O estudo de compiladores apoiado pela pedagogia de projetos e por um ambiente virtual de aprendizagem

Silvana Rossy de Brito[1,2], Aleksandra do Socorro da Silva[1,2,3] , Eloi Luis Favero[1], Orivaldo de Lira Tavares[4], Renato Lisboa Francês[1]

[1]Laboratório de Computação Aplicada (LACA/GPEDN) - Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) -Universidade Federal do Pará (UFPA)
Caixa Postal 479 –66.075-110, Belém - PA – Brasil

{srossy, aleka, favero, rfrances}@ufpa.br

[2] Instituto de Estudos Superiores da Amazônia (IESAM/NUTEIA)
66055-260 - Belém – PA – Brasil

[3]Universidade da Amazônia (UNAMA)
CEP 66060-902, Belém – PA – Brasil

[4] Centro Tecnológico (CT) - Universidade Federal do Espírito Santo (UFES)
29060-900, Vitória – ES – Brasil

tavares@inf.ufes.br

Resumo. *Dentre as possibilidades pedagógicas utilizadas para auxiliar a aprendizagem de Compiladores, a pedagogia de projetos se destaca por favorecer a integração entre disciplinas e o desenvolvimento de habilidades envolvidas no processo de desenvolvimento de um projeto de engenharia de software. Este artigo aborda o estudo de compiladores segundo a abordagem de projetos, relatando algumas das dificuldades enfrentadas pelos professores e apresentando a experiência dos autores com a utilização de um ambiente virtual de aprendizagem no ensino da disciplina.*

Abstract. *Among the pedagogic possibilities used to aid the learning of Compilers, the pedagogy of projects stands out it goes favoring the integration between disciplines and the development of abilities involved in the process development of the project of software engineering. This article presents the compilers study in context the approach of projects, relating some of the difficulties faced by the teachers and presenting the authors experience with the use of a virtual environment of learning in the teaching of the discipline.*

1. Introdução

Dentre as possibilidades pedagógicas utilizadas para auxiliar a aprendizagem de Compiladores, a pedagogia de projetos destaca-se por valorizar a integração entre disciplinas, além de favorecer a cooperação e a colaboração entre estudantes. Essa abordagem é particularmente adequada em disciplinas onde o desenvolvimento do projeto aparece de forma faseada, como é o caso do projeto de um compilador. Para apoiar esse processo de construção, acreditamos ser imprescindível a utilização adequada de

tecnologias que apóiem a cooperação, característica considerada essencial em um ambiente virtual de aprendizagem (AVA), e que, atualmente, é tratada com a utilização dos recursos da *web*[1].

Este artigo apresenta o relato da experiência com a utilização de um ambiente interativo de aprendizagem na disciplina de compiladores. Está estruturado da seguinte forma: as seções 2 e 3 apresentam os objetivos do estudo de compiladores em cursos de computação e a abordagem de projetos; a seção 4 relata algumas das dificuldades enfrentadas pelos professores; a seção 5 apresenta a experiência com a utilização de um ambiente virtual de aprendizagem para apoiar o ensino de compiladores utilizando a abordagem de projetos e na seção 6, as conclusões deste trabalho.

2. Objetivos de um curso de compiladores

Compiladores são ferramentas de tradução entre linguagens, mantendo a semântica original.

Ao final da disciplina, espera-se que os estudantes tenham a percepção concreta de que ferramentas fundamentais como métodos formais, engenharia de software, estruturas de dados e algoritmos colaboram para resolver um problema complexo. Segundo Baldwin [2003], é importante que os estudantes aprendam como e por que são construídos os compiladores, além de como construí-los.

Segundo as diretrizes curriculares do MEC, qualquer que seja a abordagem utilizada para o ensino de compiladores, é exigido que o ensino de Compiladores assegure aos aprendizes a oportunidade de aplicação das técnicas estudadas no desenvolvimento de projetos de porte realístico, com implementações de alta correção e eficiência [MEC 2003].

A tabela 1 ilustra a abrangência da disciplina de compiladores, considerando duas principais dimensões: das atividades (**análise, síntese, síntese & análise)** e dos níveis lingüísticos (**léxico, sintático e semântico**); para cada célula desta tabela temos conceitos e técnicas diferentes, com a complexidade aumentando da esquerda para a direita e de cima para baixo. Na coluna da direita temos as 4 principais disciplinas envolvidas: LF= Linguagens Formais & Autômatos; TC= Teoria da Computação; ED = estrutura de dados; OC= Organização de Computadores (além do tema TDS= Tradução dirigida por sintaxe).

