UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

SISTEMA DE APRENDIZADO DE ALGORITMOS PELA WEB IMPLEMENTADO COM GRÁFICOS VETORIAIS

ANTONIO URBANO FILHO

BLUMENAU
2006

2006/2-02

ANTONIO URBANO FILHO

SISTEMA DE APRENDIZADO DE ALGORITMOS PELA WEB IMPLEMENTADO COM GRÁFICOS VETORIAIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Maurício Capobianco Lopes, Msc. - Orientador

BLUMENAU
2006

2006/2-02

SISTEMA DE APRENDIZADO DE ALGORITMOS PELA WEB IMPLEMENTADO COM GRÁFICOS VETORIAIS

Por

ANTONIO URBANO FILHO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Maurício Capobianco Lopes, Mestre – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Doutor – FURB
Membro:	Prof.ª Joyce Martins, Mestre – FURB

Blumenau, 06 de dezembro de 2006

Dedico este trabalho a todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça, velando por mim enquanto estive sem tempo para isso, ocupado que estava em querer fazer tantas coisas ao mesmo tempo.

À minha família, em especial minha esposa Lenita, por ter estado ao meu lado durante todo o curso e principalmente nesta última etapa, que exigiu muito da sua compreensão.

Aos meus amigos, especialmente os do curso, que ajudando e empurrando um ao outro passam pela mesma caminhada e depois seguem seu caminho individual.

A todos os professores do curso, em especial ao meu orientador, Maurício Capobianco Lopes, por ter acreditado na conclusão deste trabalho e suportado meus atrasos.

“Se enxerguei além dos outros, é por que estava no ombro de gigantes”.

Sir Isaac Newton

RESUMO

Este trabalho apresenta o AlgoSVG, uma ferramenta gráfica e interativa baseada na web para o aprendizado de algoritmos. A ferramenta permite ao aluno incluir e conectar entre si símbolos de fluxograma formando o algoritmo desejado. Este algoritmo pode ser executado e testado. Para formação das telas são usados gráficos vetoriais que permitem mudança de escala sem perda de definição. Os elementos gráficos na tela são dinâmicos e interativos, respondendo a eventos como clicar de botões. A ferramenta é acessada através do navegador de web dispensando instalação de executáveis. Para geração dos gráficos foi escolhido o formato Scalable Vector Graphics (SVG), um padrão projetado para fornecer gráficos vetoriais dinâmicos e interativos na web.

Palavras-chave: Algoritmos. Fluxogramas. Gráficos vetoriais. SVG. Aprendizado pela web.

ABSTRACT

This work presents the AlgoSVG, a graphical and interactive web based tool for algorithms learning. The tool allows the student to include and connect flowchart symbols forming the desired algorithm. This algorithm can be executed and tested. For screens generation, vectorial graphics are used, which allow scale change without definition loss. The graphical elements in the screen are dynamic and interactive, answering the events as mouse clicks. The tool is accessed through the web, avoiding executables installation. For the graphics generation the Scalable Vector Graphics (SVG) format was chosen, a projected standard which supply dynamic and interactive vectorial graphics in web.

Key-words: Algorithms. Flowcharts. Vectorial graphics. SVG. Web learning.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

DOM – *Document Object Model*

GALS – Gerador de Analisadores Léxicos e Sintáticos

HTML – *HyperText Markup Language*

PHP – *Hypertext Preprocessor*

SVG – *Scalable Vector Graphics*

W3C – *World Wide Web Consortium*

XHTML – *eXtensible HyperText Markup Language*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O ensino de programação é essencial num curso de ciência da computação. É ministrado em várias disciplinas, sendo a primeira delas Introdução à Programação, onde o aluno aprende os princípios da lógica e desenvolve a capacidade de análise e resolução de problemas descrevendo-os através de algoritmos (VARGAS, 2005, p. 13). Algoritmo pode ser considerado uma seqüência de procedimentos finitos que são executados no tempo certo para atingir o objetivo desejado (CARBONI, 2003, p. 12).

Um algoritmo pode ser representado de várias formas, tais como fluxograma e Portugol. O fluxograma faz uso de símbolos geométricos e ligações que representam as estruturas de um programa. Já o Portugol representa o algoritmo usando a língua portuguesa escrita em uma sintaxe determinada (VARGAS, 2005, p. 13). O fluxograma é uma linguagem gráfica, enquanto o Portugol é uma linguagem textual. Um algoritmo pode ser criado diretamente na linguagem gráfica do fluxograma sem uma linguagem intermediária.

O desafio no processo de ensino de lógica e algoritmos consiste no fato deste ser aplicado a grupos heterogêneos de indivíduos, cada um com talentos, estilos de trabalho, forma de pensar e métodos de aprendizagem diferentes (MEDEIROS; DAZZI, 2002). Uma solução para esse desafio está em o próprio indivíduo conduzir boa parte de seu aprendizado, e uma maneira de facilitar isso é fornecer um ambiente interativo de aprendizagem.

“Os ambientes interativos de aprendizado são baseados em quatro princípios: o estudante deve construir seu conhecimento; o controle do sistema é feito, de forma mais significativa, pelo estudante; o sistema é individualizado para cada estudante; e o *feedback* é gerado em função da interação do estudante com o ambiente.” (VARGAS, 2005, p. 18).

A internet é uma ferramenta de grande importância e deve ser considerada neste contexto, pois uma aplicação web que permita ao aluno criar e testar seus algoritmos pode ser utilizada tanto em aulas práticas presenciais como à distância (MEDEIROS; DAZZI, 2002), pois está disponível em qualquer hora ou local, aumentando assim as oportunidades do aluno utilizá-la.

Entretanto, nas ferramentas hoje utilizadas na internet existem dificuldades com respeito à interatividade dos conteúdos, ou seja, na manipulação de objetos disponíveis na tela. Porém, novas tecnologias surgiram para facilitar o uso de gráficos de alta definição e manipulação interativa dos mesmos. Uma tecnologia promissora é um formato gráfico chamado *Scalable Vector Graphics* (SVG). Este formato permite criar gráficos vetoriais e

animações em páginas geradas dinamicamente. Um gráfico vetorial contém as informações sobre as curvas e as linhas de uma imagem, ao invés de informações sobre seus *pixels*. Isto permite, por exemplo, que o gráfico seja aumentado ou diminuído na tela sem perda de definição. Os gráficos SVG podem ser incluídos em páginas *HyperText Markup Language* (HTML) e *eXtensible HyperText Markup Language* (XHTML). O SVG suporta conteúdo dinâmico e interatividade. Eventos podem ser usados para manipular objetos gráficos. Essa manipulação é possível através da utilização de linguagens de *script*, como o JavaScript, que podem criar novos gráficos e modificar ou excluir partes de um gráfico, acessando os elementos SVG no *Document Object Model* (DOM). Arquivos SVG também podem ser gerados dinamicamente através das linguagens que rodam no servidor web, como *Hypertext Preprocessor* (PHP). Um arquivo SVG segue o padrão *eXtensible Markup Language* (XML) e isto permite integrá-lo facilmente com outras tecnologias (MACHADO; FURTADO; ALVES, 2002).

Assim, este trabalho descreve o desenvolvimento do AlgoSVG, uma ferramenta gráfica e interativa baseada na web para auxiliar no aprendizado de algoritmos. Esta ferramenta permite ao aluno construir o algoritmo graficamente por acrescentar e ligar entre si símbolos de fluxograma e, ao final, executar e testar o algoritmo criado. Para os gráficos foi utilizado o formato SVG e para os diálogos e demais formulários o HTML.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é implementar uma ferramenta gráfica e interativa baseada na web denominada de AlgoSVG, que auxilie no aprendizado de algoritmos.

Os objetivos específicos do trabalho são:

a) permitir que o aluno crie e teste seus próprios algoritmos via web;
b) estimular as fases de análise do problema e projeto da solução;
c) permitir a construção de algoritmos modularizados, com o conceito de sub-rotina;
d) montar as telas usando gráficos vetoriais dinâmicos e interativos.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos.

O segundo capítulo apresenta pesquisas sobre temas relacionados com este trabalho e relaciona alguns trabalhos correlatos, descrevendo suas principais características.

O terceiro capítulo descreve a especificação e implementação do AlgoSVG, as técnicas e ferramentas utilizadas e uma discussão sobre os resultados obtidos, comparando este trabalho com os correlatos citados no segundo capítulo.

As conclusões obtidas com este trabalho e algumas sugestões para futuros trabalhos encontram-se no quarto capítulo.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentadas pesquisas sobre temas relacionados com esse trabalho e também alguns trabalhos correlatos.

2.1 ALGORITMOS

Automação acontece quando uma tarefa deixa de ser desempenhada pelo homem e passa a ser realizada por máquinas, sejam computadores ou não. Para que a automação de uma tarefa tenha sucesso, é necessário que a máquina que passará a realizá-la seja capaz de desempenhar com eficiência cada uma das etapas constituintes do processo a ser automatizado, de modo a garantir a repetibilidade do mesmo. Para isso, é necessário que seja especificado com clareza e exatidão o que deve ser realizado em cada uma das fases do processo a ser automatizado, bem como a seqüência em que estas fases devem ser realizadas. À especificação da seqüência ordenada de passos que deve ser seguida para a realização de uma tarefa, garantindo a sua repetibilidade, dá-se o nome de algoritmo (SALIBA, 1992, p. 1).

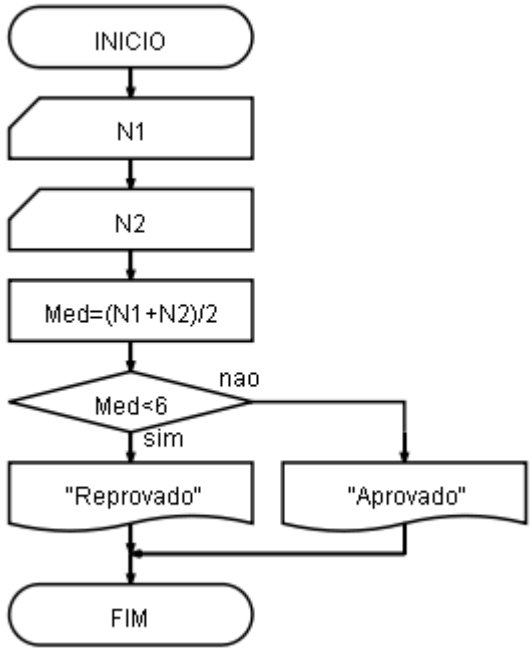
Ao contrário do que se pode pensar, o conceito de algoritmo não foi criado para satisfazer às necessidades da computação. Pelo contrário, a programação de computadores é apenas um dos campos de aplicação dos algoritmos. Na verdade, há inúmeros casos que podem servir como exemplo do uso, involuntário ou não, de algoritmos para a padronização da execução de tarefas rotineiras. Uma receita de bolo, a troca de um pneu furado ou tomar um banho são exemplos de tarefas que podem ser descritas detalhadamente e com precisão de modo a serem repetidas outras vezes (SALIBA, 1992, 1).

Voltando para o contexto computacional, para que um computador possa desempenhar uma tarefa, é necessário que esta seja detalhada passo-a-passo, numa forma compreensível pela máquina, utilizando aquilo que se chama de programa. Neste sentido, um programa de computador nada mais é que um algoritmo escrito numa forma que o computador entenda, ou seja, numa linguagem de programação (SALIBA, 2002, p. 2).

Segundo Carboni (2003, p. 12-24), o algoritmo é uma forma de descrever a lógica para solução de um problema. O algoritmo pode ser considerado uma seqüência de procedimentos finitos que serão executados em determinado período de tempo para atingir o objetivo. Todo

algoritmo é composto por instruções finitas e bem definidas. As instruções escritas em uma linguagem de programação são chamadas de programas. Entre as formas de representação do algoritmo estão o português estruturado e o fluxograma. O primeiro também é conhecido como Portugol ou pseudocódigo e representa a lógica do algoritmo usando palavras da língua portuguesa. Já o fluxograma usa um conjunto de figuras geométricas para representar a lógica do algoritmo.

Essas duas formas de representação podem ser vistas na Figura 1, que mostra o cálculo da média das notas de um aluno sob a forma de fluxograma e pseudocódigo. A forma de representação por pseudocódigo é mais rica em detalhes, como a definição dos tipos das variáveis. Já o fluxograma é mais fácil de ser visualizado e entendido. (SALIBA,1992, p. 5-8).




```
Algoritmo Media

Var N1, N2, Med : real

Inicio
      Leia N1
      Leia N2
      Med ← (N1 + N2) / 2
      Se  Med < 6
            Então
                  Escreva “Reprovado”
            Senão
                  Escreva “Aprovado”
            Fim-se
Fim.
```

Fonte: Saliba (1992, p. 7-8).

Figura 1 – Exemplo de um algoritmo para cálculo da média

O bom aprendizado de algoritmos é obtido através de muitos exercícios. Mas para obter bons algoritmos é necessário seguir uma metodologia que define passos para a sua construção.

Carboni (2003, p. 13-14) define uma metodologia com os seguintes passos:

a) identificar o problema mediante leitura atenta do enunciado;

b) retirar do enunciado quais são as “entradas de dados” que serão fornecidas;

c) retirar do enunciado quais as “saídas de dados” que devem ser geradas;

d) determinar o que deve ser feito para transformar as “entradas” em “saídas”;

e) construir o algoritmo, com as informações levantadas nos passos anteriores;

f) testar a solução.

Alguns métodos foram propostos para a construção de algoritmos, surgindo então a expressão programação estruturada. Segundo Carboni (2003, p. 161-165), programação estruturada é um método de programação que recomenda programas modulares, usando apenas as estruturas básicas de controle: seqüência, seleção e repetição. Modulares significa dividido em sub-rotinas. Seus objetivos são: trabalho em equipe, programas legíveis, menos erros, melhores testes, menor tempo de programação e reaproveitamento de rotinas. O método divide um programa em tarefas distintas, organizadas em partes independentes chamadas módulos. Cada módulo representa uma tarefa lógica (procedimento ou função) do programa. Nas chamadas dos módulos podem ocorrer passagens ou retornos de parâmetros.

2.2 FLUXOGRAMAS

O fluxograma é uma forma gráfica de representar algoritmos. No fluxograma as operações são representadas por símbolos que identificam o processo envolvido e o fluxo de execução é representado pelas setas que interligam as operações. Os símbolos usados em fluxogramas foram propostos na norma ISO 5807-1985, que deixou livre o uso de aplicações ou soluções particulares sem impor normas rígidas (MANZANO, 2004, p. 3). Os símbolos mais utilizados e seu significado ou uso podem ser vistos na Figura 2.

Símbolo	Significado	Descrição ou uso mais comum
	Terminal	Início e Fim de programa ou sub-rotina.
	Teclado	Entrada manual de dados pelo teclado.
	Cartão perfurado	Entrada de dados. Parâmetros do programa.
	Exibição	Saída de dados no monitor de vídeo.
	Documento	Saída de dados em um relatório.
	Dados em disco	Qualquer operação em arquivos de disco, inclusive abrir e fechar.
	Dados genéricos	Entrada ou saída genérica de dados.
	Processamento	Atribuição de um valor ou expressão a uma variável.
	Decisão	Tomada de decisão com desvio para um dos caminhos.
	Preparação	Usado no comando de repetição.
	Processo	Chamada de uma sub-rotina.
	Conector	Re-conecta os caminhos que foram desviados.
	Linha direcionada	Conecta os elementos e indica o fluxo da execução.
	Operação manual	Por exemplo, “Verifique se há papel na impressora”.
	Declaração	Declaração de variáveis.

Fonte: Manzano (2004, p. 8-11) e Cares (2002, p. 16).

Figura 2 – Alguns símbolos usados em fluxogramas

Neste trabalho não foram utilizados todos os símbolos do fluxograma. O Quadro 1 mostra quais os símbolos foram utilizados no AlgoSVG e quais não o foram.

Símbolos utilizados no AlgoSVG	Símbolos não utilizados
Terminal, para marcar início e fim do algoritmo ou sub-rotina	Cartão perfurado
Teclado, para comandos de entrada de dados	Exibição
Documento, para comandos de saída de dados	Dados em disco
Processamento, para comandos de atribuição	Dados genéricos
Decisão, para comandos de seleção	Conector
Preparação, para comandos de repetição	Operação manual
Processo, para chamadas de sub-rotinas	Declaração
Linha direcionada, para indicar o fluxo de execução	

Quadro 1 – Símbolos do fluxograma utilizados no AlgoSVG

Os símbolos do fluxograma representam instruções primitivas. A partir delas pode-se construir estruturas mais complexas, que informam as instruções a serem executadas, o momento de sua execução e o fluxo de processamento. Existem três estruturas básicas:

a) estruturas seqüenciais;
b) estruturas de decisão;
c) estruturas de repetição.

Na estrutura seqüencial os comandos são executados na seqüência em que aparecem. Cada comando é executado somente após o término do anterior. Um exemplo de comandos numa estrutura seqüencial pode ser visto na Figura 3.

