DRAWCODE: UMA FERRAMENTA PARA A GERAÇÃO DE CÓDIGO A PARTIR DE DIAGRAMAS DE CLASSE E DIAGRAMAS N-S

Maurício Capobianco Lopes[1], André Henkels[1], Francisco A. P. Fialho, Dr.[2]

[1]Departamento de Sistemas e Computação - Universidade Regional de Blumenau (FURB)
Rua Braz Wanka, 238 – 89.035-260 – Blumenau, SC – Brasil

[2]Departamento de Engenharia e Gestão do Conhecimento - Universidade Federal de Santa Catarina – Florianópolis, SC - Brasil

mclopes@furb.br, andre.henkels@gmail.com, fapfialho@terra.com.br

Abstract. *This work presents DrawCode, an Eclipse plugin that aims to assist the object-oriented codification process, using N-S diagram notation, considering class diagrams specifications, and further the generation code in a programming language, gathering both diagrams information. The plugin allows both N-S and class diagrams editing and uses templates for code generation. DrawCode is an available alternative for educational use in initial courses, gathering both OO benefits and legible graphical notation.*

Resumo. *Este trabalho apresenta o Drawcode, um plugin do Eclipse cujo objetivo é auxiliar o processo de codificação de softwares orientados a objetos, usando a notação de diagramas N-S, a partir das especificações contidas nos diagramas de classe, e a posterior geração do código em uma linguagem de programação, associando os elementos dos dois diagramas. A ferramenta permite a edição de diagramas de classe e N-S e conta com templates para o processo de geração de código. O DrawCode se apresenta como uma alternativa viável para uso em ambientes educacionais, sobretudo nas séries iniciais de formação, por permitir aliar os benefícios da OO com a legibilidade de uma notação gráfica.*

1. Introdução

Os grandes desafios da educação em computação propostos pela Sociedade Britânica de Computação (BRITISH COMPUTER SOCIETY, 2008) destacam, entre outros, a necessidade de compartilhamento de boas idéias e práticas entre as instituições de ensino, sobretudo nos aspectos que influenciam os estudantes que iniciam os cursos na área, bem como chama a atenção para a necessidade de proporcionar aos estudantes métodos que os permitam entender os problemas e desenvolver suas próprias soluções.

Neste contexto, um dos grandes desafios tem sido a maneira de se levar aos programadores iniciantes o aprendizado da lógica de programação de forma motivadora e com sucesso. Assim, a etapa de codificação de sistemas - ou programação - tem sido objeto de estudos de pesquisadores de educação em computação, sobretudo por ser realizada no início do processo de aprendizagem na maioria das escolas, precedendo as etapas de análise e projeto.

O advento da orientação a objetos (OO) trouxe um novo ingrediente para esta questão, pois o ensino de algoritmos sempre esteve fundamentado nos princípios da programação estruturada, cujo paradigma e forma de construção da solução dos problemas muitas vezes são incompatíveis com os princípios da OO.

Atualmente a orientação a objetos (OO) tem se firmado como o método que permite uma visão mais natural de representação de um problema. A OO agrupa os elementos conforme suas propriedade e comportamentos, possibilitando uma visão mais parecida com o mundo real. Assim, as metodologias de ensino para os estudantes em fases iniciais também devem ser readaptadas a esta nova realidade, sendo que a mudança de um paradigma para outro exige uma combinação de métodos, técnicas e ferramentas.

O foco deste artigo, entretanto, não é discutir uma metodologia completa de ensino de programação OO, mas demonstrar uma ferramenta que pode auxiliar em uma das etapas deste processo unindo, potencialmente, os benefícios da OO com as vantagens da especificação de algoritmos baseada na construção de diagramas.

O software OO tem sido construído a partir da *Unified Modeling Language* (UML), que segundo Fowler (2005, p. 25) é uma família de notações gráficas que ajudam na descrição e no projeto de sistemas de software. A UML é formada por diversos diagramas, que apóiam as diferentes etapas do processo de construção de um software. Entretanto, uma das carências da UML é a representação detalhada do corpo dos métodos de uma determinada classe, uma vez que os diagramas existentes normalmente trabalham em níveis de abstração mais elevados, delegando ao programador a tarefa de interpretar e codificar a solução algorítmica para os métodos diretamente em uma linguagem de programação.

