UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Patrícia Gerent Petry

UM SISTEMA PARA O ENSINO E APRENDIZAGEM DE ALGORITMOS UTILIZANDO UM COMPANHEIRO DE APRENDIZAGEM COLABORATIVO

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Drª. Marta Costa Rosatelli
Orientadora

Florianópolis, 2005.

UM SISTEMA PARA O ENSINO E APRENDIZAGEM DE ALGORITMOS UTILIZANDO UM COMPANHEIRO DE APRENDIZAGEM COLABORATIVO

Patrícia Gerent Petry

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Conhecimento e aprovada em sua forma final pelo Programa de Pós–Graduação em Ciência da Computação.

_____

Dr. Raul Sidnei Wazlawick
Coordenador do Curso

Banca Examinadora

_____

Drª Marta Costa Rosatelli (orientadora)
Presidente da Banca

_____

Dr. Rafael Faraco
Membro da Banca

_____

Dr. Leandro J. Komosinski
Membro da Banca

_____

Dr. Raul Sidnei Wazlawick
Membro da Banca

"Ninguém ensina nada a ninguém,
no máximo ajuda-se o outro a aprender"
Galileu

Ao meu filho, João Victor.

AGRADECIMENTOS

Agradeço primeiramente a Deus que me deu forças durante toda a jornada deste trabalho.

À professora Marta Costa Rosatelli, minha orientadora, que mesmo à distância conseguiu fazer-se presente, dando-me incentivos nos momentos mais difíceis.

Agradeço aos meus pais, José Mauro e Ivete, pelo carinho e apoio recebidos durante o desenvolvimento deste trabalho, bem como às minhas irmãs que sempre estiveram ao meu lado.

Às amigas Fátima, Gisele e Renate, meus sinceros agradecimentos por me escutarem e darem muito apoio durante o desenvolvimento deste trabalho com suas palavras de coragem e amizade.

Ao Wilson pelo seu apoio no desenvolvimento do protótipo deste trabalho.

A todos aqueles que contribuíram direta ou indiretamente com o desenvolvimento desta Dissertação.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CA	Companheiro de Aprendizagem
CAI	*Computer Assisted Instruction*
CSCL	*Computer Supported Collaborative Learning*
DOS	Sistema Operacional em Disco
EAD	Ensino A Distância
IA	Inteligência Artificial
IA-ED	Informática Aplicada à EDucação
ICAI	*Intelligent Computer Assisted Instruction*
KQML	*Knowledge Query and Manipulation Language*
LC	*Learning Companion*
LECS	*LEarning from Case Studies*
MBR	Modelagem Baseada em Restrições
SCAs	Sistemas Companheiro de Aprendizagem
SLCs	Sistemas *Learning Companion*
SQL	*Structured Query Language*
STI	Sistemas Tutores Inteligentes
UML	Linguagem de Modelagem Unificada
ZPD	Zona Proximal de Desenvolvimento
XML	*Extensible Markup Language*
WWW	*World Wide Web*

RESUMO

Esta dissertação apresenta um modelo computacional de um sistema de suporte ao ensino e aprendizagem no domínio de algoritmos. O sistema inclui um companheiro de aprendizagem virtual, que utiliza a Modelagem Baseada em Restrições (*Constraint-Based Modelling*) como forma de representação do conhecimento e raciocínio. Os Sistemas Companheiro de Aprendizagem (Sistemas *Learning Companion*) são considerados uma evolução dos Sistemas Tutores Inteligentes e têm na sua arquitetura, além dos módulos tradicionais deste tipo de sistema, um companheiro virtual que interage com o estudante na resolução de problemas, auxiliando e colaborando no aprendizado de um determinado domínio. A abordagem da modelagem baseada em restrições tem como fundamento uma teoria de aprendizagem que traz uma concepção do processo de ensino e aprendizagem em que o aluno aprende através dos seus erros. O sistema modelado e prototipado – o AlgoLC – proporciona ao estudante um auxílio individualizado de um companheiro de aprendizagem que tem um papel colaborativo, interagindo com o aluno com o objetivo de estimulá-lo a verificar seus erros e corrigi-los, e não apontando a solução do problema.

ABSTRACT

This dissertation presents a computational model of a system that supports teaching and learning in the domain of algorithms. The system includes a virtual learning companion that uses Constraint-Based Modelling as knowledge representation and reasoning. Learning Companion Systems are considered an evolution of Intelligent Tutoring Systems that include in their architecture, besides the traditional modules of this kind of system, a virtual companion that interacts with the student in the problem solving process, supporting and collaborating in learning a particular domain. The Constraint-Based Modelling approach has as a basis a theory of learning from performance errors that brings a conception to the teaching and learning process that the students learn from their errors. The system that was modelled and prototyped – named AlgoLC – provides to the student the individualised support of a learning companion that has a collaborative role, interacting with the student with the objective of stimulating him or her to verify the errors and correct them, rather than pointing out the problem solution.

1. INTRODUÇÃO

A disciplina de algoritmos, que faz parte do currículo de alguns cursos de graduação da área tecnológica, é uma barreira para muitos alunos que se iniciam na área. Em um algoritmo, o aluno precisa desenvolver seu raciocínio lógico propondo soluções de problemas. Segundo (NOBRE, 2002), algumas dificuldades vivenciadas pelo professor desta disciplina ou no ensino da programação são:

- Reconhecer as habilidades inatas de seus alunos;
- Apresentar técnicas de soluções de problemas;
- Trabalhar a capacidade de abstração do aluno, tanto na busca de possíveis soluções quanto na escolha da estrutura de dados a ser utilizada;
- Promover a cooperação e colaboração entre os alunos.

No ensino tradicional o professor em sala de aula é a figura que acompanha cada etapa apresentada pelos alunos no desenvolvimento da solução dos problemas propostos. Porém, nem sempre é possível para o professor acompanhar os alunos quando estes desenvolvem seus exercícios individualmente.

Buscando adequar-se a essas dificuldades encontradas em sala de aula, este trabalho apresenta um modelo computacional de sistema de suporte ao ensino e aprendizagem de algoritmos através de um companheiro de aprendizagem que utiliza a Modelagem Baseada em Restrições (MBR) como forma de representação do conhecimento e raciocínio. Este sistema utiliza um companheiro de aprendizagem que auxilia os alunos no desenvolvimento das possíveis soluções dos problemas a partir da identificação dos seus erros. O uso deste ambiente possibilita ao professor acompanhar e propor situações desafiadoras ao raciocínio do aluno através de relatórios que são gerados após as tentativas de solução do problema pelo aluno.

Os Sistemas Companheiro de Aprendizagem (SCAs) são considerados uma evolução dos Sistemas Tutores Inteligentes (STIs), que prevêem além dos módulos tradicionais, a existência de pares virtuais com comportamento humano que apóiam os estudantes durante o processo de ensino-aprendizagem (CHAN & BASKIN, 1988). Um STI utiliza técnicas da inteligência artificial e teorias pedagógicas para monitorar a interação de cada estudante em um determinado domínio (por exemplo, a Matemática). Um STI decide o que ensinar e como ensinar com base em informações sobre o

aprendizado do estudante (por exemplo, seu estilo de aprendizagem) (AZEVEDO, 2003), utilizando os melhores meios de instrução para esse determinado estudante. Além disso, um STI avalia se o estudante está assimilando o conteúdo e apresenta o *feedback*[1] mais apropriado a ele.

Um SCA encoraja o estudante a aprender colaborativamente, pois coopera com um estudante na resolução de problemas (CHAN & BASKIN, 1988). Seguindo a teoria sócio–cultural de Vygotsky, que enfoca a interação social no desenvolvimento cognitivo do indivíduo e conceitos da Zona Proximal de Desenvolvimento (ZPD), a interação com outras pessoas torna o indivíduo capaz de realizar atividades que, sem a ajuda externa, seriam impossíveis de acontecer (REGO, 1995).

A abordagem MBR tem como fundamento uma teoria de aprendizagem que se baseia na identificação dos erros. Esta teoria é relevante para os estudantes que não detém o conhecimento de um determinado assunto e, portanto, não são capazes de detectar os próprios erros. Esta modelagem representa o conhecimento do estudante sobre um determinado domínio, como por exemplo, algoritmos, através de um conjunto de restrições, onde cada uma delas representa um conceito declarativo que deve ser aprendido e analisado pelo estudante para a solução correta de problemas no domínio em estudo (MARTIN, 2003; MITROVIC & OHLSSON, 1999).

1.1 Objetivo Geral

Este trabalho tem como objetivo geral comprovar que a utilização de um sistema companheiro de aprendizagem colaborativo para auxílio ao ensino e aprendizagem de algoritmos facilita a identificação de dúvidas e erros dos alunos na construção de algoritmos.

1.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Fazer uma revisão sobre os STIs, enfocando os SCAs que possuem conceitos relacionados a esta dissertação;

[1] *Feedback* é a retroalimentação fornecida ao aluno por ocasião do erro.

- Investigar e analisar alguns ambientes computacionais para apoio ao ensino e aprendizagem de algoritmos;
- Identificar uma técnica de representação do conhecimento e raciocínio - modelagem baseada em restrições – apropriada aos CAs;
- Definir os componentes básicos necessários para a especificação de um modelo de companheiro de aprendizagem para o ensino e aprendizagem de algoritmos;
- Demonstrar a viabilidade da aplicação na criação de um protótipo em um sistema que suporte a aprendizagem colaborativa apoiada por computador no domínio de algoritmos, demonstrando que o estudante que utiliza o SCA como auxílio no desenvolvimento de problemas algorítmicos, erra menos na solução destes problemas;
- Demonstrar que com a aplicação do AlgoLC o professor dispõe de uma ferramenta computacional com estatísticas de aprendizado da disciplina de algoritmos, em resoluções dos problemas realizados pelos alunos.

1.3 Justificativa

A disciplina de algoritmo tem um dos maiores índices de reprovação em todas as instituições de ensino no Brasil (índice de 40% a 50%), o que a torna ponto de reflexão por parte de professores preocupados com a melhoria da qualidade no processo de ensino e aprendizagem dos alunos (RODRIGUES, 2002).

Segundo KOSLOSKY (1999), uma das maiores dificuldades encontradas no processo ensino e aprendizagem de algoritmos é entender o raciocínio desenvolvido pelos alunos. Mesmo em uma situação ideal, onde existe a figura de um mediador que possa acompanhar individualmente cada um deles, é difícil acompanhar cada etapa da solução apresentada.

A didática utilizada pelo professor para apresentar o conteúdo introdutório da disciplina também é um fator fundamental no processo de ensino de algoritmos, pois para todo problema requer uma boa capacidade de abstração do aluno. O aluno, normalmente, tem dificuldade em entender os diversos e novos conceitos referentes à execução de tarefas pelo computador (RODRIGUES, 2002).

A presente dissertação se justifica por tentar amenizar os problemas descritos acima através da criação de um modelo computacional com suporte ao ensino e aprendizagem de algoritmo utilizando um CA que auxilia os alunos no desenvolvimento das possíveis soluções dos problemas a partir da identificação dos seus erros. A escolha da utilização de um SCA deu-se pela possibilidade de interação do estudante com um companheiro de aprendizagem, que segundo Vygotsky a aprendizagem é resultante de um processo de interação com outro(s) sujeito(s) eventualmente mais capaz(es).

Os diferentes ambientes computacionais descritos no decorrer desta dissertação serviram de base para o desenvolvimento e criação do modelo AlgoLC.

1.4 Metodologia de Trabalho

Este trabalho será desenvolvido através de uma revisão de literatura dos principais conceitos envolvendo o ensino e a aprendizagem de algoritmos. Serão apresentados e descritos alguns ambientes de apoio ao ensino de algoritmos.

Em seguida, será realizado um estudo sobre Sistemas Companheiro de Aprendizagem e uma revisão da literatura sobre os mecanismos de representação do conhecimento e raciocínio em alguns ambientes computacionais existentes. Esses ambientes servirão de base para o desenvolvimento de um modelo de um sistema de apoio ao ensino e aprendizagem de algoritmos. O modelo será testado através de um protótipo disponibilizado para alguns alunos que cursam a disciplina de algoritmos do curso de Ciência da Computação.

1.5 Estrutura da Dissertação

A estrutura da presente dissertação é composta de seis capítulos.

O capítulo 1 faz uma introdução ao trabalho; destacando os objetivos, a justificativa e a metodologia.

O capitulo 2 mostra alguns conceitos relevantes ao ensino e aprendizagem de algoritmos, abordando alguns ambientes computacionais de apoio ao mesmo.

O capítulo 3 versa sobre os SCAs, a partir de uma revisão sobre os STIs.

No capítulo 4 é descrita a modelagem baseada em restrições, seus conceitos, características e implementações já existentes.

No capítulo 5 é apresentado o sistema prototipado, sua arquitetura e testes realizados.

O capítulo final relata as conclusões e aponta algumas direções para trabalhos futuros.

2. ENSINO-APRENDIZAGEM DE ALGORITMOS

2.1 Introdução

O problema da compreensão e aplicação da lógica de programação é muito comum nas disciplinas de algoritmos nos cursos em nível de graduação em Ciência da Computação. Essa grande dificuldade apresentada pelos alunos é uma das maiores causas das reprovações e desistências no decorrer do curso, seja em instituições de ensino públicas ou privadas (MARTINS, 2003). Quando não solucionada a tempo essa lacuna traz prejuízos imensos aos discentes e docentes da instituição no cumprimento do currículo do curso, tornando morosa a formação acadêmica do aluno e, muitas vezes, incompatível com o nível de atividades práticas e intelectuais que lhe serão exigidas ao longo do curso.

Segundo David Tall (2002),

> *"O computador oferece o algoritmo (a receita), não o pensamento. O aprendiz precisa pensar em como usar o algoritmo. Os alunos acham que o importante é chegar ao resultado certo, mas não é isso. No mundo dos negócios, há 20 anos, se você calculava errado o lugar de um buraco num disco de metal, perdia-se um colar e, portanto, muito dinheiro. Ter o melhor resultado não era o mais importante – era a única coisa que importava. Mas hoje há problemas que requerem outras qualidades, como intuição. Se você entende os princípios, pode usar o computador para fazer o trabalho por você. Mas se não entende, não terá chance."*

Algoritmo é um conceito que surge da matemática. Por isso a dificuldade em ensinar algoritmo por parte dos professores e o aprendizado por parte dos alunos, visto que deve-se exercitar efetivamente o pensamento matemático.

Considerado o algoritmo uma seqüência finita de ações psicológicas voltadas à realização de um objetivo, precisa-se entender a produção do algoritmo como fragmento do pensamento matemático. Segundo Muniz (2001), por conseqüência, a investigação do pensamento matemático, via algoritmo, tem exigido que se considere que tal produção possa revelar o processo de pensamento do pesquisador, um trabalho de interpretação e dedução que garanta a produção de hipóteses. A revelação do processo do pensamento matemático de forma mais explícita, além da elaboração de hipóteses,

requer um trabalho de mediação junto ao aluno para revelar elementos de análise que somente a interpretação da representação escrita do algoritmo não pode traduzir a real complexidade que é a produção matemática.

Por não identificar tais relações, o aluno abandona o processo de desenvolvimento de algoritmo, abdicando do pensamento autônomo, para então, filiar-se aos algoritmos prontos pelo colega ou professor e, portanto, assim não representam esquemas de pensamento produzido pelo próprio aluno e sim como uma reprodução de algoritmos estáticos, fechados e sem significação ao aluno (MUNIZ, 2001).

A experiência dos alunos é aprender algoritmo, mas raramente inventá-los ou mesmo analisá-los. Um exemplo é a aritmética básica. A proficiência na execução do algoritmo da divisão com números grandes pode ter relativamente pouca importância nesta era da calculadora, mas compreender como funciona o algoritmo (e não meramente como obter na calculadora o resultado) explica, por um lado, porque se obtém um padrão de repetição na expressão decimal de 1/7, por exemplo, e porque se obtém o mesmo padrão de repetição (embora "deslocado") na expressão de 4/7. A análise do algoritmo está bem dentro das capacidades dos alunos – enquanto alcançar proficiência na sua execução pode requerer trabalho compulsivo enfadonho – e mesmo assim fornece certos conhecimentos que são a base para outros estudos (GOLDENBERG, 1998).

Neste capítulo, serão abordados alguns conceitos ensinados na disciplina de algoritmos com o objetivo de apresentar as principais características do processo de ensino-aprendizagem da mesma nos cursos de Ciência da Computação. Além disso, serão apresentados alguns ambientes computacionais de apoio ao ensino e aprendizagem desta disciplina.

2.2 Conceitos Básicos

A abordagem de alguns conceitos básicos no ensino e aprendizagem de algoritmos é importante para o entendimento dos temas trabalhados nesta dissertação. Entre eles pode-se citar: lógica, raciocínio, raciocínio lógico, lógica de programação e linguagem de programação. Estes são apresentados abaixo e a seguir apresenta-se também o conceito de algoritmo em um item à parte.

Lógica

FORBELLONE (1993) diz que lógica é a arte de pensar corretamente e que ensina a colocar em ordem o pensamento.

Raciocínio

É relacionar idéias coerente, consciente e deliberadamente. Raciocinar é transformar as informações disponíveis, tornando-as mais claras, e encontrar as respostas adequadas às perguntas colocadas (CAEIRO, 2004).

Raciocínio Lógico

Raciocínio lógico é a capacidade de identificar e compreender o que há de essencial e de geral em fatos isolados, bem como perceber o conteúdo de um conceito geral em toda a sua extensão, estabelecendo relações entre os dados analisados (PORTAL, 2004).

Lógica de Programação

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo. A lógica de programação pode ser representada em inúmeras linguagens de programação, como por exemplo, Java, Pascal, C++, etc. Entretanto, as linguagens envolvem um grande número de detalhes computacionais. Para melhor representar o raciocínio da lógica de programação utiliza-se os algoritmos (GUIMARÃES, 2004).

Linguagem de Programação

Uma linguagem de programação é um conjunto finito de símbolos com os quais se escrevem programas de computador. Um programa determina o que um computador deve fazer para, a partir de um conjunto de informações de entrada, obter uma saída determinada (GUIMARÃES, 2004).

2.3 Algoritmo

Para FARRER *et al.* (1989) “algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações”.

Um computador é uma máquina que executa ordens a fim de solucionar um determinado problema. Para isso, é necessário que o aluno desenvolva seu raciocínio lógico sobre o problema em questão. Assim, utiliza-se os algoritmos para permitir que os alunos resolvam um determinado problema através de uma seqüência lógica e finita de instruções (OLIVEIRA, 1999).

Segundo SALIBA (1992), dá-se o nome de algoritmo à especificação da seqüência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetibilidade; ou seja, um algoritmo é uma lista de instruções para a execução, passo a passo, de algum processo.

Há inúmeros casos que podem exemplificar o uso (involuntário ou não) de algoritmos para a padronização do exercício de tarefas rotineiras, como por exemplo; uma receita de bolo, trocar um pneu furado ou uma lâmpada queimada.

2.4 Formas de Representação de Algoritmos

Existem diversas formas de representação de algoritmos, mas não há um consenso com relação à melhor delas. Para SALIBA (1992), o critério usado para classificar hierarquicamente estas formas está diretamente ligado ao nível de detalhe ou, inversamente, ao grau de abstração oferecido. O processo de abstração é uma abordagem dada à solução do problema, onde se consideram apenas os aspectos que são importantes para sua solução.

Algumas formas de representação de algoritmos tratam os problemas apenas em nível lógico, abstraindo-se de detalhes de implementação muitas vezes relacionados com alguma linguagem de programação específica.

Por outro lado, existem formas de representação de algoritmos que possuem uma maior riqueza de detalhes e muitas vezes acabam por obscurecer a idéia principal, o algoritmo, dificultando seu entendimento.

Dentre as formas de representação de algoritmos mais conhecidas se destacam:

- Descrição Narrativa: nesta forma de representação os algoritmos são expressos diretamente em linguagem natural. Esta representação é pouco empregada na prática porque o uso da linguagem natural muitas vezes dá oportunidade a más interpretações, ambigüidades e imprecisões.
- Fluxograma: é uma representação gráfica de algoritmos onde se têm formas geométricas diferentes que implicam em ações (instruções, comandos) distintas. Há vários padrões que definem as formas geométricas das figuras que devem ser usadas para representar cada um dos diversos tipos de instruções; contudo, nenhum deles se sobressai com relação aos demais no que diz respeito à aceitação por parte dos usuários. De modo geral, um fluxograma é uma representação gráfica do fluxo de controle de um algoritmo, denotado por setas indicando a seqüência de tarefas (representadas por retângulos) e pontos de tomada de decisão (representados por losangos).
- Pseudocódigo, também conhecido como Linguagem Estruturada ou Portugol[2]: antes de utilizar uma linguagem de programação de computadores, é necessário ordenar as ações a serem tomadas pela máquina de forma organizada e lógica, sem se preocupar com as regras rígidas da sintaxe da linguagem. Para isto, é utilizado um padrão para escrever tais ações, conhecido como pseudocódigo. A forma geral da representação de um algoritmo em pseudocódigo pode ser observada a seguir.

```
Algoritmo <Nome_do_Algoritmo>
VAR
        <declaração_de_variáveis>
Início
        <corpo_do_algoritmo>
Fim
```

Figura 1 – Representação geral de um algoritmo
Fonte: Saliba (1992)

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

Dentre as três formas de representação apresentadas o pseudocódigo será a utilizada neste trabalho.

