Raquel Oliveira Prates

Visual LED: uma ferramenta interativa para geração de interfaces gráficas

Dissertação apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para a obtenção do título de Mestre em Informática: Ciência da Computação.

Orientador: Marcelo Gattass
Co-orientador: Luiz Henrique de Figueiredo

TeCGraf

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 15 de agosto de 1994

Aos meus pais, Antonio Augusto e Maria Helena,
e aos meus irmãos, Henrique e Marcos.

Agradecimentos

À minha família, pelo carinho e apoio que sempre me deu e pelo incentivo que sempre deu aos meus estudos.

Ao Luiz Henrique de Figueiredo, pela amizade, pelo incentivo e pela sua orientação constante, dedicada e participativa, que foi fundamental para que eu chegasse ao fim deste trabalho.

Ao Professor Marcelo Gattass, por ter sugerido o trabalho, por sua confiança em mim e pela sua orientação.

Ao Carlos Henrique Levy, por todo apoio que me deu e por sua participação ativa neste trabalho, estando sempre presente para dar suporte na utilização do sistema IUP/LED, para discutir idéias e dar sugestões. Agradeço também por ter implementado comigo a solução da árvore no TeCDraw.

Aos Professores Virgílio A. F. Almeida e Osvaldo S. F de Carvalho da UFMG, por terem me incentivado e apoiado na decisão de vir para a PUC-Rio fazer o mestrado.

A Marc Brown (SRC-DEC), Jim Meehan (Adobe) e Jorge Stolfi (Unicamp), pela sua boa vontade e por terem me mandado material sobre FormsVBT.

Aos colegas e amigos do TeCGraf, por terem sempre proporcionado um ambiente de trabalho amigo e cooperativo, sempre participando com idéias e sugestões no meu trabalho.

Aos meus amigos, por terem me apoiado nos momentos difíceis e por sempre terem estado ao meu lado me dando força e me incentivando. Em especial, à "quadrilha": Andréia Diniz, Anna Hester, Arlindo Cardarett, Beatriz Castier, Carlos Eduardo Kubrusly (Turco), Eduardo Nobre (Boi), Eduardo Thadeu Corseuil, Ivan Menezes, João Luiz Campos (Açaí), Joaquim B. C. Neto, Luiz Gil Solon, Marcos Euclides Viana e Paulo Rodacki; e também a Mônica F. da Costa, Hélio Magalhães e Wanda Teixeira, pela colaboração, pela amizade, pela união espiritual, pelo apoio e pela companhia, tanto na hora do "crime", quanto na hora do trabalho. Ao Turco, agradeço ainda a ajuda com a implementação dos exemplos Motif e o apoio e companheirismo "viradouro" nos momentos de maior tensão.

A Magda e Rodrigo Prates e ao Pensionato Santa Rosa de Lima, em especial à Irmã Stella, Ana Cláudia, Flávia e Bettysa, por terem proporcionado não só uma moradia, mas também um ambiente de família e amizade.

Ao CNPq e ao convênio PUC–PETROBRÁS, pelo auxílio financeiro.

Resumo

Descrevemos e analisamos o projeto e o desenvolvimento de Visual LED, um editor gráfico de interfaces LED. O Visual LED apresenta ao usuário três vistas: a vista textual, a vista gráfica e a vista do resultado. As vistas textual e gráfica são representações editáveis do mesmo diálogo e estão sempre consistentes entre si. A vista textual apresenta a interface em LED; a vista gráfica apresenta uma representação geométrica do *layout* abstrato do diálogo, onde o usuário manipula diretamente seus elementos. O usuário pode editar em qualquer uma das duas vistas a qualquer momento, sem que seja necessário explicitar a transferência de uma para outra. A vista do resultado apresenta o *look-and-feel* final do diálogo.

Para criar o Visual LED, estudamos a interação com o usuário em outros editores gráficos, tanto genéricos quanto de interface. Neste texto, discutimos estes editores gráficos e os comparamos, tanto entre si, quanto com o Visual LED.

Durante o estudo dos editores gráficos, identificamos alguns problemas de interação com o usuário e apresentamos soluções para estes problemas. O principal problema identificado foi a impossibilidade de se editar a hierarquia de um grupo. Para este problema, propomos a solução da árvore, para os editores gráficos genéricos e os de interfaces de *layout* concreto, e a política hierárquica, para os editores gráficos de interfaces de *layout* abstrato.

Abstract

We described and analyzed the design and development of LED interface graphical editor: Visual LED. Visual LED presents to the user three views: a textual view, a graphical view and a resulting view. The textual and graphical views are editable representations of the same dialog and they are always consistent with each other. The textual view presents the abstract layout of the dialog, in which the user may direct manipulate the elements of the dialog. The user can edit in any of the two views, and is able to switch from one another at any moment without requesting an explicit switch. The resulting view presents the look-and-feel of the dialog.

To create Visual LED, we studied the interaction with others generic graphical editors and interfaces graphical editors. In the text we discuss these graphical editors and we compare them among themselves and with Visual LED.

During the study of the graphical editors, we identified some interaction problems and we present some solutions for them. The major problem identified was the impossibility of editing the hierarchy of a group. For this problem we suggest the solution of the tree, for generic graphical editors and concrete layout interfaces graphical editors, and the hierarchical politics for the abstract layout interfaces graphical editors.

Índice

Lista de Figuras

Capítulo 1

Introdução

Um sistema pode ser separado em duas partes fundamentais: a interface com usuário e a tecnologia da aplicação. A interface é responsável pela comunicação entre o usuário e a aplicação, enquanto que a tecnologia da aplicação é a parte que resolve o problema proposto; a princípio, a tecnologia independe da interface [FGIF94].

Muitas vezes, pode ser necessário modificar a interface da aplicação sem que seja necessário alterar a sua tecnologia. A recíproca também é verdadeira: pode-se alterar a tecnologia da aplicação sem precisar alterar a sua interface. A tecnologia da aplicação pode ser alterada de duas formas conceitualmente distintas: pode-se alterar a implementação das funções ou a sua funcionalidade. A alteração da implementação é, geralmente, transparente para o usuário, e não causa alterações na interface. Por outro lado, alterações na funcionalidade têm um impacto maior e, normalmente, implicam em alterações na interface.

Embora a interface não faça parte do problema a ser resolvido, ela é parte fundamental de qualquer solução, pois é a ligação entre o usuário e a aplicação. Uma interface mal projetada pode dificultar o entendimento e o uso da aplicação, fazendo, assim, com que o usuário tenha que despender tempo e energia na utilização do sistema, ao invés de concentrá-los na tarefa a ser executada. Isto pode causar frustração no usuário e, até mesmo, levá-lo a abandonar o sistema. Assim, uma boa tecnologia pode deixar de ser útil se a interface do programa que a contém não for boa [NM90].

1.1 Motivação

O TeCGraf (Grupo de Tecnologia em Computação Gráfica da PUC-Rio) desenvolve, principalmente, aplicações gráficas que requerem grande interação com o usuário. Além disso, freqüentemente é necessário que estas aplicações sejam executadas em diversas plataformas.

Buscando simplificar a criação de boas interfaces portáteis, o TeCGraf desenvolveu o IUP/LED, um sistema portátil de interface com o usuário, composto por um *toolkit* virtual (IUP) e por uma linguagem de especificação de diálogos (LED) [Lev93, FGL93].

O sistema IUP/LED tem sido utilizado em produção pelo TeCGraf com sucesso. Os elementos de interface oferecidos são suficientes, permitindo que boas interfaces sejam rapidamente projetadas. Para criar estas interfaces, o programador deve aprender LED, que é uma linguagem simples e concisa, e portanto de fácil aprendizado. No entanto, novas dificuldades foram encontradas: ao projetar uma interface, deve-se primeiro visualizá-la mentalmente e depois traduzir esta visão em comandos LED, o que pode ser uma tarefa bastante trabalhosa. Reciprocamente, ler uma descrição LED de interface e visualizá-la pode ser igualmente difícil, principalmente se o diálogo sendo descrito é grande ou se a interface é composta por mais de um diálogo, o que é freqüente.

É importante observar que o problema de transformar uma imagem mental em comandos de uma linguagem e vice-versa, existe independente do sistema adotado; este problema não é específico do sistema IUP/LED. (Veja a seção 3.7 para uma discussão mais ampla sobre as limitações de LED.)

1.2 Objetivo

Esta dissertação tem como objetivo descrever e analisar o projeto e o desenvolvimento de Visual LED, uma ferramenta interativa para geração de interfaces em LED.

O Visual LED foi desenvolvido com os seguintes objetivos:

- facilitar a geração de descrições em LED para interfaces com usuário;
- permitir a visualização da interface e do seu *layout* abstrato durante o processo de construção da mesma;
- combinar diferentes métodos de edição (descrição textual e manipulação direta de elementos) para tornar mais eficiente a criação de interfaces;
- fornecer uma interação simples e intuitiva ao usuário.

O Visual LED é um editor gráfico de três vistas, que segue o mesmo modelo de *layout* abstrato adotado por LED. A primeira vista é a vista textual, que apresenta a descrição LED do diálogo e é editável. A segunda vista é a vista gráfica, na qual o usuário manipula diretamente os elementos de interface em uma representação geométrica do *layout* do diálogo. A terceira vista apresenta o diálogo com seu *look-and-feel* final e serve como protótipo.

Para que o Visual LED pudesse ser implementado, inicialmente foram estudados outros editores gráficos, tanto genéricos quanto de interface, com ênfase nos métodos de interação destes editores. As dificuldades de interação encontradas são discutidas neste trabalho e para elas propomos algumas soluções. A principal dificuldade encontrada foi a edição em hierarquia (seção 4.2.1).