Tabela 1. Abrangência da disciplina de Compiladores

Análise	Síntese	Análise x Síntese	Disciplinas
Léxica		Erros-léxicos	LF; TC
		Tabela-símbolos	ED
Sintática		Erros-sintáticos	LF; TC
		Tabela-símbolos	ED
Semântica	TDS	Erros-semânticos	LF; ED
	TDS: Geração de código	Código de máquina:Geração & Otimização	LF; ED; OC

Assim, para atender os objetivos da disciplina e a necessidade de consolidar os conhecimentos de Teoria de Linguagens Formais e Autômatos e Teoria da Computação, a abordagem escolhida freqüentemente envolve o projeto de um compilador. É fundamental, que ao final do projeto de um compilador, o aprendiz desenvolva as habilidades necessárias

[1] *Web* é uma abreviatura da *world wide web* (www).

para justificar a escolha das ferramentas, ambientes, paradigmas e linguagens utilizadas [MEC 2003]. Muitos dos conceitos vistos apenas do ponto de vista teórico (como modularidade, manutenibilidade, portabilidade e custos de software) podem ser oportunisticamente analisados durante todo o desenvolvimento do projeto.

3. A pedagogia de projetos na aprendizagem de compiladores

Disciplinas como compiladores ou sistemas operacionais incluem freqüentemente um projeto no qual os estudantes implementam (no todo ou em parte) um pequeno exemplo do sistema em estudo. Assim, é comum o estudo de compiladores segundo a abordagem de projetos nos cursos de computação. Esses projetos encorajam a aprendizagem, fazendo com que os estudantes compreendam o assunto por meio de seus próprios exemplos. O desenvolvimento do projeto de um compilador ocorre de forma faseada, por meio da construção dos seus diversos componentes [Brito et al. 2003].

Com a abordagem de projetos, é possível favorecer a consolidação dos conteúdos estudados em Teoria da Computação como um todo e em Engenharia de Software e em Teoria de Linguagens Formais e Autômatos de forma integrada no projeto do compilador. Do contrário, o aprendizado dessas disciplinas, quando desvinculado da prática, tende a ser enfadonho, cansativo e pouco produtivo [Furtado 2003]. Uma metodologia de ensino integrada é essencial pois é na disciplina de compiladores onde acontece a consolidação entre os conceitos estudados e a prática da construção. Infelizmente, poucos professores da área de computação estão familiarizados com teorias educacionais e poucos especialistas em educação tentam aplicar teorias educacionais na educação em informática [Bem-Ari 2004], embora propostas construtivistas como a de Ben-Ari [1998 2001] e Hadjerrouit [1999] tentem resumir a teoria para professores de computação, fazendo com que cada vez mais, professores de computação mencionem explicitamente o construtivismo como a base teórica para propostas pedagógicas, para a utilização de software educacional e ambientes virtuais de aprendizagem.

Na concepção da pedagogia de projetos, os projetos se constituem em planos de trabalho e conjunto de atividades que podem tornar o processo de aprendizagem mais dinâmico, significativo e interessante para o aprendiz. Para o estudo de compiladores, essa abordagem freqüentemente envolve um projeto significativo onde os estudantes escrevem um compilador para uma linguagem de programação restrita. Segundo Aiken [1996], o projeto de um compilador freqüentemente tem dois objetivos distintos: a) apoiar o aprendizado sobre o projeto de uma linguagem e sobre a implementação de um compilador para essa linguagem; b), fornecer para os estudantes a experiência de construir um projeto de software significativo. No contexto de um curso de graduação em computação, um projeto de compilador pode ser a tarefa de construção de software mais complexa desempenhada pelos estudantes. Essa tarefa envolve a realização de pesquisas, a investigação e especificação de requisitos do software, o registro de dados, a formulação de hipóteses, a análise, aplicação e avaliação do artefato construído.

Para o sucesso da abordagem de projetos, o envolvimento dos aprendizes, a responsabilidade e a autonomia são fundamentais. Os aprendizes são co-responsáveis pelo trabalho e pelas escolhas ao longo do desenvolvimento do projeto [Tavares et al. 2001]. Em geral, essas escolhas são realizadas em equipe, motivo pelo qual a cooperação está também quase sempre associada ao trabalho de projetos. A construção de um compilador é uma tarefa complexa e exige a cooperação de um grupo de aprendizes para realizá-la. Essa

construção (especificar, escrever, testar, avaliar e documentar o compilador) constitui-se em um problema que exige o planejamento e a execução de um conjunto de atividades.

4. Dificuldades dos mediadores da aprendizagem de compiladores

Nesta seção discute-se as principais dificuldades dos mediadores em cursos de compiladores segundo a abordagem baseada em projetos. Essas dificuldades, parte relatada pela experiência dos autores e parte coletada na literatura especializada, permitiram definir uma classificação para os problemas enfrentados.

