MARCOS VINÍCIUS DE BORTOLLI

ENSINO POR TUTOR INTELIGENTE DE LINGUAGENS TEXTUAIS DE COMANDOS EM APLICAÇÕES DE SISTEMAS OPERACIONAIS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientadora: Prof. Alexandre I. Direne

CURITIBA
2001

UFPR
Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática do aluno ***Marcos Vinícius de Bortolli,*** avaliamos o trabalho intitulado ***"Ensino por Tutor Inteligente de Linguagens Textuais de Comandos em Aplicações de Sistemas Operacionais",*** cuja defesa foi realizada no dia 10 de julho de 2001, às quinze horas, no anfiteatro B do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 10 de julho de 2001.

Prof. Dr. Alexandre Ibrahim Direne
DINF/UFPR - Orientador

Profª. Dra. Maria das Graças Volpe Nunes
ICM-USP/SC

Profª. Dra. Laura Sanchez Garcia
DINF/UFPR

Agradecimentos

Agradeço às pessoas que partilharam de parte de suas vidas para que este trabalho se realizasse, a quais cito a seguir, não em ordem de importância:

À DEUS, que através de seu filho Jesus Cristo, me proporcionou a possibilidade de vitória em todas as coisas.

Ao meu orientador e amigo Alexandre, pela dedicação, competência e amizade.

Aos meus colegas de mestrado, que proporcionaram muitos momentos de companheirismo e solidariedade.

À meus pais, pelo amor e incentivo ao estudo e ao desejo de aprender.

À minha esposa Sabrina, pelo amor sacrificial em todo o tempo.

Finalmente, dedico este trabalho à minha filha ISABELA, que com seu carinho, inteligência e constante bom humor, inspirou este trabalho.

Índice geral

Lista de figuras

Lista de tabelas

Resumo

Este trabalho apresenta a concepção, projeto e implementação do sistema *ISABELA (Intelligent System for Assisting BEgginers in LAnguages),* o qual é composto de um conjunto de ferramentas utilizadas no apoio ao ensino de programação com linguagens textuais para sistemas operacionais. O ambiente oferece duas ferramentas: uma de ensino e uma de autoria. A ferramenta de ensino se propõe a cobrir a aquisição de princios e da prática no domínio de linguagens de operação de dispositivos digitais, tais como os sistemas operacionais. Os princípios são comunicados por meio de interações humano-máquina onde a máquina atua de forma passiva, sendo o aprendiz o agente responsável pela aprendizagem através da leitura de conceitos, enunciados de exemplos e a observação da dinâmica de execução dos referidos exemplos. A prática é vivenciada pela solução de problemas dados em *exercícios propostos e a implementação da solução*, os quais são avaliados e acompanhados por meio de tutoramente inteligente, de caráter reativo por parte da máquina. Este tutoramento inteligente abrange a avaliação de soluções típicas e atípicas fornecidas pelo aluno através de diagnóstico por caminhos-padrão e por análise de resultados. Finalmente, para o instrutor, o sistema oferece uma ferramenta de autoria que facilita a criação e manipulação do material expositivo e reativo destinado ao aprendiz, de acordo com o repositório de material didático da preferência do próprio instrutor.

Abstract

This work presents the conception, design and implementation of ISABELA (Intelligent System for Assisting BEgginers in LAnguages), which is composed of a set of tools to aid the teaching of computer programming with text-based languages applied to operating systems. The environment offers two tools: one for teaching/learning and one for authoring. The teaching/learning tool covers the acquisition of programming principles and programming experience through drill-and-practice examples in domains of digital device operation, such as an operating system. The principles are communicated by means of system-passive interactions where the trainee is responsible for learning through the inspection of new concepts and related commented examples, as well as through the dynamic execution of such examples. The experiential knowledge is communicated heavily by the intelligent assistance of problem solving tasks where the trainee engages in the implementation program instructions for exercises given by ISABELA. In this latter case, ISABELA evaluates the trainee's answer by means of a reactive tutoring model which embodies the idea of typical and atypical classification of problem solutions supplied by the user. ISABELA carries out the assessment through the program diagnosis according to two mains techniques: (1) standard-path matching and (2) black-box, input-output analysis. Finally, for the instructor, the system offers an authoring tool that facilitates the creation and manipulation of passive and reactive material for learning, according to the preference of the author.

1 - Introdução

1.1 - O problema principal

Um aluno de programação de computadores somente pode ser considerado apto para resolver problemas algorítmicos de maneira satisfatória quando adquire diversas habilidades que podem ser divididas em princípio e perícia (du BOULAY; SOTHCOTT, 1987). O princípio de programação diz respeito à sintaxe (como deve ser escrito) e à semântica (o que faz) isolada de cada comando. Já a perícia engloba os aspectos da lógica de programação, ou seja, as habilidades de decompor um problema e formar uma estratégia de longo prazo para a solução equivalente utilizando comandos isolados em uma determinada ordem.

No ensino de princípios da operação de dispositivos digitais, cada comando é visto isoladamente, havendo contextualização do mesmo apenas por meio de entradas individuais, sem a composição de um corpo monolítico de programa. O objetivo a ser atingido é saber como escrever corretamente, em modo interativo de interpretação, comando a comando, os termos de instruções que não admitem sub-comandos em sua estrutura.

Como a sintaxe de um comando é o primeiro grande obstáculo que um aprendiz encontra, um sistema tutorial que tenha por objetivo apoiar o ensino de programação deve se preocupar bastante com este tópico. É necessário prover um *feedback* sintático que explique melhor cada erro ou então implementar o ensino através de exemplos, até o aprendiz sentir-se confiante para escrevê-los sem ajuda.

Além do aspecto sintático, o ensino de princípios deve também ocupar-se em transmitir a semântica de cada comando. Somente sabendo com certeza o que cada comando faz é que o programador pode conseguir utilizá-los de maneira correta para implementar a solução de um problema.

Geralmente os sistemas oferecem exemplos de comandos executados, bem como um ambiente de livre exploração onde o aprendiz pode testar os comandos, verificar o resultado de cada comando até que tenha um conceito sólido e *modelos mentais do que ocorre como efeito de cada comando.*

Depois que os aprendizes sabem trabalhar com comandos isolados, é hora de aprender a colocá-los em conjunto para implementar soluções de problemas propostos. Aqui não importa somente saber o que cada comando faz, mas também como colocá-los juntos e na ordem certa para fazer o que esperamos que faça. A palavra chave portanto é ordem. Para que a solução do problema esteja correta, determinados comandos têm que ser executados em uma seqüência bem definida. Estudos empíricos apontam que os programadores iniciantes apresentam grande dificuldade em integrar os comandos e conceitos para formar um programa.

Adquirindo perícia, o programador é capaz de formular soluções mentais para problemas utilizando modelos colecionados por sua experiência. É possível abstrair a solução de um problema específico para um conjunto de sub-planos de soluções já vistas. É também importante salientar que estas soluções não dependem de sintaxe. Assim, uma vez adquirida a perícia, para que o aprendiz seja treinado em outra linguagem basta adquirir os princípios da mesma.

Assim, o aprendizado de programação de computadores passa pelo ensino de princípios e das perícias de uso dos comandos de uma linguagem textual específica. Neste processo, pode-se utilizar sistemas de diagnóstico automático de programas de alunos, os quais são uma forma de imitar o comportamento de um tutor humano. Estes sistemas analisam um programa errado e sugerem as correções ao aluno [Ramadhan; du Boulay, 1993]. Em certo sentido, o computador se torna tanto o objeto do estudo como o próprio

guia do estudo [du Boulay; Sothcott, 1987]. Desta forma, um dos principais objetivos de um sistema tutorial inteligente (ITS) voltado para o ensino de programação de computadores é apoiar o aluno durante a resolução de exercícios.

Como vimos, os princípios da programação de computadores são compostos pela sintaxe e pela semântica isolada de cada comando da linguagem específica. A perícia em programação de computadores está na capacidade de integrar comandos para resolver problemas de programação ou operação.

Neste trabalho, a ênfase principal é com o ensino de aspectos de perícia na programação de computadores. Para isso, é necessário apoiar os aprendizes na prática da solução de problemas por meio de comandos textuais para implementar operações aplicadas.

Os trabalhos anteriores próximos a este foco de problema foram muito específicos quanto à linguagem-objeto e quanto ao domínio da especialidade ensinada. Um exemplo aplicado especificamente à solução de problemas no domínio do sistema operacional de linguagem-objeto UNIX foi o protótipo RESCUER [Virvou 1991] [Virvou 1992].

Com relação a contribuições anteriores no campo de Shells com mecanismos pedagógicos genéricos, o sistema SATELIT-2 [Direne, 1997b] contemplou a construção de elementos pedagógicos essenciais para o ensino de linguagens de operação de dispositivos digitais. Entretanto, os referidos mecanismos pedagógicos só tem poder reativo com relação aos aspectos sintáticos das linguagens-objeto nos domínios específicos a serem ensinados. Os aspectos lógicos da solução de problemas de operação foram apenas estudados inicialmente mas nunca concretizados por meio da implementação de um interpretador pedagógico genérico.

Como a incorporação de aspectos genéricos aos conceitos e ferramentas requer comprovação através da aplicação dos mesmos a mais de uma linguagem-objeto em seu domínio de especialidade, utilizou-se de forma conceitual neste trabalho dois sistemas operacionais, o UNIX e o MS-DOS.

Desta forma, uma Shell de ensino alimentada com uma base de conhecimento sobre MS-DOS, deve ser capaz de produzir apoio instrucional ao aluno em cada enunciado de problema que lhe for apresentado. O comportamento desta Shell de ensino pode ser reproduzido para o UNIX (ou para qualquer outro sistema operacional), criando-se uma base de conhecimento sobre a linguagem-objeto dos comandos e seus problemas de operação no domínio específico do sistema operacional em questão. A Figura 1 mostra um exemplo do tipo de apoio que a Shell de ensino pode oferecer.

Após estes argumentos, fica claro que o problema principal neste caso é o da formulação de conceitos e construção de ferramentas para apoiar o ensino de Programação de Computadores, e que o foco mais específico é o do ensino de linguagens de comandos isolados através de Shells de ensino/aprendizagem.

Assim, conclui-se que o ensino por tutor inteligente de linguagens textuais de comandos em aplicações de sistemas operacionais cria a necessidade preemente de novos conceitos e mecanismos genéricos de interpretação pedagógica. Afim de que os objetivos deste trabalho sejam alcançados, estes mecanismos serão discorridos mais adiante.

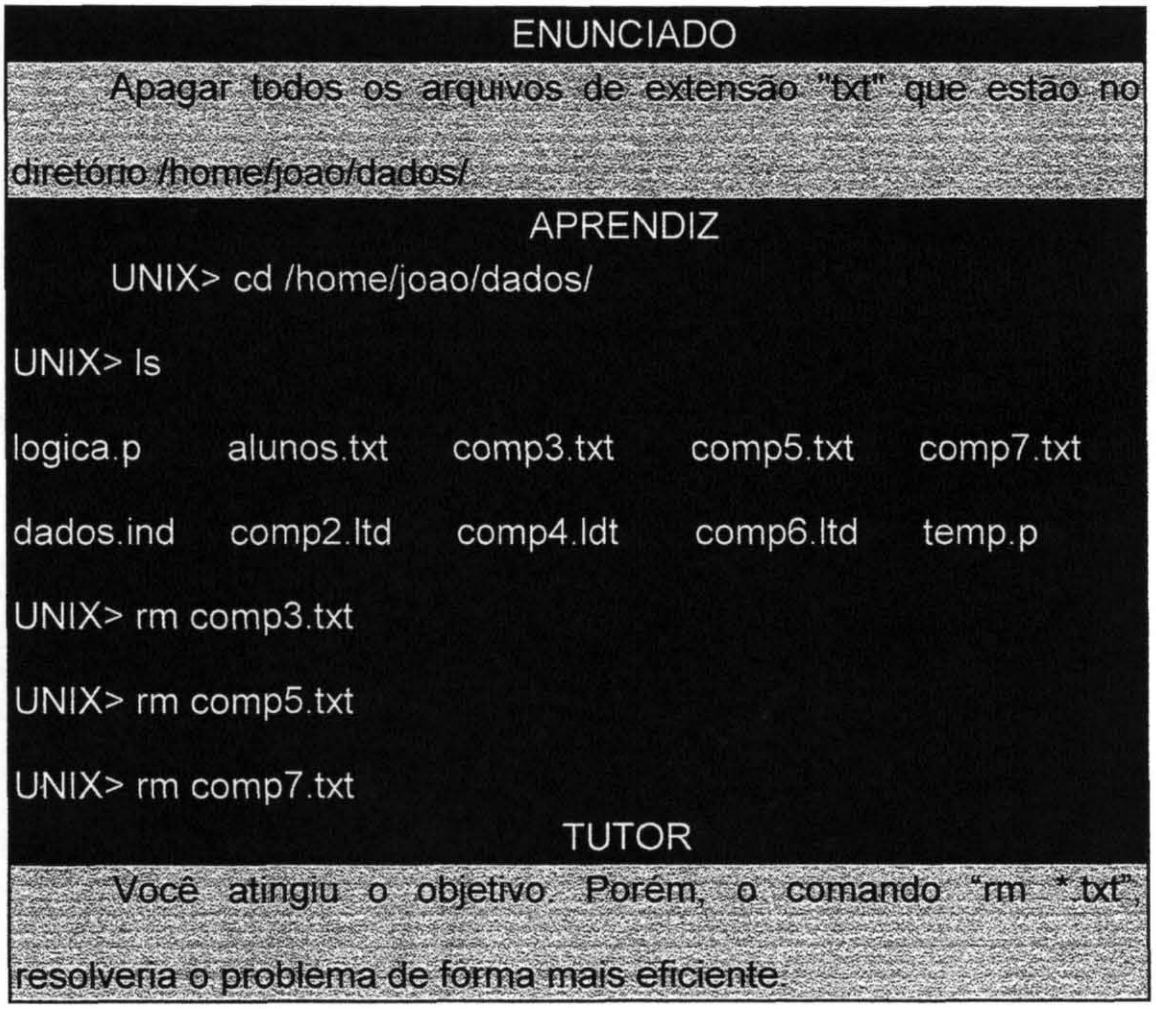

Figura 1 - Exemplo de tipo de apoio de uma Shell de Ensino

1.2 - Visão geral do sistema ISABELA

O sistema ISABELA (*Intelligent System for Assisting Begginers in LAnguages*) é um sistema que se propõe a ensinar, de forma inteligente, linguagens de sistemas operacionais textuais, como o MS-DOS, Linux, UNIX e outros. Esta inteligência está no fato de interagir com o aluno de forma a ensinar os conceitos, analisar as respostas do aluno, criticar de forma coerente as respostas e até mesmo aprender boas soluções com o aluno. Estes tópicos são supridos pela ferramenta de ensino e aprendizagem do sistema ISABELA.

