidade responsável pela viabilização de comportamento interativo contextualizado.

4.2. Comportamento do Sistema Social

Aqui, é discutido o comportamento dinâmico dos módulos do Sistema Social. Pretende-se com isso, apontar as funcionalidades que emergem no nível interno deste sistema quando seus módulos interagem.

No Sistema Social existem duas etapas de comportamento bem definidas para o processo cooperativo, quais sejam: (i) *mecanismo de raciocínio social*, (ii) *coordenação e cooperação*. Estas etapas são utilizadas exclusivamente no processo cooperativo e ocorrem apenas no agente que pede a cooperação; no agente cooperador estas etapas não figuram.

Outro comportamento evidente é o tratamento realizado para os diálogos. Tal situação não está ligada exclusivamente ao processo cooperativo e acontece nos dois lados de qualquer diálogo entre agentes na sociedade. Cada etapa é ilustrada através do acompanhamento das interações dos módulos envolvidos apresentando também os algoritmos utilizados.

Além dos comportamentos já citados, deve-se observar a existência do comportamento de manutenção. Tal comportamento é caracterizado através do acompanhamento das interações do módulo de *Manutenção Social* com outros módulos da arquitetura. Apenas no Capítulo 5, quando se descreve os protocolos de manutenção (entrada e saída), é que este comportamento é tratado em detalhes.

4.2.1. Mecanismo de Raciocínio Social

Os agentes possuem um mecanismo de raciocínio para operar sobre o modelo de suas capacidades e o modelo das capacidades dos outros agentes. Para tal, ele conta com três componentes: *AC*, *CS* e *Alocação*.

Antes de efetuar um pedido de cooperação, uma determinada tarefa é preparada pelo *Resolvedor de Tarefas*. O preparo consiste em, quando possível, decompô-la para descrevê-la num padrão entendido pelo Sistema Social. No Capítulo 5 apresentamos o formato de descrição de uma tarefa.

O mecanismo de Raciocínio Social é ilustrado na Figura 4.3. Quando o Sistema Tutor detecta a necessidade de cooperação, via *Resolvedor de Tarefas*, ele efetua um pedido de viabilização dessa cooperação ao Sistema Social. O parâmetro que acompanha este pedido é uma tarefa pedagógica (problema, diagnóstico, instrução). Este parâmetro é organizado conforme uma lista $\langle tarefa_1 \rangle \langle tarefa_2 \rangle \ldots \langle tarefa_n \rangle$.

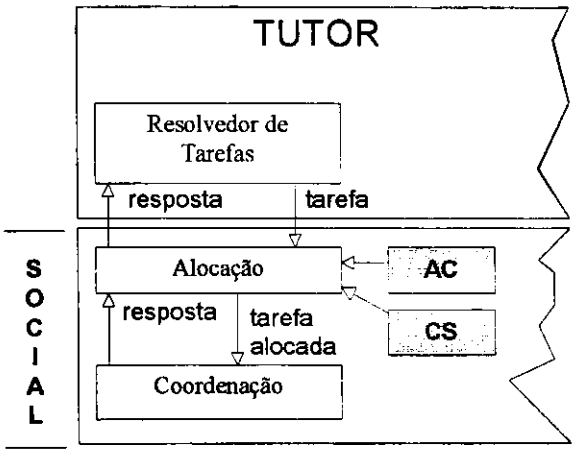


Figura 4.3: Comunicação interna entre os módulos na etapa de Raciocínio Social

O módulo de *Alocação* executa a seleção para cada tarefa da lista. O algoritmo abaixo descreve o processo.

```
alocação.executa( lista de tarefas )
para cada tarefa[i]
      se casa( AC, tarefa[i] )
            aloca o próprio agente para a tarefa[i]
      fim-se
      para cada AC-j do CS
            se casa( AC-j, tarefa[i] )
                  aloca o agente do AC-j para a tarefa[i]
            fim-se
      fim-para
fim-para
```

O algoritmo é simples, a complexidade envolvida depende basicamente da estratégia de *casamento*, que depende das necessidades do sistema. Por exemplo, poder-se-ia considerar a utilização de biblioteca de sinônimos ou regras de equivalência. O *casamento* é realizado sobre a habilidade de cada $AC \in CS$ e a habilidade necessária para realizar cada tarefa.

Uma vez concluído o processo de alocação, a estrutura abaixo, representando a tarefa alocada, é passada para o módulo de *Coordenação*. Esta estrutura possui a seguinte configuração: ⟨tarefa$_1$⟩ ⟨lista-de-agentes$_1$⟩, ⟨tarefa$_2$⟩ ⟨lista-de-agentes$_2$⟩, ..., ⟨tarefa$_n$⟩ ⟨lista-de-agentes$_n$⟩.

Entretanto, quando a tarefa não se apresenta na forma decomposta, a situação fica reduzida a um caso particular, no qual a lista possui apenas um elemento. Portanto, recai-se no seguinte esquema: ⟨tarefa⟩ → alocação → ⟨tarefa⟩ ⟨lista-de-agentes⟩.

O resultado final do processo de coordenação, que é a resposta, como ilustrado na Figura 4.3, é enviada para o Sistema Tutor através do *Resolvedor de Tarefas*. Esta resposta pode ser o resultado da execução da tarefa ou uma indicação de insucesso. Estas duas situações são detalhadas na seção seguinte.

4.2.2. Coordenação e Cooperação

O funcionamento dos módulos de *Coordenação* e *Cooperação* se dá de forma integrada, e, em certos casos, concorrente e sincronizada. A Figura 4.4 ilustra as interações entre estes dois módulos.

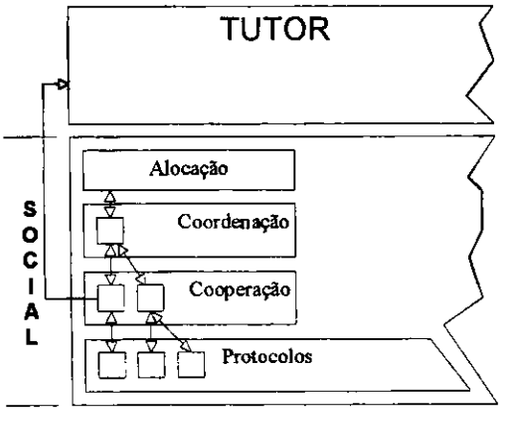


Figura 4.4: Visão integrada envolvendo Coordenação e Cooperação

Quando uma tarefa alocada é tratada pelo módulo de *Coordenação*, cria-se uma instância de coordenação com um contexto particular associado relativo à tarefa em questão. A utilização de instâncias é necessária devido à possibilidade da manipulação de

vários pedidos concorrentes de cooperação pelo Sistema Social. A possibilidade de coordenações concorrentes gera uma necessidade de controle. Uma instância de coordenação realiza as funções atribuídas ao módulo de *Coordenação* descritas na Seção 4.1.

Antes de discutirmos os algoritmos utilizados, é necessário observar que uma tarefa assume três estados distintos, quais sejam: NÃO EXECUTADA, EXECUTADA, EM ANDAMENTO. Estes estados são utilizados como palavras-chave nos algoritmos descritos nesta seção.

O primeiro algoritmo, descrito abaixo, realiza, dentre outras coisas, as inicializações necessárias para a instância de coordenação, incluindo a atribuição do estado de NÃO EXECUTADA para todas as tarefas. Além disso, o algoritmo realiza a chamada do método **`coordenação.dispara`** para iniciar as primeiras cooperações possíveis.

```
coordenação.executa( lista de tarefas alocadas )
coordenando = VERDADEIRO
insucesso = FALSO
tarefas = lista de tarefas alocadas
para cada tarefas[i]
      tarefas[i].condição = NÃO EXECUTADA
fim-para
coordenação.dispara() /* ativa o algoritmo de disparo */
espera enquanto coordenando = VERDADEIRO
```

O algoritmo, após a realização dos primeiros disparos, entra num estado de espera, no qual permanece até o encerramento das atividades cooperativas sobre a tarefa em questão. O algoritmo de disparo referenciado anteriormente é como segue.

```
coordenação.dispara()
para cada tarefas[i]
      se tarefas[i].condição = NÃO EXECUTADA
            se tarefas[i] estiver pronta para ser executada
                  cria uma instância de cooperação
                  para encaminhá-la
            fim-se
      fim-se
fim-para
```

O algoritmo de disparo analisa as tarefas no estado NÃO EXECUTADA (inicialmente todas estão neste estado) e durante a análise, para cada tarefa pronta para execução cria uma instância de cooperação para tratá-la. Uma tarefa está pronta para ser executada,

quando ocorre uma das duas situações seguintes: (i) ela não possui dependências, ou; (ii) ela possui dependências, mas todas as tarefas das quais ela depende já foram executadas.

Uma instância de coordenação deve criar as instâncias de cooperação concorrentes, como indicado no algoritmo de disparo nas linhas sublinhadas e com o fonte em itálico. Cada instância, tal como ocorre com as instâncias de coordenação, possui um contexto particular a ela associado. Quando uma instância de cooperação é criada, a seguinte tupla é repassada para essa instância: ⟨tarefa⟩ ⟨lista-de-agentes⟩ ⟨id. da tarefa⟩.

Quando uma instância de cooperação finaliza o seu trabalho, ela notifica a instância de coordenação que a criou. Neste momento, há a possibilidade de existirem várias outras instâncias de cooperação em andamento. É necessário que a instância de coordenação identifique para qual tarefa no estado EM ANDAMENTO corresponde a notificação recebida. O campo ⟨id. da tarefa⟩ existe para viabilizar este controle e é um valor numérico obtido do índice da tarefa na lista. A execução de uma instância de cooperação é tratada inicialmente através do algoritmo abaixo.

```
cooperação.executa( tarefa alocada, identificador da tarefa )
se há algum agente alocado para a tarefa
      cooperação.encaminha( tarefa, lista dos alocados )
      se algum agente executou a tarefa com SUCESSO
            comunica este sucesso para a instância de coordenação
            que o criou e finaliza a execução do algoritmo
      fim-se
fim-se

ativa um diálogo de licitação passando
como parâmetro a habilidade necessária para a execução
da tarefa e a lista dos agentes já contactados para
realizar negociação reduzida

se algum agente se propuser a executar a tarefa anunciada
      cooperação.encaminha( tarefa, lista dos proponentes )
      se algum agente executou a tarefa com SUCESSO
            comunica este sucesso para a instância de coordenação
            que o criou finaliza a execução do algoritmo
      fim-se
fim-se

comunica a situação de INSUCESSO para a instância de coordenação
que o criou
notifica a situação de INSUCESSO para SEH
finaliza a execução do algoritmo
```

O algoritmo **cooperação.executa** esgota todas as possibilidades de cooperação e consta das seguintes etapas:

(i) Realização de uma tentativa de encaminhamento com a lista de agentes alocados. Em caso de sucesso (algum agente cooperou), comunica-se o fato à instância de coordenação e finaliza-se a execução do algoritmo.

(ii) Caso não se obtenha sucesso na etapa (i), acontecerá uma licitação. O mesmo acontece, caso a etapa (i) não ocorra; esse caso pode decorrer do fato da lista estar vazia.

(iii) Se o resultado da etapa (ii) for bem sucedido (algum agente se propôs a cooperar), ocorre uma tentativa de encaminhamento, só que agora com a lista de proponentes. Em caso de sucesso, procede-se como na etapa (i).

(iv) Caso não se obtenha sucesso na etapa (iii), ou caso ela não aconteça (nenhum agente se propôs), comunica-se o insucesso para a instância de coordenação, para logo após finalizar a execução.

O algoritmo de encaminhamento usado para contactar agentes potencialmente cooperantes referenciado em dois momentos, antes e após a licitação, no algoritmo anterior, é descrito como segue.

```
cooperação.encaminha( tarefa, lista de agentes )
se há mais de um agente na lista
      ordena a lista de agentes segundo o critério de afinidade
fim-se
para cada agente[i] da lista
      se o agente[i].identificador = AC.identificador
            efetua um pedido interno de cooperação
      senão
            cria um diálogo mestre-escravo
            passando como parâmetro o identificador do agente[i]
            e a tarefa
      fim-se
      se houve sucesso na cooperação
            retorna SUCESSO
      fim-se
fim-para
retorna INSUCESSO
```

De forma sucinta, o funcionamento do algoritmo **cooperação.encaminha** é o seguinte:

(i) Inicialmente acontece um tratamento para uma possível situação de conflito[1] que ocorre quando o encaminhamento é efetuado para mais de um agente da sociedade. Detalhes sobre o critério de afinidade, que se baseia no modelo multidimensional, podem ser encontrados em **[Cos97]**.

(ii) Após a ordenação dos agentes, caso se constate a necessidade, o algoritmo passa a realizar as tentativas de encaminhamento da tarefa. Quando o encaminhamento é social, é feito através da criação de um diálogo *mestre-escravo*, no caso de encaminhamento local, acontece um pedido direto para o Sistema Tutor.

(iii) Quando uma cooperação é bem sucedida, decorrente de alguma das duas situações acima, o algoritmo finaliza a sua execução e retorna um sinal de sucesso para o algoritmo **cooperação.executa.**

(iv) No caso de todos os encaminhamentos serem mal-sucedidos, retorna uma notificação de insucesso para o algoritmo **cooperação.executa**[2].

Quando uma notificação de insucesso chega à instância de coordenação, ela aparece no formato da seguinte tupla: ⟨id. da tarefa⟩ ⟨descrição do erro⟩. O insucesso na cooperação obtido para uma determinada subtarefa acarreta no insucesso da cooperação para a tarefa toda. Quando isto acontece, é possível que haja algumas cooperações com o estado EM ANDAMENTO. Neste caso, adota-se a estratégia[3] de esperar a finalização dessas tarefas. Tal estratégia tem a prerrogativa de ser prática, embora apresente a desvantagem de gerar um consumo desnecessário de recursos nos agentes escravos, que porventura estejam executando as tarefas no estado EM ANDAMENTO. O algoritmo a

---

[1] O conflito acontece quando há mais de um agente alocado para a tarefa (antes da licitação) ou quando há mais de um agente proponente (após a licitação).

[2] De fato, esta situação dentro do contexto do MATHEMA implica numa situação que recorre à *SEH* (Sociedade de Especialistas Humanos).

[3] Em **[UG92]** adota-se a estratégia de anulação das cooperações em andamento. Isto implica a necessidade de interações (*Goal Annulation*) com os agentes escravos com o objetivo de anular as execuções das tarefas em andamento.

seguir apresenta o tratamento da situação particular de insucesso no nível de coordenação.

```
coordenação.insucesso( id. da tarefa, descrição do erro )
se ainda não houve insucesso
      insucesso = VERDADEIRO
      armazena a descrição do erro
      tarefas[id. da tarefa].condição = NÃO EXECUTADA
fim-se
para cada tarefas[i] da lista
      se tarefas[i].condição = EM ANDAMENTO
            finaliza a execução do algoritmo
      fim-se
fim-para
coordenando = FALSO
```

O algoritmo **coordenação.insucesso** realiza o tratamento adequado para viabilizar a estratégia de espera mencionada anteriormente. Para tal, armazena a descrição do erro que serve de referência para o Sistema Tutor e muda a condição da tarefa indicada para o estado NÃO EXECUTADA. A última operação efetuada é a verificação se ainda há alguma tarefa com o estado EM ANDAMENTO. Caso não haja mais nenhuma nessa condição, o processo de coordenação é finalizado. Do contrário, continua a espera das outras finalizações.

Quando uma notificação de sucesso chega à instância de coordenação, ela aparece no formato da tupla abaixo. ⟨id. da tarefa⟩ ⟨resultado da execução⟩. Neste caso, o algoritmo **coordenação.sucesso** é executado. Note que os tratamentos necessários para implementar a estratégia de espera são similares.

```
coordenação.sucesso( id. da tarefa, resultado da execução )
se ainda não houve insucesso
      armazena o resultado da execução
      tarefa[id. da tarefa].condição = EXECUTADA
      coordenação.dispara()
fim-se
para cada tarefa[i] da lista
      se tarefa[i].condição = EM ANDAMENTO
            finaliza a execução do algoritmo
      fim-se
fim-para
coordenando = FALSO
```

Se não ocorreu nenhum insucesso anteriormente, o algoritmo **coordenação.sucesso** armazena o resultado da execução da tarefa indicada, e muda o seu estado para EXECUTADA. Logo após, o algoritmo de disparo é novamente invocado para

instanciar novas cooperações que dão seguimento ao trabalho integrado dos módulos de coordenação e cooperação.

O funcionamento dos dois módulos, considerando as situações de sucesso e insucesso, pode ser resumido como segue: (i) quando houver algum insucesso de cooperação e todas as tarefas no estado EM ANDAMENTO finalizarem sua execução (estratégia de espera), o que é repassado para o módulo *Alocação* é a descrição do erro que conduziu a esse resultado; (ii) quando todas as tarefas forem executadas com sucesso; neste outro caso, o resultado é passado para o módulo *Alocação* no seguinte formato: $\langle tarefa_1 \rangle$ $\langle resultado_1 \rangle$, ..., $\langle tarefa_n \rangle$ $\langle resultado_n \rangle$.

4.2.3. Tratamento dos Protocolos

Como ja foi declarado na Seção 4.1, devido à concorrência inerente de algumas situações do Sistema Social, é possível a existência concorrente de diálogos. Cada diálogo é dotado de um contexto particular (observe D1, D2 e Dn na Figura 4.5). Dentre as inúmeras possibilidades, algumas situações podem servir de exemplo, a saber: duas licitações referentes a duas tarefas paralelas; entrada de dois ATs simultaneamente na sociedade; etc.

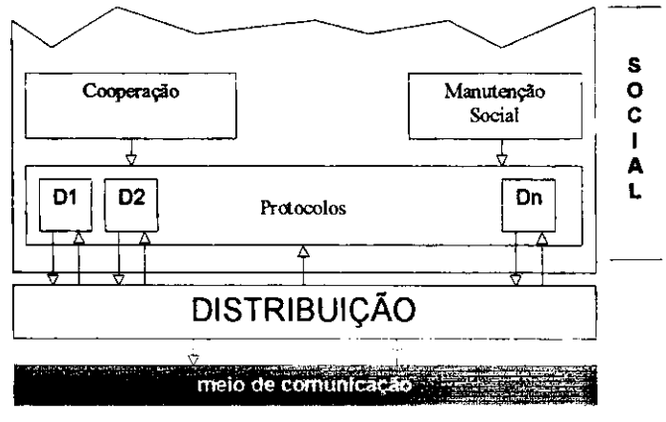


Figura 4.5: Diálogos

Para cada um dos protocolos preestabelecidos, existem duas implementações envolvidas: uma para o lado que inicia o diálogo e a outra para o lado que responde à requisição de interação. No primeiro caso, os diálogos são sempre criados e ativados a partir do

módulo *Cooperação* ou *Manutenção Social*; no segundo caso, eles são criados e ativados a partir do módulo *Controle* após a recepção da primeira mensagem enviada pelo diálogo que iniciou a interação. Após encerrar a sua execução, um diálogo comunica o fato ao módulo *Protocolos* para que ele seja destruído. A operacionalização de um protocolo é efetivada pelos diálogos. Os algoritmos referentes a cada protocolo, nos dois lados, são apresentados no Capítulo 5.

4.3. Componentes do Sistema de Distribuição

Os tratamentos finais de uma interação são realizados no Sistema de Distribuição. A Figura 4.6 corresponde somente a este sistema, cujos componentes são detalhados nesta seção. Observe que dois módulos são definidos, a saber: **Controle** e **Comunicação.**

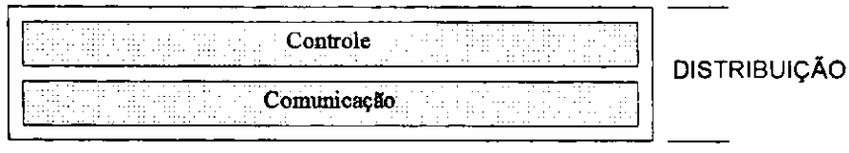


Figura 4.6: Componentes da Arquitetura no Sistema de Distribuição

Controle

Módulo responsável pela mediação entre o Sistema Social (módulo *Protocolos*) e o módulo de *Comunicação*. Suas funções são as seguintes:

(i) encaminhar as mensagens recebidas para serem tratadas pelo diálogo apropriado;

(ii) verificar a consistência das mensagens recebidas (semântica);

(iii) descartar as mensagens atrasadas;

Para encaminhar as mensagens para os diálogos apropriados, deve-se incluir nas mensagens informações adicionais. No Capítulo 5, quando a sintaxe da linguagem de interação for apresentada, estas informações adicionais serão introduzidas. As mensagens atrasadas são descartadas em duas situações: ou o diálogo responsável pelo seu

tratamento não existe mais; ou o tempo estabelecido para recebê-la, após uma difusão (*broadcast*) expirou (*time-out*).

Comunicação

Este módulo implementa o modelo de comunicação adotado para os agentes MATHEMA e é encarregado: (i) da distribuição e coleta das mensagens através do meio de comunicação, fazendo assim a mediação deste último com o módulo de *Controle*; (ii) da verificação sintática das mensagens recebidas.

A visão do módulo *Comunicação* cria uma abstração importante na independência de todo o mecanismo existente acima dele com relação ao meio e ao mecanismo de comunicação utilizados.

4.4. Comportamento do Sistema de Distribuição

O comportamento dinâmico dos módulos do Sistema de Distribuição é elucidado, conforme ilustrado na Figura 4.7, nos dois sentidos: do Sistema Social para o meio de comunicação (de cima para baixo) e vice-versa (de baixo para cima).

No sentido de cima para baixo, a função do *Controle* é nula, ele só repassa a mensagem para o módulo de *Comunicação*. O único módulo funcional neste sentido é o *Comunicação*, pois viabiliza o envio das mensagens para os agentes indicados como destinatários, conforme os modos de endereçamento descritos no Capítulo 3.

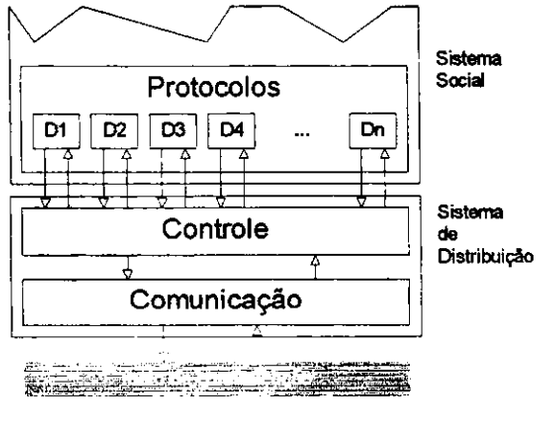


Figura 4.7: Arquitetura funcional do Sistema de Distribuição

No sentido de baixo para cima, quando uma mensagem é recebida, o módulo de *Comunicação* verifica a sua sintaxe e, em caso de sucesso, passa-a para o módulo de *Controle*. Esse módulo, a partir de então, encaminha a mensagem para o diálogo, identificado na própria mensagem, conforme mostrado no algoritmo abaixo.

```
controle.trata_chegada( mensagem )
se a mensagem não tem um diálogo destino determinado
      criar um diálogo apropriado
      se sucesso
            passa a mensagem para ser tratada por ela
            retorna OK
      senão
            retorna ERRO
      fim-se
senão
      para cada diálogo [i] em execução
            se mensagem.id-ip-dest = protocolo[i].id
                  passa a mensagem para ser tratada por ela
                  retorna OK
            fim-se
      fim-para
      retorna CHEGADA_ATRASADA
fim-se
retorna ERRO
```

Se a mensagem não foi endereçada para nenhum diálogo específico (primeira interação de um protocolo), então cria-se um diálogo antes de repassar a mensagem. O diálogo é construído a partir de informações contidas na própria mensagem, conforme apresentado no Capítulo 5. Por outro lado, se a mensagem foi endereçada para um diálogo conhecido, é necessário identificá-lo antes de repassar a mensagem.

5

Protocolos e Linguagem de Interação

Este capítulo detalha os protocolos utilizados na sociedade MATHEMA, bem como a sintaxe e a semântica da linguagem de interação utilizada na troca de mensagens entre os agentes. Cada protocolo é detalhado através de uma descrição passo-a-passo das interações que ocorrem entre os agentes e entre os módulos da arquitetura. A sintaxe da linguagem de interação é descrita através do uso de diagramas sintáticos. Comentários adicionais sobre a semântica da linguagem também são apresentados.

5.1. Protocolos de Interação

O modelo de Cooperação entre agentes concebido neste trabalho conduz à necessidade do estabelecimento de um conjunto de protocolos para disciplinar as atividades interativas envolvidas.

Os protocolos de interação são destinados ao tratamento das situações de interação entre os *AT*s na sociedade. No presente trabalho, tais protocolos estão classificados em duas categorias: protocolos de cooperação e protocolos de manutenção. A seguir, cada protocolo é descrito e ilustrado, destacando-se apenas os módulos necessários do Sistema Social.

5.1.1. Protocolos de Cooperação

Os protocolos de cooperação regem as situações de cooperação. Com relação à cooperação, existem duas situações: *mestre-escravo* e *licitação*. Os protocolos de

cooperação, no lado do agente que pede cooperação, são ativados pelo módulo *Cooperação* através de alguma instância de cooperação.

Mestre-Escravo

O protocolo *mestre-escravo* rege uma situação direta de cooperação e possui uma natureza 1 para 1. Tal protocolo é empregado quando se sabe *a priori* com quem cooperar. Quando há vários agentes aptos a cooperar, definidos no módulo de *Alocação* ou após uma negociação com a sociedade, utiliza-se um protocolo *mestre-escravo*, que é ativado pelo módulo *Cooperação* após a eleição de um agente dentre os habilitados.

No protocolo *mestre-escravo* existem dois papéis evidentes: o do mestre e o do escravo. Na Figura 5.1 ilustra-se o que ocorre em tal protocolo e logo após detalha-se o seu funcionamento. Observe que as setas orientadas e indexadas correspondem a cada um dos passos descritos a seguir. Esta estratégia de apresentação é adotada para os protocolos que ainda serão detalhados.

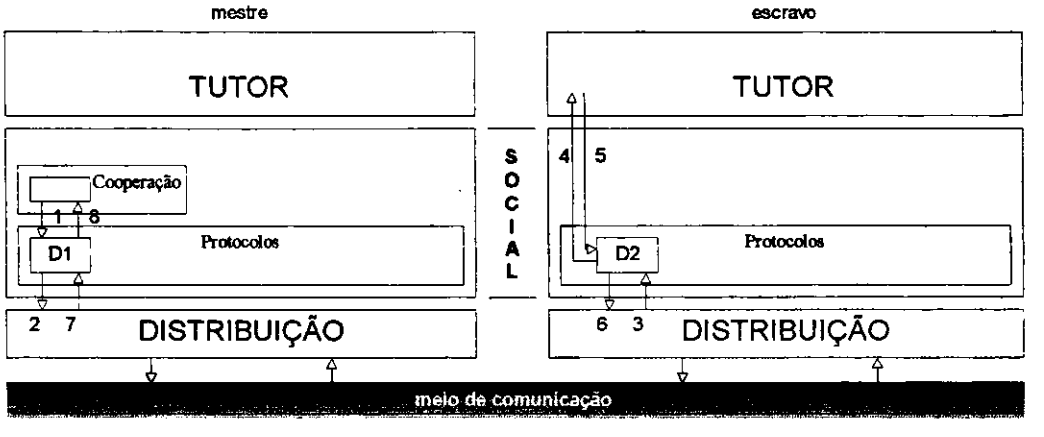


Figura 5.1: Protocolo Mestre-Escravo

(1) No lado mestre, uma determinada instância de cooperação cria um diálogo mestre-escravo. Os parâmetros utilizados na criação são: a **TAREFA** para a qual se deseja cooperação e o **IDENTIFICADOR** do agente escravo.

(2) O primeiro passo do protocolo, executado por D_1, é o envio de um **PEDIDO** para o agente escravo. A mensagem, antes de ser enviada, é preenchida com a **TAREFA** que foi passada como parâmetro na criação de D_1.

(3) No agente escravo, um outro diálogo, rotulada por D_2, responsável pelo tratamento do protocolo mestre-escravo no lado escravo, é criada após a recepção do **PEDIDO**. Logo após a criação de D_2, o Sistema de Distribuição repassa a mensagem recebida para ser tratada adequadamente.

(4) O único tratamento realizado por D_2 é o encaminhamento da **TAREFA** para o Sistema Tutor.

(5) O Sistema Tutor tenta executar a **TAREFA** e o **PRODUTO** obtido da execução é retornado para D_2, podendo este ser tanto o **RESULTADO** requerido pelo lado mestre, quanto a descrição de algum **ERRO** que porventura tenha acontecido durante o processamento.

(6) D_2 envia uma **RESPOSTA** para o agente mestre. A mensagem enviada é preenchida com o **PRODUTO** obtido da execução da tarefa pelo Sistema Tutor. Após o envio, D_2 finaliza a sua execução.

(7) No agente mestre, D_1 recebe a **RESPOSTA** enviada pelo agente escravo.

(8) D_1 encaminha para a instância de cooperação que a criou o **PRODUTO** da cooperação e finaliza a sua execução. Neste ponto, a instância de cooperação reassume o controle e continua o seu processamento.

Licitação

A Licitação é um protocolo 1 para N, ou seja, estabelece uma interação de um agente com um grupo de agentes. Tal protocolo é utilizado quando não se conhece *a priori* um agente com quem se possa cooperar. Uma Licitação acontece quando o *conjunto de agentes* selecionado pelo módulo *Alocação* é vazio, ou ainda quando há insucessos de contatos diretos para cooperação através do uso do protocolo *mestre-escravo*. Tal situação de insucesso é prevista, pois assume-se neste trabalho que o *CS*, como visto na Seção 4.1, pode não estar correto e/ou completo, ou seja, um agente alocado pode não corresponder às expectativas indicadas pelo *CS* do agente que o invocou.

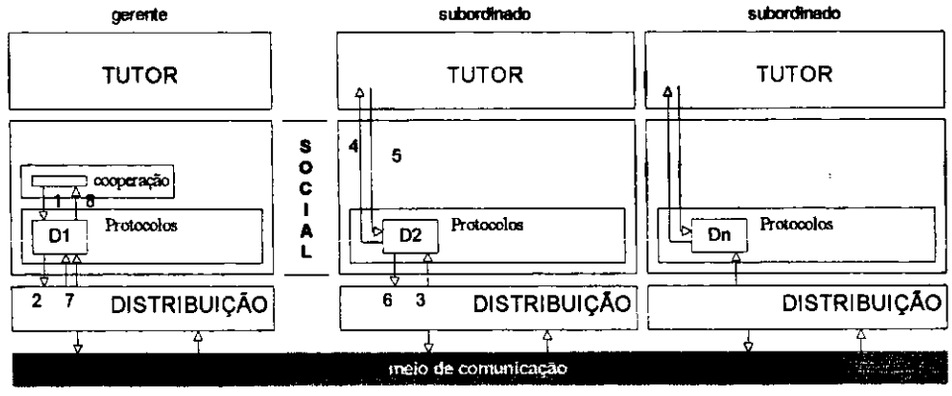


Figura 5.2: Protocolo de Licitação

O protocolo de Licitação é relativo a uma das etapas do Protocolo de Redes Contratuais apresentado no Capítulo 2, na licitação há dois papéis distintos: o do gerente, que inicia a licitação, e o do subordinado (normalmente, mais de um), contatado pelo gerente. O processo acontece como apresentado na Figura 5.2 e detalhado a seguir.

(1) No lado gerente, uma instância de cooperação cria um diálogo, rotulado por D_1, relativa à situação de licitação. Os parâmetros utilizados na criação são: a **TAREFA** para a qual se quer cooperação e uma lista com os **IDENTIFICADORES** dos agentes que não devem ser utilizados na licitação, caso esta lista seja vazia, todos os agentes da sociedade devem ser contatados.

(2) Aqui, o primeiro passo do protocolo executado por D_1 é o envio de um **ANÚNCIO**. A mensagem antes de ser enviada é preenchida com a **TAREFA** em questão.

(3) Em todos os agentes destinatários do **ANÚNCIO**, cada diálogo (D_2, ..., D_n), relativa ao tratamento da situação de licitação no lado subordinado, é criada após a recepção da mensagem. Logo após a criação, em cada agente, o Sistema de Distribuição repassa o **ANÚNCIO** recebido para ser tratado pela D_i recém criada.

(4) No intuito de analisar sua própria competência sobre a **TAREFA** anunciada, cada D_i encaminha a **TAREFA** contida na mensagem para o Sistema Tutor.

(5) O Sistema Tutor analisa a **TAREFA** e retorna um **PARECER**. Este parecer, é um valor que indica o percentual da **TAREFA** que pode ser executada através do agente. Quando o agente pode executar a **TAREFA** completamente, o valor é 100; quando não pode realizá-la, o valor é 0. Um valor entre 0 e 100 indica uma competência parcial e a existência de uma decomposição durante o processo de análise. Quando isto acontece, o **PARECER** é obtido através da expressão a seguir:

$$parecer = \frac{n^{\circ}\ de\ tarefas\ alocadas\ com\ sucesso}{n^{\circ}\ de\ tarefas\ resultantes\ da\ decomposição} \times 100$$

(6) Se o **PARECER** obtido no processo de análise for maior que zero, o D_i do agente subordinado envia para o agente gerente uma **PROPOSTA** com este valor. Após isso, cada D_i finaliza a sua execução.

(7) Após o envio do **ANÚNCIO**, o gerente através de D_1 passa a receber as **PROPOSTAS** enviadas pelo agentes voluntários, aptos a cooperar. O **IDENTIFICADOR** destes agentes e o **PARECER** dado por cada um são inseridos numa lista destinada à catalogação das propostas. O gerente continua a receber as **PROPOSTAS** durante um determinado intervalo de tempo.

(8) Quando o tempo estabelecido para a recepção expira, D_1 deixa de receber as **PROPOSTAS** e encaminha para a instância de cooperação apropriada a lista dos agentes que se propuseram a cooperar. Com isso, D_1 finaliza a sua execução e a instância de cooperação continua com o seu processamento.

5.1.2. Protocolos de Manutenção

Os protocolos de manutenção regem as situações de manutenção. Como já mencionado anteriormente, a manutenção é necessária, pois é desejável que todos os agentes estejam sempre com os seus *CS*s consistentes. Três situações devem ser tratadas: entrada, saída e atualização de um agente. Tanto na entrada quanto na saída de um agente é necessário que toda a sociedade seja notificada. Os protocolos de manutenção são ativados pelo módulo de Manutenção Social.

Entrada

O protocolo de Entrada estabelece uma interação de um agente com a sociedade (1 para N). Na entrada acontece uma troca de ACs entre o agente que está entrando e o restante da sociedade. O protocolo de Entrada deve ser o primeiro a ser executado num agente e todos os pedidos de interação devem ser recusados pelo módulo de *Controle* antes da finalização deste protocolo. A Figura 5.3 ilustra este protocolo que é detalhado como segue.

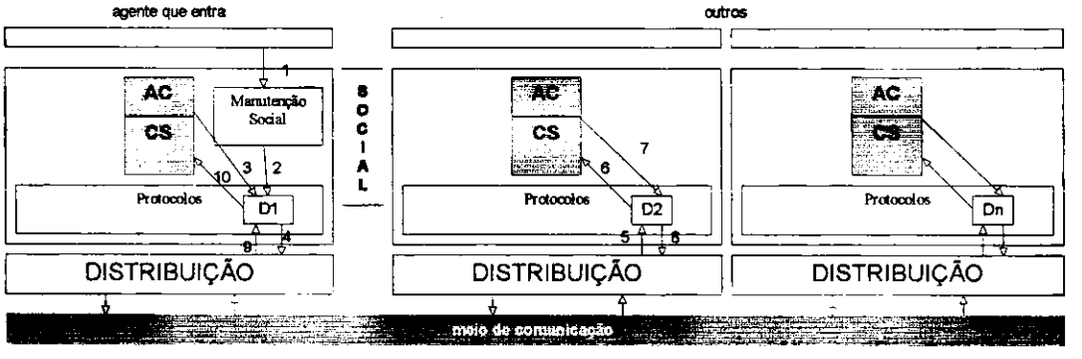


Figura 5.3: Protocolo de Entrada

(1) Quando um novo agente entra na sociedade, o Sistema Tutor pede ao Sistema Social, através do módulo Manutenção, que inicie o protocolo de entrada;

(2) O módulo *MS* cria um diálogo, rotulada por D_1 responsável pelo tratamento da situação no lado do agente que está entrando. O *AC* e uma referência para o *CS*, são passados como parâmetros na criação de D_1.

(3) D_1 prepara uma mensagem para enviar a toda sociedade. O AC passado como parâmetro na criação é colocado na mensagem.

(4) D_1 envia a mensagem preparada, uma **APRESENTAÇÃO**, para todos os agentes da sociedade.

(5) Em todos os agentes da sociedade, cada diálogo (D_2, ..., D_n), relativa ao tratamento da situação de entrada nos agentes que já estão na sociedade, é criada após a recepção da mensagem. Logo após a criação, em cada agente, o Sistema de Distribuição repassa a **APRESENTAÇÃO** recebida para ser tratada pela D_1 recém-criada.

(6) Cada diálogo (D_2, ..., D_n) cadastra em seu CS o AC do agente que está entrando.

(7) Cada diálogo (D_2, ..., D_n) prepara uma mensagem com o AC do agente em questão.

(8) Cada diálogo (D_2, ..., D_n) envia uma **APRESENTAÇÃO** de volta.

(9) Após o envio da **APRESENTAÇÃO** inicial, o agente que entra, através de D_1, passa a receber as **APRESENTAÇÕES** enviadas durante um determinado intervalo de tempo.

(10) Os ACs recebidos são cadastrados no CS do agente que entra.

Saída

O agente que sai deve avisar a todos da sua saída, através do protocolo de saída que, como no protocolo de entrada, estabelece uma interação de um agente com a sociedade (1 para N). Deve-se observar que um diálogo de saída só pode ser executada de forma exclusiva, ou seja, quando nenhum outro diálogo, de qualquer tipo de protocolo, estiver em execução. Durante a execução do protocolo de saída, todos os pedidos de criação de diálogo devem ser recusados. Após a finalização do diálogo de saída, nenhum outro diálogo deve ser executado no agente. O protocolo ilustrado na Figura 5.4 é detalhado a seguir.

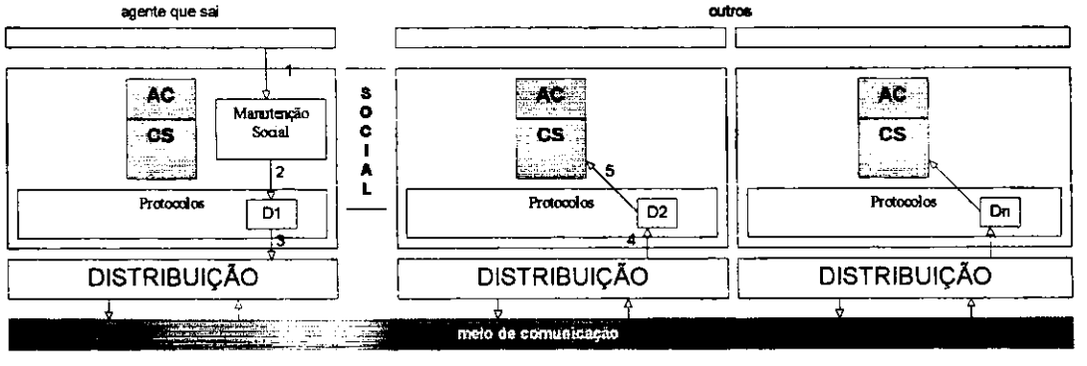


Figura 5.4: Protocolo de Saída

(1) Quando um agente sai da sociedade, antes de efetuar a sua saída, o Sistema Tutor pede ao sistema Social, através do módulo Manutenção, que inicie o protocolo de saída.
(2) O módulo MS cria um diálogo de saída, rotulada por D_1.
(3) O primeiro passo efetuado por D_1 é o envio de uma **INFORMAÇÃO** sinalizando para todos os agentes da sociedade a sua saída.
(4) Em todos os agentes da sociedade, cada diálogo (D_2, ..., D_n), relativa ao tratamento da situação de saída nos agentes que permanecerem na sociedade, é criada após a recepção da mensagem. Logo após a criação, em cada agente, o Sistema de Distribuição encaminha a **INFORMAÇÃO** recebida para o diálogo apropriado.
(5) O único tratamento realizado por cada diálogo (D_2, ..., D_n) é promover a remoção do AC relativo ao agente que sai do seu CS.

Atualização

A situação de atualização de um agente tutor é caracterizada neste trabalho através de uma saída seguida de uma reentrada. Devido a isso, a sua operacionalização utiliza os dois protocolos (entrada e saída) descritos anteriormente.

5.2. Linguagem de Interação

Uma linguagem de interação padroniza e precisa o vocabulário utilizado entre os agentes tutores da *SATA* durante a troca de mensagens. As interações são estruturadas num formato fixo para facilitar a composição e a interpretação das mesmas **[Pop93]**. Uma interação, tal como ilustrado a seguir na Figura 5.5, é composta por três partes, cada qual composta por campos com um propósito particular. Abaixo descrevemos cada uma destas partes:

(i) **DISTRIBUIÇÃO:** Composta por campos de endereçamento do agente e de identificação do diálogo, tanto para o emissor quanto para o receptor. Tais campos são úteis para os módulos de comunicação e controle do Sistema de Distribuição.

(ii) **SOCIAL:** É composto por campos úteis na interpretação da mensagem pelos diálogos.

(iii) **TUTOR**: Esta parte é preenchida com um conteúdo dependente do tipo da mensagem.

DISTRIBUIÇÃO				SOCIAL		TUTOR
REM	DEST	ID-DLG-REM	ID-DLG-DEST	PROT	PRIM	CONTEÚDO

Figura 5.5: Formato de uma interação.

O formato apresentado acima é inspirado em outros trabalhos, incluindo aqueles apresentados em **[Dem95]**. Abaixo descrevemos cada um dos campos da mensagem:

Remetente (REM): É preenchido com o identificador do agente tutor que remete a mensagem. Quem remete uma mensagem precisa se identificar, pois o destinatário geralmente precisa interagir de volta com ele.

Destinatário (DEST): Identifica o agente tutor destino. Este campo pode ser preenchido também com a palavra chave "TODOS" (difusão), ou com "TODOS EXCETO" seguida de uma lista de identificadores de agentes (endereçamento complementar).

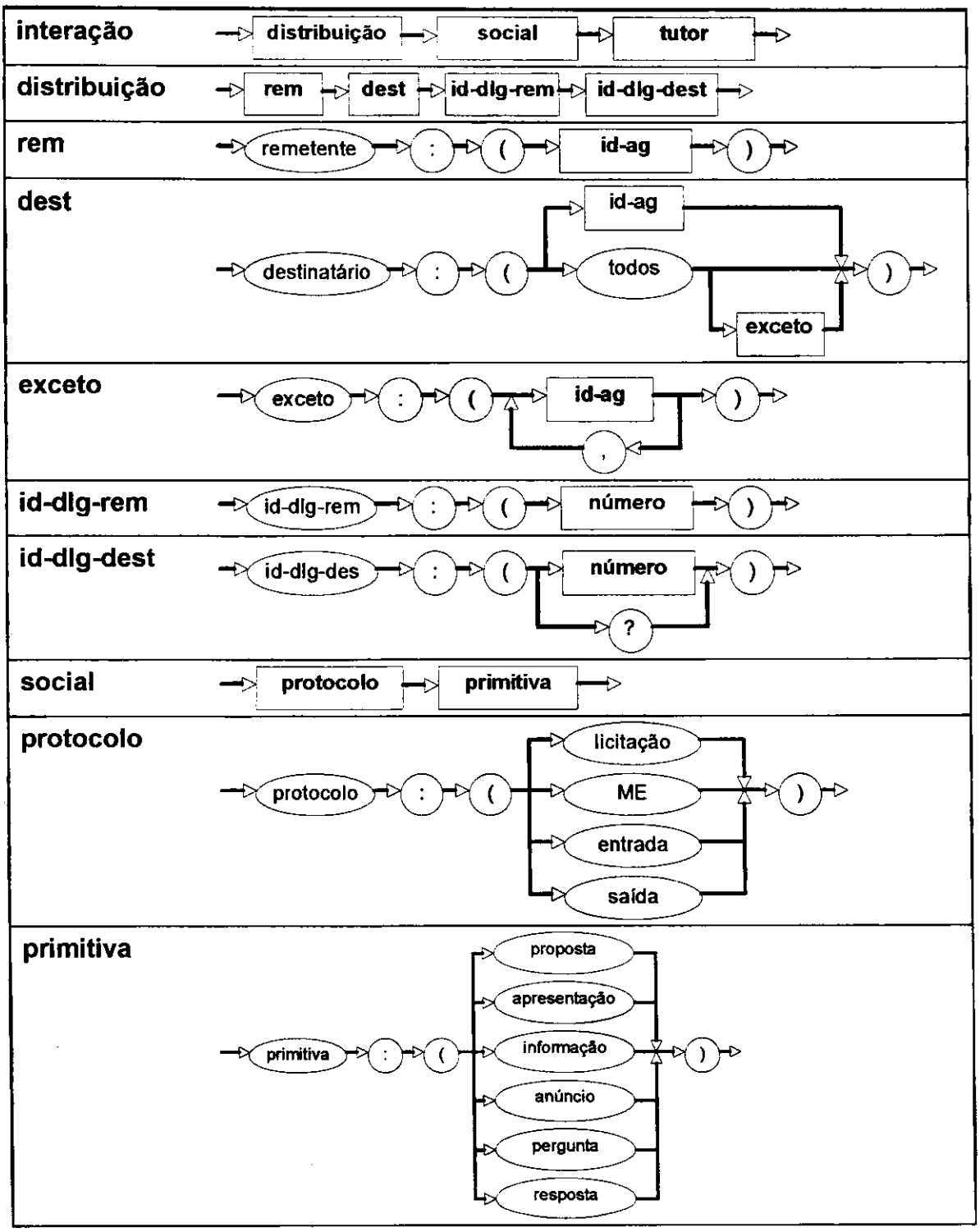
Identificação do diálogo remetente (ID-DLG-REM): Deve ser preenchido com o identificador do diálogo que remete a mensagem. Este atributo precisa ser identificado tomando como base a mesma justificativa utilizada para o atributo REM.

Identificação do diálogo destinatário (ID-DLG-DEST): Identifica o diálogo destino. Quando não se conhece o identificador do diálogo destino (primeira interação do protocolo, difusão ou endereçamento complementar), utiliza-se o caracter "?".

Tipo do Protocolo (PROT): Identifica o tipo do protocolo como: MESTRE-ESCRAVO, LICITAÇÃO, ENTRADA ou SAIDA.

Primitiva (PRIM): Identifica o tipo da interação entre os agentes contextualizando-a num protocolo, admitindo as seguintes possibilidades: APRESENTAÇÃO (entrada), INFORMAÇÃO (saída), ANÚNCIO (licitação), PROPOSTA (licitação), PERGUNTA (mestre-escravo), RESPOSTA (mestre-escravo).

Apresentamos na Tabela 5.1 os diagramas sintáticos utilizados para descrever a linguagem de interação. As elipses representam os símbolos terminais da linguagem, enquanto que os retângulos, os não-terminais. Logo após, ilustra-se uma possível interação exemplo.



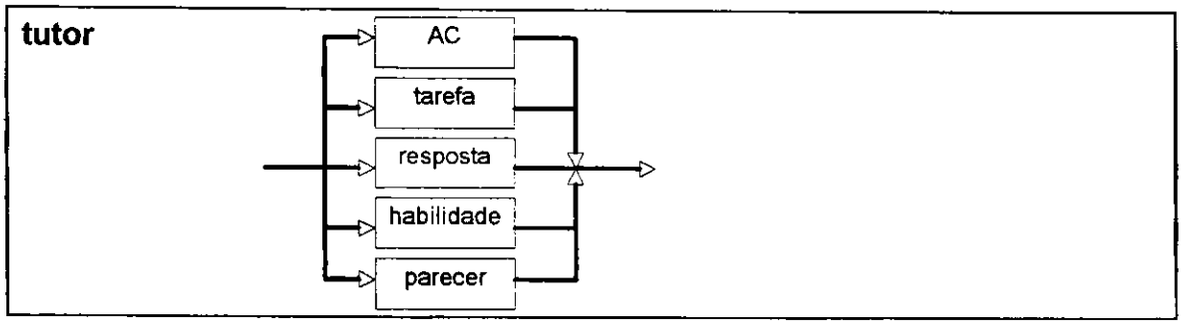


Tabela 5.1: Linguagem de Interação

```
remetente : ( "Agente1" )
destinatário : ( todos exceto : ( "Agente2" ) )
id-dlg-rem : ( 2 ) id-dlg-dest ( ? )
protocolo : ( licitação ) primitiva : ( anúncio ) ...
```

Observe que o campo destinado ao conteúdo da interação, como visto no diagrama anterior, pode ser preenchido com um AC, uma tarefa, uma resposta, uma habilidade ou um parecer. Estes elementos, tratados pelo Sistema Social ou pelo Sistema Tutor, possuem uma sintaxe própria baseada nas definições sintáticas e formais apresentadas no Capítulo 3. Apresentamos a seguir os diagramas sintáticos relativos à descrição de cada um desses elementos: *autoconhecimento* (Tabela 5.2), *tarefa* (Tabela 5.3), *resposta* e *parecer* (Tabela 5.4), *habilidade* (Tabela 5.5).

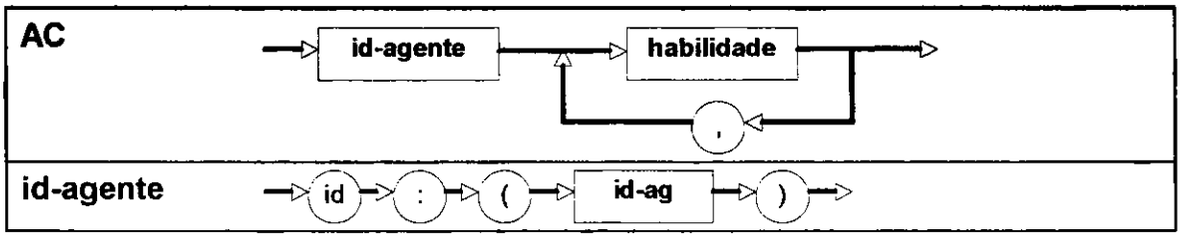
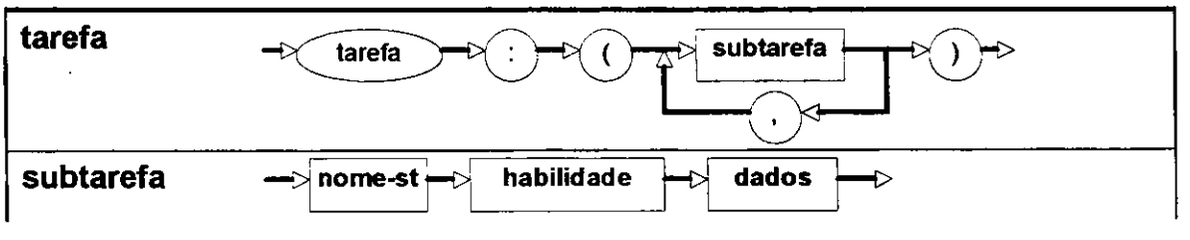


Tabela 5.2: Diagramas Sintáticos para o Autoconhecimento



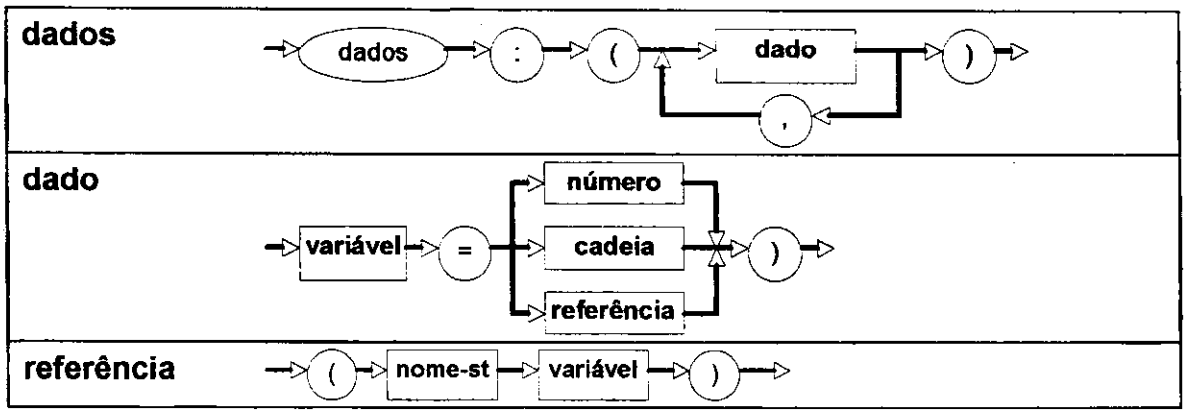


Tabela 5.3: Diagramas Sintáticos para a Tarefa

As dependências causais entre as subtarefas, quando há decomposição, são descritas através de uma referência que indica a subtarefa da qual se depende e o recurso (variável) esperado dela. Além de uma referência, um dado utilizado como entrada para uma tarefa pode ser numérico ou ainda uma cadeia de caracteres. Tanto a descrição de AC quanto a de tarefa, apresentadas acima, necessitam da descrição de habilidade que é introduzida mais adiante.

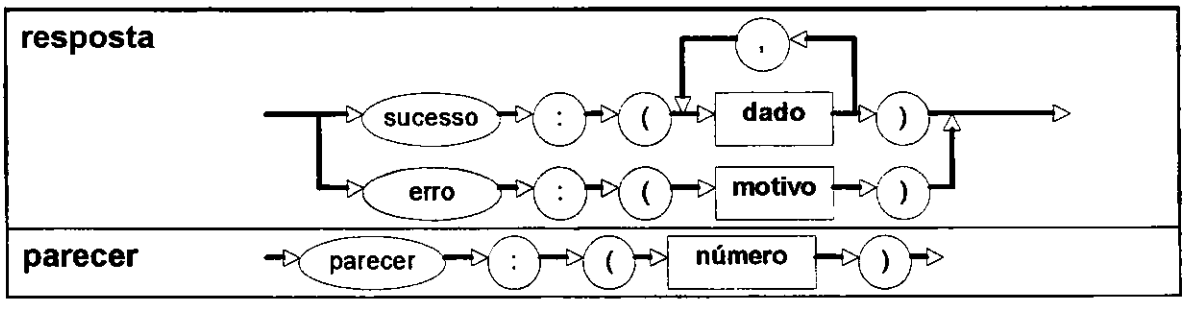


Tabela 5.4: Diagramas Sintáticos para Resposta e Parecer

Quando a resposta é um sucesso, ela vem acompanhada dos dados obtidos com a resolução da tarefa. No caso de insucesso, segue-se uma cadeia de caracteres informando o motivo pelo qual a cooperação falhou. O valor oferecido como parecer pelos subordinados durante o protocolo de licitação é numérico e compreendido no intervalo de 1 até 100 (incluindo estes dois valores).

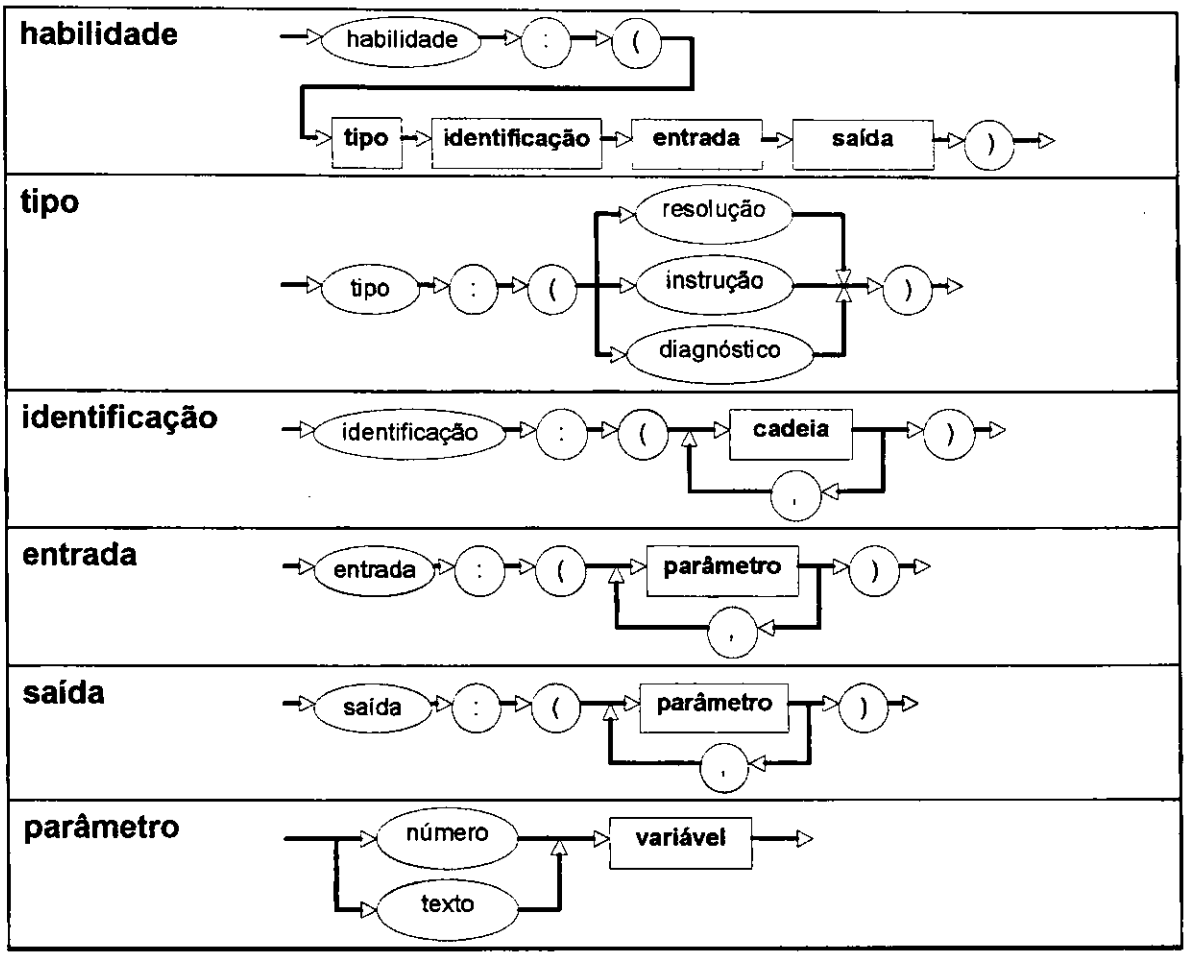


Tabela 5.5: Habilidade

Como é possível observar, há três tipos de habilidade relacionadas aos tipos de tarefas pedagógicas. Os parâmetros utilizados na descrição da interface da habilidade (entrada e saída) são, como visto nos dados de entrada e saída da tarefa, de dois tipos: numérico, texto (cadeia de caracteres).

Na Tabela 5.6 são listadas todas as palavras chaves utilizadas na linguagem de interação. A Tabela 5.7 apresenta os diagramas sintáticos dos elementos mais primitivos da linguagem, referenciados nos diagramas anteriores.

Remetente	destinatário	todos	exceto	id-dlg-rem	id-dlg-dest	protocolo
primitiva	id	tarefa	dados	sucesso	erro	parecer
habilidade	tipo	identificação	entrada	saída	número	texto

Tabela 5.6: Palavras-Chave

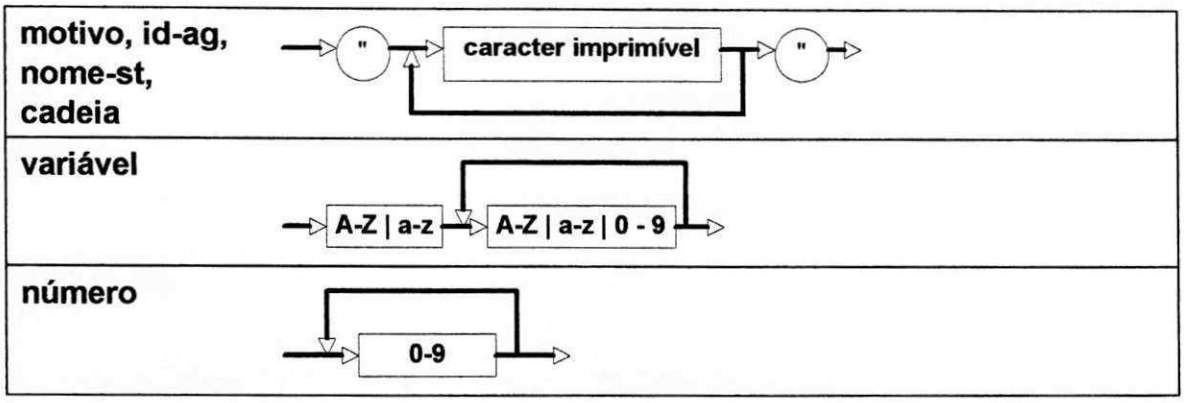


Tabela 5.7: Elementos Primitivos

Para melhor ilustrar o formato da linguagem, apresentamos logo abaixo um exemplo de uma tarefa descrita conforme a sintaxe definida. Os elementos não específicos da linguagem estão destacados em negrito.

```
tarefa : (
      "t1"
      habilidade : (
            tipo : ( resolução )
            identificação : ( "baskara" )
            entrada : ( NUM a, NUM b, NUM c )
            saída ( NUM raizes, NUM x1, NUM x2 ) )
      dados : ( a = 2, b = ( "t2" c ), c = ( "t3" c ) ) ,

      "t2"
      habilidade : (
            tipo : ( resolução )
            identificação : ( "soma" )
            entrada : ( NUM a, NUM b )
            saída: ( NUM c ) )
      dados: ( a = 2, b = 3 ),

      "t3"
      habilidade : (
            tipo : ( resolução )
            identificação : ( "multiplicacao" )
            entrada : ( NUM a, NUM b )
            saída : ( NUM c ) )
      dados: ( a = 1, b = 2 )
)
```

Exemplificando ainda, a seguir apresentamos a sintaxe de uma parte do AC de um agente tutor identificado por "Equação Grau 2", que, dentre todas as suas habilidades, sabe resolver equação de segundo grau através do método de Báskara.

```
id : ( "Equação Grau 2"
habilidade : (
      tipo : ( resolução )
      identificação : ( "baskara" )
      entrada : ( NUM a, NUM b, NUM c )
      saída ( NUM raizes, NUM x1, NUM x2 ) ),
...
```

Na Tabela 5.8 resume-se o preenchimento das interações nas situações existentes nos quatro protocolos apresentados.

Id. Destinatário	Protocolo	Primitiva	Aplicação
-	mestre-escravo	pergunta	TAREFA
-	mestre-escravo	resposta	ERRO
-	mestre-escravo	resposta	RESPOSTA
TODOS EXCETO	licitação	anúncio	HABILIDADE
-	licitação	proposta	PARECER

Id. Destinatário	Protocolo	Primitiva	Aplicação
TODOS	entrada	apresentação	DESCRIÇÃO DO AGENTE
TODOS	saída	informação	-
-	entrada	apresentação	DESCRIÇÃO DO AGENTE

Tabela 5.8: Resumo das interações utilizadas

6

Implementação

Este capítulo visa compartilhar a experiência obtida (requisitos levantados, decisões tomadas, restrições encontradas, etc) durante o processo de implementação do arcabouço de *software* que viabiliza o comportamento social dos agentes tutores do MATHEMA. Para isso, destaca-se alguns aspectos relevantes que foram identificados durante a fase de implementação.

O arcabouço, introduzido nos Capítulos 4 e 5 (arquitetura, funcionalidade, protocolos e linguagens), visa auxiliar o desenvolvimento de agentes no MATHEMA e é utilizado como uma biblioteca. Devido a esta natureza, o seu desenvolvimento foi realizado empregando-se a visão de programação do ponto grande (*programing in the large*) mencionada no Capítulo 2. No total, o código implementado em Java para o arcabouço é composto de cerca de 3.500 linhas.

A pretensão do trabalho, com relação à implementação, foi a de obter apenas um protótipo que visou basicamente a validação do que foi definido e especificado nos capítulos anteriores.

A implementação é apresentada em três momentos distintos: identificação do suporte operacional, escolha de ferramentas e implementação propriamente dita (codificação e depuração do código). Antes de se discutir a implementação, apresenta-se uma modelagem estática desenvolvida para o sistema. Finalmente, aborda-se aspectos funcionais e de interface dos testes que foram realizados para auxiliar o processo de desenvolvimento.

6.1. Identificação do Suporte Operacional

Algumas das necessidades operacionais do sistema guiaram a implementação do protótipo, e, no que segue, elas são sucintamente enumeradas.

1. **Comunicação:** disponibilidade de algum mecanismo que permita a comunicação entre os agentes da sociedade. Como, por exemplo: comunicação entre processos; invocação de método direto; *socket*; chamada remota de procedimento; invocação remota de método, entre outros.

2. **Paralelismo:** o sistema operacional utilizado para o desenvolvimento dos agentes deve oferecer suporte a multitarefa, motivado pelas notórias situações de paralelismo encontradas na arquitetura do agente.

3. **Controle de Concorrência**: considerando que na arquitetura exposta existem algumas situações de concorrência e acesso exclusivo a recursos compartilhados, é necessário então, que o sistema operacional ofereça algum mecanismo que implemente exclusão mútua.

4. **Heterogeneidade:** considerando que soluções distribuídas podem ser executadas em ambientes heterogêneos, é desejável que o sistema seja uma solução com suporte para execução em tais ambientes.

Observe que os três primeiros requisitos são imperativos, enquanto que o quarto, a heterogeneidade, é somente desejável pois a implementação em questão é de apenas um protótipo e não de um produto final com pretensões comerciais.

6.2. Escolha de Ferramentas

Antes de optar por uma ferramenta, e do início efetivo da implementação, algumas decisões precisaram ser tomadas. A primeira decisão foi a escolha de um sistema operacional. Entre as opções disponíveis, destacaram-se o Windows e Unix devido à disponibilidade e à facilidade de se encontrar ferramentas de desenvolvimento para tais

sistemas operacionais. Uma possibilidade mais atrativa foi procurar uma solução que contemplasse a independência de plataforma (heterogeneidade).

Sob o aspecto da possibilidade do uso de alguma ferramenta de apoio ao desenvolvimento de agentes, o presente trabalho restringiu-se a considerar a utilização de alguma biblioteca que diminuísse o esforço de desenvolvimento. Pode-se observar que, de um modo geral, estas bibliotecas tinham um propósito muito específico ou muito genérico, não se adequando ao estudo (comportamento, arquitetura, protocolos, linguagem) realizado para o domínio em questão.

Diversas ferramentas poderiam ter sido consideradas; entretanto, após a obtenção do modelo objeto da arquitetura, avaliou-se que o tempo de desenvolvimento seria menor do que o tempo de aprendizado de outras ferramentas de apoio que poderiam até não oferecer todos os recursos necessários. Não é o objetivo deste trabalho fazer um estudo exaustivo sobre ferramentas de apoio ao desenvolvimento de agentes, entretanto o leitor interessado pode encontrar informações detalhadas e referências em http://www.cs.umbc.edu/agents/.

A partir da decisão anterior, foi necessário optar por uma linguagem para codificar o arcabouço. Dentro deste contexto, as opções consideradas foram linguagens funcionais (*LISP*), linguagens baseadas em lógica (*Prolog*) e, por fim, orientadas a objeto (C++, *Java*, *Telescript* e *Smalltalk*).

Prolog e *Lisp* são duas linguagens usualmente ligadas com a programação em IA. Devido a isso, tais linguagens adequam-se melhor à realização do Sistema Tutor. Esta discussão está além do escopo deste trabalho, uma vez que nosso foco está nos Sistemas Social e de Distribuição, como já mencionado no Capítulo 3.

Linguagens OO são amplamente difundidas e utilizadas no contexto de desenvolvimento de agentes computacionais **[KJ98]**. Neste sentido e devido à modelagem desenvolvida com o paradigma OO, como é apresentado mais adiante, optou-se por uma linguagem OO (facilidade de derivação do código a partir do modelo).

Tanto *Java* quanto *Smalltalk* e *Telescript* são linguagens portáveis devido à utilização do conceito de maquina virtual. *Java* sempre foi a opção mais atraente e acabou sendo a escolhida, pois atende todas as necessidades colocadas na seção 6.1. A decisão pela linguagem Java acabou tornando irrelevante as questões relativas ao sistema operacional.

Finalmente, a questão relacionada ao mecanismo de comunicação foi considerada. Se observarmos um agente como um objeto, e realmente ele foi visto desta forma, algum padrão de comunicação conhecido que permitisse a interoperabilidade entre objetos distribuídos **[KJ98]** (OLE[1], CORBA[2]) poderia ter sido utilizado. Alguma linguagem padrão para proporcionar a interação entre agentes, como KQML, também foi cogitada. Foram observadas ainda outras alternativas, de mais baixo nível, dentre elas: RMI (*Remote Method Invocation*), RPC (*Remote Procedure Call*).

Como visto no Capítulo 3, na arquitetura do agente, mais especificamente no Sistema de Distribuição, existe um módulo de comunicação de baixo nível que deve ser somente responsável pelo envio e recebimento de mensagens. Evidentemente que todas as opções mencionadas anteriormente satisfazem os requisitos funcionais deste módulo. Por outro lado, essas opções oferecem outras funcionalidades dispensáveis e eventuais sobrecargas. Constatou-se enfim que o esforço de programação utilizando um mecanismo de comunicação mais primitivo seria irrelevante, considerando a implementação como um todo. Portanto, acabou-se optando pelo uso de *socket*[3]. Mais adiante falaremos mais sobre *socket* e daremos mais detalhes de como este recurso foi usado para efetivar a troca de mensagens entre os agentes.

Com *Java* pode-se desenvolver um projeto em ambientes diferentes e com ferramentas (edição e compilação) distintas. Porém para apoiar o desenvolvimento, foi utilizado totalmente o Visual J++ versão 1.1 da *Microsoft*.

---

[1] http://www.activex.com

[2] http://www.corba.org

[3] *Socket* é uma abstração de software, um "bocal" virtual aonde você pode se conectar para introduzir e retirar bits. Um *socket* existe nos dois pontos finais de uma conexão. O uso de *socket* por programadores é feito através do uso de estruturas e de chamadas de procedimentos de uma API — *Application Programming Interface*.

6.3. Modelo Objeto

Um modelo é uma abstração de algo com o propósito de entendê-lo antes de implementá-lo. Abstração é uma investigação seletiva de certos aspectos de um problema. Toda abstração é incompleta porém isto não invalida sua utilidade. Um bom modelo captura os aspectos cruciais de um problema e omite outros **[RBP91]**.

A modelagem orientada a objetos promove entre outras coisas: um melhor entendimento das necessidades, um projeto mais claro e um sistema mais fácil de manter.

Esta seção descreve sucintamente o modelo para os objetos utilizado na implementação. Apenas alguns aspectos relativos à descrição das classes são apresentados por serem mais relevantes e poderem ser utilizados como referência em outros trabalhos.

Aqui, optou-se pela notação OMT para descrever as classes e como elas se relacionam **[RBP91]**. OMT é uma notação gráfica independente de linguagem constituída por três tipos de modelo: um modelo objeto que descreve a estrutura estática de um sistema em termos de objetos e relacionamentos correspondentes às entidades do mundo real; um modelo dinâmico que descreve a estrutura de controle de um sistema em termos de eventos e estados; um modelo funcional que descreve a estrutura computacional de um sistema em termos de valores e funções. Apenas o primeiro modelo foi utilizado efetivamente e é aqui apresentado. Na Figura 6.1, apresentamos um resumo da notação OMT utilizada.

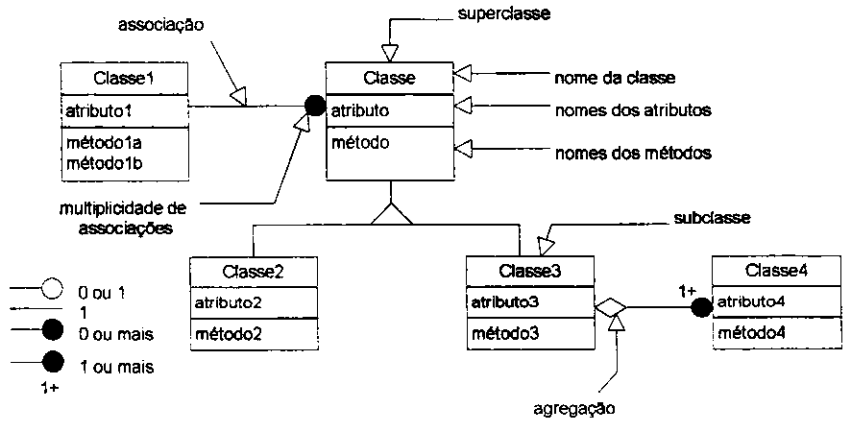


Figura 6.1: Resumo da notação OMT **[Coa97]**

Como pode se observar na Figura 6.2, para criar um agente MATHEMA, é necessário criar uma classe derivada da classe **social**. Esse esquema de derivação é bastante utilizado em outros trabalhos no domínio de agentes, pois é considerado mais natural do ponto de vista de modelagem **[KJ98]**.

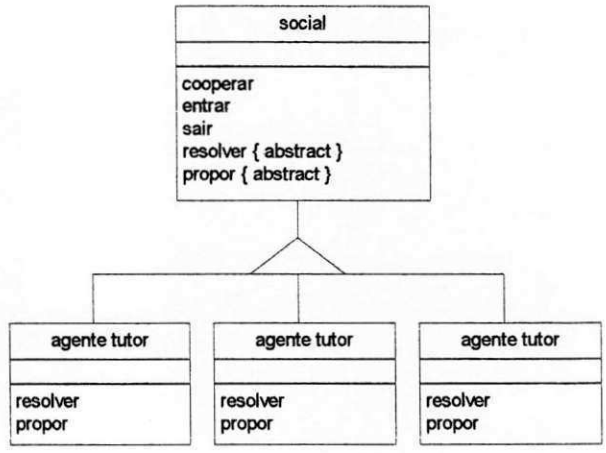


Figura 6.2: Modelo Objeto para a construção de agentes

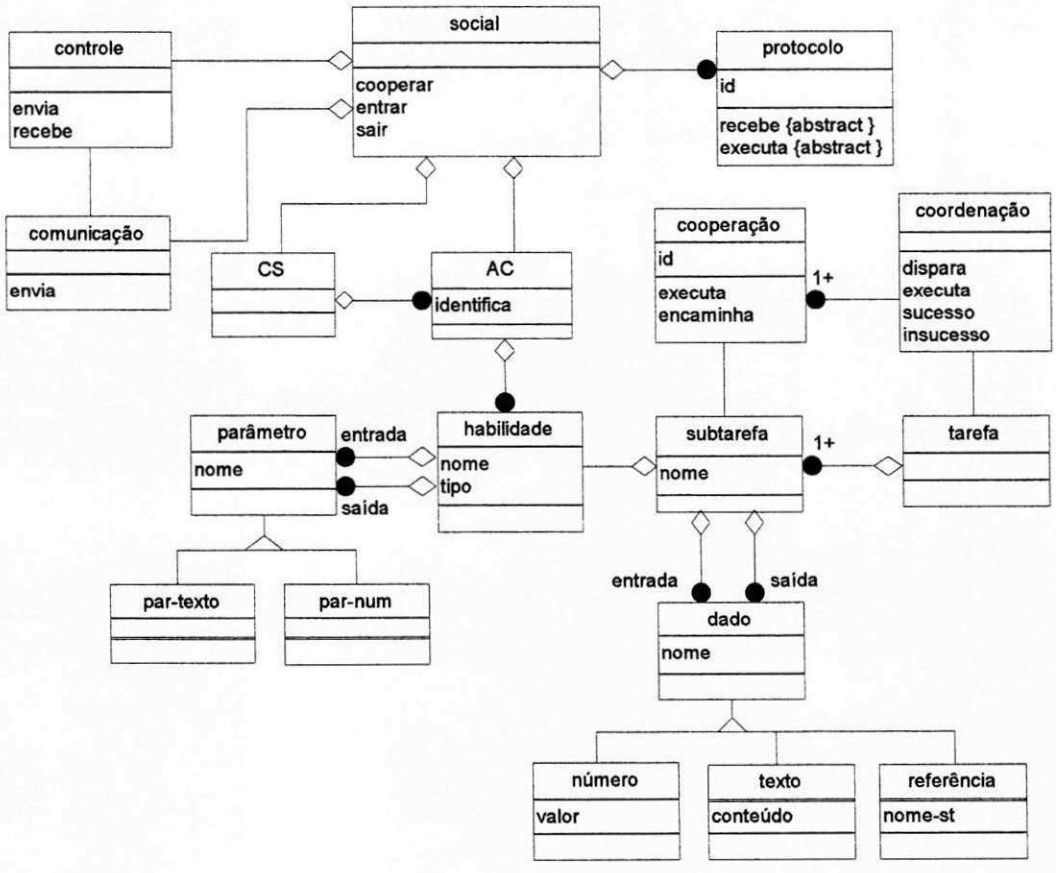


Figura 6.3: Modelo Objeto do Sistema Social (parcial)

A classe **social** possui dois métodos abstratos — resolver e propor — que precisam ser implementados em cada agente especializado. Estes métodos viabilizam a interação que existe do Sistema Social para o Sistema Tutor. Através do método cooperar, o agente tutor submete os pedidos de cooperação. A classe social define mais dois métodos, entrar e sair, que disparam respectivamente os protocolos de entrada e saída.

No modelo objeto, exibido na Figura 6.3, a classe **social** é um agregado de **protocolos**, um **CS**, um **AC**, um **controle**, e um **objeto** da classe **comunicação**. Um **CS** é constituído por vários **ACs**. Um **AC** possui um identificador e é definido por várias **habilidades**. Uma **habilidade** possui um nome, um tipo, uma entrada e uma saída (lista de parâmetros). Um **parâmetro** possui um nome e pode ser numérico (**par-num**), ou textual (**par-texto**).

Uma instância de **coordenação** pode executar várias instâncias de **cooperação** (uma ou mais). A **coordenação** está associada a uma **tarefa**, enquanto que a **cooperação** está associada a uma **subtarefa**. Uma **tarefa** é um agregado de **subtarefas** (uma ou mais). Uma **subtarefa** possui um nome, duas listas de **dados** (uma de entrada e outra de saída) e uma **habilidade** para executá-la. Os **dados** podem ser numéricos, textuais, ou uma referência a um dado de outra **subtarefa**.

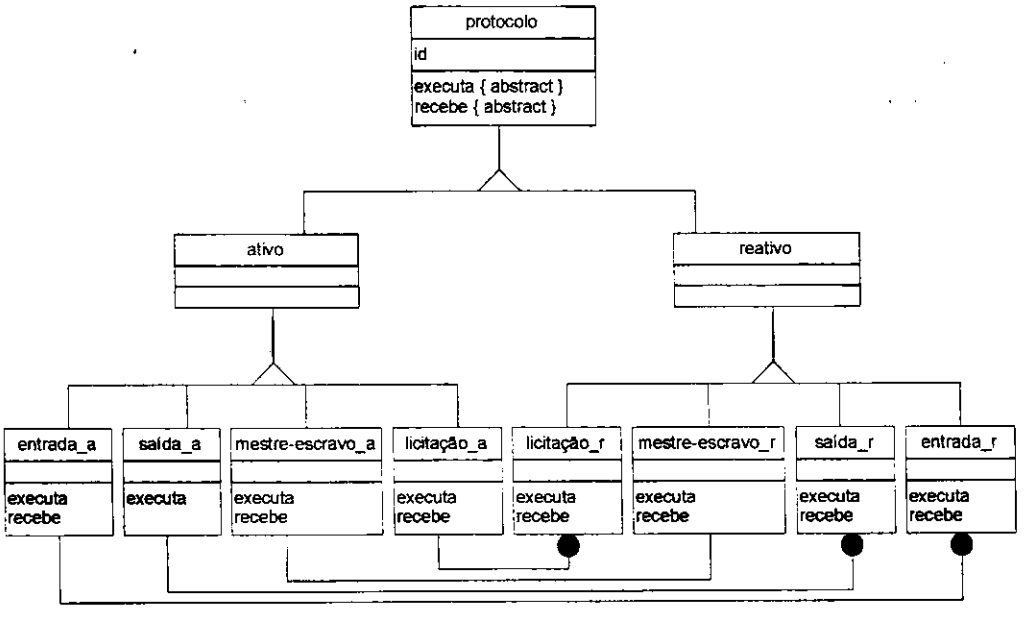


Figura 6.4: Modelo Objeto dos protocolos

Conforme a Figura 6.4, as implementações dos protocolos estão dispostas numa hierarquia. A superclasse **protocolo** possui apenas um atributo (id) e dois métodos abstratos (recebe e executa) que devem ser implementados em cada subclasse. Existem dois tipos de protocolos: **ativos** e **reativos**. Os **ativos** são criados pelo agente que inicia um diálogo, enquanto que os **reativos** são criados pelos agentes invocados por estes diálogos.

É importante ressaltar que foram omitidos detalhes dos modelos com o objetivo de tornar mais clara esta apresentação. Os modelos nas Figuras 6.2, 6.3 e 6.4 poderiam ser apresentados num único modelo, entretanto a legibilidade estaria prejudicada.

6.4. Implementação

Com finalidade didática, algumas questões de implementação, que podem ser úteis em outros trabalhos e que foram julgadas relevantes no contexto desta dissertação, são detalhadas nesta seção.

6.4.1. A Classe MAT_Agente

Como apresentado na Seção 6.4, Figura 6.2, para se obter um agente tutor seria necessário apenas criar uma classe derivada da classe **social**. Entretanto, ao se criar uma aplicação qualquer é muito comum ser necessário derivar a classe responsável pela aplicação de outras que oferecem certos recursos específicos, interface por exemplo. Porém, com *Java* só é possível realizar heranças simples. Seria então impossível, por exemplo, declarar uma classe de um agente tutor com tratamento de interface (herança múltipla). Neste sentido, devido à essa característica da linguagem *Java*, foi necessário criar um esquema um pouco diferente do apresentado na modelagem, mas não foi preciso refazer o modelo.

O esquema implementado exige que na definição da classe de um agente tutor se agregue um objeto da classe **MAT_Social**. Além disso, é necessário que a mesma classe agente tutor implemente os métodos abstratos da interface abstrata **MAT_Agente** (similar a derivar de uma classe vazia que possua apenas métodos abstratos). Abaixo,

apresentamos resumidamente as classes **MAT_Social** e **MAT_Agente** e uma possível classe **Agente1** usando tal esquema.

```
abstract interface MAT_Agente
{
      abstract public MAT_Dados Resolver( MAT_Tarefa tarefa );
      abstract public double Propor( MAT_Habilidade hab );
}

class MAT_Social
{
      ....
      public void Entrar( MAT_AC ac ) { ... }
      public void Sair(){ ... }
      public MAT_Erro Cooperar( MAT_Tarefa tarefa ) { ... }
      ...
}

class Agente1 extends Frame implements MAT_Agente
{
      MAT_Social  social;

      Agente( ... )
      {
            ....
            social = new MAT_Social( this );
            ...
      }

      public MAT_Dados Resolver( MAT_Tarefa tarefa ) { ... }
      public double Propor( MAT_Habilidade hab ) { ... }
}
```

6.4.2. Comunicação

Com base nas necessidades do modelo de comunicação discutido no Capítulo 3, dois aspectos foram considerados na implementação do mecanismo de comunicação: o caráter assíncrono da comunicação e os modos de endereçamento existentes. Além disso, outro aspecto, relacionado inerentemente com a implementação, foi levado em conta, qual seja a distribuição dos agentes. Neste contexto, considerou-se a possibilidade da execução dos agentes em quatro configurações: (i) numa mesma máquina; (ii) em máquinas diferentes de uma rede local; (iii) em redes diferentes; (iv) ou mesmo em qualquer combinação arbitrária das três configurações anteriores. No que segue, elucida-se como cada um desses aspectos foi tratado e as estratégias de solução adotadas.

A Transmissão de uma Mensagem

O modo de comunicação entre os agentes é do tipo par-a-par (*peer-to-peer*); um agente pode assumir, ora o papel de cliente, ora o de servidor, ou ambos. O módulo de *Comunicação* é autônomo; em todo agente, há um *thread* sendo executado para tratar da recepção das mensagens recebidas. A seguir, como ilustrado na Figura 6.5, descrevemos o processo de transmissão de uma mensagem observando os dois lados da comunicação.

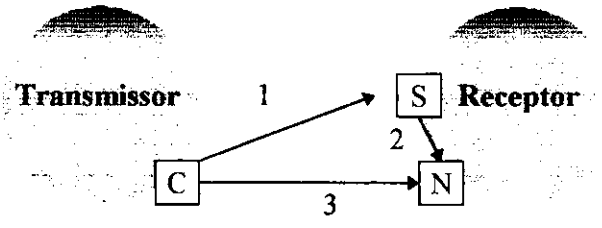


Figura 6.5: Transmissão de uma mensagem

1. Para enviar uma mensagem, a camada de comunicação do agente transmissor, cria um *socket* **C** e se conecta (*connect*) ao *socket* **S** de atendimento do agente receptor.

2. O *socket* **S** de atendimento do agente receptor, que foi criado na inicialização do módulo de comunicação está em estado de espera (*listen*). Quando aceitar uma conexão (*accept*), cria-se um novo *socket* **N**, já conectado a **C**, e continua a atender outras conexões (*listen*).

3. No agente emissor, a camada de comunicação escreve a mensagem completamente no *socket* **C** para depois fechá-lo. No agente receptor, um outro *thread* é criado para se encarregar da leitura da mensagem no *socket* **N**. A criação deste *thread* é importante pois viabiliza o tratamento paralelo de mensagens transmitidas. Logo após a finalização da leitura, o *socket* **N** é fechado.

Observe que no passo 3., a criação de um *thread* é necessária para não bloquear o processo de aceitação de conexões através do *socket* **S** e para criar um fluxo de execução que trate a recepção das mensagens nos módulos acima. Sem esse mecanismo, as mensagens recebidas seriam tratadas sequencialmente.

O Agente de Comunicação

Como foi determinado no Capítulo 3, existem três modos de endereçamento: direto, total e complementar. Para viabilizar o terceiro modo de endereçamento (grupo) foi necessária a implementação de um mecanismo especial.

Foram consideradas várias soluções para este problema, todas elas possuíam vantagens e desvantagens. Por fim, optou-se pelo esquema descrito a seguir. Todas as mensagens não são enviadas diretamente de um agente tutor para outro, elas antes são encaminhadas para um agente de comunicação (conhecido por todos) que sabe aonde cada agente está instalado (máquina e porta), como será detalhado mais adiante. Com isso, as mensagens enviadas para um grupo (complementar) são devidamente encaminhadas por este agente de comunicação (por conhecer todos) para os agentes destinos.

O agente de comunicação se instala numa máquina e numa porta conhecida para que todos os agentes da sociedade possam entrar em contato com ele. Observando a Figura 6.6, as seguintes etapas ocorrem:

(i) antes de entrar na sociedade, um agente deve se cadastrar junto ao agente de comunicação enviando uma mensagem com as seguintes informações: identificador do agente, máquina e porta aonde ele está instalado;

(ii) todas as mensagens são enviadas para o agente de comunicação, e através da análise do campo destinatário, ele encaminha as mensagens para os agentes apropriados;

(iii) após efetivar o protocolo de saída, um agente deve se descadastrar junto ao agente de comunicação enviando o seu identificador.

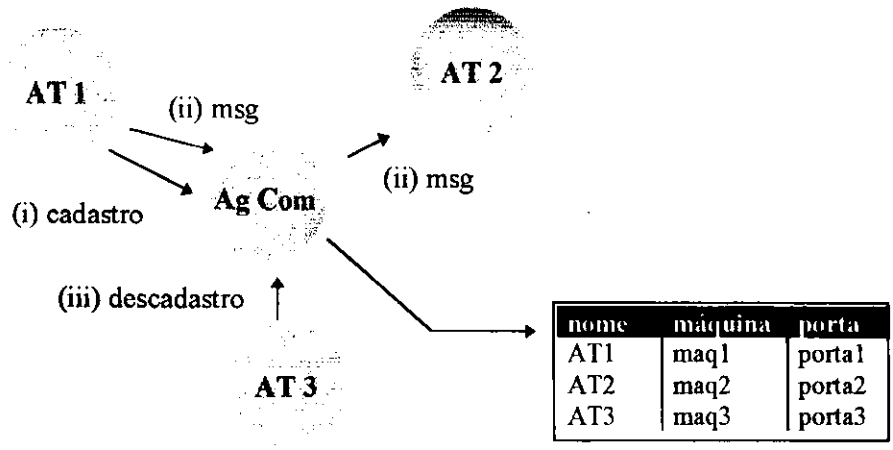


Figura 6.6: Agente de Comunicação

O agente de comunicação consegue distinguir as interações normais da sociedade das mensagem de cadastro e descadastro devido à sintaxe utilizada. Para o cadastro e o descadastro é utilizado um formato de mensagem diferente.

Dentre as diversas soluções, uma outra seria definir a priori a localização (máquina, porta) de forma estática (através de um arquivo ou embutido no código) de todos os agentes da sociedade. Esta solução foi utilizada apenas inicialmente, porém ficou evidente que o seu uso restringia o comportamento dinâmico da sociedade.

Uma outra solução cogitada foi a restrição da operacionalização da sociedade a um ambiente de rede local e, desta forma, utilizando-se de *broadcast* para encaminhar mensagens para um grupo específico. Com esta solução é necessário que cada agente da sociedade verifique se uma determinada mensagem de *broadcast* o incluí como destinatário. Entretanto esta opção foi descartada principalmente devido à possibilidade de ter a sociedade instalada numa única máquina, uma situação desejável durante o processo de implementação.

Como mencionamos anteriormente, todas as soluções consideradas oferecem prós e contras. Da mesma forma, a adoção do agente de comunicação também implicou em vantagens e desvantagens, as quais discutimos sucintamente em seguida.

Além de ser uma solução para o caso do endereçamento complementar, foi possível também com este esquema executar vários agentes pertencentes à mesma sociedade numa mesma máquina. Adicionalmente esta solução permite ainda que as máquinas estejam em redes diferentes. É possível também a criação de sociedades paralelas, basta criar dois ou mais agentes de comunicação, cada um responsável pelo encaminhamento da comunicação de uma sociedade.

Um dos pontos fracos da solução é o seu aspecto centralizador. A centralização torna o esquema vulnerável: toda a troca de mensagem na sociedade depende do bom funcionamento do agente de comunicação. Caso ele esteja instável, todo o sistema de comunicação da sociedade pode sofrer conseqüências. Com a saída inesperada do agente de comunicação todo o sistema deixa de funcionar. Outro problema é o desempenho: há uma sobrecarga causada pela retransmissão da mensagem, do agente remetente para o agente de comunicação, e deste para o agente destinatário.

Com o agente de comunicação não foi necessária a inclusão de novos atributos (número de porta e nome de máquina) na linguagem de interação. A linguagem permaneceu inalterada com a referência a agentes através do uso de identificadores e o arcabouço ficou independente da solução de implementação adotada para o modelo de comunicação, característica assumida como desejável durante o projeto.

6.4.3. Paralelismo e Controle de Concorrência

Uma possibilidade para implementar paralelismo no agente é criar processos distintos responsáveis por cada execução paralela. Outra possibilidade é utilizar *threads*, um fluxo sequencial simples de controle dentro de um processo. Um processo pode executar vários *threads* concorrentemente.

Optou-se pelo uso de *threads* pois o desempenho é melhor e a programação é bem mais simples. Fica mais fácil programar comunicação entre as execuções paralelas (chamada normal de método no lugar de comunicação entre processos) e acesso a dados compartilhados (um *thread* tem acesso a toda a área de dados do processo que o criou, com processos distintos é necessário usar uma área de memória compartilhada). Além desses dois aspectos, a sincronização e o controle de concorrência também são facilitados com o uso de *threads*. *Java* oferece algumas classes para realizar e tratar o paralelismo, dentre elas citamos **Runnable** e **Thread** que foram utilizadas na implementação.

Há alguns pontos de paralelismo no código, dentre eles podemos citar: disparo de subtarefas, envio e recepção das mensagens tanto num agente tutor quanto no agente de comunicação.

O acesso concorrente é uma situação que decorre do paralelismo. Para definir regiões críticas Java dispõe da palavra-chave **synchronized**. É possível definir uma região do código, um método, ou um objeto como "sincronizado". Há alguns recursos compartilhados no agente que precisam ser acessados de modo exclusivo, dentre eles podemos citar: o *CS*, o *AC*, uma lista de subtarefas durante uma coordenação, a lista dos protocolos.

6.4.4. O Arquivo de Configuração

Há um arquivo de configuração (*config.txt*) onde se define a porta e a máquina aonde o agente de comunicação se instala, e os tempos de espera utilizados nas situações de difusão de mensagens. Estes atributos são utilizados tanto pelo sistema quanto pelos agentes. Abaixo vemos um exemplo para este arquivo.

```
agcom.porta=3000          // porta do agente de comunicação
agcom.maquina=localhost   // máquina do agente de comunicação
dlgent.timeout=5000       // time-out para o protocolo de entrada
dlgme.timeout=5000        // time-out para o protocolo mestre-escravo
dlgneg.timeout=5000       // time-out para o protocolo de licitação
```

6.5. Testes

Considerando a não existência de nenhum Sistema Tutor (no contexto do MATHEMA), nem como protótipo, na época do desenvolvimento do arcabouço, foi necessário implementar, como teste, alguns simuladores de agentes tutores. Cada teste é um código que objetiva simular tanto o comportamento mínimo de um Sistema Tutor (habilidades e resolução de tarefas) quanto as interações deste com o Sistema Social. Com isso, possibilita-se a verificação das funcionalidades do arcabouço.

Os agentes de simulação foram três, todos apenas com habilidades pedagógicas de resolução: um destinado a trabalhar com equação de segundo grau, e os outros dois com operações matemáticas. A partir deste cenário foi possível simular todos os aspectos operacionais levantados: *time-out*, incompletude e incorretude do conhecimento social, chegada atrasada de mensagem, paralelismo e concorrência, execução de protocolo, etc. Para o leitor interessado em cenários mais complexos consulte **[Cos97]**.

Com relação à interface, no MATHEMA só há interação homem-máquina com o aprendiz (agente de interface) ou com o *SEH* (agente de manutenção), os agentes tutores interagem entre si, e, por isso, não precisam de interface homem-máquina. Os agentes de simulação são equivalentes a agentes tutores, porém foi necessário desenvolver para eles uma interface mínima de acompanhamento da execução do arcabouço e de inicialização de cooperações.

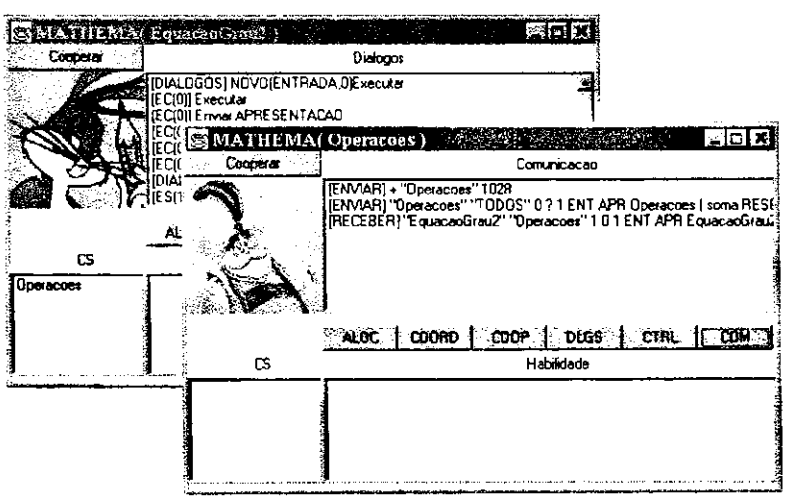


Figura 6.7: Interface de um Agente Tutor (Windows)

A linguagem Java foi suficiente para o desenvolvimento da interface do agente, que foi desenvolvido como uma aplicação separada, não como um *applet*. Na Figura 6.7, pode-se observar a interface para dois agentes.

A imagem colocada na interface do agente tutor é utilizada apenas para personalizá-lo. Uma imagem facilita o reconhecimento, e é mais rápido assim distinguir visualmente um agente do outro. Na área central, deslocado para a direita, aparecem seis botões alinhados, com os seguintes rótulos: ALOC (alocação), COORD (coordenação), COOP (cooperação), DLGS (protocolos), CTRL (controle), COM (comunicação). Quando se ativa um desses botões, a lista acima é modificada para apresentar a depuração relativa ao módulo especificado pelo botão. Na parte de baixo da janela, há duas áreas. Na área da esquerda é apresentada uma lista com os agentes da sociedade que estão com o seu AC cadastrado no CS do agente relativo à interface em questão. Quando um desses agentes é marcado, na área da direita aparece a lista de habilidades do seu AC. O botão "Cooperar", situado na parte superior à esquerda, é utilizado para disparar uma cooperação pelo usuário. Quando este botão é ativado, a caixa de diálogo, como apresentado na Figura 6.8, é exibida.

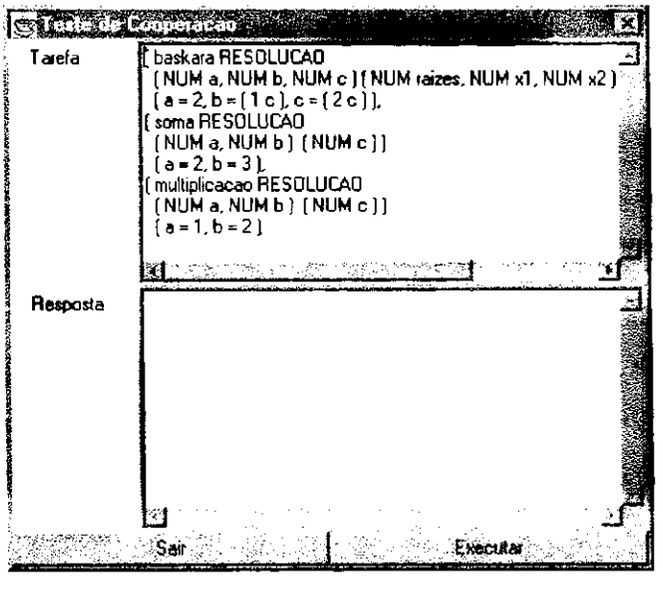


Figura 6.8: Caixa de Diálogo utilizada para realizar o teste de Cooperação (Windows)

Há uma área de edição relativa à descrição da tarefa. Com a tarefa descrita, o usuário pode acionar o botão "Executar" para disparar uma cooperação. Após a submissão da

tarefa à sociedade, o resultado (lembre-se do Capítulo 4, o resultado pode ser a solução da tarefa ou uma sinalização de erro) é exibido na área abaixo. Para abandonar a janela, basta acionar o botão "Sair". Mesmo com esta janela aberta, é possível acompanhar na janela do agente o andamento da execução da cooperação nos módulos.

Conforme apresentado na Figura 6.9, a interface do agente de comunicação é bastante trivial, nela simplesmente são exibidos os agentes da sociedade. Cada linha da tabela é composta pelas informações: nome do agente, máquina e porta.

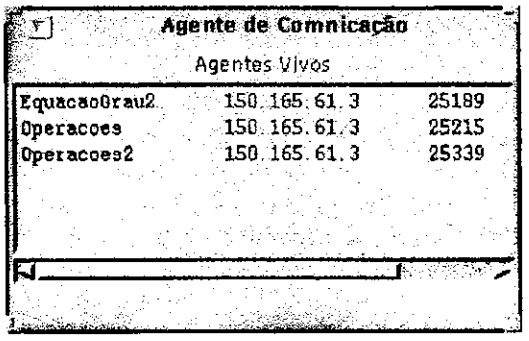


Figura 6.9: Interface do Agente de Comunicação (Openwindows)

É desejo no futuro que os agentes de interface e de manutenção, como visto no Capítulo 3, sejam implementados como *applets* para permitir o ensino e a gerência de forma remota através de um cliente (*browser*) universal.

7

Conclusão

Este trabalho está contextualizado na área de Inteligência Artificial Distribuída, mais especificamente na área de Sistemas Multiagentes aplicada no contexto do projeto MATHEMA **[Cos97]**. O documento descreveu basicamente a realização de quatro objetivos:

(i) contribuir para o estabelecimento de um modelo de organização, cooperação e comunicação para a sociedade e um modelo para os agentes tutores;

(ii) contribuir para um projeto da arquitetura para os agentes tutores que descreva a configuração e a interação de alto nível dos seus componentes;

(iii) definir a linguagem de interação e os protocolos que devem ser utilizados pelos agentes tutores na viabilização do comportamento cooperativo;

(iv) implementar um arcabouço de *software* que promova o desenvolvimento de agentes tutores no contexto do MATHEMA.

Com base no comportamento cooperativo desejado para a sociedade multiagente dos tutores do MATHEMA, e em consonância com **[Cos97]**, definiu-se o modelo para um agente tutor e para esta sociedade. Estes modelos foram apresentados através da especificação dos aspectos de organização, cooperação, comunicação e controle.

A partir de um estudo das possibilidades existentes na literatura, e após descrever tais modelos, foi projetada uma arquitetura para os agentes tutores. No Capítulo 3, tal arquitetura foi introduzida. No Capítulo 4, foram esclarecidos detalhadamente os aspectos estáticos e dinâmicos que emergem da interação entre os seus componentes.

Em seguida, foram abordados semantica e sintaticamente os protocolos e a linguagem de interação utilizados na interação entre os agentes tutores.

Por fim, apresentou-se o desenvolvimento de um sistema de *software* para ser utilizado com o objetivo de viabilizar o comportamento definido para os agentes tutores. O produto obtido com a implementação foi um protótipo, um arcabouço desenvolvido, modelado e implementado segundo o paradigma de orientação a objetos. O funcionamento deste arcabouço de *software* foi testado através de simulações, uma vez que Sistemas Tutores não estavam disponíveis e a sua implementação não está no escopo deste trabalho. Alguns detalhes de interface e alguns comentários sobre estas simulações foram apresentados no final do Capítulo 6.

Algumas das soluções realizadas durante a implementação do protótipo foram pragmáticas. Por exemplo, a linguagem de interação foi implementada a partir de uma simplificação da sintaxe descrita no Capítulo 5. Esta simplificação consistiu na eliminação de alguns símbolos da linguagem utilizados apenas como marcadores. Isto foi adotado para facilitar o desenvolvimento tanto do *parser* quanto do montador das mensagens. Por exemplo, uma mensagem com a sintaxe definida no Capítulo 5 seria `"from: (agentex) to: (agentey )..."`, entretanto da forma como foi implementada foi simplificada para `"agentex agentey ..."`. Esta simplificação, apesar de não interferir na funcionalidade, introduz uma dificuldade de leitura para um observador humano que acompanha as interações através da interface do agente, além disso, a composição do AC é dificultada.

Além dos objetivos específicos já discutidos, pretendeu-se ainda, com os resultados obtidos com o desenvolvimento desta dissertação, disponibilizar, além do arcabouço apresentado, uma experiência no domínio de Sistemas Multiagentes.

Acreditamos que os objetivos definidos foram alcançados e que o desenvolvimento do trabalho gerou experiências satisfatórias de estudo, modelagem e implementação.

A contribuição maior foi para o ambiente MATHEMA, este trabalho representa um esforço importante na direção da implementação deste ambiente. Conceitualmente e do ponto de vista de sistema de *software* este ambiente é relativamente complexo.

7.1. Perspectivas Futuras

No intuito de aprimorar as soluções encontradas neste trabalho, sugerimos alguns possíveis encaminhamentos futuros que devem contribuir para a evolução do ambiente MATHEMA, são eles:

- Sofisticação do mecanismo de alocação, utilizando técnicas de casamento de padrão mais elaboradas: *thesaurus*, regras de equivalência, entre outros. O algoritmo de casamento de padrão mencionado no Capítulo 4 e utilizado no processo de alocação foi implementado de forma trivial, através da comparação de cadeias de caracter.

- Sofisticação do esquema de resolução de conflitos com a inserção de novos conceitos, como por exemplo crença.

- Inserção de variáveis de interesse nos agentes, passando a considerar não somente os agentes como benevolentes.

- Para reutilizar o arcabouço em aplicações em contextos diferentes, ao invés de implementar os protocolos diretamente em código, poderíamos definir um esquema que possibilitasse a utilização dos protocolos através de bibliotecas. Tais protocolos seriam definidos por uma linguagem específica e interpretados em tempo de execução **[BHK93]**.

- Implementação da linguagem de interação como especificado no Capítulo 5, sem as simplificações mencionadas anteriormente.

- Com a perspectiva de manipular interações mais complexas no futuro, a linguagem talvez necessite de algum enriquecimento. Neste caso, deve-se

considerar a possibilidade de empregar linguagens já conhecidas, ou mesmo padrões, como por exemplo KQML.

- Estudo de outras alternativas de organização para a sociedade de agentes. Alguns domínios de ensino podem se ajustar bem com uma organização hierárquica. Para estes casos, os protocolos utilizados deveriam ser adequados convenientemente.

- Hoje, existe a possibilidade de laços cooperativos durante o processo de cooperação. A detecção deste comportamento não desejado de forma automática é objeto de investigação futura. Atualmente, o projetista deve evitar a ocorrência destes laços na concepção dos agentes. Uma possível solução seria adotar uma organização hierárquica.

Referências Bibliográficas

[AS97] Álvares, L.O., Sichman, J.S. "Introdução aos Sistemas Multiagentes", *XVI Jornada de Atualização em Informática - JAI'97*, pp. 1-39, Brasília (DF), agosto 1997.

[BB97] Bigus, J.P.; Bigus, J. *Constructing Intelligent Agents with Java*, John Wiley e Sons, New York (EUA), 1997.

[BDB93] Berthet, S.; Demazeau, Y., Boissier, O. "Knowing Each Other Better ", *11th International Workshop on Distributed Artificial Intelligence, Glen Arbor*, fevereiro 1993.

[BG88] Bond, A., Gasser, L. *Readings in Distributed Artificial Intelligence*, San Mateo, Morgan Kaufmann, 1988.

[BHK93] Burmeister, B., Haddadi A., Sundermeyer K. "Generic Configurable Cooperation Protocols for Multi-Agent Systems", *5th MAAMAV workshop, Ghedira and Sprumont eds., University of Neuchâtel*, 1993.

[CL96] Chan, P.; Lee, R. *The Java™ Class Libraries: An Annotated Reference*, Addison-Wesley Publishing Company, Massachusetts (EUA), 1996.

[CL97] Costa, E.B.; Lopes, M.A. "Ambientes de Aprendizagem Interativos: Uma Abordagem Teórica", *Anais do I Encontro Nacional de Inteligência Artificial - ENIA'97*, Brasília (DF), agosto 1997.

[CLF95] Costa, E.B.; Lopes, M.A.; Ferneda, E. "MATHEMA: A Learning Environment Based on a Multi-Agent Architecture", *Proceedings of the 12th Brazilian Symposium on Artificial Intelligence, Campinas, Brazil*, pp. 141-150, Wainer J.; Carvalho A. (eds), Volume 991 of Lecture Notes in Artificial Intelligence, Springer-Verlag, outubro 1995.

[Coa97] Coad, P. *Object Models: Strategies, Patterns & Applications*, second edition, Yourdon Press, Upper Saddle River (EUA), 1997.

[Cos97] Costa, E.B. *Um modelo de ambiente interativo de aprendizagem baseado numa arquitetura multi-agente*, Tese de Doutorado, Curso de Doutorado em Engenharia Elétrica, UFPB, Campina Grande, 1997.

[CP96] Costa, E.B.; Perkusich, A. "Modelling the Cooperative Interactions in a Teaching/Learning Situation", *Proceedings of Third International Conference on Intelligent Tutoring Systems - ITS'96*, Montreal (Canadá), junho, 1996.

[CP97a] Costa, E.B.; Perkusich, A. "A Multi-Agent Interactive Learning Environment Model", *Proceedings of the 8th World Conference on Artificial Intelligence in Education / Workshop on Pedagogical Agents*, Kobe (Japão), agosto 1997.

[CP97b] Costa, E.B.; Perkusich, A. "Designing a Multi-Agent Interactive Learning Environment", *Proceedings of the International Conference on Computers in Education - ICCE'97*, (Malásia), dezembro 1997.

[CPF96] Costa, E.B.; Perkusich, A.; Figueiredo, J.C.A. "A Multi-Agent Based Environment to Aid in the Design of Petri Nets Based Software Systems", *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering - SEKE'96*, Illinois-EUA, junho 1996.

[CPJ96] Costa, E.B.; Perkusich, A., Jatobá A.A. "Petri net based modelling of the cooperative interaction in multi-agent based learning environment", *Procedings of IEEE Computational Engineering in Systems Application - CESA'96*, Lille (França), junho 1996.

[CPJ 97] Costa, E.B.; Perkusich, A.; Jatobá, A.A. "Linguagem e Protocolos para Interação entre Agentes Tutores", *Anais do I Encontro Nacional de Inteligência Artificial - ENIA '97*, SBC, Brasília (DF), agosto 1997.

[DBK94] Demazeau, Y.; Boisser, O.; Koning, J-L. "Interaction Protocols in Distributed Artificial Intelligence and their Use to Control Robot Vision Systems", *6th International Conference on Artificial Intelligence and*

Information-Control Systems, Smolenice, Plander ed., World Scientific Publishing, 1994.

[Dem95] Demazeau, Y. "From Interactions to Collective Behaviour in Agent-Based Systems", *Proceedings of the European Conference on Cognitive Science*, Saint Malo (França), Abril 1995.

[DM90] Demazeau, Y.; Muller, J.P. "Decentralized artificial intelligence" In: *Decentralized AI*, pp. 3-13, Demazeau, Y.; Muller, J.P. (ed.), Amsterdam (Holanda), Elsevier Science Publishers, 1990.

[DR94] Durfee, E. H.; Rosenschein, J.S. "Distributed problem solving and multi-agent systems: comparisons and examples" In: *Proc. 13th. International Workshop on DAI*, Seattle, 1994.

[FG91] Ferber, J.; Gasser, L. "Intelligence Artificielle Distribuée", *Proceedings of the International Workshop on Expert Systems and Their Applications*, Avignon (França), 1991.

[FG96] Franklin, S.; Graesser, A. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[Fin94] Finin, T., Fritzon, R. McKay, D. e McEntire, R. "KQML as an agent communication language", *Proc. 3rd. International Conference on Information and Knowledge Management*, 1994.

[Gas92] Gasser, L. "An Overview of DAI", *Distributed Artificial Intelligence: Theory and Praxis*, pp. 10-29, Kluwer Academic Publishers, Dordrecht (Holanda), 1992.

[HS97] Huhns, M. N.; Singh M. P. *Readings in Agents*, Morgan Kaufmann Publishers, San Francisco (EUA), 1997

[Jen94] Jennings, N.R. *Cooperation in Industrial Multi-Agent Systems*, World Scientific Publising, Singapore, 1994.

[JW92] Jennings, N.R. van; Wittig T. "ARCHON: Theory and Pratice", In: *Distributed Artificial Intelligence: Theory and Praxis*, Avouris, N.M.;

Gasser L. (Eds), pp. 179-195, Kluwer Academic Publishers, Dordrecht (Holanda), 1992.

[KJ98] Knapik, M.; Johnson, J. *Developping Intelligent Agents for Distributed Systems*, McGraw-Hill, New York (EUA), 1998.

[Min85] Minsky, M. *The Society of Mind*, Simon and Schuster, New York (EUA), 1985.

[Oli93] Oliveira, F. M. "A Distributed AI Architecture Enabling Multi-Agent Cooperation", *Proceedings of the Sixth International Conference, Edinburgh, Scotland*, 1993.

[OMR93] Oliveira, E.; Moura F.; Rocha A.P. "Cooperation in a Multi-Agent Community", *Proceedings of International Conference On Artificial Intelligence, Expert Systems and Natural Language*, Avignon (França), 24 a 28 de maio de 1993.

[Pop93] Populaire, P.; Demazeau, Y.; Boissier, O.; Sichman, J.S. "Description et Implémentation de Protocoles de Communication en Univers Multi-Agents", *1ères Journées Francophones sur l'Intelligence Artificielle Dsitribuée et les Systèmes Multi-Agents*, AFCET & AFIA, Tolouse, 1993.

[RBP91] Rumbaugh, J.; Blaha, M.; Premerlani, W.; Eddy, F.; Lorensen, W. *Object-Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs (EUA), 1991.

[Sea69] Searle, J. *"Speech Acts"*, Cambridge University Press, 1969.

[SDB92] Sichman, J.S.; Demazeau, Y.; Boissier, O. "When can knowledge-based systems be called agents?", *Anais do Simpósio Brasileiro de Inteligência Artificial*, Rio de Janeiro (RJ), outubro 1992.

[Sho93] Shoham, Y. "Agent-Oriented Programming", *In Artificial Intelligence*, 60:51-92, 1993.

[Sic95] Sichman, J.S. *Du raisonnement social chez les agents: une approche fondée sur la théorie de la dépendance*, Thèse de Doctorat de l'INPG, Grenoble, France, 1995.

[Smi92] Smith, R.G. "The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver", *IEEE Transactions on Computers*, Vol. 29, IEEE, 1980.

[Tho93] Thomas, R. *An Agent Oriented Programming Language*, Phd Thesis, Stanford University, 1993.

[UG92] Urzelai, K.; Garijo, F.J. "MAKILA: A Tool for the Development of Cooperative Societies", In: Proceedings of the MAAMAW'92, *S. Martino al Cimino, Italy*, pp. 311-323, Castelfranchi, C.; Werner, W. (eds), Volume 830 of Lecture Notes in Artificial Intelligence, Springer-Verlag, (Itália), julho 1992.