Um Ambiente Integrado para auxílio ao Ensino de Ciência da Computação

Eliana S. de Almeida[1], Julian D. Herrera[1], Luiz Josué da S. Filho[1], Hyggo Oliveira de Almeida[2*], Evandro de Barros Costa[1], Bruno L. Vieira[1], Marcelo D. de Melo[1]

[1]Departamento de Tecnologia da Informação – Universidade Federal de Alagoas(UFAL)
Caixa Postal 15.064 – 91.501-970 – Maceió – AL – Brasil

[2]Departamento de Engenharia Elétrica - Universidade Federal de Campina Grande (UFCG)
Av. Aprigio Veloso, 882 – Bodocongo, 58109-970 Campina Grande - PB – Brasil
Phone:+(5583)310-1150

eliana@fapeal.br, julian@tci.ufal.br, ljsf@tci.ufal.br,
hyggo@dee.ufcg.edu.br, ebc@fapeal.br

Resumo. *Neste artigo é apresentado um ambiente integrado para o auxílio ao ensino introdutório de fundamentos da Computação que tem como objetivo dar suporte às disciplinas de Teoria da Computação, Arquitetura de computadores e Programação. O ambiente integra ferramentas de autoria de pseudo-código e micro-programação, com compilação e execução inspecionada pelo aluno, representação gráfica e mapa de memória, além de simulações com máquinas abstratas, como a de Turing. Além disso, apresenta-se também um cenário de interação do aluno com as ferramentas que compõem o ambiente.*

Palavras-chave: *Ambientes Interativos de Aprendizagem, Informática na educação, Ensino de Computação.*

Abstract. *This paper presents an Integrated Environment to aid teaching of fundamentals of computation under basic level. It aims to support learning in domains such as Theory of Computation, Computer Architecture, and Programming. The environment integrates authoring tools to micro-code and micro-programming, endowed with tools oriented to students to observe resources such as compilation and running process, graphical representation, and memory map. Also, it offers simulations with abstract machine as Turing machine. Finally, an interaction scenario involving the student and computer tools is presented.*

Key Words: *Interactive Learning Environment, Computer in Education, Computer Teaching*

1. Introdução

A experiência adquirida em anos de ensino em cursos de computação indica a falta de interesse e a dificuldade de aprendizagem do aluno iniciante no que diz respeito aos fundamentos teóricos de disciplinas como programação, arquitetura de computadores e teoria da computação. Uma das hipóteses levantadas sobre tal problema diz respeito à

[*] Bolsista CNPq no Programa de Doutorado em Engenharia Elétrica COPELE/DEE/UFCG.

grande quantidade de conceitos abstratos que dificultam a percepção do forte relacionamento entre as três disciplinas. Sendo assim, a integração prática dos conceitos destas disciplinas, com o auxílio de ferramentas, ocasionaria uma melhoria no tempo necessário para o aprendizado dos alunos em três dos mais importantes tópicos de introdução à Ciência da Computação.

Com base nestes argumentos, propõe-se um ambiente integrado e simples que inicie o aluno no entendimento de conceitos básicos e essenciais da computação, fornecendo ferramentas para a construção e execução de programas, depuração e criação de processos explicativos, que esclarecem com maior representação de detalhes a solução proposta, para traduções entre diferentes formas de representação de soluções.

Em relação ao ensino de programação, o ambiente utiliza a noção de níveis de aprendizado, desvencilhando o aluno de detalhes de sintaxe e semântica específicos de linguagens de programação, quando iniciante, focando-o assim na resolução do problema. Em um nível mais avançado, o ambiente mantém todas as restrições de sintaxe e semântica da linguagem com as quais o aluno irá se deparar quando estiver programando em outras linguagens estruturadas. Independente do nível de conhecimento do aluno, o ambiente fornece mecanismos para facilitar a resolução de problemas, provendo uma ferramenta de depuração, simulação de memória e representação gráfica da solução, baseada na ferramenta AMBAP[ALMEIDA 2001, 2002].

Em relação ao entendimento do funcionamento interno de um sistema computacional, o ambiente provê ferramentas de software que permitem, através de simuladores, explorar todas as fases envolvidas no processamento e manipulação, processos de tradução e compilação, do programa construído, destacando as traduções realizadas e a execução do código de máquina pela Unidade Central de Processamento (UCP). Estas funcionalidades tiveram como base o ambiente ASIMAV[COSTA 1994], que fornece os fundamentos necessários para o desenvolvimento de mecanismos que auxiliem no entendimento do processo de representação de um programa em diversos níveis.

