Compilador Educativo VERTO: ambiente para aprendizagem de compiladores
VERTO Educational Compiler: environment for compiler learning

Carlos Sérgio Schneider[*]
Liliana Maria Passerino[**]
Ricardo Ferreira de Oliveira[***]

Resumo:

O Compilador Educativo Verto surgiu da necessidade de desenvolver uma ferramenta para apoio pedagógico na disciplina de Compiladores do Centro Universitário Feevale. A disciplina de Compiladores no contexto de um curso de graduação, objetiva levar o aluno a perceber concretamente as diversas etapas envolvidas no processo de compilação, elaborando, ao final da disciplina, o projeto de um compilador funcional para uma gramática simples. O Compilador Verto foi escrito na linguagem de programação Java e elaborado sob os termos da licença GPL (GNU General Public License). Com este ambiente visamos proporcionar aos estudantes de Compiladores um projeto aberto e documentado para utilização em seu aprendizado, apoiando a compreensão das fases de um compilador, sobretudo as fases de síntese de código intermediário e geração de código-objeto, promovendo uma aprendizagem contextualizada.

Palavras-chave: ambientes digitais, ensino de compiladores, aprendizagem contextualizada

Abstract:

The Verto Educational Compiler has arisen from the need to develop a tool intended to provide pedagogical support to the compilers course of the Centro Universitário Feevale. This course, in the undergraduate context, aims to conduct the student to perceive in a concrete way the several phases related to the compiling process, developing, at the end of the course, a project of a functional compiler for a simple grammar. The Verto Compiler was written in the Java programming language under the terms of the GPL (GNU General Public License). With this environment we aim to provide the students of Compilers an open and documented project for use in its learning, supporting the understanding of the diverse involved stages, mainly the phases of synthesis of intermediate code and code-object generation, promoting a context learning.

Keywords: virtual learning environment, education of compilers, context learning

Ensino de Compiladores e Aprendizagem Contextualizada

[*] Mestre em Engenharia de Produção pela PPGEP/UFRGS. Professor da Ciência em Computação do Centro Universitário FEEVALE. carlos.schneider@feevale.br
[**] Doutora em Informática na Educação – PGIE/UFRGS. Coordenadora de Estágios da Licenciatura em Computação e Assessora em Tecnologias Educacionais do NATE/NEAD do Centro Universitário FEEVALE. liliana@feevale.br
[***] Mestre em Ciência da Computação pela PUC-RS. Professor da Ciência e da Licenciatura em Computação do Centro Universitário FEEVALE. ricardofo@feevale.br

Os compiladores são ferramentas que, dito de forma simplificada, realizam a leitura de um programa escrito em uma linguagem (linguagem fonte) e a traduzem para uma outra linguagem (linguagem objeto) mantendo a semântica original (Aho et al., 1995). Durante esse processo de tradução, o compilador deve realizar duas tarefas básicas: a **análise**, na qual o texto-fonte é verificado, analisado e compreendido em seus aspectos semânticos e a **síntese**, na qual o texto-objeto é gerado de forma a corresponder ao texto de entrada (Rangel, 2005).

O ensino de compiladores aborda, portanto, aspectos relativos ao estudo de técnicas e métodos específicos para análise léxica, sintática e semântica, organização da tabela de símbolos, gerenciamento de erros e geração de código. Logo, percebemos que o ensino de compiladores, além de ser fundamental na Ciência da Computação, envolve o ensino de conteúdos eminentemente procedimentais. Um conteúdo procedimental, segundo Zabala (1998) se caracteriza por ser um conteúdo no qual o aluno deve apropriar-se de um processo de construção que parte de um conjunto de ações, modos de agir, de acordo com metas específicas. Sua aprendizagem, portanto, consiste na aprendizagem de um processo composto por uma seqüência ordenada de passos ou etapas, na qual participam técnicas, estratégias e métodos como componentes do conteúdo procedimental.

Desta forma, o desenvolvimento de um trabalho prático é uma das questões fundamentais relacionada à disciplina de Compiladores que envolve geralmente o projeto de um compilador e proporciona aos alunos a oportunidade de aplicação das técnicas estudadas nesta e outras disciplinas como teoria da computação, linguagens e autômatos, sistema operacionais e arquiteturas de computadores.

