
**UNIVERSIDADE FEDERAL DO AMAZONAS**

**RICARDO DOS SANTOS CÂMARA**

# AMBIENTE VIRTUAL PARA O ENSINO E APRENDIZAGEM DA PROGRAMAÇÃO DE AGENTES – AVEPA

**Manaus**

**2010**

**RICARDO DOS SANTOS CÂMARA**

# AMBIENTE VIRTUAL PARA O ENSINO E APRENDIZAGEM DA PROGRAMAÇÃO DE AGENTES – AVEPA

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Ciências Exatas da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. José Francisco de Magalhães Netto, DSc.

Manaus

2010

**RICARDO DOS SANTOS CÂMARA**

# AMBIENTE VIRTUAL PARA O ENSINO E APRENDIZAGEM DA PROGRAMAÇÃO DE AGENTES – AVEPA

Esta dissertação foi julgada adequada à obtenção do título de Mestre em Informática e aprovada em sua forma final pelo Curso de Mestrado em Informática da Universidade Federal do Amazonas.

Manaus, 01 de Junho de 2010.

_____

Professor e orientador José Francisco de Magalhães Netto, DSc.
Universidade Federal do Amazonas

_____

Prof. Alberto Nogueira de Castro Júnior, PhD.
Universidade Federal do Amazonas

_____

Prof. Crediné Silva de Menezes, DSc.
Universidade Federal do Espírito Santo

À minha filha, Melissa da Silva Câmara

## AGRADECIMENTOS

Aos meus pais, que me proporcionaram o acesso à educação.

Ao amigo e orientador, Professor José Francisco de Magalhães Netto, pela confiança creditada em mim, sempre apontando o caminho correto, com valiosas sugestões para que o projeto fosse concluído.

Ao Tribunal de Justiça do Amazonas e Fucapi, pela liberação e flexibilidade nos horários de trabalho no período dos estudos.

## RESUMO

Com a crescente oferta de cursos à distância e de softwares educacionais que apóiam o processo de ensino e aprendizagem, a utilização de ambientes virtuais de aprendizagem (AVA) vem ganhando cada vez mais atenção por parte dos pesquisadores das áreas de Inteligência Artificial e Informática na Educação. Este trabalho propõe uma utilização diferenciada destes ambientes, criando um software educacional baseado em Sistemas Multiagente (SMA) para apoiar o processo de ensino e aprendizagem da programação orientada a agentes (AOP), ou seja, a idéia principal consiste em utilizar agentes que ensinam a programar agentes. Através desta abordagem é possível guiar o aluno por algumas áreas do processo de aprendizado, inicialmente, observando os conceitos mais relevantes e, futuramente, interagindo de forma mais ativa com os cenários que compõem o ambiente proposto.



Palavras-chave: Inteligência Artificial, Informática na Educação, Sistemas Multiagente, Ambiente Virtual de Aprendizagem, Software Educacional, Programação Orientada a Agentes.

## ABSTRACT

With the increasing availability of distance courses and educational software that support the teaching learning, the use of virtual learning environments has been gaining increasing attention from researchers in Artificial Intelligence and Computer Science in Education. This work proposes a distinct use of these environments, creating an educational software based on Multiagent Systems (MAS) to support teaching and learning process of agent oriented programming (AOP), ie, the main idea is to use agents who teach the program agents. Through this approach can guide the student through some areas of the learning process, initially by looking at the most relevant concepts and in the future, interacting more actively with the scenarios that compose the proposed environment.



Keywords: Artificial Intelligence, Computers in Education, Multiagent Systems, Virtual Learning Environment, Educational Software, Agent Oriented Programming.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| ACL | Agent Communication Language |
| AmCorA | Ambiente Cooperativo de Apoio à Aprendizagem |
| AMS | Agent Management System |
| AOP | Agent-Oriented Programming |
| AUML | Agent-Based Unified Modeling Language |
| AVA | Ambiente Virtual de Aprendizagem |
| AVAX | Ambiente Virtual de Aprendizagem em Xadrez |
| AVEPA | Ambiente Virtual para o Ensino e Aprendizagem da Programação de Agentes |
| CASE | Computer Aided Software Engineering |
| DF | Directory Facilitator |
| DTD | Document Type Definition |
| EaD | Educação a Distância |
| FAQ | Frequently Asked Questions |
| FIPA | Foundation for Intelligent Physical Agents |
| FIPA-OS | FIPA-Open Source |
| GUI | Graphic User Interface |
| IA | Inteligência Artificial |
| IIOP | Internet Inter-ORB Protocol |
| JADE | Java Agent DEvelopment Framework |
| JRE | Java Runtime Environment |
| JVM | Java Virtual Machine |
| KQML | Knowledge Query Manipulation Language |
| MEC | Ministério da Educação |
| PAT | Pedagogical and Affective Tutor |
| POO | Programação Orientada a Objetos |
| PDA | Personal Digital Assistant |
| RMA | Remote Monitoring Agent |
| RMI | Remote Method Invocation |
| SL | Semantic Language |
| SMA | Sistema Multiagente |
| SysMLOMG | Systems Modeling Language |
| TCP/IP | Transmission Control Protocol/Internet Protocol |
| UFAM | Universidade Federal do Amazonas |
| UML | Unified Modeling Language |
| XML | Extensible Markup Language |

**SUMÁRIO**

# 1 INTRODUÇÃO

A utilização e desenvolvimento de ambientes virtuais de aprendizagem podem ser considerados como práticas constantes nas atuais pesquisas em Informática na Educação e na Educação a Distância. Computadores pessoais e a Internet são algumas das principais ferramentas da chamada Era da Informação, nos quais os processos de construção do conhecimento exigem alunos e profissionais disciplinados, criativos, críticos e, de certa forma, autodidatas. A prática mostra que a demanda gerada pela dinâmica deste processo não pode ser mais atendida somente pelas aulas tradicionais, baseadas simplesmente nas interações entre professores e alunos, ocorridas nas salas de aula.

Neste contexto, entende-se que os ambientes virtuais devem complementar o processo de ensino e aprendizagem, sendo apresentados como mais um recurso educacional. Tomando esta demanda como motivação, propõe-se utilizar recursos visuais, emulando cenários baseados em Sistemas Multiagente (SMA), como forma de facilitar o aprendizado da Programação Orientada a Agentes. Os conceitos que envolvem esta técnica de programação, como troca de mensagens e comportamentos, por exemplo, são exibidos para os usuários em ambientes gráficos, facilitando seu entendimento e assimilação.

De acordo com Wooldridge (2009), no paradigma da programação orientada a agentes, um agente é determinado pelas suas crenças, capacidades e compromissos, fatores que constituem seu estado mental. Estes agentes podem interagir em comunidades, através da troca de mensagens, cooperando ou competindo. No que diz respeito à parte prática, Netto (2006) alerta para o fato de que existe uma falta de hábito na utilização de SMA, já que a atual formação acadêmica e profissional na área computação é bastante influenciada pelo paradigma de orientação a objetos.

No escopo deste trabalho, um agente computacional é caracterizado como um sistema autônomo que busca ou colabora para que metas sejam alcançadas dentro de um ambiente real ou virtual.

## 1.1 Especificação do Problema

Através de observações empíricas, é possível notar a dificuldade de alguns alunos em disciplinas como Inteligência Artificial ou Sistemas Multiagente. Em certos casos estas

dificuldades atingem níveis mais elevados, levando os estudantes a criar aversão a linguagens como Prolog e os conceitos técnicos que envolvem a IA.

É possível que este fato reflita negativamente nos cursos de Pós-Graduação em Informática, diminuindo a procura pelas linhas de pesquisa em IA.

Tomando como exemplo outras áreas tradicionais, onde alunos têm algumas dificuldades de assimilação, como a matemática, observa-se que certas iniciativas são comumente tomadas para facilitar o processo de ensino e aprendizagem, como a utilização de jogos e ambientes virtuais. Voltando o foco para a IA, percebe-se a carência deste tipo de iniciativas nessa área, ou seja, não é comum encontrar ambientes virtuais de aprendizagem ou jogos que facilitem o ensino e aprendizagem dos conceitos que envolvem as áreas de IA e SMA. Utiliza-se sim, com certa freqüência, a aplicação de IA em diversas áreas, como agentes em Educação a Distância ou IA em jogos, por exemplo.

Este fato gera uma demanda imediata: Explorar o potencial que recursos como jogos ou simulação possuem no processo de ensino e aprendizagem. Este foi o principal motivador para que as pesquisas fossem iniciadas.

## 1.2 Delimitação do Problema

A proposta consiste em facilitar a aprendizagem de conceitos da Programação Orientada a Agentes e os seus paradigmas, criando um ambiente onde o aluno possa atuar, basicamente, de duas formas:

1. De forma observacional, assimilando as características mais relevantes para o desenvolvimento de SMA's, ou seja, o ambiente é utilizado basicamente como um emulador, com a diferença de que há algumas interações que possibilitam que o usuário não seja classificado apenas como um mero observador, como a visualização de códigos e das mensagens trocadas; e
2. Ativamente, onde, após assimilar os conceitos teóricos e práticos do paradigma de programação de SMA's, ele seja capaz de interagir com os agentes computacionais, enviando e recebendo mensagens, por exemplo.

Para que as atuações descritas anteriormente sejam possíveis, é escolhido um cenário de resgate numa floresta, especificado no Capítulo 6, onde os agentes cooperam em busca de um objetivo comum, resgatar a(s) vítima(s).

## 1.3 Motivação

A crescente utilização de soluções que empregam a técnica de agentes, tanto pelo meio acadêmico quanto pelas indústrias, e a necessidade de dotar alunos e professores de ferramentas que facilitem o aprendizado desta técnica de programação e seus conceitos, são alguns dos fatores que motivaram e tornaram plausível a proposta de se utilizar recursos visuais que emulem e possibilitem a interação dos usuários com os conceitos da Programação Orientada a Agentes.

Outro fator que merece destaque, percebido por observações em salas de aula, consiste no fato de que as fronteiras entre agentes e objetos ainda não são vistas de forma clara por parte dos alunos que se aventuram no aprendizado desta nova abordagem.

## 1.4 Objetivos

A proposta apresentada tem como objetivo principal criar um software educacional baseado em comunidades de agentes, a fim de apoiar o processo de ensino e aprendizagem da Programação Orientada a Agentes. Tal objetivo pode ser alcançado através dos seguintes objetivos específicos:

- Possibilitar a simulação de cenários povoados por agentes.
- Possibilitar a visualização gráfica dos cenários e comportamentos dos agentes, como troca de mensagens, negociações e cooperações.
- Possibilitar a visualização dos códigos dos agentes e cenários.
- Possibilitar que o aluno adicione agentes num cenário previamente definido e interaja com o ambiente.
- Possibilitar que o aluno modifique o cenário e observe o comportamento dos agentes, percebendo as alterações e se adaptando as mesmas.
- Criar protótipos do ambiente virtual proposto contendo as principais funcionalidades propostas.
- Executar experimentos utilizando os protótipos desenvolvidos.

## 1.5 Contribuição Esperada

Espera-se que o ambiente proposto possa ser utilizado como um software educacional, apoiando o processo de ensino e aprendizagem da Programação Orientada a Agentes,

possibilitando, desta forma, que os alunos diferenciem, sem maiores dificuldades, esta técnica de outras, como o da Programação Orientada a Objetos, por exemplo.

Após explorar o ambiente educacional proposto, é esperado que os usuários assimilem as principais características e conceitos que envolvem a programação de agentes, bem como seus comportamentos num Sistema Multiagente, possibilitando, desta forma, aprofundar os estudos de problemas específicos que envolvem este paradigma de programação.

## 1.6 Metodologia

Foram desenvolvidos protótipos do ambiente proposto. As etapas da metodologia adotada são listadas abaixo:

1. Levantamento bibliográfico sobre metodologias de modelagem e implementação de Sistemas Multiagente. Onde foram comparadas diversas técnicas de análise e desenvolvimento de SMAs, bem como padrões de linguagens e comunicações utilizados.
2. Levantamento bibliográfico sobre as ferramentas disponíveis para modelagem e implementação de Sistemas Multiagente. Onde foram levantadas as ferramentas que facilitam o desenvolvimento de SMAs, como frameworks e GUI's disponíveis.
3. Definição de um cenário específico que foi implementado em forma de protótipo.
4. Análise, projeto, desenvolvimento e validação do ambiente virtual. Nesta etapa foram desenvolvidos e testados os protótipos do ambiente proposto.

## 1.7 Organização do Texto

O restante do texto está organizado da seguinte forma: o Capítulo 2 apresenta as metodologias e tecnologias envolvidas, como Agentes, Sistemas Multiagente, Java, JADE e NetLogo; o Capítulo 3 aborda os trabalhos correlatos, enfatizando as principais características e diferenças, comparando-os a esta pesquisa; o Capítulo 4 trata da arquitetura empregada, da análise e do projeto do sistema; o Capítulo 5 descreve a implementação e exibe os resultados obtidos nos testes e avaliações executados; o Capítulo 6 descreve alguns exemplos de uso do

protótipo implementado; por fim, o Capítulo 7 apresenta os objetivos alcançados e as considerações finais, bem como os trabalhos em andamento e futuros, que darão continuidade à pesquisa.

# 2 TRABALHOS CORRELATOS

Neste capítulo são apresentados os trabalhos correlatos que serviram de base para a proposta. São abordadas as suas principais características, bem como as diferenças relevantes, se comparados a esta pesquisa.

Inicialmente, devido à dificuldade em encontrar trabalhos com foco baseado especificamente em apoiar o aprendizado de Sistemas Multiagente através de simulações executadas por agentes, ou seja, um ambiente simulado composto por uma comunidade de agentes emulando as principais características do Paradigma da Programação Orientada a Agentes – AOP, Câmara (2009a) cita trabalhos correlatos que apóiam o ensino e aprendizagem em geral. Nesta abordagem são citados o Ambiente Virtual para a Aprendizagem de Xadrez – AVAX, proposto por Netto (2005) e o Sistema Tutor Inteligente para um Ambiente Virtual de Aprendizagem – STI, proposto por Lima (2004).

Entretanto, aprofundando ainda mais as pesquisas sobre trabalhos similares já desenvolvidos ou em desenvolvimento, foram identificados dois trabalhos com fortes semelhanças a esta proposta. Na primeira, Fasli e Michalakopoulos (2005) propõem uma ferramenta de software que apóia o ensino, a aprendizagem e a avaliação dos conceitos de Inteligência Artificial e de Sistemas Multiagente. O ambiente proposto é denominado e-Game. Trata-se de um jogo que promove a aprendizagem ativa por meio de exercícios de simulação de um ambiente de leilões. O sistema pode ser usado para ensinar e demonstrar os princípios de protocolos de negociação, os mecanismos de mercado e as estratégias dos agentes. Além disso, possibilita que os instrutores desenvolvam exercícios que podem ser usados no ensino ou avaliação dos alunos. Na segunda, proposta por Azevedo e Menezes (2007), é proposta uma ferramenta com finalidades didáticas, objetivando motivar os usuários a criar modelos de simulação de forma cooperativa.

Em seguida, mudando o foco para trabalhos que utilizam simulações de Sistemas Multiagente, os seguintes trabalhos apresentam similaridades com esta proposta.

Proença (2002) propõe um Sistema Multiagente Regulador de Tráfego, onde são simuladas estradas e cruzamentos com correspondente quantidade de tráfego associada. O objetivo é que um SMA efetue a gestão dos semáforos de forma a minimizar o tempo de espera por parte dos veículos.

Nos subtópicos seguintes os trabalhos citados são descritos de forma mais detalhada.

## 2.1 Ambiente Virtual para a Aprendizagem de Xadrez – AVAX

Netto (2005) propõe um ambiente virtual de prática e aprendizagem do jogo de xadrez, baseado na tecnologia de Sistemas Multiagente, o qual chama de AVAX.

Aqui a comunidade virtual é composta por usuários e agentes heterogêneos, de forma similar a este trabalho. Outra característica em comum consiste no fato da principal tecnologia utilizada para implementar o ambiente proposto é o JADE e a comunicação entre os agentes é baseada na FIPA. No AVAX é utilizada uma ontologia para a representação dos conceitos que envolvem o sistema. A Figura 1 ilustra os agentes que compõem uma execução do AVAX trocando mensagens.

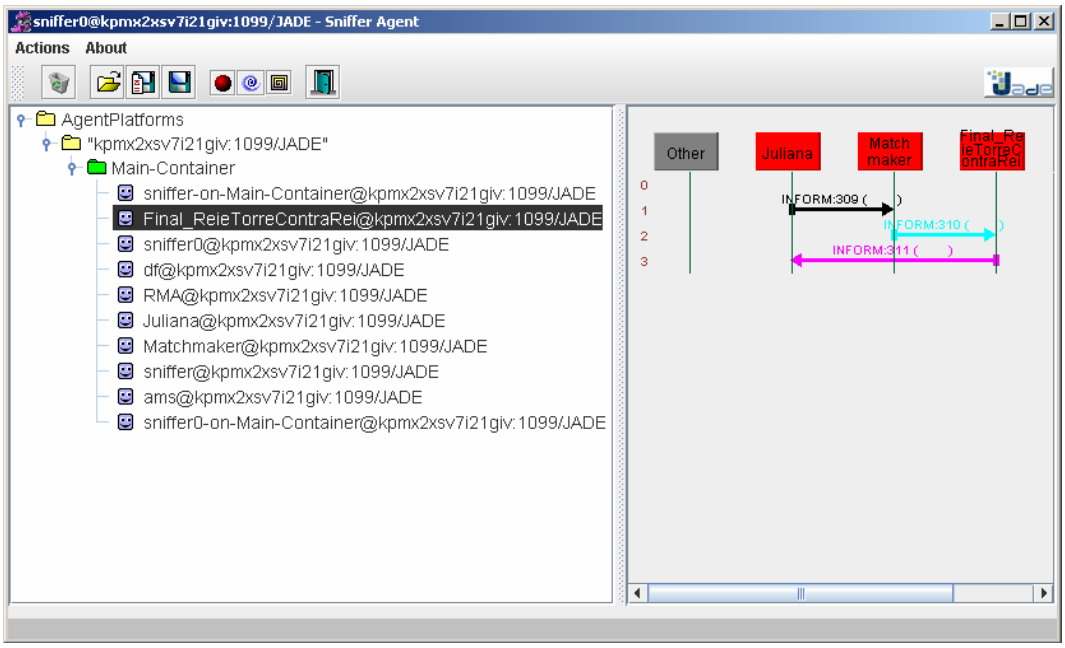


Figura 1: Troca de mensagens entre agentes no AVAX –Netto (2005)

A principal diferença entre o AVAX e o AVEPA consiste no fato do AVEPA utilizar simulação e, o AVAX, a idéia de tutores.

## 2.2 Sistema Tutor Inteligente para um Ambiente Virtual de Aprendizagem – STI

Outro trabalho correlato pode ser encontrado na proposta de Lima (2004), que explora um sistema tutor inteligente aplicado num ambiente virtual de ensino e aprendizagem. Aqui a

tutoria consiste em fornecer suporte e acompanhar o usuário durante um curso a distância, utilizando um agente inteligente que faz uso da técnica de Raciocínio Baseado em Casos.

O foco deste trabalho consiste em orientar o aluno no processo de aprendizado, realizando um trabalho de acompanhamento de forma constante, fomentando a troca de experiências entre aluno/tutor e aluno/colegas de turma. A tutoria deve fomentar um processo de aprendizado colaborativo, onde todos os envolvidos contribuem para o crescimento contínuo do grupo como um todo.

Neste trabalho, as FAQ's (perguntas freqüentes) são utilizadas como base de conhecimento, podendo ser adaptado a partir do retorno fornecido pelos usuários. A Figura 2 representa este processo.

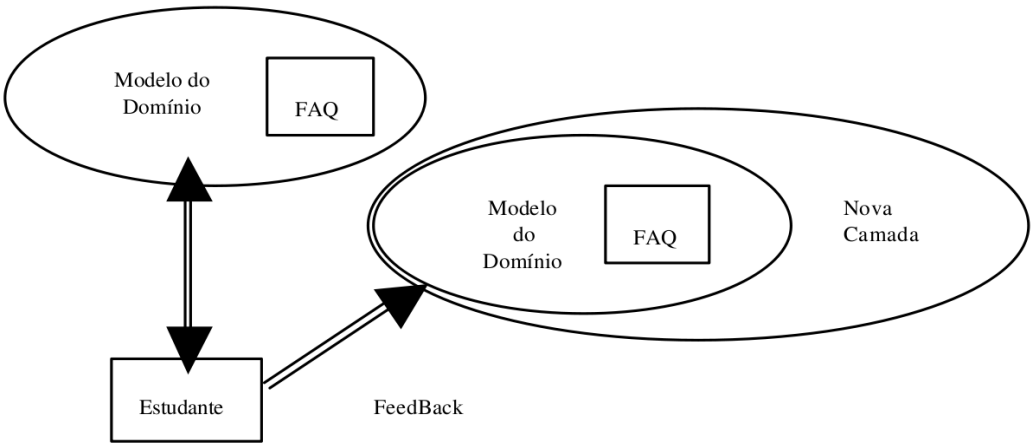


Figura 2: Acesso e adaptação do modelo de domínio do STI – Lima (2004)

Novamente o fato do AVEPA utilizar simulação com fins didáticos pode ser citado como diferença do STI.

## 2.3 e-Game

Trata-se de um jogo que promove a aprendizagem ativa por meio de exercícios de simulação de um ambiente de leilões. O sistema pode ser usado para ensinar e demonstrar os princípios de protocolos de negociação, os mecanismos de mercado e as estratégias dos agentes. Além disso, de acordo com Fasli e Michalakopoulos (2005), possibilita que os instrutores

desenvolvam exercícios que podem ser usados no ensino ou avaliação dos alunos. A Figura 3 ilustra a arquitetura do sistema.

