A Linguagem para Especificação de Figuras FIG: Definição, Implementação e Utilização

Aparecido Nilceu Marana[1]
Tomasz Kowaltowski[2]

[1]DEMAC - IGCE - UNESP
Caixa Postal 178
Rua 10, 2527 - Santana
13500-230 Rio Claro, SP, Brasil
nilceu@com001.uesp.ansp.br

[2]DCC - IMECC - UNICAMP
Caixa Postal 1170
Campinas, SP, Brasil
tomasz@dcc.unicamp.br

Abstract. This paper describes FIG, a programming language for the description of figures, presents some aspects of its implementation and exhibits a way to integrate it into LaTeX.

1 Introdução

A linguagem FIG foi definida por J.C.Setubal [Setubal (1987)] e tem por finalidade possibilitar a especificação de figuras bidimensionais de forma não interativa. Sua característica principal é a possibilidade de definição e uso de figuras de maneira análoga a procedimentos, modelando, assim, o desenho sem utilizar estrutura de dados.

Embora sendo de propósito geral, a linguagem FIG pode ser integrada a sistemas de composição gráfica de textos como o TeX [Knuth (1984)] ou LaTeX [Lamport (1986)], tornando-se uma ferramenta útil para especificação de figuras nestes sistemas. O Integrador FIG-LaTeX, apresentado em [Marana (1990)] é um exemplo deste tipo de integração.

Este texto descreve a linguagem FIG (seção 2), apresenta alguns aspectos da sua implementação (seção 3) e mostra uma possível forma de integrá-la ao LaTeX (seção 4).

2 A Linguagem FIG

Nesta seção apresenta-se uma descrição sumária da linguagem FIG e exemplos de especificações de figuras nesta linguagem. Maiores detalhes da linguagem podem ser encontrados em [Setubal (1987), Kowaltowski (1988), Marana (1990)].

2.1 Descrição Sumária

FIG é uma linguagem de propósito geral própria para especificação de figuras bidimensionais, de forma não interativa. Em FIG, as figuras são modeladas de maneira análoga a procedimentos de linguagens como Pascal ou C. Assim, nenhuma estrutura de dados é utilizada. As figuras permitem parametrização flexível e podem retornar valores através de um mecanismo especial. A "instanciação" das figuras pode ser modificada através de transformações geométricas embutidas na linguagem e o posicionamento pode ser feito de três modos: por um movimento implícito associado a "instanciação" consecutiva de diferentes figuras, por posicionamento relativo a outras figuras já "instanciadas" e por coordenadas absolutas.

FIG possui um conjunto de primitivas que constituem a "base" para especificação de quaisquer novas figuras. Estas figuras primitivas encontram-se pré-compiladas em uma biblioteca.

Um programa escrito em FIG constitui-se das seguintes partes, nesta ordem:

1. **Cabeçalho:** O cabeçalho é composto pelo nome do desenho e, opcionalmente, pela definição (em centímetros) da unidade de comprimento a ser utilizada. Se a unidade de comprimento não for explicitamente definida, ela é considerada como sendo 1 cm.

2. **Definição de Figuras:** A definição de uma figura é análoga à declaração de um procedimento na linguagem Pascal, porém, não se permite aninhamentos. Toda figura tem um movimento implícito próprio e é definida num sistema local de coordenadas, cujos eixos são paralelos aos eixos do sistema global. O sistema local é utilizado unicamente para a definição da

figura. Para que uma figura seja efetivamente desenhada, ela precisa ser "instanciada" direta, ou indiretamente, no sistema global, quando, então, sua posição, tamanho e orientação no plano do desenho ficam bem determinados.

