

**Ensino por tutor inteligente
de linguagens textuais de
comandos em aplicações de
sistemas operacionais**

Prof. Msc. Marcos Vinícius de Bortolli

Mestre em Informática pela UFPR, especialista em Administração - Gestão da Informática, pela FAE, tecnólogo em Processamento de Dados pela Funesp, gerente geral da Fábrica de Software da CPM S.A. em Pato Branco, professor da FADEP. Um dos autores dos projetos Incubadora Tecnológica Softex Gênesis Empreender, Parque Tecnológico de Pato Branco e Pato Branco Tecnópole.
mvbortolli@bol.com.br

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 princípios e da prática no domínio de linguagens de operação de dispositivos digitais, tais como os sistemas operacionais. 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 tutoramento inteligente, de caráter reativo por parte da máquina. 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.

Palavras Chave: Ambientes Interativos de Aprendizagem, Linguagens, Ferramentas de Autoria, Inteligência Artificial Aplicada à Educação.

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 classification of problem solutions supplied by the user. *ISABELA* carries out the assessment through the program diagnosis according to two main 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.

Keywords: Learning interactive environment, Languages, Authoring tools, Artificial Intelligence applied to education.

1 - Introdução

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

perícia¹. O princípio de programação diz respeito à sintaxe e à semântica 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 subcomandos em sua estrutura. Como a sintaxe de um comando é o primeiro grande obstáculo que um aprendiz encontra, um sistema tutorial para 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 o que cada comando faz é que um programador consegue utilizá-lo corretamente. 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 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

¹ du Boulay; Sothcott, 1987.

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 esperado. A palavra chave portanto é: ordem. 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. Uma vez adquirida 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 aluno². Em certo sentido, o computador se torna tanto o objeto como o próprio guia do estudo³. Um dos principais objetivos de um sistema tutorial inteligente (ITS) para o ensino de programação de computadores é apoiar o aluno na resolução de exercícios.

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

² Ramadhan; du Boulay, 1993.

³ du Boulay; Sothcott, 1987.

domínio da especialidade ensinada. Um exemplo aplicado especificamente à solução de problemas no domínio do sistema operacional de linguagem-objeto UNIX foi o protótipo RESCUER⁴.

Com relação a contribuições anteriores no campo de Shells com mecanismos pedagógicos genéricos, o sistema SATELIT-2⁵ 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.

Assim, fica claro que o problema principal neste caso é o da formulação de conceitos e construção de ferramentas para apoiar o ensino de Programação

⁴ Virvou, 1991; Virvou, 1992.

⁵ Direne, 1997.

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 premente de novos conceitos e mecanismos genéricos de interpretação pedagógica.

2 - Trabalhos Correlatos

Parece completamente razoável que deveríamos delegar parte do ensino de programação ao próprio computador. 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 aluno⁶. 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

⁶ Ramadhan; du Boulay, 1993

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 correta⁷. 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: Greaterp⁸, um sistema tutorial para o ensino da linguagem LISP e GIL⁹, 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: BRIDGE¹⁰ para o ensino de programação Pascal, SPADE¹¹ utilizado para o ensino de programação com a linguagem Logo; e DISCOVER¹² que auxilia o ensino de programação através de uma linguagem própria, semelhante ao pseudocódigo.

Como exemplos de sistemas passivos temos: Laura¹³, utilizado no apoio

⁷ Binder; Direne, 1999 - Pimentel; Direne, 1998.

⁸ du Boulay; Sothcott, 1987.

⁹ Reiser; Kimberg; Lovett; Ranney, 1988.

¹⁰ Bonar; Cunningham, 1986.

¹¹ du Boulay; Sothcott, 1987.

¹² Ramadhan; du Boulay, 1993.

¹³ Adam; Laurent, 1980.

ao ensino da linguagem Fortran; Mycroft¹⁴, que auxilia no ensino da linguagem Logo; Proust¹⁵, sistema de apoio ao aprendizado da linguagem Pascal e Bip¹⁶, 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 exceções são os sistemas SATELIT-1¹⁷ e Asimov¹⁸. 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)

¹⁴ Goldstein, 1975.

¹⁵ Johnson; Soloway, 1987.

¹⁶ Barr; Beard; Atkinson, 1976.

¹⁷ Direne, 1997.

¹⁸ Binder; Direne, 1999.

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.

Apesar do constante crescimento da área e dos méritos de sistemas de autoria tão completos quanto o EON¹⁹, 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. 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 amigável, flexibilidade no planejamento do ensino e agilidade na representação das

¹⁹ Murray, 1998.

soluções possíveis para os enunciados de exercícios criados.

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 tivesse independência de domínio e que pudesse, 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 domínio.

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.

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.

