Sistemas Tutores no Domínio da Programação
Tutoring Systems in the Field of Programming

Carlos Alberto Botelho
USCS - Universidade Municipal de São Caetano do Sul - São Caetano do Sul - SP - Brasil
ca.botelho@terra.com.br

Resumo: A literatura a respeito do ensino de programação de computadores indica que a dificuldade para o aprendizado da programação de computadores reside no fato de que é preciso conhecer a linguagem e saber como aplicá-la no desenvolvimento de soluções por intermédio do uso do computador. Nos últimos anos, com o aumento da demanda de profissionais de programação, vêm sendo desenvolvidas novas técnicas para o ensino da programação. Neste artigo, é feita uma revisão dos sistemas tutores inteligentes construídos para o ensino da programação de computadores. Na primeira parte, é apresentado um resumo da história do desenvolvimento de sistemas voltados ao ensino por computador. Na segunda parte, são abordados diversos aspectos considerados na construção de sistemas tutores inteligentes para o ensino da programação de computadores. A seguir, são descritos alguns sistemas hoje existentes, com a finalidade de ensinar programação de computadores.

Palavras-chave: Sistemas Tutores Inteligentes, Inteligência Artificial, Ensino por Computador.

Abstract: The literature regarding the teaching computer programming indicates that the difficulty of learning computer programming resides in the fact that one needs to know the language and how to apply it in the development of solutions using a computer. In the last years, with the increase in the demand for programming professionals, new techniques of teaching programming have been developed. In this article, a revision of the intelligent tutoring systems is done. In the first part, a summary of the history of the system development directed to computer teaching is done. In the second part, multiple aspects considered in the construction of intelligent tutoring systems for the teaching of computer programming are considered. Following, some systems with the goal of teaching computer programming are described.

Keywords: Intelligent Tutoring systems, Artificial Intelligence, Computer Based Teaching.

1 INTRODUÇÃO

O processo de ensino utilizando os computadores como ferramenta teve início no final da década de 1950, quando o psicólogo americano B. F. Skinner, baseado na teoria comportamentalista, propôs um método de ensino, que chamou de máquina de ensinar. Nesse método, o material a ser ensinado se divide em módulos seqüenciais, cada módulo terminando com questões a serem respondidas pelo estudante. Acertando as respostas, o estudante é autorizado a passar para o módulo seguinte.

Com base nesse método, foram criados os primeiros programas de computador voltados para o ensino: os *Computer Aided Instruction*, conhecidos por CAI.

Nos sistemas de ensino suportados pelo CAI, o aluno segue uma seqüência finita e predeterminada de passos. A cada passo, o conhecimento adquirido pelo estudante é testado e, se corresponde ao correto, passa para o próximo passo; caso contrário, o conhecimento ainda não assimilado é novamente exposto ao estudante e o teste é refeito. Essa operação se repete até que o estudante responda corretamente ao teste.

Esses sistemas têm como característica a transmissão do conhecimento feita sempre da mesma forma, independente do perfil do estudante. São hoje conhecidos como "viradores de páginas".

Durante a década de 1960, foram feitos muitos investimentos na produção de sistemas CAI, pelo governo norte-americano, por empresas da

área de Tecnologia da Informação, tais como IBM, Digital, RCA e por universidades. O sistema CAI mais bem-sucedido até hoje é o Plato (*Programmed Logic for Automatic Teaching Operations*), desenvolvido em parceria entre a *Computer Education Research Laboratory*, da Universidade de Illionois, a *Control Data Corporation* e a *National Science Foundation* para computadores de grande porte.

Em 1982, Sleeman & Brown (SLEEMAN & BROWN, 1982 *apud* FONSECA JR., 2004) revisaram o estado da arte nos sistemas CAI e, com base na psicologia cognitivista, na abordagem construtivista do ensino e nas técnicas de inteligência artificial, criaram uma nova maneira de usar o computador no ensino, que denominaram como "sistemas tutores inteligentes".

Ao contrário dos sistemas CAI tradicionais, que atuam sempre da mesma maneira para todo estudante, os STI atuam conforme as características de cada estudante.

O quadro abaixo mostra com clareza as principais diferenças entre os sistemas CAI e os sistemas STI.

Segundo (JONASSEN, 1993 *apud* FONSECA JR., 2004), um STI deve atender a três requisitos, quais sejam:

1. o conteúdo deve ser codificado, de modo que o sistema possa rapidamente acessar as informações, fazer inferências ou resolver problemas;
2. o sistema deve ser capaz de avaliar a aquisição deste conhecimento pelo aluno;
3. as estratégias tutoriais devem ser projetadas para reduzir a discrepância entre o conhecimento do especialista e o conhecimento do aluno.

Para Urretavizcaya (2001), as características mais importantes de um STI são as seguintes:

1. conhecimento do domínio restrito e claramente articulado;
2. conhecimento sobre o aluno, que permita dirigir e adaptar o ensino;
3. seqüência de ensino não-predeterminada;
4. produção de diagnósticos mais detalhados e adaptados ao aluno;
5. melhora na interação entre tutor e aluno, permitindo que o aluno faça perguntas ao tutor.

A arquitetura tradicional de um STI, demonstrada na Figura 1, contém quatro componentes básicos (WENGER, 1987 *apud* URRETAVIZCAYA, 2001), a seguir explicitados.

1. Módulo do domínio, que contém a matéria a ser ensinada.

Quadro 1:
Principais diferenças entre os sistemas CAI e os sistemas STI

ASPECTO	CAI	STI
Origem	Educação.	Ciência da Computação.
Bases teóricas	Skinner (behaviorista).	Psicologia cognitivista.
Estruturação e funções	Uma única estrutura algoritmicamente predefinida, onde o aluno não influi no seqüenciamento.	Estrutura subdividida em módulos, cujo eqüenciamento se dá em função das respostas do aluno.
Estruturação do conhecimento	Algorítmica.	Heurística.
Modelagem do aluno	Avaliam a última resposta.	Tentam avaliar todas as respostas do aluno durante a interação.
Modalidades	Tutorial, exercício e prática, simulação e jogos educativos.	Socrático, ambiente interativo, diálogo bidirecional e guia.

Fonte: GIRAFFA (1997: 19).

2. Módulo do aluno, que contém as características do aluno.
3. Módulo do tutor, que contém as estratégias e táticas para a aplicação no processo de ensino.
4. Módulo da interface, que contém a forma de comunicação entre o tutor e o aluno.

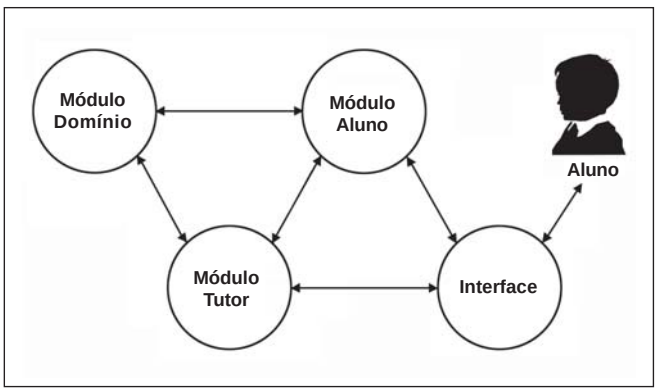

Figura 1: Os componentes de um sistema tutor inteligente
Fonte: GOULART & GIRAFFA (2001).

2 SISTEMAS TUTORES PARA ENSINO DA PROGRAMAÇÃO DE COMPUTADORES

Este trabalho abordará um segmento particular dos sistemas tutores inteligentes, que trata dos sistemas criados especificamente para o ensino da programação de computadores.

Segundo Pillay (2003), um tutor para o ensino da programação de computadores deve ter as seguintes funções:

- apresentação e explicação de conceitos de programação;
- apresentação ao estudante de diferentes tipos de problemas, cada qual abordando um aspecto das técnicas de programação;
- assistência ao estudante no desenvolvimento de soluções dos diferentes tipos de problemas, avaliando a solução;
- avaliação dos programas escritos pelo estudante, com respeito à exatidão e à eficiência;
- assistência ao estudante no processo de localização e correção de erros de semântica nos seus programas, o que implica a detecção de erros de lógica, identificação de mal-entendimentos do estudante e aconselhamento ao estudante de como corrigir os erros.