4.1. Quanto ao planejamento do curso

O projeto da linguagem é o primeiro problema enfrentado no planejamento de cursos de compiladores. Segundo Aiken [1996], especificações precisas devem ser escritas para apoiar as fases de projeto, implementação, testes e documentação do compilador. Idealmente, o projeto inteiro deveria ser implementado pelos projetistas do curso, antes de seu início, uma vez que somente uma implementação completa pode garantir que o projeto seja consistente, completo e rastreável. Entretanto, a idealização completa de um projeto de compilador freqüentemente não é viável para o professor. Pressões de tempo favorecem uma especificação em "tempo real" (enquanto o curso está acontecendo). Uma vez criado um projeto (especificação e implementação), o investimento feito já representa um forte incentivo para reusá-lo diversas vezes, em diferentes turmas. É assim que muitos professores criam projetos, repetindo tarefas de outros, havendo pouca reutilização entre instituições ou até mesmo entre professores da mesma instituição. Isso não tem acontecido em outras áreas que têm projetos amplamente compartilhados, como é o caso da disciplina de Sistemas Operacionais que disponibiliza simuladores, slides de cursos [Machado e Maia 2002] e sistemas operacionais pedagógicos [Christopher et al. 1993]. É fácil concluir que a condição de ensino melhora substancialmente se professores e instrutores compartilham os frutos de seu trabalho mais amplamente.

4.2. Quanto à linguagem a ser compilada

Na comunidade de linguagens de programação há um longo e útil debate sobre quais linguagens e conceitos de linguagens são importantes e devem ser ensinados. No contexto dos cursos de compiladores, as questões freqüentemente levantadas são: *Qual a linguagem a ser compilada ? Em que linguagem os estudantes devem escrever os seus compiladores ?* A experiência com a disciplina de compiladores reforça a idéia de [Aiken 1996] de que essas duas perguntas não são as questões mais fundamentais para a especificação de um curso de compiladores. Para atender aos objetivos da disciplina de compiladores, as preocupações que antecedem essas questões são: *o projeto está bem especificado? O projeto de compilador é possível de ser construído, documentado e avaliado?*. Ou seja, a preocupação está muito mais centrada na linguagem para a qual o compilador será desenvolvido do que nas linguagens e ferramentas nas quais será escrito. Assim, pode ser mais interessante inventar uma linguagem em vez de utilizar um subconjunto de uma linguagem existente, dado o fato que a linguagem inventada pode ser projetada para ser de fácil implementação em vez de ser fácil de utilizar (objetivo das linguagens reais). Além disso, os estudantes não ficam restritos às linguagens existentes, favorecendo a comparação entre linguagens e forçando os estudantes a pensarem conscientemente no significado das sintaxes das linguagens.

Com a abordagem de projetos, constatamos a vantagem em se combinar o uso de gramáticas lógicas para prototipação e especificação de linguagens de programação. Assim, os estudantes podem completar a especificação, em gramáticas lógicas, e a implementação (numa linguagem convencional tal como C++, Pascal e Java) de uma mini-linguagem imperativa, o que só é possível devido ao poder de abstração das gramáticas lógicas. Por exemplo, no quadro 1, temos um fragmento da especificação de um comando condicional IF/THEN/ELSE e no quadro 2, o trecho de uma gramática lógica que implementa a especificação. Nota-se que o código executável em Prolog é até menor que o código da especificação.

Quadro 1: Especificação do comando IF/THEN/ELSE [Slonneger e Kurtz 1995]

<command> ::= **if** <boolean expr> **then** <command sequence>1
else <command sequence>2 **end if**
Code(<command>) concat(*Code*(<boolean expr>),
[(JF,label(*InhLabel*(<command>)+1))],
Code(<command sequence>1),
[(J,label(*InhLabel*(<command>)+2))],
[(label(*InhLabel*(<command>)+1),LABEL)],
Code(<command sequence>2),
[(label(*InhLabel*(<command>)+2),LABEL)])…

Quadro 2: Gramática lógica [Slonneger e Kurtz 1995]

```
command(Code,Temp,InhLab,SynLab) -->
      [if], { InhLab1 is InhLab+1, label(InhLab1,Lab) },
      booleanExpr(Code1,Temp),
      [then], commandSeq(Code2,Temp,InhLab1,SynLab), [end,if],
      { concat(Code1, [['JF',Lab]],Code2, [[Lab,'LABEL']], Code) }.
```

4.3. Quanto ao acompanhamento da agenda do projeto

O desenvolvimento de um projeto de compilador é consumidor de tempo. Para Aiken [1996], é impossível para a maioria dos estudantes implementar qualquer coisa além do que uma pequena linguagem. A escolha de uma linguagem real (qualquer linguagem que tenha um significativo número de usuários) é viável apenas se o projeto do compilador envolve apenas um subconjunto desta (ex: linguagem MINILISP, representada por um subconjunto da linguagem LISP). É preciso observar, também, que existe pouco valor educacional na implementação de facilidades adicionais. Portanto, o professor deve atentar para não especificar riquezas de interface, por exemplo.