2.4.1 Variáveis

Variável é uma posição de memória, representada por um nome simbólico (atribuído pelo usuário), a qual contém, em um determinado instante, uma informação correspondente a um tipo de dado, que pode ser inteiro, real, *string* ou lógico.

2.4.2 Instruções Primitivas

As instruções primitivas são os comandos básicos que efetuam tarefas essenciais para a operação dos computadores, como entradas e saídas de dados (comunicação com o usuário e com os dispositivos periféricos), e movimentação dos mesmos na memória. Elas são descritas no corpo do algoritmo. São as seguintes as instruções primitivas:

- Instrução Primitiva de Atribuição: é a principal maneira de se armazenar uma informação numa variável. É necessário haver compatibilidade entre o tipo de dado resultante da avaliação da expressão e o tipo de dado da variável, no sentido em que esta deve ser capaz de armazenar o resultado da expressão. Mais explicitamente, se uma expressão resulta num valor lógico, então a variável deve ser também do tipo lógico. Uma exceção é o caso em que a variável é do tipo real e a expressão resulta num valor inteiro. Nesta situação, o resultado (dado do tipo inteiro) é convertido para o tipo real e posteriormente armazenado na variável. Sua sintaxe é:

<variável> ← <expressão>

Figura 2 – Instrução de atribuição
Fonte: Saliba (1992)

- Instrução Primitiva de Saída de Dados: é o meio pelo qual informações contidas na memória dos computadores são colocadas nos dispositivos de saída, como por exemplo, o monitor. Sua sintaxe é:

Escreva <lista_de_variáveis>
Ou
Escreva <string>

Figura 3 – Instrução de saída de dados
Fonte: Saliba (1992)

- Instrução Primitiva de Entrada de Dados: a semântica da instrução de entrada (ou leitura) de dados é de certa forma, inversa a da instrução de escrita: os dados são fornecidos ao computador por meio de um dispositivo de entrada e armazenados nas posições de memória das variáveis, cujos nomes aparecem na lista_de_variáveis. Sua sintaxe é:

Leia <lista_de_variáveis>

Figura 4 – Instrução de entrada de dados
Fonte: Saliba (1992)

2.4.3 Estruturas de Fluxos

As formas de representar os algoritmos utilizam apenas instruções primitivas de atribuição de entrada e saída de dados. Qualquer conjunto de dados fornecido a um algoritmo deste tipo será submetido ao mesmo conjunto de instruções, executadas sempre na mesma seqüência.

Mas, na prática, muitas vezes é necessário executar ações diversas em função dos dados fornecidos ao algoritmo. Em outras palavras, dependendo do conjunto de dados de entrada do algoritmo, deve-se executar um conjunto diferente de instruções. Além disso, pode ser necessário executar um mesmo conjunto de instruções várias vezes. De acordo com o modo como este controle é feito, são classificadas em:

- Estruturas Seqüenciais: na estrutura seqüencial os comandos de um algoritmo são executados numa seqüência pré-estabelecida. Cada comando é executado somente após o término do comando anterior.
- Estruturas de Decisão: neste tipo de estrutura o fluxo de instruções a ser seguido é escolhido em função do resultado da avaliação de uma ou mais condições. Uma condição é uma expressão lógica. Existem dois tipos de estruturas de decisão: se-então-senão e escolha-faça.

- Estruturas de Repetição: são muito comuns as situações em que se deseja repetir um determinado trecho de um programa um determinado número de vezes. Pode-se citar o caso em que se deseja realizar um mesmo processamento para conjuntos de dados diferentes. Por exemplo, um processamento de folha de pagamentos de uma empresa, em que o mesmo cálculo é efetuado para cada um dos funcionários. As estruturas de repetição são muitas vezes chamadas de laços, ou também, de *loops*. A classificação das estruturas de repetição é feita de acordo com o conhecimento prévio do número de vezes que o conjunto de comandos será executado. Assim, os laços dividem-se em:
  - Laços Contados: os laços contados são úteis quando se conhece previamente o número de vezes que se deseja executar um determinado conjunto de comandos. Então, este tipo de laço nada mais é que uma estrutura dotada de mecanismos para contar o número de vezes que o corpo do laço (ou seja, o comando composto em seu interior) é executado. Os laços contados podem ser de três tipos: para-faça, enquanto-faça, repita-até.
  - Laços Condicionais: laços condicionais são aqueles cujo conjunto de comandos em seu interior é executado até que uma determinada condição seja satisfeita. Ao contrário do que acontece nos laços contados, nos laços condicionais não se sabe de antemão quantas vezes o corpo do laço será executado, pelo fato de o mesmo estar amarrado a uma condição sujeita a modificações pelas instruções do interior do laço. Os laços condicionais podem ser de dois tipos: enquanto-faça e repita-até.

2.5 Resolução Estruturada de Problemas

No mundo do desenvolvimento de software, as décadas de 70 e 80 foram dominadas pela abordagem de solução de problemas (paradigma) chamada abordagem estruturada. A abordagem estruturada de problemas decompõe um problema em funções que consiste no método de refinamentos sucessivos ou decomposição funcional, sendo este método útil no projeto e na implementação de software. Segundo SALIBA (1992),

a idéia básica deste método é que se partindo de um dado problema, para o qual se deseja encontrar um algoritmo de solução, deve-se procurar subdividi-lo em problemas menores e de soluções mais simples. Alguns terão soluções imediatas, outros não. Estes que não têm solução direta devem ser novamente subdivididos. Assim, o processo é repetido até que se consiga encontrar um algoritmo para solucionar cada um dos subproblemas. Portanto, o algoritmo de solução original será composto pela justaposição dos algoritmos usados para solucionar cada um dos subproblemas que o problema original foi decomposto.

Este método é chamado de *top-down* (cima para baixo), pois incentiva uma abordagem gradativa do problema. A cada passo do refinamento aumenta a quantidade de detalhes tratados.

O método traz vantagens na (SALIBA 1992):

- subdivisão de algoritmos complexos, facilitando o seu entendimento (hierarquia de blocos, segundo GANE (1983));
- estruturação de algoritmos, facilitando principalmente a detecção de erros e a documentação de sistemas, e;
- modularização de sistemas que facilita a manutenção de softwares e a reutilização de subalgoritmos já implementados (alterabilidade, segundo GANE (1983)).

O algoritmo pode ser subdividido na fase de análise *top-down*, que procura resolver o problema em nível de subalgoritmos e comandos compostos, e na fase de programação estruturada, que detalha os subalgoritmos e comando composto até o nível das instruções primitivas e construções de controle de fluxo. A metodologia impõe o uso de apenas três regras básicas para formalizar a lógica de solução de um problema (MARTIN & McCLURE, 1991):

- Seqüência, que implementa procedimentos incondicionais;
- Condição, que implementa procedimentos condicionais, possibilitando o fluxo de execução em duas direções, com base na ocorrência lógica;
- Repetição, que possibilita a construção de um conjunto de procedimentos que serão executados repetidamente e controlados por uma condição lógica.

A resolução de problemas estruturados permite a visualização em diversos níveis de generalização, facilitando o entendimento e assimilação da lógica utilizada (SOUZA,

2000), juntamente com a utilização de ferramentas para a construção de algoritmos, pois estimula o aprendizado e facilita a compreensão dos fluxos de controle.

Por várias gerações de programadores, esta tem sido a abordagem mais natural para solução de problemas. Em meados de 1970 uma importante pesquisa sobre um paradigma diferente de desenvolvimento de software e solução de problemas - a abordagem da orientação por objetos. Nesta abordagem, a decomposição de dados, ao invés da decomposição de funções, torna-se a idéia central (SCHNEIDER, 2005). As funções se tornam ligadas a um modelo de dados e servem a este modelo de dados. A solução de problemas passa a ser a descrição e modelagem de como objetos interagem entre si.

Outros paradigmas de programação têm sido criados além do estruturado e do orientado por objetos. Programação funcional, exemplificada pela linguagem de programação LISP e programação lógica exemplificada pela linguagem de programação PROLOG são dois exemplos. Programação funcional tem sido largamente usada em aplicativos de inteligência artificial e PROLOG em aplicativos para aprendizagem de máquinas (SCHNEIDER, 2005).

Tanto a abordagem estruturada quanto a orientada à objetos promovem soluções específicas aos problemas que estão sendo tratados. Este trabalho terá enfoque na resolução estruturada de problemas pelo fato de que na disciplina de lógica de programação dos cursos da área tecnológica ainda adotarem algoritmo estruturado, sendo responsável por um grande índice de reprovação na disciplina. Outro fator relevante é o fato de que o aluno estuda os principais conceitos na criação de um algoritmo: declaração de variáveis e estruturas seqüencial, de decisão e de repetição. Estes são de suma importância na elaboração de um algoritmo tanto estruturado quanto orientado a objetos (SCHNEIDER, 2005).

2.6 Processo de Ensino-Aprendizagem de Algoritmos

A aprendizagem de algoritmo é essencial para todas as carreiras ligadas à Informática. Programação é, sem dúvida, a disciplina mais importante para a formação daqueles que terão no desenvolvimento de software o produto final do seu trabalho. Uma vez que a aprendizagem de algoritmos ocorre praticamente durante todo o curso, o

baixo índice de assimilação dos estudantes nas disciplinas cujos requisitos exigem esse conhecimento tem sido um grande problema enfrentado em muitas instituições de ensino (PIMENTEL, 2003).

Apesar de várias metodologias propostas terem verificado melhores índices de aprendizado no domínio de algoritmo, não se encontrou na literatura pesquisada, metodologias que possibilitem tratar cada aprendiz de maneira diferenciada. Ou seja, as metodologias geralmente são aplicadas de maneira uniforme em turmas inteiras.

Os alunos possuem origens, experiências e habilidades diferentes. Isto justifica, em parte, o fato de alunos de uma mesma classe, submetidos às mesmas condições de ensino, apresentarem resultados distintos (PIMENTEL, 2003) e reforça a necessidade do uso de técnicas variadas que permitam ampliar os resultados de ensino. É claro que, apesar de cada um aprender de forma diferente, existem também pontos comuns a certos grupos de alunos, que são os estilos de aprendizagem.

Um estilo de aprendizagem é um método que uma pessoa usa para adquirir conhecimento (GARDNER, 1995). Um estilo de aprendizagem não é o que a pessoa aprende e sim o modo como ela se comporta durante o aprendizado. O processo de ensino-aprendizagem de algoritmos é desafiador para o professor, visto que o aprendizado de algoritmos é individual a cada aluno, pois o mesmo possui formas diferentes de pensar e raciocinar em determinar diferentes soluções para um mesmo problema, de maneira correta.

Muitas teorias têm sido desenvolvidas nos campos da Educação e Psicologia para explicar como as pessoas pensam e aprendem. GARDNER (1995) coloca que os indivíduos não possuem uma inteligência fixa, mas pelo menos sete diferentes modos de aprender que podem ser aprimorados ao mesmo tempo e desenvolve ou apresenta a teoria das inteligências múltiplas.

Entre as inteligências apresentadas por GARDNER (1995), a inteligência lógica-matemática relaciona-se com o mundo através da capacidade do indivíduo em pensar, raciocinar, calcular padrões e seqüências. O uso da lógica na vida de uma pessoa é constante, visto que é ela quem possibilita a ordenação do pensamento humano (FORBELLONE, 1993).

A forma de pensar e agir diferente do indivíduo possibilita várias soluções para um mesmo problema. Para facilitar este processo, o professor deve buscar formas que

estimulem o aprendizado e formalize a lógica de programação através de ferramentas apropriadas, como já mencionado, e de incentivo ao aluno, fazendo com que o mesmo não desanime ao encontrar uma solução errônea. Ao mesmo tempo, deve haver um treinamento em técnicas de estruturação de algoritmos, fazendo com que o raciocínio lógico flua com naturalidade na determinação de soluções de problemas, fazendo com que estas sejam bem projetadas e otimizadas, propiciando ao aluno saber onde e como iniciar seu algoritmo.

Como forma de facilitar o entendimento dos processos de construção de algoritmos no início do processo de aprendizagem, o professor deve escolher preferencialmente problemas inseridos no contexto da realidade cotidiana dos alunos (FORBELLONE, 1993). Assim, o professor poderá propor estudos de casos buscando aumentar a complexidade dos algoritmos de forma gradual e restrita à lógica, sem se preocupar com as linguagens de programação, por exemplo.

Aprender a programar é um processo complexo e exigente para a maioria dos alunos. No aprendizado de algoritmos, além de conceitos básicos existem conceitos complicados de serem assimilados, como iteração, recursão, passagem de parâmetros, etc.

PIMENTEL (2003) afirma que não é raro ouvir que, “algoritmo e programação não é para todos”. Esta é uma visão simplista daqueles que não questionam a maneira de ensinar. Porém, para uma razoável parcela da comunidade científica, surgem questionamentos sobre como ensinar algoritmos. São pesquisadores preocupados em entender o processo de aprender a programar, detectando falhas e dificuldades deste aprendizado e sugerindo alternativas, de modo a facilitar o aprendizado do aluno na disciplina de algoritmo.

2.7 Como ensinar Algoritmo

O maior problema na construção de algoritmos é a complexidade, portanto, ao construir um algoritmo. O objetivo principal é vencer a complexidade do problema a ser solucionado, que pode ser traduzida como a variedade de condições que envolvem o problema, ou seja, representa a quantidade de situações diferentes que um problema pode apresentar e que devem ser previstas na solução do mesmo (GUIMARÃES, 2004).

Alguns problemas no ensino de algoritmos enfrentados pelos professores desta disciplina podem ser observados nos tópicos a seguir (RODRIGUES, 2002).

2.7.1 Motivação

Muitos professores não conseguem motivar facilmente os alunos a se interessar pela disciplina. Não fica claro para muitos alunos que estão ingressando no curso de graduação em Ciência da Computação a importância de certos conteúdos para a sua formação. O desafio é fazê-los entender que a disciplina e seu conteúdo são importantes, merecendo uma maior atenção e dedicação especial. É necessário mostrar que o conhecimento adquirido não é suficiente para resolução de problemas mais complexos que exigirão toda uma metodologia de desenvolvimento (RODRIGUES, 2004).

Ainda relacionado com a motivação há um paradigma criado dentro e fora das salas de aula. O aluno chega com a idéia fixa de que a disciplina será um problema e um obstáculo extremamente difícil de ser vencido. Além disso, alguns professores se encarregam de completar o negativismo, pois afirmam ser complicadíssimo o aprendizado e prometem avaliações muito difíceis, que podem não atingir o verdadeiro objetivo de avaliar.

Outro fator importante é fazer com que o aluno participe intensamente das aulas teóricas com perguntas e exercícios, motivando-os através de palavras de apoio, incentivo e gratificação.

2.7.2 Avaliação

As avaliações são um segundo problema do ensino tradicional da disciplina. Geralmente são extensas para o tempo alocado e são um ato isolado, especial, com data marcada e que desenvolve ao aluno uma certa repugnância e temor pela disciplina, prejudicando sua aprendizagem.

Segundo RODRIGUES (2004), a avaliação deve garantir o desempenho e se transformar em acompanhamento diário, para, desde logo, facilitar tratamentos especiais para cada dificuldade especial. O professor deve viabilizar toda instrumentação necessária para garantir o bom desempenho, progresso sustentado e aprimoramento da

qualidade. O objetivo principal é que ela sirva de parâmetro básico para que o professor possa verificar quais são as principais dificuldades da turma, analisando os pontos positivos e possíveis problemas que emperram, dificultam ou até mesmo inviabilizam a aprendizagem.

Durante o decorrer do ensinamento da disciplina é possível buscar formas alternativas de avaliação para que a mesma não seja realizada em um único momento, numa grande avaliação, e sim com base num conjunto de pequenas avaliações, que vão alimentando e reorientando os processos de mudança. O professor pode propor problemas rápidos durante as aulas e/ou passar exercícios que servirão de complemento para a nota final do aluno. Isto porque diversos alunos não conseguem expor todo conhecimento adquirido em um único momento, ficando psicologicamente abalados com a avaliação.

2.7.3 Relacionamento

Um terceiro problema diz respeito ao relacionamento professor-aluno, pois os professores ficam preocupados em mostrar competência e acabam desconsiderando a importância do item relacionamento humano na melhoria da qualidade das relações de comunicação e afetivas entre professores e alunos, fundamentais ao processo de aprendizagem. O professor deve assumir o papel de agente estimulador do aprendizado, em diálogo constante com aluno e, principalmente, numa relação de afeto e cooperação, orientar os melhores caminhos para o aprendizado da disciplina. Professor e aluno devem ser companheiros de trabalho numa busca contínua de conhecimento.

2.7.4 Didática

Este problema está intimamente relacionado com a didática usada pelo professor para apresentar o conteúdo introdutório da disciplina, pois a introdução requer uma boa capacidade de abstração. O aluno, normalmente, tem dificuldade de entender os diversos e novos conceitos referentes à execução de tarefas pelo computador.

O material didático e a preparação das aulas também devem estar sujeitos a adaptações e não seguir um conteúdo fixo. O professor deve ter a flexibilidade de saber quando a classe está preparada para um novo conteúdo e adequar o planejamento à

realidade atual. A própria aula deve ser um verdadeiro algoritmo, podendo ser usada como exemplo. Este algoritmo deve mesclar o básico para uma boa aula expositiva com a introdução de algumas dinâmicas de grupo para auxiliar o entendimento. A aula deve sempre iniciar com objetivos claros do que será abordado e principalmente com uma pequena recapitulação do que foi visto na aula anterior.

O professor deve investigar, com alguns questionamentos, se os conhecimentos necessários para a seqüência do assunto estão sólidos, utilizando mais alguns minutos, se for necessário, para que o entendimento seja completo. Ao final da aula deve haver uma conclusão com a reafirmação do que foi visto no dia e a preparação para a próxima aula.

Os professores estão sempre experimentando mudar suas aulas, baseado em relatos de alunos que não obtiveram êxito no aprendizado e também em relatos de alunos que não conseguiram um aproveitamento excelente, mas apenas o necessário para serem aprovados. Portanto, o professor deve ser capaz de fazer o aluno compreender a abstração envolvida com toda simbologia utilizada, usando a sua criatividade e tentar resolver cada problema baseando-se em situações do cotidiano. O professor também pode utilizar desenhos, caixas de papel para representação de variáveis e os próprios alunos para facilitar o entendimento.

O modo como a disciplina de algoritmos é apresentada, segundo RODRIGUES (2002, p. 5),

> *"normalmente demonstra um certo descompasso com todas as inovações e técnicas educacionais atuais. Pelo fato de ser uma disciplina da área tecnológica e o professor, via de regra, não ter formação pedagógica, fica ainda mais complicado o êxito dos alunos. Em disciplinas como essa, é notória a necessidade da "Arte de Ensinar" para o professor universitário. Vale salientar que o fracasso de grande parte dos alunos está associado a uma falta de preparo dos mesmos para o ingresso na universidade. Isto ocorre, principalmente, em instituições particulares de ensino, onde o processo seletivo é menos concorrido e acabam ingressando alunos sem nenhuma base, principalmente quanto aos conteúdos da matemática. Como esta base é indispensável para o aprendizado da disciplina, fica muito difícil a assimilação do conteúdo programático*."

Ensinar, de uma forma geral, é uma tarefa de enorme complexidade, e os processos cognitivos envolvidos ainda são desconhecidos mesmo para profissionais humanos. A tarefa de ensinar algoritmos, em particular, demanda uma enorme carga cognitiva, uma vez que não se restringe apenas à capacidade de trabalhar com uma sintaxe específica, mas também com as habilidades de decomposição de problemas e de composição de soluções possíveis (SANTOS, 2003).

O professor ajuda o aluno a encontrar o caminho correto, todavia, se não houver interesse do aluno também não haverá sucesso no aprendizado.

2.7.5 Cooperação

O diálogo se constrói e se concretiza através de ações conjuntas entre indivíduos autônomos através de parcerias que não se realizam se não houver cooperação (ERICKSON, 1989). A Cooperação envolve empenho mútuo dos participantes em um esforço coordenado para solucionar juntos um problema.

Para PANITZ (2002) cooperação é uma filosofia de interação e estilo de vida pessoal na qual as pessoas são responsáveis por suas ações, incluindo aprendizagem e respeito pelas habilidades e contribuições de seus pares. A cooperação ultrapassa o indivíduo e exige a presença de outros, principalmente na contemporaneidade, uma vez que ninguém é auto-suficiente. Mesmo vivendo em sociedade tão competitiva os indivíduos precisam agrupar-se em grupos que não podem ser apenas conjuntos de pessoas justapostas e sim pequenos grupos coletivos integrados (CORTELAZZO, 2000).