Embora o Visual LED seja uma ferramenta específica do sistema IUP/LED, muitos dos problemas tratados neste trabalho não são particulares ao sistema IUP/LED. Assim, as soluções apresentadas para estes problemas também não são específicas para o Visual LED e podem ser aproveitadas em outros trabalhos e aplicações, como foi feito com a solução da árvore (seção 4.2.1).

1.3 Estrutura da dissertação

Neste Capítulo 1, apresentamos o papel da interface em uma aplicação. Além disso, apresentamos a motivação e o objetivo dessa dissertação, e descrevemos a sua estrutura.

No Capítulo 2, descrevemos as formas de especificação de *layout* concreta e abstrata, acompanhadas de um exemplo. Para discussão, apresentamos os modelos de *layout* abstrato de Xaw, Motif e TEX, incluindo o código do diálogo exemplo para cada um destes modelos.

No Capítulo 3, descrevemos o sistema portátil de interfaces com o usuário IUP/LED. Esta descrição apresenta o *toolkit* IUP e a linguagem de descrição LED. Além disso, listamos as limitações encontradas no uso do sistema IUP/LED e que nos levaram a procurar soluções e a desenvolver o Visual LED.

Ao desenvolver o Visual LED, buscamos criar uma interação simples e intuitiva para o usuário. Para tal, estudamos a interação de diversos editores gráficos. Apresentamos este estudo no Capítulo 4. Além disso, identificamos e propomos uma solução para o problema de edição em hierarquia.

No Capítulo 5, apresentamos os editores gráficos de interface Guide, Visual Basic, OPUS, Ibuild e FormsEdit. Fazemos então uma análise crítica destes editores e sugerimos uma taxonomia para editores gráficos de interface.

Finalmente, no Capítulo 6, apresentamos o Visual LED. Descrevemos as principais características do Visual LED e explicamos algumas decisões tomadas durante o desenvolvimento do Visual LED. Comparamos então o Visual LED com os editores gráficos de interface apresentados no Capítulo 5 e o classificamos de acordo com a taxonomia introduzida naquele capítulo.

Na conclusão, Capítulo 7, ressaltamos as contribuições do Visual LED e propomos algumas melhorias e extensões no Visual LED como trabalhos futuros.

Capítulo 2

Especificação de layout de diálogos

A interface de uma aplicação pode ser composta por um ou mais diálogos, onde diálogo é um grupo de objetos de interface que estão em um contexto espacial limitado [Mar92]. Ao fazer o projeto da interface de uma aplicação, o programador deve, primeiramente, definir os diálogos necessários para a interação do usuário com a tecnologia da aplicação. Feito isso, é necessário definir, para cada diálogo, os elementos de interface que deverão ser usados e a sua disposição (*layout*) no diálogo. Neste capítulo, discutimos alguns métodos de especificação de *layout*.

2.1 Paradigmas de *layout* de diálogos

Especificar o *layout* de um diálogo significa descrever a composição visual deste diálogo. Esta especificação pode ser feita de várias formas diferentes, e não existe um consenso sobre qual delas é a melhor. Como conseqüência, o modo de se especificar o *layout* varia de um sistema de interface para outro, sendo que os dois principais paradigmas são: *layout* abstrato e *layout* concreto.

Fazer uma especificação de *layout* concretamente significa fornecer explicitamente as posições e tamanhos dos elementos de interface. A forma de fornecer estas posições varia de acordo com o sistema de interface, mas em todos é necessário saber previamente as coordenadas e tamanho de cada elemento de interface, normalmente em coordenadas de tela.

Por outro lado, uma especificação abstrata de *layout* permite que se forneça a posição dos elementos relativamente a outros objetos. O modo de fornecer estas posições relativas também varia de acordo com o sistema de interface, mas agora não é necessário se preocupar com as coordenadas ou tamanho de cada elemento, apenas com a sua posição relativa dentro do diálogo. Especificações abstratas são potencialmente capazes de capturar a intenção de *layout*.

Para exemplificar os métodos de especificação de *layout*, suponhamos que se queira descrever o *layout* do diálogo da Figura 2.1.

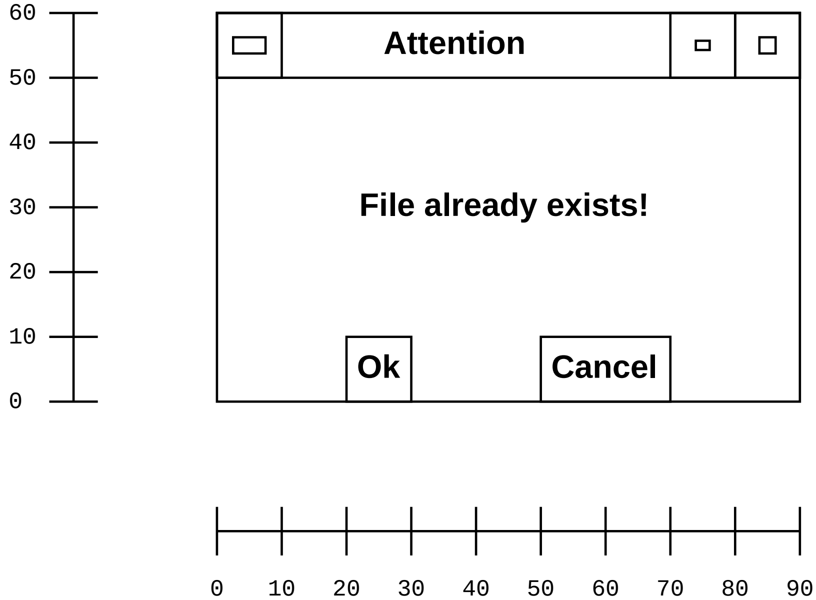

Figura 2.1: Geometria do *layout* de um diálogo.

Se a especificação fosse feita concretamente, precisaríamos fornecer os seguintes dados:

- diálogo: posição $(0, 0)$, tamanho 90×60;
- botão `Ok`: posição $(20, 0)$, tamanho 10×10;
- botão `Cancel`: posição $(50, 0)$, tamanho 20×10;
- mensagem: posição $(20, 30)$;

Se a especificação do *layout* fosse abstrata, poderíamos ter:

- botão `Ok`: 22% à direita do canto inferior do diálogo;
- botão `Cancel`: 22% à direita do botão `Ok`;
- mensagem: centralizada horizontalmente e verticalmente no espaço acima dos botões.

Cada uma destas duas formas de especificação de *layout* tem suas vantagens e desvantagens. Ao especificar o *layout* concretamente, o usuário sabe onde quer que determinado elemento se encontre; basta então definir sua posição. No *layout* abstrato, pode não ser tão simples construir o diálogo que se tem em mente: tem que se ter conhecimento do método utilizado para definir as posições dos elementos relativas umas às outras e então é necessário aplicar esses métodos para se alcançar o *layout* desejado. Por outro lado, os métodos de especificação normalmente são de fácil aprendizado, enquanto que para se conseguir determinar as posições exatas do *layout* concreto pode ser muitas vezes necessário desenhar o diálogo desejado em papel milimetrado. Além disso, fazer modificações em um *layout* de especificação concreta pode ser bastante trabalhoso, o que não acontece se a especificação for abstrata. Se o projetista de interfaces de *layout* concreto resolve inserir um novo elemento no diálogo, ou modificar a posição ou tamanho de algum elemento do diálogo, ou do próprio diálogo, pode ser necessário recalcular a posição de muitos ou de todos os elementos. Se o *layout* for abstrato, no pior caso, será necessário modificar as posições relativas dos elementos relacionados com o elemento sendo modificado ou inserido. Se a modificação for feita na posição ou tamanho do diálogo, o projetista não precisa alterar nada, pois as posições e tamanhos dos elementos do diálogo são recalculados automaticamente. Assim, as especificações abstratas permitem manter a intenção de *layout*.

Considere novamente a Figura 2.1. Se quiséssemos inserir um botão `Ignore` entre os botões `Ok` e `Cancel`, na especificação concreta teríamos que aumentar o tamanho do diálogo e trocar as posições do botão `Cancel` e da mensagem. Na especificação abstrata, teríamos que trocar a posição do botão `Cancel` para ser à direita do `Ignore`. Se aumentássemos o tamanho do diálogo e quiséssemos manter a posição relativa dos elementos, então teríamos que trocar a posição de todos os elementos na especificação concreta, enquanto que nada teria que ser feito na especificação abstrata.

2.2 Modelos de *layout* abstrato

Nesta seção, descrevemos os paradigmas de *layout* abstrato adotados nos *toolkits* de interface Xaw e Motif e no processador de textos TEX [Knu84]. Para cada um deles, apresentamos o código necessário para gerar o diálogo mostrado na Figura 2.1 e discutimos os problemas encontrados.

2.2.1 Xaw

O Xaw (*X Athena Widget*) [DEC88] é um *toolkit* de interface para o sistema X Window. O modelo de *layout* abstrato adotado por este *toolkit* é conhecido como *attachment*. Neste modelo, a posição de um filho do diálogo pode ser relativa à posição de outro filho do diálogo ou às bordas do próprio diálogo.