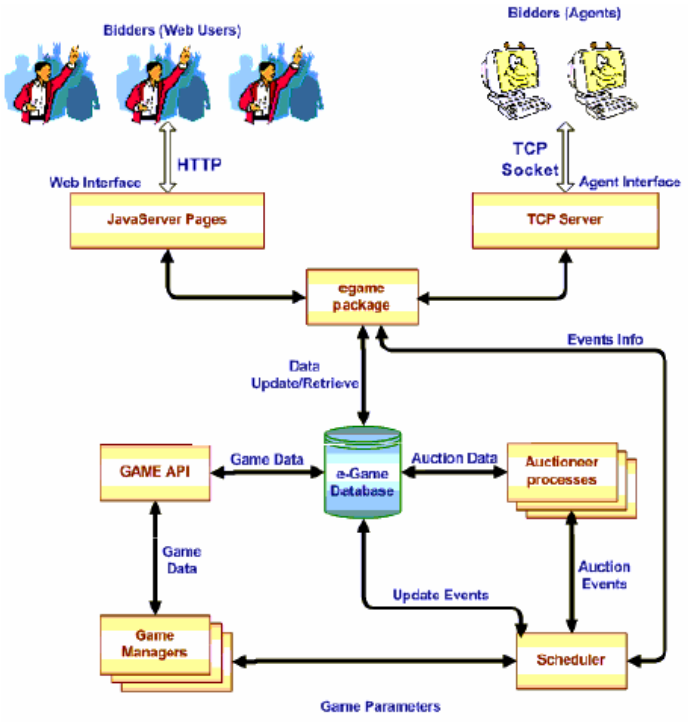


Figura 3: Arquitetura do e-Game – Fasli e Michalakopoulos (2005)

Na Figura 4 pode ser observada uma tela de execução do e-Game.

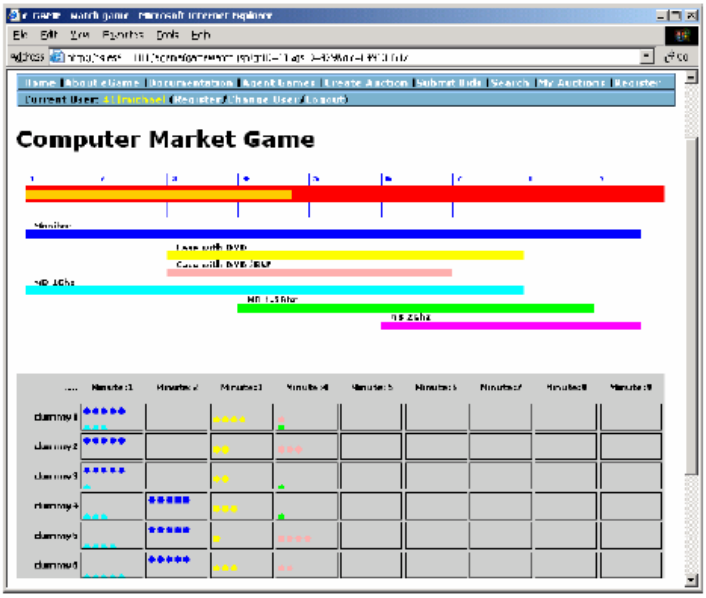


Figura 4: Tela de execução do e-Game – Fasli e Michalakopoulos (2005)

A principal diferença entre o e-Game e esta proposta consiste no foco baseado em jogos. Apesar de permitir que os alunos assumam os papéis de determinados agentes nas simulações, este trabalho não utiliza jogos de forma direta, pois não são computados escores ou utilizadas competições entre alunos e agentes, por exemplo.

## 2.4 NetPlay - uma ferramenta para construção de modelos de simulação baseado em multiagente

Conforme citado anteriormente, sua construção foi motivada pela dificuldade encontrada entre pesquisadores e usuários na utilização de alguma ferramenta que possibilita descrever modelos de simulação baseados em agentes, como o MASON, NetLogo e outros.

De acordo com Azevedo e Menezes (2007), o NetPlay possui finalidades didáticas e pode ser considerado de fácil manuseio, de forma a não exigir nenhum conhecimento do modelador sobre as características de agentes ou da linguagem de programação. Esta pode ser citada como a principal diferença entre o NetPlay e o AVEPA, pois este último tem como objetivo facilitar o aprendizado das características da AOP.

A Figura 5 ilustra a estrutura interna do NetPlay.

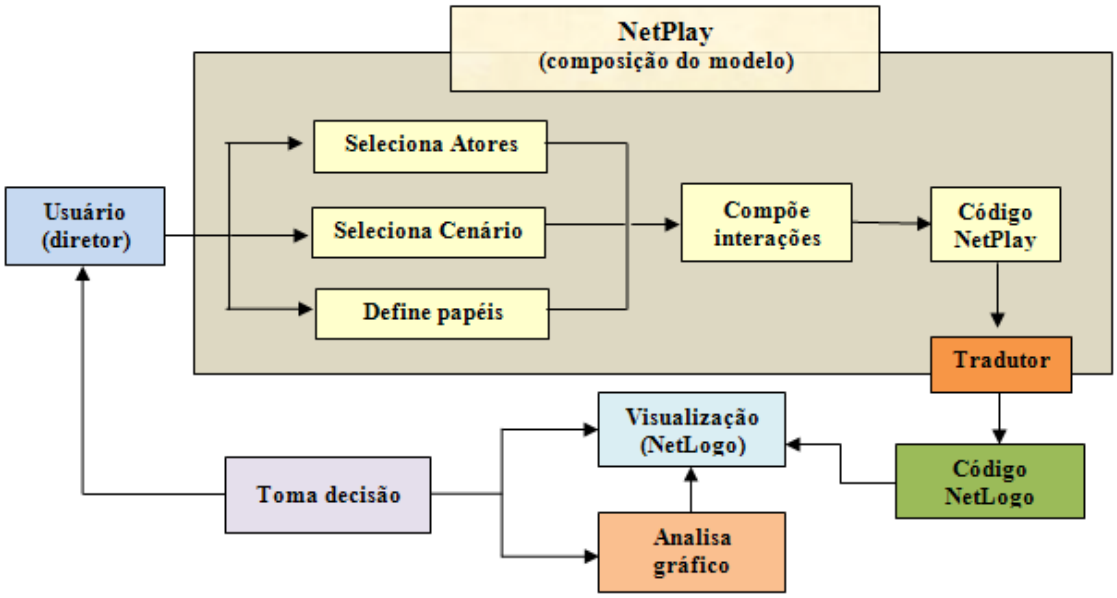


Figura 5: Estrutura interna do NetPlay - Azevedo e Menezes (2007)

## 2.5 Sistema Multiagente Regulador de Tráfego

Nesta abordagem Proença (2002) simula uma série de estradas e cruzamentos com correspondente quantidade de tráfego associada. O objetivo é que um Sistema Multiagente efetue a gestão dos semáforos de forma a minimizar o tempo de espera por parte dos veículos. A Figura 6 ilustra a arquitetura deste sistema.

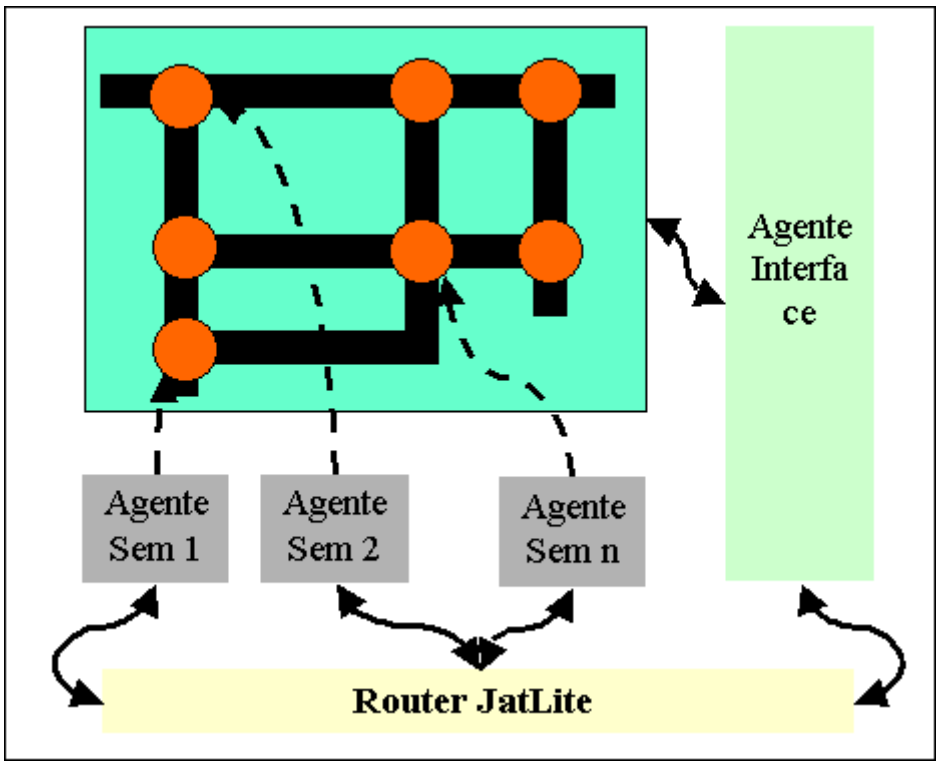


Figura 6: Arquitetura do Sistema Multiagente Regulador de Tráfego – Proença (2002)

A principal diferença para esta proposta é que não há finalidades didáticas neste simulador, ou seja, o trabalho não tem como objetivo ensinar a Programação Orientada a Agentes utilizando a simulação. Por outro lado, do ponto de vista de simuladores e de Sistemas Multiagente, este aplicativo utiliza boa parte dos conceitos envolvidos em SMA, abordando um problema clássico, o controle de tráfego. Uma das telas do sistema pode ser observada na Figura 7.

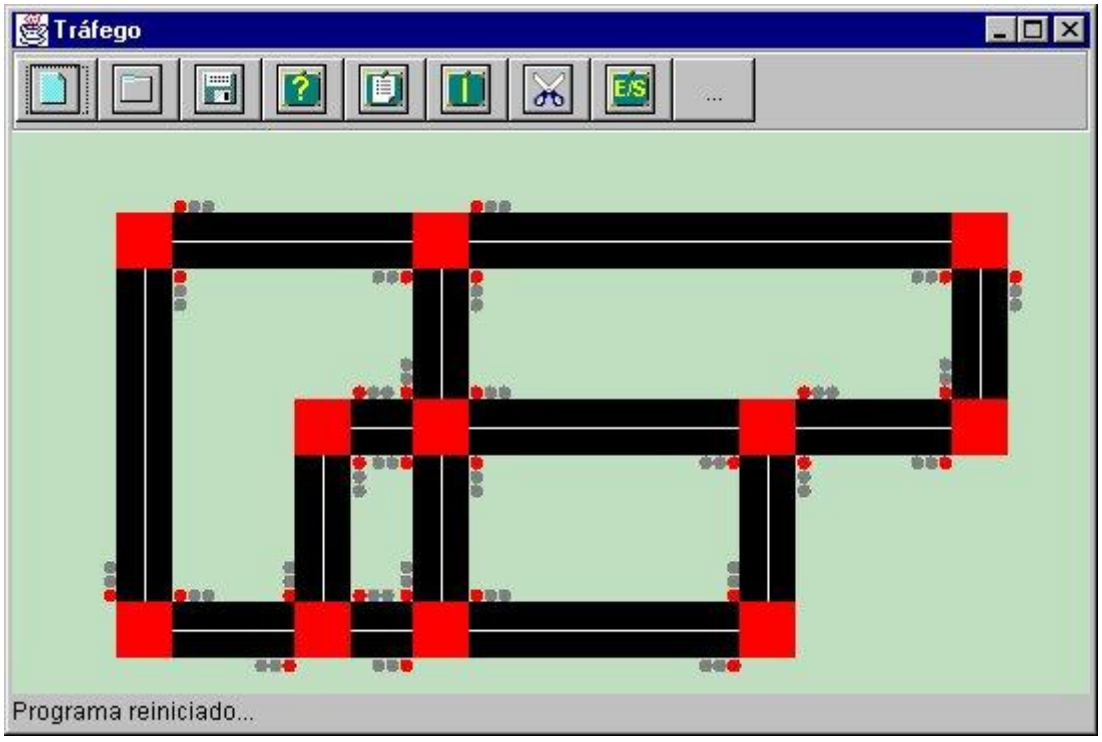


Figura 7: Uma tela do Sistema Multiagente Regulador de Tráfego – Proença (2002)

# 3 REFERENCIAL TEÓRICO

Neste capítulo são abordados as principais características e conceitos de agentes, Sistemas Multiagente, linguagens de comunicação, ontologias e as tecnologias utilizadas na documentação e desenvolvimento do protótipo, como AUML, JADE e NetLogo.

## 3.1 Sistema Multiagente Regulador de Tráfego

No contexto deste trabalho, ambientes virtuais de aprendizagem podem ser classificados como sistemas que apóiam o processo de ensino e aprendizagem pela Web, utilizando softwares educacionais. Estes ambientes são comumente utilizados para que os professores gerenciem conteúdos para os alunos ou na complementação de aulas presenciais. O Moodle ou TelEduc podem ser citados como exemplos destes ambientes. Uma tela do Moodle pode ser observada na Figura 8.

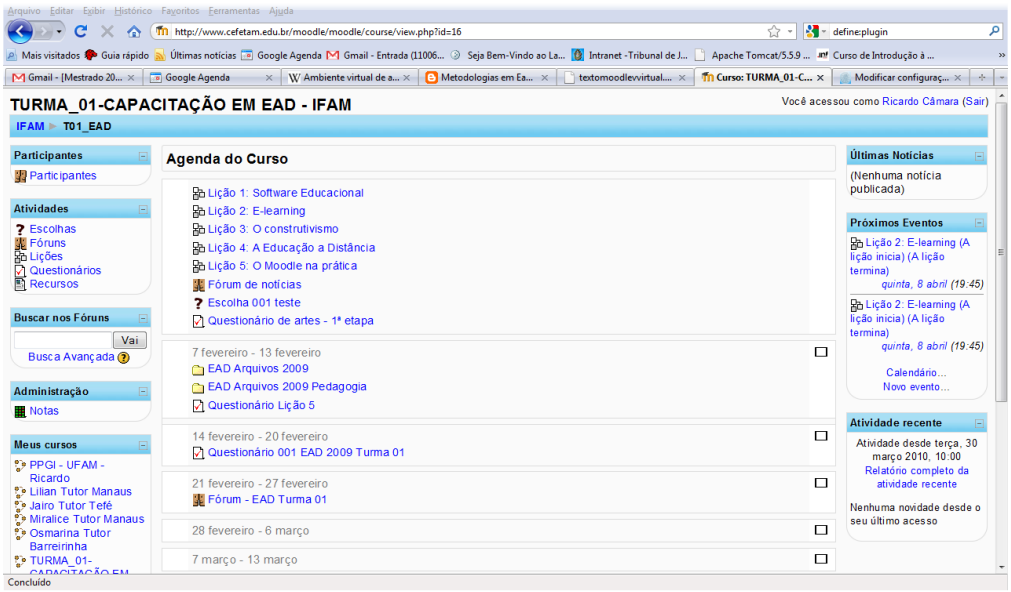


Figura 8: Uma tela do Moodle

Utilizando um AVA, como o ilustrado na Figura 8, é possível disponibilizar lições, materiais para download, participar de fóruns e chats, entre outras atividades. Ambientes Virtuais de Aprendizagem e Educação a Distância são termos fortemente

relacionados. Neste sentido, faz-se importante definir alguns termos, como EaD, ensinar, aprender, software educacional e tutoriais de auto-estudo.

### 3.1.1 Educação a Distância

De acordo com as definições e recomendações disponíveis no MEC, a Educação a Distância é a modalidade educacional na qual a mediação didático-pedagógica nos processos de ensino e aprendizagem ocorre com a utilização de meios e tecnologias de informação e comunicação, com estudantes e professores desenvolvendo atividades educativas em lugares ou tempos diversos.

É válido ressaltar que a EaD não é um método de ensino, mas uma modalidade, que pode ser adaptada a diferentes métodos e abordagens pedagógicas.

### 3.1.2 Aprender e Ensinar

De acordo com Weisz (1999), o sujeito aprende em situações funcionais quando:

- Tem a necessidade de usar todo conhecimento já construído para resolver determinada atividade.
- Há, realmente, um problema a ser resolvido e decisões a serem tomadas pelos alunos, em função do que se pretende produzir.
- O conteúdo da atividade caracteriza-se por ser um objeto sócio-cultural real.
- A organização da tarefa pelo professor garante o intercâmbio de informações.

No que tange ao ensino, há itens que são, idealmente, considerados básicos para o trabalho docente:

- Conhecer e estar atento às reações dos alunos;
- Pensar o processo de aprendizagem como algo ativo, que depende das construções cognitivas do aluno.
- Aceitar e discutir as diferentes formas de pensar.
- Manter o foco na formação dos alunos.

Partindo destes princípios, surgem alguns questionamentos:

- Como conciliar as tarefas descritas anteriormente?
- Dado um contexto, quais as formas de avaliação mais eficientes?

- Na educação convencional, estes questionamentos já são um grande desafio, então, como otimizar a utilização de tecnologias para que estas auxiliem, da melhor forma possível, no processo de ensino e aprendizagem?

Analisar o mérito das questões levantadas anteriormente foge ao escopo deste trabalho. Algumas respostas podem ser encontradas em pesquisas na área da educação, como nas publicações de Mizukami (1986) e Oliveira (1997).

### 3.1.3 Software Educacional

Conforme mencionado anteriormente, um dos objetivos desta pesquisa é a criação de um software educacional para apoiar o processo de ensino e aprendizagem de Sistemas Multiagente. Por este motivo, é importante definir quais as características de um software educacional.

Sistemas cujo objetivo consiste em estimular o aprendizado de uma forma mais autônoma e que estimulem certas habilidades cognitivas, obviamente sem a intenção de substituir o professor, podem ser classificados como softwares educacionais. É importante frisar que estes aplicativos não são necessariamente utilizados na EaD, é perfeitamente possível e, de certa forma comum, que a utilização ocorra de forma presencial, mediada ou orientada por um professor.

A Figura 9 fornece um exemplo de software educacional, chamado de Balança Interativa proposta por Castro Filho *et al* (2008).

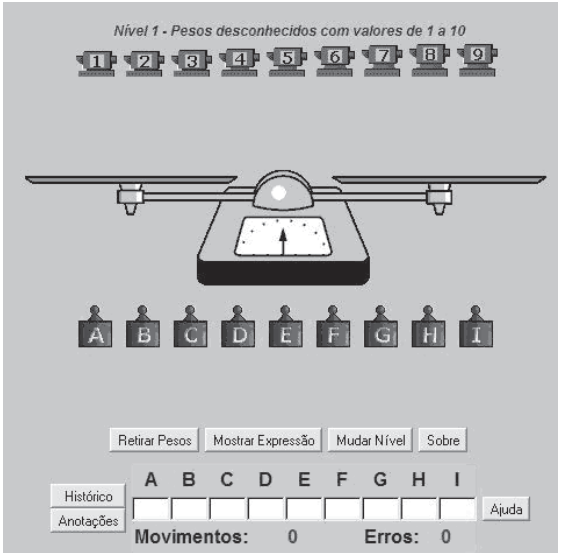


Figura 9: Tela inicial do software Balança Interativa proposta por Castro Filho *et al* (2008)

O aplicativo é baseado na manipulação simulada de uma balança na forma de um jogo, consistindo em descobrir os valores desconhecidos que são associados às letras. Em determinada situação, o aluno tem pesos conhecidos, representados por números, e desconhecidos, representados por letras. A partir deste ponto, deve-se utilizar o objeto para comparar os pesos através da igualdade ou desigualdade na balança.

### 3.1.4 Tutoriais de Auto-Estudo

Podem ser considerados como um dos recursos importantes em AVA's.

Conceituados como módulos virtuais contendo material didático, organizados para serem estudados de forma totalmente autônoma. O conteúdo é apresentado em multimídia, com textos, animações, vídeo e áudio, conforme convier.

Por não contarem com as intervenções permanentes de um tutor ou professor, devem incluir um FAQ (perguntas mais freqüentes) e exercícios com resposta automática, onde o aluno fica sabendo na hora se acertou ou não, sendo orientado para rever o conteúdo específico. Além disso, recomenda-se disponibilizar um e-mail para dúvidas.

Podem ser distribuído em CD-ROM ou disponíveis na internet.

## 3.2 Agentes

No campo da Inteligência artificial, um agente pode ser visto como uma entidade autônoma, capaz de tomar decisões sem interferências externas. Também deve possuir a capacidade de interagir com o ambiente e com outros agentes, de forma cooperativa ou competitiva. Russell e Norvig (2002) definem agente como algo que pode perceber seu ambiente através de sensores e agir nesse ambiente por meio de atuadores. A Figura 10 ilustra esta definição.

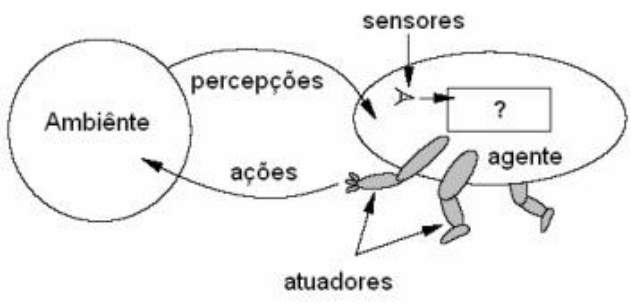


Figura 10: Definição de agente – Russel e Norvig (2002)

Certamente existem inúmeras definições para agentes, porém, ainda não há um conceito fechado universalmente. Wooldridge e Jennings (1994) comparam a questão “o que é um agente?” com “o que é inteligência?”, chegando à conclusão da sua imprevisão no cenário computacional, visto que, nos dois casos, não existe uma definição única para o tema.

É interessante citar algumas aplicações práticas da utilização de agentes para prover soluções computacionais.

Campana *et al* (2008) propõem a utilização de agentes para apoiar o acompanhamento das atividades em ambientes virtuais de aprendizagem. É apresentada uma arquitetura multiagente para ambientes virtuais de aprendizagem que contempla uma camada multiagente para AVA's. O objetivo desta camada 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.

Em outra abordagem, Jaques *et al* (2008) avaliam a efetividade de um agente pedagógico animado emocional. É apresentada a avaliação empírica realizada de um agente pedagógico animado e afetivo, chamado Pat (Pedagogical and Affective Tutor), cujo objetivo principal é fornecer suporte emocional ao aluno, motivando-o, encorajando-o a acreditar em suas próprias habilidades, e incentivando-o a aumentar seus esforços e sua motivação intrínseca.

## 3.3 Sistemas Multiagente

Um Sistema Multiagente (SMA) pode ser definido como um sistema composto por uma ou mais comunidade de agentes interagindo. Este paradigma pode ser usado para resolver problemas complexos para um agente individual ou para um sistema tradicional resolver. Soluções de comércio eletrônico e ambientes que modelam estruturas sociais são exemplos de problemas que podem fazer uso de Sistemas Multiagente.