2.8 Ambientes para o Ensino-Aprendizagem de Algoritmos

Um dos grandes problemas do modo tradicional de ensino em algoritmos é a dificuldade de motivar os alunos, de fazer com que se interessem pela disciplina e fazê-los entender que seu conteúdo é importante e também fundamental como base para outras disciplinas. No contexto da aprendizagem baseada em computador, alguns ambientes foram desenvolvidos para auxiliar o ensino e aprendizagem de algoritmos. Abaixo estão descritos alguns destes ambientes.

2.8.1 HabiPro

O *HabiPro* (*Habits of Programming*) (VIZCAÍNO *et al*., 2000) é um software pedagógico e colaborativo para desenvolver “bons hábitos” em programação. Ele não ensina a programação, mas estimula os estudantes novatos em programação a adquirirem habilidades como a observação e reflexão da estrutura do algoritmo, necessários para se tornarem bons programadores.

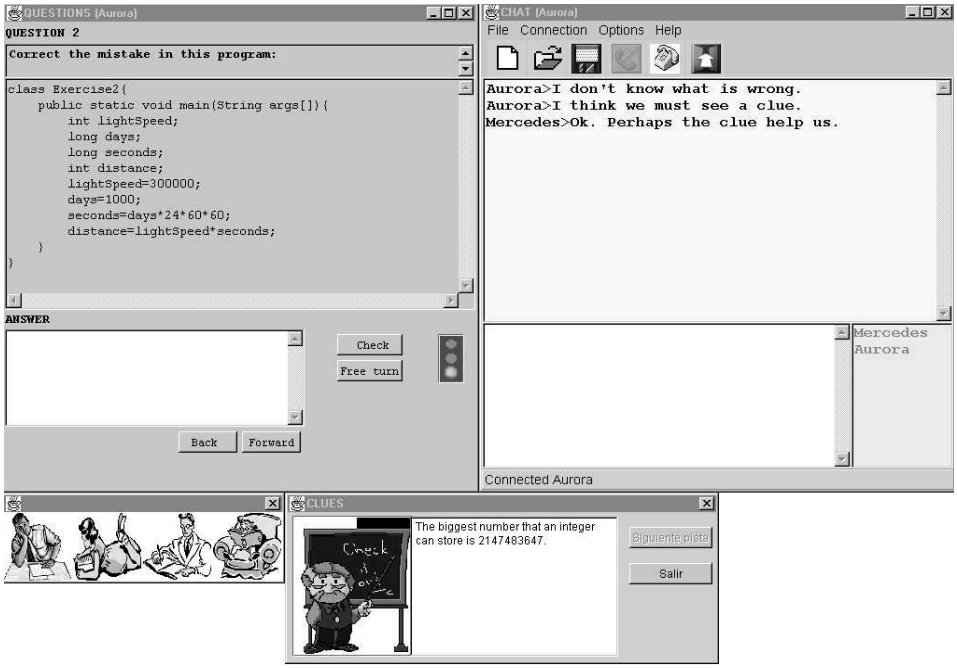

Figura 5 – Interface do HabiPro
Fonte: VIZCAÍNO *et al*., 2000

A interface desta aplicação tem duas janelas, sendo uma delas um *chat* para a comunicação entres os estudantes, e uma outra janela, que é a área de trabalho onde os alunos devem colaborativamente resolver um determinado problema de programação. Após a proposta de uma solução do grupo ser apresentada, se esta não está correta, o sistema propõe quatro tipos de ajuda:

1. Oferece suporte ao estudante de como resolver o problema;
2. Mostra a solução e uma explicação do porquê o problema ter sido resolvido com aquela técnica;
3. Mostra um exemplo similar do problema que o estudante tentou resolver e sua solução; ou
4. Mostra a solução do problema.

2.8.2 Visualg

A idéia para este programa nasceu da necessidade de uma ferramenta para os alunos iniciantes em programação exercitarem seus conhecimentos num ambiente próximo da realidade de uma linguagem de programação. O programa utiliza uma linguagem parecida com o Portugol. Com esse programa os princípios básicos da programação estruturada poderiam ser ensinados sem que a curva de aprendizagem fosse íngreme. O programa também deveria ser capaz de simular o que acontece na tela do computador com os famosos “leia” e “escreva”, bem como possibilitar o exame dos valores de variáveis e um acompanhamento passo a passo da execução do “programa” (pelo seu grande valor didático). Aliado a isto deveria estar um editor com recursos razoáveis, tais como salvar e abrir algoritmos, utilizando todos os recursos que o ambiente Windows provê (VISUALG, 2004).

A linguagem que o *Visualg* usa para a criação de algoritmos é bem simples. Permite apenas um comando por linha, e por isso não há necessidade de caracteres separadores de estruturas, como o ponto e vírgula, por exemplo; também não existe o conceito de blocos delimitados por caracteres especiais ou palavras-chave como *begin* e *end*. Não há subprogramas, comandos de desvio incondicional, ou provisão para recursividade ou variáveis locais (todas as variáveis são de escopo global). Na versão atual do *Visualg*, também não há nenhuma função ou procedimento embutido.

Segundo VISUALG (2004) o formato básico de um algoritmo é o seguinte:

```
algoritmo “semnome”
// Função :
// Autor :
// Data : 06/10/2002
// Seção de Declarações
inicio
// Seção de Comandos
fimalgoritmo
```

Figura 6 – Formato de um algoritmo em VisualG
Fonte: VisualG (2004)

A primeira linha de um algoritmo compõe-se da palavra-chave algoritmo, seguida do seu nome delimitado por aspas. A seção que se segue é a de declaração de variáveis que termina com uma linha com a palavra-chave inicio. Deste ponto em diante está a seção de comandos, que vai até uma linha onde se encontra a palavra-chave fimalgoritmo. Esta linha marca o fim do código-fonte[3], e todo texto existente a partir dela é ignorado pelo interpretador.

2.8.3 Logo

A linguagem de programação Logo foi desenvolvida com finalidades educacionais por um grupo de pesquisadores do *Massachusetts Institute of Technology*. É uma linguagem considerada, ao mesmo tempo, simples e sofisticada. Do ponto de vista educacional é uma linguagem simples, porque possui características que torna acessível o seu uso por sujeitos de diversas áreas e de diferentes níveis de escolaridade. Computacionalmente, Logo é considerada uma linguagem bastante sofisticada, por possuir características pertencentes a três paradigmas computacionais distintos: procedural, orientado a objetos e funcional (PRADO, 2000). Logo se caracteriza pela presença de um cursor representado pela figura de uma Tartaruga, que pode ser deslocada no espaço da tela por meio de alguns comandos primitivos. A manipulação da Tartaruga permite ao sujeito lidar com o espaço topológico e caracteriza um modo de fazer geometria, conhecida pelo nome de Geometria da Tartaruga. Os comandos básicos da Tartaruga são:

- parafrente <número>: desloca a Tartaruga para frente um determinado número de passos
- paratrás <número>: desloca a Tartaruga para trás um determinado número de passos
- paradireita <número>: gira a Tartaruga à direita um determinado ângulo
- paraesquerda <número>: gira a Tartaruga à esquerda um determinado ângulo
- repita <número> <lista de instruções>: repete a lista de instruções um determinado número de vezes

[3] O código fonte é um texto escrito em uma linguagem de programação, usada para criar um programa.

Segundo FERRUZZI (2001), a linguagem Logo tem algumas características:

- Aprender ensinando: No trabalho com o Logo o usuário programa o computador sem perceber que está realizando uma programação. Para programar o computador ele “ensina” a tartaruga. Ensinar a tartaruga implica em fazer uma descrição de movimentos para que esta execute determinado procedimento. É preciso descrever com formalidade e na linguagem que esta entende.
- Integração das atividades corporais com as intelectuais: O Logo possibilita integrar habilidades corporais com as intelectuais. Esta integração ajuda no desenvolvimento da lateralidade e do raciocínio, fundamentais para se conseguir construir algum trabalho.
- Intelectuais: Nas atividades é preciso que o usuário se coloque no lugar da tartaruga. No momento em que ele fornece um comando tem que se colocar no lugar da tartaruga, lembrando-se sempre que a posição desta é diferente da sua. Na tela, a tartaruga é apresentada na vertical e o usuário anda na horizontal. O usuário tem que se imaginar sendo tartaruga e descobrir quais comandos deve fornecer para que a tartaruga realize o que ele deseja. Isto possibilita ao usuário uma maior compreensão quanto ao seu conhecimento espacial, de coordenadas e dos pontos cardeais.
- Ciclo descrição – execução – reflexão – depuração: No desenvolvimento de um projeto o usuário possui uma idéia do que pretende realizar e descreve, por uma série de comandos, o que deseja que a tartaruga execute. Esta, por sua vez, “obedece”, isto é, executa cada comando e a movimentação vai sendo apresentada na tela. Nesse momento, o aluno pode visualizar o que foi executado, realizando uma reflexão sobre a sua idéia original e o que foi executado pela tartaruga. Se o resultado é o esperado seu trabalho foi realizado com sucesso. Caso contrário, ele depura o procedimento para encontrar a solução desejada. Assim, o controle de todo o processo está nas mãos do aprendiz.
- Tratamento do erro: No Logo, o erro é visto como um importante fator de aprendizagem, pois possibilita ao aluno a compreensão do motivo pelo qual aconteceu o erro e a busca de soluções para o seu problema. Tem-se assim, a

aprendizagem por descoberta. Quando o usuário erra algum conceito o programa faz uma pergunta amigável, questionando o que ela realmente quer dizer, fazendo com que o aluno repense o que deseja. Quando ocorre um erro o aluno pode depurar o procedimento identificando sua origem e usar, com isso, o erro de modo produtivo para entender melhor suas ações.

- Papel do professor: O papel do professor neste contexto é de auxiliar, discutir, estimular e animar o usuário. O professor deve acompanhar o raciocínio do aluno, sem corrigir de imediato suas ações. Deve promover o crescimento deste de modo a auxiliá-lo a encontrar seus erros e corrigi-los por si só. Discutir com o aluno mostrando o direcionamento das soluções sem, no entanto, concluir por ele.

2.8.4 Ambiente para Apoio à Aprendizagem de Programação

Este sistema é um ambiente de suporte à aprendizagem de programação utilizando o paradigma STI, descrito no capítulo seguinte, a interação com os estudantes e a identificação semi-automática de grupos de soluções de problemas propostos, visando facilitar o aprendizado e fornecendo o *feedback* adequado (GIRAFFA, 2003).

Possui como características:

- contribuição para a melhoria quantitativa e qualitativa do *feedback* que é fornecido ao aluno;
- baseia-se na tecnologia de agentes para auxiliar o professor na identificação dos erros realizados pelos estudantes durante a resolução dos problemas propostos;
- 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 arquitetura pode ser visualizada na figura 7 mostrando a interação no ambiente com agentes reais, representados por estudantes, professor e colaborador; e agentes virtuais representados pelo construtor de perfil, assistente de operação e assistente de aprendizagem.

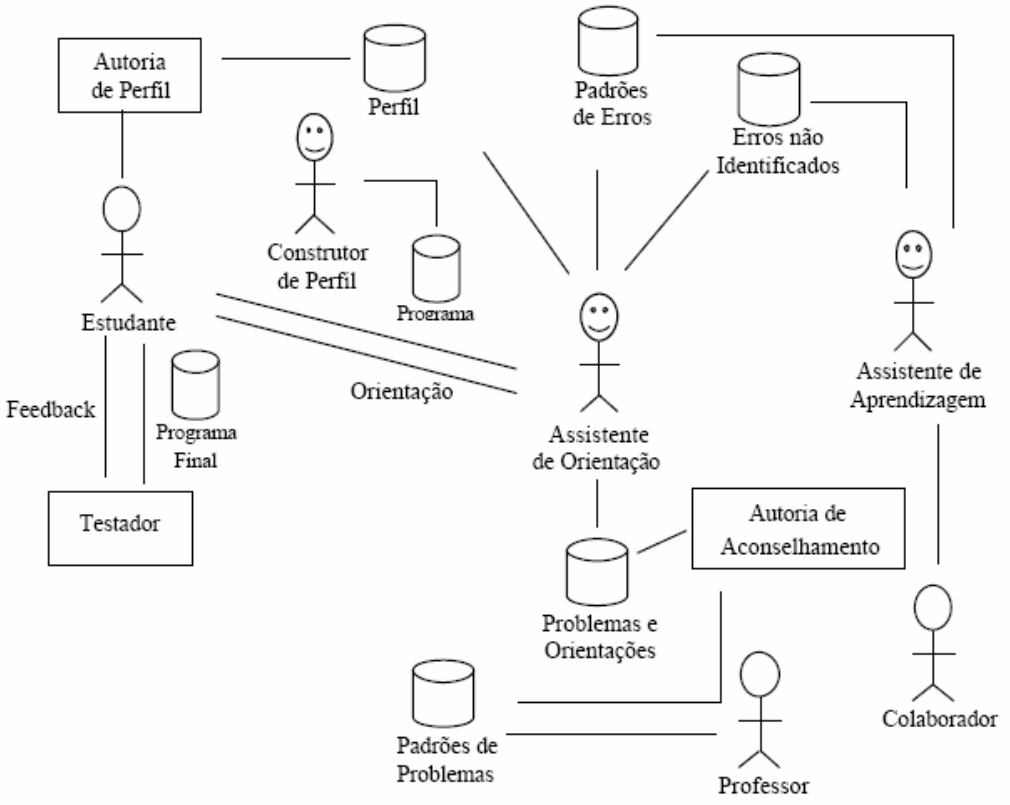

Figura 7 – Arquitetura do ambiente para apoio à aprendizagem de programação
Fonte: GIRAFFA, 2003.

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, com base 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 um erro seja encontrado, o assistente fornecerá um feedback ao estudante. A ferramenta testador serve para avaliar o programa.

2.8.5 Outros Sistemas

Diversos tipos de ferramentas e ambientes têm sido propostos com o objetivo de facilitar o aprendizado de lógica de programação. Além dos sistemas revisados anteriormente, outras ferramentas como o Portugal/Plus serão abordados a seguir.

Segundo ESMIN (1998) o objetivo principal do ambiente Portugol/Plus é fornecer uma ferramenta de apoio ao ensino da lógica de programação baseado no Portugol. O ambiente foi desenvolvido para operar em um microcomputador do tipo PC

(computador pessoal de uso doméstico), sob o sistema operacional DOS (*Disk Operating System*), e divide-se em duas partes principais: Editor de algoritmos e um compilador[4]. Durante o processo de compilação, o compilador do ambiente Portugol/Plus faz uma verificação da sintaxe das instruções usadas no algoritmo. Caso não sejam encontrados erros, um programa objeto é gerado na linguagem de programação Pascal (FARRER, 1995). Através de um compilador Pascal executa-se o programa objeto (ou código objeto).

Um outro protótipo de sistema de ensino e aprendizagem de programação implementado em *Delphi*, proposto por KEMP (2003), é aprender a codificar programas de computadores previamente prontos e criar um diagrama corresponde ao programa já implementado. Neste sistema, os estudantes terão que associar as estruturas de programação (se-então-senão, para-faça, enquanto-faça) com a representação correspondente nos diagramas.

2.9 Considerações Finais

O processo de ensino-aprendizagem de algoritmos é um problema presente nos cursos de Ciência da Computação na maioria das universidades. Ao ensinar, o professor deve ter sempre em mente que os alunos não são especialistas na área de computação e todos os conceitos apresentados são novos para eles. Portanto, o papel do professor é ser capaz de fazer com que o aluno compreenda todas as simbologias utilizadas e consiga com que o mesmo raciocine perante o problema apresentado, demonstrando soluções através das suas habilidades desenvolvidas.

Para o aluno “A” é uma letra do alfabeto. Como fazê-lo entender que “A” é agora uma variável? Portanto, nada melhor do que envolver o aluno em problemas do cotidiano, fazendo com que o mesmo comece a desenvolver seu raciocínio lógico ordenando seu pensamento. Cabe ao professor ser criativo e utilizar formas de representação que facilitem o entendimento do aluno.

---

[4] Compiladores são programas que recebem como entrada um código-fonte, analisam o conteúdo e o contexto, caso encontrem erros informam ao usuário, do contrário, convertem o código-fonte para um código-objeto geralmente em linguagem de máquina.

Alguns ambientes computacionais de ensino-aprendizagem de algoritmos pretendem ser uma ferramenta de apoio ao professor no desenvolvimento do raciocínio lógico dos alunos. A pesquisa destes ambientes serve de apoio para a criação do modelo deste trabalho, levando em consideração aspectos relacionados ao processo de ensino de algoritmos e da aprendizagem pelos alunos.

No capítulo seguinte será descrito o companheiro de aprendizagem, como sendo um companheiro virtual de aprendizagem dos estudantes, em um domínio específico, como proposta dessa dissertação: algoritmo. O SCA pode ser considerado um mecanismo de colaboração para o ensino e aprendizagem de algoritmos. Para isso, foi realizada uma pesquisa bibliográfica que envolvem conceitos de Inteligência Artificial (IA) e Sistemas Tutores Inteligentes (STIs).

3. COMPANHEIRO DE APRENDIZAGEM

3.1 Introdução

A aplicação da Informática na Educação, através da utilização de software educacional, vem demonstrando resultados positivos no processo de ensino-aprendizagem principalmente com da utilização de técnicas de Inteligência Artificial (IA) (VICARI *et al*., 2003; GIRAFFA *et al.*, 2003), atendendo a 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. Segundo RUSSELL & NORVIG (1995) a IA é um ramo da ciência dedicado à compreensão das entidades inteligentes. Um dos objetivos da IA é proporcionar ferramentas formalizadas para registrar os conhecimentos e as heurísticas dos sistemas cognitivos, ou seja, o desenvolvimento de sistemas computacionais que representem o modelo de funcionamento e o comportamento intelectual dos seres humanos na realização de uma determinada atividade.

As técnicas de IA no projeto e desenvolvimento de ambiente de ensino e aprendizagem computadorizado têm se constituído em objeto de pesquisa na área de Inteligência Artificial na Educação (IA-ED) e são significativamente representadas nos Sistemas Tutores Inteligentes (STIs).

A IA-ED é baseada em computador para o apoio à aprendizagem que suporta alguma tomada de decisão autônoma em relação às suas interações com os estudantes (ROSATELLI, 2000). Esse processo de decisão é necessariamente feito on-line, durante as interações do sistema com os usuários e, geralmente, o sistema precisa acessar vários tipos de conhecimento e processos de raciocínio para habilitar tais decisões a serem tomadas. A IA-ED constitui uma área de pesquisa bastante interessante por causa da sua constante inter-relação de idéias e é importante por sua contribuição potencial para o objetivo social de melhorar a qualidade do aprendizado (ROSATELLI, 2000).

Dentro deste contexto, vários paradigmas têm sido propostos, entre eles a utilização de STIs que possibilitaria o desenvolvimento de diferentes raciocínios e a integração de várias ações para alcançar um determinado objetivo (MARIETTO, 1997).

CHAN (2003) propôs uma nova forma de ambientes de aprendizagem inteligentes chamados de Sistemas Companheiro de Aprendizagem – SCAs (Sistemas *Learning Companions* - SLCs) (CHAN & BASKIN, 1988). Os SCAs assumem dois papéis; um como um tutor inteligente e outro como um companheiro de aprendizagem (CHAN, 2003), permitindo que os estudantes aprendam com um companheiro de aprendizagem virtual em um determinado domínio.

3.2 Sistemas Tutores Inteligentes

Os STIs como são conhecidos, constituem em uma tentativa de implementar, num sistema computacional, os métodos tradicionais de ensino e aprendizagem, exemplificados por uma interação um-a-um entre tutor e estudante. Possuem um certo grau de autonomia durante sua interação com o usuário e, para isso, precisam ter acesso a várias formas de representação de conhecimento e a vários tipos de raciocínio (SELF, 1995).

Um STI é um sistema computacional que utiliza técnicas de IA e teorias pedagógicas fazendo o tutoramento de um estudante num dado domínio (disciplina), como por exemplo, a disciplina de algoritmos. O STI modela o entendimento do estudante sobre um tópico e à medida que ele realiza determinadas tarefas no sistema (ou seja, ele interage com o sistema realizando tarefas colocadas por este) compara o conhecimento do estudante com o modelo que ele tem de um especialista naquele domínio. Se existir uma diferença, o sistema pode usar o seu modelo de domínio para, gerar uma explicação que vai auxiliar o estudante a compreender o que ficou mal entendido.

Utilizando um STI o professor dispõe de um ótimo recurso de auxílio em uma disciplina, ajudando o aluno em seu aprendizado e proporcionando um estudo individualizado em um determinado domínio. Existem ainda STIs com abordagem cooperativa, onde alguns destes sistemas oportunizam o trabalho cooperativo entre vários alunos conectados ao mesmo tempo na realização de uma mesma tarefa.