A definição de uma figura decompõe-se nas seguintes partes:

- *Nome:* Identificador através do qual a figura é declarada.
- *Parâmetros:* Os parâmetros em FIG são passados apenas por valor e podem ser de qualquer um dos tipos existentes na linguagem. Durante a "instanciação" de uma figura, pode-se omitir alguns parâmetros, desde que estes tenham um valor-padrão, ou que os não omitidos constituam uma das combinações definidas.
- *Combinações:* Uma combinação é um subconjunto do conjunto de parâmetros da figura. Para cada combinação de parâmetros, existe uma especificação alternativa para a figura em questão.
- *Resultados:* Em FIG, toda figura ao ser "instanciada" retorna, automaticamente, alguns pontos estratégicos do menor retângulo que a envolve. Além disso, se uma figura produz outros resultados que também devem ser visíveis fora do seu contexto, estes resultados devem ser atribuídos à variáveis que são declaradas na seção `Results`. Para que os resultados de uma figura possam ser utilizados, a sua "instanciação" precisa ser rotulada.
- *Corpo da Figura:* O corpo da figura deve obedecer as mesmas regras do corpo do programa principal.

Exemplo de definição de uma figura em FIG:

```
Figure exemplo ;
 Parameters{a,b = 1,c} ;
 Combinations{ 1 : a,b ;
               2 : b,c ;
               3 : a,c } ;
 Results{pto1,pto2,area} ;
 Begin
   Case exemplo
    { 1 : { ..... } ;
      2 : { ..... } ;
      3 : { ..... } ; }
 End ;
```

3. **Corpo do Programa Principal:** O corpo do programa é constituído por uma seqüência de comandos separados, necessariamente, por ponto-e-vírgulas. Os comandos da linguagem FIG são os seguintes:

   - := (atribuição a variáveis)
   - :== (atribuição a constantes)
   - `Repeat` *expressão* `times { ` *comandos* `} ;`
   - `For` *variável* `from` *expressão1* `to` *expressão2* `step` *expressão3* `do { ` *comandos* `} ;`
   - `While` *expressão* `do { ` *comandos* `} ;`
   - `If` *expressão* `then { ` *comandos1* `} else { ` *comandos2* `}`
   - `Case` *expressão*
     `{` *rótulo numérico* `: { ` *comandos1* `} ;`
     rótulo numérico `: { ` *comandos2* `} ;`
     ⋮
     rótulo numérico `: { ` *comandosn* `}`
     `}`
   - `Exit` (interrompe a execução dos comandos da figura na qual está inserido.)

As figuras devem ser "instanciadas" nos corpos do programa principal e das próprias figuras, da seguinte forma:
Rótulo Operações Nome-da-Figura Parâmetros Posicionamento

Todas as partes componentes da "instanciação" de uma figura são opcionais, exceto, é óbvio, a que se refere ao nome da figura.

Rótulo: é um identificador sucedido pelo caractere ":". Ele rotula a "instância" da figura de modo a identificá-la para permitir acessos futuros a seus resultados.

Operações: As operações são transformações geométricas embutidas na linguagem, que permitem ao usuário modificar as figuras. As operações existentes em FIG são:

- `Rotate` *ângulo , ponto*: rotaciona a figura no sentido anti-horário em *ângulo* graus sobre o ponto *ponto*. Se *ponto* for omitido, então será utilizado o ponto *[0,0]* como padrão.
- `Scale` *fx , fy , ponto*: amplia/reduz a figura utilizando *fx* e *fy* como fatores de escala para as abscissas e ordenadas, respectivamente. O fator *fy* pode ser omitido; se isto ocorrer, é utilizado o valor de *fx* em

seu lugar. O ponto *ponto* é utilizado como referência para fazer a redução/ampliação. Se for omitido é utilizado o ponto *[0,0]*.

- `Reflect` *ponto1 , ponto2*: espelha a figura em torno da linha definida pelos pontos *ponto1* e *ponto2*.

Nome-da-Figura: Nome com o qual a figura foi definida.

Parâmetros: São expressões com as quais uma figura é instanciada. Tais expressões são denominadas parâmetros efetivos e são associadas aos parâmetros formais especificados na declaração da figura. A associação entre paraâmetros efetivos e formais pode ser efetuada de duas formas: usando sintaxe posicional ou não posicional.