Mas, frente a esta necessidade, enfrentamos um problema de práxis educativa que implica processos de motivação e significação. Pois, partindo de uma concepção construtivista do processo educativo, não consideramos aprendizagem como o conhecimento de respostas corretas ou reprodução de conhecimentos já definidos. Pelo contrário, consideramos aprendizagem como uma construção de uma representação pessoal de um conteúdo que é objeto de aprendizagem (Coll, 1998). No nosso caso, esse objeto de aprendizagem é o compilador e suas funcionalidades e características. Porém, muitos estudantes não têm a expectativa de vir a estar engajados em projetos de construção de compiladores em sua vida profissional e desta forma, tendem a considerar o estudo de Compiladores menos relevante que outras disciplinas mesmo que as técnicas aprendidas possam ser utilizadas numa grande variedade de aplicações (Debray, 2003) (Henry, 2005).

Portanto, conduzir um curso de compiladores em uma disciplina de um único semestre não é uma tarefa fácil. Principalmente considerando que devido às dificuldades apresentadas pelos alunos na compreensão das técnicas de análise sintática abordadas, implica com que as fases de síntese sejam vistas de forma rápida e, em alguns casos, superficialmente. Nesse sentido, e visando auxiliar a compreensão das fases de um compilador, sobretudo das fases finais de geração de código intermediário e geração de código-objeto, foi elaborado o Compilador Educativo Verto[i] como parte de um ambiente digital de aprendizagem (ADA) para compiladores. Consideramos o Verto parte de um ADA e não o próprio ambiente pois partimos da concepção proposta por Passerino(2005) na qual um ambiente digital de aprendizagem é composto por um conjunto de ferramentas computacionais, um professor mediador, os alunos participantes, as seqüências didáticas planejadas para o processo de ensino e aprendizagem apoiadas por uma metodologia de ensino embasada na epistemologia do professor.

A seguir apresentamos as funcionalidades e aspectos técnicos do Verto para depois entrarmos nas possibilidades educativas do mesmo.

Compilador Educativo VERTO

O Compilador Educativo Verto surgiu da necessidade de desenvolver uma ferramenta para apoio pedagógico na disciplina de Compiladores do Centro Universitário Feevale. A disciplina de Compiladores no contexto de um curso de graduação, objetiva levar o aluno a perceber concretamente as diversas etapas envolvidas no processo de compilação, elaborando, ao final da disciplina, o projeto de um compilador funcional para uma gramática simples.

O Compilador Verto foi escrito na linguagem de programação Java e elaborado na forma de um software livre com licença GPL[ii] (GNU Public License). Com este ambiente visamos proporcionar aos estudantes de Compiladores um projeto aberto e documentado para utilização em seu aprendizado.

A questão inicial envolvida na elaboração do projeto Verto foi a escolha de quais seriam as linguagens fonte e objeto utilizadas. Com relação à linguagem fonte, a mesma foi especificada utilizando-se *português estruturado* com uma sintaxe próxima a da linguagem C por ser esta de amplo conhecimento dos alunos do curso de Ciência da Computação. Por outro lado, visando a simplicidade de aprendizagem da ferramenta utilizou-se um reduzido conjunto de instruções. Já a linguagem objeto escolhida foi a linguagem *César*, implementada na UFRGS pelo prof. Dr. Raul Weber para as disciplinas de Arquitetura de Computadores I e II (Weber, 2001). Os principais fatores que levaram à escolha desta linguagem como linguagem-objeto foram:

- a sua utilização pelos alunos na disciplina de Arquitetura de Computadores da Feevale;
- a sua ampla utilização em outras Universidades;
- a disponibilidade de acesso à ferramenta *César* a partir do serviço de ftp da UFRGS e, principalmente,
- a filosofia envolvida na elaboração da mesma, pois é uma ferramenta elaborada com fins didáticos, fácil de ser compreendida e dispondo das principais funções encontradas em uma máquina real.

O processo de compilação do Verto se dá em 2 etapas: inicialmente a ferramenta gera um código-intermediário em um formato macro-assembler. Este formato visa facilitar a compreensão das estruturas compiladas pois dispõe de instruções com formas mais simplificadas das instruções da máquina *Cesar*. Deste modo, promove-se um melhor entendimento de como as estruturas da linguagem Verto, elaboradas pelo aluno, são transformadas para instruções próximas à da máquina objeto. A partir desta forma intermediária, gera-se o arquivo destino final contendo as instruções no formato da máquina hipotética *Cesar* permitindo que o aluno execute e analise o algoritmo.

