UMA ARQUITETURA MULTIAGENTE PARA AMBIENTES VIRTUAIS DE APRENDIZAGEM

Diego Rodrigues Sanches

Dissertação de Mestrado em Informática

Mestrado em Informática
Universidade Federal do Espírito Santo
Vitória – ES. - 2006

Diego Rodrigues Sanches

Uma arquitetura multiagente para ambientes virtuais de aprendizagem

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática. Programa de Pós-Graduação em Informática.

Universidade Federal do Espírito Santo.

Orientador: Prof. Dr. Orivaldo de Lira Tavares.

Vitória - ES
2006

UMA ARQUITETURA MULTIAGENTE PARA AMBIENTES VIRTUAIS DE APRENDIZAGEM

Diego Rodrigues Sanches

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

Aprovada em ___ de _____ de 2006.

COMISSÃO EXAMINADORA

_____
Prof. Dr. Orivaldo de Lira Tavares
Universidade Federal do Espírito Santo
Orientador

_____
Prof. Dr. Davidson Cury
Universidade Federal do Espírito Santo

_____
Profa. Dra. Tânia Barbosa Salles Gava
Universidade Federal do Espírito Santo

_____
Profa. Dra. Cláudia Maria Garcia de Oliveira
Instituto Militar de Engenharia

A

Meus pais, professores, amigos e todos aqueles que de certa forma colaboraram para mais esta conquista.

AGRADECIMENTOS

Aos meus familiares, em especial meus pais José Carlos e Maria das Graças pela torcida, incentivo e apoio, sempre estando ao meu lado tornando mais esta etapa da minha vida possível.

Aos professores e amigos Denise Franzotti Togneri, Silvana Rossy de Brito e Luciano Lessa Lorenzoni pelo apoio e incentivo ao meu ingresso no mestrado. Pelas ótimas e construtivas discussões e troca de conhecimentos.

Ao meu orientador e amigo Prof. Orivaldo de Lira Tavares que acreditou no meu trabalho e que sempre esteve disposto a me auxiliar, me dando a força e o auxilio necessários durante mais esta jornada em minha vida acadêmica. Obrigado pela dedicação e pelas trocas riquíssimas de conhecimento.

A Deus, por tudo.

"O que sabemos é uma gota, o que ignoramos é um oceano".

Isaac Newton

RESUMO

Este trabalho apresenta uma arquitetura multiagente para ambientes virtuais de aprendizagem (AVA´s) que contempla uma camada multiagente para AVA's. Apresenta também o uso dessa arquitetura multiagente no ambiente virtual de aprendizado AVAUFES[1] para o refinamento dos requisitos e teste real de sua aplicabilidade. O objetivo da camada multiagente é a monitoração e suporte aos participantes e mediadores dos grupos desse ambiente, tornando suas atividades diárias mais simples, diminuindo o trabalho cognitivo e manual da utilização das ferramentas pelos usuários e, assim, liberando-os para o objetivo maior que é interagir e aprender com o ambiente e seus usuários.

[1] AVAUfes está sendo desenvolvido pelo grupo GAIA, da Universidade Federal do Espírito Santo.

ABSTRACT

This work presents a multiagent architecture for virtual environments of learning (AVA's) that contemplates a multiagent layer for AVA's. It also presents the use of this multiagent architecture in the virtual environment of learning AVAUFES[2] for the refinement of the requirements and real test of its applicability. The objective of the multiagent layer is the monitoring and support to the participants and mediators of the groups of this environment, simplifying their daily activities, diminishing the cognitive and manual work of the use of the tools by the users and, thus, liberating them for the greater objective which is interacting and learning with the environment and its users.

[2] The AVAUfes is being developed by group GAIA, from the Universidade Federal do Espirito Santo.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AIED	- *Artificial Intelligence in Education*
AVA'S	- Ambientes Virtuais de Aprendizagem
AORML	- AOR modeling language - linguagem de modelagem AOR
CAI	- *Computer Aided Instruction*
EAD	- Educação a Distância
FAESA	- Fundação de Assistência e Educação
ICAI	- *Intelligent Computer Aided Instruction*
ICAL	- *Intelligent Computer Aided Learning*
ICATS	- *Intelligent Computer Aided Tutoring Systems*
ILE	- *Intelligent Learning Environment*
IA	- Inteligência Artificial
IAS	- *Intelligent Authoring System*
STIs	- Sistemas Tutores Inteligentes
UFES	- Universidade Federal do Espírito Santo
WEB	- *Word Wide Web*

SUMÁRIO

Sanches, Diego Rodrigues,

Uma Arquitetura Multiagente para Ambientes Virtuais de Aprendizagem. [Vitória] 2006

165 p., 29,7 cm (UFES, M. Sc., Informática, 2006)

Dissertação, Universidade Federal do Espírito Santo,

I. Informática/UFES. II. Uma Arquitetura Multiagente para Ambientes Virtuais de Aprendizagem.

1 INTRODUÇÃO

Nos últimos anos, o aumento da capacidade de processamento dos computadores pessoais, da tecnologia de redes e da variedade de recursos computacionais à disposição, tem tornado cada vez mais possível o desenvolvimento de ambientes computacionais para apoiar a Educação a Distância (EAD). Esses ambientes permitem que professores e alunos, aqui denominados de mediadores e aprendizes, distantes ou não geograficamente, possam interagir e trocar experiências com o auxílio da tecnologia.

O desenvolvimento de ambientes para apoiar EAD tem buscado fornecer mecanismos para não apenas transmitir informações, mas fazer com que o aprendiz desenvolva as habilidades necessárias para aplicar seu conhecimento.

Diante dos recursos disponíveis na Internet e da necessidade de ambientes educacionais que suportem adequadamente as atividades dos mediadores e aprendizes, a *Word Wide Web* (Web) se apresenta como um dos recursos capazes de atender parte das expectativas dos pesquisadores da área de ensino/aprendizagem a distância. Essas soluções buscam promover a educação e treinamento em larga escala, a custos mais acessíveis do que os cursos tradicionais, permitindo a publicação de material didático, a utilização de tutoriais, aplicação de exercícios e testes, além de facilitar a comunicação entre aprendizes e mediadores e a apresentação de aulas a distância e até mesmo a utilização de simuladores.

No sentido de desenvolver facilidades para aprendizes e mediadores, uma diversidade de ambientes tem sido desenvolvida utilizando os recursos disponíveis na Web. No contexto das instituições de ensino, aprendizes e mediadores têm percebido, através da disponibilização de notas de aula, listas de exercícios e trabalhos na Internet, as vantagens de incorporar tais recursos ao processo de ensino-aprendizagem, seja ele presencial ou a distância.

Assim, o desenvolvimento de ambientes baseados na *Web* para apoiar a Educação

a Distância mostrou-se uma área de pesquisa de acelerada evolução. Essa evolução tem mostrado seus benefícios através do desenvolvimento de ambientes de aprendizagem que fazem uso das redes de computadores e das facilidades da hipermídia. Entretanto, percebe-se que a maioria desses ambientes (ou *sites*) educacionais apresenta poucos recursos para os mediadores e aprendizes acompanharem suas tarefas, além de uma grande quantidade de problemas não resolvidos.

Segundo Brito et al. (2000), características como o auxílio personalizado, o suporte à cooperação e o suporte às atividades docentes estão entre os problemas que ainda necessitam de melhores soluções, tanto do ponto de vista pedagógico quanto tecnológico. Esses problemas tornam-se mais evidentes a medida que os cursos a distância permitem um grande volume de participantes.

No contexto do mediador, ainda são relatadas experiências onde a sobrecarga de trabalho com o acompanhamento dos aprendizes é substancialmente maior do que no ensino presencial. Esse fato, segundo Brito et al. (2000), deve-se tanto ao modelo pedagógico adotado, quanto à necessidade de ferramentas para apoiar o professor nas suas atividades de acompanhamento dos alunos.

Do ponto de vista dos aprendizes, maior flexibilidade e adaptabilidade são necessárias nos ambientes de educação a distância, para que o modelo pedagógico adotado possa ser adequado ao modelo do aluno. Tais características são extraídas de um modelo do aluno que deve ser multidisciplinar, envolvendo a psicologia, a pedagogia e o suporte tecnológico adequado.

Para tratar a maioria desses problemas, além da utilização da Web, nos últimos anos, têm sido focadas bastante as técnicas de Inteligência Artificial (IA) no projeto e construção dos ambientes de aprendizagem. Isto devido às potencialidades que incorporam a tais ambientes. Diferentes pesquisas (GIRAFFA, 1999, VICCARI, 1997) apontam para o uso de técnicas de IA, a fim de prover sistemas computacionais de ensino, com capacidade de adaptação ao contexto e personalização do ambiente de acordo com as características dos aprendizes, além de permitir um alto grau de interatividade entre o ambiente e os usuários e um

controle maior das sessões de aprendizagem em ambientes multi-usuários.

A introdução de técnicas de IA nestes ambientes tem diversas finalidades, entre as quais se pode citar: melhores meios de modelagem do processo de ensino-aprendizagem; aperfeiçoamento da modelagem do estado cognitivo do aprendiz (MAES, 1996) e de estratégias de acompanhamento do aprendiz (BRITO, 2001).

1.1 Objetivos

Os avanços mais recentes no campo dos ambientes de aprendizagem inteligentes propõem o uso de arquiteturas baseadas em sociedade de agentes (COSTA, 1997, SILVA, 2000). Os sistemas multiagentes têm mostrado um potencial bastante adequado no desenvolvimento de sistemas de ensino, pois podem ser usados para facilitar a interação cooperativa dos grupos participantes dos ambientes virtuais de aprendizagem. Além disso, ambientes de aprendizagem baseados em arquiteturas multiagentes possibilitam suportar o desenvolvimento de sistemas de forma mais robusta, mais rápida e com menores custos, tornando-os mais atrativos, do ponto de vista de seu aproveitamento real.

No presente trabalho faz-se uma reflexão a respeito da profundidade do aprendizado proporcionado pelos cursos ministrados pela Internet. Procura-se fazer uma contribuição para que tais cursos venham a oferecer oportunidades de aprendizagem mais rica e efetiva que grande parte dos cursos atualmente oferecidos com o apoio de ambientes virtuais de aprendizagem.

A proposta deste trabalho é projetar e desenvolver uma camada inteligente para ambientes virtuais de aprendizagem, para torná-los mais interativos, dinâmico e de fácil utilização, que tenha como base estudos realizados sobre os projetos MimerDesk[1], Amcora[2] (TAVARES 2000, MENEZES *et al*., 2001), Famcora[3]

[1] **MimerDesk** é um ambiente groupware e de aprendizado colaborativo, baseado na web, com enfase na palavra trabalho em grupo. Ele foi projetado para uma grande variedade de usos, tais como gerenciamento pessoal, aprendizado colaborativo baseado em computador, executar projetos e configurar comunidades. Suas forças principais incluem um sistema de grupos altamente customizável que permite a muitos grupos trabalhar simultaneamente em uma base de dados

(PESSOA, 2002), FaesaOnline[4] e AvaUfes[5]. Este trabalho busca incorporar as facilidades necessárias para o acompanhamento dos aprendizes e o suporte ao administrador do ambiente e dos mediadores dos grupos de aprendizagem. A proposta é promover um ambiente de aprendizagem dinâmico, atrativo e interessante ao aprendiz com recursos para facilitar a mediação do processo de aprendizagem.

Além desta proposta, neste trabalho identificam-se diversos problemas enfrentados pelos aprendizes na utilização dos recursos que os ambientes virtuais de aprendizagem disponibilizam para o seu aprendizado. Entre os recursos analisados estão as ferramentas de bate-papo (*chats)*, fóruns, murais, e-mails, estantes, escaninhos, listas de discussão, agendas, troca de mensagens, lista de atividades dentre outras. A diversidade de recursos e a necessidade de construir um ambiente flexível e adaptável às características do aprendiz remetem à identificação de novos requisitos para tratar essas dificuldades.

Assim, a partir dos problemas e necessidades levantados, este trabalho apresenta a especificação de requisitos para uma camada multiagente para apoiar a aprendizagem em AVA's que permita solucionar vários dos problemas detectados nos ambientes de referência, tais como: manter o aprendiz informado sobre as tarefas pendentes, os prazos, a necessidade de ver novas informações postadas no ambiente, etc; permitir ao moderador e ao administrador estabelecer regras de interação com os aprendizes e essas regras serem implementadas pelos agentes, de modo a não sobrecarregar essas pessoas com tarefas de verificação do cumprimento das regras estabelecidas.

A camada multiagente proposta deve se mostra flexível e adaptável às

---

compartilhada, com as ferramentas livremente selecionadas tais como ferramentas para projectos, calendarios, tarefas, forums, links, chat, revisores, votação, arquivos, mensagens instantâneas, profiles, e várias outras

[2] **Amcora** é um Ambiente Telemático Inteligente para Apoio à Aprendizagem Cooperativa para apoiar os cursos presenciais e a distância da UFES.

[3] **Famcora** é um framework para a construção de ambientes cooperativos inteligentes de apoio a aprendizagem na Internet baseado em web services e agentes.

[4] **FaesaOnline** é um ambiente de EAD web que tem por objetivo disponibilizar materiais e ferramentas em formato de sala de aula virtual para a aprendizagem à distância.

[5] **AvaUfes** é uma evolução do Amcora que incorpora uma camada multiagente para dar um melhor suporte aos usuários do ambiente, tanto professores/mediadores quanto aprendizes/estudantes.

necessidades/características dos aprendizes e mediadores, de modo a permitir que seus usuários possam se sentir cada vez mais informados e integrados ao ambiente virtual, auxiliando-os no controle de suas atividades, mantendo-os informados sobre seus compromissos, atividades, datas, rendimento e atrasos, além de outras facilidades que podem ser especificadas pelos próprios usuários.

De modo sintético, os objetivos deste trabalho são:

1. especificar, projetar e implementar uma camada multiagente que possa ser acoplada de forma simples, rápida e transparente a um AVA, permitindo que este se torne um ambiente mais dinâmico, adaptável e inteligente, buscando com isso torná-lo mais útil, atrativo e de simples utilização e manutenção por parte de seus usuários.
2. Acoplar este *framework* ao AVAUfes, para utilizá-lo como estudo de caso e estar as reais facilidades e necessidades que esta solução irá trazer para este tipo de ambiente.

1.2 Metodologia do Trabalho

Durante a fase inicial de pesquisa e definição do trabalho foi definida de que forma e quais os ambientes deveriam ser estudados, o que deveria ser analisado, o que estes ambientes tinham em comum, quais seriam seus pontos fortes, quais seriam seus pontos fracos e qual seria a melhor maneira de se levantar estas informações. Nesta fase foi feita grande parte do levantamento bibliográfico do trabalho, identificados trabalhos correlatos, tecnologias aplicáveis, necessidades e problemas atuais, padrões amplamente utilizados e vantagens e desvantagens de suas utilizações.

De posse dessas informações cada ambiente foi analisado de forma separada e informações previamente adquiridas pelas equipes que os definiram e/ou utilizavam os ambientes foram aproveitadas para estudo. O MimerDesk foi analisado através de sua utilização durantes aproximadamente 2 anos pela turma de engenharia elétrica da UFES (Universidade Federal do Espírito Santo) e através do desenvolvimento de ferramentas para serem acopladas no ambiente que não dava

suporte a algumas necessidades do curso. O Amcora foi analisado também através de sua utilização pela UFES, onde foi utilizado por cinco anos, só que de uma forma mais abrangente, pois estava disponível para um grande número de cursos e turmas da instituição. O Famcora a maioria das informações foram levantadas através de material gerado por pesquisas realizadas pelo seu criador (PESSOA, 2002), e finalmente o FaesaOnline através de material e informações adquiridas durante seus 6 anos de utilização pela FAESA (Fundação de Assistência e Educação).

De posse destas informações foi criada a especificação de requisitos da solução, onde foram definidas suas funcionalidades, ou seja, o que a solução deveria fazer, suas interfaces, como a solução iria interagir com as pessoas (usuários), hardwares, outros sistemas e produtos.

Em paralelo a especificação de requisitos iniciou-se a analise da solução onde foram modeladas informações importantes do domínio do problema da solução, validado a importância destes requisitos e seu nível de detalhamento para as fases futuras do trabalho.

Após a analise e especificação de requisitos do trabalho um projeto inicial foi definido, tempo como objetivo definir como seria implementado o software da solução para atender os requisitos levantados, sendo estes funcionais e não funcionais.

De posse de toda essa documentação iniciou-se a implementação e testes do software, onde todos os conceitos previamente definidos foram implementados no contexto do sistema de informação.

Durante todas as fases do trabalho foi adotada uma evolução cíclica onde sempre que necessário os requisitos foram refinados e as documentações alteradas.

O resultado final de todo o trabalho está reunido nessa dissertação que foi desenvolvida durante todo o desenvolvimento deste projeto.

1.3 Estrutura da Dissertação

Esta dissertação é formada de 9 capítulos, sendo esses distribuídos da seguinte forma:

- Capítulo 2 - Ambientes de Aprendizagem: um histórico da evolução do software educacional até o surgimento dos ambientes de aprendizagem.
- Capítulo 3 - AVA UFES: a proposta do AVAUFES e os princípios adotados para apoio aos modelos de educação não presenciais e abertos, e as características dos ambientes de aprendizagem e a aplicação de agentes inteligentes na construção de ambientes de aprendizagem;
- Capítulo 4 - Requisitos para uma plataforma multiagente: levantamento de requisitos e problemas encontrados em sistemas de educação a distância em relação a aspectos relativos à interatividade, cooperação, colaboração e mediação, além da facilidade de uso e a construção do conhecimento e seus usuários;
- Capítulo 5 - *Framework* Jade: detalhamento do *framework* Jade e os benefícios que a utilização dele pode trazer para a solução proposta;
- Capítulo 6 - Modelagem dos Agentes: a modelagem conceitual dos agentes da solução;
- Capítulo 7 - Projeto dos Agentes: o modelo de *design* dos agentes.
- Capítulo 8 - Considerações finais e perspectivas futuras do trabalho: apresenta as conclusões, limitações e perspectivas para trabalhos futuros a partir dos resultados alcançados.
- Capítulo 9 – Referências

2 AMBIENTES DE APRENDIZAGEM

Este capítulo apresenta uma breve descrição da evolução do software educacional até o surgimento dos ambientes de aprendizagem. Este capítulo também discute diversos ambientes de aprendizagem envolvidos no estudo para o levantamento dos atuais problemas encontrados nesses sistemas. Além disso, busca evidenciar a adequação do paradigma de agentes para resolver grande parte desses problemas.

2.1 Educação a distância

Segundo Azevedo (1999), o uso da Internet como meio facilitador para a EAD tem provocado profundas alterações no método convencional de ensino. O professor deixa de ser o centro de fluência da informação, passando a ser um mediador, facilitador e guia no acesso as informações. O aluno deixa de ser um passivo aprendiz e torna-se um aprendiz ativo, com a responsabilidade de ser o principal responsável pela construção do seu conhecimento.

A chamada educação através dos recursos da Internet está desafiando as instituições de ensino a repensarem seus modelos pedagógicos e está exigindo o desenvolvimento de um modelo pedagógico específico. A EAD via Internet começa a ser vista por algumas instituições como uma alternativa para reduzir custos ou permitir a rápida atualização de conteúdos sem os altos custos de reimpressão do material impresso, entre outras vantagens (Azevedo, 1999).

A definição presente no Decreto n. 2.494/1998, que regulamenta o Artigo 80 da Lei de Diretrizes e Bases (Lei n. 9.394/1996), define Educação a Distância (EAD) como “uma forma de ensino que possibilita a auto-aprendizagem, com a mediação de recursos didáticos sistematicamente organizados, apresentados em diferentes suportes de informação, utilizados isoladamente ou combinados, e veiculados pelos diversos meios de comunicação”. Pierre Lévy (1999) coloca a EAD como sendo uma modalidade de ensino que explora certas técnicas de ensino a distância, incluindo as

hipermídias, as redes de comunicação interativas e todas as tecnologias intelectuais da cibercultura, favorecendo o surgimento de um novo estilo de pedagogia, que favorece ao mesmo tempo as aprendizagens personalizadas e a aprendizagem coletiva em rede. O mediador torna-se um animador da inteligência coletiva em vez de um fornecedor direto de conhecimentos.

2.2 Um breve histórico dos ambientes educacionais

As primeiras modalidades de *softwares* educacionais que surgiram no início da década de 60 foram os do tipo CAI (*Computer Aided Instruction*), que utilizavam a teoria comportamentalista, proposta por Skinner (1958) citado por Giraffa (1999) como forma de aprendizagem. Esses softwares foram denominados convencionalmente por alguns autores como *"page-turners"*, onde as lições eram preparadas sobre um assunto específico e seu mecanismo básico era apenas "virar páginas". Neles, não havia nenhuma diferenciação entre os diversos níveis de conhecimento dos usuários e também não havia geração de problemas e comentários individualizados para cada usuário (GIRAFFA, 1999).

Para superar as limitações dos sistemas do tipo CAI, técnicas de Inteligência Artificial (IA) foram incorporadas a tais ambientes, tornando possível a criação dos sistemas denominados ICAI (*Intelligent Computer Aided Instruction*) ou ICAL (*Intelligent Computer Aided Learning*). As características inteligentes incorporadas em tais ambientes permitiram personalizar as ações do software de modo individualizado, tornando esses sistemas mais interessantes para os aprendizes.

Os sistemas do tipo ICAI ou ICAL também passaram a ser chamados de sistemas tutores inteligentes (SLEEMAN, 1982; WENGER, 1987). O termo ICATS (*Intelligent Computer Aided Tutoring Systems*) também é usado para referenciar os sistemas tutores que fazem utilização de técnicas de IA.

Os sistemas do tipo ICAI diferenciam-se do tipo CAI, porque separam as estratégias de ensino do conteúdo a ser ensinado (GIRAFFA, 1999), além de possuírem um modelo dinamicamente atualizado do desempenho do aprendiz.

Os sistemas tutores inteligentes podem ainda ser subdivididos em: tutores inteligentes e assistentes inteligentes. Conforme Costa (1997), esta subdivisão é necessária para diferenciar a complexidade encontrada nos sistemas tutores inteligentes.

Uma das diferenças entre tutores e assistentes está relacionada com o seu porte (VICARI, 1990): assistentes são pequenos sistemas tutores que realizam tarefas mais simples. Outra diferença citada por Luzzi (1997) está relacionada a diferentes estratégias pedagógicas utilizadas na interação com o aprendiz: em um assistente a estratégia se baseia na monitoração do aprendiz, nos resultados obtidos ao longo do trabalho e sem muitas interrupções, buscando-se estimular o aprendiz a utilizar os recursos do ambiente e fazê-lo refletir sobre o que está fazendo. Nos tutores, esta estratégia é mais evasiva e com mais interrupções na interação entre o aprendiz e tutor, por parte do tutor.

Pode-se entender que a maior diferença entre tutores e assistentes esteja no modelo do aprendiz. No caso dos tutores, este modelo é mais forte e robusto. Em um assistente, o modelo do aprendiz é mais leve ou inexiste. Os assistentes procuram auxiliar o aprendiz no entendimento do assunto e na construção do seu conhecimento. Os assistentes, geralmente, são menos diretivos. Como eles usam um modelo e uma monitoração menos robusta do aprendiz, por conseqüência, as estratégias (seu comportamento) possuem um conjunto de táticas e ações menos elaboradas do que as dos tutores.

Outra importante classe de ambientes são os IAS (*Intelligent Authoring System*) ou ambientes de autoria ou sistemas geradores de tutores inteligentes (Giraffa, 1999), que permitem a construção desses sistemas para um certo domínio. Mais recentemente, os pesquisadores da área de AIED (*Artificial Intelligence in Education*), têm encontrado uma nova divisão para os ambientes de ensino inteligentes: os ambientes interativos de aprendizagem através do computador (Guin et al., 1995), também conhecidos como ILE (*Intelligent Learning Environment*).

2.3 Ambientes de educação a distância

Em Santos (1999) encontra-se uma descrição de ambientes na área de educação a distância. Segundo Santos (1999), o uso educacional das tecnologias de rede apóia-se em diferentes vertentes de pesquisa e desenvolvimento, e esse uso pode ser reunido em seis modalidades: (1) Aplicações de hipermídia para fornecer instrução distribuída; (2) Sites educacionais; (3) Sistemas de autoria para cursos a distância; (4) Salas de aula virtuais; (5) *Frameworks* para aprendizagem cooperativa; (6) Ambientes distribuídos para aprendizagem cooperativa. São vários os ambientes atualmente disponíveis que se classificam dentro dessas modalidades, cada um deles com suas características particulares e compromissos com diferentes abordagens. Das modalidades apresentadas por Santos (1999), foram estudados para o contexto desse trabalho os seguintes ambientes:

MimerDesk – um ambiente *groupware* e de aprendizado colaborativo, baseado na web, com ênfase no trabalho em grupo. Ele foi projetado para uma grande variedade de usos, tais como gerenciamento pessoal, aprendizado colaborativo baseado em computador, execução de projetos e configuração de comunidades. Suas forças principais incluem um sistema de grupos altamente customizável que permite muitos grupos trabalharem simultaneamente em uma base de dados compartilhada, com as ferramentas livremente selecionadas.

Projetado para facilitar os problemas com trabalhos baseados em grupos, o *MimerDesk* torna fácil trabalhar em grupos, dentro e fora de uma organização. Isto traz uma grande vantagem para organizações que têm muitos grupos de trabalho com os quais tem que estar em contato ao redor do globo. No *MimerDesk*, um usuário entra como um indivíduo para uma área de trabalho pessoal. A partir dali, ele pode se juntar aos vários grupos de trabalho. Todos os grupos têm suas próprias ferramentas e permissões de usuários.

FAESA@ONLINE (2001). Um ambiente de EAD web que tem por objetivo disponibilizar materiais e ferramentas em formato de sala de aula virtual para a aprendizagem a distância. Todos os acessos ao sistema são realizados através de um *browser web* que acessa um servidor único. O ambiente disponibiliza

ferramentas síncronas como *chats* e mensagens instantâneas e assíncronas como fóruns, quadro de avisos, estante, agendas, área de disponibilização de materiais, correio eletrônico dentre outras. Esse ambiente é atualmente utilizado pela instituição FAESA[1] para auxiliar no aprendizado de turmas totalmente presenciais e turmas totalmente a distância.

AmCorA - O projeto AmCorA (Menezes, 1999) contempla o suporte à interação entre os vários agentes envolvidos em um processo de aprendizagem, registrando, sistematizando e facilitando a recuperação do resultado das interações (SOUZA; MENEZES, 2001). Nele, cada colaborador (agente real) representa um aspecto do conhecimento (quanto à faceta ou maturidade) e possui um representante no sistema (agente virtual). A noção de agentes é também aplicada na modelagem de outras funcionalidades como, por exemplo, a realização de tarefas de apoio aos aprendizes e mediadores.

O AmCora é um *framework* para dar suporte ao desenvolvimento de aplicações de natureza cooperativa para a Internet, desenvolvido a partir de uma arquitetura em várias camadas (MENEZES *et al*., 1999), conforme ilustrado na FIG. 1.

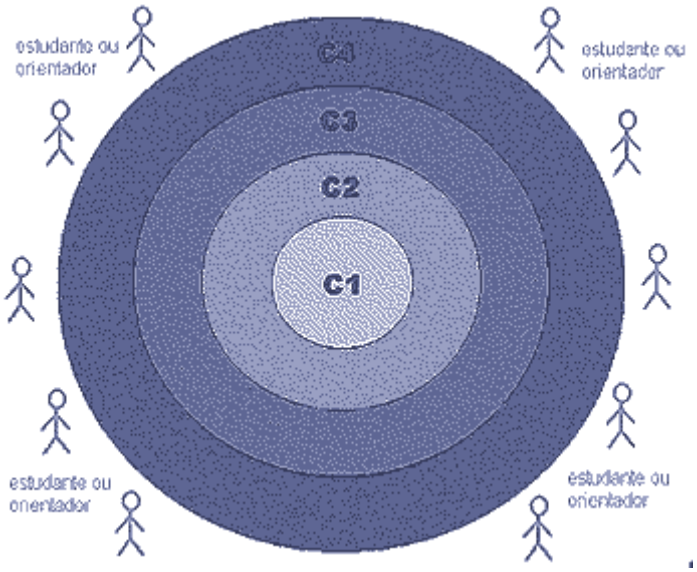

FIGURA 1 – Camadas da arquitetura do sistema AmCora
Fonte: Menezes, 1999.

[1] FAESA – Faculdades Integradas Espírito-Santenses (http://www.faesa.br).

A FIG. 1 apresenta de forma esquemática a arquitetura multicamadas do ambiente AmCora. A camada C1 é o núcleo, responsável pelo sistema de armazenamento de dados do sistema (arquivos e mensagens). A camada C2 é responsável pelos serviços básicos de comunicação (correio, transferência de arquivos, chat etc.). A camada C3 é a dos serviços inteligentes para recuperação de informação e roteamento de mensagens. A camada C4 é a camada de serviços. Nela se concentram as ferramentas de apoio ao trabalho cooperativo e de apoio ao trabalho individual, como descreveremos a seguir (MENEZES, 1999).

FAmCorA - é uma proposta de *framework*, para o desenvolvimento de aplicações do tipo CSCW/CSCL na Web (PESSOA, 2004). O *framework* está baseado em padrões abertos, com ênfase no reuso de aplicações executáveis na *Web*, e não de código fonte, serviços (APIs) ou componentes dependentes de um *middleware*. O FAmCorA faz a construção de *groupwares* na *Web* possível com o simples reuso do modelo de hipertexto. A composição e o reuso de aplicações no FAmCorA utiliza apenas conhecimento declarativo, escrito em HTML ou XML, gerado automaticamente por um ambiente visual de desenvolvimento, tornando, assim, a construção de portais CSCW/CSCL na *Web*, uma simples atividade de autoria.

AVAUFES – O AVAUFES, desenvolvido pelo grupo avaufes é uma nova proposta de ferramenta de *groupware* para consolidar o uso das novas tecnologias de informação e de comunicação nos seus cursos da UFES de graduação e pós-graduação presenciais ou não. O sistema é um ambiente virtual de aprendizagem que visa principalmente oferecer recursos para que cada indivíduo da Universidade possa interagir, mediado pelo computador e pela Internet, com outros participantes. Cria-se assim um canal multiplicador dos saberes e das experiências acadêmicas, com o propósito de apoiar as atividades didáticas e os processos decisórios que ocorrem nas diversas instâncias da UFES. O AVAUFES é o ambiente foco de nosso trabalho e será discutido com mais detalhes nesta dissertação.

QUADRO 1
Comparação entre as diversas características dos ambientes estudados

Aspectos / Ambientes	**MimerDesk**	**FAESA@ONLINE**	**AmCorA**	**FAmCorA**	**AVAUFES**
Interface com Usuário	Interface simples	Interface simples	Interface simples	Interface simples	Interface simples
Linguagem utilizada	Perl MySQL	ASP SQL SERVER	Kylix Interbase	CGI	Java EJB JBOSS MySQL
Categorias	Multi-usuário	Multi-usuário	Multi-usuário	Multi-usuário	Multi-usuário
Recursos de comunicação	*chat* mensagens-instantâneas fóruns quadro de avisos estante agenda correio eletrônico imagens sons	*chat* mensagens-instantâneas fóruns quadro de avisos estante agenda correio eletrônico imagens	*chat* mensagens-instantâneas fóruns estante agenda escaninho correio eletrônico imagens	Não se aplica	*chat* mensagens-instantâneas fóruns estante agenda escaninho correio eletrônico imagens
Plataforma	*Linux/Unix* *Web*	*Windows* Web	Linux/Unix	Linux/Unix	Linux/Unix Windows Machintosh Solaris OS2
Integração c/outras aplicações	É permitida de forma complexa	Não é permitida	Não é permitida	É permitida	É permitida de forma simples
Arquitetura	Cliente-servidor	Cliente-servidor	Cliente-servidor	Cliente-servidor	Cliente-servidor
Teorias Pedagógicas	Cooperação	Cooperação	Cooperação Construtivismo	Não se aplica	Cooperação Construtivismo
Classificação Segundo Santos (1999)	Ambiente distribuído para aprendizagem cooperativa	Salas de aula virtual	*Frameworks* para aprendizagem cooperativa	*Frameworks para aprendizagem cooperativa*	*Frameworks para aprendizagem cooperativa*

2.4 Arquitetura geral de um ambiente de aprendizagem

De um modo geral, podem-se observar diversas características comuns aos ambientes de aprendizagem analisados. Alguns elementos (módulos) desses ambientes já são contemplados nas arquiteturas dos Sistemas Tutores Inteligentes – STI como, por exemplo, na arquitetura proposta por Wenger (1987). A proposta de Wenger pode ser mapeada para um ambiente de aprendizagem se novos elementos forem incorporados, conforme pode ser observado na FIG. 2.

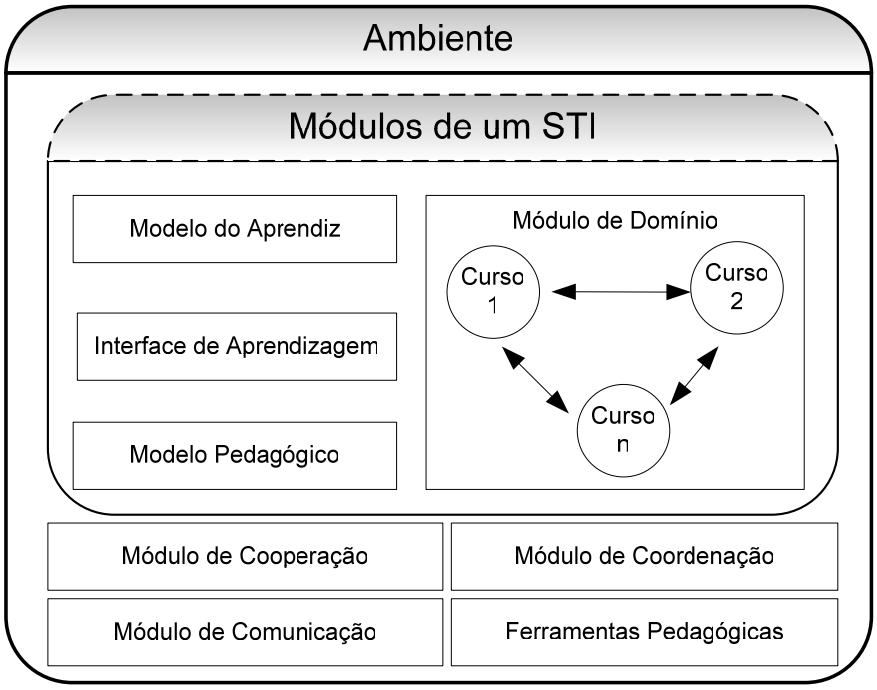

FIGURA 2 - Componentes de um ambiente de aprendizagem cooperativa

Os módulos componentes desta arquitetura são descritos a seguir.

- **Módulo do Domínio:** este módulo é responsável pela representação do conhecimento no sistema. Deve conter o conhecimento do assunto que o sistema pretende oferecer ao aprendiz (WENGER, 1987);

- **Módulo do Modelo do Aprendiz:** é responsável por manter uma imagem do aprendiz no ambiente. Este módulo capta o estado do entendimento do aprendiz a respeito do assunto que está sendo apresentado/discutido, bem como mantém informações sobre seus interesses, nível de conhecimento e ritmo de

aprendizagem;

- **Módulo do Modelo Pedagógico:** deve ser capaz de tomar decisões sobre as estratégias educacionais a serem utilizadas. Uma estratégia de aprendizagem envolve seleção e planejamento das atividades a serem apresentadas ao aprendiz (COSTA, 1997). É importante observar que esse módulo inclui as funções dos mediadores;

- **Módulo de Interface de Aprendizagem:** preocupa-se com a forma com que o conteúdo deve ser apresentado, permitindo a interação entre o aprendiz e o ambiente;

- **Módulo de Cooperação:** este módulo é responsável por prover os mecanismos que viabilizam as atividades cooperativas;

- **Módulo de Comunicação:** este módulo diz respeito aos mecanismos responsáveis pela comunicação ocorrida no ambiente;

- **Módulo de Coordenação**: este módulo é responsável pela coordenação dos vários outros módulos do ambiente;

- **Ferramentas Pedagógicas:** este módulo permite a inserção e a utilização de ferramentas pedagógicas (jogos, simuladores, micro-mundos, etc.).

2.5 O paradigma de agentes e os ambientes de aprendizagem

Um dos principais objetivos dos ambientes de aprendizagem é facilitar a aprendizagem através da flexibilidade e adaptabilidade aos seus participantes. A forma como a aprendizagem pode ser promovida é muitas vezes reforçada pelo aspecto de cooperação. Questões como a cooperação e flexibilidade do ambiente têm sido atacadas, recentemente, com um paradigma que tem se mostrado bastante apropriado: o paradigma de agentes.

Segundo Wooldridge e Ciancarini (2001), as razões para o interesse nesse paradigma são várias, mas a mais importante se deve, certamente, ao conceito de um agente como um sistema autônomo, capaz de interagir com outros agentes para satisfazer seus objetivos, bem como por ser uma forma natural para se projetar *software*. Da mesma forma como se pode compreender um sistema como sendo composto de objetos passivos, que possuem estados e sobre os quais executam-se operações, também se pode compreender outros sistemas como sendo agentes (semi) autônomos que interagem para atingir um objetivo comum.

Antes que se possa discutir o desenvolvimento de sistemas baseados em agentes e, mais especificamente, da aplicação deste paradigma no desenvolvimento de ambientes educacionais, deve-se entender o que significam os termos agentes e sistemas baseados em agentes. Infelizmente, vários conceitos, considerados chave no processamento baseado em agentes, não possuem uma definição universalmente aceita e, portanto, necessita-se adotar uma definição para contextualizar este trabalho dentro da área de pesquisa de Agentes Inteligentes.

No contexto deste trabalho, adotou-se a definição de que um agente é qualquer entidade que percebe seu ambiente através de sensores (ex.: microfone, teclado, mouse, ...) e age sobre ele através de atuadores (ex. vídeo, alto-falante, impressora, braços, ftp, ...) (RUSSELL; NORVIG, 1995). Pode-se pensar, então, em um agente como um sistema de computação que, situado em algum ambiente, seja capaz de tomar ações autônomas sobre aquele ambiente a fim de alcançar seus objetivos.

A adequação do paradigma de agentes para o desenvolvimento de ambientes de aprendizagem pode ser demonstrada pelo poder de abstração desse paradigma para se construir aplicações complexas. Esse paradigma permite a modelagem de entidades autônomas que, juntas, cooperam para atingir um objetivo comum. Ambientes de aprendizagem podem naturalmente ser observados como organizações onde atuam agentes humanos e agentes artificiais para se atingir um objetivo comum (MENEZES *et al.*, 1999). Essa distribuição do controle reduz a complexidade da aplicação, além de fornecer melhores meios de conceituar e implementar uma aplicação. De acordo com Bond e Gasser, citados por Jennings e Wooldridge (1998), três importantes características do domínio são freqüentemente

citadas para se adotar a tecnologia do agente:

- dados, controle, especialidade ou recursos são inerentemente distribuídos. Quando o domínio envolve um número distinto de entidades de resolução de problemas (ou dados origem) que está física ou logicamente distribuído (em termos de seus dados, controle, especialidade ou recurso), e que necessitam interagir uns com os outros a fim de resolver problemas, os agentes podem, freqüentemente, fornecer uma solução eficaz. Em casos como esse, os agentes fornecem uma maneira natural de modelar o problema: as entidades do mundo real e suas interações podem ser diretamente traçadas como agentes autônomos com seus próprios recursos e especialidades, e que podem interagir com outros a fim de executar suas atividades.

- o sistema é naturalmente considerado como uma sociedade de componentes autônomos cooperando. Por exemplo, um programa que filtra e-mails pode ser apresentado ao seu usuário através da metáfora de um assistente digital pessoal (MAES, 1995), e um *software* de escalonamento pode naturalmente ser apresentado como um agente autônomo, social que pode interagir com outros agentes semelhantes conforme o interesse do usuário;

- o sistema contém os componentes legados que devem ser feitos para interagir com outros, possivelmente novos componentes. Grandes organizações possuem muitas aplicações de software (especialmente sistemas de informação) que executam funções críticas da organização. Para manter o ritmo com as necessidades de negócio, esses sistemas devem ser periodicamente alterados. Entretanto, modificar os sistemas *legados* é, em geral, muito difícil: as operações internas e a estrutura do sistema se tornam *corrompidas* com o passar do tempo, projetos e documentação são perdidos e indivíduos com a compreensão do software saem da organização. Reescrever esses softwares tende a ser uma tarefa proibitivamente cara e freqüentemente impossível. A única forma de manter esses sistemas úteis é incorporá-los a uma grande comunidade cooperativa, na qual eles podem ser explorados por outras porções de software.

Embora a tecnologia de *software* tenha um importante papel a desempenhar no

desenvolvimento de aplicações, os aspectos de suas limitações devem ser analisados. Muitas aplicações que atualmente usam agentes podem ser construídas com técnicas sem agentes. O simples fato de que um domínio particular de um problema tenha dados de origem distribuídos ou envolva sistemas legados não implica, necessariamente, que uma solução baseada em agentes seja a mais plausível. Como com todo projeto de sistemas, a escolha é determinada por vários fatores. Assim, é importante identificar as situações nas quais uma solução baseada em agentes deve ser *considerada*, em oposição aquelas nas quais ela deve ser descartada.

Além das questões relacionadas à adequação da solução baseada em agentes para um problema específico, Jennings e Wooldridge (1998) ressaltam que o paradigma de agentes conduz a uma variedade de problemas. Entre esses problemas, estão (JENNINGS; WOOLDRIDGE, 1998):

- **Controlador no sistema**. Uma solução baseada em agentes não é apropriada para domínios nos quais restrições globais devem ser mantidas, em domínios onde uma resposta em tempo real deve ser garantida, ou nos quais *deadlocks* ou *livelocks* devem ser evitados;

- **Perspectiva global**. As ações dos agentes são, por definição, determinadas a partir de um estado local do agente. De qualquer maneira, uma vez que em quase qualquer sistema de agentes realístico, o conhecimento global completo não é uma possibilidade, isto pode significar que os agentes tomam decisões globais sub-ótimas. A questão de reconciliar a tomada de decisão baseada no conhecimento local, com o desejo de alcançar desempenho globalmente ótimo, é a questão base na pesquisa de sistemas multiagentes;

- **Confiança e Delegação**. Para que as pessoas aceitem a idéia de delegar tarefas aos agentes, elas devem primeiramente *confiar* neles. As pessoas e a organização devem estar acostumadas com a noção de componentes de software autônomos, uma vez que eles serão muito usados. Os usuários têm que ganhar confiança nos agentes que trabalham em seus nomes e este processo leva tempo. Durante esse período, o agente deve fazer um balanço

> entre continuamente pedir orientação (e, sem necessidade, distrair o usuário) e nunca pedir orientação (exercendo sua autoridade). Falando claramente, o agente deve conhecer as suas limitações.

São vários os aspectos relacionados a agentes que podem ser vantajosos no desenvolvimento de ambientes de aprendizagem. Segundo Eusébio, citado por Giraffa (1999), algumas vantagens que podem ser apontadas são as seguintes: a modularidade (facilidade na construção de um sistema formado por módulos quase independentes); a mudança da arquitetura nos computadores (o computador tende a ser formado, não por um único processador com uma grande memória, mas por um grupo de processadores, cada um com a sua própria memória, o que implica em uma melhor adaptabilidade desses sistemas); os raciocínios heterogêneos (o problema pode ser formado por vários subproblemas, aos quais correspondem diferentes formalismos na representação do conhecimento); as perspectivas múltiplas (muitos problemas são mais facilmente resolvidos se podem ser vistos segundo várias perspectivas); os problemas distribuídos (os problemas podem ter os seus dados disponíveis em várias localizações físicas distintas) e a confiança (a resolução de um problema pode continuar, ainda que ocorra falha em um dos agentes).

Adicionalmente a essas características, também podem ser citados: a cooperação (característica fundamental em ambientes de aprendizagem cooperativa); a facilidade de comunicação; a adaptabilidade; a flexibilidade e a redução da complexidade de produzir um modelo do mundo real através do uso de uma abstração de alto nível.

No contexto de arquiteturas, são várias as propostas para ambientes de aprendizagem. Exemplos são (LABIDI *et a*., 2000), (MENEZES *et al*., 1999) e (BRITO *et al*., 2000).

Para melhor compreender as facilidades que a tecnologia de agentes incorporam aos ambientes educacionais, o QUADRO 2 apresenta algumas das principais características dos ambientes educacionais e como a tecnologia de agentes poe dar suporte a essas características.

