FACULDADE DE INFORMÁTICA
Programa de Pós-Graduação em Ciência da Computação
PUCRS – Brasil

http://www.pucrs.br/inf

Ambientes Inteligentes para Suporte ao Ensino de Programação

Sabrina dos Santos Marczak
Lucia Maria Martins Giraffa

TECHNICAL REPORTS SERIES

Number 028
August, 2003

Contato:
smarczak@inf.pucrs.br
http://www.inf.pucrs.br/~smarczak

giraffa@inf.pucrs.br
http://www.inf.pucrs.br/~giraffa

Sabrina dos Santos Marczak é aluna do curso de mestrado do Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática (PPGCC/FACIN), PUCRS. É integrante do grupo de pesquisa GIE (Grupo de Informática na Educação) desde 2001, tendo sua pesquisa na área de ambientes inteligentes de ensino, mais especificamente em relação ao ensino de Algoritmos e Programação. Sua bolsa de pesquisa é patrocinada pelo Convênio Dell/PUCRS, onde atua como pesquisadora na área de Qualidade de Software no Centro de Pesquisa e Desenvolvimento em E-Business deste convênio.

Lucia Maria Martins Giraffa trabalha na PUCRS desde 1986. É professora titular da Faculdade de Informática, credenciada ao PPGCC/FACIN da PUCRS. Coordena o grupo de pesquisa GIE e desenvolve pesquisas nas áreas de Sistemas Multiagentes e Informática Aplicada à Educação, bem como desenvolve softwares educacionais. Também, coordena o Centro de Tecnologia XML – Microsoft/PUCRS, o qual tem por objetivo capacitar profissionais em tecnologias Microsoft. Concluiu seu doutorado em 1999, junto ao Instituto de Informática da UFRGS (RS, Brasil).

Published by PPGCC/FACIN, PUCRS
Av. Ipiranga, 6681
90619-900 – Porto Alegre – RS – Brasil

Sumário

Lista de abreviaturas **4**
Lista de figuras **5**
1 Introdução **6**
2 Ambientes de suporte ao ensino de algoritmos e programação **13**
2.1 Ambientes desenvolvidos com a tecnologia de agentes **16**
2.1.1 AME-A 16
2.1.2 SEI 19
2.1.3 Eletrotutor III 21
2.1.4 MuTanIS 24
2.1.5 AmCorA 26
2.1.1.1 AmCorA-ES 27
2.1.1.2 *Moonline* 29
2.1.1.3 Ambiente para apoio à aprendizagem de programação 32
2.1.1.4 SAmbA 34
2.1.6 LeCS 36
2.1.7 TUTA 39
2.1.8 Um Ambiente Inteligente para Aprendizagem Colaborativa (AIAC) 41
2.1.9 SEMEAI 42
2.2 Ambientes desenvolvidos com tecnologias diversas **46**
2.2.1 Curso individualizado da linguagem de programação C 46
2.2.2 ELM-ART 48
2.2.3 ADAPT 50
2.2.4 C-Tutor 53
2.2.5 PORTUGOL/PLUS 56
2.2.6 ASIMOV 58
2.2.7 LPT-TUTOR 59
2.2.8 Construção de abstrações em lógica de programação 59
2.2.9 HabiPro 61
2.2.10 Ambiente colaborativo para o aprendizado de programação 62
3 Considerações finais **65**
4 Referências bibliográficas **69**

Lista de abreviaturas

AIAC	Ambiente Inteligente para Aprendizagem Colaborativa
AIED	*Artificial Intelligent in EDucation*
CGI	*Common Gateway Interface*
EAD	Ensino a Distância
HTML	*HyperText Markup Language*
IA	Inteligência Artificial
IAED	Inteligência Artificial aplicada à Educação
IE	Informática na Educação
ITS	*Inteligent Tutoring Systems*
JAIED	*Journal of Artificial Intelligent in Education*
JSP	*Java Server Pages*
KQML	*Knowledge Query and Manipulation Language*
LAPRO	Laboratório de Programação
RT	Relatório Técnico
SBC	Sociedade Brasileira de Computação
SBIA	Simpósio Brasileiro de Inteligência Artificial
SBIE	Simpósio Brasileiro de Informática na Educação
SMA	Sistema Multiagentes
STI	Sistema Tutor Inteligente

Lista de figuras

Figura 1 - Arquitetura clássica de um STI 8
Figura 2 - Arquitetura ampliada de um STI 9
Figura 3 - Coreografia da assistência do STI ao aluno 10
Figura 4 - Representação do contexto deste trabalho 12
Figura 5 - Arquitetura do ambiente AME-A 17
Figura 6 - Arquitetura do ambiente ELETROTUTOR III 23
Figura 7 - Arquitetura do ambiente MuTanIS 25
Figura 8 - Interação entre os agentes do Ambiente do Aluno 31
Figura 9 - Arquitetura do ambiente para apoio à aprendizagem de programação 33
Figura 10 - Arquitetura do ambiente SAmbA 35
Figura 11 - Interface com o usuário do sistems LeCS 37
Figura 12 - Arquitetura do sistema LeCS 38
Figura 13 - Arquitetura do ambiente TUTA 40
Figura 14 - Interação dos agentes do AIAC 42
Figura 15 - Arquitetura do ambiente SEMEAI 43
Figura 16 - Arquitetura do curso individualizado 47
Figura 17 - Área de trabalho do ambiente ADAPT 51
Figura 18 - Estrutura do ambiente C-Tutor 55
Figura 19 - Visão do ambiente do Portugol/PLUS 57
Figura 20 - O compilador do ambiente PORTUGOL/PLUS 57
Figura 21 - Arquitetura do sistema HabiPro 62
Figura 22 - Módulos funcionais do ambiente colaborativo 63

1 Introdução

O aprendizado de programação é um desafio para muitos professores e alunos principiantes de vários cursos de graduação. As disciplinas de Algoritmos e Laboratório de Programação (LAPRO) formam o binômio central do ensino de programação de principiantes. Os nomes variam, a seriação também (ambas no primeiro semestre, Algoritmos no primeiro, LAPRO no segundo), porém a idéia geral permanece: desenvolver habilidades de resolver problemas com o uso do computador, e desenvolver habilidades cognitivas amplas, tais como análise, síntese e aplicação. Ambas necessárias ao processo de abstração que permite ao aluno receber um problema e propor uma alternativa de solução.

Muito tem se feito para estimular os estudantes a aprenderem e manterem o interesse [CHA01], procurando não deixá-los desanimar, apesar das dificuldades. No entanto, mesmo com todo o esforço, muitas questões permanecem em aberto. Especialmente a questão envolvendo a heterogeneidade da formação dos alunos e a dedicação extra-classe necessária para a fixação dos conteúdos trabalhados em aula.

Segundo [TOB01], os estudantes encontram dificuldades no aprendizado de programação por razões diversas, dentre elas destaca-se a preocupação excessiva com detalhes de sintaxe da linguagem usada, a falta de uma visão daquilo que se quer solucionar, a idealização de soluções adequadas ao problema proposto, a organização dessas soluções em passos seqüenciais e abstração do funcionamento dos mecanismos escolhidos.

As dificuldades encontradas podem ser associadas ao alto grau de repetência nas disciplinas introdutórias. Relatos de professores e alunos apontam as dificuldades

demonstradas nestas disciplinas, onde se destaca o domínio do raciocínio lógico e a resolução de problemas. Segundo [GIR02], outro aspecto apontado como causa das repetências e evasão, principalmente nos alunos de cursos noturnos, é o alto envolvimento com o trabalho, a impossibilidade de comparecimento às atividades presenciais da disciplina e o pouco envolvimento com as tarefas complementares extraclasse (trabalhos e exercícios). Geralmente a reprovação está ligada diretamente ao fato dos alunos não se envolverem nas tarefas necessárias para o bom entendimento e fixação dos conteúdos destas disciplinas.

As diferentes pesquisas analisadas e apresentadas neste Relatório Técnico (RT) apontam como alternativa para auxiliar na resolução deste problema o desenvolvimento de ambientes computadorizados de apoio ao ensino-aprendizagem de programação. A tendência observada nestes ambientes é utilizar técnicas de Inteligência Artificial (IA), a fim de prover a estes sistemas capacidade de adaptação ao contexto e de personalização do ambiente de acordo com as características dos alunos, além de permitir um alto grau de interatividade entre o ambiente e os usuários. A introdução de técnicas de IA nestes ambientes tem, também, a finalidade de propiciar mecanismos de modelagem do processo de ensino, assim como ter a possibilidade de inferir o provável estado cognitivo corrente do estudante [MUS01].

As técnicas de IA associadas aos ambientes educacionais são significativamente representadas nos Sistemas Tutores Inteligentes (STIs). Estes sistemas pertencem a uma classe de programas que possuem uma base de conhecimento separada da forma como este será utilizado. A arquitetura destes sistemas permite criar um certo grau de autonomia nos módulos, auxiliando no tratamento das questões envolvendo a personalização da assistência ao aluno. Segundo [GIR97], a utilização de técnicas de IA, no projeto e desenvolvimento de ambiente de ensino-aprendizagem computadorizados, tem se constituído em objeto de maior investigação por parte dos pesquisadores da área de Informática Aplicada à Educação (IAED), devido suas potencialidades.

Os STIs são sistemas que, interagindo com o aluno, modificam suas bases de conhecimento, podem perceber as intervenções do aluno, podem possuir a capacidade de aprender e permitem adaptar as estratégias de ensino de acordo com o desempenho do aluno. Caracterizam-se, principalmente, por construir um Modelo Cognitivo do Aluno, através da interação e da formulação e comprovação de hipóteses sobre o conhecimento do aluno. Possuem a capacidade de adequar estratégias de ensino-aprendizagem ao aluno e à situação atual [VIC90, Apud in SAN01].

Segundo [OLI94, Apud in BOL01], os STIs possuem uma organização básica com alguns componentes funcionais: *Módulo do Aluno*, *Módulo Tutor*, *Módulo de Domínio* e *Interface.* A figura 1 apresenta a arquitetura onde os módulos estão representados com suas inter-relações. Esta arquitetura é denominada arquitetura clássica.

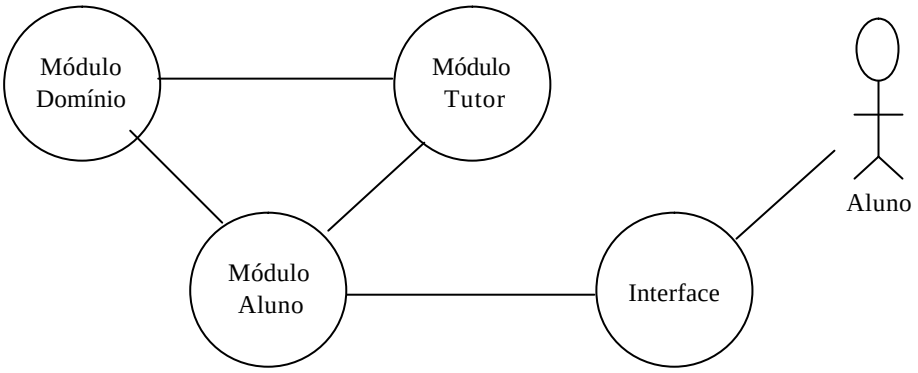

Figura 1 - Arquitetura clássica de um STI

O *Módulo do Aluno* armazena informações específicas para cada estudante de forma individual. Segundo [GIR01], este módulo representa o conhecimento e as habilidades cognitivas do aluno em um dado momento. A partir desse modelo e do conteúdo representado na base do domínio, o sistema deve ser capaz de inferir a melhor estratégia de ação a ser utilizada para cada aluno. O *Módulo Tutor*, ou Módulo Pedagógico, oferece uma metodologia para o processo de aprendizado. Possui o conhecimento sobre as estratégias e táticas para selecioná-las em função das características do aluno. As entradas deste módulo são fornecidas pelo *Módulo do Aluno*. O *Módulo de Domínio* armazena a informação que o

tutor está ensinando. A modelagem do conhecimento a ser disponibilizado é de grande importância para o sucesso do sistema como um todo. Por fim, o módulo *Interface* é o responsável por fazer a interação entre o tutor e o aluno.

Segundo [GOU01], a arquitetura clássica foi ampliada por Self [SEL99] para se tornar uma arquitetura associada ao modelo de interação que ocorre ao longo de uma sessão de trabalho entre o(s) aluno(s) e o ambiente. Esta nova arquitetura pode ser observada na figura 2.

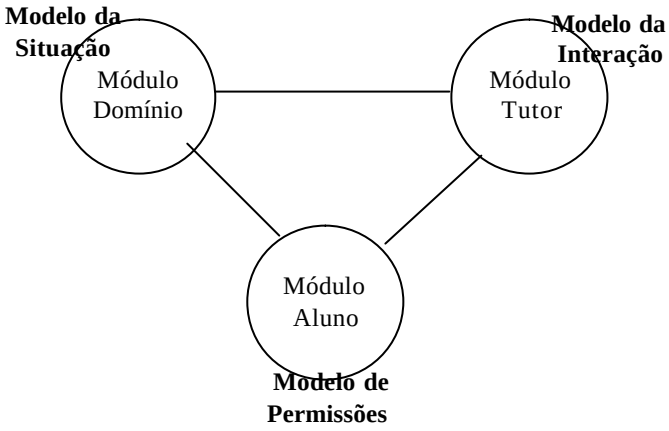

Figura 2 - Arquitetura ampliada de um STI

Segundo [GOU01], o *Módulo Domínio* não é mais uma forma de tornar as informações inter-relacionadas, mas sim um modelo dos aspectos do conhecimento sobre o domínio que o aluno pode acessar durante as interações com o STI (*Modelo da Situação*). O *Módulo Aluno* não mais relaciona somente as informações sobre a análise das interações do aluno com o domínio, mas busca uma contextualização maior destas interações em função das ações do aluno, o contexto em que elas ocorrem e a estrutura cognitiva do aluno naquele momento (*Modelo de Interação*). O *Módulo Tutor* deixou de ser o responsável pela seleção do conteúdo e estratégias para se tornar, de uma forma mais ampla, aquele que conduz o aluno de acordo com objetivos e desafios educacionais que o ambiente proporciona ao aluno (*Modelo de Permissões*).

Observa-se, no entanto, que a utilização de um tutor, com uma estratégia menos diretiva, a qual não determina totalmente os caminhos da interação entre o aluno e o conhecimento, permite que o STI atue de forma a complementar o ensino em sala de aula, auxiliando o professor em sua tarefa. Esta assistência desenvolve um processo contínuo, ao qual Giraffa [GIR99] chama de "coreografia" (vide figura 3) e que pode ser associada ao ciclo de tutoração.

Figura 3 - Coreografia da assistência do STI ao aluno

Esta coreografia representa a assistência personalizada do tutor, em função do modelo cognitivo atual do aluno, e essa assistência tem como finalidade organizar o domínio de acordo com os objetivos educacionais modelados no STI. Esta organização do domínio busca alterar o estado cognitivo do aluno (Modelo do Aluno), finalizando um ciclo na coreografia desenvolvida pelo tutor [GOU01].

Desta forma, o tutor realiza suas funções com a finalidade de assistir o aluno de acordo com suas características individuais, auxiliando-o a organizar o conteúdo apresentado. Tal assistência segue então os objetivos educacionais desempenhados pelo tutor, sendo estes, explicitados através das estratégias e táticas selecionadas.

Verifica-se que os STIs objetivam reproduzir no computador um comportamento similar ao realizado por um professor. No entanto, uma das maiores preocupações dos pesquisadores da área é a interação do STI com o aluno, considerando que um STI

tradicional é baseado em um estilo rígido de interação, significando que o sistema detém sempre o controle da mesma [SAN01].

Segundo [SAN01], uma das formas encontradas para amenizar este problema é a aplicação de técnicas de Inteligência Artificial Distribuída, a qual traz sua contribuição com a utilização de agentes inteligentes. A abordagem de agentes em STIs possibilita interações mais naturais e mais próximas entre alunos e o sistema tutor, onde a iniciativa da interação é normalmente compartilhada entre o sistema e o aluno. Tais interações são um contraste em relação aos documentos estáticos geralmente encontrados em materiais de cursos baseados na Internet.

Atualmente, as pesquisas em STIs preocupam-se com a construção de ambientes que possibilitem um aprendizado colaborativo. Neste contexto, a utilização de agentes inteligentes possibilita o desenvolvimento de diferentes raciocínios e a integração de várias ações para alcançar um aprendizado mais efetivo [SAN01]. Também, possuem a vantagem, em relação às arquiteturas tradicionais de STI, de proporcionar uma maior flexibilidade no tratamento dos elementos que compõem o sistema. Uma vez que vários agentes podem ser agrupados para representar um componente do STI.

O futuro trabalho de dissertação de mestrado da autora deste RT está inserido dentro do projeto de pesquisa PROOGRAMA [GIR02], que visa o desenvolvimento de um ambiente cooperativo de aprendizagem através da *Web* para disciplinas de algoritmo e programação básica, em cursos de terceiro grau. Busca-se, com este projeto, apresentar-se uma proposta alternativa às metodologias tradicionais dos cursos de graduação para as disciplinas de Algoritmos e LAPRO. Sendo assim, estão envolvidas no desenvolvimento deste trabalho futuro as seguintes áreas de pesquisa: o Ensino de Programação, a Informática na Educação (IE) e a IA, mais especificamente, a área de Sistemas Multiagentes (SMAs). A figura 4 representa de forma gráfica o inter-relacionamento destas áreas de pesquisa.