Assim, baseado na notação dos Diagramas de Nassi-Shneiderman (N-S) (NASSI; SHNEIDERMAN, 1973), este artigo apresenta o DrawCode (Henkels, 2007), uma ferramenta, na forma de um *plugin* para o ambiente Eclipse, que permite a edição de diagramas de classe e a codificação de métodos das classes por meio de notações gráficas N-S, com sua respectiva geração de código por meio de *templates*, já estando disponível atualmente o *template* para a linguagem Java .

Na Seção 2 são apresentadas as motivações para a construção de ferramentas que apóiem o desenvolvimento de software OO. Na Seção 3 são apresentadas as vantagens e características dos diagramas N-S. A Seção 4 trata da geração de código que é um dos aspectos importantes da ferramenta, haja vista a produtividade conferida ao processo de codificação. A Seção 5 apresenta a ferramenta desenvolvida, bem como algumas características e detalhes de implementação. A Seção 6 discute a relação da ferramenta com seus correlatos, bem como suas limitações e, finalmente, a Seção 7 apresenta algumas considerações finais sobre o trabalho.

2. Orientação a Objetos

A orientação a objetos é um paradigma de análise, projeto e programação de sistemas baseado na composição e interação entre diversas unidades de software chamadas de objetos (ORIENTAÇÃO A OBJETO, 2007). Tonsig (2003, p. 166) define objeto como “a representação de elementos físicos do mundo real, que sob o ponto de vista do problema a ser resolvido, possuem atributos e métodos comuns”. Portanto, um objeto é composto por atributos que representam suas características e métodos que são seus

comportamentos. Normalmente um software é composto por uma grande quantidade de objetos, que são agrupados em classes de objetos. Uma classe representa um conjunto de objetos com as mesmas características e comportamentos.

Para ajudar a entender estas classes de objetos e como elas se relacionam no contexto do software a UML prevê alguns diagramas (PILONE, PITMAN, 2006, p. 5). No processo de codificação, alguns diagramas destacam-se como muito necessários: o de classes que apresenta as classes do software e suas relações com as demais; o de atividades que demonstra o fluxo de comportamento dos objetos; o de comunicação que demonstra as mensagens trocadas entre os objetos; e o de seqüência que apresenta os elementos envolvidos, o tipo e a ordem das mensagens enviadas entre os elementos durante a execução do software.

Entretanto, uma das limitações da UML é a falta de um diagrama que auxilie o programador no processo de codificação de um software. A codificação consiste basicamente na construção dos algoritmos que vão dar a forma e o conteúdo ao software de forma a permiti-lo atingir os objetivos para o qual foi especificado. Assim, o uso de uma notação gráfica como é o caso dos diagramas N-S, aliado às características dos sistemas OO, pode agregar ainda mais produtividade e qualidade na especificação deste tipo de software, além de potencialmente facilitar a construção e o entendimento das soluções algorítmicas.

3. Diagramas N-S

Segundo Carboni (2003, p. 12-24), existem várias formas de descrever a lógica para solução de um problema e uma das mais utilizadas é o algoritmo, com suas formas de representação. O algoritmo pode ser considerado uma seqüência de procedimentos finitos que serão executados em determinada ordem para atingir certo objetivo.

É possível representar algoritmos de várias maneiras. Souza (2000, p. 21) cita: português estruturado, pseudocódigo, fluxogramas e diagrama N-S. Alguns são representações textuais (português estruturado e pseudocódigo) e outras gráficas ou visuais (fluxogramas e diagramas N-S). A representação gráfica facilita a estruturação do raciocínio e, consequentemente, minimiza o esforço para obtenção da solução do problema (SOUZA, 2000, p. 9), entretanto, por sua própria característica, são mais difíceis de serem representados sem a ajuda de uma ferramenta.

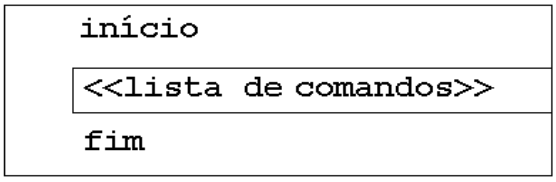
Os diagramas N-S são melhores para representar a solução estruturada de um algoritmo que os fluxogramas, pois estes dão uma visão não estrutural do problema e tendem a apresentar muitos saltos. Deste modo, neste trabalho optou-se por usar a notação deste tipo de diagrama visando perceber sua efetividade como apoio à codificação de software OO.