QUADRO 2
Agentes em ambientes de aprendizagem

Características	Agentes em ambientes de aprendizagem
Adaptabilidade	Cada módulo no ambiente (que pode ser representado por um ou vários agentes) pode ser dotado de uma grande capacidade adaptativa. Assim, o sistema se adapta a novas situações, modificando suas decisões, escolhendo novos métodos de raciocínio etc. O sistema pode alterar seu comportamento conforme as ações do aprendiz, adaptando-se a ele.
Representação do aprendiz	Bases de conhecimento de agentes podem ser modeladas para representar as ações e intenções dos aprendizes. Para isso, o modelo BDI (KINNY *et al*, 1996) mostra-se bastante adequado. Um modelo do aprendiz baseado em estados mentais permite uma descrição qualitativa do aprendiz (GIRAFFA, 1999).
Domínio de conhecimento	Pode-se definir um modelo multidimensional do conhecimento que pode ser executado por uma sociedade de agentes, conforme sugere a arquitetura do MATHEMA (COSTA, 1997).
Estratégias de acompanhamento	As estratégias de acompanhamento estão intrinsecamente ligadas ao modelo do aprendiz. Quanto melhor for a representação do modelo do aprendiz, melhor será a seleção da estratégia de acompanhamento. Além disso, a estratégia de acompanhamento (no caso da presença do mediador) pode ser executada por um ou vários agentes no sistema. Uma discussão detalhada sobre o uso de estratégias pedagógicas adaptáveis ao modelo do aprendiz é apresentada por Giraffa (1999), Curilem e Azevedo (2001) e Goulart e Giraffa (2001).
Interações do Aprendiz com o ambiente	Reatividade é uma das características que permite que sistemas multiagentes reajam de diferentes formas às ações do aprendiz (conjuntos de regras podem ser definidos representando as possíveis formas de interação com o aprendiz). Exemplos de trabalhos nessa área são: (CASTRO *et al*., 2001), (SOUZA; MENEZES, 2001), (FRAGA *et al.,* 2001), (SANTOS *et al*., 2001).
Interações entre agentes humanos	Agentes (um ou vários) podem atuar no auxílio da identificação do "par mais capaz" (VIGOTSKY, 1984), além de auxiliar nas interações entre os aprendizes (Souza e Menezes, 2000; Castro *et al*., 2001). Além disso, agentes de interface podem auxiliar os aprendizes nas interações, até mesmo no suporte ao agendamento de encontros, como é proposto por Kozierok e Maes (1993).
Coordenação	O módulo central do sistema pode utilizar as várias técnicas de coordenação, para garantir que os seus módulos (ou agentes) trabalhem de forma coordenada.
Suporte à mediação	Agentes podem auxiliar o mediador em ambientes de aprendizagem, acompanhando e gerando *logs* relacionados à utilização do ambiente pelos aprendizes.

Fonte: BRITO, 2000.

2.6 Considerações finais do capítulo

Este capítulo apresentou alguns ambientes de aprendizagem e suas principais limitações, explorando a importância do uso do paradigma de agentes para a construção e/ou melhoria desses ambientes. A adequação desse paradigma na construção de ambientes de aprendizagem pode ser observada pelo fato desses sistemas proverem uma estrutura semelhante a de uma organização, onde se podem identificar entidades cooperantes que podem ser diretamente modeladas como sendo agentes autônomos, com suas próprias capacidades e potencialidades.

3 AVAUFES UMA FERRAMENTA DE GROUPWARE VOLTADA AO ENSINO A DISTÂNCIA

A proposta do AVAUFES segue os princípios adotados pelos sistemas para apoio aos modelos de educação não presenciais e abertos. Esses modelos têm surgido com o intuito de suprir as limitações do ensino convencional (presencial) tais como, a distância geográfica, o respeito ao ritmo de aprendizagem do estudante, a dificuldade em conciliar horários, entre outras. Dentre as novas alternativas de ensino-aprendizagem estão os modelos não presenciais e abertos, sendo um dos mais conhecidos a Educação a Distância – EAD. A EAD tem sido favorecida pelo avanço e a disseminação de novas tecnologias, especialmente dos recursos multimídias e da Internet (BRITTO *et al*., 2000).

Seguindo essa tendência, o AVAUFES é um ambiente que vem sendo projetado para apoiar a interação de grupos cujos membros estejam separados geograficamente, buscando atender aos seguintes requisitos:

- permitir a criação de grupos nas mais diversas áreas;
- permitir que o mediador/instrutor disponibilize o material didático;
- permitir a criação de grupos de aprendizes;
- ser interativo, utilizando hipermídia e propondo espaços para resolução de problemas;
- disponibilizar mecanismos de comunicação síncronos (bate-papo, conferências, etc.) e assíncronos (E-mail, mural, fóruns, etc.). Tais interações ficam registradas, de forma a serem recuperadas quando necessário;
- disponibilizar ferramentas que adicionem facilidades para o trabalho em grupo;
- suportar a cooperação, coordenação e comunicação;
- possibilitar a promoção de debates, conferências e seminários que motivem discussões, enriquecendo o aprendizado através das interações entre os

indivíduos/grupos.

- possibilitar a elaboração de projetos que possam ser desenvolvidos por equipes geograficamente separadas, mas que possuem um objetivo comum;
- possibilitar a seleção, usando a área de interesse e o nível de dificuldade, da melhor pessoa para diagnosticar um problema e/ou tirar dúvidas dos aprendizes;
- possibilitar que os aprendizes/grupos divulguem suas próprias soluções;
- disponibilizar um espaço para a divulgação de outras atividades ou eventos;
- disponibilizar acessos diferenciados para professores, ex-alunos, alunos de outros períodos, profissionais, comunidades, monitores, etc;
- disponibilizar ferramentas de busca com características inteligentes. Esse requisito é importante, pois visa minimizar alguns problemas, já bastante conhecidos da Internet, que são: os usuários despendem enorme quantidade de esforço e tempo para encontrar as informações desejadas; os aprendizes podem dispersar-se facilmente durante a navegação, devido à enorme quantidade de ligações ou mesmo devido ao volume de informação disponível, cuja tendência é ainda aumentar;
- incorporar novas ligações de forma dinâmica ao ambiente;
- manter a trilha do progresso do aluno (sites visitados, nível/volume de interações participadas, problemas/exercícios resolvidos, etc.);
- disponibilizar meios para que se possa incorporar ferramentas, jogos e outros produtos ao ambiente;
- monitorar o comportamento do grupo e dos participantes;
- promover interdisciplinaridade.

Para atender a todos esses requisitos, o desenvolvimento do AVAUFES vem acontecendo por meio da modelagem e implementação de propostas que atendem parte desses requisitos. Esses projetos têm sido desenvolvidos e incorporados ao

AVAUFES na medida em que são avaliados e testados. No contexto dessa proposta, os requisitos contemplados são descritos na forma de casos de uso, apresentados a seguir.

3.1 Casos de uso

Os requisitos identificados são apresentados por meio de diagramas de caso de uso. Os casos de uso podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para os desenvolvedores chegarem a uma compreensão comum, com os usuários finais do sistema e com os especialistas de domínio (BOOCH, 1994).

Um ator é um agente que interage com o sistema, um tipo de usuário ou categoria com papel definido, podendo incluir seres humanos, máquinas, dispositivos ou outros sistemas (WOOLDRIDGE; JENNINGS, 1998). Os atores presentes no AVAUFES são descritos a seguir.

a) Aprendiz: são os usuários do sistema que participam dos grupos de aprendizagem com o objetivo de adquirir os conhecimentos e as habilidades estabelecidas nos objetivos do grupo;

b) Mediador: o mediador é responsável pelo grupo ao qual os aprendizes vão estar vinculados. O sistema permite que esse papel seja desempenhado por uma ou mais pessoas, dependendo das permissões concedidas. Esse ator possui acesso ao ambiente de aprendizagem, monitorando o processo de aprendizagem por meio do acompanhamento das interações entre os componentes do grupo, podendo intervir para favorecer a aprendizagem, fornecendo informações significativas para os aprendizes, questionar os aprendizes e propor novos desafios.

c) Administrador: o administrador é responsável pela configuração e administração dos participantes, grupos e ferramentas do ambiente. Seu principal papel manter

o ambiente em ordem, com a disponibilização e configuração das funcionalidades do ambiente para seus usuários.

3.1.1 Caso de uso principal

A FIG. 3 apresenta o diagrama principal do sistema. Nele estão contidos os atores e casos de uso principais do sistema. E ele se decompõe em: RealizarCadastroAmbiente, SolicitarSenhaAcesso, GerenciarDadosPessoais, UtilizarAmbienteEnsino, PesquisarGrupos, SolicitarEntradaGrupo, GerenciarAmbiente e GerenciarGrupo, descritos a seguir.

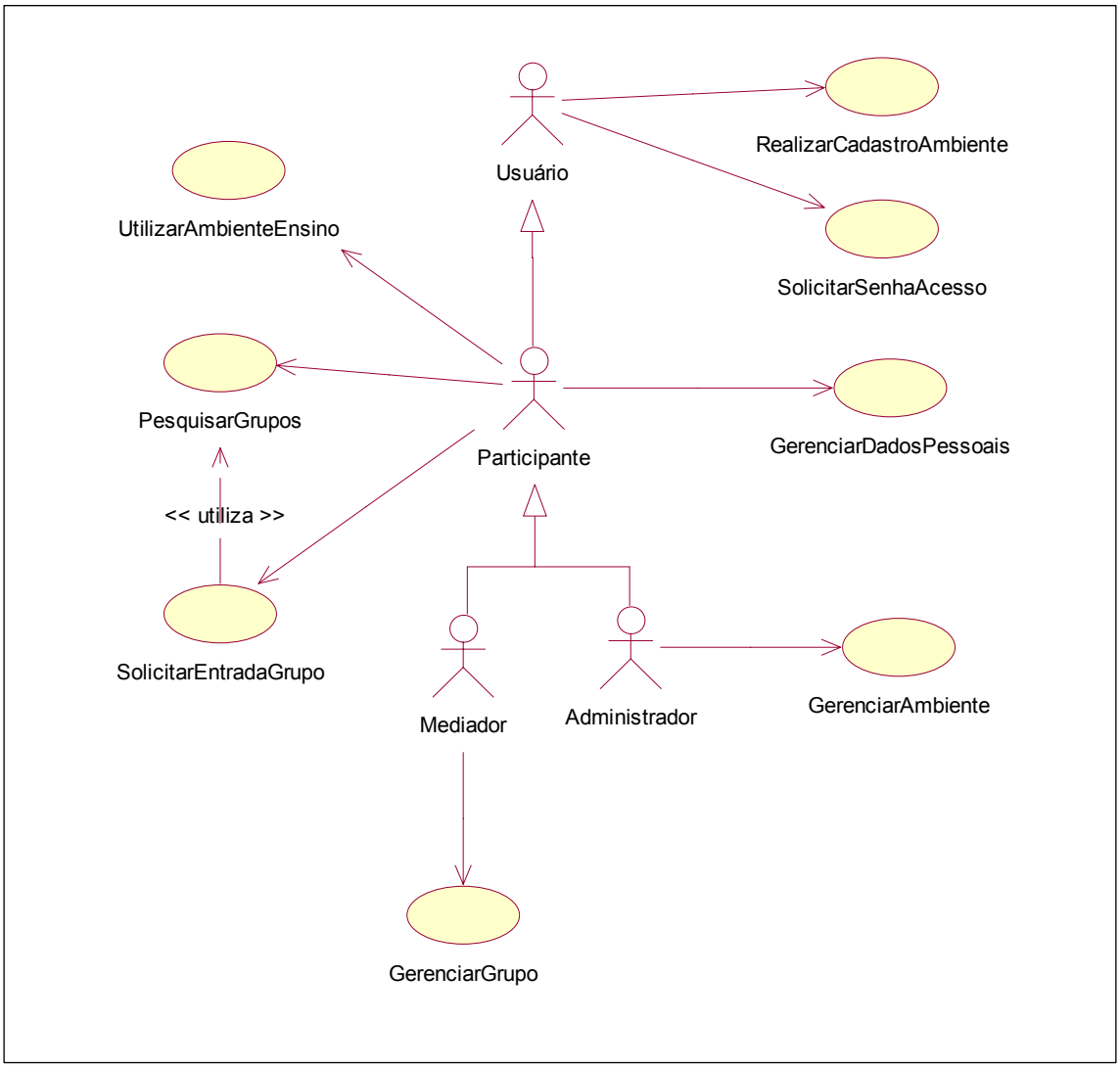

FIGURA 3 - Diagrama do caso de uso principal

3.1.2 Caso de uso RealizarCadastroAmbiente

O usuário poderá realizar seu cadastro no sistema, bastando que o mesmo informe: seu nome, endereços, telefones, data de nascimento, login e senha. A qualquer momento, o usuário também poderá alterar essas informações e em caso de esquecimento do login e/ou senha, solicitar que os mesmos sejam enviados para seu e-mail pessoal acionando o caso de uso Solicitar SenhaAcesso.

3.1.3 Caso de uso SolicitarSenhaAcesso

O usuário poderá a qualquer momento solicitar seu login e senha para acesso ao ambiente no caso de esquecimento destas informações, bastando que este forneça para o sistema seu e-mail informado no momento do cadastro no ambiente como descrito no caso de uso RealizarCadastroAmbiente.

3.1.4 Caso de uso GerenciarDadosPessoais

Este caso de uso representado na FIG. 4 é utilizado pelo participante do sistema e é decomposto em AlterarDadosPessoais, AlterarLoginSenha, AdicionarEndereço, AlterarEndereço, ExcluirEndereço, AdicionarTelefone, AlterarTelefone, ExcuirTelefone.

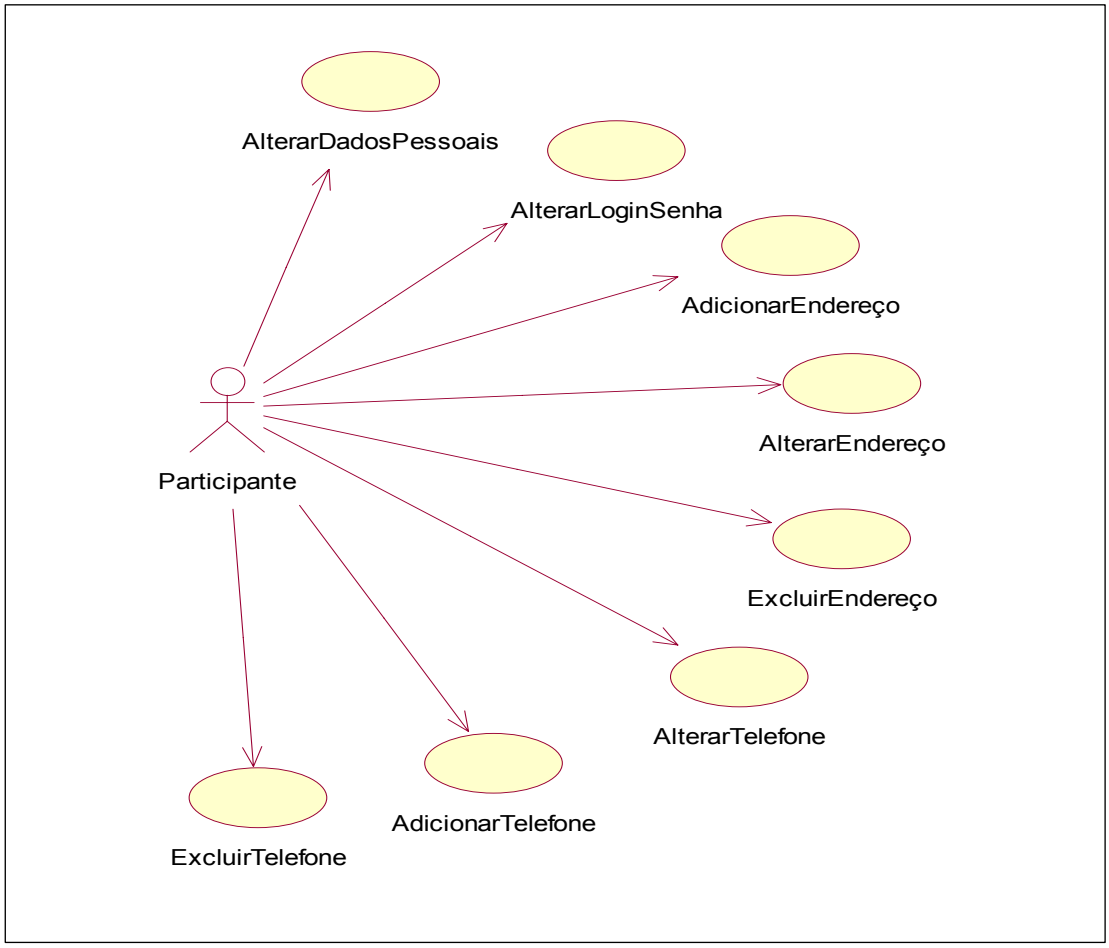

FIGURA 4 - Diagrama do caso de uso GerenciarDadosPessoais

3.1.4.1 Alterar Dados Pessoais

O participante a qualquer momento poderá alterar suas informações pessoais, bastando que o mesmo informe: seu nome, rg, emails, cpf, data de nascimento.

3.1.4.2 Alterar Login Senha

O participante a qualquer momento poderá alterar seu login e senha no ambiente, bastando que o mesmo informe os novos valores para o sistema.

3.1.4.3 Adicionar Endereço

O participante a qualquer momento poderá adicionar seus endereços de contato no ambiente, bastando que o mesmo informe: os dados de sua rua, numero, bairro, cidade, estado, cep.

3.1.4.4 Alterar Endereço

O participante a qualquer momento poderá alterar seus endereços de contato no ambiente, bastando que o mesmo informe os novos valores para seus dados de rua, numero, bairro, cidade, estado, cep.

3.1.4.5 Excluir Endereço

O participante a qualquer momento poderá excluir seus endereços de contato no ambiente, bastando que o mesmo selecione o endereço a ser excluído.

3.1.4.6 Cadastrar Telefone

O participante a qualquer momento poderá cadastrar seus telefones de contato no ambiente, bastando que o mesmo informe os dados de seu ddi, ddd, número.

3.1.4.7 Alterar Telefone

O participante a qualquer momento poderá alterar seus telefones de contato no ambiente, bastando que o mesmo informe os novos dados de seu ddi, ddd, número.

3.1.4.8– Excluir Telefone

O participante a qualquer momento poderá excluir seus telefones de contato no ambiente, bastando que o mesmo selecione o telefone a ser excluído.

3.1.5 Caso de uso UtilizarAmbienteEnsino

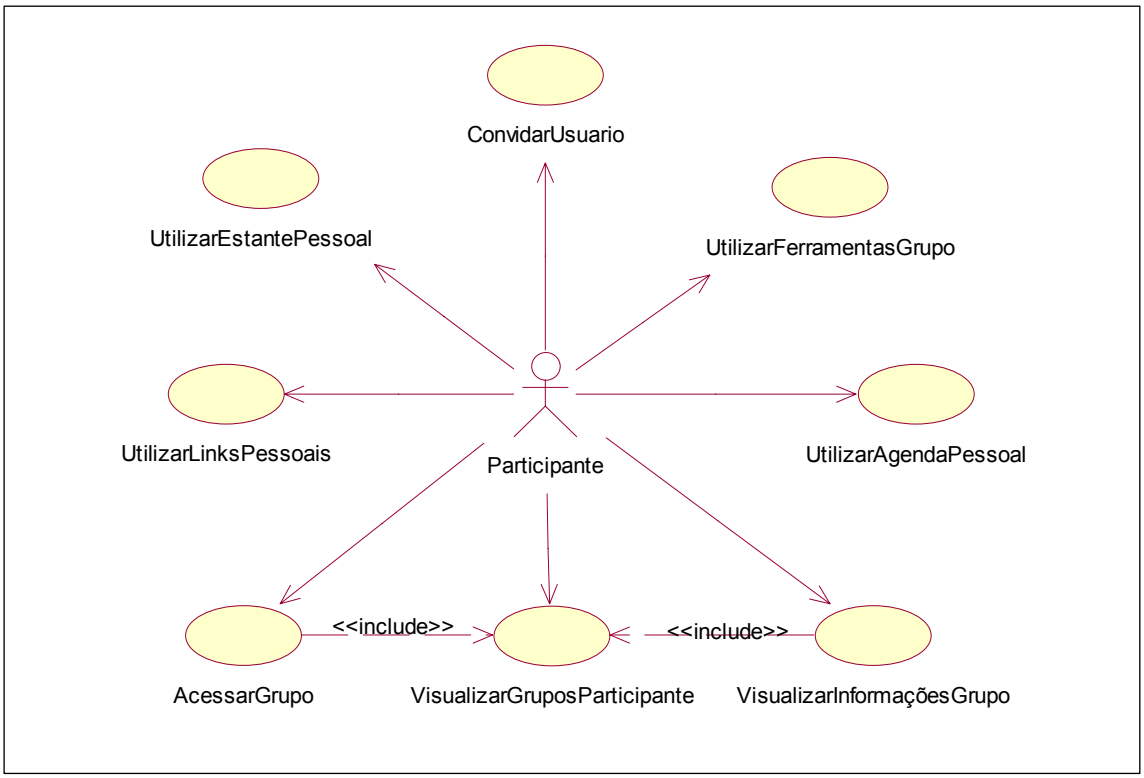

FIGURA 5 - Diagrama do caso de uso UtilizarAmbienteEnsino

Este caso de uso representado pela FIG. 5 compreende as funcionalidades disponibilizadas pelo ambiente, sendo utilizado pelos participantes e é decomposto em: VisualizarGruposParticipante, VisualizarInformaçõesGrupo, AcessarGrupo, UtilizarFerramentasGrupo, UtilizarAgendaPessoal, UtilizarEstantePessoal, UtilizarLinksPessoais, ConvidarUsuario.

3.1.6 Caso de uso UtilizarFerramentasGrupo

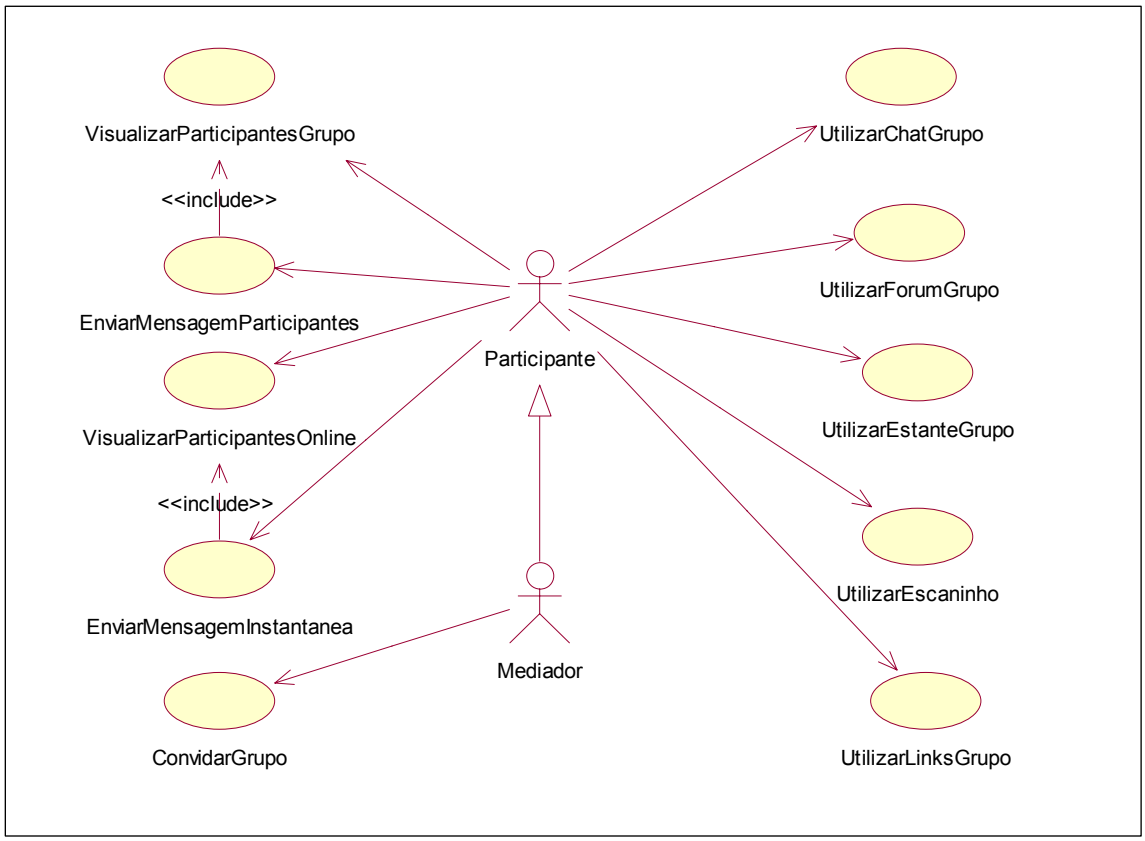

FIGURA 6 - Diagrama de caso de uso UtilizarFerramentasGrupo

A FIG.6 apresenta o caso de uso Utilizar FerramentasGrupo que compreende as funcionalidades disponibilizadas pelo ambiente, para a participação e interação entre os participantes dos grupos e é decomposto em: VisualizarParticipantesGrupo, EnviarMensageParticipante, VisualizarParticipantesOnline, EnviarMensagemInstantânea, UtilizarChatGrupo, UtilizarForumGrupo, UtilizarEstanteGrupo, UtilizarEscaninho, UtilizarLinksGrupo, ConvidarGrupo.

3.1.7 Caso de uso PesquisarGrupos

Este caso de uso é utilizado pelos Participantes do ambiente e tem como finalidade permitir que estes pesquisem os grupos disponíveis para participação e visualizem suas informações antes da realizarem a solicitação de entrada no grupo através do caso de uso SolicitarParticipaçãoGrupo.

3.1.8 Caso de uso SolicitarParticipaçãoGrupo

Este caso de uso é utilizado pelos Participantes do ambiente e tem como finalidade permitir que estes solicitem sua entrada como participantes de um determinado grupo.

3.1.9 Caso de uso GerenciarAmbiente

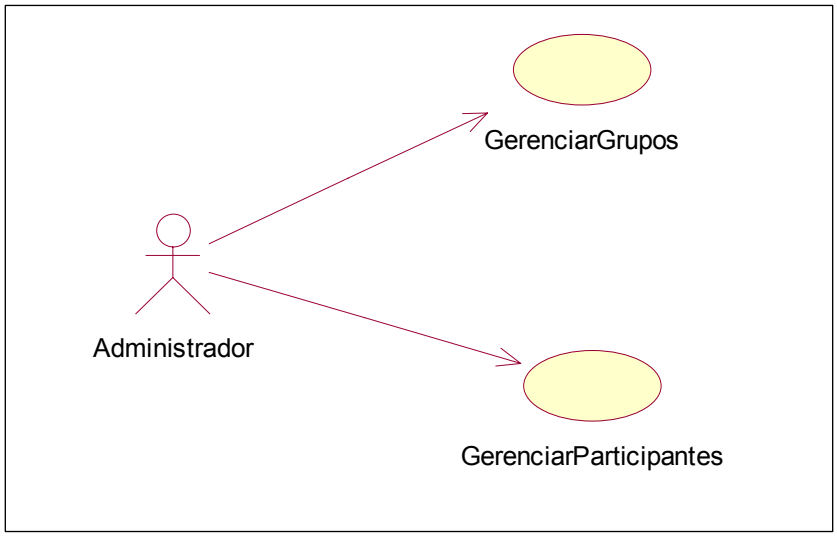

FIGURA 7 - Diagrama de caso de uso GerenciarAmbiente

Este caso de uso representado pela FIG.7 compreende toda parte de administração, gerenciamento e manutenção das informações do sistema, participantes e grupos. É utilizado pelo Administrador e decomposto em: GerenciarGrupos, GerenciarParticipantes.

3.1.9.1 Caso de uso GerenciarGrupo

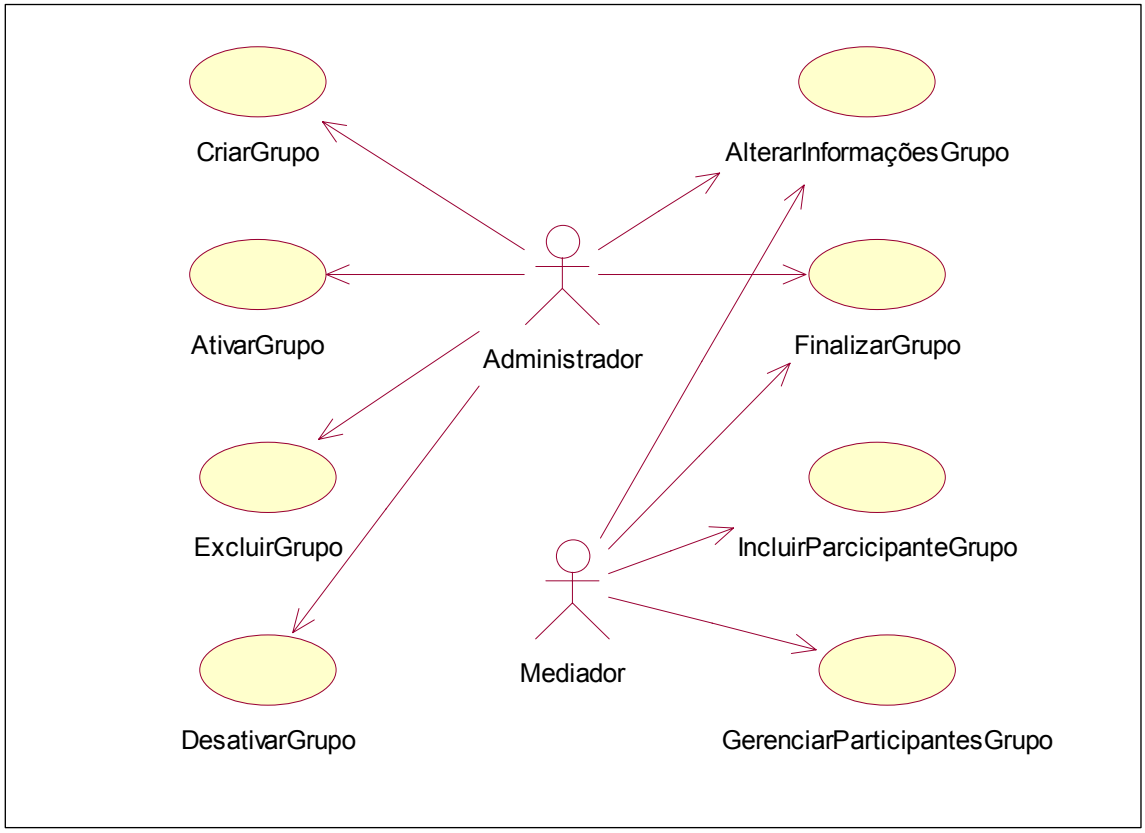

FIGURA 8 - Diagrama de caso de uso GerenciarGrupo

A FIG.8 apresenta o caso de uso do gerenciamento das informações dos grupos desde sua criação até sua finalização. Ele é utilizado pelos administradores do ambiente e mediadores dos grupos e é decomposto em: CriarGrupo, AvitarGrupo, ExcluirGrupo, DesativarGrupo, FinalizarGrupo, AlterarInforaçõesGrupo, IncluirParticipanteGrupo, GerenciarParticipantesGrupo. Com ele administradores e mediadores podem controlar todas as informações dos grupos, e seus participantes. O administrador poderá controlar todos grupos do ambiente, ao contrário do mediador que poderá controlar apenas os grupos cujo seu papel esteja como de mediador.

3.1.9.2 Caso de uso GerenciarParticipantes

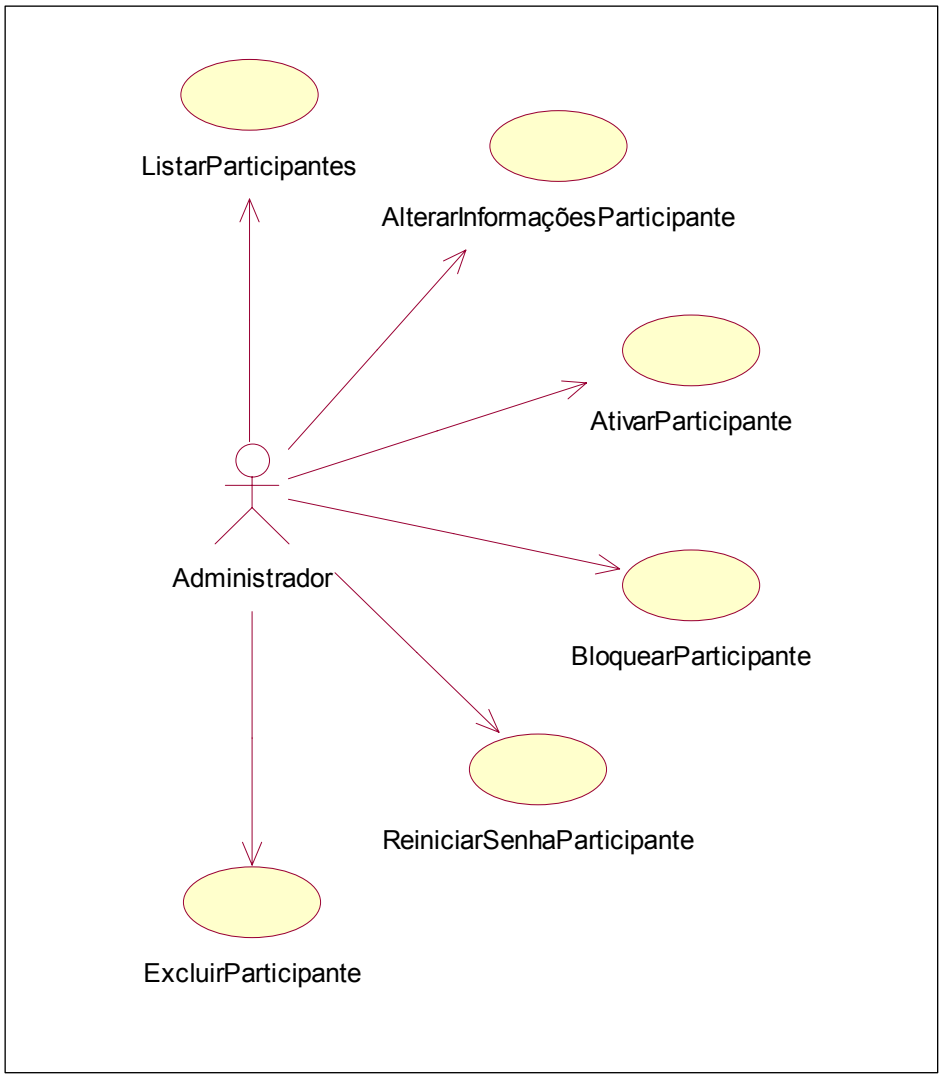

FIGURA 9 - Diagrama do caso de uso GerenciarParticipantes

A FIG.9 apresenta o caso de uso relacionado ao gerenciamento dos participantes do ambiente e é decomposto em: ListarParticipantes, AlterarInformaçõesParticipante, AtivarParticipante, BloquearParticipante, ReiniciarSenhaParticipante, ExcluirParticipante. Esse caso de uso permite que os administradores acompanhem todos os participantes do ambiente.

3.2 Análise orientada a objetos

No processo de desenvolvimento de um software o primeiro passo consiste na definição do problema a ser solucionado e das suas restrições, considerando as especificidades do ambiente de utilização e as características dos usuários.
Na modelagem do AVAUFES optou-se pelo paradigma orientado a objetos, em função das características de encapsulamento, modularidade, hierarquia e abstração. A análise orientada a objetos, visualiza o mundo como uma coletânea de objetos que interagem entre si, apresentam características próprias que são representadas pelos seus atributos (dados) e métodos (processos).

De acordo com Yourdon *et al.* (1995) existem dois argumentos primários em defesa do uso do paradigma orientado a objetos: produtividade e qualidade. A principal razão pelo qual a orientação a objetos aumenta os níveis de produtividade no desenvolvimento de um software provém do extensivo reuso de classes[1] e objetos[2]. O paradigma orientado a objetos alcança o reuso através das bibliotecas de classes e objetos, em particular através dos conceitos de encapsulamento e herança. Do ponto de vista tecnológico, o reuso na orientação a objetos é mais fácil de conseguir do que em outras metodologias para desenvolvimento de softwares. Além disso, ainda de acordo com Yourdon *et al*. (1995), objetos tendem a ter uma alta coesão porque eles encapsulam códigos e dados e, portanto, as chances de um objeto desenvolvido para um determinado sistema poder ser usado em outro contexto são boas. O reuso também reduz o tempo necessário para testes (porque as classes reusáveis e objetos já foram testados) e para projeto (porque não se perde tempo pensando em como implementar classes e objetos).

---

[1] "Uma classe é um conjunto de objetos que compartilham uma estrutura comum e um comportamento comum." (BOOCH *et al.*. 1999)
[2] "Um objeto apresenta estado, comportamento e identidade; a estrutura e comportamento de objetos similares são definidos em suas classes comuns; os termos instância e objeto são permutáveis (similares)". (BOOCH *et al.*, 1999)

Além das características citadas, a orientação a objetos acrescenta algumas características interessantes e desejáveis ao processo de desenvolvimento de um *software*, tais como (BOOCH, 1994):

- Visão do mundo real mais adequada através da utilização de objetos, com conseqüente redução do "*gap* semântico";
- Desenvolvimento incremental e evolutivo, característica extremamente desejável para que um produto possa ser desenvolvido em etapas ou por equipes distintas;
- Reuso. Devido à reusabilidade, muitas vezes é possível reaproveitar parcelas de código, projetos ou mesmo de especificações de requisitos na construção de outras partes do sistema, ou até mesmo em outros sistemas;
- O conceito de objetos e sua descrição em classes, incorporando dados e operações, constituem critérios de modularização altamente efetivos e propiciam o encapsulamento, mantendo alta coesão do sistema.

A notação comumente utilizada para a representação dos modelos na orientação a objetos é a proposta por Booch, Rumbaugh e Jacobson denominada de linguagem de modelagem unificada - UML[3]. Segundo Booch *et al*. (2000), a "UML é uma linguagem-padrão para a elaboração da estrutura de projetos de software". A UML é usada no desenvolvimento dos mais diversos tipos de sistemas e é aplicada nas diferentes fases do desenvolvimento de um sistema, desde a especificação de requisitos até a finalização com a fase de testes.

Ainda de acordo com Booch (1994), a UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir desde sistemas de informação corporativos, até aplicações baseadas na Web e sistemas complexos de tempo real.

A seguir são apresentados os produtos gerados pela análise baseada na orientação a objetos.

A FIG.10 apresenta o diagrama de classes, contendo os elementos estáticos, ou

---

[3] UML – *Unified Modeling Language*

seja, as classes que compõem o sistema e as relações entre as mesmas.

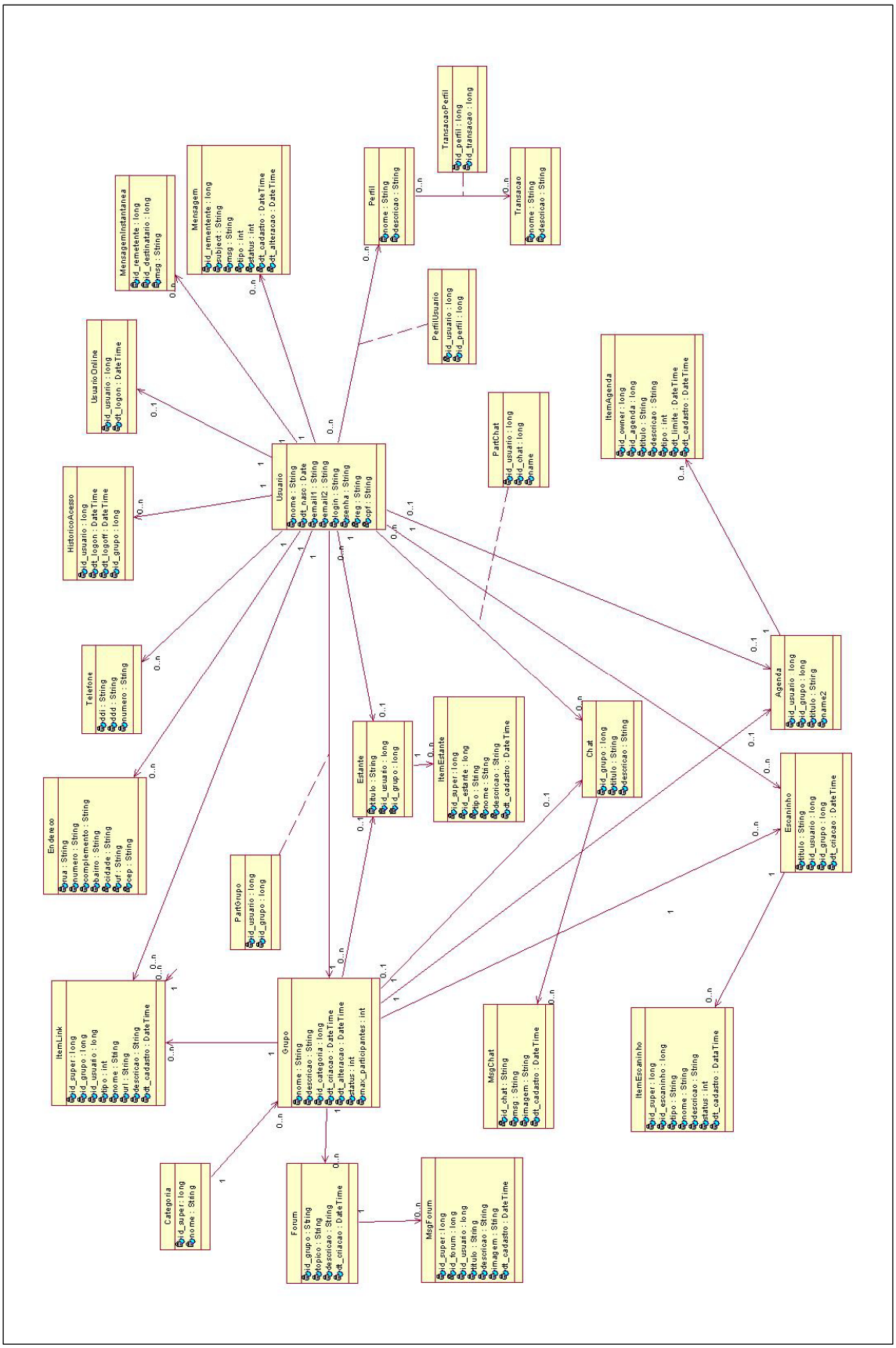

FIGURA 10 - Diagrama de classes do AVAUFES

O diagrama de classes (FIG.10) tem como objetivo especificar em modelo ideal, as classes envolvidas na construção dessa versão do sistema AVAUFES.

3.3 Diagramas de seqüência

Este tópico tem como principal objetivo descrever os principais diagramas de seqüência do sistema AVAUFES.

Um diagrama de seqüência descreve a maneira como os grupos de objetos colaboram em algum comportamento ao longo do tempo. Ele registra o comportamento de um único caso de uso. Ele exibe os objetos e as mensagens passadas entre esses objetos no caso de uso. Um projeto pode ter uma grande quantidade de métodos em classes diferentes. Isso torna difícil determinar a seqüência global do comportamento. Esse diagrama é simples e lógico, a fim de tornar óbvios a seqüência e o fluxo de controle. Em síntese: o Diagrama de Seqüência diz respeito a uma das ferramentas UML usadas para representar o *comportamento* das operações, métodos *(procedimentos ou funções)* entre objetos de um cenário. Esse diagrama é construído a partir do Diagrama de Casos e Usos. Primeiro, se define qual o papel do sistema (Uses Case), depois, é definido como o software realizará seu papel (Seqüência de operações).

Conseitos do diagrama de sequência:

- **Atores**: São entidades externas que interagem com o sistema e que solicitam serviços, gerando dessa forma eventos que iniciam processos.

- **Objetos**: Representam as instâncias das classes representadas no processo Os objetos são ilustrados como retângulos.

- **Porta**: Indica um ponto em que a mensagem pode ser transmitida para dentro ou para fora do fragmento de interação.

- **Fragmento**: Fragmentos de interação como: Alt (Alternativa), Opt (Opcional),

Break (Parar), Loop (Repetição) e outras.