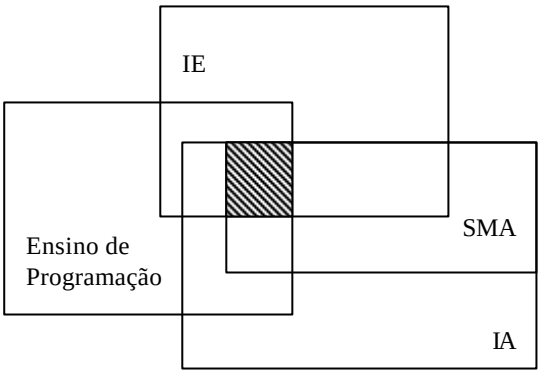

Figura 4 - Representação do contexto deste trabalho

Este RT permitiu reunir subsídios para o futuro trabalho de dissertação de mestrado a ser desenvolvido, ao longo do curso de Mestrado em Ciência da Computação da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), sob orientação da Profa. Dra. Lucia Maria Martins Giraffa. Com a revisão literária realizada e a análise dos ambientes, foi possível identificar características de ambientes com pressupostos similares ao projeto PROOGRAMA (trabalhos correlatos) e obter indicadores (requisitos) para auxiliar na proposta a ser desenvolvida na dissertação de mestrado. No capítulo 3 estas contribuições serão melhores detalhadas.

O texto está organizado da seguinte forma: o capítulo 2 descreve os trabalhos correlatos. O capítulo 3 apresenta as considerações finais. E o capítulo 4 apresenta as referências bibliográficas.

Este RT destina-se a leitores que buscam saber sobre ambientes de suporte ao ensino de algoritmos e programação. De forma geral, são descritas características dos ambientes, apresentados princípios metodológicos utilizados como base para sua especificação e aspectos computacionais, tais como arquitetura do sistema, agentes e suas respectivas funcionalidades, e tecnologias utilizadas no seu desenvolvimento. A ordem de leitura recomendada para este texto é a ordem numérica ascendente dos capítulos, sendo que é possível começar a leitura diretamente no capítulo 2, caso o leitor não esteja especialmente interessado em uma breve exposição sobre STIs e SMAs.

2 Ambientes de suporte ao ensino de algoritmos e programação

Este capítulo apresenta a descrição de alguns ambientes intencionalmente selecionados ao longo da revisão bibliográfica, utilizando como fontes:

- Anais do Simpósio Brasileiro de Informática na Educação (SBIE);
- Anais do Simpósio Brasileiro de Inteligência Artificial (SBIA);
- Anais do Congresso Nacional da Sociedade Brasileira de Computação (SBC);
- *Proceedings of International Inteligent Tutoring Systems Conference* (ITS);
- *Artificial Intelligent in Education Conference* (AIED);
- *International Journal of Artificial Intelligent in Education* (JAIED);
- Dissertações de Mestrado e Teses de Doutorado oriundas de diversos grupos de pesquisa brasileiros.

Estas fontes fornecem, de forma significativa, o registro da pesquisa que se realiza nas áreas de IAED e IE, especialmente no domínio escolhido: o ensino de programação para alunos iniciantes.

A preocupação em desenvolver ambientes para suporte ao ensino de programação remonta ao início da própria IAED. Os trabalhos *Laura* [ADA80, Apud in BIN99], utilizado no apoio ao ensino da linguagem Fortran; *Mycroft* [GOL75, Apud in BIN99], que auxilia no ensino da linguagem Logo; *Proust* [JOH87, Apud in BIN99], sistema de apoio ao aprendizado da linguagem Pascal; *Bip* [BAR76, Apud in BIN99], cuja linguagem-alvo é

Basic; *Greaterp* [BOU87, Apud in BIN99], um sistema tutorial para o ensino da linguagem LISP; *Gil* [REI88, Apud in BIN99], que acrescentou uma interface gráfica e a capacidade de visualização ao sistema *Greaterp*; *BRIDGE* [BON85, Apud in BIN99], para o ensino de programação Pascal; *SPADE* [BOU87, Apud in BIN99], utilizado para o ensino de programação com a linguagem Logo; e *DISCOVER* [RAM93, Apud in BIN99], que auxilia o ensino de programação através de uma linguagem própria, semelhante ao pseudo-código; são exemplos de que a busca pelo desenvolvimento de ambientes para auxiliar na aprendizagem de conteúdos/habilidades envolvendo a programação de computadores não é algo novo e que não está esgotado.

O ensino de programação é um domínio que possui as mesmas características do que qualquer outro domínio do conhecimento humano no que tange aos aspectos de necessidade de organização intencional dos conteúdos associados a uma proposta metodológica. Ou seja, as mudanças que se tem observado no conjunto de crenças de como os indivíduos processam informação e como aprendem, e as implicações disto no processo de aprendizagem dos alunos, são também absorvidas pelos pesquisadores de IAED. Isto é retratado na preocupação dos mesmos em desenvolver sistemas baseados em conhecimento, como é o caso dos STIs.

Cada sistema traz implícito na sua modelagem este conjunto de crenças e pressupostos psico-pedagógicos que o projetista possui e professa com relação à aprendizagem de programação.

O ensino de programação requer o desenvolvimento de um conjunto de habilidades no aprendiz (estudante). Logo, o bom entendimento do "processo" de construção da solução é tão ou mais importante do que a solução propriamente dita. Ensinar a programar necessita um "sistema de depuração de erros" que funcione de forma efetiva e permita ao aluno entender o porque errou, a dimensão e as implicações deste erro.

Em situação de aula presencial, quando o estudante consegue interagir com colegas e professores, ele tem a oportunidade de ter um *feedback*[1] que permite o entendimento de seu(s) erro(s). Entretanto, trazer todo este contexto para um ambiente computadorizado de suporte ao processo de ensino-aprendizagem é uma tarefa complexa. Vai requerer a modelagem dos conteúdos associada à metodologia na forma de problemas com graus de complexidade e um conjunto crescente de estratégias e táticas de ensino para auxiliar o aluno a entender a sua solução. E, que permita o entendimento do erro, quando ele ocorrer [GIR02].

Neste capítulo destaca-se alguns trabalhos que possuem os mesmos pressupostos que se acredita serem muito importantes em um ambiente desta natureza:

- Base de conhecimento utilizando exercícios com graus de dificuldade diferentes;
- Uma metodologia para embasar o ensino de programação;
- Uso de agentes ou alguma técnica de IA na modelagem/implementação do ambiente;
- Possibilidade de trabalho colaborativo.

Além destes aspectos, os critérios de seleção dos trabalhos também foram: a utilização da tecnologia de agentes, possuir aspectos relacionados ou inspirados na arquitetura de STI e, ainda, a possibilidade de interação entre os alunos e professores. Salienta-se que nem todos os ambientes têm, diretamente, como enfoque o ensino de algoritmos e programação, mas foram selecionados por atenderem os critérios citados acima.

Alguns dos sistemas possuem todos estes aspectos e outros apenas alguns deles. Desta forma, foram agrupados em: ambientes desenvolvidos com a tecnologia de agentes e ambientes desenvolvidos com tecnologias diversas. Esta divisão foi utilizada para facilitar a

[1] *Feedback* é a retroalimentação fornecida ao aluno por ocasião do erro. Baseia-se na heurística de um professor especialista no domínio e está diretamente relacionada a uma metodologia.

análise realizada e auxiliar no levantamento das características dos sistemas visando um futuro conjunto de requisitos necessários a serem observados na proposta futura.

2.1 Ambientes desenvolvidos com a tecnologia de agentes

2.1.1 AME-A

O AME-A é um ambiente proposto por D'Amico [DAM97] e que teve continuidade nos trabalhos de [DAM98, DAM99 e PER01]. O SMA é destinado a aplicações interativas para Ensino a Distância (EAD). A proposta pretende suportar o ensino às características psico-pedagógicas do aprendiz. Foi concebido sem a preocupação de atender as características de um domínio específico. Isto é, a proposta pretende criar um arcabouço genérico que possa ser instanciado para cada área de conhecimento.

Para antender tal proposta, a estrutura geral é muito grande e de complexa implementação. D'Amico iniciou o trabalho na sua tese de doutorado e vários trabalhos estão realizando as funcionalidades da arquitetura proposta.

A característica principal que diferencia esta proposta é a concepção da aprendizagem dividida entre estática e dinâmica. A aprendizagem estática corresponde a primeira interação do aprendiz com o ambiente, onde um agente modela o aprendiz conforme suas características afetivas, motivação e nível de conhecimento. A aprendizagem dinâmica ocorre durante a interação, quando é validado o modelo de aluno e estratégias pedagógicas em vigor.

Esta idéia não é nova nos STIs, mas foi tratada de forma explícita no trabalho de [DAM97]. A aprendizagem estática é associada à primeira interação do aluno com o sistema. É a fase onde se cria o modelo inicial do aluno.

Por se tratar de um SMA, encontra-se cada agente realizando suas tarefas e trocando mensagens entre si.

Conforme [DAM98, Apud in BOL01], cada agente é responsável por suas tarefas e agem continuamente no ambiente, com a finalidade de cooperar para promover uma aprendizagem inteligente e adaptável às características dos aprendizes.

A figura 5 apresenta os agentes envolvidos no ambiente, bem como retrata suas interações.

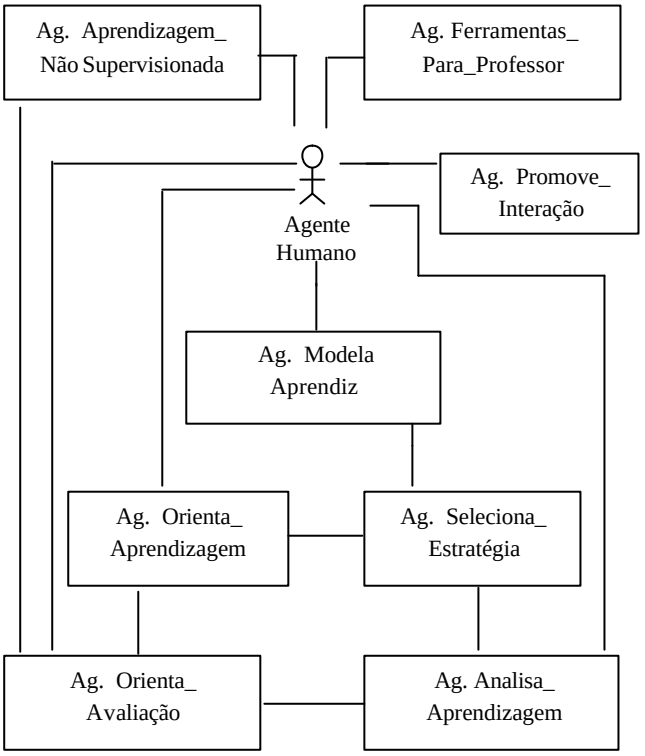

Figura 5 - Arquitetura do ambiente AME-A

A complexa sociedade proposta por [DAM99] apresenta os seguintes agentes:

- *Agente Aprendizagem_Não_Supervisionada*: é responsável pela aprendizagem livre e sem supervisão quando o aprendiz assim desejar;

- *Agente Promove_Interação*: auxilia a interação entre alunos, entre aluno e professor, e entre professores;

- *Agente Ferramentas_Para_Professor*: o professor possui uma ferramenta para auxiliá-lo na integração do material no banco de dados do ambiente. O *Agente Ferramentas_Para_Professor* tem como função orientar o professor e armazenar o material do curso. Inicialmente o professor cadastra-se no sistema, podendo incluir novos materiais, alterar e aperfeiçoar o material existente;

- *Agente Modela_Aprendiz*: o ambiente propõe-se a personalizar o ensino conforme um modelo de aluno gerado pelo *Agente Modela_Aprendiz*. Para o modelo de aluno, adotou-se a combinação de quatro pares de perfis: Extrovertido-Introvertido, Sensitivo-Intuitivo, Emocional-Racional e Perceptivo-Julgador, gerando 16 perfis psicológicos. Através de um questionário que é dado ao aprendiz no início do curso, o *Agente Modela_Aprendiz* verifica em qual dos perfis o aluno se enquadra. Essas informações que caracterizam o aluno são passadas para o *Agente Seleciona_Estratégia*;

- *Agente Seleciona_Estratégia*: para o perfil do aprendiz, foram definidos métodos de ensinar e características relevantes. Durante a interação do aluno no ambiente, o agente pode mudar o perfil do aprendiz, caso perceba alguma modificação em seu comportamento, e esta informação é passada ao *Agente Seleciona_Estratégia*. O *Agente Seleciona_Estratégia* muda a estratégia para um determinado aluno quando o *Agente Modela_Aprendiz* ou o *Agente Analisa_Aprendizagem* informar alguma alteração na aprendizagem, motivação e personalidade do aluno. Neste trabalho o *Agente Seleciona_Estratégia* seleciona uma estratégia híbrida com base em características do aluno e características das estratégias que se enquadram para um determinado perfil de aluno;

- *Agente Orienta_Aprendizagem*: busca o endereço no banco de dados e apresenta o material, conforme plano de ensino, isto é, a seqüência de ações (táticas) geradas pelo *Agente Seleciona_Estratégia*. O material de ensino, também denominado domínio do conhecimento, está armazenado sob a forma de diferentes mídias (vídeo, texto, gráfico, áudio, etc.), as quais são utilizadas para a sua apresentação. No banco de dados, encontram-se os endereços das páginas conforme uma estrutura pré-definida, a qual retrata o plano de ensino com todas as possibilidades de estratégias, onde o professor disponibilizou através do *Agente Ferramentas_Para_Professor*;

- *Agente Orienta_Avaliação*: informa ao *Agente Orienta_Avaliação* a sessão atual, informações do aprendiz e objetivo em curso. Com essas informações, o *Agente Orienta_Avaliação* busca e apresenta o material de avaliação correspondente;

- *Agente Analisa_Aprendizagem*: analisa e verifica a aprendizagem durante a interação do aprendiz e informa ao *Agente Seleciona_Estratégia*. Caso o aprendiz não estiver aprendendo, o agente seleciona outra estratégia de ensino.

Pela sua arquitetura, o ambiente possibilita a utilização de múltiplas estratégias de ensino, selecionadas em função de parâmetros que o *Agente Seleção_Estratégia* recebe de outros agentes.

Quanto à implementação, os protótipos desenvolvidos utilizaram a linguagem Java para manter a independência de plataforma e o acesso através da *Web.*

2.1.2 SEI

O sistema SEI (Sistema de Ensino Inteligente), de [TED97], é um STI voltado para o ensino de Introdução à Computação, implementado em linguagem Java, composto por uma sociedade de agentes. Além dos agentes, o SEI também conta com duas Bases de

Conhecimento: *Modelo do Estudante* e *Modelo do Domínio*. A arquitetura do ambiente, segundo [TED97], apresenta:

- Base de Conhecimento *Modelo do Estudante*: armazena informações relativas ao aluno corrente, na forma de *frames*, que são inicializados de acordo com um questionário inicial e são atualizados a cada passo da interação. São armazenadas: características cognitivas, histórico da interação, caminhos percorridos no currículo e histórico do progresso. Para acomodar as diferenças de conhecimento prévio de Computação, o *Modelo do Estudante* dispõe de três estereótipos de aluno: Usuário Especialista, Usuário Casual e Usuário Leigo. Esta atribuição inicial é feita com base na primeira interação do aluno com o sistema, sendo que posteriormente esta classificação pode ser mudada, pois tal modelo é atualizado a cada passo da interação;

- Base de Conhecimento *Modelo do Domínio:* armazena o conteúdo do curso. Este modelo está organizado como uma rede de *frames*. O sistema apresenta as informações em quatro níveis de abstração: Curso, que armazena o curso completo; Lições, que são as grandes divisões temáticas dos cursos; Tópicos, que são as unidades conceituais das lições; e Apresentações, que guardam o conteúdo que será apresentado para o aluno;

- *Agente Controlador:* gerencia a troca de informações entre os vários agentes do SEI. O *Controlador* recebe solicitações dos outros agentes do sistema e envia para aqueles que fornecem os serviços solicitados. Quando recebe as respostas, o *Controlador* as direciona para os solicitantes. Para isso, o *Controlador* mantém um registro de todos os agentes do sistema, sua localização e serviços disponíveis. Os agentes se comunicam através da troca de mensagens KQML[2] (*Knowledge Query and Manipulation Language*);

[2] KQML é uma linguagem e um protocolo de comunicação de alto nível, para troca de mensagens independentemente da sintaxe do conteúdo e da ontologia aplicável, do mecanismo de transporte (TCP/IP, SMTP, etc) e da linguagem conteúdo (SQL, Prolog, etc.). Foi desenvolvida em 1993, patrocinada pelo DARPA (*US Defense Advanced Research Projects Agency*) [SIL01].