Os problemas que resultam das dificuldades de acompanhamento da agenda do projeto remetem às seguintes conseqüências: (1) o projeto pode não ser concluído (ficar incompleto), embora possa ter outros méritos; (2) os estudantes não terem tempo, no estágio final, para refletirem acerca da experiência vivida, diante dos demais compromissos do período letivo. É conveniente que o resultado de cada projeto desenvolvido seja materializado em um produto final, que pode ser apresentado para toda a turma. Assim, os artefatos de um pequeno grupo de aprendizes podem atrair o interesse de outros, favorecendo o refinamento do processo de aprendizagem.

4.4. Quanto às características do projeto especificado

Na especificação do projeto do compilador, um conjunto de características define as facilidades e os aspectos que devem ser abordados durante o seu desenvolvimento:

- Modularidade, portabilidade e reusablidade do projeto: Projetos de compiladores normalmente são divididos em fases (análise léxica, sintática, semântica e geração de código). As fortes dependências entre essas fases são um problema inerente do projeto do compilador: um estudante que faz um analisador léxico pobre, pode ser penalizado indiretamente no sintático, porque não será possível testar o sintático completamente quando o léxico ainda contém erros, e assim por diante. Da mesma forma, sem um gerador de código funcionando corretamente, um estudante que escreve o analisador semântico não pode testar a interface de geração de código. Deve-se buscar pela independência entre as tarefas. Idealmente, o projeto deve ser completamente modular: cada módulo (correspondente a uma fase) deve ser compilado separadamente e utilizado em qualquer outro compilador, desde que possua uma interface adequada àquele módulo [Aiken 1996]. Idealmente, o projeto também deve ser portátil entre plataformas, para garantir maior reusabilidade entre os diversos artefatos produzidos.

- "Memória" do projeto: O projeto deve ser novo evitando o desenvolvimento de uma "memória" (estudantes submetem como seu próprio trabalho, projetos de anos anteriores). Pequenas mudanças na especificação do projeto podem desanimar esses estudantes.

4.5. Quanto à linguagem com a qual o projeto será implementado

Finalmente, a escolha da linguagem na qual os estudantes escrevem seus compiladores não é trivial. É importante que essa escolha facilite a minimização de erros de codificação rotineiros que podem prejudicar o tempo de desenvolvimento e o entusiasmo pelo projeto. Uma desvantagem, nesse sentido, está na limitação de ferramentas para construção de compiladores que apóiem a programação em qualquer linguagem de programação. Essa decisão depende também das linguagens de programação utilizadas em outras disciplinas.

4.6. Quanto às ferramentas para construção de compiladores

Os cursos de compiladores utilizam, freqüentemente, ferramentas para construção de compiladores (variações do LEX e do YACC). Essas ferramentas são projetadas para agilizar o desenvolvimento do projeto, ocultando detalhes de implementação. Segundo Baldwin [2003], "compiladores para compiladores" são apóiam cursos onde a meta é ensinar os estudantes a *escrever* compiladores. Entretanto, como essas ferramentas escondem detalhes de implementação das diversas fases do compilador, elas não auxiliam muito quando se espera que os estudantes compreendam como o compilador funciona. Nesse sentido, as ferramentas visuais, tais como simuladores[2] [Kaplan e Shoup 2000], podem auxiliar os estudantes na compreensão do funcionamento das partes geradas automaticamente [Baldwin 2003].

---

[2] Simuladores são usados na educação com o intuito de proporcionar aprendizagem sobre conceitos que permeiam o sistema que está sendo simulado. Podem ser usados para imitar o funcionamento de partes do compilador, como simuladores de autômatos, para demonstrar o funcionamento do analisador léxico.

4.7. Quanto aos conteúdos didáticos utilizados

Diversos livros de compiladores [Apel e Ginshurg 1988] [Slonneger e Kurtz 1995] apresentam a especificação e a implementação de um compilador (como exemplo) para um subconjunto de uma linguagem real. Além dos exemplos existentes na literatura, outros compiladores (de outros projetos) podem ser apresentados como "estudos de caso" [Acebal et al. 2002]. Esses compiladores são bem aplicados como demonstrações, mas seus módulos nem sempre são tão reutilizáveis quanto é desejável, além do que eles não compilam pequenas linguagens, como seria o caso ideal [Baldwin 2003].