Ainda segundo Pillay (2003), um tutor para o ensino da programação, para atender às funções acima, deve ter a arquitetura genérica mostrada na Figura 2. Na arquitetura proposta por Pillay, são acrescentados os seguintes módulos à arquitetura de sistemas tutores inteligentes:

- módulo especialista, que deve conter as soluções dos problemas apresentados ao estudante;
- módulo de problemas, que deve conter os problemas a serem apresentados ao estudante, se não for possível gerá-los automaticamente;
- módulo de especificação do código, que deve conter as especificações para a codificação do algoritmo numa determinada linguagem. Assim, o sistema tutor de programação pode ser construído para o ensino de qualquer linguagem;
- módulo de explicações, que deve gerar explicações, obtidas do módulo especialista, sobre os erros cometidos pelo estudante;
- módulo de aprendizagem/experiência, que deve atualizar os demais módulos, fornecendo informações sobre a interação com o estudante.

Ramadhan & Shihab (2000) dividiram os tutores para o ensino de programação em duas categorias, expressas nos itens abaixo:

- diagnosticadores automáticos de programas;
- sistemas de auxílio para a construção de programas.

Os diagnosticadores automáticos de programas diagnosticam programas construídos por estudantes e indicam os erros, fornecendo diagnósticos. Devido à grande diversidade de soluções corretas e erradas que podem ser produzidas na construção de programas, os tutores conseguem apenas realizar a análise de programas muito simples e predefinidos.

Os sistemas de auxílio para a construção de programas conduzem o estudante no aprendizado, utilizando, para isso, ambientes visuais e muita interação com o estudante.

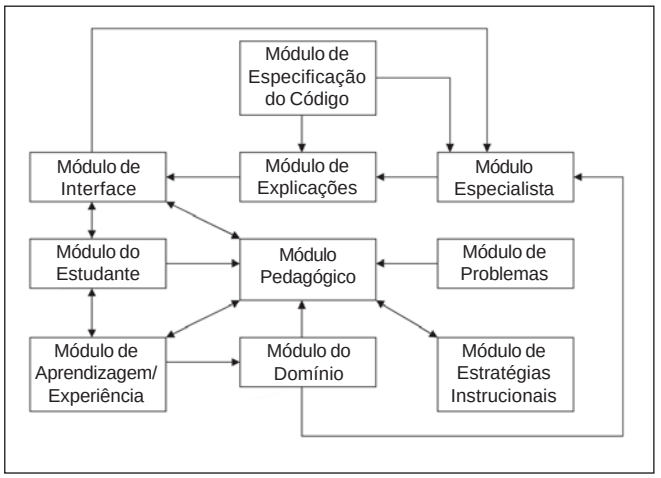

Figura 2: A arquitetura genérica de sistemas tutores de programação

Fonte: PILLAY (2003).

2.1 Diagnosticadores automáticos de programas

A descoberta de erros e a emissão de diagnóstico em programas é um processo extremamente complicado, devido às inúmeras possibilidades de formas de construção de um programa. Ducassé (1993) pesquisou diversos diagnosticadores automáticos de programas e concluiu que são empregadas três estratégias para diagnosticar programas, explicitadas na seqüência.

- Verificação com respeito à especificação: o programa em análise é comparado com alguma especificação preestabelecida, que deve ser seguida pelo programa.
- Verificação referente à linguagem: procura erros de sintaxe e semântica no programa.
- Filtragem com respeito ao sintoma: assume, como corretas, partes do código que não podem ser responsabilizadas pelos erros encontrados.

Ramadhan & Shihab (2000) classificaram esses programas, considerando o momento em que atuam, em *post-event systems* e *in-event systems*.

2.1.1 Diagnosticadores post-event

Os diagnosticadores classificados como *post-event systems* consideram que o programa está completo, ao fazerem a depuração.

Segundo Boulay (BOULAY & SOTHCOTT, 1987 *apud* SANTOS, 2003), os diagnosticadores podem ser classificados, conforme o método empregado, da seguinte forma:

- comparadores (*Specimen Answer*): neste caso, o programa do estudante é comparado com uma versão correta do programa, e as diferenças são consideradas como possíveis erros. Um sistema que usa esse método é o Laura (ADAM & LAURENT, 1980). Outro sistema, que aplica a comparação, mas empregando a técnica de modelagem baseada em restrições, é o SQL-Tutor (MITROVIC, 1997);
- analisadores de especificação (*Specification Analisys*): o programa do estudante é analisado conforme especificações do problema, fornecidas previamente. Um dos sistemas que utilizam esse método é o Proust (JOHNSON & SOLOWAY, 1984);
- dialogadores de depuração (*Debugging Dialogue*): neste caso, o sistema interage com o estudante, que orienta o processo de depuração. O sistema que emprega este método é o PDS6.

2.1.2 Diagnosticadores in-event

Alguns diagnosticadores atuam orientando o estudante durante o processo de construção do programa, de tal forma que, ao concluir o programa, este estará correto. Utilizam esse método, entre outros, o LispTutor (ANDERSON & REISER, 1985 *apud* RAMADHAN & SHIHAB, 2000), o ELM-ART (BRUSILOVSKY & WEBER, 2006)[1] e o AlgoLC (PETRY, 2005).

3 DESCRIÇÃO DOS SISTEMAS

A seguir, é apresentada uma breve descrição dos sistemas citados anteriormente.

3.1 Laura

Laura foi desenvolvido na Universidade de Caen, na França, e é um diagnosticador de pro-

[1] Disponível em: <http://apsymac33.uni-trier.de:8080/LispCourse>, em 11/06/2006.

gramas escritos na linguagem Fortran. Embora não possa ser classificado com um sistema tutor inteligente, pois não possui o módulo do estudante, Laura é uma das primeiras tentativas de construção de sistemas tutores para o ensino da programação de computadores e, por essa razão, está incluído neste trabalho.

O domínio do conhecimento de Laura é o conjunto dos modelos dos programas que resolvem os problemas propostos aos estudantes. Laura consegue verificar um programa se houver um modelo do mesmo programa no domínio do conhecimento.

O método empregado em Laura é o da transformação do programa do estudante para um padrão na forma de grafo, que é comparado com o modelo correspondente existente no domínio do conhecimento (Figuras 3 e 4).

A interface com o estudante é feita por meio de listagens demonstrativas do diagnóstico realizado (Figura 5).

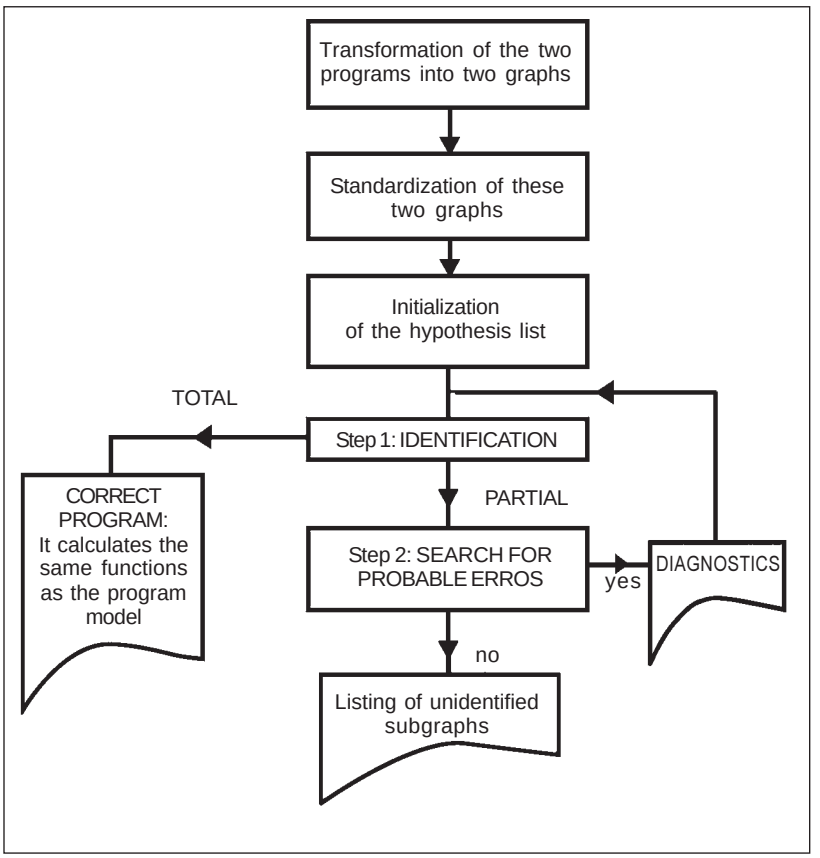

Figura 4: Fluxograma geral do Laura
Fonte: Adam & Laurent (1980: 98).