3.3.1 RealizarCadastroAmbiente

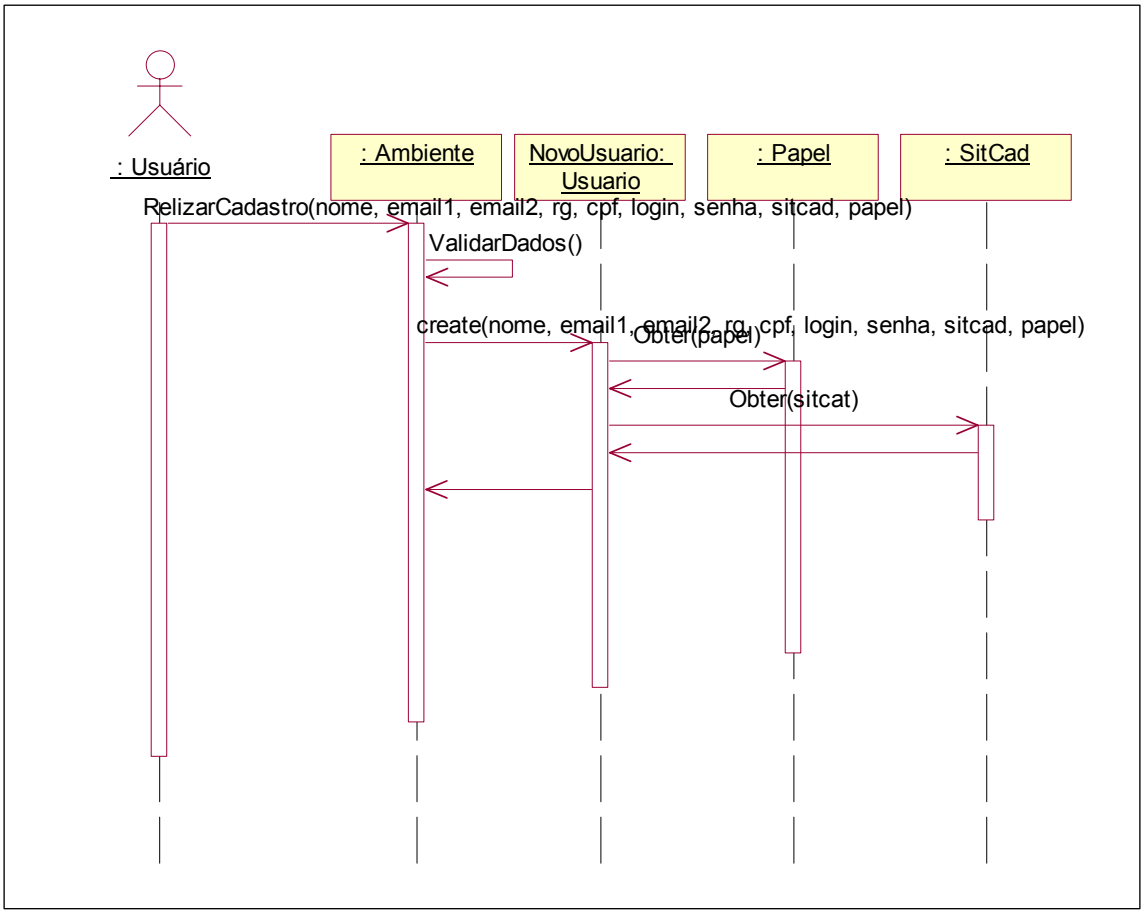

FIGURA 11 - Diagrama de Seqüência RealizarCadastroAmbiente

O diagrama de seqüência representado pela FIG.11 demonstra como os objetos Ambiente, Usuário, Papel e SitCad interagem logicamente para que o processo de cadastro do usuário seja realizado.

Inicialmente o ambiente verifica se os dados informados pelo usuário são válidos, ou seja, se foram fornecidos todas as informações nos formatos aceitos pelo sistema para a criação de novos usuários, após a validação das informações do usuário, elas são passadas para o novo objeto usuário para que ele possa ser criado, ao receber as informações o objeto usuário obtém os objetos Papel e SitCad. Se esses objetos Papel e SitCad forem válidos o novo objeto Usuário é criado no sistema.

3.3.2 SolicitarSenhaAcesso

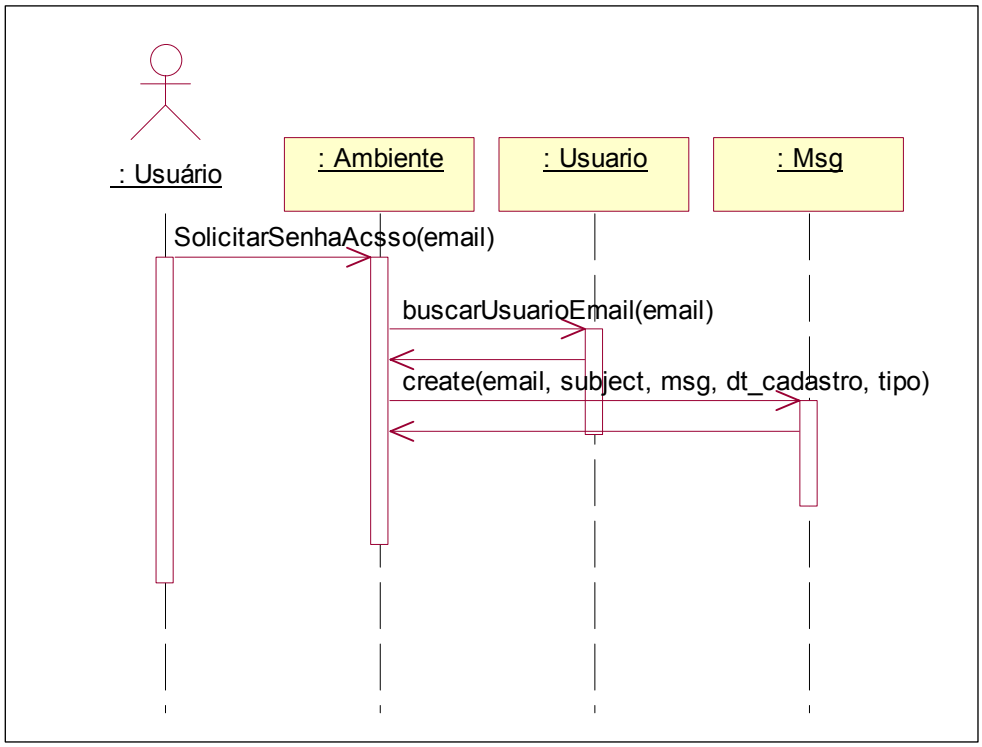

FIGURA 12 - Diagrama de Seqüência SolicitarSenhaAcesso

O diagrama de seqüência representado pela FIG.12 demonstra como os objetos Ambiente, Usuário e Msg interagem logicamente para que o processo de solicitação de senha de acesso seja realizado.

Inicialmente o usuário solicita ao ambiente sua senha de acesso informando para este seu e-mail, o ambiente interage com o objeto Usuário a fim de buscar as informações do objeto usuário que possui determinado e-mail no sistema, após localizado o objeto usuário específico o ambiente interage com o objeto Msg a fim de criar uma mensagem para o usuário com as informações de seu login e senha para a cesso ao ambiente.

3.3.3 ModificarDadosUsuario

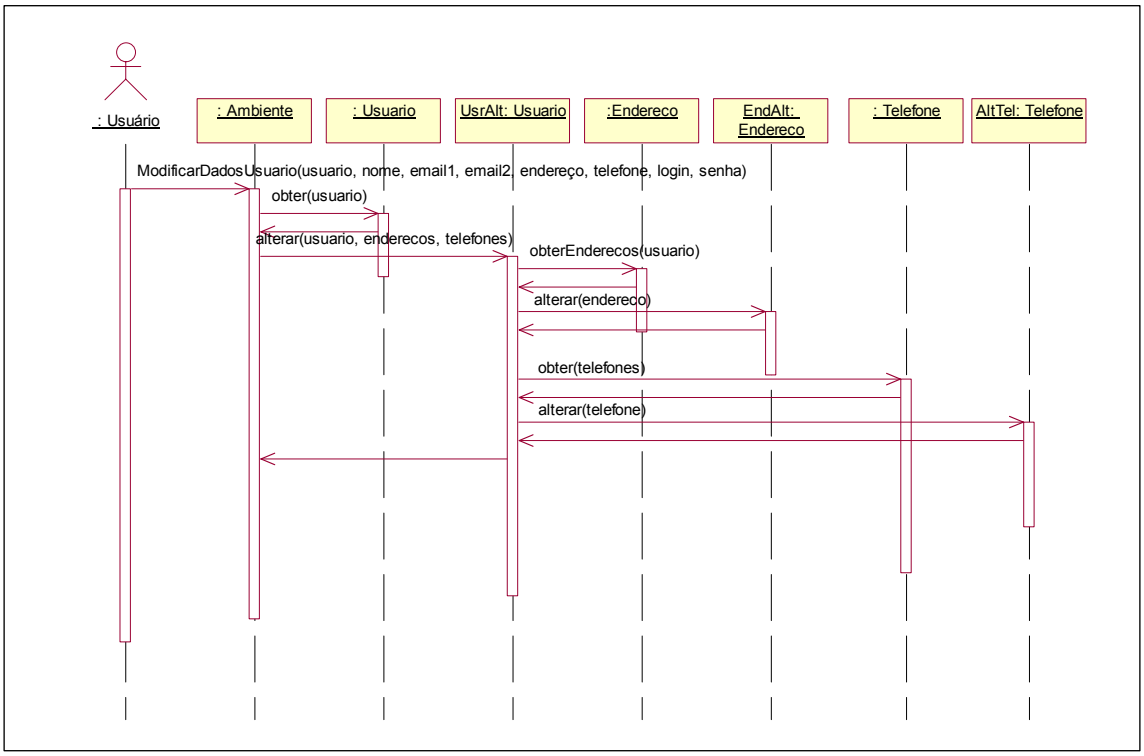

FIGURA 13 - Diagrama de Seqüência ModificarDadosUsuario

O diagrama de seqüência representado pela FIG.13 demonstra como os objetos Ambiente, Usuário, Endereço e Telefone interagem logicamente para que o processo de modificação de dados do usuário seja realizado.

Inicialmente o usuário solicita ao ambiente que seus dados sejam modificados fornecendo para este suas novas informações, o ambiente interage com o objeto Usuário a fim de buscar as informações atuais do objeto a serem alteradas, após localizado o objeto Usuário específico, este interage com os objetos Endereço e Telefone afim de obtê-los para uma eventual necessidade de alteração. Após todos os objetos obtidos, os dados são modificados de acordo com as informações fornecidas pelo usuário.

3.3.4 ModificarPapelUsuarioSistema

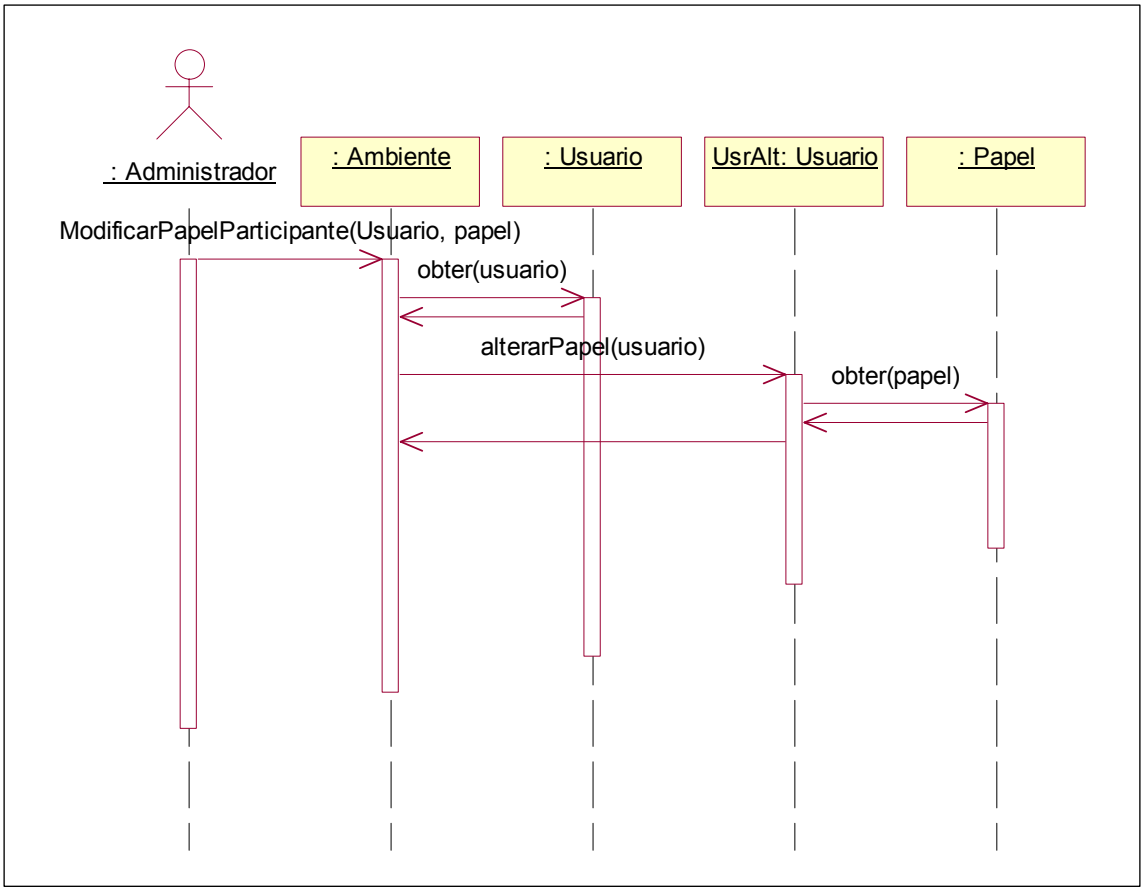

FIGURA 14 - Diagrama de Seqüência ModificaSituacaoUsuarioSistema

O diagrama de seqüência representado pela FIG.14 demonstra como os objetos Ambiente, Usuário e Papel interagem logicamente para que o processo de modificação do Papel do Usuário do sistema seja realizado.

Inicialmente o Administrador solicita ao ambiente que o papel de um determinado usuário seja modificado, fornecendo para ele suas novas informações. O ambiente interage com o objeto Usuário a fim de buscar as informações atuais do objeto a serem alteradas. O objeto Usuário interage com o objeto Papel a fim de obter as informações requeridas. Após todos os objetos obtidos, os dados são modificados de acordo com as informações fornecidas pelo administrador.

3.3.5 IncluirParticipanteGrupo

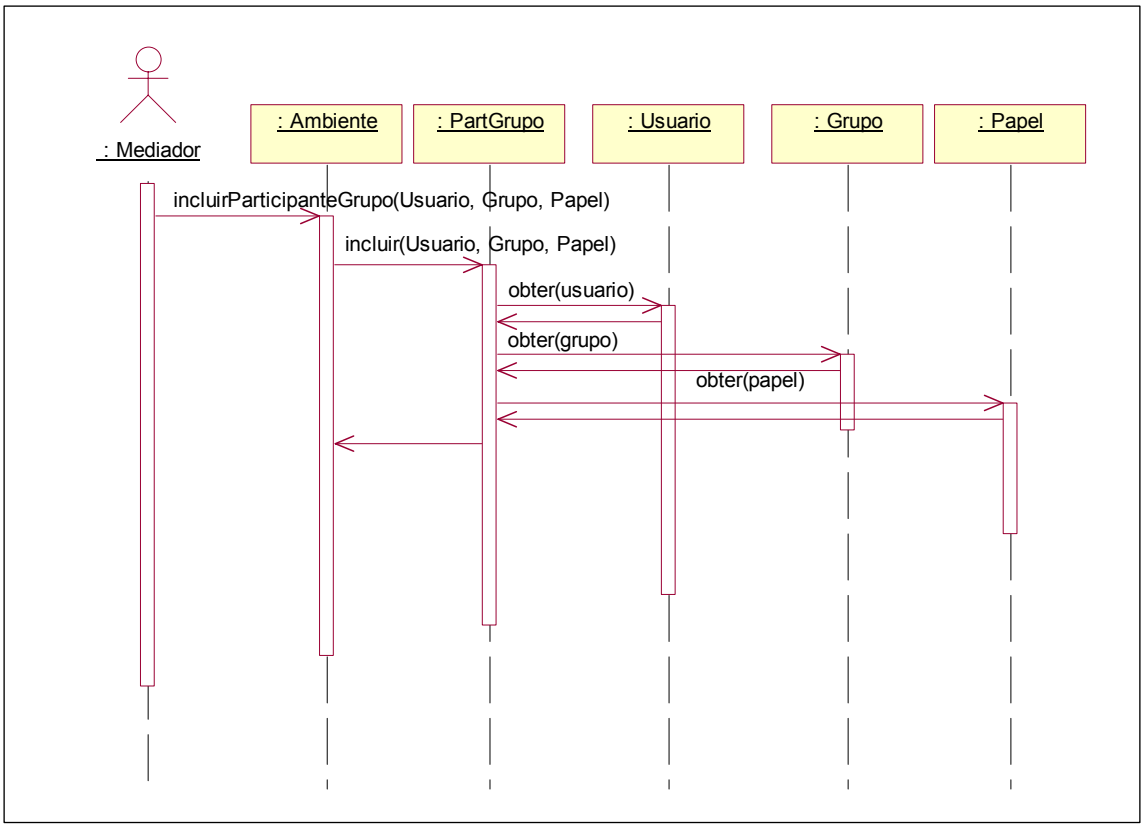

FIGURA 15 - Diagrama de Seqüência IncluirParticipanteGrupo

O diagrama de seqüência representado pela FIG.15 demonstra como os objetos Ambiente, PartGrupo, Usuário, Grupo e Papel interagem logicamente para que o processo de inclusão de participante no grupo seja realizado.

Inicialmente o Mediador solicita ao ambiente que um Usuário seja incluído em um Grupo com um determinado Papel, fornecendo para o ambiente as informações do Usuário, do Grupo e do Papel que esse Usuário irá desempenha no grupo. O ambiente por sua vez interage com o objeto PartGrupo solicitando a criação desse novo participante do grupo. O objeto PartGrupo por sua vez interagem com os objetos Usuário, Grupo e Papel a fim de buscar suas informações atuais e utilizá-las para a inclusão desse novo participante.

3.4 Digramas de estados

Os diagramas de transição de estados mostram a dinâmica interna de uma classe. Apenas os eventos e estados de uma única classe são apresentados nesses diagramas. Entende-se por eventos os fatos que ocorrem em uma classe, provocados por elementos externos (mensagens) ou internos como condições internas da classe que provocam uma troca de estado. Uma classe pode ter vários estados, caracterizados por situações em que a classe se encontra. Um diagrama de estados possui ainda estados especiais como o estado inicial e o estado final e outros estados de controle internos.

Conceitos do diagrama de estados:

- **Estado**: Condição ou situação durante a vida de um objeto na qual ele satisfaz algumas condições, executa algumas atividades ou espera por eventos.

- **Transição**: O relacionamento entre dois estados, indicando que o objeto que está no 1º estado irá passar para o segundo 2º estado mediante a ocorrência de um determinado evento e em certos casos uma condição.

- **Estado inicial**: Estado por onde se começa a leitura de um diagrama de estado.

- **Estado final**: Estado que representa o fim de uma máquina.

3.4.1 Classe usuário

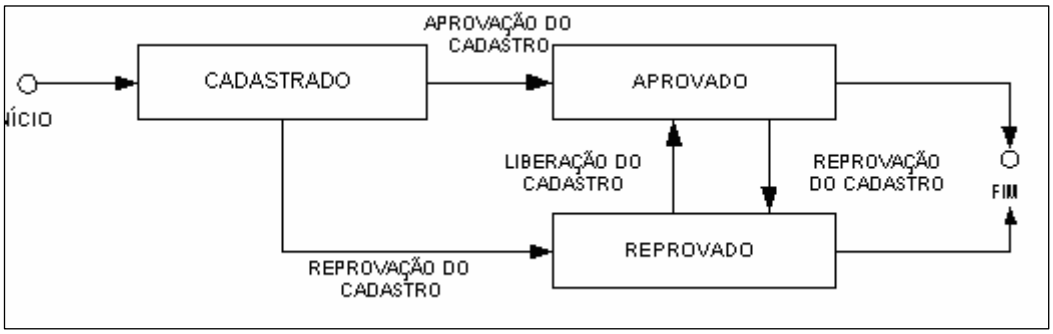

FIGURA 16 - Diagrama de Estados da Classe Usuário

O diagrama de estado representado pela FIG.16, demonstra os estados possíveis que a classe Usuário poderá adquirir durante seu ciclo de vida no sistema.

3.4.2 Classe Grupo

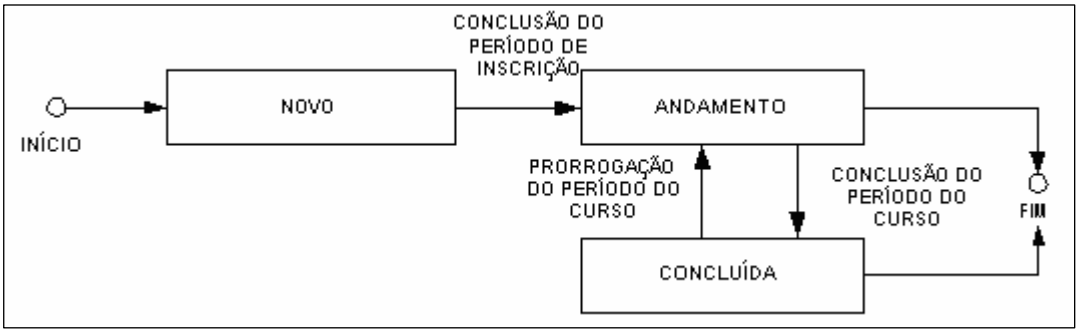

FIGURA 17 - Diagrama de Estados da Classe Grupo

O diagrama de estado representado pela FIG.17, demonstra os estados possíveis que a classe Grupo poderá adquirir durante seu ciclo de vida no sistema.

3.4.3 Classe PartGrupo

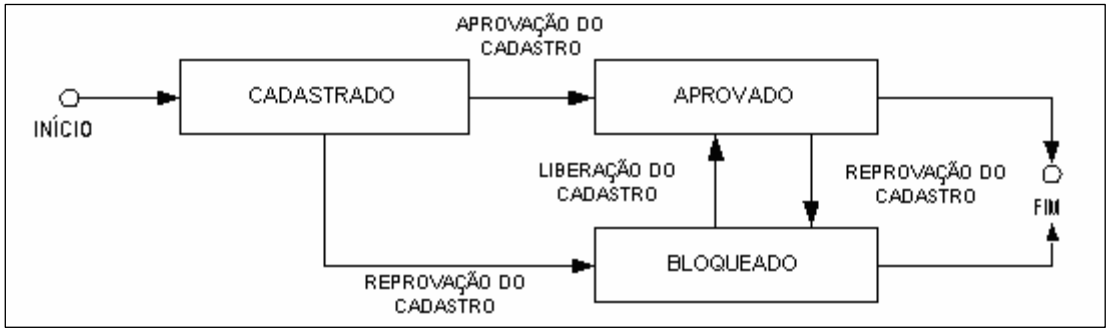

FIGURA 18 - Diagrama de Estados da Classe PartGrupo

O diagrama de estado representado pela FIG.18, demonstra os estados possíveis que a classe PartGrupo pode adquirir durante seu ciclo de vida no sistema. Esta classe representa a participação de um determinado usuário em um grupo do sistema.

4 REQUISITOS PARA UMA PLATAFORMA MULTIAGENTE PARA O GERENCIAMENTO E UTILIZAÇÃO DE AGENTES EM AMBIENTES DE INTERAÇÃO

Um ambiente de aprendizagem deve fornecer suporte às questões relativas à interatividade, cooperação, colaboração e mediação, além de facilitar o uso e a construção da aprendizagem de seus usuários. Este capítulo busca levantar alguns problemas encontrados em sistemas de educação a distância em relação a esses vários aspectos e definir formas de contorná-los.

Os problemas e requisitos identificados a seguir baseiam-se na proposta de Tavares *et al*. (2001). A diferença, fundamentalmente, está no ponto de vista pelo qual os problemas são observados. Tavares *et al*. (2001) analisaram os requisitos sob a ótica do mediador em cursos de educação a distância baseados na pedagogia de projetos. Entretanto, neste capítulo são apresentados os problemas e requisitos não só nesta ótica mais também na ótica do aprendiz em cursos de educação a distância de modo geral.

4.1 Problemas na utilização de ambientes EAD

São várias as ferramentas disponibilizadas pelos sistemas de EAD para que o aprendizado seja facilitado para seu usuário. Apesar da grande utilidade dessas ferramentas não há normalmente nesses sistemas uma forma de auxílio na configuração/manutenção de suas informações, o que faz com que, muitas vezes, o usuário fique perdido no meio de tantas opções, gerando um certo repúdio e/ou desinteresse do mesmo. Além disso, no meio de tantas opções e alternativas o usuário não consegue saber o que já realizou, o que falta realizar e qual a melhor forma de se aproveitar todos os recursos do sistema para seu crescimento pessoal. Alguns desses problemas são identificados a seguir.

4.1.1 Quanto ao acompanhamento da agenda

Alguns problemas normalmente encontrados dizem respeito ao acompanhamento das agendas dos grupos quanto ao controle e visualização dos compromissos. O participante e o próprio mediador não acessam a agenda com a freqüência desejada para visualizar as datas das atividades de seu grupo ou, simplesmente, esquecem de algum compromisso nela contido.

4.1.2 Quanto à interação

Várias são as ferramentas de interação presentes nos sistemas de EAD. Essas ferramentas visam atender os requisitos de interação entre os participantes.

Entre as ferramentas de interação assíncrona que compõem essa camada, estão:

- **Fórum**. Ferramenta para interação assíncrona disponibilizada na *Web*, onde os participantes podem interagir entre si, buscando discutir as mais diversas questões;
- **Mural**. Tem a finalidade de servir como um quadro para fornecer avisos coletivos para todos os participantes;
- **Mecanismo de Decisão**. Permite que os participantes tomem decisões de maneira objetiva. O processo consiste em enviar uma pergunta com algumas opções de resposta, que serão respondidas por meio da escolha de uma ou mais de uma das opções. Na Internet esse recurso é comumente denominado de “enquete”;
- **Listas de Discussão**. Modalidade de comunicação assíncrona onde o envio e recebimento de mensagens acontecem via correio eletrônico (e-mail). Listas de discussão podem ser criadas para uma comunidade e são muito utilizadas em ambientes de EAD para facilitar a interação entre aprendizes-aprendizes e aprendizes-mediadores.
- ***Newgroups***. Têm servidores próprios e são semelhantes aos quadros de avisos. Os usuários podem postar mensagens e responder mensagens postadas por outros usuários.

Entre as ferramentas de comunicação síncrona, têm-se:

- ***Chat*** (sala de bate-papo). Permite aos participantes comunicarem-se de forma síncrona. Esta comunicação ocorre por meio de canais (salas virtuais, também conhecidas como salas de bate-papo). O *chat* deve permitir comunicação coletiva (quando os usuários enviam e recebem mensagens de todos os usuários) e individual (um usuário escolher um integrante específico para comunicar-se diretamente com ele);
- **Mensagens Instantâneas**. Modalidade de envio e recebimento de mensagens em tempo real. Um participante consulta quais outros participantes estão *on-line*, e lhes envia uma mensagem, que será exibida em suas respectivas telas (Por exemplo, o MSN[1]);

Entre os problemas encontrados na utilização dessas ferramentas, têm-se:

- dificuldade de ser avisado sobre algum encontro organizado pelo grupo;
- dificuldade de saber como está sua participação em relação à quantidade de perguntas, respostas e outras participações;
- dificuldade de participar de eventos síncronos.

4.1.3 Quanto ao conteúdo

Em relação ao controle de conteúdos, os maiores problemas encontrados se referem à necessidade de notificação do participante quando disponibilizado novo conteúdo ou quando um conteúdo é modificado.

4.1.4 Quanto à interface

Cada participante tem suas preferências quanto à visualização dos conteúdos e das demais informações disponíveis no ambiente. Uma das dificuldades enfrentadas está na impossibilidade de configurar o ambiente para atender às exigências desses aprendizes no que diz respeito à interface de apresentação.

[1] Programa de computador para mensagens instantâneas, disponível em www.msn.com.

4.1.5 Quanto às observações pessoais

Na forma como os recursos encontram-se atualmente disponibilizados, o participante não possui recursos para incluir anotações pessoais nas informações disponibilizadas (agenda, notas de aula, etc). Esses recursos permitem a personalização do ambiente de acordo com as preferências do aprendiz.

4.2 Requisitos para uma Plataforma Multiagente para o Gerenciamento e Utilização de Agentes em Ambientes Virtuais de Aprendizagem

O objetivo desta seção é especificar os requisitos para uma camada inteligente de apoio às atividades desempenhadas pelos participantes, a partir dos problemas relatados no capítulo anterior. O sistema especificado deve permitir que o participante estruture seu próprio acompanhamento em relação ao grupo. Os requisitos apresentados estão organizados em classes de requisitos apresentadas nas subseções a seguir, conforme sugerem Tavares *et al*. (2001).

4.2.1 Estratégia de acompanhamento

O participante deve poder definir estratégias de acompanhamento de suas atividades no sistema, de acordo com suas particularidades. Entre os requisitos associados a esta categoria, estão:

- permitir ao participante definir sua forma de acompanhamento das atividades, datas, acesso ao sistema, interações com outros usuários;
- uma vez definida uma estratégia de acompanhamento, o ambiente deve auxiliar o participante na definição de outras estratégias, uma vez que já conhece suas preferências;
- permitir o uso de estratégias "*default*". O participante deve poder particularizar uma estratégia para uma ferramenta em particular;
- estratégias de acompanhamento podem ser modificadas a qualquer momento.

Neste trabalho, da mesma forma como em Tavares *et al.* (2001), as estratégias de acompanhamento são tratadas de forma semelhante às estratégias pedagógicas

dos Sistemas Tutores Inteligentes. O problema das estratégias de acompanhamento em ambientes de aprendizagem cooperativa é similar ao problema nos Sistemas Tutores Inteligentes (STIs). As estratégias pedagógicas, no contexto dos STIs, são definidas por um conjunto de regras e/ou planos para alcançar metas específicas (GIRAFFA, 1999). A diferença da proposta deste trabalho está no objetivo das estratégias definidas. Diferentemente do que nos sistemas tutores (onde as estratégias possuem o objetivo de ensinar), na visão apresentada aqui, as estratégias são definidas pelo próprio participante, como forma de tornar o ambiente mais dinâmico e adaptável às suas preferências.

4.2.2 Facilidades de acompanhamento

Deve haver ferramentas de acompanhamento para facilitar o trabalho do participante. Entre os requisitos associados a esta categoria estão:

- permitir ao participante estabelecer critérios de avaliação de seu desempenho em relação aos outros participantes, podendo definir ações para cada um dos critérios estabelecidos. Essas ações podem ser, por exemplo, notificações. Ex: Quando 50% dos participantes do grupo acessarem o *chat* ao mesmo tempo, enviar uma notificação para todos os participantes do grupo entrarem no *chat* – uma vez que pode estar havendo um encontro não agendado previamente;
- permitir ao participante ser informado quando os materiais e informações de seu grupo sofrerem alterações. Ele pode, por exemplo, somente ser informado se algum novo material for incluído (ou modificado) pelo mediador;
- permitir ao participante ser informado sobre a entrega de trabalhos e sua participação nas atividades;
- possibilitar ao participante ter acesso fácil à opinião de outros participantes, bem como sobre os consensos estabelecidos pelo seu grupo;
- automatizar as notificações esperadas.

4.2.3 Facilidades de comunicação

São necessárias facilidades de comunicação do ambiente com o participante e do participante com os outros participantes (aprendizes, mediadores e monitores), tais como: avisos sobre mensagens enviadas pelo mediador; mensagens urgentes;

quantidade de mensagens recebidas; confirmação de entrega de trabalhos; consenso entre os participantes do grupo e etc.

Os principais requisitos associados às facilidades de comunicação são:

- permitir ao participante receber notificações sobre o recebimento de mensagens, sem que necessite acessar o ambiente. O participante pode ser informado somente se a quantidade de mensagens para ele atingir um número estabelecido por ele;
- fornecer mecanismos para que o participante identifique se as mensagens postadas são de caráter formal ou informal, veja estatística prévia[2] sobre as mensagens e possa estabelecer ações para as suas observações.

4.2.4 Facilidades de agendamento

São necessários mecanismos para o agendamento e registro de reuniões, considerando o quórum mínimo e os participantes obrigatórios, bem como, o aviso sobre as reuniões agendadas pelo mediador. Os requisitos associados às facilidades de agendamento são:

- agendar reuniões síncronas levando em consideração as disponibilidades de cada participante e, principalmente, dos participantes obrigatórios, quando for o caso;
- fornecer mecanismos para tratar a questão de quórum mínimo para os encontros síncronos, conforme os critérios estabelecidos pelo mediador;
- permitir ao participante definir ações para os atrasos nas agendas ou sobre a aproximação de datas importantes.

4.3 Plataforma Multiagente para o Gerenciamento e Utilização de Agentes em Ambientes Virtuais de Aprendizagem

Para atender aos requisitos apresentados na seção anterior, esta seção apresenta uma solução em camadas, adicionando a arquitetura dos Ambientes Virtuais de

[2] Estatística prévia é aquela que se faz sem necessidade de acessar o conteúdo de onde os dados são levantados. Por exemplo, para se saber a freqüência de postagens em um fórum, não há necessidade de se acessar o fórum.

Aprendizagem (AVA's) uma camada multiagente. Nessa camada está presente uma organização de agentes montada para facilitar o uso do ambiente pelos participantes e mediadores. A arquitetura resultante está ilustrada na FIG.19.

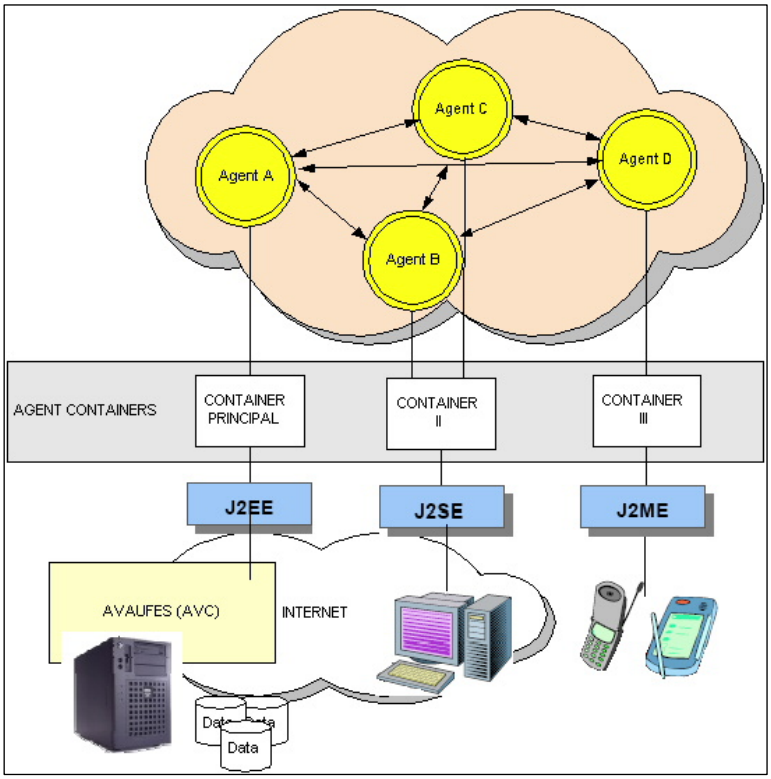

FIGURA 19 - Representação de AVA's integrados à camada Inteligente e a dispositivos de comunicação diversos

A FIG.19 demonstra como a camada inteligente pode ser integrada aos AVA's, assim como a dispositivos diversos a fim de resolver/melhorar a interação desses ambientes e seus usuários/participantes. A camada inteligente é genérica e modularizável o suficiente para se integrar a diversas tecnologias e dispositivos móveis disponíveis atualmente para facilitar a comunicação e integração de seus usuários. Esta integração se faz possível devido ao grande leque de dispositivos hoje que suportam a tecnologia Java, e também a todo o suporte a este tipo de comunicação fornecido pelo *framework* JADE[3], que torna esta comunicação simples e transparente para o sistema, possibilitando com isso uma maior integração entre os usuários destes sistemas, que poderão, por exemplo, receber alertas/avisos de seus compromissos e atividades via celular/palm. Maiores detalhes sobre o

[3] **JADE** é um *middleware* para desenvolvimento e execução de aplicações *peer-to-peer* baseada no paradigma de agentes que pode facilmente trabalhar e interoperar em ambientes de redes tradicionais ou wireless.

framework JADE e suas tecnologias serão dados no Capítulo 5.

4.4 Organização de agentes da camada inteligente

O objetivo da camada de inteligência, inicialmente, é estruturar e organizar um conjunto de agentes que possam auxiliar os aprendizes na utilização dos recursos disponibilizados em ambientes virtuais de aprendizagem, particularmente, no AVAUFES.

A proposta é criar uma camada inteligente que possa monitorar constantemente o ambiente, seus grupos, ferramentas e participantes, mantendo esses participantes informados sobre todas as atividades existentes e sobre a evolução das atividades de seus grupos. Além disso, essa organização de agentes deve auxiliar os participantes e mediadores, informando-os sobre os estados de cada atividade de seus grupos e auxiliando-os no controle e execução de suas atividades.

4.5 Identificação de papéis

Para que a organização alcance os objetivos esperados, um conjunto de papéis, cada qual com a sua responsabilidade, deve ser modelado e projetado. Esses papéis são:

- **Monitor:** Possui a responsabilidade de carregar a definição inicial dos agentes do ambiente e criá-los. Além disso, de acordo com as definições de cada agente, o Monitor monitora o comportamento de cada agente detectando sobrecargas podendo a qualquer momento criar novos agentes para a distribuição de tarefas.
- **ChatVerify:** Possui a responsabilidade de manter os participantes dos grupos informados sobre a situação do *chat* de acordo com as regras definidas por esses participantes.
- **AgendaVerify:** Possui a responsabilidade de manter o participante informado sobre as atividades agendadas pelo mediador para a grupo e sobre seus próprios agendamentos. Essas atividades podem ser, por exemplo, reuniões no chat, encontros presenciais, provas, trabalhos, etc.

- **ForumVerify:** Possui a responsabilidade de manter o participante informado sobre as discussões existentes no fórum do grupo, assim como novas discussões abertas pelo mediador e novas mensagens sobre uma discussão já existente de um assunto de seu interesse.
- **AccessVerify:** Possui a responsabilidade de manter os participantes e os mediadores informados sobre os acessos ao ambiente. Ele pode por exemplo avisar a um determinado participante sobre o acesso dos participantes de seu interesse ou alertar o mediador sobre a falta de acesso dos participantes de seu grupo.
- **MailVerify:** Possui a responsabilidade de verificar as mensagens postadas no ambiente e despachá-las para o Carteiro para que elas sejam distribuídas para seus respectivos destinatários. Além disso, esse Agente poderá fornecer informações estatísticas importantes para os mediadores dos grupos e monitorar conteúdos impróprios de determinadas mensagens.
- **EscaninhoVerify:** Possui a responsabilidade de manter o participante e os mediadores informados sobre a atualização do conteúdo do escaninho do participante. Dessa forma, participante e mediadores poderão ser informados sobre quaisquer atualizações do conteúdo do escaninho.
- **EstanteVerify:** Possui a responsabilidade de manter os participantes e os mediadores informados sobre a atualização do conteúdo da estante do grupo. Os participantes e mediadores poderão ser informados sobre a inclusão de determinados conteúdos que sejam de seu interesse ou a atualização/exclusão deles.
- **Carteiro:** Possui a responsabilidade de receber as mensagens dos agentes executores de atividades (*chat*, fórum, mural, etc) e as enviar para seus respectivos destinatários;

Os agentes identificados e seus relacionamentos são exibidos na FIG.20.

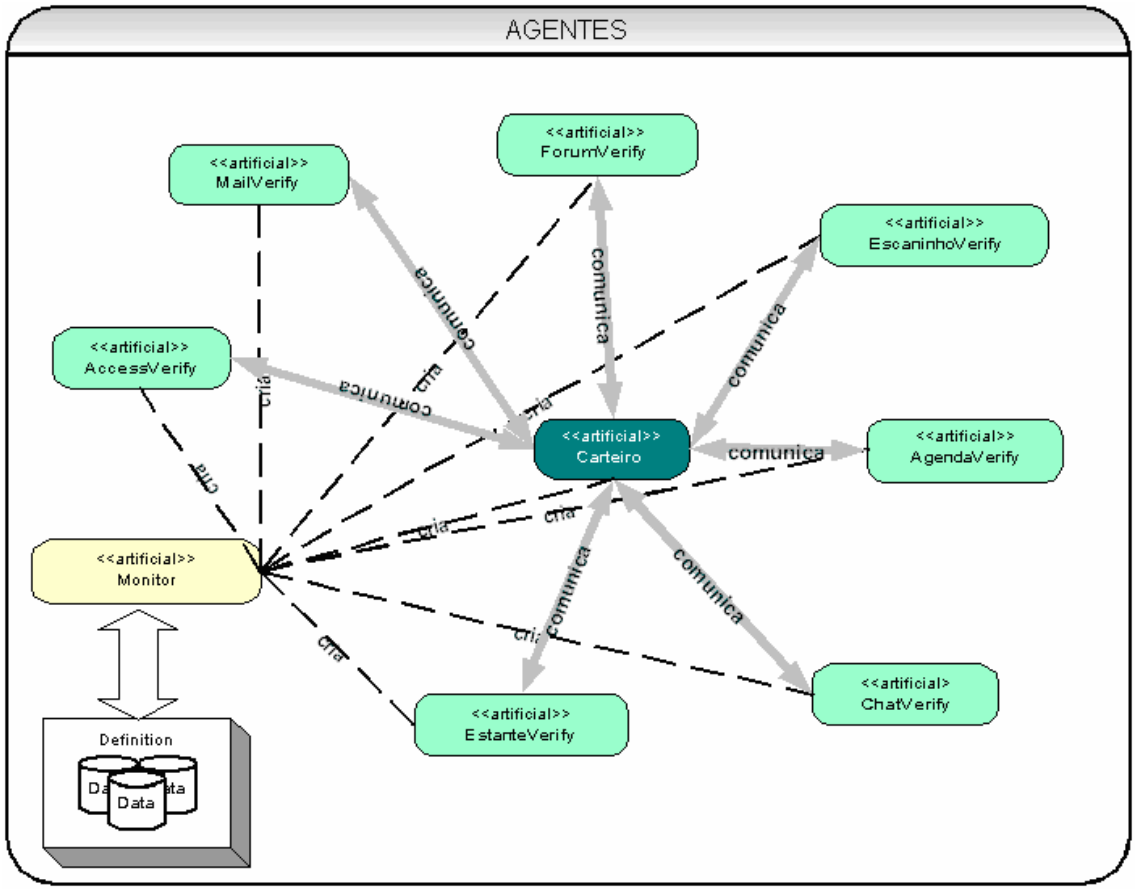

FIGURA 20 - Representação das interações entre os papéis

A FIG.20 representa as possíveis interações entre os agentes do sistema. Em amarelo está o Monitor que é responsável pela criação e monitoração dos agentes do sistema. Em verde escuro esta o Carteiro que se comunica com todos os outros agentes do sistema com o objetivo de recepção e entrega de mensagens com as informações geradas pelos agentes do ambiente. Na solução inicial não há comunicação entre os outros agentes do sistema (verde claro), mais esta comunicação será perfeitamente possível no caso de alguma necessidade futura.

A representação das interações tem como objetivo facilitar a compreensão do funcionamento do sistema.

4.6 Funcionamento da organização de agentes

Quando o ambiente é iniciado, o Monitor carrega as definições dos agentes do ambiente. Além disso, de acordo com essas definições, o Monitor cria e começa a monitorar os agentes do sistema.

Cada agente do sistema possui uma base de regras próprias, definida pelos mediadores e participantes dos grupos. Após os participantes e mediadores definirem suas regras, através da interface *Web* disponibilizada pelo sistema, a organização de agentes passa a interagir para executar as regras definidas.

Cada agente identifica a nova regra e a carrega. Cada regra definida para o agente é interpretada gerando um comportamento (*behaviour*) no agente. Caso uma regra tenha sido alterada, o agente irá detectá-la automaticamente e recarregá-la na próxima verificação.

Cada regra recebida pelo agente gera um determinado tipo de comportamento no agente. Cada agente do sistema pode gerar mensagens de alerta para os participantes e mediadores dos grupos. Essas mensagens podem ser e-mails ou mensagens instantâneas exibidas para o usuário pelo sistema. Para enviar essas mensagens os agentes do sistema se comunicam com o Carteiro que se encarrega de enviar a mensagem para seu destino correspondente.