3.2.1 Origens dos STIs

As primeiras modalidades de software educacional caracterizavam-se pelo uso do modelo comportamentalista de estímulo-resposta em que a informação é transmitida e a estratégia de ensino adotada é rígida, preestabelecida e não adaptável ao aluno. Tais sistemas são os chamados Instrução Assistida por Computador (CAI - *Computer Assisted Instruction*), que apenas usam o meio eletrônico para fazer o mesmo que era realizado no papel sem nenhum ganho significativo em nível de ensino-aprendizagem (LUZZI, 1998). A seguir, a tendência dos programas de CAI foi de utilizar as técnicas de IA fazendo com que o software se torne um elemento mais ativo no processo de interação com o aluno, ganhando uma versão mais adaptativa. Assim surgiram os software educacionais denominados Instrução Inteligente Assistida por Computador (ICAI – *Intelligent Computer Assisted Instruction*) ou STIs, onde ensino e aprendizagem estão modelados como um processo cooperativo entre sistema e estudante (LEMOS, 2001).

A existência de uma base de conhecimento, e não uma base de dados convencional, é um dos fatores que determina a diferença entre um STI e um CAI. Outros fatores importantes são que um STI possui a capacidade de diagnosticar erros do estudante em tempo real, a instrução via de regra é baseada sobre princípios cognitivos, e o sistema gera problemas e *feedback* ao estudante, entre outros.

O desenvolvimento dos STIs foi baseado, principalmente, no paradigma de Sistemas Especialistas que são programas de software que usam conhecimento e experiência para simular o desempenho de um especialista humano num domínio particular.

3.2.2 Componentes de um STI

Os STIs fornecem aos estudantes ensino individualizado em um ambiente interativo que promove a aprendizagem ativa (GOODMAN *et al*., 1997). A maioria dos STIs seguem uma estrutura tradicional incluindo três componentes: o modelo do domínio, o modelo do estudante e o modelo do tutor, além da interface gráfica com o usuário, como pode ser visualizado na figura 8.

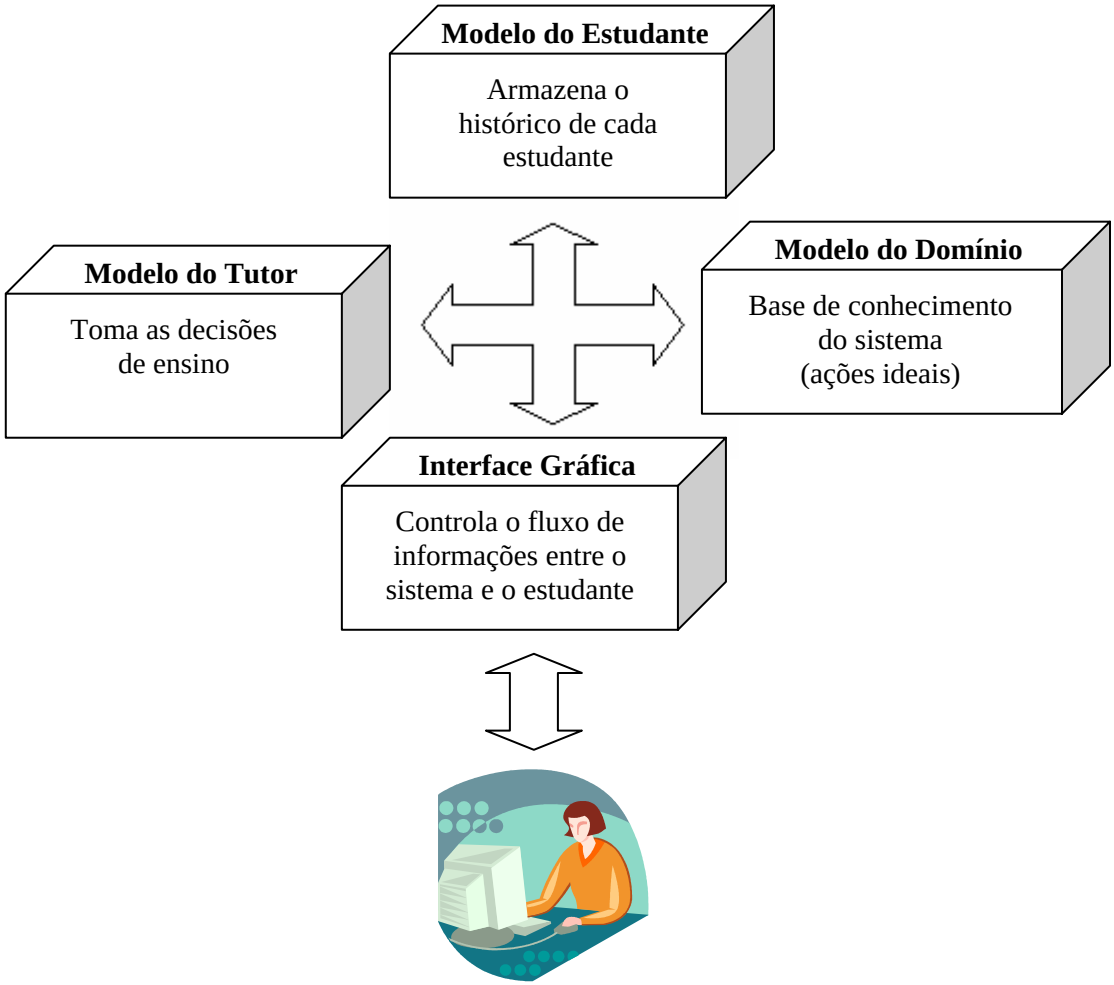

Figura 8 - Componentes de um Sistema Tutor Inteligente Genérico
Fonte: Adaptado de Rosatelli, 1999.

Os STIs percebem as características individuais de cada estudante e se comportam de forma adequada a tais características, facilitando o ensino e o aprendizado. Basicamente, um STI delimita a distância entre o conhecimento do estudante e o que está proposto no modelo do tutor. Os módulos tradicionais deste tipo de sistema são descritos a seguir (ROSATELLI, 2000):

- Interface: Inclui os elementos de entrada e saída para as interações entre o sistema e o estudante, apresentando o conteúdo da disciplina, o material didático, as intervenções do tutor, etc.
- Modelo do Tutor: Trata do conhecimento sobre o ensino. Este módulo interage com o estudante, fornecendo-lhe assistência através de ações baseadas na monitoração do desempenho do estudante.

- Modelo do Domínio: Contém o conhecimento do domínio que se deseja trabalhar com o estudante, isto é, o conhecimento sobre a disciplina a ser ensinada. É a base de conhecimento do sistema. É constituído pelo material instrucional, como lições, textos, vídeos, exercícios, etc.
- Modelo do Estudante: Representa o conhecimento e habilidades do aluno que está interagindo com o sistema. Aqui estão incluídas todas as informações pertinentes sobre o estudante, bem como o conhecimento correto e incorreto do que ele sabe ou não. Segundo GIRAFFA (1999), o modelo do estudante pode ser representado de várias formas. Uma das formas mais usadas é o modelo *overlay*, em que a representação do conhecimento do estudante é a mesma utilizada na base do domínio.
- Mecanismo de comunicação: processa o fluxo de informação dentro e fora do sistema, realizando a coordenação geral do tutor com operações de troca de informações entre os módulos com o estudante e com o sistema operacional.

Através da interação desses modelos, o STI é capaz de fazer julgamentos sobre o que o aprendiz sabe e como ele está progredindo. Os STIs representam uma ótima ferramenta como ambientes de aprendizagem. Porém, devido ao alto custo financeiro e ao elevado tempo de desenvolvimento, muitas propostas têm sido feitas para reduzir estes fatores, convergindo para um ponto em comum, permitindo o reuso do software (AZEVEDO, 2003), ou seja, construir os STIs de forma modular, permitindo apenas adaptações.

3.4 Sistemas Companheiro de Aprendizagem

Um Sistema Companheiro de Aprendizagem (SCA) assume dois papéis, um como um tutor inteligente e outro como um companheiro de aprendizagem (CHAN & BASKIN, 1990).

Segundo CHAN & BASKIN (1990) os Companheiros de Aprendizagem (CAs) possuem conhecimento similar ao do aluno, porém, num nível um pouco mais alto. Os dois realizam em conjunto as mesmas tarefas, trocando idéias de como o problema poderá ser resolvido.

Essa técnica de aprendizagem baseia-se mais no processo de construção cooperativo de conhecimento entre aluno e ajudante do que no processo de transmissão da informação. Um CA pode ser projetado como um companheiro que sugere novas idéias a serem consideradas para o estudante e/ou desafia as próprias propostas do estudante.

Um companheiro de aprendizagem atua como um par virtual num ambiente tutor inteligente encorajando o aluno a aprender, enquanto utiliza as vantagens que um STI proporciona (GOODMAN *et al*., 1997).

Aprender a complexidade das idéias envolvidas em vários domínios de conhecimento, bem como desenvolver a cognição, envolve fracasso e uma base de respostas afetivas associadas. Estas respostas de afetividade podem variar de sentimentos de interesse para sentimentos de confusão e frustração. O estudante pode desistir se ele não for capaz de recuperar o "sentimento de estar sem resposta". A meta de construir um sistema computadorizado com um CA é facilitar os próprios esforços do estudante em aprender (KAPOOR, 2001). Esse companheiro ajudará a manter a exploração do estudante com perguntas ou *feedback*, e pela observação e respostas dos mesmos. O CA é jogador, que está ao lado do estudante para ajudá-lo cada vez mais no seu aprendizado. É um sistema que é sensível à trajetória da aprendizagem dos estudantes.

Os SCAs fazem uso da abordagem pedagógica proposta por Vygotsky que, segundo a construção do conhecimento, implica em uma ação compartilhada, ou seja, através das interações sociais com o meio ou com outros pares semelhantes e eventualmente mais capazes, a relação entre sujeito e de conhecimento é estabelecida (FARACO *et al.,* 2004). O papel do companheiro de aprendizagem é ser um companheiro virtual para o estudante, interagindo com ele a fim de ajudar no seu aprendizado da mesma forma que outro companheiro humano o faria. A grande vantagem é a sua total disponibilidade para com os estudantes, incentivando-os a interagirem com o sistema e aprendendo colaborativamente (GOODMAN *et al.*, 1998).

Os SCAs procuram criar um ambiente favorável para o desenvolvimento cognitivo do aluno ao considerarem as suas capacidades individuais reais e as capacidades potenciais de aprendizado.

A capacidade individual real ou independente do aluno refere-se ao nível de desenvolvimento cognitivo já conquistado por ele, permitindo-lhe desenvolver as atividades e resolver problemas sem a colaboração de outros atores. Já o desenvolvimento potencial está ligado às atividades que o estudante somente é capaz de realizar mediante o apoio de outros pares reais ou virtuais com um maior nível de conhecimento. A diferença entre a capacidade individual de desenvolvimento de problemas dos alunos e aquilo que eles conseguiriam fazer, se recebessem apoio de outras entidades mais capazes, consiste na Zona Proximal de Desenvolvimento (ZPD) de Vygotsky (REGO, 1995).

A ZPD é criada quando existe uma parceria de colaboração, na qual o estudante mais hábil ajuda aquele que se encontra com dificuldades em alcançar seus objetivos de aprendizagem. Dentro da área de informática na educação, ZPD não é nem uma propriedade do ambiente de aprendizagem nem uma propriedade do estudante, e sim uma situação decorrente da interação dos dois e que promove o aprendizado efetivo.

Dentro de um SCA é importante que a quantidade e a qualidade do processo de interação entre o CA e o estudante seja controlada com o intuito de atender às expectativas de aprendizagem do aluno. A falta deste controle pode levar a uma saturação do aluno, caso este receba uma quantidade exagerada de informação do CA, ou mesmo a uma frustração, se as informações não são suficientes ou não ajudam efetivamente o aluno a desenvolver as atividades no curso.

Segundo FARACO (2003), não existe um consenso sobre a composição das técnicas e tecnologias utilizadas no projeto de um SCA. A cada nova proposta deve-se decidir como o SCA será composto, qual o mecanismo de comunicação, como será feita a modelagem dos alunos, do tutor e do domínio.

3.4.1 Tipos de CA

Um CA pode assumir diferentes papéis ao interagir com os estudantes:

- colaborativo: atua junto ao estudante na resolução de problemas (CHAN & BASKIN, 1990);
- competitivo: o CA e o estudante trabalham na solução dos problemas de modo independente e simultâneo e, ao terminarem, os resultados são comparados, os

erros são descobertos e corrigidos, e as soluções corretas recebem créditos pelo tutor, criando um ambiente de competição entre eles (CHAN & BASKIN, 1990);

- aprendiz: neste tipo de perfil ele comporta-se como um aprendiz do estudante humano, conseqüentemente, este estudante necessita revisar, refletir, esclarecer seu próprio conhecimento para que consiga transmiti-lo ao seu par virtual (URESTI, 2000);
- causador de problemas (*trouble maker*): aqui o agente[5] tenta motivar o estudante, provocando-o, a fim de indicar suas deficiências na matéria sendo ensinada (HIETALA & NIEMIREPO, 1998).

Na presente dissertação, o companheiro de aprendizagem terá um papel colaborativo.

Segundo PETRY (2003) com o avanço da tecnologia em diversas áreas, o processo de aprendizagem colaborativa passou a ser utilizado também fora dos limites da sala de aula. Diversos ambientes computacionais foram desenvolvidos para este fim.

A Aprendizagem Colaborativa Apoiada por Computador (CSCL – *Computer Supported Collaborative Learning*) é um paradigma que enfoca, segundo (BRAVO, 2002), a dimensão social do processo de aprendizagem e promove o uso de tecnologias como ferramenta mediadora e facilitadora no processo da construção do conhecimento. Para (MENEZES, 2002) a CSCL é uma estratégia de ensino em que dois ou mais indivíduos constroem o seu conhecimento através da discussão, reflexão e tomada de decisão, e onde os recursos de informática atuam como mediadores do processo de ensino-aprendizagem. O papel do professor é, neste ambiente, ser o facilitador no processo de aprendizagem, tendo como responsabilidades apoiar a interação entre os componentes do grupo, fomentar a colaboração e identificar e explorar oportunidades de aprendizado.

---

[5] Segundo RUSSELL & NORVIG (1995) um agente é definido como um sistema capaz de perceber seu ambiente através de sensores e atuando o ambiente por meio de executores. Um agente pode ser definido como uma entidade de software que exibe um comportamento autônomo, que está situado em algum ambiente sobre o qual é capaz de realizar ações para alcançar seus próprios objetivos de projeto e a partir do qual percebe alterações (WOOLDRIDGE, 1995).

3.4.2 Características dos CAs

Como já foi detalhado, uma das grandes vantagens apresentada pelos SCAs em ter um par virtual e mais apto dentro da arquitetura do sistema proposto é sua total disponibilidade para com os estudantes humanos, incentivando-os a interagirem com o sistema e aprendendo colaborativamente.

Uma das características dos CAs é a sua capacidade de comunicação com o estudante humano através dos vários perfis que um CA pode assumir. A comunicação facilita a cooperação, iniciativa e a autonomia a ser realizada pelo CA. Hietala & Niemirepo (1997) descrevem a complexidade que envolve o entendimento de uma conversa natural entre pares humanos e a dificuldade em desenvolver um CA que pode se comunicar com um estudante humano.

Um CA pode ajudar o estudante a refletir no seu pensamento criticando, questionando ou evoluindo em várias etapas. O estudante e o CA podem articular essas etapas através da explanação ou elaboração de questionamentos (GOODMAN, 1998). A colaboração é considerada, de acordo com Hietala e Niemirepo (1997) como uma das mais importantes qualidades do aprendizado efetivo e significativo.

Uma outra característica do CA é a habilidade para controlar as capacidades, crenças e comportamento do CA de acordo com estratégias pedagógicas particulares. Por exemplo, o aprendizado por distúrbios onde o CA é projetado como um causador de problemas provocando o usuário (CHAN *et al*, 1999).

A modelagem do estudante modela o perfil deste buscando adaptação do sistema às características do usuário. Chan *et al*. (1999) coloca que existem três abordagens de modelagem do estudante em SCAs: um único modelo do estudante provê informações para todos os CAs e tutor onde o modelo é o melhor entendimento do aluno pelo sistema; um modelo do estudante dentro do SCA onde cada agente possui sua própria interpretação do conteúdo do modelo; cada CA possui seu próprio modelo de estudante.

O modelo do tutor dentro de um SCA tem como papel o de coordenar as atividades de aprendizagem, a entrega do material didático, a escolha do CA adequado para cada estudante, a distribuição de responsabilidades entre os estudantes e o CA. Ao contrário do papel nos STIs, a interação com os alunos é feita através do CA.

No modelo do domínio, também chamado de especialista, a matéria ou problemas a ser resolvido é transferido para o aluno. O conhecimento é dividido em duas partes: a primeira refere-se ao conteúdo a ser disponibilizado para o aluno; e a segunda parte, aos procedimentos necessários para resolver os problemas sobre a matéria dentro do sistema.

3.5 Exemplos de Sistemas Companheiro de Aprendizagem

Nesta seção são apresentados alguns SCAs selecionados ao longo da revisão bibliográfica. Cada um destes sistemas traz implícito na sua modelagem a utilização de um CA.

3.5.1 AMICO

O AMICO (*Apprentissage des Mathématiques par Interaction avec des Compagnions*), segundo RASSENEUR (2002), é um protótipo de um sistema para aprimorar o aprendizado da matemática através da interação entre um estudante e vários companheiros virtuais. Ele foi projetado para ser utilizado na sala de aula durante sessões individuais de ajuda ou quando os alunos trabalham em pares. O objetivo pedagógico é obter vários tipos de justificativas e usar diferentes modos de representação nos exercícios. O AMICO foi implementado em Java utilizando a UML (Linguagem de Modelagem Unificada) para desenvolver sua arquitetura. O software oferece dois cenários diferentes de interação com companheiros virtuais: competitivo e cooperação. Foram implementadas quatro características para os CAs. Estas características seguem adaptações para diferentes estilos de aprendizagem e estratégias de ensino.

3.5.2 LuCy

O projeto de desenvolvimento de LuCy surgiu da necessidade da presença de características dos STIs e de sistemas de aprendizagem colaborativa em um único ambiente. LuCy foi desenvolvido para um STI já existente, denominado PROPA, cujo domínio de conhecimento era o ensino de habilidades em análise exploratória das

atividades de satélites, envolvendo habilidades como formulação de hipóteses, busca por evidências e avaliação da veracidade e significado dessas evidências. Analistas de atividades de satélites usam a análise exploratória para interpretar atividades de satélites baseados em informações incompletas, incertas ou inconsistentes (GOODMAN *et al*, 1997).

O CA LuCy é um par interativo que responde questões e oferece sugestões da mesma forma que um estudante colaborador real o faria. A vantagem de LuCy comparando com um par real é sua total disponibilidade para com o estudante humano e sua capacidade de apoiá-lo em dúvidas particulares. A funcionalidade de LuCy está baseada em uma arquitetura onde se fazem presentes um *parser* para tratar as sentenças dos estudantes, o modelo do domínio do conhecimento, o modelo do estudante com histórico de diálogos, seletor e gerador de respostas. Primeiramente, a sentença do estudante é analisada pelo *parser* para determinar o que o estudante quer. Uma vez determinado o significado da sentença do estudante ela é armazenada no histórico de diálogos (GOODMAN *et al.*, 1997). O próximo passo é selecionar qual o tipo de resposta dar ao aluno. LuCy pode dar uma reposta correta ou uma resposta errada, dependendo do conhecimento que o estudante tem do tema particular em estudo. Por último, a resposta é gerada e enviada ao aluno de acordo com o ritmo de interação com o CA. O objetivo do projeto de LuCy foi promover um diálogo efetivo entre os estudantes e o tutor inteligente PROPA. O diálogo promovido por LuCy encoraja o estudante a refletir sobre seu pensamento e a avaliar ações passadas.

3.5.3 LeCo-EAD

O LeCo-EAD é um ambiente SCA para ensino à distância na internet composto por múltiplos agentes *Learning Companion*[6] (AgLCs) com comportamentos próprios e sempre disponíveis para interagir com os estudantes (FARACO *et al*., 2004). O ambiente virtual de aprendizagem contempla diferentes estratégias de ensino, representadas pelos agentes companheiros do tipo colaborador, aprendiz e *trouble maker*. A partir do perfil do aluno identificado por uma escala de atitudes, o LeCo-EAD sugere qual tipo de AgLCs irá atuar como seu companheiro virtual de aprendizagem. O

[6] companheiros de aprendizagem

ambiente ainda tem o propósito de ser independente de domínio, onde tanto o conteúdo do material didático quanto o mecanismo de comunicação dos AgLCs podem ser substituídos.

O LeCo-EAD é um SCA que adota dois tipos de adaptação: a de conteúdo e a de estratégia de ensino (FARACO, 2003). A adaptação de conteúdo é realizada a partir da performance do aluno no curso, em que o sistema, através de um esquema de pré-requisitos, apresenta somente os conceitos que o aluno está apto a desenvolver. O outro tipo de adaptação ocorre quando o sistema solicita a participação do aluno na escolha do AgLC mais adequado ao seu perfil, inicialmente através da escala de atitudes e durante o curso através dos mecanismos de *feedback*.