No tocante à disciplina de teoria da computação, é proposta uma forma de representação, através de formalismos, incluindo simulações em máquinas, tais como a de Turing, de códigos gerados pelo aluno. Desta forma, espera-se facilitar a compreensão das noções de Computabilidade [BRAINERD 1974], das evidências da *Tese de Church*, dentre outras.

Além disso, como forma de fornecer o suporte de conhecimento necessário ao aluno para a utilização das ferramentas disponibilizadas no ambiente proposto, provê-se uma ferramenta de tutoria, associada ao ambiente proposto, que visa, principalmente, oferecer ao usuário uma interação mais individualizada com este ambiente e um método de auxílio ao professor na tentativa de acompanhar a aprendizagem do aluno.

O restante do artigo está organizado da seguinte forma: na Seção 2, são apresentados os trabalhos correlatos e envolvidos na problemática e no contexto dessa pesquisa; na Seção 3, é descrito o ambiente proposto; na Seção 4, é apresentado um cenário de interação do aluno com o ambiente; e, finalmente, na Seção 5, são descritas as conclusões.

2. Trabalhos correlatos

A proposta do ambiente em questão representa uma contribuição no aprendizado de conceitos de *hardware* (interpretação interna) e *software* (solução algorítmica), os quais são intercambiáveis e logicamente equivalentes. Existem muitas ferramentas na Internet para simulação de máquinas de Turing e por outro lado, muitos ambientes para desenvolvimento de software em linguagens como Java, C++, dentre outras. Porém, são poucas as soluções com foco na aprendizagem. Além disso, elas não provêem a característica integrada a que o presente trabalho propõe abordar.

A principal abordagem correlata é uma das ferramentas na qual o ambiente proposto se baseia, a ferramenta ASIMAV. ASIMAV propõe um sistema computacional que visa prover ferramentas que objetivam favorecer as atividades de aprendizagem e promover o interesse dos aprendizes em temas como Arquitetura e Organização de Computadores, Sistemas Operacionais, Linguagens de programação e Processadores de Linguagem: Interpretadores e Compiladores. O ASIMAV foi definido para, através do uso de simuladores, apresentar os conceitos fundamentais envolvidos nos temas supracitados, em uma visão unificada e estruturada em níveis de abstração. A apresentação desses conceitos é suportada pelo uso da noção de máquinas virtuais [COSTA 1994]. Essas máquinas consistem em níveis de abstração diferentes que têm correspondência na prática, pois geralmente são utilizadas pelos projetistas de software e hardware. Esses níveis são: linguagem de alto nível, linguagem de montagem (assembly) e linguagem de controle da microarquitetura (micro-assembly). A idéia de construir um simulador do funcionamento interno de um computador objetiva justamente aproximar a teoria da prática, fornecendo ferramentas que serão úteis, não apenas para os alunos "experimentarem" os conceitos que são apresentados em sala de aula, mas também para auxiliar os professores na exposição.

3. O ambiente proposto

Na sua concepção, o ambiente tenta simular o computador em diferentes níveis de abstração. O mesmo está projetado para ser composto por módulos inter-relacionados. Esses módulos, previstos para serem simples e de fácil manipulação, farão a interconexão entre as diferentes maneiras de representação de uma solução algorítmica e o aluno. Os módulos da arquitetura do ambiente, apresentada na Figura 1, são: simulador, interpretador, tradutor, responsável pela conexão entre as representações, editor e um módulo tutor.