O próprio título do clássico livro de compiladores, conhecido como "o livro do dragão" (*Compiladores, princípios técnicas e ferramentas)* [Aho et al. 1986] - sugere que o objetivo final do estudo não necessariamente é a construção de um compilador mas sim estudar e compreender os princípios, as técnicas e as ferramentas usadas na construção de um compilador. Esses conhecimentos são úteis para inúmeras aplicações principalmente hoje que a maior parte da informação disponível está na Internet em forma de linguagem natural e de objetos semi estruturados (por exemplo em XML).

4.8. Quanto a colaboração entre os aprendizes

Deficiências na comunicação entre professores, estudantes e colaboradores afetam a colaboração entre os estudantes. O projeto será bem sucedido se a participação dos aprendizes e suas contribuições tiverem sido importantes para o grupo de maneira geral. No projeto, os artefatos são os produtos gerados ou consumidos nas diversas atividades durante a construção do compilador, podendo ser os produtos (ou subprodutos) gerados nas diversas fases de compilação.

4.9. Quanto a abordagem de projetos

De um modo geral, é possível reunir alguns problemas que derivam da utilização da abordagem baseada em projetos. Esses problemas caracterizam-se como pequenas armadilhas que podem prejudicar os objetivos da disciplina. São elas: (1) os estudantes que não conseguem completar as fases iniciais podem permanecer em desvantagem ao longo do projeto; (2) concentrados no projeto, os estudantes podem refletir pouco sobre os benefícios da aprendizagem; (3) o curto espaço de tempo pode comprometer a conclusão do projeto. Vale ressaltar que até mesmo um compilador pequeno pode amedrontar estudantes que não possuem experiências anteriores com a abordagem de projetos.

Uma solução para esses problemas está na modificação de uma implementação (na verdade, na implementação parcial do compilador), em vez de criar um compilador inteiro desde o início. Segundo Holland et al. [2002] essa abordagem é popular em cursos de sistemas operacionais.

5. Algumas soluções para os problemas enfrentados: utilizando um AVA para apoiar o estudo de compiladores

Com o advento das novas Tecnologias da Informação e Comunicação (TIC) surgiu um novo e amplo espaço de possibilidades na geração de softwares e ferramentas para apoiar a educação. Como exemplo, a cooperação e a colaboração entre pessoas geograficamente distantes , vêm sendo facilitadas com o apoio da teleinformática e dos

serviços[3] da web que suportam os ambientes virtuais de aprendizagem. Entre as vantagens obtidas com o uso de tais ambientes cita-se [Silva et al. 2003]: o reuso de conhecimento, o compartilhamento de informações e a cooperação, além de possibilitarem a interação entre pessoas com diferentes interesses e níveis de conhecimento.

A primeira experiência dos autores em utilizar um ambiente de aprendizagem baseado na Web, aconteceu em 2000, com a implantação do Laboratório Virtual de Compiladores [LABCOMP 2004]. Com a utilização contínua desse ambiente, foi possível aprimorar os projetos de compiladores, por meio do compartilhamento de projetos anteriores. Uma vez que projetos anteriores podem servir como referência aos novos projetos, os estudantes podem utilizá-los como exemplos de soluções para os novos projetos. Assim, os estudantes podem desenvolver a habilidade de comparar seus compiladores. Nesse contexto, Aiken [1996] sugere também que se utilize um compilador de referência para garantir que o projeto do compilador seja possível de ser construído pelos estudantes em um período curto de tempo. A experiência evoluiu para a utilização de um ambiente interativo de aprendizagem, o TelEduc [TelEduc 2004], em duas turmas de compiladores no ano de 2003. Essa experiência permitiu um levantamento de novas dificuldades e possibilitou a especificação de um novo ambiente [Harb et al. 2003] com novas facilidades de compartilhamento e colaboração entre os estudantes.

Para auxiliar os estudantes a compreenderem os benefícios da aprendizagem de compiladores, os cursos foram organizados segundo a abordagem de projetos, nos quais os estudantes escreveram pequenos compiladores. Os projetos foram importantes para expor os estudantes às fases de compilação. A idéia do curso é de até 50% da carga horária com teoria, laboratório e orientação e 50% para trabalho em grupo, onde são desenvolvidos projetos de compiladores para mini-linguagens, considerando uma disciplina com 60 a 90 horas aula. Isto só é possível com ferramentas de alto nível que permitem a especificação e prototipação das gramáticas das mini-linguagens, nos três níveis lingüísticos (léxico, sintático e semântico/geração-de-código). Pela nossa experiência, se fornecemos a especificação de uma gramática lógica, por exemplo, em uma disciplina de até 60 horas/aula, as equipes podem desenvolver seus projetos cobrindo as três principais fases, análise léxica, sintática e semantica/geração de código. Nesses casos, são pré-requisitos uma disciplina de Linguagens Formais, uma disciplina ou curso rápido de Programação em Prolog (20 horas no mínimo) e conhecimentos de programação e estruturas de dados. Utilizamos os textos de [Slonneger e Kurtz 1995] e [Favero 2004] que apresentam compiladores completos especificados e implementados em Prolog, com o uso de gramáticas lógicas. Em Prolog uma mini-linguagem pode ser descrita em algumas centenas de linhas, tipicamente de 100 a 300 linhas. Estes compiladores foram o ponto de partida do desenvolvimento dos projetos. Adicionalmente, mostramos como podemos codificar uma gramática lógica em uma linguagem convencional como C++, Pascal, Java. Basicamente adotamos uma abordagem descendente recursiva, onde cada regra gramatical é codificada como uma função booleana. Os atributos da regra são passados como parâmetros. Com isso o aluno é guiado na codificação pela especificação executável em Prolog.