O foco principal do Compilador Verto são as fases finais do processo de compilação. Devido a isto, optou-se no projeto pela utilização de uma técnica de análise léxica simples e de um método de análise sintática recursiva descendente, ou seja, um método *Top-Down*. A escolha do método[iii] ateve-se a dois objetivos básicos: estabelecer uma transição simples entre o processo de projetar uma linguagem e a implementação de um analisador sintático e procurar elaborar o analisador mais simples possível que realizasse corretamente sua função (Metsker, 2001).

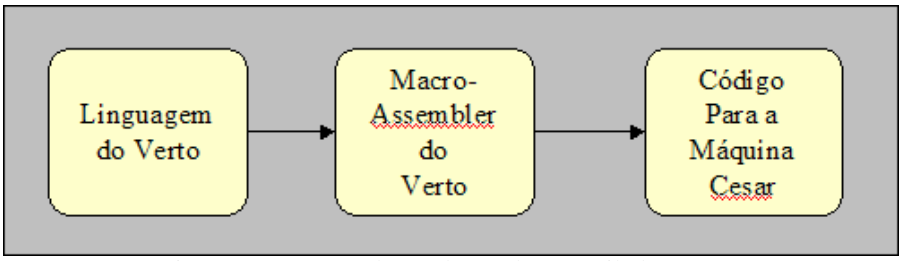

Figura 1: Esquema de Tradução do Compilador Verto.

A ferramenta dispõe de uma janela para edição de textos-fonte escritos na linguagem Verto (Figura 2).

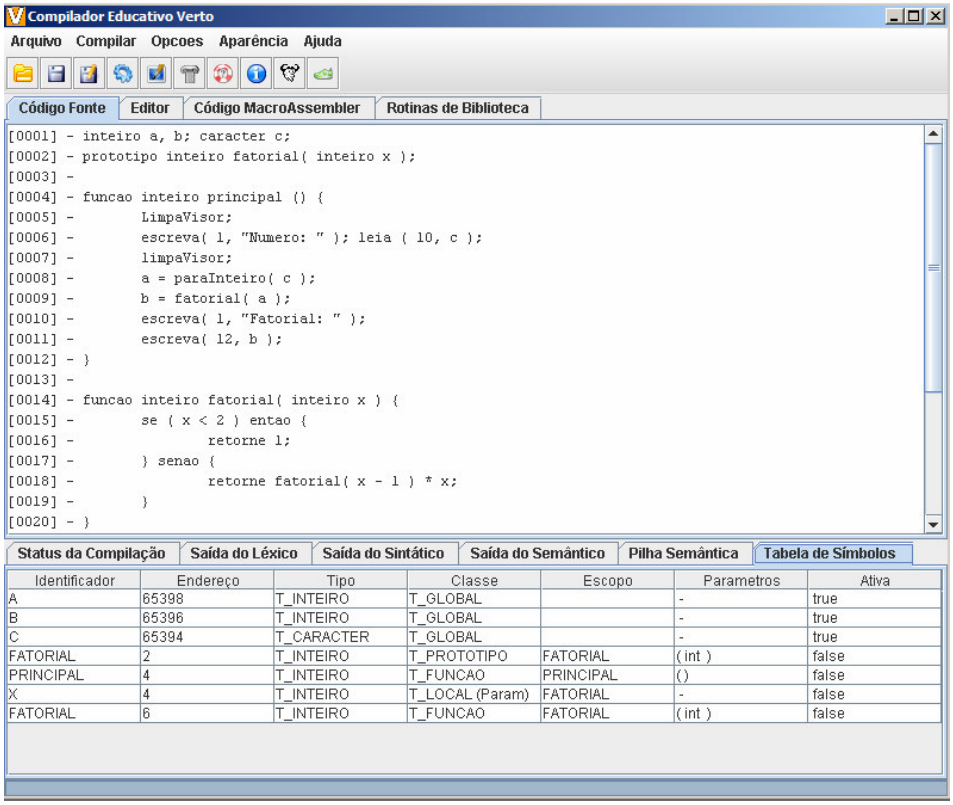

Figura 2: Tela de Edição do Compilador Verto.

A figura 3 mostra um aspecto do código de macro-assembler gerado. Cada linha deste código é comentada visando auxiliar o estudante na compreensão e finalidade das instruções geradas. As instruções criadas para este código, são basicamente as mesmas da máquina *Cesar* com adições no sentido de tornar mais fácil a leitura e o acompanhamento por parte do estudante. Desta forma, foram acrescentadas instruções para manipulação de pilha, rótulos explícitos e rótulos para cadeias de caracteres.