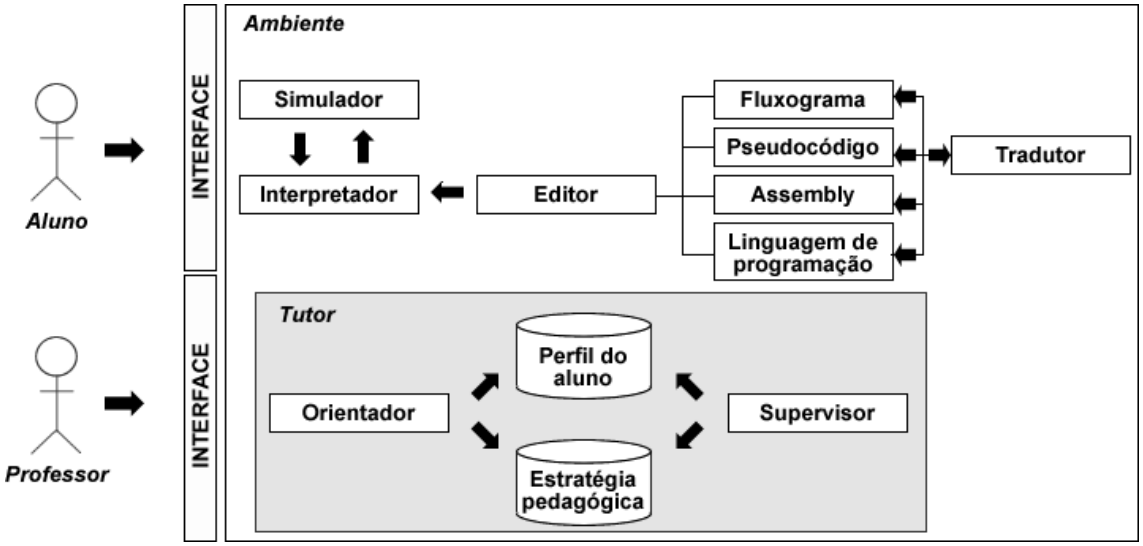

Figura 1. Arquitetura do ambiente e a interação entre os módulos.

3.1. Módulo simulador

O módulo simulador implementa a representação gráfica dos resultados das ações do módulo interpretador e fornece uma interface com o usuário para entrada e saída de dados. Este módulo possibilita também a execução passo-a-passo da solução, permitindo que o usuário a avalie detalhadamente. Em cada nível de representação da solução, o aluno tem acesso a um conjunto de ferramentas de simulação específicas que representam distintas abstrações. Por exemplo, no nível assembly, o aluno tem a oportunidade de visualizar os conteúdos de elementos de hardware, como registradores e saídas da Unidade Lógica e Aritmética (ULA), de acordo com a microarquitetura proposta em [TANENBAUM 1999]. Já no nível de representação em pseudocódigo o enfoque está na inspeção de variáveis e funções.

3.2. Módulo interpretador

O módulo interpretador tem a função de executar as instruções definidas pelo usuário e é baseado no uso de uma linguagem algorítmica de fácil aprendizagem e assimilação [CRESPO 1990], pois se aproxima da linguagem natural. Tal linguagem facilita o aprendizado, desvencilhando o aluno iniciante de detalhes específicos de linguagens de programação, enfocando apenas na resolução do problema. O interpretador fornece também uma ferramenta completa para depuração de código.

3.3. Módulo tradutor

O ambiente foi projetado utilizando-se técnicas de construção de compiladores orientadas a objetos. Nesta perspectiva, torna-se viável gerar representações alternativas da solução formulada pelo aluno. A partir da construção de todo o mecanismo de compilação/interpretação, mais especificamente da estrutura intermediária (uma árvore de objetos), o módulo tradutor é capaz de gerar outras formas de representação, seja através da simples interpretação do pseudocódigo, em representação gráfica do fluxo de execução, através de formalismos ou em representação de baixo nível.

O módulo tradutor implementa a conversão entre os diferentes níveis de representação: através de linguagem algorítmica (pseudocódigo); representação gráfica

do fluxo de dados baseada na representação algorítmica; representação em linguagem de montagem (assembly); linguagem de controle da microarquitetura (micro-assembly); e representação em alguns formalismos, como a Máquina de Turing. Essas formas de representação são descritas a seguir.