No planejamento dos projetos de compiladores, sugerimos que os seguintes aspectos sejam observados, considerando o contexto do professor:.

---

[3] Esses serviços incluem áreas para compartilhamento de arquivos (escaninhos para entrega de trabalhos; estantes para a publicação de documento, programas e para a especificação de trabalhos); fóruns de discussão; *chats*; e-mails; ferramentas para acompanhamento do projeto; etc.

- o projeto precisa ser factível no tempo da disciplina, por estudantes sem experiência prévia na construção de compiladores;
- a linguagem a ser compilada deve ser uma linguagem restrita. É aconselhável que o professor não especifique recursos ou interfaces para o compilador que comprometam, com detalhes de implementação, o desenvolvimento incremental e o tempo de execução do projeto;
- o projeto especificado deve ser altamente modular, de modo que a solução dos estudantes possa ser substituída por outra (fornecida pelo professor) e deve ser tal que os estudantes possam utilizar uma infra-estrutura de código fornecida pelo professor. Em nossa experiência, a dependência entre os módulos é minimizada implementando-se a comunicação entre os módulos a partir de arquivos em memória secundária;
- o projeto do compilador deve ser instrumentalizado com porções de código, freqüentemente incompletas, mostrando representações intermediárias de programas, de modo que auxilie os estudantes a compreender o funcionamento interno do compilador;
- módulos reutilizáveis são descritos e apresentados como exemplos de sala de aula, sem fornecer soluções completas, mas de modo que possam ser utilizados em outros projetos nos períodos subseqüentes;

No contexto do aluno, o processo de desenvolvimento do projeto do compilador deve ser especificado e descrito pelos estudantes em documento padrão de Especificação de Requisitos de Software (ERS). A arquitetura e o código devem exemplificar conceitos estudados na engenharia de software e a documentação do projeto deve ser rastreável;

Com a utilização dos recursos disponíveis em ambientes interativos de aprendizagem, foi possível observar que os projetos de compiladores podem ser amadurecidos, compartilhados e aperfeiçoados por diferentes instituições. Além disso, os estudantes se sentem encorajados a publicar gratuitamente seus projetos, o que favorece o sentimento, nos estudantes, de que a disponibilização de seus projetos gratuitamente na rede pode gerar novas linguagens para a comunidade. Adicionalmente, com o auxílio das mensagens postadas nas ferramentas de comunicação, a documentação das ferramentas utilizadas é aperfeiçoada utilizando-se um recurso de "Perguntas mais Freqüentes".

Outra consideração diz respeito às horas didáticas do professor. No sistema educacional tradicional, essas horas freqüentemente são insuficientes para os estudantes. Entretanto, para os estudantes que possuem acesso à Internet, ferramentas de colaboração e pesquisa, as facilidades disponíveis podem ser oportunas e convenientes para estudar o material no seu próprio ritmo.

Os ambientes virtuais de aprendizagem permitem a incorporação de *Applets* Java[4] que podem ajudar a criar um ambiente interativo de "aprender fazendo", através de simuladores (no Laboratório de Compiladores, *applets Java* são utilizados para simular o comportamento de analisadores léxicos e sintáticos como autômatos). Além de demonstrar conceitos, esses recursos podem aumentar a motivação e instigar um maior interesse entre estudantes.

O compartilhamento é substancialmente potencializado com a utilização dos recursos em rede. Um típico projeto de compilador utiliza muitas estruturas de dados

[4] no Laboratório Virtual de compiladores, utilizamos *applets* Java para simular o comportamento do analisador léxico, representando as transições de estados no autômato.