Nos próximos capítulos são discutidos: as tecnologias envolvidas na concepção do *framework* de agentes; a modelagem e os mecanismos inteligentes incorporados a esses ambientes para torná-los mais adaptativos e flexíveis.

5 MODELAGEM DOS AGENTES

A fase de modelagem conceitual é um passo cada vez mais importante no desenvolvimento de software. Um software bem modelado se torna fácil de implementar, de realizar manutenções e de ser compreendido.

A UML é uma linguagem de modelagem amplamente utilizada no mundo todo para sistemas orientados a objetos. No entanto ela não suporta o conceito de agentes criando assim certa ambigüidade na modelagem. A UML não consegue retratar de forma coerente a troca de mensagens entre os agentes e nem os estados de um agente, sendo uma linguagem apropriada para a modelagem de sistemas orientados a objetos, e um objeto não pode ser tratado de forma igual a um agente. Dessa forma algumas linguagens de modelagem que estendem a UML foram definidas, visando solucionar esses problemas, entre elas a AORML (WAGNER, 2002), que é usada no decorrer deste projeto.

A AORML utiliza uma representação mais detalhada de agentes. Os agentes podem ser agentes humanos, agentes artificiais ou agentes institucionais, onde cada agente possui uma representação diferenciada. Eles podem se comunicar, perceber, agir, realizar um compromisso e satisfazer uma reivindicação. Podendo ser representado de duas formas: uma representação interna, onde são representados os próprios agentes e outra representação externa onde o enfoque é o sistema como um todo (WAGNER, 2003).

5.1 AORML (Agent – Object Relationship Model Language)

A AORML (AOR modeling language - linguagem de modelagem AOR) (WAGNER, 2002) baseia-se no meta-modelo AOR (WAGNER, 2000; WAGNER, 2003). As entidades definidas no AOR são: evento, ação, alegação, compromisso, agente ou

objeto. Todas as entidades são definidas como estereótipos[1] com base na meta-classe *Class* definida pela UML. A AOR define sete tipos de agentes, sendo um deles o agente institucional. Um agente institucional representa organizações e consiste de alguns agentes internos que percebem eventos e executam ações em seu nome, exercendo determinados papéis. Os agentes internos possuem determinados direitos e deveres. A AOR especializa os relacionamentos *association*, *generalization* e *composition* definidos na UML. Além disso, ela define os relacionamentos *does*, *perceives*, *sends*, *receives*, *hasClaim* e *hasCommitment*, vinculando eventos de ação, compromissos, alegações e agentes. Todos esses relacionamentos são definidos como estereótipos com base na metaclasse *Association*.

AORML é uma extensão da linguagem de modelagem unificada (UML) que relaciona agentes e objetos. Ela representa os comportamentos dos agentes, as ações, troca de mensagens que cada agente realiza e determina diferenças entre os agentes do sistema e os objetos. As entidades estão divididas em agentes, eventos, ações, compromissos, reivindicações e objetos.

A AORML consiste de um modelo de análise (externo) e um modelo de *design* (interno).

O modelo externo, como é conhecido o modelo de análise, modela uma visão externa, de quem está observando os agentes e suas interações, considerando o domínio do problema de uma forma abrangente focando sempre no ambiente. No modelo externo as ações são também eventos, as reivindicações são também compromissos, ou seja, os modelos de ações e eventos são representados pelo mesmo diagrama de evento e a representação para reivindicação e compromisso são também representadas pelo mesmo diagrama de compromisso.

O modelo interno, como é conhecido o modelo de *design*, modela o problema na visão de cada agente, como ele se comunica com os outros agentes, como ele é notificado dos eventos no ambiente, como ele enxerga o "mundo". (WAGNER,

[1] **Estereótipo** é a imagem preconcebida de determinada pessoa coisa ou situação.

2002).

No modelo externo temos os seguintes diagramas:

- Diagrama de agentes: destacando cada classe agente, os objetos de classe mais relevantes e as relações entre esses objetos.
- Diagrama de interação de quadros: destacando as classe de eventos e as classes de *commitment*/*claim* que determinam possíveis interações entre dois tipos de agentes. Este diagrama possui quatro tipos de relacionamentos:
    - Relacionamento entre agentes com ações e eventos comunicativos: enviar e receber mensagens são eventos ou ações do tipo que relata ao agente um tipo de relacionamento que gera comunicação.
    - Relacionamento entre agentes com ações e eventos não-comunicativos: fazer e perceber os agentes são eventos que não geram comunicação entre os agentes.
    - Relacionamento entre agentes com compromisso e reivindicação: Esse relacionamento é visualizado com um conector particular que acompanha uma ação ou evento.
    - Relacionamento entre eventos sem ação: Este relacionamento é percebido pelo agente no qual ele não executa nenhuma ação, ou seja, apenas uma chamada a um método.
- Diagrama de seqüência: destacando a interação entre processos. Interação entre processos é uma seqüência de ações e eventos feitos e recebidos pelos agentes que seguem um conjunto de regras. Essa interação pode ocorrer entre o agente e o ambiente ou entre os próprios agentes (WAGNER, 2002).

No modelo interno temos os seguintes diagramas:

- Diagrama de Quadros da Reação: destacando outros agentes e as classes de ações e eventos, e também as classes de *commitment*/*claim* que determinam uma possível interação com o agente.

- Diagrama de Seqüência de Reação: destacando instâncias de interações de processos numa perspectiva interna ao agente.

5.2 Modelo externo

Durante a análise, o modelo AOR externo define o agente, o *frame* da interação, a seqüência da interação e os diagramas de padrões de interação. Os modelos externos da AOR fornecem meios para uma análise do domínio da aplicação. Tipicamente, esses modelos têm como foco, o agente, ou o grupo dos agentes que desejamos desenvolver, o modelo de estados e comportamental (WAGNER, 2003).

Na visão do observador externo adotada nos modelos externos de AOR, o mundo (isto é, o domínio da aplicação) consiste em vários tipos de elementos modeláveis, como descritos na FIG.21. Na opinião de um observador externo, as ações também são eventos, e os compromissos também são reivindicações. Conseqüentemente, um modelo externo de AOR contém, além dos tipos do agente e dos objetos de interesse, as classes do evento da ação e as classes de commitment/claim que são necessárias para descrever a interação entre os agentes principais e os outros tipos de agentes.

A FIG.21 mostra os elementos de um modelo externo de AOR, de acordo com a notação AOR.

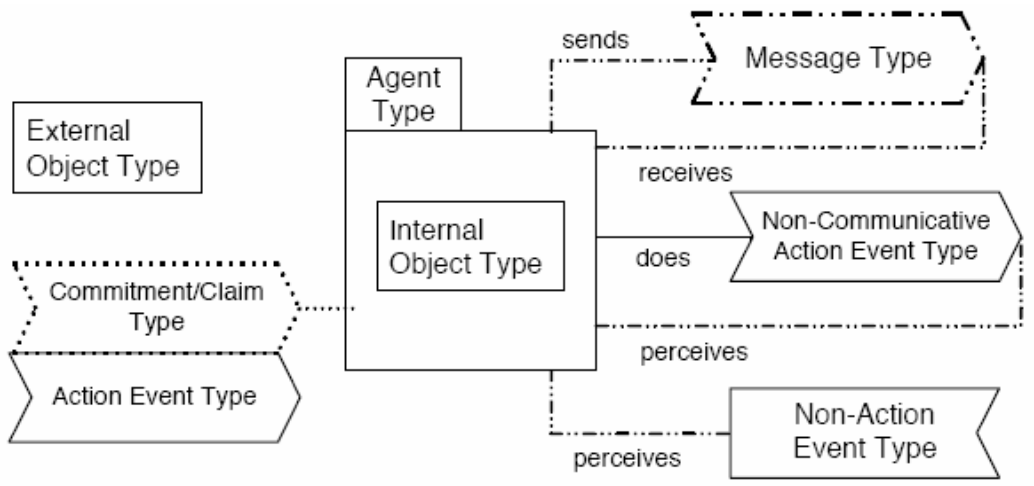

FIGURA 21 – Elementos do modelo externo da AOR
Fonte: WAGNER, 2003.

Os tipos do objeto pertencem a um ou diversos agentes (ou a tipos de agentes). Se um tipo do objeto pertencer exclusivamente a um agente, o retângulo correspondente estará dentro do agente. Se um tipo do objeto representar crenças que estão compartilhadas entre dois ou mais agentes (*AgentType*), o retângulo do objeto é conectado com os retângulos respectivos do agente por meio de um conector do *aggregation* da UML.

Na FIG.21 ainda há uma distinção entre um evento comunicativo (ou mensagem) e um evento não comunicativo da ação. A AOR também faz distinção entre eventos "de ação" e eventos "*non-action*". A figura mostra também que um "*commitment/claim*" está geralmente seguido pelo evento "da ação" que cumpre o compromisso (ou satisfaz essa reivindicação).

Um modelo externo de AOR não inclui nenhum *software* artificial. Representa apenas uma visão conceitual da análise do domínio do problema e pode também conter os elementos que são meramente descritivos e não executáveis por um programa de computador.

5.3 Modelagem do AVA-AGENTS

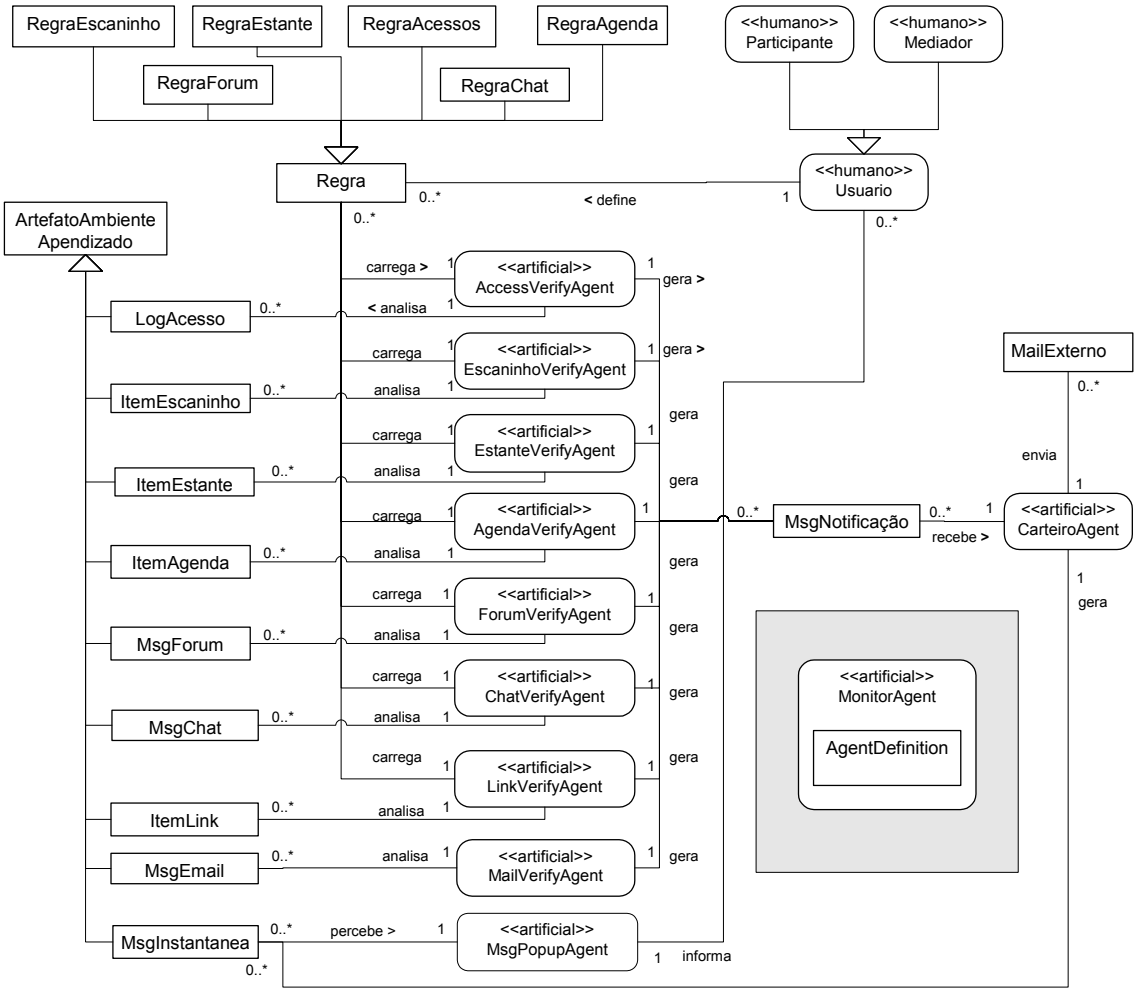

FIGURA 22 – Diagrama de Agentes

O diagrama de agentes representado pela FIG.22 que descreve as classes do agente do domínio, determina classes de objetos relevantes e os relacionamentos entre eles.

Nele, participantes e mediadores do AVAUFES são definidos como usuários (agentes humanos) da comunidade de aprendizagem.

Nessa comunidade artefatos do ambiente de aprendizado são gerados pela interação dos usuários com o ambiente e suas ferramentas, esses artefatos podem ser: mensagens do fórum, mensagens das conversas ocorridas nas salas de bate-papo, mensagens trocadas entre usuários do sistema (e-mails), arquivos postados

nas estantes pessoais e dos grupos, arquivos postados nos escaninhos dos usuários em cada grupo, informações cadastradas na agenda pessoal do usuário e do grupo, informações de acesso dos usuários no ambiente e suas ferramentas, links importantes dos usuários e grupos, e mensagens instantâneas trocadas entre os usuários no ambiente.

Para auxiliar os agentes humanos no controle e administração dos artefatos do ambiente de aprendizado, alguns agentes artificiais são propostos. O agente principal da solução é o *MonitorAgent*, este é responsável pela criação e monitoração de todos os agentes artificiais da solução. Através dele, novos agentes artificiais podem ser definidos e "plugados" no ambiente, bastando que uma nova definição de agente seja passada para sua criação e monitoração.
Para alguns destes agentes artificiais existe a possibilidade de definição de regras pelos agentes humanos para controle e monitoramento dessas "informações", tais como: AccessVerifyAgent, EscaninhoVerifyAgent, EstanteVerifyAgent, AgendaVerifyAgent, ForumVerifyAgent, ChatVerifyAgent, LinkVerifyAgent. Cada regra pode possuir determinadas características de acordo com o artefato analisado e as preferências do usuário em relação a essas informações. Além desses agentes que trabalham sobre determinadas regras, existem outros agentes para auxiliar no controle e fluxo dessas informações, sendo eles: MailVerifyAgent, MsgPopupAgent, CarteiroAgent, uma descrição detalhada de cada um destes agentes é feita a seguir.

Com a definição dessas regras os agentes artificiais passam a "trabalhar" pelos agentes humanos, deixando assim o trabalho "pesado" para os agentes artificiais, liberando o usuário (agente humano) para sua atividade principal, o aprendizado.

Os agentes artificiais propostos inicialmente na solução são:

- **MonitorAgent:** é o principal agente do sistema, ele é responsável pela criação, inicialização e gerenciamento dos agentes. As definições de cada agente e suas configurações estão definidas no objeto AgentDefinition. Para a inclusão e controle de novos agentes no ambiente, basta que esse "novo" agente seja definido e que suas informações sejam passadas para o MonitorAgent através do objeto AgentDefinition, permitindo assim que a solução seja dinâmica e que a

inclusão de novos agentes seja uma tarefa simples e fácil.

- **AccessVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de acesso ao ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora os acessos ao ambiente gerando informações referentes às estatísticas de acessos e a falta deles pelos usuários, que por sua vez são enviadas para o CarteiroAgent que é responsável pela entrega dessas informações aos seus respectivos destinatários.
- **EscaninhoVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de escaninhos do ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora os escaninhos do ambiente gerando informações a respeito de novas inclusões, atualizações e interesses por conteúdos específicos, que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.
- **EstanteVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de estantes do ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora as estantes pessoais e dos grupos do ambiente gerando informações a respeito de novas inclusões, atualizações e interesse por conteúdos específicos, que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.
- **AgendaVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de agendas do ambiente de aprendizado. De acordo com as regras definidas pelos agentes humanos (usuários), esse agente monitora as agendas pessoais e dos grupos do ambiente gerando informações a respeito de novas inclusões, atualizações e interesse por conteúdos específicos, proximidade de compromissos e expiração deles, que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.
- **ForumVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de fórum do ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora os fóruns de discussão dos grupos do ambiente gerando informações a respeito de novas inclusões, atualizações e interesse por conteúdos específicos, inclusão de conteúdos impróprios ao grupo, ausência de discussões e estatísticas sobre elas,

que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.

- **ChatVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de *chat* do ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora os participantes e suas mensagens postadas nas salas de bate-papo, gerando informações a respeito de novas discussões, discussões de conteúdos de interesse, além da possibilidade de ocorrência de discussão por definição de quórum mínimo de participantes, que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.
- **LinkVerifyAgent:** agente responsável pela análise e monitoramento dos artefatos de "*links* interessantes" pessoais e do grupo do ambiente de aprendizado. De acordo com as regas definidas pelos agentes humanos (usuários), esse agente monitora os "links interessantes" do ambiente gerando informações a respeito de novas inclusões, atualizações e interesse por conteúdos específicos, que por sua vez são enviadas para o CarteiroAgent que é responsável pela sua entrega.
- **MailVerifyAgent:** agente responsável pela verificação de existência de novos artefatos de mensagens geradas pelas ferramentas de correio e troca de mensagens do ambiente e sua atualização/envio para o *CarteiroAgent* que é responsável pela sua entrega.
- **MsgPopupAgent:** agente responsável pela verificação da existência de novos artefatos de mensagens instantâneas no ambiente e sua atualização, notificação e exibição para os usuários do ambiente (agentes humanos).
- **CarteiroAgent:** agente responsável pela recepção de *MsgNotificação* e distribuição de notificações para seus respectivos destinatários, podendo entregá-las via correio eletrônico (e-mail) ou disponibilizá-las via *MsgInstantanea* que por sua vez são detectadas pelo *MsgPopupAgent* e exibidas para os usuários (agentes humanos).

Esse grupo de agentes foi definido inicialmente com o propósito de auxiliar a monitoração e gerenciamento das informações das ferramentas contidas hoje no

ambiente AVAUFES, mais a solução proposta aceita a inclusão de novos agentes, bastando que eles sejam criados e passados para o *MonitorAgent,* que sem a necessidade de modificações no ambiente os conecta na solução, tornando-os parte do ambiente. Essa funcionalidade permite que a solução seja dinâmica e que futuramente novas ferramentas criadas para o ambiente (AVAUFES) possam também ter seus novos agentes virtuais conectados para controle, auxílio e monitoração.

5.4 Diagrama de interação de quadros

Existem dependências e relações entre os vários agentes em uma organização de agentes. Essas interações são fundamentais para que as funções do sistema sejam realizadas. Logo após identificar os agentes, deve-se, portanto, identificar as interações que ocorrem entre eles. Inicialmente, essas relações são apenas identificadas, de modo a facilitar a compreensão das hierarquias no sistema (WAGNER, 2003).

Em um modelo externo de AOR, há quatro tipos de relacionamentos designados entre agentes e eventos da ação: enviar e receber são os tipos de relacionamento que relacionam um agente com "*communicative action events*", enquanto fazer (does) e perceber (*perceives*) são os tipos do relacionamento que relacionam um agente com *"non-communicative action events*". Além desses dois tipos de relacionamentos, existem mais dois designados entre agentes e reivindicações de compromissos: *hasCommitment* e *hasClaim* (WAGNER, 2003).

Esses tipos de relacionamento são visualizados com tipos particulares de conectores como descritos na FIG.23. Observe que todos vêm com o relacionamento de multiplicidade um-para-muitos que, para simplificar o diagrama, não é mostrado explicitamente. Os nomes desses tipos de relacionamento serão geralmente omitidos em diagramas de AOR.

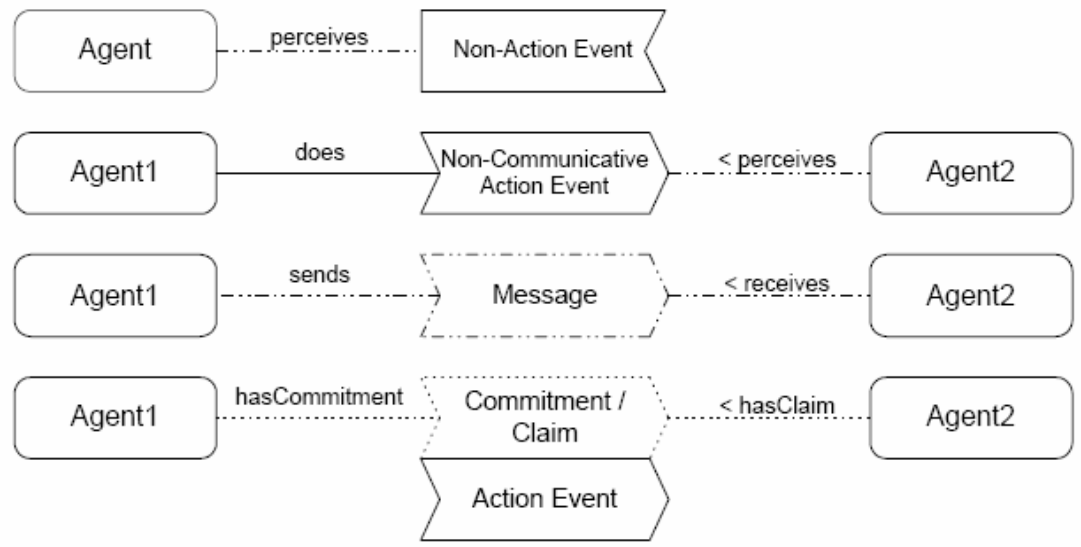

FIGURA 23: Os tipos de conectores sends, receives, does, perceives, hasCommitment and hasClaim
Fonte: Wagner, 2003.

O diagrama de interação de quadros no modelo externo da AOR descreve as possíveis interações entre 2 tipos de agentes.

Na FIG.24 é representada a interação entre o *MonitorAgent*, responsável pela criação e monitoração dos agentes, e um dos agentes artificiais do sistema monitorado por ele, neste caso o CarteiroAgent. Vale ressaltar que a comunicação representada aqui ocorre entre todos os agentes artificiais e o MonitorAgent.

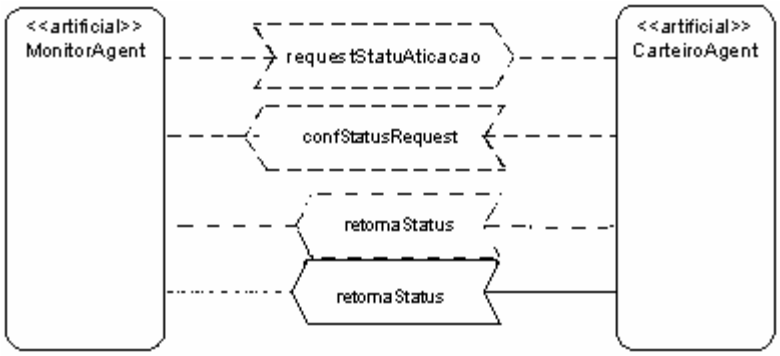

FIGURA 24 – Diagrama de Interação de Quadros – MonitorAgent e CarteiroAgent

No diagrama representado pela FIG.24, o *MonitorAgent* responsável pela criação, monitoração e gerenciamento dos agentes artificiais do ambiente fica constantemente em comunicação com os agentes verificando se eles estão ativos, e solicitando informações referentes ao nível atual de atividade e tempo de resposta desses agentes. A interação funciona da seguinte forma: O *MonitorAgent* solicita ao agente um status de ativação, que indica que o agente está funcionando e disponível para atender as solicitações do sistema, e a quantidade de requisições/tempo que esse agente está atendendo. Durante um tempo determinado o *MonitorAgent* aguarda pelo retorno de OK do agente monitorado. Se esse OK não chegar em um determinado tempo um novo agente é criado. No caso do agente estar ativo e respondendo as requisições, ele responderá a solicitação do *MonitorAgent*, que após receber o OK do agente aguarda pelas informações solicitadas. O *CarteiroAgent*, após receber a solicitação do *MonitorAgent*, gera suas informações estatísticas e as envia. Após receber essas informações, o *MonitorAgent* de acordo com suas crenças/premissas toma as devidas ações para manter o sistema ativo e com a melhor performance possível. O *MonitorAgent* também pode solicitar a destruição dos agentes, para que sejam economizados recursos do sistema.

Interações semelhantes a essa entre o *MonitorAgent* e o *CarteiroAgent* são executadas constantemente entre o MonitorAgent e todos os agentes artificiais criados e monitorados por ele. É através dessa interação que o *MonitorAgent* obtém informações sobre o funcionamento de cada agente artificial da organização de agentes controlada por ele e toma as devidas ações para manter a organização em perfeito funcionamento.

Além dessa interação de monitoração e controle entre o *MonitorAgent* e todos os agentes do ambiente, algumas outras interações são realizadas entre os agentes da solução, sendo elas: Interações entre “<agentes_artificiais> e *CarteiroAgent*” e “*MsgPopupAgen*t e Usuário”.

Uma interação entre um agente artificial da organização e o *CarteiroAgent*, ocorre sempre que um agente artificial fizer uma solicitação de envio de informações para outro agente. Essas informações podem ser enviadas por e-mails ou mensagens

instantâneas.

A FIG.25 apresenta o diagrama de interação entre o *AgendaVerifyAgent* e o *CarteiroAgent*, lembrando que uma interação semelhante é realizada entre todos os outros agentes da organização que enviam informações para destinatários através do *CarteiroAgent*.

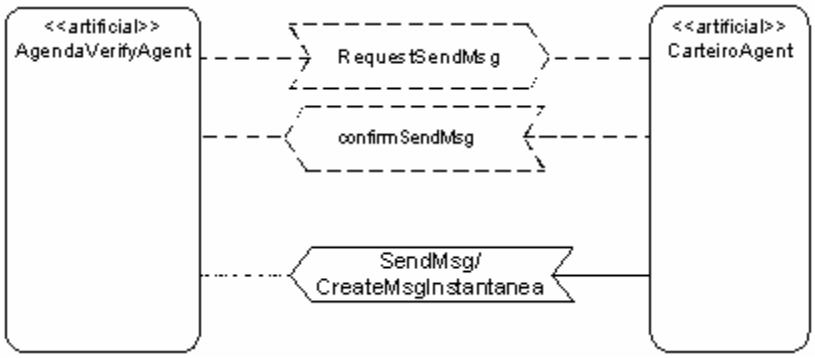

FIGURA 25 - Diagrama de Interação de Quadros – AgendaVerifyAgent e CarteiroAgent

No diagrama representado na FIG.25, o agente *AgendaVerifyAgent* envia uma mensagem para o *CarteiroAgent* solicitando o envio de uma determinada informação para determinados destinatários, o *CarteiroAgent* recebe a solicitação e a armazena em sua fila de solicitações, confirmando ao agente *AgendaVerifyAgent* a recepção da solicitação de envio da mensagem.

De acordo com a fila de procedência o *CarteiroAgent* vai enviando as mensagens para seus respectivos destinatários, podendo ser essas mensagens enviadas via e-mail ou armazenadas como MsgInstantaneas para serem exibidas pelo *MsgPopupAgent* aos agentes humanos.

Diferente da interação ente os agentes artificiais e o AgenteCarteiro, que possui características e comportamentos cadastrados em bases de regras pelos agentes humanos no ambiente, a interação do *MsgPopupAgent* e os Usuários é inicialmente estimulada pela detecção de novas mensagens instantâneas (artefatos) cadastradas no ambiente.

A FIG.26 mostra o diagrama de interação entre o *MsgPopupAgent* e os agentes humanos.

FIGURA 26 – Diagrama de Interação de Quadros - MsgPopupAgent e Usuário

No diagrama representado na FIG.26 o agente *MsgPopupAgent* percebe a existência de uma nova mensagem instantânea a ser enviada para o Usuário (agente humano). Após a percepção dessa mensagem o *MsgPopupAgent* a formata e a exibe para o usuário. Caso existam mais de uma mensagem para o mesmo usuário o agente exibe uma opção de paginação abaixo da mensagem exibida para que o usuário possa visualizá-las.

5.5 Diagrama de seqüência de interações

Um diagrama de seqüência da interação descreve em partes uma instância do processo de interação. Um processo da interação é uma seqüência de eventos com ação e de eventos sem ação, executados e percebidos pelos agentes, e seguindo um conjunto de regras (ou protocolo) que especifica o tipo do processo de interação. Os agentes podem interagir com seu ambiente, ou podem interagir uns com os outros.

Dessa forma foram especificados os seguintes diagramas de seqüência de interação no *AVAAgents*, buscando descrever os tipos de interações existentes entre os agentes do ambiente e entre os agentes e o ambiente (organização) da solução.

A FIG.27 apresenta o diagrama de seqüência de interação do agente *MsgPopupAgent* com o ambiente.

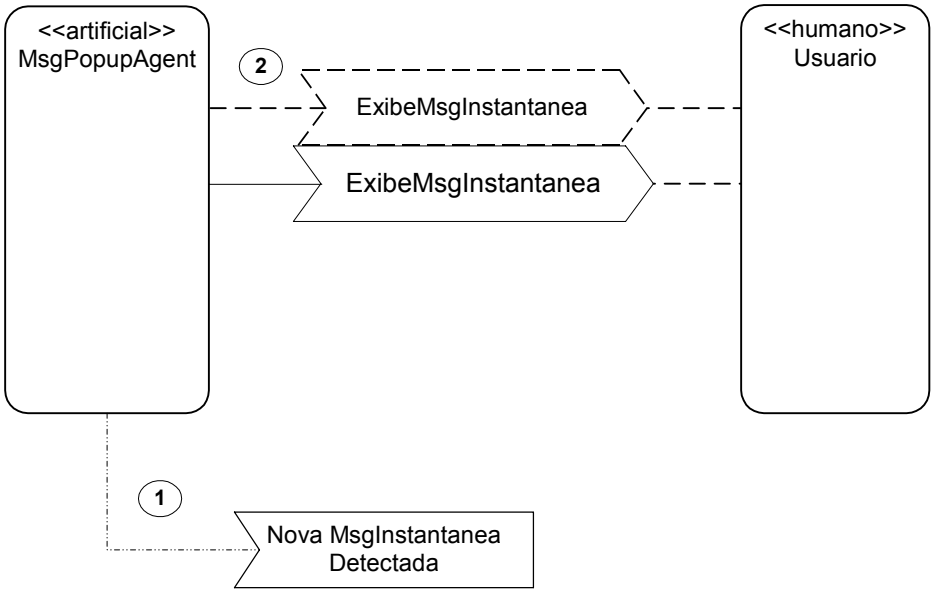

FIGURA 27 – Diagrama de Seqüência de Interação - Interação entre o MsgPopupAgent Ambiente e Usuário

No diagrama representado na FIG.27, o *MsgPopupAgent* está constantemente monitorando o ambiente, aguardando a chegada de novas mensagens instantâneas para um determinado Usuário (Agente humano) do ambiente. Ao perceber a chegada de uma nova mensagem instantânea, o agente *MsgPopupAgent* a recupera, formatando e exibindo-a ao usuário na qual a mensagem instantânea foi destinada.

Além dessa seqüência de interações entre agente virtual, ambiente e agente humano, ocorrem no ambiente, outras formas de interação entre agentes e ambientes, tais como: Interação entre o *MonitorAgent* e os Agentes Virtuais da Organização e a interação entre os Agentes (*AccessVerifyAgent, EscaninhoVerifyAgent, EstanteVerifyAgent, AgendaVerifyAgent, ForumVerifyAgent, ChatVerifyAgent, LinkVerifyAgent, MailVerify*) e o ambiente ou entre os Agentes e o agente *CarteiroAgent*.

A FIG.28 apresenta o diagrama de seqüência de interação referente às interações

entre o *MonitorAgent* e o agente *ForumVerifyAgent*.

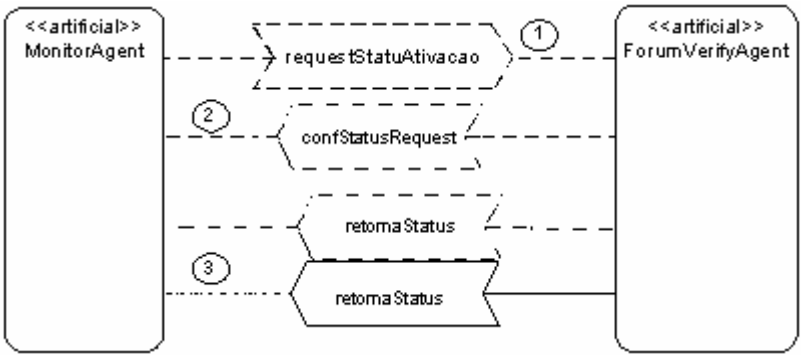

FIGURA 28 - Diagrama de Seqüência de Interação - Interação entre o MonitorAgent e o ForumVerifyAgent

O diagrama apresentado na FIG.28 mostra a seqüência de eventos de interação entre o *MonitorAgent* que cria, monitora e controla os agentes virtuais da organização e o *ForumVerifyAgent* que monitora as informações dos fóruns do ambiente. Esse diagrama mostra que inicialmente o MonitorAgent envia uma mensagem para o *ForumVerifyAgent* solicitando informações referentes ao status de sua ativação, ou seja, informações referentes as atividades executadas por esse agente. Ao receber essa mensagem o *ForumVerifyAgent* envia uma mensagem de confirmação ao MonitorAgent informando que recebeu a solicitação. Após enviar a resposta ao MonitorAgent, o *ForumVerifyAgent* recupera as informações referentes a seu status de processamento e as envia para o MonitorAgent que de posse delas pode tomar ações para melhorar a performance desse agente e da organização em si.

É importante ressaltar que a interação representada pelo diagrama da FIG.28 é realizada entre o *MonitorAgent* e todos os agentes virtuais da organização, e tem como objetivo fornecer informações ao *MonitorAgent* sobre a atual situação da organização de agentes, para que ele possa mantê-la em perfeito funcionamento.

Outras interações constantemente realizadas na organização são entre agentes virtuais responsáveis pela monitoração e controle das informações geradas pelo ambiente (como acessos, mensagens de fóruns, conversas em salas de chat,

arquivos das estantes, escaninho, compromissos agendados, *links* dos grupos dentre outras) e o agente *CarteiroAgent*.

Essa interação tem basicamente a finalidade de possibilitar o envio das informações solicitadas aos agentes virtuais pelos agentes humanos contidas nas bases de regras.

A FIG.29 apresenta o diagrama de seqüência de interação entre o agente *AccessVerifyAgent* e o *CarteiroAgent* da organização.

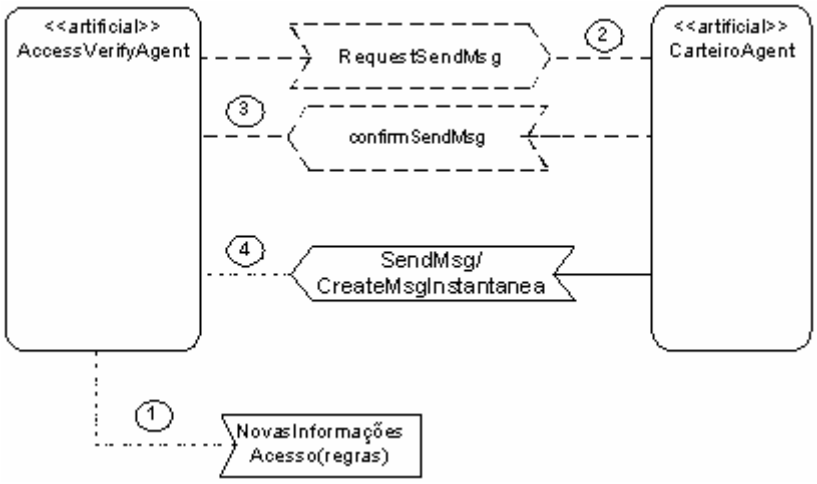

FIGURA 29 – Diagrama de Seqüência de Interação - Interação entre o AccessVerifyAgent e o CarteiroAgent

O diagrama representado pela FIG.29 mostra a seqüência de eventos de interação entre o *AccessVerifyAgent* que monitora as informações referentes aos acessos dos usuários no ambiente e o *CarteiroAgent*. O diagrama mostra que inicialmente o *AccessVerifyAgent* detecta a entrada de novas informações de acesso no ambiente que satisfazem as regras definidas pelos agentes humanos para ele. Após detectar as informações necessárias, ele as formata e solicita ao agente *CarteiroAgent* que as entregue para seus respectivos destinatários. O *CarteiroAgent* ao receber a solicitação de envio da mensagem, a armazena em sua fila de espera e envia uma mensagem de confirmação para o *AccessVerifyAgent*, informando sobre a recepção com sucesso da solicitação. Após enviar a mensagem de confirmação o *CarteiroAgent* continua processando as mensagens de sua fila de espera, enviando-

as para seus respectivos destinatários.

Mais uma vez é importante ressaltar que a interação representada pelo diagrama da FIG.29 é realizada entre os agentes *AccessVerifyAgent, EscaninhoVerifyAgent, EstanteVerifyAgent, AgendaVerifyAgent, ForumVerifyAgent, ChatVerifyAgent, LinkVerifyAgent* e o *CarteiroAgent,* mudando em cada um dos diagramas apenas o Artefato de Aprendizado monitorado e o tipo das regras correspondente a cada um desses agentes.

5.6 Considerações finais do capítulo

Neste capítulo apresentou-se a AORML, que é uma extensão da linguagem de modelagem unificada (UML) que relaciona agentes e objetos. Foram mostrados os motivos de não ser possível representar de forma completa a solução utilizando apenas a notação UML e como essa notação pode ser útil na definição da solução de agentes. Durante o capítulo foram representados os modelos de análise (externo) da solução de agentes do trabalho, parte inicial da definição do modelo de *design* (interno), que representa uma visão mais pessoal de cada agente da solução e como ele se comunica com os outros agentes e o ambiente no qual ele está contido.

6 PROJETO DOS AGENTES

De acordo com Wagner (2002), um sistema de agentes, com foco no domínio (externo) não tem objetivo, mas sim uma visão subjetiva do domínio. Em AORML é usado um modelo computacional de projeto, onde a perspectiva interna do sistema a ser construído é adotada, no contraste ao modelo externo (objetivo) que possui sua perspectiva em um modelo conceptual do domínio. No processo de internalização descrito por Wagner, o diagrama externo de AOR é transformado em um diagrama AOR interno, modelando o problema na visão de cada agente, assim como as interações (comunicação) dele dentro da organização e como ele enxerga a organização dentro do contexto do problema.

6.1 Modelo interno

O modelo Interno, como é conhecido o modelo de design, modela o problema na visão de cada agente, como ele se comunica com os outros agentes, como ele é notificado dos eventos no ambiente e como ele enxerga o “mundo” (WAGNER, 2002).

Em um modelo AOR interno, adota-se a visão interna do agente particular a ser modelado. Nessa visão de primeira pessoa, “o mundo” (isto é, o domínio de interesse) consiste em vários tipos de:

- outros agentes;
- ações;
- compromissos com outros agentes para executar determinadas ações;
- eventos, muitos deles criados por ações de outros agentes;
- reivindicações vindas de outros agentes;
- objetos diversos;
- vários relacionamentos, assim com *SentTo*, *isPerceivedBy*, etc;
- associações diversas.

Esses tipos de meta-entidade do modelo interno são mostrados na figura 30.

Um modelo interno de AOR descreve “o mundo” no modelo mental do agente. Se o foco do agente for uma organização, o modelo interno representa sua visão do “mundo”, podendo assim ser usado para projetar seu sistema de informação. Desta forma, o modelo AOR sugere o seguinte trajeto para o desenvolvimento de sistemas de informação organizacionais:

1. Na análise do domínio, desenvolver um modelo externo de AOR de uma organização (ou de um grupo das organizações) e de seu ambiente da perspectiva de um observador externo do cenário.
2. Transformar o modelo externo de AOR em um modelo interno no foco do agente do sistema de informação que será desenvolvido (tipicamente uma organização ou uma unidade organizacional). Se houver diversos focos de agentes, para cada um deles um sistema de informação deve ser modelado.
3. Transformar os modelos internos de AOR obtidos na etapa anterior em modelos de projeto de base de dados (esquemas lógicos da base de dados), tais como sistemas de gerência da base de dados objeto-relacional (SQL-99) ou em correspondentes estruturas de dados lógicas de uma linguagem foco, como Java.
4. Refinar os modelos do projeto nos modelos de implementação (estruturas lógicas de dados) fazendo uma análise em relação às limitações e desempenho.
5. Gerar o código da linguagem (tecnologia) alvo.

Um modelo interno pode ser compreendido de um ou mais dos seguintes diagramas:

- Diagrama de Quadros da Interação: destacando outros agentes e as classes de ações e eventos, e também as classes de *commitment*/*claim* que determinam uma possível interação com ele.
- Diagrama de Seqüência de Interação: destacando instâncias de interações de processos numa perspectiva interna ao agente.
- Diagramas de Padrão de Reação: que focalizam os padrões de reação dos agentes sobre considerações representadas por regras de reação.

Os diagramas de quadros da interação e os diagramas de padrão de reação podem ser fundidos em um único diagrama interno de AOR (IAORD) como demonstrado na Figura 30. Diagramas de seqüência de interação não são incluídos no IAORD, por

não estarem no mesmo nível. Todos os diagramas internos de AOR são desenhados dentro de um frame, preferivelmente com cantos arredondados e com o nome do agente que está sendo modelado no canto esquerdo superior (Wagner, 2002).

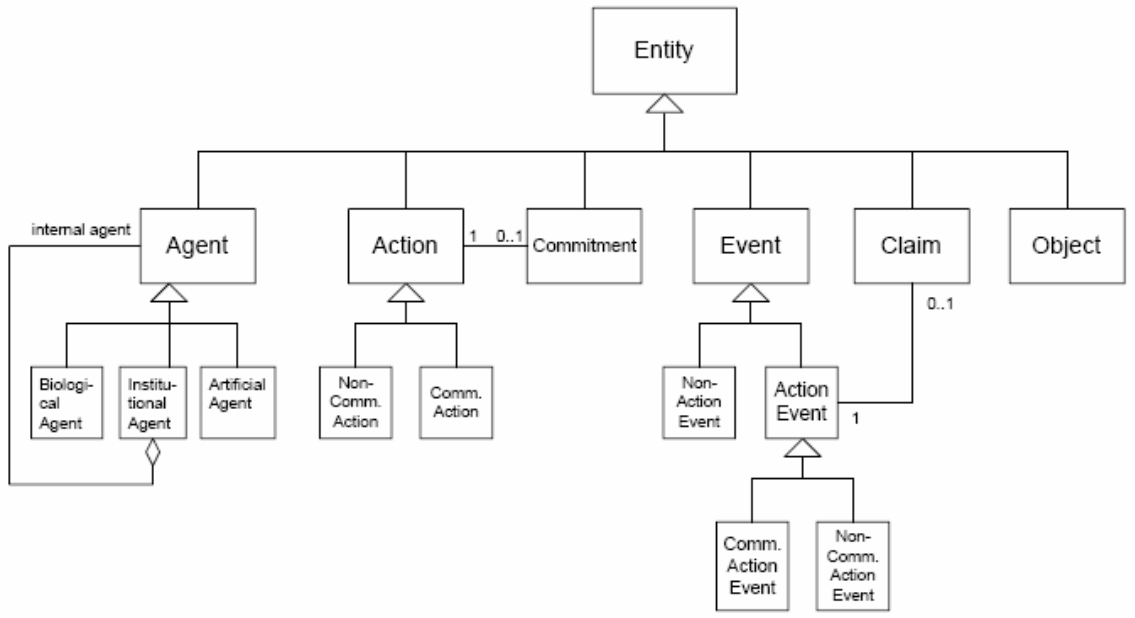

FIGURA 30 - Os tipos de meta-entidades do modelo interno
Fonte: WAGNER, 2002.

6.2 Diagrama de quadros de interação

Em um modelo interno da AOR, o comportamento reativo de um agente (tipo) em relação a outro agente (tipo) A é descrito por meio de seis tipos de entidades:

1. eventos de comunicação, ou mensagens recebidas, criadas pelas ações de comunicação das instâncias de A;
2. ações de comunicação, ou mensagens enviadas, dirigidas às instâncias de A;
3. reivindicações vindas de instâncias de A;
4. eventos não comunicativos criados pelas ações de instâncias de A;
5. compromissos enviados para instâncias de A;
6. ações não comunicativas que podem ser executadas a fim cumprir compromissos correspondentes para instâncias de A, ou na resposta aos eventos dentro do frame da interação;

O diagrama representado na figura 31 mostra um diagrama de quadro da interação entre os agentes MonitorAgent e CarteiroAgent.