- *Representação em pseudocódigo*

O ambiente permite que o aluno expresse suas soluções algorítmicas em uma linguagem próxima à linguagem natural (algorítmica ou pseudocódigo). São disponibilizados dois modos de usuário: usuário iniciante e normal. No modo de usuário iniciante, voltado para o aluno com pouca experiência em computação, a linguagem em pseudocódigo definida para esse interpretador possui grande liberdade sintática, no sentido de deixar para o usuário a opção de escolher como melhor escrever suas instruções. Por exemplo, escrever "`faça`" ou "`faca`" no comando "`enquanto <condição> faça ..... fim-enquanto`", tem o mesmo efeito. Assim como, alterar a ordem deste comando e escrever "`faça enquanto <condição> .....fim-enquanto`". Esta funcionalidade provê facilidades ao usuário para atingir o seu objetivo principal que é solucionar um determinado problema, sem maiores preocupações em fixar a sintaxe da linguagem. Já no modo usuário normal, as convenções tradicionais de sintaxe das linguagens de programação estruturadas em geral são mantidas tendo em vista a preparação do aluno para estas linguagens.

- *Representação gráfica de fluxo*

A representação gráfica do fluxo visa simular o fluxo de execução do código gerado pelo aluno, em pseudocódigo, através da criação de símbolos, pré-definidos, que representam as estruturas de controle de fluxo presentes na linguagem algorítmica. Através de um processo de utilização de cores que destacam a "trajetória percorrida" pelo fluxo de execução, como observado na Figura 2, além da representação do conteúdo das variáveis presentes na memória, garante-se um melhor entendimento da lógica de programação utilizada pelo próprio estudante.

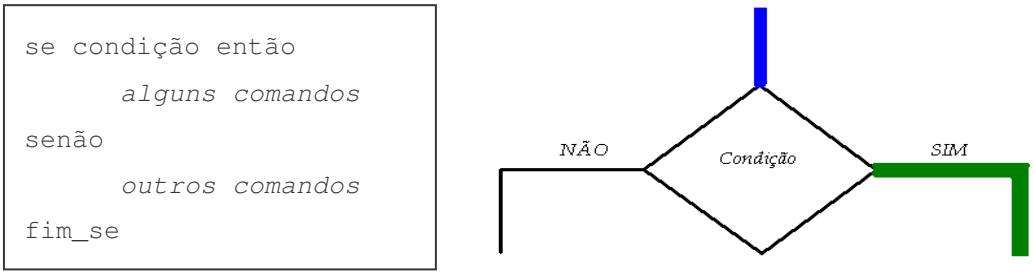

Figura 2. Pseudocódigo e sua representação no simulador gráfico.

- *Representação em linguagem de montagem e em microlinguagem*

A representação em linguagem de montagem permite que os alunos pratiquem seus conhecimentos, no que diz respeito à programação em assembly. A linguagem utilizada é baseada no Assembly definido em [TANENBAUM 1999]. Essa representação em nível de linguagem de microprogramação, baseada na arquitetura

proposta por [TANENBAUM 1999], é também destinada aos alunos que desejam testar seus conhecimentos em linguagem de mais baixo nível.

Um dos requisitos do tradutor do ambiente em questão é que duas das linguagens de destino sejam uma linguagem de montagem e uma microlinguagem, tornando-se indispensável a descrição da arquitetura à qual estas linguagens se referem. Isso se deve à relação direta que uma linguagem de montagem, e mais intimamente a microlinguagem, têm com a arquitetura do sistema computacional correspondente.

Todos os comandos de um assembly estão intimamente ligados à forma como a arquitetura de máquina está estruturada. Os acessos aos elementos básicos do hardware, como registradores, posições de memória e pilha, bem como mecanismos de entrada e saída são de interesse imediato, uma vez que cada instrução trabalha diretamente com um ou mais desses elementos. A representação em microlinguagem é ainda mais dependente da arquitetura alvo. Levando tudo isso em consideração, uma arquitetura simples, mas abrangente o suficiente para englobar todo o poder computacional da linguagem de origem, foi definida de acordo com as proposições de [TANENBAUM 1999]. De acordo com este trabalho, as microarquiteturas atuais devem ser projetadas com o intuito de suportar as necessidades das linguagens de alto nível, sendo assim o modelo adotado para o ambiente. Na Figura 3 apresenta-se a tela de edição de código do ambiente. A representação desse código em linguagem de máquina é apresentada na Figura 4*.