O diagrama N-S foi apresentado em 1973 por Isaac Nassi e Ben Shneiderman (NASSI; SHNEIDERMAN, 1973). Eles citam algumas vantagens do diagrama N-S: escopo das iterações e dos comandos de decisão e seleção é bem definido e visível, não existe transferência de controle arbitrária e a recursividade tem uma representação muito simples. Os diagramas N-S apresentam como desvantagem o problema do dimensionamento das estruturas, que normalmente obriga o desenvolvedor a redesenhá-lo novamente, consumindo muito tempo na sua elaboração, tornando-os quase inviáveis sobretudo nos processos educativos, baseados em lápis e papel. Isto fez com que eles fossem abandonados como representação algorítmica, porém, é possível eliminar este

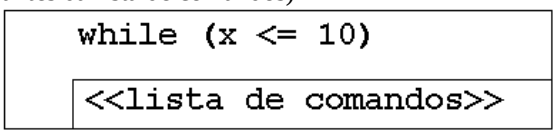
problema utilizando-se de ferramentas computacionais que automaticamente redimensionem o diagrama conforme a necessidade.

O diagrama N-S é formado por seis elementos básicos apresentados na Figura 1.

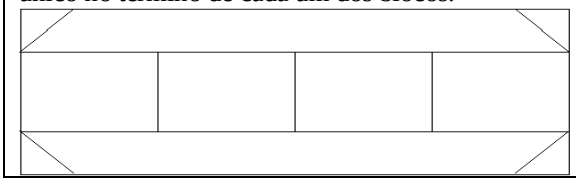
Bloco de Comandos: agrupa uma lista de comandos a serem executados formando um bloco.



Repetição: indica um grupo de comandos a serem executadas até que determinada expressão lógica deixe de ser verdadeira. Podem ser representadas de dois modos: com a expressão antes da lista de comandos ou com a expressão depois da lista de comandos (no exemplo a seguir a condição está antes da lista de comandos)

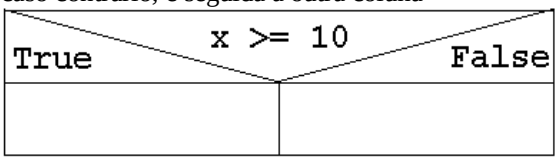


Paralelo: indica que o algoritmo em determinado ponto vai ter mais de um bloco de instruções executados ao mesmo tempo, retornando a um único no término de cada um dos blocos.

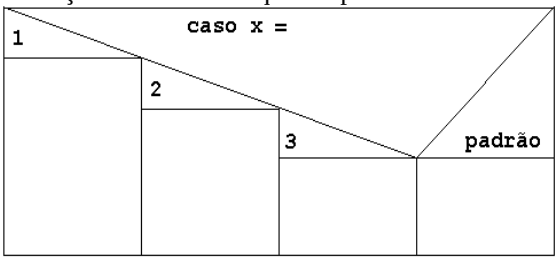


Processo: indica uma ação a ser executada, a qual pode ser um método a ser invocado ou uma atribuição.

Decisão: indica uma expressão lógica a ser verificada para desviar o fluxo de execução do algoritmo em duas possibilidades: se a expressão for verdadeira uma coluna de execução é seguida, caso contrário, é seguida a outra coluna



Seleção: indica que com base no valor de uma determinada expressão, uma das colunas de execução será seguida. Cada uma das colunas de execução tem um valor que a representa



Fonte: adaptado de Nassi e Shneiderman (1973)

Figura 1 - Simbologia dos Diagramas N-S .

A implementação de soluções de problemas baseadas em diagramas via de regra precisam ser codificadas em uma linguagem de programação, de modo que se possa tratar os dados e processar a solução. Geralmente esta etapa é feita manualmente, entretanto, sempre que possível, os programas devem ser gerados utilizando técnicas de geração de código que permitem produzir código sem nenhum erro de sintaxe a partir do projeto.

4. Geração de Código

Herrington (2003, p. 15) afirma que técnicas de geração de código podem trazer benefícios como consistência ao definir um padrão para nomes de variáveis, produtividade para o desenvolvimento de programas onde classes são geradas rapidamente, e qualidade, pois o código gerado é uniforme e superior ao criado numa codificação manual. Tal técnica também possibilita abstração em nível de linguagem, sendo possível gerar programas em outras linguagens de programação utilizando *templates*, sem comprometer a lógica envolvida.