Além da interação com o aluno, o sistema ISABELA se propõe a ser um instrumento eficaz nas mãos do instrutor humano, de forma a fornecer uma interface gráfica, flexibilidade no planejamento do ensino e agilidade na análise

interface gráfica, flexibilidade no planejamento do ensino e agilidade na análise dos resultados obtidos na aplicação de exercícios aos alunos. Estes últimos tópicos são supridos pela ferramenta de autoria do sistema ISABELA.

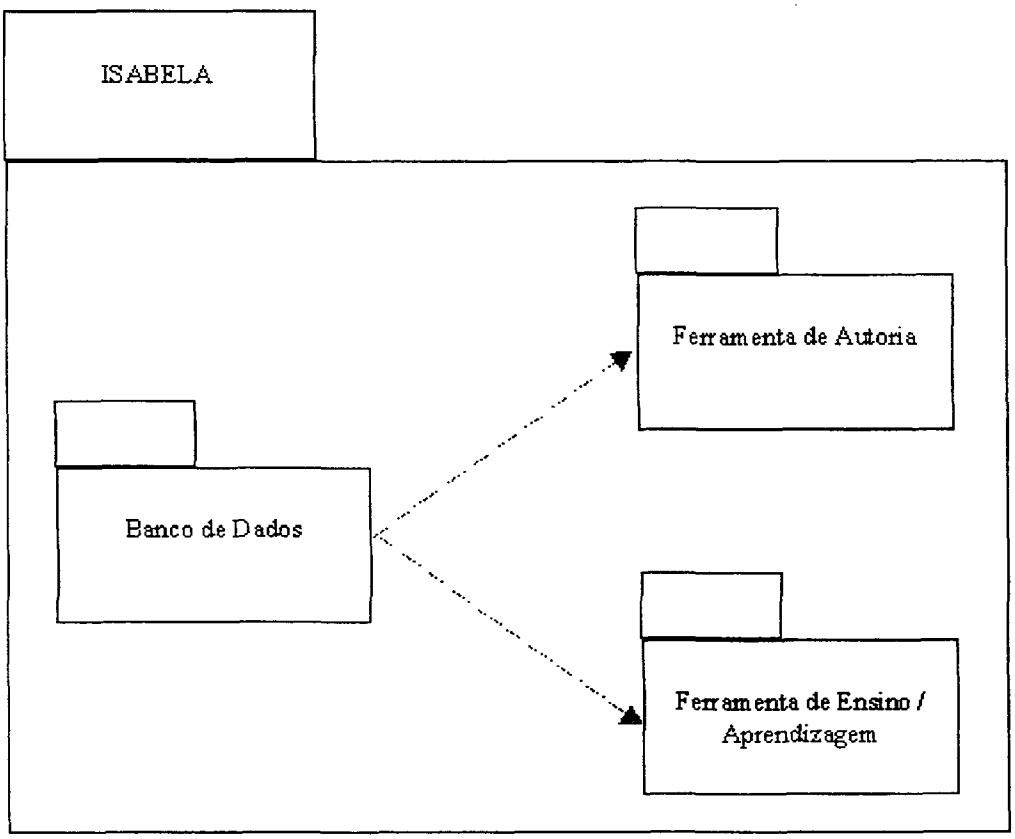

Figura 2 - Diagrama de pacotes do ISABELA

Todas as informações, tanto da ferramenta de autoria como da ferramenta de ensino e aprendizagem são armazenadas em um banco de dados relacional, o qual facilita e agiliza o acesso posterior aos dados necessários para a análise e realimentação do processo de autoria e ensino. Assim, em uma visão geral sobre o sistema ISABELA, podemos dizer que ele está dividido em 2 grandes ferramentas conectadas a um banco de dados, de acordo com a figura 2.

1.3 - Vantagens instrucionais

As características de um tutor automático que permite a exploração livre em um ambiente de descoberta guiada pode ser muito eficaz, já que reúne o auxílio à aquisição de princípio e perícia em um único sistema. Assim, como a ferramenta de ensino e aprendizagem do ISABELA é um ambiente de

descoberta guiada (*guided discovery learning environment*), o sistema oferece a vantagem de permitir que o aluno alterne a atividade de exploração livre do material de ensino com a assistência inteligente de um tutor para a solução dos problemas, onde mensagens tutoriais são fornecidas pelo diagnosticador automático caso o aprendiz cometa algum erro de lógica e/ou sintaxe.

Ao permitir atividade de autoria, o autor pode, de forma incremental progressiva, melhorar as alternativas de solução de um problema, bem como as mensagens explicativas associadas ao problema. Adicionalmente, tais alternativas podem ser originárias de boas soluções fornecidas pelos próprios alunos, as quais são armazenadas pela ferramenta de ensino e aprendizagem.

Outro fator positivo do sistema ISABELA é que mesmo as soluções erradas dos alunos são armazenadas pela ferramenta de ensino e aprendizagem com objetivo de análises futuras (como por exemplo, análises estatísticas) e de realimentar o processo de autoria e ensino baseado em tais casos de erro.

O sistema ISABELA também tem a característica de ser uma *shell de ensino*, pois permite a alteração de diversos aspectos, principalmente relacionados com o domínio que se pretende abranger. Desta forma, há possibilidade de fornecer ao aplicativo enunciados de lições e exercícios diferentes com suas respectivas soluções para serem apresentados aos alunos que utilizarem o sistema. Outro aspecto de variação da linguagem é a possibilidade de alterar o sistema operacional que está sendo ensinado, através do cadastro dos respectivos comandos textuais com suas lições e exercícios. Maiores detalhes sobre a variação da linguagem de programação e representação de soluções serão apresentados nos capítulos 3 e 4.

Apesar da tão propalada idéia da substituição do homem pela máquina, o sistema ISABELA destina-se apenas ao uso combinado do professor

humano com o software, sendo o utilizado em caráter suplementar. Isto é, o sistema ISABELA é mais um instrumento de apoio ao ensino e aprendizagem que trabalha integrado com o professor humano e não aspira de forma alguma a sua substituição.

1.4 - Estrutura da dissertação

É importante uma pequena exposição sobre a estrutura deste trabalho. Iniciando pelo capítulo 2, o mesmo procura mostrar o estado da arte que se encontra o ensino de programação apoiado pelo computador. Alguns sistemas são citados e as características que os fizeram importantes são exploradas. O capítulo se estende e faz um apanhado de "*shells*" para ensino de conceitos, ressaltando que o ensino de programação não está entre os principais domínios onde esta facilidade foi aplicada.

O capítulo 3 detalha os conceitos que definiram a construção do sistema ISABELA, e as duas ferramentas que o compõem. A primeira delas, a ferramenta de autoria *a-ISABELA,* é explicada ainda neste capítulo 3. A segunda ferramenta, denominada ferramenta de ensino *e*-ISABELA é detalhada no capítulo 4. Para um melhor entendimento, nestes dois últimos capítulos, o sistema ISABELA e seu funcionamento é explanado com muitas ilustrações e exemplos.

Finalmente, o capítulo 5 define as conclusões após a construção do protótipo do sistema ISABELA e suas possíveis futuras implementações.

2. - Trabalhos correlatos

Vários trabalhos servem como instrumento de comparação com a presente proposta, os quais podem ser subdivididos em três principais áreas afins: a) aquisição de perícia, b) Sistemas tutores para o ensino de programação de computadores e c) Ferramentas de autoria e shells de ensino.

2.1 - Aquisição de perícia

Muitas atividades humanas requerem muito treinamento para serem executadas com facilidade e precisão. Atividades complexas como a programação de computadores necessitam de muitas horas de prática, além do conhecimento sobre princípios e lógica de programação.

O ensino de domínios de habilidade prática (perícia) é feito com base em exemplos, principalmente no domínio dos conceitos visuais. Neste caso, o aprendiz já conhece bem a definição dos princípios do domínio visual e só precisa adquirir perícia por meio de classificação das imagens, através de exaustiva prática com exemplos diferentes de imagens. [Lesgold, 1989] [Lesgold, 1984].

Desta forma, há uma divisão do ensino entre princípios e prática. Em radiologia médica, uma das áreas que mais estudos tem suscitado no campo de sistemas tutores inteligentes, o conhecimento de princípios é composto de anatomia, fisiologia, geometria projetiva de radiografia e teoria médica de doenças [Lesgold, 1984]. Já a aquisição de perícia em radiologia consiste em desenvolver no aprendiz a capacidade de classificar imagens, identificar características e descrever anomalias, combinando estes conhecimentos em esquemas a serem manipulados pelo perito como se fossem uma unidade. Outros exemplos de habilidades requeridas na radiologia médica são a capacidade de visão 2D-3D, o diagnóstico diferencial, a utilização de

vocabulário técnico e a consistência de relações lógicas entre anormalidades, além de outros.

Estudos no campo das ciências cognitivas por outros autores apontam divisões semelhantes às de Lesgold. Os estudos sobre diferenças entre iniciantes e especialistas sugerem que o aprendiz passa por diversos estágios durante a aquisição de conhecimento e competência. A aquisição de conhecimento (princípio e prática) pode ser dividida em 3 fases:

a) aquisição de conhecimento declarativo (ou factual). O conhecimento inicial é codificado em nossa mente como um conjunto de fatos em uma rede semântica.

b) procedimentalização do conhecimento, onde o conhecimento é gravado na forma de rotinas ou procedimentos.

c) composição, na qual as rotinas são tornadas mais rápidas através de ligação de dois ou mais procedimentos para formar um só corpo de mesmo efeito.

A incorporação de técnicas de Inteligência Artificial em educação afim de produzir artefatos úteis educacionalmente iniciou nos anos 70. No início dos anos 80, pesquisadores apresentaram os Intelligent Tutoring Systems (ITS), advogando um paradigma aonde o computador age como um tutor de alunos, que sabe o que ensina, a quem ensina e como ensina. O projeto de tais tutores está na intersecção entre ciências da computação, educação e psicologia cognitiva, o que requer um entendimento das 3 áreas pelo pesquisador. [Nwana 1990]

Esta intersecção entre as 3 áreas se retrata no próprio projeto da interface dos ITS. Um exemplo disso pode ser o sistema RUI (Representations for Understanding Images) [Direne 1997a], o qual pode ser descrito como (1) um método e uma ferramenta de software orientados à objeto para gerenciar a

complexidade de projeto de um ITS; (2) um modelo de interpretação de diálogos independente de domínio, integrado com o método, para implementação de interações tutoriais.

O RUI possui uma *interface dual para comunicar princípios e perícia* de conceitos visuais, onde a autoria e interpretação tutorial é destinada atualmente ao ensino de conceitos visuais em domínios da Radiologia Médica.

2.2 - Sistemas tutores para o ensino de programação de computadores

Parece completamente razoável que deveríamos delegar parte do ensino de programação ao próprio computador. A máquina, em certo sentido, se tornaria tanto o objeto do estudo como o próprio guia para o estudo [du Boulay; Sothcott, 1987]. Desta forma, um dos principais objetivos de um sistema tutorial inteligente (ITS) voltado para o ensino de programação de computadores é apoiar o aluno durante a resolução de exercícios.

Para atingir tal objetivo, o sistema deve ter a capacidade de interagir com o aluno, analisando o programa, identificando erros e sugerindo novos rumos. Os sistemas tutores inteligentes foram fundamentalmente divididos em (1) ambientes livres e (2) sistemas de diagnóstico automático de programas de alunos, o qual é uma forma de imitar o comportamento de um tutor humano que analisa um programa errado e sugere as correções ao aluno [Ramadhan; du Boulay, 1993]. Isto não é simples, pois envolve diversos fatores, tais como os objetivos do problema a ser resolvido, características internas da linguagem de programação usada, erros mais comuns encontrados, uso de comandos ou estruturas de dados semelhantes e soluções corretas mas totalmente diferentes da solução tomada como referência.

Como existem diversas maneiras de se ensinar a programar, os sistemas tutores já desenvolvidos refletiram esta variedade em sua implementação. Apesar da grande variedade de métodos, podemos classificar os sistemas tutores para apoio à programação com diagnóstico automático em dois grupos: ativo e passivo. Um sistema ativo está constantemente monitorando o comportamento do aluno, fornecendo informações (feedback) quando necessário. Já o sistema de diagnóstico passivo indica que o sistema deve receber um programa completo do aluno para iniciar o processo de análise.

Atualmente, o diagnóstico ativo é a forma mais utilizada nos sistemas tutores, pois evita que o aluno se desvie demasiadamente de alguma solução correta [Binder; Direne 1999] [Pimentel; Direne 1998]. Os sistemas ativos podem ser subdivididos em: *model-tracing* e *model-answer*. Na categoria *model-tracing* enquadram-se os sistemas que possuem como principal característica o diagnóstico de programas através da utilização de regras de produção. Estes sistemas tem a capacidade de, efetivamente, entender o enunciado proposto e apresentar uma solução correta para o problema. Entre os sistemas tutores desta categoria, temos: Greaterp [du Boulay; Sothcott, 1987], um sistema tutorial para o ensino da linguagem LISP e GIL [Reiser; Kimberg; Lovett; Ranney, 1988], que acrescentou uma interface gráfica e a capacidade de visualização ao sistema Greaterp. Na segunda categoria (model-answer) temos os sistemas cujo diagnóstico é realizado com base em uma solução de referência inserida no sistema antes do início da sessão de tutoramento. Estes sistemas não entendem o enunciado do problema e não são capazes de apresentar qualquer tipo de solução, correta ou errada. Os sistemas mais significativos desta categoria são: BRIDGE [Bonar; Cunningham, 1986] para o ensino de programação Pascal, SPADE [du Boulay;

Sothcott, 1987] utilizado para o ensino de programação com a linguagem Logo; e DISCOVER [Ramadhan; du Boulay, 1993] que auxilia o ensino de programação através de uma linguagem própria, semelhante ao pseudo-código.

Como exemplos de sistemas passivos temos: Laura [Adam; Laurent, 1980], utilizado no apoio ao ensino da linguagem Fortran; Mycroft [Goldstein, 1975], que auxilia no ensino da linguagem Logo; Proust [Johnson; Soloway, 1987], sistema de apoio ao aprendizado da linguagem Pascal e Bip [Barr; Beard; Atkinson, 1976], cuja linguagem-alvo é a Basic.

Por surpreendente que seja, todos os sistemas tutores para a programação de computadores são direcionados para domínios de linguagens de programação específicas. Duas excessões são os sistemas SATELIT-1 [Direne, 1997b], e Asimov [Binder; Direne, 1999]. O SATELIT-1 é um tutor baseado em simulação o qual permite interações ativas e passivas entre o aluno e o ambiente de desenvolvimento, concentrando-se nas características sintáticas e léxicas do diálogo. A interface do SATELIT-1 consiste de 4 objetos de instrução, não exibidos ao mesmo tempo: (1) Simulador de Terminal, (2) Galeria de Exemplos, (3) Caixa de Mensagens e (4) Janela de animação. A perspectiva adotada para a engenharia do conhecimento no SATELIT-1 é orientada por parâmetros cognitivos derivados de entrevistas com instrutores e alunos. Os métodos utilizados incluíram sessões reais de treinamento nas quais foram observados diversos princípios gerais de solução de problemas.