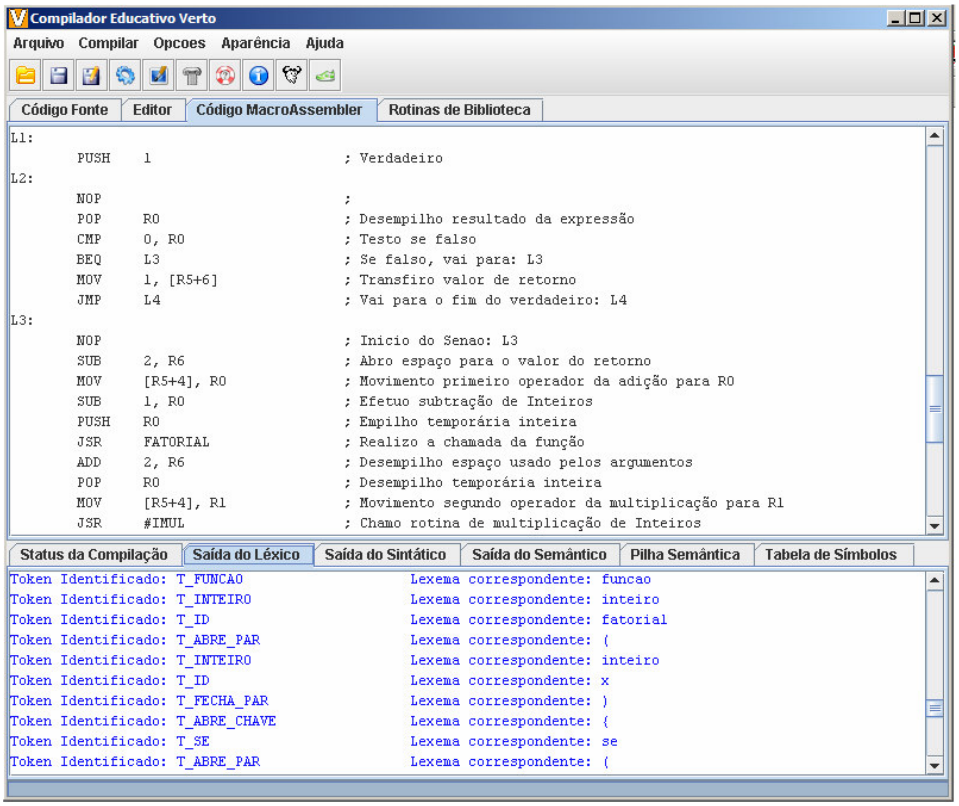

Figura 3: Tela com Código Macro-Assembler.

Como já foi mencionado, a linguagem Verto consiste de um conjunto bastante reduzido de instruções com adaptações nas funções de entrada e saída devido ao fato da máquina *Cesar* dispor de um visor limitado[iv]. O Verto, também, dispõe de um menu de ajuda para rotinas e sintaxe como apresentado na figura 4.

Figura 4: Auxílio à Sintaxe e Rotinas

Possibilidades educativas do VERTO

Antes de discutirmos as questões pedagógicas do Verto, apresentaremos as diferentes etapas do processo de interação possíveis entre o aluno e a ferramenta a fim de exemplificar os aspectos pedagógicos que serão destacados a seguir. A interação do aluno com o Verto é realizada seguindo as diferentes etapas que um compilador executa no seu processo. Isto é importante, pois como mencionamos antes, na aprendizagem de um conteúdo procedimental, o aluno não somente se apropria dos conceitos pertinentes como também das ações ou passos necessários incluídos dentro de um contexto (no

nosso caso o Verto) e que se traduz numa **ação cognitiva** (Rubstov, 1996) Assim, cada uma das fases da compilação dispõe de uma seção visualmente identificada com uma aba para que o aluno possa consultar o resultado e confrontar suas hipóteses (Figuras 2 e 3)

> pesquisas nas quais […] os estudantes de programação receberam ajudas gráficas que facilitavam a adquisição do modelo mental durante a aprendizagem mostraram uma melhoria na representação do conhecimento com relação a otros que não tiveram tais ajudas. (CAÑAS, BAJO y GONZALVO 1994, p.173 [tradução nossa])