3.2 Proust

Proust (*Program Understander for Students*) foi desenvolvido na Universidade de Yale, e é um diagnosticador de programas escritos na linguagem de programação Pascal, que analisa programas inteiramente prontos. O seu objetivo é analisar um programa mediante metas e planos previamente estabelecidos e apresentar um diagnóstico sobre as incoerências existentes no programa com respeito a essas metas e esses planos.

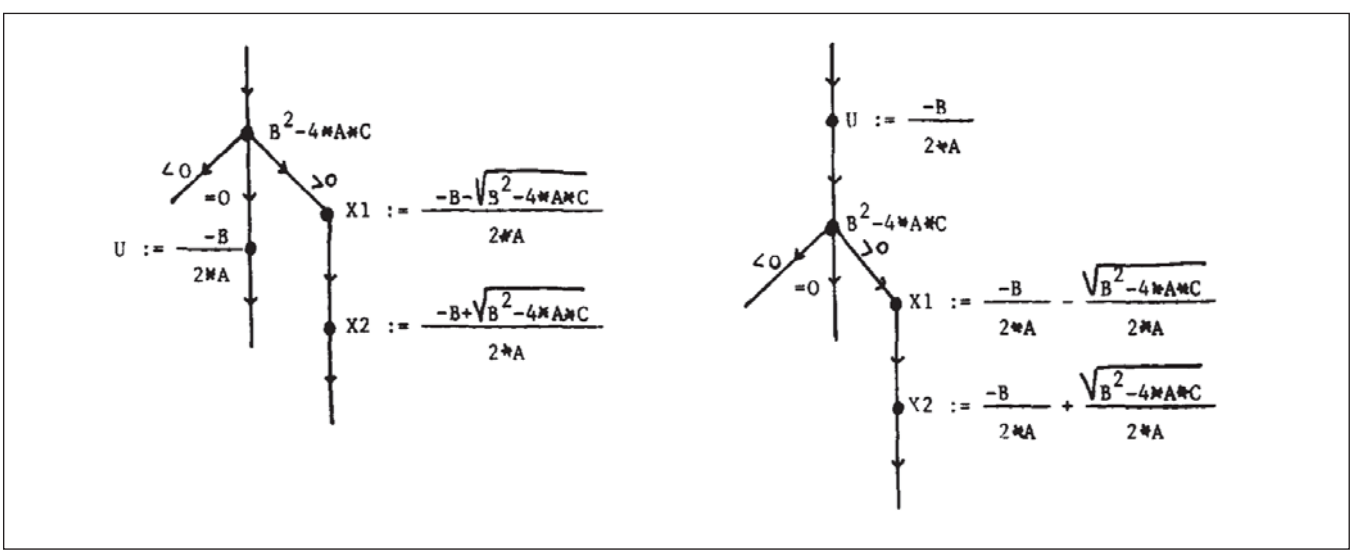

Figura 3: Modelo e programa do estudante (em grafos)
Fonte: Adam & Laurent (1980: 85).

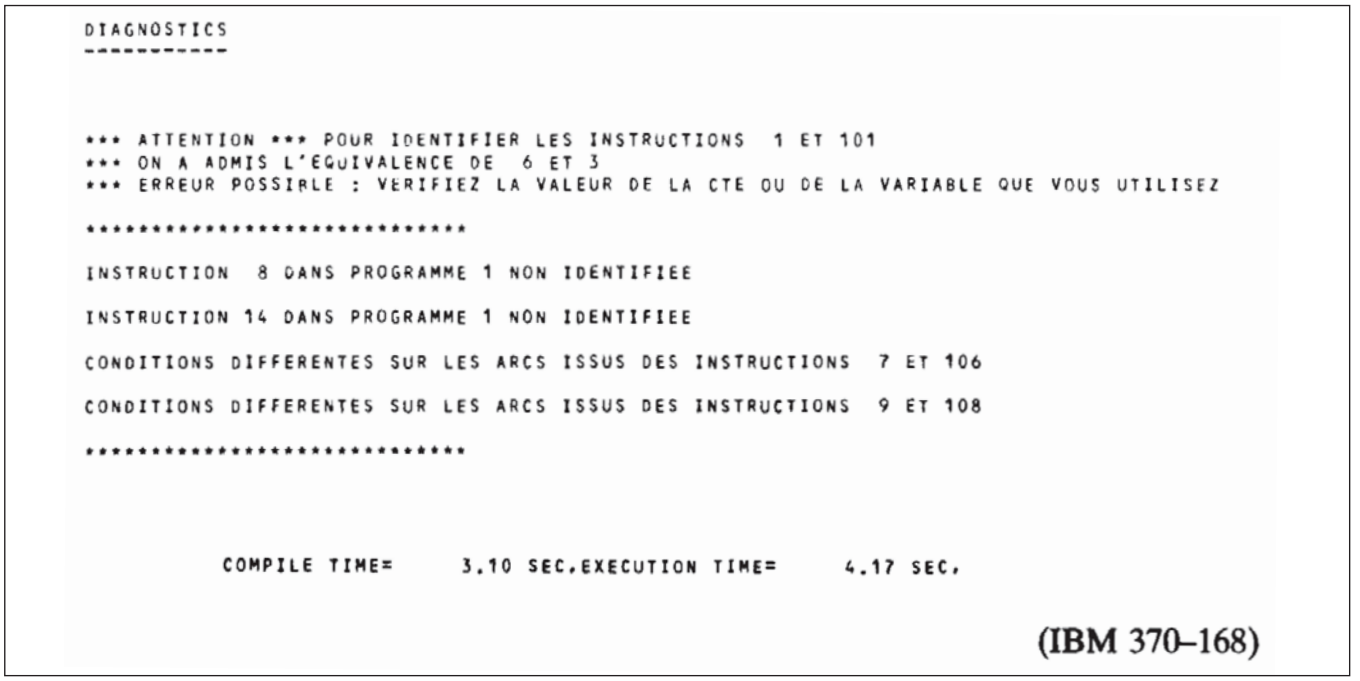

Figura 5: Listagem de diagnóstico
Fonte: ADAM & LAURENT (1980: 98).

O módulo do domínio do conhecimento contém as diversas metas e os planos que devem orientar a construção de um programa preestabelecido. Entre os planos previamente cadastrados, alguns correspondem a versões corretas do programa e outros, a versões incorretas, com os respectivos diagnósticos.

O sistema não interage com o estudante, pois age apenas quando o programa está pronto, emitindo uma relação de erros encontrados, com um diagnóstico sobre a possível origem do erro.

A análise é feita pela comparação do código escrito em Pascal, com as metas e os planos previamente cadastrados no sistema, e que não ficam disponíveis para o estudante. Encontrando algum plano que corresponda ao programa apresentado pelo estudante, é emitido um diagnóstico.

Seja o problema do cálculo da média de um conjunto de números digitados, o conjunto será finalizado pelo número 99999, que não deverá ser considerado para o cálculo da média (JOHNSON & SOLOWAY, 1984). A Figura 6 mostra a decomposição da solução em planos e objetivos; a Figura 7 mostra uma tentativa de solução de um estudante; e a Figura 8 mostra uma das soluções corretas do exercício.

Segundo Johnson & Soloway (1984), a análise de um programa feita pelo Proust pode falhar num dos seguintes casos:

- erros ocasionais, cuja baixa freqüência não justifica a sua inclusão na base de conhecimento;
- programas que contenham planos impossíveis de ser previstos;
- casos de ambigüidade, que somente podem ser resolvidos por meio de diálogo com o estudante.