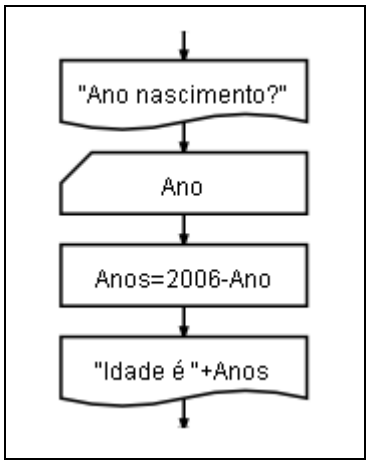


Figura 3 – Exemplo de uma estrutura seqüencial

Na estrutura de decisão o conjunto de instruções a ser executado é escolhido em função do resultado da avaliação de uma expressão lógica. Uma construção do tipo decisão pode ser entendida como uma bifurcação onde há dois caminhos que podem ser seguidos. Um será escolhido se a expressão lógica for verdadeira e outro se resultar falsa. A Figura 4 mostra exemplos de estrutura de decisão.

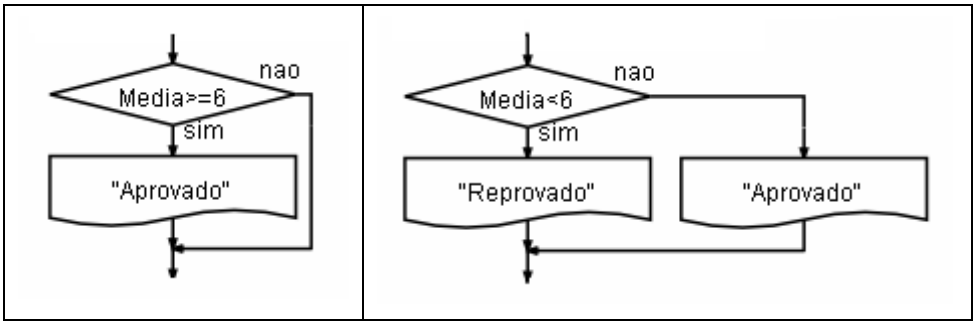


Figura 4 – Exemplos de estrutura de decisão

Na estrutura de repetição um conjunto de instruções é repetido zero ou mais vezes em função do resultado da avaliação de uma expressão lógica. As estruturas de repetição são muitas vezes chamadas de laços de repetição. Há dois caminhos que podem ser seguidos. Um é executar o conjunto de instruções e voltar para o teste da expressão lógica. E outro é sair do laço de repetição, indo para a próxima instrução do programa. Existem dois tipos de estruturas de repetição. Um tipo é o laço condicional, que avalia uma expressão lógica para decidir se executa o conjunto de instruções ou se sai do laço de repetição. O outro tipo é o laço contado, que atribui a uma variável um valor inicial e a incrementa até um valor final executando o conjunto de instruções a cada passo (SALIBA, 1992, p. 66-68). A Figura 5 mostra exemplo dos dois tipos de repetição.

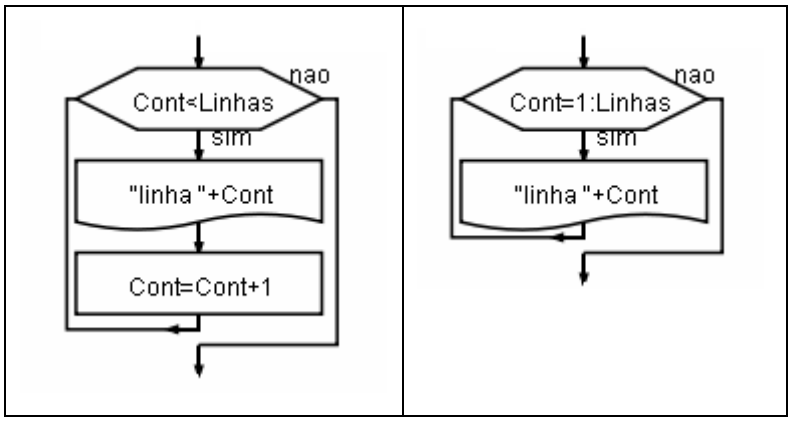


Figura 5 – Exemplos de estrutura de repetição

2.3 GRÁFICOS VETORIAIS E MATRICIAIS

Em computação gráfica pode-se classificar uma imagem, em relação à sua origem, de duas formas distintas. A primeira é o modelo geométrico vetorial, que se baseia em descrições matemáticas dos objetos gráficos e a segunda é o mapa de bits, que é a descrição da cor de cada *pixel*.

Os gráficos vetoriais criam imagens com a utilização de descrições matemáticas de objetos como círculos e linhas. Objetos simples como círculos, linhas, esferas, cubos e assim por diante, são chamados de primitivas e são utilizados na criação de gráficos mais complexos. O gráfico vetorial utiliza descrições simples para criar objetos primitivos. Usando-se um português simples, as descrições poderiam ser assim: “Desenho de uma linha do ponto A ao ponto B” ou “Desenho de um círculo com raio R centralizado no ponto P” (CORRIGAN, 1994, p. 21-23). Entre as vantagens dos gráficos vetoriais estão a mudança de escala sem perda de definição e a possibilidade de selecionar objetos individuais na imagem para serem alterados ou excluídos.

Um gráfico matricial é semelhante a um papel quadriculado onde cada quadrado é preenchido com a cor naquele ponto. Os *pixels*, forma abreviada de *picture elements*, são os blocos básicos de construção dos gráficos matriciais, ou seja, elementos individuais que combinados formam uma imagem. Devido ao gráfico matricial guardar informação para cada *pixel* individualmente, imagens muito grandes consomem muita memória. Um aspecto positivo dos gráficos matriciais é o foto realismo, quando a imagem armazenada for do mundo real (CORRIGAN, 1994, p. 3-18).

A Figura 6 ilustra algumas diferenças entre gráficos vetoriais e matriciais. A primeira diferença diz respeito à mudança de escala, já que no gráfico vetorial a figura ampliada mantém a forma original enquanto que no gráfico matricial há distorções. A segunda é quanto ao armazenamento do gráfico, pois para armazenar um círculo no gráfico vetorial basta guardar as coordenadas do ponto central e o raio da circunferência, enquanto no gráfico matricial é preciso armazenar as propriedades de cada ponto.

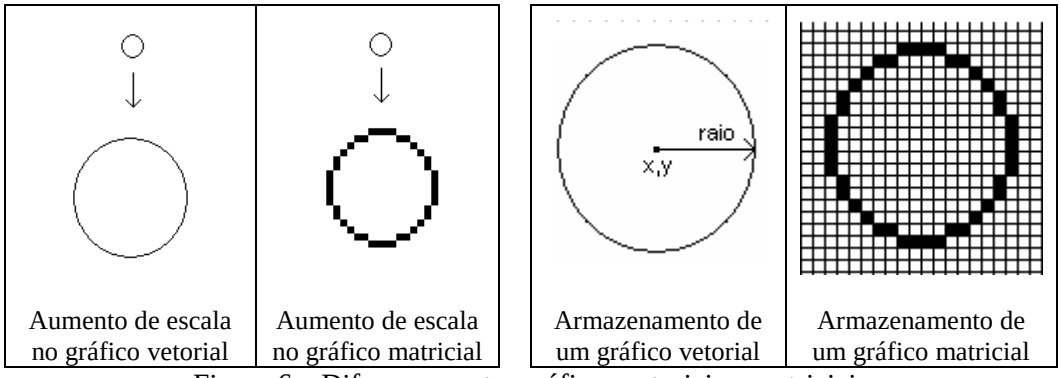


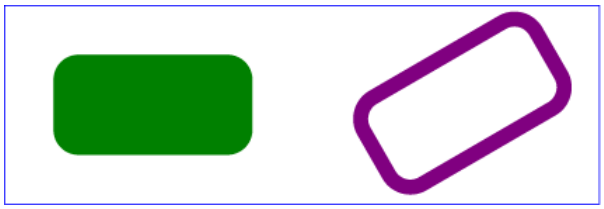
Figura 6 – Diferenças entre gráficos vetoriais e matriciais

2.4 FORMATO GRAFICO SVG

Um dos formatos que tem sido utilizado atualmente para a representação de gráficos vetoriais na web é o SVG. Ele foi proposto pelo *World Wide Web Consortium* (W3C) como um padrão para a criação de gráficos bidimensionais e animações. Basicamente, é um arquivo XML, um arquivo texto com marcações. Nas marcações são informados elementos primitivos de desenho como retângulos, linhas, polígonos, círculos e elipses (W3C, 2006).

A Figura 7 mostra um exemplo de um arquivo SVG, exibindo o código fonte do arquivo e a imagem gerada. O código fonte segue o padrão XML.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <desc>Example rect02 - rounded rectangles</desc>
  <!-- Show outline of canvas using 'rect' element -->
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="blue" stroke-width="2"/>
  <rect x="100" y="100" width="400" height="200" rx="50"
        fill="green" />
  <g transform="translate(700 210) rotate(-30)">
    <rect x="0" y="0" width="400" height="200" rx="50"
          fill="none" stroke="purple" stroke-width="30" />
  </g>
</svg>
```



Fonte: W3C (2006).

Figura 7 – Exemplo de um arquivo SVG

2.4.1 Formas geométricas no SVG

O SVG define elementos para seis primitivas gráficas: *circle*, *ellipse*, *rect*, *line*, *polyline* e *polygon*, que junto com o elemento polivalente *path*, podem ser combinados para a contrução de formas mais complexas. Cada uma dessas primitivas gráficas carrega propriedades que especificam sua posição e seu tamanho. Elas fornecem os seguintes resultados (TRAVERSA, 2001):

a) *circle*: mostra um círculo com o ponto central e o raio especificados;
b) *ellipse*: mostra uma elipse com o ponto central e os dois raios especificados;
c) *rect*: mostra um retângulo ou quadrado com o canto superior esquerdo, a largura e a altura especificadas. Pode mostrar cantos arredondados se for informado raio;
d) *line*: mostra um segmento de reta entre duas coordenadas;
e) *polyline*: mostra uma série de segmentos de retas com vértices nos pontos especificados;
f) *polygon*: igual ao *polyline*, mas une por uma linha o último ponto com o primeiro, fechando a forma;
g) *path*: mostra uma seqüência de linhas e curvas fechando ou não a forma.

No Quadro 2 e no Quadro 3 estão listadas as propriedades para definir posição, tamanho, preenchimento e tracejado das formas e no Quadro 4 os comandos para o elemento *path*.

Elemento	**Propriedades**	**Significado**
circle	*cx, cy*	Coordenada para o ponto central.
	r	Raio.
ellipse	*cx, cy*	Coordenada para o ponto central.
	rx, ry	Raios *x* e *y*.
rect	*x, y*	Coordenada para o canto superior esquerdo.
	width	Largura.
	height	Altura.
	rx, ry	Raios *x* e *y*, pode ser informado apenas um deles ou os dois.
line	*x1, y1*	Coordenada para o ponto inicial.
	x2, y2	Coordenada para o ponto final.
polyline	*points*	Lista com pares *x* e *y*, separados por branco e/ou vírgula.
poligon	*points*	Lista com pares *x* e *y*, separados por branco e/ou vírgula.
path	*d*	Seqüência de comandos e coordenadas para linhas e curvas.

Quadro 2 – Propriedades para definir posição e tamanho nas formas geométricas

Propriedades	**Significado**	
fill	Cor de preenchimento.	
fill-opacity	Opacidade da cor de preenchimento.	
fill-rule	Critério a ser usado para determinar se o ponto está dentro ou fora do polígono. Valores: *nonzero* e *evenodd*.	
stroke	Cor da linha.	
stroke-opacity	Opacidade da cor da linha, um valor real de 0 a 1, assume 1 por padrão.	
stroke-width	Espessura da linha.	
stroke-dasharray	Lista de tamanhos para alternar entre traços e falhas na linha, obtendo linhas contínuas, tracejadas, pontilhadas com diversos tamanhos e combinações.	
stroke-dashoffset	Desloca o padrão *stroke-dasharray*, iniciando alguns pontos adiante nele.	
stroke-linecap	Define como o final da linha é desenhado. Valores: *butt*, *round* ou *square*.	
stroke-linejoin	Define como junções de linhas são desenhadas. Valores: *miter*, *round* ou *bevel*.	
stroke-miterlimit	Se *stroke-linejoin* for *miter*, *stroke-miterlimit* indica o limite da ponta. Razão entre comprimento da ponta e espessura.	

Fonte: adaptado de Eisenberg (2002).

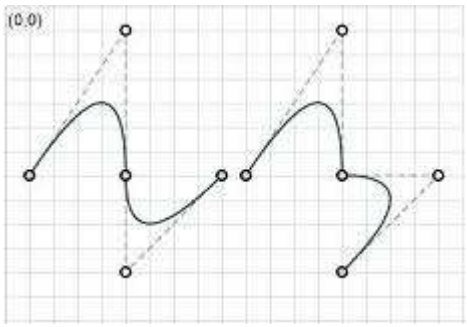
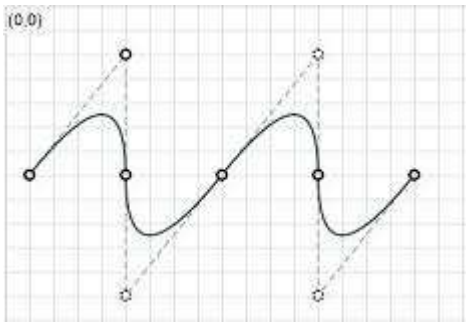
Quadro 3 – Propriedades para definir preenchimento e tracejado das figuras

Comando	**Significado**
M *x y*	*moveto*: move para a coordenada especificada sem desenhar a linha. Todo elemento *path* deve iniciar com *moveto*.
L *x y*	*lineto*: move para a coordenada especificada desenhando a linha.
H *x*	*horizontal lineto*: move para a posição *x* mantendo *y* e desenhando a linha.
V *y*	*vertical lineto*: move para a posição *y* mantendo *x* e desenhando a linha.
C *x1 y1 x2 y2 x y*	*curveto*: desenha uma curva *bezier* cúbica para o ponto *x,y* onde os pontos *x1,y1* e *x2,y2* são os pontos de controle inicial e final respectivamente.
S *x2 y2 x y*	*shorthand*/*smooth curveto*: igual a anterior mas assumindo para *x1,y1* a reflexão do último ponto de controle final.
Q *x1 y1 x y*	*quadratic bezier curveto*: desenha uma quadrática *bezier* entre o último ponto e o ponto *x,y* usando *x1,y1* como ponto de controle.
T *x y*	*shorthand*/*smooth quadratic bezier curveto*: igual a anterior, mas assumindo para *x1,y1* a reflexão do último ponto de controle
A *rx ry xAxisRotation largeArcFlag sweepFlag x y*	*elliptical arc*: desenha um arco elíptico do ponto corrente até *x,y*. O tamanho e orientação da elipse são definidos pelos dois raios *rx,ry* e uma *xAxisRotation*, que indica como a elipse inteira é rotacionada relativa à coordenada corrente do sistema. O centro *cx,cy* da elipse é calculado pelas restrições impostas pelos demais parâmetros. *large-arc-flag* e *sweep-flag* contribuem para o cálculo.
Z	*closepath*: fecha o caminho com uma linha do ponto final ao inicial.
Existe diferença entre maiúsculas e minúsculas. Uma letra maiúscula indica uma posição absoluta. Uma letra minúscula indica uma posição relativa em relação ao último ponto, exceto para a primeira instrução do elemento *path*, que sempre será uma posição absoluta. Os parâmetros são separados por branco e/ou vírgula a critério do programador.	

Fonte: adaptado de ADOBETUTORIALZ (2006).

Quadro 4 – Comandos para a propriedade *d* do elemento *path*

A Figura 8 demonstra o uso dos comandos do elemento *path* no desenho de curvas, mostrando como o comando T simplifica o desenho de curvas subseqüentes, assumindo para ponto de controle a reflexão do ponto de controle anterior. Esta estratégia se baseia no fato de que junções contínuas entre dois segmentos de curvas são muito comuns, o que justifica uma maneira para facilitar seu desenho (FROST; GOESSNER; HIRTZLER, 2003).

<table>
<tr><td colspan="2"> 
</td><td> 
</td></tr>
<tr><td>d = "M 10,70
Q 50,10 50,70
Q 50,110 90,70"</td><td>d = "M 100,70
Q 140,10 140,70
Q 180,70 140,110"</td><td>d = "M 10,70
Q 50,20 50,70
T 90,70
T 130,70
T 170,70"</td></tr>
</table>

Fonte: Frost, Goessner e Hirtzler (2003).

Figura 8 – Exemplos de curvas obtidas com comandos do elemento *path*

Os tamanhos e posições são valores numéricos reais, mas permitem um sufixo

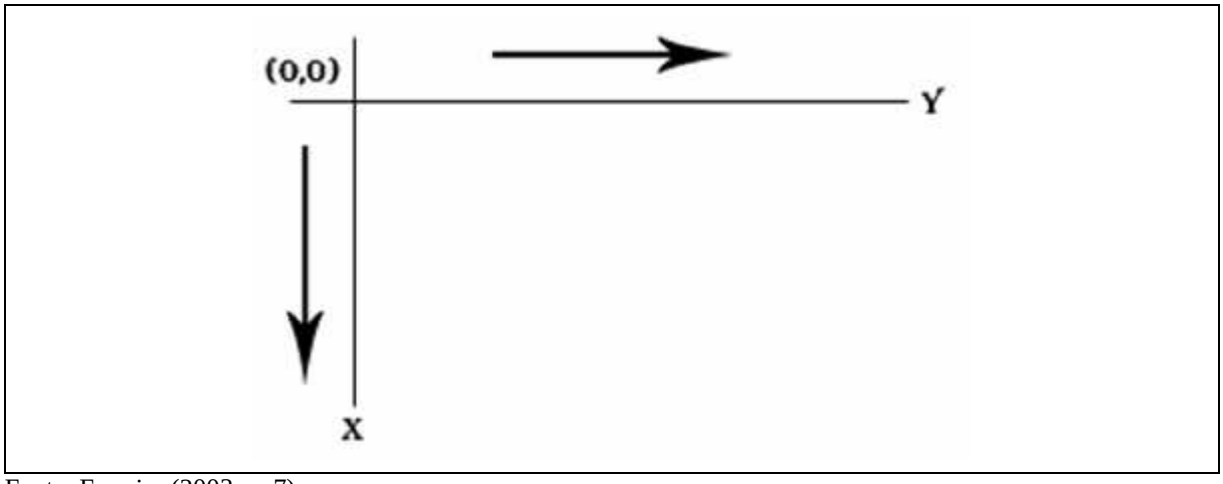
indicador da unidade de medida. Uma lista de sufixos válidos é mostrada no Quadro 5. Unidades como centímetros, milímetros ou polegadas são usadas em gráficos que serão impressos e necessitam exatidão. Nos demais casos a unidade *pixel* é mais indicada.

Unidade	**Significado**
em	Unidade igual ao tamanho do fonte corrente.
ex	Geralmente a altura da letra x minúscula (*x-height*).
px	*Pixel*, a unidade padrão, quando a unidade não é informada assume esta.
pt	*Point*, igual a 1/72 de polegada.
pc	*Picas*, igual a 1/6 de polegada.
cm	Centímetros.
mm	Milímetros.
in	Polegadas.
%	Percentual.

Quadro 5 – Sufixos indicadores da unidade de medida para os atributos

2.4.2 Sistema de coordenadas do SVG

Segundo Ferreira (2003, p. 7), os objetos SVG possuem a indicação de coordenada no eixo das abscissas e no eixo das ordenadas. Por padrão, este sistema possui o ponto (0, 0) no canto superior esquerdo, como mostrado pela Figura 9. No entanto, ele pode ser alterado, aplicando funções de rotação, translação e escalamento.



Fonte: Ferreira (2003, p. 7).

Figura 9 – Sistema de coordenadas padrão do SVG quando não for especificado outro

Para, por exemplo, converter este sistema de coordenadas, num sistema de coordenadas cartesiano, aplica-se as transformações mostradas na Figura 10.