Para elaborar um algoritmo o aluno dispõe de um editor de texto embutido no Verto. Uma vez finalizado o algoritmo, o aluno submete o programa-fonte ao compilador que gera o código macro-assembler intermediário e o código-objeto na linguagem da máquina hipotética *Cesar*. Após submeter um código na linguagem Verto, o estudante tem a oportunidade de consultar a saída de cada uma das fases da compilação. A Tabela de Símbolos, uma tabela onde constam os identificadores encontrados no código-fonte, é disponibilizada em uma aba para que o aluno possa consultar ao valores de endereços, tipos, e escopo registrados como apresentado na Figura 2. Ao realizar a análise do posicionamento dos rótulos, da lógica utilizada na tradução, do uso de endereços absolutos ou relativos o estudante pode construir um melhor entendimento das ações tomadas pelo compilador após o reconhecimento das regras sintáticas. Nesse sentido, ao confrontar suas hipóteses com os resultados obtidos o aluno desenvolve um raciocínio contextualizado fortemente embasado na resolução de problemas. A resolução de problemas é, segundo, Rubstov(1996), o processo de aquisição de "formas de ação gerais típicas", que implicam não somente a aprendizagem dos conceitos envolvidos numa ação, como a aprendizagem da própria ação e do contexto no qual ela se desenvolve. Ou seja, o problema de aprendizagem está estreitamente ligado ao conceito de "ação cognitiva" a qual enfatiza a análise do conteúdo e da estrutura para a resolução do problema de aprendizagem. Como afirma Rubstov (1996), completar uma ação cognitiva não é um processo trivial, pois o sujeito precisa desenvolver uma série de transformações que levem a uma reconstrução do objeto de aprendizagem.

Quanto a fase de Análise Léxica que envolve a verificação dos caracteres encontrados no programa fonte e o particionamento do programa em lexemas[v] válidos (Holmes, 1995), o aluno tem a possibilidade de visualizar as saídas do léxico, na qual os tokens[vi] e lexemas encontrados são listados (Figura 2).

A fase de Análise Sintática engloba, em geral, funções de grande importância: identificação de sentenças; detecção de erros sintáticos; ativação de rotinas de análise semântica; ativação de rotinas de código-objeto (José Neto, 1987). Entre as diversas técnicas de análise sintática passíveis de escolha, optou-se pela análise sintática descendente recursiva feita de forma não automática. Esta abordagem foi escolhida por ser simples, intuitiva e bastante utilizada em projetos de compiladores. Nesta fase, cada regra sintática reconhecida pode levar o compilador a disparar uma ação semântica ou a geração de uma parte do código. A aba denominada de "Saída do Sintático" registra cada regra reconhecida, de modo a ilustrar a seqüência de regras que foram reconhecidas pelo compilador (Figura 5).

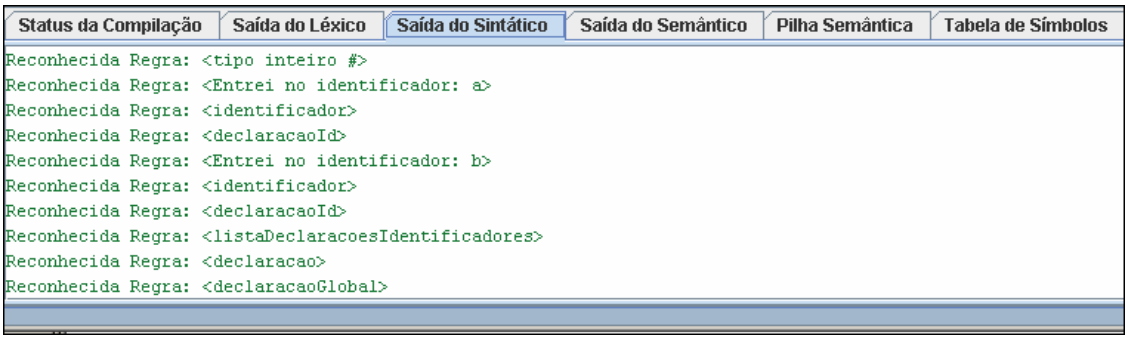

Figura 5: A aba: Saída do Sintático

A análise semântica envolve a computação de diversas informações que estão além da capacidade das gramáticas livres de contexto (Louden, 2004). Esta fase é caracterizada pela manutenção das tabelas de símbolos mapeando identificadores a seus tipos e endereços (Crespo, 1998). Nesta fase, os atributos são normalmente mantidos em uma pilha semântica. Quando o analisador sintático sinaliza o reconhecimento de uma regra, os *k* elementos do topo da pilha semântica são desempilhados para o tratamento da ação semântica. Caso haja algum resultado semântico envolvido, este é empilhado após a execução da ação semântica (Appel, 1998). Parta auxiliar o estudante na observação das variações ocorridas durante a análise semântica, uma das abas registra cada movimentação ocorrida na pilha semântica (Figura 6).