O protótipo LeCo-EAD caracteriza-se como uma ferramenta que possibilita a autoria dos cursos à distância, bem como a configuração dos agentes companheiros. No que tange à configuração dos AgLCs foi construído um mecanismo de geração dinâmica das mensagens formadas por *templates*, cujos conteúdos são preenchidos com variáveis de ligação sensíveis ao contexto da interação do aluno. O LeCo-EAD faz uso de testes psicológicos, em particular escalas de atitudes, como instrumento que procura descobrir a atitude do estudante frente a um objeto (tipo do AgLC), buscando a adaptação do sistema (FARACO *et al*., 2004).

Considerando que o LeCo-EAD adota três estratégias de ensino diferentes representadas pelos AgLCs do tipo *trouble maker*, aprendiz e colaborador, a adaptação do sistema acontece inicialmente quando a escala de atitudes sugere qual agente é mais adequado ao perfil do estudante (FARACO *et al*., 2004). Após o sistema sugerir o AgLC mais adequado, o aluno ainda pode trocar este tipo de agente, caso não esteja satisfazendo suas necessidades. Além desse tipo de adaptação, o protótipo faz uso das informações mantidas no modelo do aluno para conduzir o estudante nos conceitos que compõem o mapa conceitual do curso à distância. O modelo do aluno apresenta uma abordagem interessante porque, além de registrar o *tracing* do aluno no curso, adota um mecanismo baseado em fatores de confiança de sistemas especialistas para avaliar o desempenho do estudante.

3.6 Considerações Finais

Neste capítulo procurou-se relatar o uso de STIs na educação como uma ferramenta no processo de ensino e aprendizagem, focando os sistemas companheiro de aprendizagem como um companheiro que interage com o estudante na resolução de problemas.

O simples fato de utilizar um computador não garante a aprendizagem. A utilização do computador de maneira inteligente não é um atributo inerente ao mesmo, mas está vinculada à maneira como se concebe a tarefa na qual ele será utilizado. É preciso definir com clareza e competência o que é preciso ensinar para capacitar o aluno, dando-lhe condições para aprofundar seus conhecimentos e desenvolver importantes conceitos para que ele consiga lidar com a sua realidade.

Considerando que o modelo a ser criado teve como referência a prática pedagógica do sócio-interacionista, buscou-se nas idéias de Vygotsky um modelo pedagógico que se ajustasse à base teórica do mesmo.

Este trabalho propõe a criação de um ambiente que utilize um CA, que age como um companheiro de aprendizagem do aluno, podendo atuar para auxiliar o estudante na resolução de problemas. O ensino de algoritmos requer o desenvolvimento de um conjunto de habilidades no estudante. Portanto, o processo de construção da solução é tão importante quanto a própria solução do aluno.

Um ambiente de ensino e aprendizagem de algoritmos deve permitir ao estudante entender o porque errou, onde errou e as implicações deste erro, por isso o capítulo seguinte descreve a técnica da modelagem baseada em restrições (MBR), que será utilizada na construção do modelo proposto nessa dissertação.

4. MODELAGEM BASEADA EM RESTRIÇÕES

4.1 Introdução

A abordagem Modelagem Baseada em Restrições (MBR) (OHLSSON, 1994) tem como fundamento uma teoria de aprendizagem que se baseia na identificação dos erros. Esta teoria é relevante para os estudantes que não detêm o conhecimento de um determinado assunto e, portanto, não são capazes de detectar os próprios erros.

Esta modelagem representa o conhecimento sobre um domínio através de um conjunto de restrições de estados para a solução correta de um problema deste domínio (MITROVIC & OHLSSON, 1999). As restrições são divididas em todas as possibilidades de soluções corretas e incorretas que o estudante possa realizar. Cada restrição representa um conceito declarativo que deve ser aprendido e analisado pelo estudante (MARTIN, 2003).

MBR é um método para diagnosticar as soluções dos estudantes. A abordagem identifica erros que são extremamente importantes para os estudantes com ausência do conhecimento naquele domínio, visto que eles não são capazes de detectar os próprios erros. O modelo de estudante irá registrar todas as ações do estudante diagnosticando onde ocorreu violação ou não da restrição.

4.2 Aprendizagem por Erros

A MBR propõe que um domínio seja representado como restrição sobre soluções corretas de problemas nesse domínio. Este é representado através dos efeitos (estados da resolução do problema) que as ações do aluno podem gerar, fazendo a suposição de que é possível identificar um estado do problema que viola conceitos fundamentais do domínio. Assim, o domínio é representado pela descrição de estados nos quais uma solução pode recair e testa se essas soluções são consistentes com o problema que está sendo resolvido (BARROS, 2000).

As restrições propostas por OHLSSON (1992) são constituídas por um par ordenado (Cr, Cs), no qual Cr é a condição de relevância e Cs a condição de satisfação. A condição de relevância é usada para identificar um conjunto de estados de problema

para o qual determinada restrição se aplica e para o qual um possível erro pode ocorrer (ou seja, a restrição pode ser violada). A condição de satisfação é usada para identificar um subconjunto desses estados relevantes nas quais a restrição é satisfeita, ou seja, a condição que indica o acerto do aluno com relação à restrição. Se Cr é satisfeita em um estado do problema, para que esse estado esteja correto é preciso que Cs também seja satisfeita (ou então alguma coisa está errada). A figura 9 mostra um exemplo de restrição dentro do domínio de algoritmos, sendo que este tipo de modelagem se aplica também a vários domínios tais como Matemática, Banco de Dados, entre outros. Por exemplo, no caso de se fazer a leitura de uma variável num algoritmo:

Cr: Se o objetivo é fazer a leitura de uma variável então Cs: esta variável deve estar declarada como um inteiro, real, *string* ou lógica (senão ocorre um erro)

Figura 9 – Exemplo de uma restrição
Fonte: autor

No exemplo da figura 9 a restrição é somente relevante quando Cr for satisfeito, ou seja, se a instrução que estiver sendo examinada se referir à leitura de uma variável. Nesse caso a variável deve estar declarada como inteiro, real, *string* ou lógica; caso contrário, a restrição será violada.

Se, ao analisar a solução do aluno, Cr for satisfeita, mas Cs não, dizemos que a restrição (Cr, Cs) foi violada, indicando assim que o aluno cometeu um erro. Segundo MARTIN & MITROVIC (2002), em MBR não estamos interessados no que o estudante fez ao longo do processo de solução de um problema, mas em que estado ele está atualmente.

Neste sentido, um Sistema Tutor Inteligente (STI) deve ser capaz de observar determinadas ações do aluno (previamente especificadas), ou seja, os efeitos dessas ações, sendo que a análise dessas observações pode ser usada para a construção do modelo do estudante. No caso de uma solução incorreta, um STI deve ser capaz de diagnosticar os erros cometidos pelo aluno durante o processo de resolução e, com base nesse diagnóstico, construir um modelo mais informado do aluno.

A abordagem MBR não permite a identificação do raciocínio abstrato do aluno, devido ao fato de que as condições da restrição somente podem ser observadas

diretamente na solução do aluno (BARROS, 2000). Contanto que o estudante nunca alcance um estado que será incorreto, ele está livre para realizar qualquer ação que ele gostaria. Portanto, o modelo do domínio é uma coleção de estados de descrições na forma de:

SE <condição de relevância> é verdade para a solução do estudante, ENTÃO <condição de satisfação> deve ser também verdade, senão a solução está errada.

Figura 10 – Representação de uma restrição
Fonte: MARTIN, 2002

Em outras palavras, a condição de relevância de cada restrição é usada para testar se a solução do estudante está em um estado pedagogicamente significante. Caso a resposta for sim, a condição de satisfação é checada. Se esta condição falhou, o estudante realizou um erro um *feedback* apropriado é passado a ele (MARTIN, 2002).

As restrições que modelam o conhecimento do domínio servem para classificar estados de problemas, ou seja, devem identificar que o aluno deve receber uma determinada ação de instrução através do sistema, sugerindo novas idéias que estejam relacionadas àquela determinada restrição (MARTIN & MITROVIC, 2000). Essa abordagem pode ser usada tanto para modelar o conhecimento do domínio como para modelar o conhecimento do aluno neste domínio, através das restrições violadas.

As restrições modeladas no sistema não são exaustivas, ou seja, não há como requerer todas as possibilidades de interação. Para o estudo de domínio dessa dissertação, algoritmo, um grande número de possibilidades pode ser gerado.

4.3 STIs que utilizam MBR

A seguir serão detalhados alguns STIs que utilizam a MBR para modelar o conhecimento dos estudantes nos mais diversos domínios.

4.3.1 CAPIT

Segundo MAYO (2000), o CAPIT (*Capitalisation and Punctuation Intelligent Tutor*), projetado para crianças entre 10 e 11 anos de idade, ensina o uso de letra

maiúscula nas sentenças e das regras de pontuação em um texto. O sistema representa o domínio como um conjunto de restrições especificando os corretos padrões de uso de letra maiúscula e pontuação. O *feedback* para a criança é dado a partir da violação das restrições modeladas no sistema.

A arquitetura do CAPIT pode ser visualizada conforme figura 11.

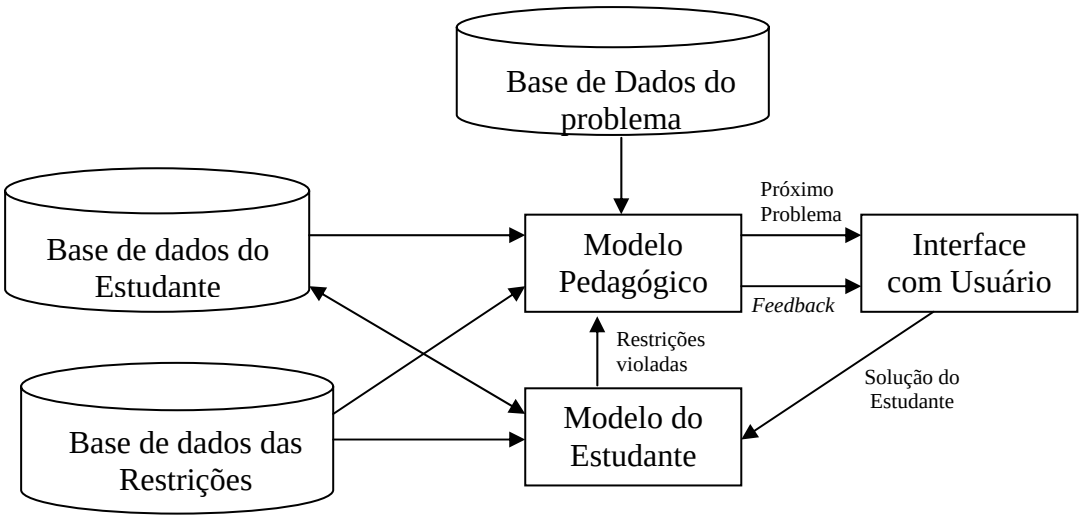

Figura 11 - Arquitetura do Sistema
Fonte: adaptado de MAYO (2000)

O módulo pedagógico resolve duas tarefas chaves de decisão: quando a criança seleciona o próximo problema através da interface e quando uma simples mensagem de erro é visualizada no momento em que a criança submete uma solução incorreta. Quando a criança submete uma solução o modelo do estudante determina, primeiramente, qual restrição é relevante para a solução atual, e depois, quais dessas restrições são satisfeitas. A violação das restrições é então passada para o módulo pedagógico, onde uma mensagem de erro é selecionada (MAYO, 2000).

4.3.2 LeCS

LeCS (*Learning from Case Studies*) é descrito como um sistema colaborativo para o ensino com estudos de caso. Neste sistema, o suporte a este método pedagógico no contexto do ensino à distância envolve habilitar a colaboração entre um grupo de estudantes geograficamente dispersos. O sistema permite a colaboração entre os

estudantes, como também colabora com estes no processo de solução do estudo de caso, que é realizado através da *World Wide Web* (*www*) (ROSATELLI, 1999).

A arquitetura do LeCS é baseada em agentes e suporta a comunicação entre eles através de um facilitador. Esta arquitetura é utilizada como uma base para implementar um ambiente colaborativo que inclui um conjunto de ferramentas que dá apoio à atividade de grupo no ensino baseado em casos (ROSATELLI *et al*., 2000).

A figura 12 mostra a arquitetura e as estruturas de comunicação utilizadas nesse sistema, que estabelecem que os agentes nunca trocam mensagens diretamente, mas através do facilitador. Segundo ROSATELLI (1999), o facilitador é um programa especial (implementado como um agente) que mantém a informação sobre cada agente no sistema. Ele é responsável pelo roteamento das mensagens. As mensagens trocadas entre os agentes usam o formato KQML (*Knowledge Query and Manipulation Language*).

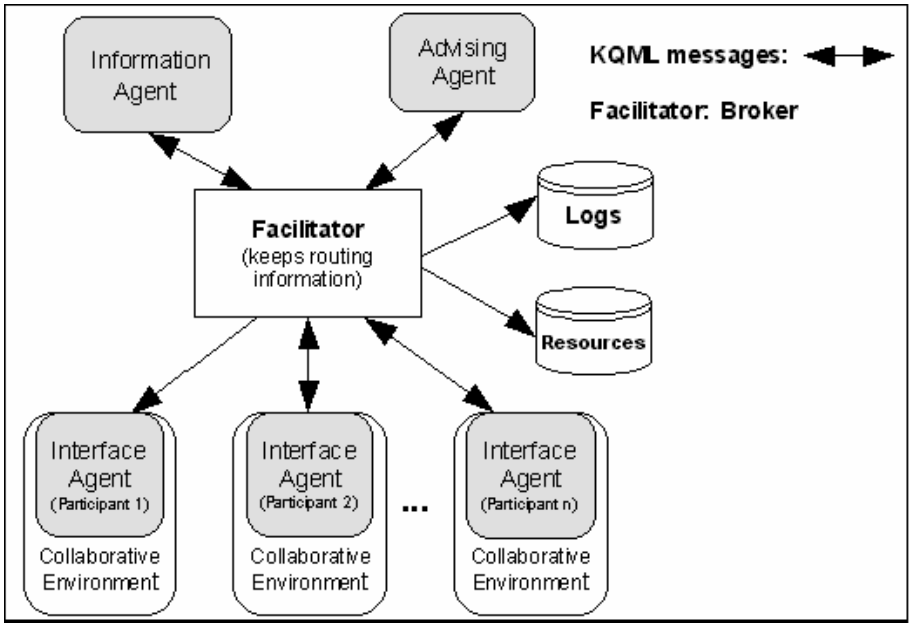

Figura 12 - Arquitetura baseada em agentes do LeCS
Fonte: ROSATELLI, *et al.* (2002)

No LeCS, a MBR foi usada para tornar o sistema apto a identificar um mal entendido a respeito do estudo de caso (ROSATELLI, 1999). De acordo com ROSATELLI (1999), a Tabela 1 apresenta exemplos de restrições que foram formuladas numa análise do estudo de caso de acordo com que tipos de mal entendidos

que ele poderia originar. Geralmente, isso é modelado pelo professor que trabalha com estudos de caso.

Tabela 1. Um exemplo de uma restrição de estado (<Cr, Cs>) de um estudo de caso no domínio de Engenharia de Produção.

Se a “companhia oferta” (Cr) então ela terá “numerosos produtos a oferecer” (Cs) (ou senão haverá um erro)
Se a “companhia produz muito” (Cr) então ela terá “destaque das demais” (Cs) (ou senão haverá um erro)
Se ela é “proposta de Mr. Barth” (Cr) então ela terá “aumento na média de produção” (Cs) (ou senão haverá um erro)

Fonte: ROSATELLI (1999)

4.3.3 SQL-Tutor

Um STI que utiliza MBR e foi desenvolvido no domínio de base de dados é o SQL-Tutor (MITROVIC, 1998; MITROVIC & OHLSSON, 1999). O SQL-Tutor ensina a linguagem de banco de dados SQL (*Structured Query Language*). De acordo com MARTIN & MITROVIC (2002) este sistema foi desenvolvido ao longo de quatro anos e na sua versão atual conta com cerca de 500 restrições. No SQL-TUTOR a MBR é usada no modelo do estudante para diagnosticar as respostas do mesmo.

Por exemplo, quando um estudante faz um erro de declaração em SQL, o sistema gera a correta solução ao lado da solução do estudante no ponto do erro. Uma lista de *feedback* é gerada do SQL-Tutor para explicar cada um dos erros na solução do estudante.

A arquitetura deste sistema é ilustrada na figura 13, incluindo os componentes básicos do sistema: a interface com o usuário, o módulo pedagógico e o módulo do estudante.

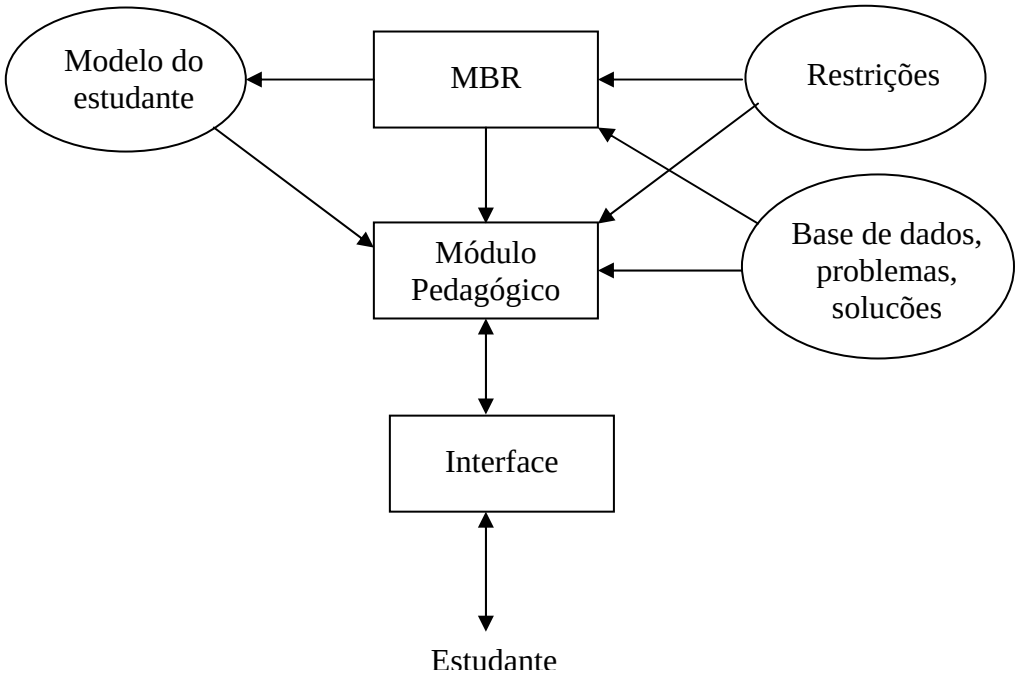

Figura 13 – Arquitetura do SQL-Tutor
Fonte: MITROVIC (1998)

O módulo pedagógico observa todas as ações geradas pelo estudante na interface, e reage a ele apropriadamente (MITROVIC, 1998). No início da interação é selecionado um problema para o estudante trabalhar. MITROVIC (1998) diz que, quando o estudante entra com a solução para o problema atual, o módulo pedagógico envia a solução para o modelo do estudante, o qual checa se a solução está correta ou incorreta, e atualiza o modelo do estudante. O módulo pedagógico gera uma mensagem de retorno apropriada. Quando o atual problema está resolvido, ou o estudante requer um novo problema para trabalhar, o módulo pedagógico seleciona um novo problema de acordo com o modelo do estudante.

O modelo do estudante registra o histórico do uso de cada restrição. Este histórico contém informações sobre como a restrição foi relevante para a solução ideal no problema praticado pelo estudante, ou seja, como foi satisfeita ou violada. Este registro é utilizado pelo módulo pedagógico (MITROVIC & OHLSSON, 1999).

O domínio do conhecimento é representado na forma de restrições. O sistema também contém um conjunto de restrições para cada base de dados especificada e a solução ideal das restrições (MITROVIC, 1998). As soluções são necessárias porque o SQL-Tutor não tem um módulo de domínio e não é capaz de resolver os problemas.

4.4 Considerações Finais

Este capítulo apresentou a MBR, que é uma abordagem baseada na teoria de aprendizagem por erros. A MBR é eficaz para a construção de modelos de domínio e de estudante nos STIs.

Foram apresentados os STIs desenvolvidos utilizando a MBR em algum dos seus módulos. Pode-se verificar que a utilização dos STIs não se restringe em apenas um domínio específico, mas sim nos mais variados podendo ter sistemas que atendam crianças na idade escolar, STIs para auxílio de uma determinada disciplina de cursos de graduação, estudos de casos em diversas áreas, todos eles tendo em algum módulo a utilização da MBR. Vale ressaltar que não foi verificado nenhum STI com MBR num domínio específico para o ensino e aprendizagem de algoritmos.

No estudo dos diversos ambientes pesquisados ao longo dessa dissertação pode-se observar aspectos relevantes que embasaram a construção do modelo apresentado no próximo capítulo, que enfoca a modelagem do AlgoLC, um sistema que inclui um companheiro de aprendizagem que utiliza a modelagem baseada em restrições. Vários foram os sistemas pesquisados sobre o domínio de estudo, o algoritmo, sendo que não foi observado um SCA para esse domínio e nem uma técnica de representação do conhecimento e raciocínio aos SCAs através da Modelagem Baseada em Restrições (MBR). Os estudos dos mais diversos ambientes contribuíram para a criação da arquitetura do AlgoLC, principalmente a arquitetura do SQL-Tutor, que é a que está mais próxima desta pesquisa.