O objetivo principal do sistema Asimov é apoiar o ensino de lógica de programação de linguagens imperativas/procedimentais. Para atingir este objetivo, procurou-se reunir diversas características positivas de sistemas já desenvolvidos em um único ambiente com características flexíveis e com independência de domínio para auxiliar alunos iniciantes na aquisição de

princípio e perícia de programação. Dentre esses aspectos, destaca-se: (1) exploração livre do ambiente de programação; (2) diagnóstico automático de programas; (3) visualização de células de memória; (4) execução compassada; (5) possibilidade de alteração de alguns parâmetros de tutoramento pelo próprio aluno; (6) facilidade na inclusão de novos problemas e mudança (desde que obedecidas determinadas restrições) da linguagem alvo do sistema.

2.3 - Sistemas de Autoria e "Shells" de Ensino

Para a construção de um tutor automático é necessário que seja disponibilizado previamente todo o conhecimento específico e características pedagógicas que serão utilizados. Ao contrário de um trabalho apoiado por um livro, o valor educacional de um tutor automático está no fato deste ser capaz de apresentar o material de ensino de forma altamente dinâmica, com base em uma gama de casos típicos e atípicos onde a variação dependerá da reação de cada aluno. A eficácia de um tutor automático depende de um acervo de classes de situações e estratégias associadas. Um tutor automático deve incorporar um subconjunto representativo destas situações para permitir que o aprendiz possa inspecionar este espaço de situações armazenadas e praticar simulações baseadas em estudos de caso.

Estas características tornam a construção de tutores automáticos cara e difícil, especialmente por incluir diversos aspectos interdisciplinares, especialmente os de carácter pedagógico. Desenvolver um ITS custa aproximadamente 50 a 500 horas de trabalho para cada 1 hora de material instrucional [Woolf & Cunningham, 1987]. Por isso se faz necessário apresentar, de forma concreta, uma generalização de seu funcionamento de forma a permitir ao professor, a inclusão de novos exemplos, enunciados de problemas e respectivas soluções.

Atualmente, o único método viável para produzir software de qualidade em grande quantidade é por meio da produção em equipe. Esta estratégia envolve um time de profissionais que individualmente desenvolvem uma tarefa segundo suas habilidades. Assim, professores cuidam da pedagogia e programadores produzem código. Infelizmente, o custo disso é proibitivo.

Portanto, há uma série de razões pelas quais os sistemas tutoriais inteligentes não tiveram grande aceitação nas salas de aula. Estas razões incluem recursos financeiros e o fato de que muitos ITS são altamente específicos sobre um domínio em particular, não permitindo aos professores reconfigurá-los para outros domínios "semelhantes" de interesse ou mesmo alterar a forma e a estratégia de apresentar o conhecimento.

Porém, com sistemas de autoria, é possível permitir que cada professor crie e altere os sistemas de ensino [Nicolson; Scott, 1986]. Nos sistemas de autoria e shells para ensino/aprendizagem, o autor pode construir sistemas completos ou então, alterar de maneira significativa o domínio do sistema atual. Os sistemas de autoria podem contribuir de diversas maneiras: a inserção de conhecimento é facilitada e eficiente, pois praticamente não exige programação e porque a representação do conhecimento é modular, podendo ser reutilizada em outros sistemas. Além disso, o autor pode verificar a validade do sistema construído através de testes imediatos. Outra vantagem do sistema de autoria é a de incluir conhecimento que o especialista no domínio normalmente não possui de maneira bem formalizada, como princípios pedagógicos.

Desta forma, construir sistemas de autoria reduz custos e pode produzir de forma automatizada material de curso para um grande grupo de autores [Murray, 1999]. Isto é possível porque a maioria dos sistemas de autoria oferecem Shells que permitem o reuso do mecanismo pedagógico essencial

que contém componentes instrucionais ajustáveis em vários níveis de complexidade.

Um exemplo disso é o sistema COCA (CO-operative Classroom Assistant), o qual contém um número de controles heurísticos configuráveis que implementam decisões a serem tomadas durante o ensino. Assim, o sistema COCA torna possível ao professor construir e reconstruir suas estratégias de ensino conforme suas necessidades [Major, 1991]. Entretanto, uma avaliação mais apurada do sistema COCA [Major, 1994] demonstrou que, apesar de fornecer considerável poder aos professores, há ainda um grande espaço entre a interface que os professores podem manipular e a interface que os sistema de autoria pode fornecer.

Para ter usabilidade no nível instrucional, o sistema de autoria deve oferecer controles instrucionais que possam ser facilmente manipulados pelo professor. Deve prover os controles críticos da estratégia de ensino sem entretanto ofertar tantas escolhas que torne a tarefa complexa demais. O sistema REDEEM [Major, 1997] se propôs a reduzir as escolhas de baixo nível do professor em troca de maior facilidade de uso. Percebe-se claramente a necessidade de se variar as diretivas pedagógicas, especialmente em domínios de aprendizagem baseados fortemente em exemplos, como é o caso da Programação de Computadores.

A demonstração de soluções pode oferecer uma saída parcial em domínios onde não há abordagem pedagógica forte. Um exemplo disso é o Demonstr8 (lê-se demonstrate) o qual é um ambiente de autoria que utiliza técnicas de autoria por demonstração no domínio da aritmética. Por meio dele, o autor pode interagir com a interface que o estudante irá usar e as produções podem ser inferidas pelo sistema [Blessing, 1997].

Apesar do constante crescimento da área e dos méritos de sistemas de autoria tão completos quanto o EON [Murray, 1998], os mesmos podem ser de difícil aplicação direta no mundo prático. Em particular, destaca-se aqui a natureza dinâmica bem particular dos programas de computador por meio dos quais as soluções de problemas neste representa um grande desafio de integração entre os modelos de domínio e pedagógico para os Shells Tutores Inteligentes a serem gerados pelas ferramentas de autoria. A isto soma-se o fato de que nenhum ambiente de autoria, dentre os mais conhecidos, foi destinado ao ensino de Programação de Computadores.

3 - Conceitos e Ferramenta para a Autoria

Em uma visão geral sobre o sistema ISABELA, podemos dizer que ele está dividido em duas ferramentas principais conectadas a um banco de dados, como apresentado anteriormente na Figura 2.

A primeira ferramenta, denominada *e-ISABELA,* é uma ferramenta de ensino e aprendizagem e se propõe a ensinar, de forma inteligente, linguagens de sistemas operacionais textuais. Pretende interagir com o aluno de forma a ensinar conceitos, analisar e comentar suas respostas, e até mesmo incorporar boas soluções fornecidas pelo aluno.

A segunda ferramenta do ISABELA, denominada *a-ISABELA,* é uma ferramenta de autoria e supre os quesitos de ser um instrumento destinado ao instrutor humano, ao fornecer uma interface gráfica, flexibilidade no planejamento do ensino e agilidade na representação das soluções possíveis para os enunciados de exercícios criados.

Por questões de ordem do texto e até mesmo de entendimento, é importante enfocar, em primeiro lugar, os aspectos de autoria do sistema ISABELA antes dos aspectos de ensino.

3.1 - Conceitos Adotados para a Autoria

Ao construir um tutor inteligente para o ensino de linguagens operacionais textuais, levou-se em conta a existência da grande variedade destas linguagens, citando como exemplo o MS-DOS, o LINUX/UNIX, OS/2, entre outras. Assim, percebeu-se a necessidade de um sistema tutor que fosse independente da linguagem de programação, podendo assim, mediante a escolha do tutor humano, ensinar ora uma linguagem operacional ora outra.

Era necessário também, que a meta-linguagem de autoria tivesse um poder expressivo suficiente para contemplar, com exatidão, os comandos das diversas linguagens-objeto operacionais. Esta meta-linguagem de autoria *também exigia um alto nível abstração na interação com o tutor humano*, de forma que este pudesse alterar, de forma prática, os conceitos, exemplos e exercícios propostos em cada linguagem de programação.

Estas considerações nortearam a construção do sistema ISABELA e mais especificamente da ferramenta de autoria *a-ISABELA*, a qual, de forma bem satisfatória, supriu as necessidades previstas em seu projeto. Passamos, então a descrever os principais elementos que formam a linguagem de autoria com mais detalhes.

3.1.1- A Linguagem de Autoria para Comandos Isolados

A linguagem de autoria do ambiente ISABELA é formada por *primitivas* que permitem a criação de 2 (dois) tipos principais de material de ensino/aprendizagem para dispositivos como sistemas operacionais. O primeiro tipo de *primitiva* de autoria se destina à criação de material com conteúdo puramente *expositivo*. Ou seja, a ferramenta de autoria permite a criação de material que será exibido ao aluno, como por exemplo, textos com ensino de conceitos e exemplos de exercícios já resolvidos. Sendo assim, além da criação dos textos que podem ser posteriormente inspecionados pelo aprendiz, a ferramenta de autoria permite a criação de exemplos dinâmicos de exercícios resolvidos, os quais também podem ser inspecionados pelo aprendiz por meio de sua execução em tempo real.

Este material *expositivo*, quando utilizado pelo estudante na ferramenta de ensino, permite que o próprio estudante esteja no controle da interação,

possibilitando a este decidir *o que* e *quanto* deseja aprender sobre determinado assunto da linguagem de programação.

Por exemplo, ao definir o material expositivo da lição 1, o tutor humano tem condições de declarar o título e o objetivo da lição e incluir quantos tópicos quiser nesta lição. Estes tópicos podem ser divididos em conceitos simples, conceitos com exemplos dinâmicos e exercícios propostos. Em cada um destes tópicos, o tutor humano pode:

- Escrever, em um editor de texto simplificado, o material expositivo dos conceitos e exercícios.
- Possíveis exemplos dinâmicos coerentes com o tópico ensinado e que serão executados em tempo real diante do aluno.
- Definir o *estado operacional* que o sistema ISABELA irá adotar, ou seja, o diretório em que atuará, os arquivos contidos ali, o formato do prompt, dentre outras variáveis de estado. Veja a tabela 5 como exemplo.

A linguagem de autoria *a-ISABELA* também permite a definição de comandos da linguagem-objeto. Este tipo de primitiva de autoria é fortemente baseado em uma linguagem de *meta-padrões* de comandos de uma linguagem-objeto, a linguagem *I-ISABELA.*

Estes *meta-padrões* são, na verdade, formas tabulares que facilitam a definição de comandos da linguagem-objeto, os quais são armazenados em tabelas relacionais para posterior utilização. Para facilitar a tarefa do autor, as primitivas que são nomes de comandos sempre são semelhantes aos comandos do sistema operacional em estudo. Estes comandos são associados à meta-linguagem *I-ISABELA,* a qual provê o correto funcionamento diante do *Simulador de Sistema Operacional* que opera no ferramenta de ensino e-ISABELA.

Comando MS-DOS	Comando *I-ISABELA*
CD	VerCaminho()
CD..	SubirNível()
CD\	IrParaRaiz()
CD Pasta	IrParaPasta()
CD\Pasta	IrParaPasta()
DIR	Listar()
REN Pasta	Renomear()
RENAME Pasta	Renomear()
DEL Pasta	Deletar()
MD Pasta	CriarPasta()
MKDIR Pasta	CriarPasta()
ERASE Pasta	Deletar()
INPUT Pasta	CriarArquivo()
DIR Pasta	Listar()
PROMPT Pasta	Prompt()

Tabela 1 - Comandos do MS-DOS e sua correspondência no I-ISABELA

Uma amostra dos comandos para o sistema operacional MS-DOS e sua respectiva correspondência na meta-linguagem *I-ISABELA* é demonstrada na Tabela 1.

3.1.2 - Linguagem de Autoria para Representar a Solução de Problemas

Ao definir os tópicos de uma lição, o tutor humano pode também estabelecer as possíveis soluções dos exercícios propostos. Isto é contemplado em uma primitiva de criação de material *reativo* para o sistema ISABELA. A organização interna das alternativas genéricas de comandos a serem fornecidos por um autor de curso para a solução de problemas também é dada por meio de uma linguagem de primitivas.

Para desenvolver tal linguagem, o primeiro passo foi analisar um grande número de soluções e identificar quais situações de variação podem ocorrer em uma solução correta. Para estas situações foram definidos *códigos de controle*, os quais compõem os comandos da meta-linguagem usada para descrever as variações nas soluções destes problemas. As principais estruturas genéricas identificadas, assim como suas descrições, podem ser encontradas na Tabela 2.

Código de controle	Estrutura	Descrição
Números seqüenciais positivos	*Comandos em seqüência*	A ordem é obrigatória. Um ou mais comandos devem ser executados em uma determinada ordem. Por exemplo, deve-se primeiro criar um diretório ("**MD Pasta**") para depois entrar nele ("**CD Pasta**"). Estes comandos executados em ordem diferente não terão o mesmo efeito
Números seqüenciais negativos	*Comandos alternativos*	Somente um deles deve ser executado. É o caso onde dois caminhos mutuamente exclusivos são possíveis para solucionar um problema. Por exemplo, para criar diretórios, pode-se usar o "**MD**" ou "**MKDIR**".
Números seqüenciais se repetem	*Comandos em qualquer ordem*	Todos comandos devem ser executados, mas a ordem não importa. Por exemplo, devo renomear três arquivos, mas a ordem em que serão renomeados não importa.
Número 0 (zero)	*Comandos opcionais*	O comando pode ou não ser executado. Como o próprio nome diz, este comando é facultativo. Por exemplo, em um exercício de renomear arquivos, um comando "**DIR**" não fará diferença no resultado, mas pode auxiliar o aluno na visualização do seu desempenho

Tabela 2 - Estruturas genéricas de soluções

Como exemplo do uso destes códigos de controle, vamos supor o seguinte exercício: Criar um diretório com o nome de ISABELA, ir para este diretório e criar um arquivo chamado TESTE.TXT e outro chamado ALEGRIA.TXT. A codificação na linguagem referente à solução correta deste exercício está descrita na Tabela 3.

Código de controle	Comando
-1	MD ISABELA
-1	MKDIR ISABELA
2	CD ISABELA
3	INPUT TESTE.TXT "OLÁ"
3	INPUT ALEGRIA.TXT "OBA !"
0	DIR
0	DIR TESTE.TXT
0	DIR *.TXT

Tabela 3 - Exemplo de uso da linguagem *I-ISABELA*

A interpretação destes comandos pelo *Analista de Soluções* seria:

- Na ordem 1, pode-se usar MD ISABELA ou MKDIR ISABELA (alternativo)
- Na ordem 2, somente o comando CD ISABELA (obrigatório seqüencial)
- Na ordem 3, pode-se usar INPUT TESTE.TXT "OLÁ" e INPUT TESTE.TXT "OLÁ", em qualquer ordem (obrigatórios em qualquer ordem), pois o

resultado será equivalente. Os dois comandos deverão ser usados para que a resposta esteja correta.