Posicionamento: Diz respeito ao posicionamento da figura no plano do desenho - equivale à transformação de translação de figuras. Em FIG existe posicionamento absoluto ou relativo, implícito ou explícito. Para efetuar o posicionamento implícito, o compilador utiliza os pontos de concatenação de figuras que são automaticamente determinados. Estes pontos dependem da direção corrente de especificação das figuras (north, south, east ou west). O valor da direção corrente é mantido na variável *cd*.
Se durante a "instanciação" de uma figura o posicionamento for omitido, então o compilador utilizará o mecanismo de posicionamento implícito de figuras, ou seja, o ponto de concatenação da figura coincidirá com o ponto determinado pelas coordenadas correntes *cc*. As variáveis *cd* e *cc* são pré-definidas e seus valores são atualizados automaticamente durante a execução do programa.
Se o posicionamento não for omitido, deve ter a seguinte sintaxe:
`with` *res* `at` *ponto* .

O identificador *res* deve ser, necessariamente, um resultado implícito ou explícito do tipo ponto da figura que está sendo "instanciada", enquanto que *ponto* pode ser um ponto qualquer do plano.

A omissão do posicionamento pode ser parcial. Quando "`with` *.res*" ou "`at` *ponto*" forem omitidos, não simultaneamente,o compilador utilizará "`with .center`[1]" ou "`at cc`", respectivamente, em seus lugares.

Se *ponto* for um valor absoluto, então tem-se posicionamento absoluto. Se for um ponto resultante da "instanciação" de outra figura, tem-se posicionamento relativo.

2.2 Exemplos de Especificações de Figuras em FIG

Nesta seção são apresentados dois exemplos de especificações de figuras utilizando-se a linguagem FIG:

Exemplo 1: Programa para construir recursivamente uma árvore binária completa de altura 3.

A possibilidade de especificar figuras recursivamente parece ser a melhor e mais poderosa característica da linguagem FIG.

Veja na Figura 1 a figura produzida pelo programa descrito a seguir.

Observe que, para se especificar uma árvore binária de altura 10, por exemplo, bastaria "instanciar" a figura `Sub_Arvore` passando-se 10 como parâmetro.

```
Picture Figura1 ;

Figure Sub_Arvore ;
 Parameters{ h = 0 } ;
 Results{ p1 , p2 , p3 } ;
 Begin
  If h = 0
   Then { r : Circle 0.75 ;
          p1 := r.lb ;
          p2 := r.tc ;
          p3 := r.rb }
   Else { tree1 : Sub_Arvore h-1 ;
          tree2 : Sub_Arvore h-1 With .p1
                    At tree1.p3 + [1,0] ;
          raiz  : Circle 0.75
                    At [tree1.p3.x + 0.5,
                        tree1.p2.y + 1] ;
          Arrow raiz.lc ,
                [tree1.p2.x,raiz.lc.y] ,
                 tree1.p2 ;
          Arrow raiz.rc ,
                [tree2.p2.x,raiz.rc.y] ,
                 tree2.p2 ;
          p1 := tree1.p1 ;
          p2 := raiz.tc  ;
          p3 := tree2.p3  }
  End ;
Begin ! corpo do programa principal
 !$B L
 scale 0.5 Sub_Arvore 3 at [10.75,10] ;
 !$E
End Picture
```

[1]Embora a utilização do ponto de concatenação da figura seja mais coerente, já que este ponto é utilizado nos casos em que há omissão total de posicionamento, na prática, a utilização do ponto `center` demonstrou ser bastante útil.

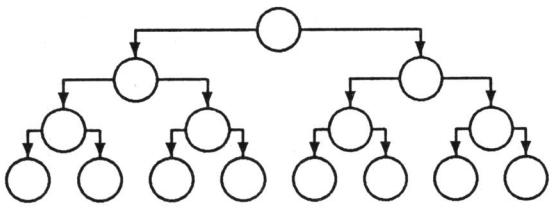

Figura 1: Árvore binária completa de altura 3.