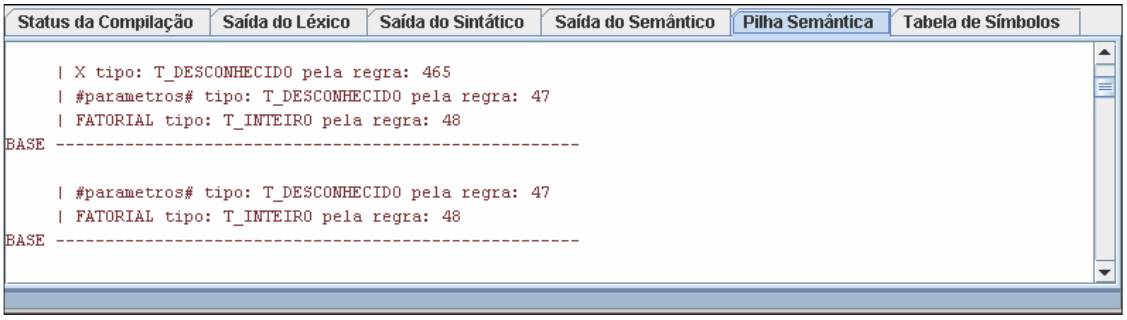

Figura 6: A aba: Pilha Semântica

Após compilar um determinado programa-fonte, sem erros, o estudante pode então, submeter o código-objeto gerado à máquina *Cesar*. A figura 7 ilustra a saída de um programa exemplo gerado a partir de uma rotina recursiva para cálculo do fatorial.

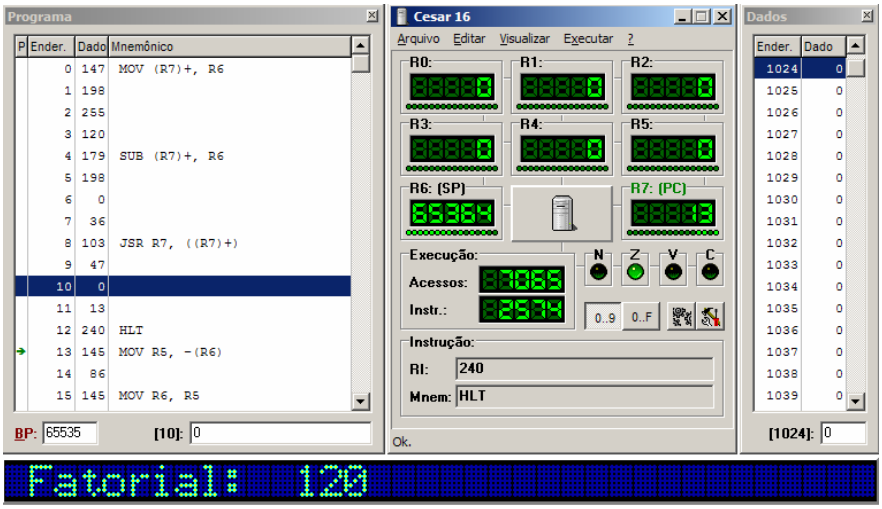

Figura 7: O resultado do processamento do programa fatorial.mem

Desta forma, visando expandir as possibilidades pedagógicas utilizadas para auxiliar a aprendizagem de compiladores, a ferramenta Verto fornece um ambiente interativo no qual o estudante pode criar programas relativamente pequenos (Kaplan, 1994) para, comparando o resultado gerado, formar conclusões a respeito de aspectos da tradução num processo de raciocínio e aprendizagem contextualizado.

Raciocínio contextualizado apóia-se, segundo Janvier (1996), em representações externas que permitem internalizar e transformar em representações internas de forma que o aluno não aprende somente sobre os conceitos sendo abordados senão que se apropria de todo o contexto que circunda a atividade de aprendizagem. O contexto é diferente da representação interna, pois define limites e condições onde se desenvolve a aprendizagem acrescentando variáveis adicionais que não encontram-se presente num processo de aprendizagem meramente teórico, na qual o aluno trabalha com representações abstratas desconectadas da realidade.