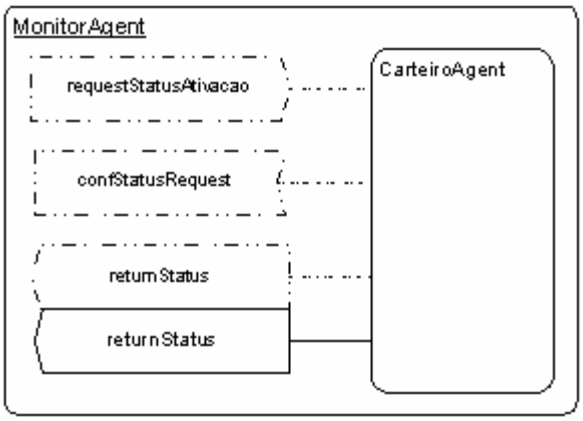

FIGURA 31 - O frame da interação entre MonitorAgent e o CarteiroAgent

A Figura 31 é a representação do diagrama de quadro de interação do MonitorAgent. Seu conteúdo pode ser descrito da seguinte forma: O agente MonitorAgent envia mensagens de solicitação de status de ativação; recebe mensagens de confirmação de solicitação de status; pode fazer reivindicações de returnStatus; e recebe eventos de returnStatus criados pelo CarteiroAgent.

Essa interação é realizada entre todos os agentes criados pelo MonitorAgent, e é repetida de tempos em tempos, tornando assim o processo cíclico, mantendo as informações referentes ao funcionamento dos agentes sempre atualizadas na estrutura de informações do MonitorAgent, que de posse delas pode decidir a melhor forma de otimizar a performance do ambiente.

Outra interação importante de ser demonstrada na solução é a interação entre os agentes de gerenciamento de artefatos de aprendizado do AvaUfes *(AccessVerifyAgent, EscaninhoVerifyAgent, EstanteVerifyAgent, AgendaVerifyAgent, ForumVerifyAgent, ChatVerifyAgent, LinkVerifyAgent, MailVerifyAgent)* e o CarteiroAgent.

No diagrama representado na figura 32 é ilustrada a interação entre o agente AgendaVerifyAgent e o CarteiroAgent, no processo de solicitação e envio de

mensagens com informações definidas para este agente.

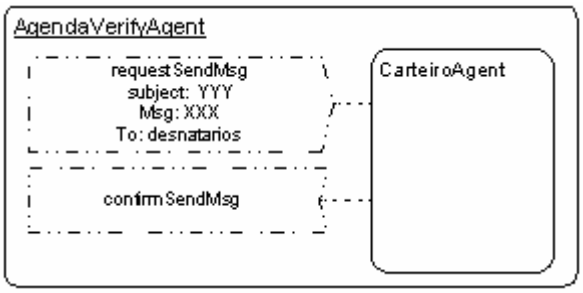

FIGURA 32 - O frame da interação entre AgendaVerifyAgent e o CarteiroAgent

A Figura 32 representa o diagrama de quadro de interação para o *AgendaVerifyAgent*. Seu conteúdo pode ser descrito da seguinte forma: O agente *AgendaVerifyAgent*, após gerar a mensagem com as informações solicitadas em sua base de regras, envia mensagens de solicitação de envio delas para o Carteiroagent; recebe mensagem de confirmação da remessa de cada mensagem enviada ao destinatário.

Essa interação também é realizada entre todos os agentes que utilizam o serviço do CarteiroAgent para o envio de mensagens e alertas.

6.3 Internalização

Um diagrama externo de AOR pode ser transformado em um diagrama interno de AOR para o foco de um dos agentes (ou de tipos do agente) da seguinte maneira:

1. omitindo o foco do agente cujo o perspectiva é modelada (o agente da primeira-pessoa);
2. alterando todos os eventos de ação direcionados para o agente “primeira-pessoa” em retângulos de eventos;
3. alterando todos os retângulos do evento de ação direcionados para o parceiro de interação do agente primeira-pessoa em retângulos de ação;
4. alterando todos os retângulos compromisso/reivindicação direcionados para o agente “primeira-pessoa” em retângulos de reivindicação;

5. alterando todos os retângulos compromisso/reivindicação direcionados para o parceiro de interação do agente primeira-pessoa em retângulos de compromisso;
6. traçando as linhas dos retângulos de todos os agentes internos mais importantes;
7. realizando um merge (fusão) das representações internas e externas dos tipos de objetos;

Toda essa transformação é chamada internalização.

Dessa forma os diagramas de frame de interação mostrados na Figura 24 e 25, foram transformados nos diagrama de frame de interação mostrados nas Figuras 31 e 32.

No diagrama AOR interno (IAORD), desde que o escopo seja um agente único, não se pode mais ver uma completa interação de padrões envolvendo dois ou mais agentes. O comportamento do agente sob consideração é modelado pela identificação dos padrões de reação e expressado em forma de regras de reação, como no IAORD representado na Figura 33.

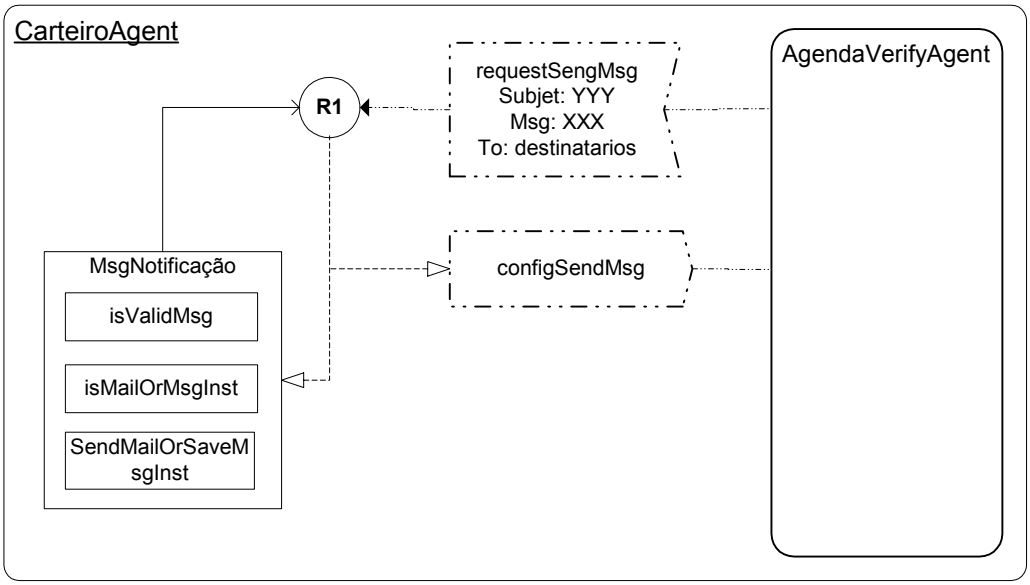

FIGURA 33 - Diagrama padrão de reação para o CarteiroAgent

O diagrama representado pela Figura 33 descreve os passos do processo de reação do *CarteiroAgent* na recepção de uma solicitação de envio de uma mensagem seja

ela e-mail ou mensagem instantânea. Esse comportamento é sempre o mesmo para o *CarteiroAgent*, independente do agente no qual a solicitação de envio da mensagem é disparada.

Outro diagrama padrão de reação importante a ser apresentado neste trabalho é o padrão de reação de todos os agentes criados e controlados pelo *MonitorAgent*, na figura 34 representado pelo *CarteiroAgent*. Esse padrão de reação ocorre toda vez que o *MonitorAgent* solicita informações referentes ao status de atividade dos agentes criados por ele, para que possa realizar o controle deles.

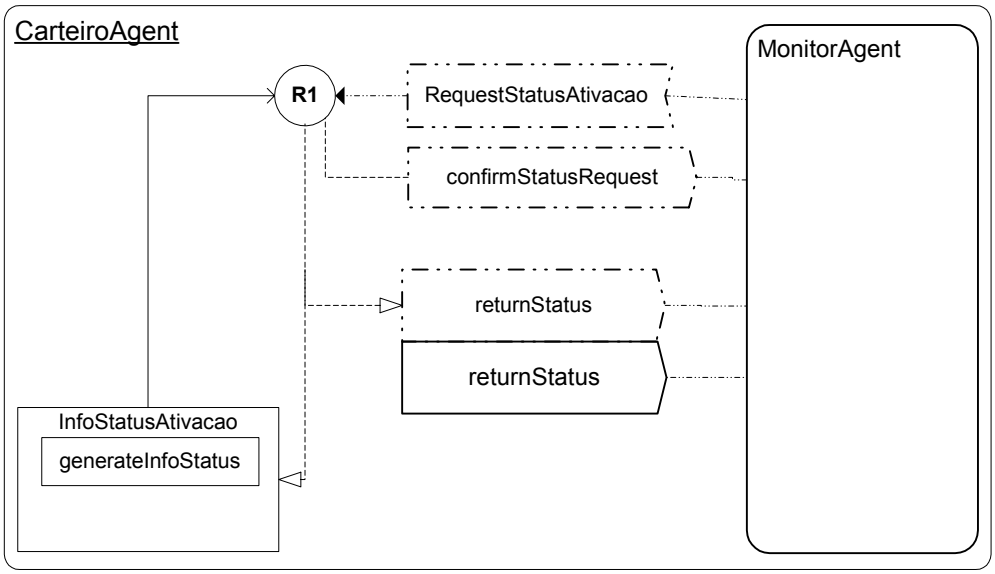

FIGURA 34 - Diagrama padrão de reação para o CarteiroAgent em relação ao MonitorAgent

Como demonstrado na Figura 34 e já destacado anteriormente, esse padrão de reação é adotado por todos os agentes criados pelo *MonitorAgent*, e é executado toda vez que o *MonitorAgent* solicita informações referentes ao *status* de trabalho dos agentes do ambiente.

6.4 Considerações finais do Capítulo

Com a definição dos modelos internos da organização de agentes chega-se a um ponto onde a modelagem do problema na visão de cada agente foi definida, assim como sua comunicação dentro da organização e como ele a enxerga dentro do

contexto do problema.

De posse dessas informações, a solução de agentes pode começar a ser desenvolvida usando nessa próxima fase as tecnologias definidas durante o decorrer do trabalho.

No próximo capítulo são apresentados os resultados obtidos na implementação do *framework* proposto e idéias de trabalhos futuros que poderão ser gerados a partir da solução inicialmente projetada e implementada.

7 INTEGRAÇÃO ENTRE O AVAUFES E A PLATAFORMA DE AGENTES

Como escrito nos capítulos anteriores, para que a plataforma de agentes fosse testada de forma concreta foi definido que ela seria acoplada ao AVAUFES, ambiente AVA desenvolvido no escopo desta dissertação.

Este capítulo visa mostrar algumas funcionalidades principais do ambiente, suas funções e como a plataforma de agentes foi acoplada a ela. Dessa forma, torna-se possível validar o sistema multiagente, em seu propósito de facilitar e auxiliar os usuários desse ambiente em suas atividades diárias.

7.1 Configurações do ambiente

O ambiente AVAUfes está hoje hospedado na própria UFES em um computador PENTIUM 2 com 1 Gbyte de memória ram, HD de 80 Gbytes e processador 2.0 gHz. O sistema operacional utilizado nesse computador é um Linux Suse 3, o servidor de aplicações Java utilizado é um JBOSS 4.0.2 com JDK 1.4.2_08 e o banco de dados MySql 3.23.58.

7.2 Configuração da plataforma de agentes

A plataforma de agentes foi desenvolvida em Java utilizando JADE, com JDK 1.4.2 para manter a compatibilidade com a atual versão do JDK utilizado no ambiente AVAUfes. A plataforma de agentes foi acoplada ao AVAUfes por meio da inclusão e um *servlet*[1] de inicialização no ambiente AVAUfes, sendo este iniciado toda vez que

[1] **Servlet** é um componente que disponibiliza ao programador da linguagem Java uma interface para o servidor web (ou servidor de aplicação), através de uma API. As aplicações baseadas no servlet geram conteúdo dinâmico (normalmente HTML) e interagem com os clientes, utilizando o modelo *request/response*. Os *servlets* normalmente utilizam o protocolo HTTP, apesar de não serem restritos a ele.

o ambiente é iniciado, e permanece ativo durante toda a sessão de funcionamento do ambiente.

7.3 Interfaces de Acesso e Utilização do AvaUfes

O ambiente AVAUfes está hoje disponibilizado para seus usuários através da Internet sob o domínio http://av.inf.ufes.br, como demonstrado na Figura 35.

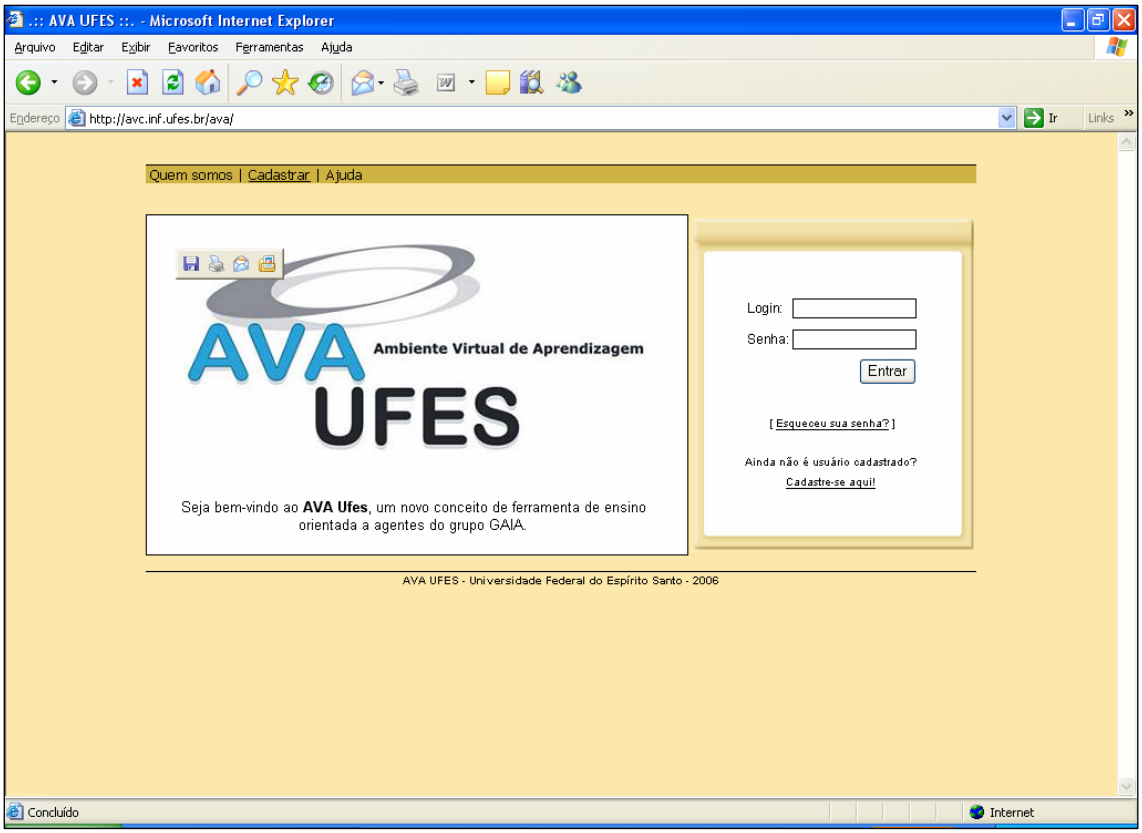

FIGURA 35 – Interface de entrada do ambiente AVAUfes

Para realizar acesso ao sistema, basta que o usuário forneça seu *login* e senha nos respectivos campos, caso o usuário não seja cadastrado no ambiente basta que este clique na opção “Cadastre-se aqui!” e informe seus dados como demonstrado na Figura 36.

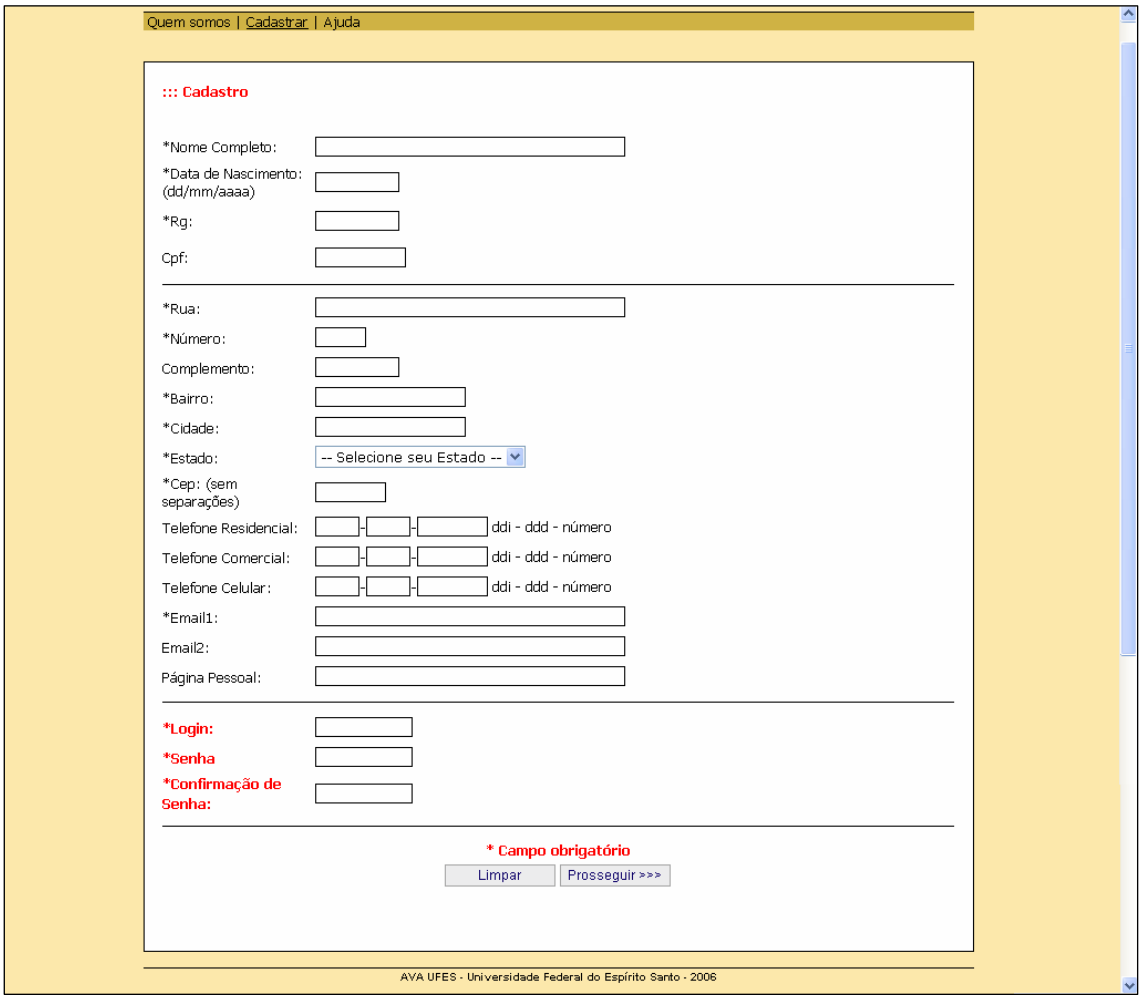

FIGURA 36 – Tela de cadastro do ambiente AvaUfes

Após realizar o cadastro no ambiente o usuário terá acesso ao ambiente e sua área de trabalho e às ferramentas (funções) pessoais. Essas funções permitem a consulta e a inscrição em grupos de estudo e a cooperação como demonstrado na Figura 37.

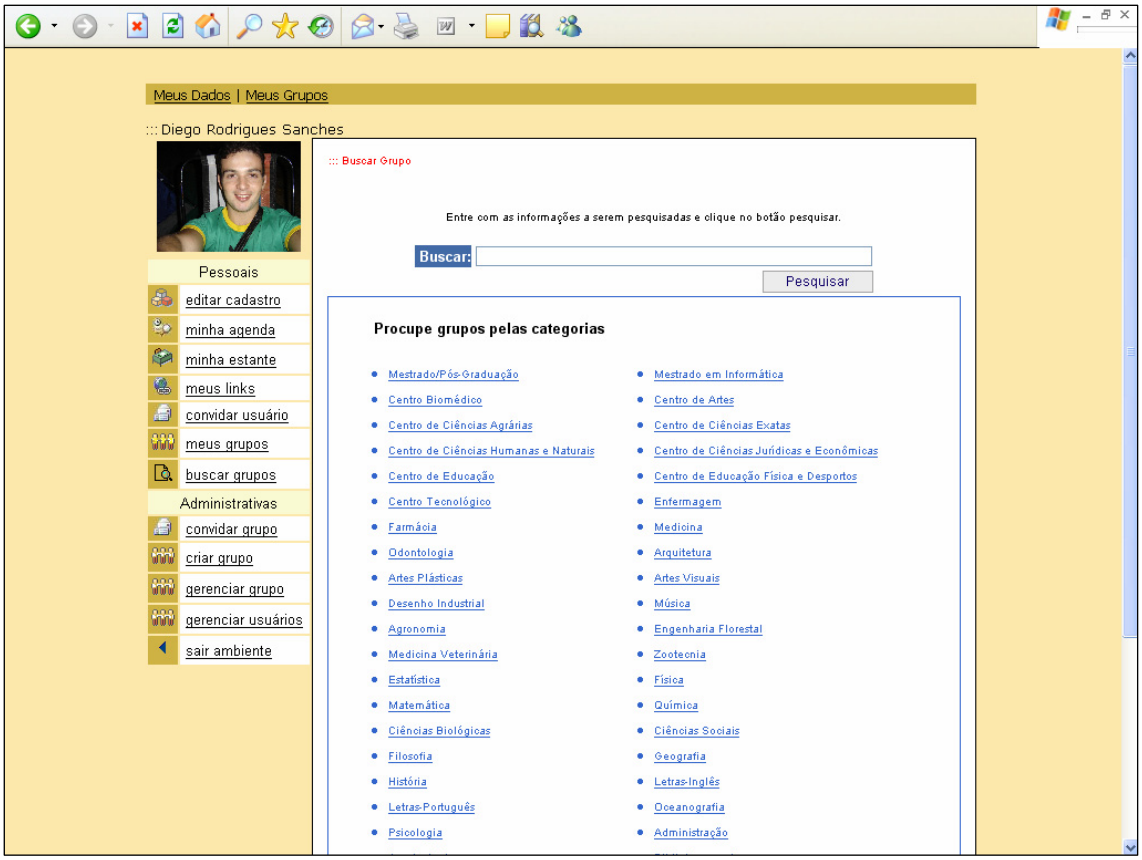

FIGURA 37 – Tela de consulta dos grupos disponíveis no ambiente

Como demonstrado na Figura 37 o ambiente permite que o participante pesquise e visualize informações referentes a um grupo de seu interesse antes de solicitar sua inscrição nesse grupo.

Após se inscrever e ter seu cadastro liberado pelo mediador do grupo, o participante passa a ter esse grupo liberado para acesso em sua opção de menu “meus grupos”. A partir daí o participante passa a ter acesso a esse grupo de estudo como demonstrado na Figura 38.

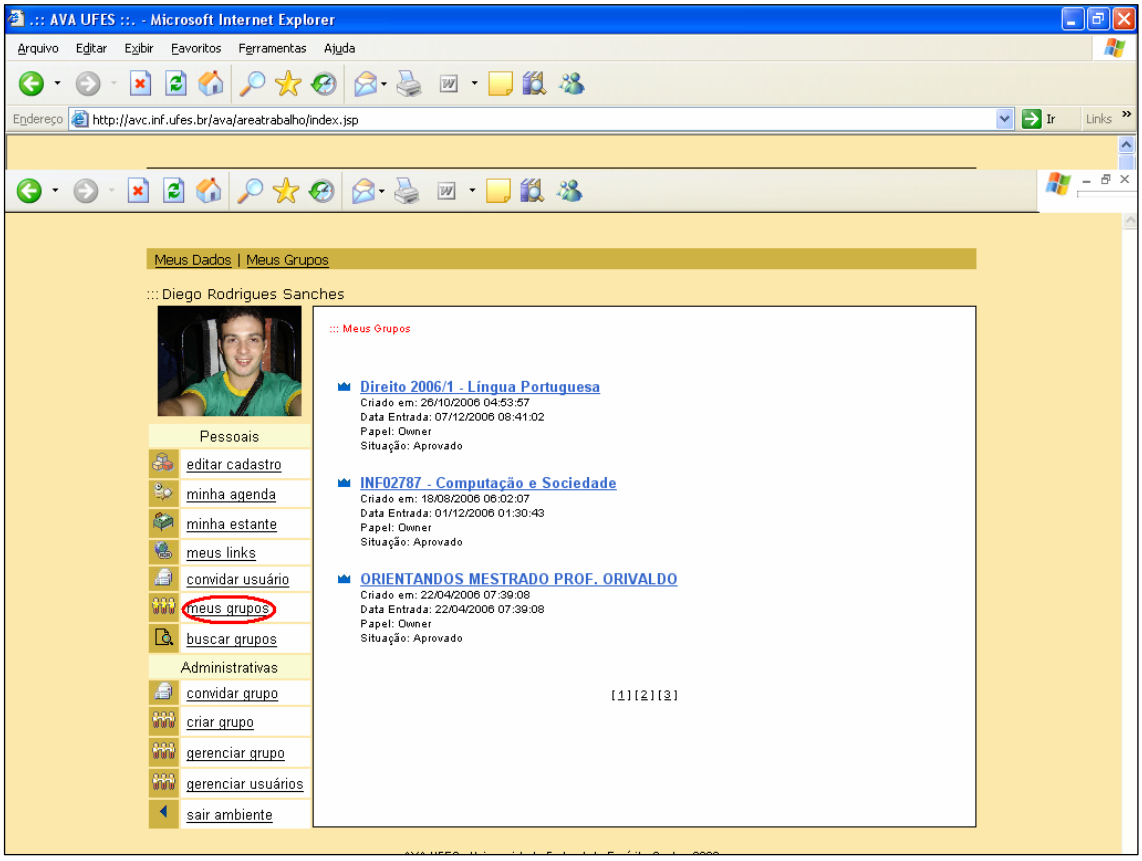

FIGURA 38 – Interface de seleção de grupos no qual o usuário está cadastrado

Ao selecionar a opção meus grupos, o usuário visualiza em sua interface de trabalho todos os grupos nos quais ele está inscrito e ativo. Ao selecionar um desses grupos, o usuário acessa esse grupo e pode usar os recursos desse grupo, podendo também interagir com todos os outros participantes do grupo usando as ferramentas disponibilizadas pelo sistema.

A Figura 39 mostra o menu de acesso às ferramentas do grupo, após o usuário ter selecionado um de seus grupos. É nesse momento que a interface de cadastro e configuração de agentes foi incluída. Como demonstrado na Figura 39 o ambiente AVAUFES disponibiliza para seus usuários um menu de ferramentas/recursos. A quantidade de recursos depende do perfil do usuário em determinado grupo. Esses recursos permitem a cooperação e troca de conhecimento entre os usuários de um grupo (aprendizes e mediadores), além de permitir que o moderador do grupo gerencie os participantes e o próprio grupo.

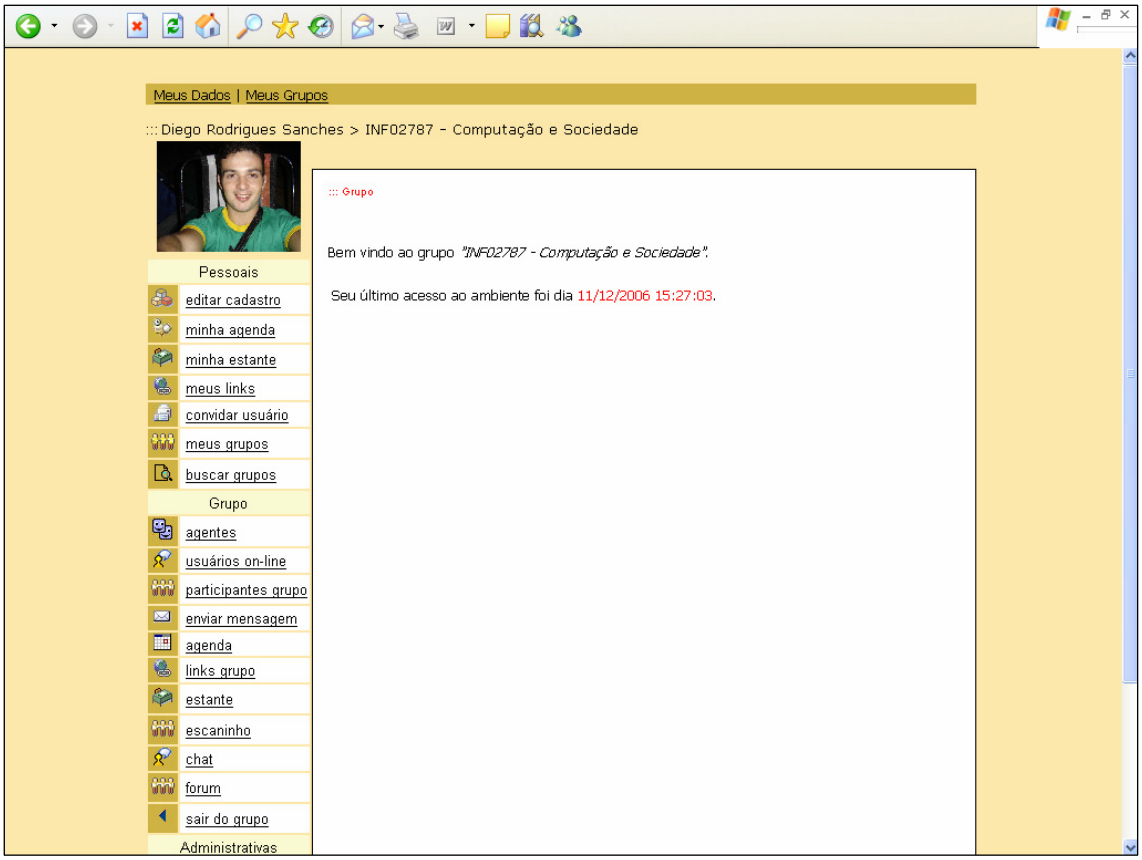

FIGURA 39 – Interface de entrada de grupo do ambiente AVAUfes

Para que os usuário do AVAUfes possa configurar as regras de seus agentes e personalizar as ferramentas de seu ambiente e grupo, foi incluída uma nova opção de menu no ambiente chamada “agentes”. Essa opção como mostra na Figura 39 é a primeira opção de menu do grupo e permite que os usuários do ambiente (aprendizes e mediadores) personalizem as configurações de seus agentes pessoais, permitindo que os agentes trabalhem a favor de cada usuário.

A Figura 40 apresenta a interface principal de acesso às funcionalidades de configuração dos agentes de cada ferramenta do ambiente.

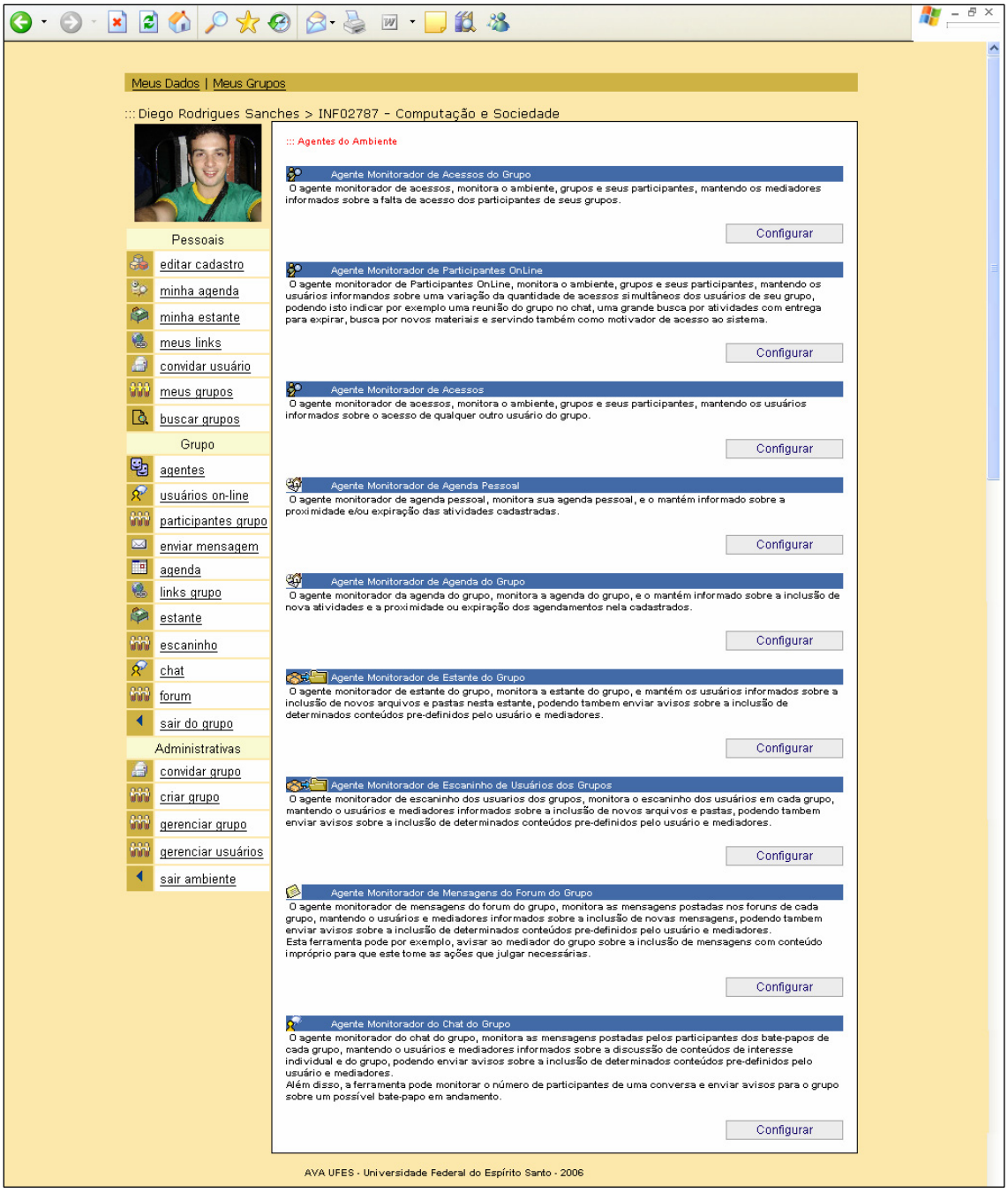

FIGURA 40 – Interface de acesso à configuração das regras de cada agente pessoal disponível pelo ambiente

Como demonstrado na Figura 40 cada agente pessoal do usuário possui uma interface de acesso para cadastro e configuração de suas regras, a tabela abaixo descreve cada agente pessoal disponível no ambiente para personalização de regras e suas funções.

A Quadro 3 descreve todos os agentes, suas ferramentas específicas e suas principais funções.

QUADRO 3
Agentes, ferramentas específicas e suas principais funções
(Continua)

Agente	Ferramenta	Principais Funções
Agente Monitorador dos Acessos do Grupo	Controle de Acessos	O agente monitorador dos acessos monitora o ambiente, os grupos e seus participantes, mantendo os mediadores informados sobre a falta de acesso dos participantes de seus grupos.
Agente Monitorador dos Participantes *OnLine*	Controle de Acessos	O agente monitorador dos Participantes *OnLine*, monitora o ambiente, grupos e seus participantes, mantendo os usuários informados sobre uma variação da quantidade de acessos simultâneos dos usuários de seu grupo, uma vez que isso pode indicar, por exemplo, uma reunião do grupo no *chat*, uma grande busca por atividades com entrega para expirar, busca por novos materiais e serve também como motivador de acesso ao sistema.
Agente Monitorador de Acessos	Controle Acessos	O agente monitorador de acessos, monitora o ambiente, os grupos e seus participantes, mantendo os usuários informados sobre o acesso de qualquer outro usuário do grupo.
Agente Monitorador da Agenda Pessoal	Agenda Pessoal	O agente monitorador da agenda pessoal, monitora sua agenda pessoal, e o mantém informado sobre a proximidade e/ou expiração das atividades cadastradas.
Agente Monitorador de Agenda do Grupo	Agenda Grupo	O agente monitorador da agenda do grupo, monitora a agenda do grupo e o mantém informado sobre a inclusão de novas atividades e a proximidade ou expiração dos agendamentos nela cadastrados.

QUADRO 3
Agentes, ferramentas específicas e suas principais funções
(Conclusão)

Agente	Ferramenta	Principais Funções
Agente Monitorador da Estante do Grupo	Estante Grupo	O agente monitorador da estante do grupo, monitora a estante do grupo e mantém os usuários informados sobre a inclusão de novos arquivos e pastas nessa estante, podendo também enviar avisos sobre a inclusão de determinados conteúdos pré-definidos pelo usuário e mediadores.
Agente Monitorador do Escaninho de Usuários dos Grupos	Escaninho	O agente monitorador do escaninho dos usuários dos grupos, monitora o escaninho dos usuários em cada grupo mantendo os usuários e os mediadores informados sobre a inclusão de novos arquivos e pastas, podendo também enviar avisos sobre a inclusão de determinados conteúdos pré-definidos pelo usuário e mediadores.
Agente Monitorador de Mensagens do Fórum do Grupo	Fórum Grupo	O agente monitorador de mensagens do fórum do grupo monitora as mensagens postadas nos fóruns de cada grupo, mantendo os usuários e os mediadores informados sobre a inclusão de novas mensagens. Pode também enviar avisos sobre a inclusão de determinados conteúdos pré-definidos pelo usuário e mediadores. Um agente dessa classe pode, por exemplo, avisar ao mediador do grupo sobre a inclusão de mensagens com conteúdo impróprio para que sejam tomadas as providências necessárias.
Agente Monitorador do *Chat* do Grupo	*Chat*	O agente monitorador do chat do grupo, monitora as mensagens postadas pelos participantes dos bate-papos de cada grupo, mantendo os usuários e os mediadores informados sobre a discussão de conteúdos de interesse individual e do grupo, podendo enviar avisos sobre a inclusão de determinados conteúdos pré-definidos pelo usuário e mediadores. Além disso, a ferramenta pode monitorar o número de participantes de uma conversa e enviar avisos para o grupo sobre um possível bate-papo em andamento.

Como demonstrado na Quadro 1, cada agente do sistema tem uma funcionalidade alvo, ou seja, atua sobre determinado recurso do ambiente (por exemplo, fórum, *chat*, estante, etc) ou sobre o próprio ambiente (por exemplo, para monitorar acessos ao ambiente) a fim de auxiliar em sua utilização, monitoramento e controle.

7.4 Agentes criados para o ambiente AVAUFES

Como escrito nos capítulos anteriores, alguns agentes foram definidos como proposta deste trabalho e disponibilizados no AVAUfes para auxiliar os usuários desse ambiente e com isso torná-lo mais dinâmico e motivador.

Para cada agente disponibilizado no ambiente foi criada uma interface para configuração e personalização desses agentes por seus usuários. Essas interfaces de configuração de cada agente são descritas nas seções seguintes.

7.4.1 Agente monitorador de acessos do grupo

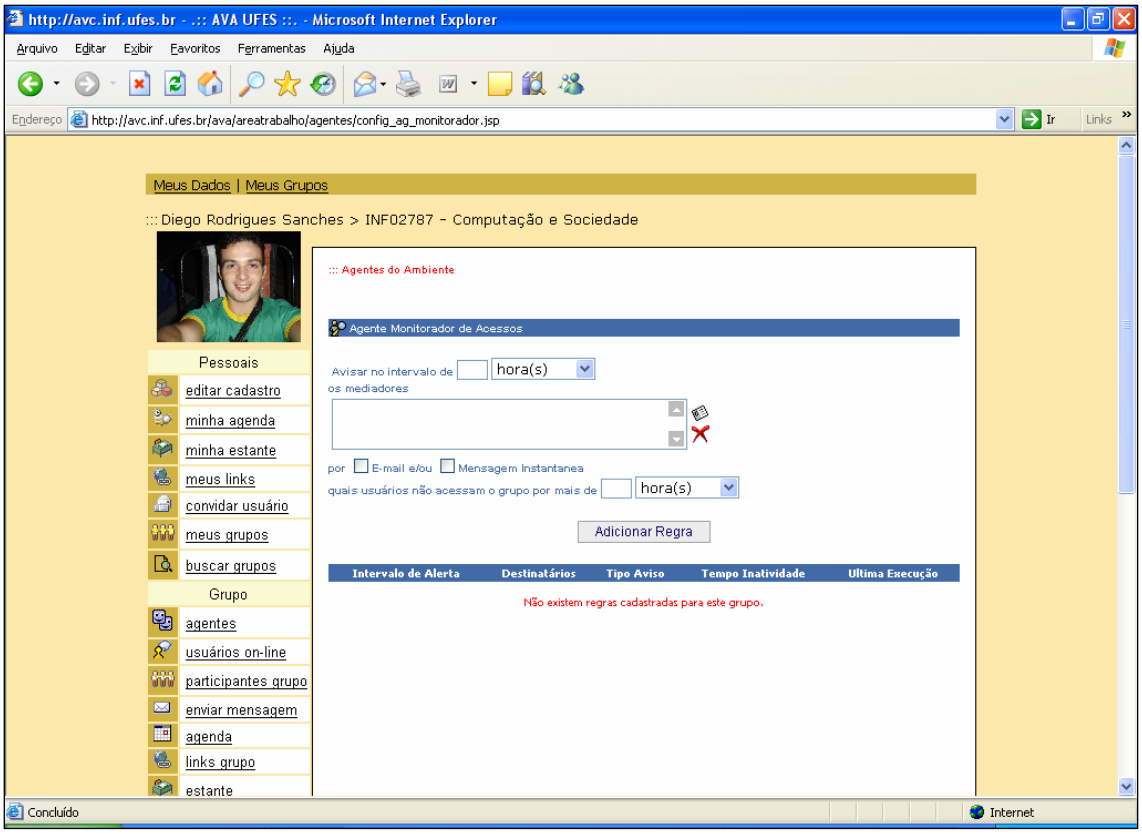

FIGURA 41 – Interface de Configuração do Agente Monitorador de Acessos do Grupo

Como apresentada na Figura 41, a interface de configuração do agente monitorador de acessos do grupo permite ao mediador criar regras referentes ao acesso dos outros participantes do grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o mediador com as informações sobre a falta de acesso dos usuários de seus grupos.

7.4.2 Agente monitorador de participantes *OnLine*

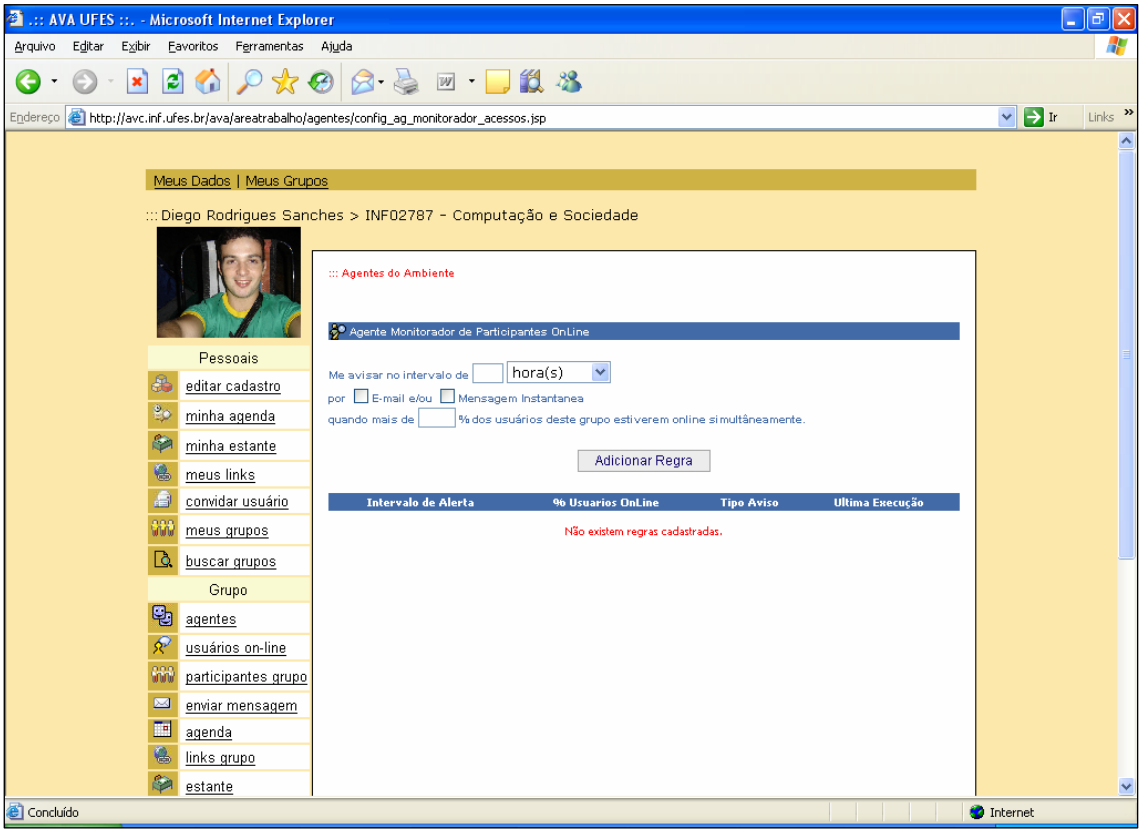

FIGURA 42 – Interface de Configuração do Agente Monitorador de Participantes OnLine

Como apresentada na Figura 42, a interface de configuração do agente monitorador de participantes *online* do grupo permite ao usuário (aprendiz ou mediador) criar regras referentes ao acesso dos outros participantes do grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário quando for atingido um quórum mínimo de usuários *online* simultaneamente no grupo.