- Opcionalmente, a qualquer momento, pode-se usar um ou mais dos comandos DIR listados (*opcionais*).

O uso de tais primitivas pode ser melhor visualizado por meio de um *Grafo Direcionado Acíclico* no qual os nodos são definidos por meio dos padrões de comandos da linguagem-objeto sendo ensinada e os arcos são definidos por meio das primitivas acima. O grafo é denominado GAS (*Grafo de Alternativas de Soluções)*. A figura 3 mostra a representação gráfica destes códigos de controle.

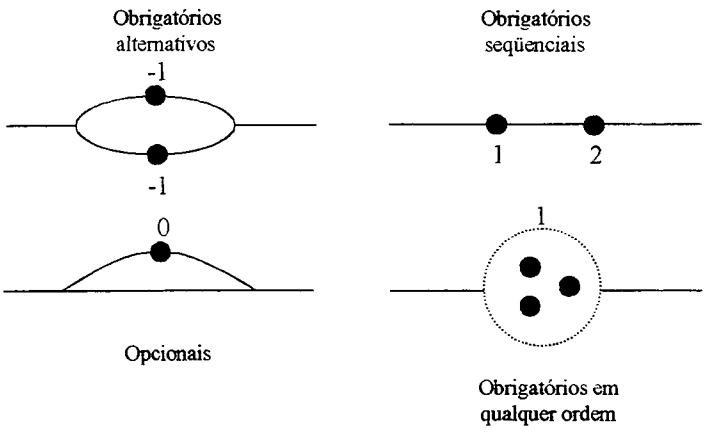

Figura 3 - Grafo de Alternativas de Soluções

Segundo este modelo de grafo, o GAS do exercício proposto anteriormente ficaria como demonstrado na figura 4.

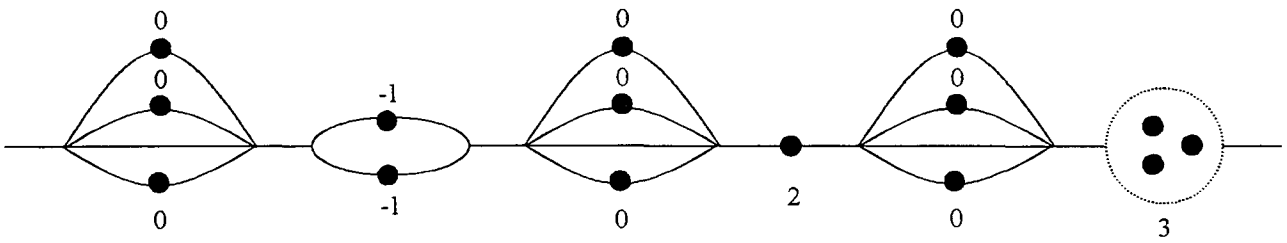

Figura 4 - Grafo GAS do exercício proposto

3.2 - A Ferramenta de Autoria a-ISABELA

A ferramenta de autoria *a-ISABELA* foi implementada utilizando-se os recursos da linguagem de programação *Visual Basic 6* e também os recursos do SGBD *MS-Access*. O motivo principal disto é a experiência prévia do autor do presente trabalho de mais de 7 anos com tais recursos como programador profissional e professor universitário. A grande quantidade existente de bibliotecas próprias e de terceiros e a facilidade e ótimo desempenho da programação visual também foram chaves para a escolha destes componentes.

3.2.1 - Arquitetura do *a-ISABELA*

A arquitetura interna da ferramenta de autoria *a-ISABELA* é composta de vários módulos, os quais são denominados *Interface, Manipulador de Material Expositivo*; *Manipulador de Material Reativo* e *Gerente de Arquivo* .

A Figura 5 descreve como estes módulos se relacionam e as próximas subseções irão descrever brevemente cada um dos módulos citados.

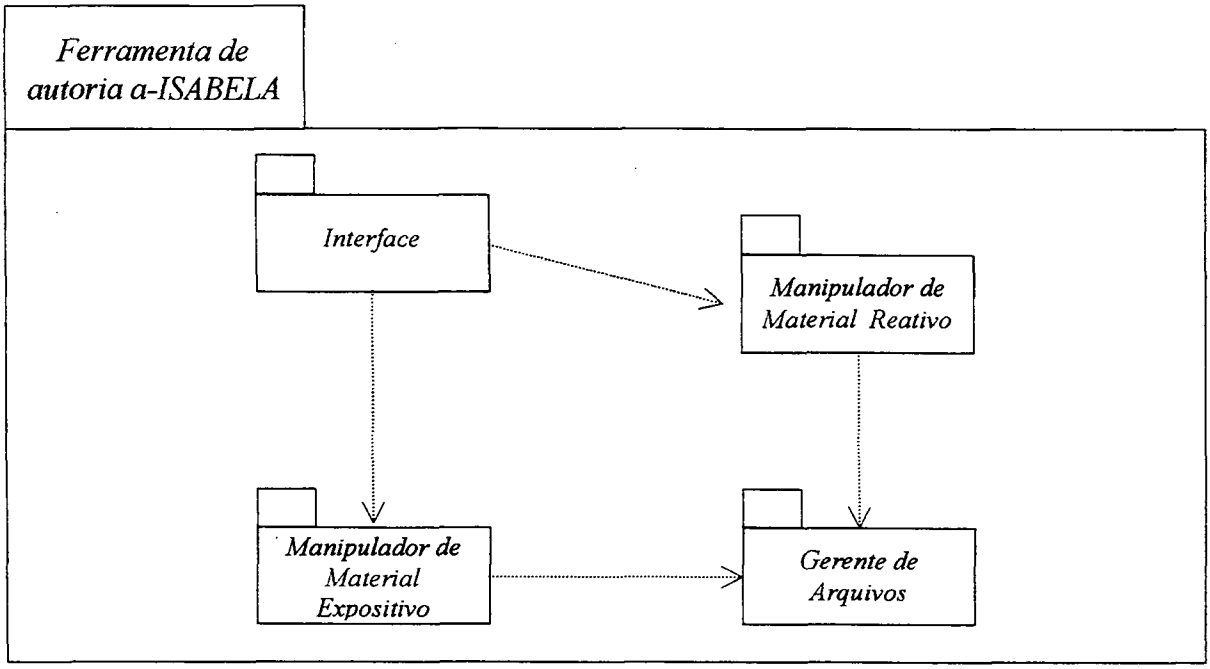

Figura 5 - A ferramenta a-ISABELA com seus módulos

3.2.2 - Interface

A interface da ferramenta de autoria *a-ISABELA* é simples e com os componentes necessários à manutenção das lições e tópicos com seus exercícios e exemplos relacionados. Possui um formulário principal, denominado *Autoria de Lições,* o qual contém:

- Um editor de texto simples, com ferramentas de edição como : tipo da fonte de letra, alinhamento de parágrafos e recursos como *Recortar / Copiar / Colar.* Este editor trabalha com padrão RTF (*Rich Text Format)*, o qual é reconhecido pelos principais editores de texto conhecidos, como o MS-WORD, StarWritter e outros. Isto significa que o autor pode utilizar, por exemplo, o MS-WORD (com muito mais recursos) para implementar os textos das suas lições, bastando associá-lo à lição desejada na ferramenta de autoria *a-ISABELA.*
- Cadastro de lições (com os campos *Lição, Título* e *Objetivo)*

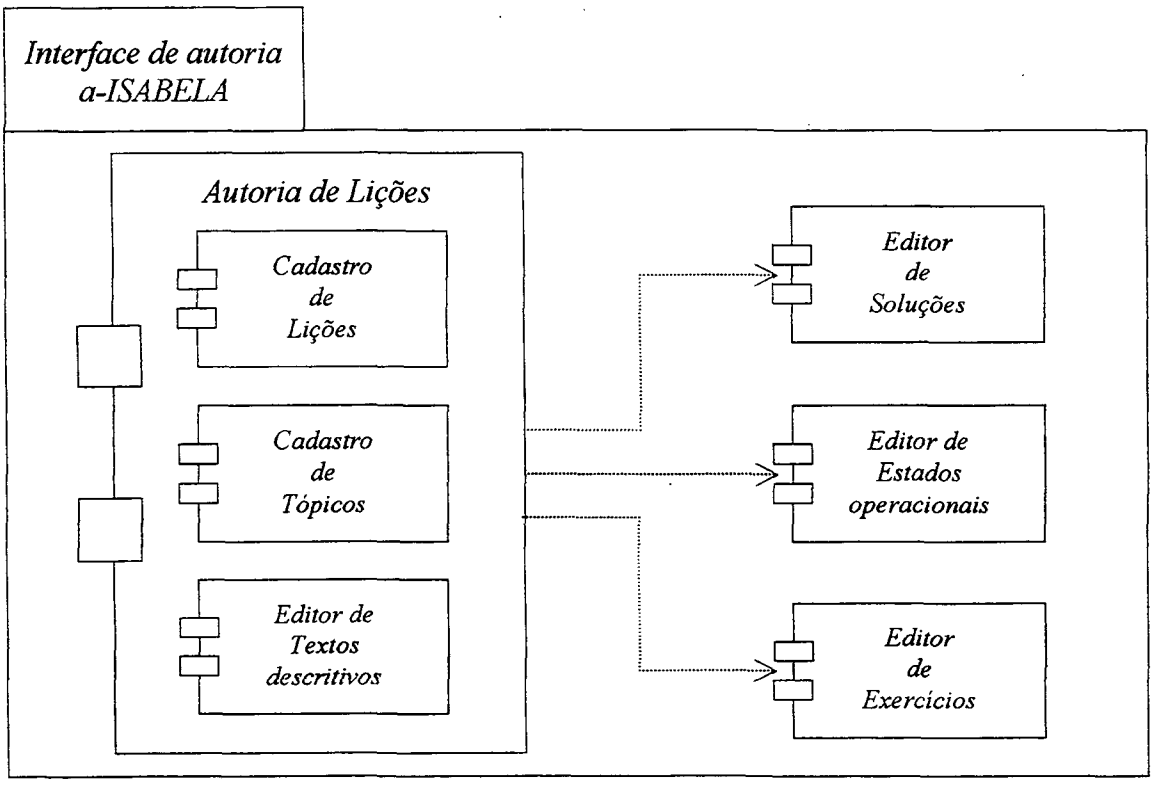

Figura.6 - Componentes da Interface da ferramenta de autoria a-ISABELA

- Cadastro de tópicos (com os campos *Arquivo, Comandos, Estado, Tópico, Tipo e Lição*)
- Acesso ao formulário de Cadastro de Soluções, o qual implementa as soluções possíveis para o *exercício* do Tópico em questão.
- Acesso ao formulário de Cadastro de Exemplos, o qual implementa um *exemplo dinâmico* para o tópico em questão.
- Acesso ao formulário de Cadastro de Estados, o qual implementa um *estado operacional* necessário para a execução do exercício ou do exemplo do tópico em questão.

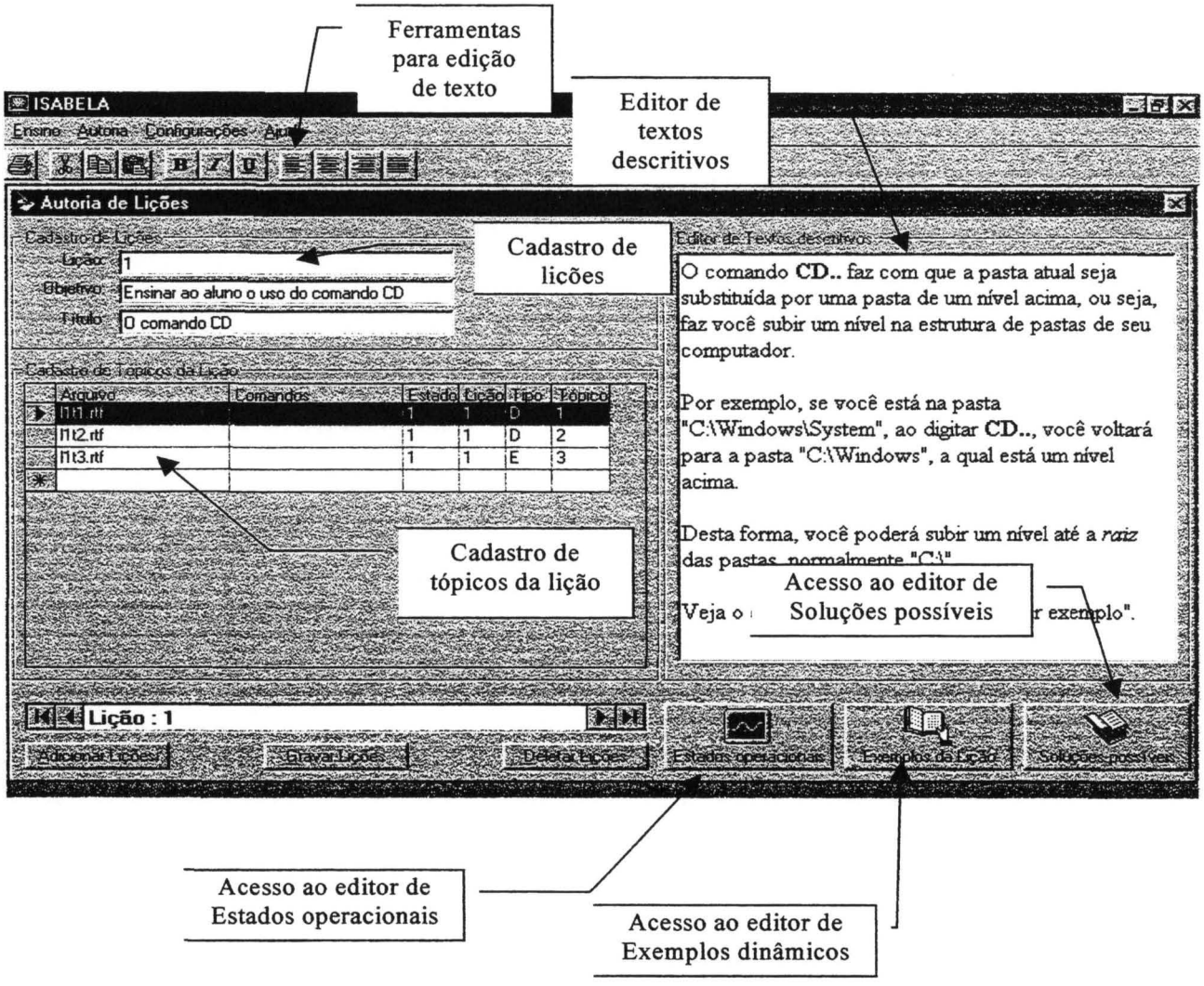

Figura 7 - Formulário *Autoria de Lições* e seus componentes

A figura 6 mostra em um diagrama de componentes como a interface da ferramenta de autoria é composta. O formulário *Autoria de Lições* da ferramenta de autoria com seus componentes é mostrado na figura 7.