Wooldridge (2009) e Sycara (1996) citam algumas características importantes que um agente inserido numa comunidade de agentes deve possuir. Dentre elas, merecem destaque:

- Autonomia: Os agentes são, pelo menos, parcialmente autônomos;

- Visões locais: Nenhum agente tem uma visão global do sistema, ou o sistema é demasiado complexo para que um agente faça uso prático de tal conhecimento;
- Computação assíncrona: A computação é não-sincronizada. Os agentes decidem quando lerão uma mensagem que está na fila, por exemplo.

Obviamente, no paradigma de programação descrito neste tópico, onde agentes autônomos interagem entre si numa comunidade, eles precisam trocar informações. Para suprir tal necessidade, existem padrões para troca de mensagens. Estes padrões são discutidos no tópico 3.5.

Podem ser citadas algumas aplicações de Sistemas Multiagente em problemas práticos.

Oliveira *et al* (2009) propõem o Global, um modelo de educação ubíqua, descentralizado, baseado em Sistemas Multiagente, disponibilizando agentes de software que executam tarefas comuns ao processo de aprendizagem ubíqua.

Em outro trabalho, Netto *et al* (2004) propõem o AmCorA: Uma Arquitetura Multiagente Baseada em FIPA. Aqui é proposta uma arquitetura para Ambientes Virtuais de Aprendizagem que estende uma arquitetura já implementada e testada, o AmCorA (Ambiente Cooperativo de Aprendizagem), ampliando suas potencialidades através de uma comunidade de agentes inteligentes representando alunos chamados clones e do emprego de ontologias e de padrões FIPA (*Foundation for Intelligent Physical Agents*) para promover a interoperabilidade.

## 3.4 Foundation for Intelligent Physical Agents – FIPA

Netto (2005) define FIPA como uma organização que visa promover a interoperabilidade entre agentes heterogêneos. Dentre as várias definições, são propostos protocolos de trocas de mensagens para realização de tarefas.

Foi fundada como uma organização sem fins lucrativos em 1996, objetivando definir um conjunto de normas para o desenvolvimento de Sistemas Multiagente, especificando a forma como os agentes devem se comunicar e interagir.

Um dos padrões propostos é a linguagem de comunicação entre agentes, chamada ACL-FIPA, que é discutida no tópico 3.5.2.

## 3.5 Linguagens de Comunicação de Agentes

Conforme mencionado anteriormente, num ambiente povoado por agentes, existe necessidade de comunicação entre eles. Com este objetivo existem as linguagens de comunicação de agentes – ACL. Neste trabalho são apresentadas duas linguagens para comunicação entre agentes: Knowledge Query Manipulation Language – KQML e ACL-FIPA.

### 3.5.1 *Knowledge Query Manipulation Language* – KQML

Finin (1996) define a KQML como uma linguagem e um protocolo para a comunicação entre agentes. Pode ser utilizada como a linguagem que um agente utiliza para interagir com outros, ou para dois ou mais sistemas inteligentes compartilharem conhecimento para solução cooperativa de problemas.

Com relação à semântica das mensagens, Wooldridge (2009) lista as performativas que podem ser observadas na Tabela 1.

Tabela 1: Parâmetros de uma mensagem no padrão KQML

| PARÂMETRO | SIGNIFICADO |
|---|---|
| :content | A informação sobre qual performativa expressa uma atitude. |
| :force | Se o remetente irá sempre negar o significado da performativa. |
| :in-reply-to | O rótulo esperado em resposta. |
| :language | O nome de uma linguagem contida no parâmetro :content. |
| :ontology | O nome da ontologia usado no parâmetro :content. |
| :receiver | O destinatário. |
| :reply-with | Se o transmissor espera uma resposta, e se sim, um rótulo para a resposta. |
| :sender | O atual remetente da performativa. |

Faraco (1998) mostra um exemplo de mensagem escrita em KQML:

Tabela 2: Exemplo de mensagem em KQML – Faraco (1998)

```
(ask-if
        :sender XX
        :receiver YY
        :in-reply-to MSG01
        :reply-with MSG02
        :language Natural
        :ontology 'teste de conteúdo'
        :content ('Este teste serve para mostrar o conteúdo de uma mensagem KQML escrito em linguagem natural')).
```

### 3.5.2 ACL-FIPA

Linguagem para comunicação entre agentes semelhante, no que diz respeito ao formato, à linguagem KQML. Como pode ser observado em FIPA (2004), a quantidade de performativas ACL é inferior ao número de performativas KQML.

Conforme pode ser observado na Tabela 3, a linguagem ACL define algumas performativas, cujo objetivo é prover a interpretação desejada para os diversos tipos de mensagens.

Tabela 3: Performativas da linguagem ACL-FIPA

| PERFORMATIVA | SIGNIFICADO |
|---|---|
| accept-proposal | Aceite de uma proposta numa negociação. |
| Agree | Aceite para desempenhar uma determinada ação. |
| Cancel | Cancelamento da execução de uma determinada ação. |
| Cfp | *Call for proposals*. Inicia uma determinada negociação. |
| Confirm | Confirma a veracidade de uma determinada mensagem. |
| Disconfirm | Inverso de *confirm*. |
| Failure | Falha na tentativa de executar uma determinada ação. |
| Inform | Informa algo aos outros agentes. |
| inform-if | Informação sobre a veracidade de determinada informação. |
| inform-ref | Informação sobre um determinado valor. |

| | |
|---|---|
| not-understood | Não entendimento de uma mensagem. |
| Propagate | Pedido de propagação de uma determinada mensagem a um conjunto de agentes. |
| Propose | Envio de proposta. |
| Proxy | Envio de uma mensagem que será reenviada a um conjunto de agentes. |
| query-if | Pedido de informação sobre a veracidade de determinada informação. |
| query-ref | Pedido de informação sobre um valor. |
| Refuse | Recusa de executar determinada ação. |
| reject-proposal | Recusa de uma proposta efetuada no contexto de uma negociação. |
| Request | Pedido a um dado agente para executar determinada ação. |
| request-when | Pedido para executar uma ação quando certa condição for satisfeita. |
| request-whenever | Pedido para executar uma ação sempre que uma condição seja verdadeira. |
| Subscribe | Pedido para ser informado acerca das alterações relacionadas com determinado fato ou informação. |

Uma das principais diferenças entre as linguagens KQML e ACL-FIPA é que esta última tem maior preocupação com a modelagem semântica e simplifica a quantidade de performativas suportadas. Gomes (2004) mostra um exemplo de mensagem escrita em ACL-FIPA:

Tabela 4: Exemplo de mensagem em KQML – Gomes (2004)

```
(inform
        :sender (agent-identifier :name "Agente LMS")
        :receiver (set (agent-identifier :name ILO))
        :content "((humano estudante))"
        :ontology "escola"
        :language fipa-sl
)
```

### 3.5.3 Ontologias

Conforme citado anteriormente, para que seja possível a interação entre agentes que fazem parte de um Sistema Multiagente, é necessário que exista uma plataforma e uma linguagem de comunicação. Entretanto, outro requisito é fundamental para que a comunicação seja bem sucedida, os agentes devem possuir um vocabulário comum bem definido, neste momento o conceito de ontologias se faz presente. Por exemplo, é usual que diferentes agentes possuam diversas terminologias para o mesmo significado ou vice-versa. Este problema pode ser resolvido se eles partilharem uma ontologia comum. De acordo com Huhns (1997), uma ontologia não é mais que a representação do conhecimento de um dado domínio, disponibilizada a todos os outros componentes de um sistema de informação.

Similar ao conceito de agentes, também existem diversas definições de ontologia propostas pelos pesquisadores da área. Segundo Gruber (1995), uma ontologia é uma especificação formal e explícita de uma abstração, uma visão simplificada de um domínio de conhecimento. Uma ontologia modela uma parte do "mundo", ou seja, um domínio de conhecimento, definindo um vocabulário comum. Com relação aos benefícios na utilização de ontologias, Noy (2001) relaciona alguns deles ao compartilhamento, reuso, estruturação da informação, interoperabilidade e confiabilidade.

No contexto de Sistemas Multiagente, uma ontologia é vista como uma representação formal de conceitos, características e relacionamentos num certo domínio específico, permitindo o entendimento comum da área de conhecimento entre pessoas e agentes de software.

Existem diversos tipos de ontologias, dentre os quais merecem destaque:

- Ontologias genéricas. Definem termos suficientemente genéricos e freqüentemente usados como base para permitir a definição de outros termos compostos. Este também é o conceito de meta-ontologia;
- Ontologias de domínio. Ontologias baseadas nas genéricas, porém, especializadas para uma determinada área. Os termos e conceitos definidos nestas ontologias são referentes à área selecionada;
- Ontologias de aplicação. Contêm as definições necessárias à modelagem do conhecimento numa área específica.

As ontologias são cada vez mais essenciais para o desenvolvimento e utilização de sistemas inteligentes assim como para a interoperabilidade entre sistemas heterogêneos.

Netto (2005) fornece um exemplo de ontologia de domínio na Figura 11.

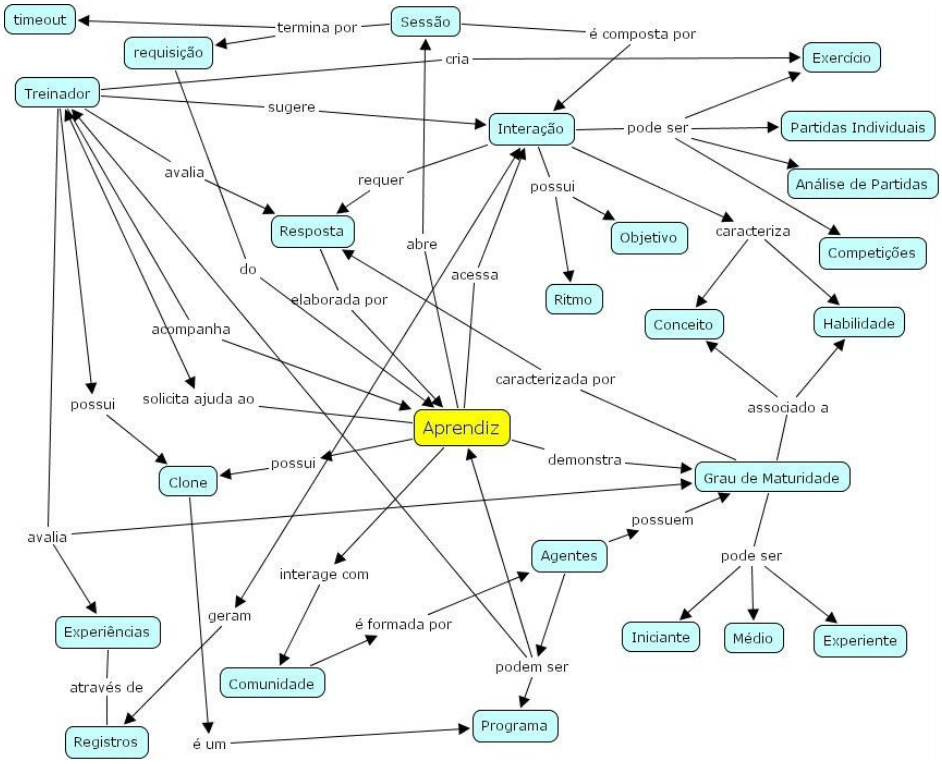


Figura 11: Ontologia de domínio da Aprendizagem em Xadrez – Netto (2005)

## 3.6 Metodologias de Modelagem

Neste tópico são apresentadas as ferramentas utilizadas na modelagem do protótipo criado nesta pesquisa. Pode-se enxergar um software como um produto gerado pelo uso de três ferramentas: Linguagem de programação, banco de dados e documentação. Atualmente é inegável a importância da documentação de sistemas, área estudada pelos pesquisadores da Engenharia de Software. Nos subtópicos 3.6.1. e 3.6.2. são descritos alguns recursos que facilitam esta documentação.

### 3.6.1 Unified Modeling Language – UML

Como a nomenclatura sugere, trata-se de uma linguagem de modelagem que, na computação, é amplamente utilizada para se modelar softwares orientados a objetos. É importante frisar que a UML é ampla e extensível, significando que ela não está restrita à modelagem de sistemas, podendo, inclusive, ser aplicada em outras áreas.

Melo (2002) lembra que são utilizados diversos diagramas que auxiliam na modelagem e documentação dos sistemas, dentre os quais merecem destaque:

- Diagrama de casos de uso: Expressa as funcionalidades e responsabilidades de cada ator dentro do sistema.
- Diagrama de classes: Booch (2000) define este diagrama como uma representação da estrutura e relações das classes que servem de modelo para objetos.
- Diagrama de seqüências: De acordo com Furlan (1998), este diagrama exibe as interações entre objetos organizadas em seqüência de tempo e de mensagens trocadas.

É válido ressaltar que os diagramas da UML atendem bem se o objetivo for modelar um sistema orientado a objetos. Porém, o protótipo construído nesta pesquisa adota outra abordagem, a Programação Orientada a Agentes. Com a possibilidade de se estender a linguagem UML, foram criados novos diagramas e outros foram alterados, dando origem a Agent Unified Modeling Language – AUML que é discutida no Tópico 3.6.2.

### 3.6.2 Agent Unified Modeling Language – AUML

Existe a necessidade de diagramas específicos para se modelar sistemas baseados em agentes. Neste sentido a UML foi estendida para a AUML, modificando diagramas já existentes ou incorporando nos diagramas e elementos.

Entretanto, como pode ser observado no site oficial da AUML - http://www.auml.org/, o esforço para manter esta linguagem pode ser considerado como estagnado no momento. Inicialmente, houve inúmeros trabalhos para se estender UML para a abordagem de agentes, porém, ocorreram algumas situações que merecem destaque:

1. UML 2.1 foi lançada, contemplando muitos dos recursos propostos na AUML;
2. O OMG Systems Modeling Language (SysML) – http://www.sysml.org, foi lançado. Trata-se de uma linguagem de modelagem de uso para aplicações de engenharia de sistemas. Suporta a especificação, análise, projeto, verificação e validação de uma ampla gama de aplicativos. É definida como uma extensão de um subconjunto da Unified Modeling Language (UML).

É importante frisar que os fatos citados anteriormente não interrompem o esforço de desenvolvimento da AUML que, apesar de não continuada, pode ser útil na modelagem de determinados cenários orientados a agentes.

## 3.7 Ferramentas Utilizadas na Implementação

Neste tópico são apresentadas as principais ferramentas utilizadas na implementação do protótipo criado nesta pesquisa. Vale frisar que o NetLogo foi utilizado para a construção do ambiente simulado, onde é possível observar os agentes se movimentando pelo cenário de resgate. Para tornar possível a interação entre usuário e o ambiente, os agentes foram escritos utilizando o Java Agent Development Framework – JADE. Estas duas tecnologias são discutidas nos subtópicos 3.7.1. e 3.7.2.

### 3.7.1 Java Agent Development Framework – JADE

De acordo com Bellifemine (2007), o JADE é uma plataforma desenvolvida na linguagem Java, contendo inúmeras classes e métodos que simplificam o desenvolvimento de Sistemas Multiagente, ou seja, trata-se de um framework amplamente documentado, que pode ser utilizado na implementação de agentes. A Figura 12 apresenta os seus componentes:

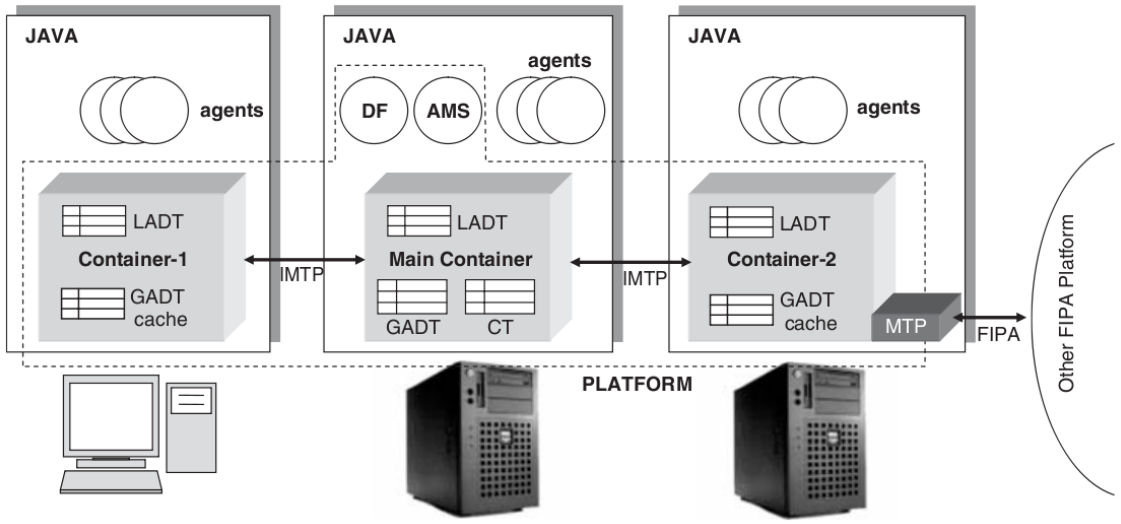


Figura 12: Relações entre os principais elementos da arquitetura JADE Bellifemine (2007)

A ferramenta também provê alguns recursos gráficos, que são úteis, dentre outras atividades, no monitoramento e depuração dos agentes. Alguns exemplos são o Remote Monitoring Agent – RMA, Sniffer, Instropector e Dummy, cujas funcionalidades são descritas a seguir:

- Remote Monitoring Agent – RMA: Permite controlar o ciclo de vida da plataforma de todos os agentes. A arquitetura distribuída do JADE também permite o controle de forma remota, tornando possível controlar a execução de agentes e de seu ciclo de vida de um host através da rede. Do ponto de vista da implementação e funcionamento do JADE, o RMA é um objeto Java e pode ser observado em execução na Figura 13. É possível iniciá-lo de duas formas:
  - ✓ A partir de uma linha de comando: **java jade.Boot rma1:jade.tools.rma.rma**; ou
  - ✓ Acrescentando o argumento –gui na inicialização do JADE: **Java jade.Boot –gui**.

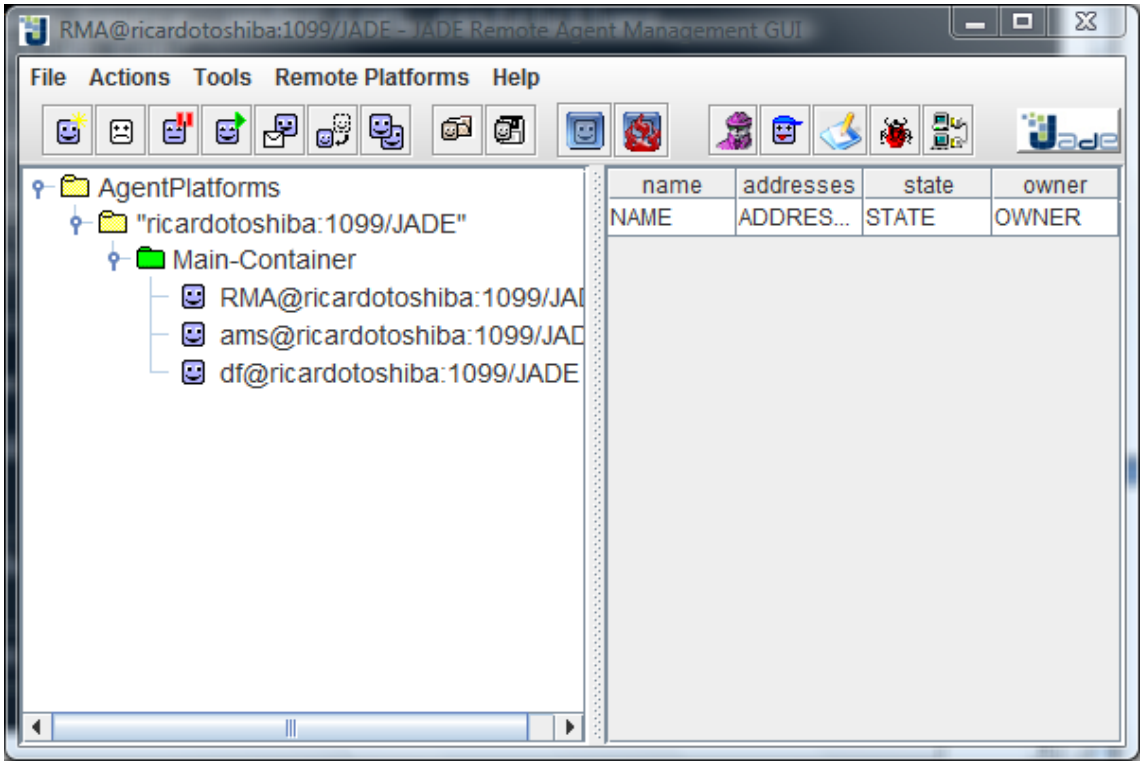


Figura 13: RMA em execução

- Sniffer Agent: Com o nome sugestivo, o Sniffer Agent é basicamente um agente FIPA-compliant com características de monitoramento. Quando o usuário decide monitorar um agente ou uma comunidade de agentes, cada

mensagem trocada é exibida na interface do Sniffer Agent. O usuário pode visualizar ou salvar estas mensagens, o que pode ser útil em análises posteriores. Uma tela de execução do Sniffer pode ser visualizada na Figura 33. Este agente pode ser iniciado de duas formas:

- ✓ A partir da barra de ferramentas do RMA; ou
- ✓ A partir da linha de comando **java jade.Boot sniffer:jade.tools.sniffer.Sniffer**.

- Instrospector Agent: Esta ferramenta permite monitorar e controlar o ciclo de vida de um agente e as mensagens trocadas. Ele também permite controlar a fila de comportamentos, permitindo executá-los passo a passo. O Introspector Agent pode ser inicializado de forma similar aos agentes Sniffer e Dummy, através da linha de comando **java jade.Boot introspector0:jade.tools.introspector.Introspector** ou pela da barra de ferramentas do RMA. A Figura 14 ilustra este agente sendo executado.