3.3 Lisp Tutor

O Lisp Tutor foi desenvolvido na Carnegie-Mellon University e se baseia na Teoria do Controle Adaptativo do Pensamento (*ACT – Adaptative Control of Tought Theory*). A sua intervenção é feita na forma *in-event*, dirigindo a construção do programa.

O módulo do domínio do conhecimento é constituído de um conjunto de regras de produção corretas, necessárias para criar um programa em Lisp.

A modelagem do estudante é feita interativamente, pelo acompanhamento do progresso do

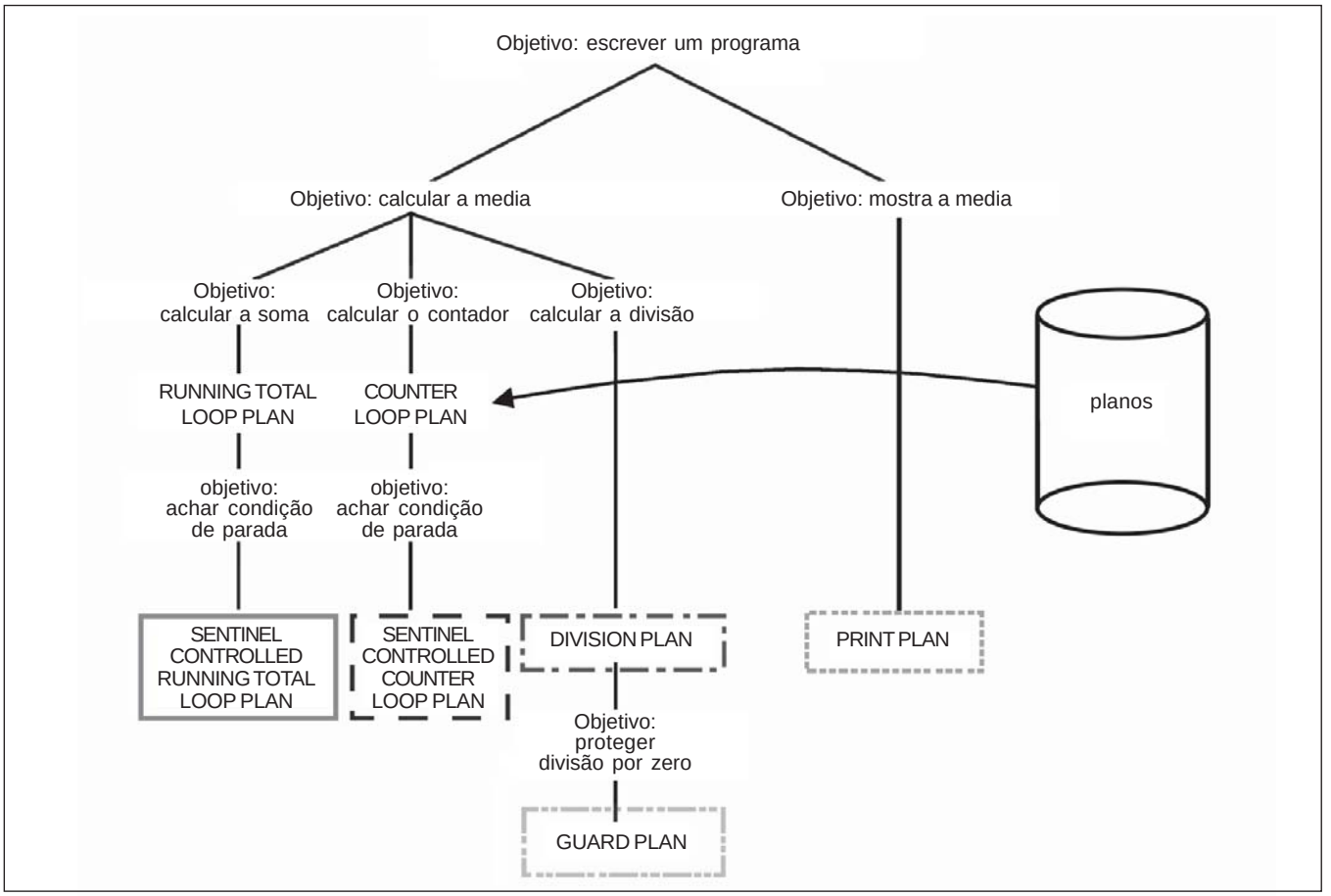

Figura 6: Decomposição da solução do problema em objetivos, subobjetivos e planos

Fonte: DELGADO (2004: 11)

```
1     PROGRAM Average( input, output );
2     VAR Sum, Count, New, Avg: REAL;
3     BEGIN
4       Sum := 0;
5       Count := 0;
6       Read( New );
7       WHILE New<>99999 DO
8         BEGIN
9           Sum := Sum+New;
10          Count := Count+1;
11          New := New+1
12        END;
13      Avg := Sum/Count;
14      Writeln( 'The average is ', avg );
15    END;
```

PROUST output:

It appears that you were trying to use line 11 to read the next input value. Incrementing NEW will not cause the next value to be read in. You need to use a READ statement here, such as you use in line 6.

Figura 7: Exemplo da análise e do diagnóstico de um programa errado

Fonte: JOHNSON & SOLOWAY (1984: 374).

```
                PROGRAM Average( INPUT, OUTPUT );
                VAR Sum, Count, New, Avg: REAL;
Counter Variable BEGIN
Plan    ----------> Count := 0;
        |      ---> Sum := 0;          Running Total Loop Plan
        |      |    Read(New); <-------------------
Running |      |    WHILE New <> 99999 DO <-------|
Total   |      |      BEGIN                       |
Variable|      -------> Sum := Sum + New; <-------|
Plan    -------------> Count := Count + 1;       |
                       Read(New);        <--------
                     END;              Valid Result Skip Guard
                   IF Count > 0 THEN  <-------------------------
                     BEGIN <-----------------------------------|
                       Avg := Sum/Count; <-------------------|
                       Writeln( Avg); <----------------------|
                     END <---------------------------------|
                   ELSE  <-----------------------------------|
                     Writeln( 'no legal inputs'); <----------|
                 END.
```

Figura 8: O mesmo programa na versão correta, com a indicação dos planos

Fonte: JOHNSON & SOLOWAY (1984: 374).

estudante. O sistema acompanha continuamente o progresso do estudante e tenta inferir o conhecimento adquirido por intermédio do seu comportamento atual. Além disso, o sistema possui um conjunto de regras de produção erradas, que ocorrem com mais freqüência. A representação do conhecimento é feita por meio do Grapes (*Goal Restricted Production System Architecture*) (ANDERSON, FARREL & SAUERS, 1982).

O modelo do tutor utiliza dois componentes: o solucionador de problemas e o conselheiro. O solucionador de problemas consiste de um interpretador Grapes, regras do modelo novato e regras erradas. O conselheiro é um interpretador de regras de produção muito semelhante ao Grapes. A estratégia de ensino empregada é "aprender fazendo", de tal forma que o estudante vai descobrindo as regras de produção enquanto resolve o problema. O tutor age como um guia de solução do problema, mas jamais especifica as produções a serem aprendidas.

O processo de ensino é iniciado com a apresentação de um problema para ser programado em Lisp. O solucionador de problemas gera todas as possibilidades de regras corretas e erradas. Na medida em que o estudante constrói o programa, o conselheiro compara cada caráter da entrada do estudante com um caráter esperado, que pode pertencer ao conjunto das regras corretas ou ao conjunto de regras erradas. Se o caráter coincide com o esperado correto, o estudante é autorizado a continuar. Se o caráter coincide com o esperado errado, é emitida uma instrução para a correção do erro. Se o caráter não for um dos esperados, o tutor solicita um novo caráter e, após certo número de tentativas, explica o próximo passo.

Segundo Ramadhan (RAMADHAN & SHIHAB, 2000), o método tem a vantagem de apresentar um diagnóstico imediato de algum conceito mal-entendido, informando-o ao estudante que, assim, nunca se afasta da solução correta. Por outro lado, esse método torna o processo de aprendizagem muito restritivo, não permitindo ao estudante, por exemplo, reiniciar a construção do programa por outro caminho.