7.4.3 Agente Monitorador de Acessos

Como apresentada na Figura 43, a interface de configuração do agente monitorador de acessos permite ao usuário (aprendiz ou mediador) criar regras referentes ao acesso dos outros participantes do grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário quando um determinado usuário de seu grupo estiver *online* no ambiente.

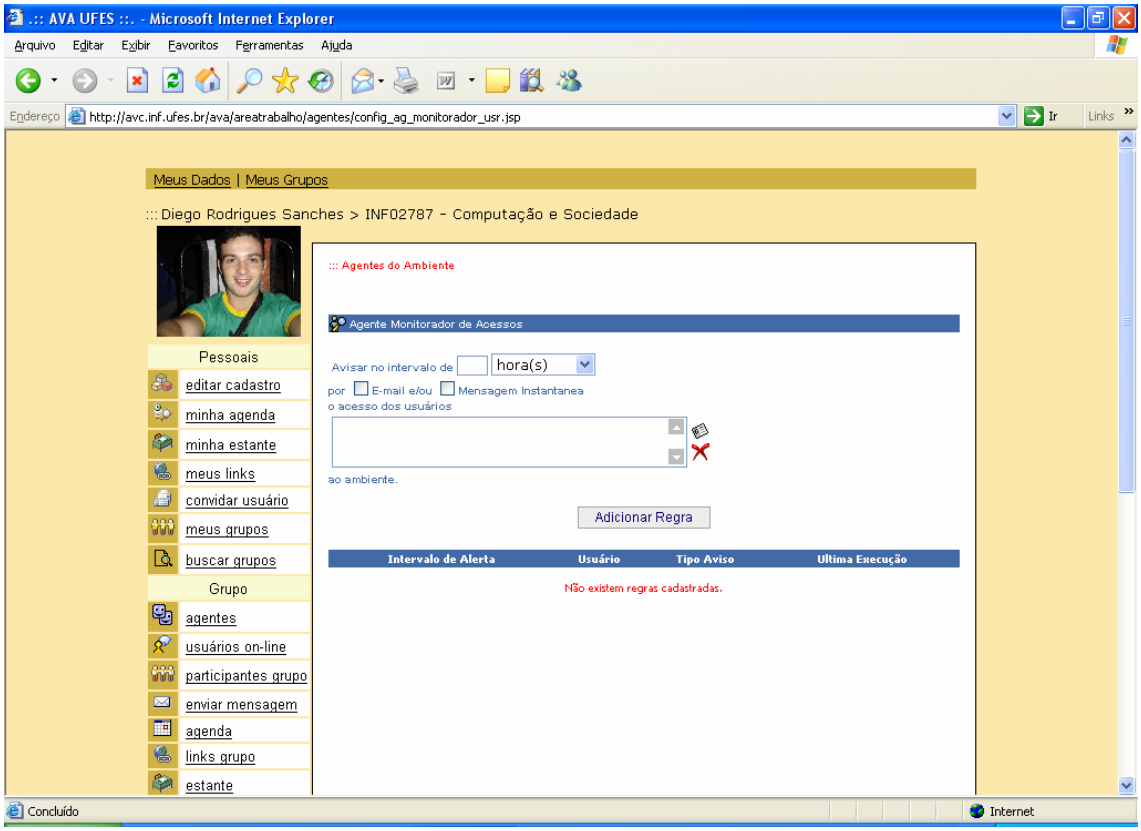

FIGURA 43 – Interface de Configuração do Agente Monitorador de Acessos

7.4.4 Agente Monitorador de Agenda Pessoal

A Figura 44 apresenta a interface de configuração do agente monitorador da agenda pessoal. Esse agente permite ao usuário (aprendizes e mediadores) criar regras referentes aos compromissos cadastrados em sua agenda pessoal. O agente, de acordo com as regras criadas pelo usuário, enviará alertas, via e-mail e mensagem instantânea, para o usuário quando um determinado compromisso de sua agenda estiver próximo de expirar ou quando um novo compromisso for cadastrado na agenda.

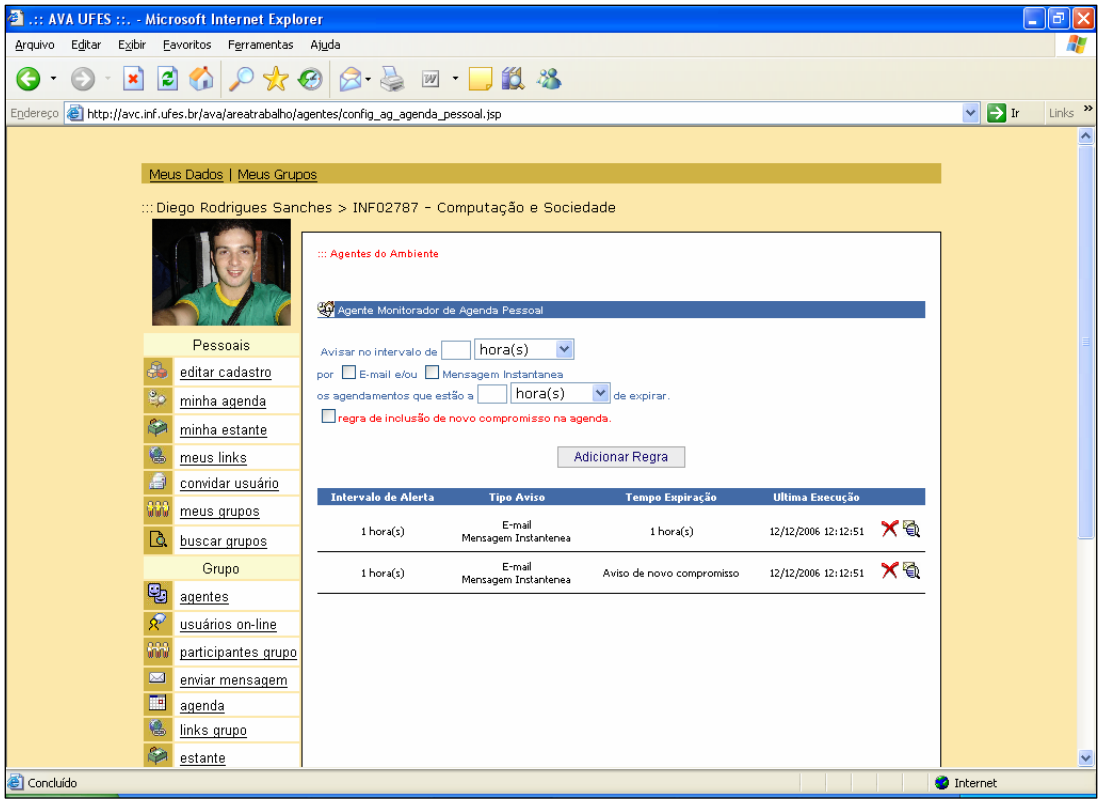

FIGURA 44 – Interface de Configuração do Agente Monitorador de Agenda Pessoal

7.4.5 Agente Monitorador da Agenda do Grupo

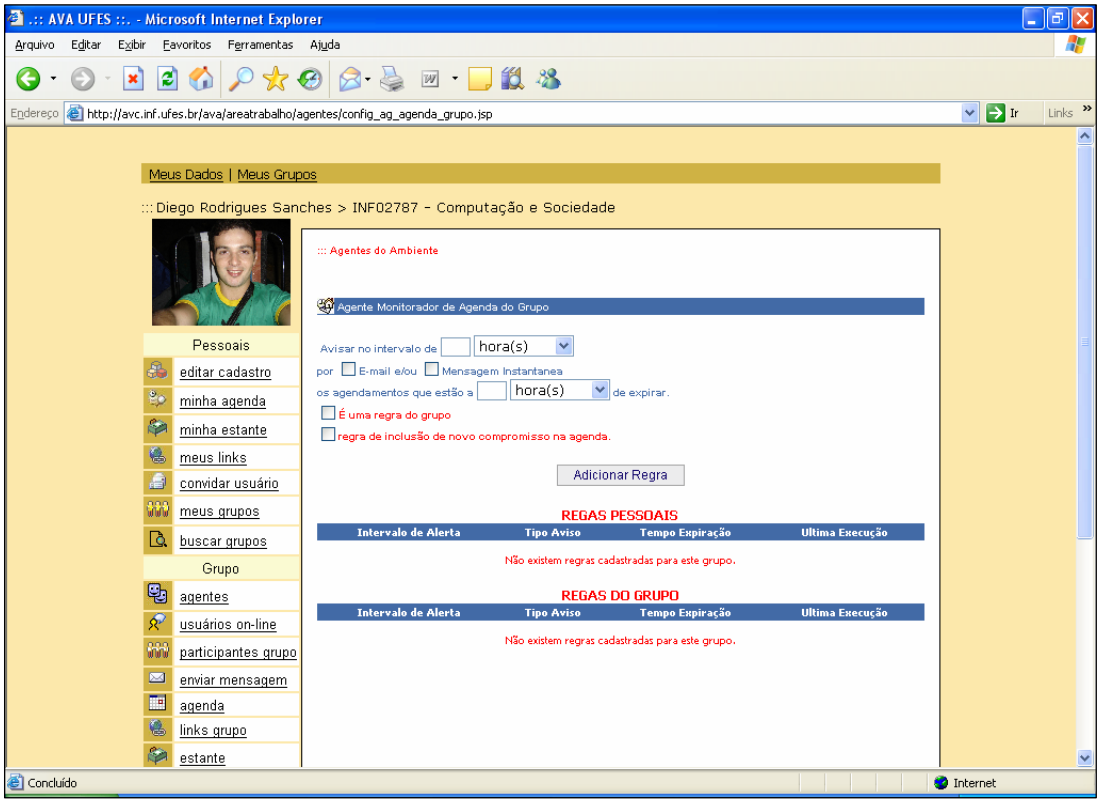

FIGURA 45 – Interface de Configuração do Agente Monitorador de Agenda do Grupo

A Figura 45 apresenta a interface de configuração do agente monitorador da agenda do grupo que permite ao usuário (aprendiz e mediador) criar regras referentes aos compromissos cadastrados na agenda de um grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário quando um determinado compromisso de sua agenda estiver próximo de expirar, de acordo com as regras criadas pelo usuário, ou quando um novo compromisso for cadastrado na agenda. No caso do usuário ser um mediador, ele ainda poderá marcar a regra como uma regra do grupo, fazendo que desta forma o agente trabalhe para o grupo, enviando alertas para todo grupo e não somente para o usuário criador da regra.

7.4.6 Agente Monitorador de Estante do Grupo

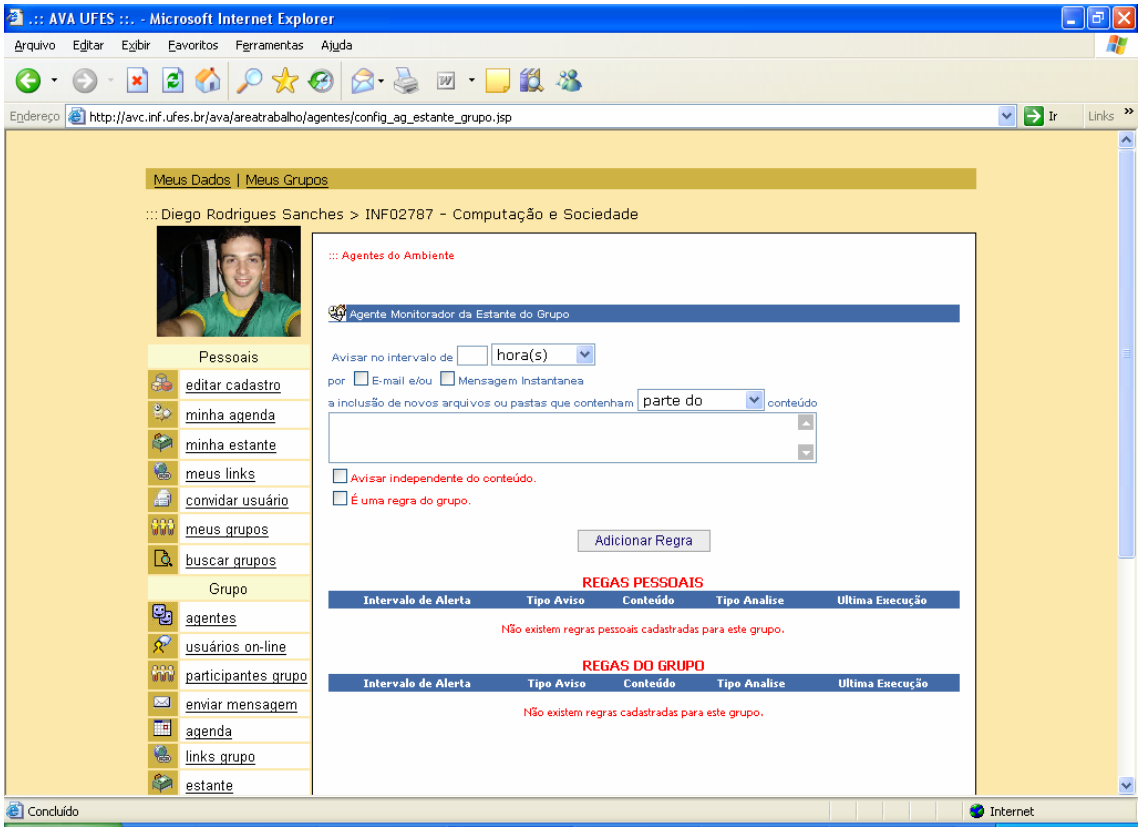

FIGURA 46 – Interface de Configuração do Agente Monitorador da Estante de Grupo

A Figura 46 apresenta a interface de configuração do agente monitorador da estante do grupo que permite ao usuário (aprendiz e mediador) criar regras referentes aos arquivos enviados para a estante do grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário quando um arquivo qualquer for inserido na

estante ou quando um determinado arquivo, com determinado conteúdo de interesse do usuário, for inserido no grupo. No caso do usuário ser um mediador, ele ainda poderá marcar a regra como uma regra do grupo, fazendo que dessa forma o agente trabalhe para o grupo, enviando alertas para todo grupo e não somente para o usuário criador da regra.

7.4.7 Agente Monitorador do Escaninho de Usuários dos Grupos

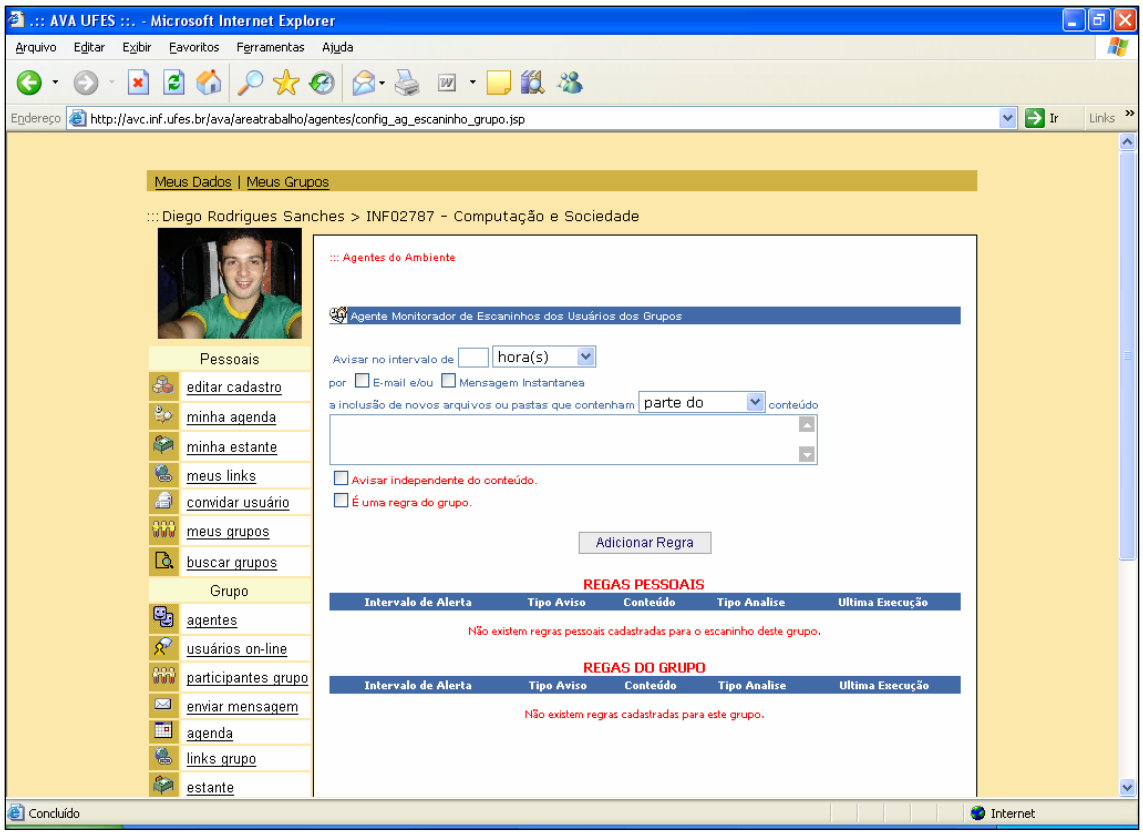

FIGURA 47 – Interface de Configuração do Agente Monitorador do Escaninho de Usuários dos Grupos

A Figura 47 apresenta a interface de configuração do agente monitorador do escaninho de usuários dos grupos que permite ao usuário (aprendiz e mediador) criar regras referentes aos arquivos enviados para seu escaninho em determinado grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário do grupo quando um arquivo qualquer for inserido em seu escaninho ou quando um arquivo com determinado conteúdo tiver sido inserido em seu escaninho. No caso do usuário ser um mediador, ele ainda poderá marcar a regra como uma regra do

grupo, fazendo que desta forma o agente trabalhe para o grupo, enviando alertas para todo grupo e não somente para o usuário criador da regra.

7.4.8 Agente Monitorador de Mensagens do Forum do Grupo

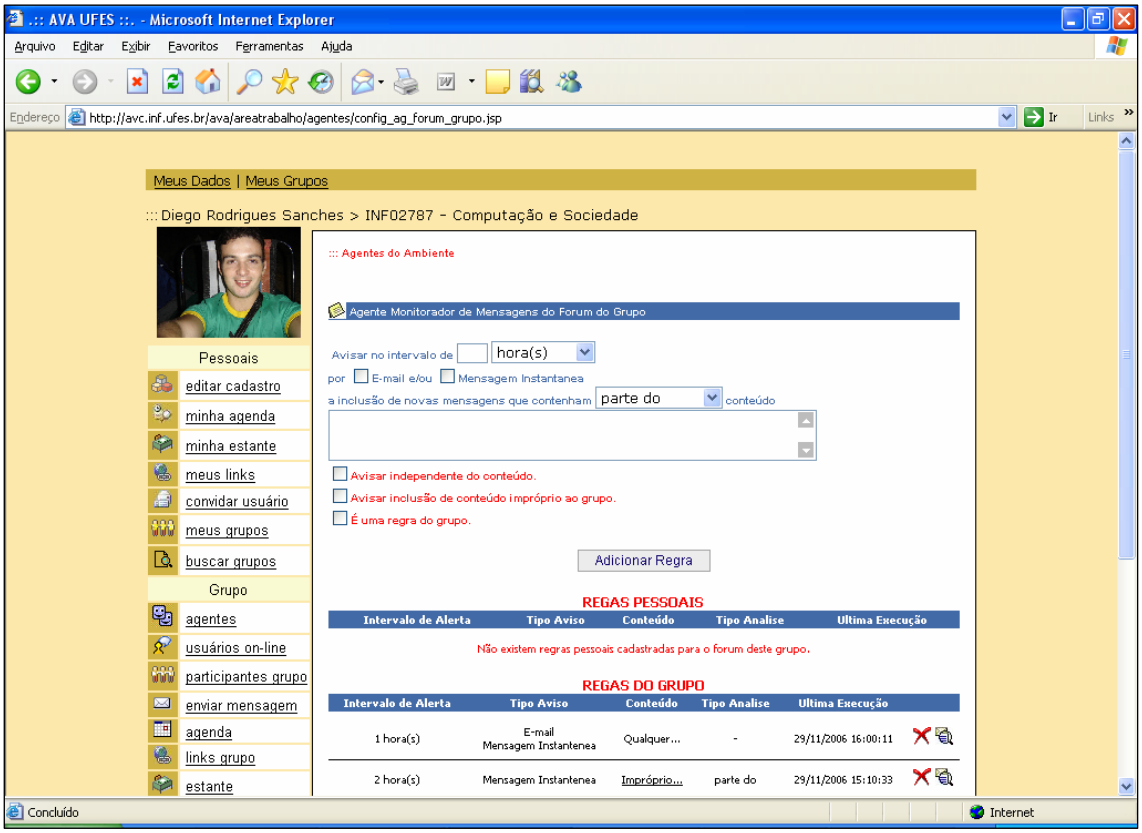

FIGURA 48 – Interface de Configuração do Agente Monitorador de Mensagens do Fórum do Grupo

A Figura 48 apresenta a interface de configuração do agente monitorador de mensagens do fórum do grupo que permite ao usuário (aprendiz e mediador) criar regras referentes ao conteúdo postado no fórum de seu grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário do grupo quando uma nova mensagem for inserida em seu fórum ou quando uma mensagem for inserida no fórum de seu grupo. No caso do usuário ser um mediador, ele ainda poderá marcar a regra como uma regra do grupo, fazendo que o agente trabalhe para o grupo, enviando alertas para todo grupo e não somente para o usuário criador da regra, e também pode criar regras de filtro de conteúdo impróprio, fazendo com que o agente o avise no caso de uma mensagem com conteúdo que ele julgar impróprio ser cadastrada no fórum do grupo.

7.4.9 Agente Monitorador do *Chat* do Grupo

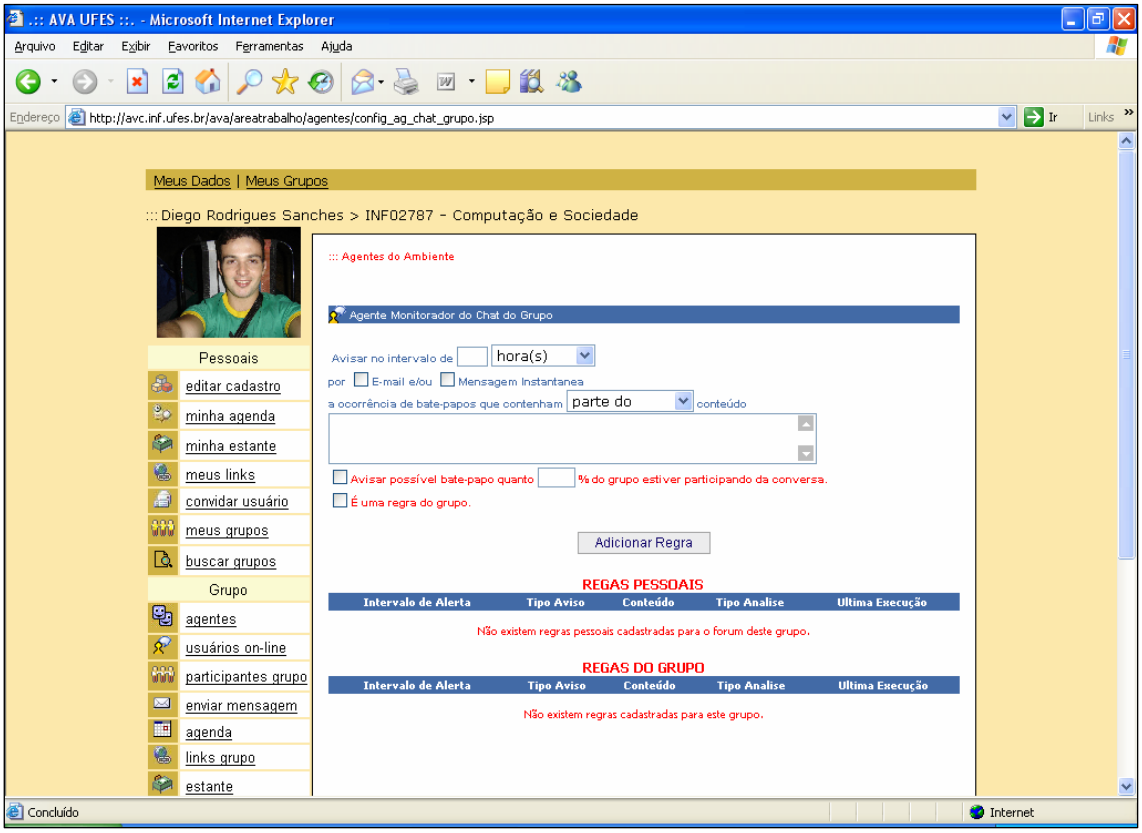

FIGURA 49 – Interface de Configuração do Agente Monitorador do Chat do Grupo

A Figura 49 apresenta a interface de configuração do agente monitorador de *chat* do grupo que permite ao usuário (aprendiz e mediador) criar regras referentes ao conteúdo das conversas realizadas no *chat* de seu grupo. O agente enviará alertas, via e-mail e mensagem instantânea, para o usuário do grupo quando um determinado conteúdo for discutido no *chat* de seu grupo ou quanto uma porcentagem X de usuários de seu grupo estiver simultaneamente na sala de bate-papo do grupo. No caso do usuário ser um mediador, ele ainda poderá marcar a regra como uma regra do grupo, fazendo que o agente trabalhe para o grupo, enviando alertas para todo grupo e não somente para o usuário criador da regra.

7.5 Considerações finais do Capítulo

É bastante significativo o ganho que os agentes trouxeram para os usuários participantes dos grupos e do AVAUfes em si, tanto em relação à produtividade dos

usuários quanto à facilidade de uso e controle dessas ferramentas.

Apesar da necessidade inicial do usuário aprender a usar mais essas funcionalidades do ambiente, o ganho em volume de informação e na agilidade de se obter essas informações resultantes das ações dos agentes permite que os usuários fiquem mais tranqüilos em relação aos seus compromissos e às atividades realizadas por seu grupo, pois agora estão constantemente sendo avisados sobre o que está acontecendo em seus grupos e agora podem investir mais atenção na aprendizagem.

8 CONSIDERAÇÕES FINAIS

No cenário mundial atual, quanto ao uso da tecnologia da informação para apoiar a educação, diversos ambientes virtuais de apoio à aprendizagem (AVA) têm surgido, com diferentes propostas pedagógicas que visam explorar os recursos disponíveis na Internet para potencializar a aprendizagem.

Apesar dessa grande diversidade de sistemas (AVA) e propostas, percebe-se ainda frustração nos aprendizes e mediadores ao usarem esses recursos. Um componente importante dessa frustração é o fato desses AVA oferecerem suporte somente a algumas tarefas, não existindo um ambiente integrado e dinâmico capaz de suportar atividades colaborativas como o compartilhamento das atividades, o registro de evolução de atividades, o apoio às atividades de mediação, cooperação e, finalmente, o suporte personalizado para o aprendiz e para o mediador, no que se diz respeito ao uso diário e contínuo das funcionalidades desses ambientes.

O que se tem, no cenário atual de ambientes virtuais de aprendizagem, são propostas que ainda carecem de um suporte mais adequado e personalizado para resultarem efetivamente em contribuições que melhorem o processo de ensino-aprendizagem.

Foi com base nesse cenário que este trabalho foi proposto, visando inicialmente levantar a maioria dos problemas e carências desses ambientes, e propor com base nesse levantamento uma forma simples e acoplável de solução e melhoria dos AVA.

A solução apresentada tem base em diversas tecnologias, o que tornou o desenvolvimento do trabalho muito proveitoso, no que se diz respeito a aplicar conhecimentos das áreas da Ciência da Computação e da Pedagogia, dentre outras.

Para um estudo de caso mais aprofundado, foi desenvolvido o AVAUFES, que desde o início de sua implantação foi utilizado como laboratório com diversas turmas da instituição (UFES) a fim de refinar os requisitos e por a prova as soluções propostas durante o trabalho.

A utilização da plataforma multiagente nesse ambiente tornou sua utilização mais simples e dinâmica. No caso do mediador houve uma melhora significativa no que diz respeito ao controle de seu grupo, deixando que a própria plataforma multiagente trabalhe por ele nas atividades do dia-a-dia, mantendo-o sempre informado da atual situação de seu grupo, liberando-o para sua função principal que é auxiliar na aprendizagem dos participantes de cada grupo. Já em relação ao aprendiz, ele se tornou mais integrado ao ambiente, pois a plataforma multiagente permite um acompanhamento mais direto e contínuo de seu grupo, atividades e compromissos diários. Os aprendizes do grupo estão mais informados sobre o acompanhamento das atividades de cada grupo e sobre o uso das funções disponíveis no ambiente. O controle das funções e ferramentas se tornou uma atividade mais simples e dinâmica, pois o sistema passou a trabalhar pelo aprendiz e não simplesmente com ele, uma vez que cada participante passou a ter um conjunto de agentes configuráveis de acordo com suas necessidades e vontades, e estes por sua vez estão 24 horas por dia no ar monitorando todo o ambiente, seus grupos e informações a fim de mantê-lo informado de tudo que acontece em seu grupo de trabalho.

8.1 Perspectivas futuras

Durante o desenvolvimento deste trabalho um grande leque de trabalhos relacionados a ele foi sendo vislumbrado, envolvendo novas funções a serem acopladas ao ambiente e novos agentes que poderão ser definidos para o auxilio e solução de diversos problemas.

Com o curto intervalo de tempo e a necessidade de se desenvolver um ambiente novo que pudesse ser utilizado como base para a plataforma

multiagente, construiu-se um número razoável de funcionalidades e agentes necessários para suportar os requisitos levantados. Entretanto, vários requisitos identificados durante o trabalho não constam no projeto. Entre eles citam-se:

- definição de agentes para auxílio gráfico à utilização das ferramentas do ambiente. Através da presença de assistentes pessoais, para acompanhar o aprendiz durante toda a sua navegação dentro do ambiente, inclusive posicionando-o a respeito das últimas ações realizadas por ele próprio e pelo grupo do qual faz parte;
- definição de agentes para criação e definição de interfaces adaptativas, permitindo, por exemplo, que o sistema configure o ambiente e ferramentas de acordo com as preferências e perfil do usuário (ex: forma de visualização das mensagens nas ferramentas de comunicação, ordenação de menus por ferramentas mais utilizadas, ordenação de grupos mais acessados);
- definição de uma camada para ser acoplada a plataforma multiagente, que utilize uma linguagem formal, simples e direta para definir novos agentes para o sistema, sem que haja a necessidade de codificação na solução em si;
- definição de uma interface baseada em avatares, onde o ambiente possa se apresentar como um Ambiente Virtual de Aprendizagem Incorporado, onde o aprendiz, não só utilize as ferramentas do sistema, mas também possa participar de um universo virtual totalmente novo.

A utilização da plataforma multiagente em outros ambientes virtuais de aprendizagem também é importante para testar seu real nível de acoplamento a soluções diversas. É muito importante que a utilização do ambiente e da plataforma multiagente estejam em constante acompanhamento, pois com o aumento do uso do sistema e o avanço das atuais tecnologias, novas idéias e necessidades irão aparecer. Essa continua evolução é o que pode garantir o real sucesso deste trabalho.

REFERÊNCIAS

AZEVEDO, Breno Fabrício Terra; TAVARES, Orivaldo de Lira; CURY, Davidson. *MUTANTIS - Uma Arquitetura Multi-Agente para a Autoria de Tutores Inteligentes*. Dissertação (Mestrado em Informática) - CT/UFES. Vitória-ES, 1999.

Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. (2003): Jade – A White Paper. EXP in search of innovation. *Special issue on Jade, v*. 3, n. 3, p. 6-19, Sept. 2003.

BIGUS, Joseph P.; BIGUS, Jennifer. *Constructing intelligent agents using JAVA.* 2. ed. New York: Willey, 2001

BOOCH, Grady. *Object-Oriented Analysis and Design with Applications*. 2. ed. Massachusets: Addison-Wesley, 1994.

BRITO, Silvana R.; TAVARES, Orivaldo De Lira; MENEZES, Crediné S. *A* Society of Intelligent Agents for an Environment of Cooperative Online Learning. Proceedings of the IC-AI'2000. Regular Research Report (RRR): Monte Carlo Resort, Las Vegas, Nevada, USA., p. 699-703, 26-29 June, 2000.

BRITO, Silvana Rossy de. *MEDIADOR – Ambiente para a aprendizagem orientada a projetos com suporte inteligente à mediação*. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Espírito Santo, Vitória, 2001.

BRITO, Silvana R.; TAVARES, Orivaldo De Lira; MENEZES, Crediné S. *An Environment of Cooperative Online Learning for Software Engineering*. Proceedings of the International Conference on Engineering and Computer Education (ICECE2000). São Paulo/BR, ago/2000.

BRITO, Silvana Rossy; TAVARES, Orivaldo De Lira; MENEZES, Crediné Silva. Um ambiente para educação continuada em engenharia de software. In: SBIE´99 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 10,1999, Curitiba-PR. *Anais*... p. 405-407, Curitiba: UFPR, 1999.

BRITO, Silvana Rossy; TAVARES, Orivaldo de Lira. Agentes Inteligentes: definições, aplicações e comunicação. *Revista Engenharia - Ciência & Tecnologia*, CT/UFES, Vitória-ES, Ano 2, N. 9, p. 13-20, março-abril/1999. ISSN 1414-8692.

BRITO, Silvana Rossy; TOGNERI, Denise Franzotti; TAVARES, Orivaldo de Lira; MENEZES, Crediné Silva; FALBO, Ricardo de Almeida. *Um Sistema Multiagente para Gerência de Reuniões em Ambientes de Aprendizagem Cooperativa*. In: Workshop de Ambientes de Aprendizagem baseados em Agentes, 2.,2000, Maceió: UFAL, nov. 2000.

BRITO, Silvana Rossy; GAVA, Tânia Barbosa Salles; TAVARES, Orivaldo de Lira, MENEZES, Crediné Silva. Metodologias para desenvolvimento de sistemas multiagentes: Visão geral e comparação. In: SBIA – SIMPÓSIO BRASILEIRO DE INTELIGÊNCIA ARTIFICIAL, 9, 2001. *Anais*… Fortaleza: 2001.

BLOOM, B. S., et al. *Taxionomia dos objetivos educacionais*. Porto Alegre: Globo, 1974.

BRUCKMAN, Amy. *MOOSE Crossing: Construction, Community, and Learning in a Networked Virtual World for Kids, 1997.* [online] Disponível em: http://asb.www.media.mit.edu/people/asb/thesis/index.html. [capturado em: 16 jul. 1999]

BUCHANAN, W.J.; NAYLOR, M. and SCOTT, A.V. *Enhancing Network Management using Mobile Agents*. Proceedings of the 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems 3 - 7, April, 2000, Edinburgh, Scotland. Disponivel em: <http://computer.org/proceedings/ecbs/0604/06040218abs.htm>. Acesso em: 03 mar. 2001.

BUNUEISTER, Birgit. *Models and methodologies for agent-oriented analysis and design*. In: FISCHER, Klaus (ed.). Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems. 1996. DFKI Document D-96-06.

CASTRO, Glauco Palassi Cupertino de Castro; CURY, Davidson; MENEZES, Crediné Silva de. SAMIR: Assistente pessoal de perguntas. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12.,2001, Vitória-ES. *Anais*..., Vitória: UFES, 2001.

COSTA, Evandro Barros. *Um Modelo de Ambiente Interativo de Aprendizagem Baseado numa Arquitetura Multiagente*. 1997. 145p. Tese (Doutorado em Processamento da Informação) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal da Paraíba.

EJB (2003): Sun Microsystems Enterprise JavaBeans Specification, Version 2.1, Nov. 2003.

EJB (2005): Enterprise JavaBeans. Site oficial disponível em <http://java.sun.com/products/ejb/index.jsp>. Acesso em: 30 mar. 2006.

FIPA. FIPA ACL message structure specification. 2002. Disponível em: <http://www.fipa.org/specs/fipa00061/SC00061G.html>. Acesso em: 30 mar. 2005.

___. FIPA contract net interaction protocol specification. 2001. Disponível em: <http://www.fipa.org/specs/fipa00029/XC00029F.html>. Acesso em: 30 mar. 2005.

FIPA (2005): Página oficial da organização FIPA disponível em <http://www.fipa.org/>. Acesso em: 30 mar. 2005.

GAVA, Tânia B. S.; MENEZES, Crediné S. Ambientes cooperativos para aprendizagem orientada à projeto. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 10., 1999. *Anais*..., Curitiba: UFPR, 1999.

GAVA, Tânia B. S.; MENEZES, Crediné S. Moonline: Um ambiente de aprendizagem cooperativa baseado na WEB para apoio às atividades extraclasse. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória-ES. *Anais*..., Vitória: UFES, 2001.

GIRAFFA, L.M.M. *Uma arquitetura de tutor utilizando estados mentais*. 1999. 177 p. Tese (Doutorado em Ciência da Computação) CPGCC/UFRGS, Porto Alegre.

GIRAFFA, L.M.M.; NUNES,M.A; VICCARI,R.M. Multi-Ecological: proposta de Ambiente de Ensino Inteligente utilizando arquitetura de Sistemas Multiagentes. In: ENIA: ENCONTRO NACIONAL DE INTELIGÊNCIA ARTIFICIAL, 1, 1997, Brasília. *Anais...* Brasilia:UnB,1997.

GOULART, Rodrigo R. V.; GIRAFFA, Lucia M. M. Utilizando a tecnologia de agentes na construção de sistemas tutores inteligentes em ambiente interativo. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12.,2001, Vitória-ES. *Anais*..., Vitória: UFES, 2001.

GRÉGOIRE, R.; LAFERRIÈRE, T. *Project-based collaborative learning with network computers:* teacher's guide. Canada's Schoolnet, 2001. Disponível em: <http://www.tact.fse.ulaval.ca/ang/html/projectg.html#anchor387673>. Acesso em 10 agosto 2005.

IGLESIAS, Carlos A.; GARIJO, Mercedes; GONZÁLEZ, José C.; VELASCO,

Juan R *A Methodological Proposal for Multiagent Systems*. In: B. GAINES, B. e MUSEN, M. (eds). *10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW96)*, Banff, Canada, November 1996. vol. 1, pp. 25-1/17.

IGLESIAS, Carlos A.; GARIJO, Mercedes. *Position Paper on Software Engineering for Agent Systems*. [online]. Disponível: http://www.cs.vu.nl/~treur/SIG1.ciglesias.html. [capturado em 10 de julho de 2000].

Jade (2005): Página official do framework Java Agent Development (Jade) disponível em <http://jade.tilab.com/>. Acesso em: 30 mar. 2005.

JavaBeans (2005): Especificação do padrão JavaBeans. Disponível em <http://java.sun.com/beans>. Acesso em: 12 ago. 2005.

JBoss (2005): Página oficial do servidor de aplicações JBoss disponível em <http://www.jboss.com/developers/index>. Acesso em: 10 fev. 2005.

JENNINGS, N. R.; WOOLDRIDGE, M. J. *Agent Technology: foundations, applications, and markets*. New York: Springer, 1998.

KINNY, D.; GEORGEFF M.; RAO, A. A methodology and modelling technique for systems of BDI agents. In: VAN DER VELDE, W.; PERRAM, J. (eds.): *Agents Breaking Away. Proceedings on Workshop on Modelling Autonomous Agents in a MultiAgent World*, MAAMAW-96, 7, pag. 56-71. Springer-Verlag: Berlin, Germany, 1996. MAAMAW'96, LNAI 1038, Springer, Berlin, Germany,1996.

LABIDI, Sofiane, SILVA; Josenildo C.; COUTINHO, Luciano R.; COSTA, Nilson S.; COSTA, Evandro de Barros. Agent-Based Architecture for a Cooperative Learning Environment. In: SBIE´00 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11., 2000, Maceió-AL. *Anais*..., Maceió: UFAL, 2000.

MAES, Patie. Artificial Life Meets Entertainment: Life Like Autonomous Agents. *Communications of the ACM*, v. 38, n. 11, nov. 1995.

MAGELA, R. *Produzindo software orientado a objetos: projeto*. Rio de Janeiro: FUZION, 1998.

MARTIN-FLAT, Jean-Philippe; ZNATY, Simon. A simple typology of distributed network management paradigms. In: IFIP/IEEE INTERNATIONAL WORKSHOP

ON DISTRIBUTED SYSTEMS: OPERATIONS & MANAGEMENT, 8., 1997, Sydney.

MENEZES, C.S.; CURY, D.; CAMPOS, G.H.B.; CASTRO Jr., A. N.; TAVARES, O. L. *AmCorA – Um Ambiente Cooperativo para a Aprendizagem Construtivista Utilizando a Internet,* Projeto de Pesquisa: DI/CT/UFES, 1999.

MENEZES, C.S., CURY, D., CAMPOS, G.H.B. AmCorA: Um Ambiente Cooperativo para a Aprendizagem Construtivista Utilizando a Internet. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO; 10., 1999a; Curitiba; *Anais*...Curitiba: UFPR, 1999. p. 333-340.

MENEZES, C.S; CASTRO, A. N.; PINT, S. C. C.; CURY, D.; TAVARES, O. L. *Um framework para ambientes inteligentes de apoio à aprendizagem cooperativa.* [online]. Disponível em <http://www.gaia.ufes.br> . [Capturado em 10.12.2001]

NWANA, Hyacinth; NDUMU, Divine; LEE, Lyndon; COLLIS, Jaron. ZEUS: A Tool-Kit for Building Distributed Multi-Agent Systems. *Applied Artificial Intelligence Journal*, v. 13, n. 1, p. 129-86, 1999.

PESSOA, José Marques; MENEZES, Crediné Silva de. QSABE II: A Cooperative Service for Knowledge Appropriation and Diffusion Using the Internet. In: IEEE ICECE2000 – INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION. *Anais*… Sao Paulo, 2000.

PRESSMAN, R. S. *Software Engineering: A practitioner's Approach*. 5. ed. Boston: Mc-Graw-Hill, 2000. 860 p.

PESSOA, J. M.; Netto, H. V.; Menezes, C. S. "FAmCorA: um *framework* para a construção de ambientes cooperativos inteligentes de apoio a aprendizagem na Internet baseado em web services e agentes". In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 13., 2002, São Leopoldo–RS, *Anais*... São Leopoldo: UNISINOS, 2002, p.94-103.

RUSSELL, Stuart; NORVIG, Peter. *Artificial Intelligence - A Modern Approach*. New Jersey: Prentice-Hall, 1995.

RUSSEL, Stuart J.; NORVIG, Peter. *Artificial intelligence*: a modern approach. 2. ed. New Jersey: Prentice-Hall, 2003.

SANTOS, Cássia Trojahn; FROZZA, Rejane; DAHMER, Alessandra; GASPARY, Luciano Paschoal. Dóris – Um agente de acompanhamento

pedagógico em sistemas tutores inteligentes. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12.,2001, Vitória-ES. *Anais*..., Vitória, UFES, 2001.

SANTOS, Marcelo Coelho dos; PESSOA, José Marques. Aplicando Mapas Conceituais para Recuperar Informação em um Ambiente Virtual de Aprendizagem. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória-ES. *Anais*..., Vitória, UFES, 2001.

SANTOS, Neide. Estado da arte em espaços virtuais de ensino e aprendizagem. *Revista Brasileira de Informática na Educação – SB*C, n.4, 1999. Disponível em: <http://www.inf.ufsc.br/sbc-ie/revista/nr4/070TU-santos.htm>. Acesso em 25 abr, 2001.