3.2.3 - Manipulador de Material Expositivo

O *Manipulador de Material Expositivo* da ferramenta de autoria *a-ISABELA* é responsável, como seu próprio nome sugere, pela manipulação do material didático a ser exposto ao aluno.

Por manipulação entende-se que a ferramenta pode criar, alterar e excluir quaisquer itens referentes ao material expositivo. Por exemplo, ao definir o material expositivo da lição 1, o tutor humano tem condições de declarar o título e o objetivo da lição e incluir quantos tópicos quiser nesta lição. Nesta classe de material didático expositivo podem ser incluídos :

- *Textos descritivos*, os quais contém o ensino de conceitos e respectivos exemplos descritivos. Um exemplo disto pode ser observado no *Editor de Textos descritivos* da figura.., a qual contém o formulário *Autoria de Lições.*
- *Exemplos dinâmicos*, nos quais os comandos estudados são executados em tempo real diante do aluno no *Simulador do Sistema Operacional*, auxiliando em muito a compreensão dos conceitos ensinados, pois alia a teoria com a prática. Estes exemplos são manipulados pelo *Editor de Exemplos dinâmicos*, que utiliza a linguagem própria do sistema operacional em estudo. Assim, o autor não precisa conhecer uma meta-linguagem muito específica, bastando aplicar seus conhecimentos no sistema operacional em questão. Supondo que o conceito ensinado se refira ao comando **REN** aliado ao uso de caracteres curingas, uma aplicação possível de *exemplo dinâmico* deste comando poderia contemplar (1) um comando **DIR** inicial para visualizar o conteúdo do diretório; (2) o comando **REN** com arquivos e caracteres curinga; (3) um último comando **DIR** para visualizar o resultado. Para executar tal exemplo no ISABELA, o autor precisaria preencher uma simples tabela como a mostrada na tabela 4. Posteriormente, a ferramenta

de ensino *e-ISABELA* se responsabiliza pela exibição do exemplo no tópico especificado.

Lição	Tópico	Ordem	Comando
6	3	1	DIR
6	3	2	REN *.TXT *.BAK
6	3	3	DIR

Tabela 4 - Exemplo dinâmico em uma tabela relacional da ferramenta *a-ISABELA*

- *Enunciados de exercícios*, os quais são tarefas baseadas nos conceitos vistos e que deverão ser resolvidos por meio de comandos, e tais comandos executadas pelo aluno no *Simulador de Sistema Operacional*. Com isso, os comandos dados pelo anuno serão analisados pela ferramenta de ensino *e-ISABELA* com base no *material reativo* correspondentemente inserido pelo autor.
- *Estados Operacionais do Sistema*, que definem com precisão em que estado operacional o sistema ISABELA irá adotar a solução, ou seja, o diretório no qual atuará para executar comandos, os arquivos contidos ali, o formato do prompt, dentre outras variáveis de estado. Por exemplo, para que o *exemplo dinâmico* proposto anterior funcione, é necessário que exista no diretório de trabalho somente alguns arquivos com extensão *.TXT, os quais serão renomeados. Como o comando **REN** pode ser perigoso em algumas situações, o ISABELA prevê sua utilização somente no diretório denominado ISABELA, conforme definido pelo autor. Este estado operacional desejado poderia ser alcançado pelos comandos listados na tabela 5. Observe que os comandos são os mesmos do sistema operacional em estudo.

Estado	Ordem	Oculto	Comando
4	1	Sim	DEL C:\ISABELA*.*
4	2	Sim	INPUT C:\ISABELA\T1.TXT
4	3	Sim	INPUT C:\ISABELA\T2.TXT
4	4	Sim	INPUT C:\ISABELA\T3.TXT

Tabela 5 - Exemplo de Manipulação do Estado operacional do Sistema

Este material didático *expositivo*, quando utilizado pelo estudante na ferramenta de ensino *e-ISABELA*, permite que o próprio estudante esteja no comando da interação, com possibilidade de decidir *o que* e *quanto* deseja aprender sobre determinado assunto do domínio.

3.2.4 - Manipulador de Material Reativo

Ao definir os tópicos de uma lição, o tutor humano pode também estabelecer as possíveis soluções dos exercícios propostos. Para tanto, o autor pode se utilizar do *Manipulador de Material Reativo*.

Para entendermos melhor isto, é necessária uma breve explicação de como a ferramenta de ensino reage às respostas do aluno. A ferramenta de ensino *e-ISABELA* compara a solução do aluno com uma base pré-definida de soluções e, de acordo com as respostas do aluno, classifica cada uma e induz o aluno a um caminho diferente, fazendo-o avançar para o próximo tópico, refazer o exercício ou até mesmo rever as lições anteriores. Além disso, o ISABELA fornece mensagens específicas indicando os erros cometidos, dicas de solução e amplia o conhecimento do aluno com comandos equivalentes ou ideais.

Para contemplar esta reatividade, o *Manipulador de Material Reativo* possibilita a criação de múltiplas soluções para cada exercício, utilizando comandos que são interpretados pelo *Simulador de Sistema Operacional* (veja item 4.2.4).

Estas soluções podem ser classificadas em ideais, corretas e erradas. Para cada solução, também é possível criar mensagens específicas, além das mensagens padrões pré-existentes. Por exemplo, suponha o seguinte enunciado : "Renomear todos os arquivos com extensão **.txt** para extensão **.doc**". Como *estado operacional*, o exercício propõe uma pasta corrente

"C:\ISABELA" que contém três arquivos, a saber : **a1.txt, a2.txt, a3.txt**.

Lição	Tópico	Solução	Ordem	Comando
6	4	3	0	DIR
6	4	3	0	DIR *.TXT
6	4	3	0	DIR *.DOC
6	4	3	1	REN *.TXT *.DOC
6	4	4	-1	REN T3.TXT T3.DOC
6	4	4	-1	REN T2.TXT T2.DOC
6	4	4	-1	REN T1.TXT T1.DOC
6	4	4	0	DIR *.DOC
6	4	4	0	DIR
6	4	4	0	DIR *.TXT

Tabela 6 - Comandos dos itens de solução do exercício sobre renomear arquivo

Em um ambiente MS-DOS, a solução ideal deste exercício seria o comando "**REN *.TXT *.DOC**" ou o comando equivalente "**RENAME *.TXT *.DOC**". Porém, o aluno poderia utilizar os comandos "**REN A1.TXT A1.DOC**", "**REN A2.TXT A2.DOC**", e "**REN A3.TXT A3.DOC**", em qualquer ordem. Estes últimos comandos estão corretos, mas não são os ideais, pois são mais trabalhosos e, proporcionalmente ao número de comandos e/ou caracteres, mais propensos a erros. Esta classificação deve ser feita pelo autor e uma amostra das soluções possíveis (3 e 4) está na Tabela 6.

É necessário também classificar as soluções e indicar os *comandos chaves* necessários para a solução do problema. O campo *Tipo* determina esta classificação em (0) ideal, (1) correta e (2) errada. Observe a Tabela 7 que apresenta tal classificação pelo autor.

Solução	Mensagem	Tipo	Comando chave	Usuário
3	11	0	RENAME	0
4	13	1	REN ou RENAME aliado com caracteres curingas	0

Tabela 7 - Soluções do exercício

Nas tabelas de soluções de exercícios há sempre um campo denominado *Mensagem,* o qual permite a atribuição de uma mensagem específica para aquela solução. Por exemplo, digamos que para o exercício proposto anteriormente, o aluno responda com a opção "**REN A1.TXT**

deve reconhecer isto e indicar que (1) o aluno atingiu o objetivo do exercício; (2) que a resposta do aluno não é a ideal; (3) sugerir o comando "REN ou RENAME aliado com caracteres curingas" para uma solução ideal do exercício. Para tanto, o *Manipulador de Material Reativo* prevê a entrada de *mensagens de soluções.*

Estas mensagens também podem ser manipuladas pelo autor de forma bem simples, pois são compostas apenas textos descritivos e um meta-comando denominado **&1**, o qual é substituído pelo *comando chave* de cada solução.

A tabela 8 demonstra uma amostra das mensagens para o exercício proposto, sendo que as mensagens de número 11 e 13 foram cadastradas pelo autor e as mensagens 14 e 15 são mensagens padrões para as soluções atípicas.

Mensagem	Descrição
11	Muito bem ! Você atingiu o objetivo proposto. Lembre-se que o comando **&1** é equivalente.
13	Você atingiu o objetivo proposto. Porém, o comando **&1** seria muito mais eficiente e rápido.
14	Muito bem ! Sua resposta não está de acordo com os padrões informados, porém sua resposta está aparentemente correta. Caso confirmada posteriormente como correta, será incluída nos padrões.
15	Você atingiu o objetivo, porém de forma muito atípica. Discuta esta solução e seus possíveis desdobramentos com seu professor. Recomendo fortemente que você refaça o exercício tentando utilizar o comando **&1**.

Tabela 8 - Exemplos de mensagens

3.2.5 - Gerente de Arquivos

A ferramenta de autoria *a-ISABELA* contempla um *Gerente de Arquivos,* o qual manipula e gerencia os diversos tipos de arquivos diferentes necessários para o desempenho do sistema ISABELA. Estes arquivos contêm

basicamente os dados relacionados com o sistema operacional em estudo e a sua estruturação em lições e tópicos.

O Gerente de Arquivos possui uma estrutura simples baseada em componentes da própria linguagem de programação *Visual Basic 6,* os quais provêm o acesso a cada tipo de arquivo. Estes arquivos podem ser divididos em :

- Arquivos RTF (*Rich Text Format)*, os quais armazenam os textos descritivos dos tópicos da lição. O Gerente de arquivos acessa este tipo de arquivo através do componente *RtfText.*
- Arquivos gráficos, como ícones e bitmaps, que são ilustrativos às mensagens e diálogos que a ferramenta de ensino *e-ISABELA* utiliza. O Gerente de arquivos acessa-os através do componente *PictureBox* e através das propriedades *Picture* e *Icon* presentes na maioria dos componentes da linguagem *Visual Basic 6.*
- Arquivos do SGBD MS-Access, os quais contêm a maior parte das estruturas de autoria, armazenadas em tabelas relacionais. O acesso a estes dados é por meio da própria linguagem *Visual Basic 6*, através principalmente do componente *DataControl* e da linguagem *SQL (Structured Query Language)*, a qual é padrão de acesso em banco de dados relacionais.

Estes recursos provêm um desempenho bem satisfatório tanto na manutenção dos dados como na consulta aos mesmos. A estrutura do Gerente de Arquivos pode ser melhor visualizada na figura 8.

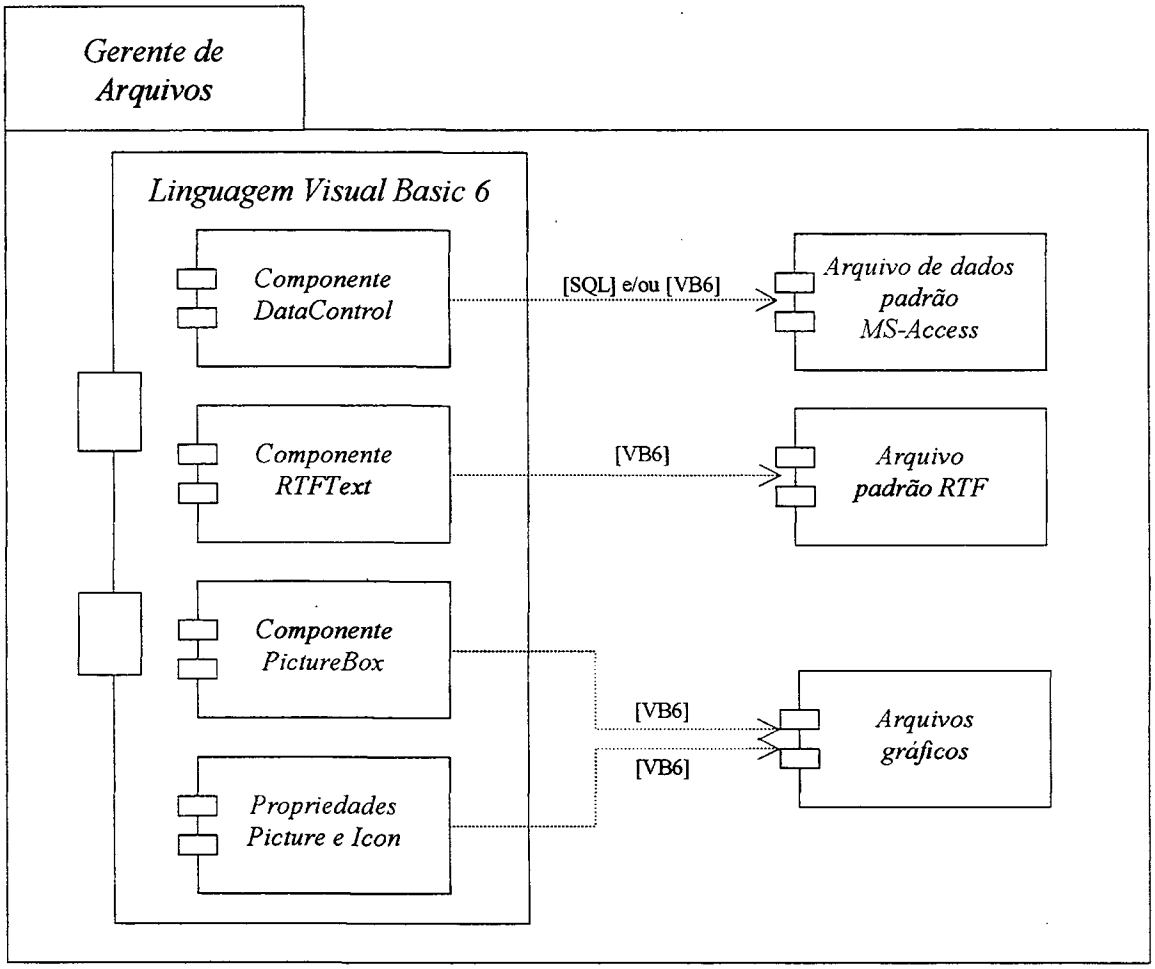

Figura 8 - Estrutura do Gerente de Arquivos

4 - Conceitos e Ferramenta para o Ensino

4.1 - Fundamentos Adotados para o Ensino

A inteligência de um tutor automatizado está no fato de interagir com o aluno de forma a ensinar os conceitos, analisar as respostas do aluno, responder de forma coerente às respostas e até mesmo aprender boas soluções com o aluno. Além disso, é importante que o aluno possa ter a liberdade de escolher o que aprender e como aprender.

Estes princípios foram a base do projeto do sistema ISABELA, os quais foram contemplados na ferramenta de ensino e aprendizagem *e-ISABELA*. Assim, o ISABELA pretende ser um sistema de ensino/aprendizagem por descoberta-guiada, que é a combinação de ferramentas tanto para a exploração livre de conceitos como para a assistência inteligente (guia) na solução de problemas.