Os elementos do Xaw possuem uma série de atributos que são usados para definir a sua posição. Quando o usuário deseja definir a posição de um elemento em relação a outro, ele pode fornecer as distâncias horizontal e vertical do elemento relativo a um elemento fixo ou às bordas do diálogo, em *pixels*, ou pode, simplesmente, especificar se um elemento está abaixo ou à direita de outro. O usuário pode definir ainda que a distância entre os elementos se mantenha constante, caso o tamanho do diálogo seja alterado. Neste caso, quando o tamanho do diálogo é alterado, os tamanhos dos elementos também são alterados para manter a distância constante.

O código da Figura 2.2 é uma tentativa de criar o diálogo da Figura 2.1. No entanto, não conseguimos gerar um espaçamento entre os dois botões, nem entre o botão `Ok` e a borda. Este espaçamento poderia ser acrescentado em *pixels* ou determinando-se a posição inicial dos botões; no entanto, este espaço seria mantido fixo, não se alterando quando o tamanho dos diálogos fosse alterado.

2.2.2 Motif

O Motif [OSF91a, OSF91b] também é um *toolkit* de interface para o sistema X Window. Assim como no Xaw, o Motif também adota o também adota o modelo de *attachment* para modelo de *layout* abstrato. No Motif, porém, é possível se definir uma restrição para cada um dos quatro lados do elemento. Um elemento de interface do Motif pode ser relacionado não só às bordas do diálogo ou a outro elemento de interface, mas também a uma posição relativa dentro do diálogo ou à posição inicial de um outro elemento de interface. Estas restrições são definidas através de atributos dos elementos.

O código da Figura 2.3 é uma tentativa de criar o diálogo da Figura 2.1. No diálogo criado com o código da Figura 2.3 não conseguimos o espaçamento entre os botões e as bordas. Como no Xaw, é possível inserir este espaçamento, porém de forma fixa. O diálogo gerado pelo código da Figura 2.3 também não centraliza o *label* no espaço entre os botões e o topo do diálogo. Podemos centralizar o *label* fixamente, mas não abstratamente.

2.2.3 TEX

No modelo de *attachment* usado no Xaw e no Motif, o usuário define restrições às posições dos elementos de interface a partir de posições de outros elementos. No modelo de *boxes-and-glue* adotado pelo TEX, o usuário não se preocupa em definir a relação entre as coordenadas das posições dos elementos, mas sim a relação entre o posicionamento dos objetos. No modelo de *boxes-and-glue*, a posição dos elementos de interface é especificada, principalmente, agrupando-os e definindo a direção do grupo como sendo horizontal ou vertical. Quando o grupo é horizontal (`hbox`), os elementos que o compõem estão alinhados pelos seus lados superiores; quando o grupo é vertical (`vbox`), eles estão alinhados pelos seus lados esquerdos. O usuário pode definir ainda a existência de espaços expansíveis (`glue`, `fill`) entre elementos consecutivos de um grupo.

O código mostrado na Figura 2.4 é uma tentativa de criar em TEX o diálogo da Figura 2.1. O diálogo obtido é exatamente o desejado.

```
void main(int argc, char *argv[])
{
  XtAppContext context;
  Widget form, label, button1, button2, toplevel;

  toplevel = XtAppInitialize(&context, "attention", 0,
                             NULL, &argc, argv, NULL, NULL, 0);

  form = XtVaCreateManagedWidget("form",
        formWidgetClass, toplevel,
        NULL);

  label = XtVaCreateManagedWidget("File already exists!",
        labelWidgetClass, form,
        XtNleft,      (XtEdgeType) XtChainLeft,
        XtNright,     (XtEdgeType) XtChainRight,
        XtNtop,       (XtEdgeType) XtChainTop,
        NULL);

  button1 = XtVaCreateManagedWidget(\"Ok",
        commandWidgetClass, form,
        XtNbottom,    (XtEdgeType) XtChainBottom,
        XtNleft,      (XtEdgeType) XtChainLeft,
        XtNfromVert,  (XtArgVal)   label,
        NULL);

  button2 = XtVaCreateManagedWidget("Cancel",
        commandWidgetClass, form,
        XtNright,     (XtEdgeType) XtChainRight,
        XtNbottom,    (XtEdgeType) XtChainBottom,
        XtNfromHoriz, (XtArgVal)   button1,
        XtNfromVert,  (XtArgVal)   label,
        NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(context);
}
```

Figura 2.2: Código do diálogo em Xaw.

```
void main(int argc, char *argv[])
{
  XtAppContext context;
  Widget form, label, button1, button2, toplevel;

  toplevel = XtAppInitialize(&context, "attention", 0,
                             NULL, &argc, argv, NULL, NULL, 0);

  form = XtVaCreateManagedWidget("form",
        xmFormWidgetClass, toplevel,
        NULL);

  label = XtVaCreateManagedWidget("File already exists!",
        xmLabelWidgetClass, form,
        XmNtopAttachment,    XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_NONE,
        XmNleftAttachment,   XmATTACH_FORM,
        XmNrightAttachment,  XmATTACH_FORM,
        NULL);

  button1 = XtVaCreateManagedWidget(\"Ok",
        xmPushButtonWidgetClass, form,
        XmNleftAttachment,   XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        NULL);

  button2 = XtVaCreateManagedWidget("Cancel",
        xmPushButtonWidgetClass, form,
        XmNrightAttachment,  XmATTACH_FORM,
        XmNbottomAttachment, XmATTACH_FORM,
        NULL);

  XtRealizeWidget(toplevel);
  XtAppMainLoop(context);
}
```

Figura 2.3: Código do diálogo em Motif.

```
\def\box#1{\vbox{\hrule\hbox{\vrule #1 \vrule}\hrule}}

\box{
  \vbox to 5cm
  {
    \vfill
    \hbox to 6cm
    {
      \hfill
      { File already exists! }
      \hfill
    }
    \vfill
    \hbox to 6cm
    {
      \hfill
      \box{Ok}
      \hfill
      \box{Cancel}
      \hfill
    }
  }
}

\end
```

Figura 2.4: Código do diálogo em $\TeX$.

Capítulo 3

O sistema IUP/LED

Neste capítulo, descrevemos o sistema portátil de interface com o usuário IUP/LED. Como o Visual LED é uma ferramenta para desenvolvimento de interfaces em LED, muitas das suas características foram determinadas por características do sistema IUP/LED. É preciso então entender as idéias básicas do IUP/LED para entender o Visual LED.

3.1 O que é o IUP/LED

O sistema IUP/LED é composto por uma linguagem de especificação de diálogos LED e por um *toolkit* virtual IUP. Este sistema foi projetado para facilitar a criação de interfaces e para solucionar algumas dificuldades encontradas no trabalho de produção do TeCGraf, sendo a principal delas a dificuldade de se especificar, de forma portátil, boas interfaces com o usuário [Lev93].

O sistema IUP/LED permite que as interfaces geradas tenham tanto um *look-and-feel* fixo quanto um *look-and-feel* nativo. Poder optar entre o *look-and-feel* fixo ou nativo é uma vantagem do sistema IUP/LED, pois, se um usuário trabalha com diversas aplicações em uma mesma máquina, então será mais fácil para ele usar uma nova aplicação se ela tiver o *look-and-feel* nativo, ou seja, semelhante ao das outras aplicações. Por outro lado, um usuário que trabalha com uma mesma aplicação em diversas máquinas vai preferir um *look-and-feel* fixo, ao qual ele já está acostumado e com o qual ele é eficiente.

3.2 IUP

O IUP é um *toolkit* virtual contendo aproximadamente cinqüenta funções. O seu objetivo é possibilitar a construção e manipulação de diálogos portáteis, onde cada diálogo é formado por elementos de interface que interagem com o usuário.

Um sistema ser portátil significa que ele pode ser transportado de uma plataforma para outra com pouca ou nenhuma modificação [FGL93]. Foi para alcançar esta portabilidade que o IUP foi implementado como um *toolkit* virtual: ao invés de utilizar as rotinas específicas de acesso ao sistema de interface nativo, o IUP utiliza módulos que isolam estas rotinas, os *drivers*. A Figura 3.1 ilustra a arquitetura do IUP.

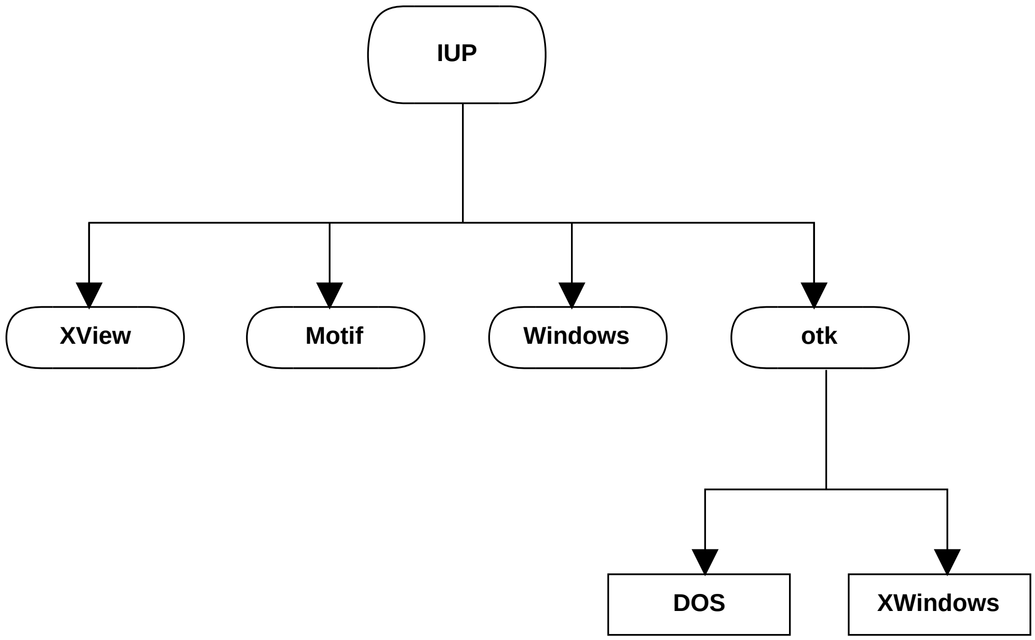

Figura 3.1: Arquitetura do IUP.