5. SISTEMA AlgoLC

5.1 Introdução

Este capítulo apresenta um Sistema Companheiro de Aprendizagem (SCA) para o ensino de algoritmos. O Companheiro de Aprendizagem (CA) apresentado utiliza a Modelagem Baseada em Restrições (MBR) e traz uma concepção do processo de ensino e aprendizagem em que o aluno aprende através dos seus erros. No caso, o aluno tem um auxílio individualizado de um CA, que envia mensagens estimulando o aluno a verificar seus erros e corrigi-los, ao contrário de, por exemplo, propor a melhor solução. Como já citado no capítulo 3, o CA neste modelo terá um papel colaborativo.

O presente capítulo detalha a modelagem do sistema, descrevendo sua arquitetura e módulos principais, a modelagem das restrições no domínio de algoritmos e o funcionamento do sistema. Em seguida é apresentado o protótipo AlgoLC que foi desenvolvido segundo a abordagem acima descrita. O AlgoLC caracteriza-se como um SCA que utiliza a MBR para dar suporte ao ensino e aprendizagem no domínio de algoritmos.

5.2 Arquitetura

A Figura 14 apresenta a arquitetura do SCA baseado em restrições para o ensino de algoritmos, denominado AlgoLC. O AlgoLC é capaz de fazer análises sobre o que o aluno sabe e como ele está progredindo na construção de um algoritmo.

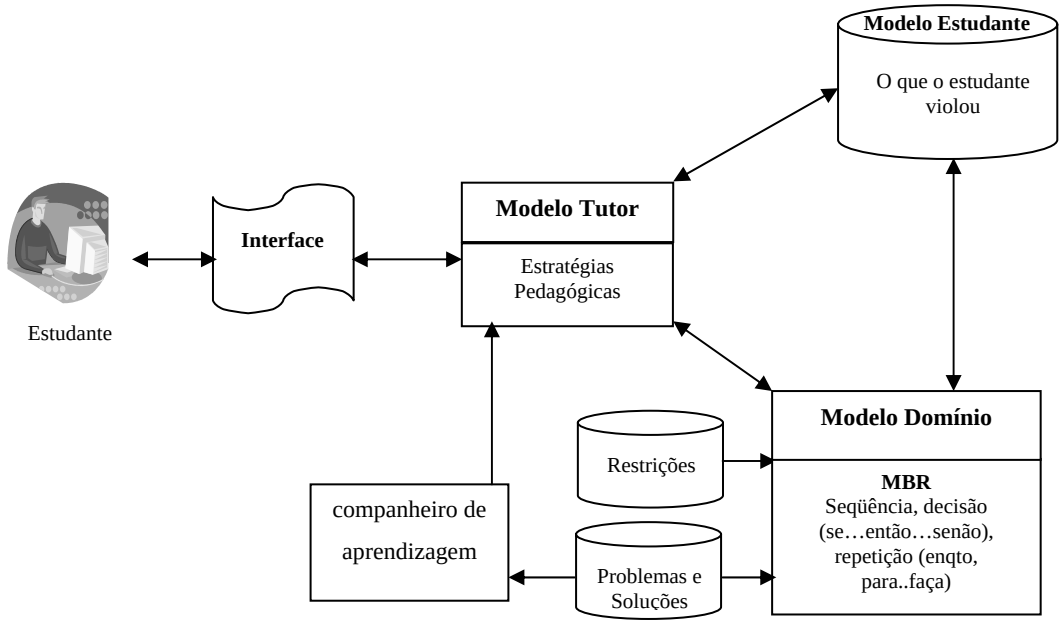

Figura 14 - Arquitetura do AlgoLC
Fonte: autor

O modelo de domínio contém as informações corretas que o aluno deverá aprender, ou seja, é a base de conhecimento do sistema. Neste caso, este modelo diz respeito aos exercícios e às restrições que não devem ser violadas em relação à estrutura de um algoritmo, quando o aluno está construindo o mesmo. O sistema contém um conjunto de restrições e a solução ideal das restrições. As restrições dizem respeito aos procedimentos necessários para a resolução dos problemas sobre a matéria dentro do sistema. Nessa etapa as atividades, operações e ações utilizadas na solução dos problemas são devidamente detalhadas por um especialista humano. Este conhecimento será utilizado quando o CA identifica uma situação em que o estudante realizou uma operação não correta ou quando o mesmo não consegue evoluir durante a solução do problema. O tutor também poderá utilizar esse conhecimento quando identificar que o estudante e o CA não evoluíram no processo de interação.

O modelo do estudante consiste nas características gerais do estudante (nome, histórico, etc.) e também se refere ao conhecimento correto ou incorreto do estudante, ou seja, quais restrições o estudante violou ou não. Esse registro é utilizado pelo modelo do tutor para determinar algumas estatísticas importantes, como por exemplo, onde o estudante mais errou e qual restrição foi mais violada. Os problemas apresentados são escolhidos pelo sistema baseados no modelo do estudante. As restrições que são

relevantes para cada questão são comparadas no modelo do estudante. Nesse modelo o conhecimento do estudante é estruturado como uma sobreposição (*overlay*) do modelo de domínio. Para cada problema a ser resolvido no sistema, um modelo individual de sobreposição armazena o nível de conhecimento do estudante sobre aquele problema. Esse nível de conhecimento pode representar se o estudante sabe ou não sabe, uma medida qualitativa do que ele sabe, ou uma medida quantitativa do quanto o estudante conhece o assunto. Cada aluno possui associado seu modelo do estudante com informações referentes ao desempenho destes no desenvolvimento dos problemas no sistema.

Os critérios utilizados para armazenar informações referentes aos resultados obtidos no desenvolvimento de problemas pelo aluno:

- número de vezes em que o CA interfere no desenvolvimento dos problemas;
- número de acertos no desenvolvimento dos problemas propostos;
- número vezes em que ocorreram violações das restrições na resolução dos problemas;
- quais as restrições que foram mais violadas pelo aluno.

O modelo do tutor possui as estratégias pedagógicas que tratam do conhecimento sobre o ensino daquele domínio. Dentro de um SCA o tutor tem o papel de coordenar as atividades de aprendizagem: apresentação dos exercícios propostos, o controle de pré-requisitos e a interação entre estudante e CA.

Quando o estudante submete uma solução, esta é passada ao modelo do estudante. O modelo do estudante determina, primeiramente, quais restrições são relevantes para a solução atual e, após, quais restrições são satisfeitas. Ou seja, o que foi violado e o que não foi. O CA intervém automaticamente, através de mensagens ao estudante, de acordo com as restrições que foram violadas.

O CA conduz o aluno aos desafios propostos na resolução dos problemas, observando e verificando todas as ações geradas pelo estudante na interface e reagindo a ele quando uma restrição foi violada. O CA deste protótipo é autônomo e colaborativo, sempre interagindo com o estudante quando for apropriado, sem ser solicitado pelo estudante. O modelo do tutor interage com o CA, escolhendo as mensagens de *feedback* nos diferentes casos através da violação do MBR.

Para este protótipo, o CA reage durante a construção do algoritmo, retornando uma mensagem apropriada quando encontrou uma violação, como pode ser visualizada na figura 15 através de um exemplo.

```
1  Algoritmo Teste1
2  Var
3      X : inteiro
4  Início
5      X
```

Figura 15 – Exemplo de um algoritmo 1
Fonte: autor

Se a linha não estiver violando a restrição, nenhuma intervenção é feita junto ao estudante e este pode continuar seu exercício. Caso uma linha tenha a restrição violada uma mensagem de erro “Erro!” é enviada ao estudante e o CA busca no banco de dados as possíveis soluções para a dúvida do estudante naquela linha de restrição violada. O estudante só consegue ir adiante após o problema solucionado. Para o exemplo na linha 5 da figura 15, os tipos de mensagens de auxílio enviadas ao estudante podem ser observadas na figura 16.

```
Pode ser realizada uma atribuição
Pode ser realizada uma leitura com o comando leia
Pode ser realizada uma escrita com o comando escreva
```

Figura 16 – Exemplo de mensagens disparadas pelo CA para um erro em uma única linha
Fonte: autor

Observando um outro exemplo, como o apresentado na figura 17, na linha 3 ocorre a violação de várias restrições na qual o CA aciona e gera ao estudante o *feedback* apropriado, através de mensagens, para que ele encontre a melhor alternativa para o seu algoritmo.

```
1  Algoritmo Teste2
2  Início
3      X = B
```

Figura 17 – Exemplo de um algoritmo 2
Fonte: autor

Para o exemplo da figura 17 as mensagens selecionadas pela violação das restrições podem ser observadas na figura 18. Vale ressaltar que nem todas as mensagens são apropriadas à solução do algoritmo, pois depende da escolha do usuário. Ou seja, se o estudante acredita que B é uma variável, a mesma deve ser declarada corretamente, portanto a última mensagem não será válida para solução do estudante. Por outro lado, se o estudante acredita que B é uma informação, portanto deve estar entre aspas, então B não será declarada.

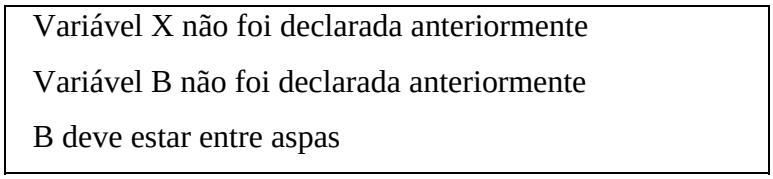

Figura 18 – Exemplo de mensagens disparadas pelo CA para um erro em uma única linha
Fonte: autor

Esse mecanismo de comunicação que o CA solicita ao estudante através da sua interferência a cada linha, é importante para o bom andamento da melhor solução do estudante naquele exercício.

A interface do usuário é o meio pelo qual o aluno se comunica com o sistema. É função da interface apresentar o material de instrução ao aluno e capturar informações do aluno, a fim de monitorar o seu progresso e o seu comportamento para manter o modelo do aluno atualizado.

5.3 Modelagem das Restrições

O modelo de domínio do AlgoLC consiste de um conjunto de 6 (seis) problemas e 33 (trinta e três) restrições (Apêndice A) que foram modeladas a partir da identificação de problemas usuais e freqüentes no ensino de algoritmos. As restrições criadas para o protótipo tendem a evoluir. O modelo do estudante é um *overlay* do modelo de domínio que inclui o número de vezes que cada restrição foi utilizada e quantas vezes ela foi violada.

As restrições modeladas foram agrupadas em quatro níveis que dizem respeito ao nível de dificuldade de um exercício. O aluno tem a liberdade de escolher por qual nível iniciar seus exercícios:

1: declaração de variáveis: se o aluno pretende desenvolver algoritmos que envolvem somente a declaração de variáveis.

2: estruturas seqüenciais: se o aluno quer desenvolver algoritmos que envolvem tanto declaração de variáveis, quanto estruturas seqüenciais.

3: estruturas de decisão: se o aluno quer desenvolver algoritmos que envolvem declaração de variáveis, estruturas seqüenciais e estruturas de decisão.

4: estruturas de repetição: se o aluno quer desenvolver algoritmos que envolvem declaração de variáveis, estruturas seqüenciais, estruturas de decisão e estruturas de repetição.

5.4 Funcionamento

Para iniciar a interação com o sistema o aluno deve digitar seu nome e senha, e em seguida escolher o grupo de exercícios que deseja trabalhar. Por exemplo, exercícios envolvendo apenas estruturas seqüenciais, utilizando comandos simples de leitura, escrita e atribuição. Após a escolha, o aluno deve iniciar escrevendo o seu algoritmo a partir da estrutura da figura 19, que é apresentada pelo sistema.

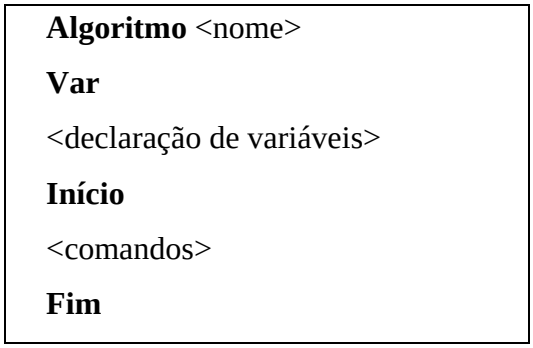

Figura 19 – Representação de um algoritmo no AlgoLC
Fonte: autor

Quando o aluno tem dúvida ou quando ele deseja fazer uma verificação ele aciona o CA, que faz a verificação linha a linha do algoritmo já escrito e interage com o aluno,

não respondendo a sua dúvida, mas auxiliando-o na melhor resposta. O CA guarda um histórico das vezes em que o aluno o solicitou para auxílio e ao final pode informar ao aluno os assuntos em que o mesmo precisa se exercitar mais. Assim, o sistema armazena um histórico do aluno que está disponível para o professor, que tem a possibilidade de verificar os erros mais freqüentes do aluno e posteriormente, na sala de aula, dar mais ênfase em determinado assunto.

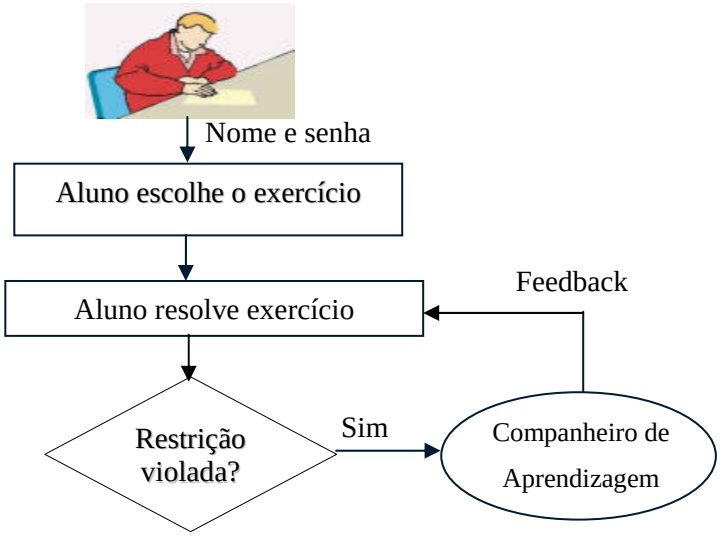

Figura 20 – Fluxograma do funcionamento do AlgoLC
Fonte: autor

5.5 Interfaces

A tela de abertura do AlgoLC pode ser visualizada na figura 21, onde o estudante deve entrar com seu nome de usuário e senha.

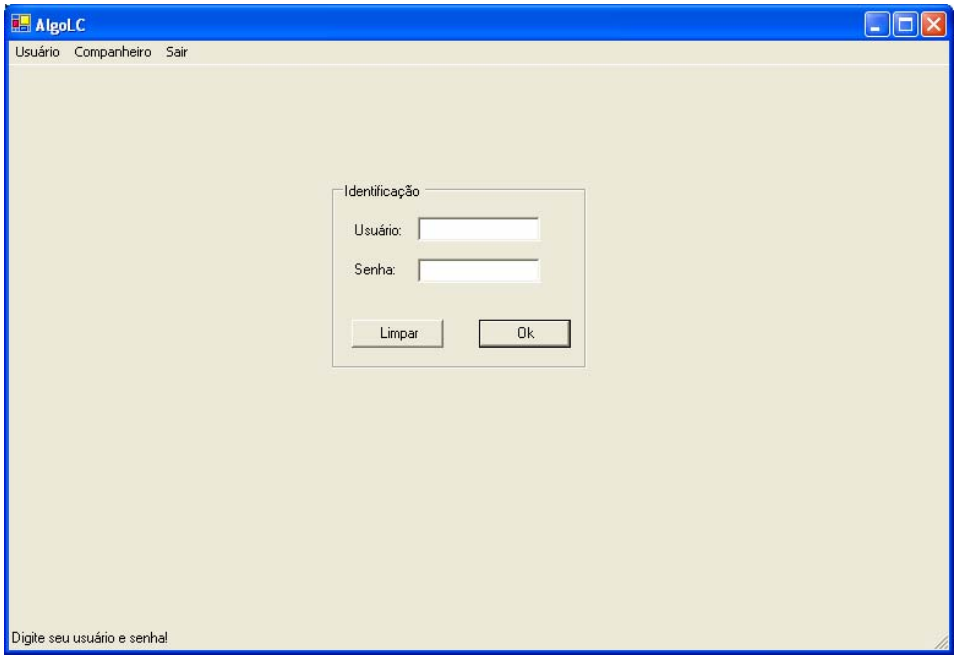

Figura 21 – Tela inicial do AlgoLC
Fonte: autor

Na barra de menu tem-se as seguintes opções: usuário, companheiro e sair. Na opção usuário o aluno pode trocar de usuário ou visualizar todos os exercícios. Na opção companheiro, têm-se as opções analisar algoritmo e a opção relatórios. A opção analisar algoritmo faz a verificação linha a linha da solução proposta pelo estudante. A opção relatórios que gera um relatório do aluno, que permite verificar quais restrições e quantas vezes o aluno a violou durante o exercício, e também um relatório geral que mostra a quantidade de vezes que cada restrição foi violada por aquele aluno na solução dos exercícios já resolvidos.

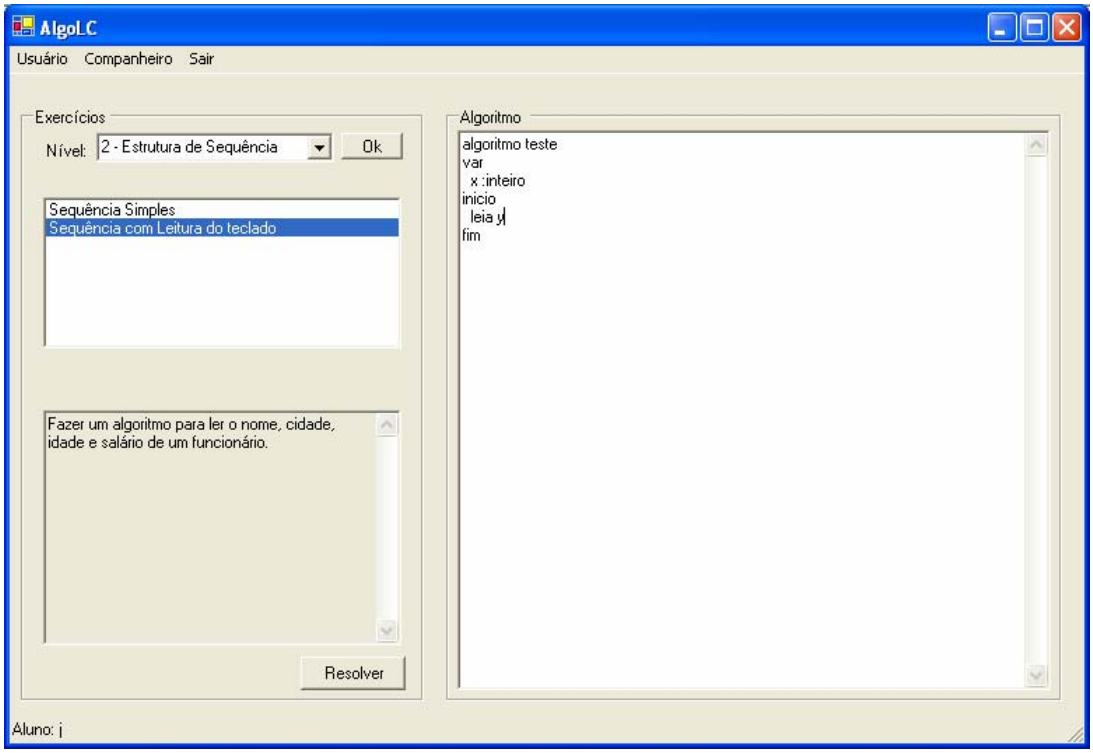

Figura 22 – Tela do AlgoLC
Fonte: autor

Após entrar no sistema o estudante visualiza a tela da figura 22. Essa tela é dividida em duas áreas: exercícios e algoritmo. Na área de exercícios o aluno faz a escolha do exercício desejado. Na área denominada algoritmo o estudante digita a solução do problema proposto.

O estudante tem a opção de listar todos os exercícios clicando no menu usuário e em seguida no item exercícios, ou escolher o nível do exercício que ele deseja resolver. Após a escolha do nível, o estudante pode optar pelo o tipo de exercício que deseja resolver. Por exemplo, na figura 22 o estudante fez a escolha do nível 2 do exercício: estrutura de seqüência. Em seguida, surge uma sub-lista de tipos de exercícios dentro do nível escolhido. O estudante faz sua escolha e o enunciado do algoritmo é apresentado. Para iniciar sua solução o estudante deve clicar no botão resolver. A solução deve ser digitada na área à direita da figura 22, denominada algoritmo.