Na versão mais recente do Lisp Tutor, a verificação pode ser feita sob o comando do estudante, como um compilador. Dessa maneira, ao invés de o tutor verificar cada caráter digitado, deverá fazê-lo quando acionado pelo estudante. Assim, o estudante pode escrever o programa da maneira que quiser. Quando solicitado, o tutor fará a verificação do programa, examinando o texto da esquerda para a direita e de cima para baixo. Quando encontrar um erro, interromperá a verificação para a correção do erro, ignorando o restante do programa, procedendo da mesma maneira que faria na verificação por caráter.

Na versão antiga, caso haja mais do que um caminho para a mesma solução de um problema, o tutor não consegue identificar o caminho escolhido. Na versão mais recente, ao compilar o programa, o tutor poderá identificar o caminho escolhido pelo estudante e indicar erros de programação, como já indicava, e também erros de planejamento. Para isso, é fornecida, no editor, uma interface para o planejamento do programa.

O estudante interage com o sistema por meio de um editor de textos, que corrige automaticamente erros primários de sintaxe, tais como balanceamento de parêntesis, aspas etc. Contém também uma interface para o planejamento do programa, utilizada na construção de programas mais complexos.

3.4 ELM-ART

O sistema ELM-ART (*Episodic Learner Model-Adaptive Remote Tutor*) é um sistema tutor inteligente para o ensino da linguagem Lisp, utilizando a *Web*, que atua na forma *in-event*.

Segundo Brusilovsky (BRUSILOVSKY & WEBER, 2000), ELM-ART é como um livro inteligente *on-line* com um ambiente solucionador de problemas integrado. Todo o material do curso (apresentação de novos conceitos, testes, exemplos e problemas) é fornecido na forma de hipermídia.

No ELM-ART, o módulo do domínio do conhecimento é constituído de uma base de conhecimento contendo o conhecimento sobre a solução de problemas em Lisp, que é representado na forma de rede de conceitos, planos e regras. O coração desse conhecimento é a rede de conceitos Lisp (*LCN – Lisp Conceptual Network*), que represen-

ta todos os conceitos usados no curso e todas as relações entre eles.

Na primeira vez em que o tutor é usado, o modelo do estudante é inicializado com a obtenção de dados do estudante e do seu conhecimento anterior sobre a linguagem e o uso dos recursos de computação. Durante a interação, o desempenho do estudante é registrado e a seqüência de apresentação do material a ser ensinado depende das suas respostas às perguntas formuladas em cada tópico. Encerrada uma sessão, os dados obtidos da interação são armazenados e utilizados no início de uma próxima sessão.

Durante uma sessão de uso, observa-se que o tutor apresenta o conteúdo na forma de tópicos, numa seqüência lógica e hierárquica de abordagem dos conceitos da linguagem Lisp. Em cada tópico, é apresentada uma explicação sobre o conceito principal e uma lista de exercícios.

A lista de exercícios é apresentada, conforme o tópico, como uma pergunta com as respostas na forma de múltipla escolha ou como uma pergunta e uma janela de texto para a resposta na forma dissertativa. Caso as respostas estejam corretas, o próximo tópico é liberado.

No caso de resposta errada, é apresentado um *feedback* a respeito do erro cometido. Em múltipla escolha, é apresentada a resposta correta e um novo exercício, com menor grau de dificuldade, sobre o mesmo assunto. No caso da resposta dissertativa errada, é apresentada uma janela de diagnóstico e a mesma janela da resposta, para a correção. Esse processo é repetido até que o estudante resolva os exercícios corretamente.

Se o estudante tentar abordar um tópico fora da seqüência, será avisado desse fato e aconselhado a voltar à seqüência proposta pelo tutor.

A interface com o estudante é feita por intermédio de uma janela, contendo os seguintes elementos:

- no lado esquerdo, a relação dos tópicos apresentados numa estrutura hierárquica. O estudante pode escolher o tópico que quer abordar ou seguir a indicação do próximo tópico na mesma relação, que é apresentado em destaque;
- na parte superior, uma relação de *links* para dar suporte aos itens abaixo.
  - o Manual: manual sobre o uso do sistema.
  - o Fórum: acesso a um fórum de discussões sobre o sistema. Na janela de fórum, são disponibilizados, também, acesso a *chat*, documentação e perguntas ao tutor.
  - o Tutor: envio de perguntas ao tutor.
  - o *Help*: ajuda sobre o tópico em estudo.
  - o *Model*: informações sobre o estado atual do modelo do estudante.
  - o *Options*: informações sobre como o tutor deve interagir com o estudante.
  - o *Search*: pesquisa de palavras no sistema.
  - o *Remarks*: abertura de uma janela de texto para o registro de comentários sobre o tópico em estudo.
  - o *Statistics*: estatística sobre o desempenho do estudante até o momento.
  - o Lisp: apresentação de uma janela com os conceitos de Lisp.

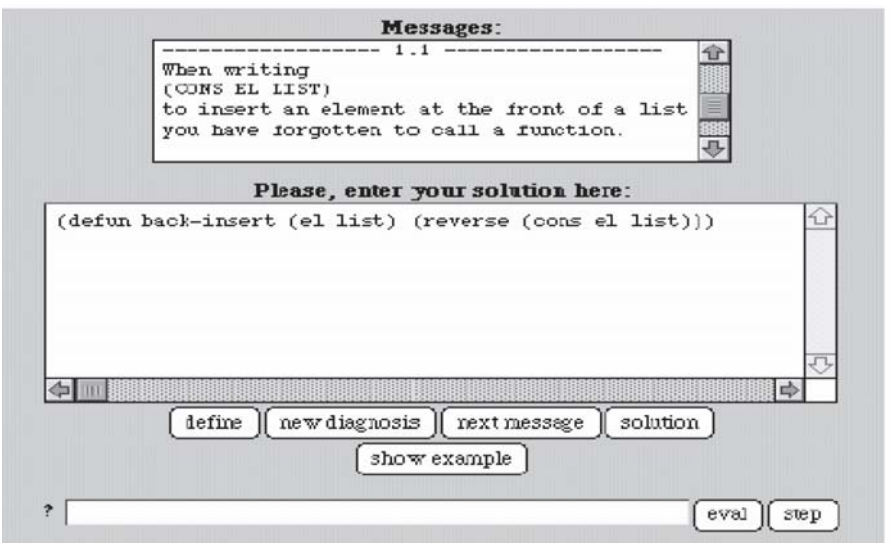

Figura 9: Tela de diálogo entre o ELM-ART e o estudante

- Na parte central, o espaço de diálogo entre o tutor e o estudante.

3.5 SQL-Tutor

O SQL-Tutor foi desenvolvido por Mitrovic (1997) para ensinar a linguagem SQL de acesso a bases de dados para estudantes universitários, utilizando a técnica da modelagem baseada em restrições, aplicada na representação do domínio do conhecimento e no modelo do estudante.

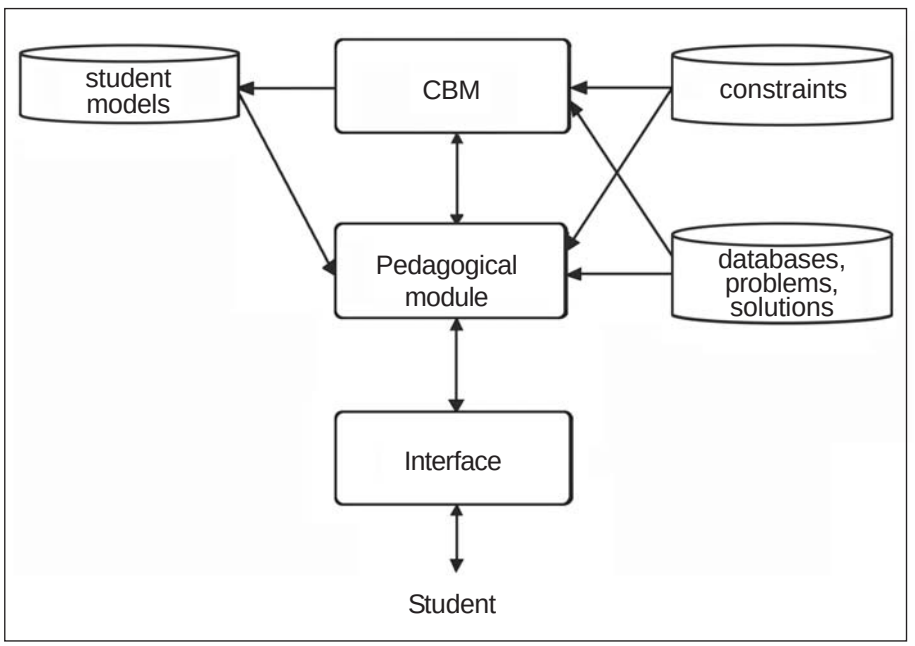

Figura 10: Arquitetura de um SQL-Tutor
Fonte: Mitrovic & Ohlsson (1999).