Com esta arquitetura, o IUP possibilita que as interfaces tenham tanto um *look-and-feel* fixo quanto um *look-and-feel* nativo. Para prover um *look-and-feel* nativo, o IUP utiliza o *driver* específico da plataforma. O *look-and-feel* fixo é alcançado utilizando o *driver* `otk` integrado a um sistema de interface totalmente portátil escritos pelo TeCGraf, originalmente para DOS sem windows.

3.3 LED

LED é uma linguagem de expressões para especificação de *layout* de diálogos. Esta linguagem é simples, de fácil compreensão, e descreve diálogos textualmente. O modelo de especificação de diálogos adotado por LED segue o paradigma de *layout* abstrato e é baseado no modelo de *boxes-and-glue* do TEX.

LED permite a especificação de vários diálogos para uma mesma aplicação. Os códigos dos diálogos são independentes entre si e são independentes do código da tecnologia da aplicação. Esta independência permite que se altere a interface ou a tecnologia da aplicação sem que a alteração de uma implique, necessariamente, na alteração da outra.

Outra característica importante de LED é que ela é interpretada em tempo de execução, o que implica que não é necessário recompilar a aplicação quando se altera a sua interface.

Além disso, para se visualizar a interface, não é necessário gerar a tecnologia da aplicação. Portanto, é possível criar rapidamente um protótipo da interface da aplicação. Mudanças neste protótipo podem ser feitas e testadas rapidamente, pois só é necessário alterar o arquivo contendo a descrição em LED da interface.

3.4 Elementos de interface

Os elementos que compõem o diálogo em LED são especificados pela sua funcionalidade, que é fornecida por parâmetros, enquanto que sua aparência é definida por atributos. Ao se especificar um elemento de interface, é obrigatório fornecer os parâmetros, enquanto que é opcional fornecer os atributos; esta distinção está clara na sintaxe de LED. A sintaxe de uma expressão em LED é:

$$v = f[a](p),$$

onde:

- v é o nome (opcional) que deverá ser utilizado pela aplicação para acessar o elemento de interface que está sendo definido pela expressão $f[a](p)$;
- f é o tipo de elemento sendo descrito;
- a é a lista de atributos para esse elemento de interface;
- p é a lista de parâmetros que definem a funcionalidade dos elementos do tipo f.

São vários os elementos de interface disponíveis em LED; eles podem ser divididos em quatro categorias: agrupamento, composição, preenchimento e primitivos. A seguir, apresentamos estas categorias e seus respectivos elementos.

A primeira categoria é a de agrupamento, que define uma funcionalidade comum para um grupo de elementos. Os elementos desta categoria são:

- `dialog`: compõe um diálogo de interação com o usuário;
- `radio`: agrupa `toggle`'s, restringindo que apenas um deles esteja ativo de cada vez;
- `menu`: agrupa `item`'s e `submenu`'s.

Os elementos de composição são os responsáveis por definir a disposição geométrica dos elementos que são seus descendentes, como em TEX, há dois elementos nesta categoria:

- `hbox`: exibe horizontalmente seus elementos;
- `vbox`: exibe verticalmente seus elementos.

A próxima categoria é a de preenchimento. Em TEX, esta categoria possui dois elementos de preenchimento: `hfill` e `vfill`. O `hfill` ocupa proporcional e dinamicamente os espaços vazios em um `hbox`, e o `vfill` faz o mesmo em um `vbox`. Como o conceito de `hfill` e `vfill` é o mesmo—a única diferença é a direção do grupo ao qual eles pertencem—optou-se em LED por se ter apenas o elemento `fill` que ocupa proporcional e dinamicamente os espaços vazios, tanto em `hbox`'s quanto em `vbox`'s.

A última categoria é a de elementos primitivos, que são os elementos que efetivamente interagem com o usuário final. Esta categoria é composta pelos seguintes elementos:

- `button`: botão;
- `canvas`: área de trabalho;
- `frame`: coloca uma borda em volta de um elemento de interface;
- `hotkeys`: teclas de funções;
- `image`: imagem estática;
- `item`: item de menu;
- `label`: texto estático;
- `list`: listas de *strings*;
- `submenu`: submenu de menu;
- `text`: captura um texto de uma ou mais linhas;
- `toggle`: botão de dois estados;
- `valuator`: captura um valor numérico.

Destes elementos, o `canvas` é o único que não tem o mesmo comportamento em todos os sistemas de interface existentes. O tratamento de eventos no `canvas` é responsabilidade total da aplicação. Na Figura 3.2, mostramos o código LED e na Figura 3.3 o diálogo gerado por ele para o exemplo da Figura 2.1. Note a semelhança com a especificação TEX da Figura 2.4.

```
dialog
(
 vbox
 (
  fill (),
  hbox
  (
   fill (),
   label ( "File already exists!" ),
   fill ()
  ),
  fill (),
  hbox
  (
   fill (),
   button ( \"Ok",do_Ok ),
   fill (),
   button ( "Cancel",do_Cancel ),
   fill ()
  )
 )
)
```

Figura 3.2: Código do diálogo em LED.

3.5 Estrutura interna

Internamente ao sistema IUP/LED, cada diálogo é organizado e armazenado em uma estrutura hierárquica do tipo árvore. A raiz da árvore é sempre um elemento do tipo `dialog` e tem apenas um filho. Os outros nós internos da árvore são ou elementos de composição (`hbox` ou `vbox`) ou o elemento de agrupamento `menu`, que pode ter um ou mais filhos, ou ainda podem ser ou o elemento de agrupamento `radio` ou o elemento primitivo `frame`, que sempre têm um único filho. As folhas da árvore são ou elementos de preenchimento `fill` ou algum elemento primitivo (exceto `frame`). A Figura 3.4 mostra a árvore do diálogo da Figura 3.2.

Figura 3.3: Diálogo gerado pelo sistema IUP/LED.

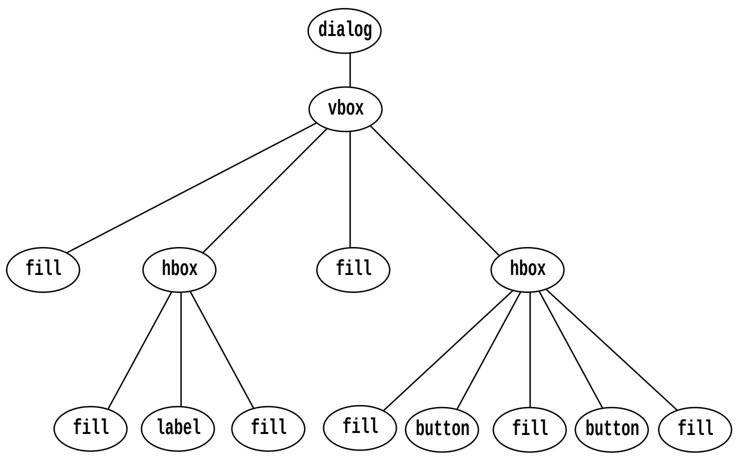

Figura 3.4: Árvore do diálogo da Figura 3.2.

3.6 Atributos

O sistema de atributos de elementos possui um mecanismo de herança: as variáveis definidas em um elemento são automaticamente exportadas para seus filhos. Isto permite a alteração de atributos de toda uma sub-árvore e, conseqüentemente, que mudanças globais sejam feitas com facilidade. Por outro lado, mudanças locais também podem ser feitas com facilidade, pois, se um elemento define o valor de um atributo já definido por um de seus ancestrais, então o novo valor passa a ter prioridade para ele e seus descendentes.

Os atributos não reconhecidos são mantidos, mas não interpretados, pelo IUP/LED. Isto permite que a tecnologia da aplicação use o sistema de atributos, e não variáveis globais, para armazenar os estados dos elementos de interface. Esta característica permite também que o usuário faça ajuste fino da interface para uma determinada plataforma.

3.7 Limitações do sistema IUP/LED

O sistema IUP/LED vem sendo utilizado com sucesso no TeCGraf no desenvolvimento das interfaces de suas aplicações. A linguagem LED foi facilmente aprendida e assimilada pelos programadores do grupo. Boas interfaces têm sido projetadas com rapidez. No entanto, surgiram novas dificuldades. A maior dificuldade encontrada em LED, e discutida nesta seção, não é, porém, específica de LED, mas é uma dificuldade encontrada em qualquer linguagem de especificação de diálogos.

O primeiro passo de um projetista de interface para construir uma interface é imaginar esta interface. Ao fazer isto, o projetista cria em sua mente uma imagem da interface que ele deseja construir. Em seguida, ele precisa transformar a imagem que ele tem em mente em uma descrição LED. Esta tarefa é bastante complexa e dificilmente se consegue de imediato escrever uma descrição LED que corresponda exatamente à interface imaginada, a não ser nos casos mais simples. Uma vez escrita a descrição LED inicial, o projetista inicia o seguinte processo:

- visualiza a interface gerada;
- compara a interface gerada com a desejada;
- identifica as diferenças entre as duas interfaces;
- identifica a parte da descrição LED responsável pela diferença entre as duas interfaces;
- modifica a descrição LED.