```
<g transform="translate(0,400) scale(1,-1)">
<!-- todos os conteúdos aqui dentro, podem usar o sistema cartesiano -->
<!—- o valor 400, acima, se refere ao height e deve estar de acordo  -->
</g>
```

Fonte: Ferreira (2003, p. 7).

Figura 10 – Conversão para o Sistema de coordenadas cartesiano

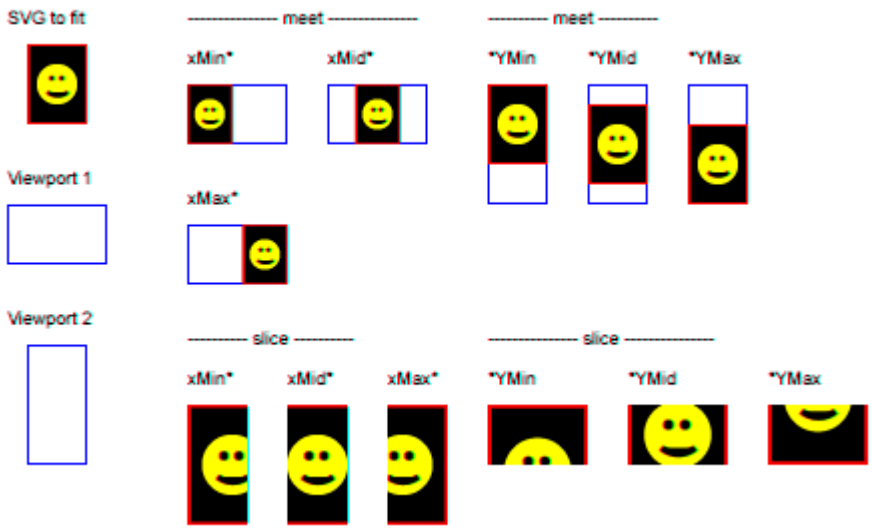
Conforme descrito em W3C (2006), o *canvas* descreve o espaço onde o conteúdo SVG é renderizado. O *canvas* é infinito para cada dimensão do espaço, mas a renderização ocorre em uma região retangular finita do *canvas*, chamada *viewport*. Uma *viewport* pode ser estabelecida pelo elemento `SVG` como exemplificado no Quadro 6. Novas *viewports* podem ser estabelecidas dentro de uma outra *viewport*.

```
<svg xmlns='http://www.w3.org/2000/svg' width="800px" height="600px"
    version="1.1">
  <!-- rest of SVG graphic would go here -->
  <svg width="600px" height="400px" version="1.1"
     viewBox="0 0 800 500" preserveAspectRatio="none"
     xmlns="http://www.w3.org/2000/svg">
</svg>
```

Fonte: W3C (2006).

Quadro 6 – Estabelecimento de uma *viewport*

Se for necessário especificar que um dado conjunto de gráficos se ajuste a uma viewport em particular deve-se usar o atributo `viewBox`. O atributo `viewBox` recebe como parâmetro uma lista de quatro números `min-x`, `min-y`, `width` e `height`, separados por branco ou vírgula, que especificam um retângulo que deve ser mapeado nos limites da *viewport* estabelecida. Os gráficos são ajustados proporcionalmente à largura e à altura da *viewport*. Se outro comportamento for desejado para ajustar os gráficos, usa-se mais um atributo chamado `preserveAspectRatio`, que informa o critério para ajustar o `viewBox` dentro da *viewport* e recebe como parâmetro a sintaxe `preserveAspectRatio="<align> [<meetOrSlice>]"`, combinados como mostra a Figura 11.




```
Exemplos:  preserveAspectRatio="none"
           preserveAspectRatio="xMinYMin meet"
           preserveAspectRatio="xMidYMin slice"
```

Fonte: W3c (2006)

Figura 11 – Combinações para o atributo `preserveAspectRatio`

2.4.3 Navegadores que suportam o SVG

Os navegadores mais recentes suportam o padrão SVG nativamente ou através de *plugins*. Na realização deste trabalho foram feitos testes apenas na plataforma Windows e nos navegadores FireFox, Opera e Internet Explorer com *plugin* Adobe SVG Viewer. Para outras plataformas também existem navegadores que suportam o padrão SVG.

O FireFox, a partir da versão 1.5, e o Opera, a partir da versão 8.0, suportam nativamente o padrão SVG. O Internet Explorer não suporta o padrão, nem mesmo na versão 7, mas com a instalação de um *plugin* consegue mostrar gráficos SVG. O *plugin* mais popular é o *Adobe SVG Viewer* liberado para o público na versão 3.03 e para desenvolvedores na versão 6.0. A versão 3.03 do *Adobe SVG Viewer* contém atualizações de segurança recentes que não foram incorporadas na versão 6.0, porém esta última possui mais características do SVG implementadas, além de ser estável (ADOBE, 2003).

2.4.4 Inserindo um gráfico SVG numa página web

Existem várias maneiras de incluir gráficos SVG em páginas da web. Uma delas é chamada de SVG *inline* onde o SVG é mesclado no código fonte de uma página web. Outra maneira é ter um arquivo unicamente SVG separado e referenciá-lo na página web (SCHEPERS; QUINT, 2005).

O Quadro 7 e a Figura 12 mostram um exemplo de SVG *inline*, exibindo um formulário HTML com um gráfico SVG. O código em negrito faz com que funcione com o *plugin* da Adobe. A vantagem desta solução é que há apenas um objeto *document*. Fica mais leve para o navegador e evita escrever código para comunicação entre documentos diferentes. A desvantagem é que no SVG *inline* nem tudo funciona corretamente, como eventos do mouse, por exemplo, que não sinalizam objetos individuais, só o mais externo.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:svg="http://www.w3.org/2000/svg"  xml:lang="en">
<head> <title>Teste SVG com PHP</title> </head>
  <object id="AdobeSVG"
    classid="clsid:78156a80-c6a1-4bbf-8e6a-3cd390eeb4e2"> </object>
  <?import namespace="svg" implementation="#AdobeSVG"?>
<body> <p> Exemplo de pagina web com graficos SVG </p>
<form method="post" action="proximo.html">
  <input type="submit" value="Proximo"/>
  <input type="hidden" name="passo" value="1"/> </form><p/>
<svg:svg width="720px" height="630px">
 <svg:rect id="P001" x="5" y="5" width="100" height="20" fill="yellow"
    stroke="black" stroke-width="1" />
  <svg:text x="10" y="20">INICIO</svg:text>
  <svg:path id="S001" d="M 39,25 L 39,40 35,40 40,45 45,40 41,40 41,25"
    fill="black" stroke-width="1"/> </svg:svg> </body> </html>
```

Quadro 7 – Arquivo exemplo.svg com SVG *inline*

Figura 12 – Arquivo exemplo.svg com SVG *inline*

A outra estratégia é ter arquivos unicamente SVG separados e referenciados de arquivos HTML por qualquer uma das *tags* `<object>`, `<embed>`, `<iframe>` ou `<frame>`. Nesta estratégia um arquivo gráfico tem internamente apenas o padrão SVG e é referenciado

pelos arquivos HTML através de uma das *tags* listadas. Quando o aplicativo é desenvolvido em módulos, distribuídos em arquivos diferentes, os gráficos já ficam em um arquivo unicamente SVG, sem a necessidade de mesclar padrões diferentes em um mesmo arquivo. O Quadro 8 mostra o arquivo principal do aplicativo AlgoSVG desenvolvido neste trabalho, onde foram usados *frames*, bem como as duas telas principais do aplicativo. Na linha 3 encontra-se a expressão `rows="*,0"`. Isto faz a primeira tela ser mostrada enquanto a segunda fica escondida. Para mostrar a segunda tela, o conteúdo do atributo `rows` é modificado para `"0,*"` com um comando do *script*. A linha 12 define o frame que referencia o arquivo SVG. A Figura 13 mostra as duas telas principais do AlgoSVG.

```
<html><head><title>Desenho de Fluxograma pela Web com Graficos SVG</title>
<script LANGUAGE="JavaScript"> ... ... </script> </head>
<frameset name="principal" id="principal" frameborder="yes" rows="*,0">
 <frame name="pjhtml" id="pjhtml" scrolling="no"  src="faseprojeto.html">
 <frameset rows="30px,*" frameborder="yes" name="frmset" id="frmset">
  <frameset cols="70%,*">
   <frame name="modulos" id="modulos" scrolling="no" src="modulos.html"
    marginheight="1">
   <frame name="debug" scrolling="no" src="debug.html" marginheight="1">
  </frameset>
 <frameset cols="70%,*">
  <frame name="painel" id="painel" scrolling="yes" src="fluxograma.svg">
  <frameset rows="50%,25%,*">
  <frame name="vars" id="vars" scrolling="yes" src="vars.html">
  <frame name="saidas" id="saidas" scrolling="yes" src="saidas.html">
  <frame name="pilha" id="pilha" scrolling="yes" src="pilha.html">
 </frameset></frameset></frameset></frameset></html>
```

Quadro 8 – Arquivo principal do AlgoSVG sem as linhas de *script*

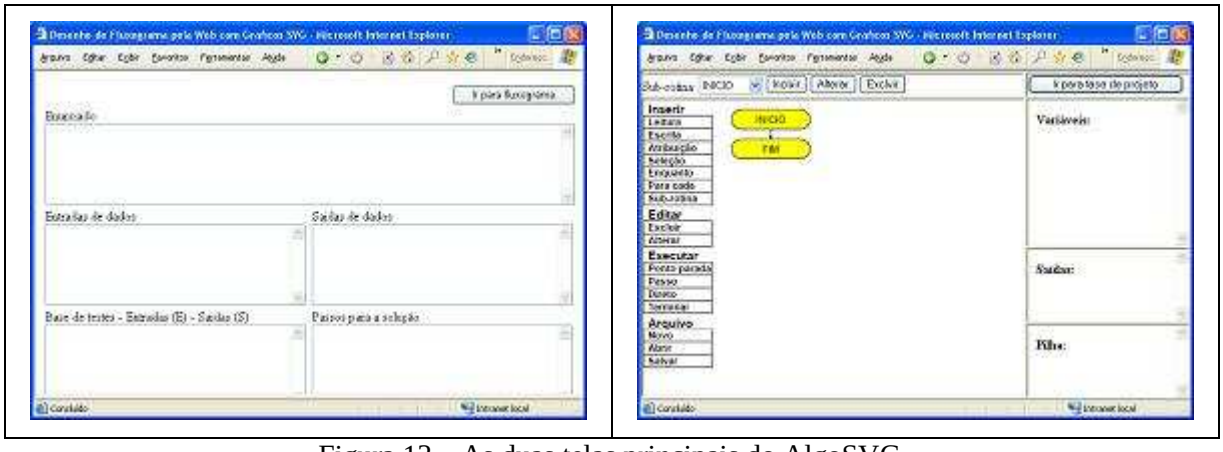


Figura 13 – As duas telas principais do AlgoSVG

2.5 TRABALHOS CORRELATOS

Para o desenvolvimento deste trabalho foram pesquisados alguns trabalhos correlatos desenvolvidos para auxiliar o aluno no aprendizado de algoritmos. Serão apresentados aqui dez trabalhos pesquisados. Estes trabalhos permitem ao aluno executar um algoritmo visualizando o conteúdo das variáveis durante a execução. Cinco destes trabalhos exibem o algoritmo apenas no formato Portugol. Os cinco restantes exibem no formato fluxograma.

2.5.1 Aplicativos que executam algoritmos sem exibir fluxograma

Em Tagliari (1996) é apresentado um programa escrito em Visual Basic que auxilia no estudo de algoritmos através de exemplos pré-definidos. O usuário pode executar os algoritmos, interagir com o sistema, bem como visualizar e alterar o conteúdo das variáveis. O algoritmo é escrito em Portugol e na execução passo a passo a linha que está ativa é grifada. O programa não permite ao usuário escrever seu próprio algoritmo.

Em Schmitt (1998) é descrito o protótipo de um programa semelhante ao descrito em Tagliari, desenvolvido em Delphi e permitindo a utilização de matrizes no algoritmo.

Almeida et al. (2002) descreveu o AMBAP, um aplicativo escrito em Java que permite ao aluno descrever um algoritmo em Portugol e executá-lo passo a passo. O documento ainda descreve o módulo Tutor, baseado em cliente servidor e acessando uma base de estratégias pedagógicas e de perfis de usuários, mas este módulo não foi implementado.

Medeiros e Dazzi (2002) apresentam uma aplicação baseada na web para auxiliar o ensino de algoritmos e programação. Foi desenvolvido em Delphi e permite digitar um algoritmo e executá-lo, visualizando o conteúdo das variáveis.

Vargas (2005) descreve uma ferramenta de apoio ao ensino de programação escrita em Delphi, que permite a entrada de algoritmos em Portugol e a sua execução passo a passo com exibição do conteúdo das variáveis. Utilizou a ferramenta Gerador de Analisadores Léxicos e Sintáticos (GALS) para gerar o compilador da linguagem Portugol.

2.5.2 O aplicativo Construtor

O aplicativo Construtor, que acompanha o livro Construção de Algoritmos (FERNANDES; BOTINI, 1998), permite montar o fluxograma com os símbolos básicos e executá-lo passo a passo visualizando as variáveis. Permite a execução contínua ou com pontos de parada, além de oferecer funções de conversão do algoritmo para as linguagens Pascal, C, Clipper e Portugol e armazenar o algoritmo em disco.

A Figura 14 mostra uma tela do aplicativo Construtor. Há uma barra de componentes que podem ser inseridos no fluxograma. Uma seta vermelha indica o ponto onde será incluído o componente. Clica-se no fluxograma para posicionar a seta vermelha no local desejado e a seguir clica-se no componente a ser incluído. Uma janela se abre solicitando dados adicionais conforme o tipo da instrução.