- *Agente Estudante:* manipula informações relacionadas aos alunos e determina o perfil do aluno corrente, baseado nas informações contidas no *Modelo do Estudante*. Quando a sessão tutorial começa, o *Agente Estudante* recebe do *Controlador* a identificação do usuário, para que possa recuperar o *Modelo do Estudante* correspondente. Quando o aluno utiliza o sistema pela primeira vez, um arquivo de *Modelo do Estudante* é criado e inicializado de acordo com os resultados da interação inicial;

- *Agente Domínio:* recupera informações do *Modelo do Domínio*, conforme solicitado pelo *Controlador* e avalia as respostas do aluno aos exercícios propostos, auxiliando assim a determinar o desempenho e o perfil do aluno corrente;

- *Agente Tutor:* toma as decisões pedagógicas do SEI, determinando o *que* o SEI vai ensinar, *como* e *quando* o fará. Outra tarefa importante deste agente é determinar as estratégias de resposta ao aluno. Por exemplo, se o aluno corrente está apresentando um desempenho fraco, o *Tutor* adota uma estratégia "encorajadora", fazendo comentários a cada passo da interação, explicando suas decisões e motivando o aluno a superar suas dificuldades. Quando o aluno melhora seu desempenho, o *Tutor* modifica sua estratégia de resposta. O *Tutor* raciocina com base em informações obtidas dos modelos do *Estudante* e do *Domínio*, para determinar o que ensinar, quando e como fazê-lo. O conhecimento do agente *Tutor* é modelado através de regras de produção.

2.1.3 Eletrotutor III

O Eletrotutor III, desenvolvido por [BIC98], é a terceira versão do STI Eletrotutor, proposto originalmente por [SIL92]. Foi antecedido pelo Eletrotutor II, uma versão tutorial para a *Web* que não possue arquitetura de STI e não utiliza agentes na sua modelagem. O

Eletrotutor I foi desenvolvido em PROLOG e com proposta *stand alone*[3], sem a possibilidade de trabalhar cooperativamente.

O Eletrotutor III implementa um ambiente distribuído de ensino-aprendizagem inteligente (*Intelligent Learning Environment*) baseado em uma arquitetura multiagente, na qual os agentes possuem as seguintes características: perceber dinamicamente as condições do ambiente; tomar decisões para afetar condições do ambiente; interpretar percepções, resolver problemas, extrair inferências e determinar ações. O conteúdo disponibilizado é de Eletrodinâmica, uma subárea da Física que estuda alguns fenômenos da Eletricidade, abordando as relações entre algumas grandezas elétricas como Corrente Elétrica, Tensão, ou Diferença de Potencial, Resistência e Potência Elétrica.

Segundo [BIC98], neste ambiente é de vital importância a coordenação do comportamento dos agentes e da maneira pela qual eles compartilham seus conhecimentos, objetivos, habilidades e seus planos para, em conjunto, tomar as ações necessárias para solucionar um problema. Para que diferentes agentes autônomos possam cooperar mutuamente a fim de atingirem seus objetivos, é necessário que a sociedade possua organização (arquitetura) e comunicação. A organização diz respeito à natureza e à função da sociedade e de seus elementos constituintes e a comunicação é o principal instrumento que os agentes utilizam para desenvolver a coordenação de suas ações.

A fim de poder atuar sobre o ambiente, cada agente possui uma representação interna parcial do mundo que o cerca. Para isso, empregou-se a metáfora de estados mentais para modelar a base de conhecimento que representa os estados do ambiente onde o agente está inserido.

A sociedade de agentes proposta na terceira versão do Eletrotutor contém agentes autônomos se que comunicam uns com os outros. Cada agente possui funções e objetivos dentro de sua especialidade. A figura 6 representa a arquitetura elaborada para este ambiente.

[3] Isto é, cada aluno trabalha em uma máquina, isoladamente.

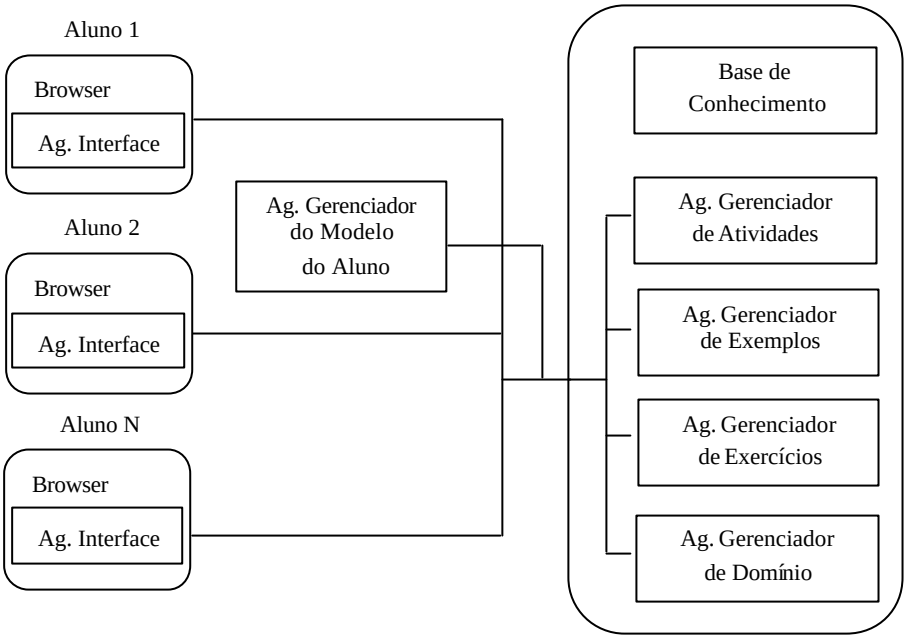

Figura 6 - Arquitetura do ambiente ELETROTUTOR III

Segundo [BOL01], os agentes do ambiente Eletrotutor III são:

- *Agente Gerenciador do Domínio*: possui a função de recuperar informações referentes ao domínio sobre cada ponto a ser apresentado ao aluno. Ele pode trocar mensagens com os *Agentes Gerenciador Modelo do Aluno* e *Agente Interface*;

- *Agente Gerenciador de Exercícios*: possui a tarefa de propor exercícios e avaliar respostas do aluno. A troca de mensagens pode ser feita com o *Agente Gerenciador Modelo do Aluno* ou com o *Agente Interface*;

- *Agente Gerenciador de Exemplos*: responsável pela tarefa de propor exemplos ao aluno. Pode trocar mensagens com os *Agentes Gerenciador Modelo do Aluno* e *Agente Interface*;

- *Agente Gerenciador de Atividades*: responsável pela tarefa de propor atividades extras ao aluno. Pode trocar mensagens com os *Agentes Gerenciador Modelo do Aluno* e *Agente Interface*;

- *Agente Gerenciador do Modelo do Aluno*: responsável por construir e manter uma base de conhecimento que modele o estado cognitivo dos alunos que estejam conectados ou que tenham estado conectados ao sistema. Quando a sessão tutorial inicia, ele recebe informação do *Agente Interface* sobre a identificação do aluno, para que possa recuperar o Modelo do Aluno correspondente e conforme o andamento das lições decidir qual a estratégia de ensino a ser utilizada com este aluno;

- *Agente Interface*: possui como função o controle do *browser* no ambiente do aluno. As mensagens são recebidas e enviadas a todos os agentes. Este agente não possui inteligência, apenas repassa as atividades do aluno na interface do sistema.

O ambiente foi desenvolvido em Java, a fim de manter a independência entre plataformas e o acesso através da *Internet.* Os agentes trocam mensagens através do *Remote Method Invocation* (RMI). As mensagens trocadas seguem um protocolo baseado na linguagem KQML. A base de conhecimento está armazenada em um banco de dados relacional, o Oracle (versão 8.0.4).

2.1.4 MuTanIS

O MuTanIS, de [AZE99], é um STI para ajudar o aprendiz a assimilar os conceitos de um determinado domínio. Este domínio pode ser alterado, dentro do sistema, a partir de um ambiente de autoria, criado em conjunto com o MuTanIS. O ambiente foi construído para o domínio de Conceitos de Orientação a Objetos para cursos de graduação de Ciência da Computação.

Sua arquitetura, apresentada na figura 7, é constituída de uma sociedade de agentes inteligentes, onde cada um destes possui suas tarefas e se comunica com os demais agentes a fim de obter ou fornecer informações. Os agentes podem estar localizados em um único computador ou distribuídos através da Internet. Para realizar a troca de informações entre os agentes, foi utilizada a linguagem KQML.

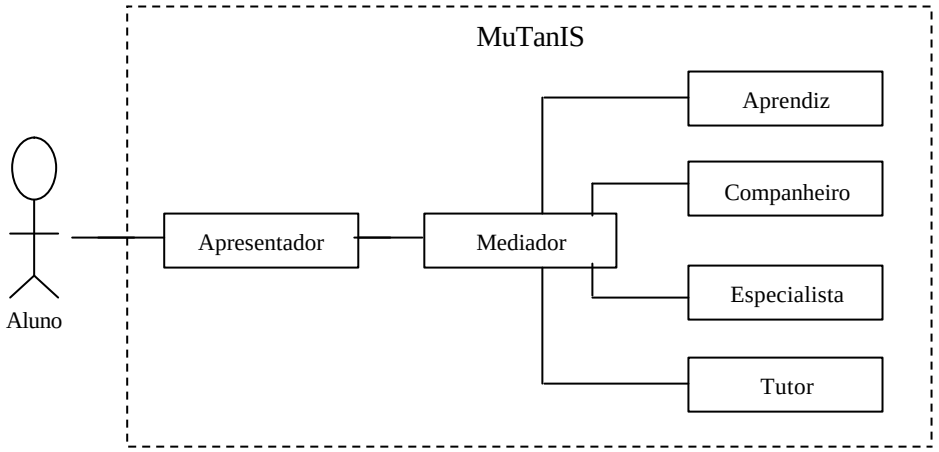

Figura 7 - Arquitetura do ambiente MuTanIS

Segundo [AZE99], o agente *Apresentador* tem a função de realizar a interface do STI com o usuário, oferecendo todos os serviços relativos à apresentação de informações ao usuário e todos os serviços relativos à obtenção de informações fornecidas pelo mesmo.

O agente *Mediador* atua como um elo de ligação entre todos os agentes, fazendo um roteamento de mensagens entre eles. Quando o STI for inicializado, todos os agentes devem enviar ao *Mediador* os serviços que eles oferecem (as tarefas que eles podem executar) e as necessidades de cada um (as informações de que precisam). Assim, o *Mediador* será capaz de saber quais os serviços oferecidos por cada agente e quais as necessidades de cada um.

O agente *Aprendiz* é responsável por criar e manter modelos dos aprendizes que utilizam o STI e fornecer informações ao agente *Tutor*, para identificação de deficiências no modelo do aprendiz e para controle do progresso do usuário. Para realizar estas tarefas, o agente possui uma base de conhecimento com as características do aprendiz, que constituem o modelo do aprendiz. Este modelo é construído dinamicamente, à medida que o usuário utiliza o STI.

O agente *Companheiro* age como um companheiro de estudo do usuário, podendo atuar de diversas formas, como por exemplo, fornecendo dicas para auxiliar o usuário na resolução de problemas, na realização de tarefas e/ou nas questões elaboradas pelo STI. Estas dicas só são apresentadas caso o aprendiz solicite-as. Assim como o agente *Aprendiz*, este também possui uma base de conhecimento, mas, neste caso, com as dicas referentes a todas as perguntas que podem ser feitas ao aprendiz.

O agente *Especialista* contém uma representação do conhecimento a ser comunicado ao usuário, agindo como a fonte do conhecimento a ser apresentado. Este agente é capaz de gerar explicações, respostas, tarefas e questões que devem ser apresentadas ao usuário.

O agente *Tutor* projeta e regula as interações instrucionais com o usuário. É quem decide as atividades pedagógicas que serão utilizadas: avisos, suporte, explicações, tarefas diferentes, entre outras. O agente *Tutor* determina o que o STI vai apresentar ao usuário, como isso deve ser realizado e quando deve ser feito. Para tal, age de acordo com informações obtidas dos agentes *Aprendiz* e *Especialista*.

2.1.5 AmCorA

O AmCorA (Ambiente Cooperativo de Aprendizagem), proposto por Menezes [MEN99], é um conjunto de sistemas idealizado para apoiar a aprendizagem em ambiente

cooperativo utilizando a Internet. A proposta segue a abordagem construtivista e pretende prover ao aluno condição para que ele resolva os problemas de forma cooperativa interagindo com agentes humanos e artificiais.

Para facilitar a interação entre os agentes envolvidos, o ambiente utiliza-se de ferramentas inteligentes para facilitar o desenvolvimento de tarefas voltadas à educação, as quais possuem uma arquitetura baseada em multiagentes que possibilita a descrição e realização de agentes reais e virtuais.

Algumas destas ferramentas desenvolvidas com a orientação de Menezes, conforme sua relevância para esta pesquisa, são apresentadas abaixo.

2.1.1.1 AmCorA-ES

O AmCorA-ES, proposto por [BRI00], é um ambiente multiagente para ensino de Engenharia de *Software* através da Internet, que possui os seguintes objetivos: fornecer a aquisição de informação e material bibliográfico; prover interação entre os diversos colaboradores; permitir aos aprendizes demonstrarem o conhecimento adquirido, bem como proporcionar uma auto-avaliação sobre este conhecimento; promover palestras e facilitar a comunicação entre profissionais. A modelagem do ambiente utilizou uma metodologia de análise e projeto de sistemas para a organização da estrutura de agentes que deveria ser criada. Esta metodologia descreve os papéis dos agentes, bem como a maneira como estes os desempenham no ambiente.

Durante a fase de análise, foram gerados dois modelos abstratos: um modelo de papéis, que identificou os papéis básicos do sistema, relacionados às suas responsabilidades e funções; e um modelo de interação, que definiu os protocolos de comunicação baseados nos papéis identificados, dizendo respeito aos objetivos das trocas de mensagens entre estes.

Na fase de projeto, os dois modelos foram mapeados em um modelo de agentes (conjunto de papéis) e um modelo de trabalhos (vários trabalhos podem estar associados a um único papel).

Segue uma breve descrição das responsabilidades dos agentes pertencentes a arquitetura do AmCorA-ES:

- *Estudantes*, *Colaboradores*, *Professores* e *Administrador*: representados por agentes humanos que interagem direto com os agentes de interface (*Atendentes*);

- *Atendentes*: agentes de interface responsáveis pela interação com os agentes humanos;

- *Comunicador*: responsável pelo fluxo de mensagens do ambiente;

- *Gerador de clones*: responsável pela geração de agentes que representem cada um dos usuários participantes do ambiente, ou seja, de clonar um agente humano em um agente virtual;

- *Clone*: armazena, trata e retorna informações sobre um usuário;

- *Gerenciador de interações* síncronas: responsável pelo armazenamento e manipulação das interações síncronas entre os agentes humanos;

- *Gerenciador de interações assíncronas*: trabalha de forma análoga ao agente *Gerenciador de interações síncronas*, mas tratando as interações assíncronas;

- *Compositor de grupo*: gerencia os grupos, seus perfis e respectivas áreas de estudo;

- *Gerador de gerenciador de grupo*: para cada grupo formado, é gerado um agente para gerenciá-lo;

- *Gerenciador de grupo*: monitora as atividades de um grupo;

- *Gerenciador de informações*: responsável por realizar o *backup* das informações nas suas respectivas tabelas;

- *Gerenciador de treinamento*: armazena e gerencia cursos, palestras e outras informações apresentadas aos colaboradores, professores e instrutores.

2.1.1.2 *Moonline*

O *Moonline* (Monitoria *On-line*) é um ambiente modelado com agentes, baseado na *Web*, para Apoio à Aprendizagem. Foi desenvolvido por [GAV00]. Seu principal objetivo é permitir que os membros de uma comunidade de aprendizagem virtual possam interagir para esclarecer suas dúvidas, fazendo perguntas uns aos outros e obtendo respostas. Essas perguntas estão associadas a contextos específicos, que são blocos de informação dentro das notas de aula. Os aspectos principais do *Moonline* são:

- O ambiente está centrado em um curso formal, que leva em consideração não somente as disciplinas que um aluno está cursando em determinado período, mas também aquelas que já foram cursadas e aquelas que virá a cursar;

- Disponibiliza um ambiente personalizado para cada um de seus usuários, oferecendo facilidades para que estes possam realizar suas tarefas da melhor maneira possível. Esses usuários são classificados em alunos, monitores e professores;

- Para cada um destes ambientes personalizados existem agentes inteligentes atuando como assistentes pessoais, realizando diversas operações com o objetivo de facilitar a realização das tarefas desempenhadas por cada um dos usuários.

O *Ambiente do Aluno* possui como objetivo melhorar o processo de ensino-aprendizagem do aluno, permitindo que ele possa esclarecer dúvidas de qualquer natureza, tanto relacionadas a seu curso como a alguma disciplina específica. O aluno também pode registrar suas anotações pessoais; pesquisar sobre perguntas/respostas feitas em relação aos assuntos, bem como elaborar novas perguntas; consultar o material bibliográfico relacionado às disciplinas que está cursando; e comunicar-se com outros alunos, professores ou monitores de forma síncrona ou assíncrona.

Existem três agentes envolvidos na realização das atividades desempenhadas pelos alunos, são eles:

- O *Agente Recuperador de Perfil de Pergunta*, que tem a função de receber uma pergunta de um aluno, extrair seu perfil e verificar se a pergunta pertence ao contexto ao qual está ligada;

- O *Agente Recuperador de Conjunto de Perguntas/Respostas Similares*, o qual compara o perfil da pergunta de entrada com os perfis das perguntas existentes na base de conhecimento para um dado contexto e apresenta ao aluno o conjunto de perguntas/respostas encontrado para que este verifique se alguma irá lhe ajudar a solucionar a dúvida;

- O *Agente que Analisa o Estágio de Aprendizagem do Aluno*, que tem por responsabilidade averiguar se algumas das perguntas selecionadas anteriormente foi elaborada pelo próprio aluno. Caso tenha sido, indaga o aluno sobre porquê refazer a pergunta e atualiza sua base de crenças sobre o aluno baseado no resultado desta interação.