Este processo é análogo ao ciclo de implementação, compilação e teste da tecnologia da aplicação. No entanto, como LED é interpretada, este ciclo é bem mais rápido. Normalmente, o programador LED não escreve toda a descrição da interface de uma só vez, a menos que seja uma interface simples. O procedimento usual é construir a interface por partes, repetindo assim o processo descrito acima para cada uma das partes. Embora IUP/LED permita uma prototipagem rápida, alcançar o diálogo desejado pode ser uma tarefa demorada e trabalhosa.

Reciprocamente, ler a descrição LED de uma interface e visualizá-la pode ser igualmente difícil, principalmente porque uma interface é freqüentemente composta por vários diálogos complexos, o que implica em expressões LED complexas.

No capítulo 6, apresentamos uma ferramenta chamada Visual LED, que fornece soluções para as limitações descritas acima.

Capítulo 4

Seleção e agrupamento em editores gráficos

Na fase inicial da implementação do Visual LED, estudamos as formas de interação usadas em editores gráficos genéricos e editores gráficos de interfaces. Estudamos a criação e seleção de primitivas, assim como as operações permitidas sobre elas. Analisando as vantagens e desvantagens das diversas interações estudadas, implementamos em Visual LED uma interação que fosse simples e intuitiva para o usuário (seção 6.3).

As operações básicas necessárias em um editor gráfico de interfaces não diferem muito das operações necessárias em um editor gráfico genérico. Algumas das operações sobre primitivas comuns a ambos são: criar, selecionar, copiar, apagar, mover, agrupar, desagrupar, trocar os atributos.

Neste capítulo, estudamos a interação com o usuário adotada Para as operações básicas nos editores gráficos genéricos Corel DRAW (versões 3 e 4) e idraw. Enfocamos, especialmente, a política de interação usada nas operações de agrupamento e desagrupamento, uma vez que esta operação seria amplamente usada em Visual LED, dado o modelo de *layout* abstrato de LED (seção 3.3). Os editores gráficos de interfaces são descritos e discutidos no próximo capítulo.

4.1 Seleção

A operação de seleção de um único elemento no Corel DRAW e no idraw é similar; a seleção de vários elementos é idêntica. No idraw, a seleção de um elemento é feita pressionando-se o botão da direita do *mouse* sobre o elemento a qualquer momento, ou ligando-se o modo de seleção e pressionando-se o botão da esquerda do *mouse*. Se um objeto que já estava selecionado for novamente selecionado, ele continua selecionado. No entanto, se outros objetos estavam selecionados, eles são de-selecionados e só o elemento fica selecionado. No Corel DRAW, sempre é necessário estar no modo de seleção para se selecionar um elemento. A seleção é feita pressionando-se o botão da esquerda do *mouse* sobre o elemento desejado.

Quando um elemento está selecionado, a operação de escala por manipulação direta fica ativa para este elemento. Se um elemento que já estava selecionado for novamente selecionado, então a operação por manipulação direta possível sobre os elementos selecionados é trocada de escala para rotação, mas nenhum elemento é de-selecionado.

Existem duas maneiras de selecionar vários objetos de uma só vez no Corel DRAW e no idraw. Na primeira, seleciona-se o elemento com a tecla ⟨`Shift`⟩ pressionada, causando o acréscimo deste elemento à lista dos elementos selecionados. Na outra forma, que chamamos de **fence**, pressiona-se o botão do *mouse* em uma posição onde não existe nenhum elemento. Fazendo isso e movendo-se o *mouse*, mantendo-se o botão pressionado, surge uma área retangular. Quando o botão é solto, todos os elementos que estiverem completamente dentro desta área serão selecionados. A região determinada por uma *fence* não é necessariamente retangular. Dependendo do editor gráfico, esta região é determinada por um retângulo ou uma curva fechada (chamada **laço**).

4.2 Agrupamento

A operação de agrupamento tem funções distintas nos editores gráficos de interfaces de *layout* abstrato e nos editores gráficos genéricos e de interfaces de *layout* concreto.

Nos editores gráficos genéricos e de interfaces de *layout* concreto, a função de se agrupar objetos é facilitar a manipulação de objetos que para o usuário têm algum tipo de relacionamento lógico, mas não necessariamente geométrico.

Nos editores gráficos de interfaces de *layout* abstrato, a operação de agrupamento é fundamental, pois a composição dos diálogos é feita agrupando-se os elementos de interface. Neste caso, o agrupamento é uma operação não apenas lógica, mas também geométrica.

Apesar de a função da operação de agrupamento ser diferente nos editores gráficos genéricos e de interfaces de *layout* concreto, e nos editores gráficos de interfaces de *layout* abstrato, alguns problemas encontrados nesta operação são comuns a todos eles. Discutimos estes problemas nesta seção.

4.2.1 Editores gráficos genéricos

A operação de agrupamento no Corel DRAW 3 e no idraw é feita da seguinte forma: seleciona-se os elementos que devem fazer parte do grupo e, em seguida, seleciona-se a opção de agrupamento. O grupo passa a ser um único elemento; todas as operações que incidem sobre o grupo incidem sobre todos os seus **filhos**, isto é, os elementos que fazem parte do grupo. A esta política de agrupamento, damos o nome de **política simples**. É importante notar que um agrupamento não implica em nenhuma mudança na posição ou na aparência dos componentes do grupo. Podemos dizer então que o agrupamento nos editores gráficos genéricos é uma operação topológica e não geométrica.

Um problema

Se, por um lado, a simplicidade é uma grande vantagem da política simples, por outro lado, ignorar a hierarquia de grupos é uma grande desvantagem. Como um grupo passa a ser tratado como uma primitiva simples, não se tem mais acesso direto aos seus componentes. Assim sendo, para se fazer qualquer operação em um elemento interno ao grupo, deve-se primeiro desfazer todo o grupo até chegar ao nível do elemento desejado, fazer a operação desejada, e refazer todo o grupo. No pior caso, pode ser necessário desfazer todos os níveis de agrupamento e refazer tudo após a alteração.

O exemplo a seguir mostra um caso onde este problema é claro. O usuário deseja fazer o desenho mostrado na Figura 4.1. Inicialmente, ele desenha um segmento de reta e um triângulo, e os agrupa formando uma seta (Figura 4.2). A seguir, ele cria dois retângulos e cria um outro grupo contendo os retângulos e a seta (Figura 4.3). Finalmente, ele cria um retângulo em volta do grupo existente e cria mais um grupo (Figura 4.1), o que facilita operações de mover e duplicar todo o desenho. Feito tudo isso, se o usuário resolve trocar a cor do triângulo da seta, então todos os agrupamentos terão que ser desfeitos; caso contrário, a operação de troca de cor afetará todos os elementos do grupo, trocando também as cores dos retângulos e da reta. Uma vez desfeitos todos os grupos, o usuário consegue selecionar o triângulo e mudar sua cor, mas é necessário reagrupar todos os elementos após a mudança.

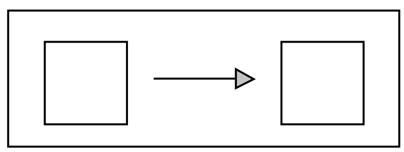

Figura 4.1: Grupo total.

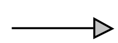

Figura 4.2: Grupo contendo seta.

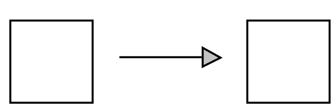

Figura 4.3: Grupo contendo seta e retângulos.

Na prática, este problema é ainda mais grave pois, mesmo em diagramas com poucas primitivas, é possível que a hierarquia de grupos seja complexa.

Uma melhoria a ser introduzida nos editores gráficos que usam a política simples seria então **edição em profundidade**, isto é, permitir percorrer a hierarquia de um grupo e editar seus componentes. Se fosse possível editar em profundidade, poder-se-ia fazer mudanças de atributos em qualquer elemento da hierarquia facilmente, sem que fosse necessário desagrupar para depois reagrupar. Teríamos, então, ao mesmo tempo, a conveniência dos grupos e o acesso direto às primitivas.

Permitir edição em profundidade não é simples: a principal questão a ser resolvida é como o usuário vai percorrer a hierarquia. Pode ser difícil selecionar diretamente no desenho o elemento desejado, pois vários elementos podem estar sobrepostos em uma mesma posição da tela. Para editar em profundidade, o ideal seria que todos os elementos de um desenho pudessem ser vistos simultaneamente e sem ambigüidade, o que pode não ser possível na vista em que se está desenhando, se há sobreposição de elementos opacos.

Uma solução

O Corel DRAW 4 já apresenta uma solução para os problemas apresentados acima e permite a edição em profundidade. Para percorrer a hierarquia e acessar os elementos internos a um grupo, seleciona-se o elemento com a tecla ⟨`Control`⟩ pressionada. Se o elemento já está selecionado, então o seu filho naquela posição é selecionado. Se o elemento é o mais profundo da hierarquia e não tem mais filhos, então o seu pai é selecionado. Não é possível selecionar simultaneamente mais de um elemento interno à hierarquia. Se o elemento não fizer parte de um grupo, então a tecla ⟨`Control`⟩ é ignorada e a seleção simples é feita normalmente.