Exemplo 2: Programa para construir a figura exibida na Figura 2.

```
Picture Figura2 ;
 Figure Conjunto ;
   Parameters{larg = 2,alt = 1} ;
   Results{p1,p2,p3,p4} ;
   Begin
     cd := south ;
     r1 : Rectangle larg , alt ;
     Arrow ;
     c1 : Circle alt / 2  ;
     Arrow ;
     r2 : Rectangle lrg , alt ;
     Arrow ;
     c2 : Circle alt / 2 ;
     Arrow ;
     r3 : Rectangle larg , alt ;
     Text ".fig" At r1.center ;
     Text "FIG"  At c1.center ;
     Text ".pas" At r2.center ;
     Text "Pascal" At c2.center ;
     Text ".ps" At r3.center ;
     p1 := r1.rb ;  p2 := r2.rt ;
     p3 := r2.rb ;  p4 := r3.rt
   End ;

 Figure Niveis ;
   Begin
     vh := 4 ; vl := 3 ;
     conj : Conjunto vh,vl At [10.75,10] ;
     l := dist(conj.lt,conj.rt) + vl ;
     h := dist(conj.p1,conj.p4) -  1 ;
     Rectangle l , h At conj.center ;
     Rectangle l - 0.3 , h - 0.3
               At conj.center ;
     Latext "\thinlines" ;
     Line conj.p1 , conj.p1 + [3,0] ,
          conj.p2 + [3,0] , conj.p2 ;
     Line conj.p3 , conj.p3 + [3,0] ,
          conj.p4 + [3,0] , conj.p4 ;
     Text "Nivel 1" At  conj.p2 +
         [6,dist(conj.p2,conj.p1) / 2] ;
     Text "Nivel 2" At  conj.p4 +
         [6,dist(conj.p3,conj.p4) / 2] ;
     Latext "\thicklines" ;
   End ;

Begin ! corpo do programa principal
  !$B Fig2sib L
  scale 0.3 niveis with. lb at [1,1] ;
  !$E
End Picture
```

3 Aspectos da Implementação da Linguagem FIG

Esta seção visa descrever alguns aspectos da primeira versão do compilador que foi desenvolvido para implementar a linguagem FIG. Nesta versão, o compilador traduz um programa escrito em FIG para um código-objeto escrito na linguagem Pascal. A execução deste programa Pascal gera arquivos contendo comandos de LaTeX, ou de PostScript [Adobe (1986), Reid (1988)], dependendo da opção do programador. A execução destes comandos, por sua vez, produz as figuras inicialmente especificadas em FIG. A opção é indicada ao compilador, através das diretivas de compilação, `!$B L` e `!$B P`, para LaTeX e PostScript, respectivamente.

Esta primeira versão do compilador FIG foi desenvolvida em Pascal, através do compilador Turbo Pascal 5.0 [Borland (1987)].

3.1 Independência de Pacote Gráfico

Na Figura 2 pode-se visualizar os passos da tradução de um programa-fonte escrito em FIG para rotinas/comandos de um determinado pacote gráfico. Com o intuito de deixar a linguagem mais flexível, independente de um pacote gráfico específico, a tradução é feita em dois níveis. No primeiro, o programa FIG é traduzido para um programa escrito em uma linguagem de alto nível – no caso, Pascal. No segundo nível, este programa é convertido em chamadas a procedimentos do particular pacote gráfico. Desse modo, quando se desejar utilizar outro pacote, basta alterar este segundo nível de tradução, o que é relativamente simples, já que isto implica apenas em redefinir as figuras primitivas da linguagem FIG que se encontram numa biblioteca. Todas as dependências a um determinado pacote gráfico estão encapsuladas no conjunto das primitivas.

3.2 Passos de Compilação

O compilador FIG gera, como código objeto, um programa escrito na linguagem de alto nível Pascal. Em Pascal, as constantes e variáveis são declaradas explicitamente no início de cada bloco. Já em FIG, as constantes podem ser declaradas em qualquer parte do programa (através da utilização do símbolo de atribuição a constantes `:==`) e os tipos das variáveis

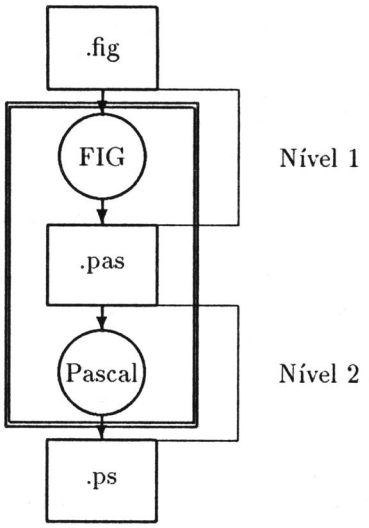

Figura 2: Níveis de tradução de um programa FIG.

são determinados analisando-se o contexto no qual aparecem pela primeira vez. Logo, não é possível entremear as fases de análise e de geração de código durante a compilação de um programa FIG, fato este que caracteriza compiladores de um único passo. Portanto, foi necessário desenvolver um compilador de dois passos.