4.1.1 - Exploração Livre de Conceitos

A aquisição de princípios de programação deve ocorrer por meio de livre exploração do material instrucional, de acordo com a vontade do aprendiz. Para tanto, a meta-organização necessária para que aprendizes de linguagens de operação e manutenção de sistemas operacionais deve conter: (1) uma interface gráfica com a exposição de conceitos e exemplos; (2) um meio pelo qual o aprendiz possa colocar em prática o que aprendeu.

Ou seja, de forma detalhada, supõe-se necessário apresentar uma lista de elementos instrucionais constituídos: (1) da apresentação de cada comando da linguagem seguido da explicação de sua função principal;(2) os detalhes das variações de sintaxe e semântica isoladas do comando; (3) exemplos de uso do comando; (4) uma fase de solução de problemas onde sistema de ensino pode então guiar o aprendiz em direção à respostas corretas, o que

será melhor explicado na seção seguinte.

Tomamos como exemplo o comando **RENAME** do MS-DOS. Ao apresentar ao aluno, é necessário:

- Dizer para que serve:

O comando RENAME vem do inglês e significa "renomear". Como o próprio nome diz, o comando altera o nome de arquivos ou pastas conforme desejo dos usuários do sistema MS-DOS.

- Dar um exemplo de uso:

É um comando útil para evitar sobreposição de arquivos em uma cópia, por exemplo.

- Utilizar um exemplo prático

Por exemplo, se estamos na pasta "C:\ISABELA" e digitamos **RENAME TESTE.TXT TESTE.BAK**, estaremos renomeando o arquivo TESTE.TXT para TESTE.BAK

- Utilizar um exemplo dinâmico

Clique no botão "Ver exemplo" e observe como funciona este comando.

- Dizer quais os comandos equivalentes

Um comando equivalente é a forma abreviada **REN**, o qual lhe dá todo o poder do comando **RENAME**. Utilize a forma abreviada para economizar digitação de caracteres.

- Dizer suas diversas sintaxes

Como outros comandos que já vimos anteriormente, o comando **RENAME** pode ser utilizado com caracteres "curingas". Assim, se você quiser renomear vários arquivos com a mesma extensão, por exemplo, você pode utilizar o caractere "*" para facilitar seu trabalho.

- Dar a possibilidade do aprendiz testar seu entendimento sobre os conceitos expostos.

Exercícios : Altere o nome do arquivo TESTE.TXT para EXEMPLO.TXT

4.1.2 - Ensino Guiado

A aquisição de perícia de programação deve ocorrer por meio do monitoramento inteligente dos passos de solução de problemas fornecidos

pelo aprendiz. Este monitoramento pode ser automatizado em boa parte por um sistema tutor que guia o aprendiz em relação à solução do problema, tanto nos erros como nos acertos em relação ao exercício.

Isto pode ser atingido dentro dos domínios de ensino de operação e manutenção de sistemas operacionais por meio do uso do grafo GAS (ver tópico 3.1.2), criado por autores de curso como representação interna das alternativas para a solução de um problema.

A manipulação de tal grafo por meio de um interpretador pedagógico pode ser atingida por meio de duas formas de tutoramento por monitoramento de erros. Da mesma forma, mensagens de erro podem ser montadas para servir de explicação para os aprendizes. As próximas duas subseções abordam estes detalhes.

4.1.2.1 - Metodologia de Verificação de Erros

O primeiro estágio na verificação de erros por meio de comparação com *caminhos padrão*. Isto é feito basicamente da seguinte forma: a ferramenta de ensino *e-ISABELA* compara a solução do aluno com uma base pré-definida de soluções. Se encontrar uma solução que se equipare à resposta do aluno, então classifica a resposta do aluno conforme o *caminho-padrão* encontrado e retorna uma mensagem coerente ao aluno.

Cada uma destas classificações induz o aluno a um caminho diferente. Como exemplo destes caminhos, pode-se fazer o aluno avançar para o próximo tópico, refazer o exercício ou até mesmo rever as lições anteriores.

Mais do simplesmente induzir o caminho do aluno, o ISABELA fornece mensagens específicas indicando erros cometidos, dicas de solução e ampliando o conhecimento do aluno com comandos equivalentes ou ideais para o referido problema.

Para detalhar isto, vamos imaginar o mesmo exemplo do tópico 3.2.4, supondo enunciado : "Renomear os arquivos com extensão **.txt** para extensão **.doc**". Como *estado operacional*, o exercício propõe uma pasta corrente "**C:\ISABELA**" que contém três arquivos, a saber : **a1.txt, a2.txt, a3.txt**.

Como vimos, as soluções ideais seriam: (1) o comando "**REN *.TXT *.DOC**"; (2) ou o comando equivalente "**RENAME *.TXT *.DOC**". Uma solução correta, porém não ideal seria utilizar os comandos "**REN A1.TXT A1.DOC**", "**REN A2.TXT A2.DOC**", e "**REN A3.TXT A3.DOC**", em qualquer ordem. Acrescentando que o aluno poderia utilizar alguns comandos **DIR** para visualizar melhor seu andamento, teríamos como *caminhos-padrão* os comandos listados nas tabelas 9, 10 e 11.

Lição	Tópico	Solução	Ordem	Comando
6	4	3	0	DIR
6	4	3	0	DIR *.TXT
6	4	3	0	DIR *.DOC
6	4	3	1	REN *.TXT *.DOC

Tabela 9 - *Caminho-padrão* da solução 3

Lição	Tópico	Solução	Ordem	Comando
6	4	4	0	DIR
6	4	4	0	DIR *.TXT
6	4	4	0	DIR *.DOC
6	4	4	1	RENAME *.TXT *.DOC

Tabela 10 - *Caminho-padrão* da solução 4

Lição	Tópico	Solução	Ordem	Comando
6	4	5	-1	REN T3.TXT T3.DOC
6	4	5	-1	REN T2.TXT T2.DOC
6	4	5	-1	REN T1.TXT T1.DOC
6	4	5	0	DIR *.DOC
6	4	5	0	DIR
6	4	5	0	DIR *.TXT

Tabela 11 - *Caminho-padrão* da solução 5

O aluno poderia utilizar qualquer uma das hipóteses acima e ela estaria correta. Porém, não há como prever em uma linguagem de sistemas operacionais toda a gama de combinações possíveis de comandos para um determinado exercício que o aluno poderia digitar. Suponha que o aluno digite

o comando "**REN T*.* T*.DOC**". É uma resposta que não tem um *caminho-padrão* na base de soluções, sendo classificada como errada. Porém, os arquivos ***.txt** foram renomeados para ***.doc**. Frente a esse caso, torna-se necessário uma análise mais aprofundada da resposta do aluno.

Para resolver isto, a ferramenta de ensino *e-ISABELA* utiliza um método CAIXA-PRETA de análise por resultado. Ou seja, após a resposta do aluno, o sistema verifica se o resultado foi alcançado verificando o *estado operacional* atual. De acordo com o enunciado do exercício, o *estado operacional* ideal para uma resposta correta é que no diretório ISABELA exista os arquivos **T1.DOC**, **T2.DOC**, **T3.DOC**. Assim, para a resposta acima, a ferramenta de ensino e-*ISABELA* utiliza uma *Tabela Relacional de Verificação de Resultados*, conforme demonstrada na tabela 12.

Lição	Tópico	Ordem	Comando	Retorno
6	4	1	CD\	C:\
6	4	2	CD\ISABELA	C:\ISABELA
6	4	3	DIR T1.DOC	T1.DOC
6	4	4	DIR T2.DOC	T2.DOC
6	4	5	DIR T3.DOC	T3.DOC

Tabela 12 - Exemplo de *Tabela Relacional de Verificação de Resultados*

Com esta tabela, o *Interpretador Genérico* (descrito no tópico 4.2.4) tem condições de avaliar o *estado operacional* atual, executando os *comandos* e comparando seu *retornos* com o desejado. No caso do exemplo, o comando dado pelo aluno seria considerado *aparentemente* correto, pois atingiu o objetivo.

Considere agora outra hipótese: (1) o aluno apaga todos os arquivos ***.txt** do diretório ISABELA; (2) de outro diretório qualquer, copia 3 quaisquer arquivos para dentro do diretório ISABELA com o nome respectivo de T1.DOC, T2.DOC e T3.DOC. Para isto o aluno utilizaria basicamente os comandos **DEL** e **COPY**. Pela análise CAIXA PRETA de resultados estaria também *aparentemente* correto. Para resolver isto, o *Interpretador Genérico* avalia

também quais os comandos utilizados na resposta do aluno e compara estatísticamente com as soluções pré-definidas da base.

Classificação	Descrição
0	Resposta correta e ideal.
1	Resposta correta, mas não ideal.
11	Aparentemente correta, com grandes chances de estar exata
12	Aparentemente correta, com grandes chances de estar errada
2	Errada

Tabela 13 - Quadro de classificação das respostas do aluno

Desta forma, o último exemplo seria interpretado como *aparentemente correto mas com grandes chances de estar errado*, pois pela estatística não utiliza nenhum dos comandos obrigatórios de uma solução tipicamente correta, que no caso seriam os comandos **REN** ou **RENAME**. Assim, de acordo com as respostas do aluno, a ferramenta de ensino *e-ISABELA* classifica cada resposta em uma das 5 categorias listadas na tabela 13.

4.1.2.2 - Metodologia de Explicação e Recuperação de Erros

Assim, de acordo com o quadro de classificação de respostas do aluno e das soluções pré-definidas da base, as mensagens fornecidas pela ferramenta de ensino poderia ser divididas em :

- *Mensagem de erros específicos da linguagem.* Há casos em que o autor de curso, seja baseada em sua experiência anterior ou nas estatísticas fornecidas pelo próprio sistema ISABELA, percebe a existência de um erro comum entre seus alunos e decide fornecer uma mensagem específica para a mesma. Por exemplo, ao perceber que seus alunos insistemente trocam as letras do comando CD por DC, o ISABELA poderia apresentar a mensagem : "Você inverteu as letras do comando CD. Verifique sua digitação...".
- *Mensagem de sub-utilização de recursos.* Este é um caso que está relacionado com a perícia do aluno. Somente com a experiência, o aluno

pode perceber que, para ir ao diretório raiz, o comando **CD** é mais eficiente que muitos **CD..** Caso utilize o comando **CD..**, a resposta é correta mas não ideal, e cabe ao ISABELA instruir o aluno com uma mensagem semelhante a: "Você atingiu o objetivo proposto. Porém, o comando CD\ é muito mais eficiente e rápido do que o uso de um ou mais CD..", para que o aluno possa melhorar seu desempenho com a utilização do comando ideal para o exercício.

- *Mensagem de equivalência*, que são utilizadas para ampliar o conhecimento do aluno com comandos equivalentes ao utilizados na resposta. Por exemplo, ao utilizar o comando **MKDIR** em uma resposta, o aluno poderá receber uma mensagem semelhante à: "Muito bem ! Você atingiu o objetivo proposto. Lembre-se que o comando **MD** é equivalente."
- *Mensagem de erros genéricos*, os quais fornecem uma mensagem padrão a respostas erradas que não estão catalogadas ainda. Assim, o ISABELA responderia com uma mensagem semelhante à: "O(s) comando(s) digitados não atingiram o objetivo proposto pelo exercício."
- *Mensagem de acerto padrão*: os quais fornecem uma mensagem padrão às respostas corretas (e ideais) que não se encaixam em nenhuma das alternativas anteriores. Esta mensagem seria semelhante à: "Muito bem ! Você atingiu o objetivo proposto pelo exercício."

Conforme descrito no tópico 4.1.2.1, o *Interpretador Genérico* utiliza uma heurística que avalia soluções atípicas por meio de um método de *análise CAIXA-PRETA de resultados,*o qual poderia ser resumido da seguinte forma:

1) *o aluno digita uma série de comandos e pede a avaliação do tutor.*

2) O tutor compara as respostas com o padrão.

3) Caso não encontre o padrão, tutor analisa o resultado.

4)Caso a análise de resultado seja positiva, o tutor avalia o quanto a resposta é correta. Como é feito esta análise:

a) O tutor compara quantos comandos do usuário encontram-se nas soluções padrões. Destes comandos, os comandos opcionais são descartados. Isto porque um simples DIR ou LS poderia somar pontos.

c) É feito uma média de comandos encontrados, com a fórmula

total = total - opcionais / obrigatórios

Se este total de comandos for maior que 50 %, então considera-se que a resposta é *aparentemente correta com grandes chances de estar exata*, apresentando uma mensagem semelhante a: "Muito bem ! Sua resposta não está de acordo com os padrões informados, porém sua resposta está aparentemente correta. Caso confirmada posteriormente como correta, será incluída nos padrões."

Caso contrário, considera-se que a resposta do aluno esta *aparentemente correta mas com grandes chances de estar errada*, ou seja, é inadequada e pode ser até mesmo perigosa, pois não contém os *comandos chave* de uma solução típica. Para avisar o aluno quanto à isso, apresenta-se uma mensagem semelhante a:"Você atingiu o objetivo, porém de forma muito atípica. Discuta esta solução e seus possíveis desdobramentos com seu professor. Recomendo fortemente que você refaça o exercício tentando utilizar o comando REN ou RENAME".

Para um aproveitamento posterior das respostas dos alunos, seja para estatísticas ou para inclusão de novas soluções, o sistema ISABELA realiza a gravação das respostas dos alunos na base de dados.

4.2 - A Ferramenta de Ensino e-ISABELA

4.2.1 - Arquitetura do e-ISABELA

A arquitetura interna da ferramenta de ensino e-ISABELA é composta de vários módulos, os quais podem ser denominados respectivamente de Interface; Apresentador de Material Expositivo; Simulador de Sistema Operacional, Interpretador Pedagógico Genérico; Gerente de Arquivos.

Estes módulos estão estritamente ligados entre si pela linguagem de programação Visual Basic e sua divisão serve para efeitos didáticos de apresentação da ferramenta de ensino. Observe a figura 9, a qual descreve a arquitetura interna da ferramenta. Para maiores detalhes, as próximas subseções irão descrever brevemente cada um dos módulos citados.