A figura 8 ilustra como esses agentes interagem entre si.

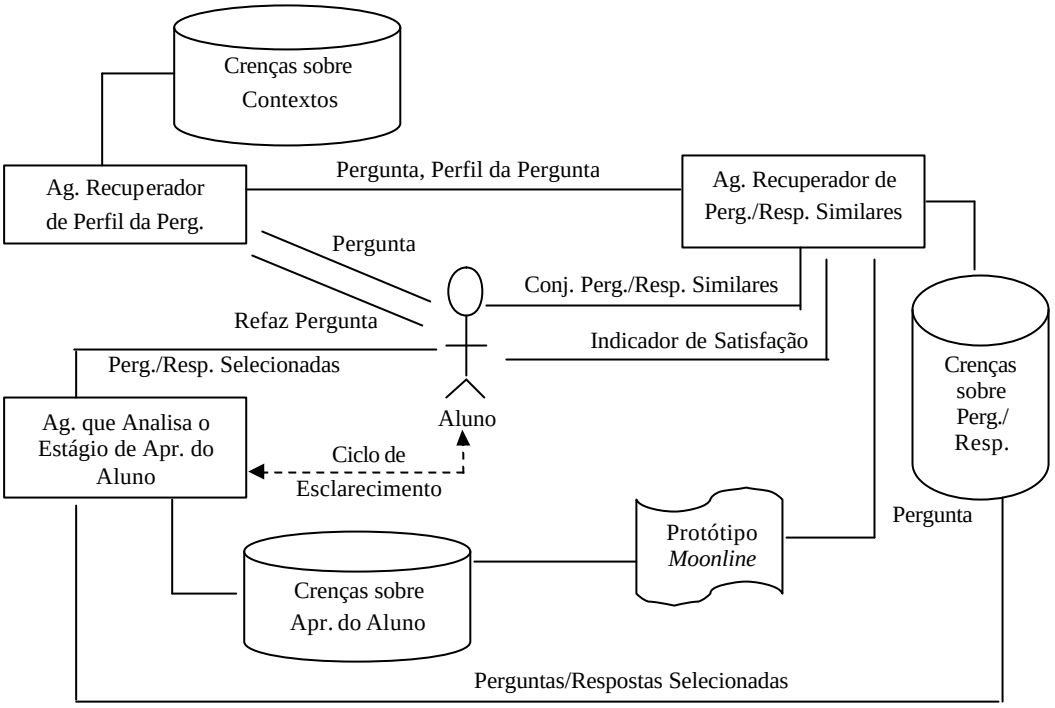

Figura 8 - Interação entre os agentes do Ambiente do Aluno

Quanto ao *Ambiente do Monitor*, este possui agentes para facilitar a realização das tarefas dos monitores, tais como responder perguntas aos alunos, organizar documentos, interagir com os professores, entre outras. Além disso, cada monitor possui sua área individual, para poder organizar seus documentos pessoais, bem como pode interagir com os demais monitores, visto que uma disciplina pode ter mais de um monitor. Os agentes disponíveis para este ambiente são, segundo [GAV01]:

- O *Agente de Interface,* que possui o perfil do monitor e atualiza-o sempre que necessário. É o responsável por se comunicar com o monitor e delegar funções aos demais agentes. Também se comunica com os agentes do *Ambiente do Aluno* para buscar informações sobre o aluno caso o monitor solicite;

- O *Agente Classificador de Perguntas*, que tem por função classificar as perguntas que chegam para o monitor, agrupando-as por tópicos específicos;

- O *Agente Resposta*, o qual auxilia o monitor a responder as dúvidas dos alunos, buscando outras fontes de informação caso o monitor tenha dificuldades de respondê-las ou enviando as perguntas para o professor responder;

- O *Agente Organizador de Documentos*, que organiza a área de trabalho do monitor, separando a documentação por disciplina, grupo de trabalho e/ou aluno. Também permite que o monitor registre as notas dos alunos e envie-as aos professores.

Por fim, o *Ambiente do Professor* possui agentes que realizam praticamente as mesmas tarefas desempenhadas para os monitores, sendo que os professores só recebem as perguntas que os agentes ou monitores humanos não conseguiram responder.

O ambiente foi disponibilizado para os alunos, monitores e professores das disciplinas introdutórias de programação, oferecidas pelo Departamento de Informática da Universidade Federal do Espírito Santo (UFES). Atualmente, está em fase de alteração de características do ambiente devido às observações decorrentes de sua utilização. Estão previstas pequenas alterações a fim de melhorar à performance dos agentes.

2.1.1.3 Ambiente para apoio à aprendizagem de programação

Este sistema, desenvolvido por [CHA01], é um ambiente de suporte à aprendizagem de programação utilizando o paradigma multiagente, possuindo como enfoque principal a interação com os estudantes e a identificação semi-automática de grupos ou padrões de soluções de problemas propostos, visando facilitar o aprendizado e fornecendo o *feedback* adequado. O primeiro paradigma de programação visado é o funcional, mas a arquitetura proposta pode ser aplicada a qualquer outro paradigma de linguagem. Possui como características:

- Baseia-se na tecnologia de agentes para auxiliar o professor na identificação dos erros cometidos pelos estudantes durante a construção de soluções para os problemas propostos;

- Pretende contribuir para a melhoria quantitativa e qualitativa do *feedback* que é fornecido ao aluno;

- Os agentes detectam padrões de erros nas respostas e enfatizam a utilização do mínimo de recursos da linguagem, necessário para se chegar a uma solução correta;

A figura 9 mostra a interação no ambiente proposto, com agentes reais representados por humanóides e agentes virtuais representados por humanóides com olhos na boca.

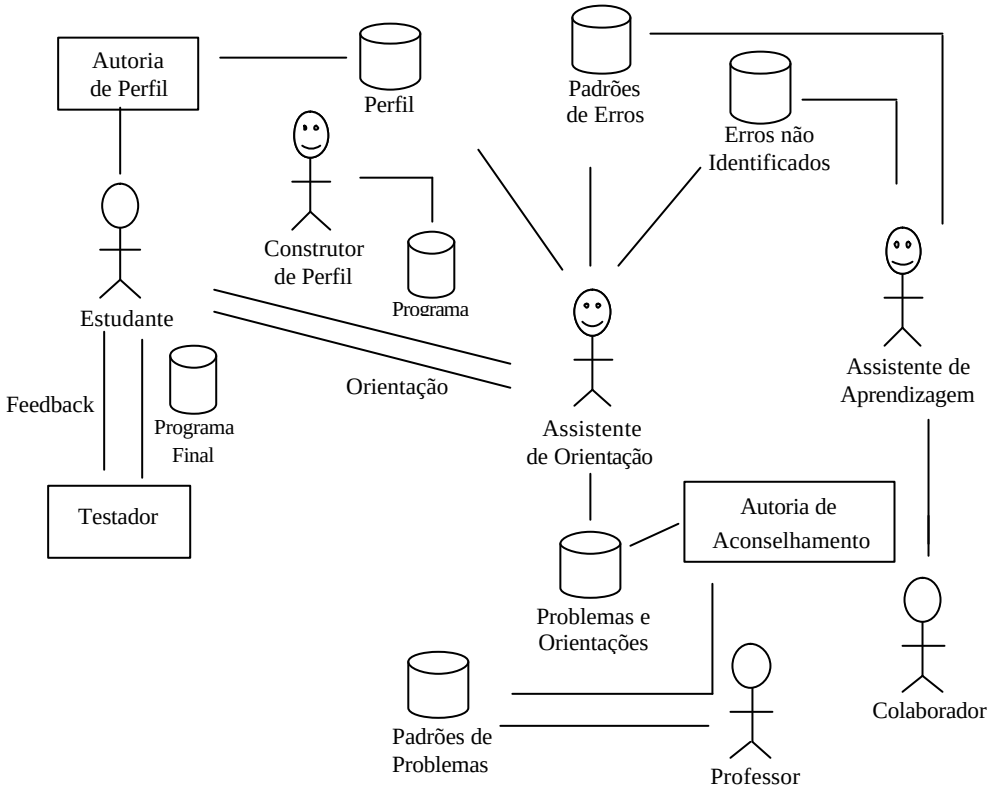

Figura 9 - Arquitetura do ambiente para apoio à aprendizagem de programação

Ao interagir com o ambiente o estudante constrói seu perfil através da ferramenta de autoria de perfil e envia um programa feito em seu editor de programação funcional ao Assistente de Orientação. O assistente, baseado no perfil do estudante e na base de padrões de erros, envia uma orientação ao Estudante informando se o programa contém ou não erros. Caso seja identificado um erro, o assistente fornecerá um *feedback* ao estudante e uma mensagem sugerindo que este último repense a solução, tentando encontrar um outro caminho.

Se o Assistente de Orientação não detectar mais nenhum erro no programa do estudante, ele sugere que o programa seja enviado à ferramenta Testador para ser avaliado.

2.1.1.4 SAmbA

O SambA (Suporte à Cooperação em um Ambiente de Aprendizagem para Programação) é um ambiente cooperativo de aprendizagem, desenvolvido por [NOB02], para cursos baseados em soluções de problemas, onde um curso de Introdução à Programação foi usado como estudo de caso. Através da *Web* o usuário pode acessar remotamente a base de dados. Tendo então, disponibilizadas as aulas, questões e ferramentas voltadas para o trabalho cooperativo.

O ambiente oferece áreas individualizadas para alunos e professores, sendo que cada uma delas disponibiliza apenas as ferramentas relacionadas às atividades do usuário ativo.

Quanto a estas atividades, o *Ambiente do Aluno* permite que estes desenvolvam e apresentem uma solução para um problema dado pelo professor, teçam comentários sobre soluções apresentadas por outros grupos, esclareçam dúvidas e interajam em grupos para desenvolvimento de projetos ou troquem informações entre colegas.

Já o *Ambiente do Professor*, permite que este disponibilize material didático (conteúdo de aulas, questões, etc.), selecione enunciados de questões e soluções de problemas a serem enviadas aos alunos, faça considerações (comentários) sobre as soluções obtidas e submeta-as aos alunos, acompanhe o processo de aprendizagem dos grupos e/ou alunos individualmente e, por fim, gerencie a formação dos grupos de trabalho.

Para o desenvolvimento do ambiente descrito, foi elaborada uma arquitetura multiagente, onde se utilizou uma metodologia baseada em papéis, denominada GAIA, para identificar-se os agentes, seus papéis e interação entre os agentes do sistema. A figura 10 apresenta a arquitetura geral do sistema.

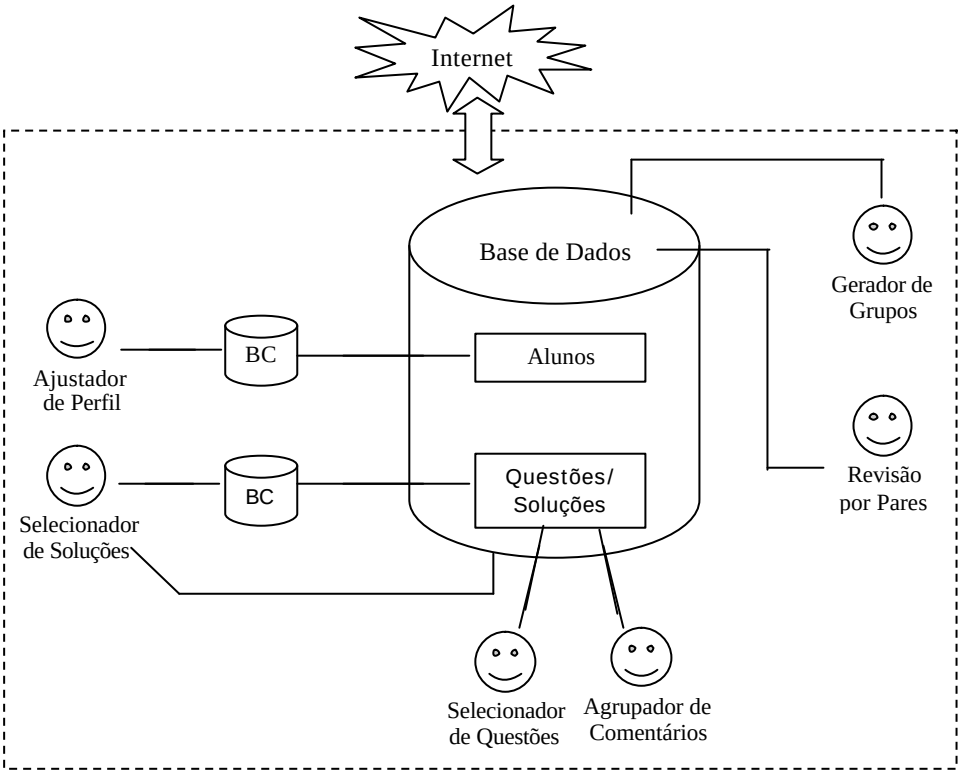

Figura 10 - Arquitetura do ambiente SAmbA

Segundo [NOB02], o agente *Gerador de Grupos* tem como responsabilidade gerar grupos de alunos seguindo critérios estabelecidos pelo professor. O agente *Ajustador de Perfil* traça o perfil do aluno, atualizando-o através do acompanhamento das atividades

realizadas pelo mesmo. O agente *Selecionador de Questões* seleciona exercícios adequados às necessidades dos alunos, segundo seus perfis de aprendizagem. Já o agente *Selecionador de Soluções*, seleciona soluções a serem analisadas pelo professor. Esta análise é realizada com o auxílio de uma base de conhecimento composta por um conjunto de esqueletos de soluções e meta-informações elaboradas pelo próprio professor.

Quanto ao agente *Revisão por Pares*, distribui as soluções encontradas pelos alunos ou grupos para que estes possam analisar e criticar as soluções desenvolvidas por outros colegas. E, por fim, o agente *Agrupador de Comentários*, que permite o agrupamento de comentários sobre as soluções apresentadas, buscando proporcionar ao aluno a obtenção do refinamento da sua solução [NOB02].

Utilizou-se na implementação do protótipo o servidor de banco de dados Interbase e Delphi 6.0 para implementação da aplicação para Internet, com o auxílio do *software* DreamWaver[4] 4.0 para *design* da interface.

2.1.6 LeCS

O LeCS (*Learning from Case Studies*), de [ROS00], é um sistema inteligente que possui uma arquitetura multiagente para aplicações de EAD. O ambiente dá suporte à aprendizagem colaborativa através da *Web*, usando o método de ensinar com estudos de casos (*Case Based Resoning*). O sistema inclui as ferramentas necessárias para desenvolver a solução para um determinado caso (a ser modelado pelo professor) e desempenha funções de apoio ao processo de aprendizado dos alunos.

O LeCS foi desenvolvido pelo Departamento de Computação e Estatística da Universidade Federal de Santa Catarina (UFSC), em conjunto com a Universidade do Vale do Itajaí (Univali) e a Unidade de Aprendizagem Baseada em Computador da Universidade de Lees (Leeds – UK).

[4] Fabricado por Macromedia, Inc (http://www.macromedia.com/software/dreamweaver).

A figura 11 mostra a interface com o usuário do sistema LeCS, a qual é composta por uma lista de participantes, um *browser*, uma área para a representação gráfica da solução, um *chat*, um editor de texto colaborativo e uma área de intervenções do sistema. O *browser* é usado para acessar as páginas *Web* que apresentam o conteúdo dos estudos de caso e os passos para guiar o desenvolvimento da solução deste casos. O *chat* é o lugar onde pode acontecer as discussões sobre o assunto. O editor de texto é usado para responder as questões colocadas em cada etapa.

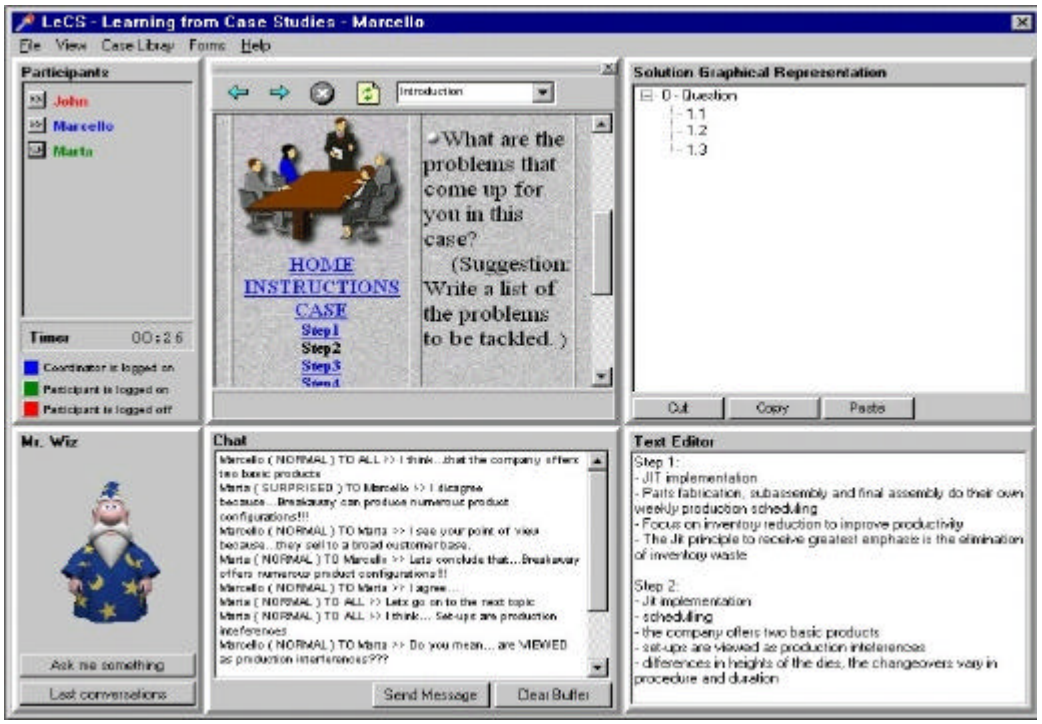

Figura 11 - Interface com o usuário do sistems LeCS

Quanto à comunicação entre os agentes, segundo [BOL01], foi definido que ela não acontece diretamente entre os mesmos, mas sim através de um *Facilitador* (*Facilitator*). O *Facilitador* é um programa especial, implementado como um agente, que guarda a informação sobre cada agente no sistema e é responsável pelo roteamento dessas mensagens, trabalhando como um *broker*. Há dois banco de dados implementados: o primeiro é o próprio *Facilitador* que armazena todas informações necessárias para rotear as

mensagens e o segundo é um mecanismo de *logs* que armazena todas as trocas de mensagens. A figura 12 apresenta a arquitetura do sistema e a estrutura de comunicação usada entre os agentes.