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 e suas descrições, podem ser encontradas na Tabela 1.

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 1 mostra a representação gráfica destes códigos de controle.

| Código de controle | Estrutura | Descrição |
|--------------------------------|-----------------------------------|---|
| Números seqüenciais positivos | <i>Comandos em seqüência</i> | A ordem é obrigatória. Um ou mais comandos devem ser executados em uma determinada ordem. Estes comandos executados em ordem diferente não terão o mesmo efeito |
| Números seqüenciais negativos | <i>Comandos alternativos</i> | Somente um deles deve ser executado. É o caso onde dois caminhos mutuamente exclusivos são possíveis para solucionar um problema. |
| Números seqüenciais se repetem | <i>Comandos em qualquer ordem</i> | Todos comandos devem ser executados, mas a ordem não importa. |
| Número 0 (zero) | <i>Comandos opcionais</i> | O comando pode ou não ser executado. Como o próprio nome diz, este comando é facultativo. |

Tabela 1 - Estruturas genéricas de soluções

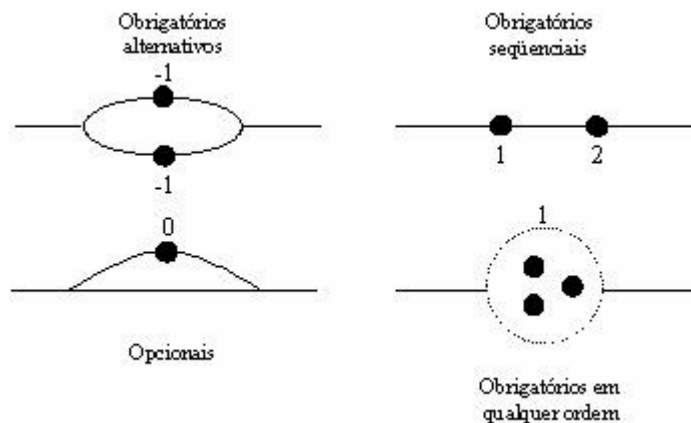


Figura 1 - Grafo de Alternativas de Soluções

3.2 - A ferramenta de autoria a-ISABELA

A ferramenta de autoria *a-ISABELA* foi implementada utilizando-se os recursos da linguagem de programação orientados a objetos e também do SGBD *MS-Access*. 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.

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 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*.

4 - Conceitos e Ferramenta 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. É importante também 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*. 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 - 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 amigável 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 guiar o aprendiz em direção à respostas corretas.

Ao apresentar um comando ao aluno, é necessário: (1) dizer para que

serve; (2) dar um exemplo de uso; (3) utilizar um exemplo prático; (4) utilizar um exemplo dinâmico; (5) dizer quais os comandos equivalentes; (6) dizer suas diversas sintaxes; (7) dar a possibilidade do aprendiz testar seu entendimento sobre os conceitos expostos.

4.2 - 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 seção 3.1), 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.

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 que 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. 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. 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. Porém a resposta do aluno, apesar de aparentemente correta, pode ser incorreta e perigosa se utilizar comandos inapropriados. Para resolver isto, o *Interpretador Genérico* avalia também quais os comandos utilizados na resposta do aluno e compara estatisticamente com as soluções pré-definidas da base. 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 2.

| 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 2 - Quadro de classificação das respostas do aluno

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.

Mensagem de sub-utilização de recursos. Este é um caso que está relacionado com a perícia do aluno, e cabe ao ISABELA instruir o aluno quanto a isso.

Mensagem de equivalência, que são utilizadas para ampliar o conhecimento do aluno com comandos equivalentes ao utilizados na resposta.

Mensagem de erros genéricos, os quais fornecem uma mensagem padrão a respostas erradas que não estão catalogadas ainda.

Mensagem de acerto padrão: os quais fornecem uma mensagem padrão

às respostas corretas (e ideais) que não se encaixam em nenhuma das alternativas anteriores.

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.

4.3 - A ferramenta de ensino e-ISABELA

A interface da ferramenta de ensino e-ISABELA utiliza padrões visuais simples, porém eficientes, 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*, 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. Para a navegação entre 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.

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. Os pontos que merecem

destaque em relação ao que foi desenvolvido e alcançado são: o diagnóstico automático de soluções, a independência de domínio, 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*.

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. 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.

6 - Referências Bibliográficas

du BOULAY, B.; SOTHCOTT, C. (1987) Computers teaching programming: an introductory 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.

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.

VIRVOU, M. (1991). **Error diagnosis for an active help system for Unix**,

Anais da Conferencia PEG, pp. 467-475.

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.

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

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

PIMENTEL, A.; DIRENE, A. I. (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.

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.

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

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

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

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

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

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.