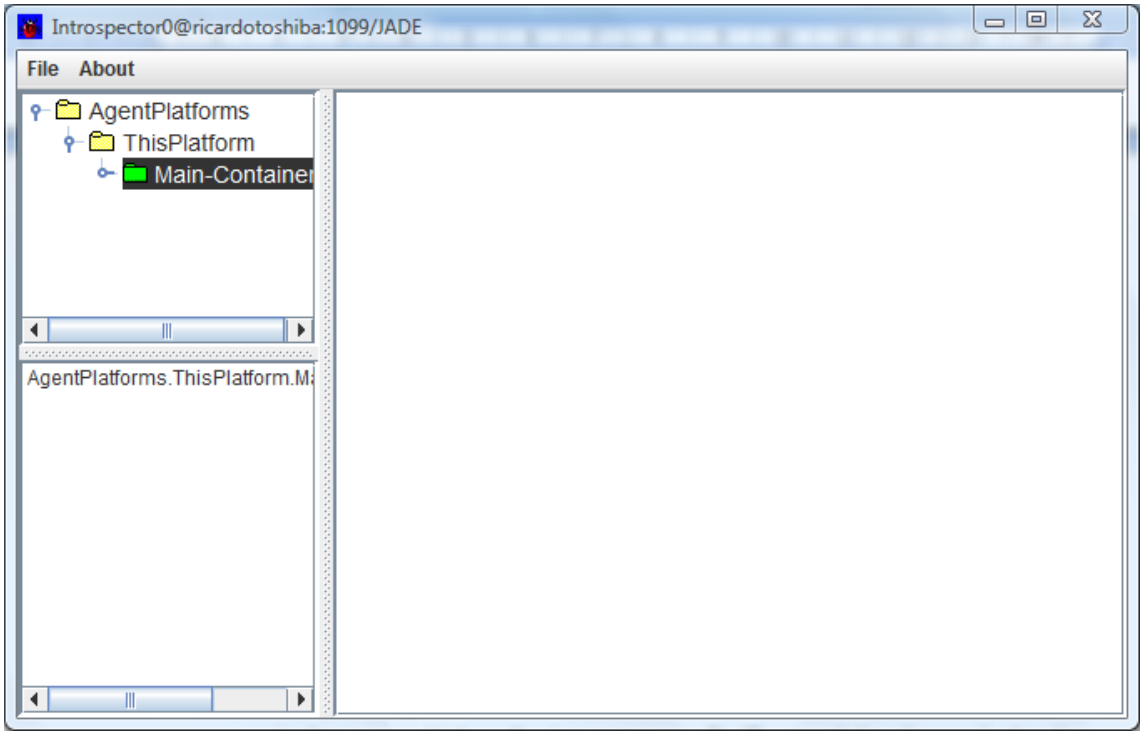


Figura 14: Introspector Agent sendo executado

- Dummy Agent: Esta ferramenta permite que os usuários interajam com os agentes JADE de forma personalizada. A interface permite compor e enviar mensagens ACL e mantém uma lista de todas as mensagens enviadas e

recebidas. Assim como no Sniffer, também é possível salvar a lista de mensagens. Uma tela de execução do Dummy pode ser visualizada na Figura 15. Novamente há duas formas de se executar este agente:

- ✓ A partir da barra de ferramentas do RMA; ou
- ✓ A partir da linha de comando **java jade.Boot dummy1:jade.tools.DummyAgent.DummyAgent**.

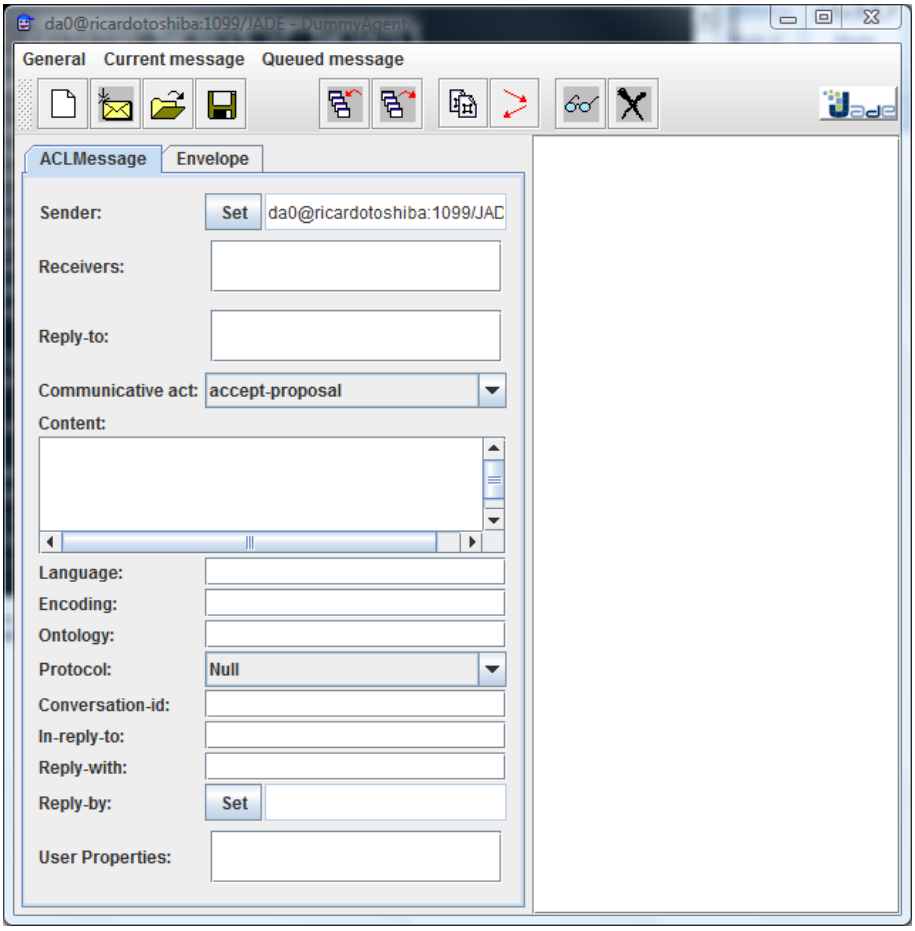


Figura 15: Dummy Agent sendo executado

É importante frisar que existem diferentes formas de executar os agentes implementados em JADE, dentre elas:

- Configurar o JADE e efetuar a compilação e execução através de linhas de comando; ou
- Conforme Câmara (2009b) propôs, integrar o JADE ao Eclipse e utilizar esta IDE como plataforma de desenvolvimento e execução dos agentes.

### 3.7.2 NetLogo

De acordo com as descrições contidas no NetLogo User Manual (2009), é definido como um ambiente de modelagem programável para simular fenômenos naturais e sociais. Foi escrito por Uri Wilensky em 1999 e, desde então, está em desenvolvimento contínuo no Center for Connected Learning and Computer-Based Modeling. As suas principais características são listadas abaixo.

- Adequado para a modelagem de sistemas complexos de desenvolvimento ao longo do tempo. Instruções podem ser enviadas para centenas ou milhares de agentes, todos operando de forma independente. Esta característica torna possível que se explore a conexão entre o comportamento de cada componente e os resultados que emergem da interação de muitos indivíduos.
- Permite que os alunos executem as simulações, explorando os mais variados comportamentos sob diversas condições. É também um ambiente de autoria, permitindo que estudantes, professores, pesquisadores e outros usuários criem seus próprios modelos.
- Dispõe de extensa documentação e tutoriais. Acompanha uma biblioteca de modelos, que é uma grande coleção de simulações que podem ser utilizadas e modificadas. Estas simulações abrangem diversas áreas do conhecimento, incluindo a biologia, medicina, física, química, matemática, ciência da computação, economia e psicologia social.
- Também pode ser utilizada como uma ferramenta participativa, através de um recurso de simulação chamado HubNet. Utilizando computadores interligados em rede ou dispositivos portáteis, tais como PDA’s e celulares, cada aluno pode assumir o controle de um agente numa simulação. Pelo aspecto participativo, esta característica é considerada de suma importância no escopo deste trabalho.

Algumas telas de execução do NetLogo podem ser observadas nas Figuras 32 e 36.

# 4 ANÁLISE E PROJETO DO SISTEMA

Neste capítulo são descritas as funcionalidades do software educacional implementado. São utilizados diagramas UML e AUML para documentar o sistema, como pode ser observado nos subtópicos seguintes.

## 4.1 Requisitos do Sistema

Para facilitar o aprendizado da Programação Orientada a Agentes, faz-se necessário criar um software educacional que utilize recursos visuais, mais especificamente de simulação, para ilustrar as características deste paradigma de programação.

Dividindo esta tarefa em duas etapas, pode-se destacar a necessidade do aluno entender a diferença da Programação Orientada a Objetos da Programação Orientada a Agentes. Nesta fase a simples simulação certamente é de grande importância, pois conceitos simples, como troca de mensagens e colaboração entre os agentes, podem ser ilustradas através de um ambiente simulado.

Após entender a diferença de programação entre os dois paradigmas e alguns conceitos básicos já citados, o aluno deve ser capaz de interagir com o ambiente, adicionando novos agentes, mudando a posição dos mesmos e até assumindo o papel de algum agente, disparando mensagens e observando o comportamento da comunidade de agentes.

Por fim, é interessante que este software educacional seja documentado e disponibilizado num ambiente virtual de aprendizagem, como o Moodle ou TelEduc, por exemplo.

Os subtópicos seguintes mapeiam os requisitos em diagramas UML e AUML.

## 4.2 Diagrama de Casos de Uso

A Figura 16 mostra o diagrama de casos de uso do sistema. Vale ressaltar que é utilizado o estereótipo <<agent>> para distinguir este tipo das classes do software educacional.

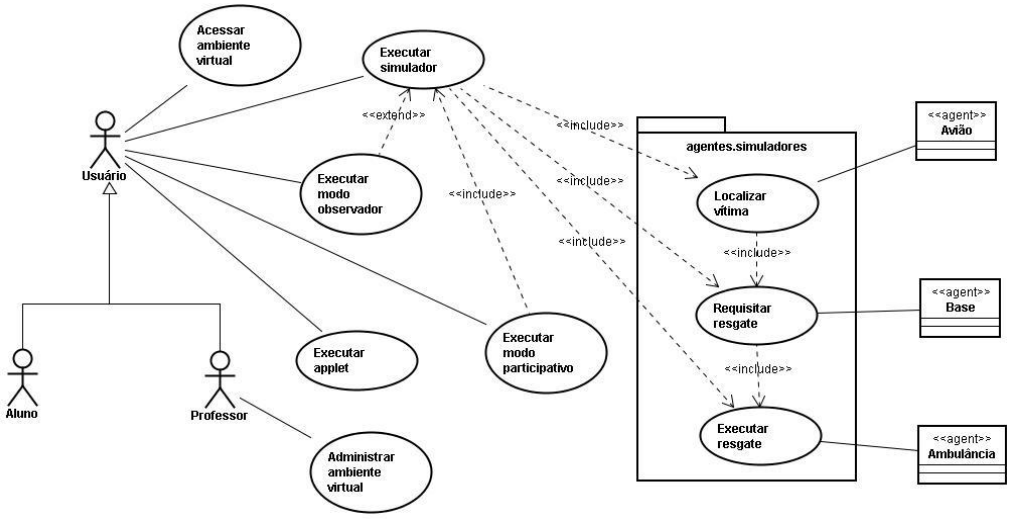


Figura 16: Diagrama de casos de uso do AVEPA

A seguir os casos de uso são documentados, obedecendo aos padrões da linguagem UML.

### 4.2.1 Caso de uso: Acessar ambiente virtual.

O usuário acessa o ambiente virtual que contém o software educacional, lições, instruções de utilização e outras funcionalidades.

Ator: Usuário.

Cenário principal

1. O usuário, através de um browser, acessa o ambiente virtual de aprendizagem.
2. O usuário informa as suas credenciais de acesso.

Cenário alternativo

Credenciais inválidas.

2.1. Se o login for inválido, comunicar o usuário e oferecer opção de recuperar senha por email.

### 4.2.2 Caso de uso: Executar simulador.

O usuário executa o ambiente simulado para visualizar os principais conceitos da Programação Orientada a Agentes, como a cooperação e troca de mensagens.

Ator: Usuário.

Cenário principal

1. O usuário configura se as mensagens serão exibidas na tela e escolhe o número de vítimas, de aviões e de ambulâncias.
2. O usuário executa o ambiente, onde é criada a comunidade de agentes que simula o cenário de resgate.
3. O usuário clica em executar resgate, iniciando a simulação.
4. Também é possível visualizar os códigos dos agentes (ambulância, avião e base).
5. A condição de parada da simulação é o resgate de todas as vítimas.

### 4.2.3 Caso de uso: Executar modo observador.

O usuário acessa o ambiente multiagente no modo observador, possibilitando visualizar a troca de mensagens no padrão FIPA-ACL entre os agentes. Pode ser estendido do caso de uso Executar Simulador.

Ator: Usuário.

Cenário principal

1. O usuário executa o ambiente multiagente no modo observador.
2. É possível visualizar as mensagens trocadas entre os agentes, seja em tempo real, integrado ao ambiente simulado, ou através da recuperação de cenários executados anteriormente.

Cenário alternativo

Cenário a recuperar não localizado.

2.1. Se não for possível recuperar o cenário a executar, encerrar o ambiente observador.

### 4.2.4 Caso de uso: Executar modo observador.

O usuário acessa o ambiente multiagente no modo participativo, possibilitando que se assuma o papel de algum agente e dispare as mensagens. Inclui o caso de uso Executar Simulador.

Ator: Usuário.

Cenário principal

1. O usuário executa o ambiente multiagente no modo participativo.
2. O usuário escolhe qual o papel irá assumir no cenário.
3. Ao assumir o papel de um agente, o usuário dispara uma mensagem ACL-FIPA para outro.
4. O caso de uso Executar Simulador é realizado.

Cenário alternativo

Mensagem não entendida.

3.1. Se a mensagem disparada pelo aluno não for entendida, retornar NOT-UNDERSTOOD, informar qual item não foi entendido e mostrar uma mensagem correta como exemplo.

### 4.2.5 Caso de uso: Executar applet.

O usuário executa o ambiente simulado, através de um applet, para visualizar os principais conceitos da Programação Orientada a Agentes, como a cooperação e troca de mensagens.

Ator: Usuário.

Cenário principal

1. O usuário configura se as mensagens serão exibidas na tela e escolhe o número de vítimas, de aviões e de ambulâncias.
2. O usuário executa o ambiente, onde é criada a comunidade de agentes que simula o cenário de resgate.
3. O usuário clica em executar resgate, iniciando a simulação.
4. Também é possível visualizar os códigos dos agentes (ambulância, avião e base).
5. A condição de parada da simulação é o resgate de todas as vítimas.

### 4.2.6 Caso de uso: Administrar ambiente virtual.

Cenário principal

1. O professor, através de um browser, acessa o ambiente virtual de aprendizagem.

2. O professor informa as suas credenciais de acesso.

3. O professor em ativar edição. A partir desta funcionalidade é possível disponibilizar arquivos para downloads, criar, alterar ou excluir lições, instruções de utilização e outras funcionalidades.

Cenário alternativo

Credenciais inválidas.

2.1. Se o login for inválido, comunicar o professor e oferecer opção de recuperar senha por email.

### 4.2.7 Caso de uso: Localizar vítima.

O avião se move pelo cenário procurando as vítimas.

Agente: Avião.

Cenário principal

1. O avião se movimenta aleatoriamente pelo ambiente, tentando localizar as vítimas.

2. Ao passar sobre uma vítima, destacá-la e enviar uma mensagem ACL-FIPA para a base, informando a posição da mesma.

### 4.2.8 Caso de uso: Requisitar resgate.

A base requisita resgates a serem efetuados.

Agente: Base.

Cenário principal

1. Ao receber a informação da localização de uma vítima, a base envia uma mensagem ACL-FIPA para a ambulância, requisitando que o resgate seja executado.

### 4.2.9 Caso de uso: Executar resgate.

A ambulância executa os resgates solicitados.

Agente: Ambulância.

Cenário principal

1. Ao receber a requisição de resgate, a ambulância executa a ação solicitada.

## 4.3 Diagrama de Classes

A Figura 17 mostra o diagrama de classes do sistema.

Apesar de ser um software educacional em parte desenvolvido sob o paradigma de agentes, a interface inicial, que executa as chamadas ao Sistema Multiagente, foi implementada utilizando recursos de orientação a objetos, o que justifica a utilização do diagrama de classes na modelagem do sistema.

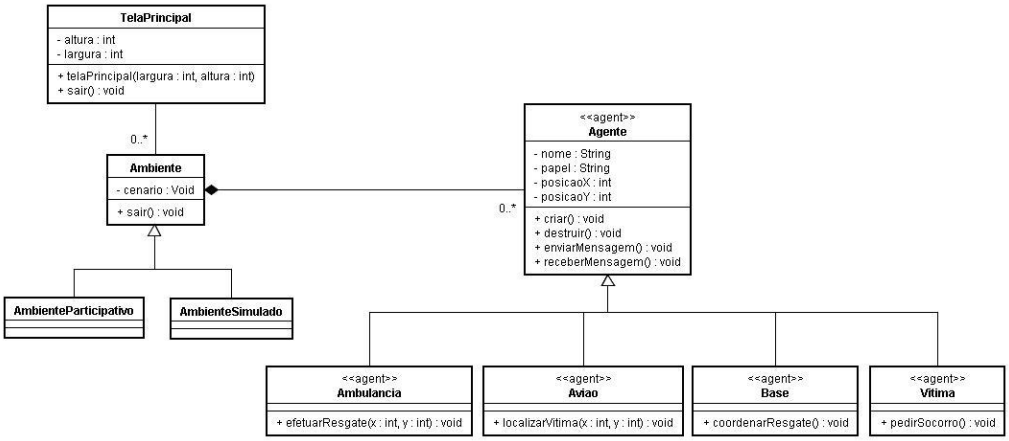


Figura 17: Diagrama de classes do AVEPA

A seguir são descritas as classes modeladas, bem como seus atributos e métodos.

- Classe TelaPrincipal: Tela inicial do sistema, onde o usuário pode efetuar chamadas a outros módulos, como o ambiente simulado ou multiagente.

Tabela 5: Atributos da classe TelaPrincipal

| ATRIBUTO | VISIBILIDADE | TIPO | DESCRIÇÃO |
|---|---|---|---|
| altura | Privado | Int | Altura da tela. |
| largura | Privado | Int | Altura da tela. |

Tabela 6: Métodos da classe TelaPrincipal

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| telaPrincipal(largura: int, altura: int) | Público | Construtor da classe TelaPrincipal. |
| sair(): void | Público | Fecha a tela. |

- Classe Ambiente: Generaliza os ambientes utilizados, participativo ou simulado.

Tabela 7: Atributos da classe Ambiente

| ATRIBUTO | VISIBILIDADE | TIPO | DESCRIÇÃO |
|---|---|---|---|
| cenario | Privado | void | Cenário a ser executado. |

Tabela 8: Métodos da classe Ambiente

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| sair(): void | Público | Fecha a tela. |

- Classe AmbienteParticipativo: Especialização da classe Ambiente. Possibilita que os usuários interajam com a comunidade de agentes.
- Classe AmbienteSimulado: Especialização da classe Ambiente. Possibilita que os usuários observem os comportamentos dos agentes que compõem o ambiente.
- Classe Agente: Subclasse de Agent. Também é modelada utilizando composição, ou seja, no contexto deste trabalho é parte da classe Ambiente. Representa os agentes que compõem o software educacional proposto.

Tabela 9: Atributos da classe Agente

| ATRIBUTO | VISIBILIDADE | TIPO | DESCRIÇÃO |
|---|---|---|---|
| nome | Privado | String | Nome do agente. |
| Papel | Privado | String | Papel que desempenha no ambiente. |
| posicaoX | Privado | int | Posição do agente no eixo x ao ser instanciado. |
| posicaoY | Privado | int | Posição do agente no eixo y ao ser instanciado. |

Tabela 10: Métodos da classe Agente

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| criar(): void | Público | Instanciar o agente no cenário. |
| destruir(): void | Público | Eliminar o agente do cenário. |
| enviarMensagem(): void | Público | Enviar mensagem para outros agentes. |
| receberMensagem(): void | Público | Receber mensagem de outros agentes. |

- Agente Ambulancia: Herda as características de Agente. Utilizado nos ambientes simulado e multiagente.

Tabela 11: Métodos do agente Ambulancia

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| efetuarResgate(x: int, y: int): void | Público | Efetua resgates numa determinada posição xy. |

- Agente Aviao: Herda as características de Agente. Utilizado nos ambientes simulado e multiagente.

Tabela 12: Métodos do agente Aviao

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| localizarVitima(x: int, y: int): void | Público | Movimenta-se aleatoriamente pelo cenário em busca de vítimas a serem resgatadas. |

- Agente Base: Herda as características de Agente. Utilizado para monitorar os resgates efetuados.

Tabela 13: Métodos do agente Base

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| coordenarResgate(): void | Público | Monitora os resgates efetuados. |

- Agente Vitima: Herda as características de Agente. Utilizado nos ambientes simulado e multiagente.

Tabela 14: Métodos do agente Vitima

| MÉTODO | VISIBILIDADE | DESCRIÇÃO |
|---|---|---|
| pedirSocorro(): void | Público | Envia mensagens de socorro para que seja efetuado o resgate. |

## 4.4 Diagramas de Seqüência

As Figuras 18 a 21 mostram os principais diagramas de seqüência do sistema. Por se tratar de um protótipo, optou-se por modelar e documentar as rotinas principais do sistema.

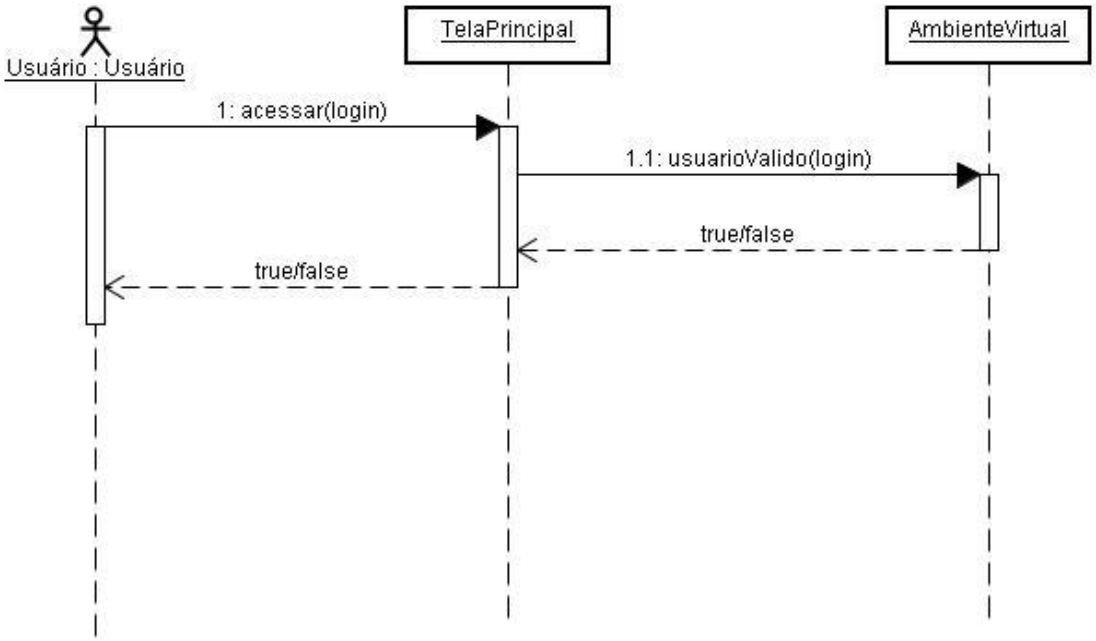


Figura 18: Diagrama de seqüência para acessar o ambiente virtual de aprendizagem

A Figura 18 exibe os passos necessários para que o usuário acesse o ambiente virtual de aprendizagem. Como pode ser observado, trata-se de uma rotina de autenticação.