Templates são basicamente arquivos com marcações especiais, que devem levar em consideração o motor que vai ser utilizado. O motor de *templates* é um artefato de

software que possibilita a junção de um conjunto de informações a um arquivo de marcações. Um dos motores de *templates* mais utilizados atualmente é o Velocity (APACHE SOFTWARE FOUNDATION, 2007) que foi escrito na linguagem Java e é um projeto de código aberto. Suas indicações são inúmeras com destaque para criação de páginas com informações dinâmicas, geração de código, mensagens pré-formatadas e transformação de dados em formato XML. Os arquivos de *template* criados para serem utilizados no Velocity, devem ser escritos conforme a *Velocity Template Language* (VTL) que provê comandos para controlar o funcionamento do motor no momento do processamento do arquivo de *template*.

5. DrawCode

Baseado nas premissas apresentadas anteriormente, foi especificada e construída a ferramenta DrawCode (Henkels, 2007), que é um *plugin* para o ambiente de desenvolvimento Eclipse que possibilita gerar código a partir de diagramas de classe e diagramas N-S, utilizando para isso o motor de *templates* Velocity.

O Eclipse é um projeto de código fonte aberto com o objetivo de criar uma plataforma de desenvolvimento composto de várias partes extensíveis e ferramentas para apoiar a construção, implantação e todo o ciclo de desenvolvimento de programas (ECLIPSE FOUNDATION, 2007). O Eclipse foi projetado para ser extensível, provendo mecanismos que permitem serem adicionadas funcionalidades ao ambiente de desenvolvimento por meio dos chamados *plugins* (BARROS et al., 2003).

As funcionalidades necessárias pesquisadas para o funcionamento do *plugin* DrawCode são apresentadas na Figura 2 na forma de requisitos funcionais.

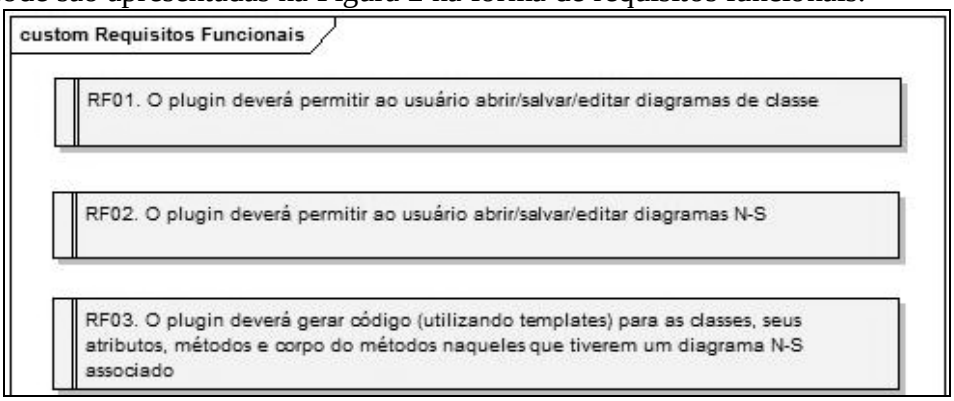


Figura 2 - Requisitos funcionais do *plugin*

Por se tratar de um *plugin* para o ambiente Eclipse, o mesmo deve ser utilizado dentro daquele ambiente. A Figura 3 apresenta a tela do Eclipse onde o usuário criou um projeto chamado projeto_drawcode_01 e dentro dele criou um diagrama de classes chamado diagrama_classe.drcd e dois diagramas N-S, nomeados como metodo1.nsd e metodo2.nsd.