Quando um elemento é selecionado, esta seleção é indicada tanto graficamente na vista contendo o desenho, quanto textualmente em uma barra de mensagem acima da área de trabalho. A indicação gráfica da seleção de um elemento no primeiro nível de hierarquia é diferenciada da indicação gráfica da seleção de elementos em outros níveis. A seleção de elementos no primeiro nível da hierarquia é indicada por quadrados nos vértices e no ponto médio de cada aresta do seu *bounding box* (Figura 4.4). Se o elemento não pertence ao primeiro nível da hierarquia, as marcas utilizadas para indicar a *bounding box* do elemento são círculos, ao invés de quadrados. A indicação textual descreve o objeto selecionado. Retângulos e elipses são descritos pela sua altura, largura e centro; curvas são descritas pelo número de pontos especificados; textos são descritos pelo seu fonte, tamanho do fonte e posição; finalmente, os grupos são descritos pelo número de elementos que contêm. A descrição textual pode ser ambígua, pois os elementos não têm nome.

Caso haja sobreposição de elementos, o elemento da frente sempre é selecionado nas áreas de sobreposição. Para resolver este problema, o Corel DRAW permite que o usuário trabalhe em um modo onde os elemento são representados em *wireframe*. Assim, é possível visualizar todos os elementos e selecionar qualquer um deles. No entanto, selecionar o elemento desejado na vista *wireframe* pode ser confuso. Quando se tem a interseção de vários elementos, e logo de várias arestas destes elementos, pode ser difícil identificar a aresta pertencente ao elemento desejado. Além disso, para selecionar um elemento, é necessário posicionar o cursor sobre uma de suas arestas e pressionar o botão da esquerda do *mouse*, isto é, na vista *wireframe* os objetos opacos perdem o seu interior.

Como a indicação gráfica da seleção de um elemento é feita indicando a sua *bounding box*, uma outra dificuldade apresentada em alguns casos é diferenciar a indicação gráfica da seleção do grupo da indicação gráfica da seleção de um de seus filhos. Isto acontece quando um grupo tem a mesma *bounding box* que um, ou mais, de seus filhos. A Figura 4.4 ilustra esta situação. Cada uma das retas mais a elipse sobre ela formam um grupo; estes dois grupos formam um segundo grupo. Quando se tem a indicação da seleção mostrada na Figura 4.4, não é possível saber quem está selecionado: o grupo formado pela reta e elipse da esquerda, o grupo formado pela reta e elipse da direita, ou ainda, o grupo formado por estes dois grupos; a indicação gráfica nos três casos é a mesma.

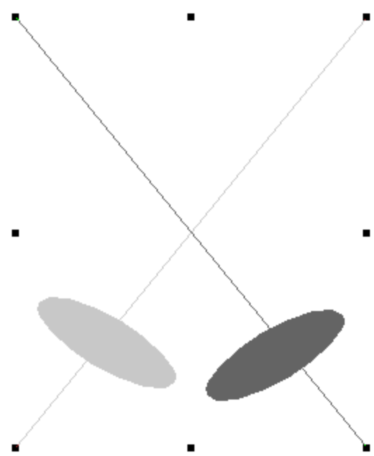

Figura 4.4: Grupo.

Neste caso, a única forma de resolver a ambigüidade seria recorrer à indicação textual da seleção. Todavia, os grupos formados por cada uma das retas e elipses têm exatamente a mesma indicação textual, não sendo então possível identificar o elemento selecionado.

Outra solução

Buscando resolver os problemas apresentados pela política simples e pela solução adotada pelo Corel DRAW 4, apresentamos aqui uma nova solução para editores gráficos genéricos: como a hierarquia de um grupo é abstratamente uma árvore, criamos uma árvore para representar a hierarquia de um desenho de um editor gráfico.

Na árvore, todos os elementos do desenho são representados simultaneamente e sem ambigüidade. Desta forma, o usuário pode selecionar qualquer elemento do desenho, a qualquer momento. Para facilitar a visualização da relação entre os nós da árvore e os elementos que eles representam, a árvore é apresentada em uma vista auxiliar.

Para efetuar uma operação em um elemento qualquer da hierarquia, o usuário seleciona o objeto a ser modificado na árvore e nele aplica a operação desejada. A seleção de objetos pode ser efetuada tanto na vista da árvore quanto na vista da edição. Quando um elemento é selecionado em uma vista, a seleção é indicada também na outra. Uma vez selecionado o objeto, a operação deve ser aplicada normalmente na vista do desenho. A vista contendo a árvore é, portanto, apenas um mecanismo auxiliar de seleção. Como esta vista não é necessária durante todo o trabalho do usuário, ela só é apresentada sob demanda. Uma idéia semelhante foi implementada no modelador de sólidos Genesys, onde a hierarquia refletia construções CSG [Fis91].

Esta solução transforma operações trabalhosas e demoradas em operações simples e imediatas. No entanto, para implementar esta solução, é necessário poder acessar e modificar a estrutura de dados do editor gráfico. Isto impede que esta solução seja simulada para editores gráficos comerciais, ou seja, implementada como uma camada acima destes editores, pois geralmente não é fornecido ao usuário acesso programável a essas estruturas de dados.

Para podermos avaliar esta solução da árvore, implementamo-na no TeCDraw, um editor gráfico do TeCGraf [TeC94]. A solução se mostrou adequada para o problema de edição em profundidade. Ela é simples de ser utilizada, e permite a seleção de qualquer elemento em qualquer profundidade na hierarquia sem ambigüidade. Além disso, nos casos em que não existe ambigüidade, o usuário pode fazer a seleção diretamente na vista do desenho se desejar.

4.2.2 Editores gráficos de interfaces de *layout* concreto

Nos editores gráficos de interfaces de *layout* concreto, a função e os problemas do agrupamento são os mesmos de um editor gráfico genérico. Na verdade, o problema de visualização na vista de edição deveria ser amenizado, pois em uma interface não é comum ter elementos sobrepostos. No entanto, tanto o Guide quanto o Visual Basic permitem que esta sobreposição ocorra. Portanto, a solução da árvore apresentada acima para os editores gráficos genéricos pode ser aplicada também a editores gráficos de interfaces de *layout* concreto. No entanto, não pudemos avaliar esta solução nos editores gráficos de interfaces de *layout* concreto: não pudemos simular a solução em um editor comercial, pois estes não fornecem um acesso programável à estrutura de dados; além disso, o TeCGraf não possui um editor gráfico de interfaces de *layout* concreto.

4.2.3 Editores gráficos de interfaces de *layout* abstrato

Nos editores gráficos de interfaces de *layout* abstrato que usam a política simples para agrupamento, a ausência de edição em profundidade é ainda mais séria. Se, nos editores gráficos genéricos e nos editores gráficos de interfaces de *layout* concreto, a operação de agrupamento é apenas uma maneira de facilitar a manipulação de elementos, nos editores gráficos de interfaces de *layout* abstrato, ela é essencial para a composição de diálogos.

Como a função da operação de agrupamento nos editores gráficos de interfaces de *layout* abstrato é diferente da sua função nos editores gráficos genéricos e nos editores gráficos de interfaces de *layout* concreto, surge a questão de se a solução da árvore continua sendo adequada para resolver o problema. Para esclarecer esta dúvida, temos que levar em conta algumas diferenças entre os editores gráficos de interfaces de *layout* abstrato e os genéricos e de interfaces de *layout* concreto. Primeiramente, em um editor gráfico de interfaces de *layout* abstrato, os elementos jamais se sobrepõem dentro de um diálogo; logo, todos os elementos são sempre visíveis. Sendo assim, é possível fazer a seleção do objeto diretamente na vista de edição. A árvore também não facilitaria a visualização do *layout* dos diálogos pelo usuário, pois é uma representação topológica e não geométrica. Deste modo, ainda seria necessário fazer a visualização mentalmente. Podemos concluir que a solução de apresentar a árvore para os editores gráficos genéricos e de interfaces de *layout* concreto não seria adequada para editores gráficos de interfaces de *layout* abstrato.

Capítulo 5

Outros editores gráficos de interface

Neste capítulo, descrevemos alguns editores gráficos de interface: Guide, Visual Basic, OPUS, ibuild e FormsEdit. Descrevemos a forma de interação de cada um deles e discutimos suas vantagens e desvantagens. Finalmente, fazemos uma análise comparativa entre estes editores. Como no capítulo anterior, o objetivo deste estudo é identificar técnicas de interação úteis para o Visual LED.

5.1 Guide

O Guide é um editor gráfico de interfaces para o OpenWindows da SUN [SUN90]. Assim sendo, as interfaces criadas por ele têm o *look-and-feel* do OpenLook; estas interfaces são implementadas utilizando o *toolkit* XView [Hel90].

No *toolkit* XView, a posição dos elementos pode ser especificada absoluta ou relativamente. Para possibilitar a especificação relativa, o XView fornece ao programador uma matriz bidimensional. O programador posiciona os elementos consecutivamente na linha ou na coluna, conforme ele determine. No entanto, este posicionamento relativo não mantém o *layout* inalterado quando o diálogo sofre alterações de tamanho, isto é, o XView não fornece *layout* abstrato, somente concreto.

Apesar do Guide utilizar o XView, ele só adota o posicionamento absoluto e, conseqüentemente, o *layout* concreto. No entanto, o usuário não precisa especificar numericamente as coordenadas dos objetos se não quiser, pois o posicionamento dos objetos pode ser feito por manipulação direta. Como o *layout* é concreto, os objetos têm posição fixa e não mudam de tamanho ou posição quando se altera o tamanho do diálogo.

O Guide permite agrupamentos e desagrupamentos, mas estes não têm nenhum significado hierárquico: sua única função é facilitar a manipulação dos objetos na tela, como nos editores gráficos genéricos. A política de agrupamento adotada é então a política simples.