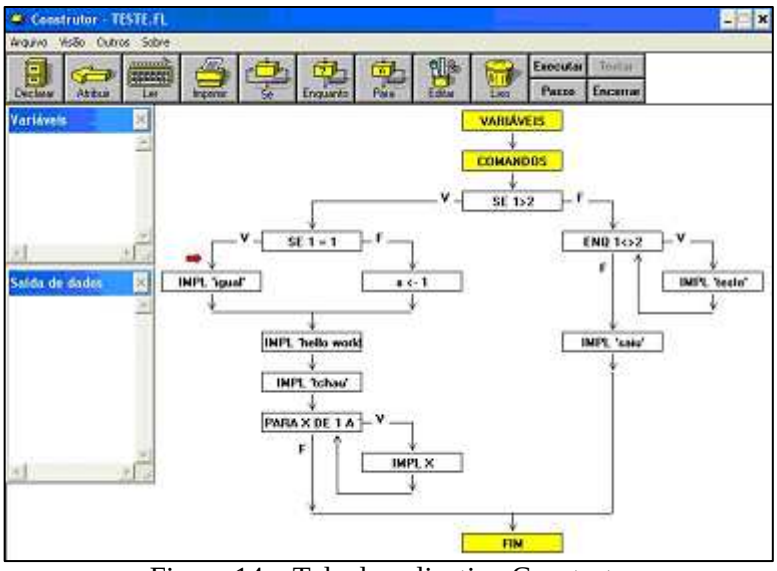


Figura 14 – Tela do aplicativo Construtor

O aplicativo suporta variável do tipo Lógica, Literal ou Vetor. O vetor é uma tabela de valores inteiros identificados por um índice entre colchetes junto ao nome da variável. O aplicativo suporta apenas vetores unidimensionais e não implementa variáveis alfanuméricas. No comando imprimir é possível informar um literal alfanumérico, mas ele deve estar entre apóstrofos.

O algoritmo pode ser executado normalmente, passo a passo ou com pontos de parada. Com os pontos de parada marcados executa-se normalmente e ao atingir o ponto marcado a execução pára. A execução então poderá continuar normalmente ou como passo a passo.

2.5.3 Ambiente para teste de mesa utilizando fluxograma

Cares (2002) desenvolveu em Delphi o Ambiente para teste de mesa utilizando fluxograma. A ferramenta disponibiliza os símbolos do fluxograma numa barra. O usuário constrói o algoritmo com os símbolos e depois o executa passo a passo visualizando o conteúdo das variáveis. A visualização é feita numa grade onde cada coluna se refere a uma variável e cada linha a um novo estado. Com isso consegue-se também um histórico dos valores ao longo da execução do algoritmo. A Figura 15 mostra a tela do aplicativo de Cares. São permitidas variáveis dos tipos inteiro, real, lógico e *string*.

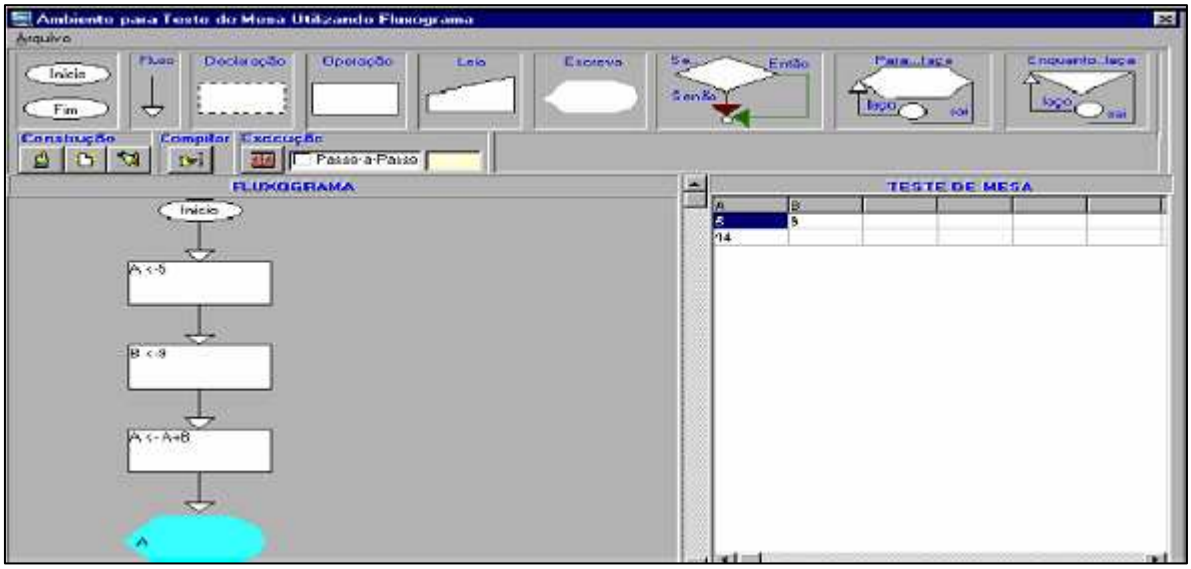


Figura 15 – Tela do aplicativo de Cares

2.5.4 Protótipo de um sistema de geração e animação de fluxogramas

Freitas (2003) implementou em Delphi um sistema que, a partir de um algoritmo em Portugol, gera o fluxograma e o executa passo a passo exibindo o valor das variáveis. As variáveis são criadas dinamicamente durante a execução. As variáveis são armazenadas como *strings*, mas se forem usadas em comandos de atribuição, seleção ou repetição deverão conter apenas valores inteiros ou reais. Em comandos de leitura e escrita, podem ter conteúdo alfanumérico. Literais alfanuméricos são aceitos apenas em comandos de escrita. A ferramenta permite ler, editar e salvar um arquivo texto contendo um algoritmo em Portugol. Não salva o fluxograma, por isso ao abrir um algoritmo existente deve-se novamente clicar no botão para compilar e gerar fluxograma. A tela deste aplicativo pode ser vista na Figura 16.

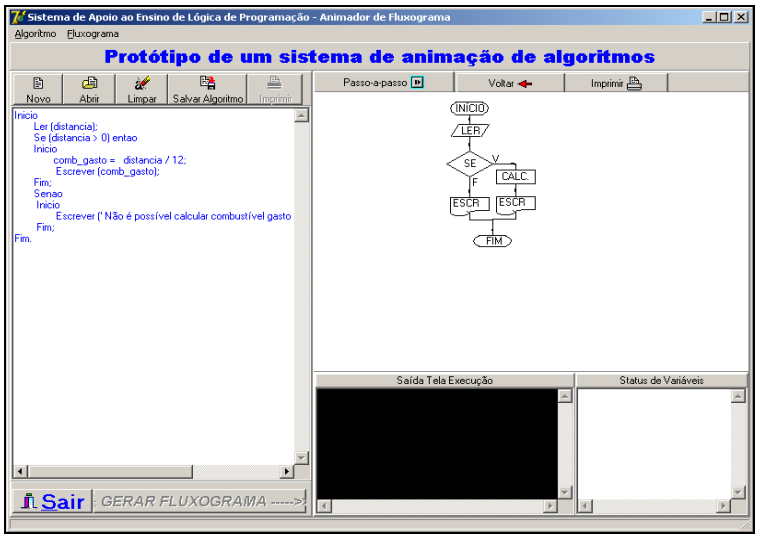


Figura 16 – Tela do aplicativo de Gilberto Freitas

2.5.5 O aplicativo CIFluxProg

Santiago e Dazzi (2004) apresentaram o CIFluxProg escrito em C++. Consiste de dois programas distintos. Com o primeiro, o aluno constrói um algoritmo montando os símbolos do fluxograma e o executa passo a passo. Com o segundo, digita o algoritmo em Portugol e o executa passo a passo. Nos dois casos o algoritmo pode ser salvo em disco no formato Portugol e aberto posteriormente em qualquer dos dois programas. A tela deste aplicativo pode ser vista na Figura 17. O sistema disponibiliza uma barra de ferramentas com os símbolos do fluxograma. Para montar um fluxograma o usuário apenas aponta o mouse para o ícone que corresponde ao símbolo desejado, e clica sobre ele para que seja criado na tela do computador um desenho representando o símbolo correspondente. Um símbolo inserido no fluxograma pode ser apagado clicando-se nele e em seguida clicando-se no botão apagar. O sistema suporta o aninhamento de símbolos. O aninhamento acontece quando se tem dentro de um símbolo do tipo Laço de Repetição ou Desvio Condicional um outro símbolo também destes mesmos tipos. Quando se quer apagar um aninhamento não há a necessidade de apagar cada símbolo, basta apagar o mais externo para que todos os símbolos vinculados internamente a este sejam também apagados do fluxograma. Todos os símbolos disponibilizados na barra de ferramentas possuem campos editáveis pelo usuário. É nesses campos que o usuário deve inserir os nomes e valores de variáveis, condições lógicas, etc.

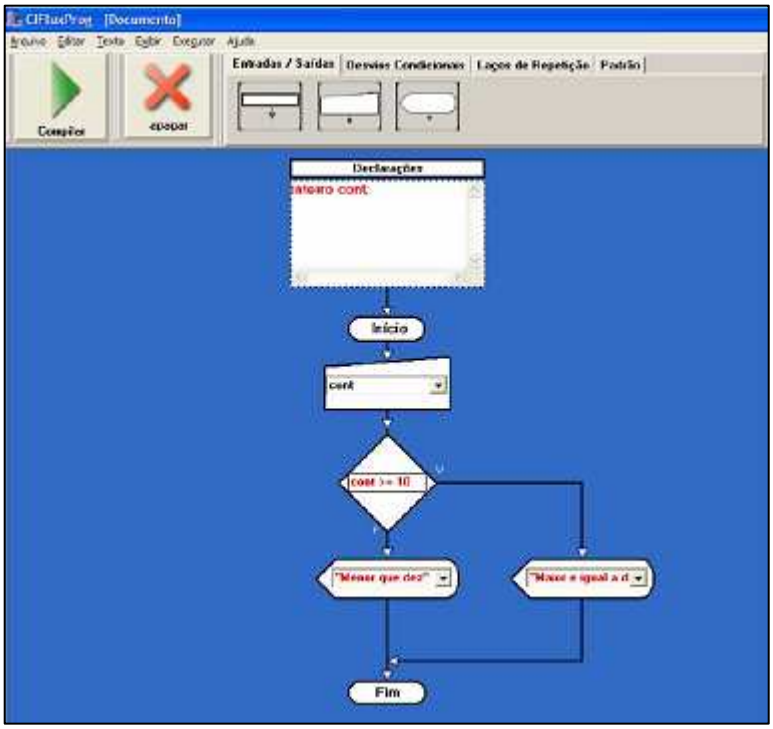


Figura 17 – Tela do aplicativo CIFluxProg

3 DESENVOLVIMENTO DO TRABALHO

Este capítulo descreve a especificação do AlgoSVG, um sistema de aprendizado de algoritmos pela web implementado com gráficos vetoriais. Na primeira seção estão descritos os requisitos principais do sistema, relacionando os requisitos funcionais, ou seja, tudo o que o sistema deve fazer e os requisitos não funcionais, ou seja, as restrições que o aplicativo deve atender. A segunda seção mostra a especificação do sistema. A terceira detalha as técnicas e ferramentas utilizadas bem como a operacionalidade da implementação. Na quarta seção são apresentados os resultados e discussões acerca do trabalho, comparando-o com os seus correlatos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Esta seção compreende o levantamento das funcionalidades necessárias para o aplicativo proposto. O Quadro 9 apresenta os requisitos funcionais do sistema e o Quadro 10 apresenta os requisitos não funcionais.

Requisito Funcional	**Implementado**
RF01: Permitir digitar informações na fase de projeto do algoritmo	X
RF02: Permitir incluir um comando no algoritmo	X
RF03: Permitir alterar um comando existente	X
RF04: Solicitar os dados para incluir ou alterar o comando	X
RF05: Permitir excluir um comando existente	X
RF06: Permitir incluir uma sub-rotina no algoritmo	X
RF07: Permitir alternar a visualização entre as sub-rotinas e o bloco principal	X
RF08: Permitir executar sem parada o algoritmo	X
RF09: Permitir executar passo a passo o algoritmo	X
RF10: Permitir encerrar prematuramente a execução do algoritmo	X
RF11: Permitir marcar pontos de parada no algoritmo	X
RF12: Destacar com outra cor a instrução sendo executada	X
RF13: Mostrar as variáveis e seus conteúdos durante a execução	X
RF14: Mostrar as saídas do algoritmo durante a execução	X
RF15: Mostrar a pilha de chamada de sub-rotinas durante a execução	X
RF16: Permitir parâmetros nas chamadas de sub-rotinas	X
RF17: Fornecer retorno com novos valores vindos da sub-rotina	X
RF18: Permitir salvar o algoritmo criado	X
RF19: Permitir abrir um algoritmo salvo previamente	X
RF20: Avisar se houver um algoritmo não salvo	X
RF21: Não abrir se o arquivo for inválido	X

Quadro 9 – Requisitos funcionais do AlgoSVG

Requisito não funcional	Tipo	Implementado
RNF01: O aplicativo deve rodar num navegador de web	Implementação	X
RNF02: Os gráficos devem ser vetoriais no formato SVG	Conformidade	X
RNF03: Código no cliente deve ser escrito em JavaScript	Conformidade	X
RNF04: Código no servidor deve ser escrito em PHP 4.5	Conformidade	X
RNF05: Deve funcionar no Firefox 1.5	Software	X
RNF06: Deve funcionar no Opera 9.0	Software	X
RNF07: Deve funcionar no IE6 com *plugin* da Adobe 6.0	Software	X
RNF08: Fornecer um menu para as operações disponíveis	Usabilidade	X
RNF09: Permitir interagir no fluxograma com o mouse	Usabilidade	X
RNF10: Fornecer diálogos ao incluir e alterar os comandos	Usabilidade	X
RNF11: Dispor os painéis em molduras flexíveis	Usabilidade	X
RNF12: Salvar e abrir no servidor com rotinas PHP 4.5	Conformidade	X
RNF13: Fornecer as telas do aplicativo por um servidor web	Implementação	X
RNF14: Usar a sintaxe do JavaScript nas expressões	Conformidade	X

Quadro 10 – Requisitos não funcionais do AlgoSVG

3.2 ESPECIFICAÇÃO

Os tópicos seguintes descrevem a especificação do aplicativo AlgoSVG. As funcionalidades do aplicativo são apresentadas na forma de casos de uso. Também são descritas as principais classes modeladas.

3.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso é mostrado na Figura 18 e descreve as funcionalidades que o aplicativo AlgoSVG fornece para o aluno. Um detalhamento de cada um dos casos de uso é fornecido a seguir.

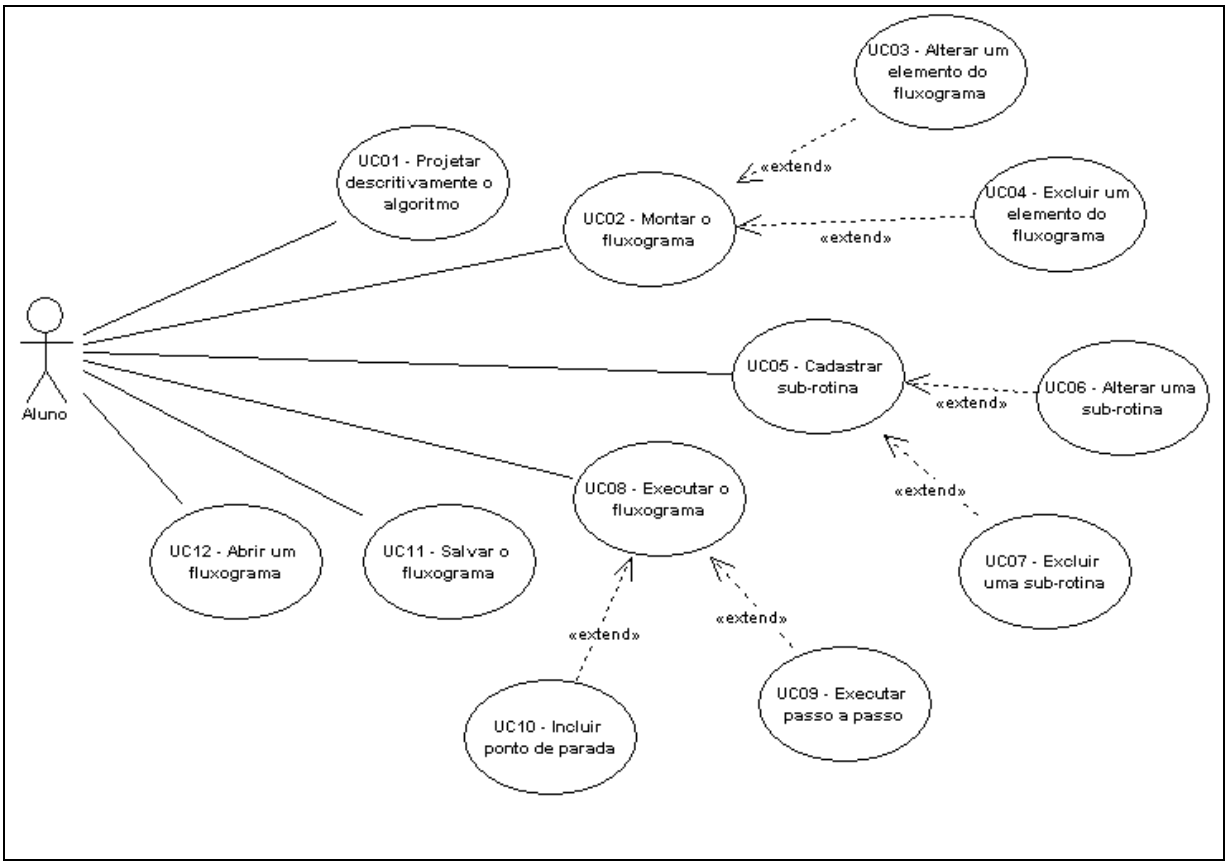


Figura 18 – Diagrama de casos de uso do AlgoSVG

UC01 – Projetar descritivamente o algoritmo
Cenário principal (relativo ao requisito funcional RF01): 1) Aluno identifica o problema mediante a leitura do enunciado. 2) Aluno obtém do enunciado quais as “entradas de dados” que serão fornecidas. 3) Aluno obtém do enunciado quais as “saídas de dados” que devem ser geradas. 4) Aluno define um conjunto de dados de entrada com as respectivas saídas para teste. 5) Aluno determina o que deve ser feito para transformar as “entradas” em “saídas”.