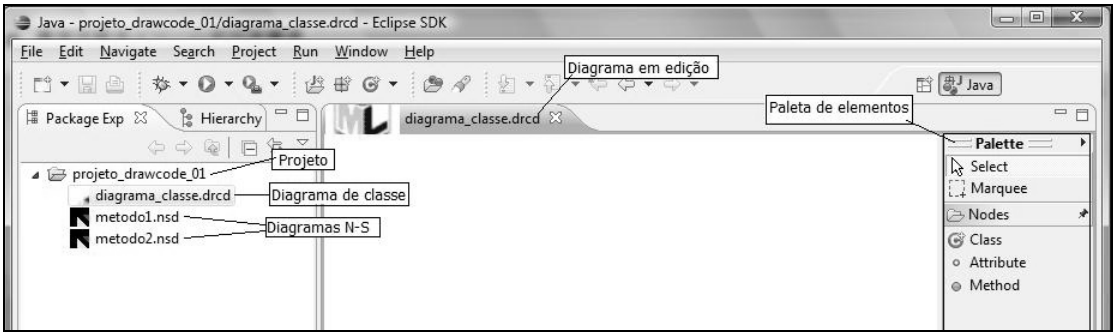


Figura 3 – DrawCode no ambiente Eclipse

A Figura 4 apresenta um diagrama de classes em edição, onde foi definida a classe `Aluno` com os atributos `nota1`, `nota2`, `nota3` e o método `getSituacao`. Para incluir uma classe, o usuário clica na paleta sobre o elemento `Class` e em seguida em alguma parte do diagrama. Para incluir atributos ou métodos clica-se nos botões `Attribute` ou `Method` e em seguida na área respectiva no diagrama.

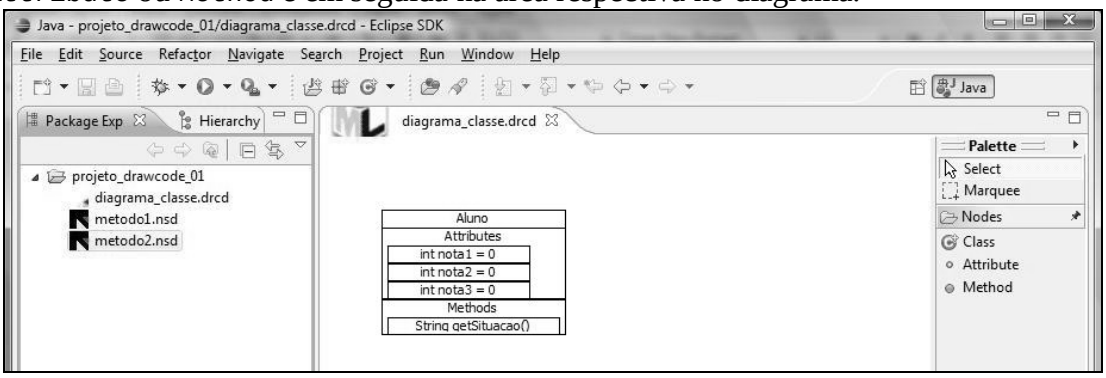


Figura 4 – Edição do diagrama de classe: incluindo uma classe com atributos e métodos

A Figura 5 apresenta um diagrama N-S em edição, onde foi desenvolvido um algoritmo que soma três notas e faz a divisão por três calculando a média. Se a média encontrada for maior ou igual a seis, o algoritmo retorna a *String* "aprovado". Caso contrário vai retornar a String "reprovado".

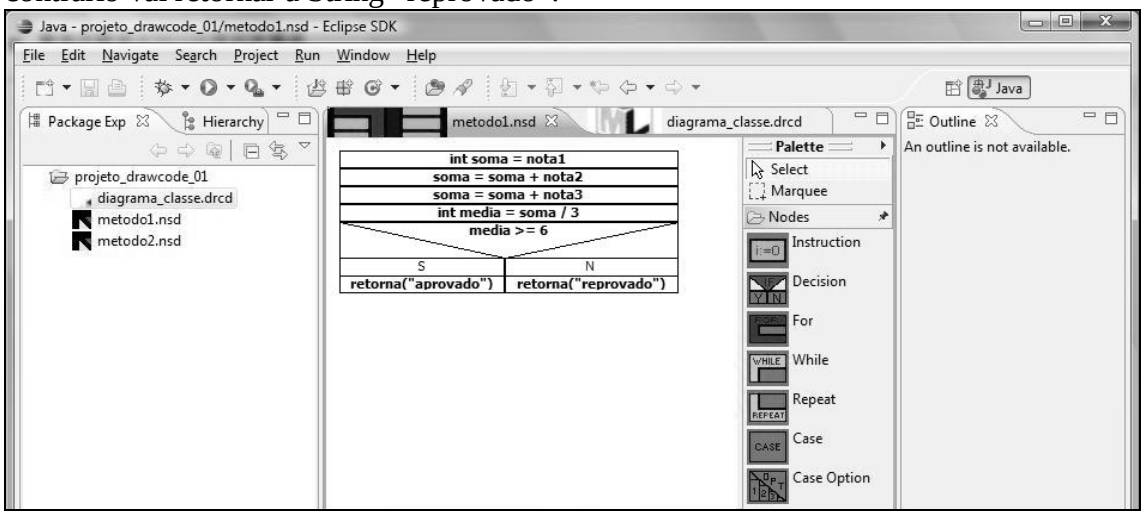


Figura 5 – Edição do diagrama N-S: definindo o algoritmo

Para fazer a associação entre o método existente na classe e o algoritmo definido no diagrama N-S é preciso editar o diagrama de classes e dar um duplo clique sobre o método, associando-o ao arquivo correspondente ao diagrama N-S.

Depois de criada a classe e definido o algoritmo para atender o propósito do método, é possível então gerar o código para o diagrama de classes. Este processo contempla o corpo da classe e o corpo dos métodos que tiverem um diagrama N-S associado. A Figura 6 apresenta o resultado da geração de código do diagrama de classes, apresentado nos exemplos anteriores. O ambiente Eclipse possui um mecanismo para identação automática do código o qual foi utilizado no código gerado para melhorar sua legibilidade. Nenhum outro tipo de alteração no conteúdo do código foi realizado.

```
//código gerado automaticamente pelo drawcode
public class Aluno {
	int nota1 = 0;
	int nota2 = 0;
	int nota3 = 0;
	public String getSituacao() {
		// Corpo de metodo gerado automaticamento pelo drawcode
		int soma = nota1;
		soma = soma + nota2;
		soma = soma + nota3;
		int media = soma / 3;
		if (media >= 6) {
			return ("aprovado");
		} else {
			return ("reprovado");
		}
	}
}
```

Figura 6 – Resultado da geração de código a partir do diagrama de classe

Algumas características na implementação da ferramenta estão destacadas na Tabela 1.

Característica	Descrição
modelos	foram criados modelos lógicos e físicos separados para representar o diagrama de classes e o diagrama N-S;
uso do padrão de projeto Visitor	este padrão permitiu que as classes criadas para representar o modelo ficassem simples com apenas algumas funcionalidades básicas. Para as operações necessárias foram criadas classes separadas, com papéis distintos: as classes "visitáveis" que são os elementos do modelo e as classes "visitadoras" que representam as funcionalidades existentes sobre o modelo;
interface gráfica	foi utilizado o *framework* GEF para criar as representação gráfica da ferramenta. Este *framework* foi extremamente útil uma vez que um dos principais problemas dos diagramas N-S é sua representação visual, visto que ao incluir, excluir ou alterar os elementos existentes, a representação precisa ser redesenhada para que se mantenha a visualização correta de cada um dos elementos presentes no diagrama, exigindo tratamento diferenciado para cada um dos símbolos gráficos presentes no diagrama, de acordo com suas características.
persistência	utilizou-se a biblioteca XStream pelo seu alto nível de abstração para converter objetos Java no formato XML e do formato XML para objetos Java;

Tabela 1 - Características de implementação do DrawCode

Outra característica importante é a geração de código, cujo processo é formado por três artefatos: o motor de *templates*, a classe responsável por instanciar e inicializar o motor e o modelo de dados do diagrama com os elementos que o usuário inseriu. Para gerar o código, a classe responsável instancia o Velocity, carrega o *template* para aquele

elemento, seta os valores do contexto e comanda que o Velocity faça o processamento do *template*. Este processo é afetado pelos elementos apresentados na Tabela 2.

texto inserido pelo usuário	é o texto contido nos elementos dos diagramas de classe e N-S. Pode ser o texto da condição de um comando de decisão, o texto de um comando simples ou ainda o texto de um elemento do tipo método no diagrama de classes;
templates para os elementos dos diagramas	são os arquivos criados para representar cada um dos elementos dos diagramas de classe e N-S;
templates com as tabelas de tradução	são os arquivos criados para representar uma tabela de tradução. A tabela de tradução fornece maior flexibilidade à utilização do *plugin*, pois permite ao próprio usuário definir uma tabela de palavras a serem substituídas. Para facilitar este mapeamento, a entrada definida pelo usuário passa por um analisador léxico, que transforma uma *String* em uma lista de *tokens*

Tabela 2 - Elementos do processo de geração de código do DrawCode

6. Resultados e Discussões

Durante a revisão bibliográfica sobre os temas relacionados ao trabalho, foram pesquisadas ferramentas para edição de diagramas de classe assim como para edição de diagramas N-S, que tenham características e permitam oferecer funcionalidades próximas ao *plugin* desenvolvido. Para a edição de Diagramas N-S destacam-se o NSD Editor (KALT, 1996), criado no ambiente Borland Delphi e que suporta as principais estruturas e permite gerar código para as linguagens C e Pascal e o Structorizer (STRUCTORIZER, 2007), criado na linguagem pascal e que possui características semelhantes ao NSD Editor, mas não permite a geração de código.

Para a edição de diagramas de classe as ferramentas em destaque são *plugins* para o ambiente Eclipse, sendo elas o Jupe (JUPE, 2007) que é um editor de diagramas de classe que suporta as principais notações deste tipo de diagrama e o TopCased (TOPCASED, 2007) que disponibiliza um conjunto de ferramentas de engenharia de software, permitindo a criação de diversos tipos de diagramas da UML além de permitir a geração de código a partir do diagrama de classes.

Na Tabela 3 é apresentado um comparativo entre o Drawcode e os trabalhos correlatos pesquisados. Ressalta-se que durante a pesquisa não foram encontradas ferramentas criadas na forma de *plugin* para o ambiente Eclipse que possibilitassem a edição de diagramas N-S, caracterizando ainda mais a relevância do trabalho ora apresentado.

	Jupe	Topcased	NSDEditor	Structorizer	Drawcode
Associa o corpo de um método a algum tipo de diagrama					**X**
Edita Diagrama de Classes	**X**	**X**			**X**
Edita outros diagramas da UML		**X**			
Edita Diagrama N-S			**X**	**X**	**X**
Gera algum tipo de código	**X**	**X**	**X**		**X**
Permite incluir nova linguagem para geração de código					**X**

Tabela 3 – Comparativo entre Drawcode e seus correlatos

O Drawcode apresenta algumas limitações com possibilidades de extensões a serem implementadas em futuras versões (Tabela 4).

a) no diagrama de classes permitir a edição das demais representações deste tipo de diagrama como associações, agregação, herança, entre outros;
b) no diagrama N-S: - permitir a mudança de posição de um elemento. Na atual versão, alterar a ordem de dois comandos, por exemplo, exige que seja excluído e criado novamente um dos dois comandos; - permitir a consulta dos atributos existentes na classe, bem como dos parâmetros dos métodos; - implementar as estruturas de paralelismo e bloco de comandos ainda não disponíveis; - incluir o recurso de complementação automática de código para apresentar os comandos ou atributos existentes no contexto daquele elemento;
c) validar as expressões escritas pelo usuário no diagrama de classes ou N-S. Na atual versão, se o usuário produzir uma expressão incorreta, a mesma será replicado no código gerado, uma vez que não existem formalismos na linguagem dos comandos de ambos os diagramas;
d) possibilitar engenharia reversa, permitindo ler código fonte de programas já existente e montar o diagrama de classes e seus respectivos diagramas N-S.
e) validar a simbologia dos diagrama N-S no contexto da OO, propondo ajustes nos símbolos existentes e/ou novas construções como, por exemplo, a representação de tratamentos de exceções.

Tabela 4 – Limitações e possibilidades de extensão do DrawCode

7. Considerações Finais

Via de regra, os editores de diagramas de classe estão inseridos em contextos de pesadas ferramentas CASE muito distantes do uso de alunos de séries iniciais. Além disso, estas ferramentas não apóiam o processo de codificação, devendo o mesmo ser feito diretamente em uma linguagem de programação que muitas vezes torna mais penoso o processo de aprendizagem de construções algorítmicas. Assim, no contexto de uma metodologia que inicie com o ensino de OO desde as séries iniciais de um curso na área de computação - como a proposta em Lopes(2007), por exemplo - o trabalho ora desenvolvido apresenta uma ferramenta que une os benefícios da OO com a legibilidade proporcionada pelos diagramas N-S, cada uma contribuindo da melhor maneira possível para a representação e resolução de problemas. Além disso, a ferramenta facilita o contato com ambientes reais de programação, uma vez que permite a geração de código em qualquer linguagem de programação por meio da construção de *templates*.

Também como contribuição deste trabalho, destaca-se que mesmo para os desenvolvedores já experientes, o Drawcode pode ser uma ferramenta de produtividade e qualidade, preenchendo uma lacuna existente nos diagramas da UML, que não apresenta nenhum de seus diagramas com estas características e que sustentem inclusive o processo de geração de código com este grau de interação.