Como o Guide só gera interfaces para o OpenLook, não existe nenhum compromisso com portabilidade ou *look-and-feel* nativo. Sendo assim, na vista na qual se monta a interface, a aparência dos diálogos e de seus objetos é bem próxima da sua aparência final. A interface pode ser testada a qualquer momento durante a sua construção. O protótipo para teste é criado imediatamente e é muito próximo da interface final. No entanto, para se obter um protótipo preciso e fiel é necessário compilar os arquivos criados pelo Guide, pois algumas propriedades dos elementos de interface só são definidas quando o código da interface é compilado. Estes arquivos contêm a descrição da interface e *stubs* (cabeçalho) das funções de *callback* da tecnologia da aplicação; elas devem ser preenchidas pelo programador. O protótipo é somente composto pela interface com funções de *callback* que não geram nenhuma ação.

Esta característica do Guide de criar arquivos de *stubs* é perigosa e requer bastante atenção do programador. Caso o programador já tenha implementado as funções de *callback* e deseje fazer uma modificação na interface, ele deve estar atento para não gerar um novo arquivo de *stubs* sobre o modificado, o que poderia causar perda do trabalho já feito.

5.2 Visual Basic

O Visual Basic [MS92] é um editor gráfico de interfaces para microcomputadores compatíveis com o IBM-PC. Existem versões do Visual Basic tanto para o sistema operacional MS-DOS, quanto para Windows. O Visual Basic é, na verdade, mais do que um editor de interfaces, pois ele permite, além da criação da interface, a programação das funções de *callback*.

O Visual Basic adota o *layout* concreto. O usuário pode definir a posição e tamanho dos elementos por manipulação direta. Os atributos dos elementos podem ser definidos através de um formulário.

O usuário pode selecionar um ou mais elementos de uma vez. Para selecionar um elemento, basta posicionar o cursor sobre ele e pressionar o botão do *mouse*. Para acrescentar outros elementos aos selecionados, deve-se selecioná-los com a tecla ⟨`Shift`⟩ pressionada. Se o usuário desejar selecionar vários elementos de uma só vez, ele pode usar uma *fence* retangular. Apesar de o Visual Basic permitir que elementos se sobreponham, só é possível acessar o elemento da frente nas áreas onde ocorre a sobreposição.

Como o agrupamento não é necessário em editores gráficos de *layout* concreto o Visual Basic não oferece a opção de agrupamento. O objetivo do agrupamento é facilitar a manipulação e operação sobre diversos elementos ao mesmo tempo. No entanto, isto pode ser feito selecionando-se os objetos desejados. A desvantagem de se usar a seleção múltipla ao invés de grupos é que é necessário selecionar os mesmos elementos, toda vez que se deseje trabalhar com eles.

Os elementos `frame` e `picture` do Visual Basic são elementos agrupadores de outros elementos. No entanto, não é possível inserir e retirar elementos de um grupo. Para que um elemento pertença a um grupo, este elemento deve ser criado dentro do grupo. Analogamente, para que um elemento seja retirado de um grupo, este elemento deverá ser apagado.

É possível fazer prototipação rápida da interface sendo construída em Visual Basic. A qualquer momento da construção, se a opção `run` for selecionada, é mostrado o diálogo.

Este diálogo tem o mesmo *look-and-feel* do diálogo resultante, e o comportamento dos seus elementos pode ser testado. Assim como o Guide, o Visual Basic não é multi-plataforma e não gera interfaces portáteis.

Ao se selecionar a opção `run`, são compiladas as funções de *callback* que já tiverem sido implementadas. Neste caso, a função de *callback* é ativada quando seu elemento é pressionado. Desta forma, o Visual Basic oferece mais que uma prototipação rápida.

A linguagem usada para programar as funções de *callback* em Visual Basic é a linguagem BASIC, que é popular e de fácil aprendizado. Além de permitir a programação das funções de *callback*, o Visual Basic contém um *debugger*, facilitando esta programação. Assim, o Visual Basic é um sistema que permite o desenvolvimento completo da aplicação, tanto da interface quanto da tecnologia.

5.3 OPUS

O editor gráfico de interface por manipulação direta OPUS (*On-line Penguims User Interface Specifier*) permite a criação de interfaces na linguagem Penguims sem programação [HM90]. Tanto o OPUS, quanto a sua linguagem Penguims, são implementados em C++ e usam o *toolkit* ARTKit e os elementos de interface oferecidos por ele.

O usuário do OPUS manipula diretamente caixas retangulares, cujas posições e bordas definem a posição e o tamanho do elemento de interface na interface final. Além destas caixas, o OPUS fornece aos usuários alguns elementos abstratos, isto é, elementos que não são visíveis na interface final, mas que visam facilitar a especificação da interface.

São três os elementos abstratos: `frames`, linhas de referência e restrições. Os `frames` permitem que se faça agrupamentos de elementos de interface ou de grupos já compostos. As linhas de referência são horizontais ou verticais e são sempre acopladas aos `frames`. A posição destas linhas de referência pode ser definida explicitamente pelas suas coordenadas, calculando-se máximos, mínimos e médias de restrições, ou ainda por uma constante de proporcionalidade da sua distância para seu `frame`. As linhas de referência facilitam o posicionamento de objetos de interface. O último e mais interessante destes elementos é a restrição. Restrições permitem relacionar as posições de uns objetos em relação a outros através de equações algébricas. Para relacionar dois objetos, o usuário define um objeto como fixo e outro como relativo. A escolha dos objetos fixo e relativo é feita por manipulação direta, utilizando para isto o elemento de interface de restrição (Figura 5.1): o círculo fica sobre a posição relacionada do elemento fixo e a seta aponta para a posição relacionada do elemento relativo. Feito isto, o usuário fornece a equação que calcula as coordenadas do objeto relativo em função das coordenadas do objeto fixo.

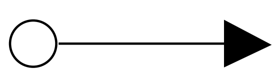

Figura 5.1: Representação da restrição.

A especificação do *layout* de um diálogo no OPUS é feita concretamente, estipulando-se posição e o tamanho dos elementos de interface. No entanto, é possível abstrair a especificação do *layout*: ao invés de defini-lo fornecendo explicitamente as posições e os tamanhos dos objetos, pode-se defini-lo estabelecendo-se a posição de um objeto em relação a outro, usando para isso as restrições e linhas de referência.

São duas as estratégias possíveis para cálculo da interface final em OPUS: *outside-in* e *inside-out*. Na estratégia *outside-in*, determina-se o tamanho da janela externa e seus componentes são determinados relativamente. Na estratégia *inside-out*, os componentes interiores têm seu tamanho natural definido e os componentes envolventes mudam de tamanho conforme o espaço requerido por seus filhos. Em uma mesma especificação de interface, o OPUS permite que se use ambas as estratégias, o que em muitos casos é necessário para se conseguir chegar a uma boa interface. O exemplo abaixo ilustra uma interface que utiliza as duas estratégias de interface.

Observe, na Figura 5.2, que o `Frame #2` tem seu tamanho determinado pelo `Frame #1`; neste caso, usou-se a estratégia *inside-out*. Por outro lado, as dimensões do `Frame #3` são determinadas pelo `Frame #2`, utilizando-se a estratégia *outside-in*. Esta interface poderia ser a de um editor gráfico, por exemplo, onde o `Frame #1` conteria o `canvas` e o `Frame #3` conteria um conjunto de botões.

O OPUS permite a edição dentro da hierarquia de um grupo. Ao se especificar uma interface, o OPUS apresenta diversas janelas, uma para cada grupo. Apenas um nível de hierarquia é visível em cada janela. Se no nível de profundidade apresentado em uma janela existe um grupo, então este grupo é visto como sendo apenas um elemento, ou seja, seus filhos não são visíveis. Os filhos de um grupo podem ser vistos e editados na janela relativa ao grupo. Este método de edição em hierarquia é simples e eficiente, mas pode se tornar confuso quando se tem muitos grupos e, conseqüentemente, muitas janelas.

O OPUS possui duas vistas: a vista programável, onde a edição é feita, e a vista de execução, que mostra o *look-and-feel* final do objeto. A vista programável contém todas as janelas mostrando os diversos grupos, os elementos abstratos (`frames`, linhas de referência e restrições) representados por retângulos, linhas e setas e os elementos de interface representados como caixas retangulares sem qualquer decoração (Figura 5.2). A vista de execução não está sempre presente; ela é mostrada apenas sob demanda. O objetivo desta vista é facilitar a visualização do relacionamento entre os diversos objetos da interface e permitir a exploração do comportamento dinâmico da interface. Para atingir este objetivo, a interface é mostrada com o seu *look-and-feel* final; na verdade, a interface mostrada não é exatamente igual à interface final, pois nesta vista os elementos abstratos ainda estão visíveis.