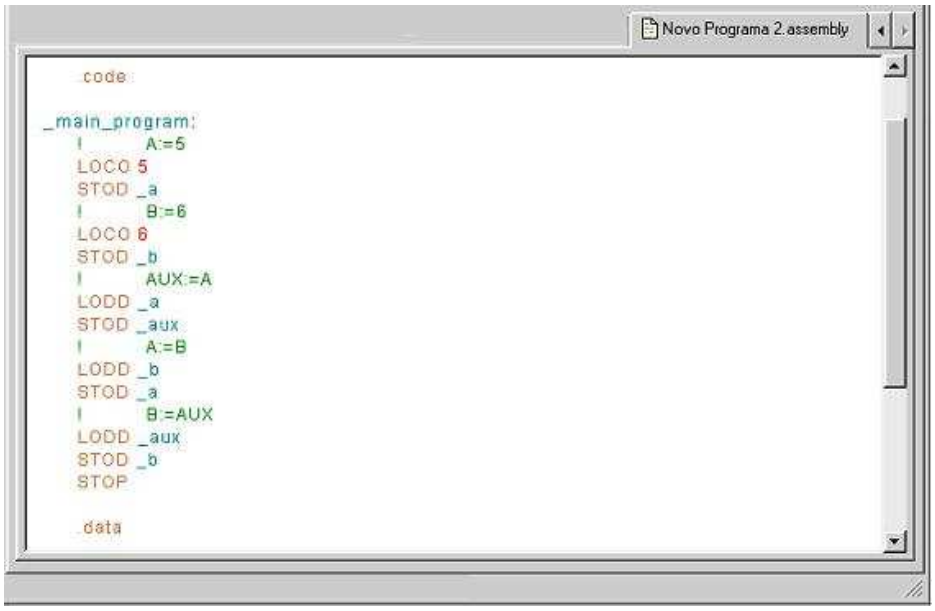

Figura 3. Um exemplo de código traduzido para linguagem de montagem.[1]

[*] Disponível em http://www.ufal.br/tci/ambap/versaodois/simula.zip

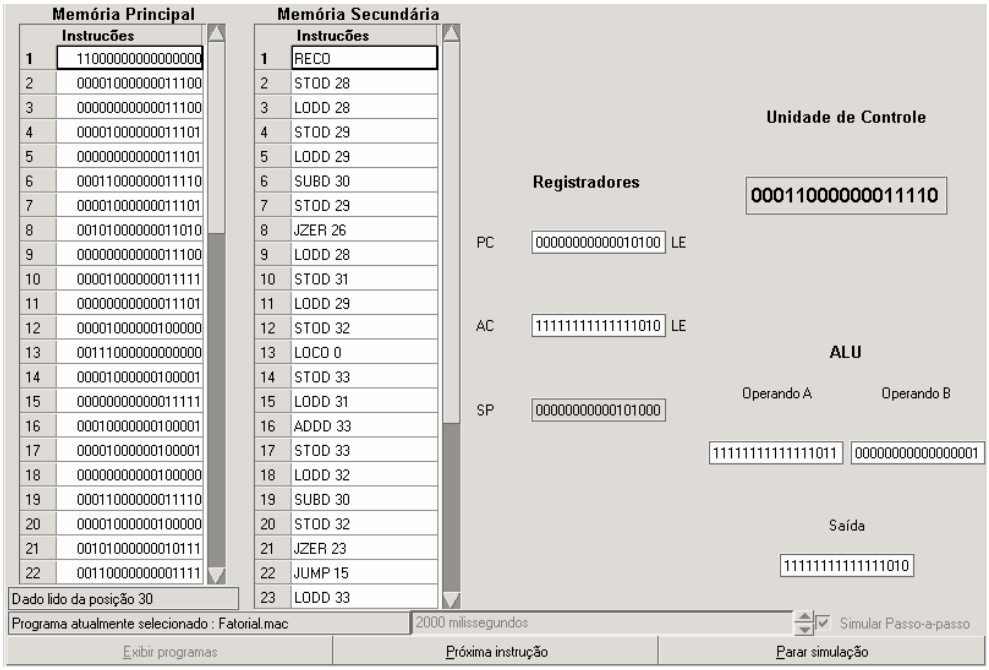

Figura 4. Um exemplo de tradução de assembly para microlinguagem.