No primeiro passo da compilação é feita a análise do programa FIG e gerada uma representação interna deste programa. Por tratar-se de uma técnica simples e eficiente, optou-se pela análise sintática descendente recursiva para a implementação do compilador [Aho et al.(1986), Kowaltowski (1983)].

Quanto à representação interna, ela constitui-se numa "árvore do programa". Desse modo, ao analisar o comando condicional

`IF` *expressão* `THEN {` *comandos1* `}`
`ELSE {` *comandos2* `}`

são armazenados: uma marca associada ao símbolo terminal `IF`, a *expressão*, os *comandos1* e os *comandos2*, enquanto que os símbolos terminais: `{`, `}`, `THEN` e `ELSE` são desprezados. Evidentemente, ao serem armazenados os comandos e as expressões envolvidas, as mesmas regras são aplicadas.

No segundo passo é gerado, a partir da árvore, o código-objeto em Pascal.

3.3 Tabela de Símbolos

A tabela de símbolos é tratada como uma pilha que contém símbolos de diferentes categorias: variáveis, constantes, figuras, resultados e parâmetros. Inicialmente, ela é composta apenas pelas variáveis, constantes e figuras pré-definidas. Com o decorrer da análise, os identificadores definidos localmente nas figuras são colocados temporariamente na tabela, enquanto estas estiverem sendo analisadas. Ao ser analisado o corpo do programa principal, a tabela de símbolos conterá, além dos símbolos inicialmente inseridos, apenas os identificadores das figuras definidas pelo usuário e as variáveis e constantes "declaradas" nesta região, visto que apenas estes identificadores são visíveis no corpo do programa principal.

3.4 Tratamento de Erros

O tratamento de erros é bastante simples: ao detectar um erro léxico, sintático ou semântico, o compilador interrompe a análise e envia uma mensagem para o usuário informando qual foi o erro encontrado e a linha em que ocorreu.

3.5 Código Objeto

O código objeto é gerado pelo compilador FIG na linguagem Pascal. Cada figura definida em FIG é traduzida para esta linguagem em um procedimento com o mesmo nome. Além dos procedimentos referentes às figuras, o programa Pascal gerado é constituído, em geral, de chamadas a procedimentos pré-definidos que se encontram compilados numa unidade a parte (biblioteca). Tais procedimentos são responsáveis, entre outras coisas, pelas transformações geométricas que podem ser aplicadas às figuras e pela atualização dos seus respectivos retângulos envolventes.

O projeto da linguagem FIG baseou-se em alguns aspectos no projeto da linguagem Pascal, principalmente no que tange aos comandos: condicional, de atribuição e de repetição. Dessa forma, o código gerado para tais comandos é quase que uma cópia dos mesmos.

3.6 O Posicionamento das Figuras

O aspecto mais problemático desta implementação diz respeito ao posicionamento das figuras. Suponha o seguinte comando de "instanciação": `xyz with .center at cc`. Para posicionar a figura `xyz` com o centro do seu retângulo envolvente no ponto `cc`, é preciso antes conhecer as dimensões de `xyz`. Como não é utilizada nenhuma estrutura de dados, a única forma de determinar as dimensões da figura `xyz` é "instanciá-la". Isto acarreta em dupla "instanciação" da figura. Na primeira "instância" (virtual) não há chamadas a rotinas do pacote gráfico, ou seja, a figura não é efetivamente construída; a "instância" virtual serve apenas para determinar as dimensões de uma figura.

Com estas informações, torna-se possível o cálculo dos fatores de translação (Δx,Δy) necessários para posicionar a figura `xyz` com o centro de seu retângulo envolvente no ponto `cc`, basta para isso "concatenar" tais fatores na matriz de transformação com a qual ela será novamente "instanciada". Nesta segunda "instância" (real) pode haver ou não chamadas às rotinas do pacote gráfico, dependendo do fator de visibilidade do local em que foi efetuada a "instanciação".