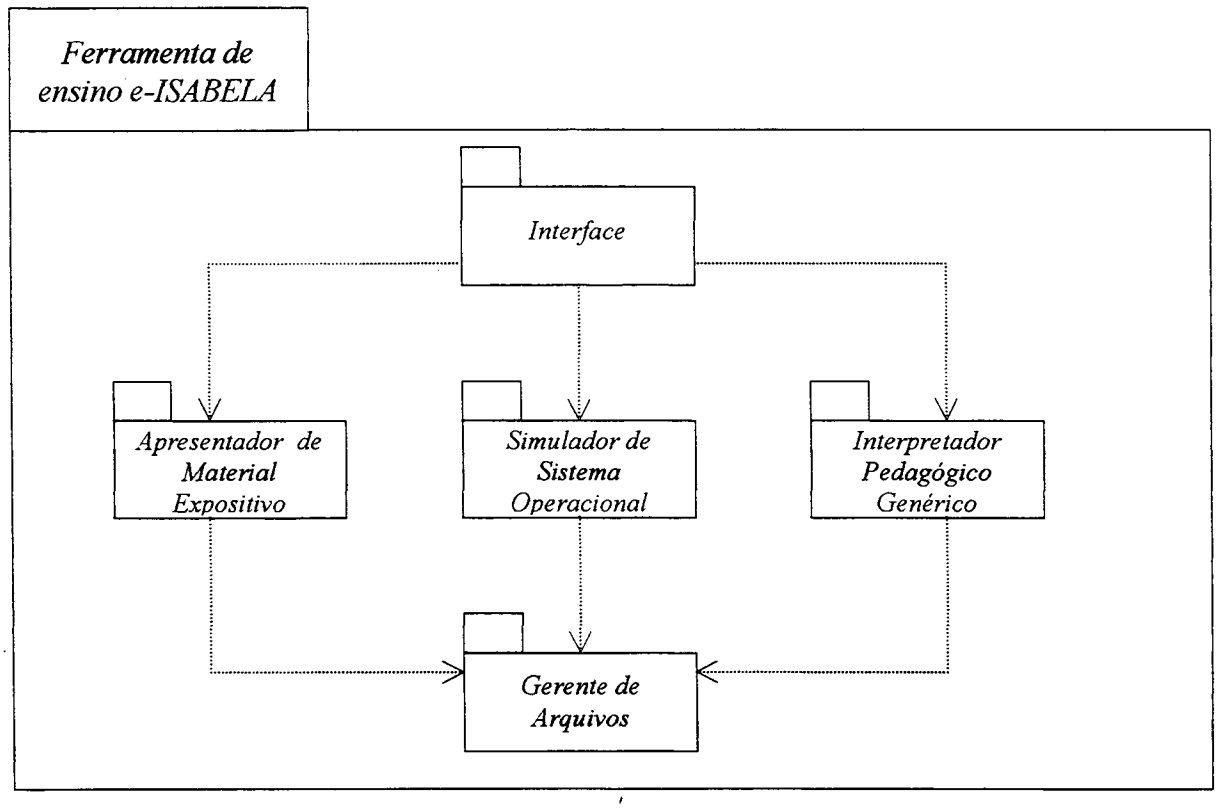

Figura 9 - Arquitetura interna da ferramenta de ensino e-ISABELA

4.2.2 - Interface

A interface da ferramenta de ensino e-ISABELA utiliza padrões visuais simplificados para comunicar-se devidamente com o aluno. A *Interface* está estritamente ligada aos módulos *Apresentador de Material Expositivo,*

Simulador de Sistema Operacional e Interpretador Pedagógico Genérico (que é interno), e utiliza como base o formulário *Tela de Lições.*

A interface da ferramenta de ensino provê algumas facilidades para o aluno, como uma barra de ferramenta para edição de texto, com recursos como *Recortar, Copiar, Colar, Imprimir.* Estes recursos foram providenciados para que o aluno possa ter a liberdade de utilizar o material expositivo em estudo fora do sistema ISABELA.

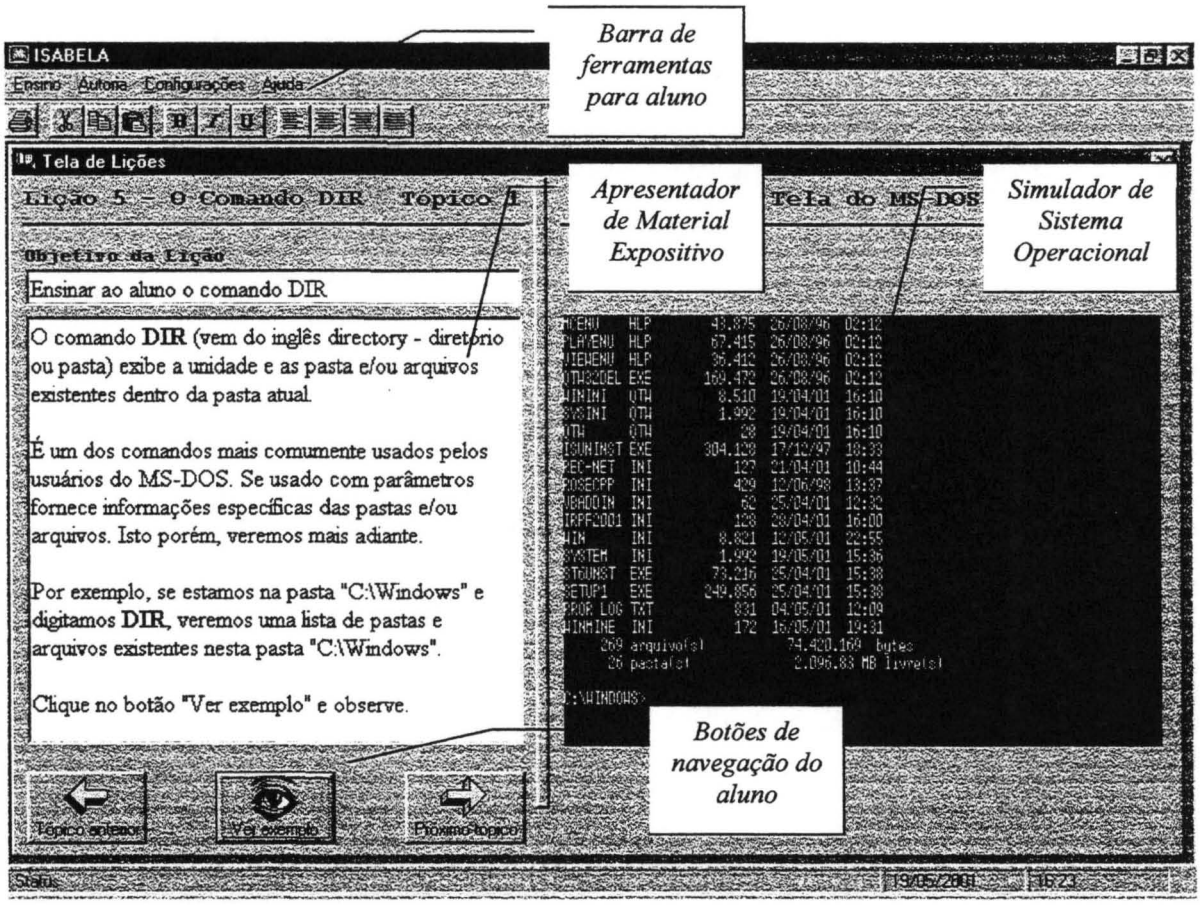

Figura 10 - Interface da ferramenta de ensino e-ISABELA.

Se o usuário do sistema ISABELA tiver o nível de autor de curso, poderá utilizar, além das facilidades do aluno, também os recursos de *Negrito, Itálico, Sublinhado* e alinhamento de parágrafos. Isto para facilitar a alteração textos descritivos dos tópicos pelo autor. Veja ferramenta de autoria no capítulo 3.

Para a navegação entres os tópicos de lição (*Apresentador de Material Expositivo*) e para visualização dos *exemplos dinâmicos*, o aluno possui alguns

botões de navegação na parte inferior da tela. Observe na figura 10 que há um *exemplo dinâmico* em execução no *Simulador de Sistema Operacional.*

4.2.3 - Apresentador de Material Expositivo

O *Apresentador de Material Expositivo* é um controlador genérico de material de curso, principalmente quando este material é de caráter puramente expositivo.

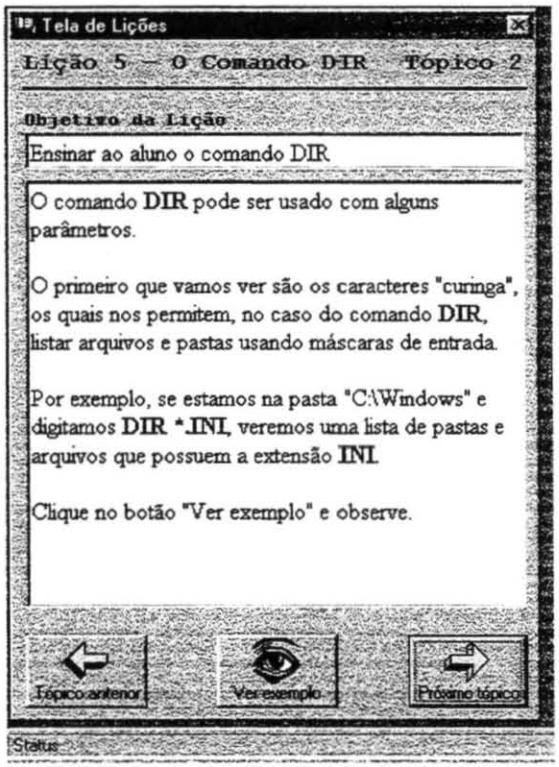

Figura 11 - O Apresentador de Material Expositivo

No *Apresentador de Material Expositivo* pode-se visualizar (1) qual a lição (número e título); (2) qual o tópico (número e objetivo); (3) e certamente os *textos descritivos* dos conceitos, exemplos estáticos e enunciados de exercícios da lição atualmente em estudo pelo aluno. A figura 11 pode prover um melhor entendimento desta ferramenta.

4.2.4 - Simulador do Sistema Operacional

O *Simulador do Sistema Operacional* presta-se à, como seu próprio nome sugere, simular de forma mais real possível o funcionamento do sistema operacional em estudo. Através dele, o aluno pode observar os *exemplos*

dinâmicos sugeridos e pode praticar em exercícios os comandos aprendidos através do material expositivo.

O *Simulador do Sistema Operacional* utiliza os comandos da meta-linguagem *I-ISABELA* para:

- Disponibilizar um *estado operacional* apropriado ao objetivo de ensino. Por exemplo, para um determinado exercício de renomear arquivos, o *Simulador do Sistema Operacional* deve providenciar o diretório e os arquivos necessários ao exercício. Veja o tópico 3.2.3.
- Reagir adequadamente aos comandos digitados pelo aluno. Isto possibilita uma interação operacional bem satisfatória com o aluno. Por exemplo, caso o aluno digite um comando sintaticamente errado, como por exemplo **DRI** em vez de DIR, o Simulador do Sistema Operacional deve imediatamente perceber o erro e apresentar uma mensagem semelhante à: "Comando ou nome de arquivo inválido". E obviamente, ao se digitar o comando correto **DIR**, deve listar todos os arquivos e pastas do diretório atual. Todos estes comandos serão avaliados posteriormente pelo *Interpretador Pedagógico Genérico.*

O *Simulador do Sistema Operacional* provê ainda um *prompt de comando* para a entrada de comandos pelo aluno e um botão denominado *Avaliar exercício* que aciona o *Interpretador Pedagógico Genérico* (veja tópico 4.2.5). A figura 12 pode fornecer um melhor entendimento do *Simulador do Sistema Operacional.*

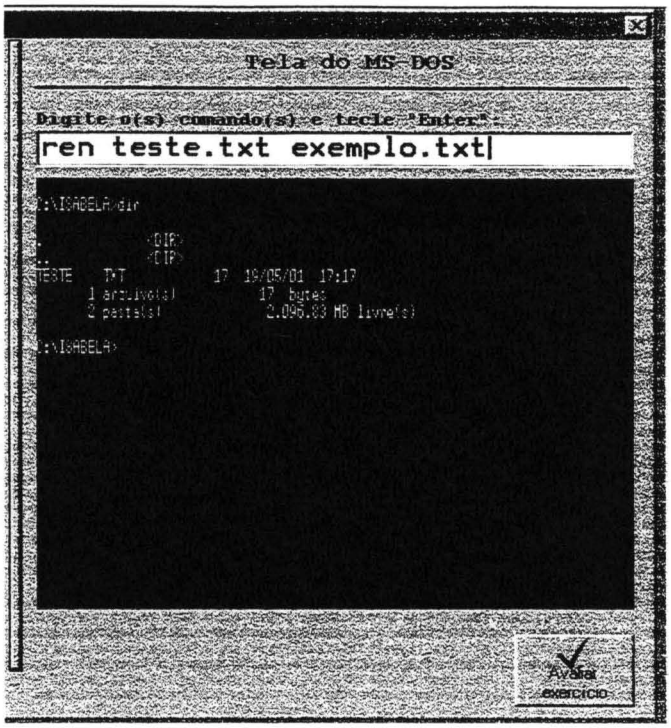

Figura 12 - Simulador do Sistema Operacional

4.2.5 - Interpretador Pedagógico Genérico

O *Interpretador Pedagógico Genérico* é o monitorador inteligente dos erros e acertos dos aprendizes. Após ser acionado pelo *Simulador do Sistema Operacional,* é capaz de avaliar as respostas dos alunos, fornecidas comando-a-comando, e verificar se estão corretas ou incorretas, com base na metodologia de verificação e montagem de explicações, descrita nas seções 4.1.2.1e 4.1.2.2 acima.

Para entender melhor o seu funcionamento, verifique o diagrama de atividades do *Interpretador Pedagógico Genérico* demonstrado na Figura 13. Após a devida verificação da resposta do aluno, o *Interpretador Pedagógico Genérico* apresenta uma mensagem coerente com o *quadro de classificação* (veja tópico 4.1.2.1) da resposta. Um exemplo de mensagem *aparentemente correta* está representada na Figura 14.

Após esta mensagem, o *Interpretador Pedagógico Genérico* apresenta, conforme o caso, algumas opções de ação ao aluno como, por exemplo, *Repetir exercício, Próximo Tópico* ou *Rever Lições.* Após a escolha da ação

desejada pelo aluno, o *Interpretador Pedagógico Genérico* aciona o *Gerente de Arquivos* para a gravação da resposta do aluno para posterior acompanhamento.