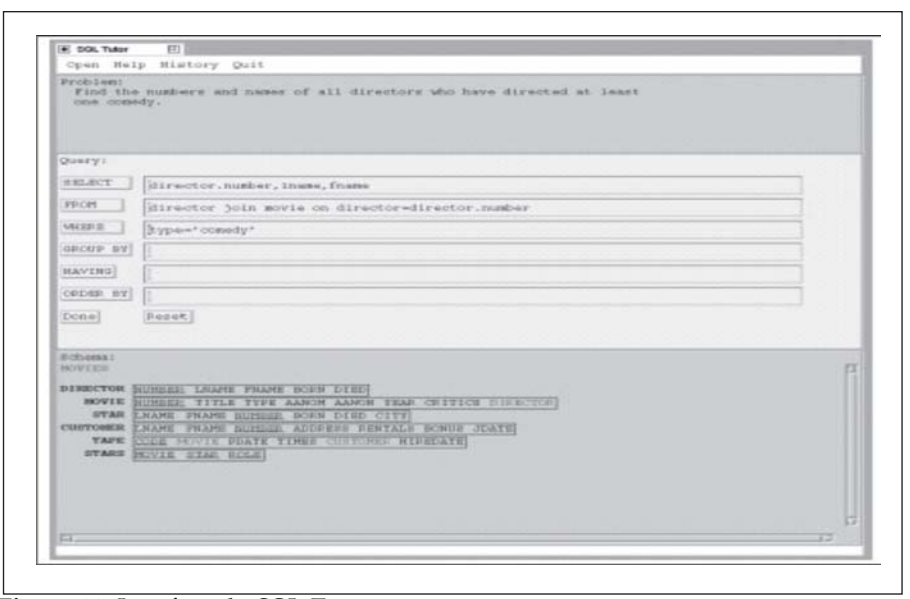

Figura 11: Interface do SQL-Tutor
Fonte: Mitrovic (1997).

O sistema é composto de uma interface, uma base de conhecimento, um modelador do estudante e um módulo pedagógico que determina o *timing* e o conteúdo das ações pedagógicas (Figura 10).

O módulo da interface solicita a identificação do estudante, empregada para localizar o modelo desse estudante. Caso seja a primeira entrada do estudante no sistema, é criado um modelo para esse estudante.

A tela principal do SQL-Tutor (Figura 11) é composta de três partes, que são visíveis durante todo o tempo que durar a interação do estudante com o sistema. Na parte superior, é apresentado o texto do problema corrente; na parte do meio, são apresentadas as cláusulas do comando *Select* e, na parte inferior, o esquema do banco de dados escolhido.

A análise da solução do problema é feita somente quando for acionada pelo estudante. Nesse momento, o módulo pedagógico envia a solução proposta pelo estudante para o modelador do estudante, que analisa a solução, identifica erros e atualiza o modelo do estudante. Caso haja erros, o módulo pedagógico apresenta a mensagem de *feedback* apropriada. A base do conhecimento é um conjunto desordenado de restrições.

O processamento de uma solução apresentada pelo estudante é feito em duas etapas (Figura 13). Na primeira etapa, a solução é verificada contra uma rede que contém as condições de relevância para o problema, sendo selecionadas apenas as condições que se apresentam como relevantes para a solução. Na segunda etapa, são selecionadas as condições de satisfação

```
(p 186
"Check the numerical constant you are using in the WHERE clause!"
(and (not (null (where ss)))
     (not (null (where is)))
     (bind-all '?n (find-names ss 'where) bindings)
     (attribute-in-db (find-schema (current-database *radni*)) '?n)
     (equalp (find-type '?n) 'numeric)
     (match '(?*d1 ?n ">" (?is ?c1 numericp) ?*d2) (where ss) bindings)
     (match '(?*d3 ?n ">=" (?is ?c2 numericp) ?*d4) (where is) bindings))
(equalp (string-to-number '?c2) (1+ (string-to-number '?c1)))
"WHERE")
```

Figura 12: Exemplo de uma restrição
Fonte: MITROVIC (1997).

que correspondem às soluções selecionadas como relevantes.

O modelo do estudante é composto das restrições que forem relevantes no processo de aprendizagem do estudante, contendo o número de vezes em que a restrição foi relevante e o número de vezes em que a restrição não foi violada.

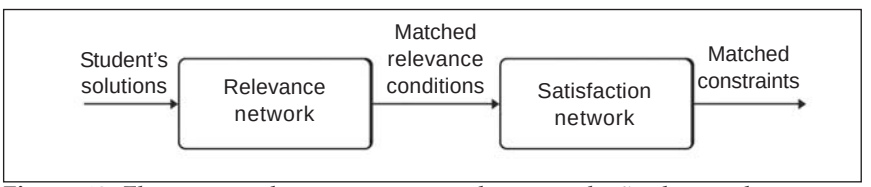

Figura 13: Fluxograma do processamento de uma solução do estudante
Fonte: MITROVIC (1997).

O módulo pedagógico gera mensagens de *feedback*, baseado no modelo do estudante, e seleciona exercícios, baseado nas restrições violadas e nas restrições cuja condição de relevância ainda não foram utilizadas pelo estudante.

3.6 AlgoLC – um sistema para o ensino e aprendizagem de algoritmos utilizando um companheiro de aprendizagem colaborativo

Petry (2005) propôs um sistema para o ensino de algoritmos, que denominou como AlgoLC, utilizando um companheiro de aprendizagem, com a aplicação da Técnica de Modelagem Baseada em Restrições.

A arquitetura do AlgoLC (Figura 14) mostra um sistema com os mesmos componentes já vistos nos anteriores, acrescido do módulo do companheiro de aprendizagem.

O modelo do domínio contém as informações corretas na forma de restrições que não podem ser violadas pelo estudante.

O modelo do estudante contém, além das suas características pessoais, as restrições violadas e não-violadas. Esses dados são utilizados pelo tutor para selecionar a seqüência de abordagem da matéria.

O modelo do tutor contém as estratégias pedagógicas que tratam do conhecimento sobre o ensino do domínio.

O companheiro de aprendizagem interage com o modelo do tutor e com o estudante, para orientar o processo de aprendizagem por meio das informações recebidas do tutor. No caso do protótipo descrito por Petry (2005), o companheiro de aprendizagem reage a cada linha do algoritmo digitada pelo estudante, apresentando a mensagem de *feedback* apropriada no caso de violação de alguma restrição. Se nenhuma restrição for violada, o companheiro de aprendizagem não apresenta qualquer reação. Na Figura 15, é demonstrado o funcionamento do sistema.