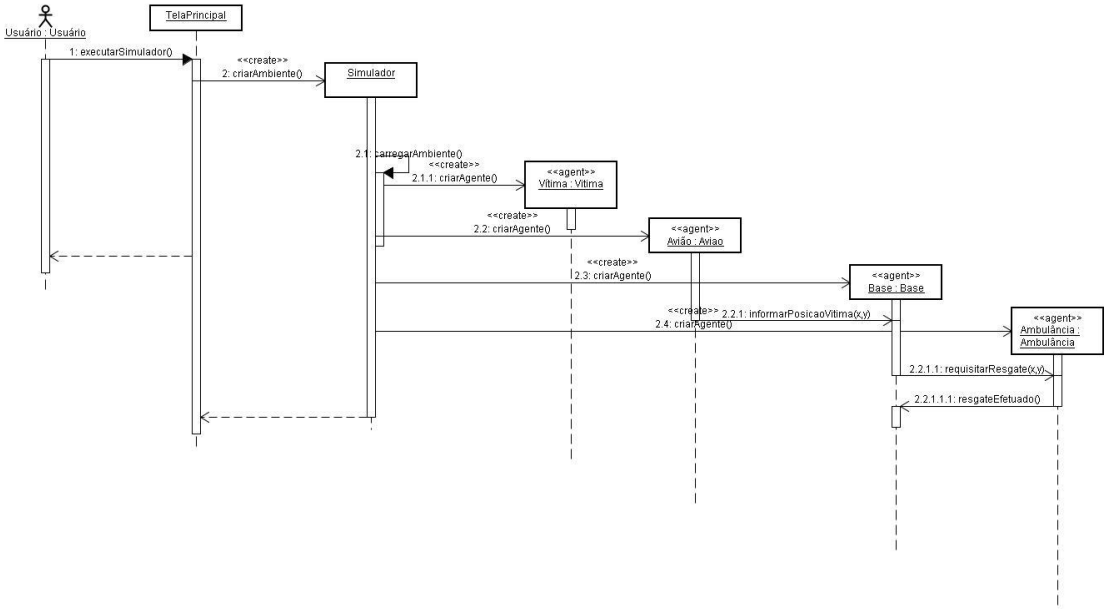


Figura 19: Diagrama de seqüência para executar o ambiente simulado

A Figura 19 exibe os passos necessários para que o usuário execute o ambiente simulado. Ao disparar a mensagem executarSimulador(), o objeto Simulador é criado. Em seguida a comunidade de agentes também é criada e é possível observar as mensagens trocadas entre os mesmos.

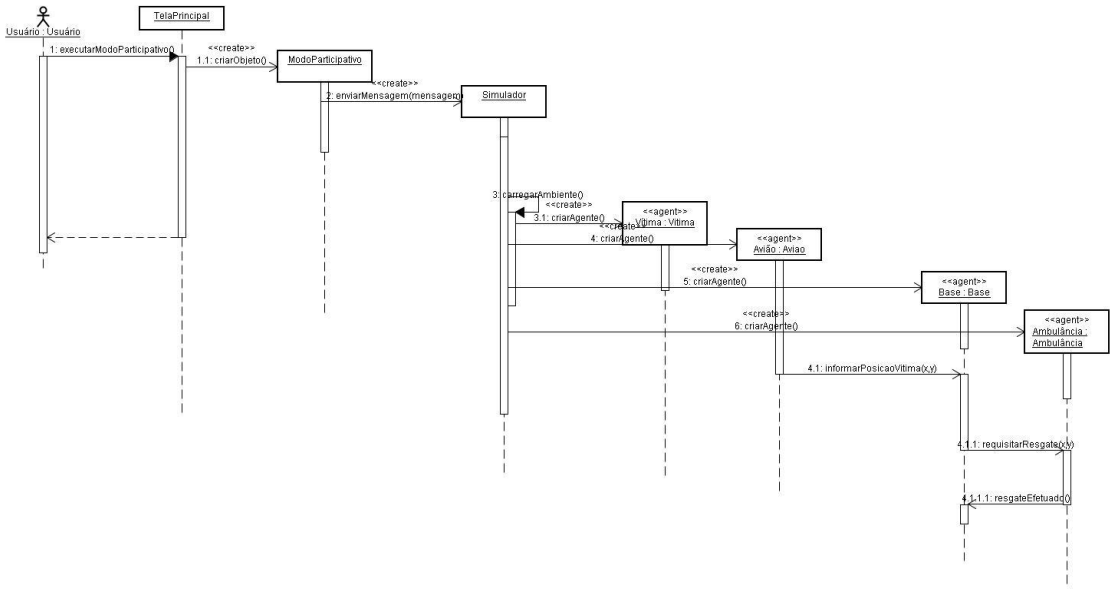


Figura 20: Diagrama de seqüência para executar o ambiente no modo participativo

A Figura 20 exibe os passos necessários para que o usuário execute o ambiente no modo participativo. Ao disparar a mensagem executarModoParticipativo(), o objeto

ModoParticipativo é criado e uma interface é utilizada para que o usuário possa interagir com o cenário, enviando e recebendo mensagens ACL-FIPA. Ao enviar a mensagem, pode-se observar o comportamento da comunidade de agentes através do ambiente simulado.

A Figura 21 ilustra a troca de mensagens entre os agentes para que um resgate seja executado, tornando possível observar a cooperação do ponto de vista da abordagem de agentes.

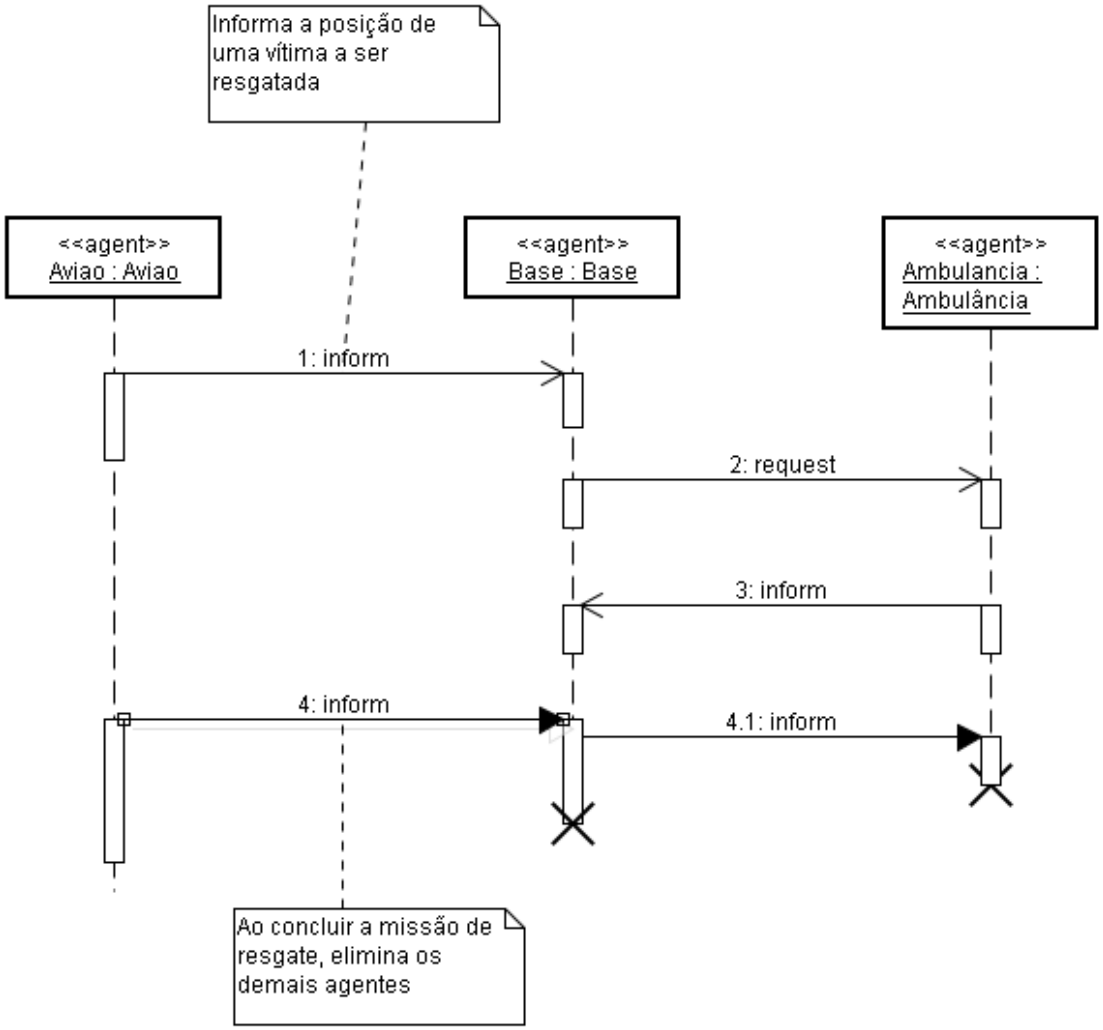


Figura 21: Troca de mensagens entre agentes que cooperam para executar um resgate

Ao avistar uma vítima no cenário, o Avião envia um inform para a Base. Esta, por sua vez, envia um request para que a Ambulância efetue o resgate. Ao concluir a missão, o Avião envia um inform encerrando os demais agentes do ambiente.

## 4.5 Diagrama de Comunicação

A Figura 22 mostra o diagrama de comunicação em alto nível do ambiente.

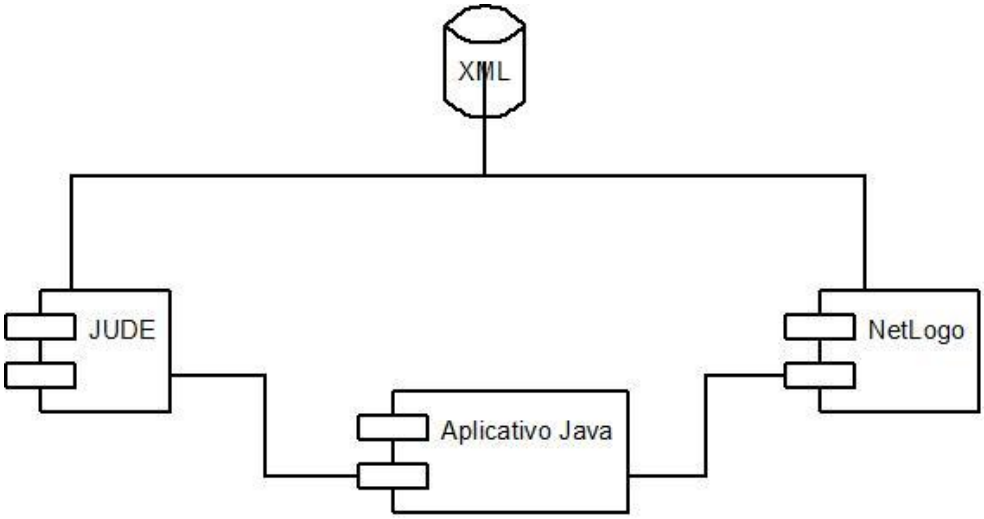


Figura 22: Diagrama de comunicação do AVEPA

De forma geral, existem três componentes principais, o ambiente multiagente, representado pelo JUDE, o ambiente simulado, representado pelo NetLogo, e o aplicativo principal. A comunicação entre os ambientes e a recuperação e armazenamento de cenários é feita através de um arquivo XML. Os detalhes sobre esta integração são discutidos no Capítulo 5.

## 4.6 Diagrama de Estados

As Figuras 23 e 24 mostram os diagramas de estados do sistema, onde é possível observar, por exemplo, a mudança dos estados da vítima até que o resgate seja efetuado.

A utilização deste diagrama na modelagem e documentação é considerada importante, pois alguns agentes mudam de estado no decorrer da execução, como a Vítima, de aguardando resgate para resgatada, por exemplo.

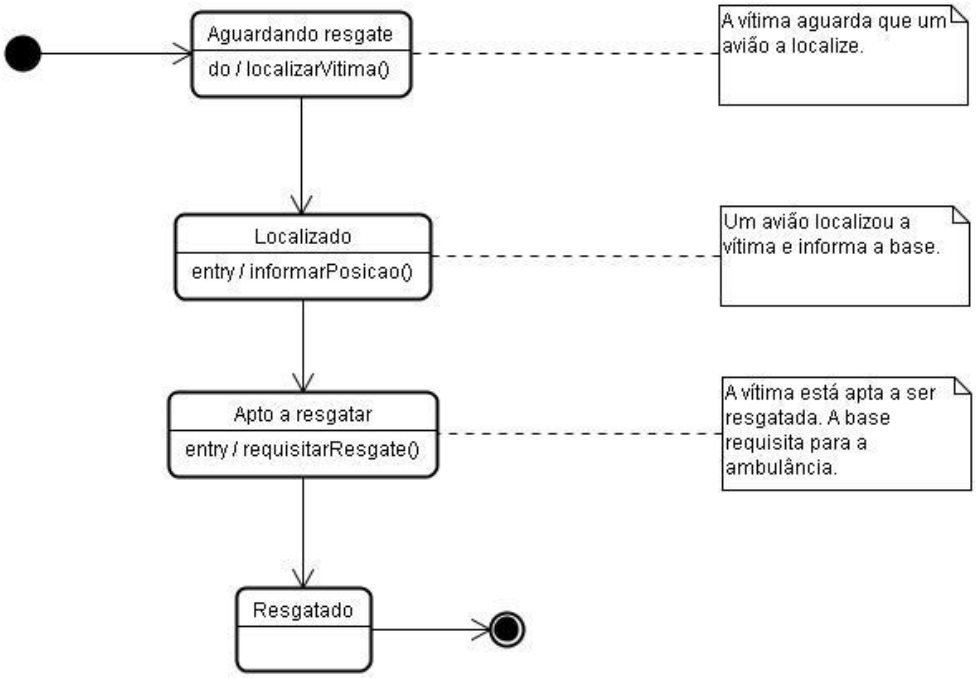


Figura 23: Diagrama de estados para executar o resgate

A Figura 24 ilustra os possíveis estados do agente Ambulância no cenário do resgate.

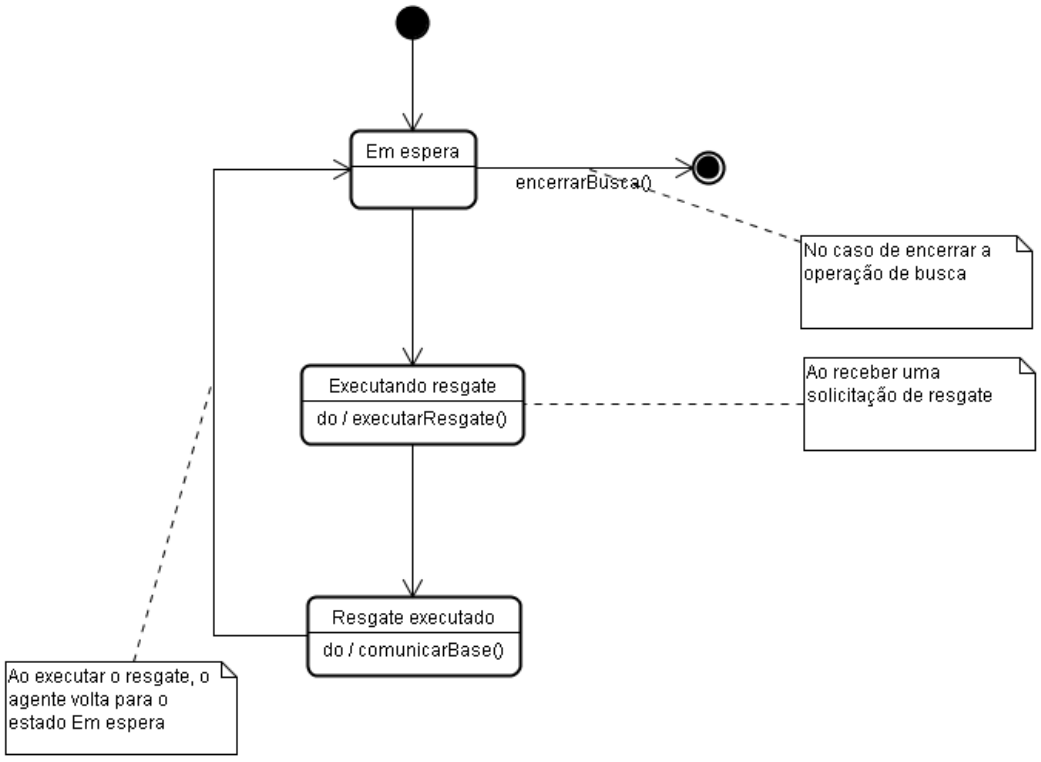


Figura 24: Diagrama de estados do agente Ambulância

Inicialmente a Ambulância se encontra no estado de espera, aguardando uma requisição de resgate. Ao ser efetuada a requisição, o estado é alterado para executando resgate. Ao executar o resgate, a Base é comunicada e a Ambulância retorna para o estado em espera, aguardando novas requisições ou que a busca seja encerrada.

## 4.7 Diagrama de Atividades

A Figura 25 mostra o diagrama de atividades do sistema, onde é possível visualizar o fluxo dos eventos até que o resgate seja efetuado.

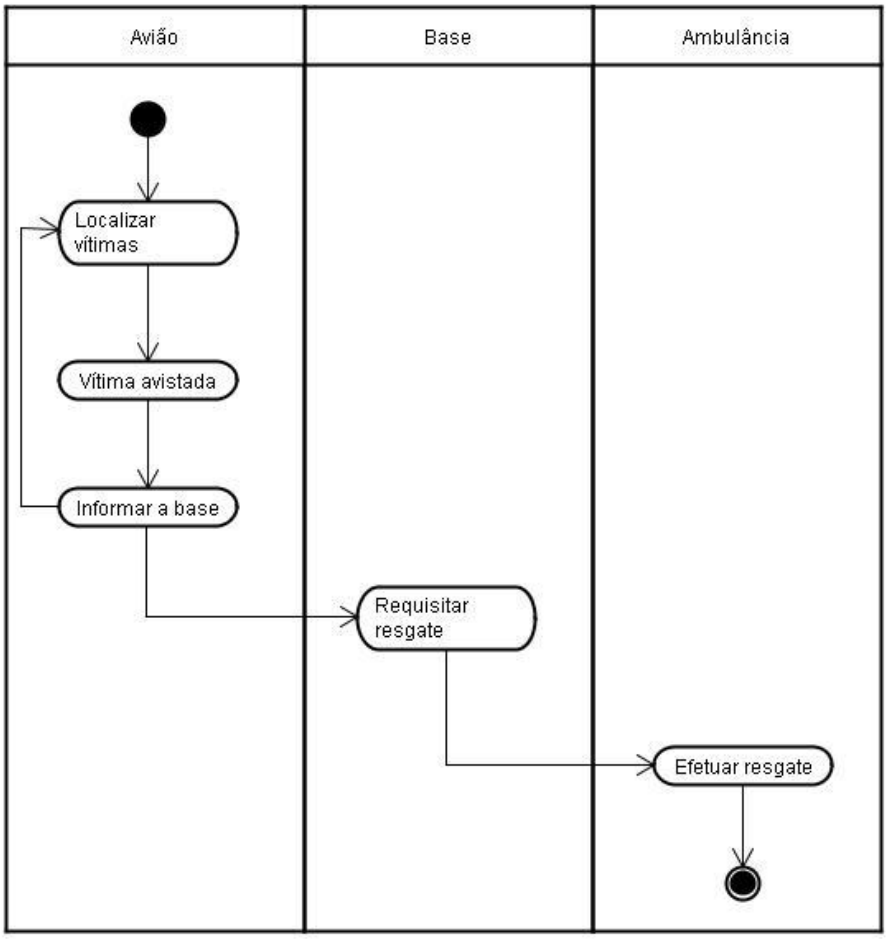


Figura 25: Diagrama de atividades para executar o resgate

## 4.8 Diagrama de Componentes

A Figura 26 mostra o diagrama de componentes, onde é possível visualizar, de forma geral, os itens que compõem o AVEPA.

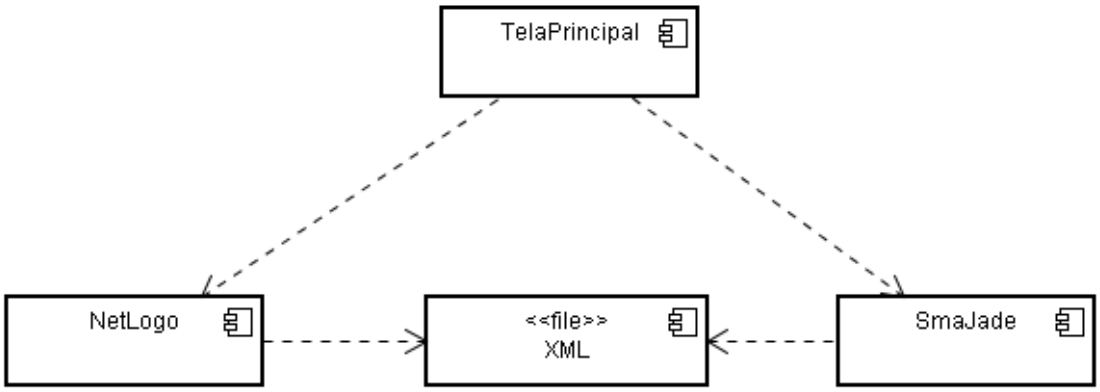


Figura 26: Diagrama de componentes do AVEPA

## 4.9 Diagrama de Implantação

A Figura 27 mostra o diagrama de implantação, onde é possível visualizar a estrutura física necessária para que o protótipo seja executado. Faz-se necessário, além dos módulos que compõem o sistema, da máquina virtual Java instalada.