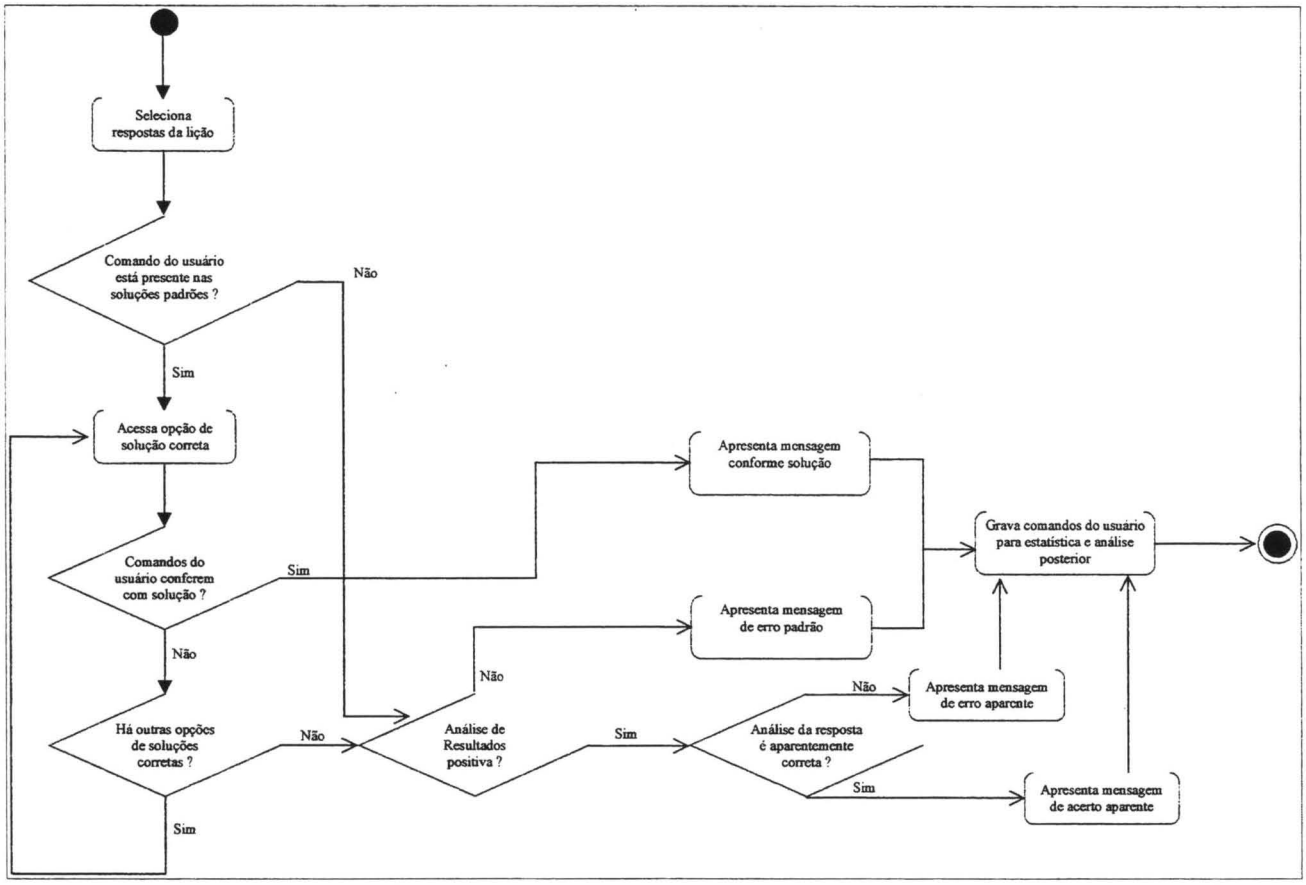

Figura 13 - Diagrama de atividades do Interpretador Pedagógico Genérico

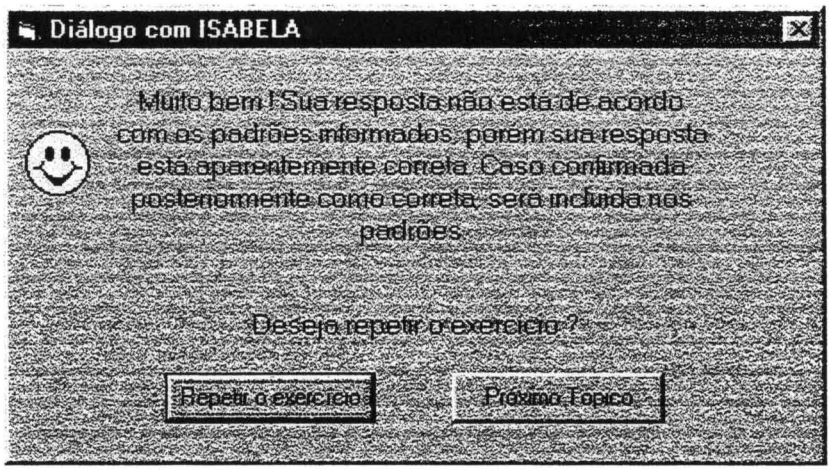

Figura 14 - Exemplo de mensagem aparentemente correta

Ressalta-se que os mecanismos pedagógicos do sistema ISABELA são genéricos somente para um subconjunto dos aspectos das linguagens de programação representáveis com as próprias ferramentas de autoria do ISABELA.

4.2.6 - Gerente de Arquivos

O Gerente de Arquivos da ferramenta *e-ISABELA* realiza operações semelhantes às do Gerente de Arquivos da ferramenta a-ISABELA, descrito na seção 3.2.5, *tornando-se desnecessário novas explicações.*

5 - Conclusão e Trabalhos Futuros

No desenvolvimento do sistema *ISABELA* procurou-se reunir diversas características positivas de sistemas já desenvolvidos para auxiliar alunos na aquisição de princípio e perícia de programação.

Apesar de que nenhuma avaliação formal do sistema ISABELA foi feita com autores de curso nem com aprendizes de linguagens de sistemas operacionais, alguns pontos merecem destaque em relação ao que foi desenvolvido e alcançado no ISABELA. Estes pontos são: o diagnóstico automático de soluções, a independência da linguagem de programação, um ambiente de descoberta guiada e uma ferramenta de autoria.

Um instrutor, ao ensinar programação, geralmente considera dois aspectos relevantes com relação às respostas para problemas que esperam dos alunos: não existe apenas uma solução correta e nem todas as soluções corretas podem ser aceitas como respostas para o problema. O sistema *ISABELA* atende a ambas as expectativas através do diagnóstico automático de soluções através de *caminhos-padrão* e por *análise CAIXA-PRETA de resultados*.

A independência da linguagem de programação permite que o professor altere o conteúdo do sistema, ou seja, a linguagem-alvo e os enunciados dos problemas. Esta característica de mudança de conteúdo torna o sistema uma *shell* para a montagem de sessões de tutoramento com linguagens diferentes. Turmas com diferentes graus de experiência e até mesmo com diferentes linguagens operacionais como alvo de estudo podem se beneficiar das características tutoriais do sistema, sem que exista a necessidade de reprogramação a cada nova turma ou a cada nova mudança.

O ambiente de descoberta guiada oferece uma rica oportunidade para aprendizagem, pois engloba a exploração livre e a programação guiada. A

ligação entre estas características é muito eficaz para o aprendizado, já que reúne o auxílio à aquisição de princípio e perícia em um único sistema.

A ferramenta de autoria *a-ISABELA* permite ao autor criar e manipular o material expositivo e reativo para seu aprendiz, de forma prática e eficaz.

O sistema ISABELA também possui conceitos originais e um grande potencial para o uso prático de ferramentas de software a serem implementadas. A sua originalidade advêm principalmente da inexistência de ambientes de autoria genéricos na área de ensino por tutor inteligente de linguagens textuais de comandos em aplicações de sistemas operacionais. Além disso, o diagnóstico por *caminhos-padrão* e por análise CAIXA-PRETA de resultados contribuem para ampliar o leque dos métodos de diagnóstico no ensino de linguagens operacionais por tutores inteligentes.

Na ferramenta de ensino *e-ISABELA,* a utilização de exemplos dinâmicos contribuem significamente para aumentar a percepção dos conceitos em foco na lição e a utilização de exercícios acompanhados de tutoramento inteligente reforçam com a prática o aprendizado do aluno. Já na ferramenta de autoria *a-ISABELA*, a utilização de tabelas relacionais aliadas à meta-linguagem *L-ISABELA* provê uma facilidade para autores mais leigos em programação.

Como metas futuras de pesquisa a partir deste trabalho, pode-se citar a criação de mecanismos que permitam a modelagem do aprendiz de forma a acompanhá-lo em sua evolução de novato para especialista. Com isso, a própria ferramenta de ensino poderá também se tornar mais ativa, decidindo sobre o material mais adequado para um aprendiz específico.

Além disso, propõe-se mais duas linhas de investigação. A primeira está relacionada com a metodologia e a ferramenta de ensino/aprendizagem. Propõe-se um estudo mais aprofundado de como estudantes aprendem

diferentes linguagens de sistemas operacionais e porque eles cometem erros. Além disso, também devem ser analisados os sucessos atípicos na solução de problemas. A partir destas soluções corretas e atípicas, espera-se incluir no *Interpretador Pedagógico Genérico* mais um mecanismo genérico de identificação e geração de explicações.

A segunda linha de investigação está relacionada com os conceitos e a linguagem de autoria. Uma expansão da linguagem de autoria deveria incluir principalmente recursos para definir *contra-exemplos* de soluções para um problema. Tais *contra-exemplos* expressam erros tipicamente cometidos por estudantes como *caminhos-padrão.* Isto resultaria na criação de "catálogos de erros" (*bug catalogues*), definidos de acordo com os estados operacionais para os comandos envolvidos.

Estes catálogos de erros contribuirão para completar o valor pedagógico do sistema *ISABELA,* além de estar de acordo com as tendências de pesquisa na comunidade científica que investiga o ensino de linguagens de programação auxiliado por computador.

Finalmente, em termos de apresentação gráfica, pode-se implementar a ferramenta de autoria com detalhamento dos dados necessários campo-a-campo a fim de facilitar a manutenção pelo autor. Na ferramenta de ensino, uma ampliação dos potenciais da estrutura de apresentação de lições do sistema ISABELA que permita uma navegação mais abrangente pelo estudante, certamente contribuirá para ampliar o ambiente de exploração livre do sistema.

6 - Referências Bibliográficas

[Adam; Laurent, 1980]

ADAM, A. & LAURENT, J. (1980). **LAURA, A system to debug student programs**. *Artificial Intelligence, 15*, 75-122

[Barr; Beard; Atkinson, R. C., 1976]

BARR, A., BEARD, M. & ATKINSON, R. C. (1976) **The computer as a tutorial laboratory: the Stanford BIP Project**. *International Journal of Man-Machine Studies,* 8, 567-596

[Binder; Direne, 1999]

BINDER, F.V., & DIRENE, A. I., (1999) **Conceitos e ferramentas para apoiar o ensino de lógica de programação imperativa**. *Anais do X Simpósio Brasileiro de Informática na Educação - SBIE99*, Curitiba - PR, páginas 145-152

[Blessing, 1997]

BLESSING, S. B., **A programming by demonstration authoring tool for model-tracing tutors**. *International Journal of Artificial Intelligence in Education*, (8):223-261

[Bonar; Cunningham, 1985]

BONAR, J. R.; CUNNINGHAM, R.(1985) *Bridge : tutoring the programming process*. p. 409-434.

[Direne 1997a]

DIRENE, A. I. (1997) **Authoring intelligent systems for teaching visual concepts**. *International Journal of Artificial Intelligence in Education (IJAIED),* Volume 8(1), pp 44-70

[Direne 1997b]

DIRENE, A. I. (1997) **Intelligent Training Shells for the Operation of Digital Telephony Stations**. *Anais da World Conference on Artificial Intelligence and Education* AIED'97. Kobe - Japan, pp 71-78

[du Boulay; Sothcott, 1987]

du BOULAY, B.; SOTHCOTT, C. (1987) **Computers teaching programming: an introdutory survey of the field**. In: LAWLER, R. W.; YAZDANI, M. *Artificial intelligence and education: learning environment and tutoring systems*. v. 1, cap. 16, p. 345-372.

[Goldstein, 1975]

GOLDSTEIN, I. P. (1975) **Summary of Microft: a system for understanding simple picture programs**. *Artificial Intelligence*, 6, 249-288

[Johnson; Soloway, 1987]

JOHNSON, W. L., & SOLOWAY, E. (1987). **PROUST: An automatic debugger for PASCAL programs**. In G. P. Kearsley (Ed.), *Artificial Intelligence: Applications and methodology.*Redding, MA: Addison-Wesley

[Lesgold, 1984]

LESGOLD, A. M., **Acquiring Expertise**. In Anderson, J. R. and Kosslyn, S. M., editors, *Tutorials in learning and memory: essays in honor of Gordon Bower.* W. H. Freeman.

[Lesgold, 1989]

LESGOLD, A. M.; RUBINSON, H.; GLASSER, P. F. R.; KLOPFER, D.; and WANG, Y.; (1989). **Expertise in a complex skill: Diagnosing x-ray pictures**. In Chi, M.; Glasser, R.; and Farr, M; editors, *The nature of expertise*. Lawrence Erlbaum

[Major, 1991]

MAJOR, N., & REICHGELT, H.(1991) **Using COCA to build an intelligent tutoring system in simple algebra**. In: *Intelligent Tutoring Media*, vol 2, 159-168.

[Major, 1994]

MAJOR, N.P. (1994) **Evaluating COCA - What do teachers think ?** *Proceedings of the World Conference on Educational Multimedia and Hypermedia* - EDMEDIA 94. AACE Press.

[Major, 1997]

MAJOR, N., AINSWORTH, S., & WOOD D. (1997). **REDEEM: Exploiting Symbiosis Between Psychology and Authoring Environments.** *International Journal of Artificial Intelligence in Education* 8, 317-340

[Murray, 1998]

MURRAY, T. (1998). **Authoring knowledge-based tutors: Tools for content, instructional strategy, student model, and interface design**. *Journal of the Learning Sciences*, 7(1) 5-64.

[Murray, 1999]

MURRAY, T., **Authoring intelligent tutoring systems: An analysis of the state of the art**. *International Journal of Artificial Intelligence in Education* (10) 98-129

[Nicolson; Scott, 1986]

NICOLSON, R. I.; SCOTT, P. J. (1986) **Computers and education: the software production problem**. *In: British journal of educational technology*. v. 17, n. 1, p. 26-35.

[Nwana, 1990]

NWANA, H. S., **Intelligent Tutoring Systems: an overview**. *Artificial Intelligence Review* (1990) 4, 251-277

[Pimentel; Direne, 1998]

Andrey PIMENTEL e Alexandre DIRENE (1998). **Medidas Cognitivas no Ensino de Programação de Computadores com Sistemas Tutores Inteligentes**. *Anais do IX Simpósio Brasileiro de Informática na Educação* (SBIE-98), Fortaleza-CE, pp 206-215.

[Ramadhan; du Boulay, 1993]

RAMADHAN, H.; du BOULAY, B. (1993) **Programming environments for novices**. *LEMUT, Enrica.Cognitive models and intelligent environments for learning programming*. Springer Verlag, p. 125-134.

[Reiser; Kimberg; Lovett; Ranney, 1988]

REISER, B.; KIMBERG, D., LOVETT, M. & RANNEY, M. (1988) **Knowledge representation and explanation in GIL, an intelligent tutor for programming**. *Cognitive science laboratory report.* NJ, USA: Princeton University.

[Woolf; Cunningham, 1987]

WOOLF, B.P. & CUNNINGHAM, P.A. (1987) **Multiple knowledge sources in intelligent teaching systems**. *IEEE Expert*, Summer 1987.

[Virvou, 1991]

VIRVOU, M. (1991). **Error diagnosis for an active help system for Unix**, *Anais da Conferencia PEG,* pp. 467-475.

[Virvou, 1992]

VIRVOU, M. (1992) *A Human Plausible Reasoning Theory in the Context of an Active Help System for Unix Users*. Tese de doutorado, School of Cognitive and Computing Sciences - University of Sussex.