"padrões" e possui algum nível de código repetitivo e de baixo nível (ex. *templates* para as instruções assembly). Segundo as diretrizes curriculares, a matéria Compiladores deve ser precedida do estudo de conceitos teóricos de linguagens e autômatos, sistemas operacionais e arquiteturas de computadores. Entretanto, presumivelmente os estudantes tiveram um curso de estruturas de dados (ou pesquisa operacional) antes do curso de compiladores e podem, já possuir, componentes que podem ser utilizados no projeto do compilador (estrutura *hashing*, por exemplo). Com os recursos de compartilhamento existentes em ambientes interativos de aprendizagem, esse código de apoio pode estar disponível e documentado em uma área compartilhada e pode ser reusado em diferentes projetos. Fornecer uma área compartilhada para o código de apoio (incluindo toda a descrição desse código e de suas interfaces) fornece um nível moderado de abstração, removendo uma grande porcentagem de possíveis armadilhas, além de permitir que os estudantes focalizem seus esforços nos aspectos mais importantes do projeto.

Uma área de compartilhamento é fundamental, para que os estudantes realizem as reflexões necessárias sobre as possíveis soluções no projeto de compilador. Se os estudantes tiverem uma única opção de recurso, dificilmente realizarão essas reflexões. O compartilhamento dos projetos permite a avaliação e a comparação entre os projetos desenvolvidos. Da mesma forma, a disponibilização de porções intermediárias de código enriquecem as opções para os estudantes. Segundo Rivas [2002], a riqueza de opções é de grande importância, constituindo-se uma medida do nível de estruturação do curso. Indubitavelmente, a riqueza de alternativas para conceitos e ferramentas constitui-se em um valioso arsenal para o estudante.

A percepção dos aprendizes é valorizada com a utilização dos recursos telemáticos. Permite aos estudantes compreenderem as atividades de outros e ajustarem suas próprias atividades através da reflexão sobre os resultados alcançados. Na experiência com o TelEduc, a percepção dos aprendizes foi valorizada pela abordagem chamada *feedback* compartilhado [Togneri et al. 2002], que consiste em coletar e apresentar para a comunidade, de forma automatizada, informações sobre as atividades individuais dos aprendizes, dentro de um espaço de trabalho compartilhado. Esse espaço, no TelEduc, foi construído com a utilização dos portifólios individuais e de grupos, que permitiram transmitir um senso de atualização contínua das ações individuais e o progresso global da turma. A grande vantagem de utilizar essa ferramenta está no fato de que os estudantes estão comunicando suas atividades a outros, refletindo, dessa forma, sobre suas ações e permitindo que os demais possam fazer comentários sobre elas e observar as conseqüências das ações efetuadas. Os portifólios representam espaços (pastas e arquivos) onde os estudantes postam os artefatos (códigos, documentação, etc.) produzidos nas fases do projeto. Esses artefatos podem ser comentados por professores ou por outros estudantes, desde que o autor tenha compartilhado o artefato.

6. Conclusões

Os ambientes interativos de aprendizagem têm sido propostos para apoiar comunidades virtuais de aprendizagem, visando a construção coletiva de conhecimento. Este artigo relatou a experiência dos autores no ensino de compiladores e reflete as idéias desenvolvidas através dessa experiência, favorecendo a utilização de AVAs como forma de apoiar o processo de construção do conhecimento. É improvável, entretanto, que a experiência de aprender sobre compiladores com apoio telemático seja única. Apesar disso, por se tratar de uma disciplina onde a abordagem de projetos freqüentemente se mostra mais adequada, acreditamos que a experiência com a disciplina de compiladores permita a

especificação de requisitos e o projeto de AVAs que valorizem a percepção, o reuso de conhecimento e facilitem o planejamento de cursos por meio do compartilhamento de recursos, até mesmo entre grupos de mediadores e aprendizes de diferentes instituições.

Os problemas levantados contribuem para especificação de requisitos iniciada por Silva et al. [2003], e coletados após a experiência dos autores em diferentes instituições e com diferentes AVAs. A continuação deste trabalho consiste, portanto, em desenvolver um ambiente com componentes de software que atendam aos requisitos especificados.

Referências

ACEBAL, C. F.; CASTANEDO, R. I.; LOVELLE, J. M. C. "Good Design Principles in a Compiler University Course", ACM SIGPLAN Notices, April 2002 (37:4). p. 62 – 73.

AHO, A. V., SETHI, R., ULLMAN, J. D. "Compilers: priciples, techniques, and tools". Massachusetts: Addison Wesley Publishing Co., 1986.

AIKEN, A. "Cool: A Portable Project for Teaching Compiler Construction". *ACM Sigplan Notices* 31(7), pages 19-26, July, 1996. Disponível em <http://theory.stanford.edu/~aiken/publications/papers/sigplan96.ps>

APPEL, A.; GINSHURG, M. "Modern compiler Implementation in C". Austrália: Cambridge University Press, 1988.

BALDWIN, D. "A Compiler for Teaching about Compilers". Proceedings of the Technical Symposium on Computer Science Education 34th SIGCSE'03 Technical Symposium On Computer Science Education, fev. 2003. p. 220-223. Reno, Nevada, USA. ISSN:0097-8418. Disponivel em <http://doi.acm.org/10.1145/611892.611974>.