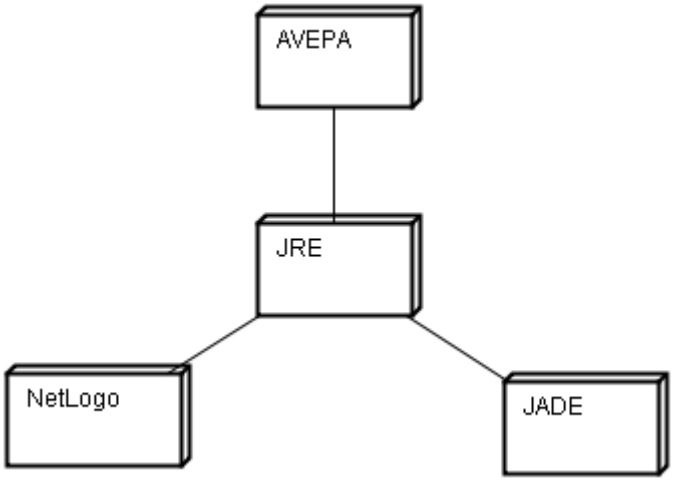


Figura 27: Diagrama de implantação do AVEPA

## 4.10 Arquitetura Proposta

Neste tópico é descrita a arquitetura geral e interna do AVEPA, sendo ilustradas nas Figuras 28 e 29.

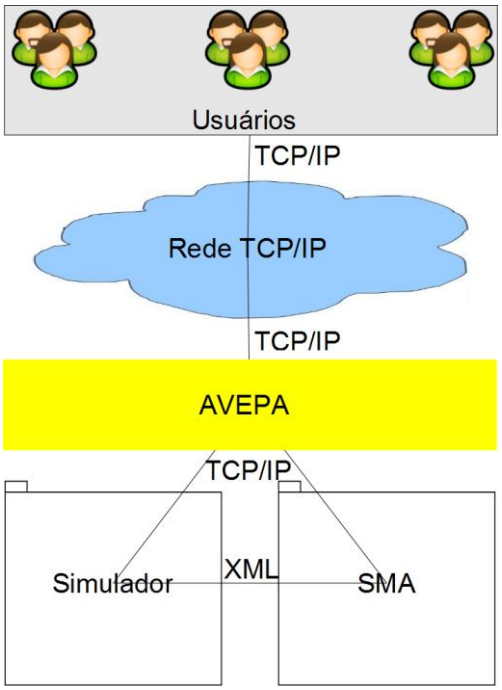


Figura 28: Visão geral da arquitetura do AVEPA

Analisando a arquitetura geral do sistema, observa-se que os usuários interagem com o aplicativo através de uma interface principal, implementada em Java. O ambiente é composto por dois módulos principais interligados, um Simulador e o Sistema Multiagente. Estes módulos são capazes de armazenar os eventos num arquivo XML, o que possibilita, dentre outras funcionalidades, a recuperação de cenários executados para análises posteriores.

A Figura 29 mostra a arquitetura interna do sistema.

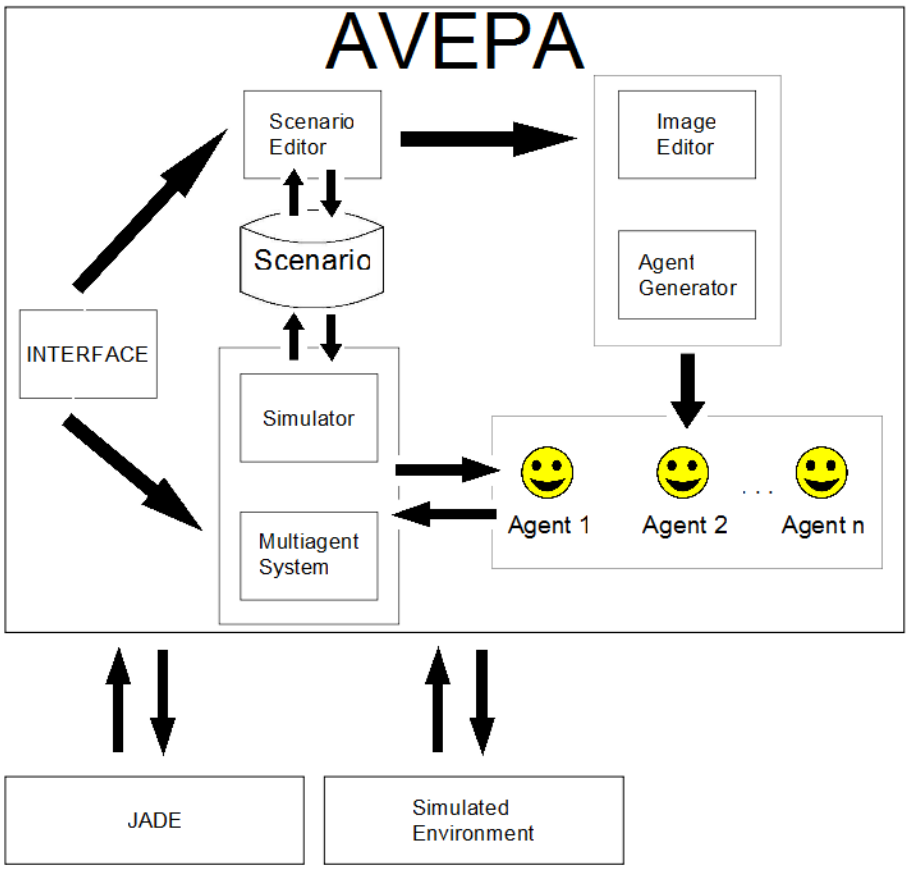


Figura 29: Arquitetura interna do AVEPA

Como pode ser observado na Figura 29, o software foi projetado de forma modular, integrando algumas tecnologias já disponíveis, como o JADE e NetLogo. Claramente ocorre a interação dos usuários com o sistema, que aciona os módulos de Simulação e Multiagente, armazenando as ações numa base de dados ou num arquivo XML.

# 5 IMPLEMENTAÇÃO E TESTES

Após a análise e projeto do sistema, é possível executar o desenvolvimento do software proposto. É válido ressaltar que a ferramenta desenvolvida nesta pesquisa trata-se de um protótipo com finalidades experimentais, com as execuções descritas no fim deste capítulo, mais especificamente na seção que aborda os testes.

Este capítulo descreve os padrões, técnicas e tecnologias utilizadas na implementação e testes da ferramenta. Também são descritos os requisitos para que o software seja executado.

## 5.1 Requisitos de Software

Existem duas versões implementadas, uma onde é possível executar através de um Applet Java, na qual se faz necessário somente um navegador Web, como o Mozilla Firefox ou o Internet Explorer, por exemplo, e a Java Runtime Environment – JRE. Entretanto, este ambiente é mais limitado, possibilitando apenas a visualização do cenário simulado. A Figura 30 ilustra uma tela de execução.


Figura 30: Tela de execução do Applet Java no Google Chrome

A outra versão traz o ambiente completo, ou seja, é possível executar o cenário simulado e os agentes em JADE, possibilitando, inclusive, que o aluno assuma o lugar de um agente e envie mensagens para os outros, observando suas reações e comportamentos. As Figuras 31, 32 e 33 ilustram algumas telas de execução do sistema.

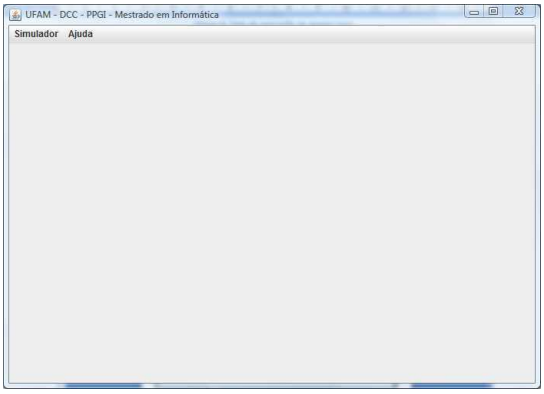


Figura 31: Tela principal do sistema

A Figura 31 exibe a tela principal do sistema, onde, através dos menus, é possível executar o ambiente simulado ou visualizar os agentes implementados em JADE trocando mensagens.

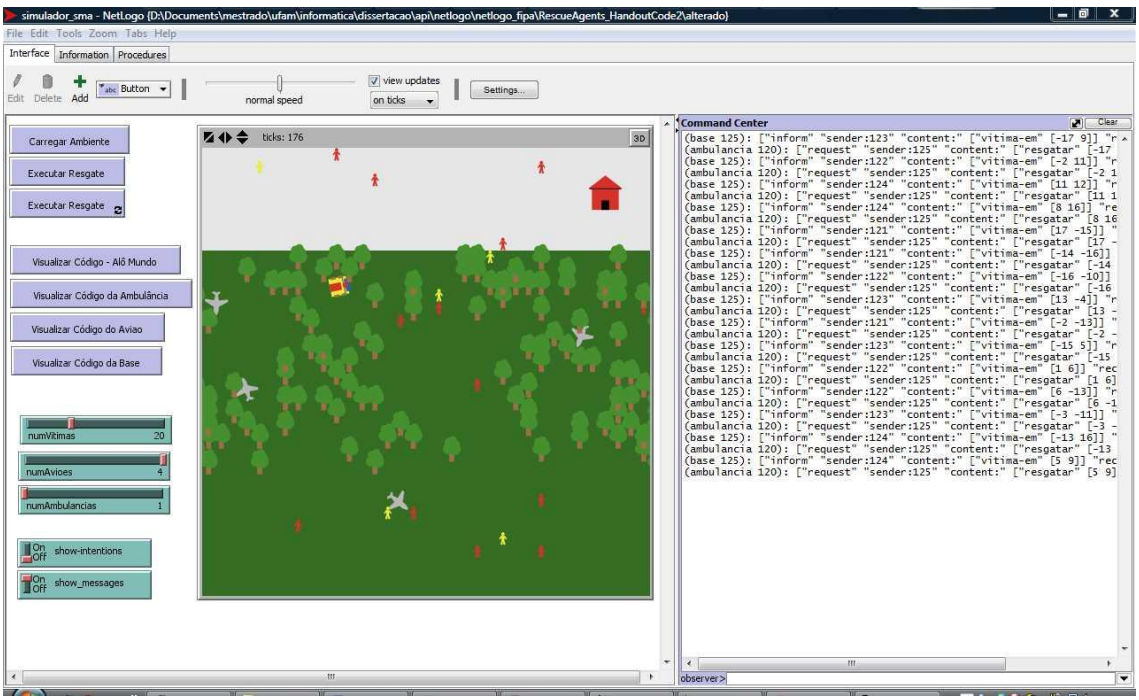


Figura 32: Tela de execução do ambiente simulado

A Figura 32 mostra uma tela de execução utilizando o NetLogo, onde é possível observar os agentes trocando mensagens sobre a posição das vítimas, bem como a movimentação dos mesmos pelo ambiente.

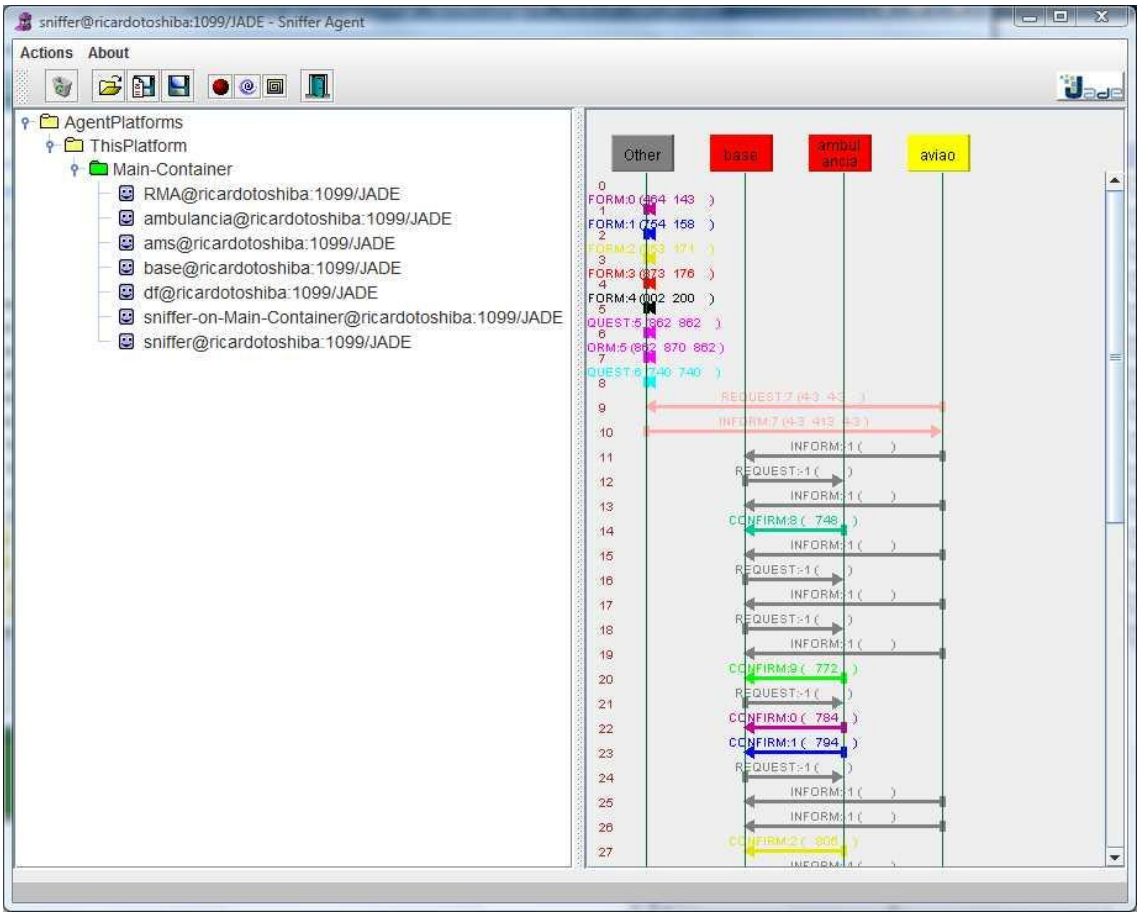


Figura 33: Visualizando a troca de mensagens dos agentes escritos em JADE

Os eventos ocorridos no ambiente simulado estão sincronizados com o JADE, isto significa que, ao localizar uma vítima, por exemplo, as mensagens enviadas pelo Avião podem ser visualizadas nos dois ambientes, JADE e NetLogo. A cada simulação o NetLogo salva um arquivo .xml contendo informações como as localizações das Vítimas. Isto possibilita que os ambientes sejam recuperados para análises posteriores, por exemplo.

A Figura 33 mostra os agentes implementados em JADE trocando mensagens. Por se tratar de um protótipo, optou-se por utilizar o Sniffer Agent, um recurso do próprio framework que permite a visualização gráfica das mensagens trocadas.

Na implementação do aplicativo foram utilizadas as ferramentas listadas a seguir.

- ✓ Java Development Kit;
- ✓ Eclipse SDK;

- ✓ JADE 3.7;

NetLogo 4.1.

É válido ressaltar que todas estas ferramentas são livres e, algumas, de código aberto.

Para executar o aplicativo, é necessário que os seguintes softwares estejam instalados:

- ✓ Máquina Virtual Java 1.5 ou mais recente;
- ✓ JADE 3.7 – http://jade.tilab.com/;
- ✓ NetLogo 4.1. – http://ccl.northwestern.edu/netlogo/.

Os detalhes de instalação dos softwares relacionados podem ser obtidos nos sites de cada ferramenta ou em grupos de discussão na Web.

## 5.2 Descrição dos Agentes

O ambiente proposto contém quatro agentes, que são descritos nos subtópicos seguintes.

### 5.2.1 Ambulância

Responsável por efetuar os resgates, quando estes são solicitados pela Base. No caso de vários resgates serem solicitados simultaneamente, a Ambulância enfileira os mesmos para, em seguida, executá-los de forma seqüencial.

### 5.2.2 Avião

Desempenha um papel mais ativo, movimentado-se de forma aleatória pelo ambiente, com o objetivo de localizar vítimas. Ao atingir este objetivo, envia uma mensagem para a Base, informando as coordenadas para que o resgate seja executado. O código em JADE do agente Avião pode ser observado na Tabela 15.

Tabela 15: Código em JADE do agente Avião

```
package dissertacao.agentes.teste;
```

```
public class Aviao extends Agent {
        protected void setup(){
                addBehaviour(new CyclicBehaviour(this) {
                        public void action() {
                                //recebendo mensagem
                                ACLMessage msg= receive();
                                if (msg!=null)
                                        System.out.println("Agente " +
myAgent.getLocalName() + " recebendo mensagem " + msg.getContent());
                                block();
                        }
                });

                //objetos e variaveis utiliazadas na leitura do arquivo .xml gravado
pelo NetLogo
                LerArquivo teste = new LerArquivo();
                int i = 0;

                //envia mensagens para "a1" e "a2"
                while (!teste.lerArquivo(i).equals("</vitima>")) {
                        //não há linha a ser lida
                        if (teste.lerArquivo(i).equals("nulo")){
                                //System.out.println("Retornou nulo!!");
                                i--;
                        }

                        //encontra o fim do .xml
                        else if (teste.lerArquivo(i).equals("</vitima>")){
                                System.out.println("Fim do resgate");
                                doDelete();
                        }

                        //foi lida uma tag <>
                        else if (teste.lerArquivo(i).startsWith("<") ||
```

```
teste.lerArquivo(i).startsWith("version")                                    ||
teste.lerArquivo(i).startsWith("encoding")){
                        System.out.println("Tag XML");
                }

                //vitima localizada, envia uma mensagem (inform) para a
base
                else {
                        ACLMessage msg = new
ACLMessage(ACLMessage.INFORM);
                        msg.setContent("vitimaEm(" + teste.lerArquivo(i) +
")");
                        msg.addReceiver(new AID( "ufam_base",
AID.ISLOCALNAME));

                        send(msg);
                        }

                        i++;
                }
        }
}
```

### 5.2.3 Base

Responsável por receber as informações sobre as localizações das Vítimas e solicitar os resgates para a Ambulância. O código em JADE do agente Base pode ser observado na Tabela 16.

Tabela 16: Código em JADE do agente Base

```
package dissertacao.agentes.teste;

import jade.core.AID;
import jade.core.Agent;
```

```
import jade.core.behaviours.*;
import jade.lang.acl.*;

public class Base extends Agent {
   protected void setup() {
                addBehaviour(new CyclicBehaviour(this) {
                        public void action() {
                                ACLMessage msg = receive();

                                //trata o recebimento da mensagem e envia um request para que a ambulância efetue o resgate
                                if (msg!=null) {
                                        System.out.println(myAgent.getLocalName() + " - mensagem recebida - " + msg.getContent());

                                        //envio do request para a ambulancia
                                        ACLMessage encaminhar = new ACLMessage(ACLMessage.INFORM);

        encaminhar.setPerformative(ACLMessage.REQUEST);
                                        encaminhar.setContent(msg.getContent());
                                        encaminhar.addReceiver(new AID("ufam_ambulancia", AID.ISLOCALNAME));
                                        send(encaminhar);
                                }
                                block();
                        }
                });
        }
}
```

### 5.2.4 Vítima

Utilizada no ambiente simulado e multiagente, porém, não tem papel ativo em nenhum dos cenários, tendo em vista que fica somente aguardando ser localizada, ou seja, não há troca de mensagens com os demais agentes.

## 5.3 Padrões Adotados

Neste tópico são listados os padrões adotados para a implementação dos componentes.

- ✓ Arquivos .java da camada de apresentação: Representam os componentes que dão acesso às principais funcionalidades do protótipo. A tela principal do aplicativo pode ser citada como exemplo.
- ✓ Arquivos .java da camada de aplicação: Representam os agentes implementados em JADE.
- ✓ Arquivos .nlogo da camada de aplicação: Representam os agentes implementados em NetLogo, ou seja, o ambiente simulado.
- ✓ Arquivos .xml da camada de aplicação: Representam as posições dos agentes no cenário, possibilitando a integração entre o JADE e o NetLogo, bem como a recuperação de cenários para análises futuras.
- ✓ Arquivos .nls da camada de aplicação: Encapsulam os métodos de troca de mensagens e alguns comportamentos dos agentes no ambiente simulado.
- ✓ Arquivos .bat: Representam os scripts que são executados da camada de apresentação, no caso do aplicativo ser executado sobre o MS-Windows.
- ✓ Arquivos .sh: Representam os scripts que são executados da camada de apresentação, no caso do aplicativo ser executado sobre o Linux.

## 5.4 Execução do Sistema

Nesta seção podem ser observadas as principais funcionalidades do protótipo em execução.

A Tabela 17 mostra um algoritmo em alto nível da execução do AVEPA no modo observador.

Tabela 17: Algoritmo para a execução do modo observador

```
modoObservador()
        Carregar a interface do AVEPA;
        Executar o ambiente simulado;
        Customizar a quantidade de agentes no cenário;
        Posicionar agentes no ambiente de forma aleatória;
        Iniciar a simulação;
        Enquanto existirem vítimas faça
                O avião se movimenta aleatoriamente pelo cenário;
                Ao localizar uma vítima, o avião envia um inform para a base
                contendo a posição da mesma;
                Ao receber o inform, a base envia um request para que a ambulância
                efetue o resgate;
        Fim enquanto;
Fim.
```

Na tabela 18 pode ser observado o algoritmo para a execução do AVEPA no modo participativo.

Tabela 18: Algoritmo para a execução do modo participativo

```
modoParticipativo()
        Carregar a interface do AVEPA;
        Executar o ambiente participativo;
        Escolher o agente que irá assumir o papel;
        Posicionar agentes no ambiente de forma aleatória;
        Enviar mensagem ACL;
        Iniciar a simulação;
        Enquanto existirem vítimas faça
                O avião se movimenta aleatoriamente pelo cenário;
                Ao localizar uma vítima, o avião envia um inform para a base
                contendo a posição da mesma;
                Ao receber o inform, a base envia um request para que a ambulância
                efetue o resgate;
        Fim enquanto;
```

| Fim. |
|---|

O sistema consiste numa comunidade de agentes que trocam mensagens, cooperando entre si, a fim de resgatar a(s) vítima(s). Esta troca de mensagens e o comportamento dos agentes podem ser observados através do ambiente simulado ou dos agentes em JADE, neste último caso, utilizando o Sniffer. Vale ressaltar que os ambientes, JADE e NetLogo, estão sincronizados em tempo real, ou seja, todos os eventos são replicados de forma fiel ao que está sendo observado no ambiente simulado. É possível, também, assumir o papel de um dos agentes e disparar as mensagens no ambiente, possibilitando que o estudante observe o comportamento dos demais agentes ao receberem uma mensagem específica.