3.7 As Primitivas da Linguagem FIG

As primitivas da linguagem FIG são figuras pré-definidas e pré-compiladas que ficam armazenadas na biblioteca do sistema. Na definição de tais figuras encontram-se as chamadas às rotinas do pacote gráfico responsáveis pela efetiva construção dos desenhos. As primitivas da linguagem constituem-se num conjunto básico sobre o qual toda figura é definida.

Nesta versão do compilador FIG, implementou-se a maioria das primitivas previstas no projeto original da linguagem. Para cada primitiva existem duas versões: uma utilizando os recursos gráficos do LaTeX e outra os recursos gráficos do PostScript.

É interessante observar que é perfeitamente possível a inserção de novas primitivas na biblioteca, aliás, recomenda-se fortemente esta prática, já que ela pode melhorar sobremaneira o tempo de compilação/execução de um programa, principalmente se a referida figura for freqüentemente utilizada no programa. As primitivas implementadas são:

- Circle
- Ellipse
- Line
- Rectangle
- Polygon
- Arrow
- Arc
- Text

4 O Integrador FIG-LaTeX

O Integrador FIG-LaTeX, apresentado em [Marana (1990)], provê uma forma de utilização da linguagem FIG, através de sua integração ao sistema LaTeX. Neste integrador, quando o usuário de LaTeX, ao compor seu documento, precisa especificar figuras, ele "instancia" ambientes `\begin{FIG}` .... `\end{FIG}` para especificá-las utilizando a linguagem FIG. Dessa forma, o arquivo criado contém comandos LaTeX e código FIG. Este arquivo é então submetido a um pré-processamento que separa o código FIG do resto do texto. Este código é compilado e executado, gerando arquivos contendo as especificações das figuras em LaTeX ou em PostScript. Posteriormente, estes arquivos são reinseridos no texto, com o auxílio do comando `\special`.

A opção PostScript é útil quando o dispositivo no qual será impresso o documento hospeda um interpretador para esta linguagem. A versão de PostScript é mais abrangente que a de LaTeX. Isto decorre das limitações dos recursos gráficos suportados por LaTeX.

Veja a seguir o aspecto geral que os arquivos preparados para serem submetidos ao Integrador FIG-LaTeX devem apresentar. Tais arquivos devem ter extensão `.FLT` (abreviação para FIG-LaTeX).

```
\documentstyle[...]{...}
        ⋮
   preâmbulo
        ⋮
\begin{document}
        ⋮
   texto LaTeX
        ⋮
  \begin{FIG}
        ⋮
    texto FIG
        ⋮
  \end{FIG}
        ⋮
   texto LaTeX
        ⋮
\begin{FIG}[p]
        ⋮
    texto FIG
        ⋮
  \end{FIG}
        ⋮
   texto LaTeX
        ⋮
\end{document}
```

Cabe observar que uma "instância" do ambiente FIG pode conter definições de novas figuras ou, exclusivamente, comandos da linguagem que não se constituem em definições de figuras, mas sim em comandos destinados à construção efetiva de desenhos. Neste último caso, o usuário pode, opcionalmente, indicar, através dos argumentos `P` ou `L`, se

os recursos gráficos a serem utilizados na tradução do desenho são os de PostScript ou os de LaTeX, respectivamente. Se estes argumentos forem omitidos, então utiliza-se o argumento padrão especificado pelo usuário quando este executa o Integrador FIG-LaTeX. Por exemplo, `FiG-LaTeX Arq.FLT P` indica que a opção-padrão é PostScript.

5 Considerações Finais

O projeto da linguagem FIG, bem como a proposição de integração a sistemas de composição gráfica foram baseados, em parte, nas linguagens PIC [Kernighan (1982)] e IDEAL [Van Wyk (1980)] e em suas respectivas integrações ao sistema TROFF [Ossanna (1979)]. Nestas integrações, ambas as linguagens funcionam como pré-processadores para TROFF.

A Linguagem FIG é mais uma opção para a construção de figuras bidimensionais. Esta opção vem acompanhada de algumas vantagens, tais como: utilização de equipamentos simples para processamento das figuras e facilidade na especificação de figuras com estruturas hierárquicas complexas.