SHERMAN, S. J. *Cooperative Learning and Science*. Handbook of Cooperative Learning Methods. Westport: Praeger; 1994. p. 227-44.

SHOHAM, Y. Agent-oriented Programming. Artificial Intelligence. *Springer Veralg*, Berlin, v. 60, p. 51- 92, 1993.

SLEEMAN, D. Assessing aspects of competence In basic algebra. In: SLEEMAN, D.; BROWN, J.S. (Eds.), *Intelligent Tutoring Systems.* New York: Academic Press, 1982.

SMYSER, B.M. *Active and Cooperative Learning*, 1993.

SOUZA, Renata Silva; MENEZES, Crediné Silva. Aplicando técnicas de recuperação de informações para facilitar a interação em ambientes cooperativos: uma abordagem multiagentes. In: SBIE´01 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12.,2001, Vitória-ES. *Anais*..., Vitória: UFES, 2001.

SOUZA, Renata Silva; MENEZES, Crediné Silva. Social Interctions in Cooperative Learning Environments. In: IEEE ICECE2000 – INTERNATIONAL CONFERENCE ON ENGINEERING AND COMPUTER EDUCATION. *Anais*… Sao Paulo, 2000.

SOUZA, Renata Silva, MENEZES, Crediné Silva. Um Sistema Inteligente para apoio à Interação em Ambientes Cooperativos de Aprendizagem. In: SBIE´00 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11.,2000, Maceió-AL. *Anais*..., Maceió: UFAL, 2000.

TAVARES, Orivaldo de Lira; BRITO, Silvana Rossy; SOUZA, Renata Silva;

MENEZES, Crediné Silva. Ambiente de apoio à mediação de aprendizagem: Uma abordagem orientada por processos e projetos. In: SBIE´00 SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11.,2000, Maceió-AL. *Anais*..., Maceió: UFAL, 2000.

TAVARES, Orivaldo de Lira, BRITO, Silvana Rossy, SOUZA, Renata Silva, MENEZES, Crediné Silva. Ambiente de apoio à mediação de aprendizagem: Uma abordagem orientada por processos e projetos. *Revista de Informática na Educação*. 2001.

VICCARI, R. *Um Tutor Inteligente para a programação em Lógica: Idealização, Projeto e Desenvolvimento,* Universidade de Coimbra, 1990.

VIEIRA, Fábia Magali Santos. *A pedagogia de projetos*. [online]. Disponível: http://www.connect.com.br/~ntemg7. [Capturado em 10 de maio de 2000].

VIGOTSKY, L.S. *A Formação Social da Mente.* Martins Fontes, 1984.

XDoclets (2005): Página oficial do XDoclet disponível em <http://xdoclet.sourceforge.net/xdoclet/index.html>. Acesso em: 12 out. 2005.

Wagner G., 2002 The Agent-Object-Relationship metamodel: Towards a unified conceptual view of state and behavior. Technical report, Eindhoven Univ. of Technology, Fac. of Technology Management, Disponível em http://AOR.rezearch.info. Acesso em: 12 mar. 2006.

Wagner G., 2003 The Agent-Object-Relationship Meta-Model: Towards a Unified Conceptual View of State and Behaviour, *Information Systems,* v. 28, n. 5, p. 475 – 504, 2003.

WENGER, E. *Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge.* Morgan Kaufmann Publishers, Inc. California, USA, January 1987.

WOOLDRIDGE, M. CIANCARINI, P. Agent-Oriented Software Engineering: The State of the Art. In: CIANCARINI, P. e WOOLDRIDGE, M. (eds). *Agent-Oriented Software Engineering*. Springer-Verlag Lecture Notes in AI Volume 1957, Jan, 2001.

WOOLDRIDGE, M., JENNINGS, N. R., KINNY, D. A methodology for agent-oriented analysis and design. In: ETZIONI. O.; MULLER, J. P. e BRADSHAW, J. (eds). *Agents '99: Proceedings on*. Autonomous Agents, Seattle, WA, 1999, Pag. 69 - 76. [online]. Capturado em 20-nov-1999. Disponível em:

http://www.csc.liv.ac.uk/~mjw/pubs/

WOOLDRIDGE, M., JENNINGS, N. R., KINNY, D. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, v. 3, n. 3, p. 285-312, 2000. [online]. Capturado em 01-fev-2001. Disponível em: http://www.csc.liv.ac.uk/~mjw/pubs/

WOOLDRIDGE, Michael; JENNINGS, Nicholas R. *Agent Theories, Architetures e Languages*: a Survey Intelligent Agents. Berlin: Springer-Verlag, 1995.

YU, Lei, SCHMID; Beat F. *A Conceptual Framework For Agent Oriented and Role Based Workflow Modeling* [online]. Disponível: http://www.knowledgemedia.net/netacademy/publications.nsf/all_print_pk/131. [capturado em 10 de setembro de 2000].

APENDICE A

Framework JADE - Java Agent Development Framework

Introdução

Jade é um *middleware* para desenvolvimento e execução de aplicações *peer-to-peer* baseada no paradigma de agentes que pode facilmente trabalhar e interoperar em ambientes de redes tradicionais ou *wireless*. Os dois maiores aspectos do modelo conceitual são: topologias de sistemas distribuídos com redes *peer-to-peer* e arquitetura de componentes com paradigma de agentes. A topologia de rede especifica como os vários componentes de um sistema são conectados enquanto especifica o que supostamente um componente espera do outro.

Jade desenvolvido e suportado pelo CSELT da Universidade de Parma, na Itália, é *open source* (LGPL7). Segundo (JADE, 2003), o principal objetivo do Jade é simplificar e facilitar o desenvolvimento de sistemas multiagentes, garantindo um padrão de interoperabilidade entre sistemas multiagentes através de um abrangente conjunto de agentes de serviços de sistema, os quais tanto facilitam como possibilitam a comunicação entre agentes, de acordo com as especificações da FIPA: serviço de nomes (naming service) e páginas amarelas (*yellow-page service*), transporte de mensagens, serviços de codificação e decodificação de mensagens e uma biblioteca de protocolos de interação (padrão FIPA) pronta para ser usada. Toda sua comunicação entre agentes é feita via troca de mensagens. Além disso, lida com todos os aspectos que não fazem parte do agente em si e que são independentes das aplicações, tais como: transporte de mensagens, codificação e interpretação de mensagens e ciclo de vida dos agentes. Ele pode ser considerado como um “middle-ware” de agentes que implementa um framework de desenvolvimento e uma plataforma de agentes. Em outras palavras, uma plataforma de agentes construída no padrão FIPA e um pacote, leia-se bibliotecas, para desenvolvimento de agentes em Java.

De acordo com (BELLIFEMINE, 2003), Jade foi escrito em Java devido a características particulares da linguagem, particularmente pela programação orientada a objetos em ambientes distribuídos heterogêneos. Foram desenvolvidos

tanto pacotes Java com funcionalidades disponíveis para serem usadas, quanto interfaces abstratas para se adaptarem de acordo com as funcionalidades da aplicação de agentes.

Modelo Peer-to-peer

Em um modelo *peer-to-peer* não existe distinção entre as funções de cada ponto, ao contrário de um modelo cliente-servidor onde os papéis são bem definidos, cada par é capaz de misturar iniciativa e capacidade, ou seja, cada um pode iniciar a comunicação, ser o sujeito ou o objeto de uma requisição, e ser pró-ativo. A aplicação lógica não mais precisa estar concentrada em um servidor, mas distribuída entre todos os pontos da rede. Cada par é capaz de descobrir cada um, podendo entrar, uni-se e liberar-se a qualquer momento. O sistema é totalmente distribuído assim como o valor do serviço é distribuído através da rede permitindo que novos modelos de negócio possam ser habilitados.
Uma importante característica deste modelo é que ao contrário de um modelo cliente servidor, onde o cliente deve saber onde está o servidor, mas os servidores não precisam saber onde estão os seus clientes, a localização dos clientes e servidores é totalmente arbitrária e o serviço deve prover formas de entrada, união e liberação de pares a qualquer momento, assim como mecanismos de busca e localização de outros pares. A FIG.50 mostra alguns modelos existentes atualmente.

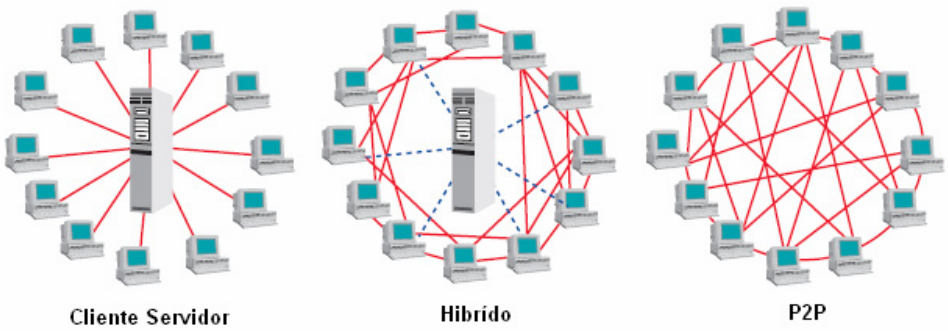

FIGURA 50 – Modelos de arquiteturas de redes de computadores distribuídas

No modelo *peer-to-peer* (P2P) os pontos são totalmente autônomos e descentralizados. A falta de um nó de referência causa dificuldades em manter uma coerência na rede e consequentemente problemas na localização de pares. A

complexidade e a largura de banda tendem a crescer exponencialmente com o número de nós.

Peer-to-peer e o paradigma de agentes

O paradigma de agentes aplica conceitos de inteligência artificial e teorias de comunicação na tecnologia de objetos distribuídos. O paradigma é baseado em uma abstração de agentes, ou seja, um componente de software que é autônomo, pró-ativo e social.

O paradigma de agentes é intrinsecamente *peer-to-peer*, cada agente é um ponto que potencialmente necessita iniciar uma comunicação com outro agente, assim como é capaz de fornecer serviços para outros agentes. O papel da comunicação é muito importante em um sistema baseado em agentes, e este sistema engloba três características importantes:

- ***Agentes são entidades ativas que podem dizer "não", e eles são facilmente conectáveis.***

Essas propriedades são as bases para a escolha de mensagens baseadas em comunicação assíncrona em vez de RPC (*Remote Procedure Call* - chamada de procedimento remoto): um agente que quer se comunicar deve apenas enviar uma mensagem a um certo destinatário. Esse tipo de comunicação na realidade permite que o receptor selecione quais mensagens ele vai atender e quais vai descartar ou quais mensagens devem ser tratadas primeiro e quais devem esperar para serem atendidas. Isso também permite que o agente emissor não fique bloqueado esperando uma resposta do agente receptor, sem qualquer dependência temporal entre as partes. Desta forma o agente receptor pode não estar disponível ou até mesmo ainda não existir no momento da comunicação.

- ***As ações e comunicações executadas pelos agentes são consideradas como sendo apenas um tipo de ação***

Fazer com que a comunicação esteja no mesmo nível das ações permite a um

agente, por exemplo, ‘raciocinar’ sobre um plano que inclua ambas: ações físicas (ex: virar para a esquerda) e ações de comunicação (ex: pedir para fechar uma porta).

Para fazer com que a comunicação seja planejável, efeitos e pré-condições de cada possível comunicação devem estar muito bem definidos.

- ***Comunicações transportam um significado semântico***

Quando um agente é um objeto de uma ação comunicativa (ex: quando recebe uma mensagem), ele deve ser capaz de entender o significado da ação e porque esta ação deve ser executada. Isso leva a uma necessidade de uma semântica universal, ou seja, a uma padronização.

FIPA – The Foundation for Intelligent Physical Agents

Em 1996 a Tilab (oficialmente CSELT) promoveu a criação da FIPA, uma associação sem fins lucrativos de companhias e organizações, compartilhando um mesmo esforço e objetivo de padronizar as especificações para a tecnologia de agentes.
Em 1997 foi lançado o primeiro conjunto de especificações e mais tarde, em 2002, a FIPA lançou o seu padrão FIPA2000. Desde junho de 2005 a FIPA foi incorporada ao IEEE, como sendo o décimo primeiro comitê, para promover estudos da tecnologia baseada em agentes e a interoperabilidade de seus padrões com outras tecnologias.
O padrão FIPA é focado em interoperabilidade, como conseqüência focado no comportamento externo dos componentes do sistema, deixando em aberto os detalhes internos de implementação e a arquitetura como demonstrado na FIG. 51.

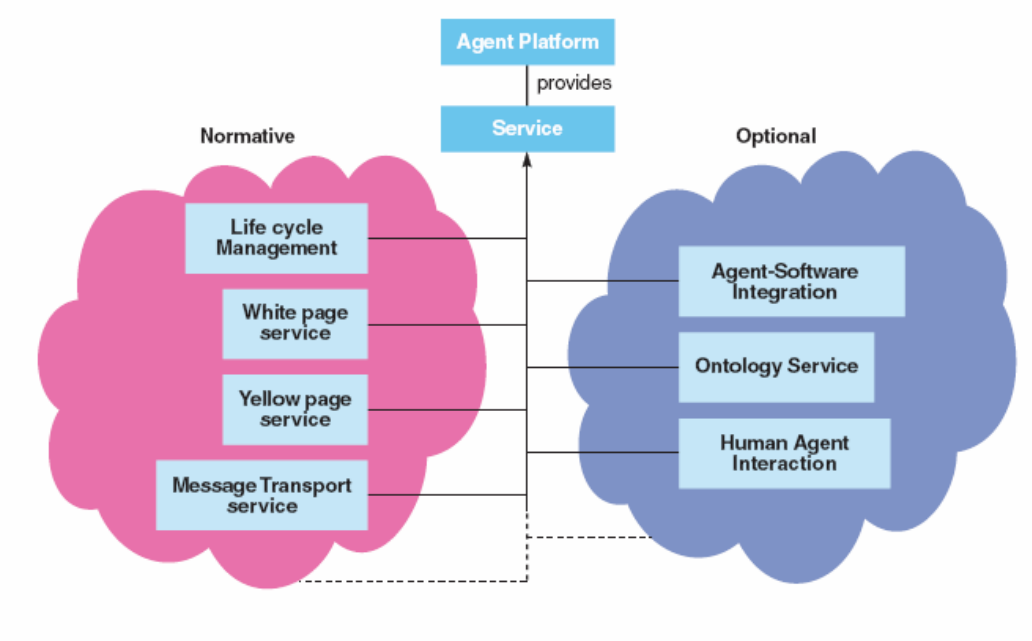

FIGURA 51 - Padrão FIPA – Serviços fornecidos pela plataforma

De fato, a arquitetura interna do JADE é a única que está completamente em conformidade com os padrões da FIPA. Estes padrões adotam totalmente o paradigma de agentes e, em particular, ele define um modelo de plataforma de agentes além do conjunto de serviços que devem ser disponibilizados. O conjunto desses serviços e dessas interfaces padronizadas permite que uma sociedade de agentes exista, opere e se gerencie.

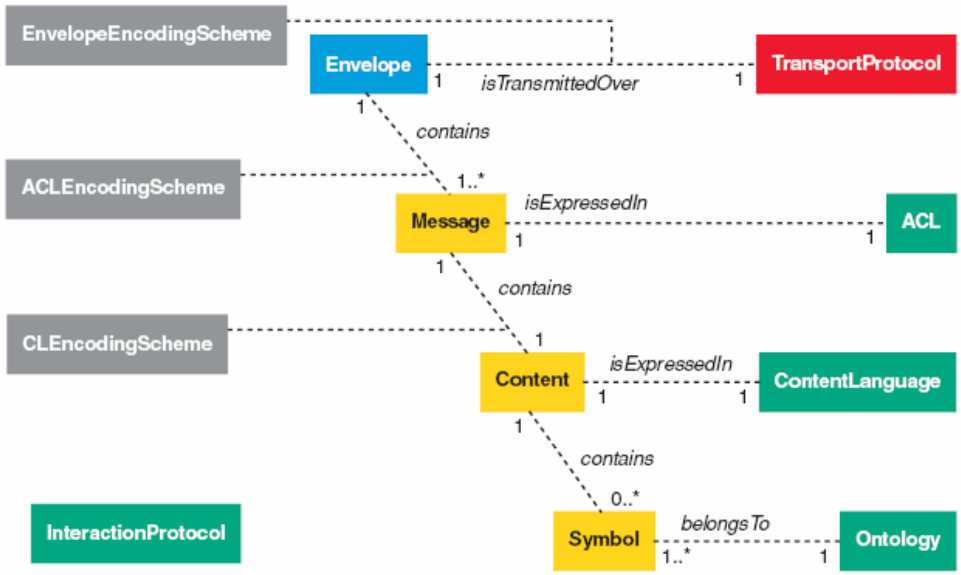

FIGURA 52 - Componentes do modelo de comunicação padronizado pela FIPA

Middleware

O termo *middleware* significa *"software que permite a interconexão entre duas diferentes e separadas aplicações"* no escopo do JADE seriam todas as bibliotecas de alto nível que permite facilmente e de forma mais efetiva o desenvolvimento de aplicações fornecendo serviços genéricos, úteis não somente para uma única, mas para uma variedade de aplicações como: acesso a dados, controle de recursos, entre outros. Os serviços são fornecidos pelos sistemas operacionais, mas a idéia por detrás do *middleware* é fornecer API's independentes da plataforma agregando facilidades nativas dentro de simples blocos reutilizáveis. O *middleware* baseado nessa abordagem permite uma redução considerável no tempo de desenvolvimento.

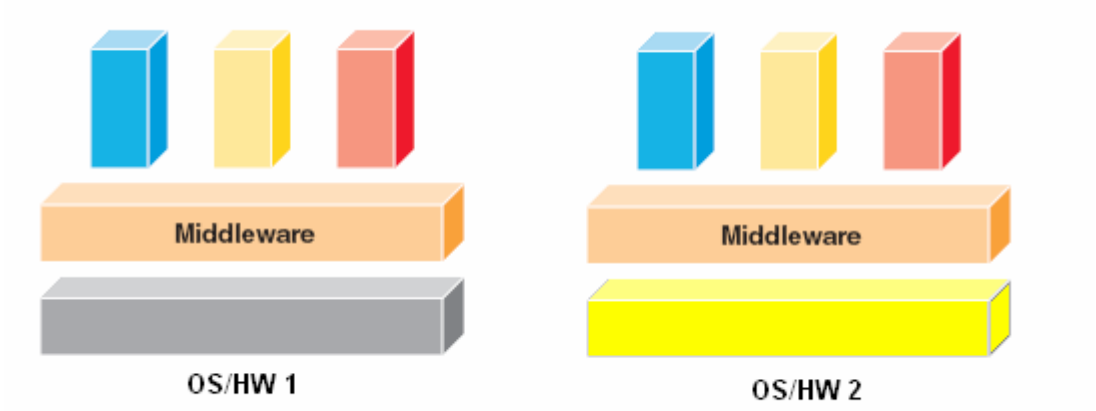

FIGURA 53 - Função de um middleware

Como demonstrado pela FIG.53 o conceito do *middleware* é fornecer API's independentes de plataforma, adicionando as aplicações que os utilizam funcionalidades nativas dentro de blocos de componentes reutilizáveis.

A plataforma JADE

Jade é um *middleware* desenvolvido pela Tilab para o desenvolvimento de aplicações distribuídas multi-agentes baseada na arquitetura de comunicação *peer-to-peer*. Ambos, a inteligência, a iniciativa, a informação, o controle e os recursos podem ser totalmente distribuídos em terminais móveis ou em computadores em redes normais. O ambiente pode evoluir dinamicamente com os pontos, que em

JADE são chamados de agentes, que podem aparecer e desaparecer de acordo com a necessidade e requerimentos do ambiente da aplicação. A comunicação entre os pontos, não importando se a rede é *wireless* ou não, é sempre simétrica, ou seja cada ponto pode ser quem inicia ou quem responde a uma ação de comunicação. JADE é baseado em Java e é guiada pelos seguintes princípios:

- Interoperabilidade: JADE está em conformidade com os padrões da FIPA, como conseqüência os agentes JADE podem interoperar com outros agentes, contando que estes estejam em conformidade com o padrão.
- Uniformidade e portabilidade: JADE fornece um conjunto homogêneo de API's que são independentes da rede e da versão de Java. O JADE pode operar sobre o J2EE, J2SE e J2ME.
- Fácil de usar: A complexidade do *middleware* é escondida atrás de um simples e intuitivo conjunto de API's de fácil utilização.

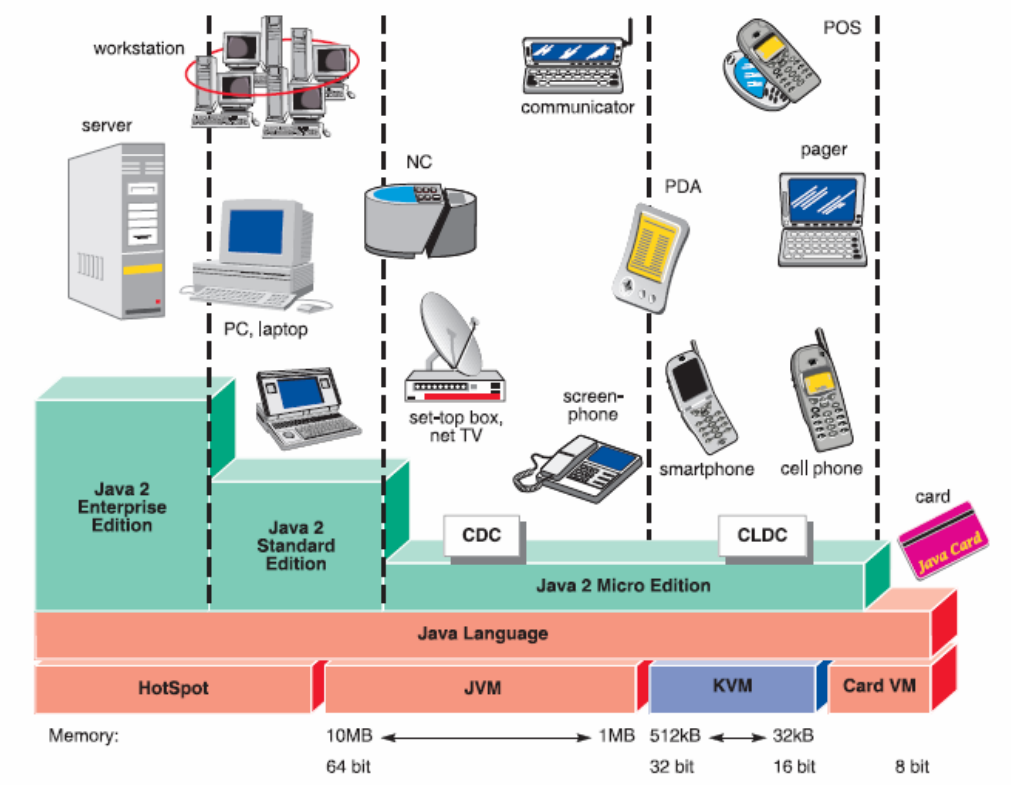

FIGURA 54 - Plataforma Java

A FIG. 54 apresenta as camadas da plataforma Java. Nela são apresentadas as versões J2EE para aplicações *enterprise*, J2SE para aplicações diversas e J2ME para dispositivos móveis e com recursos reduzidos/limitados.

Modelo da arquitetura

JADE inclui todas as bibliotecas necessárias para o desenvolvimento de agentes e um ambiente de execução que fornece os serviços básicos. O ambiente de execução deve ser iniciado antes da execução dos agentes. Cada instância do ambiente de execução é chamado de *container*, o conjunto de *container*es é chamado de plataforma que fornece uma camada homogenia que esconde os agentes da diversidade e complexidade das camadas inferiores (hardware, sistemas operacionais, etc...)

JADE é bastante versátil e é capaz de se integrar a sistemas com recursos limitados, como telefones móveis, e também com sistemas mais complexos, como plataformas J2EE e .NET.

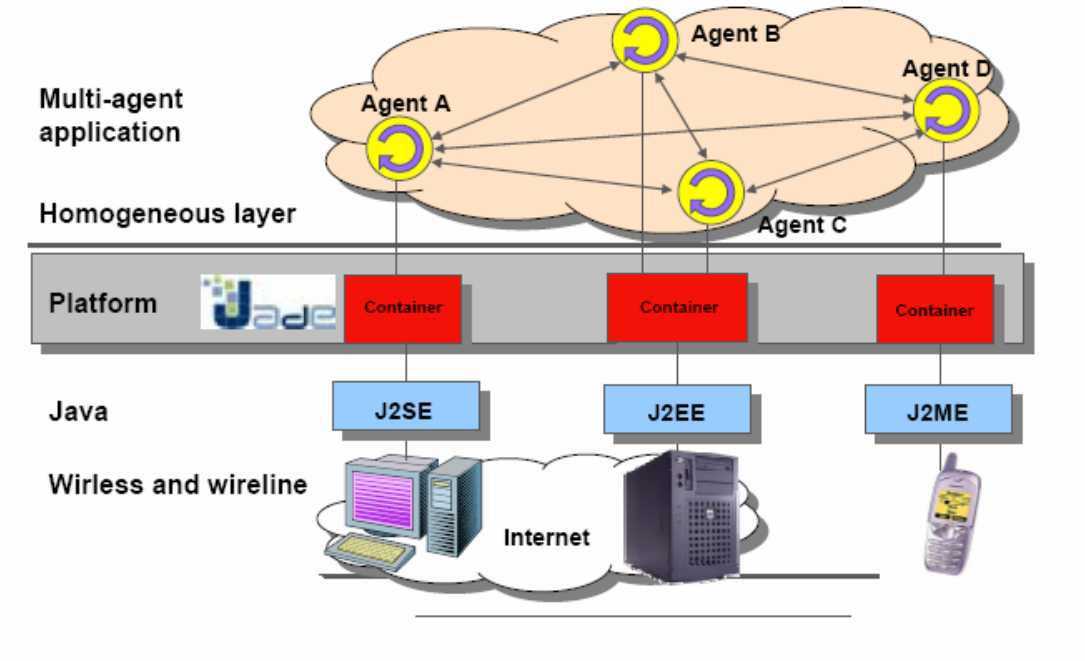

FIGURA 55 – Arquitetura Jade

Como demonstrado na FIG. 55, o modelo de arquitetura do Jade tem as seguintes características:

- Uma aplicação JADE é composta por uma coleção de componentes ativos chamados de agentes.
- Cada agente tem um nome único.
- Um agente é um ponto que pode comunicar-se com outros agentes de forma

bidirecional.

- Um agente vive em um container.

Modelo de comunicação

No framework JADE a comunicação é baseada na troca de mensagens assíncronas.

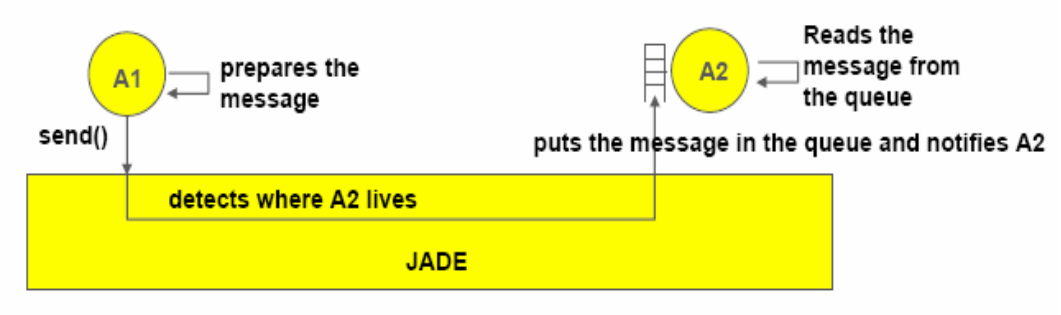

FIGURA 56 - Modelo de comunicação

Como demonstrado na FIG. 56, cada agente tem um conjunto de caixas de correio onde as mensagens para esses agentes são inseridas. Quando uma mensagem é colocada na caixa de correio o agente é notificado. Porém, é o agente que decide quando ele vai ler e responder a determinada mensagem.

Modelo funcional

Do ponto de vista funcional, JADE fornece os serviços básicos necessários para aplicações distribuídas em ambientes fixos ou móveis. JADE permite que cada agente possa dinamicamente descobrir outros agentes e comunicar com eles de acordo com o paradigma *peer-to-peer*. Do ponto de vista da aplicação, cada agente é identificado por um nome e a ele é fornecida uma série de serviços. Ele pode registrar e modificar seus serviços e/ou procurar por agentes para buscar serviços. Ele pode controlar seu ciclo de vida e se comunicar com outros pontos.

Os agentes se comunicam trocando mensagens assíncronas, o modelo de comunicação mais aceito para comunicação entre entidades heterogêneas que não conhecem nada um do outro. Para se comunicar, o agente apenas envia uma mensagem para o destinatário. Os agentes são identificados pelo nome (não existe

a necessidade de uma referência ao objeto de destino) e, como conseqüência, não existe dependência temporal entre os agentes em comunicação. O agente transmissor e o receptor não precisam estar disponíveis ao mesmo tempo. O receptor pode mesmo não existir (ainda) ou podem não ser conhecido diretamente pelo transmissor que pode apenas especificar uma propriedade (ex: agentes interessados em livros) como destino.

Apesar deste tipo de comunicação, a segurança é preservada, uma vez que para aplicações que exigem segurança, JADE fornece um mecanismo para autenticar e verificar direitos assinalados aos agentes. Quando necessário, uma aplicação pode verificar a identidade do transmissor e prevenir ações não permitidas.

Todas as mensagens enviadas por um agente devem se transportadas através de um envelope contendo apenas às informações necessárias à camada de transporte. O que permite entre outras coisas, criptografar a mensagem separadamente do envelope. A estrutura da mensagem está em conformidade com a linguagem ACL definida pela FIPA e inclui campos, como variáveis indicando o contexto da mensagem, a que se refere e um tempo de expiração que pode ser esperado antes da resposta da mensagem, a fim de suportar complexas interações e múltiplas conversações em paralelo.

Para suportar a implementação de conversações complexas, o JADE fornece um conjunto de *skeletons* para padrões de interações típicas na execução de tarefas específicas, assim como negociações, leilões e delegação de tarefas. Por usar esses *skeletons* (implementados como classes Java abstratas) programadores podem livrar-se da carga de ter que lidar com, sincronização, tempos de expiração, condições de erro e em geral sobre todos os aspectos não relacionados com a lógica da aplicação.

Para facilitar a criação e a manipulação do conteúdo das mensagens, JADE fornece suporte para conversões automáticas entre formatos na troca de conteúdos incluindo XML e RDF. Este suporte é integrado com algumas ferramentas de criação

de ontologias tais como Protégé[1], permitindo programadores criarem graficamente suas ontologias.

JADE não é transparente no que diz respeito a suporte a sistemas com inferência, se uma inferência é necessária a uma aplicação específica, ele permite que programadores reusem seus sistemas preferidos. Ele já foi testado e integrado com JESS e prolog.

Para aumentar a escalabilidade e para se adaptar a sistemas com recursos limitados, JADE fornece a possibilidade de execução de múltiplas tarefas paralelas em uma única *thread*. Muitas tarefas elementares como comunicação, podem ser combinadas para formarem estruturas de tarefas mais complexas como máquinas de estado finito concorrentes.

Em J2SE e ambientes Java, JADE suporta código móvel e execução de estados. Um agente pode parar a execução em um *host*, migrar para um *host* diferente (sem a necessidade do código deste agente já existir nesta máquina) e reiniciar a sua execução do ponto onde foi interrompido. Essa funcionalidade permite, por exemplo, a distribuição de carga computacional em tempo de execução movendo agentes de máquinas com mais carga para máquinas com menos carga, sem nenhum impacto na aplicação.

A plataforma também inclui serviços de nomes (assegurar que um agente possua apenas um nome) e serviço de páginas amarelas que podem ser distribuídos entre múltiplos *hosts*. Grafos podem ser criados para definir a estrutura de domínios do serviço de agentes.

Outra característica muito importante da plataforma é o conjunto de ferramentas gráficas que permitem depurar, monitorar e gerenciar fases do ciclo de vida da aplicação. Isto significa que existe a possibilidade de controle remoto dos agentes mesmo que eles já tenham sido instalados e executados. Conversações entre agentes podem ser emuladas, trocas de mensagens podem ser capturadas, tarefas

[1] Protege – editor de ontologias e framework de base de conhecimento código aberto e gratuito.

podem ser monitoradas e o ciclo de vida do agente pode ser controlado.

Jade em um ambiente móvel

O ambiente de execução do JADE pode ser executado em uma grande classe de equipamentos desde servidores até telefones celulares. Para endereçar de forma correta, memória e limitações de processamento de dispositivos móveis e características de redes sem fio, como por exemplo, largura de banda, latência, intermitência e variabilidade de endereçamento IP e, ao mesmo tempo, ser eficiente quando executado em ambiente de rede fixa, JADE pode ser configurado para se adaptar às características do ambiente no qual irá ser executado. A arquitetura JADE é modular de modo a permitir a ativação de certos módulos ao invés de outros e poder alcançar os objetivos de conectividade, memória e capacidade de processamento.

Um módulo chamado LEAP permite otimizar todos os mecanismos de comunicação quando trabalha com dispositivos com recursos limitados conectados através de redes *wireless*. Ativando este módulo, o JADE divide o *container* em duas partes o *front-end* e o *back-end*. Ao *back-end* é designado parte das funcionalidades do *container*, deixando o *front-end* mais leve em termos de processamento e memória. O *front-end* é capaz de detectar a perda de conexão com o *back-end* e restabelecer assim que possível. Ambos os lados possuem o mecanismo de *store-and-forward*, ou seja, a capacidade de armazenar mensagens que por ventura não possam ser enviadas imediatamente para serem enviadas assim que a conexão for restabelecida.

O QUADRO 4 a seguir descreve as características do *framework* Jade, tecnologias suportadas, e padrões técnicos definidos para suporte a ambientes móveis.

QUADRO 4 – Características JADE

JADE	
Características técnicas e funcionais	Distribuída, modular com comunicação ponto-a-ponto. Conformidade com os padrões FIPA. Gerenciamento do ciclo de vida do agente. Serviços de páginas brancas e páginas amarelas com a oportunidade de criação. Grafos em tempo de execução. Ferramentas gráficas para debugar, gerenciar e monitorar. Suporte para código de agente, execução de migração. Suporte para protocolos de comunicação complexos. Suporte para mensagens com conteúdo e gerenciamento incluindo XML e RDF. Suporte para integração com páginas JSP através de taglibs. Suporte camada de segurança. Protocolos de transporte selecionáveis em tempo de execução: RMI JICP (JADE protocolo proprietário), HTTP e IIOP.
Disponibilidade	Código aberto, Licença LGPL
Ambiente de rede	Já testado com Bluetooth, GPRS, W-LAN e a Internet.
Terminais	Todos os terminais que suportam Java MIDP1.0 já testado no Nokia 3650, Motorola Accompli008, Siemens, Compaq iPaq, Psion5MX, HP Joranda 560.
Linguagem	Java: J2EE, J2SE, J2ME CLDC/MIDP1.0

Plataforma JADE

O Jade define três agentes necessários para a administração da plataforma como mostrado na FIG. 57 a seguir:

- **Agent Management System (AMS)**
- **Agent Communication Channel (ACC)**
- **Directory Facilitator(DF)**

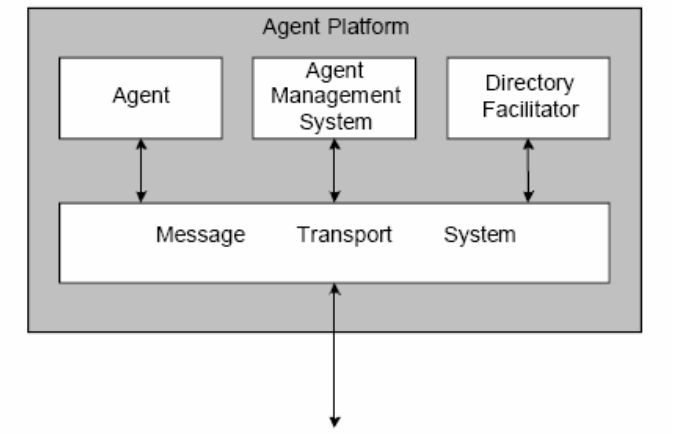

FIGURA 57 – Referência da arquitetura da plataforma de agentes FIPA

O *Agent Management System* (AMS) é o agente que fornece o controle supervisório sobre o acesso e o uso da plataforma. Somente um AMS pode existir em uma única plataforma. O AMS fornece serviço de *write-page* e de ciclo de vida, mantendo um diretório de identificadores de agentes (AID) e o estado do agente. Todo agente deve se registrar no AMS para ganhar um ID válido.

O *Directory Facilitator* (DS) é o agente que fornece um serviço padrão de *yellow-page* para a plataforma.
O *Message Transport System* também conhecido como *Agent Communication Channel* (ACC) é um componente de software que controla todas as trocas de mensagens da plataforma, incluindo mensagens de/para plataformas remotas.

O JADE está em total conformidade com este modelo de arquitetura e quando o Jade é executado, imediatamente o AMS e o DF são criados e o módulo ACC é configurado para permitir a comunicação. A plataforma de agentes pode ser dividida entre vários *hosts* como demonstrado na FIG. 58, mas somente uma aplicação Java, conseqüentemente uma JVM é executada em cada *host*. Cada JVM é um container básico de agentes que fornece um completo ambiente de execução para a ativação do agente e permite que vários agentes executem concorrentemente em um mesmo *host*. O *container* principal ou *front-end* é um *container* de agentes onde os agentes DF e AMS vivem e onde o RMI *registry*, utilizado internamente pelo JADE, é criado. Os outros *containeres* de agentes conectam-se no *container* principal e fornecem um completo ambiente para a execução de qualquer conjunto de agentes JADE.

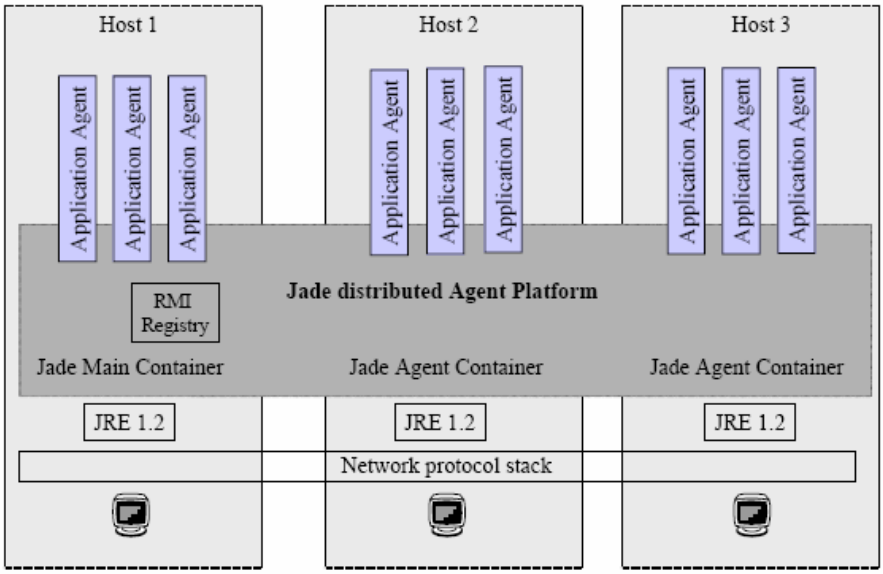

FIGURA 58 - Plataforma de agentes distribuída sobre vários containeres

O Agente em Jade

FIPA nada especifica sobre as estruturas internas dos agentes, fato que foi uma escolha explícita em conforme com a opinião de que a interoperabilidade pode ser garantida apenas especificando os comportamentos externos dos agentes (Ex: ACL, protocolos, linguagens de conteúdo e etc.) (FIPA, 2003). Para o Jade, um agente é processo autônomo e independente, que tem uma identidade e requer comunicação com outros agentes, seja ela por colaboração ou por competição, para executar totalmente seus objetivos (JADE, 2003).

Em outras palavras, pode-se concluir que Jade é absolutamente neutro no que diz respeito à definição de um agente. Ou seja, ele não limita ou especifica que tipo de agente pode ser construído pela plataforma.

Em termos mais técnicos um agente em Jade funciona como uma *"thread*" que emprega múltiplas tarefas ou comportamentos e conversações simultâneas. Esse agente Jade é implementado como uma classe Java chamada *Agent* que está dentro do pacote *jade.core*. Essa classe *Agent* atua como uma super classe para a criação de agentes de software definidos por usuários. Ela provê métodos para executar tarefas básicas de agentes, tais como:

- Passagens de mensagens usando objetos *ACLMessage (*seja direta ou *multicast*);
- Suporte completo ao ciclo de vida dos agentes, incluindo iniciar ou carregar, suspender e “matar” (*killing)* um agente;
- Escalonamento e execução de múltiplas atividades concorrentes;
- Interação simplificada com sistemas de agentes FIPA para a automação de tarefas comuns de agentes (registro no DF, etc).

JADE também provê suporte ao desenvolvimento de agentes móveis. Devido a sua própria arquitetura, os agentes podem migrar e clonar-se entre os containeres. Além disso, disponibiliza uma biblioteca (*jade.domain.mobility)* que contém a definição de ontologias para a mobilidade em JADE, vocabulário com uma lista de símbolos usados e todas as classes Java que implementam essas ontologias.

Do ponto de vista do programador, um agente JADE é simplesmente uma instância da classe *Agent*, na qual os programadores ou desenvolvedores deverão escrever seus próprios agentes como subclasses de *Agent*, adicionando comportamentos específicos de acordo com a necessidade e objetivo da aplicação, usando um conjunto básico de métodos e as capacidades herdadas que a classe *Agent* dispõe, Entre as capacidades da classe *Agent* estão os mecanismos básicos de interação com a plataforma de agentes (registro, configuração, gerenciamento remoto, etc).

Na FIG. 59 temos uma descrição de uma arquitetura interna de um agente genérico em JADE.

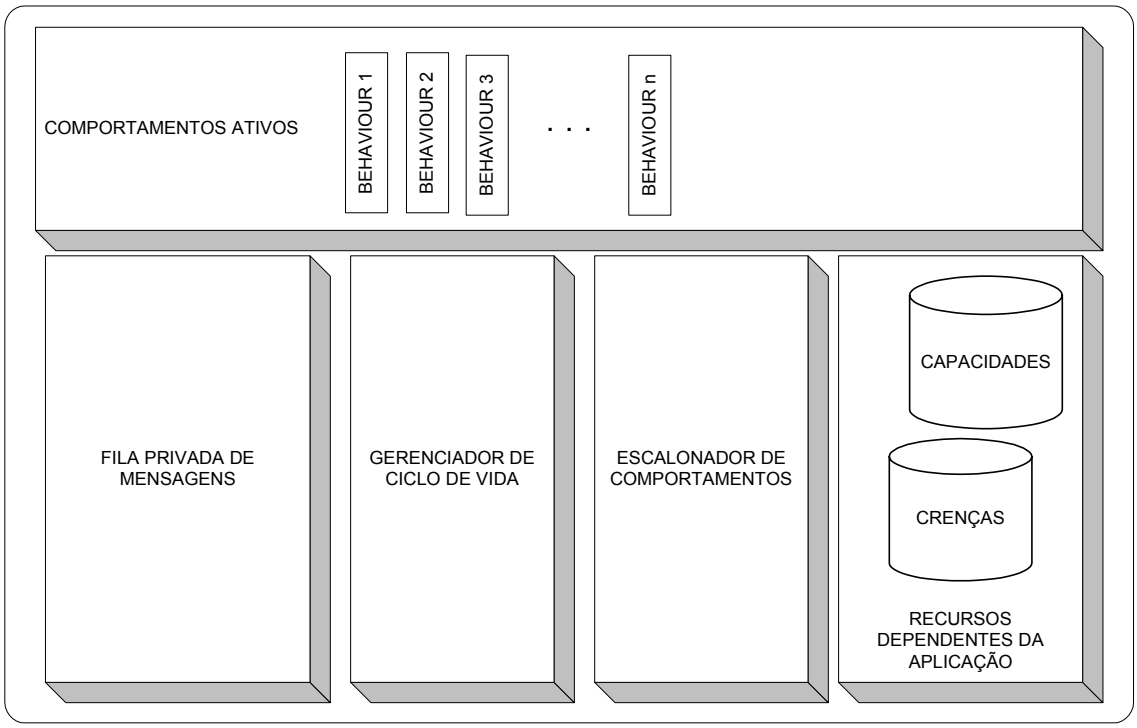

FIGURA 59 - Arquitetura interna de uma agente genérico em JADE
Fonte: JADE, 2003.