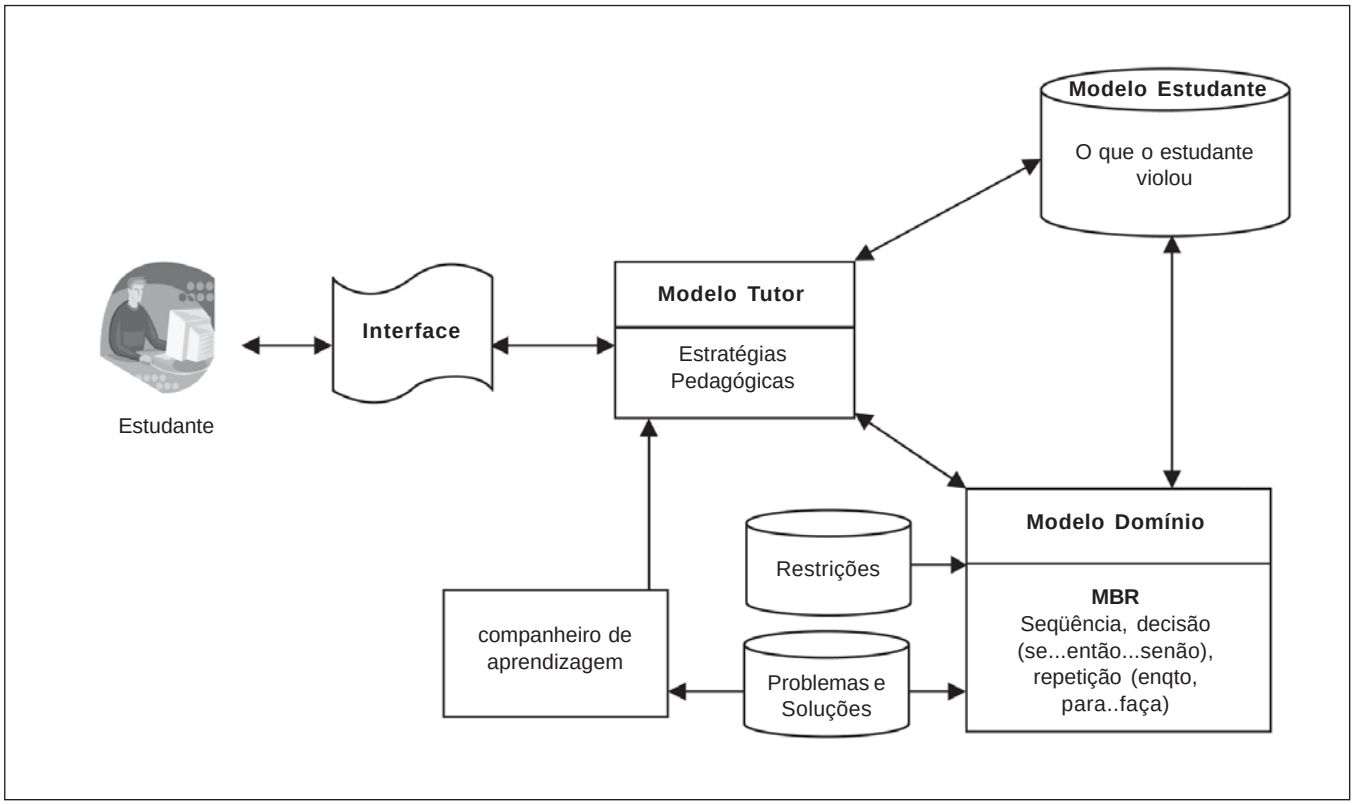

Figura 14: A estrutura do AlgoLC
Fonte: PETRY (2005).

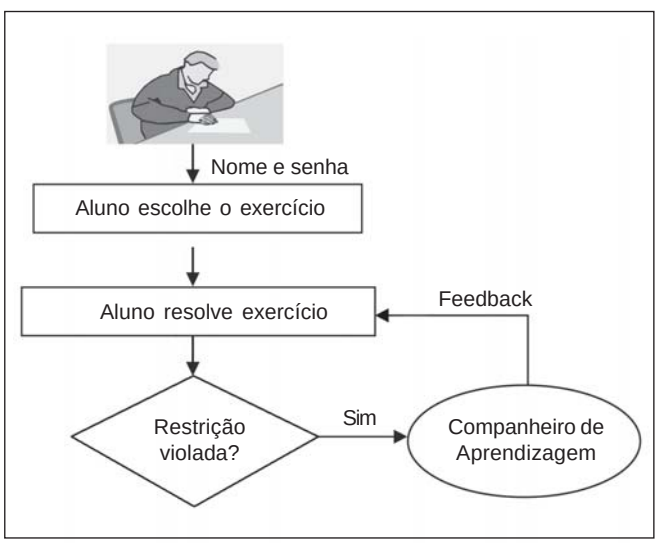

Figura 15: Fluxograma de funcionamento do AlgoLC
Fonte: PETRY (2005).

Após se identificar e escolher o processo a seguir na tela representada na Figura 16, o estudante recebe a tela de trabalho representada na Figura 17, onde escolhe o grau de dificuldade do exercício que deseja realizar. Na área "Algoritmo", o estudante desenvolve o algoritmo que resolve o exercício proposto.

A cada linha digitada, o AlgoLC verifica a violação de alguma restrição e, se houve violação, é apresentada uma mensagem de erro correspondente à restrição violada (Figura 18). Quando o exercício é completado, é apresentada uma tela (Figura 19), indicando que não há mais erros a corrigir. O sistema apresenta também relatórios demonstrando o desempenho do estudante (Figuras 20 e 21).

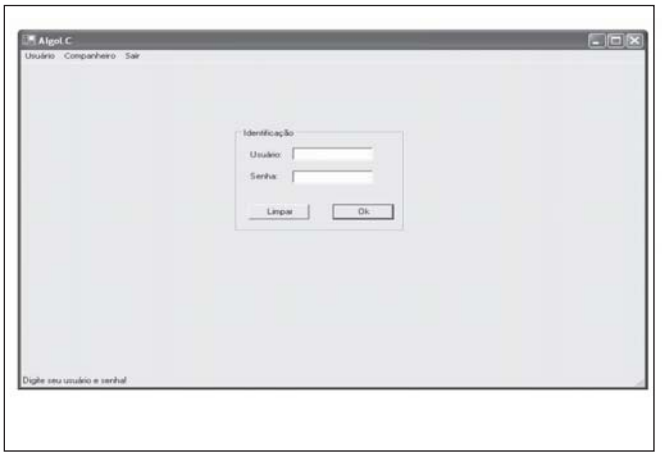

Figura 16: Tela inicial do AlgoLC
Fonte: PETRY (2005).

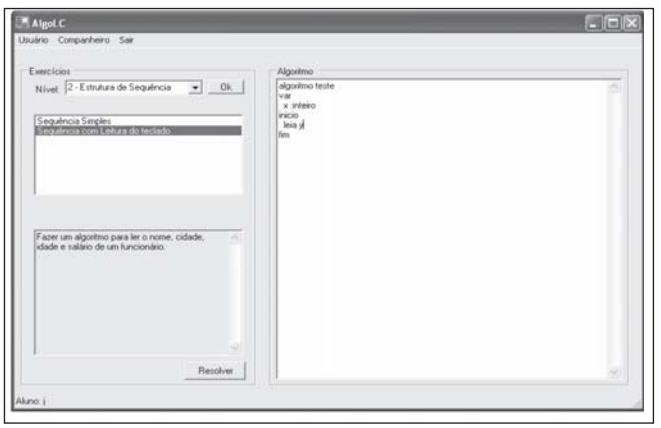

Figura 17: Tela de trabalho do AlgoLC
Fonte: PETRY (2005).

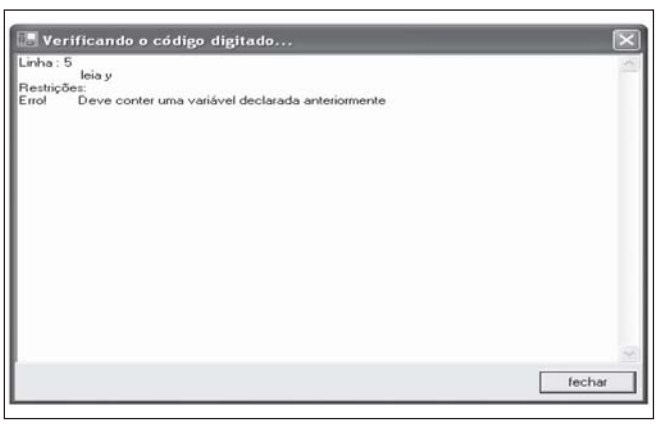

Figura 18: Tela de demonstração de violação de restrição
Fonte: PETRY (2005).