### 5.4.1 Executando o Applet

Ao acionar o menu SIMULADOR – EXECUTAR APPLET, o usuário executa o ambiente simulado através de um Applet Java, implementado em NetLogo. A Figura 30 mostra uma tela de execução desta funcionalidade.

O ambiente é composto por 7 buttons, 3 sliders e 2 switches, contendo as seguintes funcionalidades:

- ✓ Button Carregar Ambiente: Carrega o cenário escolhido para implementação do protótipo, composto por uma floresta e uma praia, contendo 4 agentes: Avião, Base, Ambulância e Vítima. A posição das Vítimas, Aviões e Ambulâncias são escolhidas aleatoriamente, isto significa que são alteradas a cada execução do simulador.
- ✓ Button Executar Resgate (passo a passo): A cada clique efetuado neste botão um passo do resgate é executado. A simulação iterativa é descrita no próximo parágrafo. Pode ser utilizado caso o usuário necessite visualizar a simulação passo a passo.
- ✓ Button Executar Resgate (continuamente): Ao pressionar este botão a simulação é executada. O sistema consiste numa comunidade de agentes trocando mensagens de forma cooperativa, buscando um objetivo comum, resgatar as vítimas. A troca de mensagens foi simplificada, em resumo, os

Aviões se movimentam aleatoriamente pelo ambiente e, caso aviste alguma Vítima, envia uma mensagem para a Base informando as suas coordenadas (INFORM). A Base, por sua fez, envia uma mensagem para a Ambulância solicitando que o resgate seja efetuado (REQUEST). A Ambulância envia uma confirmação ao efetuar o resgate (CONFIRM). Esta seqüência de troca de mensagens é repetida até que todas as vítimas sejam resgatadas. É válido ressaltar que o NetLogo disponibiliza um recurso onde é possível aumentar ou diminuir a velocidade da simulação. Também há possibilidade de parar e continuar a simulação pressionando este botão.

- ✓ Button Visualizar Código – Alô Mundo: O NetLogo disponibiliza dois recursos onde é possível visualizar texto, o Output e Command Center, este último foi escolhido para exibir códigos dos agentes e a troca de mensagens, como pode ser observado na Figura 38. O agente AloMundo é um código básico e de fácil entendimento da utilização do JADE, por este motivo a escolha em exibi-lo no protótipo.
- ✓ Button Visualizar Código da Ambulância: Este botão permite visualizar o código em JADE do agente Ambulância.
- ✓ Button Visualizar Código do Avião: Este botão permite visualizar o código em JADE do agente Avião.
- ✓ Button Visualizar Código da Base: Este botão permite visualizar o código em JADE do agente Base.
- ✓ Slider numVitimas: Permite escolher a quantidade de Vítimas que serão carregadas no ambiente.
- ✓ Slider numAvioes: Permite escolher a quantidade de Aviões que serão carregados no ambiente.
- ✓ Slider numAmbulancias: Permite escolher a quantidade de Ambulâncias que serão carregadas no ambiente.
- ✓ Switch show-intentions: Switch com dois estados: on e off. Quando ligado exibe a intenção da Ambulância, ou seja, qual será seu próximo passo, neste caso, o seu próximo resgate. A Figura 34 mostra uma execução da simulação com esta opção habilitada.

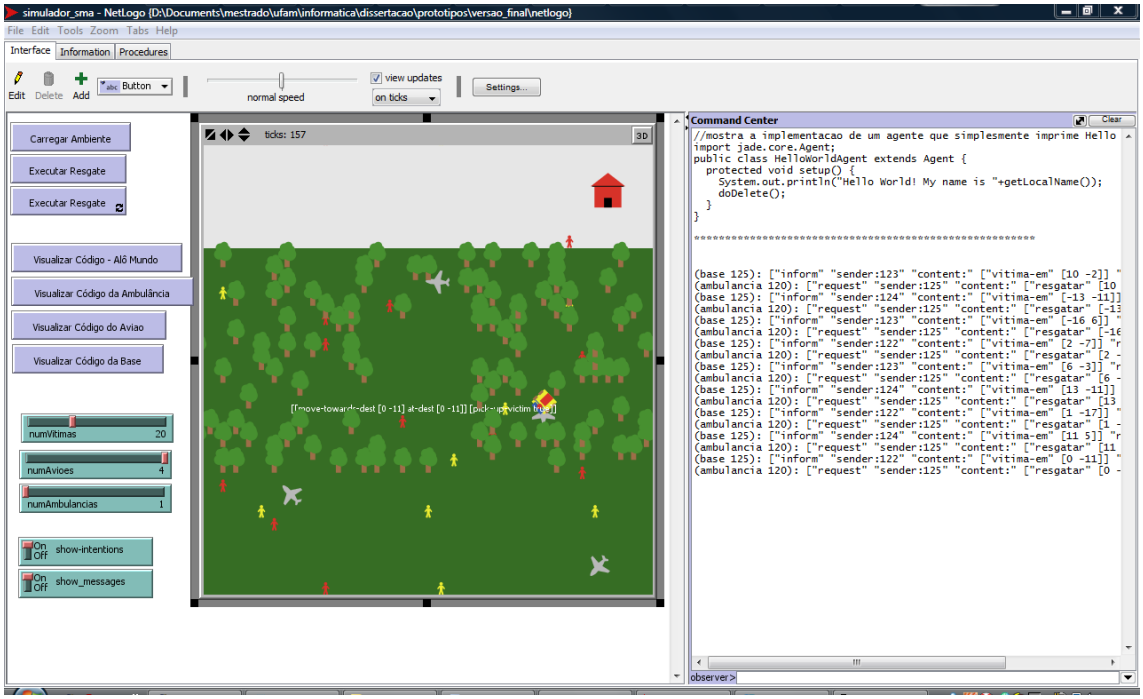


Figura 34: *Switch show-intentions* habilitado

- ✓ *Switch show-messages*: Segue o estilo do *show-intentions*, se estiver habilitado exibe as mensagens trocadas entre os agentes.

### 5.4.2 Executando o Ambiente Simulado

Ao acionar o menu SIMULADOR – EXECUTAR SIMULADOR, o usuário executa o ambiente simulado, implementado em NetLogo. A Figura 34 mostra uma tela de execução desta funcionalidade.

Todas as funcionalidades descritas no tópico 5.3.1. estão presentes neste ambiente.

Nesta tela, que pode ser observada na Figura 32, correspondente ao ambiente simulado, onde o usuário pode observar diversas características da AOP, como troca e recebimento de mensagens, cooperação, negociação autonomia, entre outras, diferenciando, desta forma, a programação orientada a agentes da programação orientada a objetos. Esta funcionalidade corresponde à primeira parte da proposta, que os alunos aprendam observando o ambiente, através de simulação.

### 5.4.3 Executando os Agentes em JADE

Ao acionar o menu SIMULADOR – EXECUTAR AGENTES EM JADE – MODO OBSERVADOR, o usuário executa os agentes implementados em JADE. Neste cenário é possível visualizar a troca de mensagens através do Agente Sniffer, um recurso do próprio framework, como pode ser visto na Figura 33.

Alguns detalhes de implementação merecem destaque, como:

- ✓ Para possibilitar a integração entre os agentes em JADE e o ambiente simulado, optou-se por utilizar um arquivo .xml, atualizado na simulação todas as vezes que uma vítima é localizada. Os agentes monitoram este arquivo e percebem quando ele sofre alguma alteração, replicando a ação no JADE. Como já foi comentado anteriormente, além de sincronizar os dois ambientes, esta técnica possibilita que as execuções sejam armazenadas e analisadas posteriormente, o que pode ser útil na análise dos ambientes já executados.
- ✓ A posição dos agentes é gerada aleatoriamente.

### 5.4.4 Executando o Ambiente Participativo

Ao acionar o menu SIMULADOR – EXECUTAR AGENTES EM JADE – MODO PARTICIPATIVO, o usuário, além de executar os agentes implementados em JADE, tem a possibilidade de assumir o papel de um dos agentes e interagir com a comunidade de agentes, através da troca de mensagens. A Figura 35 mostra a tela implementada com a finalidade de disparar a mensagem para os agentes. Aqui também é possível visualizar a troca de mensagens através do Agente Sniffer.

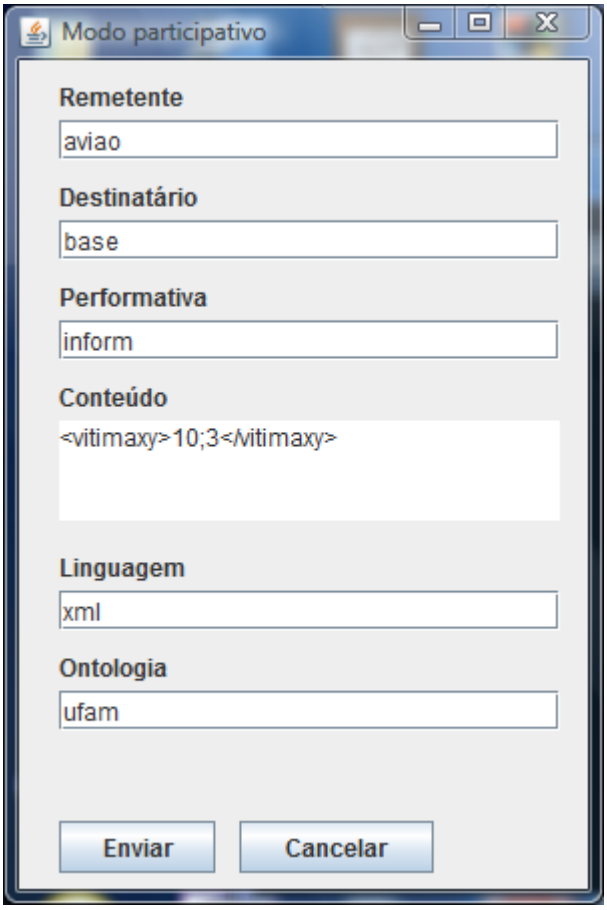


Figura 35: Tela para envio de mensagem ACL-FIPA aos agentes

Na Figura 32 pode-se observar o comportamento da comunidade de agentes após a mensagem ilustrada na Figura 35 ser disparada.

Como o objetivo principal da implementação do protótipo é efetuar os experimentos, optou-se por permitir que o usuário assuma somente o papel do Avião, localizando as Vítimas e enviando as mensagens para a Base.

Nesta tela, que pode ser observada na Figura 35, correspondente ao ambiente participativo, onde o usuário pode interagir com o Sistema Multiagente, enviando mensagens e observando o comportamento da comunidade de agentes. Esta funcionalidade corresponde à segunda parte da proposta, que os alunos põem em prática os conceitos já assimilados sobre a AOP, como o formato das mensagens, como enviá-las, qual performativa usar, quais as aplicações de ontologias, entre outras.

## 5.5 Testes

Foram realizados os principais tipos de testes em cada fase do projeto, tais como:

- Teste Caixa-Branca: Realizado no decorrer do desenvolvimento, onde foram rastreados os métodos que compõem o sistema, como troca de mensagens e funções aleatórias que interferem na configuração dos ambientes. Pressman (2005) afirma que o objetivo desta etapa é percorrer todos os caminhos independentes de um módulo ao menos uma vez. Nesta fase dos testes foram executadas as seguintes atividades:
  - ✓ Executar as decisões lógicas para valores booleanos;
  - ✓ Executar os loops;
  - ✓ Rastrear os atributos e estruturas de dados dos agentes.
- Teste Caixa-Preta: Teste utilizado para demonstrar, na prática, que os métodos estão implementados de forma correta. Isso significa que nesta etapa os agentes foram testados exaustivamente, executando-os, carregando os ambientes e verificando se a saída necessária era a esperada. Pressman (2005) afirma que os testes de caixa preta podem ser vistos como uma atividade complementar aos de caixa branca, onde se procura descobrir as seguintes falhas:
  - ✓ Métodos incorretos ou ausentes;
  - ✓ Falhas na interface;
  - ✓ Falhas nas estruturas de dados ou atributos;
  - ✓ Falhas de desempenho;
  - ✓ Falhas de inicialização e término.
- Teste de integração: Teste utilizado para verificar o funcionamento do protótipo após a junção de todos os componentes, a saber:
  - ✓ Interface principal;
  - ✓ Ambiente simulado, desenvolvido em NetLogo;
  - ✓ Ambiente multiagente, desenvolvido em JADE;
  - ✓ Scripts.

Após executar os testes descritos anteriormente, todas as falhas encontradas foram corrigidas. Os testes de caixa preta são descritos juntamente com o capítulo 6 – Exemplos de Uso.

# 6 EXEMPLOS DE USO

Objetivando intensificar a fase de testes práticos e citar exemplos práticos de utilização do sistema, o aplicativo foi exposto a diversos roteiros de uso, conforme pode ser observado nos subtópicos seguintes.

Esta etapa pode ser considerada como uma continuação dos testes de caixa preta.

## 6.1 Roteiro 1: Execução do Simulador – Alterando a Quantidade de Agentes

Para a primeira etapa da bateria de execuções, deve-se executar o ambiente simulado alterando a quantidade de agentes a cada execução. O roteiro é descrito a seguir.

1. Abrir o AVEPA. A tela referente a esta ação pode ser observada na Figura 31.
2. Executar o ambiente simulado, alterando a configuração do cenário. Esta ação pode ser observada na Figura 36.

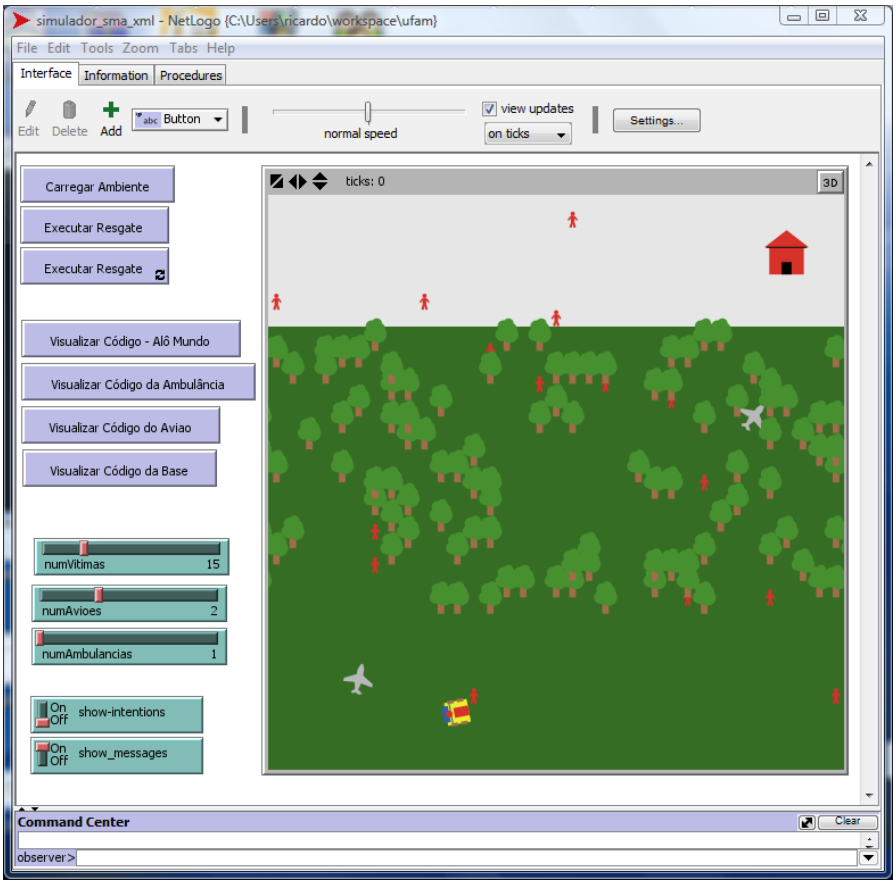


Figura 36: Executando e customizando o ambiente simulado

3. Executar a comunidade de agentes. Esta ação pode ser observada na Figura 37.

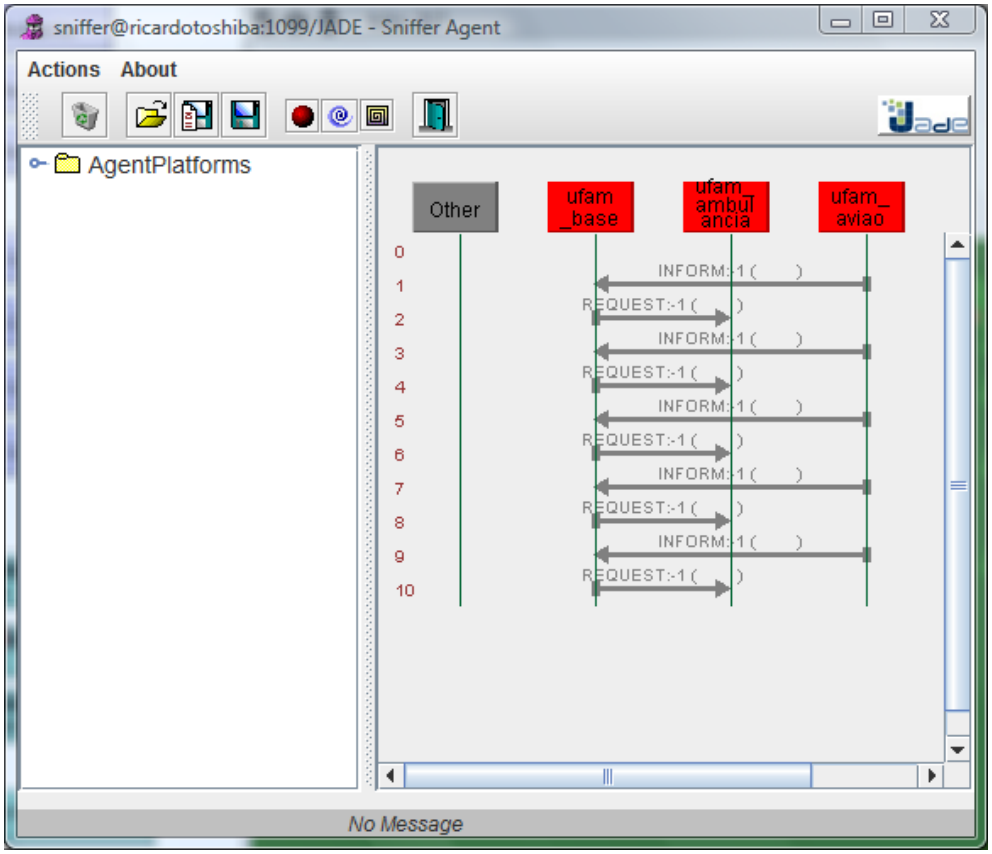


Figura 37: Tela de execução dos testes sobre a comunidade de agentes

## 6.2 Roteiro 2: Visualizando os Códigos dos Agentes

O objetivo desta segunda etapa consiste em executar o ambiente participativo repetidas vezes, disparando diferentes mensagens para a comunidade de agentes e observando seu comportamento. Este roteiro é descrito a seguir.

1. Abrir o AVEPA. A tela referente a esta ação pode ser observada na Figura 31.
2. Executar o ambiente simulado. Esta ação pode ser observada na Figura 36.
3. Clicar num dos botões de visualizar código. Esta ação pode ser observada na Figura 38.

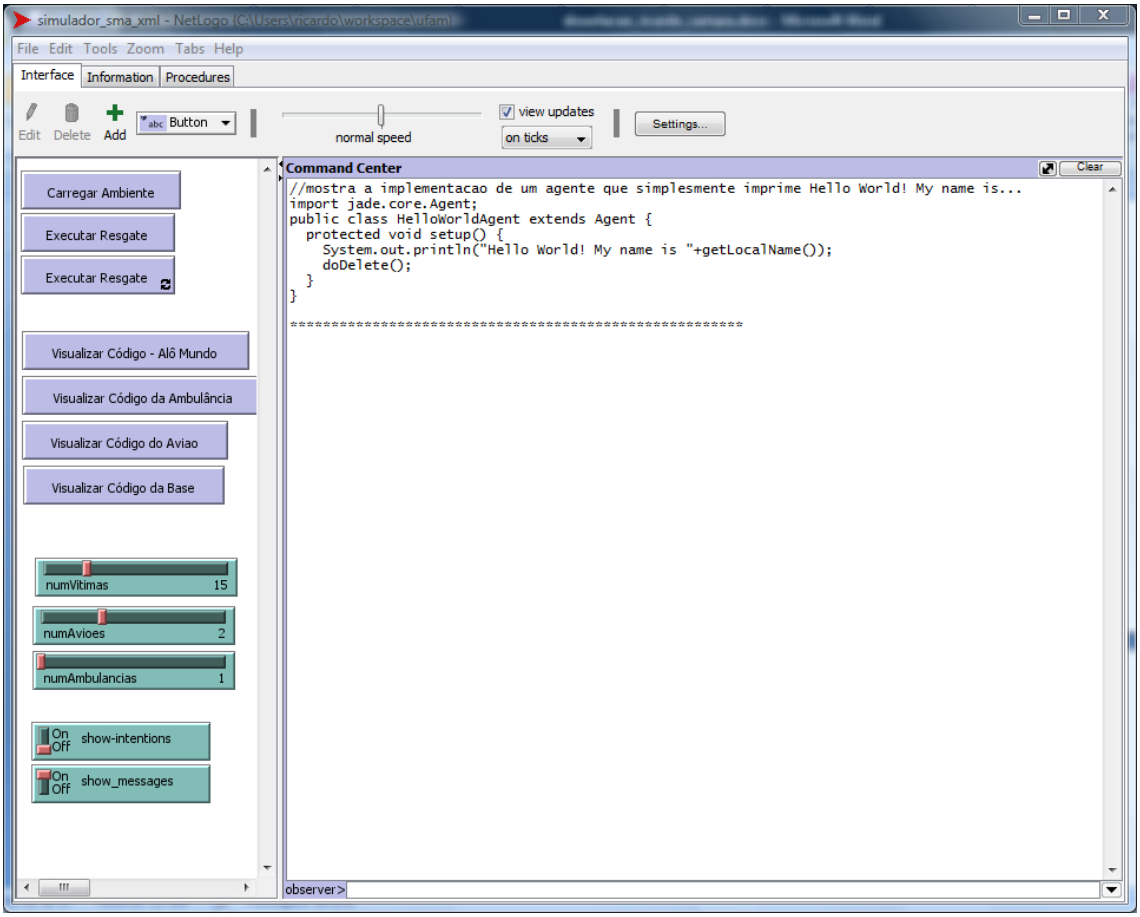


Figura 38: Teste da visualização do código de um agente

## 6.3 Roteiro 3: Execução do Ambiente Participativo para Testar a Troca de Mensagens

O objetivo desta etapa consiste em executar o ambiente participativo repetidas vezes, disparando diferentes mensagens para a comunidade de agentes e observando seu comportamento. Como já foi citado no Capítulo 5, corresponde à segunda fase da proposta, a de interação do usuário com o ambiente, após o entendimento dos principais conceitos e características da AOP. Este roteiro é descrito a seguir.