Quadro 11 – Detalhes do caso de uso: projetar descritivamente o algoritmo

UC02 – Montar o fluxograma
Cenário principal (relativo aos requisitos funcionais RF02 e RF04):
1) Aluno escolhe: leitura, escrita, atribuição, seleção, enquanto, para cada ou sub-rotina.
2) Sistema entra no estado “Inserindo comando” com a instrução escolhida marcada.
3) Aluno clica no local desejado para inserção no algoritmo.
4) Sistema abre tela pedindo dados para a instrução.
5) Aluno digita os dados para a instrução e confirma.
6) Sistema insere a instrução no algoritmo.
7) Aluno pode continuar no passo 3 inserindo o mesmo tipo de instrução em outro local.
Cenário alternativo:
No passo 5 o aluno em vez de confirmar pode cancelar a tela dos dados.
5.1) Sistema não insere a instrução no algoritmo.
5.2) Aluno pode continuar no passo 3 inserindo o mesmo tipo de instrução em outro local.

UC03 - Alterar um elemento do fluxograma
Cenário principal (relativo aos requisitos funcionais RF03 e RF04):
1) Aluno escolhe a opção Editar – Alterar.
2) Sistema entra no estado “Alterando comando”.
3) Aluno clica no comando a alterar.
4) Sistema abre tela com os dados atuais da instrução para o aluno alterar.
5) Aluno altera os dados e confirma.
6) Sistema altera a instrução com os novos dados.
7) Aluno pode continuar no passo 3 escolhendo outra instrução para alterar.
Cenário alternativo:
No passo 5 o aluno em vez de confirmar pode cancelar a tela dos dados.
5.1) Sistema não altera a instrução no algoritmo.
5.2) Aluno pode continuar no passo 3 escolhendo outra instrução para alterar.

UC04 - Excluir um elemento do fluxograma
Cenário principal (relativo ao requisito funcional RF05):
1) Aluno escolhe a opção Editar – Excluir.
2) Sistema entra no estado “Excluindo comando”.
3) Aluno clica no comando a excluir.
4) Sistema exclui o comando e se for um comando composto exclui também os sub-comandos.
5) Aluno pode continuar no passo 3 escolhendo outra instrução para excluir.

Quadro 12 – Detalhes do caso de uso: montar o fluxograma

UC05 – Cadastrar sub-rotinas
Cenário principal (relativo a requisito funcional RF06):
1) Aluno clica no botão “Incluir sub-rotina”.
2) Sistema pede os dados para o cabeçalho da sub-rotina.
3) Aluno digita nome para a sub-rotina e possíveis parâmetros que ela vai aceitar.
4) Sistema cria uma nova sub-rotina.
Cenário alternativo:
No passo 3 o aluno em vez de confirmar pode cancelar os dados para a sub-rotina.
3.1) Sistema cancela a inclusão da sub-rotina.

UC06 - Alterar uma sub-rotina
Cenário principal:
1) Aluno clica no botão “Alterar sub-rotina”.
2) Sistema abre tela com os dados atuais da sub-rotina ativa para o aluno alterar.
3) Aluno altera os dados do cabeçalho da sub-rotina e confirma.
4) Sistema altera o cabeçalho da sub-rotina com os novos dados.
Cenário alternativo (relativo a requisito funcional RF07):
No passo 1 o aluno pode visualizar outra sub-rotina diferente da atual.
1) Aluno clica na lista de seleção na moldura Sub-rotina e escolhe uma diferente da atual.
2) Sistema esconde o painel da sub-rotina atual e ativa o da sub-rotina escolhida.
Cenário alternativo:
No passo 3 o aluno em vez de confirmar pode cancelar os dados para a sub-rotina.
3.1) Sistema cancela a alteração da sub-rotina.
Cenário de exceção:
No passo 1 a sub-rotina ativa é o bloco principal.
1.1) O sistema ignora o botão Alterar quando a sub-rotina ativa é o bloco principal.

UC07 - Excluir uma sub-rotina
Cenário principal:
1) Aluno clica no botão “Excluir sub-rotina”.
2) Sistema exclui a sub-rotina e torna o bloco principal como ativo.
Cenário de exceção:
No passo 1 a sub-rotina ativa é o bloco principal.
1.1) O sistema ignora o botão Excluir quando a sub-rotina ativa é o bloco principal.

Quadro 13 – Detalhes do caso de uso: cadastrar sub-rotinas

UC08 - Executar o fluxograma
Cenário principal (relativo aos requisitos funcionais RF08, RF12, RF13, RF14, RF15, RF16 e RF17):
1) Aluno escolhe a opção Executar – Direto.
2) Sistema executa o algoritmo do início ao fim mostrando os dados durante o processo.
Cenário alternativo:
No passo 2 pode haver uma instrução de entrada de dados.
2.1) Sistema abre tela pedindo dados para a entrada necessária.
2.2) Aluno digita o dado e confirma.
2.3) Sistema continua no passo 2.
Cenário alternativo (relativo ao requisito funcional RF11):
No passo 2 uma instrução a ser executada está marcada com ponto de parada.
2.1) Sistema cancela a execução direta, mas não destrói o ambiente de execução.
Cenário alternativo (relativo ao requisito funcional RF10):
No passo 2 o aluno escolhe a opção Executar – Terminar.
2.1) Sistema destrói o ambiente de execução.

UC09 - Executar passo a passo
Cenário principal (relativo aos requisitos funcionais RF09, RF12, RF13, RF14, RF15, RF16 e RF17):
1) Aluno escolhe a opção Executar – Passo.
2) Sistema executa uma instrução e pára sem destruir o ambiente de execução.
3) Aluno escolhe novamente a opção Executar – Passo.
4) Sistema continua no passo 2.
Cenário alternativo:
No passo 2 pode haver uma instrução de entrada de dados.
2.1) Sistema abre tela pedindo dados para a entrada necessária.
2.2) Aluno digita o dado e confirma.
2.3) Sistema continua no passo 2.
Cenário alternativo (relativo ao requisito funcional RF10):
No passo 2 o aluno escolhe a opção Executar – Terminar.
2.1) Sistema destrói o ambiente de execução.

UC10 - Incluir ponto de parada
Cenário principal (relativo ao requisito funcional RF11):
1) Aluno escolhe a opção Executar – Ponto parada.
2) Sistema entra no estado “Marcando pontos de parada”.
3) Aluno marca um ponto de parada podendo continuar no passo 3 marcando outros pontos.

Quadro 14 – Detalhes do caso de uso: executar o algoritmo

UC11 – Salvar o algoritmo
Cenário principal (relativo ao requisito funcional RF18):
1) Aluno escolhe a opção Salvar.
2) Sistema abre uma nova tela pedindo nome do arquivo.
3) Aluno digita o nome do arquivo.
4) Sistema grava o algoritmo num arquivo com o nome informado.
Cenário de exceção (relativo ao requisito funcional RF12):
No passo 4 o algoritmo atual pode estar em execução.
4.1) Sistema destrói o ambiente de execução.
4.2) Sistema continua no passo 4.
Cenário de exceção (relativo ao requisito funcional RF11):
No passo 4 o algoritmo atual pode ter pontos de parada marcados.
4.1) Sistema continua no passo 4, pois os pontos de parada são salvos com o algoritmo.

Quadro 15 – Detalhes do caso de uso: salvar o algoritmo

UC12 – Abrir um algoritmo
Cenário principal (relativo ao requisito funcional RF19): 1) Aluno escolhe a opção Abrir. 2) Sistema abre uma nova tela pedindo nome do arquivo. 3) Aluno digita o nome do arquivo. 4) Sistema busca o algoritmo no arquivo informado. Cenário de exceção (relativo ao requisito funcional RF20): No passo 1 pode haver no aplicativo um algoritmo alterado e não salvo. 1.1) Sistema apresenta a mensagem “O algoritmo atual não está salvo. Continuar mesmo assim?”. Cenário de exceção (relativo ao requisito funcional RF21): No passo 4 o arquivo informado pode não existir ou não ser um arquivo válido. 4.1) Sistema apresenta a mensagem “Arquivo inválido: não é SVG”.

Quadro 16 – Detalhes do caso de uso: abrir um algoritmo

3.2.2 Diagrama de classes do AlgoSVG

A Figura 19 mostra o diagrama de classes do AlgoSVG. As classes são as seguintes:

a) `Fluxograma`: é a classe do objeto que contém todas as informações, blocos e elementos do fluxograma. Para salvar o fluxograma salva-se este objeto com todos os seus componentes, salvando assim o estado inteiro do fluxograma. Ao recuperar um fluxograma salvo, é este objeto que é substituído no aplicativo pelo objeto lido. Possui a propriedade `contaObjs` para dar nomes seqüenciais aos elementos que são criados;

b) `Bloco`: é uma classe abstrata da qual derivam classes cujos objetos conterão elementos de fluxograma;

c) `BlocoRotina`: é a classe do objeto que contém ou o bloco principal ou uma sub-rotina. Possui a propriedade `visibility` para torná-lo visível ou escondido, permitindo com isso alternar entre o bloco principal e as sub-rotinas;

d) `BlocoPrincipal`: contém as intruções de fluxograma do bloco principal;

e) `BlocoSubrotina`: contém as instruções de fluxograma de uma sub-rotina, bem como o nome da rotina e os parâmetros aceitos por ela;

f) `BlocoInstrucoes`: contém as intruções de fluxograma de um caminho de desvio. Instruções de fluxograma do tipo seleção e repetição possuem caminhos internos de desvio. Um objeto desta classe é um caminho de desvio;

g) `Instrucao`: é uma classe abstrata da qual derivam as classes cujos objetos são instruções de fluxograma. A propriedade `id` define um nome único no sistema para a instrução e consiste da concatenação do tipo da instrução com um número

seqüencial. A propriedade `expr` guarda o argumento da instrução, ou seja, os dados para o comando a ser executado. O método `tipoInstr(id)` retorna o tipo da instrução que está concatenado no nome único no sistema da instrução;

h) `InstrucaoLeitura`: é uma classe para objetos de instrução de fluxograma do tipo leitura. Esta instrução mostra na tela um texto solicitando um dado de entrada e espera este dado ser digitado. A propriedade `pergunta` contém o texto a ser mostrado. A propriedade `expr` contém o nome da variável que receberá a entrada;

i) `InstrucaoEscrita`: é uma classe para objetos de instrução do tipo escrita. Esta instrução mostra na tela uma saída de dados. A propriedade `expr` contém uma expressão a ser exibida na tela de saída;

j) `InstrucaoAtribuicao`: é uma classe para objetos de instrução do tipo atribuição. A propriedade `expr` contém um argumento de atribuição, contendo o nome da variável receptora e a expressão para produzir o valor a ser atribuído. Os métodos `attribVar(expr)` e `attribExpr(expr)` retornam o nome da variável e a expressão de atribuição, respectivamente;

k) `InstrucaoChamadaSubrotina`: é uma classe para objetos de instrução do tipo chamada de sub-rotina. A propriedade `expr` contém o nome da sub-rotina a ser chamada e uma lista dos parâmetros que serão passados para a sub-rotina. Os métodos `subNome(expr)` e `subParams(expr)` retornam o nome da sub-rotina e a lista de parâmetros, respectivamente;

l) `InstrucaoRepeticao`: é uma classe abstrata da qual derivam as classes cujos objetos contêm um caminho de desvio com instruções de fluxograma e executa essas instruções zero ou mais vezes;

m) `InstrucaoParaCada`: é uma classe para objetos de instrução do tipo “para cada”. Esta instrução usa uma variável que recebe um valor inicial e é incrementada até um valor final. A cada passo as instruções no caminho de desvio são executadas. Os métodos `paraVar(expr)`, `paraInicial(expr)` e `paraFinal(expr)` retornam o nome da variável contadora, o valor inicial e o valor final, respectivamente;

n) `InstrucaoEnquanto`: é uma classe para objetos de instrução do tipo enquanto. Esta instrução avalia a expressão na propriedade `expr` e, se resultar verdadeiro, executa as instruções do caminho de desvio;

o) `InstrucaoSelecao`: é uma classe para objetos de instrução do tipo seleção. Esta instrução avalia a expressão na propriedade `expr` e, se resultar verdadeiro, executa

as instruções no caminho de desvio denominado `sim`. Se por outro lado retornar falso, serão executadas as instruções no caminho de desvio denominado `nao`;

p) `AmbienteExecucao`: é uma classe para objetos de um ambiente de execução. Ao executar o fluxograma a ferramenta cria um ambiente de execução para o bloco principal. A cada chamada de sub-rotina um novo ambiente é criado para esta chamada. Pode haver recursividade, isto é, uma rotina ser chamada mais de uma vez antes de terminar as anteriores e, neste caso, haverá ambientes de execução separados para cada chamada sem importar se é o mesmo nome de sub-rotina. Nas chamadas pode haver passagem de parâmetros. Cada parâmetro pode ser uma variável ou uma expressão. Se for uma variável, no retorno da chamada a variável receberá o valor alterado na sub-rotina. A propriedade `prev` aponta para o ambiente de execução anterior e, quando a chamada se encerra, o ambiente é destruído e o anterior ativado. A propriedade `passoAtual` aponta para o objeto de instrução sendo executado. Ao final da execução da instrução, ou no retorno da chamada, é buscada a próxima instrução do bloco de instruções. A propriedade `modulo` aponta para a sub-rotina a que a instrução pertence. A propriedade `vars` contém as variáveis locais da sub-rotina ou do bloco-principal. A propriedade `names` contém uma lista com os nomes das variáveis locais. As propriedades `prmCal`, `prmSub` e `prmLen` guardam informações para retornar os valores para as variáveis ao final da chamada. A propriedade `fill` guarda a cor original da instrução sendo iniciada para restaurar posteriormente a sua cor correta. O método `execDireto` executa o fluxograma continuamente até o final ou até encontrar um ponto de parada.

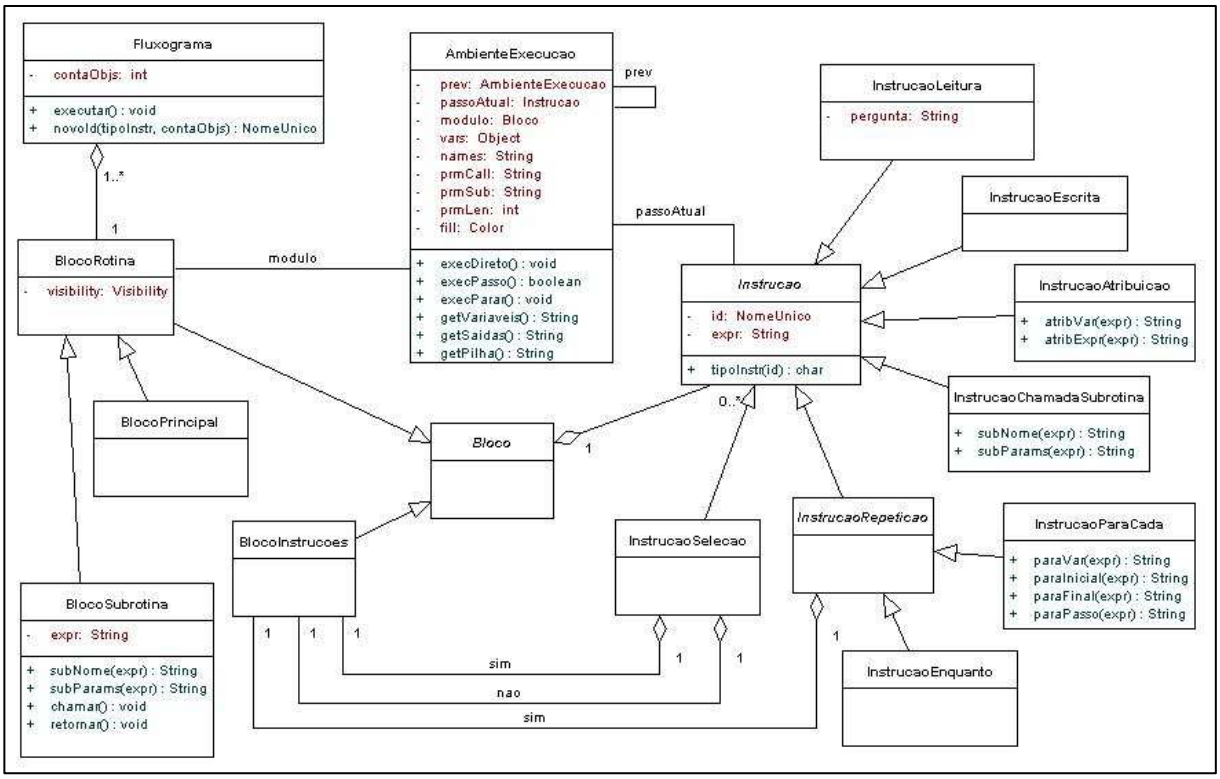


Figura 19 – Diagrama de classes do aplicativo AlgoSVG

3.3 IMPLEMENTAÇÃO

Nesta seção é detalhada a implementação do AlgoSVG, mostrando as ferramentas utilizadas no seu desenvolvimento e a sua operacionalidade.