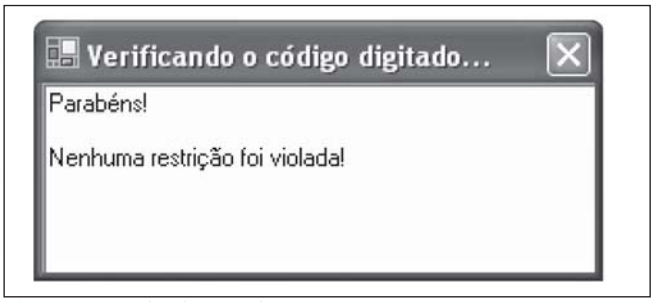

Figura 19: Tela de conclusão
Fonte: PETRY (2005).

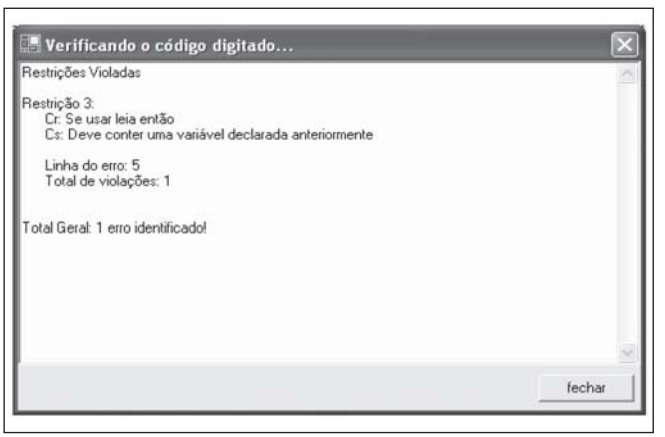

Figura 20: Relatório por estudante
Fonte: PETRY (2005).

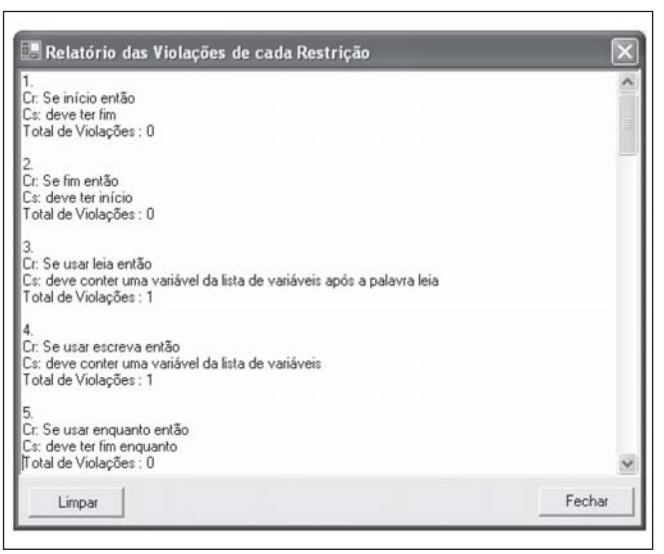

Figura 21: Relatório por restrição
Fonte: PETRY (2005).

4 CONCLUSÃO

A maior parte dos sistemas tutores inteligentes foi construída para a finalidade de pesquisa. Suas descrições são encontradas na literatura acadêmica, mas não houve seqüência na sua implementação. Assim, apenas o ELM-ART foi encontrado para testes, sendo que os demais foram abordados a partir das descrições encontradas em artigos acadêmicos dos autores e de outros autores que os usaram como referência.

Há muitas referências a sistemas que ensinam a programar, mas sem o módulo do aluno, o que os descaracteriza como sistemas tutores inteligentes. Um desses sistemas foi desenvolvido no Brasil, num projeto do Senac (ASA – Animação e Simulação de Algoritmos). No entanto, esse sistema ensina apenas alguns aspectos fundamentais da construção de programas em Pascal e o módulo do aluno contém apenas os erros localizados na análise dos programas.

Com o avanço da tecnologia voltada para o uso da *Web*, estão sendo feitas muitas pesquisas para o desenvolvimento de ferramentas de ensino a distância. Neste caso, técnicas como a de aprendizagem colaborativa, com sistemas que facilitam a utilização de *chat*, passam a ter um papel importante no processo de ensino através de computadores.

5 REFERÊNCIAS BIBLIOGRÁFICAS

ADAM, A. & LAURENT, J. Laura: a system to debug student programs. *Journal of Artificial Intelligence*, n. 15, p. 75-122, 1980.

ANDERSON, J. R.; FARRELL R. G. & SAUERS, R. Learning to plan in Lisp. *Cognitive Science*, 8, 2, 87-129, April.-June, 1984.

BRUSILOVSKY, P. L. & WEBER, E. E. G. ELM-ART: an intelligent tutoring system on World Wide Web. *In*: FRASSON, C.; GAUTHIER, G. & LESGOLD, A. (Eds.). *Intelligent tutoring systems*. Lecture notes in computer science. V. 1.086. Berlin: Springer-Verlag, 2000. p. 261-269.

BOULAY, B. du & SOTHCOTT, C. Computer teaching programming: an introductory survey on the field. *In*: LAWLER, R. W & YAZDANI, M. (Ed.) *AI and Education*: learning environments and intelligent tutoring systems. New York: Ablex Publishing, 1987.

DELGADO, K. V. 2004. *Diagnóstico baseado em modelos num sistema tutor inteligente para programação com padrões pedagógicos*. Monografia em Tópicos de Computação, Universidade de São Paulo. São Paulo: USP.

FONSECA JR., R. D. 2004. *Um modelo para sistemas inteligentes adaptativos*. Dissertação (Mestrado) – Departamento de Ciência de Computação da Universidade de Brasília. Brasília-DF: UnB.

GIRAFFA, L. M. M. 1997. *Seleção e adoção de estratégias de ensino em sistemas tutores inteligentes*. Exame de Qualificação de Doutorado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul. Porto Alegre: UFRGS.

GOULART, R. R. V. & GIRAFFA, L. M. M. Relatório Técnico nº 011/2001. Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul – PUC-RS. Porto Alegre: PUC-RS, 2001.

JOHNSON, W. L. & SOLOWAY, E. Proust: knowledge-based program understanding. Proceedings of the 7th International Conference on Software Engeneering. Orlando, Florida: IEEE Computer Society, 1984.

JONASSEN, D. H. The physics tutor: integrating hypertext and expert systems. *Journal of Educational Tecnology Systems*, v. 22(1), p. 19-28, 1993.

MITROVIC, A. *SQL-Tutor: a preliminary report*. Technical Report No. TR-COSC 08.97, Christchurch. New Zealand: Computer Science Department, University of Canterbury, 1997.

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

PETRY, P. G. 2005. *Um sistema para o ensino e aprendizagem de algoritmos utilizando um companheiro*

de aprendizagem colaborativo. Dissertação (Mestrado) – Universidade Federal de Santa Catarina. Florianópolis: UFSC.

PILLAY, N. Developing intelligent programming tutors for novice programmers. *ACM Press*, New York, NY, USA, 2003.

RAMADHAN, H. Discover: an intelligent system for discovery programming. *Journal of Cybernetics & Systems*, v. 31, p. 87-114, 2000.

RAMADHAN, H. A & SHIHAB, K. Intelligent systems for active program diagnosis. *Science & Technology*, Special Review, 2000.

SANTOS, G. dos. 2003. Autoria e interpretação tutorial de soluções alternativas para promover o ensino de programação de computadores. Dissertação (Mestrado) – Universidade Federal do Paraná. Curitiba: UFPR.

SLEEMAN, D. & BROWN, J. S. Introduction: intelligent tutoring systems. *Academic Press*, New York, 1982.

URRETAVIZCAYA, L. M. Sistemas inteligentes en el ámbito de la educación. *Revista Iberoamericana de Inteligencia Artificial*, n. 12, p. 5-12, 2001.

WENGER, E. Artificial intelligence and tutoring systems: computational and cognitive approaches to the communication of knowledge. San Francisco, CA: Morgan Kaufmann Publishers Inc., 1987.