- *Representação em termos de formalismos*

Com relação à representação por meio de formalismos ilustrativos das evidências de Church, esse módulo do ambiente, ainda em fase de desenvolvimento, tem como objetivo prover uma ferramenta computacional que, através de um conjunto de procedimentos, define sua capacidade de computabilidade e uma função recursiva equivalente. Além disso, apresentar simulações desse conjunto de procedimentos em alguns formalismos, como a Máquina de Turing, Post, Markov, etc, sendo um meio facilitador da aprendizagem de Teoria da Computação, visto que o aluno ainda não dispõe de um método de visualização de toda essa teoria de uma forma prática.

3.4. Módulo editor

O módulo editor oferece um ambiente adequado e adaptado para escrever a solução do problema proposto. Dirigido à sintaxe, provê algumas funcionalidades como, por exemplo, um esquema de cores para destacar as diversas estruturas sintáticas da linguagem. Além disso, o editor identifica alguns erros comuns de sintaxe, explicitando-os através de mensagens, acrescidas de "dicas" de como "repará-los". Isto minimiza o trabalho do professor em laboratório, além de induzir o aluno a tentar reparar o erro sozinho. Na Figura 5 é apresentada a tela de edição de código do ambiente*, com destaque para o esquema de cores.*

* Disponível em http://www.ufal.br/tci/ambap/versaodois/ambap.zip

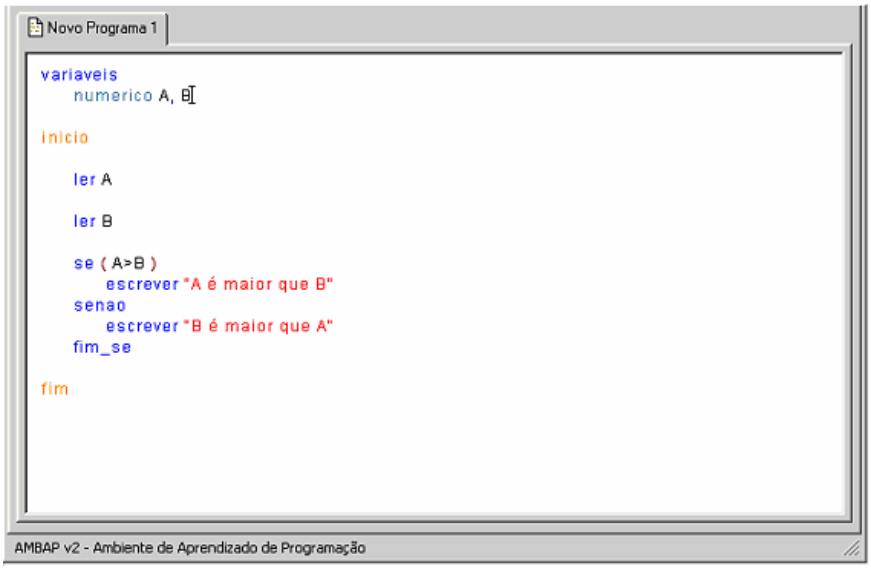

Figura 5. Um exemplo de pseudocódigo no editor.

3.5. Módulo tutor

O módulo tutor tem como objetivo auxiliar e supervisionar o aluno durante o processo de aprendizado. Para tanto, visa proporcionar ao usuário interações personalizadas com o ambiente, a partir de atividades de resolução de problemas. Pode ser decomposto nos seguintes sub-módulos:

- *Base de Recursos Pedagógicos*: repositório de conteúdos pedagógicos (conceitos, exemplos, problemas propostos e respectivas soluções, dicas etc), bem como a seqüência em que eles devem ser vistos.
- *Base de Perfis dos Alunos:* armazena informações referentes ao desempenho do aluno, isto é, o conteúdo visto por ele, informações inferidas durante a interação com o módulo orientador (tais como erros cometidos, tempo gasto para a resolução, freqüência de solicitação de ajuda etc), e soluções dadas pelo aluno aos problemas propostos.
- *Orientador:* implementa um guia para o aluno durante o processo de aprendizado, exibindo o conteúdo pedagógico de acordo com a estratégia definida pelo professor.
- *Supervisor:* permite que o professor inspecione o perfil do aluno, podendo acompanhar seu desempenho e então gerar novas estratégias pedagógicas personalizadas para um aluno ou para um grupo.

4. Cenário de execução

A seguir é apresentado um cenário de interação entre os componentes do ambiente.