3.3.1 Técnicas e ferramentas utilizadas

Aqui são detalhadas algumas técnicas usadas no desenvolvimento deste aplicativo. As rotinas escritas para editar o fluxograma, executar o algoritmo e demais facilidades presentes no aplicativo foram escritas em JavaScript.

3.3.1.1 Módulo de execução de algoritmo

O módulo de execução percorre os elementos do algoritmo conforme as ligações entre

eles e a cada elemento monta sua instrução correspondente. Se na instrução for necessário avaliar uma expressão, ela é entregue ao próprio JavaScript para fazê-lo através do comando `eval()`, mas num ambiente de variáveis próprio para não conflitar com as variáveis do aplicativo.

Se a instrução a executar for uma instrução de atribuição, primeiramente é criada a variável que vai receber o resultado da expressão, caso esta variável ainda não tenha sido criada. O Quadro 17 mostra a função para criar uma variável caso ela ainda não exista. Esta função recebe como parâmetro a variável que vai receber o resultado da expressão. Esta variável pode estar indexada e neste caso a função irá criar um vetor unidimensional com 255 posições, que é o tamanho máximo para um vetor no JavaScript. Apenas vetores unidimensionais foram implementados no AlgoSVG. As variáveis são criadas no ambiente de variáveis locais da sub-rotina, ou seja, em `dadosExec.vars`. Para verificar se a variável já existe é testado se o `typeof` da variável está `undefined`. A variável é criada com o valor `null` e seu tipo será definido depois, quando a ela for atribuído algum valor.

```
// Se expr tiver:   Total = Total + Valor
// O comando será:  if (eval(typeof dadosExec.vars.Total)== "undefined") {
//                    eval(dadosExec.vars.Total = null)}
//                    dadosExec.names += "Total|" }
function defineTalvez(variavel) {
  var ii = (variavel.indexOf("[") )
  if (ii>=0) {
    var nome = variavel.substr(0,ii)
    if (eval("typeof dadosExec.vars."+nome)=="undefined") {
      eval("dadosExec.vars."+nome+" = new Array(255)" )
      dadosExec.names += nome+"|"
    }
  }
  else
  if (eval("typeof dadosExec.vars."+variavel)=="undefined") {
    eval("dadosExec.vars."+variavel+" = null")
    dadosExec.names += variavel+"|"
  }
}
```

Quadro 17 – Função para criar uma variável ainda não existente

Após criar a variável ou confirmar que ela existe, o módulo de execução executa a instrução de atribuição do elemento do fluxograma, entregando a expressão para ser avaliada pelo JavaScript. O Quadro 18 mostra o código para avaliar uma expressão de atribuição, onde `dadosExec.vars` é o ambiente de variáveis locais da sub-rotina e `expr` a expressão contida no elemento do fluxograma. O comando para avaliação da expressão está em um bloco `try-catch` para tratar exceções e mostrar a mensagem ao aluno, em caso de erro de sintaxe.

```
// Se expr tiver:   Conta = Conta + 1
// o comando será:  with (dadosExec.vars) {Conta = Conta + 1}
// que equivale a:  dadosExec.vars.Conta = dadosExec.vars.Conta + 1
with (dadosExec.vars)
  try {
    eval(expr)
  }
  catch(E) {
    alert(E.name+": "+E.message+"\nErro no comando:\n"+ cmd)
  }
```

Quadro 18 – Avaliação de uma expressão de atribuição no ambiente de variáveis locais

Se a instrução a executar for uma instrução de seleção ou repetição, a expressão da instrução é avaliada no ambiente de variáveis locais da sub-rotina e o resultado retornado em `dadosExec.aux`, que poderá ser testado para escolher qual dos blocos de instruções, `sim` ou `nao`, será selecionado. A função `buscaBloco` retorna um objeto da classe BlocoInstrucoes. As expressões `cmdYes` ou `cmdNot` são constantes definidas no AlgoSVG e indicam o tipo de bloco a retornar. Como as instruções do fluxograma e os blocos de instruções são nódulos do DOM, a expressão `passo = bloco.firstChild` é usada para retornar o próximo elemento do fluxograma a executar dentro de um bloco de instruções.

```
// Se expr tiver:   Conta < 4
// o comando será:  with (dadosExec.vars) dadosExec.aux = (Conta < 4)
// que equivale a:  dadosExec.aux = (dadosExec.vars.Conta < 4)
bloco = null
with (dadosExec.vars) {
  if (catchEval("dadosExec.aux = ("+expr+")") )
    bloco = buscaBloco(dadosExec.passoAtual,cmdYes)
  else
    bloco = buscaBloco(dadosExec.passoAtual,cmdNot)
  }
if (bloco!=null)//apenas blocos de seleção tem o bloco "nao"
  passo = bloco.firstChild
if (passo!=null) {
  dadosExec.passoAtual = passo
else
  dadosExec.passoAtual = buscaProximaInstrucao()
```

Quadro 19 – Avaliação de uma expressão de seleção ou repetição

Nas chamadas de sub-rotinas, um novo ambiente é criado e ativado, ficando o ambiente anterior em espera até o retorno daquela chamada. A criação dinâmica de novo ambiente a cada chamada permite a chamada recursiva de sub-rotinas, situação que ocorre quando a mesma sub-rotina é chamada antes que a outra termine sem sobrescrever o ambiente anterior, mas criando um novo ambiente. As variáveis definidas na sub-rotina são criadas no novo ambiente e inicializadas com os valores passados na chamada. No retorno da sub-rotina, os valores dos parâmetros passados podem ter sido alterados, e por isso são retornados para o bloco que chamou, na variável da mesma posição do parâmetro. No caso de ter sido passada uma expressão em vez de uma variável, o novo valor é descartado, pois apenas variáveis

podem receber valores no retorno.

O Quadro 20 mostra o método `constructor` para o ambiente de execução. O atributo `prev` é uma referência para outro ambiente de execução, ou seja, o ambiente de onde veio a chamada da sub-rotina. Se este atributo contiver `null` é porque está executando o bloco principal e não uma sub-rotina, e ao final do bloco a execução do algoritmo é encerrada. Se o atributo `prev` não estiver `null`, no final do bloco é feita a passagem dos valores alterados na sub-rotina para as variáveis no bloco anterior, o ambiente anterior passa a ser o ambiente ativo e continua a execução na instrução seguinte, no ambiente restaurado.

```
function AmbienteExec(prev,passo,bloco)
{this.prev = prev         // ambiente da sub-rotina chamadora
  this.passoAtual = passo // elemento do fluxograma sendo executado
  this.modulo = bloco     // referencia para o bloco sendo executado
  this.vars = new Object  // variáveis locais da sub-rotina
  this.names = ""         // nomes das variáveis, separadas por "|"
  this.prmCall = ""       // parâmetros na instrução de chamada
  this.prmSub = ""        // parâmetros no cabeçalho da sub-rotina
  this.prmLen = 0         // quantidade de parâmetros
  this.fill = null}       // cor original do elemento em passoAtual
```

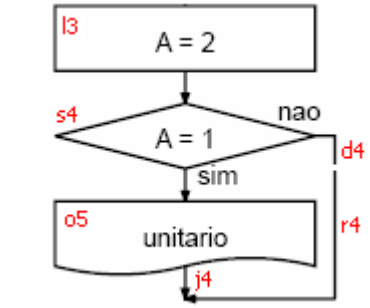
Quadro 20 – *constructor* do objeto para ambiente de execução

3.3.2 Estrutura de dados interna para os elementos do fluxograma

O fluxograma é montado com elementos SVG. Esses elementos ficam armazenados no DOM, que é uma estrutura de dados hierárquica dos objetos no documento. A Figura 20 mostra uma parte de um fluxograma com três instruções: um comando de atribuição seguido de um comando de seleção, que contém um comando de escrita no caminho `sim`. Na figura são mostrados o desenho do fluxograma, a estrutura hierárquica do DOM e o código SVG. O código não está completo, pois muitos atributos foram removidos para deixá-lo mais legível.

O comando de atribuição é composto por uma marcação `<SVG>` que contém outras três marcações: `<path>` para a seta acima do retângulo, `<rect>` para o retângulo e `<text>` para a legenda. Os elementos recebem um identificador único através do atributo `id`. O AlgoSVG gera identificadores únicos concatenando uma letra indicadora do tipo de instrução mais um número seqüencial. Os tipos de instrução e suas letras correspondentes podem ser vistos no Quadro 21. O comando de atribuição do exemplo recebeu `id="l3"`, a letra “L” concatenada com o número 3, na marcação `<svg>`. As marcações subordinadas recebem o mesmo `id`, mas com um sufixo. A marcação `<rect>` recebeu `id="l3_shape"` e a marcação `<text>` recebeu `id="l3_txt"`.

O comando de seleção é mais complexo. É composto por uma marcação `<svg>` que contém, além de marcações `<path>` e `<text>`, outras marcações `<svg>` para o bloco `nao`, o bloco `sim` e um bloco auxiliar que contém as setas e linhas auxiliares da estrutura de seleção. O comando de seleção recebeu `id="s4"`, o bloco `nao` recebeu `id="n4"`, o bloco `sim` recebeu `id="y4"`, o bloco auxiliar recebeu `id="a4"`, a linha inicial do caminho `nao` recebeu `id="d4"`. A seta final do bloco `nao` recebeu `id="r4"` e a linha final do bloco `sim` recebeu `id="j4"`.




```
<svg id="l3" x="10" y="30"><!-- comando atribuição -->
  <path d="M50,0 L50,7 L51,7 L50,10 L49,7 L50,7 Z"/>
  <rect id="l3_shape" x="10" y="10" width="80"/>
  <text id="l3_txt" x="50" y="24">A = 2</text>
</svg>
<svg id="s4" x="10" y="60"><!-- comando seleção -->
  <text x="85" y="15">nao</text><!—legenda nao -->
  <text x="60" y="35">sim</text><!—legenda sim -->
  <path d="M50,0 L50,7 L51,7 L50,10 L49,7 L50,7 Z"/>
  <path id="s4_shape" d="M10,20L50,10L90,20"/>
  <text id="s4_txt" x="50" y="24">A = 1</text>
  <svg id="n4" x="91" y="30"/><!-- bloco nao -->
  <svg id="y4" x="0" y="30"><!-- bloco sim -->
    <svg id="o5" x="0" y="0"><!-- comando escrita -->
      <path d="M50,0L50,7L51,7L50,10L49,7L50,7Z"/>
      <path id="o5_shape" d="M10 10L10 30Q30,35"/>
      <text id="o5_txt" x="50" y="24">unitario</text>
    </svg>
  </svg>
  <svg id="a4" x="0" y="0"><!—- setas de retorno -->
    <polyline id="d4" points="90,20 96,20 96,30"/>
    <path id="r4" d="M96,30L96,70L53,70L53,71"/>
    <path id="j4" d="M50,60L50,67L51,67L50,70"/>
  </svg>
</svg>
```

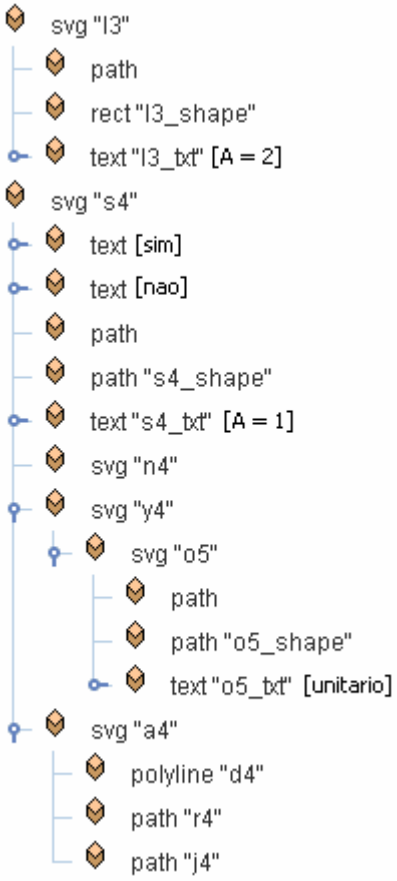


Figura 20 – Estrutura de dados interna dos elementos do fluxograma

```
cmdBegin  = "b" // instrução inicio
cmdEnd    = "e" // instrução fim
cmdCall   = "c" // instrução chamada de sub-rotina
cmdFor    = "f" // instrução para cada
cmdSelect = "s" // instrução seleção
cmdJoin   = "j" // seta ao final do bloco sim
cmdReturn = "r" // seta ao final do bloco nao
cmdDepart = "d" // seta no início do bloco nao
cmdInput  = "i" // instrução leitura
cmdOutput = "o" // instrução escrita
cmdLet    = "l" // instrução atribuição
cmdWhile  = "w" // instrução enquanto
cmdYes    = "y" // bloco sim
cmdNot    = "n" // bloco não
cmdArms   = "a" // linhas adicionais dos comandos select, while e for
```

Quadro 21 – Tipos de instrução ou elemento no AlgoSVG e suas letras correspondentes

Como foi visto, o fluxograma no AlgoSVG é uma estrutura hierárquica de viewports e elementos gráficos. A Figura 21 mostra em diagramas como estão dispostos instruções e blocos de instruções. O bloco principal e as sub-rotinas ocupam a mesma posição e tamanho no fluxograma, mas através do atributo `visibility` apenas um se apresenta visível.

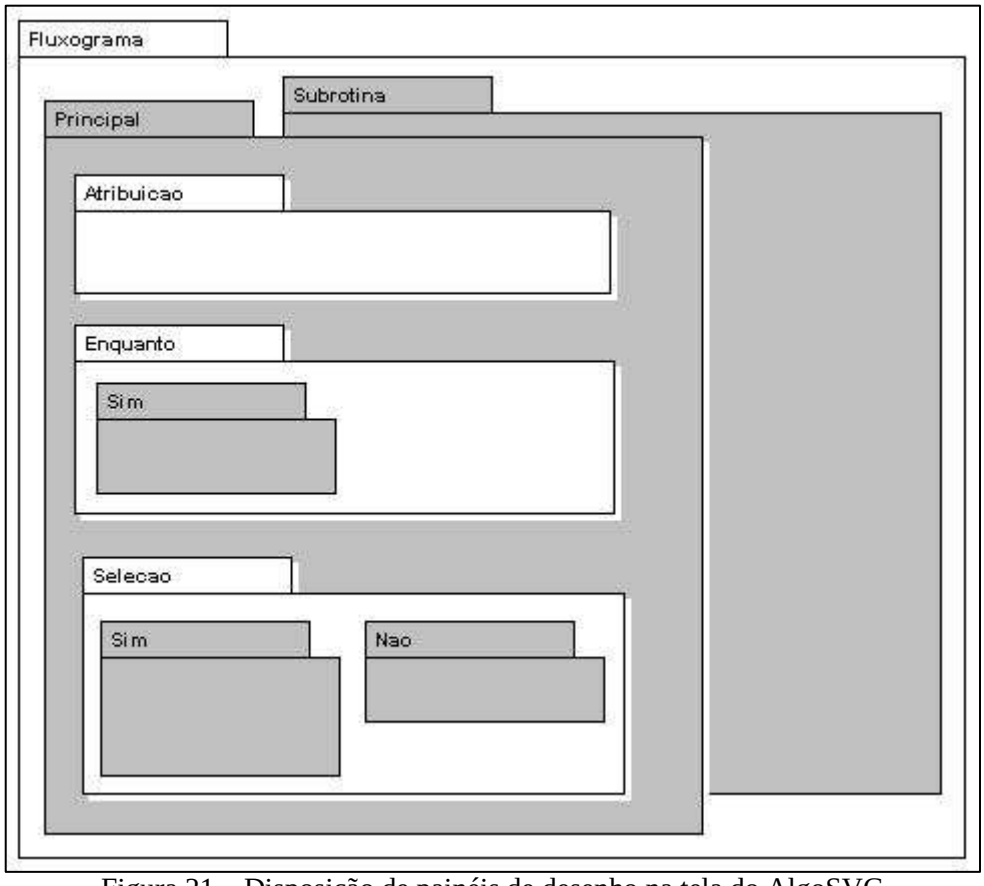


Figura 21 – Disposição de painéis de desenho na tela do AlgoSVG

3.3.3 Cálculo da posição dos elementos no desenho do fluxograma

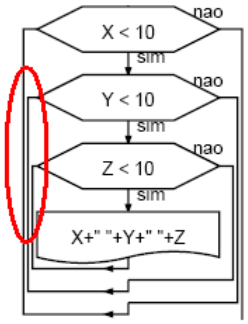
O AlgoSVG calcula a posição dos elementos, refazendo todo o conjunto sempre que um elemento é inserido ou excluído do fluxograma. A função `reposicionaTudo()`, mostrada no Quadro 22, chama três outros processos em seqüência, que calculam atributos diferentes. Os dois primeiros processos apenas calculam e gravam os atributos. O último calcula e chama a função de redesenho. Os três processos são recursivos e fazem uma busca em profundidade na estrutura.

```
function reposicionaTudo() {
  reposicionaEsq(moduloAtual)
  reposicionaDir(moduloAtual)
  reposicionaY(moduloAtual)
}
```