O fato da linguagem FIG ser muito parecida sintaticamente com a linguagem Pascal contribui para sua assimilação, visto que Pascal é uma linguagem bastante difundida.

Em FIG as figuras podem ser traduzidas também para PostScript. Isto torna-a interessante, uma vez que PostScript está se tornando um padrão para descrição de páginas e, como conseqüência, seu interpretador está sendo incorporado aos dispositivos de saída.

Alguns projetos têm sido realizados visando facilitar a inserção de figuras em documentos a serem compostos em ambiente TeX. Micah Beck, por exemplo, criou um pacote, ao qual denominou TransFig [Beck (1989), (1990)], que torna possível tal inserção. O usuário utiliza um editor gráfico para especificar suas figuras que são convertidas em código Fig[2]. Este código é traduzido, segundo opção do usuário, em comandos PostScript, LaTeX, PiCTeX, PIC ou (E)EPIC, que, finalmente, podem ser convenientemente agregados ao resto do documento.

TransFig é nitidamente mais abrangente e poderoso do que o projeto apresentado neste texto. Deve-se ressaltar, no entanto, que as filosofias dos projetos são distintas e que existem aplicações em que a linguagem FIG apresenta melhores resultados. A especificação de uma árvore binária completa é um exemplo de aplicação em que FIG leva vantagem sobre Transfig. A especificação de tal figura, via FIG, pode ser feita para uma árvore genérica (com altura n), fazendo uso de sua característica recursiva. Posteriormente, esta especificação pode ser facilmente alterada. Em Transfig, a construção desta mesma figura deve ser feita passo a passo, através de um editor gráfico, duplicando subpartes, quando possível; futuras alterações poderiam implicar na reconstrução completa da figura.

[2]Tanto o editor, quanto o código Fig, foram originalmente desenvolvidos por Supoj Sutanthavibul, na Universidade do Texas, tendo em comum com a linguagem FIG, desenvolvida por J. C. Setubal, apenas o nome.

6 Referências

Adobe Systems, *PostScript Language Reference Manual*, Addison-Wesley, 1986.

Adobe Systems, *PostScript Language Tutorial and Cookbook*, Addison-Wesley, 1986.

A. V. Aho, R. Sethi and J. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, 1986.

M. Beck, TransFig: Portable Figures for TeX, *Cornell University, Dept. of Computer Science, Technical Report*, #89 - 967, 1989.

M. Beck and A. Siegel, TransFig: Portable Graphics for TeX, TUGboat *The Communications of the TeX Users Group*, vol. 11, no.3 (Proceedings of the 1990 Annual Meeting), 1990.

Borland Inc., *Turbo Pascal Owner's Handbook*, Borland Inc., 1987.

B. W. Kernighan, PIC - A Language for Typesetting Graphics, *Software - Practice and Experience*, vol. 12, no. 1, pp. 1-21, 1982.

D. E. Knuth, *The TeXbook*, Addison-Wesley, 1984.

T. Kowaltowski e J. C. Setubal, FIG: Uma Linguagem para Especificação de Figuras, *ANAIS do VIII Congresso da Sociedade Brasileira de Computação-XV Semish*, 1988.

T. Kowaltowski, *Implementação de Linguagens de Programação*, Guanabara Dois, 1983.

L. Lamport, *LaTeX: A Document Preparation System*, Addison-Wesley, 1986.

A. N. Marana, Integrador FIG-LaTeX, *Tese de Mestrado, DCC - IMECC- UNICAMP*, 1990.

J. F. Ossanna, NROFF/TROFF User's Manual, *Computing Science Technical Report*, no.54, AT&T Bell Laboratories, 1979.

Glenn C. Reid, *PostScript Language Program Design*, Addison-Wesley, 1988.

J. C. Setubal, FIG: Uma Linguagem para Especificação de Figuras *Tese de Mestrado, DCC - IMECC - UNICAMP*, 1987.

C. J. Van Wyk, A Language for Typesetting Graphics, *Standford Department of Computer Science, Report no.STAN-CS-80-803*, 1980.