> Ao realizar uma aproximação entre a situação e sua representação [...] o raciocínio contextualizado favorece a articulação de variáveis correlatas e contribui para o sucesso do processo de resolução (JANVIER, 1996, p. 123)

E ainda mais, as representações do raciocínio contextualizado "*permitem colocar em maior relevo as relações essenciais ou fundamentais, servem também para organizar a solução (ou, ainda, a forma de colocar em ação o processo de resolução), calibrá-la e dirigi-la*" (JANVIER, 1996, p. 123).

Por isso, é preciso que o ensino utilize metodologias que adotem o **raciocínio contextualizado**, ou seja introduzam o **contexto** na aula favorecendo a aprendizagem contextualizada no ensino de compiladores. Esta abordagem traz como conseqüência um outro aspecto importante, que é a integração de disciplinas correlatas, num processo transdisciplinar embasado num enfoque globalizador da aprendizagem. Segundo Zabala (1998, 2002) os métodos globalizados organizam os conteúdos de forma transdisciplinar ao permitir que uma realidade seja explicada sem fragmentação, pois os conteúdos trabalhados procedem de diferentes disciplinas, como o caso de estrutura de dados, sistemas operacionais, linguagens, entre outras disciplinas que podem e devem estar co-relacionadas com compiladores através de um processo de ensino contextualizado e globalizador.

Portanto, Verto visa a construção do conhecimento pela interação entre o aluno e a ferramenta, através da experimentação com exemplos simples e a análise dos resultados obtidos pela compilação. Desta forma, o aluno pode testar suas hipótese e desenvolver experimentações que promovam a generalização e a contextualização da aprendizagem, assim como a troca de experiência e interação com os colegas e professores, respeitando o ritmo de cada aluno, de forma a criar um ambiente de aprendizagem consistindo"[...] *de um problema ou espaço de projeto (incluindo problema de contexto, problema de representação/simulação e problema de manipulação de espaço), casos relacionados, fontes de informação, ferramentas cognitivas, conversação, apoio contextual e social para as pessoas que participam*" (JONASSEN , 1996, p. 80). E que atenda as características definidas por Jonassen(1999) sobre aprendizagem construtivista:

- **Ativa:** no sentido de permitir o controle do processo para o aluno através da manipulação e da ação;
- **Construtiva:** ao permitir que o aluno construa seus próprios modelos mentais e

crenças com relação ao objeto em estudo e pela reflexão sobre a ação;

- **Reflexiva:** os alunos devem refletir sobre suas próprias experiências e sobre as experiências do grupo;
- **Intencional**: Todo comportamento humano tem um objetivo, que pode ser simples ou complexo. A aprendizagem está relacionada com a intencionalidade do sujeito por trás da ação executada;
- **Complexa**: contrária à noção da simplificação dos problemas, para um conhecimento que não seja "fragmentado". Problemas reais são complexos, mal-estruturados e geralmente envolvem diversas áreas do conhecimento;
- **Contextualizada**: A aprendizagem é um processo que acontece dentro de um contexto. Aprender sobre algo é aprender sobre isso dentro de um contexto;
- **Colaborativa/Cooperativa/Coloquial:** permitindo o diálogo, a troca de experiências, o trabalho em grupo pela colaboração/cooperação, a argumentação, o consenso e a discussão.

Considerações Finais e Trabalhos Futuros

Além das possibilidades educativas mencionadas, a ferramenta Verto pode ser utilizada também, como ferramenta auxiliar à outras disciplinas como Arquitetura de Computadores e Paradigmas de Linguagens de Programação. No estudo da máquina hipotética César, adotada por algumas universidades, a ferramenta Verto pode contribuir para a elaboração de problemas simples de forma rápida para posterior aperfeiçoamento por parte do aluno. Além disto, o estudante pode elaborar a solução do problema proposto inicialmente utilizando-se do *macro-assembler* criado para a linguagem de modo a ter facilitado o desenvolvimento do algoritmo. Na disciplina de Paradigmas de Linguagens de Programação, a visualização da Tabela de Símbolos, o tratamento dos tipos e a diferenciação do cálculo de endereço para variáveis estáticas e dinâmicas, são recursos da ferramenta que já vêm sendo explorados pelos alunos do Centro Universitário Feevale.