BEN-ARI, M. "Constructivism in computer science education". In: *Journal of Computers in Mathematics and Science Teaching,* 20(1), 2001, 45–73.

BEN-ARI, M. "Situated learning in computer science education". *Computer Science Education* (aceito para publicação). Disponível em: <http://stwi.weizmann.ac.il/g-cs/benari/articles/sit-cs.pdf>. Acesso em 21 mai 2004.

BEN-ARI, M. Constructivism in Computer Science Education. Proceedings of the twenty ninth SIGCSE technical symposium on Computer science education , 1998, P. 257 - 261.

BRITO, S.R., TAVARES, O.L., MENEZES, C.S. "Um ambiente virtual para aprendiza-gem de compiladores com suporte inteligente à mediação. In: Internacional". In: *Conference on Engineering and Computer Education*, Santos. ICECE/COPEC, 2003.

CHRISTOPHER, W., PROCTER, S., ANDERSON, T. The Nachos instructional operating system. In: Winter USENIX Conference. pages 479-488. January, 1993.

FAVERO, E.L. "Programação em Prolog: Uma abordagem prática", Editora da UFPA. (em publicação, 2004)

FURTADO, O. J. V. "O ensino de Linguagens Formais vinculado ao ensino de Compi-ladores". In: XI Workshop de Educação em Computação. Disponível: <http://bioinfo. cpgei.cefetpr.br/anais/SBC2003/pdf/arq0056.pdf>. Acesso em: 15 dez 2003.

HADJERROUIT, S. "A constructivist approach to object-oriented design and programming". Proceedings of the 4th annual SIGCSE/SIGCUE on Innovation and technology in computer science education , 1999, p. 171 - 174.

HARB, M. P. A. A., BRITO, S. R., SILVA, A. S., FAVERO, E. L., TAVARES, O. L., FRANCÊS, C. R. L. "AmAm: ambiente de aprendizagem multiparadigmático". multiparadigmático. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2003, Rio de Janeiro. Rio de Janeiro: NCE-IM-UFRJ. p. 223-232.

HOLLAND, D.; LIM, A.; SELTZER, M. "A New Instructional Operating System". Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education, Mar. 2002. p. 111 – 115.

KAPLAN, A; D. SHOUP. "CUPV — A Visualization Tool for Generated Parsers", Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education, Mar. 2000. p. 11 –15.

LABCOMPL. Laboratório Virtual de Compiladores. Disponível em: <http://www.inf.ufes.br/~tavares/labcomp2000/>. Acesso em 10 jan 2004.

MACHADO, F. B., MAIA, L. P. "Arquitetura de Sistemas Operacionais". 2 ed . Rio de Janeiro: LTC, 1997.

MEC. "Diretrizes Curriculares de Cursos da área de Computação e Informática". Disponível:<http://www.mec.gov.br/sesu/ftp/curdiretriz/computacao/co_diretriz. rtf>.

RESLER, D.; DEAVER, D. M. "VCOCO: A Visualisation tool for teaching compilers". Proceedings of 6th annual conference on the teaching of computing, Dublin City Univ., Ireland, 1998, p.199-202.

RIVAS, L. G. R. "Some Thoughts on the Teaching of Informatics Engineering". Disponível em < http://luisguillermo.com/english/default.htm>. Acesso em 10 jan 2004.

SILVA, A.S.; BRITO, S.R.; FAVERO, E.L.; HERNÁNDEZ-DOMÍNGUEZ, A. TAVARES, O.L.; FRANCES, C.R.L. "Uma arquitetura para desenvolvimento de ambientes interativos de aprendizagem baseado em agentes, componentes e framework". In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. Rio de Janeiro: NCE-IM-UFRJ, 2003. p. 203-212.

SLONNEGER, K; KURTZ, B. L. "Formal Syntax and Semantics of Programming Languages: A Laboratory-Based Approach", Addison-Wesley, NY, 1995.

TAVARES, O. L; BRITO, S. R.; SOUZA, R. S. ; MENEZES, C. S. "Ambiente de apoio à mediação de aprendizagem: Uma abordagem orientada por processos e projetos". *Revista de Informática na Educação*, set., 2001, p. 77-87.

TelEduc. "Ambiente de Ensino a Distância". Disponível em: http://teleduc.nied.unicamp.br/teleduc/. Acesso em 15 jan 2003.

TOGNERI, D. F., FALBO, R. A., MENEZES, C. S. "Supporting Cooperative Requirements Engineering with an Automated Tool". In: Workshop on Requirements Engineering, 5., 2002, Valência – Espanha. Anais… Valência: CYTED, 2002.