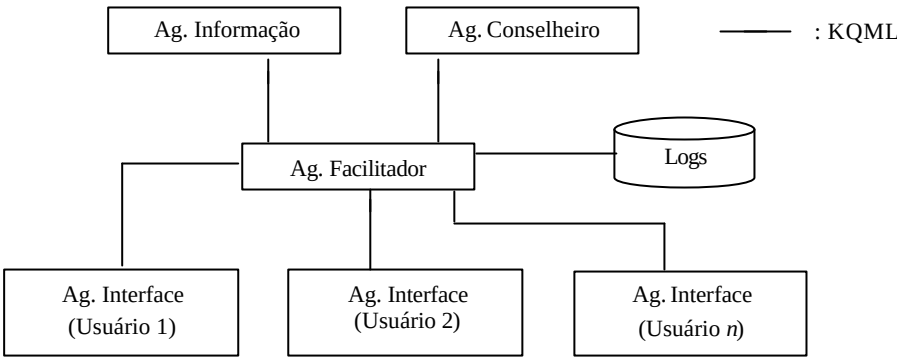

Figura 12 - Arquitetura do sistema LeCS

Segundo [ROS00, Apud in BOL01], a arquitetura inclui três classes de agentes: *Agente Interface*, *Agente de Informação* e *Agente Conselheiro*.

- *Agente Interface* (*Interface Agent*): todas as intervenções do sistema são apresentadas através deste agente. O *Agente Interface* armazena as informações sobre os usuários individualmente: o que é digitado no editor de texto, o número de participantes do *chat*, a etapa em que o indivíduo esta trabalhando, a resposta a cada questão e o tempo gasto em cada etapa. Baseado nestas informações, o *Agente Interface* faz intervenções sobre tempo e participação. Utiliza um banco de dados que contém informações adicionais: o endereço dos agentes, o nome pelo qual ele é conhecido, etc;

- *Agente Informação* (*Information Agent*): lida com o domínio e o conhecimento pedagógico. O *Agente Interface* e o *Agente Conselheiro* podem acessar o *Agente Informação*. A informação armazenada por este agente é dividida em duas diferentes categorias: material didático e base de conhecimento. O material didático consiste de páginas HTML (*Hipertext Markup Language*), imagens e textos. A base de conhecimento refere-se ao domínio e representação das soluções dos casos desenvolvidos. Além disso, este agente armazena as interações do *chat*;

- *Agente Conselheiro* (*Advising Agent*): enquanto que o *Agente de Informação* tem acesso ao banco de dados e as bases de conhecimento, o *Agente Conselheiro* tem um mecanismo para entender sobre suas ações e para reconhecer situações onde algum suporte é necessário. O *Agente Conselheiro* executa o algoritmo para gerar uma árvore de soluções com a informação fornecida pelo *Agente Interface* e retorna a representação para este agente. Quando aplicável, ele gera uma intervenção sobre o mal entendimento de caso de estudo e envia uma requisição para uma intervenção do *Agente Interface*.

A comunicação dos agentes está baseada na *Agent Comunication Language* (ACL) e as mensagens trocadas entre os agentes usam a linguagem KQML.

O sistema foi implementado em linguagem Delphi, numa arquitetura cliente-servidor. O servidor hospeda as seções, as quais são associadas com um grupo de estudantes trabalhando cooperativamente. O cliente roda na máquina do estudante permitindo dessa forma que vários estudantes possam participar de uma mesma sessão.

2.1.7 TUTA

O ambiente TUTA (TUTor baseado em Agentes), desenvolvido por Silva [SIL00, SIL00a], permite o aprendizado cooperativo de um grupo de alunos geograficamente distantes através de uma série de sessões de ensino-aprendizagem. O professor especifica o curso, constituído de um conjunto de sessões, e acompanha os alunos conectados ao ambiente. Diversas estratégias didáticas podem ser especificadas pelo professor sem que seja necessário alterar-se os códigos dos programas implementados pelo TUTA. Como estudo de caso, foi considerado o domínio da Orientação a Objetos.

O ambiente do tutor possui os elementos de sua arquitetura representados por agentes implementados em Java, inseridos no contexto da Arquitetura de uma Classe

Virtual Adaptativa (ACVA)[5]. O acoplamento das camadas *SF_Grupo* e *SF_Básicos* permite que se tenha um sistema tutor dedicado a aprendizagem de um domínio em particular e associado a um nível de conhecimento. Segundo [SIL00], os principais elementos que devem fazer parte dessa camada são: o *servidor de entidades didáticas* associado a um domínio; o *comportamento dos alunos* durante a sessão, que são considerados através dos *perfis individuais* e *de grupo*; o funcionamento do tutor do ponto de vista didático, o qual depende da recuperação e execução de *estratégias didáticas* e a interação amigável dos estudantes como sistema tutor, que é controlada pela *interface*. A figura 13 apresenta a arquitetura do ambiente TUTA, em função dos elementos e agentes que a compõem.

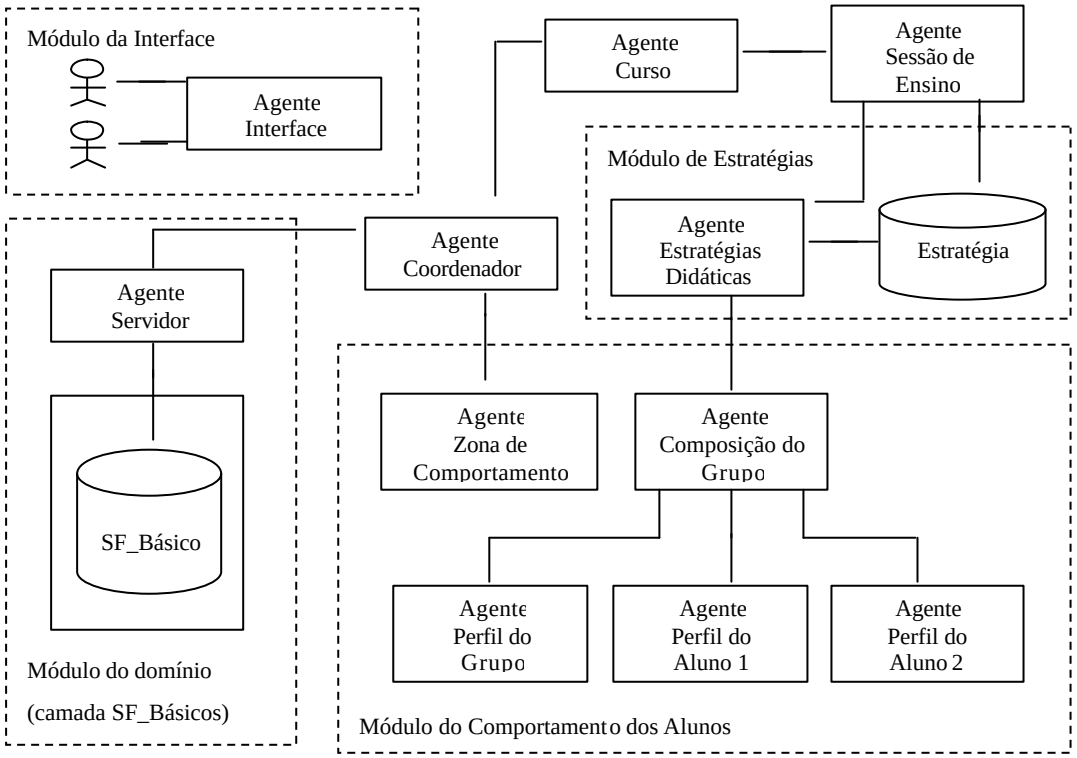

Figura 13 - Arquitetura do ambiente TUTA

[5] A ACVA considera que uma classe virtual é composta de diversos grupos heterogêneos, onde cada grupo permite que um conjunto de estudantes de diferentes níveis de conhecimento participe de uma sessão de ensino-aprendizagem. Segundo [SIL00], a ACVA é composta das seguintes camadas: *de suporte*, que armazena e controla a comunicação entre os participantes; *de serviços de formação básicos* (SF_Básicos), que permite a cada controlador de grupo (tutor) utilizar recursos em comum; *de serviços de formação de grupo* (SF_Grupo), que é a responsável por verificar se o conhecimento do estudante está associado àquele previsto para o grupo ao qual pertence; e *de serviços de formação da classe virtual* (SF_CV), que controla as mudanças de grupo pelos estudantes conforme seus níveis de conhecimento.

2.1.8 Um Ambiente Inteligente para Aprendizagem Colaborativa (AIAC)

O objetivo do AIAC, de [AZE01], é oferecer suporte à aprendizagem colaborativa, onde aprendizes, reunidos em grupos, trabalharão juntos em tarefas de aprendizagem. O ambiente poderá ser utilizado em uma rede local de computadores ou através da Internet, possibilitando que aprendizes e professores possam estar fisicamente localizados em lugares diferentes.

O AIAC é baseado em uma arquitetura multiagente, onde existe uma sociedade de agentes, cada um desses com suas tarefas específicas e comunicando-se entre si. Os agentes integrantes da arquitetura são descritos a seguir:

- *Apoio*: atua junto aos aprendizes ou professor, ajudando-os nas atividades envolvidas no processo de aprendizagem, fornecendo ferramentas de apoio, tais como agenda, troca de mensagens, etc;

- *Aprendiz Individual*: determina o perfil de cada aprendiz;

- *Aprendiz em Grupo*: com finalidade semelhante a do agente *Aprendiz Individual*, mas determinando o perfil de cada grupo;

- *Companheiro*: coopera e promove a colaboração entre os aprendizes de um grupo;

- *Observador*: fornece ao professor as informações necessárias sobre o desempenho dos aprendizes;

- *Especialista*: manipula as informações sobre o domínio para o qual foi construído o ambiente;

- *Tutor*: determina qual conteúdo será abordado, como será esse processo e quando ele será realizado. Também é responsável por avaliar o desempenho de cada aprendiz;

- *Apresentador*: responsável pela interface com o grupo de aprendizes;

- *Mediador*: gerencia a troca de mensagens entre os agentes, bem como identifica qual agente é responsável por responder a um serviço solicitado.

A figura 14 apresenta a interação entre os agentes do AIAC.

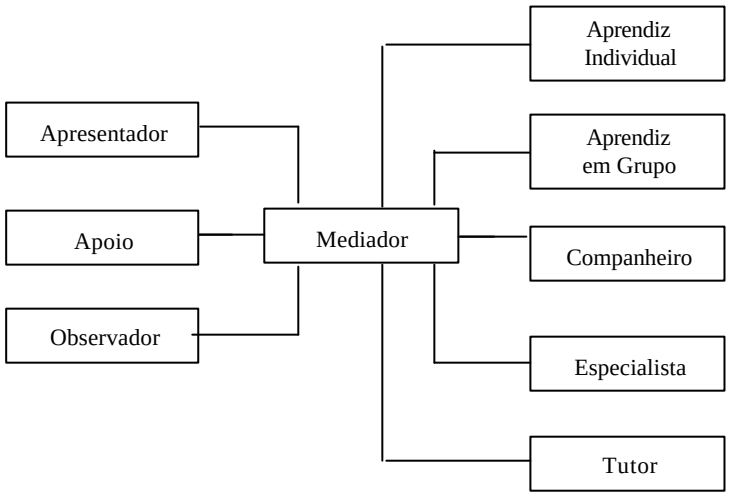

Figura 14 - Interação dos agentes do AIAC

Selecionou-se para implementação do protótipo do ambiente a linguagem Java devido às suas características de portabilidade e segurança. Em conjunto, também foi selecionada a ferramenta para construção das bases de conhecimento dos agentes, denominada JESS (*Java Expert System Shell*). Definiu-se que os agentes se comunicarão através da troca de mensagens KQML.

2.1.9 SEMEAI

O ambiente SEMEAI (SistEma Multiagente de Ensino e Aprendizagem na Internet), de [GEY01], tem como objetivo promover o aprendizado à distância, usando a tecnologia

de agentes para adaptar-se às características do aprendiz, procurando reproduzir condições ideais para o desenvolvimento de uma aprendizagem plena. A idéia fundamental do modelo do SEMEAI é conceber um ambiente de tutoria aplicável a ambientes distribuídos que permita, em interações com o aluno, auxiliando assim no processo de EAD.

O ambiente disponibiliza três atividades fundamentais em sistemas de tutoria:

- A adaptabilidade ao perfil do aluno;
- A seleção automática de estratégias de ensino adequadas, conforme as características psico-pedagógicas do aprendiz;
- A personalização do currículo de ensino.

A arquitetura do SEMEAI é multiagente. Cada agente possui determinadas funções dentro do contexto do ambiente. A figura 15 apresenta a arquitetura proposta e as interações entre os agentes do sistema, indicadas através das setas.

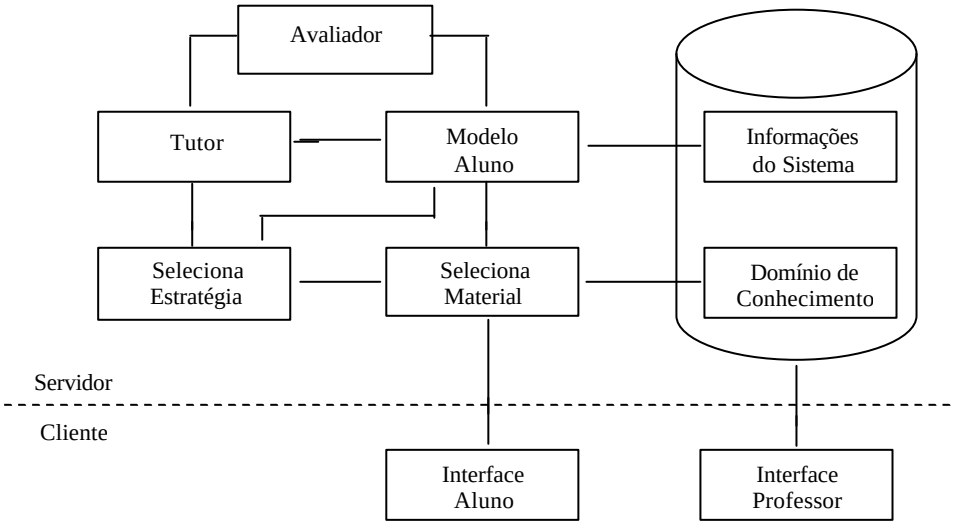

Figura 15 - Arquitetura do ambiente SEMEAI

Segundo [GEY01], o agente *Modelo Aluno* influencia as atividades de seleção de estratégias, as quais são de responsabilidade do agente *Seleciona Estratégia*. Ajuda a determinar o perfil do aluno e armazena-o.

O agente *Tutor* é responsável por encaminhar a geração da seqüência das aulas para o *Modelo do Aluno*, assim como por encaminhar o uso do método mais provável de ensino para o mesmo, a partir das informações recebidas dos agentes *Interface* e *Modelo Aluno*.

A função do agente *Avaliador* é auxiliar o processo de ensino qualitativo, avaliando os resultados das atividades oferecidas ao aluno durante sua interação com o sistema. Revisa os métodos de ensino utilizados em determinado momento pelo *Tutor*, de forma a identificar a necessidade de alterar o perfil do aluno conforme seu grau de aproveitamento no andamento do curso.

O agente *Seleciona Estratégia* possui a capacidade para decidir pela mudança de método dentro de uma estratégia, ou pela mudança da própria estratégia para criação das seqüências de ensino. Durante a escolha ou reavaliação de uma estratégia, o agente deve procurar o esgotamento das possibilidades da estratégia atual.

A tarefa do agente *Seleciona Material* consiste em criar roteiros para exploração das lições armazenadas no repositório do sistema de acordo com o método informado. Estes roteiros ou seqüências são os planos pelos quais pretende-se apresentar o conteúdo aos alunos, colaborando em seu processo de aprendizagem.

A interface é dividida em dois agentes básicos, *Interface Aluno* e *Interface Professor*, responsáveis pela interação direta com o aluno e com o professor ou especialista, respectivamente.

O agente *Interface Aluno* expõe os conteúdos selecionados ao aluno. Também é responsável por sinalizar o início do processo de tutoria após a conexão com o ambiente.

A principal função do agente *Interface Professor* é servir como assistente às atividades de inserção, revisão e exclusão de conteúdos no repositório de informações do sistema. Auxilia o professor na classificação dos conteúdos de acordo com categorias pré-definidas e configuráveis, bem como possui métodos para manipular o conhecimento armazenado na base.

No domínio de conhecimento (repositório de dados) estão armazenadas todas as informações, como conteúdos e exercícios, relativos aos cursos disponibilizados pelo sistema. O repositório utiliza o esquema de armazenamento baseado em *frames.*

O autor considera que, por possuir uma arquitetura genérica, o ambiente proporciona independência de domínio de conhecimento, podendo o mesmo ser utilizado em diferentes áreas de ensino.

Quanto à implementação, tem-se as seguintes características:

- O servidor *Web* adotado foi o Apache, executado sobre o sistema operacional Linux;
- A tecnologia base escolhida para desenvolvimento das interfaces do sistema foi JSP (*Java Server Pages*), a qual é uma linguagem embutida em servidor com capacidade para gerar páginas HTML dinamicamente. Possibilita acessar JavaBeans™;
- Possui adicionado ao sistema o JDK 1.3 (*Java Developer Kit*) e o servidor JSP Jakarta Tomcat que funciona em conjunto com o Apache;
- Para o armazenamento de dados, a opção foi o servidor PostgreSQL™;
- Para o lado cliente, tem-se a opção de utilizar o *browser* Internet Explorer ou Netscape Navigator, ambos com suporte a Java e JavaScript;

- O sistema possui uma camada de *software* base que oferece recursos para a criação dos agentes e comunicação entre os mesmos. Esta camada, desenvolvida inteiramente em Java, é o suporte básico para todas as tarefas desempenhadas pelos componentes do sistema.

2.2 Ambientes desenvolvidos com tecnologias diversas

2.2.1 Curso individualizado da linguagem de programação C

O sistema hipertexto proposto por [KAY94], para ensino da linguagem de programação C, utiliza o *browser* Mosaic e possui as seguintes características:

- Ao iniciar, o usuário (aluno) é questionado sobre seu conhecimento de programação. O material apresentado inicialmente é baseado nas respostas fornecidas pelo usuário. Este material do curso é individualizado para cada aluno;

- Um modelo de aluno mantém informações sobre o mesmo, armazenando seu conhecimento e suas preferências;

- Possui estereótipos de aluno e complementa o modelo de aluno com as informações pessoais fornecidas pelos próprios;

- Define a ordem de apresentação do conteúdo conforme o conhecimento do aluno;

- Possui diferentes formas de apresentar o mesmo conteúdo. Por exemplo, um mesmo conteúdo pode ser apresentado sob forma de conceito ou exemplos para um determinado aluno, conforme sua preferência;

- Novos exercícios são propostos conforme o desenvolvimento das atividades;

- Caso a resposta apresentada pelo aluno não seja considerada suficiente, a seção do curso é repetida com exercícios semelhantes;

A arquitetura do sistema é ilustrada na figura 16 e seus respectivos módulos são explicados a seguir.

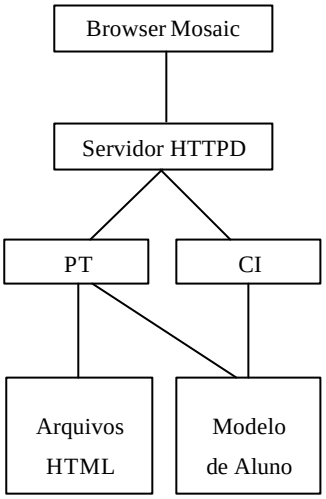

Figura 16 - Arquitetura do curso individualizado

- CI (*Concept Inventory*): responsável por definir a ordem de apresentação das questões oferecidas aos alunos na entrevista inicial. Comunica-se com o Modelo de Aluno, fazendo atualização de seu conteúdo após analisar as respostas do usuário;

- PT: processa a personalização das páginas; acessa as páginas do curso consultando o modelo do usuário e retornando as páginas personalizadas;

- Modelo de aluno: os componentes do modelo são organizados em contextos, que são um conjunto de relacionamentos estruturais que definem uma visão. Por exemplo, a entrevista inicial é baseada em uma visão definida para ser uma entrevista;

- O modelo de aluno é um conjunto de funções. Em outra versão da ferramenta, pretende-se criar um servidor com um simples protocolo de acesso;

- Implementado com CGI (*Common Gateway Interface*), apresentando páginas em HTML na interface com o usuário;

- Exige identificação (usuário e senha) para personalizar as páginas.

2.2.2 ELM-ART

O sistema ELM-ART (*Episodic Learner Model – Adaptative Remote Tutor*), de [BRU96], é um STI para o ensino da linguagem de programação LISP, através da *Web*. O ambiente foi desenvolvido no sistema ELM-PE (*Episodic Learner Model – Programming Environment*), um ambiente inteligente de aprendizagem que suporta programação baseada em exemplos, análise das soluções do problema, teste das soluções e depuração das mesmas.

Como se trata de um curso para ser oferecido à distância, disponibiliza o material do curso (apresentação de conceitos, exemplos, etc), no formato hipermídia, e fornece suporte à solução de problemas *on-line*. O ambiente difere-se de demais hiper-livros disponíveis na *Web* por dois aspectos: por saber que material apresentar ao aluno e suportar sua aprendizagem e navegação pelo material; e por todos os exemplos e problemas apresentados não serem apenas textos, mas experiências de vida. Algumas características do sistema:

- O material consiste em dois componentes básicos: o livro-texto e o manual de referência;

- O livro-texto está hierarquicamente estruturado em unidades, em diferentes níveis. Cada uma destas unidades é apresentada ao aluno como uma página HTML, contendo seu respectivo conteúdo;

- Os problemas são apresentados em páginas interativas separadas, as quais são formulários que podem ser preenchidos;

- O manual de referência fornece acesso ao material do curso. Cada página do manual possui breves explicações sobre um conceito e um *link* relacionado àquela unidade que trata o conceito;

- O manual também pode ser visto como um glossário e/ou um índice de um livro-texto tradicional;

- São usados dois tipos de *link*: hierárquico, o qual relaciona uma unidade a suas subunidades, e baseado no conteúdo.

O sistema conhece a estrutura pedagógica do domínio da linguagem LISP e a relação entre os diversos conceitos. Logo, o sistema sabe a estrutura de qualquer exemplo de solução de problemas.

Um aspecto negativo desta estrutura é que o aluno pode perder-se no conteúdo apresentado. Para auxiliar o aluno através da navegação pelo material, o sistema usa duas técnicas de hipermídia: anotação adaptativa e classificação de *links*.

- Anotação adaptativa significa que o sistema usa notações visuais, tais como ícones, fontes e/ou cores, para representar o tipo de cada *link*;

- Classificação de *links* é usada para representar *links* similares entre casos, já que o sistema pode verificar a similaridade entre dois ou mais casos. Sempre o primeiro *link* é o mais relevante em relação a um determinado caso.

Os conceitos que são pré-requisitos não são diretamente apresentados ao aluno, mas podem ser disponibilizados de duas formas. A primeira, quando um aluno acessa uma página que ainda não foi consultada por ele, então o sistema notifica-o que há pré-requisitos

que ainda não foram aprendidos e apresenta *links* para o livro-texto e/ou para o manual de referência, onde os conceitos relacionados se localizam. A segunda é quando o aluno tem dificuldades em compreender algum exemplo e este solicita ajuda, através de um botão especial para tal, e como resposta a esta solicitação de ajuda o sistema apresenta *links* para todas as páginas onde o pré-requisito aparece relacionado.

A implementação do sistema ELM-ART foi baseado no servidor hipermídia em LISP nomeado CL-HTTP, o qual é um servidor HTTP (*HyperText Transfer Protocol*) implementado em LISP que oferece uma interface CGI para acesso às URLs (*Uniform Resource Location*). Para que o servidor responda a determina URL, é necessário que esta esteja associada a uma determinada função de resposta também implementada em LISP.

2.2.3 ADAPT

A ferramenta ADAPT (*ADA Packages Tool*), de [VIK96], foi projetada para ajudar estudantes a adquirir conhecimento na linguagem de programação Ada, especialmente na utilização do conceito de pacotes, oferecido pela linguagem.

Os estudantes devem possuir algum conhecimento em linguagens procedurais, tais como Pascal ou C, bem como devem estar aprendendo a linguagem com o mínimo de instruções possíveis, mas com acesso a materiais de apoio apropriados.

O foco da ferramenta está no planejamento e não na codificação de soluções de problemas. Um dos objetivos da Engenharia de *Software* é que programadores criem e reusem componentes de *software*. Em Ada, isto é feito através dos pacotes. Pacotes representam não apenas uma nova estrutura sintática, mas uma maneira diferente de manipular e acessar dados.

Os usuários de ADAPT aprendem sobre pacotes e planos necessários para usá-los através da resolução de exercícios. Em um exercício, o aluno recebe um problema e deve escrever um procedimento, uma função ou um pacote para solucioná-lo.

O ambiente de trabalho é dividido em duas janelas, uma à direita com a apresentação do problema a ser resolvido e outra à esquerda, onde será desenvolvida sua solução, conforme pode ser observado na figura 17.

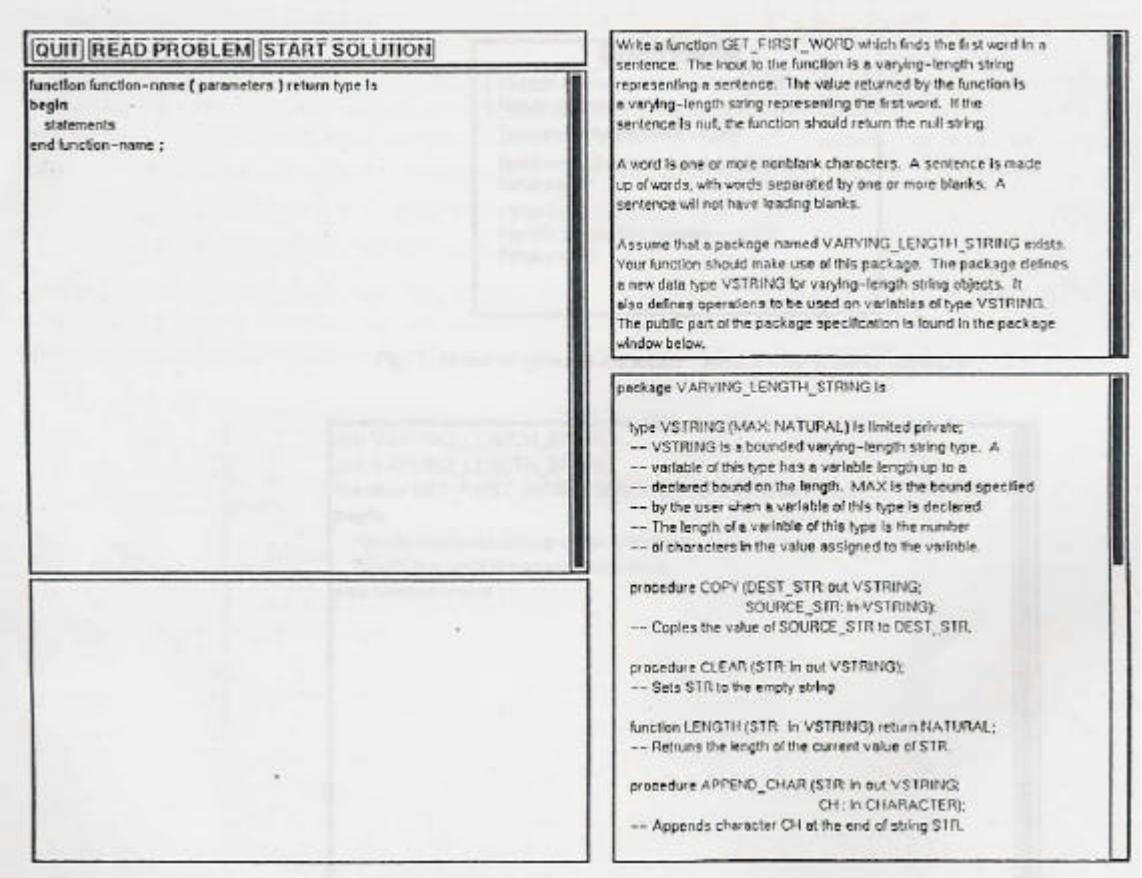

Figura 17 - Área de trabalho do ambiente ADAPT

O problema a ser resolvido é apresentado na janela superior à direita (vide figura 17). Se este problema exige o uso de um pacote para solucioná-lo, a especificação do mesmo aparece na janela inferior, também à direita.

Quando o aluno termina de ler o enunciado do problema e de verificar a especificação do pacote, deve pressionar o botão disponibilizado no topo da janela da esquerda para iniciar. Após, um *template* para o tipo de módulo necessário para o desenvolvimento da solução é apresentado na parte superior da janela de solução, localizada à esquerda na tela. Podem aparecer três tipos de itens na janela de solução: código Ada, *templates* de código e descrição de planos em linguagem natural.

Os *templates* de código devem ser elaborados com o objetivo de completarem a solução. Dependendo da complexidade, um *template* de código é elaborado por uma coleção de planos ou por simples linhas de código Ada para substituir o *template*.

Segundo [VIK96], o processo de planejar uma solução pode ser dividido em três níveis: estratégico, tático e de implementação.

- No nível mais alto, o de planejamento estratégico, o programador determina planos independentemente da linguagem de programação;

- No próximo nível, o tático, o programador constrói planos mais detalhados por partes do problema, mas estes ainda não são descritos em função de uma linguagem de programação em específico;

- No último nível, o de implementação, o programador determina uma forma de implementar cada um dos planos através de uma linguagem específica.

Pode haver várias maneiras de se planejar em alto nível (planejamento estratégico) uma solução de um problema. Para ajudar o estudante, quando este escolher substituir um plano em alto-nível, ADAPT disponibiliza um menu *pop up* com as várias soluções possíveis planejadas para que este apenas selecione uma delas.

Quando o aluno termina de elaborar a solução, esta é salva e o controle é passado para o ambiente Ada, o qual irá testar a solução apresentada.

2.2.4 C-Tutor

O ambiente C-Tutor, de [SON97], é um STI para ensino da linguagem C, composto por dois subsistemas: um Programa Analisador (*Program Analyzer*) e um Ambiente de Aprendizagem (*Learning Environment*).

Quanto ao Programa Analisador:

- É composto por um Sistema de Engenharia Reversa (*Engineering Reverse System*) e um Sistema Didático (*Didactic System*);

- ExBug (*Execution-guided deBugger*) é o Sistema Didático que analisa os programas dos alunos;

- GOES (*Goal Extraction System*) é o Sistema de Engenharia Reversa que gera automaticamente uma descrição de um problema através de um simples código de um programa, o qual é representado por um modelo de programa, gerado pelo professor;

- A descrição do problema possui objetivos que devem ser atingidos pelo aluno;

- A partir deste modelo de programa, automaticamente, GOES gera uma descrição para o problema proposto;

- A descrição, por sua vez, é gerada através da extração de planos do programa e destes são extraídos objetivos. Por fim, para completar a descrição do programa, se necessário, são acrescentadas informações sobre objetos usados no mesmo;

- Estes planos são extraídos de constantes e variáveis, e descrevem os estereótipos de seqüências de ações no programa e os objetivos representam conceitos de programação. Ambos estão relacionados hierarquicamente.

O Ambiente de Aprendizagem tem como características:

- A Rede de Conteúdos (*Curriculum Network*) é o Ambiente de Aprendizagem do ambiente C-Tutor;

- Este não é somente o módulo de tutoria do ambiente, mas também o módulo de conhecimento e modelo de aluno;

- Constrói o conhecimento dos objetivos e planos como gráficos genéticos e ensina conceitos de programação;

- Para selecionar o tópico a ser ensinado, refere-se tanto aos erros encontrados pelo ExBug como ao modelo do aluno;

- O diagnóstico de erros realizado é baseado nos artefatos gerados pelo aluno, procurando compreender onde seu código pode não estar bem estruturado;

- A Rede de conteúdos é composta de objetivos, planos e alguns exercícios;

- Objetivos e planos estão na mesma base de conhecimento que ExBug e GOES;

- A base de conhecimento do GTutor é representada como uma estrutura de *frame*, onde objetivos e planos são os principais componentes desta base.

A figura 18 apresenta uma visão geral da estrutura do ambiente C-Tutor.

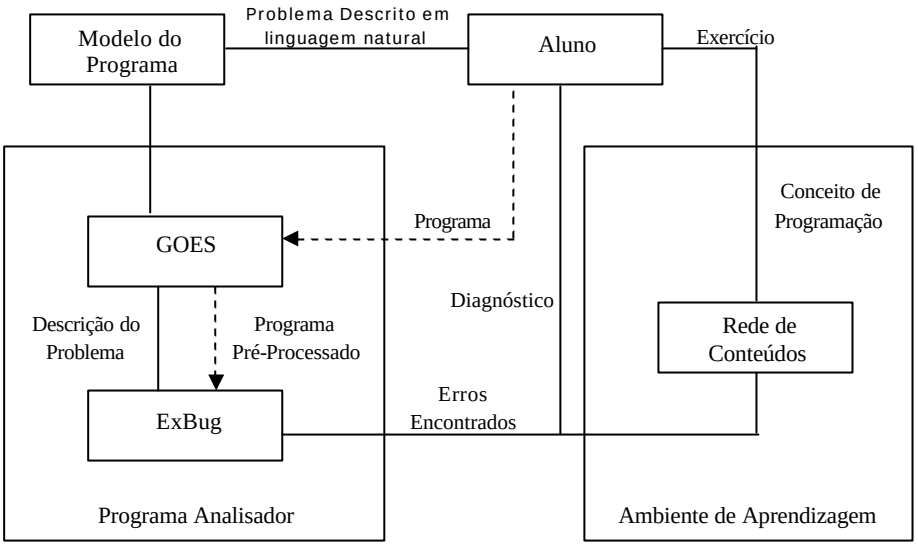

Figura 18 - Estrutura do ambiente C-Tutor

Há dois ciclos de interação no C-Tutor: Aluno-Sistema e Professor-Aluno-Sistema.

- Quanto ao ciclo Aluno-Sistema, o sistema ensina um conceito de programação e propõe a elaboração de um programa ao aluno. O programa gerado é passado ao sistema GOES com o objetivo deste remover erros sintáticos. O ExBug analisa este programa e retorna um diagnóstico. Baseado neste diagnóstico, o sistema decide se irá apresentar outro exercício ou um novo conceito ao aluno;

- Quanto ao ciclo Aluno-Professor-Sistema, o professor descreve um problema em linguagem natural ao aluno e fornece uma solução codificada deste problema ao sistema. O GOES analisa este programa e gera uma descrição do problema. A seguir, o aluno apresenta sua solução e o ciclo Aluno-Sistema se repete.

Cada módulo do C-Tutor foi implementado em LISP, em uma *workstation*, enquanto a interface Windows foi implementada na linguagem C, em um ambiente X/Motif.

2.2.5 PORTUGOL/PLUS

O sistema PORTUGOL/PLUS, desenvolvido por [ESM98], é uma ferramenta de apoio ao ensino de lógica de programação baseado no Portugol[6]. O ambiente pretende estimular os alunos a praticar e exercitar o desenvolvimento de algoritmos em Portugol.

O público-alvo do ambiente são alunos dos cursos de Ciência da Computação, Informática e demais cursos que possuam disciplinas envolvendo lógica de programação. O sistema divide-se em duas partes: o Editor de Algoritmos e o Compilador.

O Editor de Algoritmos permite a digitação e a manipulação de arquivos que contêm os algoritmos através de uma interface dirigida por menus. São eles:

- Arquivo: permite a manutenção (criar, salvar, abrir) de arquivos;
- Editar: permite executar ações sobre os arquivos em edição (abertos);
- Localizar: ajuda na busca de trechos específicos escritos no arquivo em uso;
- Compilar: torna possível a execução dos algoritmos criados. Após editado e salvo, este menu é usado para realizar a compilação do algoritmo. Depois de corrigidos seus erros, é possível ver seu comportamento, se o resultado obtido é o desejado. O menu Compilar, bem como um exemplo de algoritmo escrito em Portugol, podem ser vistos na figura 19;
- Janela: possibilita alterar a visualização dos arquivos abertos;

[6] Portugol é uma pseudo-linguagem algorítmica bastante utilizada na descrição de algoritmos, que faz uso de comandos em Português, facilitando o aprendizado da lógica de programação.

- Opções: permite alterar valores configuráveis pelo usuário dentro do ambiente, tal como cor.

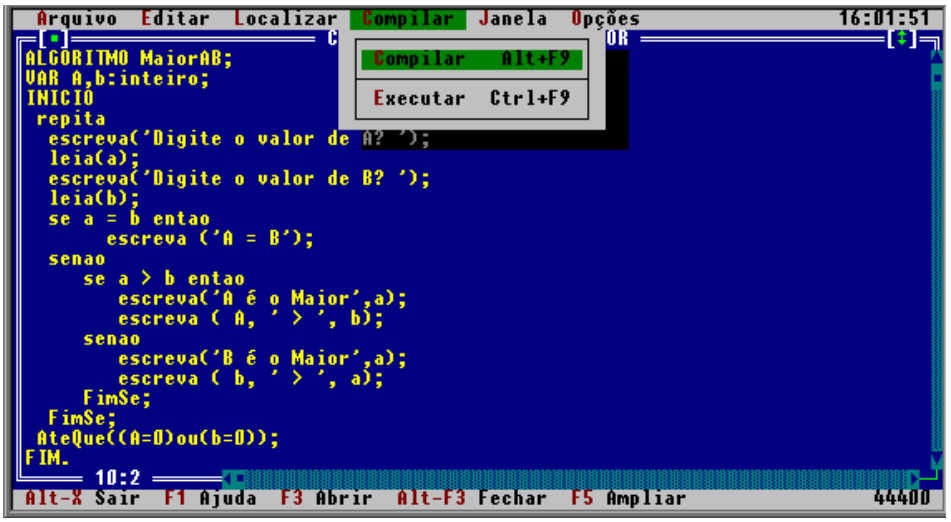

Figura 19 - Visão do ambiente do Portugol/PLUS

O compilador do ambiente PORTUGOL/PLUS, durante o processo de compilação, que pode ser verificado na figura 20, realiza a verificação da sintaxe das instruções usadas no algoritmo. Caso não sejam encontrados erros, um programa objeto é gerado na linguagem de programação Pascal. Este pode ser executado através de um compilador Pascal.

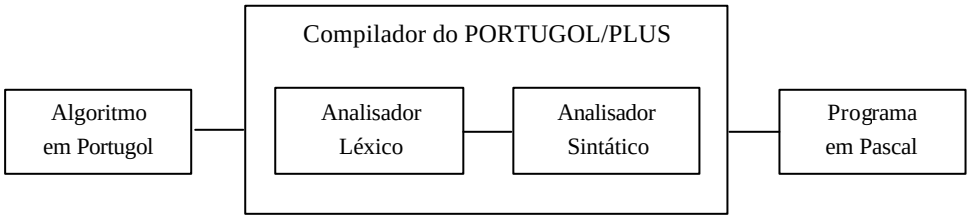

Figura 20 - O compilador do ambiente PORTUGOL/PLUS

O ambiente foi desenvolvido em linguagem Pascal, para o ambiente DOS. O autor pretende desenvolver uma versão para a plataforma Windows.

2.2.6 ASIMOV

O sistema ASIMOV, desenvolvido por [BIN99], é composto de um conjunto de ferramentas para apoio ao ensino de programação de linguagens imperativas. Seus principais objetivos são:

- Facilitar a aquisição de princípios de programação e de perícia em programação por meio da identificação e explicação de erros de lógica encontrados nos programas de alunos iniciantes;

- Possibilitar a alteração de diversas características de conteúdo;

- Possibilitar o uso deste conteúdo em variadas situações de aprendizagem.

O sistema possui uma interface contendo quatro áreas bem distintas: edição de programas, visualizador, entrada e saída de dados do programa e enunciado do exercício proposto. Através da interface é possível editar textos, compilá-los de acordo com uma gramática flexível, diagnosticar automaticamente a solução do aluno e interpretar o programa.

Para disponibilizar tais tarefas, foram utilizadas diversas técnicas computacionais, tais como a construção de compiladores e interpretadores, redes semânticas e casamento de padrões. Erros de lógica em programas de alunos são identificados através da comparação da solução do mesmo com uma solução pré-determinada pelo professor, denominada de solução de referência. A comparação entre as duas soluções é realizada através de um algoritmo de casamento de padrões.

2.2.7 LPT-TUTOR

O LPT-TUTOR, de [CID00], é um ambiente *Web* de aprendizagem que tem por objetivo geral facilitar o desenvolvimento de habilidades de uso da tecnologia de informação e comunicação pelos alunos da disciplina de Linguagens e Técnicas de Programação do Bacharelado em Informática da Universidade da Região de Joinville (UNIVILLE).

O ambiente disponibiliza conteúdos, atividades e exercícios a serem desenvolvidos durante as aulas; possibilita atividade de revisão e fixação dos conteúdos em horários extra-classe e extra-campus; e oferece recursos de integração e trabalho em grupo através de uma lista de discussão.

A estrutura do ambiente foi desenvolvida baseada em uma *homepage*, dividida em três quadros que representam uma sala de aula:

- O quadro *tutor* está localizado no canto superior direito do ambiente e permite, através do acionamento de ícones, obter informações a respeito das atividades a serem desenvolvidas na disciplina;

- O *quadro-negro* se localiza à esquerda do ambiente e permite a apresentação de slides relativos aos conteúdos das aulas;

- A *escrivaninha* está localizada no canto inferior direito do ambiente e permite o acesso ao detalhamento dos conteúdos.

2.2.8 Construção de abstrações em lógica de programação

Mattos [MAT00] propôs uma ferramenta didática motivado pela dificuldade do ensino da lógica de programação através da abordagem abstrata, onde o aluno deve pensar na solução de um problema através de conceitos.

A ferramenta tem como objetivos introduzir o conceito de análise de requisitos nas primeiras fases do ensino de computação, de tal forma que o aluno acostume-se a realizar uma análise mais detalhada dos problemas que são propostos, e auxiliar o aluno a construir abstrações sobre a solução destes problemas.

Possui como público-alvo alunos de disciplinas introdutórias de programação de cursos de Computação, sendo que os alunos de tais disciplinas da Universidade Regional de Blumenau, entidade onde se realizou a pesquisa, foram os motivadores deste estudo.

A primeira versão da ferramenta constituía um sistema especialista, desenvolvido em CLIPS[7], que representava o conhecimento do especialista em uma árvore de decisões. O enunciado do problema é apresentado e o sistema ia lançando perguntas ao aluno, buscando induzi-lo a pensar na solução do problema, através da análise do seu contexto. Nesta versão, a ferramenta conduzia a um melhor entendimento do contexto do problema a ser solucionado, mas não contribuía significantemente para o processo de generalização da solução desenvolvida. Alunos com dificuldades em abstrair conceitos continuavam excluídos do processo de aprendizagem. Tal situação levou ao desenvolvimento, em linguagem Java, de uma nova versão da ferramenta.

A segunda versão utiliza Raciocínio Baseado em Casos (RBC) e análise de linguagem natural para obter propostas de soluções abstratas para os problemas propostos. Esta versão explora os casos apresentando-os aos alunos e solicita que estes tentem produzir soluções adotadas anteriormente em problemas semelhantes, sintetizando-as e aplicando-as na nova solução.

Uma terceira versão está em desenvolvimento, unificando as duas anteriores e aprimorando suas características gerais.

---

[7] *CLIPS*: abreviação para *C Language Integrated Production System*. Projetado em 1985 pela NASA/Johnson *Space Center* [BAR01]. É um sistema especialista que fornece um completo ambiente para a construção de regras e/ou sistemas especialistas baseados em objetos [RIL02].

2.2.9 HabiPro

HabiPro (Hábitos de Programação), de [VIZ00], é um ambiente colaborativo criado para desenvolver "bons hábitos"[8] de programação. Não possui o objetivo de ensinar programação propriamente dita, mas sim de desenvolver em estudantes iniciantes habilidades tais como observação e reflexão, as quais são necessárias para que se tornem bons programadores.

A interface da aplicação é dividida em duas janelas: uma de *chat*, que permite a comunicação entre os estudantes, e outra denominada *Área de Trabalho*, destinada a compartilhar as soluções de problemas resolvidos de forma colaborativa pelos alunos.

A janela *Área de Trabalho* pode apresentar quatro tipos de exercícios: buscar o erro, colocar um programa na ordem correta, predizer o resultado a ser apresentado pelo programa e completar um determinado programa.

Quanto aos aspectos de interação com o aluno e tratamento de erro, o sistema apresenta quatro tipos de ajuda, as quais têm o objetivo de ajudá-los a encontrar a solução correta e obter informações sobre os mesmos:

- O primeiro tipo apresenta dicas sobre como o problema pode ser solucionado, levando o grupo a refletir sobre a possível solução;
- O segundo, apresenta a solução e uma explicação porque a mesma foi solucionada com determinada técnica;
- O terceiro, apresenta um exemplo similar, com sua respectiva solução. Neste caso o grupo precisa observar a técnica que foi usada e relacioná-la com o problema proposto;
- O último tipo simplesmente apresenta a solução do problema.

[8] A expressão é do autor do ambiente.

HabiPro possui uma arquitetura cliente-servidor, conforme está representado na figura 21.

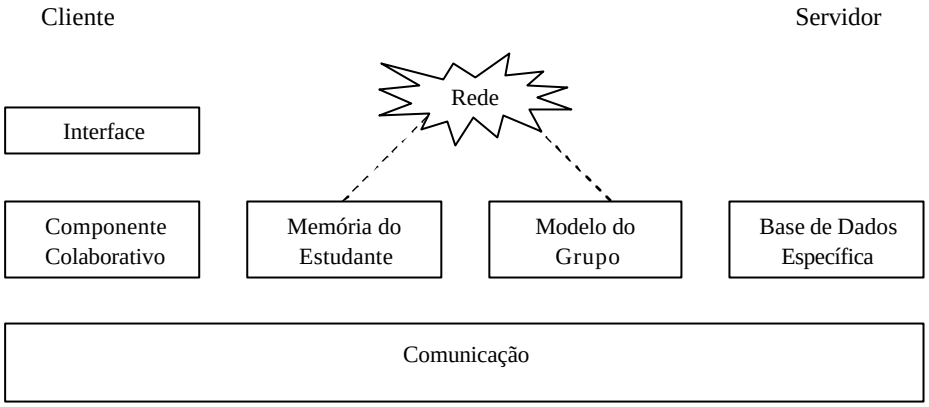

Figura 21 - Arquitetura do sistema HabiPro

O lado cliente é formado por: uma interface, que é a janela de *chat*; um componente colaborativo, que proporciona a colaboração entre os estudantes; uma memória do estudante, que armazena as ações realizadas pelos alunos; e um componente de comunicação, que permite envio e recebimento de mensagens entre o lado cliente e o servidor.

O lado servidor é composto por: uma parte de comunicação, que permite aos estudantes estarem conectados à aplicação, bem como o envio de recebimento de mensagens; uma base de dados específica, para armazenar exercícios, soluções, exemplos, dicas, etc; e um modelo do grupo, que armazena quais são suas habilidades, tipos de exercício preferido e tipos de erros cometidos.

2.2.10 Ambiente colaborativo para o aprendizado de programação

O projeto Suporte Tecnológico para o Aprendizado Colaborativo de Desenvolvimento de *Software*, proposto por [TOB01], tem por objetivo a definição, o projeto (*design*) e a implementação de um ambiente, mediado por computador, de aprendizado de programação colaborativo. O ambiente possui as seguintes características:

- Voltado para a área de desenvolvimento de *software*, seguindo o paradigma procedimental;

- Permite o envolvimento de estudantes, professores e sistemas inteligentes, oferecendo meios para geração e discussão de idéias, resolução de problemas, acesso e localização de informação *on-line* útil, e motivação à participação dos estudantes;

- O uso do ambiente pode ser caracterizado por um cenário com três etapas, não necessariamente seqüenciais, são elas: a aquisição de informação e conhecimento básico sobre um determinado tópico pelo estudante; a consolidação desse conhecimento pelo estudante, através da solução cooperativa de problemas; e a realimentação e aprendizado pelo sistema, através da interação com o estudante;

Para satisfazer os requisitos especificados, o ambiente deverá conter os módulos funcionais apresentados na figura 22.

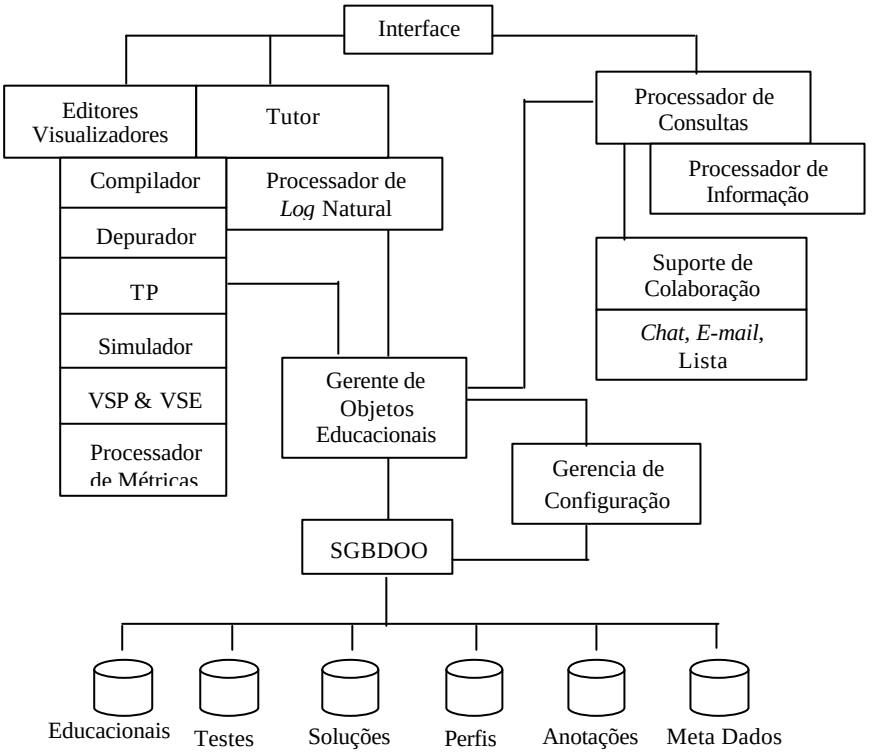

Figura 22 - Módulos funcionais do ambiente colaborativo

- O acesso à informação na *Web*, segundo [TOB01], oferece o *Processador de Consultas* e o *Processador de Informação*, o qual extrai, trata semanticamente, valida, transforma, classifica, filtra e minera os dados;

- Para manipulação de programas, o estudante lança mão de editores e visualizadores de objetos, através dos quais pode ativar compiladores, depuradores ou simuladores;

- Para comunicação, são disponibilizadas ferramentas para listas de discussões, *e-mail* e *chat*;

- O *Verificador de Semelhança de Programas* (VSP) tem por função comparar programas indicando o grau de semelhança entre os mesmos. De forma análoga, o *Verificador de Semelhança de Explicações* (VSE) compara textos explicativos dos programas;

- O módulo *Testador de Programas* (TP) submete-o, após compilá-lo, a um conjunto de testes do tipo caixa preta, visando verificar a funcionalidade básica do sistema;

- O módulo *Tutor*, baseado em uma interface em língua natural, tem o intuito de propiciar um relacionamento mais "humano" entre o sistema e os estudantes [TOB01].

No momento, a definição geral da arquitetura do sistema está sendo finalizada, sendo que protótipos de alguns dos módulos estão disponíveis, enquanto outros estão em fase de implementação.

A análise e estudo destes ambientes permitiram que pudesse ser observada as tendências e as características julgadas importantes num ambiente para suporte ao ensino-aprendizagem de programação para iniciantes.

No próximo capítulo estas conclusões e observações são apresentadas.

3 Considerações finais

Após a detalhada leitura dos ambientes analisados, percebe-se a preocupação em disponibilizar os conteúdos relacionado ao curso/disciplina que o ambiente atende, levando-se em conta uma proposta metodológica. Esta constatação vem ao encontro das crenças do projeto PROOGRAMA [GIR02], onde acredita-se que a modelagem de uma ambiente educacional é altamente dependente da proposta metodológica que embasa o trabalho do professor. Logo, a quantidade de agentes, recursos, etc, que um ambiente vai possuir depende da concepção pedagógica da instituição onde ele será utilizado.

Os ambientes estudados possuem os seguintes aspectos em comum:

- Estruturação do conteúdo em exemplos e exercícios;
- Uso de *feedback* (retroalimentação) para os erros dos alunos;
- Possibilitar múltiplas formas de representação do conhecimento, quanto à apresentação dos exercícios como do *feedback*. Entenda estas múltiplas representações o uso de gráficos, desenhos, textos, etc;
- Preocupação de registrar o trabalho do aluno para posterior análise.

Quanto à preocupação de relacionar o material disponibilizado com o conteúdo que se pretende ensinar para o aluno, ela está diretamente relacionada as estratégias de ensino utilizadas. Ou seja, novamente, encontra-se a necessidade de ter-se uma proposta

metodológica clara que permita coordenar e integrar todas as funcionalidades disponibilizdas nos ambientes.

Como próxima etapa, pretende-se fazer um estudo das estratégias de ensino encontradas nos ambientes e sua aplicação no ensino de programação, no contexto do projeto PROOGRAMA. Isto é, pretende-se aprofundar o estudo das estratégias identificadas, buscar-se a parte teórica e relacioná-las ao trabalho que vem sendo desenvolvido nas disciplinas de LAPRO I e LAPRO A, anteriormente citadas.

Uma vez realizada esta nova etapa, passar-se-á a uma alternativa de viabilização das mesmas, em relação a implementação. Espera-se com estas próximas etapas identificar que ajustes devem ser feitos na sociedade de agentes e funcionalidads da arquitetura geral do ambiente do PROOGRAMA ora em desenvolvimento.

Quanto aos aspectos de arquitetura dos sistemas e o uso de agentes, os ambientes possuem um número variado de agentes, com diferentes papéis. No ententanto, observa-se que as constatações apresentadas no trabalho de [BOL01] se comprovam:

- Existem agentes que realizam "tarefas" para o professor e para o aluno. Como exemplo, cita-se agentes organizadores de material, envio de mensagens selecionadas, controladores de atividades realizadas, etc;

- Existem agentes que "auxiliam o processo de aprendizagem" e estão ligados aos aspectos metodológicos/pedagógicos, como por exemplo, agente assistente, etc.

A complexidade de se modelar e implementar tais agentes é distinta. Os agentes associados às questões pedagógicas envolvem processos mais sofisticados de raciocínio e deliberação (construção/seleção de planos de ação). Observa-se que, em muitos dos protótipos, apenas os agentes executores de tarefas estão operacionais e os pedagógicos parcialmente implementados quando da conclusão dos trabalhos de mestrado e/ou

doutorado. Estes aspectos são importantes de serem considerados quando da definição futura do trabalho a ser desenvolvido na dissertação.

Acredita-se que o trabalho será facilitado no sentido de que a arquitetura geral do sistema é definida no escopo do projeto PROOGRAMA. Esta dissertação trabalhará para auxiliar a resolver problemas relativos a uma determinada parte da arquitetura. A definição da arquitetura está prevista no cronograma do projeto até agosto do corrente ano. Logo, auxilia na definição dos requisitos do ambiente, uma vez que permite ao grupo do projeto a comparação de algumas hipóteses a cerca das possíveis funcionalidades de um ambiente desta natureza.

A questão metodológica já foi definida no escopo do projeto e vem sendo testada nas disciplinas de LAPRO I e A, ministradas pela orientadora desta autora.

Existe uma parceria entre os pesquisadores dos projetos AmCorA e PROOGRAMA. Esta parceria visa a troca de experiências e de resultados. Assim, espera-se que exista alguma semelhança intencional entre estes ambientes.

Continuando a análise dos ambientes estudados, observou-se que muitos deles têm como característica prover uma área individualizada para cada tipo de usuário a fim de que as atividades disponibilizadas sejam somente àquelas permitidas para um determinado aluno. Muitas destas atividades tem por objetivo serem desenvolvidas em período extra-classe, a fim de que o aluno reforce e pratique os conceitos aprendidos em aula. Enquanto que outras buscam atender a uma das dificuldades constatadas: a do aluno poder participar presencialmente das atividades de aula (muito comum em cursos noturnos). Um exemplo disto são as atividades oferecidas através da *Web*, onde o aluno pode formular sua solução e discuti-la posteriormente com seu grupo de colegas.

Outra característica identificada, diretamente relacionada ao ensino de programação, foi a disponibilização de atividades que objetivam a apresentação de uma solução para um problema proposto. Para tal, os alunos devem analisar o problema proposto, discutirem

possíveis soluções e codificarem a solução escolhida, caso seja considerado interessante. Este estudo pode ser realizado de forma cooperativa ou individualmente, conforme o quê o ambiente se propõe a atingir.

Alguns ambientes fornecem *feedback* direto ao aluno permitindo-o avaliar sua evolução no processo de aprendizagem. Isto é feito através da análise das soluções apresentadas; sendo que, em alguns dos ambientes apenas é levado em consideração os erros cometidos pelos alunos no planejamento da solução e, em outros sistemas, a coerência da codificação com a sintaxe da linguagem.

Tem-se também ambientes com o objetivo específico de ensinar a importância da análise de requisitos de um sistema antes de desenvolver-se uma solução, bem como outros que enfatizam a elaboração de planos para solução de problemas; buscando ambos proporcionar as habilidades de observação e reflexão por parte do aluno.

Quanto ao material utilizado como referência ao longo da revisão bibliográfica realizada, alerta-se para a heterogeneidade do mesmo, tanto qualitativa como quantitativa, o que causou uma certa dificuldade na organização do texto de uma forma geral. Constatou-se a falta de uma apresentação mais abrangente do ambiente, onde fosse possível obter informações tanto em relação aos aspectos pedagógicos quanto computacionais, mais especificamente detalhes sobre arquitetura, algoritmos e/ou tecnologias utilizadas na implementação dos mesmos. Muitos dos detalhes de alguns ambientes foram adquiridos diretamente com os autores em função da orientadora da autora conhecer os próprios. Também, notou-se a falta de informação quanto ao estado atual da pesquisa de alguns ambientes, bem como a falta de acesso aos mesmos, o que dificulta estudos e análises mais aprofundada destes, especialmente em relação a aspectos computacionais.

Além das contribuições pessoais e para o grupo de pesquisa, espera-se que este RT contribua para a pesquisa de IE auxiliando na identificação de aspectos a serem considerados no desenvolvimento de um ambiente para ensino de programação de iniciantes.

4 Referências bibliográficas

[AZE99] AZEVEDO, Breno F.; TAVARES, Orivaldo L.; CURY, Davidson. MuTanIS: Um Sistema Tutor Inteligente Multiagente para o Ensino-Aprendizagem de Conceitos. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 10, 1999, Curitiba, PR. **Anais...** Curitiba: UFPR, 1999.

[AZE01] AZEVEDO, Breno F.; TAVARES, Orivaldo L. Um Ambiente Inteligente para Aprendizagem Colaborativa. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[BAR01] BARRETO, Jorge M. **Inteligência Artificial no limiar do século XXI.** 3.ed. Florianópolis: O Autor, 2001.

[BIN99] BINDER, Fábio V.; DIRENE, Alexandre I. Conceitos e Ferramentas para Apoiar o ensino de Lógica de Programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 10, 1999, Curitiba, PR. **Anais...** Curitiba: UFPR, 1999.

[BIC98] BICA, Francine. **Eletrotutor III - Uma Abordagem Multiagente para o Ensino a Distância.** Porto Alegre: CPGCC, UFRGS, 1998. (Dissertação de Mestrado em Ciência da Computação)

[BOL01] BOLZAN, Willian. **Estudo comparativo sobre Sistemas Tutores Inteligentes para Web utilizando Agentes.** Porto Alegre: PPGCC, FACIN/PUCRS, 2001. (Trabalho Individual I, Mestrado em Ciência da Computação)

[BRI00] BRITO, Silvana R.; TAVARES, Orivaldo L.; MENEZES, Crediné S. A Society of Intelligent Agents for an Environment of Cooperative Online Learning. *Online*. Disponível na Internet http://www.inf.ufes.br/~tavares/IC_AI_2000_ SROSSY_ENG.html. (Capturado em mar. 2002)

[BRU96] BRUSILOVSKY, Peter; SCHWARZ, Elmar; WEBER, Gerhard. ELM-ART: An Intelligent Tutoring System on World Wide Web. In: Intelligent Tutoring Systems Conference, 3, 1996, Montréal, Canada. **Proceedings...** Montréal: 1996.

[CHA01] CHAVES, Thais H. *et al.* Um Ambiente para Apoio à Aprendizagem de Programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[CID00] CIDRAL, Alexandre. LPT-TUTOR: Um Ambiente Web de Aprendizagem como Suporte as Atividades Presenciais no Ensino de Técnicas de Programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11, 2000, Maceió, AL. **Anais...** Maceió: UFAL, 2000.

[DAM97] D'AMICO, C.; VICCARI, R.; ALVARES, L. A Framework for Teaching and Learning Environments. In: SIMPÓSIO DE INFORMÁTICA NA EDUCAÇÃO, 8, 1997, São Paulo, SP. **Anais...** São Paulo: ITA, 1997.

[DAM98] D'AMICO, C.*et al.* Adapting Teaching Strategies in a Learning Environment on WWW. In: The WebNet World Conference of the WWW, Internet & Intranet, Florida, USA. 1998. **Proceedings...** Florida: 1998.

[DAM99] D'AMICO, C. **Aprendizagem estática e dinâmica em ambientes multiagentes de ensino-aprendizagem.** Porto Alegre: CPGCC, UFRGS, 1999. (Tese de Doutorado)

[ESM98] ESMIN, Ahmed A. PORTUGOl/PLUS: Uma Ferramenta de Apoio ao Ensino de Lógica de Programação Baseado no Portugol. *Online*. Disponível na Internet http://www.c5.cl/ieinvestiga/actas/ribie98/118.html. (Capturado em abr. 2002)

[GAV00] GAVA, Tânia B.; MENEZES, Crediné S. *Moonline*: Um Sistema Multiagentes Baseado na Web para Apoio a Aprendizagem. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11, 2000, Maceió, AL. **Anais...** Maceió: UFAL, 2000.

[GAV01] GAVA, Tânia B.; MENEZES, Crediné S. *Moonline*: Um Ambiente de Aprendizagem Cooperativa Baseado na Web para Apoio às Atividades Extra-classe. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[GEY01] GEYER, Cláudio *et al.* SEMEAI: SistEma Multiagente de Ensino e Aprendizagem na Internet. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[GIR97] GIRAFFA, Lucia M. **Seleção e adoção de Estratégias de Ensino em Sistemas Tutores Inteligentes.** Porto Alegre: CPGCC/UFRGS, 1997. (Estudo de Qualificação)

[GIR01] GIRAFFA, Lúcia M. M., VICCARI, Rosa M.. **Fundamentos dos Sistemas Tutores Inteligentes**. Porto Alegre. Capítulo de livro (a ser publicado).

[GIR02] GIRAFFA, Lucia M.; PINTO, Sérgio C.; OLIVEIRA, João B. **PROOGRAMA: Ambiente Cooperativo de Suporte a Aprendizagem de Algoritimo e Programação.** Porto Alegre: PPGCC, FACIN/PUCRS, 2002. (Relatório Técnico Nº 24)

[GOU01] GOULART, Rodrigo R. **Utilizando a Tecnologia de Agentes na construção de sistemas Tutores Inteligentes em Ambiente Interativo.** Porto Alegre: PPGCC/FACIN, PUCRS, 2001. (Dissertação de Mestrado em Ciência da Computação)

[KAY94] KAY, J.; KUMMERFELD, R. An Individualised Course for the C Programming Language. *Online*. Disponível na Internet http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Educ/kummerfeld.html (Capturado em mar. 2002)

[MAT00] MATTOS, Mauro M. Construção de Abstrações em Lógica de Programação. In: CONGRESSO NACIONAL DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 20, 2000, Curitiba. **Anais...** Curitiba: PUCPR, 2000.

[MEN99] MENEZES, Crediné S.; CURY, Davidson. AmCorA: Um Ambiente Cooperativo para a Aprendizagem Construtivista Utilizando a Internet. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 10, 1999, Curitiba, PR. **Anais...** Curitiba: UFPR, 1999.

[MUS01] MUSA, Daniela L. *et al.* Agente para auxílio à avaliação de aprendizagem em ambientes de ensino na Web. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[NOB02] NOBRE, Isaura A. **Suporte à Cooperação em um Ambiente de aprendizagem para Programação (SAmbA).** Vitória: Departamento de Informática, UFES, 2002. (Dissertação de Mestrado em Informática)

[PER01] PEREIRA, A.; D'AMICO, C.; GEYER C. Gerenciamento do Conhecimento do ambiente AME-A. *Online*. Disponível na Internet em: http://www.inf.ufrgs.br/~adriana/vcied.doc (Capturado em abr. 2002)

[RIL02] RILEY, Gary. What is CLIPS? Online. Disponível na Internet http://www.ghg.net/clips/WhatIsCLIPS.html (Capturado em mai. 2002)

[ROS00] ROSATELLI, M., SELF, J.; THIRY, M. LeCS: a collaborative case study system. In Inteligent Tutoring Systems Conference, 5, 2000, Montréal, Canada. **Proceedings...** Montréal: 2000.

[SAN01] SANTOS, Cássia T. *et al.* DÓRIS – Um agente de acompanhamento pedagógico em sistemas tutores inteligentes. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[SEL99] SELF, J. The defining characteristics of intelligent tutoring systems research: ITSs care, precisely. International Journal of Artificial of Artificial Intelligence in Education, Leeds, England, 1999.

[SIL92] SILVEIRA, R. **Inteligência Artificial em Educação:** um modelo de sistema tutorial inteligente para microcomputadores. Porto Alegre: PUCRS, 1992. (Dissertação de Mestrado em Educação)

[SIL00] SILVA, Aleksandra do Socorro. **TUTA: Um Tutor Baseado em Agentes no Contexto do Ensino a Distância.** Campina Grande: Centro de Ciências e Tecnologia, UFPA, 2000. (Dissertação de Mestrado em Ciência da Computação)

[SIL00a] SILVA, Aleksandra S.; DOMINGUEZ, Arturo H. TUTA: Um Tutor Baseado em Agentes. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 11, 2000, Maceió, AL. **Anais...** Maceió: UFAL, 2000.

[SIL01] SILVA, Flávio S.; MENESES, Eudenia X. Integração de Agentes de Informação. In: Jornada de Atualização em Inteligência Artificial, 1, 2001, Ceará. **Anais...** Ceará: UNIFOR, 2001.

[SON97] SONG, J. *et al.* An Intelligent Tutoring System for Introductory C Language Course. In: Computers & Education, v. 28, N. 2, 1997.

[TED97] TEDESCO, P. **SEI – Sistema de Ensino Inteligente.** Recife: DI, UFPE, 1997. (Dissertação de Mestrado em Ciência da Computação)

[TOB01] TOBAR, Carlos M. *et al.* Uma Arquitetura de Ambiente Colaborativo para o Aprendizado de Programação. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 12, 2001, Vitória, ES. **Anais...** Vitória: UFES, 2001.

[VIK96] VIKKI, Fix; WIEDENBECK, Susan. An Intelligent Tool to Aid Students in Learning Second and Subsequent Programming Languages. In: Computers & Education, v. 27, N. 2, 1996.

[VIZ00] VIZCAÍNO, Aurora *et al.* In: Intelligent Tutoring Systems Conference, 5, Montréal, Canada, June 2000. **Proceedings...** Montréal: 2000.