UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
MESTRADO EM ENGENHARIA DE ELETRICIDADE

YONARA COSTA MAGALHÃES

ESPECIFICAÇÃO DE UMA SOCIEDADE DE AGENTES PARA UM SISTEMA DE APRENDIZAGEM COOPERATIVA À DISTÂNCIA

São Luís
2003

YONARA COSTA MAGALHÃES

ESPECIFICAÇÃO DE UMA SOCIEDADE DE AGENTES PARA UM SISTEMA DE APRENDIZAGEM COOPERATIVA À DISTÂNCIA

Dissertação apresentada ao Curso de Mestrado em Engenharia de Eletricidade da Universidade Federal do Maranhão, para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Sofiane Labidi

São Luís
2003

Magalhães, Yonara Costa

Especificação de uma sociedade de agentes para um sistema de aprendizagem cooperativa à distância / Yonara Costa Magalhães. – São Luis, 2003.

215 f.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Maranhão, 2003.

1. Computador – Ensino à distância. 2. Sistema multiagente. 3. Ensino cooperativo à distância. I. Título.

CDU 004:37.018.43

YONARA COSTA MAGALHÃES

ESPECIFICAÇÃO DE UMA SOCIEDADE DE AGENTES PARA UM SISTEMA DE APRENDIZAGEM COOPERATIVA À DISTÂNCIA

Dissertação apresentada ao Curso de Mestrado em Engenharia de Eletricidade da Universidade Federal do Maranhão, para obtenção do título de Mestre em Ciência da Computação.

Aprovada em / /

BANCA EXAMINADORA

_____

Prof. Sofiane Labidi (Orientador)
Doutor em Ciência da Computação
Universidade Federal do Maranhão

_____

Prof. Edison Nascimento
Doutor em Engenharia de Eletricidade
Universidade Federal do Maranhão

_____

Prof. Denis Vinícius Coury
Doutor em Engenharia de Eletricidade
Universidade de São Paulo (Escola de Engenharia de São Carlos)

Ao sempiterno Deus.

AGRADECIMENTOS

À Deus, Senhor da sabedoria e do conhecimento.

À minha família, que sempre me apoiou e em especial a minha mãe e minhas irmãs e sobrinhos.

Ao Prof. Sofiane Labidi, pelo incentivo, dedicação e orientação segura.

Ao amigo Nilson Santos pelas inúmeras revisões textuais.

Aos demais amigos: Bruno Feres, Érika Hohn, Mário Meireles, Valeska Trinta, Josenildo Costa, Jacson, Fernando Tanaka, Luciano Coutinho, pelas dicas e apoio.

À Rosângela Val, pela ajuda bibliográfica.

À coordenação da CPGEE, na pessoa da Professora Mª da Guia e ao corpo técnico da mesma: Violeta, Ione, Lílian e Alcides, sempre presentes.

Aos professores: Edson Nascimento e Zair Abdelouahab.

Aos colegas do projeto MATHNET que contribuíram com o enriquecimento deste trabalho.

RESUMO

Processo de ensino-aprendizagem cooperativo à distância da sociedade de agentes inteligentes do ambiente MATHNET. Apresenta-se a arquitetura da sociedade de agentes do MATHNET que provê o processo de ensino-aprendizagem. Destaca-se o Agente Tutor e o modelo de comunicação, baseado em troca de mensagens e que utiliza a Linguagem de Comunicação FIPA-ACL, entre o Agente Tutor e os demais agentes participantes dessa sociedade. Descrevem-se as principais características e o papel (responsabilidades) do Agente Tutor. Constrói-se o seu modelo conceitual. Modelam-se os seus principais Casos de Uso, as interações com os demais agentes dessa arquitetura e os protocolos de comunicação do Agente Tutor com os outros agentes. Cria-se o Agente Tutor utilizando para isto uma ferramenta de construção de agentes. Implementa-se o Agente Tutor e seus Casos de Uso para demonstrar sua relevância e importância dentro do sistema e o modelo de comunicação aqui proposto.

Palavras-chave: Aprendizagem baseada em computador. Sistemas Multiagentes (SMA). MATHNET. Agente Tutor. Interações. Linguagem de Comunicação de Agentes. FIPA-ACL.

ABSTRACT

Cooperative process of teach-learning at a distance of the society of intelligent agents of environment MATHNET. It is presented architecture of the society of agents of the MATHNET that to provide the process with teach-learning. It is in exchange for distinguished the Tutor Agent and the model of communication, based messages and that it uses the FIPA-ACL Agent Communication Language, between the Tutor Agent and excessively the participant agents of this society. The main characteristics and the paper (responsibilities) of the Tutor Agent describe. Its conceptual model is constructed. Its main Cases Use, interactions with several agents of this architecture and the protocols of communication of the Tutor Agent with the other agents are shaped. The Tutor Agent creates itself using a tool of construction of agents. One implements the Tutor Agent and its Cases Use to inside demonstrate to its relevance and importance of the system and the model of communication.

Keywords: Computer Based Learning. Multi-agent Systems (MAS). MATHNET. Tutor Agent. Interactions. Agent Communication Language. FIPA-ACL.

“A maravilhosa disposição e harmonia do universo só pode ter tido origem segundo o plano de um Ser que tudo sabe e tudo pode. Isto fica sendo a minha última e mais elevada descoberta.”

(Isaac Newton)

APÊNDICES

SUMÁRIO

LISTA DE SIGLAS

ARPA	- Agência de Projetos de Pesquisa Avançada dos Estados Unidos
BA	- Agente Broker
BDI	- Banco de Dados Inteligente
CA	- Ato Comunicativo
CSCL	- Aprendizagem Cooperativa Suportado por Computador
LCAE	- Linguagem de Cooperação de Agentes Educacional
FIPA	- Fundação para Agentes Físicos Inteligentes
FTP	- Protocolo de Transferência de Arquivo
HTTP	- Protocolo de Transferência Hipertexto
IAD	- Inteligência Artificial Distribuída
KQML	- Linguagem de Manipulação e Consulta de Conhecimento
KSE	- Força Tarefa de Compartilhamento de Conhecimento
LC	- Linguagem de Comunicação
LCA	- Linguagem de Comunicação de Agentes
LMD	- Linguagem de Modelagem de Domínio
MATS	- Sistema Tutorial Multiagente
OMG	- Grupo de Gerenciamento de Objeto
RDP	- Resolução Distribuída de Problema
SHIECC	- Sistema Hipermídia Inteligente de Ensino Cooperativo Computadorizado
SMA	- Sistema Multiagente
SMTP	- Protocolo de Transferência de Mensagem Simples
STI	- Sistema Tutor Inteligente
TA	- Agentes Tutores
TCP	- Protocolo de Controle de Transmissão
UML	- Linguagem de Modelagem Unificada
US DARPA	- Agência de Projetos e Pesquisas Avançadas do Departamento de Defesa dos Estados Unidos

LISTA DE FIGURAS

LISTA DE QUADROS

1 INTRODUÇÃO

Os processos de ensino-aprendizagem são geralmente encarados como sendo compostos por estratégias meramente transmissivas. Isto devido a maioria partir do pré suposto que o conhecimento é um mero conteúdo ou um dado, ou seja, algo já pronto e acabado e que o ensino-aprendizagem deve, portanto, se resumir na transmissão destes dados pelos professores e à sua assimilação pelos alunos. Assim, o conhecimento é identificado como o resultado desta assimilação onde um pólo se manifesta ativo (o professor) e outro passivo (todos os alunos).

Entretanto, o conhecimento não é estático. Ele deve ser visto como a capacidade de estabelecer conexões entre informações aparentemente desconexas, processá-las, analisá-las, relacioná-las, armazená-las e avaliá-las segundo critérios de relevância, organizando-as em sistemas de forma a construir um significado, ou seja, perceber e compreender os fatos e/ou objetos em relação a outros. Deve-se conceber o conhecimento como algo inacabado, resultado do trabalho interativo de alunos e professores, tendo como ponto de partida as informações socialmente disponíveis.

Por isto, o professor é o mediador e orientador dos processos que permitirão a construção dos significados e não apenas simples repassadores de informações.

É nesse contexto que os Sistemas Tutoriais Inteligentes (STI) são aplicados. STIs são sistemas computacionais com modelos de conteúdo instrucionais que especificam **o que** ensinar e estratégias de ensino que especificam **como** ensinar (MURRAY, 1999, grifo nosso). Sendo um foco de pesquisa da

Inteligência Artificial na área da educação, cujo objetivo é a descoberta das formas de fazer do computador um tutor inteligente.

Sua característica mais importante é prover ensino adaptativo, permitindo a realização de atividades apropriadas para que o nível de habilidade do usuário mude, à medida que aumenta a experiência no ensino, ou seja, é um sistema cujo principal objetivo é ensinar um determinado assunto de forma a se preocupar com o aluno, não apenas expondo todo o conteúdo de um domínio restrito, mas apresentá-lo da melhor maneira possível a fim de que o aluno consiga compreender ao máximo todo o ensinamento. E, embora, o aluno esteja inserido em um grupo isto é feito de forma individualizada levando-se em conta características do aluno como: conhecimento sobre determinado assunto, preferências e aspectos psicológicos, entre outros.

Neste trabalho é apresentado o MATHNET, que é um ambiente baseado em uma arquitetura Multiagentes, isto é, em uma sociedade de agentes inteligentes, onde cada agente possui as suas tarefas e se comunica com os demais agentes realizando o processo de ensino-aprendizagem pela cooperação entre os agentes humanos e artificiais existentes neste sistema; e, nos conceitos de STI.

1.1 Objetivo do trabalho

O objetivo desta dissertação é de propor a especificação de uma Sociedade de Agentes (SA) para um sistema de aprendizagem cooperativa à distância, mostrando a aplicação desse serviço no ambiente MATHNET de ensino – aprendizagem cooperativa computadorizada, enfatizando o Agente Tutor e o modelo de comunicação entre os agentes existentes baseado na troca de mensagens.

Especificamente pretende-se:

a) Definir quais agentes são necessários e suficientes no MATHNET;

b) Definir os seus papéis (roles);

c) Propor protocolos de interação entre os agentes nessa sociedade;

d) Identificar os principais Casos de Uso que envolvam o Agente Tutor;

e) Identificar os agentes necessários e suficientes para esses Casos de Uso;

f) Definir as colaborações e os protocolos de comunicação do Agente Tutor com os demais agentes participantes desses Casos de Uso;

g) Implementar os Casos de Uso identificados.

1.2 Justificativa e relevância

A abordagem multiagentes para os STIs permitiu que novas facilidades fossem introduzidas como a modelagem de múltiplas estratégias de ensino e as múltiplas representações do conhecimento associadas a diversas formas de exploração do mesmo. De modo geral, esta abordagem trouxe a outros sistemas a flexibilidade necessária devido às características inerentes dos agentes. Além disso, é através da modelagem das interações entre os agentes de uma sociedade que as possíveis trocas de mensagens podem ser identificadas e implementadas. Assim que, ao se levar em conta todos esses aspectos, torna-se clara a importância de um processo de especificação bem feito.

A importância da especificação dentro de qualquer projeto deve-se ao fato de que ela permite: 1) Documentar o projeto; 2) Guiar o projeto de implementação de forma a permitir que o mesmo seja bem concebido; 3) Retirar as ambigüidades que possam existir; e, 4) Demonstrar como ocorre a comunicação entre os agentes, dentro desse ambiente.

Deste modo, a especificação dessa sociedade de agentes que compõe o sistema MATHNET, delimitada em seus componentes, permitirá uma visão clara desse sistema, bem como, uma melhor compreensão de porquê a comunicação entre os agentes (artificiais e humanos) é de vital importância no processo de ensino-aprendizagem. Além disso, o Agente Tutor em foco nesse trabalho estará trabalhando em conjunto com outros agentes modelados em trabalhos anteriores e que pertencem a este mesmo projeto.

1.3 Organização da dissertação

Esta dissertação é composta por oito capítulos. No Capítulo 2, apresenta-se o ambiente MATHNET, sua arquitetura e a sociedade de agentes que o compõe. No Capítulo 3, são apresentados os fundamentos necessários para compreender a comunicação entre Agentes, devido à cooperação existente entre eles, e que serve de embasamento para o modelo de comunicação de agentes proposto neste trabalho. No Capítulo 4, destaca-se como foco principal deste trabalho o Agente Tutor no contexto MATHNET, com suas características e responsabilidades. No Capítulo 5, apresentam-se os diagramas de interação dessa sociedade de agentes que foram construídos em Linguagem de Modelagem Unificada (UML) para os casos de uso que envolvem o Professor e o Aprendiz. No Capítulo 6, são mostrados os

diagramas contendo os protocolos de comunicação entre os agentes e que foram construídos com base nos diagramas de Interação descritos no Capítulo 5. Já no Capítulo 7, visualiza-se a implementação de casos de uso que envolve a comunicação do agente Tutor com outros agentes do MATHNET, utilizando-se para isso a linguagem Java e a ferramenta ZEUS. Por fim, no Capítulo 8, conclui-se esse trabalho discutindo os resultados alcançados e mostrando as suas perspectivas futuras.

2 AGENTES E SOCIEDADE DE AGENTES

Neste capítulo destaca-se conceito de Agente, baseada em suas características, o ambiente MATHNET com sua arquitetura, componentes, objetivos, aplicabilidade e a sociedade de agentes que o compõe. Possibilitando dessa forma descrever o quadro do ambiente no qual está inserido este trabalho.

2.1 Definindo um Agente

A padronização dos agentes, em seus vários aspectos, tem sido alvo de um constante interesse por parte de vários pesquisadores e organizações. De modo que, dessas organizações surgiram propostas de padronização para esta área.

Assim instituições como a OMG (OMG, 2001), FIPA (GREAVES, 2000), US Darpa (DARPA, 2001), KQML (OMG, 2000), AgentLink (UNIVERSITY OF SOUTHAMPTON, 2001) e Climate (OMG, 2000; CLIMATE, 2001) voltaram-se para programas de pesquisa que abordam vários aspectos da tecnologia de agentes (arquitetura, linguagem, protocolo, etc.) na tentativa de desenvolver e promover algum tipo de padronização que pudesse vir a ser aceita pela maioria.

Todo esse interesse em torno desse tipo de padronização deve-se ao fato de que a tecnologia de agentes está integrada à aplicação de várias outras tecnologias (ver Fig. 1), adicionando-lhes desse modo um conjunto de novas habilidades às aplicações já existentes (ODELL, 2000). De modo que, essa vasta aplicabilidade desperta um interesse natural na comunidade de pesquisadores.

Em relação à definição do que seja um agente isto ainda é um tópico em discussão, mas em geral concorda-se que um agente é uma parte de software dotado de alguma inteligência, que possui um papel bem definido e um conjunto de

crenças, podendo ser autônomo e pró-ativo trabalhando em benefício do usuário representando os seus interesses, para prover algum serviço ou dado. Além disso, realiza tarefas mais eficientemente do que uma pessoa realizando ações altamente repetitivas e computacionalmente pesadas (GALAN, 2000).

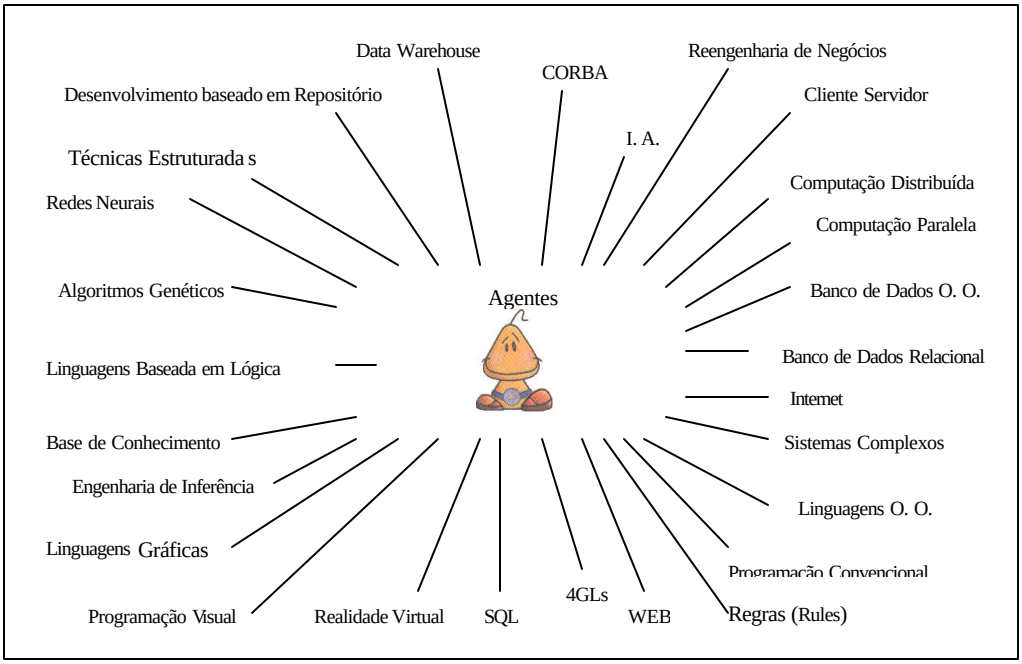

Figura 1 – Agentes podem ser usados com outras tecnologias

Em vista da diversidade de opiniões, uma definição de agente que pode ser adotada é: “uma entidade autônoma que pode interagir com seu ambiente” (ODELL, 1998) (ver Fig. 2). Entretanto, como o próprio autor expressa mais adiante neste mesmo artigo, esse é o sentido mais geral de um agente, pois eles não são particularmente proveitosos a menos que sejam autônomos, comunicativos (habilidade social), reativos e móveis. Sendo que algumas das propriedades que o caracterizam são (OMG, 2000; ODELL, 1998):

a) **autonomia** – é a capacidade de atuar sem uma intervenção externa direta;

b) **interativo/comunicativo (Habilidade Social)** – comunica-se com o ambiente e outros agentes (humanos ou outras entidades) através de uma linguagem de comunicação. Sendo deste modo sociáveis;

c) **reativo/adaptativo** – é capaz de perceber e responder, de maneira adequada, às mudanças de seu ambiente (mundo físico, interface gráfica, coleção de outros agentes, Internet ou uma combinação desses elementos);

d) **mobilidade** – habilidade para transportar-se de um ambiente para outro;

e) **pró-ativo** – orientado a objetivos, decidido. Não apenas reage ao ambiente;

f) **inteligente** – o estado é formalizado pelo conhecimento (isto é, objetivos, crenças, planos, suposições) e interage com outros agentes usando linguagem simbólica;

g) **coordenativo** – capaz de executar algumas atividades em um ambiente compartilhado com outros agentes; e,

h) **cooperativo/colaborativo** – capaz de se coordenar com outros agentes para alcançar propósitos comuns.

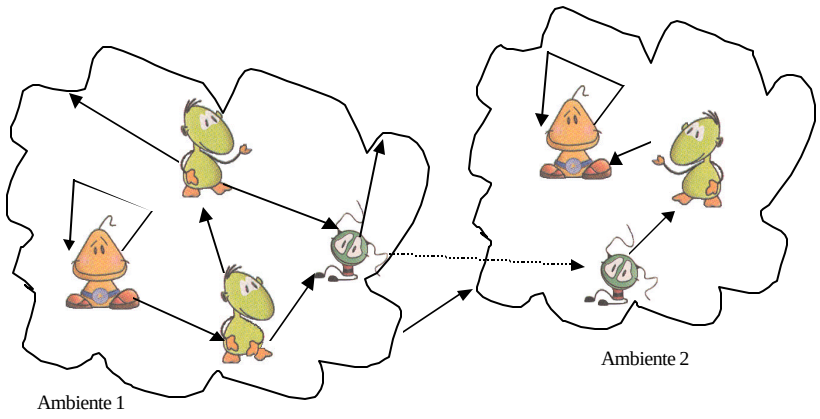

Figura 2 - Características dos Agentes

A utilização de agentes justifica-se por (ODELL, 1998):

a) fazer e projetar decisões;

b) solucionar grandes problemas de otimização combinatorial;

c) detectar e responder a oportunidades e mudanças em situações dinâmicas complexas.

No caso MATHNET, justifica-se a utilização de agentes devido ao fato de se requerer a existência das características de autonomia, mobilidade, pró-atividade, cooperatividade e interatividade.

Isto proporciona aos alunos e professores um ambiente de aprendizagem cooperativa eficiente. Todas essas características dos agentes são necessárias para o desenvolvimento do sistema MATHNET.

2.2 Definindo uma SA

Quando uma pessoa precisa resolver um problema para o qual não detém conhecimento suficiente ela requisita o auxílio de outra pessoa (ou de outros recursos, como: livros, apostilas, etc.) capaz de solucioná-lo. Da mesma forma que no mundo real, os agentes, em algumas situações, podem executar tarefas com ou sem a ajuda de outros agentes, para alcançar um objetivo.

A Figura 3, a seguir, representa de forma simples, a idéia de trabalho cooperativo quando um grupo de agentes trabalha em conjunto visando alcançar um objetivo comum e onde cada um é responsável por realizar tarefas diferentes.

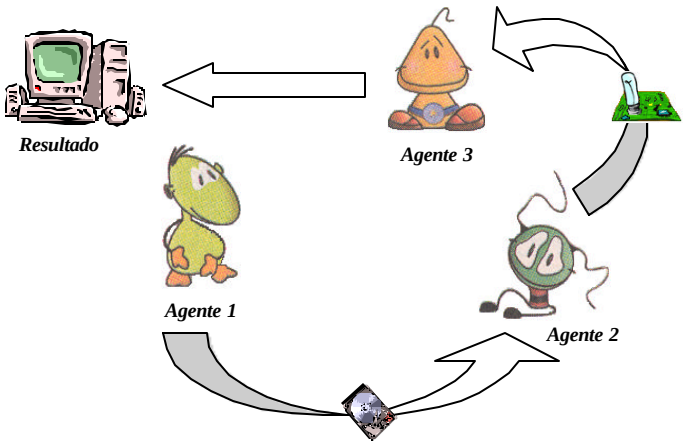

Figura 3 – Trabalho cooperativo

Isso não implica apenas na divisão de tarefas, que parece óbvio, mas também existe a necessidade de comunicação entre esses agentes para que a tarefa seja realizada com sucesso.

A interação entre os agentes é, portanto, uma característica importante, entretanto, ela não é suficiente para construir uma sociedade de agentes. Para tal, é necessário coordenação (seja através de competição, cooperação ou a combinação de ambas). Essa sociedade de agentes é chamada de Sistemas Multiagentes (SMA) e são sistemas compostos de agentes coordenáveis através de seus relacionamentos uns com os outros (OMG, 2000).

A tecnologia de agentes tem origem na Inteligência Artificial Distribuída (IAD). Sendo que a IAD dividida em duas principais áreas de investigação (JENNINGS, 1996): a Resolução Distribuída de Problemas (RDP) (UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2002a, 2002b) e SMA.

A RDP estuda as técnicas para resolver problemas específicos dividindo o trabalho entre muitos módulos que cooperam interagindo e trocando conhecimentos sobre o problema e a sua solução, ou seja, os agentes cooperam uns com os outros, dividindo e compartilhando conhecimento sobre o problema e sobre o processo de

obter uma solução. Assim, nesta abordagem, os agentes são projetados especificamente para resolver aquele problema ou classe de problemas.

Sob um ponto de vista externo, um sistema RDP é visto como uma unidade. O processo de coordenação das ações dos agentes é definido em tempo do projeto. Pode-se representá-la conforme a Figura 4, a seguir. Onde inicialmente parte-se do problema, passando pela criação dos agentes até chegar à solução do problema pela utilização desses agentes.

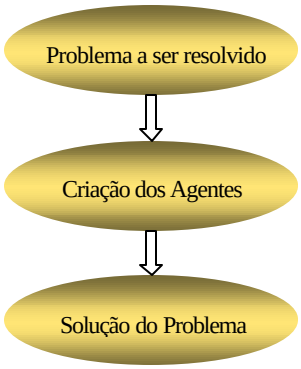

Figura 4 - RDP

Já na área de SMA, o projetista não volta sua atenção para um problema específico, mas para um domínio específico. Nesta abordagem, a idéia consiste em coordenar o comportamento inteligente de um conjunto de agentes autônomos, cuja existência pode ser anterior ao surgimento de um problema em particular. Os agentes devem então raciocinar a respeito das ações e sobre o processo de coordenação em si.

Suas arquiteturas são mais flexíveis e a organização do sistema está sujeita a mudanças visando adaptar-se às variações no ambiente e/ou no problema a ser resolvido. A SMA estuda o comportamento inteligente em uma sociedade de agentes autônomos, como coordenar seus conhecimentos, metas e planos para

resolver problemas. Deste modo, pode-se representá-la conforme a Figura 5 a seguir.

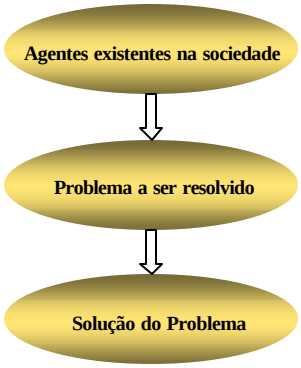

Figura 5 - SMA

Na abordagem SMA as seguintes características são encontradas:

a) a decomposição das tarefas é feita pelos agentes;

b) os agentes são autônomos;

c) os agentes são hábeis em solucionar mais de um problema;

d) é um sistema aberto, ou seja, agentes podem entrar e sair dessa sociedade quando quiserem, possibilitando que a sociedade se adapte à novas situações. A migração de agentes inteligentes e autônomos entre sociedades não é o foco desse trabalho, maiores detalhes podem ser vistos em (HÜBNER, 1995);

e) o ambiente dos agentes pode ser alterado, ou seja, a organização interna do mundo dos agentes pode sofrer alterações.

Assim que, os SMA são sistemas que se caracterizam por interações entre diversos agentes para cumprirem um objetivo final. E embora, cada agente possa ter um objetivo diferente, ou até mesmo, não ter nenhum (simplesmente reativos), diferentes agentes poderão ter diferentes tarefas mas que no conjunto levam à resolução do problema.

Em relação à avaliação de desempenho de um SMAs, deve-se levar em conta algumas propriedades não funcionais, como performance, escalabilidade e estabilidade (LEE et al., 1998). A seguir descreve-se brevemente cada um desses conceitos aplicados à SMA, que são:

a) **performance** – assim como ocorre em sistemas distribuídos os indicadores de performance básicos em SMA são os custos computacionais e o *throughput*. Performance está associada a uma medida que usa um conjunto de indicadores das principais saídas do sistema e seu consumo de recursos, onde indicadores típicos incluem: *throughput*, tempo de resposta, número de tarefas/agentes concorrentes, tempo computacional e *overhead* de comunicação. Neste caso, as variáveis que afetam a performance incluem, mas não exclusivamente, o número de agentes existentes, o número de tarefas/objetivos concorrentes que os agentes executam, a organização dos agentes e o tipo de protocolos de coordenação empregado;

b) **escalabilidade** - Está relacionada com a capacidade que o sistema tem de fazer aumentar a sua capacidade de trabalho à medida que o tamanho do sistema cresce. Assim que em SMA, escalabilidade significa que o aumento da carga de um agente é causado pela sua necessidade de interagir com mais agentes devido a um aumento do tamanho da sociedade; e,

c) **estabilidade** – este não é um conceito muito claro, pois intuitivamente um SMA está em equilíbrio quando as propriedades estatísticas dos seus indicadores de performance ficam estacionárias para uma dada

variação na carga externa do sistema. A instabilidade ocorre então quando ocorrem perturbações de alguma ordem como ruídos (interferências eletrônicas que atrapalham a comunicação), variações nos valores de parâmetro, inclusão ou desabilitação de agentes.

Existem razões para a aplicação de SMAs ao invés de utilizar apenas agentes isolados, pois em muitos casos sua utilização incluem flexibilidade, escalabilidade, descentralização e robustez. Assim que, dentre essas razões estão:

a) a divisão das funções entre muitos agentes provê modularidade, flexibilidade e características modificativas e extensivas. Já que um único agente que faz várias coisas representa um obstáculo para a rapidez, a confiabilidade e torna difícil a manutenção (ver Figura 6);

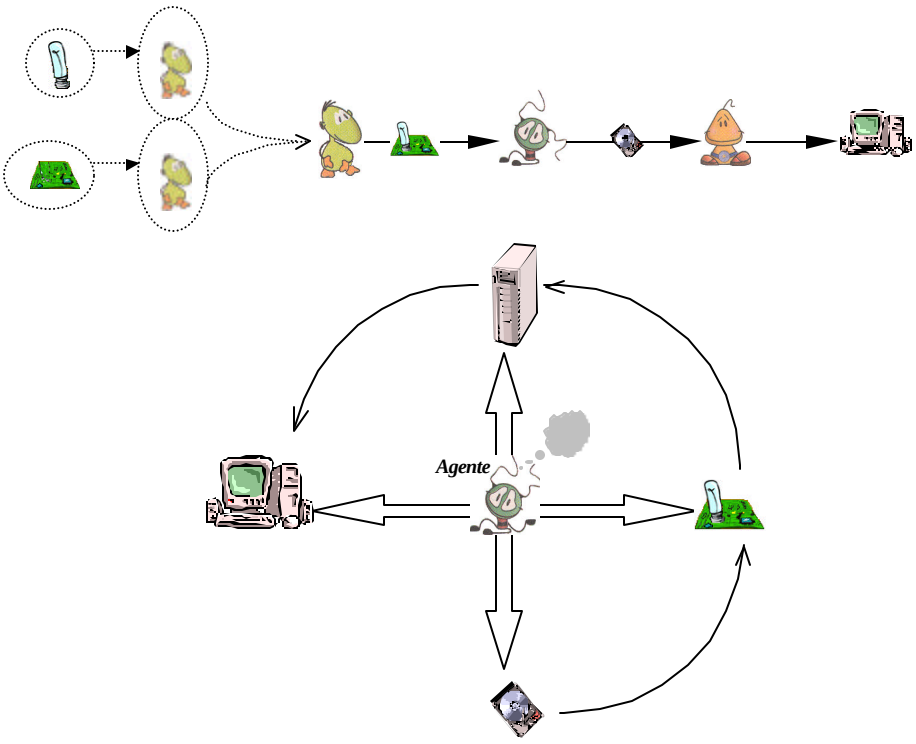

Figura 6 - SMA x Agente isolado

b) o conhecimento especializado não está freqüentemente disponível por um único agente. Assim que, o conhecimento extendido sobre vários

agentes pode ser integrado para uma visão mais completa quando necessário (descentralização);

c) aplicações que requerem computação distribuída são melhor suportadas por SMA, pois os agentes podem ser projetados como componentes autônomos de granularidade fina que atuam em paralelo.

Para suportar SMA um ambiente deve (OMG, 2000):

a) prover uma infraestrutura específica para comunicação e interação de protocolos;

b) ser tipicamente aberto e não ter um "designer" centralizado ou um controle de função *top-down*;

c) possuir agentes que sejam autônomos, adaptativos e coordenadores.

2.3 Ambiente MATHNET

O projeto MATHNET baseou-se no Sistema Hipermídia Inteligente de Ensino Cooperativo Computadorizado (SHIECC) (LABIDI et al., 1998b; FERREIRA et al., 1998; FERREIRA, 1998) e MATHEMA (COSTA, 1997), unindo os vários conceitos utilizados nesses dois sistemas com o objetivo de desenvolver um sistema para efetivamente explorar o paradigma CSCL (Aprendizagem Cooperativa Suportado por Computador), ou seja, um sistema baseado em agentes e que provê um ambiente de aprendizagem cooperativa à distância suportado por computador (NUNES et al., 2000).

Deste modo, a proposta do MATHNET é criar um ambiente onde estudantes são separados em vários grupos organizados para cooperar e aprender através da interação com seu próprio grupo, com o sistema, o professor e outros

grupos. Sendo que estes grupos podem usar os mais variados recursos, desde multimídia até a tecnologia da Intranet. Integrando deste modo a idéia dos STI com ambientes de rede o MATHNET define um ambiente cooperativo tutor inteligente (LABIDI et al., 2000a).

Os grupos são chamados de áreas cooperativas. Uma área cooperativa é geralmente formada por três estudantes interagindo com o sistema através de um terminal. Esta quantidade foi sugerida pelo fato de que um número inferior não proporcionaria a quantidade de interações adequadas a um trabalho cooperativo e um número superior poderia acarretar problemas, considerando-se o fato de que os alunos estarão interagindo com o computador (LABIDI et al., 1998). É importante ressaltar a existência no MATHNET do conceito de grupos virtuais. Isto significa que os grupos podem ser formados com pessoas geograficamente separadas, sem que haja prejuízos ao processo de ensino-aprendizagem.

Desta forma, os estudantes cooperam usando o computador como um tutor ativo. O terminal também pode ser visto como uma ferramenta para mediar as interações com o professor e os outros grupos de estudantes. As interações são suportadas através de recursos de comunicação de uma rede de computadores, que pode ser uma LAN, MAN ou WAN através de uma INTRANET ou INTERNET. Um professor interagindo com o sistema através de um terminal é considerado como uma área cooperativa específica (ver Fig. 7).

A aprendizagem no MATHNET é um processo que ocorre em virtude da cooperação existente entre todos os elementos que o compõe (Aprendiz, Professor, Especialista/Engenheiro do Conhecimento e os agentes artificiais). E é nesse contexto que o processo de aprendizagem cooperativa ocorre no âmbito da aplicação do domínio por parte dos aprendizes. Esse processo é esquematizado no

MATHNET como sendo composto por seis Fases/Atividades Pedagógicas (ver Fig. 8) (COUTINHO et al., 2000).

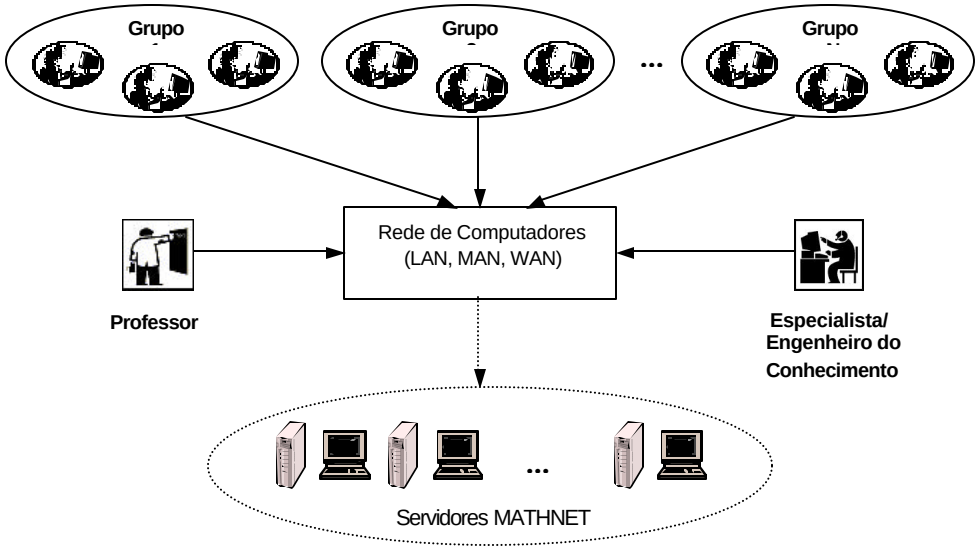

Figura 7 – Arquitetura do Ambiente MATHNET

Cada uma dessas fases/atividades é responsável por um determinado aspecto no processo de ensino-aprendizagem do MATHNET. Deste modo, à cada atividade estão associadas funções específicas que são desempenhadas com o uso de estratégias pedagógicas apropriadas, escolhidas de acordo com o modelo do aprendiz.

Na Figura 8, pode-se ainda observar que em determinados momentos algumas atividades são realizadas paralelamente à outras ao longo do tempo, ou seja, na verdade após a fase de Preparação dos Grupos, em qualquer momento, mais de uma fase/atividade estará sendo realizada ao mesmo tempo durante esse processo.

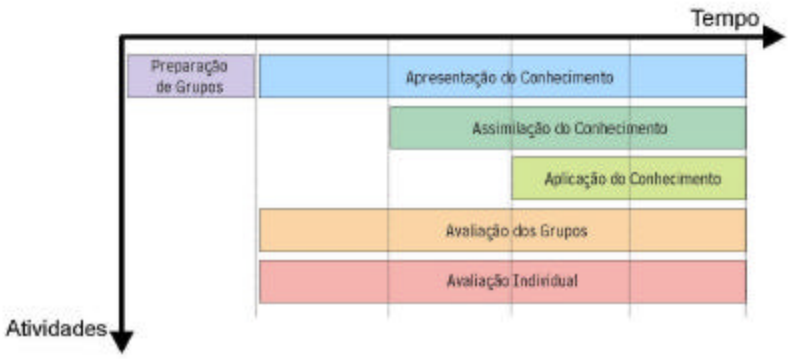

Figura 8 – Atividades Pedagógicas do MATHNET

As estratégias pedagógicas aplicadas no MATHNET visam alcançar uma maior eficiência no processo de ensino-aprendizagem de acordo com as diferenças comportamentais dos aprendizes em cada uma dessas fases (SERRA, 2001). Existem as estratégias:

a) **Global** – que é definida ou modificada pelo Professor nos intervalos entre as sessões de ensino-aprendizagem, ou seja, essa estratégia faz parte do planejamento pedagógico realizado antes do início de uma sessão de ensino-aprendizagem. Essa estratégia diz respeito à seqüência das atividades pedagógicas e ao tempo alocado para cada uma delas; e,

b) **Específicas de Atividades** - que podem ser modificadas pelo Agente Estrategista ou Professor durante as fases/atividades em uma sessão, ou seja, dependendo da fase/atividade em que o aprendiz se encontre pode-se aplicar estratégias que mais se adeqüem a cada um desses momentos. Dentre as estratégias pode-se citar como, por exemplo: Fase de Apresentação → Expositiva e Argüição; Fase de Assimilação → Discussão; Fase de Aplicação → Negociação e Competição.

Desta forma, existe no sistema uma flexibilidade em relação às estratégias que podem ser aplicadas, já que elas podem ser modificadas, a qualquer hora, pelo Professor. É o Agente Estrategista, interagindo com o Agente de Modelagem do Aprendiz, o responsável por selecionar a estratégia mais adequada de ensino com base no modelo do aprendiz apresentado.