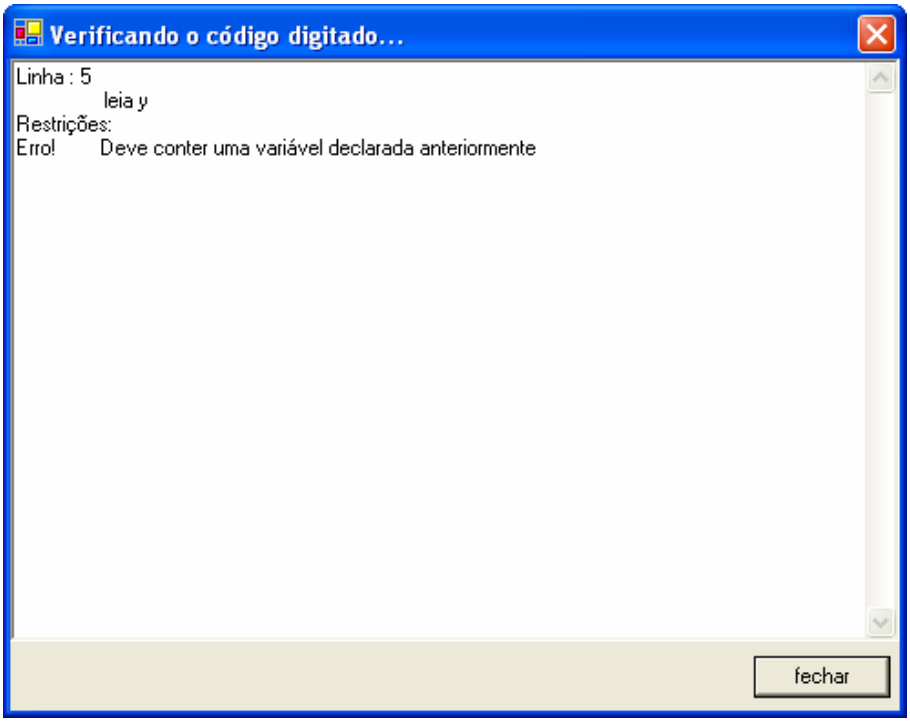

Figura 23 – Tela de verificação da solução proposta com violação
Fonte: autor

Após a digitação de cada linha do algoritmo como solução proposta (utilizando a tecla <enter> para pular de linha), o CA é acionado automaticamente caso alguma restrição naquela linha tenha sido violada. Caso ocorra violação, surge então a tela representada na figura 23 de verificação do código do algoritmo naquela linha em que o estudante está resolvendo o problema. Essa análise é feita com base nas restrições modeladas (ver apêndice A) onde se observa, para a linha do algoritmo escrito pelo estudante, qual condição é relevante e, dentre as condições que são relevantes naquele caso, quais são satisfeitas. As mensagens são associadas às restrições utilizadas e se as mesmas foram ou não violadas. Observa-se na figura 23 que na linha cinco ocorreu um erro. O CA não dá a resposta imediata ao aluno e sim possibilidades de soluções ao erro do estudante.

Ou seja, a partir da verificação de quais condições são relevantes para aquela linha e se as respectivas condições de satisfação são relevantes, estas são apresentadas na forma de mensagens ao estudante. Sendo assim, as mensagens de *feedback* são relacionadas diretamente à cada restrição incluída no modelo do domínio. E, quando uma restrição é violada, o CA envia mensagens apropriadas ao aluno, descrevendo as possíveis violações daquela restrição.

Após a correta solução do problema proposto, a tela da figura 24 é visualizada.

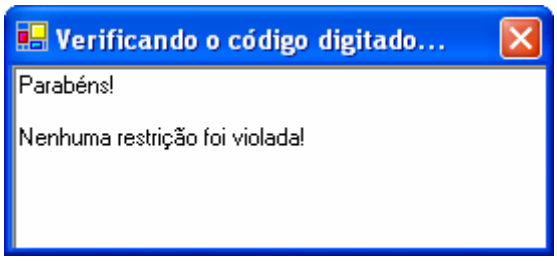

Figura 24 – Tela de verificação da solução proposta sem violação
Fonte: autor

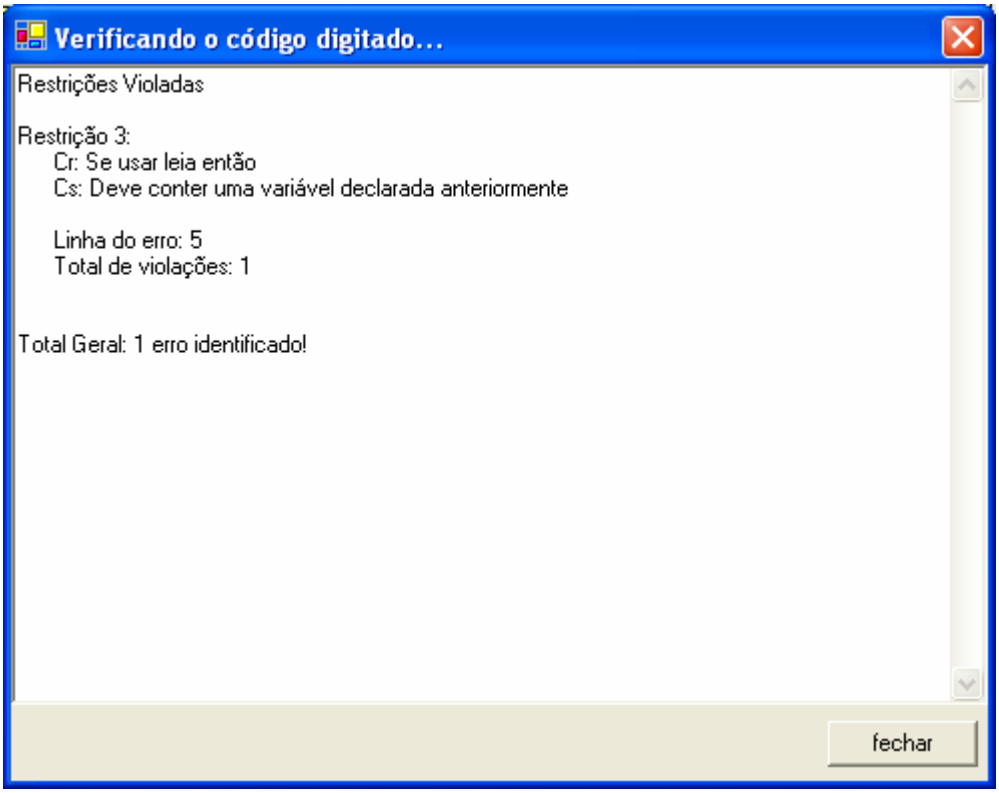

Figura 25 – Tela de relatório por aluno
Fonte: autor

A figura 25 representa a tela de relatório por aluno, onde são apresentadas todas as estatísticas referentes a violações das restrições do exercício que está sendo realizado pelo aluno.

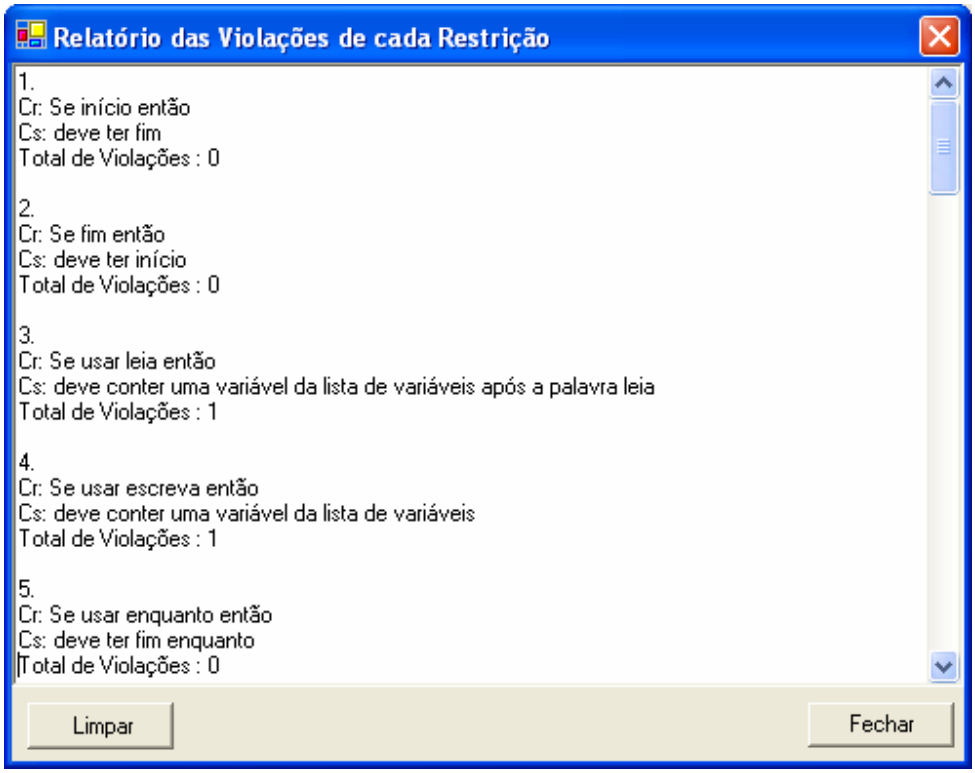

Figura 26 – Tela de relatório das restrições
Fonte: autor

A figura da tela 26 representa o relatório das restrições, onde são apresentadas todas as estatísticas referentes à violação de cada uma das restrições de todos os exercícios que foram realizados pelo aluno até o momento.

5.6 Testes e Resultados

O protótipo do AlgoLC foi testado com 15 (quinze) alunos do curso de Ciência da Computação da Universidade do Sul do Estado de Santa Catarina, que cursam a disciplina de algoritmo (1ª fase). Dos quinze alunos que realizaram o teste, 9 (nove) eram repetentes desta disciplina. A experiência dos alunos com a disciplina de algoritmo é pouca, visto que apenas um dos alunos já tinha realizado um curso de lógica de programação. Os outros alunos tinham tido o contato com este domínio apenas nesta disciplina.

Os exercícios utilizados no teste se restringiram aos de declaração de variáveis e estruturas seqüenciais (ver apêndice B), totalizando 6 (seis) exercícios. Os testes foram realizados durante 3 (três) horas. Primeiramente, os alunos instalaram o sistema AlgoLC

em seu computador pessoal, o que levou em torno de 30 (trinta) minutos. Logo em seguida houve uma explicação do sistema e os alunos começaram a realizar os exercícios que estavam disponíveis, seguindo a ordem: declaração de variáveis e estruturas seqüenciais. Após a resolução de cada exercício, era solicitado ao aluno enviar a tela de relatório por aluno ao professor. Ao final da resolução de todos os exercícios, o aluno gerava o relatório das restrições, onde se visualizava as estatísticas referentes à violação de cada restrição utilizada em todos os exercícios. Das 35 (trinta e cinco) restrições criadas, para esse teste foram utilizadas 24 (vinte e quatro) restrições. As restrições de 5 a 15 não foram utilizadas durante o teste.

Tabela 2. Demonstração das violações das restrições pelos alunos durante o teste.

Aluno	**Restrições Utilizadas e Violadas**																	
	3	**4**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**30**	**31**	**32**	**33**	**34**	**35**
1																		
2										X		X						X
3		X		X		X			X	X			X			X		
4																		
5	X								X								X	
6										X	X			X				
7				XX		X	X			X					X			
8	X	X	X										X					
9			X		X					X								
10	X									X						X		
11										X								
12		X						X		X				X	X			X
13					X					X								
14		X																
15										X							X	

Fonte: autor

As restrições 1, 2, 26, 27, 28 e 29 (apêndice A) não foram violadas por nenhum aluno, por isso não constam na tabela. Apenas um aluno (aluno 7) violou duas vezes a mesma restrição (restrição 17), que se refere à atribuição de variáveis do tipo inteiro recebendo valores reais.

Dos quinze alunos que utilizaram o AlgoLC, somente dois (alunos 1 e 4) conseguiram resolver todos os exercícios sem violar nenhuma restrição. A restrição 23 (apêndice A) foi violada 10 (dez) vezes. Essa restrição diz respeito à atribuição de informações à variáveis do tipo *string*, onde a informação a ser atribuída à variável deve estar entre aspas (“ ”). Um tipo de erro freqüente observado pelo professor nos testes da ferramenta AlgoLC, é que os alunos criavam uma variável do tipo *string* corretamente, mas quando atribuíam um valor à essa variável, esse valor poderia ser uma informação ou outra variável do tipo *string*, pelo fato da informação não estar entre aspas. Por exemplo, nome=“Ana”, é diferente de nome=Ana, pelo fato de Ana com aspas ser uma informação e Ana sem aspas poder ser uma outra variável.

Durante os testes, todos os alunos, após o acionamento do CA com o *feedback* apropriado, conseguiram resolver os dois grupos de exercícios propostos. Com a ferramenta computacional AlgoLC os alunos solucionavam os problemas através da interação do CA, possibilitando um aprendizado adequado ao erro cometido e, com isso, num próximo grupo de problemas, os alunos errariam menos. Esse resultado também fornece uma indicação que o sistema apresentado permite que o professor da disciplina observe com relativa precisão as dúvidas surgidas e os erros cometidos com maior freqüência pelos alunos na construção de algoritmos e em sala de aula procurar sanar melhor essas dúvidas e erros.

Assim, utilizando o AlgoLC como suporte ao ensino e aprendizagem de algoritmos em sala de aula, o professor pode dispor de mais uma ferramenta para saber exatamente o ponto do maior grau de dificuldade da maioria dos estudantes e de cada estudante de uma determinada turma.

6. CONCLUSÃO E RECOMENDAÇÕES FUTURAS

Do ponto de vista da educação, o computador pode ser apreciado como um novo meio de dar suporte a idéias e abordagens pedagógicas já existentes. No que diz respeito à construção conjunta do conhecimento, os trabalhos de Vygotsky, que colocam a interação social como estando no centro do processo de desenvolvimento, embasam a perspectiva teórica atual de que o desenvolvimento cognitivo é mediado e construído através da interação com os outros, e os significados são negociados e estabelecidos através da interação numa vasta gama de contextos sociais, entre eles, o contexto da aprendizagem baseada em computador.

O AlgoLC atendeu os objetivos do presente trabalho, que apresentou uma ferramenta de apoio ao ensino e aprendizagem de algoritmos que utiliza um companheiro de aprendizagem colaborativo. Esta ferramenta permite um suporte adequado ao professor na identificação de dúvidas e erros dos alunos na construção de algoritmos.

Os ambientes para o ensino e aprendizagem de algoritmo e os STI que utilizaram um companheiro de aprendizagem num determinado domínio, que foram pesquisados e analisados embasaram a modelagem do sistema e implementação do protótipo AlgoLC.

O teste realizado com o protótipo, apresentou os resultados esperados, no sentido que os alunos que participaram do teste apoiaram a utilização da mesma como um suporte à disciplina de programação de computadores. Todos os alunos não encontraram dificuldades em manuseá-la. Também afirmaram a importância do companheiro de aprendizagem não ser um agente que fornece ao aluno a resposta direta da resolução do algoritmo. Muito dos alunos citaram como um ponto a favor da ferramenta, o fato do companheiro de aprendizagem ser autônomo e interferir na medida da criação do algoritmo, principalmente fornecendo possíveis soluções, ou seja, fazendo com que o aluno raciocine na escolha da opção.

O modelo proposto e o protótipo desenvolvido, que foi submetido a um teste com alunos da disciplina de algoritmos, serviram para mostrar a eficácia de uma ferramenta de apoio a aprendizagem de algoritmo com o apoio de um companheiro de aprendizagem virtual interagindo colaborativamente com o aluno.

Como recomendação futura, sugere-se que o protótipo criado sirva de base para a formalização de um sistema que possa ser utilizado como apoio ao processo de ensino e aprendizagem na disciplina de algoritmo nos diversos cursos em nível de graduação da área tecnológica. Esse sistema poderia envolver exercícios mais dirigidos à área de atuação do curso do aluno ou exercícios que representem o dia-a-dia da maioria dos alunos, como por exemplo, cálculo do salário, média das notas, etc.

Na implementação atual do sistema, as telas de relatório estão vinculadas ao usuário-estudante. Ou seja, esta não inclui um usuário-professor, onde o professor poderia acompanhar todas as etapas já realizadas pelos alunos, bem como relatórios referentes à violação de restrições por estes estudantes, exercícios que estes apresentam maior dificuldade, tempo de realização de cada exercício, etc. Estas tarefas que a ferramenta poderá adotar são propostas para dar continuidade ao ambiente AlgoLC. Outro fator observado é a importância de um sistema completo em rede com um módulo para o usuário-professor, onde este poderia ter todas as estatísticas de todos os alunos de várias turmas nos diversos cursos que tenham no seu currículo a disciplina de algoritmo.

O modelo criado nesta dissertação se restringe somente a algoritmos estruturados não fazendo nenhuma referência à orientação a objetos. A criação de um módulo que atenda os conceitos de orientação a objetos fica também, como recomendação para trabalhos futuros.

7. REFERÊNCIAS BIBLIOGRÁFICAS

AZEVEDO, B. F. T.; TAVARES, O. L. *Um Sistema Tutor Inteligente para Suporte à Aprendizagem de "Conceitos de Orientação a Objetos"*.
Disponível em: www.inf.ufes.br/~tavares/sticoo.html. Acessado em: 01/10/2003.

BARROS, L. N. e SANTOS, E. T. *Um Estudo sobre a Modelagem do Domínio de Geometria Descritiva para a Construção de um Sistema Tutor Inteligente*. Anais do XI Simpósio Brasileiro de Informática Educativa - SBIE 2000. Maceió – AL, 2000. pgs. 259-266.

BRAVO, C.; REDONDO, M. A.; ORTEGA, M.; VERDEJO, M. F. *Collaborative Discovery of Model Design*. S.A. Cerri, G. Gouardères, and F. Paraguaçu (Eds): ITS 2002, LNCS 2363. Springer-Verlag Berlin Heidelberg, pp. 671-680.

CAEIRO, C. M.; SERRA, D. R.; JORGE, J. D. *Estudo sobre Inteligência Artificial*. Acessado em 27-12-2004.
http://www.citi.pt/educacao_final/trab_final_inteligencia_artificial/raciocinio.html

CHAN, T. W. e BASKIN, A. *Studying with the prince. The computer as a learning companion*. In Proceedings of the ITS-88 Conference, Montreal, Canada, 194-200, 1988.

CHAN, T. W. e BASKIN, A. B. *Learning Companion Systems*. In: C. Frasson & G. Gauthier (Eds.), Intelligent Tutoring System: At the Crossroads of Artificial Intelligence and Education, Cap. 1, New Jersey: Ablex Publisching Corporation, 1990.

CHAN, T. W.; CHOU, C. Y.; LIN, C. J. *User Modeling in Simulatin Learning Companions*. Artificial Intelligence in Education. IOS Press, 1999.

CHAN, T. W.; CHOU, C. Y.; LIN, C. J. *Redefining the Learning Companion: the Past, Present, and Future of Educational Agents*. Computers & Education. 40, 255-269 (SSCI), 2003.

CORTELAZZO, I. B. C. *Colaboração, Trabalho em equipe e as tecnologias de comunicação: relações de proximidade em cursos de pós-graduação*. Doutorado em educação. Faculdade de educação, USP, 2000.

ERICKSON, F. *Research currents: Learning and Collaboration in Teaching*. Language arts., 1989.

ESMIN, A. A. A. *Portugol/Plus: uma ferramenta de apoio ao ensino de lógica de programação baseado no portugol*. IV Congresso RIBIE, Brasília, 1998.

FARACO, R. *Uma Arquitetura de um Sistema Learning Companion para Ensino a Distância*. Programa de pós-graduação em Engenharia de Produção, UFSC. Tese de doutorado, 2003.

FARACO, R.A., ROSATELLI, M.C.; GAUTHIER, F.A. (a) *Adaptivity in a learning companion system*. In Proc. of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004), in press. IEEE Press, 2004.

FARACO, R. A., ROSATELLI, M. C. & GAUTHIER, F. A. (b) *A Learning companion system for distance education in computer science*. In Anais do XII WEI - Workshop de Educação em Computação, XXIV Congresso da Sociedade Brasileira de Computação, CD ROM. Salvador: SBC, 2004.

FARRER, H.; BECKER, C.G.; FARIA, E. D.; CAMPOS, F. F Fº; MATOS, H. F.; SANTOS, M. A.; MAIA, M. L. *Pascal Estruturado*. 2ª.ed. Rio de Janeiro : Guanabara Koogan, 1995.

FARRER, H.; BECKER, C.G.; FARIA, E. D.; MATOS, H. F.; SANTOS, M. A.; MAIA, M. L. *Algoritmos Estruturados*. 2ª ed. Rio de Janeiro : Guanabara Koogan, 1989.

FERRUZZI, E. C. *Considerações sobre o LOGO*. Seminário Apresentado no GEIAAM – Grupo de Estudos de Inteligência Artificial Aplicada à Matemática-UFSC - Setembro/2001. Acessado em 23-01-2004.

FORBELLONE, A. L V.; EBRSPÄCHER, H. F. *Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados*. São Paulo: Makron Books, 1993.

GANE, C.; SARSON, T. *Análise Estruturada de Sistemas*. Rio de Janeiro: LTC, 1983.