Como demonstrado na FIG. 59, na parte superior estão os comportamentos ativos do agente que representariam as ações/intenções que cada agente tem. O modelo computacional de um agente em JADE é multitarefa, onde tarefas (ou comportamentos) são executadas concorrentemente. Cada funcionalidade ou serviço provido por um agente deve ser implementado como um ou mais comportamentos. É importante frisar que JADE permite uma variedade de comportamentos em um mesmo agente.

Na parte inferior esquerda da FIG. 59, existe uma fila privativa de mensagens ACL. Todo agente JADE tem essa fila onde decide quando ler as mensagens recebidas e quais mensagens ler.

Ao centro da FIG. 59 estão o escalonador de comportamentos e o gerenciador do ciclo de vida. O primeiro é responsável por escalonar a ordem de execução dos comportamentos (por exemplo, seguirão alguma ordem de precedência ou não). O gerenciador de ciclo de vida é o controlador do estado atual do agente. Como uma característica importante do modelo de agentes JADE é autonomia, cada agente possui a autonomia implementada pela possibilidade de controlar completamente

sua *thread* de execução. O gerenciador de ciclo de vida é o meio que os agentes utilizam para determinar seu estado atual (ativo, suspenso, etc).

No lado direito da FIG. 59 estão os recursos de agentes dependentes da aplicação. Nesse local são armazenadas as crenças e capacidades que o agente adquiriu na execução da aplicação e elas ficam a cargo do desenvolvedor do agente.

Ciclo de Vida

Um agente JADE pode estar em um dos vários estados de acordo com ciclo de vida (*Agent Platform Life Cycle)* das especificações FIPA. Na FIG.60, temos a representação do ciclo de vida de um agente definido pela FIPA e seus possíveis estados.

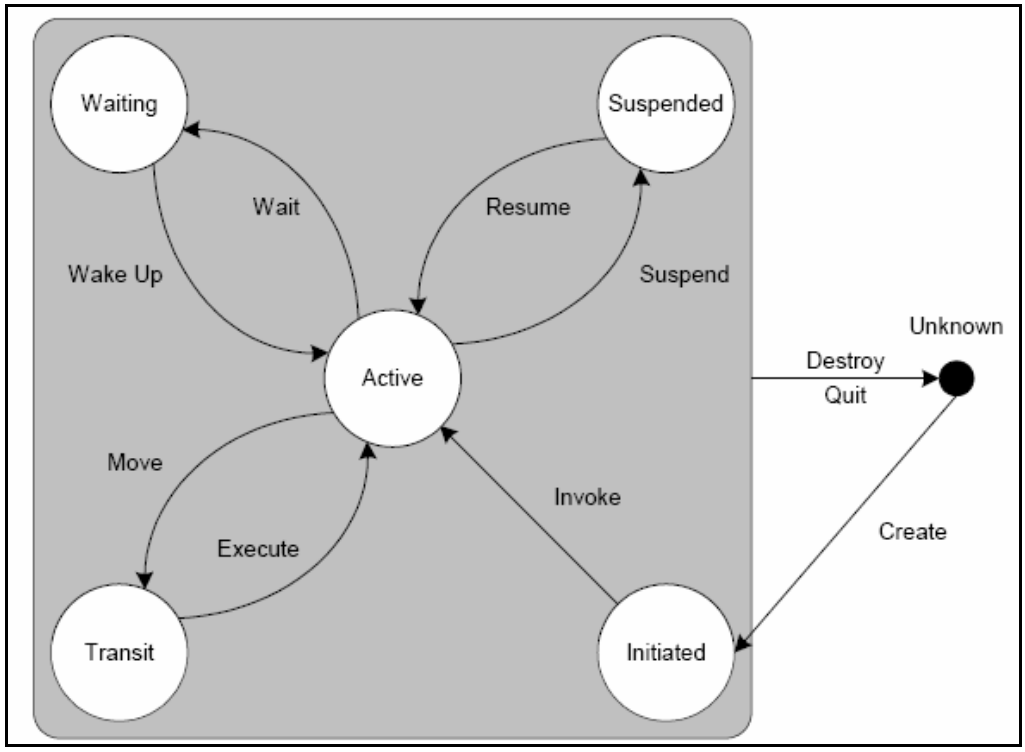

FIGURA 60 - Ciclo de Vida de um agente definido pela FIPA
Fonte: FIPA, 2003.

Esses estados são representados em JADE como constantes estáticas da classe *Agent.* São eles:

- **AP_INIATED:** o objeto da classe *Agent* foi instanciado, mas ainda não se registrou no AMS, não tem nome ou endereço e não pode se comunicar com

outros agentes.

- **AP_ACTIVE:** o objeto da classe *Agent* está registrado no AMS, tem nome formal e endereço e tem acesso a todas funcionalidades do JADE.
- **AP_SUSPENDED:** o objeto da classe *Agent* está no momento interrompido. Sua *thread* interna está suspensa e nenhum comportamento está sendo executado.
- **AP_WAITING:** o objeto da classe *Agent* está bloqueado, esperando por algo. Sua *thread* interna está "dormindo" (*sleeping)* sob um monitor Java e irá acordar quando alguma condição ocorrer (geralmente o recebimento de uma mensagem).
- **AP_DELETED:** o agente está definitivamente "deletado" ou encerrado. Sua *thread* interna terminou sua execução e o agente não está mais registrado no AMS.
- **AP_TRANSIT:** um agente móvel entra nesse estado enquanto tiver migrando para uma nova localização. O sistema continua a armazenar mensagens em um *buffer* para que sejam enviadas para essa nova localização.
- **AP_COPY:** esse estado é usado internamente pelo JADE para agentes que foram clonados.
- **AP_GONE:** esse estado é usado internamente pelo JADE, quando um agente móvel migrou para uma outra localização e se encontra em um estado estável.

A troca de estados é feita por meio de métodos como podemos visualizar na FIG.60 (reiniciar, esperar, suspender, *etc).* Existem métodos públicos disponibilizados pela classe *Agent* de JADE que fazem as transições entre esses estados acima citados. São eles:

- *public void doActivate ()* – Faz a transição do estado *suspenso* para o *ativo* ou *em espera* (estado que agente estava quando *doSuspend ()* foi chamado). Esse método é chamado da plataforma de agentes e reinicia a execução do agente. Chamar *doActivate ()* em um agente que não esteja suspenso não tem efeito.
- *public void doSuspend ()* – Faz a transição do estado *ativo* ou *em espera* para o estado de *suspenso.* O estado original do agente é salvo e será restaurado pela chamada ao método *doActivate ()*. Esse método pode ser chamado pela plataforma de agentes ou pelo próprio agente que interrompe todas suas atividades. Mensagens que chegam para um agente suspenso são armazenadas pela plataforma e são entregues assim que o agente reinicie suas atividades. Chamar *doSuspend ()* de um agente suspenso não tem efeito.

- *public void doWait ()* – Faz a transição do estado *ativo* para o estado *em espera*. Esse método pode ser chamado pela plataforma de agentes ou pelo próprio agente que causa um bloqueio dele mesmo, interrompendo todas suas atividades até que alguns eventos ocorram. Um agente em estado de espera sai desse estado quando o método *doWake ()* é chamado ou quando uma mensagem chega. Chamada ao método *doWait ()* em agentes suspensos ou em espera não causa efeito nenhum.
- *public void doWake ()* – Faz a transição do estado *em espera* para o estado *ativo.* Esse método é chamado pela plataforma de agentes e reinicia a execução do agente. Chamada ao método *doWake ()* em agente que não está em espera não tem efeito.
- *public void doDelete ()* – Faz a transição dos estados *ativo, suspenso* ou *em espera* para o estado "*deletado".* Esse ato causa a destruição do agente. Esse método pode ser chamado pela plataforma de agentes ou pelo próprio agente. Chamar *doDelete ()* em um agente já destruído não causa efeito nenhum.
- *public void doMove (Location* destino*)* – Faz a transição do estado *ativo* para o estado *em trânsito.* Esse método tem o objetivo de dar suporte à mobilidade de agentes e é chamado pela plataforma de agentes ou pelo próprio agente para iniciar o processo de migração.
- *public void doClone (Location* destino, *java.lang.String* novoNome) – Faz a transição do estado *ativo* para o estado "*copy"*. Esse método tem o objetivo de dar suporte à mobilidade de agentes e é chamado pela plataforma de agentes ou pelo próprio agente para iniciar o processo de clonagem.

Vale ressaltar que é permitido ao agente executar seus comportamentos/*behaviours* apenas quando estiver no estado *ativo.* Outro fato que também merece ser ressaltado é que se em qualquer dos comportamentos de um agente for chamado o método *doWait ()*, o agente como um todo e todas suas atividades serão bloqueadas, não só o comportamento que chamou o método. Ao invés disso, o método *block ()* da classe *Behaviour* evitaria essa bloqueio total, uma vez que permite a suspensão de apenas um comportamento, e além de se desbloquear caso uma mensagem seja recebida.

Principais Métodos da Classe Agent

Segue abaixo alguns dos principais métodos da classe *Agent*:

- ***protected void setup ()*** – Esse método protegido é indicado para códigos inicializadores da aplicação. Os desenvolvedores podem sobrescrever esse método fornecendo comportamentos necessários ao agente. Quando esse método é chamado, o agente já está registrado no AMS e está apto a enviar e receber mensagens. Porém, o modelo de execução do agente ainda é seqüencial e nenhum escalonamento de comportamento foi efetivado ainda. Logo, é essencial adicionar pelo menos um comportamento para o agente neste método, além de tarefas comuns de inicialização como registro no DF, para torná-lo apto a fazer alguma coisa.
- ***public void addBehaviour (Behaviour* comportamento) -** Esse método adiciona um novo comportamento ao agente. Ele será executado concorrentemente com outros comportamentos. Geralmente é usado no método *setup ()* para disparar algum comportamento inicial, porém pode ser usado também para gerar comportamentos dinamicamente. O método *removeBehaviour ()* faz justamente o inverso: remove determinado comportamento do agente.
- ***public final void send (ACLMessage* mensagem)** - Envia uma mensagem ACL para outro agente. Esse agente destino é especificado no campo *receiver* da mensagem no qual um ou mais agentes podem ser definidos como receptores.
- ***public final ACLMessage receive ()*** – Recebe uma mensagem ACL da fila de mensagens do agente. Este método é não-bloqueante e retorna a primeira mensagem da fila caso haja alguma. Retorna *null* caso não haja mensagens.

Existem ainda métodos bastante particulares e úteis que podem ser encontrados na documentação do JADE. Os acima citados são os mais básicos, primordiais para qualquer sistema multiagente, por menor que seja.

Comportamentos/Behaviours

As ações ou comportamentos que um agente desempenha dentro de um sistema multiagente são fundamentais para que o objetivo final da aplicação seja alcançado.

No JADE, o desenvolvedor que pretender implementar um agente deve obrigatoriamente herdar a classe *Agent* e implementar tarefas específicas para o agente escrevendo uma ou mais subclasses *Behaviour,* instanciando-as e adicionando-as ao agente.

JADE contém comportamentos específicos para a maioria das tarefas comuns na programação de agentes, tais como envio e recebimento de mensagens e até construção de tarefas mais complexas como demonstrado na FIG. 61.

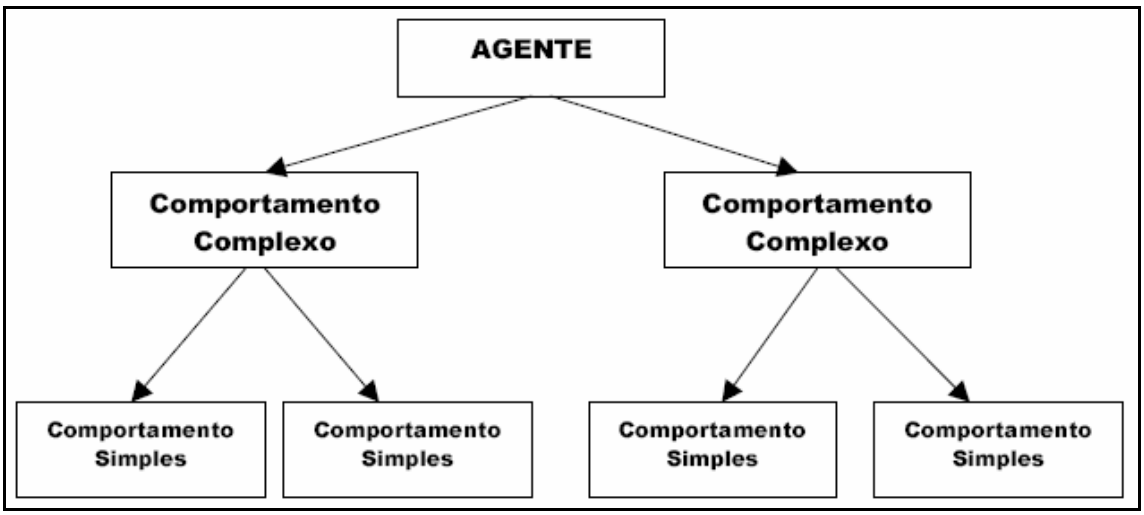

FIGURA 61 - Comportamentos compostos por sub-comportamentos em JADE

A estrutura de comportamentos do JADE dá-se através de um escalonador, interno à super classe *Agent*, que gerencia automaticamente o escalonamento desses comportamentos (*behaviours).* Do ponto de vista de programação concorrente um agente é um objeto ativo com uma *thread* de controle interna. JADE usa o modelo de uma *thread* por agente ao invés de uma *thread* por tarefa ou conversação, com o objetivo de manter um número pequeno de *threads* necessárias para executar a plataforma de agentes. Caso criasse uma nova *thread* para cada comportamento, o desempenho do sistema poderia ser comprometido.

O escalonamento desses comportamentos é feito por um escalonador, implementado na super classe *Agent* e abstrato para o programador, que realiza a política de *round-robin*[1] não preemptivo*,* em todos os comportamentos disponíveis na fila de pronto (BELLIFEMINE 2003). No escalonamento *round-robin* cada

comportamento adicionado ao agente é colocado em uma fila. Quando esse comportamento passa para o estado de execução, ou seja, o processo passa para a CPU, existe um tempo limite para a sua utilização. Quando esse tempo denominado *time-slice* ou quantum expira, sem que antes o processador seja liberado pelo processo, ele volta ao estado de pronto, dando a vez para outro processo. Isso permite a execução da classe de um comportamento completo. Ou seja, se a tarefa ou comportamento cede sem o controle ter sido completado, ela será re-escalonada no próximo ciclo. Ela poderá não ser re-escalonada caso esteja bloqueada (um *behaviour* pode se bloquear no instante em que espera mensagens para evitar desperdício de tempo de CPU e evitar uma “espera ocupada”).

Em outras palavras, um comportamento é executado por vez por um determinado tempo (*quantum*). Quando acaba esse tempo, o comportamento pode voltar para o fim da fila, caso não tenha acabado de executar sua tarefa. O fato de ser não preemptivo significa dizer que não existe prioridade entre os comportamentos, ou seja, cada comportamento adicionado deve ir para o fim da fila e todos têm a mesma fatia de tempo.

Classe Behaviour

Behaviour é uma classe abstrata do JADE disponível no pacote *jade.core.behaviours.* Uma classe abstrata é uma classe que possui alguns métodos implementados e outros não, e não pode ser instanciada diretamente. Ela tem como finalidade prover a estrutura de comportamentos para os agentes JADE implementarem.

O principal método da classe *Behaviour* é o *action()*. É nele que serão implementadas as tarefas ou ações que este comportamento irá tomar. Outro importante método é o *done()*, que é usado para informar ao escalonar do agente se o comportamento foi terminado ou não. Ele retorna *true* caso o comportamento tenha terminado, quando é removido da fila de comportamentos, e retorna *false* quando o comportamento não tenha terminado obrigando o método *action()* a ser executado novamente.

Outros importantes atributos e métodos da classe *Behaviour* são:

- *protected Agent myAgent* – Retorna o agente a qual esse comportamento pertence.
- *void block ()* – Bloqueia esse comportamento. Pode ser passado como parâmetro um tempo de bloqueio em milisegundos: *void block (long* tempo).
- *void restart ()* – Reinicia um comportamento bloqueado.
- *void reset ()* – Restaura o estado inicial do comportamento.
- *boolean isRunnable ()* – Retorna se o comportamento está bloqueado ou não.

Classes Derivadas do Behaviour

Como já mencionado anteriormente, JADE possui várias classes de comportamentos prontas para uso pelo desenvolvedor, adequando-as de acordo com a necessidade específica do agente. Hierarquicamente, elas estão estruturadas da seguinte forma:

- Classe *jade.core.behaviours.**Behaviour***
- Classe *jade.core.behaviours.**CompositeBehaviour***
- Classe *jade.core.behaviours.**FSMBehaviour***
- Classe *jade.core.behaviours.**ParallelBehaviour***
- Classe *jade.core.behaviours.**SequentialBehaviour***
- Classe *jade.core.behaviours.**ReceiverBehaviour***
- Classe *jade.core.behaviours.**SimpleBehaviour***
- Classe *jade.core.behaviours.**CyclicBehaviour***
- Classe *jade.core.behaviours.**OneShotBehaviour***
- Classe *jade.core.behaviours.**SenderBehaviour***
- Classe *jade.core.behaviours.**TickerBehaviour***
- Classe *jade.core.behaviours.**WakerBehaviour***

Como classes mais gerais temos a *CompositeBehaviour, ReceiverBehaviour e a SimpleBehaviour.* Sendo que, algumas delas são subdividas em outras classes para tornar mais específicas suas finalidades.

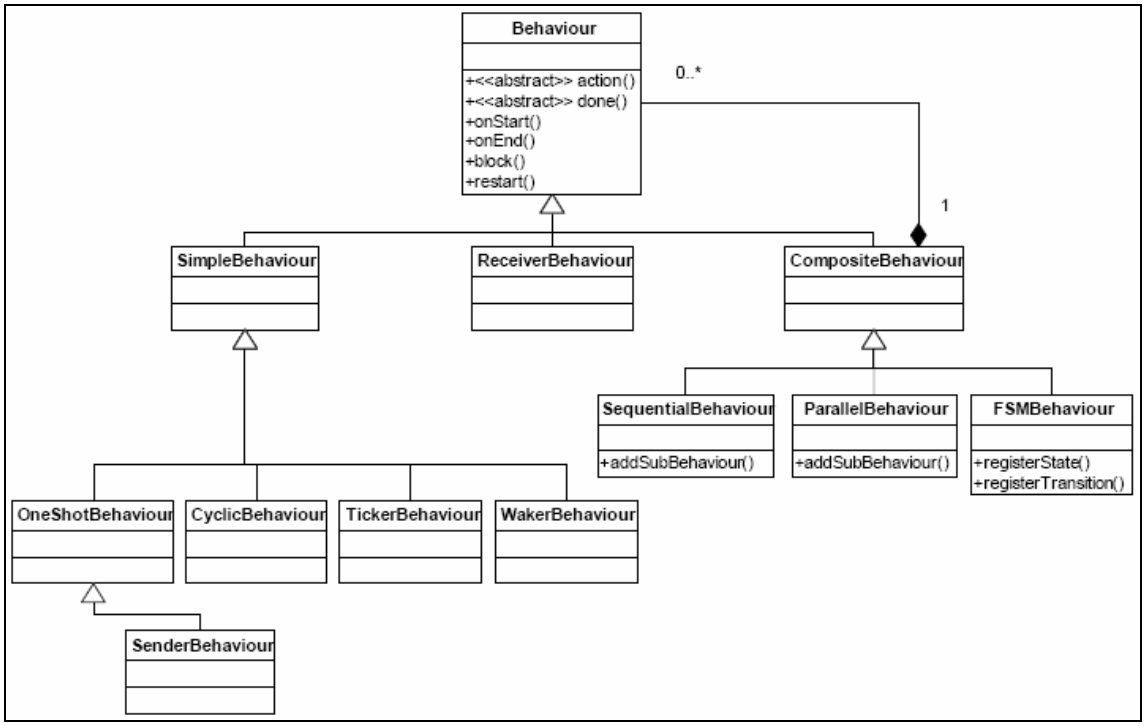

FIGURA 62 - Hierarquia das classes derivadas de *Behaviour* em UML

A FIG.62 apresenta um diagrama de classes em UML que mostra a hierarquia de classes que derivam da classe abstrata *Behaviour.* As classes *CompositeBehaviour* e *SimpleBehavior* são mostradas como classes derivadas da *Behaviour* (a classe *ReceiverBehaviour* foi omitida no diagrama por não ter classes filhas). Também é possível visualizar as classes que herdam de *CompositeBehaviour* e *SimpleBehaviour* e um pequeno resumo do tipo de tarefas que elas modelam.

A classe *ReceiverBehaviour* implementa um comportamento para recebimento de mensagens ACL. Ela encapsula o método *receive ()* como uma operação atômica11 e o comportamento é encerrado quando uma mensagem ACL é recebida. Possui o método *getMessage ()* que permite receber a mensagem.

Já a classe *CompositeBehaviour* é uma classe abstrata para comportamentos, como o próprio nome diz, compostos por diferentes partes. Ela controla internamente comportamentos filhos dividindo seu “quantum” de execução de acordo com alguma política de escalonamento. Possui três classes herdadas disponibilizadas pelo JADE. São elas:

- *jade.core.behaviours.**FSMBehaviour*** – Comportamento composto baseado no

escalonamento por máquina de estados finitos (***F**inite **S**tate **M**achine*). O *FSMBehaviour* executa cada comportamento filho de acordo com uma máquina de estados finitos definido pelo usuário. Mais especificamente cada comportamento-filho representa um estado na máquina de estados finitos. Ela fornece métodos para registrar os estados (sub-comportamentos) e as transições que definem como dar-se-á o escalonamento desses comportamentos-filho. Os passos básicos para se definir um *FSMBehaviour* são:

1) Registrar um comportamento único como estado inicial através do método *registerFirstState* passando como parâmetros o *Behaviour* e o nome do estado.

2) Registrar um ou mais comportamentos como estados finais através do método *registerLastState*

3) Registrar um ou mais comportamentos como estados intermediários através do método *registerState*

4) Para cada estado, registrar as transições dele para os outros estados através do método *registerTransition.*

- *jade.core.behaviours.**ParallelBehaviour*** - Comportamento composto com escalonamento concorrente dos comportamentos filhos. *ParallelBehaviour* executa seus comportamentos filhos concorrentemente e finaliza quando uma condição particular em seus sub-comportamentos é atingida. Por exemplo, quando um número "X" de comportamentos-filho terminarem ou um comportamento-filho qualquer terminar ou quando todos os comportamentos-filho terminarem. Essas condições são definidas no construtor da classe, passando como parâmetro as constantes ***WHEN_ALL***, quando for todos, ***WHEN_ANY***, quando for algum, ou um valor inteiro que especifica o número de comportamentos filhos terminados que são necessários para finalizar o *ParallelBehaviour.* O método para adicionar comportamentos-filho é o *addSubBehaviour.*
- *jade.core.behaviours.**SequentialBehaviour*** - Comportamento composto com escalonamento seqüencial de comportamentos-filho. Ou seja, os comportamentos-filho são executados numa ordem seqüencial e só é encerrado quando o último comportamento-filho for finalizado. O método para adicionar comportamentos-filho é o *addSubBehaviour.*

Por último, temos a classe abstrata *SimpleBehaviour.* Ela é um comportamento

atômico. Isso porque modela comportamentos que são feitos para serem únicos, monolíticos e que não podem ser interrompidos. Possui quatro classes herdadas disponibilizadas pelo JADE. São elas:

- *jade.core.behaviours.**CyclicBehaviour*** – Comportamento atômico que deve ser executado sempre. Essa classe abstrata pode ser herdada para criação de comportamentos que se manterão executando continuamente. No caso, o método done() herdado da super classe Behaviour, sempre retorna false, o que faz com que o comportamento se repita como se estivesse em um "*loop*" infinito.
- *jade.core.behaviours.**TickerBehaviour*** – Comportamento executando periodicamente tarefas específicas. Ou seja, é uma classe abstrata que implementa um comportamento que executa periodicamente um pedaço de código definido pelo usuário em uma determinada freqüência de repetições. No caso, o desenvolvedor redefine o método *onTick* () e inclui o pedaço de código que deve ser executado periodicamente. O período de "ticks" é definido no construtor da classe em milisegundos.
- *jade.core.behaviours.**WakerBehaviour*** **-** Comportamento que é executado depois de determinado tempo ser expirado. Em outras palavras, é uma classe abstrata onde uma tarefa "*OneShot*" é executada apenas depois que determinado tempo for expirado. No caso, a tarefa é inserida no método *handleElapsedTimeout* () o qual é chamado sempre que o intervalo de tempo é transcorrido.
- *jade.core.behaviours.**OneShotBehaviour*** – Comportamento atômico que é executado uma única vez. Essa classe abstrata pode ser herdada para a criação de comportamentos para tarefas que precisam ser executadas apenas uma única vez. Tem como classe filha a classe *SenderBehaviour.*
- *jade.core.behaviours.**SenderBehaviour*** – é um comportamento do tipo *OneShotBehaviour* para envio de mensagens ACL. Essa classe encapsula o método *send*() como uma operação atômica. No caso, esse comportamento envia determinada mensagem ACL e se finaliza. A mensagem ACL é passada no construtor da classe.

Particularidades

Devido ao modelo não-preemptivo de escolha de comportamentos dos agentes, os programadores de agentes devem evitar uso de "*loops*" infinitos e até executar atividades muito longas dentro do método *action*() dos *Behaviours*. Isso porque enquanto o método *action* de algum comportamento estiver sendo executado, nenhum outro comportamento desse mesmo agente poderá ser executado até que o fim do método *action* ocorra.

Além disso, como não há armazenamento em pilha, o método *action* () sempre será executado do início. Não sendo possível, por exemplo, interromper um comportamento no meio de um *action*(), passar o controle para outros comportamentos e voltar para o comportamento original no local onde tinha sido interrompido (BELLIFEMINE, 2003).

Troca de Mensagens

Toda a troca de mensagens realizada no JADE é feita através de métodos próprios e com o uso de instâncias da classe *ACLMessage.* Essa classe possui um conjunto de atributos que estão em conformidade com as especificações da FIPA, implementando a linguagem FIPA-ACL. Assim, um agente que pretenda enviar uma mensagem deve instanciar um objeto da classe *ACLMessage*, preenchê-los com as informações necessárias e chamar o método *send*() da classe *Agent*. Caso for receber mensagens, o método *receive()* ou *blockingReceive* () da classe *Agent* deve ser chamado.
Outro meio de enviar ou receber mensagens no JADE é através do uso das classes de comportamentos *SenderBehaviour* e *ReceiveBehaviour*. Fato que torna possível que as trocas de mensagens possam ser escalonadas como atividades independentes de um agente.

A classe *ACLMessage* implementa, como o próprio nome diz, uma mensagem na linguagem FIPA-ACL de acordo com as especificações da FIPA. Todos seus atributos podem ser acessados via métodos *set* e *get* (por exemplo, *getContent* e *setContent)*. Além disso, a *ACLMessage* define um conjunto de constantes

(ACCEPT_PROPOSAL, AGREE, CANCEL, CFP,CONFIRM**,** etc) que são usadas para se referir às performativas da FIPA. Essas constantes são referidas no construtor da classe com o objetivo de definir o tipo de mensagem.

Os métodos da classe *ACLMessage* são simples e abrangentes. Sendo os principais:

- *public void* ***addReceiver (AID*** **idAgente)** – Adiciona o AID de um agente como receptor ou destinatário da mensagem. Em outras palavras, determina quem receberá a mensagem;
- *public void* ***removeReceiver*** **(AID idAgente)** – Remove o AID do agente da lista de receptores;
- *public ACLMessage* ***createReply*** **()**- Cria uma nova mensagem ACL de resposta à determinada mensagem. Assim, o programador só necessita definir o ato comunicativo (*communicate-act)* e o conteúdo da mensagem;
- *public static java.lang.String[]* ***getAllPerformativeNames*** **()** - Retorna um array de *strings* com todas as performativas;
- *public Iterator* ***getAllReceiver*** **()** – retorna um *iterator* contendo todos os AIDs dos agentes receptores da mensagem;
- *public java.lang.String* ***getContent*** **()** - Retorna uma *string* contendo o conteúdo da mensagem;
- *public void* ***setContent (java.lang.String*** **conteúdo)** – Define o conteúdo da mensagem a ser enviada;
- *public void* ***setOntology (java.lang.String*** **ontologia)** – Define a ontologia da mensagem.

No padrão FIPA todas as trocas de mensagens entre agentes devem se basear em *strings* ou textos e é o que ocorre em JADE no qual os métodos *getContent* e *setContent* trabalham com *strings*. Porém, JADE também permite que não só *strings* sejam transmitidas por mensagens, mas também outros objetos Java. O método *setContentObject* da classe *AclMessage* permite definir como conteúdo de uma mensagem um objeto Java do tipo *java.io.Serializable*. Já o método *getContentObject* retorna o conteúdo definido pelo método anterior.

Nas figuras 63 e 64 estão exemplos do uso desses dois métodos.

```
ACLMessage msg = new ACLMessage (ACLMessage.INFORM) ;
Date data = new Date () ;
try{
  msg.setContentObject (data);
} catch (IOException e) {}
```

FIGURA 63 - Exemplo do uso de setContentObject

```
ACLMessage msg = receive ();
try {
  Date data =  (Date) msg.getContentObject();
}
catch (UnreadableException e) {}
```

FIGURA 64 - Exemplo do uso de getContentObject

Entretanto, o próprio JADE não incentiva o uso desses métodos por não estarem em conformidade às especificações da FIPA.

Interoperabilidade

Conforme já foi mencionado, a arquitetura de JADE é baseada em *Java Virtual Machine* em diferentes *hosts*. Para manter as comunicações entre diferentes ambientes JADE utiliza-se do protocolo IIOP[2] (*Internet Inter-ORB Protocol*). Para a comunicação de agentes na mesma plataforma JADE são utilizados outros meios (RMI ou via eventos). Ou seja, percebe-se que JADE possui um comportamento variado em relação a forma com que realiza sua comunicação, onde o mecanismo é selecionado de acordo com a situação do ambiente. Isso ocorre com um objetivo único de atingir o menor custo possível nas trocas de mensagens. No caso, quando

[2] Introduzido na segunda especificação de CORBA (*Common Object Request Broker Architecture*), o IIOP permitiu que CORBA se tornasse uma solução definitiva para interoperabilidade entre objetos que não estão presos a uma plataforma ou padrão específico. Mais informações sobre CORBA e IIOP podem ser encontradas no livro **Distributed Systems – Concepts and Design** de Tim Kindberg, George Coulouris & Jean Dollimore, 3ª edição, editora Addison Wesley Pub 2001.

um agente JADE envia uma mensagem, as seguintes situações são possíveis:

- se o agente receptor reside no mesmo container de agentes, então o objeto Java representante da mensagem ACL é passado para o receptor via eventos. Sem invocações remotas;
- se o agente receptor reside na mesma plataforma JADE, mas em containeres diferentes, a mensagem ACL é enviada usando Java RMI;
- se o agente receptor reside em uma diferente plataforma de agentes, o protocolo IIOP é usado de acordo com o padrão FIPA.

Isso é ilustrado na FIG. 65. Essa figura mostra as possibilidades que JADE tem para realizar suas trocas de mensagens. O meio mais eficiente é escolhido de acordo com a localização do agente receptor da mensagem conforme foi explicado anteriormente. Note que na parte inferior da figura existem as formas possíveis de comunicação. Esse *cache* local, localizado no ACC (parte central da Figura), armazena as referências dos objetos dos outros *containeres*. Essas referências são adicionadas ao *cache* sempre que uma mensagem é enviada.

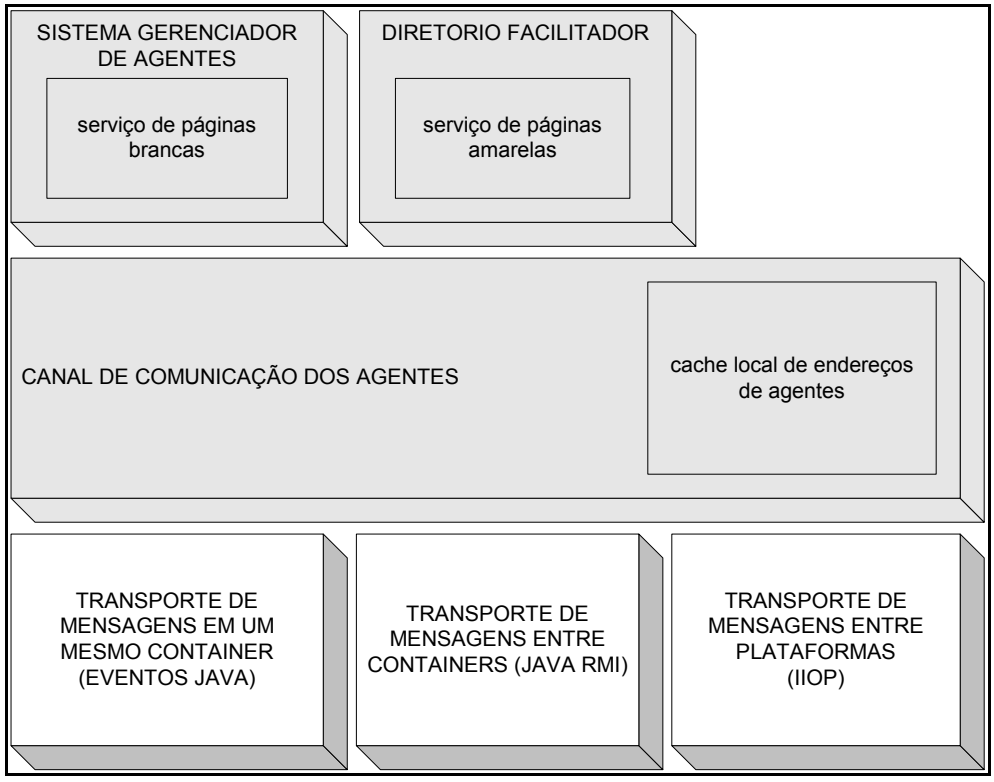

FIGURA 65 - Interoperabilidade no JADE

Bibliotecas de Pacotes do Jade

JADE oferece inúmeros pacotes de classes que visam auxiliar tanto na implementação quanto na monitoração de sistemas multiagentes. Os principais pacotes que compõem o JADE são:

- ***Jade.core*** – Esse pacote implementa o "kernel" do sistema. Ou seja, é o "coração" do *Jade* que inclui classes importantes como a classe *Agent e AID.* Ele fornece uma passagem de mensagens simples com protocolos de mensagens variados e um ambiente de execução *multithreading* para softwares de agentes.
- ***Jade.core.behaviours*** – Esse sub-pacote do pacote *jade.core* contém a classe *Behaviour.* Essa classe, como próprio nome já diz, implementa os comportamentos básicos de um agente. Ou seja, suas tarefas, intenções ou objetivos. Esses comportamentos foram abordados no tópico *6.14 Comportamentos/Behaviours.*
- ***Jade.lang.acl*** – Esse pacote contém todo o suporte necessário para a implementação de uma mensagem no formato FIPA-ACL *,* incluindo a classe *ACLMessage*, o analisador (*parser*), o codificador (*encoder*), e uma classe para representar *templates* (padrões) de mensagens ACL. Esse assunto foi abordado no tópico 6.18 (*Troca de Mensagens)..*
- ***Jade.content*** – Esse pacote contém, como conteúdo, classes para suportar as ontologias definidas pelo usuário e as linguagens de conteúdo.
- ***Jade.domain*** - Esse pacote e seus sub-pacotes contêm todas as classes Java que representam as entidades gerenciadoras de agentes (*Agent Management)* definidas pela FIPA. Tais como:
    - *AMS Agent:* gerencia o ciclo de vida dos agentes no ambiente e guarda informações sobre eles.
    - *DF Agent:* mantêm as páginas amarelas (*yellowpages)* de informações sobre os agentes.
- ***Jade.gui*** – Esse pacote contém componentes que podem ser usados para construir interfaces gráficas Swing13 genéricas, para mostrar e editar propriedades relacionadas aos agentes JADE como *Agent Identifiers (AIDs),* descrições de agentes, *ACLMessages, etc.*
- ***Jade.mtp –*** Esse pacote contém todas as interfaces Java que todo protocolo de transporte de mensagens (*Message Transport Protocol* - MTP) deve implementar

para que fique totalmente integrada com o *framework* do JADE como também a implementação desses protocolos.

- ***Jade.proto*** – Contém classes para modelar protocolos de interação padronizados (ex: *fipa-request, fipa-query, fipa-contract-net, fipa-subscribe,* e outros definidos pela FIPA), como também classes abstratas para que os programadores desenvolvam seus próprios protocolos. Para cada protocolo de interação, de acordo com as especificações FIPA, dois papéis (*roles*) podem ser desempenhados para cada agente:
  - *Initiator:* O agente contacta um ou mais agentes para iniciar uma conversação de acordo com um específico protocolo de interação.
  - *Responder:* Em resposta a uma mensagem recebida de outro agente, o agente continua em uma nova conversação seguindo um protocolo específico.
- ***Jade.wrapper*** - Juntamente com o *jade.core.Profile* e o *jade.core.Runtime* esse pacote contém todo o suporte necessário para que aplicações Java externas usem JADE como um tipo de biblioteca e executem o "*run-time*" do JADE, agentes JADE e os *containeres* de agentes pela própria aplicação.
- ***Jade.util*** – Esse pacote contém classes úteis, e em particular classes para tratamento de propriedades e o sub-pacote *leap* que é uma substituição para o *framework* Java que não foi suportada pelo J2ME14.
- ***Jade.tools*** – Pacote que possui ferramentas úteis para a administração do desenvolvimento dos agentes JADE.

Outras Classes de Jade

JADE possui uma grande variedade de classes prontas para serem usadas pelos desenvolvedores de sistemas multiagentes. Por ter código aberto, inúmeras novas classes *addons* para JADE estão sendo desenvolvidas e disponibilizadas gratuitamente. Sendo assim impossível falar de todas as classes que JADE possui, porém descreve-se apenas algumas delas visando demonstrar outras funcionalidades que essa plataforma possui.

Classe AID

Localizada no pacote *jade.core.AID*, a classe AID representa um identificador do agente JADE. Esse identificador é único e é utilizado por tabelas internas do JADE para armazenar nomes e endereços dos agentes e também como referência para que os outros agentes possam trocar mensagens. Geralmente composto por um nome *string* e um endereço composto por uma porta (a *default* é 1099) e o *host* da máquina em que o agente reside.

Classe MessageTemplate

A classe *MessageTemplate* permite construir padrões para as mensagens serem compatíveis com os padrões definidos. Usando os métodos dessa classe, o programador pode criar um ou mais padrões para cada atributo da mensagem ACL. Além disso, padrões básicos podem ser combinados com operadores AND, OR ou NOT com o objetivo de construir regras de validações mais complexas.

Classe DFService

Essa classe provê um conjunto de métodos estáticos para a comunicação com um serviço de DF complacente com as especificações da FIPA. Inclui métodos para registrar, cancelar registros, alterar e pesquisar em um DF. Está presente no pacote *jade.domain.*

SocketProxyAgent

Essa classe provê um agente JADE que pode ser usado para comunicação com clientes remotos via *sockets*. É uma ferramenta útil que pode por exemplo prover interações com Java *Applets* em *browsers* ou tratar conexões com *firewalls*.

Pacote jade.gui

O pacote *jade.gui* possui inúmeras classes que podem ser usadas para a construção de aplicações Java Swing para agentes. As classes principais são:

- ***jade.gui.GuiAgent*** - Essa classe abstrata é uma extensão da classe *jade.core.Agent*, tem como objetivo gerenciar as *threads* de agentes ativos em aplicações que utilizem uma GUI. Isso é necessário porque quando o programa instancia uma GUI, o Java inicia uma nova *thread* que, logicamente, é diferente da *thread* agente. Essa fica ativa por causa da *thread* agente uma vez que os comportamentos do agente geralmente vão depender de ações do usuário (ex: pressionando um botão, movendo mouse, etc). Portanto, um mecanismo apropriado para gerenciar a interação entre essas duas *threads* ativas se faz necessário. É aí que o *GuiAgent* entra. O que ele faz é definir um espaço de execução para cada *thread* ao invés de uma *thread* ficar requisitando a outra para executar métodos. Esse controle é feito através de eventos (*GuiEvents)* em métodos que a própria classe disponibiliza (onGuiEvent *()* e *postGuiEvent ())*. Em suma, seria um agente para GUIs que controla agentes JADE.
- ***jade.gui.GuiEvent*** - Essa classe define um objeto do tipo *GuiEvent* que é usado para notificar um evento para um *GuiAgent*. Possui dois atributos obrigatórios: a origem do evento e um inteiro identificando o tipo de evento.
- ***jade.gui.AclGui*** – A *AclGui* é uma classe derivada de JPanel do pacote Swing do Java com a adição de todos os controles necessários para editar e visualizar corretamente os campos de uma mensagem ACL. Existem basicamente duas maneiras de usar a *AclGui*:
    - Modo Não Estático – A *AclGui* funciona como uma JPanel comum, sendo adicionada ao Container de um objeto Java. Os métodos *setMsg ()* e *getMsg ()* podem ser usados para mostrar/editar determinada mensagem no *panel*. Os métodos *setEnabled ()* e *setSenderEnabled ()* podem ser usados para habilitar e desabilitar modificações nos campos da mensagem ou no campo do remetente.
    - Modo Estático – No caso, ela provê métodos estáticos que disparam uma janela de dialogo temporária, que inclui o painel *AclGui* e botões de OK e CANCEL, permitindo a visualização e edição de determinadas mensagens. Esses métodos são *editMsgInDialog ()* e *showMsgInDialog ()*.
- ***jade.gui.AIDGui*** - É uma classe derivada da *javax.swing.JDialog* que mostra em uma gui o AID de um agente. O método destinado a esse fim é o *ShowAIDGui ().*

Além das classes citadas acima, JADE dispõe ainda nesse pacote mais de 20 classes para utilização relacionadas com Gui's com as mais variadas utilidades, tais como *JadeLogoButton, AboutJadeAction, AgentTree, BrowserLauncher, VisualAIDList* entre outras.

GLOSSÁRIO

Agentes – São programas de softwares desenvolvidos para auxiliar o usuário na realização de alguma tarefa ou atividade. Normalmente apresentam dezenas de propriedades, entre elas destacam-se autonomia, personalização, reatividade e comunicabilidade.

Algoritmo – Processo de cálculo em que um certo número de regras formais resolve, na generalidade e sem exceções, problemas da mesma natureza; qualquer procedimento que permita mecanizar a obtenção de resultados de tipo determinado.

Ambiente de Autoria – É um programa equipado com diversas ferramentas que permitem o desenvolvimento de projetos de forma autônoma, no qual o usuário organiza as informações da forma que achar mais conveniente visando a obtenção de um resultado final.

Log – É uma espécie de registro integral contínuo e ininterrupto de ações em um determinado sistema.

Mapas Conceituais – Idealizados por John Novak, são utilizados como uma linguagem para descrição e comunicação de conceitos. Representam uma estrutura que vai desde os conceitos mais abrangentes até os menos inclusivos. São utilizados para auxiliar a ordenação e a seqüenciação hierarquizada de conteúdos.

Middleware - no campo de computação distribuída, é um programa de computador que faz a mediação entre outros softwares. É utilizado para mover informações entre programas ocultando do programador diferenças de protocolos de comunicação, plataformas e dependências do sistema operacional.

Open Source - O software chamado *open source*, ou em português, código aberto, é um tipo de software cujo código fonte é visível publicamente. O *software* de código aberto respeita as quatro liberdades definidas pela Free Software Foundation, porém, não estabelece certas restrições como as contidas na GPL. É advogado pela Iniciativa do Código Aberto (*Open Source Initiative*)

Peer-to-Peer - O P2P ou Peer-to-Peer é uma tecnologia para estabelecer uma espécie de rede de computadores virtual, onde cada estação possui capacidades e responsabilidades equivalentes. Difere da arquitetura cliente/servidor, no qual alguns computadores são dedicados a servirem dados a outros. Esta definição, porém, ainda é demasiado sucinta para representar todos os significados do termo Peer-to-Peer.

Wireless - Rede sem fio (também chamada rede wireless ou rede Wi-Fi) refere-se à uma rede interligada sem fios, isto é, por canais de comunicação alternativos (como rádio-frequência, infravermelho ou laser). O termo começou a ser usado no Reino Unido, logo depois que uma rádio começou a transmitir através dessa estratégia. Este tipo de redes, quando locais, também se designa por Wlan, *wireless* LAN. O termo WiFi vem da abreviação de Wireless Fidelity (uma noção no nível abstrato que implica uma conexão confiável a uma fonte), é um conjunto de padrões de compatibilidade para wireless local area networks (WLAN) baseado nas especificações IEEE 802.11.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo