Especificações Formais para Linguagens Visuais de Programação

CLARISSE SIECKENIUS DE SOUZA[1]

DELLER JAMES FERREIRA[1,2]

[1]ICAD
Departamento de Informática — PUC-Rio
Rua Marquês de São Vicente, 225
22451-041 Rio de Janeiro, RJ, Brasil
email: clarisse@icad.puc-rio.br e deller@inf.puc-rio.br

[2]Departamento de Estatística e Informática
Intituto de Matemática e Física — UFGo
Caixa postal 131
74001-970 Goiânia, Go, Brasil

Abstract Este trabalho apresenta o projeto de LvPASCAL, uma linguagem visual de programação baseada em PASCAL. Após uma breve análise da variedade de conceitos associados à *programação visual*, propõe-se que a emergência de reais instâncias deste tipo de código gráfico exige formalização sintática e motivação cognitiva. A formalização sintática é indispensável à compilação de programas e em LvPASCAL ela é feita através de L-SYSTEMS. A motivação cognitiva é indispensável para que um programador possa escrever e entender código nesta linguagem. Em LvPASCAL a motivação das imagens é perceptiva, associando formas geométricas a noções de controle e modularização de programas.

Keywords linguagens de programação, linguagens visuais, L-Systems, estruturas sintáticas, percepção visual

1. Introdução

Exemplos de linguagens visuais de programação (LvP's) relatados em publicações especializadas [Chang'90; Crimi et al.'90; Lakin'87; Smith et al.'82] revelam *linguagens de programação* (LP), para os vários autores, são aquelas mais precisamente classificadas como linguagens de consulta, linguagens de representação e linguagens de comando. O conceito fundamental associado a uma LP — o de ser uma ferramenta para escrever sistemas de computação [Ghezzi & Jazayeri'82], dotada de componentes lexicais, sintáticos e semânticos, que dão conta de abstrações de dados, controle e elementos modulares [Appleby'91] — raramente se explicita nas abordagens até aqui propostas.

Apesar alguns autores tratarem extensivamente da questão das estruturas de dados [Shu'88; Card et al.'91], LvP's que adicionem a este tratamento o de estruturas de controle e de modularidade não têm sido apresentadas. Um dos grandes desafios relativos a controle, em particular, é encontrar uma solução harmoniosa para a sua feição estrutural (controle em tempo de compilação) e a sua feição temporal (controle em tempo de execução). Outro destes desafios, e possivelmente ainda maior que o primeiro, é o de que se projete uma LvP capaz de ser tão geral e expressiva quanto, por exemplo, a linguagem PASCAL [Jensen & Wirth'85]. Somente assim poder-se-ia classificar com legitimidade uma linguagem visual como de PROGRAMAÇÃO.

Este trabalho reporta os primeiros resultados de projeto de LvPASCAL — uma LvP de caráter geral, inspirada em PASCAL. Argumenta-se que a generalidade exigida de uma LP tradicional só é atingível no meio visual através de imagens ABSTRATAS. Conseqüentemente, a EXPRESSIVIDADE da linguagem, julgada a partir do ponto de vista de um usuário-programador, é um ponto crítico tanto comparativamente à sua contraparte textual, quanto a linguagens visuais de comando ou de consulta. Defende-se, aqui, que uma das melhores opções para se garantir interpretabilidade aos *tokens* de uma linguagem visual de programação é fazer com que as imagens a eles associadas tenham MOTIVAÇÃO PERCEPTIVA e que as sentenças visuais a partir deles formadas tenham SINTAXE FORMAL.

Nas seções seguintes a esta, encontram-se: na segunda, uma breve análise sobre estrutura e expressividade de LvP's; na terceira, exemplos de LvPASCAL; e,

finalmente na quarta, uma série de conclusões preliminares sobre oportunidades e limitações de linguagens de programação cujos *tokens* são visuais.

2. Linguagens Visuais de Programação: Estrutura e Expressividade

Uma linguagem de programação se destina essencialmente a possibilitar a codificação e execução de sistemas de computação. Uma LP de propósito geral possui estruturas de dados e controle, e não raro de modularização, formalmente definidas através de 3 componentes básicos: o lexical (palavras reservadas e palavras válidas para a nomeação de entidades de programa), o sintático (conjunto de regras estruturadoras de um programa), e o semântico (conjunto de regras mapeadoras de estruturas sintáticas em estados de uma máquina abstrata equivalente ao programa).

Obviamente existe um destino associado ao de possibilitar a codificação de programas que é o de que as LP's possibilitem a sua manutenção. Conseqüentemente, além de elas deverem ser legíveis por máquina, elas devem ser legíveis por pessoas. Uma característica distintiva entre LP's é justamente o grau de legibilidade dos programas nelas escritos. Linguagens como PASCAL, por exemplo, têm o objetivo explícito de atingirem um bom grau de redigibilidade (facilidade de escrever) e legibilidade (facilidade entender) de programas.

LP's textuais atendem aos requisitos de construção e manutenção de software através de conexões diretas entre os *tokens* da linguagem (itens lexicais) e palavras do inglês (na maioria dos casos). Além disso, freqüentemente, muitas das regras sintáticas são igualmente inspiradas nas estruturas da língua inglesa, facilitando ao programador as inferências corretas sobre a semântica de seu programa.

Uma LP visual, portanto, precisa encontrar imagens com o grau de redigibilidade e legibilidade indispensável à construção e manutenção de software em geral. Uma observação dos exemplos de linguagens visuais bem-sucedidas em computadores, contudo, revela que a facilidade de expressar e de entender idéias através de imagens em tela está geralmente associada a analogias existentes entre a semântica da forma visual e eventos ou objetos cujo conhecimento faz parte do conhecimento de mundo do usuário. Esta associação provê regras de formação de sentenças visuais de natureza muito menos sintática do que heurística. O exemplo mais claro disto é a consagrada linguagem visual do *desktop* da maioria dos computadores pessoais e estações de trabalho. As sentenças visuais bem-formadas são aquelas que obedecem a princípios análogos aos de manipulação física de objetos correspondentes aos ícones da linguagem, antes que a princípios de estruturação gramatical (como os formadores de listas, árvores ou grafos, por exemplo).

Ora, a substituição de sintaxe por heurística é extremamente problemática para qualquer linguagem de propósito geral. A aplicar-se o princípio da analogia, uma LvP com bom grau de redigibilidade e legibilidade demandaria seja um conjunto muito extenso e muito provavelmente heterogêneo de analogias para cada estrutura de dados, controle e modularização de programas, seja uma única e poderosa analogia que desse conta de tudo o que é possível programar.

A não aplicar-se o princípio da analogia, uma LvP utilizável deveria prover: (a) um conjunto de imagens primitivas (e regras de construção de imagens válidas) para fazer as vezes das tradicionais palavras reservadas e rótulos de entidades de programa: (b) um conjunto de regras estruturadoras de imagens em fluxos de controles, dados e módulos de programação compiláveis; e finalmente (c) um conjunto de regras de mapeamento destas últimas imagens estruturadas em estados da máquina abstrata equivalente. Ainda mais importante do que (a), (b) e (c), a LvP só seria utilizável se alguém pudesse escrever programas com ela e subseqüentemente fazer a manutenção dos mesmos.

Parece dispensável argumentar que o meio visual apresenta, para esta finalidade, uma barreira de altíssima complexidade, explicando com facilidade a escassez (senão a inexistência) de legítimas linguagens visuais de programação. Uma revisão das principais propostas até aqui apresentadas, revela o projeto de:

— linguagens visuais de comando

Trata-se de linguagens cujas estruturas têm muito baixa complexidade, não havendo maior dificuldade do que a de articular ação, objeto e parâmetros [Chang'90,Shu'88].

— linguagens visuais de consulta

Trata-se de linguagens com estruturas ligeiramente mais complexas que as anteriores, uma vez que normalmente encontram-se envolvidas expressões booleanas, cuja articulação pode apresentar aninhamentos profundos [Crimi et al'90]. Entretanto, como somente operadores lógicos apresentam desafios estruturais, tais linguagens mostram-se tratáveis visualmente através de gramáticas espaciais [Lakin'87].

— linguagens visuais de representação

Trata-se propriamente de linguagens de output apenas, ou linguagens de visualização, onde a tarefa de

entender sobrepõe-se quase totalmente à de expressar. Nestas linguagens, estruturas quaisquer expressas em uma linguagem outra (e.g. algoritmos, equações, teoremas) são traduzidas em imagens, como alternativa de representação [Shu'88].

Visando projetar uma genuína LvP, dotada de construtores de dados, controles e modularizações necessárias e suficientes para uma programação estruturada, buscamos especificar uma linguagem visual dotada de SINTAXE FORMAL e MOTIVAÇÃO COGNITIVA. A sintaxe formal é necessária para a compilação de programas quaisquer. A motivação cognitiva é necessária para qualquer programador poder usá-la para gerar ou alterar código. Mas, ao contrário do apoio da linguagem natural humana no caso das LP's textuais, não há um código visual tão natural e genérico quanto a língua inglesa para apoiar as LvP's. A alternativa escolhida foi a MOTIVAÇÃO PERCEPTIVA, ou seja: a seleção de imagens estruturadas e abstratas cuja impressão à vista **motive** uma associação com as noções semânticas de programação.

No atual estágio desta pesquisa, somente estruturas de controle e modularização estão sendo tratadas. Estruturas de dados não foram ainda incorporadas ao ambiente. Além disto, as LP's selecionadas para uma versão visual são as do padrão ALGOL. Especificamente, a LP de base para a pesquisa é o PASCAL.

A seguir será apresentado o perfil de LvPASCAL, cuja sintaxe formal está especificada através de L-SYSTEMS [Lindemeyer'87; Prusinkiewicz et al.'88] e cuja motivação perceptiva tem base geométrica, valendo-se de segmentos de reta e polígonos para a geração de imagens estruturadas.

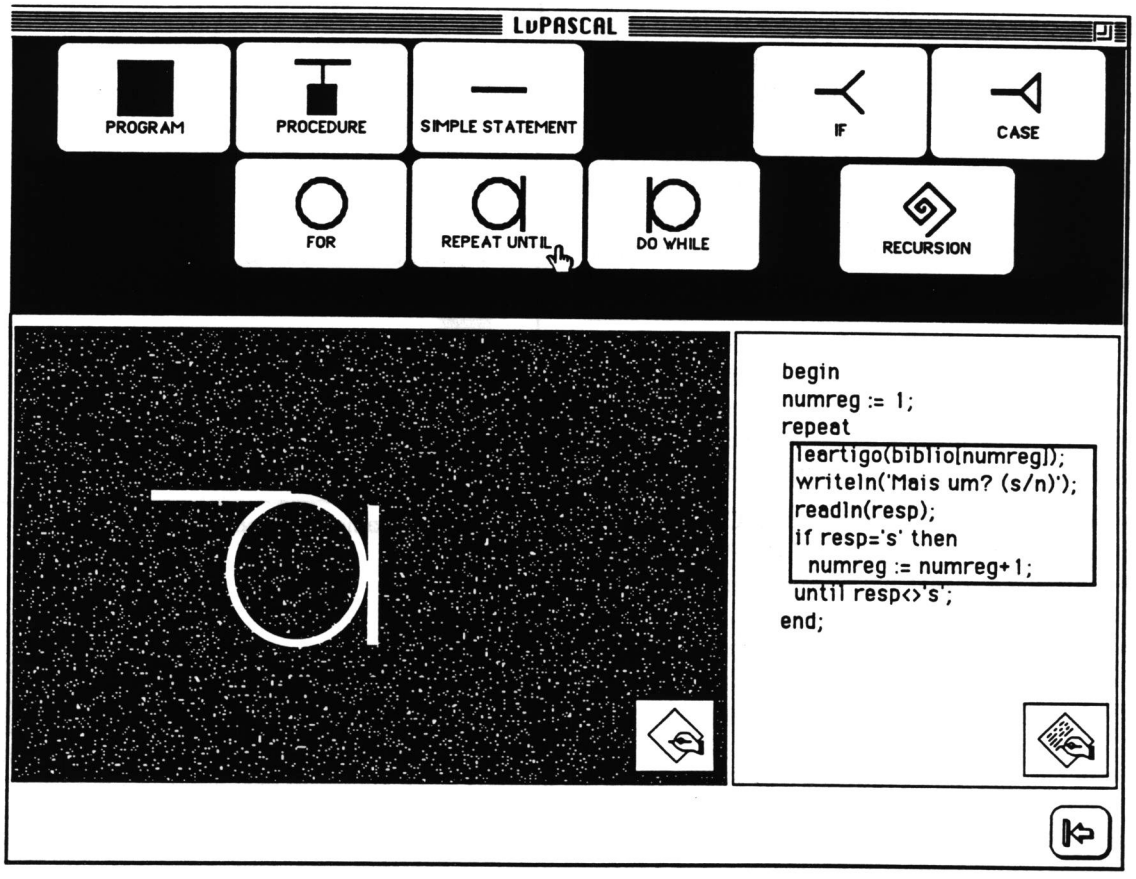

Figura 1: Aspecto da tela de interface de LvPASCAL

3. Um Exemplo de Linguagem Visual de Programação

Dentre as estruturas modulares e de controle existentes em PASCAL, foram selecionadas aquelas que aparecem na Figura 1: 3 diferentes níveis de modularização — `program`, `procedure` e `statement` 3 diferentes estruturas de iteração — `for`, `do while` e `repeat until` — além da recursão; e 2 estruturas de seleção — `if` e `case` Apesar de as imagens que aparecem nos botões de interface serem de fato uma estilização das

imagens realmente geradas pelo programa (como será visto a seguir), elas revelam a motivação perceptiva que associa sua forma geométrica à noção semântica dos controles e abstrações modulares em questão.

Um programa é a unidade máxima, por isto representado como um quadrado (caixa, recipiente que a tudo engloba). Uma instrução simples é uma unidade mínima, por isto representada como um único segmento de reta (primitiva de traçado em LvPASCAL). Já os procedimentos são sub-programas (donde o mesmo quadrado, somente em escala reduzida) e ao mesmo tempo equivalentes a instruções (donde o segmento de reta a que se ligam). As estruturas de seleção binária e eneária são representadas através de uma bifurcação na seqüência de instruções (`if`) ou de uma forquilha de `n` dentes, sendo `n` o número de casos existentes no programa (na Figura 1 vê-se a representação estilizada).

As estruturas iterativas e a recursiva (um caso especial de iteração) são representadas a partir de um círculo (imagem facilmente associada à noção de (`loop`) e de uma espiral poligonal (imagem também facilmente associada a objetos que operam ou desdobram-se a partir de si mesmos) [Hofstadter'79]. De fato, a visualização do círculo (como será visto a seguir) obtém-se por uma generalização da regra sintática geradora de polígonos regulares. Assim, uma iteração contendo `n` instruções é representada como um círculo que contém um polígono regular de `n` lados. O `for` é controlado por uma contagem explícita de iterações; o `do while` é comandado por um teste booleano antes da iteração e o `repeat until` por um teste booleano depois da iteração. Daí os segmentos de reta perpendiculares à seqüência linear de instruções nos últimos dois casos.

Assim como acontece na linguagem PASCAL, a noção de bloco é fundamental na LvPASCAL. Por isto, cada imagem representa apenas um nível de abstração. Para "explodir" as 4 instruções do `repeat until` naquilo que de fato são, o programador passa por sucessivos níveis de detalhamento, nos quais as imagens que aparecem são do tipo das que se vêem na Figura 2.

Evidentemente, para um programa "visual" valer ser escrito (mesmo que aqui só estejamos tratando de parte do problema geral) é preciso que ele possa ser compilado. Ou seja, deve-se poder escrever um *parser* para uma imagem complexa candidata a programa. Para exemplificar, é preciso que dada a imagem na área de desenho da Figura 1, possa-se construir uma estrutura equivalente à do procedimento que aparece escrito na área de texto. Dentro do procedimento (entre begin e end), há uma instrução simples e um `repeat until` (dentro do qual há 4 instruções). Na LvPASCAL, o recurso que permite a existência de um *parser* para imagens complexas são os L-SYSTEMS.

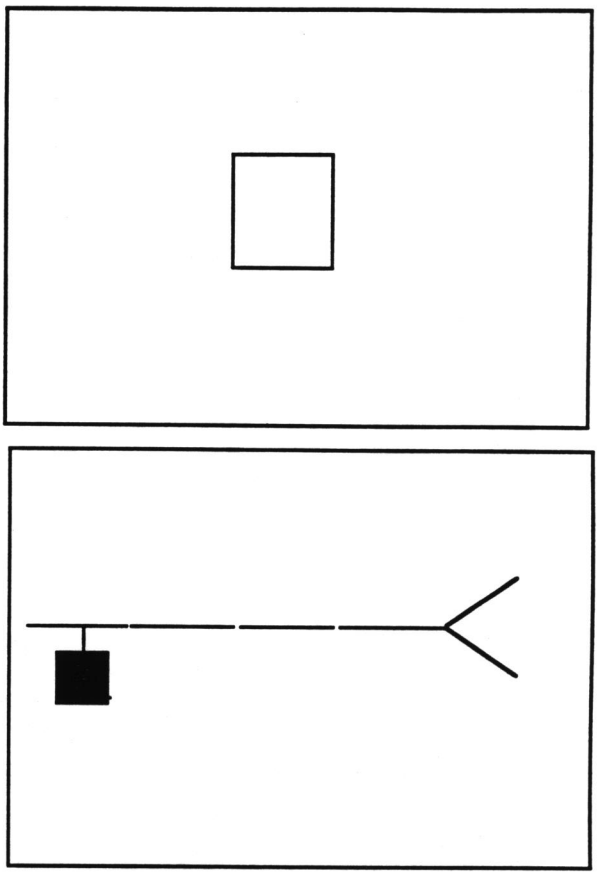

Figura 2: Detalhamentos sucessivos de um `loop`

L-SYSTEMS são sistemas de reescritura do tipo α -> β sem contudo haver diferença entre símbolos terminais e não-terminais [Hopcroft&Ulmann'79]. Um desenhador gera imagens especificadas por este formalismo através da iteração da aplicação das produções gramaticais. O programa por nós utilizado [Bourke'90] oferece ao usuário uma linguagem para a elaboração de gramáticas livres de contexto (0L), sensíveis a contexto (1,1L) e estocásticas. Para os L-Systems mais simples (0L), as produções desta linguagem podem ter o formato sugerido nas regras da Tabela 1.

Imagem ::= Traçado | Traçado, Imagem	
Traçado ::= {Traçado}^{n}	n é número inteiro de iterações
Traçado ::= '[', Traçado, ']' | '{', Traçado, '}'	'[ ]' determinam pushdown e popup das estruturas; '{ }' determinam imagem poligonal
Traçado ::= Rotação, Translação	
Rotação ::= {Mod-Ang},{[ ] | '+' | '-'}, α | 'l'	α é um ângulo de rotação
Translação ::= {Mod-Esc}, f | {Mod-Esc}, F	
Mod-Esc ::= '<', esc | '>', esc | {'<'|'>'}, Mod-Esc	esc é um número real
Mod-Ang ::= '(', ang | ')', ang | {'('|')'}, Mod-Ang	ang é um número real

Tabela 1: Regras para a escrita de produções no sistema de Bourke [Bourke'90]

Um exemplo das descrições de imagens que aparecem na Figura 1 a partir de L-SYSTEMS é o da: imagem do `repeat until`:

axioma:

AA [< B] C) - AA

produções:

A -> F

B -> {-FB}^{360}

C -> f

parâmetros:

$\alpha = 1^o$

incremento angular = 89

incremento escalar = 57.5

Vale retomar algumas das características interessantes de LvPASCAL. Primeiramente, como visto no caso acima, a geração da imagem do círculo, através das produções de L-SYSTEMS, é equivalente à de um polígono regular qualquer, aplicada a seguinte produção:

X -> {+ α F X}^{n}

n = no de lados

$\alpha = 360/n$

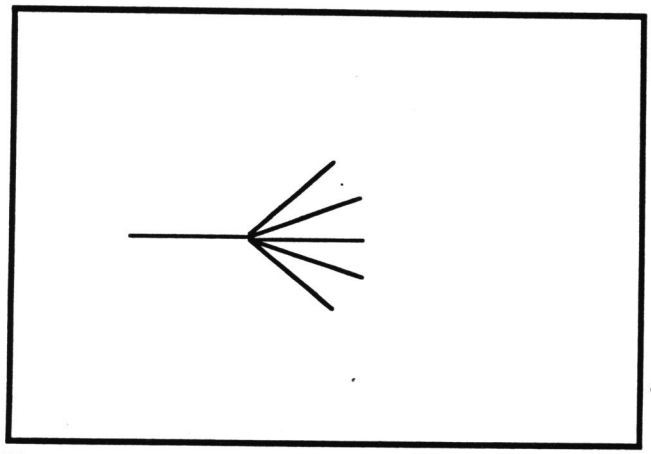

Figura 3: Detalhamento de uma estrutura `case`

Daí, para qualquer estrutura iterativa contendo mais de 2 instruções simples ou estruturadas, pode-se refinar a visualização do *loop* através do traçado do polígono de tantos lados quanto o número de instruções (veja-se Figura 2). O mesmo ocorre com estruturas de seleção — `if` e `case`. A imagem genérica da primeira coincide com a sua imagem específica, já que um dos caminhos equivale ao `then` e outro ao `else`. Já a estrutura `case` pode conter vários caminhos. Assim, a sua visualização refinada pode mostrar quantos são estes caminhos, através de uma produção de imagem que trabalha com sucessivos decrementos angulares. Por exemplo, a visualização de uma estrutura case com 5 casos é a que se vê na Figura 3.

Para ilustrar o funcionamento completo de LvPASCAL, considere-se o programa descrito a seguir. Este programa possibilita a captura de dados para referências bibliográficas e foi compilado e executado numa estação SUN. Interessam a este trabalho somente as estruturas de controle e modularização, podendo-se excluir de consideração todas as estruturas e declarações de dados.

A construção da estrutura do programa através de LvPASCAL sucede-se como sugerido na Figura 1. O programador aciona um dos botões da interface e aparecem no campo de desenho e no campo textual os códigos visuais e computacionais equivalentes ao controle ou módulo em questão. Assim, o programa-exemplo oferece visualizações sucessivas, como apontado na Figura 4. Acrescentamos à visualização de estrutura uma visualização de execução, para ressaltar os efeitos de harmonização entre visualização em tempo de ompilação e visualização em tempo de execução.

4. Conclusões Preliminares

A pesquisa aqui relatada aponta para problemas e soluções bastante estimulantes no campo de LINGUAGENS VISUAIS. Inicialmente, apesar de o uso de LvP's para efetivo desenvolvimento de software ainda

parecer uma realidade distante, o aparente tabu de que as LvP's têm de ter natureza icônica e ser reguladas por princípios de analogia em manipulação direta de objetos parece começar a ser superado. Este trabalho mostra que é possível contemplar regularidades e generalizações [de Souza'93] a partir de imagens abstratas, sem com isto incorrer-se necessariamente em visualizações sem qualquer modivação cognitiva [de Souza'94].

```
program referencias;
const
        sizearq = 100;
type
        string40 = packed array [1..40] of char;
        string10 = packed array [1..10] of char;
        artigo =
                record
                autor: string40;
                titulo: string40;
                revista: string40;
                volume: integer;
                paginas:string10;
                ano: integer;
                end;
        bibliografia = array [1..sizearq] of artigo;
var
        biblio: bibliografia;
        item: artigo;
        numreg: integer;
procedure leartigo (var item:artigo);
begin
        writeln('Autor(es): ');
        readln(item.autor);
        writeln('Titulo: ');
        readln(item.titulo);
        writeln('Periodico: ');
        readln(item.revista);
        writeln('Volume: ');
        readln(item.volume);
        writeln('Pagina(s): ');
        readln(item.paginas);
        writeln('Ano: ');
        readln(item.ano);
end;
procedure montarquivo(var biblio:bibliografia;
                                var numreg:integer);
var     resp: char;
begin
        numreg:=1;
        repeat
                leartigo(biblio[numreg]);
                write('Mais um? (s/n)');
                readln(resp);
                if resp='s' then numreg:=numreg+1;
                until (resp <> 's') or (numreg > sizearq)
end;
procedure mostrartigo (var biblio:bibliografia;
                                var numreg: integer);
var     i: integer;
begin
        for i := 1 to numreg do
          begin
                writeln('Item: ', i);
                writeln('Autor(es): ',biblio[i].autor);
                writeln('Titulo: ',biblio[i].titulo);
                writeln('Periodico: ',biblio[i].revista);
                writeln('Volume: ',biblio[i].volume);
                writeln('Paginas: ',biblio[i].paginas);
                writeln('Ano: ',biblio[i].ano);
                writeln;
          end;
          i := i + 1;
end;
begin   {Programa Principal}
        montarquivo(biblio,numreg);
        mostrartigo(biblio,numreg);
end.
```

A utilização de L-SYSTEMS para formalizar a sintaxe da LvP garante a codificação e a compilação de programas — pelo menos até o nível de estruturas de controle e modularização aqui apresentadas — de maneira bastante análoga ao que se faz em LP's textuais. Esta característica está dando suporte ao projeto de LvPASCAL como efetiva ferramenta gráfica de programação [Ferreira's. J. D.].

Uma vez que as LvP's tornam-se uma realidade mais genérica, as aplicações computacionais do ambiente por elas favorecido se multiplicam. Por exemplo, o processo de programação estruturada dá suporte a aplicações de *Computer-Aided Software Engineering*, na medida em que a correção de programas pode ser visualmente detectada e controlada. Se atentarmos para os recursos de depuração de programa, para ilustração, é possível a um programador visualizar os caminhos de **execução** do programa e identificar pontos de desvio ou interrupção de fluxo indesejados ou imprevistos.

Ainda como aplicação de bastante interesse, pode-se destacar a educativa, ou tutorial. Um ambiente que associa formalmente as estruturas sintáticas e semânticas de um código textual e de um código visual permite ao aprendiz de programação uma percepção muito mais penetrante dos construtos da linguagem do que seria possível mediante a existência de apenas uma das alternativas. Neste particular, PASCAL é uma LP especialmente adequada, tendo em vista que freqüentemente é através dela que se ensinam os fundamentos da programação [Salmon'91].

Todas estas facilidades convergem para um incentivo considerável ao desenvolvimento de linguagens visuais para a área de *end-user programming*. Nela, objetiva-se que o usuário final possa desenvolver rapidamente programas, dentro de um ambiente amigável para o qual em muito contribuem as interfaces gráficas. Entretanto, ao contrário das abordagens como Visual Basic©, por exemplo, em que a real programação gráfica só se pode obter a nível de parâmetros de objetos visualizáveis (botões, ícones, janelas, e outros), LvPASCAL estende esta possibilidade ao próprio código. Em Visual Basic, o código é representado e construído textualmente, restringindo-se os recursos gráficos ao destaque de termos sintaticamente relevantes (através de **highlight** e **identação** para a compilação ou interpretação).

A conclusão corrente mais importante para nós, porém, é a de que a noção de gramáticas espaciais de suporte a visualizações em geral pode transcender as aplicações já existentes em linguagens de comando, de consulta ou de representação. Esta extensão permite que se conceba um paradigma único para o tratamento códigos gráficos em situações tão diversas quanto interfaces de usuário,

visualização de dados, algoritmos, equações ou teoremas, e especificação e codificação de sistemas.

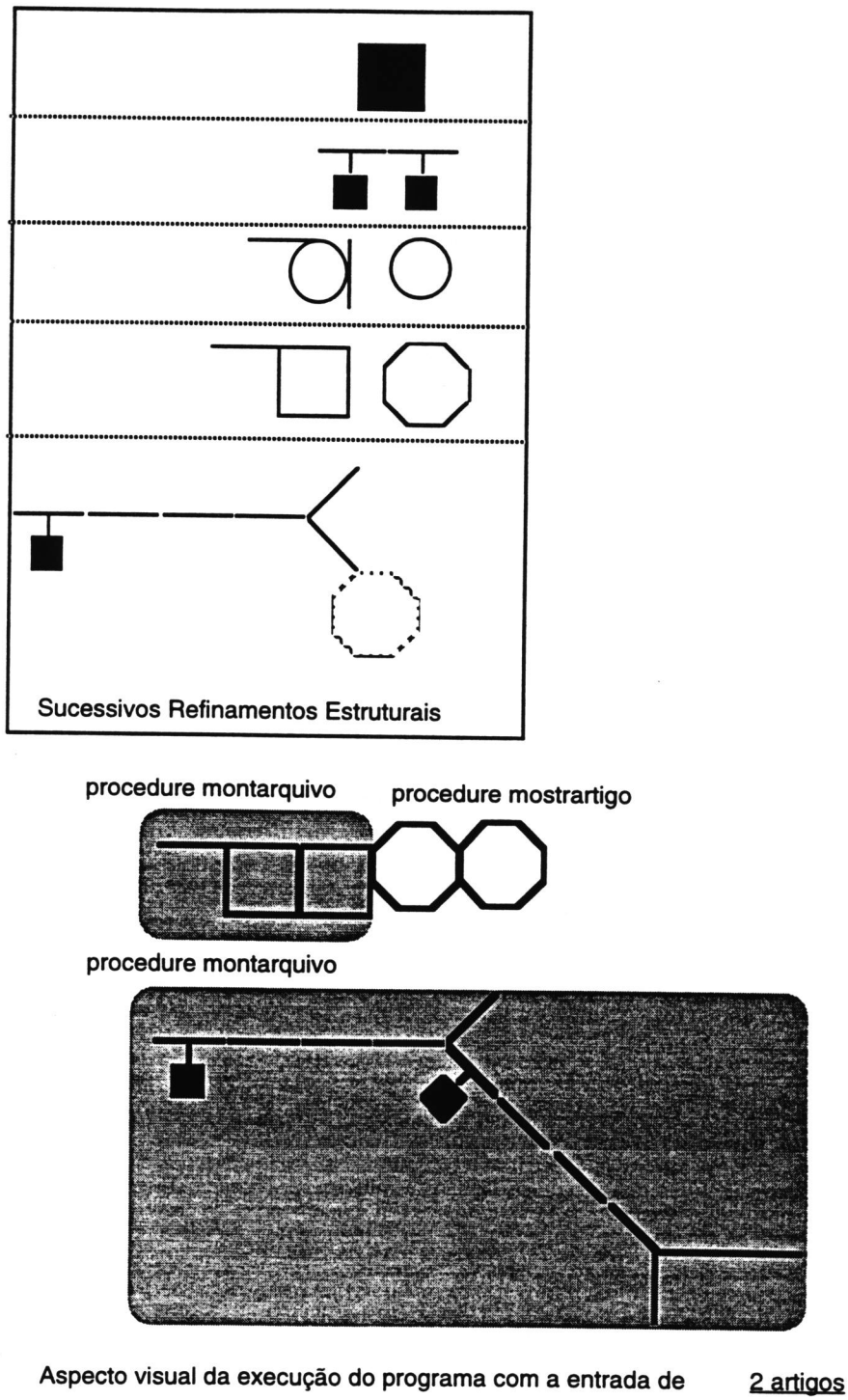

Figura 4: Estrutura e execução visualizadas em LvPASCAL

5. Referências

Appleby'91 — Appleby,D. (1991) *Programming Languages*. New York, NY. McGraw-Hill Inc.

Bourke,P. (1990) *L System*. Program and User Notes, available in Internet via Public FTP.

Card,S.K.; Robertson,G.G; and Mackinlay,J.D.(1991) The The Information Visualizer. Proceedings of CHI'91. ACM Press. pp. 181 — 188

Chang,S.K. (1990) *Visual Languages and Visual Programming*. New York, NY. Plenum Press.

Crimi,C., Guercio,A., Pacini,G., Tortora,G., and Tucci,M. (1990) Automating Visual Language Generation. *IEEE Transactions on Software Engineering*, **16**(10). pp.1122-1135

de Souza,C.S. (1993) Regularidade e Generalização em Interfaces Gráficas. Anais do SIBGRAPI VI. Recife,Pe. pp. 183-191

de Souza,C.S. (1994) Structure and Expressiveness of Visual Computer Languages. Aguardando publicação.

Ghezzi,C. and Jazayeri,M. (1982) Programming Language Concepts. New York. John Wiley.

Hofstadter,D. (1979) Gödel, Escher and Bach: an Eternal Golden Braid. New York. Basic Books.

Hopcroft,J.E. and Ullman,J.D. (1979) *Introduction to Automata Theory, Languages and Computation*. Reading, Ma. Addison-Wesley.

Ferreira, D.J. (s.d.) LvPASCAL — Fundamentos de Projeto para uma Linguagem Visual de Programação Estruturada. Em preparação

Jensen,K. & Wirth,N. (1985) Pascal User Manual and Report. Heidelberg. Springer-Verlag.

Lakin,R. (1987) Visual Grammars for Visual Languages. *Proceedings of AAAI'87*. pp. 683-688

Lindemeyer,A. (1987) Models for Multicellular Development: Characterization, Inference and Complexity of L-systems. In A. Kelmenová and J. Kelmen (eds.): *Trends, Techniques and Problems in Theoretical Computer Science*. Lecture Notes in Computer Science 291, Springer-Verlag, Berlin. pp. 27-40

Prusinkiewicz,P., Lindemeyer,A. and Hanan,J. (1988) Developmental Models of Herbaceous Plants for Computer Imagery Purposes. *Computer Graphics* **22**(4). pp. 141-150

Salmon,W.I. (1991) Structures and Abstractions. Boston,Ma. Richard Irwin, Inc.

Schneiderman'83 — Schneiderman,B. (1983) Direct Manipulation: A Step beyond Programming Languages. *IEEE Computer*, **16** (8). pp. 57-69

Shu,N. (1988) *Visual Programming*. New York, NY. van Nostrand Reinhold.

Smith,D.C.,Irby,C.,Kimball,R.,Verplank,W., and Harslem,E. (1982) Designing the Star User Interface. *Byte* **7**, pp. 242-282