Quadro 22 – Função no AlgoSVG que comanda o reposicionamento de todos os elementos

A função `reposicionaEsq()`, mostrada no Quadro 23, busca por elementos de repetição, gravando neles o atributo `esq`, que indica o deslocamento para a esquerda da linha de retorno da estrutura.

```
function reposicionaEsq(bloco) {
  var esq = 0
  var aux = 0
  var node = bloco.firstChild
  while (node!=null) {
    switch (buscaTipo(node) ) {
      case cmdFor:
      case cmdWhile:
        aux = reposicionaEsq(buscaBloco(node,cmdYes) )
        fillAttr(node,"esq="+aux )
        aux += 2
        if (aux>esq)
          esq = aux
        break
    }
    node = node.nextSibling
  }
  return esq
}
```



Quadro 23 – Função que posiciona as linhas de retorno dos comandos de repetição

A função `reposicionaDir()`, mostrada no Quadro 24, busca por elementos de seleção e repetição gravando neles o atributo `dir`, e nos blocos `nao` o atributo `x`.

```
function reposicionaDir(bloco) {
  var dir = 0; var dy = 0; var dn = 0; var aux = 0
  var yes = null; var not = null; var node = bloco.firstChild
  while (node!=null) {
    switch (buscaTipo(node) ) {
      case cmdSelect:
        yes = buscaBloco(node,cmdYes)
        not = buscaBloco(node,cmdNot)
        dy = reposicionaDir(yes)
        dn = reposicionaDir(not)
        if (dy==1) dy==0
        fillAttr(not,"x="+(90+dy) )
        if (dn==0)
          aux = (dy)//+2
        else
          if (dn==1)
            aux = (dy+90)
          else
            aux = (dy+dn+90)
        aux += 2; fillAttr(node,"dir="+(aux) )
        if (aux>dir) dir = aux
        break
      case cmdFor:
      case cmdWhile:
        aux = reposicionaDir(buscaBloco(node,cmdYes) )
        fillAttr(node,"dir="+aux ); aux += 2
        if (aux>dir) dir = aux
        break
      default:
        if (dir==0) dir = 1
        break }
    node = node.nextSibling }
  return dir }
```

Quadro 24 – Função no AlgoSVG que reposiciona os elementos no eixo X

A função `reposicionaY()`, mostrada no Quadro 25, recalcula a posição vertical de todos os elementos. Esta função, por ser a última a ser chamada, faz também o redesenho, chamando duas outras funções que redesenham as estruturas de seleção e repetição, respectivamente. As funções de redesenho apenas refazem as linhas e setas auxiliares dos comandos de seleção e repetição, pois a posição dos demais elementos é feita automaticamente na alteração dos atributos x e y. Os elementos são reposicionados pela alteração do atributo y. Apenas os blocos nao precisam ser reposicionados no eixo x, e uma vez reposicionados, todos os seus elementos seguem junto.

```
function reposicionaY(bloco) {
  var y = 0; var y1 = 0; var y2 = 0; var tipo = ""
  var node = bloco.firstChild
  while (node!=null) {
    fillAttr(node,"y="+(y) )
    tipo = buscaTipo(node)
    switch (tipo) {
      case cmdSelect:
        y1 = reposicionaY(buscaBloco(node,cmdNot) )
        y2 = reposicionaY(buscaBloco(node,cmdYes) )
        if (y1>y2)
          y2 = y1
        y += (y2+40)
        refazSelect(node)
        break
      case cmdFor:
      case cmdWhile:
        y2 = reposicionaY(buscaBloco(node,cmdYes) )
        y += (y2+40)
        refazDesvio(node)
        break
      default:
        y += 30
        break
    }
    node = node.nextSibling
  }
  return y
}
```

Quadro 25 – Função no AlgoSVG que reposiciona os elementos no eixo Y

3.3.4 Operacionalidade do AlgoSVG

O sistema inicia na tela de projeto, que pode ser vista na Figura 22, onde o aluno pode analisar o problema e escrever uma solução seguindo os seguintes passos:

a) identificar o problema mediante leitura atenta do enunciado;

b) retirar do enunciado quais as “entradas de dados” que serão fornecidas;

c) retirar do enunciado quais as “saídas de dados” que devem ser geradas;

d) definir um conjunto de dados de entrada com as respectivas saídas para teste;

e) determinar o que deve ser feito para transformar as “entradas” em “saídas”.

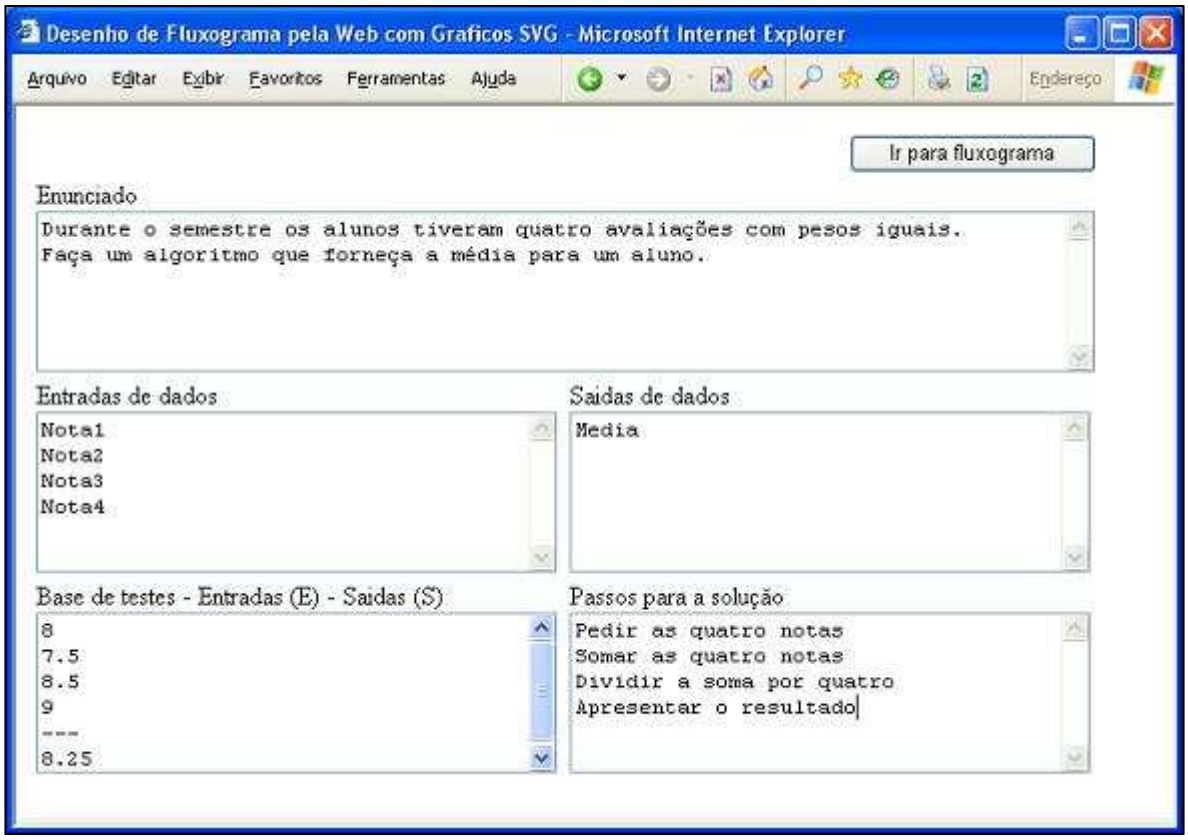


Figura 22 – Tela do AlgoSVG: fase de projeto do algoritmo

Estando pronta a análise do algoritmo clica-se, no botão `Ir para fluxograma` e o navegador passa a apresentar a tela seguinte, onde é feita a construção do fluxograma. Não existe no AlgoSVG uma integração automática entre as fases de projeto e de montagem. O aluno, após ter definido os passos para a solução na fase de projeto, passa para a montagem do fluxograma, onde ali inclui os elementos no fluxograma para corresponderem aos passos para a solução do problema que ele definiu na fase de projeto.

A Figura 23 mostra a tela de montagem do fluxograma. No exemplo mostrado na figura o aluno já incluiu os elementos do fluxograma e executou o algoritmo. Pode-se ver nas molduras `Variáveis` e `Saídas` o último estado do algoritmo ao final da execução.

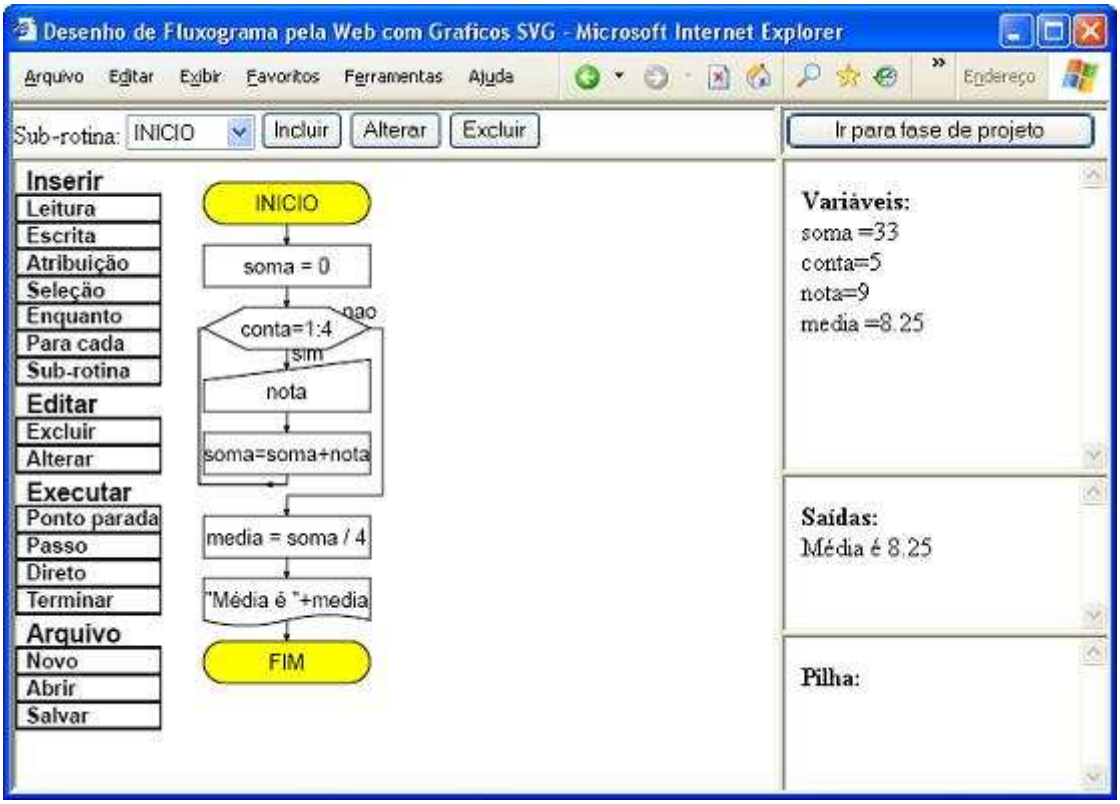


Figura 23 – Tela do AlgoSVG: montagem e execução do fluxograma

No lado esquerdo da tela para montagem do fluxograma existe um menu onde o aluno seleciona o tipo de instrução que quer inserir. O aluno então clica sobre o fluxograma no local onde quer a instrução. O cursor do mouse muda de figura sempre que passa no painel gráfico sobre um elemento que aceita clique, informando ao aluno que ali é um ponto para inclusão. Assim que o aluno clica no local de inserção, surge uma tela menor pedindo os dados para montar a instrução, como mostrado na Figura 24, onde a tela pede dados para o comando `Leitura`.

Figura 24 – Tela do AlgoSVG: dados para a instrução Leitura

A Figura 25 mostra o processo de inclusão de sub-rotina. No exemplo vê-se que já existe a chamada para a sub-rotina com a legenda `TrazMedia(mf)`. O AlgoSVG trabalha com o conceito de sub-rotina, e se for desejado retorno de valor, deve-se passar uma variável já existente nos parâmetros da chamada, por isso no exemplo da Figura 25 existe a instrução `mf = 0` antes da chamada da sub-rotina e a variável `mf` sendo passada como parâmetro. No canto superior esquerdo da tela existe uma moldura que mostra a sub-rotina selecionada para visualização, neste caso sendo o próprio programa principal. Neste frame existem três botões, sendo que para incluir uma nova sub-rotina deve-se clicar no botão `Incluir`. Uma janela se abre pedindo os dados para montar a nova sub-rotina.

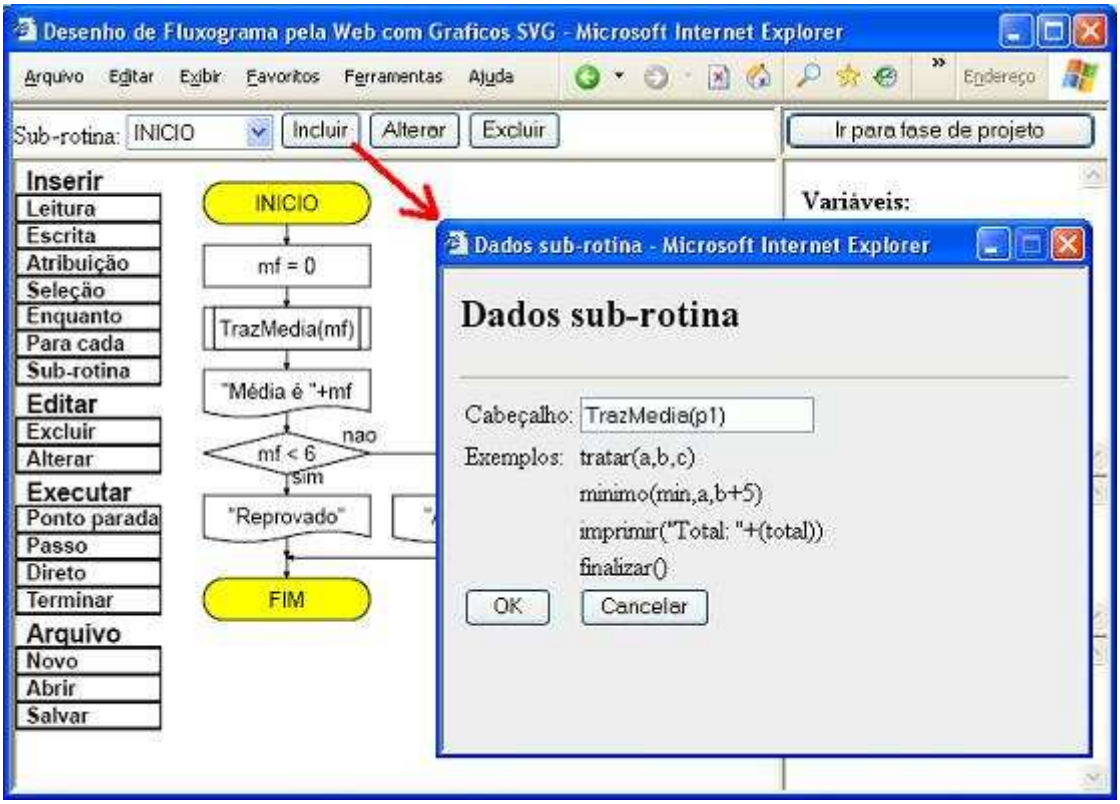


Figura 25 – Tela do AlgoSVG: incluindo uma sub-rotina

Não necessariamente a chamada da sub-rotina precisa ser feita antes, mas pode-se criar a sub-rotina e depois colocar as chamadas para ela. Ao incluir a sub-rotina o aplicativo já a seleciona para visualização, escondendo a que estava selecionada anteriormente. Na Figura 26 pode-se ver o processo para alterar a sub-rotina ativa. Seleciona-se a sub-rotina desejada no controle de listas e ela passa a ser apresentada na tela.