1. Abrir o AVEPA. A tela referente a esta ação pode ser observada na Figura 31.
2. Executar o ambiente participativo.
3. Disparar mensagens, observando o comportamento dos agentes. Esta ação pode ser observada na Figura 39.

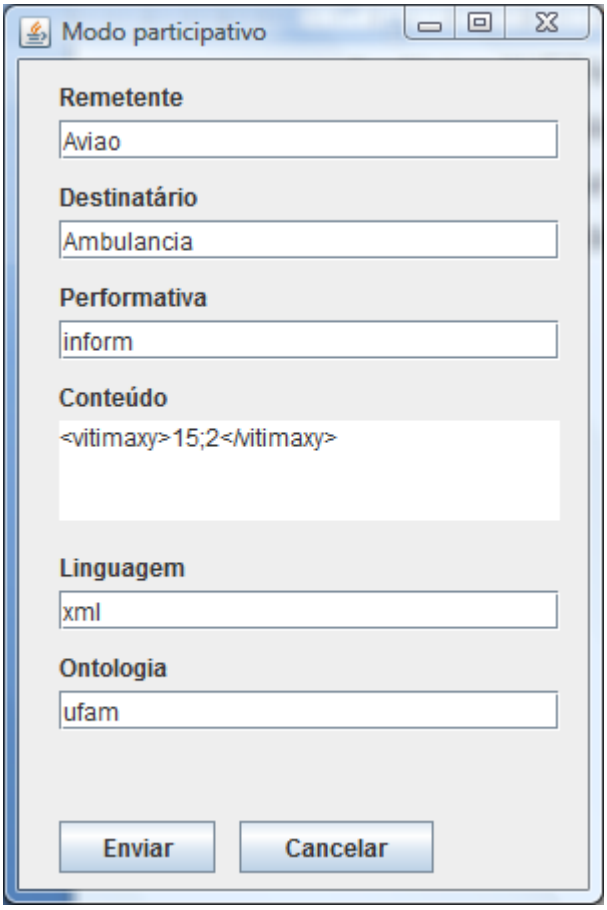


Figura 39: Teste da participação do usuário efetuando comunicação com a comunidade de agentes

É válido ressaltar que os procedimentos descritos nos tópicos 6.1 a 6.4 foram repetidos sucessivas vezes sem a detecção de falhas, ilustrando os principais exemplos de uso e concluindo, desta forma, os testes de caixa preta no protótipo.

## 6.4 Roteiro 4: Execução do Simulador – Alterando a Quantidade de Ambulâncias

A fim de testar o comportamento dos agentes quando se insere mais de uma ambulância no cenário, optou-se por executar o ambiente simulado com duas ambulâncias. O roteiro é descrito abaixo.

1. Abrir o AVEPA. A tela referente a esta ação pode ser observada na Figura 31.
2. Executar o ambiente simulado, alterando a configuração do cenário. Esta ação pode ser observada na Figura 36.

3. Alterar a quantidade de ambulâncias de uma para duas. Esta configuração pode ser observada na Figura 40.

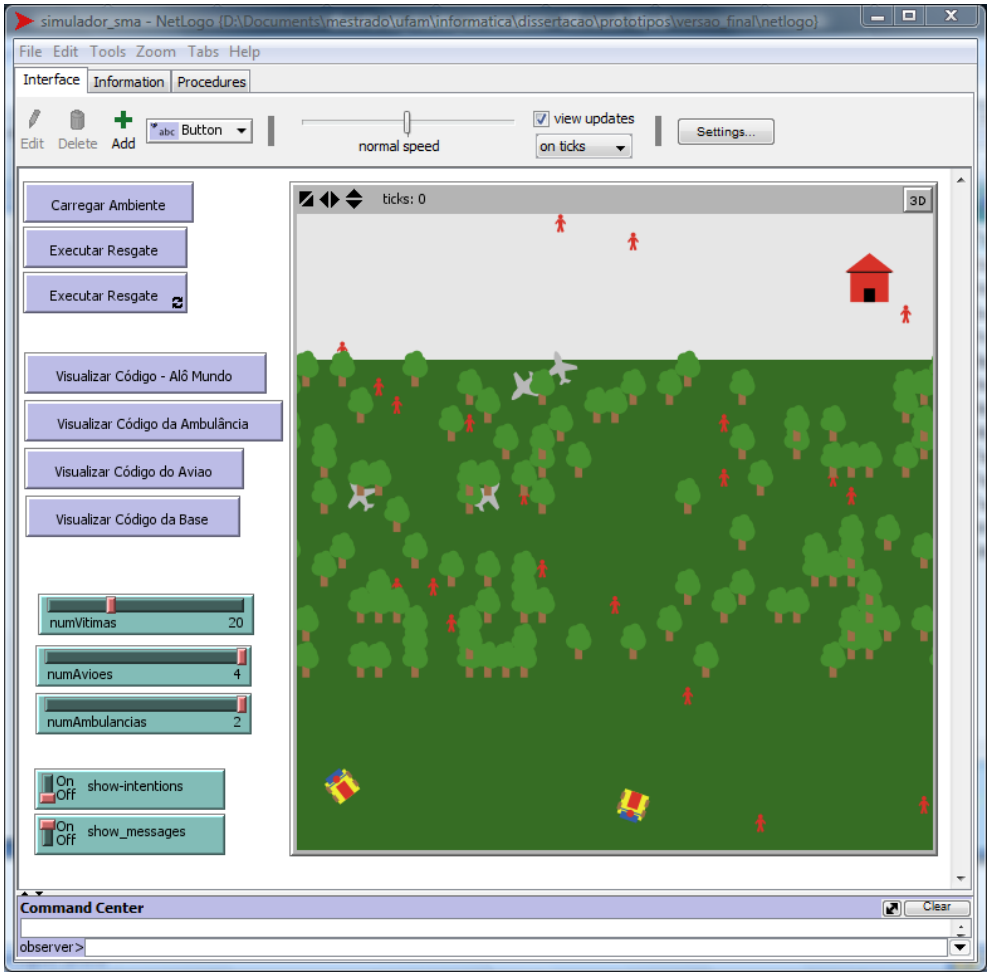


Figura 40: Inserindo duas ambulâncias no cenário

4. Clicar em executar resgate.

Neste exemplo de uso, além de todas as funcionalidades já citadas, o aluno teria a oportunidade de observar a negociação entre as ambulâncias, de forma a otimizar os resgates, trabalhando de forma cooperativa e mais eficiente. Porém, vale ressaltar que esta troca de mensagens entre as ambulâncias não foi implementada, pois este não é o principal foco do trabalho. Esta funcionalidade é proposta como trabalho futuro no Capítulo 7.

## 6.5 Contribuições Pedagógicas do Projeto

O principal resultado esperado do software é que sejam agregadas contribuições educacionais, ou seja, que os usuários aprendam conceitos e características de Sistemas Multiagente e da Programação Orientada a Agentes, bem como explorem algumas das tecnologias disponíveis atualmente para este paradigma de desenvolvimento.

Utilizando o aplicativo como um mero observador, o aluno explora os aspectos fundamentais da AOP, como autonomia, comunicação, cooperação ou competição entre os agentes. Ainda nesta etapa, é possível visualizar os códigos dos agentes que compõem o cenário, possibilitando o estudo e entendimento dos mesmos, desta forma os usuários podem explorar uma das ferramentas utilizadas na programação de agentes, o JADE. A troca de mensagens é um dos principais aspectos dos Sistemas Multiagente, por este motivo também é desejado que os usuários percebam que esta troca de mensagens possui um padrão e utiliza performativas, o que é atingido possibilitando visualizar todas as mensagens trocadas na tela, como pode ser observado na Figura 41.

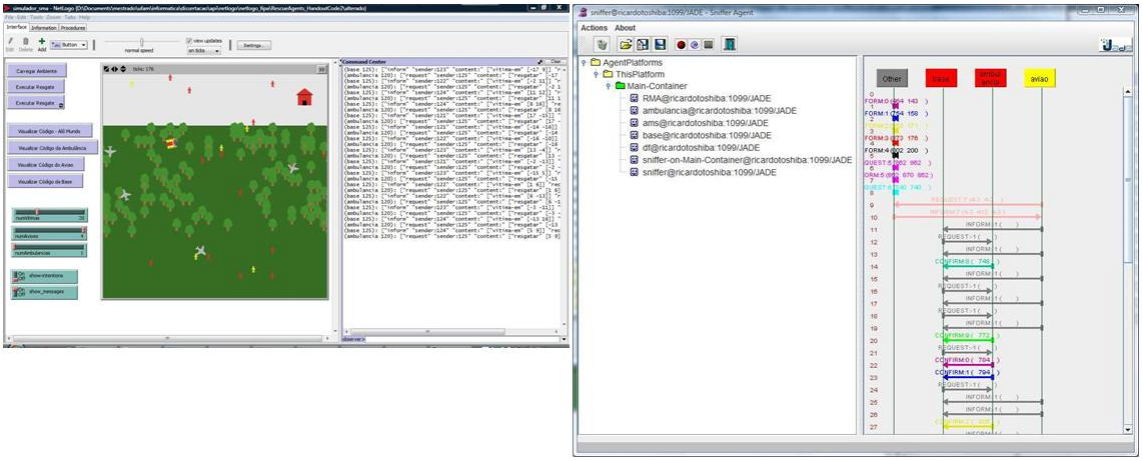


Figura 41: Execução do ambiente Simulado e Multiagente

Assimilados os principais conceitos, o aluno tem a possibilidade de atuar nos cenários, assumindo o papel de algum agente, enviando mensagens e observando o comportamento da comunidade de agentes após este tipo de interação.

Com todos estes recursos possibilitados pelo protótipo, este pode ser considerado como um software educacional, que facilita o processo de ensino e aprendizagem da Programação Orientada a Agentes.

# 7. Conclusões

A principal contribuição deste trabalho é a proposta de um software educacional, desenvolvido sob o paradigma de Sistemas Multiagente, para apoiar o processo de ensino e aprendizagem da Programação Orientada a Agentes, no que diz respeito aos conceitos teóricos e a parte prática do desenvolvimento de agentes.

Precedendo a implementação do protótipo, foi desenvolvida uma arquitetura geral e interna do ambiente a ser implementado. Numa fase posterior, por influência das ferramentas utilizadas na implementação, foi necessário pesquisar como efetuar a integração dos ambientes simulado e multiagente, desenvolvidos em NetLogo e JADE, respectivamente.

Foi desenvolvido um protótipo, onde agentes cooperam em resgates, possibilitando aos alunos observarem, através de simulação, o comportamento de uma comunidade de agentes, trocando mensagens, competindo ou cooperando para que se atinja um objetivo. Num nível mais avançado, os usuários podem assumir o papel de agentes dentro dos cenários, trocando mensagens com os demais e observando, de forma geral, os comportamentos dos seus componentes. Nesta fase, os conceitos que envolvem agentes, como troca de mensagens e ontologias, devem ser claros para os usuários do aplicativo. Do ponto de vista dos professores, o software pode ser utilizado como uma ferramenta educacional, complementando aulas presenciais, semipresenciais ou à distância.

Uma das dificuldades encontradas no decorrer da pesquisa foi utilizar uma metodologia de modelagem para SMA, tendo em vista que a AUML está parada e que os diagramas da UML 2.0 não contemplam todas as necessidades requeridas por este paradigma de programação. Existem pesquisas voltadas para a modelagem de SMA's em andamento, como a descrita por Guedes (2009).

Uma contribuição, que pode ser classificada como indireta, foi a integração de três tecnologias já existentes, o JADE, NetLogo e XML. É válido ressaltar que foram utilizadas soluções ad hoc para possibilitar estas integrações. Faz-se necessário definir o que é considerado integração destes ambientes no escopo deste trabalho. Na evolução das pesquisas houve a necessidade de se implementar dois cenários, um simulado e outro participativo. Para a implementação dos agentes e seus respectivos comportamentos, a ferramenta escolhida foi o JADE, porém, para o cenário emulado, houve a necessidade de uma ferramenta mais específica, o NetLogo surgiu como uma boa opção, minimizando o esforço na construção da

parte gráfica, bastando definir os comportamentos e eventos necessários para os agentes se movimentarem no cenário, por exemplo. Efetuadas as escolhas, surgiu um problema, como reconstituir no ambiente simulado (NetLogo) uma requisição efetuada na comunidade de agentes, implementada em JADE e vice-versa?

Optou-se por uma medida simples e eficaz para solucionar a questão anterior. Fazendo uso de uma base de dados que armazenasse as mensagens trocadas no JADE, como a requisição para se efetuar um resgate, ou eventos ocorridos no NetLogo, como a localização de uma vítima, bastaria que os agentes consultassem esta base de tempos em tempos e, ao perceberem a ocorrência de um evento, o replicariam no outro lado. Nesta fase do trabalho a tecnologia XML forneceu a solução. Armazenando estas informações num ou em vários arquivos XML, resolve-se o problema da integração e ainda possibilita que cenários já executados anteriormente sejam recuperados, o que pode ser útil em análises futuras.

## 7.1. Trabalhos Futuros

Os trabalhos citados a seguir podem ser considerados como continuação ou variantes desta pesquisa ou simplesmente idéias que surgiram no decorrer do desenvolvimento dos trabalhos.

Construção de um plugin para o JADE que automatize a simulação de um ou vários agentes.

Modelar agentes e cenários utilizando XML ou uma tecnologia similar, o que possibilitaria a usuários com poucos conhecimentos na área de Sistemas Multiagente modelarem e executarem diversos cenários. Possibilitar a geração automática de código através destes modelos e definições de alguns comportamentos dos agentes.

Possibilitar que os usuários criem novos agentes e programem seus comportamentos em tempo de execução, tornando, desta forma, os cenários altamente dinâmicos. É válido ressaltar que na versão atual do protótipo é possível configurar a quantidade de agentes antes de uma execução.

Definir uma ontologia que padronize a troca de mensagens entre os agentes.

Otimizar resgates no caso de haver mais de uma ambulância no cenário, ou seja, estes agentes trocariam mensagens e dividiriam a tarefa de resgatar as vítimas.

Como pode ser observado, analisando os trabalhos futuros propostos, a execução desta pesquisa abriu um leque de possibilidades para se incrementar o software educacional que foi desenvolvido. Algumas ações podem, inclusive, dar origem a novas abordagens ou arquiteturas. Por exemplo, se for possível modelar cenários e agentes com XML ou com uma

tecnologia similar, gerando códigos a partir destes modelos, uma nova forma de modelagem e desenvolvimento de Sistemas Multiagente estaria disponível, o que demandaria a criação e utilização de novas ferramentas para este fim.

# REFERÊNCIAS

AUML Web Site. Disponível em: <http://www.auml.org/>. Acesso em: Março/2010.

BELLIFEMINE, F.; CAIRE, G.; GREENWOOD, D.. Developing multi-agent systems with JADE. Editora Wiley, 2007.

BOOCH, G; RUMBAUGH, J e JACOBSON, I: UML, Guia do Usuário: tradução; Fábio Freitas da Silva, Rio de Janeiro, Campus, 2000.

CÂMARA, R.S.; NETTO, J.F.M.; MAIA, R.J.M.. Usando XML para Facilitar o Desenvolvimento de Ambientes de Aprendizagem Baseados em Sistemas Multiagente. In: XIX Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza/CE. XIX SBIE, 2008.

CÂMARA, R.S.; NETTO, J.F.M.. Uma Abordagem Baseada em Sistemas Multiagente para Aprendizagem de Conceitos de Agentes e Sistemas Multiagente. In: I Escola Regional de Informática - Regional Norte 1, 2009, Manaus. I Escola Regional de Informática, 2009a.

CÂMARA, R.S.; NETTO, J.F.M.; MAIA, R.J.M.. Construindo Agentes inteligentes com JADE e Eclipse. Mundo Java, 01 set. 2009b.

CAMPANA, V.F.; SANCHES, D.R.; TAVARES, O.L.; SOUZA, S.F.. Agentes para Apoiar o Acompanhamento das Atividades em Ambientes Virtuais de Aprendizagem. In: XIX Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza/CE. XIX SBIE, 2008.

CASTRO FILHO, J. A., FREIRE, R. S.; FERNANDES, A. C.. Quando objetos digitais são efetivamente para aprendizagem: o caso da matemática. In: XIX Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza/CE. XIX SBIE, 2008.

FARACO R.A.. Uma Arquitetura de Agentes para Negociação Dentro do Domínio do Comércio Eletrônico. Dissertação de Mestrado, UFSC, Florianópolis, 1998.

FASLI, M.; MICHALAKOPOULOS, M.. Supporting Active Learning through Game-Like Exercises icalt, pp.730-734, Fifth IEEE International Conference on Advanced Learning Technologies (ICALT'05), 2005.

FININ, Tim. UMBC KQML Web. Lab for Advanced Information Technology, 1996. Disponível em: <http://www.cs.umbc.edu/kqml>. Acesso em Janeiro/2010.

FIPA Foundation for Intelligent Physical Agents. Disponível em: <http://www.fipa.org>. Acesso em: Janeiro/2010.

FURLAN, J. D. Modelagem de Objetos Através da UML: São Paulo, Brasil, Makron Books, 1998.

GOMES, E.R.; SILVEIRA, R.A.; VICCARI, R.M.. Objetos Inteligentes de Aprendizagem: Uma Abordagem baseada em Agentes para Objetos de Aprendizagem. Anais do XV SBIE - Simpósio Brasileiro de Informática na Educação. Manaus AM, 2004.

GRUBER, T.R. (1995) Toward principles for the design of ontologies used for knowledge sharing. In: Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers.

GUEDES, G.T.A.; VICARI, R.M.. Applying AUML and UML 2 in the Multi-agent Systems Project. Book Series Lecture Notes in Computer Science, 2009.

HUHNS, M. N., e SINGH, M. P.. Ontologies for Agents, IEEE Internet Computing, pp. 81-83, 1997.

JAQUES, P.A.; LEHMANN, M.; JAQUES, K.S.F.. Avaliando a Efetividade de um Agente Pedagógico Animado Emocional. In: XIX Simpósio Brasileiro de Informática na Educação, 2008, Fortaleza/CE. XIX SBIE, 2008.

LIMA, D. R.; ROSATELLI, M. C.. Um Sistema Tutor Inteligente para um Ambiente Virtual de Aprendizagem. In: Workshop sobre Informática na Escola, 2004, Florianópolis. WIE, 2004.

MEDEIROS, J.B.. Alucinação e magia na arte: o ultimatum futurista de Almada Negreiros. 1991. 100 f. Monografia (Departamento de Letras)–Faculdade de Filosofia, Letras e Ciências Humanas, USP, São Paulo, 1991.

MELO, A.C.. Desenvolvendo Aplicações com UML. Rio de Janeiro: Brasport, 2002.

MIZUKAMI, M.G.N.. Ensino: as abordagens do processo. 14. ed. São Paulo: EPU, 1986. 119p.

NetLogo User Manual. Disponível em: <http://ccl.northwestern.edu/netlogo/docs/>. Acesso em: Março/2010.

NETTO, J.F.M.. Uma Arquitetura para Ambientes Virtuais de Convivência: uma Proposta Baseada em Sistemas Multiagente. Tese de Doutorado, UFES, Vitória, 2006.

NETTO, J.F.M.; MENEZES, C.S.; CASTRO JÚNIOR, A.N.. AmCorA: Uma Arquitetura Multiagente Baseada em FIPA. In: XV Simpósio Brasileiro de Informática na Educação, 2004, Manaus/AM. XV SBIE, 2004.

NETTO, J.F.M.; TAVARES, O.L.; MENEZES, C.S.. Um Ambiente Virtual para a Aprendizagem de Xadrez. In: Workshop de Jogos Digitais na Educação - SBIE 2005, 2005, Juiz de Fora. Anais do XVI Simpósio Brasileiro de Informática na Educação - SBIE, 2005.

NOY, N. F.; MCGUINNESS, D. L. (2001) Ontology Development 101: A Guide to Create Your First Ontology. Knowledge Systems Laboratory Technical Report KSL-01-05, Stanford University. 25p.

OLIVEIRA, J.M.; RABELLO, S.; BARBOSA, J.L.V.. Um Modelo Multi-agente Descentralizado para Ambientes de Educação Ubíqua. In: XX Simpósio Brasileiro de Informática na Educação, 2009, Florianópolis/SC. XX SBIE, 2009.

OLIVEIRA, M. K. de (1997) Vygotsky: aprendizado e desenvolvimento, um processo sócio histórico. São Paulo, Scipione, 111 pp. (4a Edição)

PRESSMAN, R. S.. Software Engineering: A Practitioner's Approach, McGraw-Hill, 6th ed, Nova York, NY, 2005.

PROENÇA, H.P.M.C.. Sistema Multiagente Regulador de Tráfego. Disponível em: <http://paginas.fe.up.pt/~eol/SMA/20012002/trab_trafego/SMA_Relat.htm>. Acesso em: Junho/2009.

RUSSELL, S.; NORVIG, P.. Artificial Intelligence: A Modern Approach. 2a Ed. Editora John Wiley&Sons, Inglaterra, 2002.

Site do Ministério da Educação. Disponível em http://portal.mec.gov.br/index.php?option=com_content&view=article&id=289&Itemid=356. Acesso em: Março/2010.

Site Oficial do Moodle. Disponível em http://moodle.org/. Acesso em: Março/2010.

Site Oficial do TelEduc. Disponível em http://moodle.org/. Acesso em: Março/2010.

SYCARA, K., PANNU, A., WILLIAMSON, M., ZENG, D. (1996), Distributed intelligent agents, IEEE Expert, Vol. December pp.36-46.

WEISZ, T. O Diálogo entre o Ensino e a Aprendizagem. São Paulo: Ática, 1999.

WOOLDRIDGE, M., JENNINGS, N.R.. Intelligent Agents: Theory and Practice, Knowledge Engineering Review, 1994. Disponível em http://www.doc.mmu.ac.uk/STAFF/mike/ker95.ps. Acesso em: Março/2010.

WOOLDRIDGE, M.. An Introduction to MultiAgent Systems. Second edn. John Wiley & Sons (2009).