As fases/atividades pedagógicas são descritas a seguir (LABIDI et al., 2000a; COUTINHO et al., 1999; SERRA et al., 2001). Elas servem como base para a fase de modelagem, onde os Casos de Uso, modelados em UML (ver Apêndice A), do MATHNET serão especificados:

a) **Preparação de Grupos** - onde é gerado um perfil inicial de cada aprendiz para a formação dos grupos;
b) **Apresentação do Conhecimento** - onde é feita a apresentação do conhecimento ao aprendiz. É baseada no perfil de cada um deles montado pelo Agente de Modelagem do Aprendiz e que é usada pelo Agente Estrategista para definir a apresentação mais adequada;
c) **Assimilação do Conhecimento** - nela os alunos interagem entre se com o objetivo de assimilar o conhecimento apresentado através de discussões. É na verdade a primeira atividade realmente cooperativa;
d) **Aplicação do Conhecimento** - nela os alunos aplicarão o conhecimento anteriormente apresentado através da realização de várias atividades e pela resolução de problemas. Essa atividade é importante para a fixação e para o acompanhamento da assimilação do conhecimento apresentado por parte do professor;

e) **Avaliação dos Grupos** - para essa avaliação final o sistema utiliza informações adquiridas em todas as atividades desde o momento que os alunos começam a assistir a apresentação do conhecimento;

f) **Avaliação Individual dos aprendizes** - é feita de forma semelhante à Avaliação dos Grupos, mas levando-se em conta o aprendiz individualmente.

O MATHNET encontra-se estruturado de forma a constituir uma sociedade de agentes que cooperam entre si para realizar o processo de ensino-aprendizagem, baseando-se deste modo na arquitetura multiagentes.

O sistema MATHNET possui um conjunto de agentes que utilizam os recursos do sistema de forma distribuída. Esses agentes encontram-se classificados, conforme a Tabela 1 (LABIDI et al., 2000a; LABIDI et al., 2000b):

AGENTES DO MATHNET	
HUMANOS	Professor
	Aprendizes
	Engenheiro do Conhecimento
ARTIFICIAIS	Tutor
	Domínio
	Modelagem do Aprendiz
	Estrategista
	Busca

Quadro 1 – Os agentes do MATHNET

No MATHNET, tanto os agentes artificiais quanto os agentes humanos formam uma sociedade que cooperam entre se para alcançar um objetivo comum, que nesse caso é o Ensino-Aprendizagem, sendo que cada um deles tem um papel bem definido dentro dessa sociedade (ver Fig. 9).

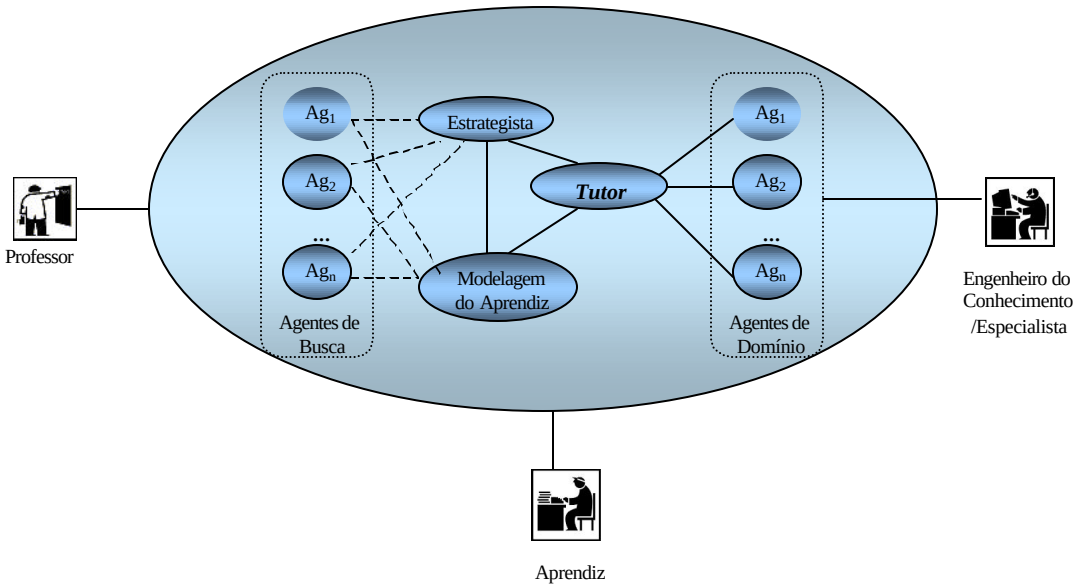

Figura 9 – Arquitetura Multiagentes do MATHNET

Esse conjunto principal de agentes artificiais, provê todas as funções tutoras principais para o aprendiz (LABIDI at al., 2000a):

a) **Agente Tutor** – é responsável pela apresentação do conhecimento para diferentes estudantes em diferentes áreas cooperativas, ou seja, apresentar aos aprendizes o conteúdo adequado para cada um deles. Ao mesmo tempo em que é responsável por controlar a interação dos grupos de estudantes com o sistema durante o processo de ensino-aprendizagem e observar o comportamento tanto dos grupos quanto de cada aprendiz individualmente ao longo de todo o processo para realizar avaliação;

b) **Agente de Domínio** – embora exista um Agente de Domínio para cada unidade de conhecimento, todos possuem a mesma estrutura. Cada agente é responsável por prover o conteúdo (texto, vídeo, som, etc.) que será apresentado ao aprendiz. E cada Agente de Domínio é responsável pela representação de um conhecimento específico. Foi desenvolvida uma Linguagem de Modelagem de Domínio (LMD) para

modelar (armazenar e distribuir) o domínio desse conhecimento (COSTA, 2002). Interage com os Agentes Engenheiro do Conhecimento, Tutor e Professor;

c) **Agente Estrategista** – é responsável por definir e propor as estratégias pedagógicas a serem aplicadas durante o processo de aprendizagem. A escolha da melhor forma de apresentar um conteúdo é baseada em critérios pedagógicos e em informações obtidas pelo modelo do Aprendiz. Não existe apenas uma única estratégia pedagógica aplicada no MATHNET. O Agente Estrategista interage com os Agentes de Modelagem do Aprendiz e Tutor;

d) **Agente de Modelagem do Aprendiz** – este agente conhece, mantém e representa as informações sobre o aluno individualmente e seus grupos dentro do sistema ao longo do processo de ensino-aprendizagem. Possuindo uma grande interação como o Agente Estrategista desde que as decisões pedagógicas sejam principalmente baseadas nos resultados obtidos pelo modelo do aprendiz (COUTINHO et al.,2000; SERRA, 2001). Além disto, interage com o Agente Tutor;

e) **Agentes de Busca** - esses agentes são responsáveis por buscar informações que auxiliem os aprendizes a responderem suas dúvidas no MATHNET, ou seja, sua principal função no MATHNET é auxiliar os aprendizes, individualmente ou em grupo, a encontrar informações para responder suas dúvidas. O próprio Agente de Busca pode oferecer-se para buscar alguma informação por conta própria, o que denota um certo grau de pró-atividade. Por serem móveis, os agentes

de busca podem funcionar de maneira autônoma e assíncrona. A utilização do paradigma de agentes móveis permite que um aprendiz possa criar um Agente de Busca, que por sua vez pode criar e depois enviar vários outros agentes-filhos, simultaneamente, para cada uma das bases de recursos disponíveis no sistema. Desse modo, a busca é realizada em paralelo aumentando o desempenho do serviço (NUNES, 2001). Os Agentes de busca podem ser acionados pelo Agente Estrategista ou pelo Agente de Modelagem do Aprendiz.

Os agentes humanos utilizam uma interface de sistema específica, onde suas necessidades e direitos são considerados:

a) **Professor** – esse agente humano pode assumir diferentes papéis no sistema fazendo as vezes também de Engenheiro do Conhecimento. O Professor pode interagir com o grupo específico ou com todos os grupos a qualquer momento. O sistema provê um mecanismo de envio/recebimento de mensagens que pode ser utilizado diretamente, ou quando possível (LABIDI et al., 2000b). Dentre as atividades que o Professor pode realizar no sistema encontram-se: 1) formar grupos e reorganizá-los, quando necessário; 2) fornecer o conteúdo a ser ensinado; 3) alterar as estratégias pedagógicas propostas e/ou adotadas; 4) realizar a avaliação individual e dos grupos; 5) supervisionar e interagir com as áreas cooperativas;

b) **Aprendizes** – são os alunos. E encontram-se alocados em grupos compostos por três pessoas. Antes de começar uma sessão de aprendizagem o sistema MATHNET é responsável por montar um perfil inicial de cada aluno e organizá-los em grupos;

c) **Engenheiro do Conhecimento** – pode também ser conhecido como Especialista, pode ser um papel assumido pelo próprio Professor. Foi desenvolvida uma Ferramenta de Autoria para o MATHNET de modo a permitir que o domínio de conhecimento seja modelado de acordo com a LMD (COSTA, 2002).

O Agente Tutor será mais detalhado no Capítulo 4, sendo apresentada suas características, papel e importância dentro do ambiente MATHNET.

2.4 Considerações Finais

Neste capítulo apresentou-se uma sucinta descrição sobre o sistema de agentes, a estrutura da sociedade de agentes existente no ambiente MATHNET, seus conceitos básicos, os agentes humanos e artificiais que compõe essa SA e as Atividades Pedagógicas que definem todo o processo de ensino-aprendizagem no MATHNET.

Nos próximos capítulos serão enfatizados os conceitos que envolvem a comunicação entre agentes, as linguagens de comunicação existentes, bem como suas características.

3 LINGUAGEM DE COMUNICAÇÃO DE AGENTES

Neste capítulo serão destacados os conceitos básicos de comunicação, o conceito de Linguagem de Comunicação de Agentes (LCA), seus componentes e suas características desejáveis, bem como alguns tipos de LCA, dentre elas KQML e FIPA-ACL. Isto serve como embasamento para a proposta de comunicação de agentes na forma de troca de mensagens proposta neste trabalho.

3.1 Teoria dos Atos da Fala

Antes de mencionarmos alguns dos tipos de linguagem de comunicação de agentes é importante ressaltar a Teoria dos Atos da Fala ou Ações da Fala (SEARLE, 1980; WOOLDRIDGE, 2000) também conhecida como *speech acts*, que é a base da formação dos dois tipos de LCAs que serão apresentadas nas seções 3.3.1 e 3.3.2.

A comunicação humana é usada como modelo para a comunicação entre agentes. Uma base para analisar a comunicação humana é a teoria das ações de fala ou de discurso. Esta teoria vê a comunicação humana na forma de *ações*, tais como pedidos, sugestões, compromissos e respostas. Deste modo, a teoria parte do princípio que as pessoas pensam sobre o que vão dizer para atingir os seus objetivos. Elas planejam os seus atos de fala para afetar as crenças, objetivos e estados emocionais dos ouvintes. A idéia é que tal linguagem pode ser modelada como operadores em um sistema de planejamento, logo os atos de fala podem ser integrados em planos.

Um plano é formado por ações que são consideradas performativas. Essa teoria propõe então uma espécie de categorização de primitivas de comunicação como, por exemplo, *inform, ask-to-do, answer, propose,* etc. Essas primitivas são associadas a ações e suas conseqüências.

Sendo assim, a teoria de atos de fala pode ser utilizada como uma ferramenta para a interpretação do significado pretendido com as mensagens trocadas durante a interação, pois o reconhecimento dos atos de fala nos permite identificar as ações e os personagens envolvidos (emissor e receptor) em uma comunicação.

Uma ação de fala tem três aspectos:

a) **Locução (*locution*)** – que é a declaração física do interlocutor, ou seja, representa o que é dito para o ouvinte. Sendo que essa emissão de palavras e sentenças possui algum significado;

b) **Ilocução (*illocution*)** – expressa o significado desejado para a declaração feita pelo interlocutor, ou seja, demonstra o modo (sentido/intenção) que o emissor transmite para o ouvinte;

c) **Perlocução (*perlocution*)** - ação resultante da locução, ou seja, é uma ação que ocorre como resultado de uma ilocução.

Sendo que uma ilocução é dividida em duas partes, que são: a força de ilocução e uma proposição (uma afirmação que pode ser verdadeira ou falsa). Por exemplo, na frase a seguir:

“Estou com frio.”

Pode-se fazer as seguintes observações, quanto aos aspectos desta frase:

a) A frase é uma locução;

b) O modo com que o emissor expressa esta frase, é a ilocução;

c) Caso o ouvinte interprete corretamente a frase isto acarretará uma ação, que pode ser, por exemplo: desligar o ar condicionado, pegar um cobertor, etc. A ação resultante do entendimento da frase é a perlocução.

A Teoria dos Atos da Fala usa o termo performativa para identificar a força de ilocução das declarações que não deixam dúvida quanto ao seu propósito. A força de ilocução pode ser classificada de forma abrangente nos seguintes tipos:

a) Assertiva – que define a constatação de fatos;

b) Diretiva - define os comandos de uma estrutura mestre-escravo;

c) Comprometedora - as que causam compromissos;

d) Declarativas – as que expressam a afirmação de fatos; e,

e) Expressivas – as que expressam emoções.

Em suma, a teoria das ações de fala ajuda a definir o tipo de mensagem por meio do conceito de força de ilocução, que restringe a semântica da comunicação. A ação desejada pelo emissor de uma mensagem é, então, claramente definida, e o receptor não tem nenhuma dúvida quanto ao significado da mensagem. Esta restrição facilita o projeto dos agentes de software.

O conteúdo da mensagem em um dado protocolo pode ser ambígua, não ter resposta, requerer a decomposição e ajuda de outros para respondê-la, no entanto, o protocolo deve claramente identificar o tipo de mensagem enviada.

3.2 Entendendo a comunicação entre agentes

Em uma sociedade de agentes, um dos aspectos mais importantes é a capacidade desses agentes cooperarem para alcançar a solução de um problema comum. Deste modo, pode-se entender que a cooperação implica na necessidade de comunicação entre eles (ver Fig. 10). A despeito dos diferentes pontos de vista sobre a definição do que seja um *agente*, a maioria dos autores concorda que a capacidade de interação e a interoperabilidade são as características desejáveis para este tipo de entidade (FININ et al., 1997).

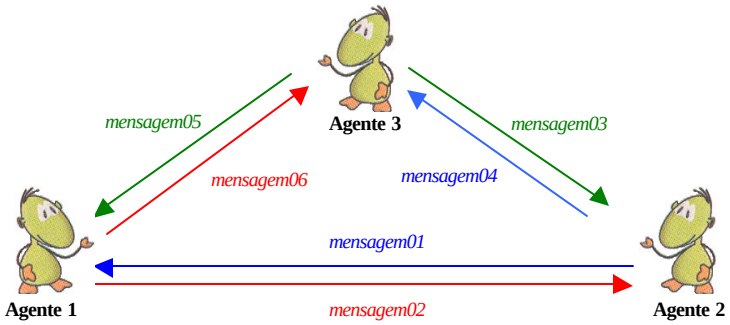

Figura 10 – Cooperação implica comunicação

Mas para que haja interação o conhecimento deve ser compartilhado. Pois, compartilhar conhecimento não significa apenas a comunicação do mesmo, mas também um entendimento mútuo dos participantes envolvidos sobre o significado desse conhecimento, ou seja, não basta apenas ter uma linguagem comum que permita que essas entidades se comuniquem, faz-se necessário que haja também um entendimento correto dos termos utilizados dentro de um

determinado contexto, por exemplo: dois especialistas na área de informática trocam conhecimento e conseguem se comunicar (fazerem-se compreendidos) porque ambos possuem vocabulário suficiente e correto das expressões e termos utilizados durante aquele diálogo, o que não acontece caso um leigo tente se integrar nesse contexto.

Comunicação e entendimento do conhecimento andam juntas na questão de *agentes de software*. Assim, da mesma forma que na linguagem humana existe uma correlação entre símbolos (sintaxe) e o significado dos termos e expressões utilizadas (semântica), de forma que uma possa expressar corretamente o sentido da outra, a LCA deve definir uma sintaxe e uma semântica (FIPA, 2001).

Em um ato de comunicação estão envolvidos os seguintes elementos (ver Fig. 11):

a) **protocolos de interação** – são estratégias de alto nível propostas pelos agentes de software que guiam suas interações com outros agentes;

b) **linguagem de comunicação (LC)** - é o meio pelo qual as atitudes relativas ao conteúdo da troca são comunicadas, sugerindo se esse conteúdo é uma afirmação, um pedido ou alguma forma de consulta; e,

c) **protocolo de transporte** – que é o mecanismo de transporte atual usado tecnicamente para a comunicação, como: TCP, SMTP, FTP e HTTP.

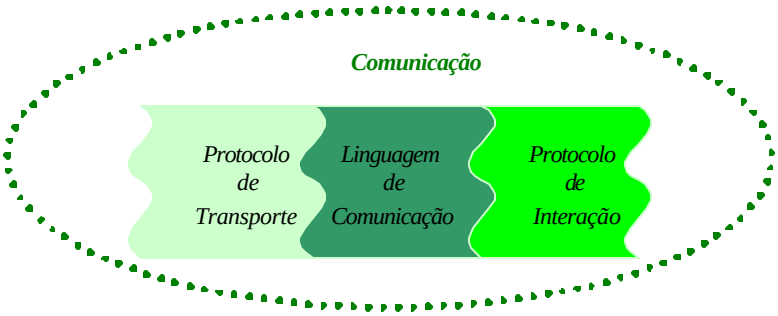

Figura 11 – Componentes da comunicação

Desse modo não se deve confundir linguagem de comunicação com protocolo. Pois quando se utiliza ou menciona-se protocolo no contexto de linguagens de comunicação na verdade pretende-se referir a algum dos seguintes significados:

a) protocolo de transporte (HTTP, SMTP, FTP, etc.);

b) um *framework* de alto nível para interação, assim como negociação, protocolos de teoria de jogos, etc; ou,

c) trocas das primitivas de comunicação (mensagens).

Uma linguagem de comunicação pode usar protocolos do primeiro tipo como mecanismos de transporte, que são usados por protocolos do segundo tipo como um modo de implementá-los e usualmente inclui protocolos do terceiro tipo como parte de sua descrição. Porém, definitivamente, uma linguagem de comunicação não é simplesmente um protocolo em si (FININ et al., 1997).

Para que haja esse entendimento mútuo dois aspectos devem estar presentes:

a) tradução de uma linguagem de representação para outra;

b) compartilhamento do conteúdo semântico da representação do conhecimento, entre diferentes aplicações.

Em resumo, para que os agentes de software possam interagir e operar conjuntamente, de modo efetivo, três requisitos são fundamentais (FININ et al., 1997):

a) existir uma linguagem comum que, em termos lingüísticos, significa que uma sintaxe, uma semântica e uma prática comum devem ser compartilhadas;

b) ter um entendimento comum do conhecimento trocado, ou seja, a todos os envolvidos deve estar claro o significado das expressões que estão sendo trocadas;

c) possuir a habilidade de trocar, independente do conteúdo da mensagem, nos itens 1 e 2.

Deste modo, a interação é vista como uma troca (comunicação) de informações e conhecimento que pode ser mutuamente entendida.

Uma linguagem de comunicação de agentes deve permitir então que os agentes passem a execução de ações para outros agentes, monitore suas execuções, relate o progresso, o sucesso e a falha, recuse alocação de tarefas, confirme o recebimento de mensagens, etc. Isto significa que uma coleção de agentes pode ser necessária para efetuar uma tarefa que freqüentemente inclui pessoas que estão executando algum tipo de trabalho e onde parte desse trabalho foi delegada para os agentes (COHEN et al., 1995; COSTA, 1997).

Para comunicar-se com outros agentes, um agente deve ser capaz de participar de um diálogo (HUHNS et al., 1999). Seu papel no diálogo pode ser ativo ou passivo, ou ambos, admitindo-se que eles podem funcionar como mestres, escravos ou pares, respectivamente. Para participarem de um diálogo, os agentes precisam para isso trocar mensagens.

Essas mensagens, em relação ao fato dos agentes poderem ser ativos ou passivos, podem ser de dois tipos: consultas ou afirmações. Todo agente, ativo ou passivo, deve ser capaz de aceitar informações. No caso mais simples, esta informação é comunicada para um agente externo por meio de uma mensagem de afirmação. Para assumir o papel de ativo ou passivo em um diálogo, o agente deve adicionalmente ter capacidade de responder uma consulta, ou seja, ele deve saber:

a) aceitar uma consulta de uma fonte externa; e,

b) respondê-la por meio de uma afirmação.

Para um agente ser ativo em um diálogo, ele deve saber formular consultas e fazer afirmações. Assim, ele pode potencialmente controlar outros agentes ou subagentes, como bancos de dados ou redes neurais. O Quadro 2, a seguir, resume o tipo de agente de acordo com sua capacidade de comunicação.

Capacidade/Agente	Básico	Passivo	Ativo	Par
Receber afirmações	✔	✔	✔	✔
Receber consultas		✔		✔
Enviar afirmações		✔	✔	✔
Enviar consultas			✔	✔

Quadro 2 – Capacidades dos agentes

Deste modo, pode-se entender a tabela anterior da seguinte forma: um agente Básico somente é capaz de receber informações; o agente Passivo é aquele que recebe e envia afirmações e apenas recebe consultas; o Ativo recebe/envia afirmações além de ser capaz de enviar perguntas/consultas; e, Par é aquele capaz de receber e enviar afirmações e consultas, ou seja, ele possui todas as características do Ativo e do Passivo.

De acordo com as características em relação às capacidades dos agentes mencionadas acima, os agentes artificiais utilizados no MATHNET são classificados segundo suas capacidades conforme o Quadro 3, a seguir.

Agentes Artificiais MATHNET	Básico	Passivo	Ativo	Par
Tutor	-	-	-	✔
Modelagem do Aprendiz	-	-	-	✔
Estrategista	-	-	-	✔
Domínio	-	✔	-	-
Busca	-	-	-	✔

Quadro 3 – Classificação dos Agentes Artificiais MATHNET segundo suas capacidades

Na seção a seguir, destacam-se as características que uma LCA deve possuir.

3.3 Características desejáveis em uma LCA

O valor de uma linguagem de comunicação pode ser medido através de alguns requisitos no momento de sua concepção. Esses requisitos servem como guias para nortear o que se deseja para uma linguagem de comunicação de agente. Tais parâmetros são igualmente relevantes e complementares para a construção de uma boa LCA (ver Fig. 12).

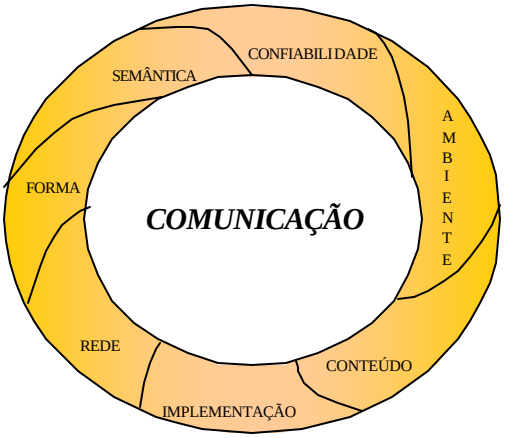

Figura 12 – Características desejáveis de uma LCA

As características são:

a) **Forma** – uma boa linguagem de comunicação deve ser informativa, sintaticamente simples e legível para as pessoas. Além disso, deve ser: concisa, fácil de analisar e gerar, e possuir uma sintática extensível de modo a poder ser integrada a uma grande variedade de sistemas;

b) **Conteúdo** – uma linguagem de comunicação expressa as ações comunicativas (quando uma mensagem pretende executar alguma ação em virtude de ter sido enviada), enquanto que a linguagem de conteúdo expressa fatos sobre o domínio. A linguagem, portanto, deve possuir um conjunto bem definido de ações comunicativas (primitivas). E embora esse conjunto possa ser extendido, ele deve possuir um conjunto mínimo de primitivas que capturam nosso entendimento sobre o que constitui uma ação comunicativa e que seja independente da aplicação (Banco de Dados, sistema Orientado a Objeto, base de conhecimento, etc.), assegurando deste modo à utilização da linguagem por uma variedade de sistemas;

c) **Semântica** – a semântica de uma LC deve exibir as propriedades esperadas de uma semântica de qualquer outra linguagem, sendo fundamentada em uma teoria e não possuindo ambigüidades. Uma descrição semântica provê um modelo de comunicação que seria útil para a performance da modelagem;

d) **Implementação** – deve ser eficiente e rápida, provendo uma boa combinação com a tecnologia de software existente e com uma interface de fácil uso. A linguagem deve favorecer a implementação;

e) **Rede** – pelo fato de alguns conceitos de LC envolverem rede de comunicação ela deve estar atenta às novas tecnologias de rede. Com isso, a linguagem deve suportar todas as conexões básicas (ponto-a-ponto, *muticast* e *broadcast*), conexões síncronas e assíncronas. Deve possuir um rico e suficiente conjunto de primitivas que possa servir como base sob a qual uma linguagem de alto nível e protocolos de interação possam ser construídos, de modo que esses protocolos de alto nível sejam independentes do mecanismo de transporte (TCP/IP, e-mail, HTTP);

f) **Ambiente** – a LC deve prover ferramentas para lidar com ambientes heterogêneos e dinâmicos, já que esse é o tipo de ambiente em que os agentes são requisitados a trabalhar. Também deve suportar: interoperabilidade com outras linguagens e protocolos, descoberta de conhecimento em grandes redes e ser facilmente legada aos sistemas;

g) **Confiabilidade** – uma LC deve dar suporte a uma comunicação confiável e segura, além de trocas seguras e privadas entre os agentes

de modo a garantir o isolamento e a integridade de dados. E ainda, prover um modo de garantir a autenticação dos agentes e suportar razoavelmente mecanismos para identificação e sinalização de erros e alertas (*warnings*).

Existem várias linguagens de comunicação entre agentes como Telescript (INTELLIGENT, 2002; PUC DO RIO GRANDE DO SUL, 2002) que é uma linguagem interpretada que trabalha independentemente de todos os protocolos de transporte e cujo ambiente foi desenvolvido especificamente para a utilização de agentes móveis; KQML (Linguagem de Manipulação e Consulta de Conhecimento ou Knowledge Query and Manipulation Language) e FIPA-ACL (COMTEC, 2002; FIPA, 2002) dentre outras.

Tanto a KQML quanto a FIPA-ACL são linguagens que também podem ser utilizadas na comunicação com os agentes móveis. Agentes móveis são aqueles que podem se transportar para um outro sistema de agentes no qual esteja contido um objeto com o qual o agente móvel precisa interagir (WHITE, 1997).

Na seção a seguir, essas duas linguagens, serão apresentadas com maiores detalhes.

3.4 Tipos de LCA

3.4.1 KQML

Os agentes podem se comunicar se eles têm uma linguagem de representação comum ou usam linguagens diferentes, mas que podem ser

traduzidas de uma para outra e que possam compartilhar um *framework* do conhecimento necessário para interpretar as mensagens que são trocadas. Na prática, a comunicação entre agentes de software envolve conhecer com quem se está conversando e como encontrá-los, bem como, conhecer como iniciar e manter uma troca de mensagens. Essa é a primeira preocupação de KQML. A segunda é a semântica.

A KQML foi desenvolvida dentro do Knowledge Sharing Effort (KSE) patrocinado pelo DARPA, em 1993. Ela é independente do mecanismo de transporte (TCP/IP, SMTP, etc.), independente da linguagem conteúdo que pode ser KIF (KIF, 2002; MASUOKA et al., 1999), SQL, Prolog, etc., e da ontologia (vocabulário) assumida pelo conteúdo.

KQML baseia-se na teoria dos atos de fala e no uso de performativas. Performativas são, em linguagem natural, expressões que substituem aquilo que o emissor diz ou afirma que ele/ela está fazendo algo. Deste modo, uma ação é executada mediante algo que foi dito.

Assim, pode-se definir KQML como sendo uma linguagem e um conjunto de protocolos que suportam agentes de software na identificação, com conexão, e troca de informações com outros agentes, ou então, é uma linguagem e um protocolo de comunicação, de alto nível, por troca de mensagens independentemente da sintaxe do conteúdo e da ontologia aplicável.

A KQML é composta por três camadas conforme a Figura 13, a seguir (FININ, 1997).

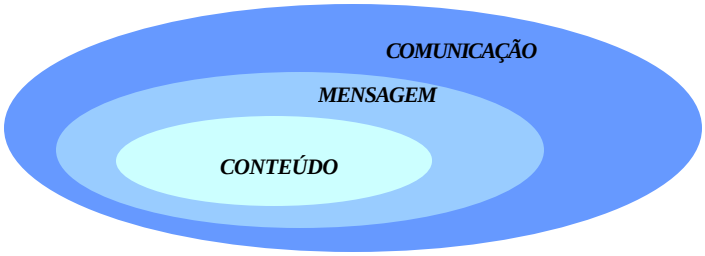

Figura 13 – Camadas da comunicação em KQML

As três camadas são:

a) **Comunicação** - envolve um conjunto de características da mensagem que descrevem parâmetros de comunicação de alto nível, como as que identificam o emissor e receptor, e um identificador único associado com a mensagem;

b) **Mensagem** – é usada para codificar uma mensagem que um agente gostaria de transmitir para outro. Ela é o núcleo da linguagem e determina os tipo de interações que podem existir com um agente "falando" KQML. A primeira função dessa camada é identificar o protocolo a ser usado para entregar a mensagem e o ato de fala ou performativa anexada ao conteúdo. Esse conteúdo, por sua vez, pode ser uma afirmação, uma consulta, um comando ou qualquer outra coisa do conjunto de performativas conhecidas. Essa camada inclui também características opcionais que descrevem a linguagem de conteúdo e algum tipo de descrição do conteúdo. Essas características fazem ser possível para implementações KQML analisar, rotear e propriamente entregar mensagens apesar do conteúdo deles ser inacessível;

c) **Conteúdo** – é a que contém o conteúdo propriamente dito da mensagem. Este conteúdo pode carregar expressões em qualquer tipo de linguagem de representação como *strings* ASCII ou notação binária. Não existe nenhuma restrição na linguagem quanto ao uso de caracteres não ASCII.

A KQML é projetada para ser usada com vários mecanismos (atualmente há implementações que usam TCP/IP, SMTP e-mail, HTTP e CORBA). O agente de KQML pode falar diretamente a outro agente ou pode enviar mensagens a múltiplos agentes do mesmo grupo.

O ambiente operacional para agentes KQML é altamente distribuído, heterogêneo e extremamente dinâmico. Para satisfazer as exigências de tal ambiente, a linguagem KQML provê ferramentas formais que trabalham com outras linguagens (*Common Lisp*, C, etc.) e demais protocolos de rede (FTP, HTTP, SMTP, TCP/IP).

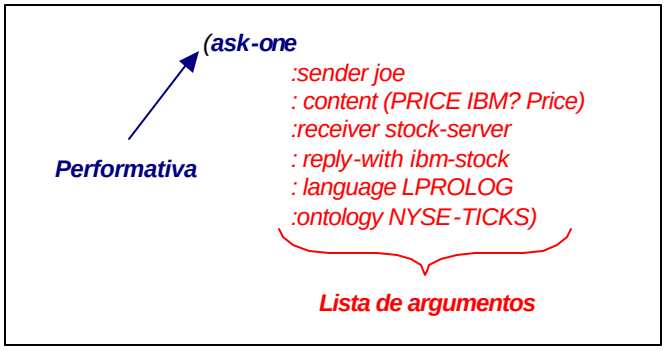

Figura 14 – Elementos de uma mensagem KQML

A sintaxe KQML baseia-se em listas que contém frases aninhadas por parênteses, conforme a Figura 14 mostrada acima.

O primeiro elemento dessa lista é a performativa (*ask-one*) e os demais são argumentos (Ex: *:sender Joe*) dessas performativas expressos em termo de

pares compostos de palavra-chave e valor. A sintaxe da linguagem revela as raízes das implementações iniciais feitas em linguagem de programação Common Lisp (ABEL, 1995), o que a torna totalmente flexível.

A semântica da mensagem é definida pelo conteúdo dos campos, ou seja, pela mensagem em si. É o conteúdo dos campos *linguagem* e *ontologia* que definem a semântica da mensagem, enquanto os símbolos restantes são apenas parâmetros. Alguns autores fizeram inclusive propostas de revisão da KQML inicialmente formulada pelo DARPA (LABROU, et al., 1997b) oferecendo alterações na sintaxe das mensagens e nos parâmetros das performativas reservadas e que, embora tenham sido mínimas, são mudanças significativas no conjunto das performativas reservadas em relação aos seus significados e intenções.

Devido ao fato das mensagens serem uma seqüência linear de caracteres, à semelhança da sintaxe de *Lisp*, isto faz com que as mensagens sejam facilmente lidas, analisadas e convertidas para outros formatos. Logo, a sintaxe é simples e permite a adição de novos parâmetros, quando necessário em futuras revisões da linguagem.

Existe um conjunto mínimo predefinido e não fechado de performativas reservadas que cobre um repertório básico de ações comunicativas e que pode ser extendido adicionando-se performativas e protocolos associados às mesmas, desde que os agentes envolvidos concordem com a interpretação dada a essas performativas adicionais. As performativas constituem a camada de mensagem da linguagem e são interpretadas como atos de fala. As performativas KQML encontram-se identificadas no Quadro 4, a seguir:

Performativas	Identificadores
Informações genéricas	Tell, achieve, cancel, untell, unachieved
Consultas básicas	evaluate, ask-if, ask-in, ask-one, ask-all
Múltiplas respostas para consultas	stream-in, stream-all
Resposta	Reply, sorry
Geradoras	standby, ready, next, rest, dicard, generator
Definição de capacidade e notificação	Advertise, subscribe, monitor, import, export
Rede	register, unregister, forward, broadcast, route, transport-address

Quadro 4 – Algumas performativas de KQML

Algumas dessas performativas de comunicação são orientadas a protocolo (por exemplo, *subscribe*) e outras estão relacionadas a aspectos de práticas não protocoladas (como *advertise, recruit, etc.*). Vários protocolos de comunicação básicos são suportados em KQML.

Outro elemento importante em KQML é a existência de uma classe especial de agentes chamada facilitadores de comunicação (*subscribe, broker, recruit* e *recommend).* Estes são na verdade, agentes responsáveis por executar vários serviços usuais de comunicação, manter registro do serviço de nomes, transferir mensagens para os serviços chamados, rotear mensagens baseado no conteúdo, prover serviços de tradução e mediação, e prover o “encontro” entre fornecedores e clientes da informação (ver Fig. 15). Sua principal função é conhecer o endereço e o nome dos agentes da sociedade, passando esta informação aos demais de modo que eles possam trocar mensagens usando apenas o nome do agente como referência. Todos os agentes têm que saber o endereço físico do facilitador para se comunicar com outros agentes (FININ et al., 1994).

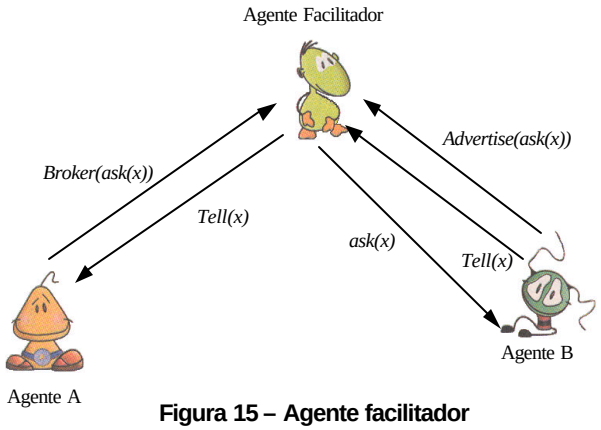

Figura 15 – Agente facilitador

Na Fig. 15, mostrada acima, o Agente **A**, pede ao Agente Facilitador para encontrar um agente que possa processar uma performativa *ask(x)*. O agente **B**, independentemente, informa ao Facilitador que ele está querendo aceitar performativas do tipo *ask(x)*. Desta forma, o agente Facilitador tendo recebido ambas mensagens, envia ao Agente **B** a consulta, obtém a resposta e a encaminha para o Agente **A**

O problema de como os agentes encontram os facilitadores não é o objetivo desse trabalho, mas existe uma variedade de possíveis soluções (GENESERETH et al., 1994). Algumas aplicações baseadas em KQML utilizam técnicas simples para isto, dependendo do projeto. No projeto PACT (FININ et al., 1994 apud CUTKOSKY, 1993, p. 28-38) todos os agentes usam um facilitador comum, que é central, responsável por localizar os demais agentes participantes, "registrando-os" no momento em que são iniciados. Já no ARPI (BURSTEIN, 1994), quando cada agente é iniciado existe um módulo de roteamento que anuncia este fato ao agente facilitador local, registrando-o em um banco de dados local, deste modo a localização e o estabelecimento de contato com o facilitador local é uma das funções de uma API construída para este ambiente.

A existência desses facilitadores pode prover os meios para descoberta de conhecimento em redes grandes, especialmente se os facilitadores podem cooperar com outras aplicações de descoberta de conhecimento disponíveis na Web (WEB, 2002).

A semântica KQML é ainda um assunto em aberto, pois no momento existem apenas descrições de linguagem natural do significado das performativas e seus usos (protocolos).

A eficiência da comunicação KQML tem sido pesquisada, existindo uma pressão para que haja uma redução dos custos de comunicação através da redução do tamanho das mensagens e também pela eliminação de uma fração substancial de *strings* duplicadas (FININ et al., 1997).

A conversação entre agentes que utilizam essa linguagem pode ser feita através da comunicação direta com outros agentes (endereçando-os pelo nome simbólico), pelo *broadcast* de suas mensagens ou através da solicitação de serviços aos agentes parceiros ou facilitadores para entregar uma mensagem utilizando performativas apropriadas. KQML permite interações síncronas (o agente cliente espera pela resposta) e assíncronas (o agente cliente continua o seu raciocínio ou ação que só é interrompido mais tarde, quando receber a resposta).

Essa linguagem tem demonstrado ser altamente efetiva na integração de diversas ferramentas e sistemas, permitindo novas ferramentas de interação e suportando uma infraestrutura de comunicação de alto nível, reduzindo deste modo o custo de integração e disponibilizando uma comunicação flexível através de múltiplos sistemas de rede.

Têm-se utilizado KQML em diversos tipos de aplicações, como bancos de dados distribuídos e/ou heterogêneos, jogos, experimentos de integração de tecnologia da ARPA (Agência de Projetos de Pesquisa Avançada dos Estados Unidos), etc.

A KQML, embora possua desvantagens, é largamente utilizada pela comunidade que trabalha com agentes. Dentre as desvantagens pode-se citar:

a) Apesar de atender as principais características de uma linguagem de comunicação de agentes, KQML ainda não tratou adequadamente as questões de segurança e autenticação de agentes em mensagens KQML que devem ser melhoradas. Mesmo se novas performativas ou parâmetros de mensagem podem ser introduzidos permitindo a encriptação do conteúdo ou de toda a mensagem KQML, melhorando desta forma a segurança;

b) A falta de uma concordância semântica formal aceita universalmente, já que sua semântica é informalmente definida em linguagem natural, o que acaba abrindo diversas interpretações por parte de cada desenvolvedor que utiliza a linguagem (O'BRIEN et al., 1998). E, embora, tenha sido definida uma semântica formal para KQML (LABROU, 1996), baseada em pré-condições, pós-condições e condições para completar cada perfomativa, o que não está claro é se isto tem sido largamente adotada pelos usuários (FININ et al., 1997).

Existem algumas ferramentas para a construção de agentes que utilizam a KQML como linguagem de comunicação, algumas delas são AgentBuilder

(AGENTBUILDER, 2001), Intelligent Agent Factory (BITS AND PIXELS, 2002) e JATLite (JATLITE, 2002), etc.

3.4.2 FIPA-ACL

A Fundação de Agentes Físicos Inteligentes (FIPA) é uma associação sem fins lucrativos cujo propósito é promover o sucesso de aplicações, serviços e equipamentos que utilizam a tecnologia de agentes. Seu objetivo é fazer e disponibilizar especificações com o intuito de maximizar a interoperabilidade entre sistemas de agentes, ou seja, a FIPA é uma organização para especificar padrões para agentes de software.

FIPA-ACL é uma linguagem padrão da FIPA de comunicação entre agentes. É uma linguagem de alto nível e baseada na teoria dos atos da fala. Originou-se da combinação da afinidade e praticidade que os usuários tinham com KQML e a formalização da ARCOL (BREITER et al., 1996), que foi mais expressiva e mais formalmente definida do que KQML e que possuía um pequeno conjunto de primitivas.

Devido a sua origem de KQML, a FIPA-ACL possui a mesma sintaxe para suas mensagens, mas a semântica é diferente.

FIPA-ACL especifica a comunicação entre agentes e possui uma semântica formal associada a ela composta por 5 níveis (ver Fig. 16):

a) Protocolo – define as regras sociais para a estruturação do diálogo entre os agentes;

b) Ato Comunicativo (CA) – define o tipo de comunicação que está sendo executado. Ex: *request*, *confirm*, etc.;

c) Mensagem – define a meta-informação sobre a mensagem do agente, incluindo informações do tipo: emissor, receptor, contexto, etc.;

d) Linguagem de conteúdo – define a gramática e a semântica associada para expressar o conteúdo de uma mensagem. Ex: PROLOG;

e) Ontologia – define o vocabulário e o significado dos termos e conceitos usados no conteúdo da expressão.

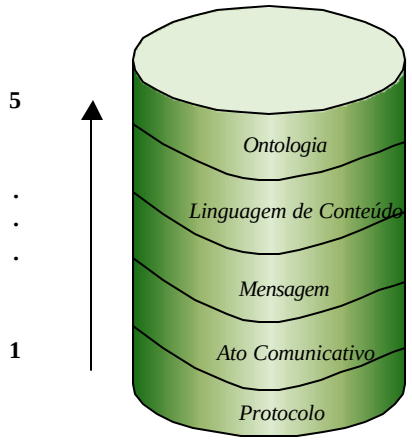

Figura 16 – Comunicação FIPA-ACL em níveis

O nível 1, o Protocolo é que fornece a estrutura para o diálogo entre os agentes, enquanto que os níveis de 2 a 5 (Ato Comunicativo e Mensagem, respectivamente), estão contidos em cada mensagem FIPA-ACL.

E do mesmo modo que a mensagem KQML, a mensagem FIPA-ACL também pode ser composta por *n* expressões aninhadas (ver Fig. 17).

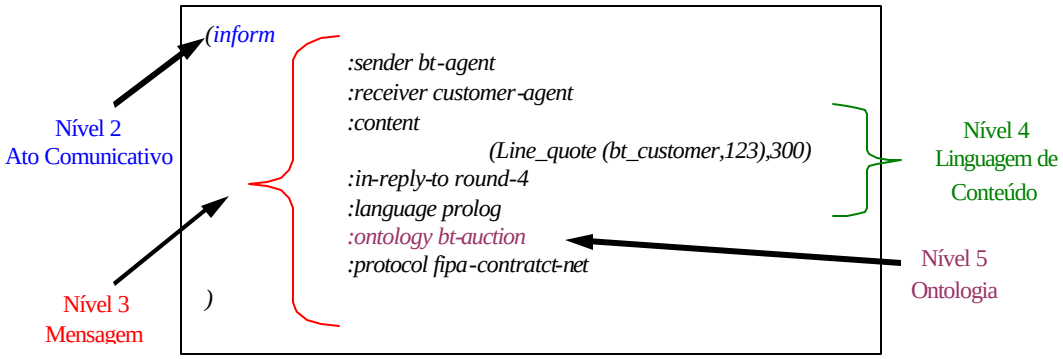

Figura 17 - Estrutura da mensagem FIPA-ACL

Assim como na KQML, os CA's em FIPA-ACL encontram-se também classificados segundo a categoria a que pertencem, conforme pode ser visto em Anexo A. Deste modo, essas categorias agrupam conjuntos de CA's que tem funções em comum, mas que se diferenciam por suas características intrínsecas.

Além disso, a FIPA-ACL (FIPA, 2001; FIPA, 2002) define um conjunto padrão de CA's que podem ser primitivos ou compostos. Os primitivos são aqueles que representam ações atômicas (possui apenas uma ação) e podem ser combinados dentro de expressões mais complexas, que são os CA's compostos (ver Quadro 5).

KQML utiliza o termo perfomativa para se referir às primitivas de comunicação. Em FIPA-ACL, as primitivas de comunicação são chamadas de ações comunicativas. Apesar dos nomes diferentes, as performativas e ações comunicativas são o mesmo tipo de entidade. Uma breve descrição sobre o significado de cada um desses CA's encontra-se descrito conforme a tabela que se encontra no Apêndice B (FIPA, 2001).

ATO COMUNICATIVO	COMPOSTO	PRIMITIVO
ACCEPT-PROPOSAL	X	
AGREE	X	
CANCEL	X	
CFP	X	
CONFIRM		X
DISCONFIRM		X
FAILURE	X	
INFORM		X
INFORM-IF (macro ato)	X	
INFORM-REF (macro ato)	X	
NOT-UNDERSTOOD	X	
PROPOSED	X	
QUERY -IF	X	
QUERY -REF	X	
REFUSE	X	
REJECT-PROPOSAL	X	
REQUEST		X
REQUEST-WHEN	X	
REQUEST-WHENEVER	X	
SUBSCRIBE	X	

Quadro 5 – Atos Comunicativos primitivos e compostos

Além dos CA's, a documentação da FIPA definiu uma quantidade de Protocolos de Interação de Agentes padrão que são responsáveis por definir a sequência de mensagens que representa uma conversação ou diálogo completo entre agentes. A Figura 18, a seguir, ilustra o protocolo FIPA-Request (O'BRIEN et al., 1998) que consiste de uma troca ordenada de atos comunicativos entre dois agentes iniciados por um ato comunicativo *request*. O agente emissor do *request* pode antecipadamente saber que pode ter como resposta um *not-understood*, *refuse* ou *agree*.

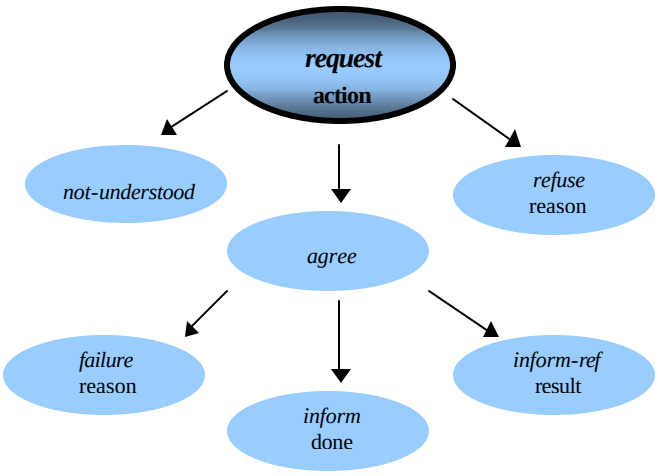

Figura 18 – Protocolo FIPA-Request

Entretanto, essa especificação feita pela FIPA não restringe os desenvolvedores a apenas esses protocolos, podendo eles mesmos adotar seus próprios protocolos ou então seguir uma estrutura de conversação que pode aparecer durante a interação entre agentes. Deste modo, se um agente utiliza o nome de um protocolo padrão do FIPA, este agente deve estar de acordo com o protocolo definido na documentação do FIPA.

FIPA-ACL foi escolhida para definir os protocolos de comunicação do Agente Tutor, com os demais agentes da sociedade, por ter as características necessárias a uma LCA, citadas anteriormente, e por ser uma padronização adotada pela FIPA para a comunicação entre agentes de software.

Dentre as ferramentas de construção de agentes que utilizam a FIPA-ACL, pode-se citar a Comtec Agent Platform (COMTEC, 2002) e ZEUS (ver Apêndice F), esta última foi escolhida para a construção dos agentes deste trabalho.

3.4.3 Semelhanças e diferenças básicas ente KQML e ACL

KQML e FIPA ACL são quase idênticas com respeito aos conceitos básicos e princípios que observam (LABROU et al., 1999). Embora a FIPA-ACL adote uma sintaxe muito semelhante de KQML, incluindo a forma como as mensagens e seus parâmetros são apresentados, existem algumas diferenças em termo da nomenclatura das mensagens e parâmetros, ou seja, elas possuem palavras-chaves reservadas diferentes para representar as mensagens e parâmetros.

Basicamente, as duas diferem nas suas respectivas apresentações semânticas, pois a KQML quando originalmente criada como parte do consórcio KSE, descreveu informalmente a semântica de suas performativas através de descrições em linguagem natural. Pesquisas posteriores têm sido dirigidas para a necessidade de se formular uma semântica mais precisa (LABROU, 1996), apesar de não deixar claro se a semântica proposta tem sido universalmente adotada. Enquanto isso, a ACL foi inicialmente projetada para ter uma semântica formalmente definida.

Por causa dessa diferença, torna-se impossível obter um mapeamento exato entre as performativas do KQML e das primitivas da FIPA ACL, entretanto no Apêndice C, é apresentado um possível mapeamento de algumas performativas/CA's (FIPA, 2001). Por outro lado, essas diferenças podem ser imperceptíveis para programadores que não utilizam a arquitetura de Banco de Dados Inteligente (BDI). BDI é uma sub-área de pesquisa envolvendo agentes cognitivos modelados através de estados mentais, detendo-se nos agentes que utilizam a tríade de estados mentais compostos por: *belief* - crença, *desire* - desejo e *intention* - intenção em seus agentes (ZAMBERLAM et al., 2001).

Uma outra diferença entre as duas LCAs é o tratamento de primitivas de anúncio e definição de capacidades e de registro. Estas primitivas são importantes para a utilização do modelo de três camadas (ver seção 3.4.1). Em KQML, tarefas, tais como registro, atualização e anúncio de capacidades, são tratadas como objetos de primeira classe e são chamadas primitivas de *facilitação*. A FIPA-ACL não provê, ainda, primitivas de facilitação.

Alguns usuários, acostumados a utilizar KQML, querem que sejam incluídas as primitivas de facilitação, tais como *broker*, *recommend* e *recruit*, em FIPA-ACL. Estes pedidos mostram que uma LCA precisa ser uma cuidadosa mistura de teoria e prática.

E embora existam estas diferenças entre essas duas linguagens, KQML e FIPA-ACL, vistas por alguns como vantagens ou desvantagens, ambas continuam sendo as LCAs mais utilizadas quando se trata da comunicação entre agentes.

3.4.4 Padronização de LCA

O conceito de padrão de uma linguagem de comunicação para agentes de software que seja baseada na teoria dos Atos da Fala tem encontrado grande apoio tanto entre os interessados em teorias de comunicação de agentes quanto entre os que visam construir sistemas de agentes na prática (LABROU et al., 1997c). Muitos acreditam que o desenvolvimento de uma efetiva e rica LCA padrão, seja uma das chaves para o paradigma de agentes.

O problema com a definição de um padrão de LCA é que muitos dos SMAs que foram construídos até hoje, utilizam linguagens personalizadas para seus

problemas e seus ambientes de aplicação. Essas linguagens têm primitivas, cujos significados são restritos aos limites de seus respectivos SMA. E mesmo quando essas primitivas de comunicação são definidas utilizando-se uma sem ântica formal, tais formalismos raramente são úteis para outros sistemas que tenham seus próprios contextos.

Essa tarefa então, de definir uma LCA padrão tem uma ampla variedade de complicadas decisões de natureza pragmática, porém, nenhuma é tão difícil quanto a definição clara de uma semântica baseada em ações de comunicação.

Muitas têm sido as tentativas de se definir a semântica das ações de comunicações (COHEN et al., 1995; SINGH, 1998). Estas tentativas, em geral, apresentam os seguintes problemas (LABROU et al., 1997a):

a) Estão presas a alguma teoria de agentes específica que pode não ser aplicável a todos os agentes que precisam utilizar a LCA; e,

b) Introduzem formalismos complexos que não permitem a implementação dos sistemas de agentes.

3.5 Considerações finais

Para realizar a cooperação assume-se que os agentes devem estar habilitados para se comunicarem. Para uma comunicação significativa considera-se o uso de mensagens. Um conjunto pré-definido de mensagens padrão é proposto pelas linguagens de comunicação KQML e FIPA-ACL e foram aqui apresentadas.

Nesse capítulo, também foram apresentados os conceitos básicos que permitem entender como a comunicação entre agentes ocorre, utilizando-se para isto uma LCA (composta de sintaxe e semântica) e que permite o entendimento correto dos termos utilizados durante a troca de mensagens entre os agentes.

Dentre as LCAs apresentadas, enfatizou-se a FIPA-ACL, com a estrutura de suas mensagens, os CA's que a compõe e semântica, bem como o fato dela ser a LCA utilizada neste trabalho, por ser uma padronização adotada pela FIPA, para comunicação de agentes.

No próximo capítulo será detalhado o Agente Tutor mostrando suas principais características, funções e Casos de Uso identificados no sistema MATHNET.

4 AGENTE TUTOR

Neste capítulo apresenta-se como o Tutor é definido em sistemas multiagentes. Além disso, destaca-se o papel (características e responsabilidades) do Agente Tutor no MATHNET, suas interações com os demais agentes dessa SA e os principais Casos de Uso do MATHNET nos quais o Tutor participa em relação às Fases/Atividades Pedagógicas apresentadas no Capítulo 2.

4.1 Abordagens sobre o Agente Tutor

A necessidade de agentes independentes que possam realizar tarefas em grupo, através da interação e colaboração de outros agentes, e a utilização de rede de computadores com o objetivo de prover um ambiente de ensino-aprendizagem que possibilite aos alunos aprender interagindo e criando uma compreensão sobre as informações recebidas, tem sido objeto de várias pesquisas e têm gerado um grande número de modelos utilizando arquiteturas SMA.

Existem várias propostas para arquiteturas SMA de sistemas de ensino-aprendizagem e em cada uma delas o Tutor é visto e definido de várias formas possuindo características próprias/particulares e que se adeqüa ao ambiente no qual está sendo modelado. Todas elas, entretanto, têm como objetivo comum construir um ambiente de ensino-aprendizagem que proporcione o desenvolvimento do aprendizado de forma significativa e duradoura através principalmente da interação e cooperação entre os agentes existentes nessas sociedades.

4.2 Papel do Agente Tutor no MATHNET

A teoria SMA pode ser integrada adequadamente no desenvolvimento de sistemas tutores inteligentes cooperativos como o MATHNET, pois ela pode oferecer modelos adaptados para modelagem das interações entre os diferentes integrantes do sistema, enfatizando a aprendizagem cooperativa onde os estudantes aprendem colaborando com todo o ambiente (ver Fig. 19).

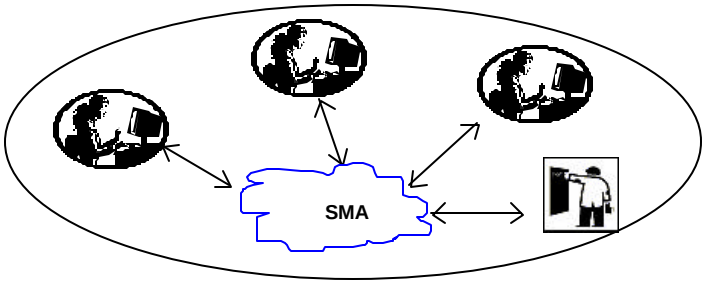

Figura 19 - Aprendizagem Cooperativa em um SMA

O MATHNET adota uma arquitetura Multiagentes, conforme mostrado na seção 2.3. Nas próximas seções, serão apresentados maiores detalhes do funcionamento desta sociedade destacando o papel fundamental do Agente Tutor no ambiente MATHNET.

4.2.1 Agente Tutor no MATHNET

A definição e implementação do Agente Tutor, bem como suas interações com os demais agentes, é o foco principal deste trabalho, do ponto de vista da comunicação de agentes.

Deste modo, para delimitar os principais Casos de Uso do Tutor foi necessário descrever as responsabilidades inerentes a ele e o papel que ele exerce dentro da sociedade de agentes MATHNET.

Destacam-se as seguintes responsabilidades (LABIDI et al., 2000a; NUNES, 2001):

a) Ajudar na formação dos grupos (a partir das informações fornecidas pelo Agente de Modelagem do Aprendiz);

b) Apresentar o conhecimento para os estudantes nas diferentes áreas cooperativas. Decidindo e coordenando o que será apresentado ao aprendiz, quando e como;

c) Controlar o fluxo de interação dos grupos de estudantes com o sistema durante o progresso do curso;

d) Auxiliar o professor na tarefa de manutenção das atividades que existem no sistema;

e) Propor a elaboração de listas de atividades que podem ser trabalhadas pelo aprendiz e mantidas pelo professor;

f) Disponibilizar ao aprendiz suas listas de atividades que devem ser executadas durante a fase de Aplicação de Conhecimento.

g) Solicitar o fornecimento de dicas e ajuda para o aprendiz durante o processo de ensino-aprendizagem;

h) Auxiliar a Ferramenta de Autoria na definição do Modelo de Domínio;

i) Aplicar as estratégias pedagógicas;

j) Observar o comportamento dos grupos para proceder avaliação;

k) Observar o comportamento individual para proceder avaliação.

O Agente Tutor interage com os Agentes de Modelagem de Aprendiz, Estrategista, Aprendiz, Domínio e com o Professor.

Pode-se então, afirmar que o Tutor tem um papel importante dentro do ambiente MATHNET, pois ele auxilia desde o momento da preparação dos grupos, na disponibilização do conhecimento, na manutenção das atividades, na interação com o Aprendiz e o Professor, na aplicação de estratégias pedagógicas e na avaliação de um modo geral.

O Tutor exerce um papel central dentro do MATHNET já que durante todo o processo de ensino-aprendizagem ele encontra-se presente. Sendo participante de todas as fases pedagógicas através das interações com os demais agentes da sociedade, como um elo entre os agentes, ora facilitando as tarefas, ora sendo parte atuante na realização destas. Isto pode ser visto melhor através das interações descritas na seção 4. 2.2, da delimitação dos Casos de Uso apresentados na seção 4.5 pelos diagramas feitos em UML apresentados no Capítulo 5.

4.2.2 Interações do Agente Tutor com os demais agentes no MATHNET

Em um SMA existem vários tipos de interações entre os agentes. No MATHNET, os agentes humanos utilizam os recursos do sistema para interagir entre

si. Por sua vez, os agentes artificiais interagem entre si e com os agentes humanos, com o objetivo de dar suporte computacional ao ensino-aprendizagem.

Assim que, no MATHNET, existem desde as interações *Intragrupos* (que ocorrem dentro de uma área cooperativa entre os alunos do grupo quando cooperam entre si para alcançar objetivos comuns), *Intergrupo* (que são as que ocorrem entre os grupos e através das quais os alunos têm oportunidade de conhecer outros colegas, trocar experiências e conhecimentos, etc.), *Agentes artificiais & professor* (refletem todas as interações nas quais o professor interage com todos os agentes artificiais do MATHNET) e *Professor & grupos/alunos* (são aquelas nas quais o Professor interage com os alunos/grupos propiciando um melhor andamento da aprendizagem, principalmente no caso de dúvidas não resolvidas pelos outros alunos, grupos ou agentes artificiais podendo enviar e receber mensagens desses alunos) (LABIDI et al., 2000b):

Embora existam outras interações além das que foram citadas acima, as interações do Agente Tutor com os demais agentes humanos e artificiais que compõe a sociedade de agentes MATHNET, constituem o foco deste trabalho e são aqui destacados.

a) **Agente Tutor - alunos/grupos**: os alunos interagem com o Agente Tutor durante todo o decorrer do *curso*. Por exemplo, o Agente Tutor poderá ter a iniciativa de fornecer aos aprendizes dicas, sugerir discussões, etc. Os aprendizes podem pedir ao Tutor que repita parte específica de uma apresentação, pedir/aceitar ajuda, expor dúvida, aceitar/rejeitar sugestões de interações inter e intragrupos, etc.;

b) **Agente Tutor - Professor**: o Tutor pode interagir com o Professor no momento da formação dos grupos, ou seja, nesse caso, o próprio Tutor pode sugerir, independentemente do Professor, uma formação inicial para os grupos, que pode ser ou não aceita por ele;

c) **Agente Tutor - Agente de Modelagem do Aprendiz**: as interações entre esses agentes ocorrem quando o Tutor necessita obter o perfil do Aprendiz. A partir deste perfil o Tutor consegue identificar os possíveis conteúdos a serem assistidos e a partir de que ponto, bem como as demais tarefas (discussões, problemas, etc.) que o aluno pode realizar durante aquela sessão de aprendizagem;

d) **Agente Tutor - Agentes de Domínio**: o Tutor interage com os Agentes de Domínio para poder disponibilizar o conteúdo (conhecimento) armazenado neles aos alunos do sistema. Desta forma, após o aluno iniciar uma sessão de aprendizagem, ele pode aprender um assunto a partir do momento em que o Agente Tutor disponibilizar esse conteúdo;

e) **Agente Tutor - Agente Estrategista**: ocorrem interações entre o Tutor e o Agente Estrategista com o objetivo do Tutor aplicar a melhor estratégia pedagógica selecionada pelo Agente Estrategista (utilizando o perfil do Agente de Modelagem do Aprendiz para definir sua escolha).

Descrever as interações é basicamente, ou mais especificamente, descrever a cooperação existente entre os agentes. Essa descrição é que tornar possível Identificar cooperações e parceiros de cooperação, ou seja, onde o objetivo

de um agente é satisfeito pela cooperação com outros agentes, neste caso pode então ser necessário compartilhar recursos, sincronizar ações ou coordenar o comportamento.

4.3 Agente Tutor nos SMAs

A efeito comparativo com outros sistemas, usando o conceito de Tutor artificial, destaca-se o sistema MATS (SOLOMOS et al., 1999) e o de Kirchhof & Fontes (KIRCHHOF et al., 1999).

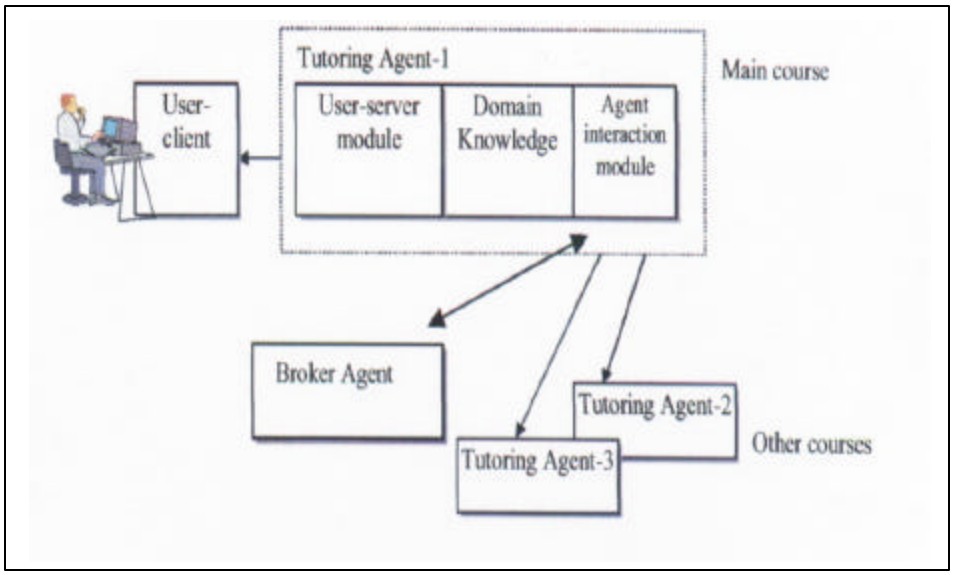

Figura 20 – Arquitetura MATS

No MATS é um protótipo que modela “um estudante – vários professores” em uma situação de aprendizagem, sendo que ele é um sistema tutorial multiagente distribuído (ver Fig. 20) (SOLOMOS et al., 1999).

Cada agente MATS representa um tutor capaz de ensinar um assunto distinto e todos os tutores MATS são também capazes de colaborar com uns com os outros para solucionar dificuldades de aprendizagem que os estudantes possam ter

(SOLOMOS et al., 1999). Esses Agentes Tutores cobrem diferentes assuntos, que são definidos como áreas de um especialista que eles podem ensinar e se comunicam com outros agentes através de uma Linguagem de Cooperação de Agentes Educacional (EACL), baseada em KQML.

A arquitetura definida por Kirchhof e Fontes (ver Fig. 21) é multiagentes e tem como objetivo apoiar as atividades de ensino-aprendizagem como as que são utilizadas em STIs. Nesta arquitetura, existe um certo número de agentes cooperativos, coordenando o comportamento inteligente do conjunto de agentes existentes. Deste modo que, existe uma sociedade de agentes autônomos, onde cada um possui uma função bem definida, e com bases de conhecimentos de domínio armazenadas em um servido WWW (KIRCHHOF et al., 1999).

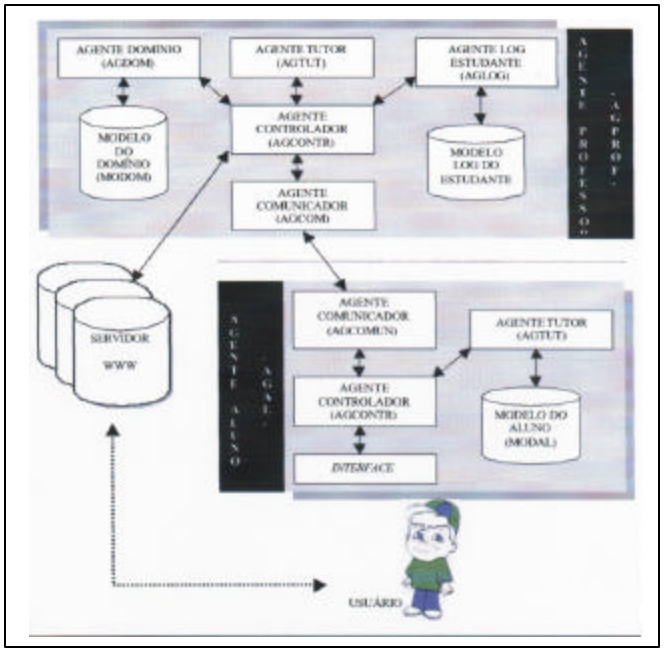

Figura 21 – Interação entre os agentes da sociedade

Além disso, esta arquitetura permite que o aluno seja monitorado durante toda a interação com o sistema, que o Tutor planeje e tome as decisões

pedagógicas necessárias adaptando as estratégias de acordo com o tutoramento do aluno, etc. A comunicação entre esses agentes é feita através de trocas de mensagens.

No Quadro 6, pode-se observar um quadro comparativo entre essas SMAs. Embora, o Agente Tutor tenha sido visto sobre diferentes aspectos em cada uma delas, não se pode afirmar qual a mais correta. Pois, foram baseadas nas características próprias e nos objetivos propostos de cada arquitetura, de forma a atender às necessidades do ambiente. Isto não diminui sua relevância do Tutor dentro de cada SMA.

	AGENTE TUTOR		
Características	*MATS*	*Kirchhof e Fontes*	*MATHNET*
É constituído por uma sociedade de agentes tutores.	✔		
Armazena/Representa uma unidade de aprendizagem (conteúdo/aula/subdomínio)	✔		
Aplica a estratégia pedagógica			✔
Responsável por escolher/tomar decisões pedagógicas.		✔	
Apresenta uma unidade de aprendizagem (conteúdo/aula/subdomínio).	✔		✔
Pode propor uma formação inicial para dividir em grupos os alunos.			✔
Monitora/acompanha as ações de cada aluno.	✔	✔	
Elo de ligação/comunicação entre os agentes artificiais e humanos do sistema.			✔

Quadro 6 – O Tutor nos diferentes SMAs

Pode-se observar através da tabela acima, que o grau de complexidade de cada Agente Tutor varia em função da concepção do sistema no qual se encontra inserido. Desta forma, algumas vezes, o Agente Tutor é estruturado de maneira que ele centraliza basicamente todas as ações para realizar o processo de ensino-aprendizagem, desde o planejamento da estratégia a ser aplicada, passando pelo

armazenamento do conteúdo a ser assistido pelo Aprendiz e o seu tutoramento ao longo do “curso”.

Na seção a seguir, com base nas características destacadas acima, e nas Fases/Atividades Pedagógicas descritas na seção 2.2, serão apresentados os principais casos de Uso do MATHNET em relação ao Tutor.

4.4 Modelagem das interações no MATHNET

A modelagem permite a compreensão de um sistema, entretanto nenhum modelo é inteiramente suficiente, pois, é a interconexão entre vários modelos que torna possível entender qualquer aspecto global, ainda que seja o sistema mais trivial (LARMAN, 2000). Além disso, os modelos podem ser expressos através de diagramas, que facilitam a comunicação, de forma correta e sem ambigüidades, entre os desenvolvedores e programadores, permitindo que ambos os lados possam ser entendidos (LARMAN, 2000).

Na modelagem do MATHNET, está baseada no processo unificado UML, devido por ser extensível e adaptável a qualquer sistema, por poder realizar o mapeamento direto dos modelos para as linguagens de programação Orientada a Objeto (O.O.) e vice-versa, além de ser um padrão para modelagem de sistemas O.O e baseado em Agentes.

Os Casos de Uso representam os requisitos funcionais do sistema como um todo e envolvem a interação dos atores com o sistema, não sendo tão gerais, mas também não são muito específicos (PAGE-JONES, 2001). Deste modo, os comportamentos que um sistema apresenta são modelados como casos de uso

durante o processo de captação e análise de requisitos (BOOCH et al., 2000), pois é através da utilização dos Casos de Uso e de diagramas que os representam e de suas respectivas especificações, que os requisitos do usuário para o sistema serão identificados.

O processo de ensino-aprendizagem no MATHNET é dividido seis grandes pacotes (ver Fig. 22). Esses pacotes correspondem às características das Fases/Atividades Pedagógicas descritas anteriormente na seção 2.2.

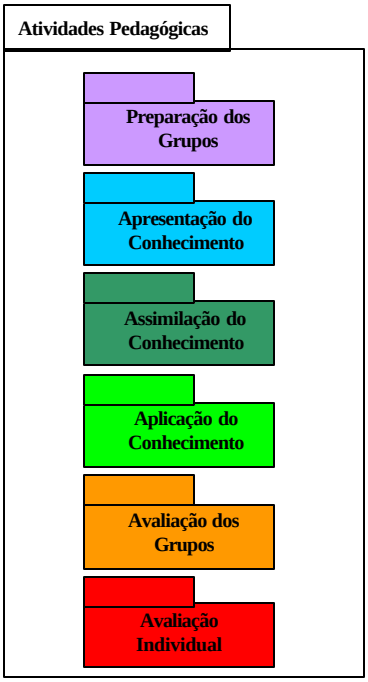

Figura 22 – Pacotes do processo de ensino-aprendizagem

A fase de análise é baseada nas Atividades Pedagógicas e nas características do Agente Tutor, para poder identificar os principais Casos de Uso do MATHNET. Os que foram aqui identificados contemplam todo o processo de ensino-aprendizagem, passando por todas as atividades pedagógicas, permitindo que o aprendiz apreenda um conhecimento e que o professor acompanhe e coordene as atividades desenvolvidas ao longo de todo o processo. Os Casos de Uso encontram-

se descritos no Quadro 7 a seguir, sendo classificados conforme às Atividades Pedagógicas.

ATIVIDADES PEDAGÓGICAS	CASOS DE USO
Preparação de Grupos	Organizar Grupo Estabelecer Critério Estabelecer Aceitação Estabelecer Coesão Iniciar Sessão
Apresentação do Conhecimento	Assistir Apresentação Tirar Dúvida Monitorar
Assimilação do Conhecimento	Escolher dinâmica de grupo Monitorar Tirar dúvida Discutir assunto Orientar Assimilação
Aplicação do Conhecimento	Escolher Dinâmica de Grupo Monitorar Orientar Aplicação Elaborar Lista de Atividades Manter Base de Atividades Realizar Lista de Atividades Realizar Atividade Resolver Problema Tirar Dúvida Consultar Banco de Dúvidas Enviar Mensagem para o Professor Recuperar Informações
Avaliação do Grupo	Avaliar Grupo
Avaliação Individual	Avaliar Aluno

Quadro 7 – Casos de Uso do Tutor em relação às Atividades Pedagógicas

A seguir descreve-se, sucintamente, os principais casos de uso identificados na tabela acima:

a) **Organizar Grupo** - através deste caso de uso, o professor obterá do sistema a divisão de uma turma de aprendizes em grupos, a partir de critérios de formação de grupo estabelecidos por ele;

b) **Estabelecer Critério** - é um caso de uso, abstrato sendo extendido em dois outros: Estabelecer Aceitação e Estabelecer Coesão, que são os critérios definidos pelo professor e utilizados pelo Tutor para a formação de grupos;

c) **Estabelecer Aceitação** - neste caso de uso, o professor estabelece seu Critério de Aceitação para os grupos de alunos a serem formados pelo sistema;

d) **Estabelecer Coesão** - através deste caso de uso, o professor estabelece como o tutor deve calcular a coesão de um grupo a partir de um sociograma da turma;

e) **Iniciar Sessão** - disponibiliza ao aprendiz uma sessão de aprendizagem. Nela, o aprendiz estará apto a assistir apresentação de qualquer conteúdo disponibilizado pelo sistema, realizar atividades dentro dessa sessão de aprendizagem, obter o esclarecimento de dúvidas;

f) **Assistir Apresentação** – é disponibilizado ao aprendiz o conteúdo (unidade de conhecimento) selecionado por ele. Começando deste modo a sessão de aprendizagem pela apresentação do conhecimento;

g) **Tirar Dúvida** - descreve como um aprendiz deve proceder para obter do sistema a explicação para uma dúvida. Este caso de uso pode ser iniciado nas fases de Apresentação, Assimilação e Aplicação do Conhecimento. Ele é composto por três outros casos de uso: "Consultar Banco de Dúvidas", "Enviar mensagem para o Professor" e "Recuperar Informações". Esse último já se encontra implementado e foi denominado Serviço de Busca (NUNES, 2002);

h) **Monitorar** - através deste caso de uso o professor deve ser capaz de conhecer e acompanhar as ações do aprendiz ao longo de uma

sessão de aprendizagem (Apresentação/Assimilação/Aplicação de Conhecimento);

i) **Escolher Dinâmica de Grupo** – neste caso de uso o professor deve ser capaz de escolher a dinâmica que será aplicada ao(s) grupo(s) do sistema nas fases de Assimilação e Aplicação do Conhecimento;

j) **Discutir Assunto** - através deste caso de uso, o aprendiz será capaz de interagir com outros aprendizes (do mesmo grupo ou de grupos diferentes) para a discussão de um determinado assunto;

k) **Orientar Assimilação** – através deste caso de uso, o professor deve ser capaz de orientar a assimilação de conteúdo pelo aprendiz;

l) **Orientar Aplicação** - através deste caso de uso, o professor deve ser capaz de orientar a aplicação de um conteúdo pelo aprendiz;

m) **Elaborar Lista de Atividades** – neste caso de uso, o Tutor deve ser capaz de selecionar uma ou mais atividades da base de atividades para compor uma lista de atividades para o aprendiz, automatizando dessa forma essa tarefa para o professor. O professor também pode compor (criar, alterar e excluir) uma lista de atividades;

n) **Manter Base de Atividades** - através deste caso de uso, o professor deve ser capaz de realizar a manutenção (inclusão/alteração/exclusão) de atividades que irão compor a base de atividades do sistema. Cada atividade poderá estar associada a um ou mais conteúdos (unidades de conhecimento) e que poderá ser passada ao aprendiz;

o) **Realizar Lista de Atividades** - através deste caso de uso, o aprendiz tem acesso às listas de atividades associadas a ele e pode escolher uma delas para trabalhar;

p) **Realizar Atividade** - através deste caso de uso, o aprendiz tem acesso a atividade que lhe foi repassada, pode trabalha nessa atividade e depois apresentar os resultados de cada atividade realizada. Este é um caso de uso abstrato já que os resultados podem ser os mais variados (problema, questão, projeto, pesquisa, redação, etc.) cada um deles deve ser delimitado e especificado, podendo ser um caso de uso diferente;

q) **Resolver Problema** - através deste caso de uso, o aprendiz deve poder visualizar uma coleção de problemas a serem resolvidos, selecionar um ou mais problemas e propor uma resolução para o(s) problema(s) selecionado(s) (BORGES, 2002);

r) **Avaliar Grupo** - através deste caso de uso, o professor deve ser capaz de obter a avaliação do desempenho dos grupos de aprendizes através de parâmetros estabelecidos por ele;

s) **Avaliar Aluno** - através deste caso de uso, o professor deve ser capaz de obter do sistema a avaliação do desempenho de cada aprendiz individualmente ao longo da sessão de aprendizagem ou no final da sessão, utilizando para isto alguns parâmetros estabelecidos pelo mesmo. O professor também poderá solicitar ao sistema, para efeito de avaliação, que o aprendiz responda a alguns problemas relacionados ao assunto em estudo.

Uma descrição mais detalhada destes Casos de Uso pode ser vista no Apêndice E, onde um formulário padrão do MATHNET foi utilizado para descrever mais detalhadamente os Atores envolvidos e a interação entre os agentes em cada um desses Casos de Uso. Baseando-se no levantamento e descrição feitos anteriormente, dos principais Casos de Uso do MATHNET, os mesmos são apresentados nos diagramas da Fig. 23 e da Fig. 24, a seguir.

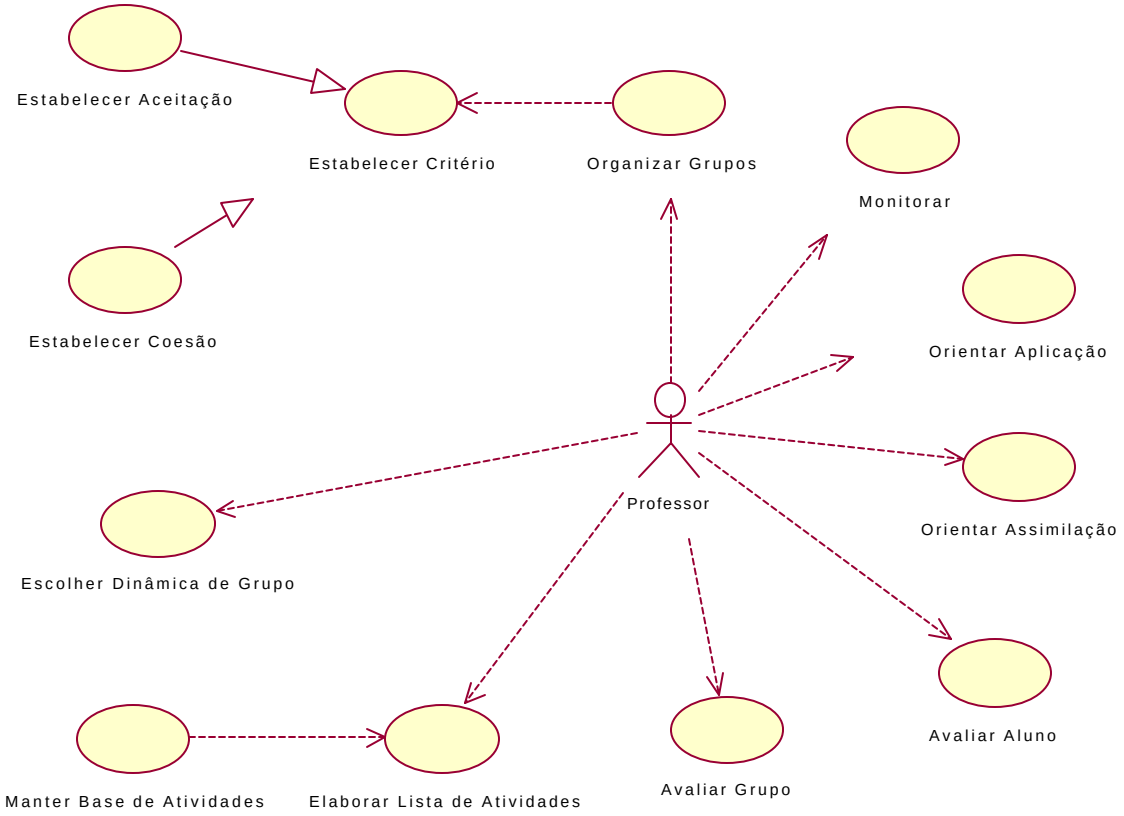

Figura 23 – Diagrama de Casos de Uso do Tutor em relação ao Professor

No diagrama acima, destaca-se a figura do Professor e os Casos de Uso com os quais ele se relaciona. Obtendo, assim, uma visão geral da importância do papel do Professor no sistema, pois ele pode participar ativamente desde o momento da formação dos grupos até a avaliação final do Aprendiz. Pode-se observar ainda que o Professor, por participar de vários Casos de Uso, é um dos principais atores deste sistema.

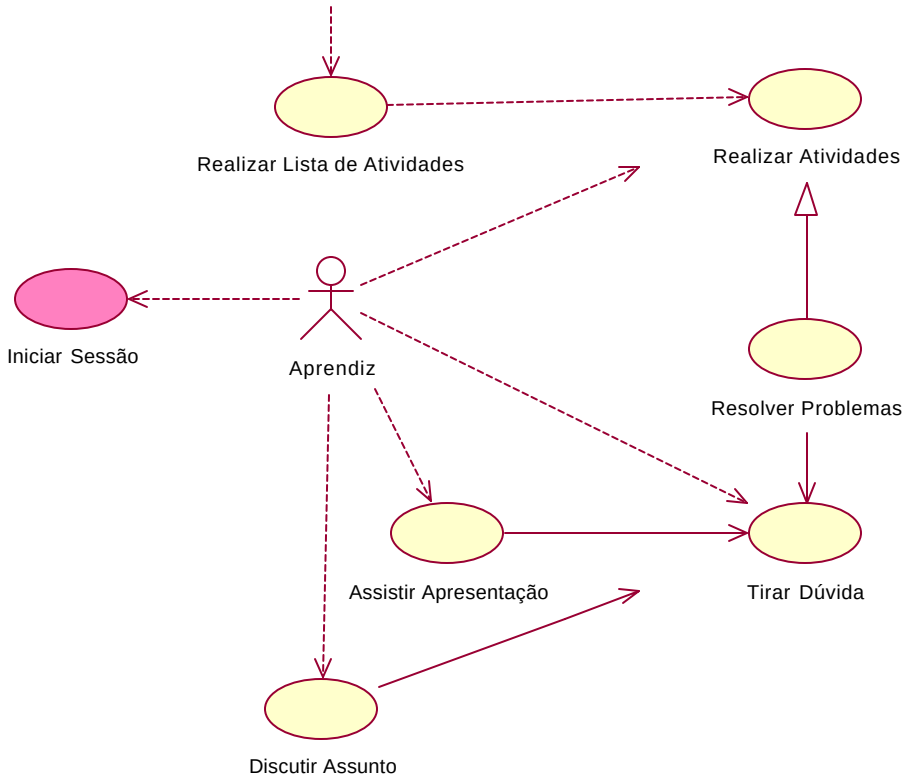

Figura 24 – Diagrama de Casos de Uso do Tutor em relação ao Aprendiz

O Aprendiz, na figura acima, é o Ator principal dos Casos de Uso em destaque. Sendo que o todo o processo de ensino-aprendizagem no MATHNET é iniciado a partir do momento em que o Aprendiz resolve iniciar uma sessão de aprendizagem. A figura geral contendo todos os principais Casos de Uso do MATHNET e seus Atores pode ser vista no Apêndice D.

Assim, em cada um dos diagramas mostrados enfatiza aspectos diferentes em relação ao Professor e ao Aprendiz. Assim, pode-se ter uma visão geral do MATHNET, seus principais Atores e os seus principais casos de uso.

4.5 Considerações finais

Neste capítulo apresentou-se como o Agente Tutor é estruturado no MATHNET. Em especial, seu papel, características, comportamento e interações com os demais agentes dessa SA. Ressaltou-se sua importância dentro dessa sociedade, bem como os principais Casos de Uso e os Atores envolvidos.

No Capítulo 5, a seguir, serão detalhados os Casos de Uso aqui apresentados, os agentes suficientes e necessários em cada um deles e os atores participantes e seus respectivos diagramas de Interação.

5 MODELAGEM DA SOCIEDADE DE AGENTES MATHNET

Neste capítulo são apresentados os Diagramas de Colaboração e Seqüência dos principais Casos de Uso do MATHNET, destacando-se aqueles em que a atuação do Tutor é mais evidenciada. Esses diagramas permitem que nos Casos de Uso sejam delimitados os agentes suficientes e necessários, bem como a descrição do seu Fluxo de Eventos.

5.1 Conceitos básicos

Uma dos mais importantes aspectos na busca do conhecimento é a obtenção de modelos. Por modelo entende-se um procedimento, de qualquer natureza (prático, matemático, gráfico, verbal, etc.), capaz de, em todos os aspectos relevantes, reproduzir uma relação de antecedentes (causas) e conseqüentes (efeitos) de forma idêntica como essa relação ocorre no "universo" real. Por concentrar-se nos aspectos relevantes, o modelo corresponderá a uma simplificação do evento real, nisto residem sua força e suas vantagens, pois se pode: prever como se comportará o universo ou determinar como nele introduzir uma determinada configuração final (tecnologia).

Deste modo, a modelagem do MATHNET aqui proposta destaca os principais casos de uso do sistema e os principais aspectos do Agente Tutor como foco deste trabalho.

Os diagramas aqui construídos serão de dois tipos: Diagrama de Colaboração e Diagrama de Seqüência, que juntos compõe o Diagrama de Interação. Para isto, foi utilizada uma ferramenta para a modelagem Orientada a

Objeto chamada RationalRose (RATIONALROSE, 2001), que como todo software CASE, tem por objetivo facilitar, tornar mais consistente e menos sujeita a erros a modelagem do projeto do sistema. Fornecendo deste modo um instrumento suficiente para uma modelagem visual, ou seja, modelar graficamente utilizando a notação UML descrevendo o sistema a ser desenvolvido. A modelagem visual permite uma melhor captação dos detalhes de um problema complexo, desconsiderando os detalhes irrelevantes ao projeto. Além disto, esta ferramenta trabalha com outros tipos de modelagem como Booch, OMT e OOSE (BOOCH et al., 2000) e também provê um mecanismo para visualizar o sistema a ser modelado em diferentes perspectivas.

Assim que, serão utilizados protocolos de cooperação para definir os possíveis fluxos de mensagens entre os agentes cooperadores. Esses protocolos podem ser derivados dos gráficos/diagramas de interação. Alguns desses diagramas serão vistos nas seções a seguir e apresentarão o fluxo de mensagens previsto para os agentes envolvidos nos Casos de Uso que foram definidos no Capítulo 4.

Entretanto, faz-se necessário, antes de iniciar a leitura das próximas seções, compreender alguns conceitos que foram utilizados aqui:

a) **Base de Atividades** - é utilizada para armazenar as atividades disponíveis no sistema e que deve ser apresentada ao Aprendiz, sendo que é o Agente de Domínio o responsável por acessar essas atividades disponibilizando-as ao Aprendiz. O Agente de Domínio contém tanto as referências às atividades desta Base quanto as referências às atividades externas, ou seja, que se encontram na internet. Deste modo, a partir dessa base, pode-se obter informações como: autor, localização, formatos (texto, áudio, vídeo, HTML, imagens etc.). O que fica nos Agentes de

domínio. Detalhes sobre a estrutura dessa base não são abordados neste trabalho. É também nos Agentes de Domínio que se localizam as Unidades de Conhecimento, ou seja, os conteúdos a serem assistidos pelos Aprendizes;

b) **Estereótipos de "interface", armazenamento e controle** (ver Fig. 25) - representam um conjunto de classes ou um serviço. Sendo utilizados pelo RationalRose para representar entidades como: 1) estereótipo de interface é responsável em ser o elo de ligação entre o ator e os demais agentes do sistema; 2) os de armazenamento servem, como o próprio nome já diz, para representar entidades que refletem algum tipo de armazenamento de informações/dados; e, 3) e os de controle representam, em geral, os agentes ativos no sistema, ou seja entidades que executam ações dentro do sistema.

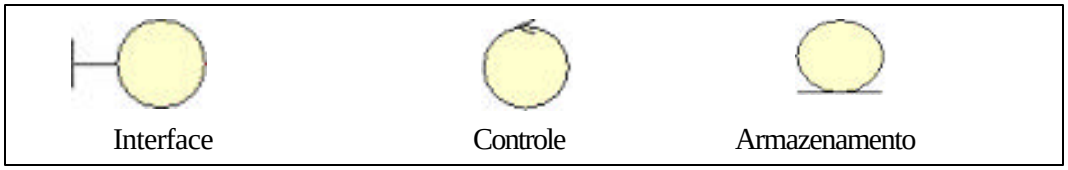

Figura 25 - Estereótipos

Na modelagem do MATHNET os casos de uso apresentados nas seções a seguir, tiveram como enfoque o Professor e o Aprendiz (ver Quadro 8), bem como as interações do Agente Tutor com os demais agentes desta sociedade. Não se levando em conta o Especialista/Engenheiro de Conhecimento que foi o foco de outro trabalho que construiu uma Ferramenta de Autoria para o ambiente MATHNET (COSTA, 2002).

ATORES	CASOS DE USO
Professor	Organizar Grupo Estabelecer Critério - Estabelecer Aceitação - Estabelecer Coesão Monitorar Orientar Aplicação Orientar Assimilação Elaborar Lista de Atividades - Manter Base de Atividades Realizar Lista de Atividades Avaliar Grupo Avaliar Aluno
Aprendiz	Iniciar Sessão Assistir Apresentação Realizar Atividades - Resolver Problemas Tirar Dúvida - Consultar Banco de Dúvidas - Enviar mensagem para o Professor - Recuperar Informações Discutir Assunto

Quadro 8 – Casos de Uso do MATHNET

Os casos de uso que não são o foco deste trabalho não se encontram detalhados nas próximas seções por não abrangerem o que se pretende demonstrar neste trabalho. Contudo, isto não diminui nem restringe a importância de cada um deles no ambiente do MATHNET. Para ver a especificação de cada um desses casos de uso feita através do formulário padrão utilizado pelo MATHNET para as especificações iniciais dos casos de uso, veja o Apêndice E.

5.2 Caso de Uso "Assistir Apresentação"

No diagrama (ver Fig. 26), o Aprendiz, após iniciar uma sessão de aprendizagem, pode decidir por iniciar a apresentação de um determinado conteúdo que seja de seu interesse. Deste modo, o Agente Tutor disponibiliza ao Aprendiz, com base no perfil (histórico: desempenho, último assunto estudado, etc.) solicitado

ao Agente de Modelagem do Aprendiz, todos os possíveis conteúdos a serem assistidos naquele momento.

Assim, logo após o Aprendiz selecionar um destes itens, o Agente Tutor solicita ao Agente de Domínio (que detém o conteúdo) que disponibilize esse conteúdo para que o Aprendiz para que o mesmo comece a interagir com ele, ou seja, comece a "assistir" a aula.

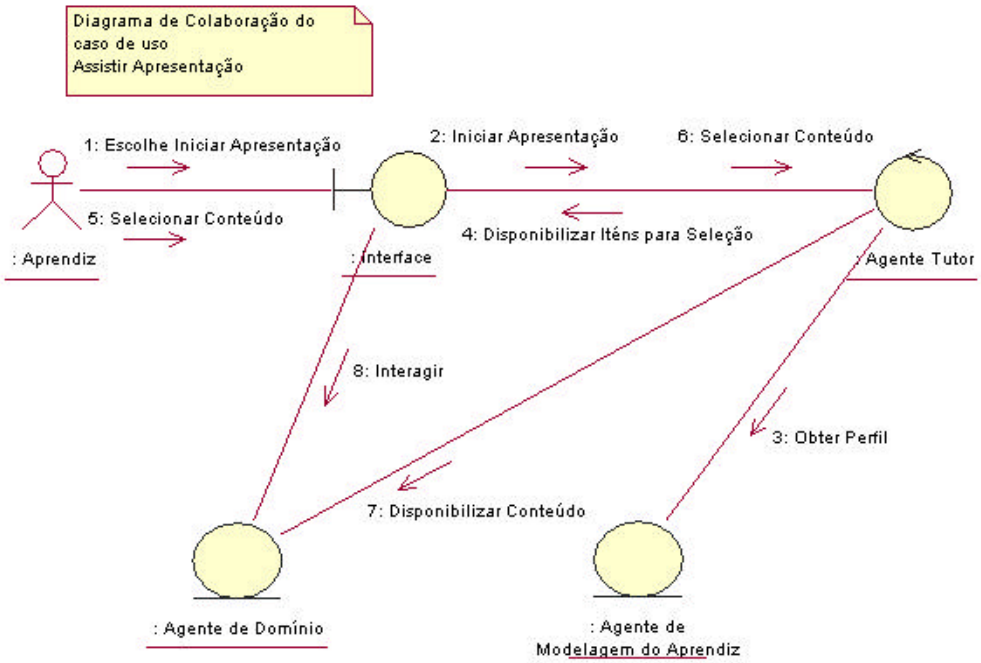

Figura 26 – Diagrama de Colaboração "Assistir Apresentação"

O termo conteúdo utilizado aqui se refere a aula que será assistida pelo Aprendiz e que pode estar em diferentes formatos (texto, áudio, vídeo, HTML, imagens, etc.) e que também poderá incluir interatividade com o Aprendiz ao longo da apresentação e não apenas uma posição passiva do Aprendiz diante da aula que está sendo apresentada.

No caso do Aprendiz necessitar tirar alguma dúvida durante a apresentação o Caso de Uso Tirar Dúvida será iniciado.

Em caso de haver uma interrupção do fluxo de apresentação (algum tipo de falha quanto ao conteúdo) o próprio sistema enviará ao Aprendiz uma mensagem de aviso. Com isto, o Aprendiz pode: fazer uma nova tentativa para continuar a assistir a apresentação ou encerrar a apresentação. Neste último, o Caso de Uso termina com fracasso.

Caso o conteúdo selecionado não tenha sido concluído o Aprendiz começará a apresentação a partir do mesmo ponto onde havia parado.

Um outro modo de ver o mesmo Caso de Uso é através do Diagrama de Seqüência (ver Fig. 27) que permite visualizar de forma temporal o fluxo de eventos envolvidos. Assim, vê-se o início do Caso de Uso a partir de uma ação do Aprendiz de iniciar uma apresentação de um conteúdo pretendido que se encontra em um dos Agentes de Domínio passando a interagir com ele.

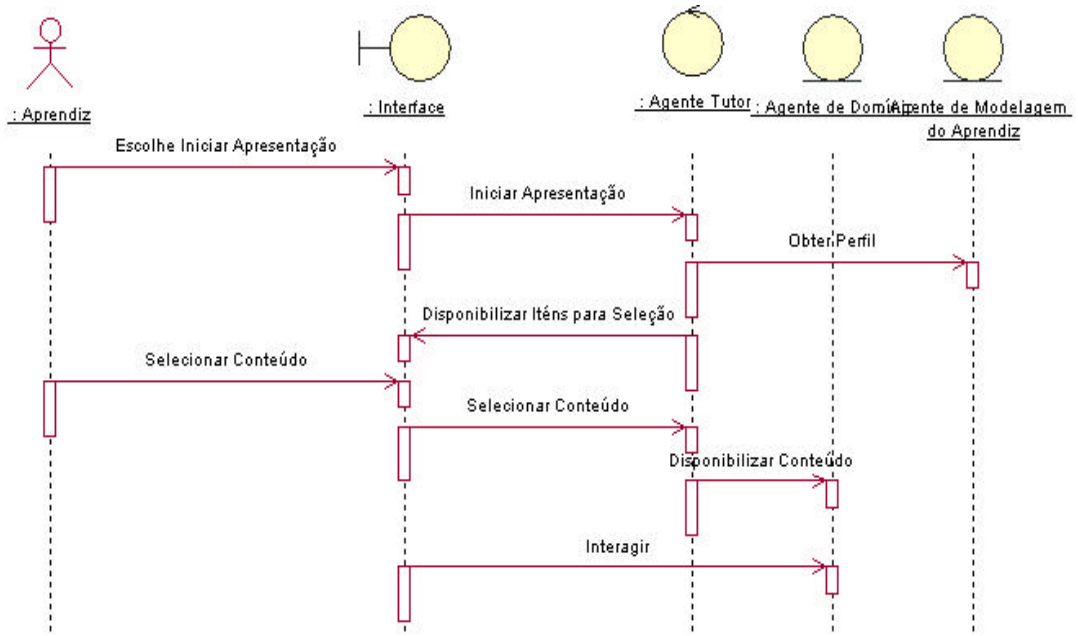

Figura 27 – Diagrama de Seqüência "Assistir Apresentação"

5.3 Caso de Uso “Organizar Grupos”

Através deste caso de uso o Professor obterá do sistema a divisão de uma turma de aprendizes em grupos, a partir de critérios de formação de grupo estabelecidos por ele. Para isto, o Professor deve ter feito *login* no sistema e o Agente Tutor deve possuir a relação dos alunos participantes da turma.

Em seu cenário básico (ver Fig. 28 e 29), este caso de uso se inicia quando o professor informa ao Agente Tutor que deseja organizar a turma em grupos. Deste modo, o Agente Tutor consulta a última organização da turma em grupos, ou seja, a última organização dos aprendizes em grupos e os últimos critérios de formação de grupos utilizados, disponibilizando-os ao Professor.

Assim que, o Professor, baseado nas informações apresentadas, informa ao Agente Tutor um ou mais novos critérios de formação de grupos, deste modo o caso de uso “Estabelecer Critérios” é iniciado e solicitando ao Agente Tutor que organize a turma em grupos. O Agente Tutor então, apresenta ao Professor o resultado da organização de grupos, baseado nos critérios estabelecidos pelo Professor. Este informa ao Agente Tutor que confirme esta organização apresentada e Agente Tutor assim o faz.

No caso de não existir a última organização da turma em grupos, o Agente Tutor avisará o Professor sobre isto e o próprio Agente Tutor fará uma divisão “qualquer” da turma em grupos e exibe-a ao Professor. Essa divisão “qualquer” pode ser implementada através de um algoritmo que faça uma divisão aleatória, ou seja, uma divisão que não leve em consideração os critérios informados pelo professor. Por exemplo: se existem 9 pessoas na turma serão formados grupos com três componentes, na ordem em que aparecem. A formação inicial dos grupos é feita através da construção de um Sociograma (ver seção 5.6) (SERRA, 2001).

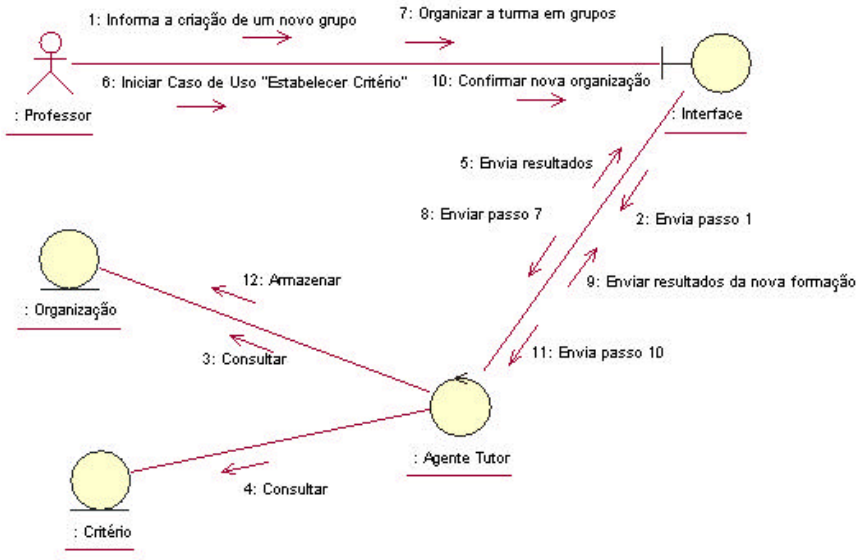

Figura 28 – Diagrama de Colaboração "Organizar Grupos"

Pode ocorrer também que o Professor não estabeleça novos critérios de formação de grupos. Deste modo, o Agente Tutor informará ao Professor que um ou mais novos critérios devem ser definidos e o Professor decide então definir novos critérios. Entretanto, caso o Professor decida continuar a operação sem definir novos critérios, o Agente Tutor irá considerar os últimos critérios definidos.

Caso o Agente Tutor não encontre critérios para dividir a turma em grupos, ele informará ao Professor sobre o ocorrido e não realizará a divisão da turma em grupos. Este fato também será avisado ao Professor.

Existe ainda a possibilidade do Agente Tutor não conseguir realizar a divisão da turma em grupos de modo a satisfazer simultaneamente todos os critérios estabelecidos pelo Professor. Deste modo, o Agente Tutor informará ao Professor sobre essa impossibilidade e disponibilizará a ele a formação encontrada que mais se aproxima dos critérios estabelecidos.

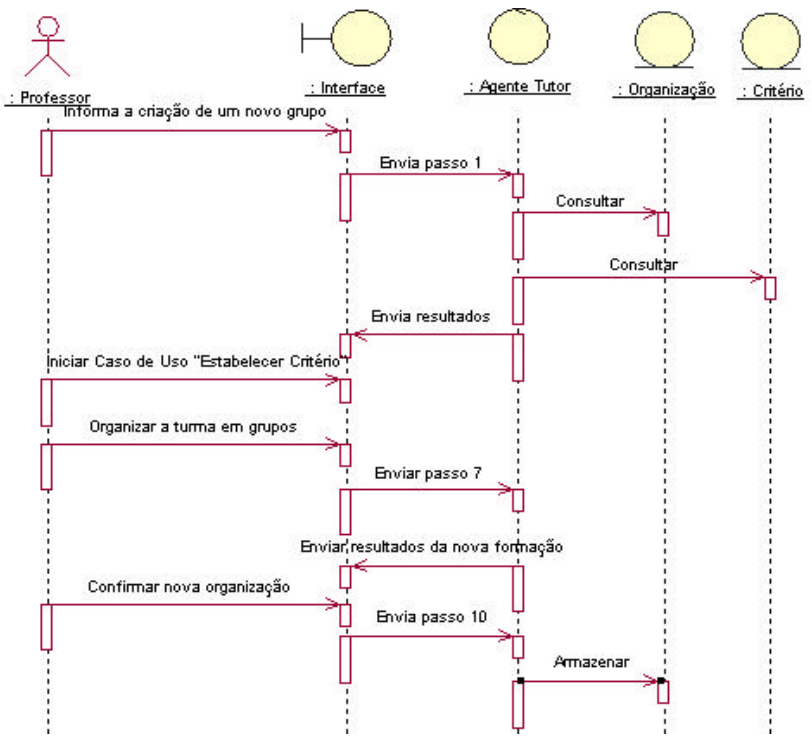

Figura 29 – Diagrama de Seqüência "Organizar Grupos"

O Professor pode também não concordar com a divisão da turma feita pelo Agente Tutor, podendo alterar manualmente a divisão apresentada. Está decisão de dividir manualmente (sem o auxílio do sistema) a turma em grupos pode ser tomada pelo Professor a qualquer momento.

Ao final da realização com sucesso deste caso de uso o Aprendiz fará parte de um grupo e estará apto a iniciar uma sessão de aprendizagem (*logar* no sistema).

5.4 Caso de Uso "Estabelecer Critério"

É o que se pode chamar de Caso de Uso Abstrato (OLIVEIRA, 2001), ou seja, aquele que possui apenas as características comuns aos casos de uso que estão contidos nele, ou seja, na verdade esses outros casos de uso descrevem novos comportamentos além dos que foram definidos em Estabelecer Critério.

Deste modo, Estabelecer Critério é extendido em dois outros: Estabelecer Aceitação e Estabelecer Coesão. Este caso de uso é iniciado a partir do Caso de Uso Organizar Grupo.

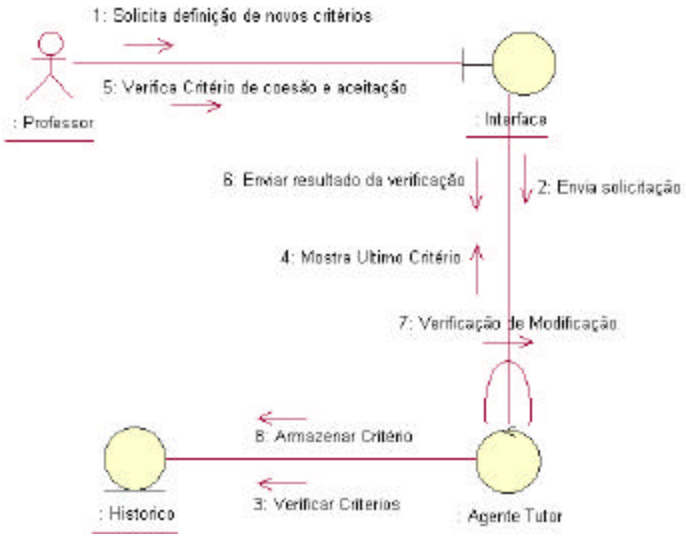

Figura 30 – Diagrama de Colaboração "Estabelecer Critério"

Através das Fig. 30 (acima) e 31, pode-se observar que o Professor informa ao Agente Tutor que deseja definir novos critérios, que são utilizados para a formação de grupos. O Tutor disponibiliza então os detalhes do último critério que foi estabelecido. O Professor modifica esses critérios. O Tutor valida cada uma das modificações feitas e as apresenta ao Professor que decide armazenar o critério com suas modificações realizadas.

No momento em que o Professor decide modificar o critério é que os Casos de Uso Estabelecer Aceitação ou Estabelecer Coesão serão iniciados. No caso do Tutor não validar as modificações feitas, o Professor será informado do ocorrido e será solicitado que ele cancele as modificações e faça outras.

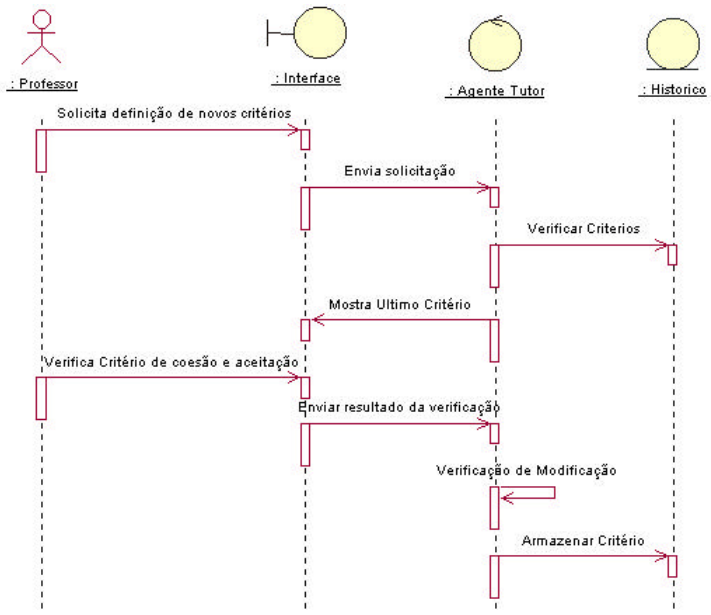

Figura 31 – Diagrama de Seqüência "Estabelecer Critério"

O professor poderá, a qualquer momento, cancelar toda a operação e recomeçar todo o processo do início novamente.

5.5 Caso de Uso "Estabelecer Aceitação"

Por Critérios de Aceitação entende-se que sejam os Padrões de Grupo, os Níveis de Aceitação baseados nesses padrões e os Perfis de todos os alunos da turma, juntamente com um perfil da turma como um todo, que é derivado dos perfis individuais dos aprendizes.

O Perfil de Aluno, definido para o MATHNET, é composto por três atributos específicos como: 1) a capacidade de liderança; 2) o senso de cooperação; e, 3) o nível de conhecimento do domínio. O Agente de Modelagem de Aprendiz é o responsável por gerar um valor, variando de 1 (muito baixo) a 5 (muito alto) para cada um desses atributos. Já o Perfil de Grupo pode ser definido com sendo um agrupamento de perfis dos Alunos que formam um dado Grupo.

Padrão de Grupo é um esquema que define implicitamente uma coleção de perfis de grupo. No MATHNET, existem dois Padrões de Grupo definidos pelo professor: o Padrão de Grupo Ideal (que define as melhores formações de grupo para estudar um determinado assunto, usando uma dada estratégia) e o Padrão de Grupo Médio (define a forma geral dos grupos que são minimamente aceitáveis pelo Professor) (COUTINHO et al., 2001). Deste modo, mesmo que o professor não consiga que todos os grupos a serem formados se enquadrem no Padrão Ideal, o Padrão médio permitirá especificar as características mínimas a respeito da composição de um grupo.

Os Níveis de Aceitação são definidos através do Padrão de Grupo Ideal e do Padrão de Grupo Médio e varia de 0 (não aceitável) a 3 (alta aceitação).

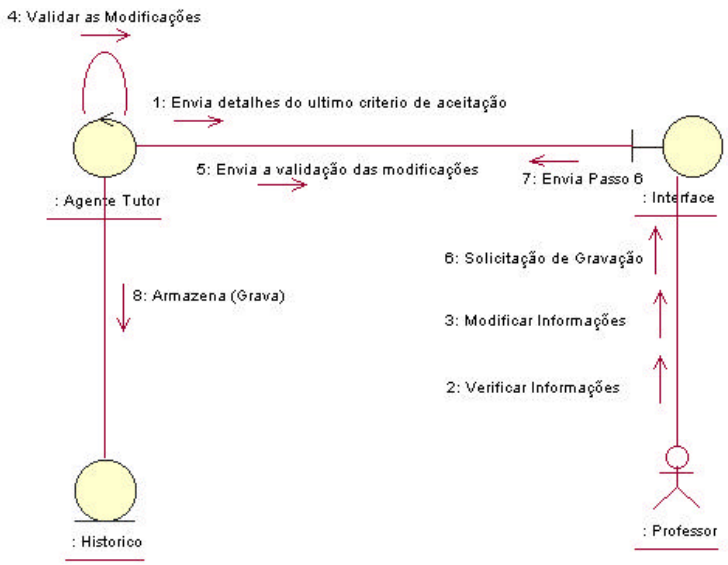

Figura 32 – Diagrama de Colaboração "Estabelecer Aceitação"

Assim, conforme o Diagrama de Colaboração (ver Fig. 32, acima) e o Diagrama de Seqüência (ver Fig. 33, a seguir), o Tutor disponibiliza ao Professor os detalhes do último Critério de Aceitação estabelecido. O Professor confere os perfis dos aprendizes da turma e o Perfil da Turma (Verificar Informações) e decide

modificar algumas dessas informações, como o Padrão de Grupo e/ou seus perfis de grupo (incluindo ou excluindo). O Tutor valida as modificações feitas para cada ação do Professor, este por sua vez encerra as alterações e informando ao Tutor que o critério pode ser armazenado.

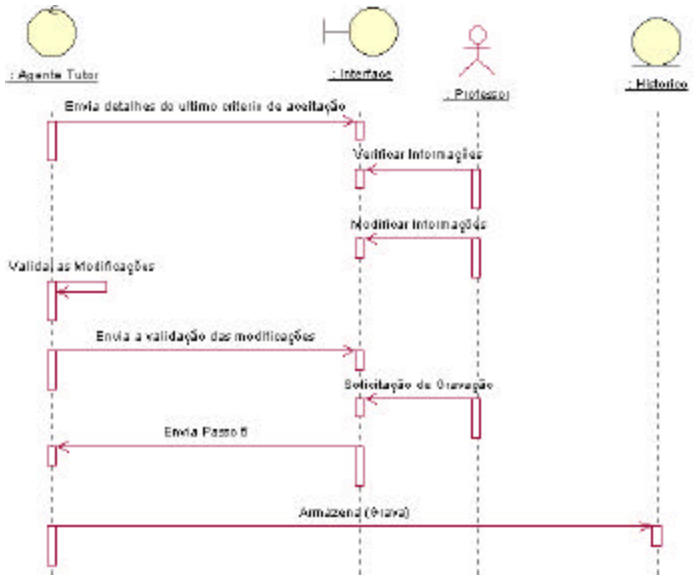

Figura 33 – Diagrama de Seqüência "Estabelecer Aceitação"

No caso de não haver nenhum Critério de Aceitação estabelecido, o Tutor cria um critério inicial (*default*) e o disponibiliza ao Professor. Há ainda como Cenário Alternativo a possibilidade de modificar a constituição dos Perfil do Aluno e os Níveis de Aceitação.

5.6 Caso de Uso "Estabelecer Coesão"

Por Critério de Coesão entende-se que o mesmo seja composto por: Regras de Coesão, Níveis de Coesão (baseados nessas regras) e o Sociograma da Turma (OLIVEIRA, 2001).

Sociograma é um grafo onde os *nós* representam cada aluno da turma e os arcos as escolhas de cada aluno, ou seja, com quem o aluno tem preferência

para trabalhar. Essas escolhas são descritas pelos próprios alunos através de um questionário que foi anteriormente respondido e que serve de base para a construção desse sociograma. Na Figura 34, tem-se um exemplo de um possível sociograma gerado a partir de algumas preferências de uma turma de alunos fictícia.

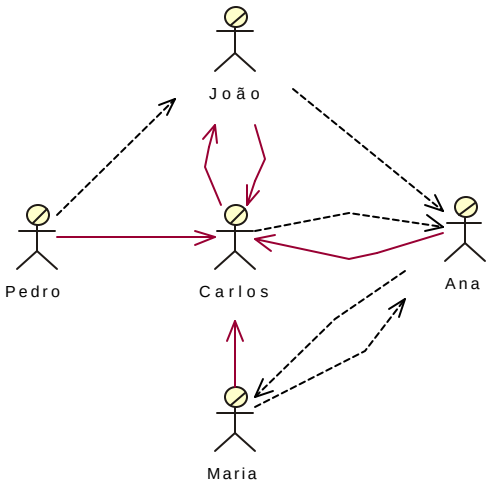

Figura 34 – Um possível Sociograma

As linhas cheias representam primeira escolha de cada aluno e as linhas tracejadas segunda escolha. E é através deste Sociograma que as Regras de Coesão e os Níveis de Coesão foram definidos.

Regras de Coesão são critérios que se referem à formação de grupos. E como o próprio nome já diz, baseia-se em regras que são derivadas de trabalhos na área da pedagogia aplicada à aprendizagem cooperativa (HAIDT, 1999) e que utilizam sociogramas como forma de dividir, da melhor maneira possível, uma turma de alunos em grupos onde se aplica a aprendiz. No MATHNET foram definidas quatro regras que servem como base para definir os Níveis de Coesão (OLIVEIRA, 2001).

Níveis de Coesão são pesos que variam de 0 a 3 (associados às regras) e que de forma crescente definem a não coesão (valor 0) até o altamente coeso (valor 3).

Assim que, o Tutor disponibiliza ao Professor os detalhes do último Critério de Coesão estabelecido, ele confere essas informações e decide modificar algumas delas, dentre elas o Sociograma. Quando essa informação de que um novo Sociograma deve ser construído chega ela, é repassada ao Tutor. O Professor especifica o tempo limite de espera pelas respostas dos Aprendizes que o Tutor deve aguardar.

Pela Figura 35, pode-se observar como o Critério de Coesão é estabelecido. O Tutor como primeira medida envia um questionário com as perguntas do Sociograma aos Aprendizes e eles respondem o questionário. O Tutor ao expirar o tempo limite de espera das respostas desse questionário faz:

a) a coleta dessas informações;
b) a construção de um novo Sociograma da turma; e,
c) a disponibilização desse Sociograma ao Professor. O professor, por sua vez, encerra as modificações e o Tutor armazena o novo Critério de Coesão que foi estabelecido.

No caso de não haver nenhum Critério de Coesão estabelecido o Tutor cria um critério inicial (*default*) e mostra-o ao Professor. As Regras de Coesão também podem ser modificadas.

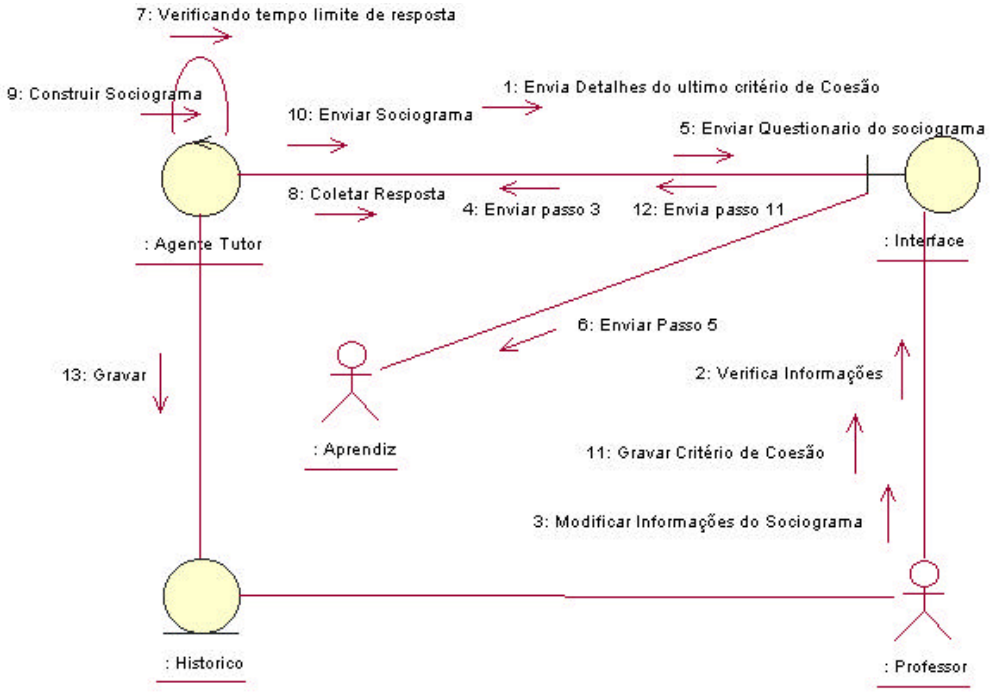

Figura 35 – Diagrama de Colaboração "Estabelecer Coesão"

É possível também que ao final do tempo limite de espera algum Aprendiz não tenha respondido ao questionário. Neste caso, o Tutor constrói o Sociograma baseado apenas nas respostas daqueles aprendizes que responderam o questionário e ao mesmo tempo informa ao Professor sobre o ocorrido.

5.7 Caso de Uso "Manter Base de Atividades"

A Base de Atividade é representada pelo estereótipo de Armazenamento (*:Atividades*) e nela estão contidas todas as atividades que poderão ser realizadas no sistema. Na manutenção pode-se incluir, alterar ou excluir qualquer atividade.

Por atividade entende-se uma determinada tarefa a ser cumprida pelo Aprendiz durante a fase de Aplicação do Conhecimento.

Esta manutenção pode ser vista conforme o Diagrama de Colaboração da Fig. 36 e no Diagrama de Seqüência da Fig. 37, onde se têm descrito o cenário básico deste Caso de Uso que é a inclusão de uma nova atividade.

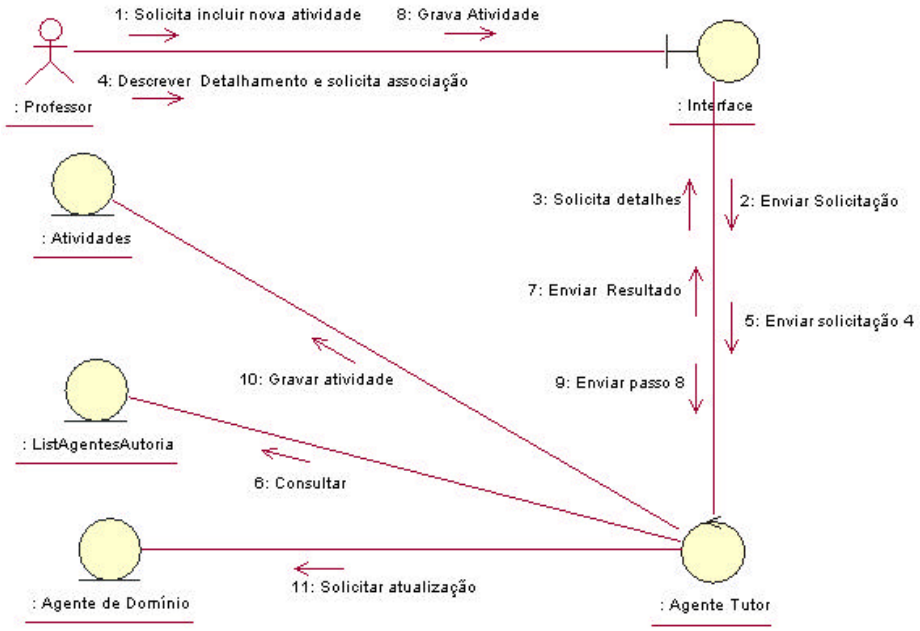

Figura 36 – Diagrama de Colaboração "Manter Base de Atividades"

Desta forma a manutenção dessa Base é iniciada a partir do momento em que o Professor decide incluir uma nova atividade nessa base. O Tutor então, solicita ao Professor que informe os detalhes dessa atividade (Ex: enunciado, valores, etc.), este por sua vez a compõe, associando-a a uma ou mais Unidades de Conhecimento obtidas pelo Tutor ao consultar a Lista de Agentes de Domínio. O Professor informa então ao Tutor que deseja armazenar a atividade e o Tutor assim o faz e depois ele solicita ao Agente de Domínio que ele atualize as suas referências de atividades (recurso/fonte). Esta atualização não ocorrerá apenas no cenário básico apresentado, mas toda vez que ocorrer qualquer manutenção (incluir/alterar/excluir).

Caso o Professor decida alterar alguma atividade existente, ele deve selecionar a atividade desejada, o Agente Tutor disponibiliza então os detalhes dessa atividade. O Professor realiza as modificações que julga serem necessárias e informa ao Agente Tutor para armazenar a atividade modificada e assim ele o faz.

No caso dessa modificação referir-se à remoção de alguma unidade de conhecimento associada a essa atividade então o Agente Tutor antes de remover essa associação deverá executar um procedimento que verifica antes se esta atividade não pertence a nenhuma Lista de Atividades. Caso o Professor queira excluir a atividade o Agente Tutor também deverá realizar essa verificação antes de permitir a exclusão. Nessa verificação o Professor será avisado pelo Agente Tutor no caso de haver algum impedimento para a realização da modificação solicitada.

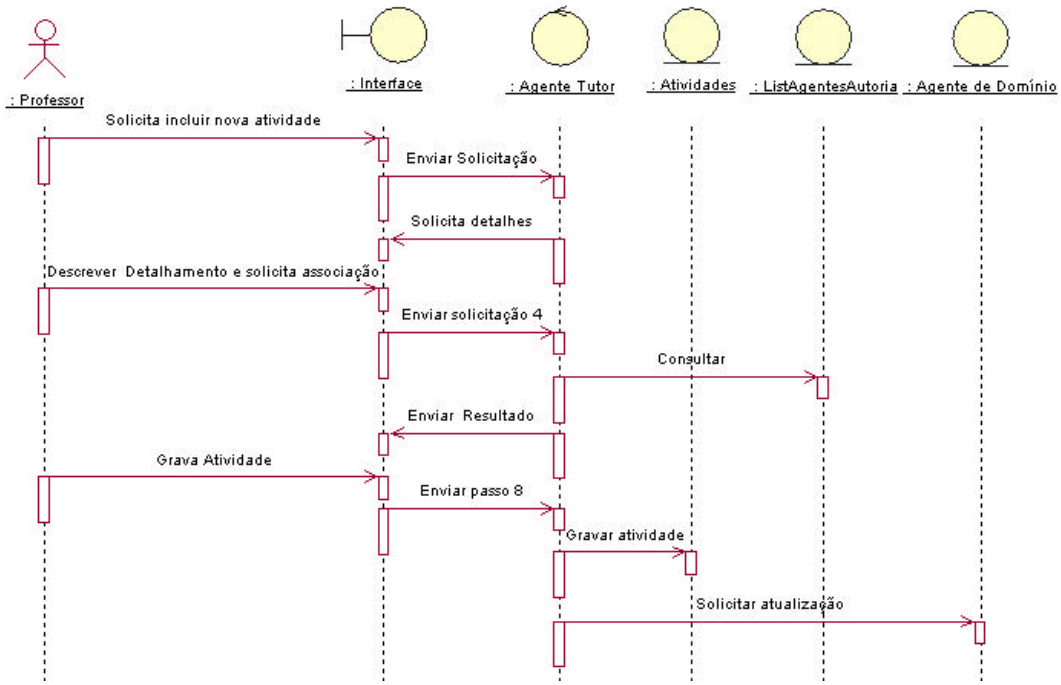

Figura 37 – Diagrama de Seqüência Manter Base de Atividades"

Se por acaso ocorrer alguma falha no momento de armazenar a atividade o Agente Tutor enviará ao Professor uma mensagem de aviso sobre o ocorrido e daí

o Professor poderá duas atitudes: faz uma nova tentativa de salvar a atividade ou cancela a operação.

5.8 Caso de Uso “Iniciar Sessão”

Ao final deste Caso de Uso o Aprendiz terá à sua disposição uma sessão de aprendizagem a qual permitirá que seja realizado o processo de ensino-aprendizagem.

Para que esse caso de uso seja iniciado faz-se necessário que exista pelo menos um Aprendiz que pertença a um grupo do sistema e que detenha um código de usuário e senha válidos para realizar com sucesso o *login*.

Este caso de uso (ver Fig. 38) começa a partir do momento que o Aprendiz envia suas informações de *login* (código de usuário e senha) para que o Tutor realize a autenticação dessas informações.

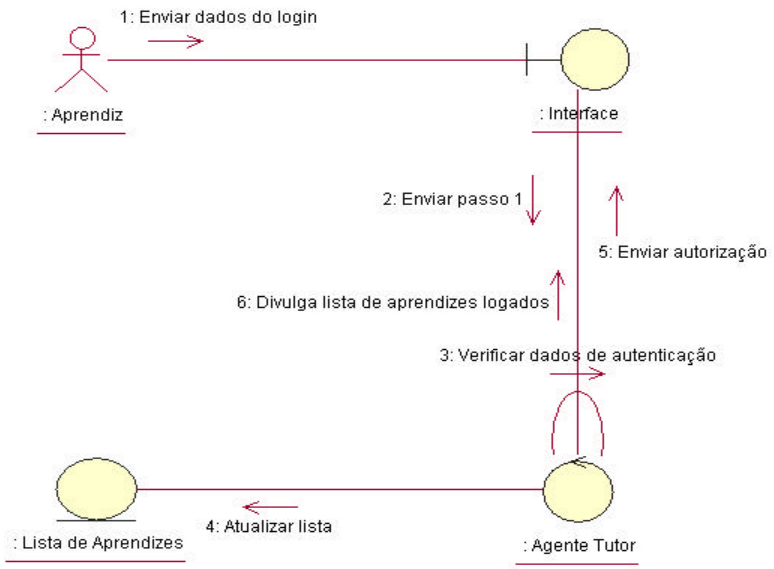

Figura 38 – Diagrama de Colaboração “Iniciar Sessão”

O Tutor ao receber essas informações realiza a autenticação do Aprendiz e caso ocorra sucesso o Agente Tutor: 1) atualiza a lista de aprendizes que se

encontram atualmente *logados* no sistema; 2) envia ao Aprendiz uma autorização (código de acesso) para participar da sessão; e, 3) divulga a lista dos aprendizes atualmente l*ogados* a todos os participantes do sistema.

Caso o Aprendiz não seja autenticado pelo sistema, o Agente Tutor envia ao Aprendiz uma mensagem de aviso sobre o ocorrido e o Aprendiz pode tomar duas decisões: ou ele tenta novamente fazer o *login* ou cancela a operação e sai do sistema.

No diagrama da Figura 39, abaixo, pode-se observar melhor como o fluxo de mensagens deste caso de uso ocorre. Sendo que as três últimas mensagens ocorrem “ao mesmo tempo”.

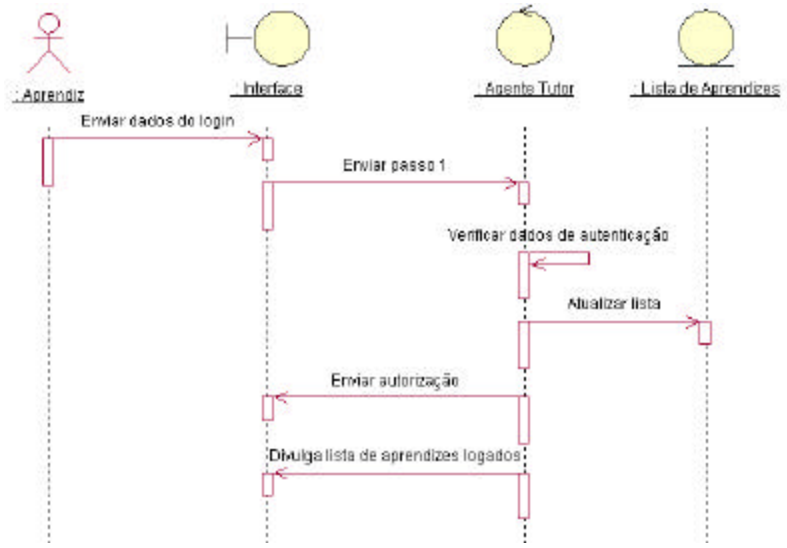

Figura 39 – Diagrama de Seqüência “Iniciar Sessão”

5.9 Caso de Uso “Elaborar Lista de Atividades”

Através deste caso de uso o Agente Tutor deve ser capaz de selecionar uma ou mais atividades da Base de Atividades para compor uma lista de atividades que é associada a um ou mais Aprendizes, automatizando dessa forma essa tarefa para o professor.

O professor pode ou não aceitar a lista elaborada pelo Agente Tutor ou então modifica-la. O professor também pode compor (criar, alterar e excluir) uma lista de atividades a qualquer momento. Essas listas na verdade possuem apenas as referências às atividades que se encontram na Base de Atividades e/ou no Agente de Domínio referente ao conteúdo estudado.

Deste modo, este caso de uso (ver Fig. 40 e 41) será iniciado a partir do momento em que o Tutor decide criar uma Lista de Atividades, baseado no Agente de Modelagem do Aprendiz, por dois motivos: porque ele identificou que não havia nenhuma Lista associada ao Aprendiz ou porque ele identificou que o Aprendiz necessita fazer mais atividades para reforçar o conteúdo que está sendo estudado.

Então, o Tutor identifica através do Agente de Modelagem do Aprendiz quais as atividades se referem à unidade de conhecimento associada àquele Aprendiz. O Agente Tutor então constrói a Lista contendo a(s) referência(s) à essa(s) atividades, associa essa lista ao(s) Aprendiz(es) e a armazena.

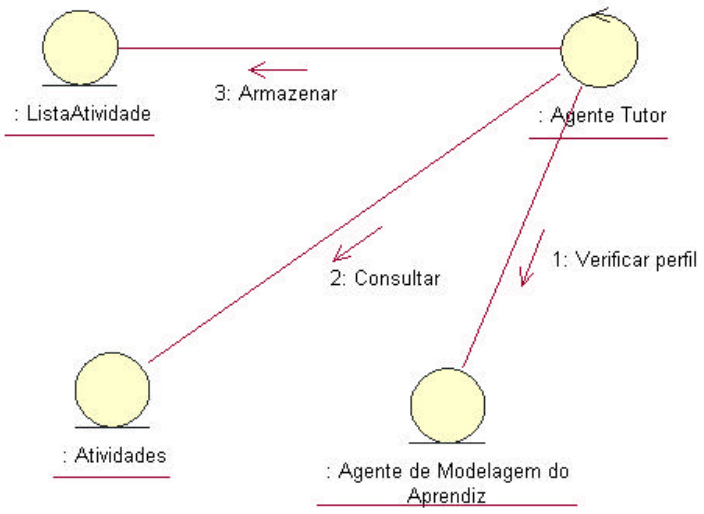

Figura 40 – Diagrama de Colaboração "Elaborar Lista de Atividades"

Como já foi dito inicialmente, o Professor também pode elaborar uma Lista de Atividades a qualquer momento. Com isso, o Agente Tutor mostra ao

Professor as referências das atividades que se encontram na Base de Atividades e nos Agentes de Domínio e que estão associadas às unidades de conhecimento do Aprendiz em questão. O Professor constrói a Lista selecionando as atividades desejadas e a associando ao(s) Aprendiz(es). E por fim, o Agente Tutor armazena essa Lista.

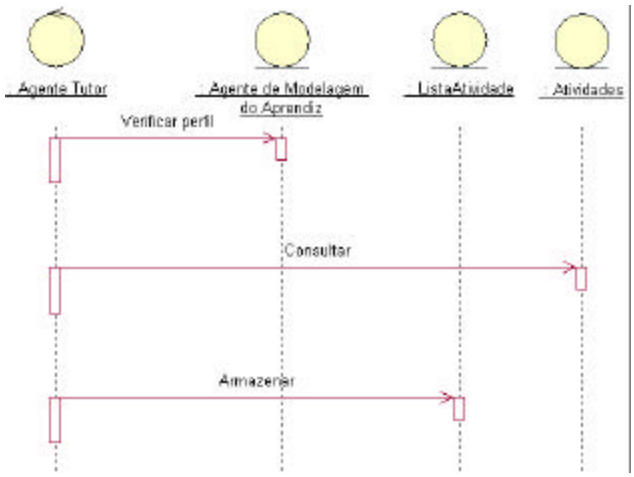

Figura 41 – Diagrama de Seqüência "Elaborar Lista de Atividades"

No caso do Professor decidir alterar alguma Lista de Atividades (feita por ele ou pelo Tutor) o Agente Tutor disponibilizará a ele uma relação de todas as Listas existentes, o Professor selecionará uma delas para alterar e o Tutor lhe disponibilizará a Lista selecionada. O Professor após alterar o conteúdo dessa Lista (referências) e/ou as associações ao(s) Aprendiz(es), informa ao Tutor que deseja salvar essas alterações, o Agente Tutor valida essas modificações e armazena a Lista com as modificações realizadas.

Essa validação consiste do Tutor verificar se o professor pode alterar (incluir/alterar) o conteúdo (referências) da Lista no caso dela não ter sido ainda utilizada ou então se ele pode desassociar um Aprendiz dessa lista (no caso do Aprendiz não ter ainda trabalhado ou está trabalhando essa lista).

Como resultado final do sucesso desse caso de uso uma Lista de Atividades será elaborada e permitirá que o Aprendiz faça as atividades nela descrita.

5.10 Caso de Uso “Realizar Lista de Atividades”

Através deste caso de uso (ver Fig. 42 e 43) o Aprendiz pode trabalhar em alguma Lista de Atividades associada a ele e este caso de uso será iniciado a partir do momento em que o Aprendiz decidir realizar alguma atividade e desde que exista pelo menos uma lista de atividades associada ao aprendiz.

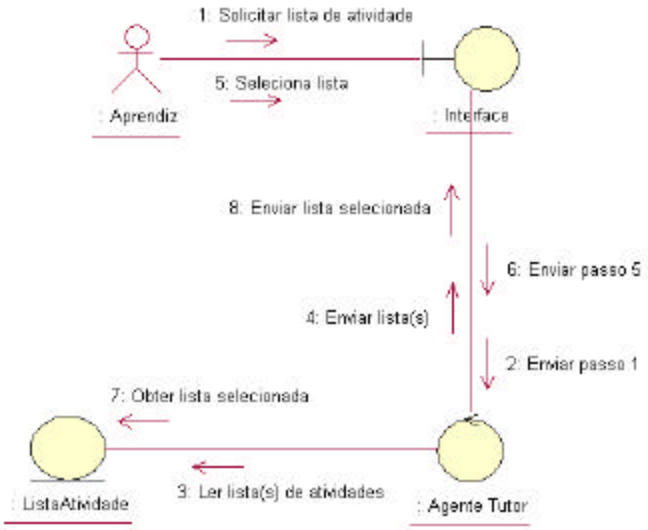

Figura 42 – Diagrama de Colaboração “Realizar Lista de Atividades”

Assim, o Aprendiz solicita ao Agente Tutor a(s) Lista(s) de Atividades associadas a ele, o Tutor então disponibiliza uma relação dessa(s) Lista(s) dentre as quais o Aprendiz escolhe uma para trabalhar. Deste modo, o Agente Tutor disponibiliza a Lista selecionada e o Aprendiz passa a trabalhar na atividade de acordo com o Caso de Uso Realizar Atividade. Ao mesmo tempo em que o Aprendiz trabalha nessa lista o Tutor vai armazenando informações sobre ele.

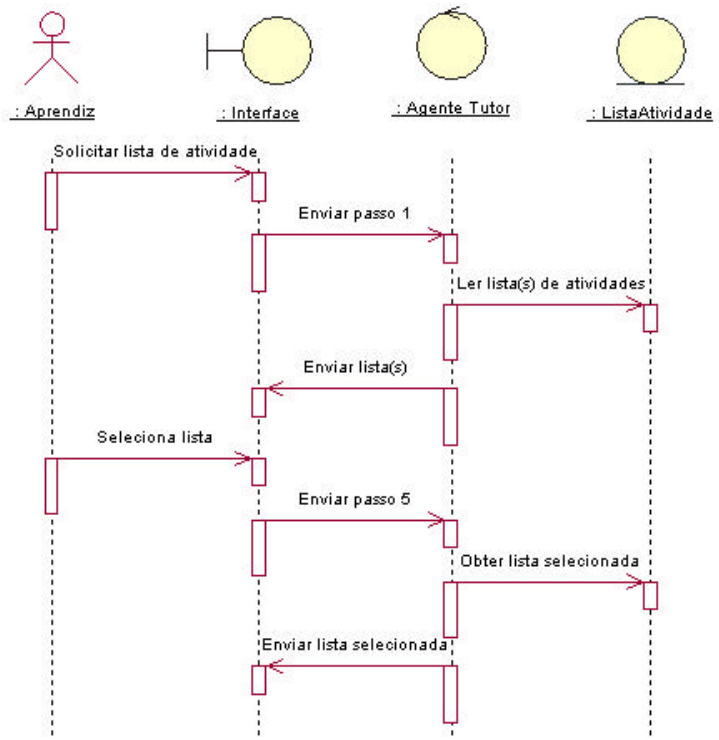

Figura 43 – Diagrama de Seqüência "Realizar Lista de Atividades"

Realizar atividade se repete até que o aprendiz decida não mais realizar nenhuma atividade ou até que ele conclua todas as atividades associadas a ele daquela lista.

Como resultado final da realização com sucesso deste caso de uso o Aprendiz poderá acessar as suas Listas e dar início a alguma atividade.

5.11 Caso de Uso "Realizar Atividade"

Através deste caso de uso o Aprendiz tem acesso a atividade que lhe foi repassada, trabalha nessa atividade e apresenta os resultados de cada atividade realizada.

Para cada atividade a ser desenvolvida existirá um caso de uso associado a ela. Pois, cada atividade possui características próprias para o desenvolvimento de sua resolução.

Deste modo, uma atividade pode ser definida como:

a. Problema (Caso de uso “Resolver Problemas”). Ver detalhamento na especificação do próprio caso de uso.
b. Questão (Caso de uso “Resolver Questão”). Questão é uma estrutura composta de: a) enunciado (pergunta) que exige um comentário (texto) em resposta.
c. Projeto (Caso de Uso “Construir Projeto”). Projeto é uma estrutura composta de: a) uma sentença que estabelece o(s) objetivo(s) do projeto; b) uma lista de dados extraídos do enunciado do projeto; c) uma lista de conhecimentos que espera-se que sejam utilizados nos passos de desenvolvimento do projeto; d) uma estratégia, ou seja, a divisão do projeto em etapas; e) uma ou mais listas dos passos de resolução associados às etapas da estratégia;
d. Pesquisa (Caso de uso “Elaborar Pesquisa”). Pesquisa é uma estrutura composta de: a) Tema, que é um texto descritivo que delimita o conteúdo da pesquisa; b) Características, que identifica as características especiais;
e. Redação/Artigo (Caso de uso “Elaborar Redação/Artigo”). Redação é uma estrutura composta de: a) Tema, que é um texto que descritivo que delimita conteúdo da redação; b) Estilo, que identifica a estrutura da redação que deve ser aplicada; c) Característica, que serve para identificar algumas características especiais que devem ser abordados durante o desenvolvimento da redação (número de linhas, considerar algum ponto, etc.).

As atividades definidas acima não se encontram inteiramente delimitadas podendo sofrer alterações na estrutura aqui proposta, além disso, outras atividades, além das que foram apresentadas aqui, podem ainda ser definidas e adicionadas ao MATHNET. A cada atividade desenvolvida pelo Aprendiz o resultado podendo ser:

a. Número – como resultado de uma operação matemática;
b. Texto;
c. Figura - que pode ser um Fluxograma, Organograma, Planta Baixa, Gráfico, etc.;
d. Foto – uma Imagem fotográfica;
e. Vídeo;
f. Som;

O caso de uso (ver Fig. 44 e 45) inicia-se quando o Aprendiz no caso de uso Realizar Lista de Atividades escolhe uma das atividades listadas para trabalhar.

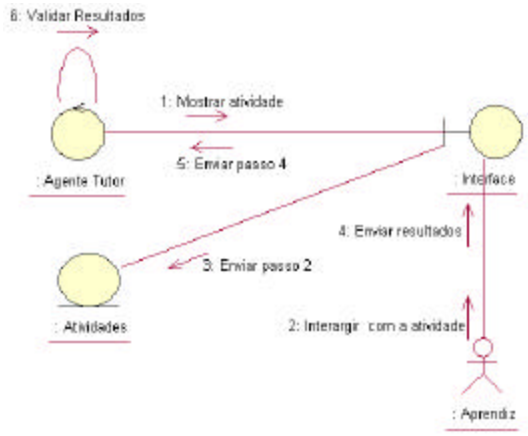

Figura 44 – Diagrama de Colaboração "Realizar Atividade"

É importante ressaltar, que na verdade este caso de uso é abstrato, ou seja, generaliza os demais casos de uso de acordo com a atividade a ser desenvolvida (questão, problema, pesquisa, etc.).

Assim, em linhas gerais neste caso de uso o Agente Tutor é o responsável por exibir a atividade selecionada em detalhes para o Aprendiz, este trabalha nela até obter os resultados que serão submetidos ao Agente Tutor para serem validados.

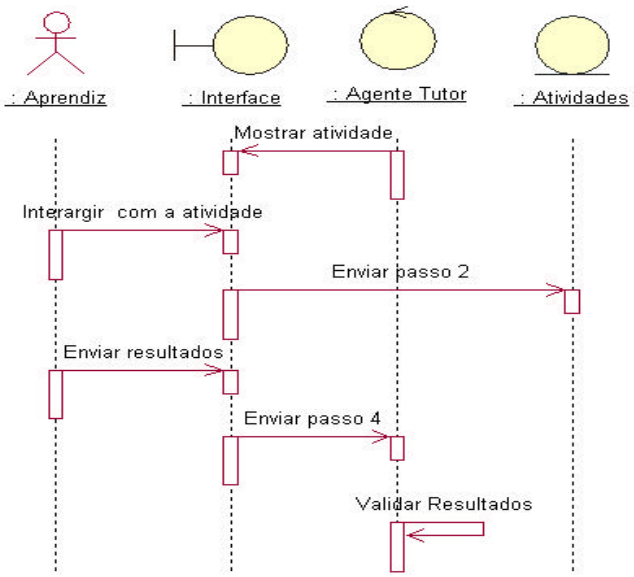

Figura 45 – Diagrama de Seqüência "Realizar Atividade"

Dependendo da atividade escolhida que um caso de uso específico será iniciado, além disso, durante a o desenvolvimento de qualquer uma das atividades um outro caso de uso o "Tirar Dúvidas" pode ser iniciado a qualquer momento pelo próprio Aprendiz ou então pelo próprio sistema ao verificar que o mesmo está tendo dificuldade.

Deste modo, este é um caso de uso bastante simples pois na verdade é através dele que serão iniciados os outros casos de uso, conforme descrito anteriormente, é que serão efetivamente realizados pelo Aprendiz.

A realização, com sucesso deste caso de uso, permitirá que o Aprendiz desenvolva algum tipo de atividade com o intuito de reforçar os conceitos aprendidos durante a apresentação de algum conteúdo (aula), colocando em prática a teoria exposta ao longo do "curso".

5.12 Considerações finais

É a partir dos Casos de Uso apresentados neste capítulo que pode-se identificar, para cada um deles, os agentes participantes, ou seja, aqueles que são suficientes e necessários para realizá-lo.

Neste capítulo, foram modelados alguns dos principais Casos de Uso do Professor e do Aprendiz, dando ênfase principalmente àqueles dos quais participa o Agente Tutor e suas interações com os demais agentes.

No próximo capítulo, serão detalhados as comunicações do Agente Tutor no MATHNET através do detalhamento de seus principais Casos de Uso.

6 COMUNICAÇÃO DOS AGENTES NO MATHNET

Neste capítulo será descrita a seqüência de mensagens que podem ser trocadas entre os agentes envolvidos com o Tutor no MATHNET. Os protocolos de comunicação definidos baseiam-se nas especificações de FIPA-ACL, nos Diagramas de Interação apresentados no Capítulo 5 e nos Casos de Uso definidos no Capítulo 4.

6.1 Comunicação: um trabalho em conjunto

Já foi citado anteriormente que os agentes no MATHNET não trabalham sozinhos, mas sim dentro de uma sociedade. Logo, para realizar suas tarefas eles precisam interagir com outros agentes, e para isso, é necessário que eles se comuniquem. No MATHNET, a comunicação é um suporte à cooperação.

A comunicação entre os agentes que compõe a Sociedade de Agentes MATHNET tem como objetivo dispor aos usuários (Aprendiz e Professor) um ambiente que contemple o processo de ensino-aprendizagem pela cooperação entre esses usuários dentro de um ambiente computadorizado, de modo a levar em conta os perfis individuais e de grupo e buscando a estratégia que mais se adeqüe às necessidades dos aprendizes dentro daquilo que é requisitado pelo professor para que seja alcançado.

A seguir, serão mostrados os protocolos definidos para reger estas comunicações através de alguns diagramas derivados dos diagramas de interação mostrados no Capítulo 5.

6.2 Comunicação do Agente Tutor no Caso de Uso “Assistir Apresentação”

A Fig. 46, mostra a seqüência de troca de mensagens que define um protocolo de comunicação entre o Agente Tutor, um Agente de Domínio e o Agente de Modelagem do Aprendiz no caso de uso “Assistir Apresentação”.

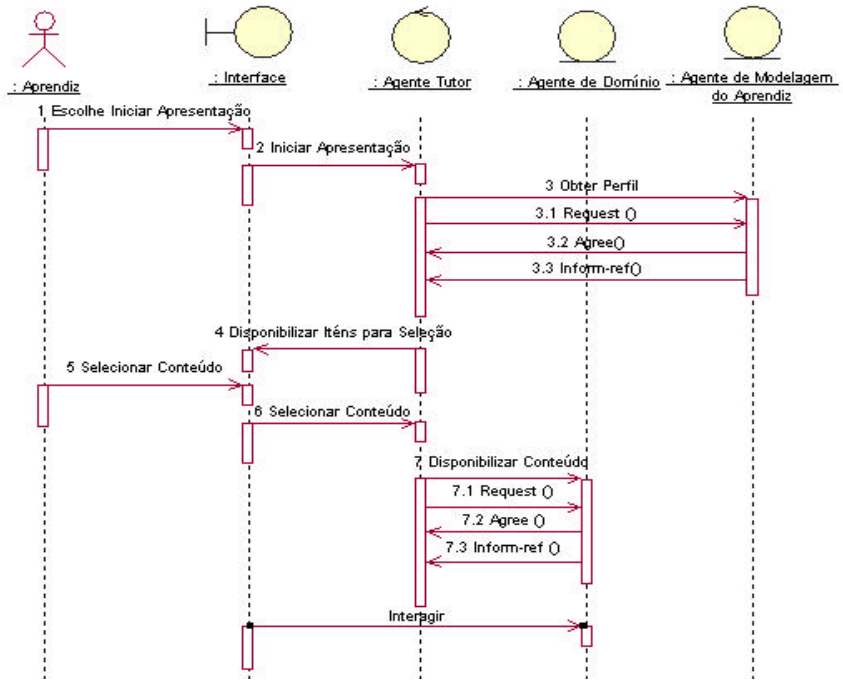

Figura 46 – Comunicação entre os agentes no Caso de Uso “Assistir Apresentação”

Deste modo, após o Agente Tutor receber uma solicitação do usuário indicando que ele deseja assistir algum conteúdo, o Agente Tutor solicitará ao Agente de Modelagem do Aprendiz o perfil deste aprendiz. O protocolo, neste caso de uso, funciona da seguinte maneira:

a) O Agente Tutor solicita ao Agente de Modelagem do Aprendiz que uma dada tarefa (que é a obtenção do perfil do aprendiz) seja realizada, utilizando para isto uma mensagem do tipo *request()* e passando como parâmetro o código do usuário (utilizado pelo Aprendiz para se logar no sistema);

b) se o Agente de Modelagem do Aprendiz aceitar essa solicitação, ele a responderá com uma mensagem do tipo *agree()* ao Agente Tutor;

c) o Agente de Modelagem do Aprendiz informará ao Agente Tutor, através de uma mensagem do tipo *inform-ref()*, o resultado da execução da tarefa solicitada, que será uma lista de itens cujo conteúdo o Aprendiz estará apto a assistir.

Esta lista, com todos os conteúdos possíveis de serem assistidos, é disponibilizada pelo Agente Tutor para o Aprendiz que selecionará qual conteúdo deseja assistir. Esta informação é repassada para o Agente Tutor que:

a) faz uma solicitação ao Agente de Domínio para que realize uma tarefa que é identificação do Agente de Domínio responsável pelo conteúdo e sua posterior disponibilização ao Aprendiz através de uma mensagem do tipo *request()*, cujo parâmetro é o código do conteúdo a ser assistido e o código de usuário do Aprendiz, que serve de identificação dentro do sistema;

b) o Agente de Domínio, ao receber essa solicitação e a aceitar, envia uma mensagem do tipo *agree()* ao Agente Tutor;

c) o Agente de Domínio informará ao Agente Tutor que o resultado da sua solicitação foi encontrado e o conteúdo encontrado passa então a ser disponibilizado para o Aprendiz que começa a assisti-lo.

6.3 Comunicação do Agente Tutor no Caso de Uso "Manter Base de Atividades"

A Fig. 47, do caso de uso "Manter Base de Atividades", mostra a seqüência de troca de mensagens que define um protocolo de comunicação entre o Agente Tutor e um Agente de Domínio.

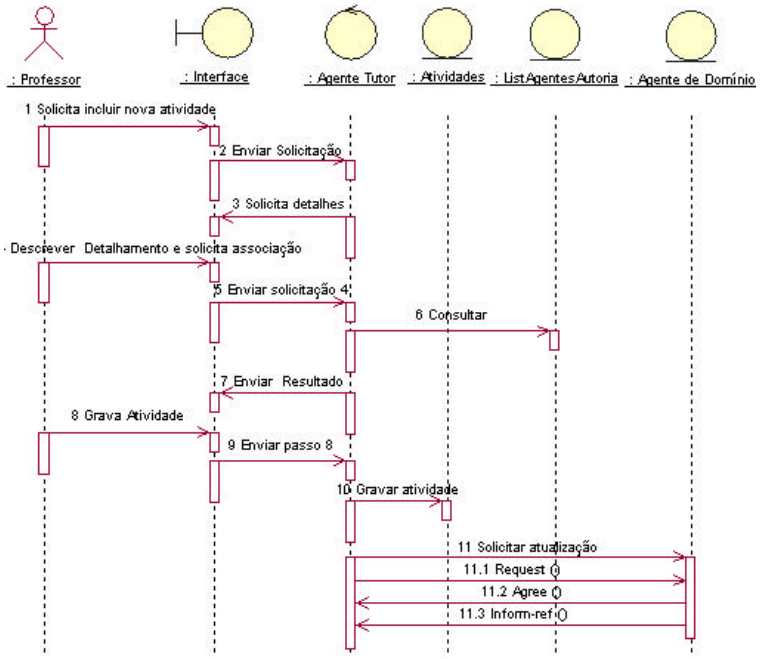

Figura 47 – Comunicação entre os agentes no Caso de Uso "Manter Base de Atividades"

Assim que, como já foi explicado anteriormente (ver seção 5.7), neste caso de uso o professor poderá fazer a manutenção (incluir, alterar e excluir) de uma atividade que esteja armazenada na Base de Atividades do sistema. Qualquer que seja o tipo de operação que esteja sendo feita na Base de Atividades, será necessário ao final que o Agente Tutor solicite ao Agente de Domínio que ele atualize as referências dele sobre os recursos/fontes de cada uma dessas atividades. Deste modo que:

a) o Agente Tutor solicita ao Agente de Domínio, através de uma mensagem do tipo *request()* que ele atualize a sua informação sobre a

atividade que foi modificada (através de uma alteração, por inclusão de uma nova atividade ou pela exclusão da mesma), enviando como parâmetros: o código de identificação da atividade que está sendo modificada, o tipo de operação realizada sobre ela (Inclusão/Alteração/Exclusão) e um outro parâmetro opcional que indicará os novos valores (recurso/fonte) caso uma inclusão ou alteração tenha sido realizada;

b) o Agente de Domínio, ao receber essa mensagem e a aceitando-a, envia uma outra mensagem do tipo *agree()* ao Agente Tutor confirmando a sua aceitação;

c) o Agente de Domínio passa então a fazer as alterações necessárias quanto ao recurso/fonte que foi modificado e ao final envia uma mensagem do tipo *inform-ref()* ao Agente Tutor para avisa-lo de que tal modificação já foi realizada.

6.4 Comunicação do Agente Tutor no Caso de Uso "Elaborar Lista de Atividades"

Na Fig. 48, que representa o caso de uso "Elaborar Lista de Atividades e descrito anteriormente na seção 5.19, é mostrada a seqüência das trocas de mensagens que definem um protocolo de comunicação entre o Agente Tutor e um Agente de Modelagem do Aprendiz.

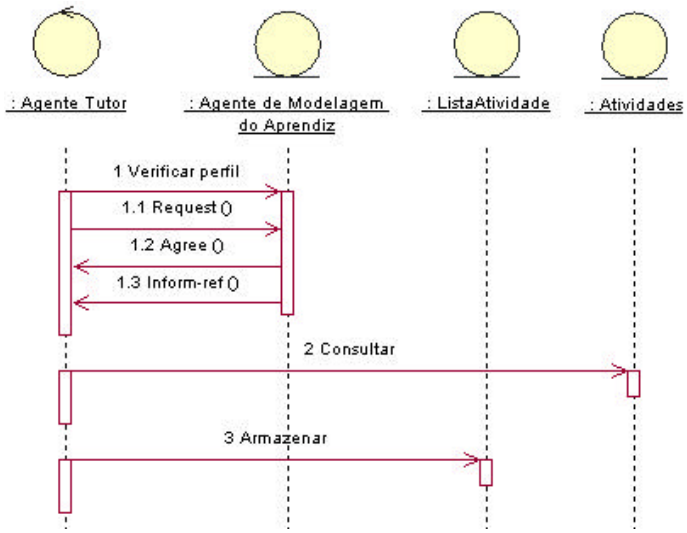

Figura 48 – Comunicação entre os agentes no Caso de Uso "Elaborar Lista de Atividades"

Esta comunicação entre os agentes dá-se no início do caso de uso onde:

a) o Agente Tutor envia uma mensagem do tipo *request()* ao Agente de Modelagem do Aprendiz, solicitando que o mesmo realize a tarefa de enviar o perfil de um determinado Aprendiz, para que o Agente Tutor possa construir uma Lista de Atividades para esse Aprendiz, enviando como parâmetros: o código de identificação do Aprendiz e o código do conteúdo/unidade de conhecimento referente ao tipo de atividade que deverá compor essa lista;

b) o Agente de Modelagem do Aprendiz ao receber essa solicitação e a aceitando, envia uma outra mensagem em resposta à solicitação do Agente Tutor que aceita realizar tal procedimento;

c) o Agente de Modelagem do Aprendiz ao obter as informações sobre o perfil do Aprendiz solicitado, envia uma mensagem do tipo *inform-ref()* ao Agente Tutor com os resultados obtidos.

6.5 Considerações Finais

Através da modelagem apresentada neste capítulo, pode-se observar o fluxo de mensagens ocorridas entre o Agente Tutor e os demais agentes do MATHNET e o protocolo de comunicação existente entre eles.

Os diagramas mostrados servirão como base para a definição e implementação do Agente Tutor e de sua comunicação com os demais agentes no MATHNET.

No próximo capítulo, será mostrada a definição e criação do Agente Tutor e dos demais Agentes da sociedade MATHNET, bem como sua implementação onde será demonstrada a comunicação baseada em troca.

7 IMPLEMENTANDO O AGENTE TUTOR E A COMUNICAÇÃO ENTRE AGENTES

Neste capítulo apresenta-se a implementação do Agente Tutor e de suas comunicações com os demais agentes, aplicando os modelos propostos neste trabalho.

7.1 Introdução

ZEUS é um toolkit que se caracteriza por ser um ambiente gráfico que fornece uma biblioteca de componentes de software e ferramentas que facilitam uma rápida construção e desenvolvimento de sistemas multi-agentes (ZEUS, 2001).

Esta ferramenta é utilizada nesse trabalho para a construção do Agente Tutor e dos demais agentes participantes por ser uma ferramenta de fácil instalação e manipulação, por possibilitar a construção de agentes que utilizam FIPA-ACL como linguagem de comunicação (adotada como modelo de comunicação neste trabalho) e por permitir a utilização de JAVA.

Embora, existam outras ferramentas para a construção de agentes, como AgentBuilder Pro (AGENTBUILDER, 2001), Comtec Agent Platform (FIPA, 2002), JATLite (JATLITE, 2002), etc., o toolkit ZEUS possui as características desejáveis delimitadas pela equipe do MATHNET para o desenvolvimento do processo de ensino-aprendizagem. Maiores detalhes sobre esta ferramenta, pode ser vista no Apêndice F.

7.2 Definição e criação de Agentes no ZEUS

7.2.1 Criando o Agente Tutor no ZEUS

Um dos primeiros passos nessa implementação foi a criação e definição dos agentes participantes, utilizando-se para isto a ferramenta ZEUS. Foram então criados (ver Fig. 49):

Figura 49 – Definições iniciais sobre os agentes na ferramenta ZEUS

a) As opções do Projeto (nome do arquivo, localização, etc.);

b) uma ontologia;

c) os agentes participantes; e,

d) as tarefas que eles podem realizar;

Em relação à construção de uma ontologia, no ZEUS, ela corresponde conseqüentemente à identificação dos **fatos** envolvidos e seus respectivos **atributos** em relação ao ambiente. Por ontologia pode-se referir como sendo a especificação formal e explícita de como representar objetos, conceitos e outras

entidades que são assumidas existirem em alguma área de interesse e as relações entre elas. Deste modo, ontologia pode ser descrita como uma idéia explícita, ou uma representação de alguma conceitualização (MONTESCO et al., 2001).

Por **fatos** refere-se a tudo aquilo que pode ser manipulado pelos agentes enquanto que, **atributos** são as características/propriedades que identificam esses fatos e na qual pode-se armazenar informações/dados a serem manipulados pelo sistema.

Desta forma, foram definidos os seguintes **fatos** e **atributos** (ver Fig. 50):

a) **Conteúdo programado** – armazena informações sobre a quantidade de vezes em que um determinado assunto foi assistido pelo aluno, quais os alunos que deverão assisti-lo e o grupo a que ele pertence;

b) **Conteúdo** – são todos os conteúdos possíveis de serem assistidos pelos alunos e que foram cadastrados pelo professor;

c) **Aluno** – refere-se a todos os alunos participantes do sistema e que encontram-se habilitados (através de um usuário e senha) a acessá-lo;

d) **Referência** – contém informações específicas sobre o conteúdo a ser ministrado, do tipo: a localização do arquivo que detém o referido conteúdo, o tipo de aplicativo que permite abrir este arquivo (word, excel,acrobat,etc.) e o conteúdo associado a ele;

e) **Grupo** – possui todos os possíveis grupos/áreas que os alunos podem participar. Cada aluno deve participar de pelo menos um grupo para que possa dar início a apresentação de algum conteúdo e também pode estar participando de mais de um;

f) **Grupo_Aluno** - permite identificar a qual grupo ou grupos cada aluno pertence, ou seja, quais as áreas que o aluno pode acessar. Ex: História, Geografia, Matemática, etc.

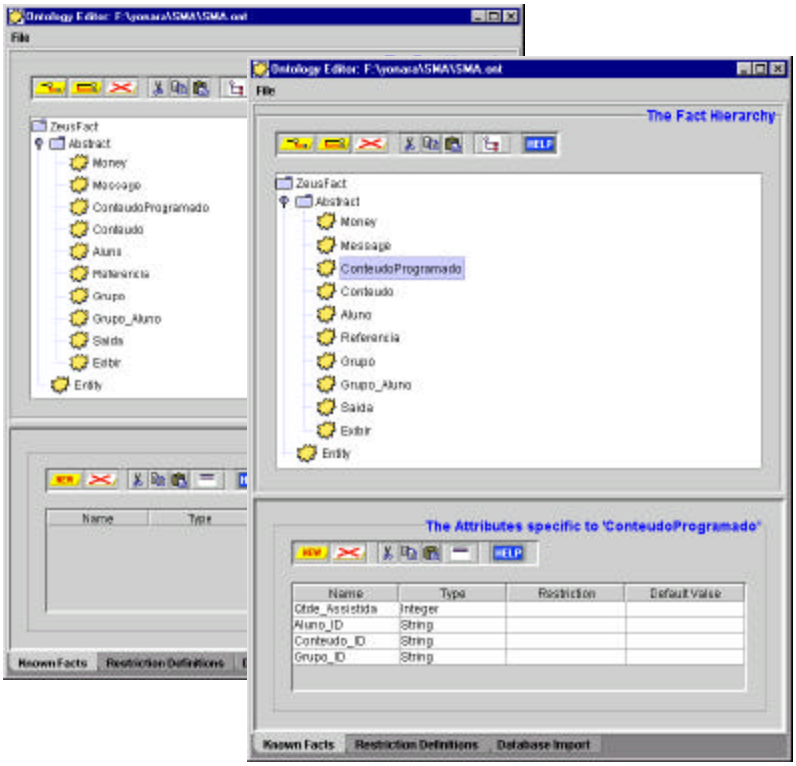

Figura 50 – Definição da Ontologia: fatos e atributos

Em resumo pode-se ver no Quadro 9, a seguir, os **fatos** e os **atributos** criados e definidos para o ambiente do Caso de Uso "Assistir Apresentação".

FATOS					
CONTEÚDO PROGRAMADO	CONTEÚDO	ALUNO	REFERÊNCIA	GRUPO	GRUPO_ALUNO
ATRIBUTOS					
Qtde_Assistida Aluno_ID Conteudo_ID Grupo_ID	Nome Conteúdo_ID	Nome Aluno_ID	Nome Localizacao Aplicativo Referencia_ID Conteúdo-ID	Grupo_ID Nome	Grupo_ID Aluno_ID Qtde_Programada

Quadro 9 – Fatos e atributos de "Assistir Apresentação"

É através do Painel de Definição de Agentes do ZEUS (ver Fig. 51 e 52) que a criação do Agente Tutor e dos demais agentes dessa sociedade é feita. Para

isto, foram definidos alguns recursos iniciais para este agente. Tais recursos são na verdade compostos dos **fatos** e das **instâncias**.

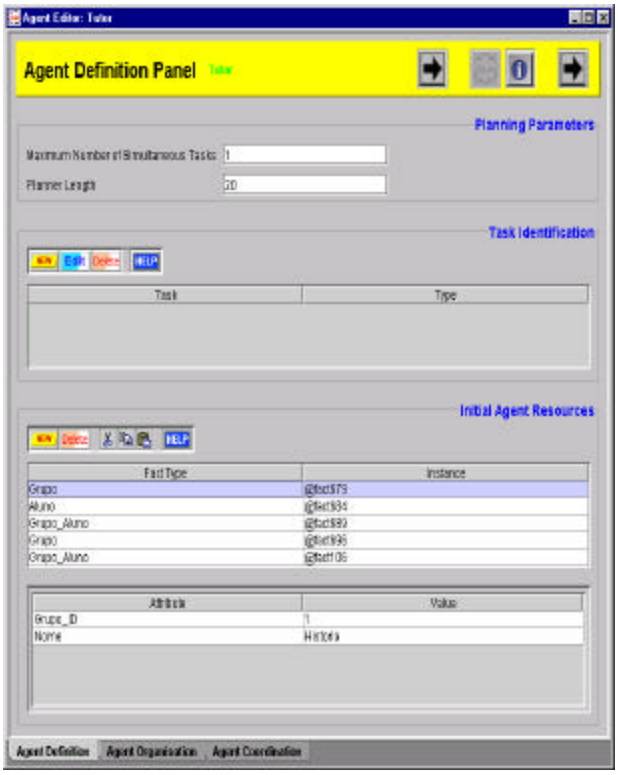

Figura 51 – Criação e definição do Agente Tutor no ZEUS

Os **fatos** são instanciados de forma que possam representar as informações necessárias aos agentes. Estas informações (recursos iniciais) formam a base de conhecimento do Agente Tutor. Desta forma, o Agente Tutor é então composto dos **fatos** Grupo, Aluno e Grupo Aluno.

Assim que, foi necessário "alimentar" tais **fatos** com informações que são utilizadas pela interface da implementação do Caso de Uso "Assistir Apresentação".

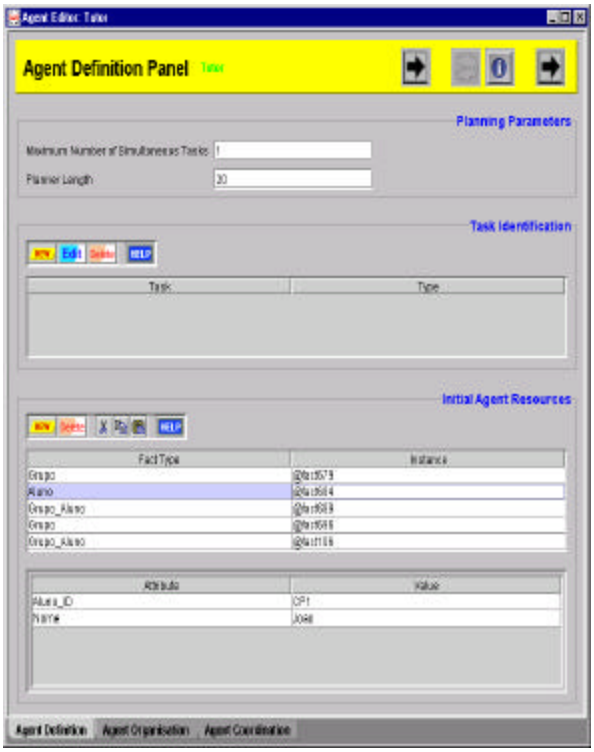

Figura 52 – Exemplo: definindo os recursos iniciais do Agente Tutor no ZEUS

Pode-se, então definir alguns recursos iniciais para o Agente Tutor conforme o Quadro 10, abaixo:

RECURSOS INICIAIS – Agente Tutor					
GRUPO		ALUNO		GRUPO_ALUNO	
GRUPO_ID	1	ALUNO_ID	CP1	ALUNO_ID	CP1
NOME	História	NOME	João	GRUPO_ID	1
GRUPO_ID	2	ALUNO_ID	CP2	QTDE_PROGRAMADA	1
NOME	Geografia	NOME	Maria	ALUNO_ID	CP1
				GRUPO_ID	2
				QTDE_PROGRAMADA	2
				ALUNO_ID	CP2
				GRUPO_ID	1
				QTDE_PROGRAMADA	4

Quadro 10 – Alguns Recursos iniciais do Agente Tutor no ZEUS

Tais Recursos iniciais podem ser expandidos com o intuito de aumentar a base de conhecimento do Agente Tutor de forma a abranger o objetivo desejado.

Em relação às tarefas a serem realizadas pelo Agente Tutor foi definido que seriam implementadas via código sem que houvesse uma prévia definição no ZEUS.

É importante ressaltar que os fatos, atributos e instâncias estruturados nesta seção, foram baseados nas características e o papel do Agente Tutor explicados na seção 4.2.1.

7.2.2 Criando os demais agentes participantes no ZEUS

Foram delimitados e criados mais dois agentes, além do Agente Tutor, que são necessários para auxiliar a comunicação entre os agentes no MATHNET, o ModelAprendiz e o Domínio.

Em relação ao Agente de Domínio, existirá, na verdade, um agente de domínio específico que é responsável por deter todo o conhecimento (conteúdo a ser assistido) sobre um determinado assunto (domínio). Assim que, existirão vários agentes de Domínio (Domínio, Domínio2, etc.) (ver Fig. 53) um para cada área de conhecimento (COSTA, 2002).

Para cada um dos agentes foram definidas algumas tarefas básicas que podem ser realizadas e também um conjunto de recursos iniciais (informações/dados manipulados pelo sistema e relacionados com os fatos e atributos citados anteriormente).

Dentre as tarefas básicas a serem executadas pelos agentes de Domínio estão a de Fornecer o Conteúdo ao aprendiz e a de Exibir as Referências (local onde se encontra o arquivo contendo o conteúdo especificado). Cada Agente de Domínio pode, para cada conteúdo, possuir uma ou mais referências sobre um mesmo assunto.

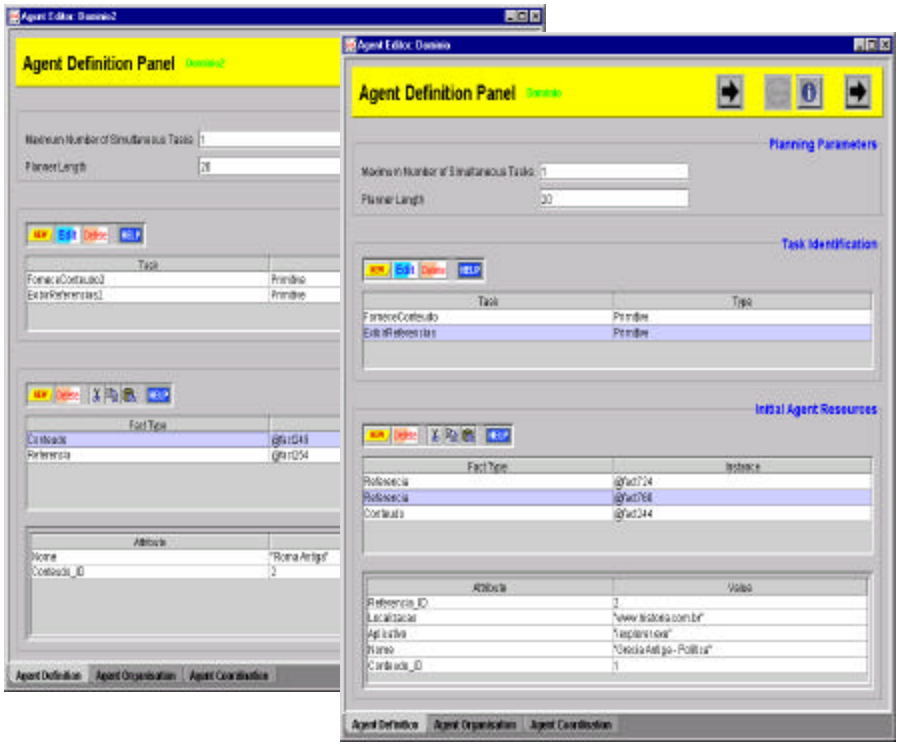

Figura 53 – Criação e definição de alguns Agentes de Domínio no ZEUS

Foram também definidos alguns recursos iniciais para os agentes de Domínio como mostrado no Quadro 11, a seguir.

RECURSOS INICIAIS – Agente de Domínio			
REFERÊNCIA		CONTEÚDO	
REFERÊNCIA_ID	2	NOME	“Grécia Antiga”
LOCALIZACAO	“www.historia.com.br”	CONTEUDO_ID	1
APLICATIVO	“iexplorer.exe”		
NOME	“Grécia Antiga – Política”		
CONTEUDO_ID	1		

Quadro 11 – Alguns Recursos iniciais de um Agente de Domínio no ZEUS

Para o Agente de Modelagem de Aprendiz, criado no ZEUS com o nome de ModelAprendiz (ver Fig. 54), também foram definidos alguns recursos iniciais básicos.

Em relação às tarefas, estas não foram previamente definidas no ZEUS, mas são diretamente implementadas via código.

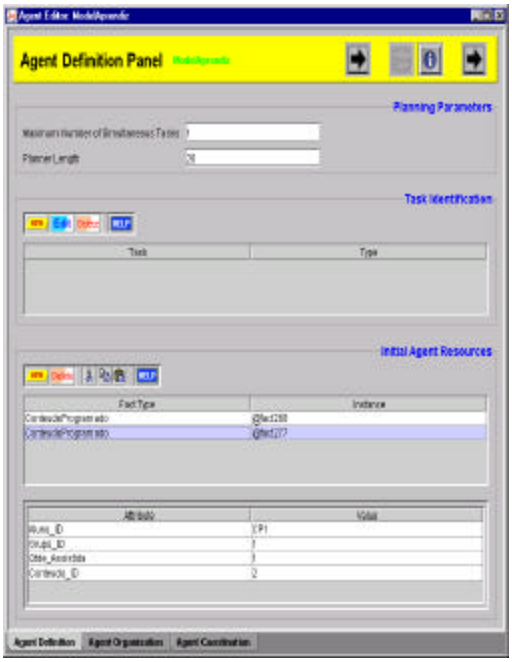

Figura 54 – Criação e definição do Agente de Modelagem do Aprendiz no ZEUS

Um exemplo dos recursos iniciais disponíveis para o Agente de Modelagem do Aprendiz pode ser visto no Quadro 12, a seguir.

RECURSOS INICIAIS – Agente de Modelagem do Aprendiz	
CONTEUDOPROGRAMADO	
ALUNO_ID	CP1
GRUPO_ID	1
QTDE_ASSISTIDA	1
CONTEUDO_ID	1
ALUNO_ID	CP1
GRUPO_ID	2
QTDE_ASSISTIDA	1
CONTEUDO_ID	2

Quadro 12 – Alguns Recursos iniciais do Agente de Modelagem do Aprendiz no ZEUS

Após a definição desses agentes com suas características a implementação do Caso de Uso “Assistir Apresentação” será descrita na próxima seção.

7.3 Implementando o Caso de Uso “Assistir Apresentação”

Conforme já citado, o objetivo de um processo de comunicação é trocar informações. Quando essa comunicação é descrita nos termos de envio e recebimento de mensagens, as partes envolvidas devem ser capazes de inferir o que o remetente intencionava quando a proferiu, ou seja, se foi solicitada alguma informação ou se estava comandando algum ato. Caso não exista alguma forma de estruturação nestas mensagens, esta inferência torna-se um processo ineficiente.

Sendo assim, as mensagens devem estar contidas por restrições formais e estruturadas para a facilidade de sua interpretação. Por exemplo, distinguindo tipos de mensagens nas quais a intenção do remetente possa ser imediatamente reconhecida pela mensagem em si.

Desta forma, é que o modelo de comunicação escolhido foi baseado em troca de mensagens, através do uso de performativas e da linguagem de comunicação de agentes FIPA-ACL.

Nesta seção, apresenta-se a implementação do Caso de Uso “Assistir Apresentação” que melhor representa a comunicação entre o Agente Tutor e os demais agentes, pois através neste Caso de Uso tem-se a participação de vários Agentes de Domínio e também do Agente de Modelagem do Aprendiz, fazendo dele um ótimo ambiente de simulação do processo de ensino-aprendizagem.

Para isto foi escolhida a linguagem Java por esta possuir orientação à objetos, multithreading, portabilidade e suporte à rede (DEITEL et al., 2001; ECKEL, 1998; SILVA et al., 1997). Neste caso, utiliza-se para isto a versão 1.2.2 do JDK e o Jbuilder versão 3. Tal implementação pode ser vista, em maiores detalhes, no Apêndice G.

Foi feita uma implementação que utiliza uma interface bastante prática e funcional e que é baseada nos agentes criados e definidos no ZEUS e descritos anteriormente.

A tela inicial do programa pode ser vista a seguir, através da Fig. 55.

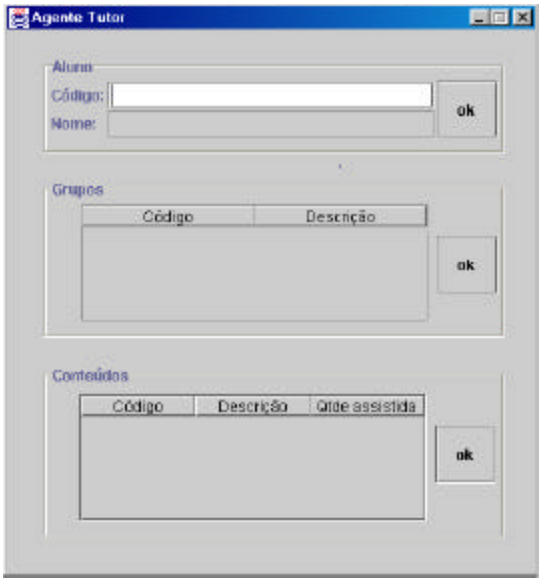

Figura 55 – Tela do programa Assistir Apresentação

O primeiro passo é a identificação do aluno como usuário do sistema. Para isso o aluno deve informar o seu código. Este código é enviado pelo Agente Tutor para o Agente de Modelagem do Aprendiz.

É o Agente de Modelagem do Aprendiz que, através deste código, identifica o aluno no sistema, ou seja, o seu perfil (nome, grupo que ele pertence, etc.) e repassa para o Agente Tutor essas informações sobre o aluno listando-as na tela (ver Fig. 56). A parte desta implementação pode ser vista na Fig. 57.

Isto significa que um aluno pode estar participando de mais de um Grupo (Área de Conhecimento) e que, portanto, ele deve escolher em que Grupo ele deseja estar interagindo naquele momento.

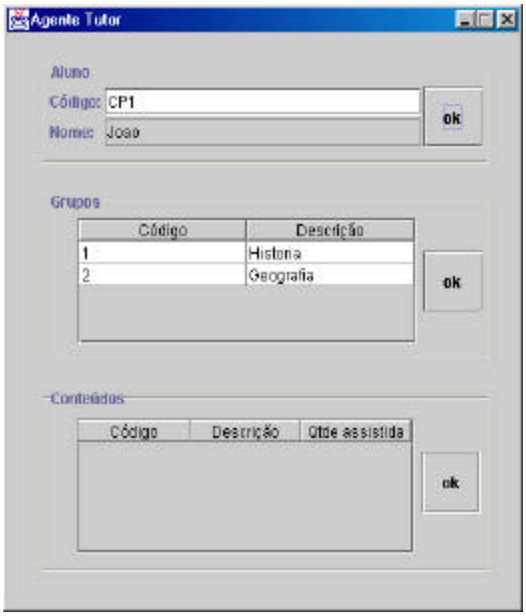

Figura 56 – Grupos para o Aluno João

Nesta implementação não houve a preocupação com relação à segurança. Por isso não é pedida a senha e também não é feita a validação desta. Isto é de responsabilidade do Caso de Uso “Iniciar Sessão”.

```
int i=0;
while(i<fatos.length)
{
                //fato=(Fact)enum.nextElement();

                if(fatos[i].getValue("Aluno_ID").equals(Aluno) &&
fatos[i].getValue("Grupo_ID").equals(Grupo_ID))
                {
                   /*Performative perf=new Performative("inform");
                   perf.setReplyWith("Informacoes_Aluno");
                   perf.setReceiver(event.getSender());
                   //perf.setContent(fato.getValue("NomeConteudo"));
                   perf.setContent(fato.getValue("data" + fato));
                   mbox.sendMsg(perf);*/

        fato2 = agent.OntologyDb().getFact(Fact.FACT,"Conteudo");
        fato2.setValue("Conteudo_ID",fatos[i].getValue("Conteudo_ID"));
                    Goal g = new Goal(agent.newId("goal"), fato2,
                (int)(agent.now())+6, 0.1,agent.whoami(),(double)(agent.now()+3));

                   agent.Engine().achieve(g);
```

Figura 57 – Agente de Modelagem do Aprendiz para mostrar os possíveis conteúdos

Para dar prosseguimento a este processo o aluno deve selecionar um dos Grupos, por exemplo: História e pressionar o botão Ok. Neste momento, as

informações sobre o aluno (seu código) e o Grupo desejado são repassados para o Agente Tutor que localiza os Agentes de Domínio que contém este Grupo e disponibiliza na tela todos os conteúdos programados para ele assistir e que foram definidos nos **fatos** e **instância** no ZEUS (ver Fig. 58). Deste modo, informações como o código do Conteúdo que o aluno deve assistir e a quantidade de vezes em que este aluno assistiu tal conteúdo.

Tais informações possibilitam que aluno possa saber quais os conteúdos que ele assistiu e o que falta assistir. Ou seja, tudo o que foi programado para ele fazer dentro deste Grupo.

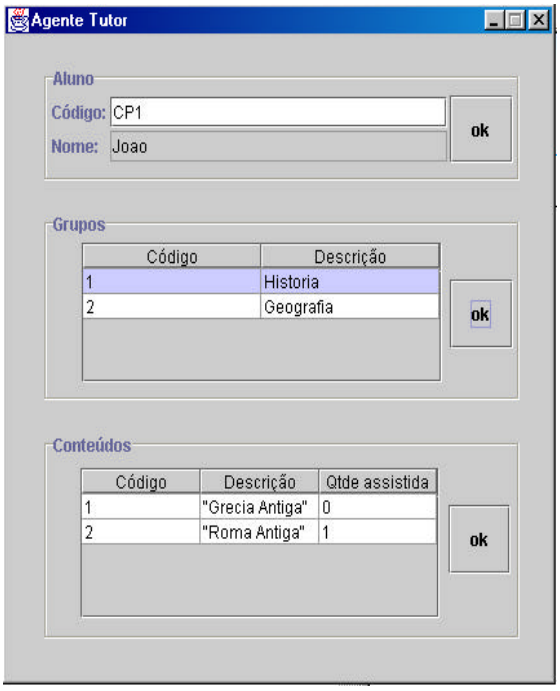

Figura 58 – Conteúdos programados para o aluno João no grupo História

O passo seguinte é selecionar qual dos conteúdos listados o aluno deseja interagir naquele momento. Desta forma, o aluno seleciona o conteúdo desejado e pressiona o botão Ok, por exemplo, sobre a opção Grécia Antiga. Neste momento, em uma segunda tela, as referências sobre aquele conteúdo devem ser mostradas

(ver Fig. 59). Pelas referências o arquivo ou página na internet pode ser localizada pelo aluno.

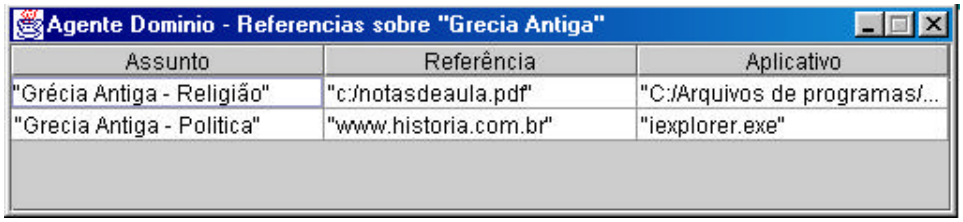

Figura 59 – Referências para João no Grupo História de Grécia Antiga

Para exibição das Referências o seguinte trecho de código a seguir foi feito (ver Fig. 60). Para cada Agente de Domínio este código foi implementado e mostra todas as referências.

```
public void exibirReferencias()
{
        System.out.println("-------dentro de exibirReferencias--------");

        Fact[] fatos2;
        fatos2=agent.ResourceDb().all("Conteudo");
        TabelaReferencias tabela=new TabelaReferencias("Agente " + agent.whoami()
+ " - Referencias sobre " + fatos2[0].getValue("Nome"));

        int i=0;
        Fact[] fatos;
        fatos=agent.ResourceDb().all("Referencia");
        while(i<fatos.length)
        {

tabela.insere_dados(fatos[i].getValue("Nome"),fatos[i].getValue("Localizacao"),fatos[i].getValue("Aplicativo"))
;
          i++;
    }
    tabela.pack();
    tabela.setVisible(true);
        Performative perf=new Performa tive("inform");
        perf.setReplyWith("Referencias_ok");
        perf.setReceiver("Tutor");
        perf.setContent("");
        mbox.sendMsg(perf);

}
```

Figura 60 – Código para Exibir as referências

Como existem vários Agentes de Domínio, um para cada conhecimento/conteúdo, logo teve que ser implementado uma forma de localizar todos os conteúdos possíveis de serem assistidos pelo aluno a partir do momento

em que foi definido qual grupo que ele deseja assistir o conteúdo. Assim que, foi criado uma espécie de facilitador para localizar esses vários agentes de Domínio e listar o conteúdo de cada um deles, para que os mesmos pudessem ser visualizados corretamente pelos alunos.

O aluno pode a qualquer momento desistir de Assistir a Apresentação. Os conteúdos possíveis de serem assistidos podem estar localizados em qualquer lugar, desde que esteja corretamente indicada a sua localização física.

7.4 Considerações finais

Neste capítulo, mostrou-se a criação do Agente Tutor, dos Agentes de Domínio e do Agente de Modelagem do Aprendiz baseada nas características e responsabilidades de cada um deles. Utilizando-se para isto a ferramenta ZEUS.

Alem disso, foi mostrada a implementação do Caso de Uso “Assistir Apresentação” para formalizar a comunicação entre agentes. Este Caso de Uso representa bem a comunicação entre agentes pelo fato de que existem outros agentes da SA MATHNET participando e por ser de fácil entedimento. Desta forma, o modelo de comunicação proposto neste trabalho, baseado em troca de mensagens e utilizando FIPA-ACL pôde ser mostrado.

8 CONCLUSÃO

A Educação à Distância por apresentar-se na esfera pedagógica como mais uma opção metodológica, que possui sua relevância e características próprias, impõe a necessidade de novas aprendizagens, possibilitando inovação nos procedimentos de ensino e a introdução de diferentes tecnologias no processo educacional (rádio, TV e computadores) como forma de atender às novas demandas deste surgidas deste processo de aprendizagem.

Mas é a adoção dessas novas tecnologias, principalmente dentro da Educação à Distância, que vêm provocando mudanças no processo de ensino-aprendizagem, e conseqüentemente, questionando os métodos didáticos convencionais e sua eficiência pedagógica (baseados no uso quase que exclusivo de sala de aula e com a exigência da presença física e simultânea de professores e alunos) estando a exigir uma redefinição no papel do professor e em sua interação com os alunos.

Dentre as novas tecnologias que estão sendo utilizadas pela Educação à Distância é a utilização do computador que tem demonstrado ser um valioso instrumento nesse modelo e o mais revolucionário. Uma das formas de utilizá-lo é através de softwares educacionais. Esses softwares têm por objetivo auxiliar o professor no processo de ensino-aprendizagem, fazendo com que o mesmo tenha ao seu dispor valiosos recursos para ajudá-lo com seus alunos e auxiliá-los no aprendizado dos mais diversos conteúdos.

Assim sendo, pode-se afirmar que neste caso, o ensino-aprendizagem computadorizado consiste basicamente de quatro “ingredientes”: o computador, o software educacional, o professor (capacitado para usar o computador como ferramenta educacional) e o aluno.

São esses quatro “ingredientes” que se bem utilizados e em sintonia favorecem o processo de ensino-aprendizagem, através da cooperação existente entre todas as partes envolvidas, pois é através da interatividade destes quatro elementos que será criado, para o professor e os alunos, um ambiente favorável para construção/produção do conhecimento de forma dinâmica e lucrativa para ambas as partes.

É com esse objetivo que o MATHNET foi projetado, buscando-se para isso integrar as mais recentes tecnologias de software e hardware disponíveis no mercado para construir tal ambiente. A partir disso, uma sociedade de agentes foi projetada e definida para proporcionar as facilidades e flexibilidades requeridas, bem como a independência e a interatividade entre as partes envolvidas.

Sobretudo, procurou-se aqui interligar os demais agentes já especificados em trabalhos citados anteriormente com o intuito de mostrar como essa Sociedade de Agentes trabalha em conjunto para obter o resultado pretendido, deixando de lado a visão isolada sobre o Agente Tutor e enfocando principalmente a interação dele com os demais agentes da sociedade.

Conforme destacado anteriormente na introdução em relação aos objetivos especificados na seção 1.1, o presente trabalho conseguiu atingir os seguintes objetivos propostos:

a) definiu os agentes necessários e suficientes no MATHNET de modo que o processo de ensino-aprendizagem fosse realizado com êxito e não houvesse redundância e nem ambigüidades nas tarefas realizadas pelos Agentes. Tal SA foi concebida pela equipe do Projeto MATHNET, baseada em pesquisas e sob orientação especializada;

b) Identificou e delimitou os principais Casos de Uso do MATHNET nos quais o Agente Tutor participa e que se encontram descritos na seção 4.4. Tais casos foram obtidos através da análise de requisitos utilizado formulário próprio do MATHNET(ver Apêndice E) e os quais foram modelados utilizando a UML e posteriormente implementados;

c) Identificou, através dos Casos de Uso, os agentes participantes a partir da definição dos papéis de cada um no sistema conforme descrito na seção 2.3. Tais papéis foram definidos ao longo da concepção do projeto MATHNET e fruto de intensas pesquisas em artigos, livros e da concepção inicial do sistema;

d) Definiu as colaborações (originadas das interações intragrupos e intergrupos, descritas na seção 4.2.2) e os protocolos de comunicação e interação do Agente Tutor com os demais agentes participantes desses Casos de Uso (descritos no Capítulo 6). Comunicação essa baseada na troca de mensagens entre os agentes utilizando FIPA-ACL, que é fruto das interações entre os agentes e que serve como suporte à cooperação existente nessa sociedade.

Sugerem-se as demais implementações da comunicação existente entre o Agente Tutor e os demais agentes, bem como dos outros agentes que compõe a SA do MATHNET, utilizando o modelo de comunicação aqui proposto. E desta forma, "interligar" os agentes já modelados em outras dissertações, como as que apresentaram o Agente de Modelagem do Aprendiz (SERRA, 2001), os Agentes de Domínio (COSTA, 2002) e Agentes de Busca (NUNES, 2001).

Espera-se ter contribuído com esse trabalho não apenas para auxiliar na operacionalização do sistema MATHNET, mas também como uma forma de

contribuir para a especificação de uma SA e para a construção de um ambiente que implemente os aspectos favoráveis para o processo de ensino-aprendizagem computadorizado, de modo a se ter no meio educacional mais um ambiente que contemple com sucesso este processo.

REFERÊNCIAS

ABEL, Mara. **Introdução à Linguagem de Programação Common Lisp**. Porto Alegre, 1995. UFRS. Instituto de Informática. Disponível em: <http://www.ec.ucdb.br/pub/windows9x/lisp/lisp.pdf>. Acesso em: 17 ago. 2001.

AGENTBUILDER. [2001]. Disponível em: <www.agentbuilder.com>. Acesso em: 25 jul. 2001.

BITS AND PIXELS. Desenvolvido por Bits and Pixels Incorporation, 1996. Disponível em: < http://www.bitpix.com/agtdemo/agtfctry/agtfctry.htm>. Acesso em: 11 mar. 2002.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML, guia do usuário**. 5.ed. Rio de Janeiro: Campus, 2000. 472 p.

BORGES, Hélder Pereira. **Assiste de Resolução de Problemas para o sistema MATHNET**. 2002. 124 f. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Maranhão, São Luís.

BREITER, P.; SADEK, M. D. **A rational agent as a kernel of a cooperative system: implementing a Logical Theory of Interaction**. In: EUROPEAN CONFERENCE ON ARTFICIAL INTELLIGENCE (ECAI) – WORKSHOP AGENT THEORIES, ARCHITECTURES AND LANGUAGES, Budapeste. 1996. p. 261-276.

BURSTEIN, Mark. In: **The ARPA/Rome Lab Knowledge-Based Planning and Scheduling Initiative Workshop**. San Francisco: Morgan Kuafmann Publishers, 1994. Disponível em: <http://www.computer.org/intelligent/ex1995/x1toc.htm> Acesso em : 12 mai. 2001.

CLIMATE.[1999]. Criação:Projeto MONTAGE e Consórcio CLIMATE. Disponível em: <http://www.fokus.gmd.de/research/cc/ecco/climate/climate.phtml>. Acesso em: 15 jun. 2001.

COHEN, Philip R.; LEVESQUE, Hector J. **Communicative actions for artificial agents**. In: FIRST INTERNATIONAL CONFERENCE ON MULTI-AGENT SYSTEMS (ICMAS-95). California: AAAI Press, 1995. p.65-72.. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/7240/http:zSzzSzwww-internal.cse.ogi.eduzSzCHCCzSzPublicationszSz..zSzPaperszSzsharonPaperzSzourkqml8.pdf/cohen95communicative.pdf> Acesso em: 4 out. 2001.

COLLIS, Jaron; NDUMU, Divine. **The Zeus Agent Building Toolkit: ZEUS Technical Manual**. Release 1.0, set. 1999. Disponível em: <http://www.deinf.ufma.br/~rgirardi/Teaching/Disciplinas/IA/planoIA.htm>. Acesso em: 18 set. 2001.

COMTEC Agent Plataform. Disponível em: <http://fipa.comtec.co.jp/ap/>. Acesso em 20 mar. 2002.

COSTA, Evandro de Barros. **Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura**. 1997. 133 f. Tese (Doutorado em Processamento da Informação) – Universidade Federal da Paraíba. Campina Grande.

COSTA, Nilson Santos. **Modelagem e construção de uma Ferramenta de Autoria para um Sistema Tutorial Inteligente**. 2002. 171 f.. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Maranhão. São Luís.

COUTINHO, Luciano Reis; LABIDI, Sofiane; SERRA JR, Gentil; TEIXEIRA, Cenidalva. **A Learner Modeling Agent for Cooperative Learning**. In: SIMPOSIUM BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE). Maceió (AL). 2000.

COUTINHO, Luciano Reis; SERRA JR., Gentil; LABIDI, Sofiane. **Group formation for cooperative learning: a genetic algorithm approach**. In: IASTED International Conference. Calgary. 2001.

COUTINHO, Luciano Reis; SERRA JR., Gentil; LABIDI, Sofiane, NUNES, Helena Maria Pereira. **Learner Modeling and Groups Formation in MATHNET**. 1999

DARPA. Disponível em: <http://www.darpa.mil/>. Acesso em: 20 jun. 2001.

DEITEL, Harvey. M.; DEITEL, P.J. **Java, como programar**. 3.ed. Porto Alegre: Bookman, 2001. 1201 p.

ECKEL, B. **Thinking in Java**. Prentice Hall, 1998. 1098 p. ISBN: 0136597238.

FININ, Tim; LABROU, Yannis; MAYFIELD, James. **KQML as an Agent Communication Language**. In ``Software Agents", AAAI/MIT Press, 1997. p. 291-316. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/5548/http:zSzzSzwww.autopubs.comzSz~dhudekzSz.zSzdocszSzkqmlAsAcl.pdf/finin95kqml.pdf> Acesso em: 5 out. 2001.

FERREIRA, Jeane Silva. **Concepção de um Ambiente Multi-Agentes de Ensino Inteligente Integrando o Paradigma de Aprendizagem Cooperativa.** 1998. Dissertação (Mestrado em Ciência da Computação) - Universidade Federal do Maranhão. São Luís.

FERREIRA, Jeane Silva; LABIDI, Sofiane. **Modelagem do aprendiz baseado no Paradigma de Ensino Cooperativo**. Departamento de Informática - UFMA. São Luís, 1998.

FININ, Tim; FRITZSON, Richard; McKAY, Don; McENTIRE, Robin. **KQML as an Agent Communication Language**. 8 f. In: THIRD INTERNATIONAL CONFERENCE ON INFORMATION AND KNOWNLEDGE MANAGEMENT (CIKM'94). Maryland: ACM Press, 1994. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/6693/http:zSzzSzwww.csee.umbc.eduzSzpubzSzARPAzSzkqmlzSzpaperszSzkqmlacl.pdf/finin94kqml.pdf > Acesso em: 10 out. 2001.

FIPA Specification Repository. Disponível em: <http://www.fipa.org/repository>. Acesso em: 30 mar. 2002. Documentos nº XC00037H e XC00061E.

FIPA Specification Repository. Disponível em: <http://www.fipa.org/>. Documento: XC00018A. Acesso em: 23 jul. 2001.

FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 329 p.

GENESERETH, Michael R.; KATCHPEL, Steven P. **Software agents. Communications of the ACM**, 37(7):48-53, 147, 1994. Disponível em: <http://logic.stanford.edu/sharing/papers/agents.ps> Acesso em: 27 ago. 2001.

GALAN, Alan K. **JiVE: JAFMAS integrated Visual Environment**. 2000. 88 f. Dissertação (Mestrado) – University of Cincinnati. Ohio. Disponível em: <http://www.ececs.uc.edu/~abaker/JiVE/PDF/Thesis.pdf>. Acesso em: 22 ou. 2001.

GREAVES, Marck. **Communication and conversation in FIPA agents**. In jul. 2000. Disponível em: <http://fipa.umbc.edu/18/greaves.pdf>. Acesso em: 10 jun. 2001.

HAIDT, Regina Célia C.. **Curso de didática geral**. São Paulo: Ática, 1999. ISBN: 8508046286

HÜBNER, Jomi Fred. **Migração de agentes em sistemas multi-agentes abertos**. 1995. 131 f. Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Porto Alegre. Disponível em: <http://www.inf.furb.br/~jomi/publicacoes/dissertacao/dissertacao.pdf>. Acesso em: 22 out. 2001.

HUHNS, M.N.; STEPHENS, L.M. **Multiagent systems and society of agents.** In G. Weiss, editor, Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence. MIT Press, 1999.

INTELLIGENT Agents. Disponível em: <http://www.doc.ic.ac.uk/~nd/surprise_95/journal/vol2/mkh/article2.html> Acesso em: 14 jun. 2002.

JATLITE. Disponível em: < http://java.stanford.edu/>. Acesso em: 10 mar. 2002.

JENNINGS, N. R. **Coordination tecniques for distributed artificial intelligence**. In: O´HARE, G.M.P.; JENNINGS, N. R. (Eds.). Foundations of distributed artificial intelligence. New York: Jonh Wiley & Sons, 1996. p. 187-210.

KIRCHHOF, Lisiane T.; FONTES, Roberto D. **Definição de uma Arquitetura Mulitiagentes para apoio ao Processo de Ensino-Aprendizagem.** SBIE 1999. Disponível em: <http://www.inf.pucrs.br/~giraffa/sbie99/kirchhof.pdf>. Acesso em: 24 ago. 2001

KIF Knowledge Interchange Format. Disponível em: <http://www.ime.usp.br/~eudenia/jaia/texto/node54.html>. Acesso em: 15 mar. 2002.

LABIDI, Sofiane; FERREIRA, Jeane Silva. **Technology-assisted instruction applied to cooperative learning: the SHIECC project**. In: IEEE INTERNATIONAL CONFERENCE FROTIERS IN EDUCATION (FIE'98). Arizona. 1998. p.4-7.

LABIDI, Sofiane; SILVA, Josenildo C.; COUTINHO, Luciano R.; COSTA, Nilson. **Agent Architecture for Cooperative Learning Environment**. In: SOCIEDADE BRASILEIRA DE INFORMÁTICA E EDUCAÇÃO (SBIE). Anais do SBIE'2000, Alagoas: 2000a.

LABIDI, Sofiane; SILVA, Josenildo C.; COUTINHO, Luciano R. **MATHNET: Agent-Based Tutoring System for Supporting Cooperative and Distant Learning**. In: COMPUTERS AND ADVANCED TECHNOLOGY IN EDUCATION (CATE). Cancun (Mexico): 2000b.

LABROU, Yannis. **Semantics for an Agent Communication Language**. 1996. Dissertação (PhD em Ciência da Computação) - University of Maryland Graduate School. Baltimore (Maryland). Disponível em: <http://www.cs.umbc.edu/~jklabrou/publications/atal97.14pp.pdf>. Acesso em: 22 jul. 2001.

LABROU, Yannis; FININ, Tim . **Semantics and conversation for an agent communication language**. In: FIFTEENTH INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE (IJCAI). Nagoya. 1997a. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/3864/http:zSzzSzwww.cs.umbc.eduzSz~jklabrouzSzpublicationszSzlnai98.pdf/labrou96semantics.pdf > Acesso em: 14 out. 2001.

LABROU, Yannis; FININ, Tim. **A proposal for a new KQML Specification**. Baltimore: 1997b. Computer Science and Electrical Engineering Department (University of Maryland Baltimore County - UMBC). Disponível em: <http://www.csee.umbc.edu/~jklabrou/publications/tr9703.pdf > Acesso em: 22 out. 2001.

LABROU, Yannis; FININ, Tim; PENG, Yun. **Agent communication languages: The current landscape**. Intelligent Systems, Vol. 14, No. 2, March/April 1999, IEEE Computer Society. Disponível em: <http://www.csee.umbc.edu/~jklabrou/publications/ieeeIntelligentSystems1999.pdf>. Acesso em: 20 out. 2001.

LABROU, Yannis; FININ, Tim. **Towards a standard for an Agent Communication Language**. 1997c. American Association for Artificial Intelligence Fall Symposium on "Communicative Actions in Humans and Machines", Cambridge, MA. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/3864/http:zSzzSzwww.cs.umbc.eduzSz~jklabrouzSzpublicationszSzfss97.pdf/towards-a-standard-for.pdf> Acesso em: 2 nov. 2001.

LARMAN, Craig. **Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos**. Porto Alegre: Bookman, 2000. 492p.

LEE, L. C.; NWANA H. S.; NDUMU, D. T.; DE WILDE, P. **The stability, scalability and performance of multi-agent systems**. BT Technology Journal, 16v. N.3, 1998. 94-103p. . Disponível em: < http://citeseer.nj.nec.com/cache/papers/cs/17902/http:zSzzSzwww.labs.bt.comzSzprojectszSzagentszSzpublishzSzpaperszSzbttj98-scalability.pdf/lee98stability.pdf>. Acesso em: 15 fev. 2002.

MASUOKA, Ryusuke; SATO, Akira; GENESERETH, Michael R. **KIF as FIPA-CLL Content Language – An antology for actions**. Stanford University: jun. 1999. Disponível em: <http://lettuce.ms.u-tokyo.ac.jp/~masuoka/papers/kif-in-fipa-cll.pdf> Acesso em: 15 mar. 2002.

MONTESCO, Carlos A. Estombelo; MOREIRA, Dilvan de Abreu. **Uma linguagem de comunicação para agentes na Internet baseada em Ontologias**. Universidade de São Paulo (USP-São Carlos): nov. 2001. Disponível em: <http://www.icmc.sc.usp.br/~percival/ucl.pdf>. Acesso em 10 dez. 2002.

MURRAY, Tom. **Authoring Intelligent Tutoring Systems: in the analysis of the state of the art**. International Journal of Artificial Intelligence in Education, 10, 98-129. 1999.

NUNES, Helena Maria Pereira. **Serviço de busca baseado em agentes móveis para o ambiente mathnet de ensino cooperativo computadorizado**. 2001. 135 f.. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Maranhão. São Luís.

NUNES, Helena Maria Pereira; LABIDI, Sofiane. **Increasing Interactions within MATHNET using Mobile Agents**. In: 4th WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS (SCI). Orlando. 2000.

NUNES, Helena Maria Pereira; LABIDI, Sofiane. **Mobile Agents for informatio extraction in MATHNET System**. In: IEEE International Conference Frontiers in Education FIE-2002. November 6-9, 2002. Boston, USA.

O'BRIEN P. D.; NICOL, R. C. **FIPA – Towards a standard for software agents**. BT Technology Journal, 16v., N.3, July 1998. Disponível em: <http://www.sc-server1.bt.com/bttj/archive.htm>. Acesso em: 17 out. 2001.

OLIVEIRA, Roosevelt Vieira. **O Processo Unificado de Desenvolvimento de Software na captura e análise de requisitos de um sistema de ensino-aprendizagem cooperativo computadorizado**. 2001. Monografia (Especialização) - Universidade Federal do Maranhão. São Luís.

OMG. Disponível em: <http://www.omg.com> Acesso em: 15 jun. 2001.

OMG. **Agent technology green paper**. Technical Report ec/2000-08-01, Object Management Group, August 2000. Disponível em: <http://www.jamesodell.com/publications.html> Acesso em: 9 jun. 2001

ODELL, James. **Introduction to Agents**, 2000. Disponível em: <http://www.jamesodell.com> Acesso em: 16 jun. 2001.

ODELL, James. "**Agents", a one-hour overview presentation**, August, 1998. Disponível em: <http://www.jamesodell.com>. Acesso em: 16 jun. 2001.

PAGE-JONES, Meilir. **Fundamentos do Desenho Orientado a Objeto com UML**. São Paulo: MAKRON Books, 2001. Trad.: Celso Roberto Paschoa, 462p.

PUC DO RIO GRANDE DO SUL. Lúcia Giraffa. Disponível em: <http://pucrs.campus2.br/~nani/trabalhos/linguagens.htm> Acesso em: 17. jun. 2002.

RATIONALROSE. Desenvolvido por: Rational Software Corporation. Disponível em: <http://www.rational.com/> Acesso em: 3 jul. 2001.

SEARLE, J. R. **Speech Acts: an essay in the philosophy of language**. New York: Cambridge, 1980.

SERRA JR., Gentil. **Agente de Modelagem do Aprendiz para o Ambiente Mathnet de Ensino Cooperativo e Computadorizado**. 2001. 102 f.. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Maranhão. São Luis.

SILVA, Luciano; GARCIA, Nilza Aréan. **Introdução à Linguagem Java**. Departamento de Ciência da Computação – IME/USP. Instituto de Pesquisas Tecnológicas – IPT. Fevereiro de 1997.

SINGH, M. **A semantic for speech acts**. Annals of Mathematics and Artificial Intelligence, 1998. vol. 8. Número: I-II. 47-71 p. Disponível em: <http://citeseer.nj.nec.com/cache/papers/cs/10268/http:zSzzSzwww.csc.ncsu.eduzSzfacultyzSzmpsinghzSzpaperszSzmaszSzamai.pdf/singh98semantics.pdf>. Acesso em: 2 nov. 2001.

SOLOMOS, Konstantinos; AVOURIS, Nikolaos. **Learning from Multiple Collaborating Intelligent Tutors: an agent-based approach**. Journal of Interactive Learning Research (JILR), Nortfolk, v. 10, n. 3,4, p. 243-262, 1999. Disponível em: <http://www.ee.upatras.gr/hci/hcien/papers/j16_Solomos_Avouris_99.pdf> Acesso em: 15 jul. 2001.

UNIVERSITY OF SOUTHAMPTON. Coordenação do Departamento de Eletrônica e Ciência da Computação. Disponível em: <http://www.agentlink.org>. Acesso em: 15 jun. 2001.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. Patrícia Augustin Marques. Disponível em: <http://www.inf.ufrgs.br/~pjaques/papers/dissertacao/cap3.htm> Acesso em: 20 mar. 2002a.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. Disponível em: <http://protem.inf.ufrgs.br/bazzan/cmp504/Cecilia/corpo_coordenacao_agentes_po.html> Acesso em: 20 mar. 2002b.

WEB Mining. Disponível em: http://www.cs.umbc.edu/~ajoshi/web-mine/. Acesso: 11 nov. 2002.

WHITE, J. **Mobile agents** In Software Agents, Bradshaw, J. Ed., MIT Press. 1997.

WOOLDRIDGE, Michael. Semantic Issues in the Verification of Agent Communication Languages. **Journal of Autonomous Agents and multiagent systems** (**JAAMAS)**, Netherlands, n.1, mar. 2000. p. 9-31. Disponível em: <http://www.kluweronline.com/issn/1387-2532>. Acesso em: 18 ago. 2001.

ZAMBERLAM, Alexandre de Oliveira; GIRAFFA, Lúcia Maria Martins, **Modelagem de Agentes utilizando a arquitetura BDI**. Rio Grande do Sul, abril de 2001. PUCRS. Disponível em: <http://www.inf.pucrs.br/tr/tr008.pdf> Acesso em: 5 fev. 2002.

ZEUS. Disponível em: <http://more.btexact.com/projects/agents/zeus/ >. Acesso em: 16 set. 2001.

APÊNDICE A - Uma introdução à linguagem UML

1 DEFININDO UML

A UML (Linguagem de Modelagem Unificada) é uma linguagem-padrão para a elaboração da estrutura de projetos de software (BOOCH et al., 2000), tendo por objetivo especificar, construir, visualizar e documentar sistemas complexos de software. Tudo isso, através da: utilização de uma linguagem gráfica padrão, composta de vários diagramas que representam diferentes aspectos de um mesmo objeto de estudo e que conectados entre si permitem a correta compreensão de qualquer aspecto desse sistema e sem ambigüidades; possibilidade de mapear os modelos em linguagens de programação (Java, C++, etc.) ou até tabelas de banco de dados relacionais ou Orientados a Objeto e pela utilização da engenharia reversa; e utilização de linguagens que possibilitam o gerenciamento de versões, modelagem das atividades de planejamento do projeto,etc.

Essa notação padronizada permite que qualquer pessoa envolvida na produção, instalação e manutenção de software possa expressar como o projeto do sistema funciona e os componentes envolvidos. Contemplando, desta forma, tanto os elementos conceituais, como processos comerciais e funções de sistema, quanto os elementos concretos, como classes de linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis, atendendo, deste modo, aos gerentes de projeto e aos programadores/desenvolvedores.

Além disso, os sistemas para os quais essa modelagem pode ser aplicada, sem qualquer problema, pode abranger sistemas de informação corporativos a serem distribuídos a aplicações baseadas em Web e até sistemas complexos embutidos de tempo real.

O entendimento dessa linguagem baseia-se na correta compreensão de três elementos principais (BOOCH et al., 2000), que são: os blocos básicos de construção da UML (cada um deles representando uma semântica bem definida), as regras que determinam como esses blocos de construção deverão ser combinados e alguns mecanismos básicos que se aplicam a toda a linguagem.

A UML, por ser somente uma linguagem, é independende do processo de desenvolvimento de software que se quer aplicar no projeto. E, como toda linguagem, a UML fornece um conjunto de palavras (vocabulário) e as regras de combinação dessas palavras que permitem, deste modo, a comunicação. Por ser uma linguagem de modelagem seu vocabulário e regras tem o foco voltado para a representação conceitual e física de um sistema.

1.1 Diagramas de UML

Embora todos os diagramas de UML sirvam para visualizar, especificar, construir e documentar eles o fazem levando-se em conta aspectos diferentes de um sistema. Esses aspectos podem ser: estáticos (estrutura descrita é sempre válida em qualquer ponto no ciclo de vida do sistema), representados através de diagramas estruturais, e dinâmicos, representados através de diagramas comportamentais.

Os diagramas de UML são:

a) **Diagrama de Classes** - é composto por classes, interfaces e colaborações e seus relacionamentos (ver Fig. 1). Uma classe define os atributos compartilhados por todos os objetos criados a partir dela e estabelece relacionamentos com outras classes (FURLAN, 1998). Deste modo, uma classe descreve as propriedades (conjunto de

atributos) e comportamento (conjunto de operações) de um tipo de objeto.

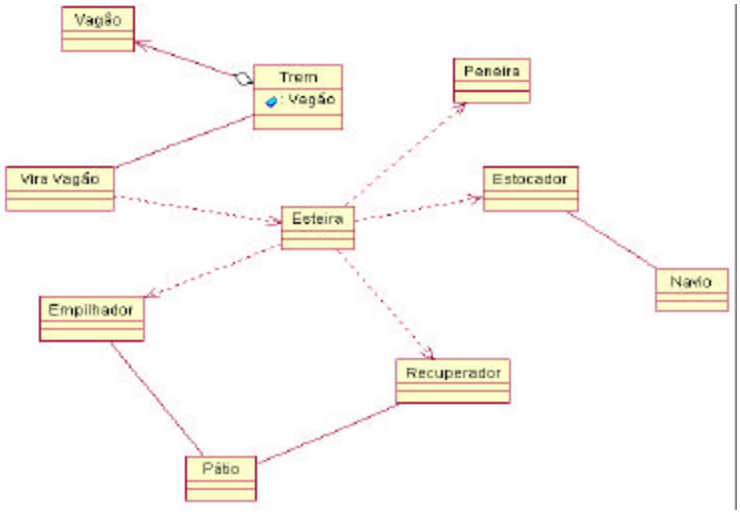

Figura 1 – Exemplo de um Diagrama de Classe

b) **Diagrama de Objetos** – permite modelar as instâncias das classes contidas no Diagrama de Classe mostrando um conjunto de objetos e seus relacionamentos em um determinado ponto no tempo. Graficamente é composto por um conjunto de vértices e de arcos. Objeto é uma instância de uma classe e instanciar uma classe significa criar um novo objeto a partir dessa classe (ver Fig. 2).

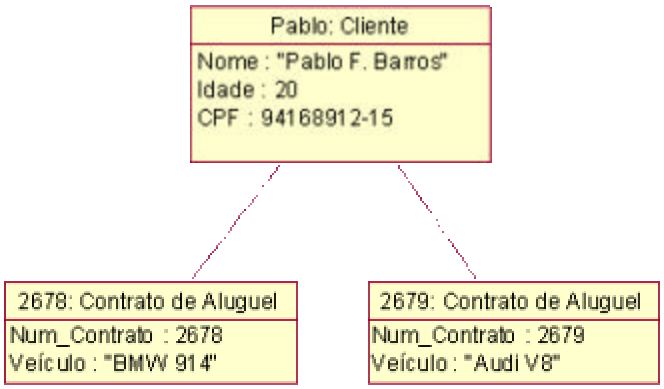

Figura 2 – Exemplo de um Diagrama de Objetos

c) **Diagrama de Componentes** – Nele são mostradas as dependências entre componenetes de software, inclusive componenetes de código fonte, código binário e outros componentes executáveis (ver Fig. 3).

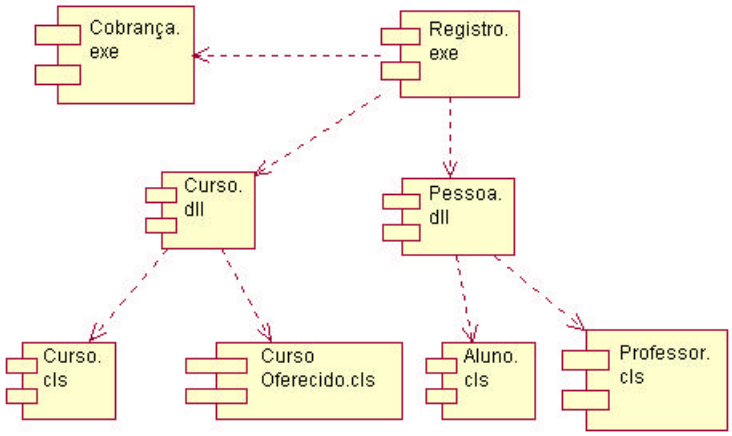

Figura 3 – Exemplo de um Diagrama de Componentes
Fonte: FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 231 p.

d) **Diagrama de Implantação** – Mostra elementos de configuração de processamento *run-time* e os componentes de software, processos e objetos que neles se mantêem (ver Fig. 4). Nele estão incluídos a parte física do sistema (computadores e outros dispositivos de interconexão).

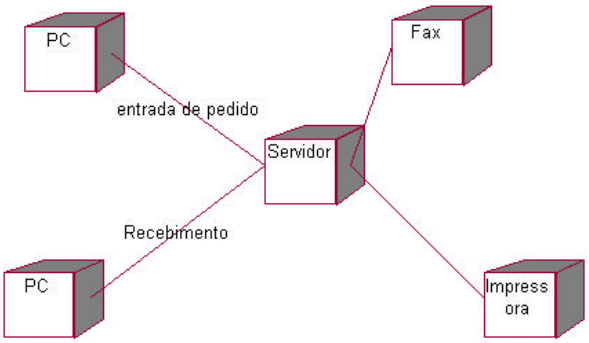

Figura 4 – Exemplo de um Diagrama de Implantação
Fonte: FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 81 p.

e) **Diagrama de Casos de Uso** – os casos de uso representam uma seqüência de transações ocorridas num sistema em resposta a um evento iniciado por um ator sobre o próprio sistema (ver Fig. 5). Um ator é uma entidade externa ao sistema que se modela e que pode interagir com ele; um exemplo de ator poderia ser um usuário ou qualquer outro sistema. As relações entre casos de uso e atores podem ser as seguintes: i) um ator se comunica com um caso de uso; ii) um caso de uso extende outro caso de uso; iii) um caso de uso utiliza outro caso de uso. Esse diagrama tem por objetivo organizar os comportamentos do sistema ou de parte dele, descrevendo um conjunto de sequências de ações (fluxo de eventos), mediante sua interação com os usuários e/ou outros sistemas. Deste modo, não é necessário especificar como esse comportamento será implementado.

Figura 5 – Exemplo de um Diagrama de Caso de Uso
Fonte: FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 266 p.

f) **Diagrama de Sequência** - tem como foco, a ordem temporal das mensagens. Mostrando as interações entre um conjunto de objetos, ordenadas em relação ao tempo (ver Fig. 6). Deste modo, um objeto origem pode solicitar (chamar) uma operação de um outro objeto. Essas mensagens podem ser: simples, síncronas e assíncronas. Esse diagrama permite indicar qual é o momento em que se envia e se completa uma mensagem mediante o tempo de transição.

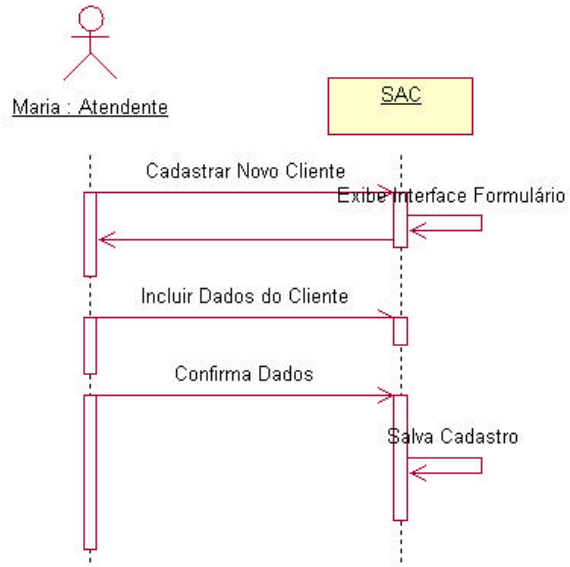

Figura 6 – Exemplo de um Diagrama de Seqüência

g) **Diagrama de Colaboração** - seu foco está na organização estrutural de objetos que enviam e recebem mensagens, ou seja, mostra a interação entre vários objetos e os relacionamentos que existem entre eles (ver Fig. 7). A diferença para um diagrama de seqüência é que o diagrama de colaboração mostra as relações entre os objetos na seqüência do tempo em que se produzem as mensagens. Objeto é uma instância de uma classe. Um relacionamento é uma instância de uma associação que conecta os objetos do diagrama. Um relacionamento pode ser reflexivo se conecta a um elemento consigo mesmo. A existência de um relacionamento entre os objetos indica que pode existir um intercâmbio de mensagens entre os objetos conectados. Durante a execução de um diagrama de colaboração se criam e se destroem objetos e relacionamentos.

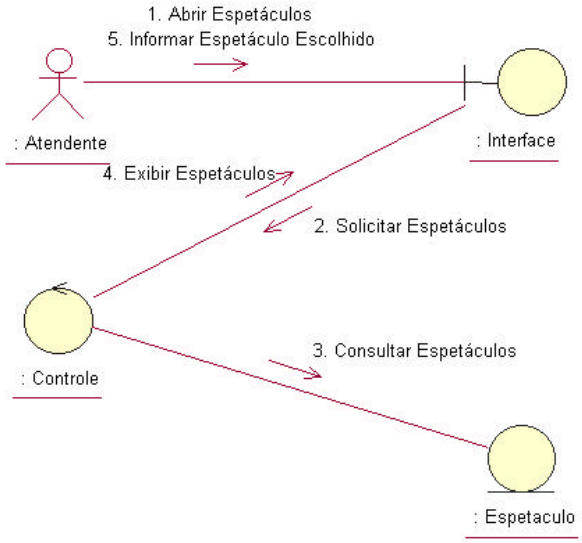

Figura 7 – Exemplo de um Diagrama de Colaboração

h) **Diagrama de Gráfico de Estados** - seu foco está no estado de mudança de um sistema orientado por eventos. Representam a seqüência de estados pelos quais um objeto ou uma interação entre objetos passa durante seu tempo de vida em resposta a estímulos (eventos) recebidos, ou seja, representa o que podemos denominar de um conjunto uma máquina de estados (ver Fig. 8). Um estado em UML é quando um objeto ou uma interação ao satisfazer uma condição, provoca alguma ação ou se encontra esperando um evento. Quando um objeto ou uma interação passa de um estado para outro em decorrência de um evento se diz que ele sofreu uma transição.

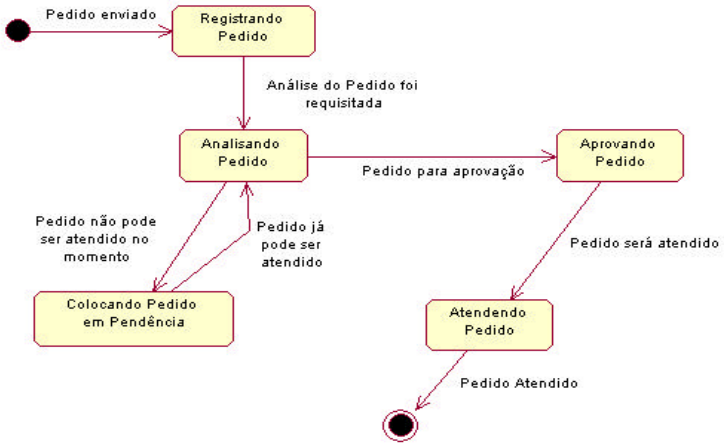

Figura 8 – Exemplo de um Diagrama de Gráfico de Estados
Fonte: FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 224 p.

i) **Diagrama de Atividade** - tem como foco, o fluxo de controle de uma atividade para outra. Esse diagrama corresponde a um caso especial dos diagramas de estado, pois os estados são considerados estados de ação, ou seja, os estados que são acionados através de uma ação interna e uma ou mais transições que se sucedem até finalizar esta ação e as transições são provocadas pela finalização das ações que têm lugar nos estados de origem (ver Fig. 9). Os diagramas de atividade são utilizados para mostrar o fluxo das operações que se desencadeiam em um procedimento interno do sistema.

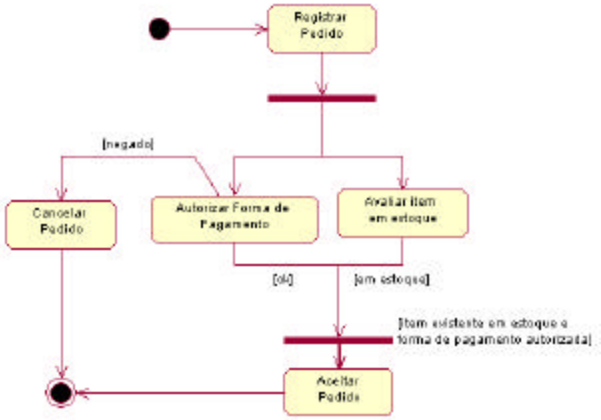

Figura 9 – Exemplo de um Diagrama de Atividade

Os diagramas de Sequência e Colaboração são chamados de Diagramas de Interação. Deste modo, quando se refere à Diagrama de Interação está-se referindo aos de Sequência ou Colaboração. Já os diagramas de Componente e Implantação são conhecidos como Diagramas de Implementação.

Todos esses diagramas aqui apresentados são utilizados na UML para modelar algum tipo de aspecto que o desenvolvedor deseja ressaltar. Entretanto, isto não significa que todos esses modelos serão utilizados durante o desenvolvimento de um projeto. Cabe ao projetista/desenvolvedor escolher quais serão utilizados e que melhor representem a finalidade e comportamento/funcionamento interno do sistema.

1.2 Notações Genéricas

Além dos diagramas citados, a UML provê um conjunto de elementos que podem ser utilizados em qualquer dos diagramas citados anteriormente.

No MATHNET foi utilizado a notação de Pacote (ver Capítulo 4) para expressar grandes porções do sistema, ou seja, o Pacote é um mecanismo de propósito geral que serve para organizar elementos de modelo em grupos e que pode também está aninhado dentro de outros pacotes, o que implicaria numa subordinação de um pacote por outro (ver Fig. 10).

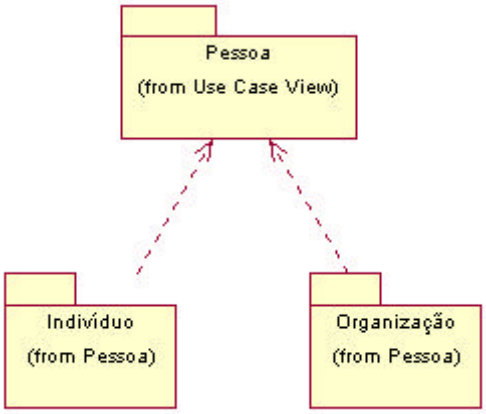

Figura 10 – Exemplo de Pacotes
Fonte: FURLAN, José Davi. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998. 83 p.

Desta forma, os pacotes resultam da subdivisão do sistema em porções/partes que possuem uma mesma lógica funcional.

Além dos pacotes, existem ainda outras notações genéricas, mas que não são de interesse neste trabalho.

APÊNDICE B - Descrição dos Atos Comunicativos de FIPA-ACL

ATO COMUNICATIVO	DESCRIÇÃO
ACCEPT-PROPOSAL	É a aceitação de uma proposta que foi previamente submetida. Onde o agente emissor da aceitação informa ao agente receptor que pretende, no futuro, que o agente receptor execute a ação, uma vez que determinada pré-condição é ou torna-se verdadeira.
AGREE	É a ação de concordar em executar alguma ação, possivelmente no futuro. É precedido por um *request* para realizar uma ação. Deste modo, o agente emissor informa o acordo ao receptor que ele pretende executar a ação, mas não até que dada pré-condição seja verdadeira.
CANCEL	É a ação de cancelamento feito anteriormente por um *request*. Permitindo que um agente interrompa outro agente de continuar a execução (ou a espera da execução) de uma ação que foi anteriormente pedida. A tentativa de cancelar uma ação que já foi executada resultará numa mensagem de *refuse* que será enviada de volta ao emissor do pedido.
CFP	É a ação de chamar para propor a execução de uma dada ação. Inicia o processo de negociação para a compor a busca por propostas para executar uma dada ação. O protocolo atual sob o qual o processo de negociação é estabelecido é conhecido também pelo acordo prévio ou é explicitamente iniciado no parâmetro de *:protocol* da mensagem.
CONFIRM	O emissor informa ao receptor que uma dada proposta é verdadeira, aonde o receptor é conhecido por estar duvidoso a respeito da proposta. Deste modo, o agente emissor: 1) acredita que de certa forma a proposta é verdadeira; 2) entende que o agente receptor também possa acreditar que a proposta é verdadeira; 3) acredita que o receptor está em dúvida sobre a veracidade da proposta.
DISCONFIRM	O emissor informa ao receptor que uma dada proposta é falsa, aonde o receptor é conhecido por acreditar, ou provavelmente acredita que, a proposta é verdadeira. É usado quando um agente deseja alterar a atitude mental conhecida de outro agente. Deste modo, o agente emissor: 1) acredita de certa forma que a proposta é falsa; 2) entende que o agente receptor também possa acreditar que a proposta é falsa; 3) acredita que o receptor acredita que é uma proposta duvidosa. Do ponto de vista do receptor, quando ele receber um *disconfirm* ele acredita que: 1) o emissor acredita que a proposta contida na mensagem é falsa; 2) o emissor deseja que o receptor acredite que a proposta é falsa também.
FAILURE	É a ação de informar a outro agente que houve a tentativa de realizar uma ação, mas a tentativa falhou. Ou seja, informa que um ato foi considerado possível para emissor, mas não foi completado por alguma razão. Assim, o agente receptor é levado a acreditar que: 1) a ação não foi realizada; 2) a ação é (ou, durante o tempo da tentativa de execução da ação, foi) possível.
INFORM	O emissor informa ao receptor que uma dada proposta é verdadeira. O agente emissor: 1) acredita de certa forma que a proposta é verdadeira; 2) entende que o agente receptor também pode acreditar que a proposta é verdadeira; 3) já não acredita que o receptor tenha algum conhecimento sobre a veracidade da proposta. Do ponto de vista do receptor, ele: 1) acredita que o emissor acredita na proposta que está no conteúdo da mensagem; 2) o emissor deseja que o receptor acredite na proposta também.
INFORM-IF (macro ato)	É uma macro ação onde o agente da ação informa ao receptor se a proposta é ou não verdadeira. Um agente que executa um *inform-if* realmente executará um *inform* padrão. O conteúdo do ato *inform* dependerá das crenças do agente passadas. Se for falso ele enviará um *refuse*.

ATO COMUNICATIVO	DESCRIÇÃO
INFORM-REF (macro ato)	É uma macro ação onde o emissor informa ao receptor o objeto que corresponde a um descritor definido, como por exemplo um nome. Ou seja, permite que o emissor informe ao receptor algum objeto que o emissor acredita corresponder a um descritor, como um nome ou outra descrição de identificação. Corresponde a uma possibilidade infinita de disjunções de *inform*
NOT-UNDERSTOOD	O emissor informa ao receptor que ele percebeu que o receptor executou alguma ação, mas que ele não entendeu que o receptor já havia feito. Isso pode acontecer por várias razões como: quando a mensagem recebida não pode ser identificada ou o conteúdo da mensagem não pode ser processado.
PROPOSED	É a ação de submeter uma proposta para executar uma certa ação, dada certas pré-condições serem verdadeiras.
QUERY-IF	É a ação de perguntar a outro agente caso uma dada proposta seja verdadeira ou não. Ou seja, o agente emissor está pedindo ao receptor um *inform* se a proposta é verdadeira. Deste modo, o agente que executa um *query-if*: 1) não tem conhecimento do valor verdadeiro da proposta; 2) acredita que o outro agente conheça se a proposta é verdadeira.
QUERY-REF	É a ação de perguntar a outro agente pelo objeto referido na expressão. Ou seja, é o ato de perguntar a outro agente para que informe o solicitante do objeto identificado por um descritor definido. Deste modo, o emissor está pedindo para o receptor executar um *inform*, contendo o objeto que corresponde ao descritor definido. Um agente executando um *query-ref*: 1) não conhece qual objeto corresponde ao descritor; 2) acredita que o outro agente conhece qual o objeto que corresponde ao descritor.
REFUSE	É a ação de recusar a execução de uma dada ação e explicar a razão da recusa. Isto acontece quando o agente não pode encontrar todas as pré-condições para ação ser executada. O agente receptor dessa mensagem acredita que: 1) a ação não foi feita; 2) a ação não é possível (do ponto de vista do emissor).
REJECT-PROPOSAL	É a ação de rejeitar a proposta de execução de uma ação durante a negociação. O agente emissor da rejeição informa ao receptor que ele não tem intenção de que o receptor execute uma dada ação sob dadas pré-condições.
REQUEST	O emissor requer que o receptor execute alguma ação, que pode ser um outro ato comunicativo. No conteúdo da mensagem está a descrição da ação a ser executada, em alguma linguagem que o receptor entende.
REQUEST-WHEN	O emissor quer que o receptor execute alguma ação quando uma dada proposta for verdadeira. Ou seja, permite que um agente informe outro agente que uma certa ação deverá ser executada, tão logo uma dada pré-condição, expressa como uma proposta, seja verdadeira. Existe um compromisso de assegurar que a ação será executada assim que a condição tornar-se verdadeira. Esse compromisso é mantido até que: a condição torne-se verdadeira, ou que o agente emissor faça um *cancel* do *request-when* ou até quando o agente decidir que é improvável que ele honre o compromisso, podendo ser enviada uma mensagem de *refuse* ao emissor.
REQUEST-WHENEVER	O emissor quer que o receptor execute uma ação tão logo alguma proposta torne-se verdadeira e desde então toda a vez que a proposta tornar-se verdadeira novamente. E além disso, se a proposta subseqüentemente tornar-se falsa essa ação será repetida tão logo ela torne-se verdadeira novamente. Representa um compromisso contínuo de reavaliar uma dada proposta e fazer algo quando o seu valor mudar.
SUBSCRIBE	É o ato de requerer com o objetivo persistente de notificar o emissor sobre o valor de uma referência e notificar novamente a todo o momento que o objeto identificado pela referência mudar. É uma versão persistente do *query-ref* pelo fato que o receptor do *subscribe* enviará ao emissor *inform* toda vez que o objeto identificado pela referência mudar.

APÊNDICE C - Correspondência entre performativas KQML e CA's de FIPA-ACL

PERFORMATIVAS KQML	CA's DE FIPA-ACL
Broadcast, transport-address, forward	*Request*
Ask-if	*Query-if*
Tell	*Inform**
Untell	*Inform**
Deny	*Inform* ou disconfirm*
Subscribe	*Subscribe*
Error	*Not-understood*
Sorry	*Failure*

***** Depende de quais parâmetros estão sendo utilizados

APÊNDICE D - Diagrama Geral dos Principais Casos de Uso do MATHNET

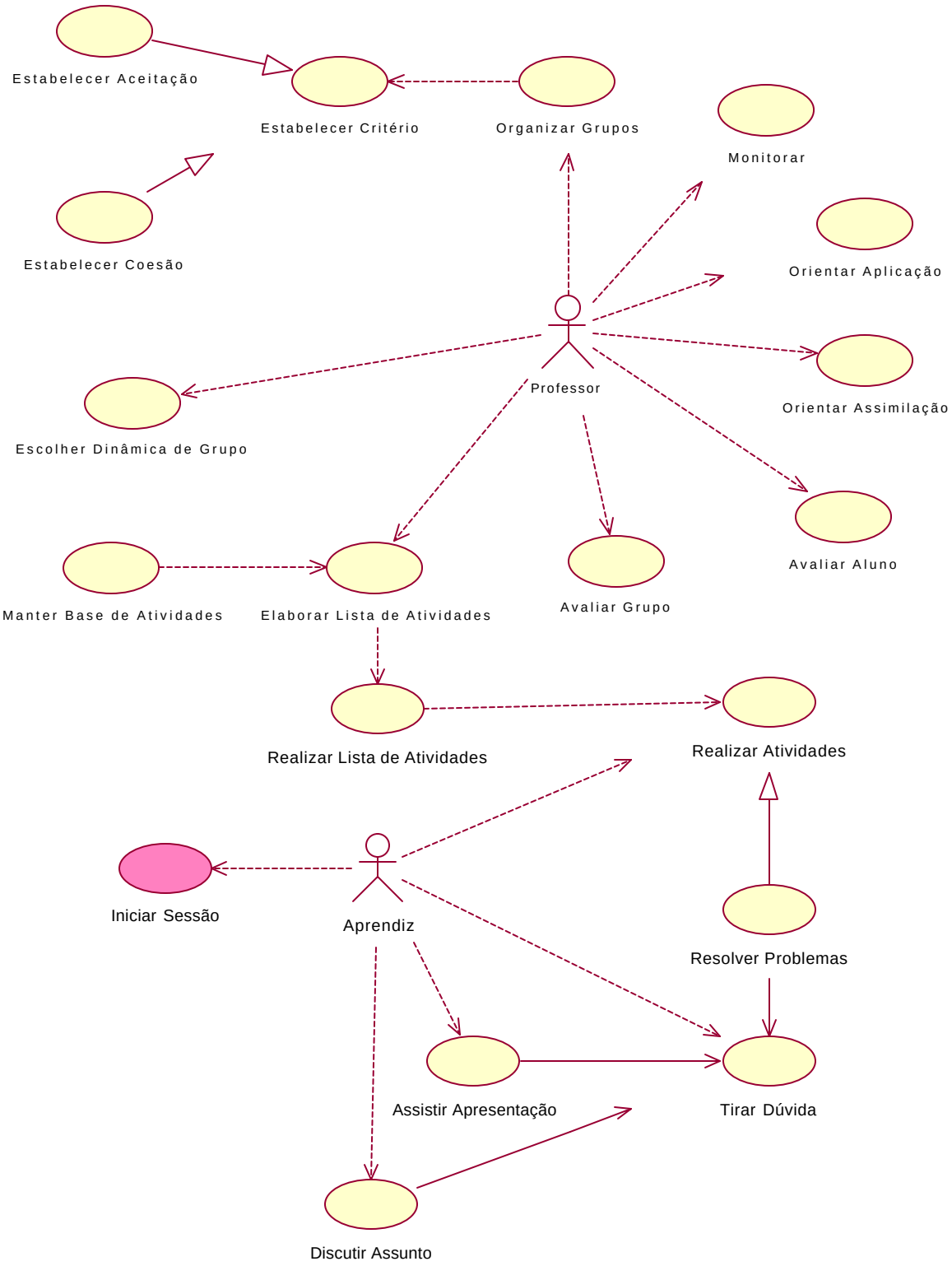

Figura 1 – Diagrama dos Principais Casos de Uso do MATHNET (visão geral)

APÊNDICE F – Ferramenta ZEUS

1 DEFININDO A FERRAMENTA ZEUS

O pacote ZEUS é gratuito, de código aberto e constituído de um conjunto de componentes escritos em JAVA, que podem ser descritos em três grupos funcionais ou bibliotecas e que se encontram expressas conforme a Fig. 1, que são:

a) **Biblioteca de Componentes** – é uma coleção de softwares que implementa as funcionalidades necessárias de um sistema multi-agentes. Fornecendo: um sistema de mensagens baseado nos protocolos TCP/IP (que facilita a intercomunicação entre agentes executados em sistemas operacionais diferentes), uma linguagem de comunicação de agentes baseada em performativas a FIPA-ACL, uma biblioteca de coordenação de estratégias, suporte para diferentes tipos de organização, um sistema de planejamento de ações para atingir os objetivos e uma estrutura para armazenamento de informação;

b) **Ferramenta construtora de agentes** – há vários editores que guiam os analistas pelas fases de desenvolvimento do agente. Durante este processo o analista deve descrever o agente pela forma como este deve atuar e pelas tarefas que irá desempenhar. Entretanto, para que a ferramenta construa o agente na linguagem de programação Java é necessário que o programador forneça à aplicação diversas informações. A ferramenta fornece, ainda, um guia passo-à-passo para a descrição dessas informações. Elementos como: estado inicial do agente, a sua capacidade de planejamento, a descrição de tarefas a cumprir, as relações entre agentes e um descritor de tarefas, são

informações necessárias à ferramenta para que o agente seja construído de forma automática.

c) **Conjunto de utilitários** – as ferramentas de visualização agrupam a informação e permitem a sua visualização de várias formas e a qualquer momento. Esta foi solução dos criadores do ZEUS para que a análise e depuração dos agentes pudessem ser tarefas mais fáceis. Os componentes desta ferramenta de visualização são vários, dentre eles destacam-se: um visualizador das relações entre agentes, das tarefas destes e do ponto de execução das tarefas, um servidor de nomes, um módulo de estatísticas de todos os agentes e um módulo individual para visualização dos estados de cada agente. É dada ao visualizador a possibilidade de visualizar apenas aquilo que deseja e como o deseja.

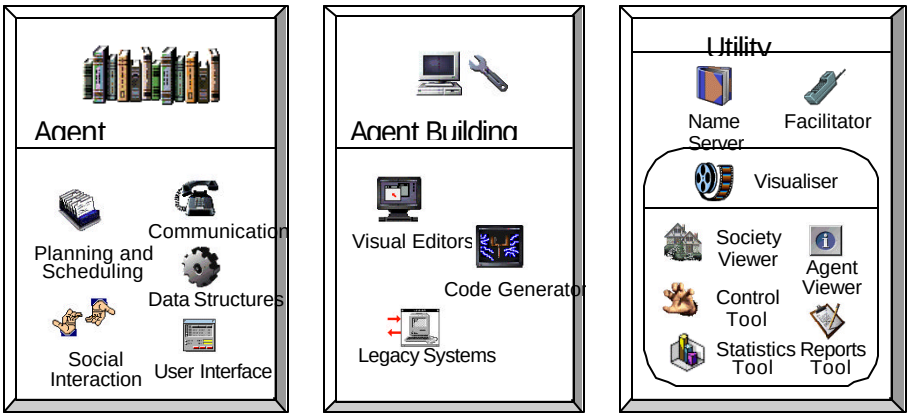

Figura 1 – Componentes da Ferramenta ZEUS

1.1 Camadas dos agentes em ZEUS

Cada agente construído no ZEUS é uma entidade composta por três camadas (COLLIS et al. ,1999) (ver Fig. 2):

1. Camada de Definição – inclui as capacidades do agente de raciocinar, seus objetivos, recursos, habilidades, crenças, preferências, etc.;
2. Camada Organizacional – nela são descritos os relacionamentos dos agentes com outros agentes do sistema.
3. Camada de Coordenação – especifica que cada agente é modelado como uma entidade social em termos de coordenação e técnicas de negociação que eles possuem.

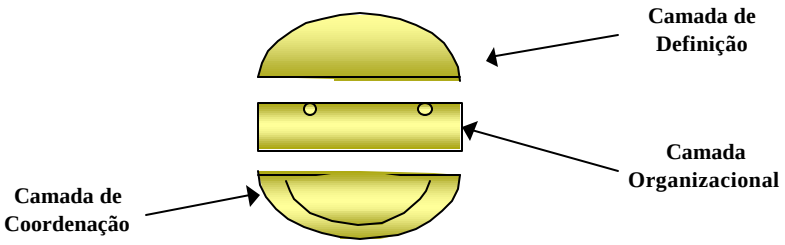

Figura 2 – Camadas de um agente no ZEUS

1.2 Metodologia de criação e desenvolvimento de agentes em ZEUS

Além disso, ZEUS possui ainda uma metodologia para o desenvolvimento e criação de agentes. Esta metodologia foi desenvolvida com o objetivo de facilitar a identificação dos agentes, seus atributos, tarefas, habilidades e protocolos, tendo como resultado final a criação do agente em si.

Essa criação é composta por 7 fases (ver Fig. 3), cada qual ao final produzindo um resultado dentro de todo o processo de criação de agentes (COLLIS et al. ,1999).

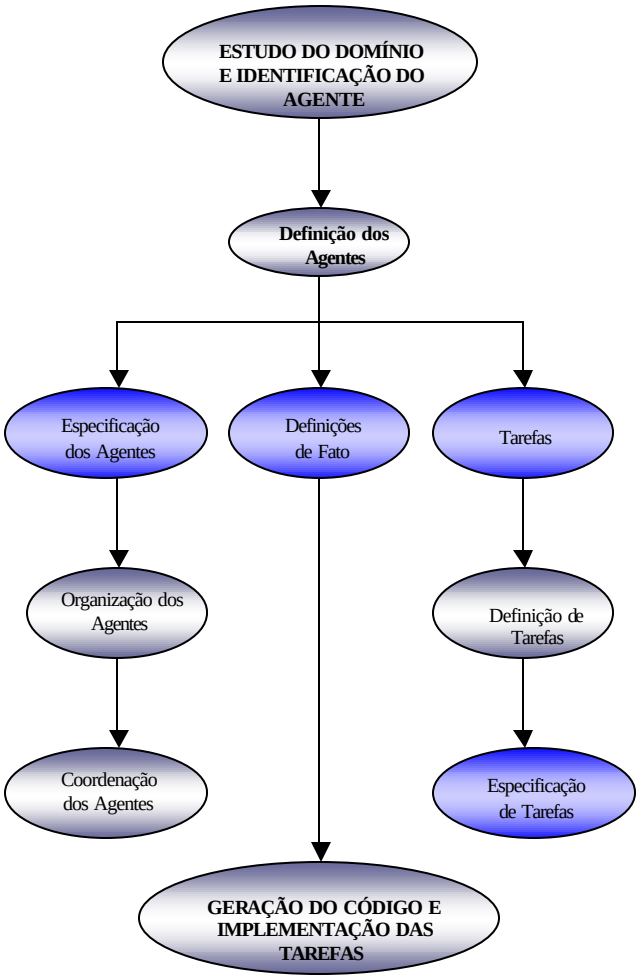

Figura 3 – Passos da metodologia de criação de agentes no ZEUS

Estas fases podem ser descritas brevemente como:

a) **Estudo do Domínio e Identificação do Agente** – o desenvolvedor analisa o domínio do problema para a identificação quais entidades serão agentes, para tal, estas entidades devem possuir a característica de atuar autonomamente, ou seja, serem capazes de atuar sem a interação humana;

b) **Definição dos Agentes** – durante esta fase outros candidatos a agentes podem ser identificados, pois nela objetivo principal é identificar os atributos relevantes para cada um dos agentes. Além

disso, nesta fase identifica-se o conjunto inicial de recursos possuídos e tarefas a executar pelos agentes.

c) **Organização dos Agentes** – neste estágio as informações sobre as habilidades dos agentes são requeridas e pode ser executada juntamente com a fase de Definição de Tarefas, identificando deste modo, o conhecimento de cada agente.

d) **Definição das Tarefas** – nesta fase as tarefas que foram identificadas durante a Definição do Agente são definidas em termos de custos, duração, pré-condições, produtos e restrições;

e) **Coordenação do Agente** – nela encontram-se envolvidos os protocolos de coordenação adequados para cada agente, e, provavelmente, requerer interação social com outros agentes quando executa seus deveres designados.

Ao final destas fases os agentes estarão criados e poderá se gerar seus respectivos códigos e implementar suas tarefas.

APÊNDICE G – Implementação de “Assistir Apresentação”

```
/*Domínio.java
/*
        This software was produced as a part of research
        activities. It is not intended to be used as commercial
        or industrial software by any organisation. Except as
        explicitly stated, no guarantees are given as to its
        reliability or trustworthiness if used for purposes other
        than those for which it was originally intended.

        (c) British Telecommunications plc 1999.
*/

/*
This code was automatically generated by ZeusAgentGenerator version 1.1
                DO NOT MODIFY!!
*/

import java.util.*;
import java.io.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.actors.*;
import zeus.agents.*;

public class Dominio {
  protected static void version() {
    System.err.println("ZeusAgent - Dominio version: 1.1");
    System.exit(0);
  }

  protected static void usage() {
    System.err.println("Usage: java Dominio -s <dns_file> -o <ontology_file> [-gui
ViewerProg] [-e <ExternalProg>] [-r ResourceProg] [-name <AgentName>] [-debug] [-h] [-
v]");
    System.exit(0);
  }

  public static void main(String[] arg) {
  try {
    ZeusAgent agent;
    String external = null;
    String dns_file = null;
    String resource = null;
    String gui = null;
    String ontology_file = null;
    Vector nameservers = null;
    String name = new String ("Dominio");
    Bindings b = new Bindings("Dominio");
    FileInputStream stream = null;
    ZeusExternal user_prog = null;
```

```
      for( int j = 0; j < arg.length; j++ ) {
        if ( arg[j].equals("-s") &&  ++j < arg.length )
          dns_file = arg[j];
        else if ( arg[j].equals("-e") &&  ++j < arg.length )
          external = arg[j];
        else if ( arg[j].equals("-r") &&  ++j < arg.length )
          resource = arg[j];
        else if ( arg[j].equals("-o") &&  ++j < arg.length )
          ontology_file = arg[j];
        else if ( arg[j].equals("-gui") &&  ++j < arg.length )
          gui = arg[j];
        else if ( arg[j].equals("-debug") ) {
          Core.debug = true;
          Core.setDebuggerOutputFile("Dominio.log");
        }
        else if ( arg[j].equals("-v") )
          version();
        else if ( arg[j].equals("-name")  && ++j < arg.length )
           name = name  + arg[j];
        else if ( arg[j].equals("-h") )
          usage();
        else
          usage();
      }

       b = new Bindings(name);
      if ( ontology_file == null ) {
        System.err.println("Ontology Database file must be specified with -o option");
        usage();
      }
      if ( dns_file == null ) {
        System.err.println("Domain nameserver file must be specified with -s option");
        usage();
      }

        nameservers = ZeusParser.addressList(new FileInputStream(dns_file));
        if ( nameservers == null || nameservers.isEmpty() )
          throw new IOException();

        agent = new ZeusAgent(name,ontology_file,nameservers,1,20,true,false);

        AgentContext context = agent.getAgentContext();
        OntologyDb db = context.OntologyDb();


/*
        Initialising Extensions
*/
        Class c;

        if ( resource != null ) {
          c = Class.forName(resource);
          ExternalDb oracle = (ExternalDb) c.newInstance();
          context.set(oracle);
          oracle.set(context);
```

```
        }
        if ( gui != null ) {
          c = Class.forName(gui);
          ZeusAgentUI ui = (ZeusAgentUI)c.newInstance();
          context.set(ui);
          ui.set(context);
        }

/*
        Initialising ProtocolDb
*/
        ProtocolInfo info;
        info = ZeusParser.protocolInfo(db,"(:name
\"zeus.actors.graphs.ContractNetRespondent\" :type Respondent :constraints ((:fact (:type
ZeusFact :id var97 :modifiers 1) :type 0 :strategy
\"zeus.actors.graphs.DefaultRespondentEvaluator\")))");
        if ( info.resolve(b) )
          agent.addProtocol(info);

/*
        Initialising TaskDb
*/
        AbstractTask t;
        t = ZeusParser.primitiveTask(db,"(:Primitive ForneceConteudo :time (1) :cost (0)
:produced_facts ((:type Conteudo :id var289 :modifiers 1 :attributes ((Conteudo_ID 1)(Nome
\"Grecia Antiga\")))))");
        if ( t.resolve(b) )
          agent.addTask(t);
        t = ZeusParser.primitiveTask(db,"(:Primitive ExibirReferencias :time (1) :cost (0)
:produced_facts ((:type Exibir :id var221 :modifiers 33 :attributes ((Codigo 1)))))");
        if ( t.resolve(b) )
          agent.addTask(t);
/*
        Initialising OrganisationalDb
*/
        AbilityDbItem item;
/*
        Initialising ResourceDb
*/
        Fact f1;
        f1 = ZeusParser.fact(db,"(:type Referencia :id fact724 :modifiers 0 :attributes
((Referencia_ID 1)(Localizacao \"c:/notasdeaula.pdf\")(Aplicativo \"C:/Arquivos de
programas/Adobe/Acrobat 5.0/Reader/AcroRd32.exe\")(Nome \"Gr\u00e9cia Antiga -
Religi\u00e3o\")(Conteudo_ID 1)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Referencia :id fact760 :modifiers 0 :attributes
((Referencia_ID 2)(Localizacao \"www.historia.com.br\")(Aplicativo \"iexplorer.exe\")(Nome
\"Grecia Antiga - Politica\")(Conteudo_ID 1)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Conteudo :id fact244 :modifiers 0 :attributes ((Nome
\"Grecia Antiga\")(Conteudo_ID 1)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
```

```
/*
        Initialising External User Program
*/

        if ( external != null ) {
           c = Class.forName(external);
           user_prog = (ZeusExternal) c.newInstance();
           context.set(user_prog);
        }


/*
        Activating External User Program
*/

        if ( user_prog != null )
           user_prog.exec(context);
      }
 catch (ClassNotFoundException cnfe) {
     System.out.println("Java cannot find some of the classes that are needed to run this
agent. Please ensure that you have the followingin your classpath :
zeus_install_dir\\lib\\zeus.jar, zeus_install_dir\\lib\\gnu-regexp.jar, java_install_dir\\jre\\rt.jar
Where zeus_install_dir is the directory that you have installed  Zeus in , and java_install_dir
is the directory that you have  installed Java in");
   cnfe.printStackTrace();}
      catch(Exception e) {
         e.printStackTrace();
      }
   }
}
```

/***Domínio_ext.java***

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import zeus.agents.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.gui.fields.*;
import zeus.actors.*;
import zeus.actors.event.*;
import zeus.generator.util.*;

public class Dominio_ext implements ZeusExternal,MessageMonitor{

    private AgentContext agent;
    private Button button;
    private MailBox mbox;

   public void exec(AgentContext agentContext){
       System.out.println("----------Dominio_ext----------");
                agent = agentContext;
```

```
                mbox=agent.MailBox();
                ZeusAgent zAgent = (ZeusAgent)agent.Agent();
                zAgent.addMessageMonitor(this,1);

    }

  public void exibirReferencias()
  {
                System.out.println("-------dentro de exibirReferencias--------");

                Fact[] fatos2;
                fatos2=agent.ResourceDb().all("Conteudo");
                TabelaReferencias tabela=new TabelaReferencias("Agente " + agent.whoami() +
" - Referencias sobre " + fatos2[0].getValue("Nome"));

                int i=0;
                Fact[] fatos;
                fatos=agent.ResourceDb().all("Referencia");
                while(i<fatos.length)
                {

tabela.insere_dados(fatos[i].getValue("Nome"),fatos[i].getValue("Localizacao"),fatos[i].getVal
ue("Aplicativo"));
                  i++;
        }
        tabela.pack();
        tabela.setVisible(true);
                Performative perf=new Performative("inform");
                perf.setReplyWith("Referencias_ok");
                perf.setReceiver("Tutor");
                perf.setContent("");
                mbox.sendMsg(perf);

  }

  public void messageDispatchedEvent(MessageEvent event){}

  public  void messageNotDispatchedEvent(MessageEvent event){}

  public void messageQueuedEvent(MessageEvent event) {}

  public void messageReceivedEvent(MessageEvent event)
  {
    if (event.getMessage().getType().equals("accept-proposal") &&
event.getMessage().getSender().equals("Tutor"))
    {
                exibirReferencias();
                System.out.println("dentro de messageReceivedEvent: " );
    }
  }
}

/*ExibirReferencias.java
/*
                This software was produced as a part of research
```

*/

/*
This stub file was automatically generated by ZeusAgentGenerator version 1.1
*/

```
import java.util.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.actors.TaskContext;
import zeus.actors.ZeusTask;

public class ExibirReferencias extends ZeusTask {
  protected void exec() {
    /*
    Add the task execution code here. The following variables are defined:
      protected Fact[][]  inputArgs;
      protected Fact[][]  outputArgs;
    Before exec() is called inputArgs will contain the input
    Facts consumed by the task. After execution, set outputArgs to
    contain the output Facts produced by the task.
    */

    // The Input Facts:


    // The Output Facts:

    Fact[] _var221;       // Exibir

    /* USER CODE STARTS */

    System.out.println("-Expected Input-");
    for(int i = 0; i < expInputArgs.length; i++ )
      System.out.println(expInputArgs[i].pprint());

    System.out.println("-Input-");
    for(int j = 0; j < inputArgs.length; j++) {
      System.out.println("Input Fact["+j+"]");
      for(int i = 0; i < inputArgs[j].length; i++)
        System.out.println(inputArgs[j][i].pprint());
    }

    System.out.println("-Expected Output-");
    for(int i = 0; i < expOutputArgs.length; i++ )
      System.out.println(expOutputArgs[i].pprint());

    System.out.println("-Output-");
```

```
      _var221 = new Fact[1];
      _var221[0] = new Fact(Fact.FACT,expOutputArgs[0]);
      System.out.println(_var221[0].pprint());

      /* USER CODE ENDS */
      outputArgs = new Fact[1][];
      outputArgs[0] = _var221;
    }
}


/*ForneceConteudo
/*
                This software was produced as a part of research
                activities. It is not intended to be used as commercial
                or industrial software by any organisation. Except as
                explicitly stated, no guarantees are given as to its
                reliability or trustworthiness if used for purposes other
                than those for which it was originally intended.

                (c) British Telecommunications plc 1999.
*/

/*
This stub file was automatically generated by ZeusAgentGenerator version 1.1
*/

import java.util.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.actors.TaskContext;
import zeus.actors.ZeusTask;

public class ForneceConteudo extends ZeusTask {
  protected void exec() {
      /*
      Add the task execution code here. The following variables are defined:
        protected Fact[][]  inputArgs;
        protected Fact[][]  outputArgs;
      Before exec() is called inputArgs will contain the input
      Facts consumed by the task. After execution, set outputArgs to
      contain the output Facts produced by the task.
      */

      // The Input Facts:


      // The Output Facts:

      Fact[] _var289;       // Conteudo

      /* USER CODE STARTS */

      System.out.println("-Expected Input-");
      for(int i = 0; i < expInputArgs.length; i++ )
```

```
      System.out.println(expInputArgs[i].pprint());

    System.out.println("-Input-");
    for(int j = 0; j < inputArgs.length; j++) {
      System.out.println("Input Fact["+j+"]");
      for(int i = 0; i < inputArgs[j].length; i++)
        System.out.println(inputArgs[j][i].pprint());
    }

    System.out.println("-Expected Output-");
    for(int i = 0; i < expOutputArgs.length; i++ )
      System.out.println(expOutputArgs[i].pprint());

    System.out.println("-Output-");
    _var289 = new Fact[1];
    _var289[0] = new Fact(Fact.FACT,expOutputArgs[0]);
    System.out.println(_var289[0].pprint());

    /* USER CODE ENDS */
    outputArgs = new Fact[1][];
    outputArgs[0] = _var289;
  }
}

/*ModelAprendiz.java
/*
        This software was produced as a part of research
        activities. It is not intended to be used as commercial
        or industrial software by any organisation. Except as
        explicitly stated, no guarantees are given as to its
        reliability or trustworthiness if used for purposes other
        than those for which it was originally intended.

        (c) British Telecommunications plc 1999.
*/

/*
This code was automatically generated by ZeusAgentGenerator version 1.1
                DO NOT MODIFY!!
*/

import java.util.*;
import java.io.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.actors.*;
import zeus.agents.*;

public class ModelAprendiz {
  protected static void version() {
    System.err.println("ZeusAgent - ModelAprendiz version: 1.1");
    System.exit(0);
  }

  protected static void usage() {
```

```
    System.err.println("Usage: java ModelAprendiz  -s <dns_file> -o <ontology_file> [-gui
ViewerProg] [-e <ExternalProg>] [-r ResourceProg] [-name <AgentName>] [-debug] [-h] [-
v]");
    System.exit(0);
  }

  public static void main(String[] arg) {
  try {
    ZeusAgent agent;
    String external = null;
    String dns_file = null;
    String resource = null;
    String gui = null;
    String ontology_file = null;
    Vector nameservers = null;
    String name = new String ("ModelAprendiz");
    Bindings b = new Bindings("ModelAprendiz");
    FileInputStream stream = null;
    ZeusExternal user_prog = null;


    for( int j = 0; j < arg.length; j++ ) {
      if ( arg[j].equals("-s") &&  ++j < arg.length )
        dns_file = arg[j];
      else if ( arg[j].equals("-e") &&  ++j < arg.length )
        external = arg[j];
      else if ( arg[j].equals("-r") &&  ++j < arg.length )
        resource = arg[j];
      else if ( arg[j].equals("-o") &&  ++j < arg.length )
        ontology_file = arg[j];
      else if ( arg[j].equals("-gui") &&  ++j < arg.length )
        gui = arg[j];
      else if ( arg[j].equals("-debug") ) {
        Core.debug = true;
        Core.setDebuggerOutputFile("ModelAprendiz.log");
      }
      else if ( arg[j].equals("-v") )
        version();
      else if ( arg[j].equals("-name")  && ++j < arg.length )
         name = name  + arg[j];
      else if ( arg[j].equals("-h") )
        usage();
      else
        usage();
    }

     b = new Bindings(name);
    if ( ontology_file == null  ) {
      System.err.println("Ontology Database file must be specified with  -o option");
      usage();
    }
    if ( dns_file == null ) {
      System.err.println("Domain nameserver file must be specified with -s option");
      usage();
    }
```

```
        nameservers = ZeusParser.addressList(new FileInputStream(dns_file));
        if ( nameservers == null || nameservers.isEmpty() )
          throw new IOException();

        agent = new ZeusAgent(name,ontology_file,nameservers,1,20,false,false);

        AgentContext context = agent.getAgentContext();
        OntologyDb db = context.OntologyDb();


/*
        Initialising Extensions
*/
        Class c;

        if ( resource != null ) {
          c = Class.forName(resource);
          ExternalDb oracle = (ExternalDb) c.newInstance();
          context.set(oracle);
          oracle.set(context);
        }
        if ( gui != null ) {
          c = Class.forName(gui);
          ZeusAgentUI ui = (ZeusAgentUI)c.newInstance();
          context.set(ui);
          ui.set(context);
        }

/*
        Initialising ProtocolDb
*/
        ProtocolInfo info;
        info = ZeusParser.protocolInfo(db,"(:name
\"zeus.actors.graphs.ContractNetRespondent\" :type Respondent :constraints ((:fact (:type
ZeusFact :id var95 :modifiers 1) :type 0 :strategy
\"zeus.actors.graphs.DefaultRespondentEvaluator\")))");
        if ( info.resolve(b) )
          agent.addProtocol(info);
        info = ZeusParser.protocolInfo(db,"(:name \"zeus.actors.graphs.ContractNetInitiator\"
:type Initiator :constraints ((:fact (:type ZeusFact :id var94 :modifiers 1) :type 0 :strategy
\"zeus.actors.graphs.DefaultInitiatorEvaluator\")))");
        if ( info.resolve(b) )
          agent.addProtocol(info);

/*
        Initialising OrganisationalDb
*/
        AbilityDbItem item;
/*
        Initialising ResourceDb
*/
        Fact f1;
        f1 = ZeusParser.fact(db,"(:type ConteudoProgramado :id fact268 :modifiers 0 :attributes
((Aluno_ID CP1)(Grupo_ID 1)(Qtde_Assistida 0)(Conteudo_ID 1)))");
```

```
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type ConteudoProgramado :id fact277 :modifiers 0 :attributes
((Aluno_ID CP1)(Grupo_ID 1)(Qtde_Assistida 1)(Conteudo_ID 2)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);


/*
        Initialising External User Program
*/

        if ( external != null ) {
          c = Class.forName(external);
          user_prog = (ZeusExternal) c.newInstance();
          context.set(user_prog);
        }


/*
        Activating External User Program
*/

        if ( user_prog != null )
          user_prog.exec(context);

      }
 catch (ClassNotFoundException cnfe) {
     System.out.println("Java cannot find some of the classes that are needed to run this
agent. Please ensure that you have the followingin your classpath :
zeus_install_dir\\lib\\zeus.jar, zeus_install_dir\\lib\\gnu-regexp.jar, java_install_dir\\jre\\rt.jar
Where zeus_install_dir is the directory that you have installed  Zeus in , and java_install_dir
is the directory that you have  installed Java in");
   cnfe.printStackTrace();}
      catch(Exception e) {
        e.printStackTrace();
      }
   }
}
```

/***ModeloTabelaReferencias**

```
import javax.swing.JTable;
import javax.swing.table.AbstractTableModel;
import javax.swing.JScrollPane;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.JOptionPane;
import java.awt.*;
import java.util.*;
import java.awt.event.*;

public class ModeloTabelaReferencias extends AbstractTableModel
{
```

```
    private String[] columnNames = {"Assunto","Referência","Aplicativo"};
    private Vector dados=new Vector();
    public int getColumnCount() { return 3; }
    public int getRowCount() { return dados.size();}
    public Object getValueAt(int row, int col) { return ((String[])dados.get(row))[col]; }
    public void setValueAt(Object value)
    {
      dados.add(value);
      fireTableRowsInserted(0, getRowCount());
    }
    public String getColumnName(int col) {
        return columnNames[col];
     }
    public void limpa_tabela() {dados.clear();}
}

/*Tutor.java
/*
                This software was produced as a part of research
                activities. It is not intended to be used as commercial
                or industrial software by any organisation. Except as
                explicitly stated, no guarantees are given as to its
                reliability or trustworthiness if used for purposes other
                than those for which it was originally intended.

                (c) British Telecommunications plc 1999.
*/

/*
This code was automatically generated by ZeusAgentGenerator version 1.1
                    DO NOT MODIFY!!
*/

import java.util.*;
import java.io.*;
import zeus.util.*;
import zeus.concepts.*;
import zeus.actors.*;
import zeus.agents.*;

public class Tutor {
  protected static void version() {
    System.err.println("ZeusAgent - Tutor version: 1.1");
    System.exit(0);
  }

  protected static void usage() {
    System.err.println("Usage: java Tutor -s <dns_file> -o <ontology_file> [-gui ViewerProg] [-
e <ExternalProg>] [-r ResourceProg] [-name <AgentName>] [-debug] [-h] [-v]");
    System.exit(0);
  }

  public static void main(String[] arg) {
  try {
    ZeusAgent agent;
```

```
String external = null;
String dns_file = null;
String resource = null;
String gui = null;
String ontology_file = null;
Vector nameservers = null;
String name = new String ("Tutor");
Bindings b = new Bindings("Tutor");
FileInputStream stream = null;
ZeusExternal user_prog = null;


for( int j = 0; j < arg.length; j++ ) {
   if ( arg[j].equals("-s") &&  ++j < arg.length )
      dns_file = arg[j];
   else if ( arg[j].equals("-e") &&  ++j < arg.length )
      external = arg[j];
   else if ( arg[j].equals("-r") &&  ++j < arg.length )
      resource = arg[j];
   else if ( arg[j].equals("-o") &&  ++j < arg.length )
      ontology_file = arg[j];
   else if ( arg[j].equals("-gui") &&  ++j < arg.length )
      gui = arg[j];
   else if ( arg[j].equals("-debug") ) {
      Core.debug = true;
      Core.setDebuggerOutputFile("Tutor.log");
   }
   else if ( arg[j].equals("-v") )
      version();
   else if ( arg[j].equals("-name")  && ++j < arg.length )
       name = name  + arg[j];
   else if ( arg[j].equals("-h") )
      usage();
   else
      usage();
}

 b = new Bindings(name);
if ( ontology_file == null ) {
   System.err.println("Ontology Database file must be specified with  -o option");
   usage();
}
if ( dns_file == null ) {
   System.err.println("Domain nameserver file must be specified with -s option");
   usage();
}

   nameservers = ZeusParser.addressList(new FileInputStream(dns_file));
   if ( nameservers == null || nameservers.isEmpty() )
      throw new IOException();

   agent = new ZeusAgent(name,ontology_file,nameservers,1,20,false,false);

   AgentContext context = agent.getAgentContext();
   OntologyDb db = context.OntologyDb();
```

```
/*
        Initialising Extensions
*/
        Class c;

        if ( resource != null ) {
          c = Class.forName(resource);
          ExternalDb oracle = (ExternalDb) c.newInstance();
          context.set(oracle);
          oracle.set(context);
        }
        if ( gui != null ) {
          c = Class.forName(gui);
          ZeusAgentUI ui = (ZeusAgentUI)c.newInstance();
          context.set(ui);
          ui.set(context);
        }

/*
        Initialising ProtocolDb
*/
        ProtocolInfo info;
        info = ZeusParser.protocolInfo(db,"(:name
\"zeus.actors.graphs.ContractNetRespondent\" :type Respondent :constraints ((:fact (:type
ZeusFact :id var108 :modifiers 1) :type 0 :strategy
\"zeus.actors.graphs.DefaultRespondentEvaluator\")))");
        if ( info.resolve(b) )
          agent.addProtocol(info);
        info = ZeusParser.protocolInfo(db,"(:name \"zeus.actors.graphs.ContractNetInitiator\"
:type Initiator :constraints ((:fact (:type ZeusFact :id var107 :modifiers 1) :type 0 :strategy
\"zeus.actors.graphs.DefaultInitiatorEvaluator\")))");
        if ( info.resolve(b) )
          agent.addProtocol(info);

/*
        Initialising OrganisationalDb
*/
        AbilityDbItem item;
/*
        Initialising ResourceDb
*/
        Fact f1;
        f1 = ZeusParser.fact(db,"(:type Grupo :id fact679 :modifiers 0 :attributes ((Grupo_ID
1)(Nome Historia)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Aluno :id fact684 :modifiers 0 :attributes ((Aluno_ID
CP1)(Nome Joao)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Grupo_Aluno :id fact689 :modifiers 0 :attributes
((Aluno_ID CP1)(Grupo_ID 1)(Qtde_Programada 3)))");
        if ( f1.resolve(b) )
```

```
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Grupo :id fact696 :modifiers 0 :attributes ((Grupo_ID
2)(Nome Geografia)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);
        f1 = ZeusParser.fact(db,"(:type Grupo_Aluno :id fact106 :modifiers 0 :attributes
((Aluno_ID CP1)(Grupo_ID 2)(Qtde_Programada 5)))");
        if ( f1.resolve(b) )
          agent.addFact(f1);


/*
        Initialising External User Program
*/

        if ( external != null ) {
          c = Class.forName(external);
          user_prog = (ZeusExternal) c.newInstance();
          context.set(user_prog);
        }


/*
        Activating External User Program
*/

        if ( user_prog != null )
          user_prog.exec(context);

      }
 catch (ClassNotFoundException cnfe) {
     System.out.println("Java cannot find some of the classes that are needed to run this
agent. Please ensure that you have the followingin your classpath :
zeus_install_dir\\lib\\zeus.jar, zeus_install_dir\\lib\\gnu-regexp.jar, java_install_dir\\jre\\rt.jar
Where zeus_install_dir is the directory that you have installed  Zeus in , and java_install_dir
is the directory that you have  installed Java in");
   cnfe.printStackTrace();}
      catch(Exception e) {
        e.printStackTrace();
      }
   }
}
```