Finalmente, no contexto dos grandes desafios em computação levantados no início deste texto, ferramentas como o DrawCode podem ser extremamente úteis justamente por permitir aos estudantes em séries iniciais construir e testar seus próprios modelos de solução de problemas utilizando, desde o início do curso, metodologias compatíveis com as exigências de formação de egressos destes cursos e possibilitando o uso de recursos e ferramentas computacionais que tem o potencial talvez não de ser motivadoras por si mesmas, mas que dentro de um adequado contexto de formação não os afugentem em longas e tediosas aulas onde se constróem modelos usando giz, quadro-negro, lápis e borracha.

8. Referências Bibliográficas

APACHE SOFTWARE FOUNDATION. **Velocity overview.** [S.l.], 2007. Disponível em: <http://velocity.apache.org/engine/devel/overview.html>. Acesso: 15 nov. 2007.

BARROS, Leliane Nunes de et al. **Desenvolvendo plugins**. [S.l.], 2003. Disponível em: <http://www.ime.usp.br/~articuno/eclipse/desenvolvendo_plugins.html>. Acesso em: 08 abr. 2007.

BRITISH COMPUTER SOCIETY. Education themes: grand challenges in computing. [S.1.], 2008. Disponível em: < http://www.bcs.org/server.php?show=nav.8107>. Acesso em: 01 mar. 2008.

CARBONI, Irenice de Fátima. **Lógica de programação.** São Paulo: Pioneira Thomson Learning, 2003.

ECLIPSE FOUNDATION. **About the eclipse foundation**. [S.l.], 2007. Disponível em: <http://www.eclipse.org/org/>. Acesso em: 08 nov. 2007.

FOWLER, Martin. **UML essencial:** um breve guia para a linguagem-padrão de modelagem de objetos. 3. ed. Tradução João Tortello. Porto Alegre: Bookman, 2005.

HENKELS, André. **Drawcode: um plugin do eclipse para a geração de código a partir de diagramas de classe e diagramas N-S**. 2007. 102f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Sistemas e Computação, Universidade Regional de Blumenau, Blumenau.

HERRINGTON, Jack. **Code generation in action.** Greenwich: Manning, 2003.

JUPE. **Jupe is a UML plugin for Eclipse**. [S.l.], 2007. Disponível em: <http://jupe.binaervarianz.de/>. Acesso em: 17 nov. 2007.

KALT, Marcel. **NSD-Editor.** [S.l.], 1996. Disponível em: <http://diuf.unifr.ch/softeng/student-projects/completed/kalt/NSD.html>. Acesso em: 15 nov. 2007.

LOPES, Maurício C. iPOO: uma metodologia para o ensino introdutório de orientação a objetos. In: **Anais...**. I Workshop sobre Educação em Informática, , 2007, Torres (RS). v. 1.

NASSI, Isaac; SHNEIDERMAN, Ben. **Flowchart Techniques for Structured Programming**. New York, v. 8, n.2, p. 12-26, Agosto 1973.

ORIENTAÇÃO A OBJETO. In: WIKIPÉDIA, a enciclopédia livre. [S.l.]: Wikimedia Foundation, 2007. Disponível em: <http://pt.wikipedia.org/wiki/Orientação_a_objeto>. Acesso em: 10 set. 2007.

PILONE, Dan; PITMAN, Neil. **UML 2 rápido e prático**: guia de referência. Tradução Armando Figueiredo. Rio de Janeiro: Alta Books, 2006.

SOUZA, Eliane Moreira Sá de. **Um modelo para processo ensino-aprendizagem de procedimentos lógicos em diversos domínios**. 2000. 202 f. Tese (Doutorado em Engenharia de Produção) – Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis.

STRUCTORIZER. **Structorizer:** about the project. [S.l.], 2007. Disponível em: <http://structorizer.fisch.lu/>. Acesso em: 17 nov. 2007.

TONSIG, Sergio Luiz. **Engenharia de software:** análise e projeto de sistemas. São Paulo: Futura, 2003.

TOPCASED. **UML Tools.** [S.l.], 2007. Disponível em: <http://topcased-mm.gforge.enseeiht.fr/website/modeling/uml/index.html>. Acesso em: 17 nov. 2007.