1. Supondo um primeiro momento, em que o aluno vai aprender lógica de programação. Esse aluno estaria interessado apenas em criar soluções algorítmicas para a solução de problemas. Nesse caso, o ambiente provê um editor dirigido à sintaxe, com liberdade sintática, fazendo com que o aluno não

perca seu foco de resolução de problema. Com sua solução expressa, o aluno poderia executar seu algoritmo, dessa forma validando sua solução.

2. Outra opção seria a execução passo-a-passo com o simulador de gráfico do fluxo de execução, em que o aluno poderia acompanhar a execução de seu algoritmo a cada passo, analisando o impacto de cada operação no resultado final. Assim, o ambiente permite ao aluno identificar mais facilmente o fluxo de execução do programa.
3. Através do módulo tradutor o aluno pode converter uma solução expressa em pseudocódigo para representação em fluxograma.
4. Em um segundo momento, o interesse do aluno seria transformar as instruções por ele definidas em instruções em linguagem de montagem. Neste, caso, o ambiente provê o editor de pseudocódigo, para a codificação de suas instruções, e também o tradutor para gerar as instruções na linguagem alvo.
5. Além disso, este aluno gostaria de observar como estas instruções são executadas pela máquina. Para tanto, o ambiente fornece tradução de linguagem de montagem para microlinguagem, e um simulador para auxiliar o entendimento da execução das instruções.

5. Conclusões

Neste artigo é apresentado um ambiente integrado para auxílio ao ensino de fundamentos da Computação. O ambiente provê ferramentas de suporte ao ensino de disciplinas básicas como programação, teoria da computação e arquitetura de computadores. Essas ferramentas se utilizam de recursos pedagógicos e de simulação para fornecer facilidades de aprendizado ao aluno que está se iniciando em computação.

Tal como comentado anteriormente, boa parte do ambiente já foi implementado, sendo inclusive utilizado em sala de aula, tendo seus resultados parciais publicados. Atualmente, os módulos restantes estão sendo implementados, para em seguida serem integrados ao ambiente e validados, inicialmente, pelos alunos de Ciência da Computação da Universidade Federal de Alagoas.

6. Referências

[ALMEIDA 2001] ALMEIDA, A. A. M. *et al*, AMBAP: Um Ambiente de Aprendizado de Programação (Resumo). Anais do 6º Congresso de Iniciação Científica – ASSER – UNICENP, pág 91, São Carlos –São Paulo, Brasil, 2002.

[ALMEIDA 2002] ALMEIDA, E. S. *et al*, AMBAP: Um Ambiente de Aprendizado de Programação. Anais do XXII Congresso da Sociedade Brasileira de Computação – X WEI, Florianópolis-SC (Brasil).

[BRAINERD 1974] BRAINERD, W. S. and Landweber, L. H., Theory of Computation, New York: John Wiley, 1974.

[COSTA 1994] COSTA, E. de B. *et al*, ASIMAV: ambiente de Simulação de Máquinas Virtuais. Anais do II Congresso Ibero-Americano de Informática na Educação, Lisboa, Portugal, 1994.

[CRESPO 1990] CRESPO, S., ILA – Interpretador de Linguagem Algorítmica, 1990, (http://www.inf.unisinos.br/~crespo/ila/ila.htm).

[EVARISTO 2000] EVARISTO, J. e Crespo, S., Aprendendo a Programar programando numa Linguagem Algorítmica Executável (ILA), Book Express, 2000.

[GORDON 1998] GORDON, M. J. C., Programming Language Theory and its Implementation, C. A. Hoare Series Editor. Prentice Hall International, 1998.

[HOLMES 1995] HOLMES, J., Object-oriented Compiler Construction, Prentice-Hall Inc,1995.

[MENDES 2002] MENDES, A. J. N., Software Educativo para Apoio à Aprendizagem de Programação. Universidade de Coimbra. Portugal, 2002, (http://www.c5.cl/ieinvestiga/actas/tise01/pags/charlas/charla_mendes.htm).

[POLYA 1978] POLYA, C., A Arte de Resolver Problemas, Ed. Interciência, 1978.

[PRATT 1996] PRATT, T.W. and Zelkowitz, M.V., "Programming Languages: Design and Implementation". Terceira edição, Prentice Hall, 1996.

[TANENBAUM 1999] TANENBAUM, A. S., Organização Estruturada de Computadores, Terceira Edição, LTC, 1999.