GARDNER, H. *Inteligências Múltiplas: a Teoria na Prática*. Porto Alegre: Artes Médicas, 1995.

GIRAFFA, L. M. M. *Uma Arquitetura de Tutor utilizando Estados Mentais*. Programa de pós-graduação em Ciência da Computação – UFRGS. Tese de doutorado, 1999.

GIRAFFA, L.; MARCZAK, S. (a) *Ambientes Inteligentes para Suporte ao Ensino de Programação*. Technical Reports Series. Number 028, 2003.

GIRAFFA, L.; MARCZAK, S.; BLOIS, M.; ALMEIDA, G. (b) *Modelando um Ambiente de Aprendizagem na Web: a Importância da Formalização do Processo de Desenvolvimento*. XIV SBIE – Simpósio Brasileiro de Informática na Educação. Rio de Janeiro, 2003.

GOLDENBERG, E. P. "Hábitos de Pensamento: um Princípio Organizador para o Currículo". Revista Educação e Matemática, nº 48, 1998. Acesso em 28-04-2005.

http://www.apm.pt/apm/revista/educ48/educ48_6.htm

GOODMAN, B.; SOLLER, A.; LINTON, F. e GAIMARI, R. *Encouraging Student Reflection and Articulation using Learning Companion*. International Journal of Artificial Intelligence in Education 9, pgs. 237-255, 1998.

GOODMAN, B.; SOLLER, A.; LINTON, F.; GAIMARI R. *Encouraging Student Reflection and Articulation using a Learning Companion*. in Proceedings of the 8th World Conference on Artificial Intelligence in Education (AI-ED 97), Kobe, Japan, 151-158. Acessado em Janeiro-2004. http://sra.itc.it/people/soller/documents/LuCy-AI-ED97.doc, 1997.

GUIMARÃES, A. *Introdução a Lógica de Programação*. Acessado em 28-12-2004.

http://twiki.im.ufba.br/pub/MAT146/Sem20022/MAT146.pdf

HIETALA, P. & NIEMIREPO. *Collaboration with Software Agents: What if the Learning Companion Agent Makes Errors?* Artificial Intelligence in Education. IOS Press, 1997.

HIETALA, P. & NIEMIREPO. *The Competence of Learning Companions Agents*. International Journal of Artificial Intelligence in Education, 1998.

KAPOOR, A.; MOTA, S.; PICARD, R. W. *Towards a Learning Companion that Recognizes Affect. Proceedings from Emotional and Intelligent II: The Tangled Knot of Social Cognition*, AAAI Fall Symposium, November 2001. Acessado em Janeiro-2004. http://vismod.media.mit.edu/tech-reports/TR-543.pdf

KEMP, R.; TODD, E.; LU, J. Y. *A Novel Approach to Teaching an Understanding of Programming. Artificial Intelligence in Education*. Eds.-IOS Press, pp. 449-451, 2003.

KOSLOSKY, M. A. N. *Aprendizagem baseada em casos um ambiente para ensino de lógica de programação*. Dissertação apresentada a Universidade Federal de Santa Catarina para obtenção do grau de Mestre em Engenharia de Produção. 1999.
Disponível em http://www.eps.ufsc.br/disserta99/koslosky/cap1.html#1.3, acessado em março de 2004.

LEMOS, M. A. *O Papel das Tecnologias Multimídia/Hipermídia em Sistemas Tutores Inteligentes*. Seminário, Universidade de São Paulo, Escola Politécnica - Engenharia Elétrica. Programa de pós-graduação da Universidade de São Paulo, 2001.

LUZZI, F.; FERREIRA, R. Z.; SENNA, R. C.; GIRAFFA, L. M. M. e BASTOS, R. M. *Assistente Inteligente para Suporte ao Ensino de Química Orgânica*. IV Congresso RIBIE, Brasília, 1998.

MARIETTO, M. G. B.; OMAR N. e FERNANDES, C. T. *Tendências nas Áreas de*

Sistemas de Tutoria Inteligente e Modelagem do Aprendiz. VIII Simpósio Brasileiro de Informática na Educação, 1997.

MARTIN, J. e McCLURE, C. *Técnicas Estruturadas e Case*. São Paulo: Makron, McGraw-Hill, 1991.

MARTIN, B. e MITROVIC, A. *Tailoring Feedback by Correcting Student Answers*. Proc. ITS'2000, LNCS Vol. 1839, Springer-Verlag. pp. 383-392, 2000.

MARTIN, B. e MITROVIC, A. *Automatic Problem Generation in Constraint-Based Tutors*. In: S. Cerri, G. Gouarderes and F. Paraguacu (eds.): Proceedings of 6th International Conference on Intelligent Tutoring Systems. Berlin: Springer-Verlag, pp. 388-398, 2002.

MARTIN, B. e MITROVIC, A. ITS: Domain Modelling: Art or Science? Artificial Intelligent in Education. H.U. Hoppe et al. (Eds.). IOS Press, 2003.

MARTINS, S. W.; CORREIA, L. H. A. *O Logo como Ferramenta Auxiliar no Desenvolvimento do Raciocínio Lógico – um Estudo de Caso*. International Conference on Engineering and Computer Education - ICECE 2003, Santos/SP. Acessado em 28-12-2004. http://www.inf.ufsc.br/~scheila/icece2003.PDF

MAYO, M.; MITROVIC, A.; McKENZIE, J. *CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation*. In Kinshuk, C. Jesshope e T. Okamoto, Proceedings of IEEE International Workshop on Advanced Learning Technologies, Los Alamitos: IEEE Computer Society Press, 151-154, 2000.

MENEZES, C. S.; PESSOA, J. M; VESCOVI, H. N.; CURY, D.; TAVARES, O. L.; GAVA, T. B. S.; CARDOSO, E. P.; BAZZARELLA, L. B.; CASTRO, A. N. J. *Educação a distância no ensino Superior – Uma proposta baseada em Comunidades de Aprendizagem usando Ambientes Telemáticos*. XII Simpósio Brasileiro de Informática na Educação – SBIE – UNISINOS 2002, pp. 168-177.

MITROVIC, A. (a) *Experiences in Implementing Constraint-Based Modeling in SQL-Tutor*. In: B. P. Goetll, H. M. Halff, C. L. Redfield, e V. J. Shute (eds.): Proceedings

of 4th International Conference on Intelligent Tutoring Systems. Berlin: Springer-Verlag, pp, 414-423, 1998.

MITROVIC, A. (b) *Learning SQL with a Computerizes Tutor*. Technical Symposium on Computer Science Education. Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education. Atlanta, Georgia, United States. Pp. 307 – 311, 1998.

MITROVIC, A., e OHLSSON, S. *Evaluation of a constraint-based tutor for a database language*. International Journal of Artificial Intelligence in Education 10, 238-256, 1999.

MUNIZ, C. A. *Educação Matemática nas séries iniciais. A criança das séries iniciais faz matemática?* Mesa Redonda – VII SBEM, RJ, 2001. Acesso em 25-04-2005.

http://www.sbem.com.br/ANAIS/VII%20ENEM/ARQUIVOS/mesa_5A.pdf

NOBRE, I. A. M.; MENEZES, C. S. *Suporte à Cooperação em um Ambiente de Aprendizagem para Programação (SAmbA).* XII Simpósio Brasileiro de Informática na Educação – SBIE – UNISINOS 2002, pp. 337-347.

OHLSSON, S. *Constraint-based Student Modeling.* In: J. E. Greer e G. McCalla (eds.): Student Modeling: The Key to Individualized Knowledge-Based Instruction. Berlin: Springer Verlag, pp.167-189, 1994.

OHLSSON, S. *Constraint-based student modeling.* Journal of Artificial Intelligence in Education 3(4), p.429-447, 1992.

OLIVEIRA, A. B; BORATTI, I. C. *Introdução à Programação – Algoritmos*. Florianópolis: Bookstore, 1999.

PANITZ, T. *Collaborative Versus Cooperative Learning*. Acessado em 03/03/2005. http://home.capecod.net/~tpanitz/tedsarticles/coopdefinition.htm

PETRY, P. G.; DEGEN, G.; PERES, F.; OLIVEIRA, R. *Tecnologias de Cooperação na Pedagogia de Projetos*. In: XXIII CONGRESSO DA SOCIEDADE

BRASILEIRA DE COMPUTAÇÃO - SBC2033, WIE2003 - IX WORKSHOP SOBRE INFORMÁTICA NA ESCOLA. 2003.

PIMENTEL, E. P.; FRANÇA, V. F.; OMAR, N. A *Caminho de um Ambiente de Avaliação e Acompanhamento Contínuo da Aprendizagem em Programação de Computadores*. II Workshop de Educação em Computação e Informática do Estado de Minas Gerais, 2003. Acessado em 29-12-2004.

http://www.inf.pucpcaldas.br/eventos/weimig2003

PORTAL do Psicólogo. *Principais Termos e Conceitos utilizados na Elaboração de Laudos da Avaliação Psicológica*. Acessado em 28/12/2004.

http://www.portaldopsicologo.com.br/diversos/termosutilizados.htm

PRADO, M. E. B. B. *Logo - Linguagem de programação e as implicações pedagógicas*. Nied-Unicamp. Disponível em http://www.nied.unicamp.br/oea, serviu de base para o debate - via Internet - ocorrido na Oficina: Programação Logo e implicações pedagógicas, uma das ações internacionais desenvolvidas em 2000 pelo Nied-Unicamp, junto ao projeto Rede Telemática para Formação de Educadores, financiado pela OEA.

RASSENEUR, D.; DELOZANNE, E.; JACOBONI, P. e GRUGEON, B. *Learning with Virtual Agents: Competition and Cooperation in AMICO*. S.A. Cerri, G. Gouardères, and F. Paraguaçu (Eds.): ITS 2002, LNCS 2363, pgs. 61-70, 2002.

REGO, T.C. *Vygotsky: Uma perspectiva histórico-cultural da educação*. Ed. Vozes, 1995.

RODRIGUES, M. C. J. *Como Ensinar Programação?* Jornal Computação Brasil da

Sociedade Brasileira de Computação, 2002. http://www.unit.br/methanias/artigos.htm

RODRIGUES, M. C. J. *Experiências Positivas para o Ensino de Algoritmos*. IV ERBASE – IV Escola Regional de Computação Bahia-Sergipe. Feira de Santana/BA, 2004. Acessado em 05-01-2005.

http://www.uefs.br/erbase2004/documentos/weibase/Weibase2004Artigo001.pdf

ROSATELLI, M. C. e SELF, J. A. *Supporting distance learning from case studies*. In S. P Lajoie, e M. Vivet (Eds.), Proceedings of 9th international conference on artificial intelligence in education (pp. 457-564). Amsterdam: IOS Press, 1999.

ROSATELLI, M. C. *Um Ambiente Inteligente para Aprendizado Colaborativo no Ensino a Distância Utilizando o Método de Casos*. Tese de doutorado na UFSC, 1999.

ROSATELLI, M. C. *Novas tendências da pesquisa em inteligência artificial na educação*. In R. C. Nunes (Ed.), VIII Escola de Informática da SBC Sul, pp. 179-210. Porto Alegre: Editora da UFRGS, 2000.

ROSATELLI, M. C., SELF, J. A. e THIRY, M., LeCS: *A Collaborative Case Study System*. In: C. Frasson, G. Gauthier, e K. VanLenh (eds.): Proceedings of 5th International Conference on Intelligent Tutoring Systems. Berlin: Springer-Verlag, pp. 242-251, 2000.

ROSATELLI, M. C., SELF, J. A., & CRISTOFOLETTI, T. V. D. *An agent-based system for supporting learning from case studies*. In F. J. Garijo, J. C. Riquelme, & M. Toro (Eds.), Advances in Artificial Intelligence – IBERAMIA 2002 (LNCS 2527), pp.745-754. Berlin: Springer-Verlag, 2002.

RUSSELL, S. e NORVIG, P. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence, New Jersey, 1995.

SALIBA, W. L. C. *Técnicas de Programação – Uma Abordagem Estruturada*. São Paulo: Makron, McGraw-Hill, 1992.

SANTOS, G.; DIRENE, A. I.; GUEDES, A. L. P. *Autoria e Interpretação de Soluções Alternativas para Promover o Ensino de Programação de Computadores*. XIV SBIE – Simpósio Brasileiro de Informática na Educação. Rio de Janeiro, 2003.

SCHNEIDER, B. Tradução do livro de AED 1: *An Object-Oriented Introduction to Computer Science Using Eiffel*. Acesso em 26-02-2005.

SELF, J. *Computational Mathetics: Towards a Science of Learning Systems Design.* Acessado em 01/10/2003. http://www.cbl.leeds.ac.uk/~jas/cm.html, 1995.

SOUZA, E. M. S; GRANDI, G.; SOUZA, O. R. M.; DAZZI, R. L. S. *Um novo Enfoque para o Processo Ensino-Aprendizagem de Lógica e Algoritmos em Cursos de Graduação*. Anais do I Simpósio Catarinense de Computação. Itajaí-SC, 2000, Pgs. 81-89.

SOUZA, E. M. S. *Um Modelo para Processo Ensino-Aprendizagem de Procedimentos Lógicos em Diversos Domínios*. Tese apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina, 2000.

TALL, D. *A Resposta é o de menos*. Entrevista. Notícias selecionadas no mundo da Educação, 2002. Acesso em 26-04-2005.

http://novaescola.abril.com.br/noticias/fev_02_27/index_fev_02_27b.htm

URESTI, J. A. R. *Should I Teach My Computer Peer? Some Issues in Teaching a Learning Companion*. Proceedings of Intelligent Tutoring Systems, Montreal, Canada, Springer, 2000.

VICARI, R.; SILVEIRA, R. A.; GOMES, E. R. *Modelagem de Ambientes de Aprendizagem baseado na utilização de Agentes FIPA*. XIV SBIE – Simpósio Brasileiro de Informática na Educação. Rio de Janeiro, 2003.

VISUALG. Acessado em 03-09-2004.
http://www.apoioinformatica.inf.br/visualg/linguagem.htm

VIZCAÍNO, A.; CONTRERAS, J.; FAVELA, J.; PRIETO, M.. *An Adaptive, Collaborative Environment to Develop Good Habits in Programming*. Intelligent Tutoring Systems, Montreal, pp.262-271, 2000.

VISCAÍNO, A. *HabiPro: A Collaborative Application to Develop Programming Skills*. HCT, 2000 - 4th Human Centred Technology Postgraduate Workshop.

University of Sussex, School of Cognitive and Computing Sciences.

http://www.informatics.susx.ac.uk/research/hct/hctw2000/papers/vizcaino.pdf

WOOLDRIDGE, M. e JENNINGS, N. R. *Agent Theories, Architectures, and Languages: a Survey*. Wooldridge and Jennings Eds., Intelligent Agents, Berlin: Springer-Verlag, 1-22, 1995.

APÊNDICE A – Restrições

1. Cr: Se início então
   Cs: deve ter fim (senão acionar companheiro de aprendizagem)

2. Cr: Se fim então
   Cs: deve ter início (senão acionar companheiro de aprendizagem)

3. Cr: Se leia então
   Cs: deve conter uma variável declarada (senão acionar companheiro de aprendizagem)

4. Cr: Se escreva então
   Cs: deve conter uma variável da lista de variáveis OU uma string após a palavra escreva (senão acionar companheiro de aprendizagem)

5. Cr: Se usar enquanto então
   Cs: deve ter fim enquanto (senão acionar companheiro de aprendizagem)

6. Cr: Se (enquanto variável_1 < variável_2) então
   Cs: variável_1 variável_2 devem ser inicializadas OU lidas antes do enquanto (senão acionar companheiro de aprendizagem)

7. Cr: Se (enquanto variável_1 < variável_2) então
   Cs: O valor da variável_1 e da variável_2 devem ser inteiro ou real declaradas na lista de variáveis (var) (senão acionar companheiro de aprendizagem)

8. Cr: Se (enquanto variável_1 < variável_2) então
   Cs: O valor da variável_1 deve ser menor que o valor da variável_2 para entrar no laço (senão acionar companheiro de aprendizagem)

9. Cr: Se (enquanto variável_1 < variável_2) então
   Cs: O valor da variável_1 deve ser incrementado até se igualar a variável_2 (senão acionar companheiro de aprendizagem)

10. Cr: Se (enquanto variável_1 <= variável_2) então
    Cs: O valor da variável_1 deve ser incrementado até ser maior que a variável_2 (senão acionar companheiro de aprendizagem)

11. Cr: Se (enquanto variável_1 > variável_2) então
    Cs: O valor da variável_1 deve ser decrementado até se igualar a variável_2 (senão acionar companheiro de aprendizagem)

12. Cr: Se (enquanto variável_1 >= variável_2) então
    Cs: O valor da variável_1 deve ser decrementado até ser menor que a variável_2 (senão acionar companheiro de aprendizagem)

13. Cr: Se (enquanto variável_1 = “sim”) então
    Cs: O valor da variável_1 deve ser uma string declarada na lista de variáveis (senão acionar companheiro de aprendizagem)

14. Cr: Se (enquanto variável_1 = “sim”) então
    Cs: O valor da variável_1 deve ser inicializado OU lido antes do enquanto (senão acionar companheiro de aprendizagem)

15. Cr: Se (enquanto variável_1 = “sim”) então
    Cs: O valor da variável_1 deve ser lido dentro do enquanto até se igualar ao “sim” (senão acionar companheiro de aprendizagem)

16. Cr: Se (variável_1 ← <expressão>) então
    Cs: A variável_1 deve ser declarada na lista de variáveis (senão acionar companheiro de aprendizagem)

17. Cr: Se (variável_1 ← <expressão>) então

    Cs: O tipo da <expressão> deve ser de mesmo tipo da variável_1 (senão acionar companheiro de aprendizagem)

18. Cr: Se <expressão> então

    Cs: Pode ser usado os símbolos +, -,*, /, not, and, or (senão acionar companheiro de aprendizagem)

19. Cr: Se (var) então

    Cs: Todas as variáveis desta lista devem ser declaradas (senão acionar companheiro de aprendizagem)

20. Cr: Se após a linha do var e antes da linha de inicio existir um nome diferente de string, lógico, inteiro ou real e não iniciar com números ou caracteres especiais então

    Cs: a palavra encontrada é uma variável válida (senão acionar companheiro de aprendizagem)

21 Cr: se palavra encontrada for válida para nome de variável então

    Cs: deve haver dois pontos (:) após a palavra para definir o tipo de dado desejado (senão acionar companheiro de aprendizagem)

22 Cr: Se tipo de dado então

    Cs: o tipo de dado pode ser string, real, inteiro ou lógico (senão acionar companheiro de aprendizagem)

23. Cr: Se *string* então

    Cs: Inicia e termina com “ ” quando atribui-se uma informação à variável do tipo string (senão acionar companheiro de aprendizagem)

24. Cr: Se real então

Cs: Tem ponto na atribuição (senão acionar companheiro de aprendizagem)

25. Cr: Se lógico então

Cs: Valor da variável = verdade ou falso (senão acionar companheiro de aprendizagem)

26: Cr: se algoritmo então

Cs: deve haver um nome válido após a palavra algoritmo iniciando por letras do alfabeto (senão acionar companheiro de aprendizagem)

27: Cr: se inicio então

Cs: antes deve ter a algoritmo (senão acionar companheiro de aprendizagem)

28: Cr: se inicio então

Cs: não pode ter sido declarada anteriormente (senão acionar companheiro de aprendizagem)

29: Cr: se fim então

Cs: deve ser a última linha do algoritmo (senão acionar companheiro de aprendizagem)

30: Cr: se <variável> após leia então

Cs: pode não ter sido declarada anteriormente (senão acionar companheiro de aprendizagem)

31: Cr: se variável após início então

Cs: pode ser feita uma atribuição (senão acionar companheiro de aprendizagem)

32: Cr: se variável após início então

Cs: pode ser feita uma leitura com o comando leia (senão acionar companheiro de aprendizagem)

33: Cr: se variável após início então

Cs: pode ser feita uma escrita com o comando escreva (senão acionar companheiro de aprendizagem)

34. Cr: Se <expressão> for uma variável <variável_2> então

Cs: Variável <variável_2> deve estar declarada anteriormente (senão acionar companheiro de aprendizagem)

35. Cr: Se <expressão> for uma *string* então

Cs: A <expressão> deve estar entre aspas (senão acionar companheiro de aprendizagem)

APÊNDICE B – Exercícios Utilizados nos Testes

Declaração de Variáveis

1. Escreva um algoritmo para declarar 3 variáveis do tipo Inteiro.
2. Declare 2 variáveis do tipo Literal.
3. Defina 5 variáveis como sendo do tipo Real.
4. Fazer um algoritmo para declarar as variáveis nome, cidade, idade e salário de um funcionário.

Estruturas Seqüenciais

1. Fazer um algoritmo para declarar as variáveis nome, cidade, idade e salário. Após, atribuir as seguintes informações a elas:
   Nome = "Ana"
   Cidade = "Floripa"
   Idade = 34
   Salário = 456.89

2. Fazer um algoritmo para ler o nome, cidade, idade e salário de um funcionário.