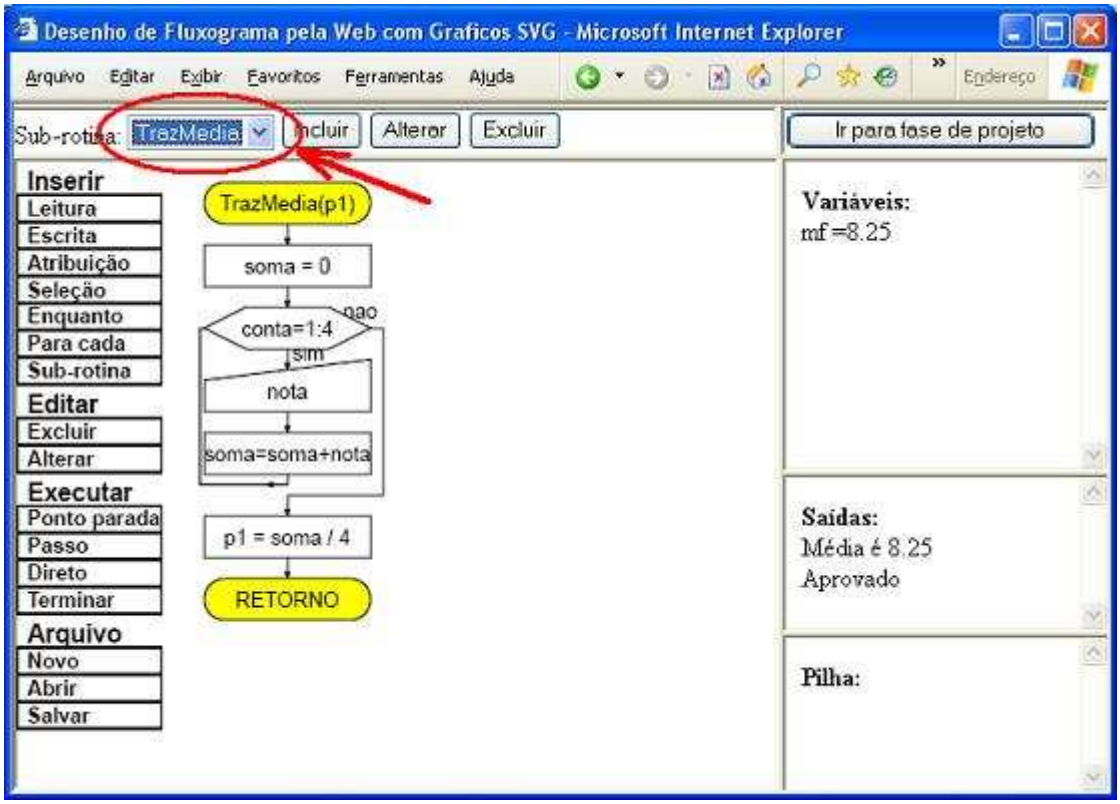


Figura 26 – Tela do AlgoSVG: sub-rotina sendo visualizada

No menu do lado esquerdo existe o grupo de botões denominado `Editar`. Neste grupo existem os botões `Excluir` e `Alterar`. Clicando-se em `Excluir`, este botão fica selecionado e todo clique sobre um elemento do fluxograma vai excluir o elemento. Clicando-se em `Alterar` esse outro botão fica selecionado e todo clique do mouse sobre um elemento do fluxograma irá abrir a janela de dados da instrução, permitindo a alteração.

Um outro grupo de botões no menu do lado esquerdo é o `Executar`. Clicando-se no botão `Ponto parada`, ele fica selecionado e todo clique sobre um elemento do fluxograma irá marcá-lo como ponto de parada. Numa execução direta, o algoritmo será executado. Se chegar numa instrução marcada, a execução é interrompida, podendo ser continuada depois por comando do aluno. Uma instrução marcada como ponto de parada fica sinalizada com cor vermelha. Clicando-se a segunda vez na instrução o ponto de parada é removido e a instrução volta para cor original. Clicando-se no botão `Passo`, apenas uma instrução do algoritmo é executada. Clicando-se no botão `Direto` o algoritmo é executado sem pausa até chegar ao fim ou a uma instrução marcada como ponto de parada. Clicando-se no botão `Terminar` a execução é interrompida e o ambiente de execução é destruído. Durante a execução direta ou passo a passo as variáveis, a pilha de sub-rotinas chamadas e as saídas são mostradas nas molduras no lado direito da tela. A instrução sendo executada é sinalizada em cor verde.

3.4 RESULTADOS E DISCUSSÕES

Para atingir seu objetivo este trabalho valeu-se do estudo de trabalhos anteriores, procurando levar para um novo contexto à criação e execução interativa de algoritmos, visando fornecer mais uma ferramenta para auxiliar o aluno no seu aprendizado. Um comparativo entre as características do AlgoSVG e os trabalhos correlatos pode ser visto no Quadro 26.

	Tagliari (1996)	Schmitt (1998)	Almeida et al. (2002)	Medeiros e Dazzi (2002)	Vargas (2005)	Construtor	Cares (2002)	Freitas (2003)	Santiago e Dazzi (2004)	AlgoSVG
Executa direto e em passos, exibe variáveis, sinaliza instrução atual	x	x	x	x	x	x	x	x	x	x
O aluno pode escrever seus próprios algoritmos	-	-	x	x	x	x	x	x	x	x
Permite salvar o algoritmo	-	-	x	x	x	x	-	x	x	x
Permite matrizes unidimensionais	-	x	x	x	x	x	-	-	-	x
Variáveis do tipo string	x	x	x	-	x	-	x	-	x	x
O usuário pode declarar as variáveis e definir seus tipos	-	-	x	-	x	x	x	-	x	-
Exibe fluxograma e permite aninhar comandos de seleção e repetição	-	-	-	-	-	x	x	x	x	x
Converte entre portugol e fluxograma	-	-	-	-	-	x	-	x	x	-
Permite pontos de parada	-	-	-	-	-	x	-	-	-	x
Permite sub-rotinas, chamadas recursivas e exibe a pilha de execução	-	-	x	-	-	-	-	-	-	x
O usuário pode alterar o conteúdo das variáveis durante a execução	x	x	-	-	-	x	x	-	-	-
Executa na web	-	-	-	x	-	-	-	-	-	x
Utiliza gráficos vetoriais na web	-	-	-	-	-	-	-	-	-	x
Linguagem de programação usada no desenvolvimento	Visual Basic	Delphi	Java	Delphi	Delphi		Delphi	Delphi	C++	SVG + JavaScript
Formato de entrada do algoritmo	Portugol	Portugol	Portugol	Portugol	Portugol	Fluxograma	Fluxograma	Portugol	Portugol/Fluxograma	Fluxograma

Quadro 26 – Comparativo entre o AlgoSVG e os trabalhos correlatos

O diferencial do AlgoSvg em relação aos outros trabalhos correlatos é exibir gráficos

vetoriais na web. Isto permite ser utilizado pelo aluno em qualquer lugar com acesso a web, disponibilizando os algoritmos salvos no servidor. Sendo disponibilizado a partir de um servidor web, as atualizações do aplicativo ficam facilitadas, pois uma vez atualizado no servidor estará disponível aos alunos imediatamente.

No AlgoSVG as variáveis não podem ser definidas pelo aluno, mas são criadas automaticamente pelo aplicativo durante a execução do algoritmo. Duas características presentes em outros trabalhos não foram implementadas no AlgoSVG que é a conversão do fluxograma para Portugol e alteração das variáveis durante a execução.

Uma limitação no AlgoSVG é a falta de funções para *zoom* e *pan*, que são úteis para algoritmos muito grandes que extrapolam o tamanho da tela.

O trabalho não pôde ser testado em um ambiente real de ensino, pois a implementação do AlgoSVG não terminou em tempo para ser usada no semestre letivo, mas pelas suas características conclui-se que será de grande auxílio no ensino de algoritmos.

4 CONCLUSÕES

O objetivo deste trabalho era criar um assistente com interface gráfica para o ensino de algoritmos através de fluxogramas, porém colocando-o num contexto ainda pouco explorado que é a web, e com uma nova tecnologia que é o SVG, um padrão para gráficos vetoriais dinâmicos e interativos na web. O AlgoSVG disponibiliza uma interface gráfica no navegador de web, onde o aluno pode montar e testar o algoritmo. A própria ferramenta calcula a posição dos elementos, refazendo todo o conjunto sempre que um elemento é inserido ou excluído do fluxograma. Instruções compostas podem ser aninhadas, permitindo instruções de seleção e repetição dentro umas das outras.

O AlgoSVG permite a construção de algoritmos modularizados com o conceito de sub-rotina. Ele permite criar sub-rotinas e chamá-las com passagem e retorno de parâmetros. Chamadas recursivas são possíveis, pois a cada chamada a ferramenta salva o ambiente anterior e cria um novo para a nova instância da sub-rotina. No retorno da sub-rotina os parâmetros modificados são repassados e o ambiente da sub-rotina é destruído.

Para estimular no aluno as fases de análise do problema e projeto da solução, foi disponibilizada uma tela onde o aluno pode ler o enunciado do problema e em campos específicos escrever quais são os dados de entrada necessários, quais as saídas desejadas, dados de exemplo para teste e os passos gerais para a solução do problema. A qualquer momento, mesmo durante o teste do algoritmo, o aluno pode alternar entre as telas de projeto e de montagem do fluxograma. Não foi feita integração mais completa entre as telas de projeto e de montagem do algoritmo, como gerar elementos no fluxograma a partir dos dados na tela de projeto ou usar para dados de entrada durante a execução do algoritmo os dados de teste definidos na fase de projeto.

O AlgoSVG permite salvar no servidor os algoritmos criados e recuperá-los posteriormente. Para salvar é necessário que a pasta no servidor esteja configurada com permissão de gravação. Devido às restrições de segurança, um aplicativo web não pode ler ou gravar na máquina cliente, apenas enviar ou solicitar dados para o servidor web. O aplicativo possui rotinas rodando no servidor que se encarregam se salvar e recuperar os algoritmos no espaço do servidor. O aplicativo não tem controle de acesso por usuários, mas todos têm acesso à mesma pasta de salvamento.

A utilização do padrão SVG neste trabalho demonstrou que esta tecnologia é viável para este tipo de aplicação e também para qualquer outra área que necessitar de gráficos

interativos e dinâmicos na web. Os elementos gráficos do SVG interagem facilmente com outras tecnologias como o JavaScript. Por sua facilidade de programação, o padrão SVG permite que num projeto o esforço para o desenvolvimento gráfico e interativo seja menor, transferindo mais recursos de tempo e programação para o fim a que se destina o software.

4.1 EXTENSÕES

Em trabalhos futuros a forma de salvamento dos algoritmos no servidor pode ser mudada para banco de dados em vez de arquivos em disco, o que permitirá integrar o aplicativo a uma base de estratégias de ensino e perfis de usuário, como proposto, mas não implementado, por Almeida et al. (2002).

Outra alteração sugerida é deixar o aluno definir as variáveis e seus tipos, em vez de fazer isso dinamicamente. Isto além de dar mais controle ao aluno sobre os tipos das variáveis, evitaria alguns problemas, como ao criar um vetor ter de alocar o tamanho máximo, por não ser possível determinar qual o maior índice será usado para aquele vetor no algoritmo.

Como sugestão de melhoria na ferramenta fica implementar as duas características presentes em outros trabalhos similares, que são importar e exportar algoritmos em linguagem Portugol e permitir alterar as variáveis durante a execução. E ainda implementar funções para *zoom* e *pan*, que são úteis para algoritmos muito grandes que extrapolam o tamanho da tela.

Uma ferramenta similar para o ensino de orientação a objetos poderia ser desenvolvida utilizando o formato SVG.

REFERÊNCIAS BIBLIOGRÁFICAS

ADOBE. **Adobe SVG Viewer pre-release download area**: version 6.0 preview 1. San Jose, 2003. Disponível em: <http://www.adobe.com/svg/viewer/install/beta.html>. Acesso em: 11 nov. 2006.

ADOBETUTORIALZ. **SVG Viewer**: SVG path. [S.l.], 2006. Disponível em: <http://www.adobetutorialz.com/categories/Print-and-Web-Publishing/SVG-Viewer>. Acesso em: 12 nov. 2006.

ALMEIDA, Eliana Silva de et al. **AMBAP**: um ambiente de apoio ao aprendizado de programação. Maceió, 2002. Não paginado. Disponível em: <http://www.students.ic.unicamp.br/~ra030015/publicacoes/arquivos/2002SbcAmbap.pdf>. Acesso em: 10 dez. 2006.

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

CARES, Paula Lorena Lovera. **Ambiente para teste de mesa utilizando fluxograma**. 2002. 92 f. Trabalho de Conclusão de Curso (Ciências da Computação) - Centro de Educação Superior de Ciências Tecnológicas, da Terra e do Mar, Universidade do Vale do Itajaí, Itajaí.

CORRIGAN, John. **Computação gráfica**: segredos e soluções. Tradução Hugo de Souza Melo e Mariza de Andrade Flores. Rio de Janeiro: Ciência Moderna, 1994.

EISENBERG, David. **SVG essentials**: basic shapes. Sebastopol, 2002. Disponível em: <http://www.oreilly.com/catalog/svgess/chapter/ch03.html>. Acesso em: 13 nov. 2006.

FERNANDES, Antonio Luiz Bogado; BOTINI, Joana. **Construção de algoritmos**. Rio de Janeiro: Senac Nacional, 1998.

FERREIRA, Helder Filipe Patrício Cabral. **Scalable vector graphics**: processamento estruturado de documentos. 2003. 32 p. Relatório (Mestrado em Engenharia Informática) - Faculdade de Engenharia da Universidade do Porto, Porto. Disponível em: <http://lpf-esi.fe.up.pt/~ped/docs/SVG2.pdf>. Acesso em: 10 dez. 2006.

FREITAS, Gilberto. **Protótipo de um sistema de geração e animação de fluxogramas**. 2003. 56 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FROST, Jon; GOESSNER, Stefan; HIRTZLER, Michel. **Learn SVG the web graphics standard**: curves - quadratic bezier spline. Columbia, 2003. Não paginado. Disponível em: <http://learnsvg.com/html/bitmap/chapter04/page04-1.htm>. Acesso em: 13 nov. 2006.

MACHADO, Vinícius Ponte; FURTADO, Elizabeth Sucupira; ALVES, Francisco José de Azevedo. Uma nova tecnologia para a construção dos materiais didáticos utilizados em EAD. In: Congresso Internacional de Educação a Distância, 9, 2002, São Paulo. **Anais...** São Paulo: ABED, 2002. Não paginado. Disponível em: <http://www.abed.org.br/congresso2002/trabalhos/texto47.htm>. Acesso em: 30 mar. 2006.

MEDEIROS, Clavius Leandro; DAZZI, Rudimar Luís Scaranto. Aprendendo algoritmos com auxílio da web. In: Congresso Brasileiro de Computação, 2, 2002, Itajaí. **Anais...** Itajaí: UNIVALI, 2002. Não paginado. Disponível em: <http://intranet.dcc.ufba.br/pastas/materias/MAT146/SobreAlgoritmos.pdf>. Acesso em: 10 abr. 2006.

MANZANO, José Augusto Navarro Garcia. **Revisão e discussão da norma ISO 5807 - 1985 (E)**. São Paulo, 2004. 31 p. Disponível em: <http://www.cantareira.br/thesis/v1n1/navarro.pdf>. Acesso em: 25 out. 2006.

SALIBA, Walter Luiz Caram. **Técnicas de programação**: uma abordagem estruturada. São Paulo: Makron Books, 1992.

SANTIAGO, Rafael de; DAZZI, Rudimar Luís Scaranto. Ferramenta de apoio ao ensino de algoritmos. In: Seminário de Computação, 13, 2004, Blumenau. **Anais eletrônicos...** Blumenau: FURB, 2004. p. 79-86. Disponível em: <http://www.inf.furb.br/seminco/2004/artigos/96-vf.pdf>. Acesso em: 16 abr. 2006.

SCHEPERS, Doug (Org.); QUINT, Antoine (Org.). **SVG WIKI**: SVG and HTML. [S.l.], 2005. Disponível em: <http://wiki.svg.org/SVG_and_HTML>. Acesso em: 27 out. 2006.

SCHMITT, Fátima Aparecida Benthien da Silva**. Protótipo de um ambiente para o ensino de algoritmos**. 1998. 50 f. Monografia (Pós-Graduação) - Curso de Pós-Graduação em Tecnologia de Sistemas, Universidade Regional de Blumenau, Blumenau.

TAGLIARI, Alessandra. **Protótipo de um software para o auxílio ao aprendizado de algoritmos**. 1996. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TRAVERSA, Eddie. **SVG - the art is in the code**: basic shapes. [S.l.], 2001. Disponível em: <http://www.webreference.com/authoring/languages/svg/intro/3.html>. Acesso em: 13 nov. 2006.

VARGAS, Karli Schubert. **Ferramenta para apoio ao ensino de introdução à programação**. 2005. 93 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

W3C. **Scalable Vector Graphics (SVG):** XML graphics for the web, 2006. Disponível em: http://www.w3.org/Graphics/SVG/. Acesso em: 25 out. 2006.