Notas

[i] A origem do nome Verto provém do latim que significa “verter” no sentido de tradução de uma linguagem para outra.
[ii] O projeto encontra-se disponível em: http://verto.sf.net
[iii] As fases de compilação estão implementadas em classes separadas de modo a permitir o estudo estanque das fases da compilação: Lexico.java, Sintatico.Java, Semantico.java e VertoAsm (geração do código).
[iv] O visor do César é uma linha de apenas 36 caracteres como pode ser visto na figura 7.
[v] Os lexemas são palavras ou conjuntos de símbolos válidos de uma determinada linguagem.
[vi] Os tokens são categorias associadas a um ou mais lexemas

Referências Bibliográficas

AHO, A. V.; SETHI, R.; ULLMAN, J. D. **Compiladores: princípios, técnicas e ferramentas**. Rio de Janeiro: LTC. 1995. 344 p.
APPEL, A. W. **Modern compiler implementation in Java**. Cambridge: Cambridge University Press. 1998. 547 p.

CAÑAS, J.J.; BAJO, M.T.; GONZALVO, P. (1994). **Mental Models and computer programming.** Intl. Journal of Human-Computer Studies, 40, 795-811.
COLL, C. et al. **O construtivismo na Sala de Aula.** São Paulo: Ed. Ática, 1996 (Série Fundamentos).
CRESPO, R. G. **Processadores de linguagens: da concepção à implementação**. Lisboa: IST Press. 1998. 435 p.
DEBRAY, S. Making compiler design relevant for students who will (most likely) never design compiler. **Proceedings of the 33th SIGCSE technical symposium on Computer science education**, Covington, Kentucky, USA, v.34, n.1, p. 341-345, mar. 2002. Disponível em: <http://delivery.acm.org/10.1145/570000/563473/p341-debray.pdf >. Acesso em: 14 out. 2005.
HENRY, T. R. Teaching compiler construction using a domain specific language. **Proceedings of the 36th SIGCSE technical symposium on Computer science education**, St. Louis, Missouri, USA, v.37, n.1, p. 7-11, fev. 2005, Disponível em: < http://delivery.acm.org/10.1145/1050000/1047364/p7-henry.pdf>. Acesso em: 14 out. 2005.
HOLMES, J. **Object-oriented compiler construction**. Englewood Cliffs, NJ: Prentice Hall. 1995. 483 p.
JANVIER, C. **Contextualização e representação na utilização da matemática.** Em: GARNIER, C. BEDNARZ, I. e ULANOVSKAYA, I. **Após Vygotsky e Piaget: perspectiva social e construtivista. Escola Russa e Ocidental.** Porto Alegre: Artes Médicas, 1996.P. 111-126
JONASSEN, D. et alli. **Learning with Technology. A constructivist perspective**. New Jersey: Prentice Hall, 1999.

JONASSEN, David. **O uso das novas tecnologias na educação à distância e a aprendizagem construtivista**. Em Aberto, Brasília ano 16 nº 70 abr/jun 1996. p. 70-88.
JOSÉ NETO, J. **Introdução à compilação**. Rio de Janeiro: LTC. 1987. 222 p.
KAPLAN, R. M. **Constructing language processors for little languages**. New York: John Wiley & Sons. 1994. 452 p.
LOUDEN, K. C. **Compiladores: princípios e práticas**. São Paulo: Pioneira Thomson Learning. 2004. 569 p.
METSKER, S. J. **Building parsers with Java**. Boston: Addison Wesley. 2001. 371 p.
PASSERINO, L. M. **Pessoas com autismo em ambientes digitais de aprendizagem : estudo dos processos de interação social e mediação**. Porto Alegre: PGIE/UFRG, 2005. 315 p. Tese de Doutorado
RANGEL, J. L. M. **Compiladores**. Disponível em: <http://www-di.inf.puc-rio.br/~rangel/comp/COMP1.ZIP>. Acesso em: 13 out. 2005.
RUBTSOV, V. **A atividade de aprendizado e os problemas referentes à formação do pensamento teórico em escolares.** In: GARNIER, C. BEDNARZ, I. e ULANOVSKAYA, I. **Após Vygotsky e Piaget: perspectiva social e construtivista. Escola Russa e Ocidental.** Porto Alegre: Artes Médicas, 1996.p.129-137
WEBER, R. F. **Fundamentos de arquitetura de computadores**. Porto Alegre: Sagra Luzzato. 2001. 299 p.
ZABALA, A. **A Prática Educativa: como ensinar.** Porto Alegre: ArtMed, 1998.p.224
ZABALA, Antoni. **Enfoque Globalizador e Pensamento Complexo :** uma proposta para o currículo escolar. Porto Alegre: Artmed, 2002.p.248