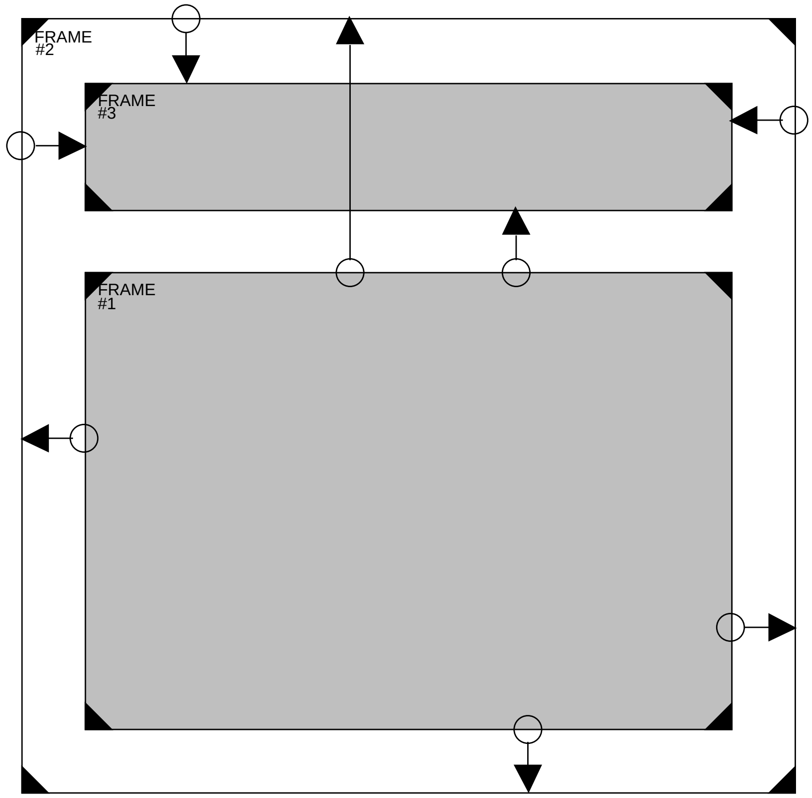

Figura 5.2: Construção de uma interface em OPUS usando ambas as estratégias.

5.4 Ibuild

O ibuild é o editor gráfico de interfaces do sistema InterViews, que permite criar interfaces gráficas por manipulação direta para aplicações em *workstations* [IV91].

Os diálogos são definidos pelo seu *layout* abstrato. O modelo de descrição de *layout* abstrato é baseado no modelo de *boxes-and-glue* do TEX e logo possui os elementos de composição `hbox` e `vbox` e os de preenchimento `hfill` e `vfill`. Os diversos elementos primitivos são combinados usando-se os elementos de composição e preenchimento para criar os diálogos. Os elementos de interface criados são representados por caixas e textos.

Uma vez criados, os elementos primitivos e os de preenchimento podem ser manipulados diretamente. Para agrupá-los em um `hbox` ou `vbox`, seleciona-se os objetos desejados, na ordem desejada, e seleciona-se a opção de agrupamento em `hbox` ou `vbox`. Uma vez agrupados, o grupo passa a ser tratado como um elemento único. No entanto, não é apresentado ao usuário nenhuma representação visual dos `hbox`'s ou `vbox`'s. Isso significa que o usuário, ao construir um diálogo no ibuild, não consegue visualizar sua hierarquia, a não ser mentalmente.

No ibuild é possível editar elementos internos à hierarquia de um diálogo. Para isso, seleciona-se a opção `Narrow` do menu, e em seguida, seleciona-se um elemento de interface do diálogo pertencente ao nível que se deseja visualizar. É então apresentada uma lista contendo todos os elementos do diálogo que estão na hierarquia diretamente acima do elemento selecionado. O nível de hierarquia que se deseja visualizar é selecionado desta lista. No entanto, nem sempre é simples saber, pela lista, o nível de hierarquia que se deseja visualizar. Apesar de os elementos estarem listados em ordem ascendente do elemento selecionado até o primeiro nível, a lista só contém o tipo do elemento e não a sua instância. Uma vez selecionado o nível, apenas os elementos abaixo dele na hierarquia continuam sendo visíveis. Os elementos filhos do grupo selecionado passam a poder ser editados como se nunca tivessem sido agrupados. No entanto, ao se visualizar apenas um nível interno à hierarquia do diálogo, perde-se a noção do diálogo como um todo. Para evitar que se perca esta noção do diálogo, o ibuild permite que se crie e trabalhe em várias vistas simultaneamente.

Uma vista é criada a partir de outra e inicialmente mostra exatamente o mesmo que a vista original (Figura 5.3). Para fazer edição em um nível na hierarquia e continuar tendo uma visão global do diálogo, basta criar uma nova vista a partir da original contendo todo o diálogo e editar a hierarquia apenas em uma das vistas (Figura 5.4). No entanto, esta solução apresenta outros problemas. Uma modificação feita em uma vista é refletida em todas as outras, o que pode gerar diálogos incoerentes em alguns casos. Por exemplo, considere o diálogo mostrado na Figura 5.4. Em uma das vistas é mostrado o diálogo total, contendo uma `vbox`, que por sua vez contém vários filhos. Um desses filhos é uma `hbox` contendo um `toggle Male`, outro `toggle Female` e um `hfill`. Quando movemos o `toggle Female` (primeira vista da Figura 5.5), a operação é refletida na vista contendo todo o diálogo, gerando então um diálogo incoerente (segunda vista da Figura 5.5). É possível salvar arquivos contendo interfaces incoerentes. A princípio, estas situações são de transição de uma situação válida para outra, mas mesmo assim são indesejáveis.

O ibuild gera três tipos de arquivos: arquivos de descrição da interface, arquivos para se gerar o protótipo da interface, e arquivos para fazerem a ligação entre a interface e a tecnologia da aplicação. Antes de se implementar todo o sistema, pode-se testar o protótipo da interface. Para gerar o protótipo da interface, o arquivo criado pelo ibuild deve ser compilado, e o executável gerado, o que é feito pelo próprio ibuild. Para fazer a ligação da interface com a tecnologia da aplicação, um dos arquivos gerados pelo ibuild deve ser modificado. No ibuild, como no Guide, se for necessário modificar a interface, o usuário deve tomar cuidado para não gerar um novo arquivo sobre o modificado, perdendo assim o trabalho feito.

Uma característica interessante e prática do ibuild é que um elemento composto pelo usuário pode tornar-se um elemento de interface e ser adicionado do menu em tempo de execução. Além disso, também em tempo de execução, o usuário pode adicionar ou retirar elementos de interface básicos ao menu de primitivas, permitindo assim que ele customize a interface do ibuild de acordo com as suas necessidades.

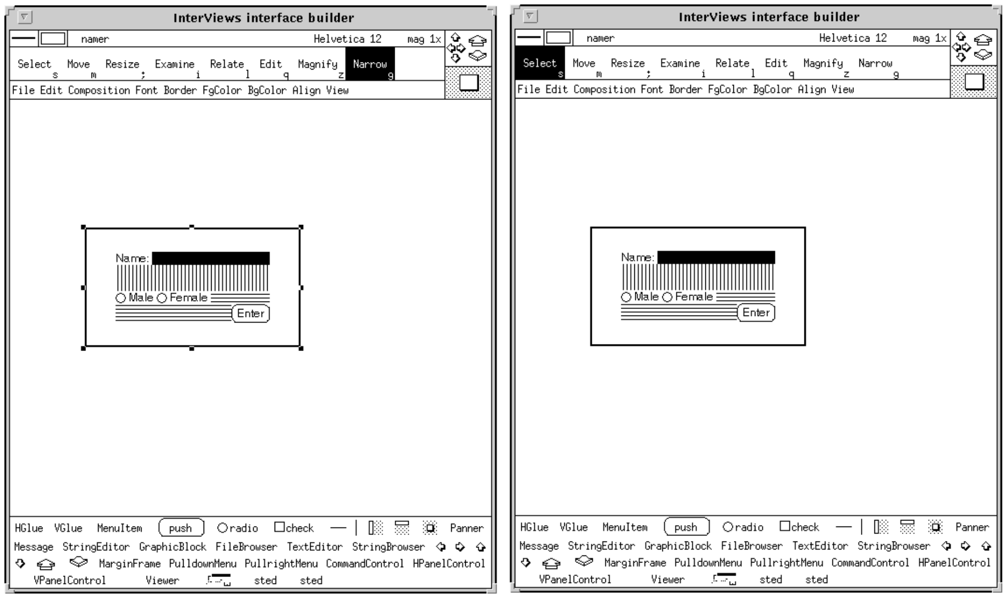

Figura 5.3: Vista original e a segunda vista criada a partir dela.

5.5 FormsEdit

O FormsEdit [ABB89, DEC89] é um editor gráfico de interfaces de *layout* abstrato. Este editor é composto por três vistas, sendo que duas destas são editáveis, e a outra usada apenas para a visualização do resultado. Cada uma das duas vistas editáveis contém uma representação do diálogo. Uma vista apresenta uma representação textual (em LISP), enquanto a outra apresenta uma representação gráfica. O usuário pode trabalhar na vista que desejar a qualquer momento, e as modificações são refletidas nas outras duas vistas.

O modelo de *layout* adotado segue o paradigma de *boxes-and-glue* do TEX (seção 2.2.3). Assim sendo, os diálogos são compostos pelo agrupamento de elementos primitivos. Os principais elementos de composição são `fill`, `glue` e `bar` (onde `fill` são espaços incolores expansíveis, `glue` são espaços incolores de tamanho fixo e `bar` espaços negros de tamanho fixo).

Na vista textual, é apresentada ao usuário uma descrição da interface em LISP. O usuário pode trabalhar nesta vista, mas após feitas as modificações desejadas, ele deve solicitar um `reparse` para que as outras vistas sejam atualizadas.

Na vista gráfica, o usuário manipula diretamente os elementos de interface. Todos os elementos de interface são visíveis nesta vista, até mesmo aqueles que não têm representação no resultado, como `hbox`, `vbox` e `fill`. Os elementos primitivos são representados por textos ou desenhos e, sempre, por sua *bounding box*. Elementos de composição e de agrupamento são representados por uma borda em torno de seus filhos. Esta representação possibilita a visualização de toda a hierarquia do diálogo. No entanto, nem sempre esta visualização é

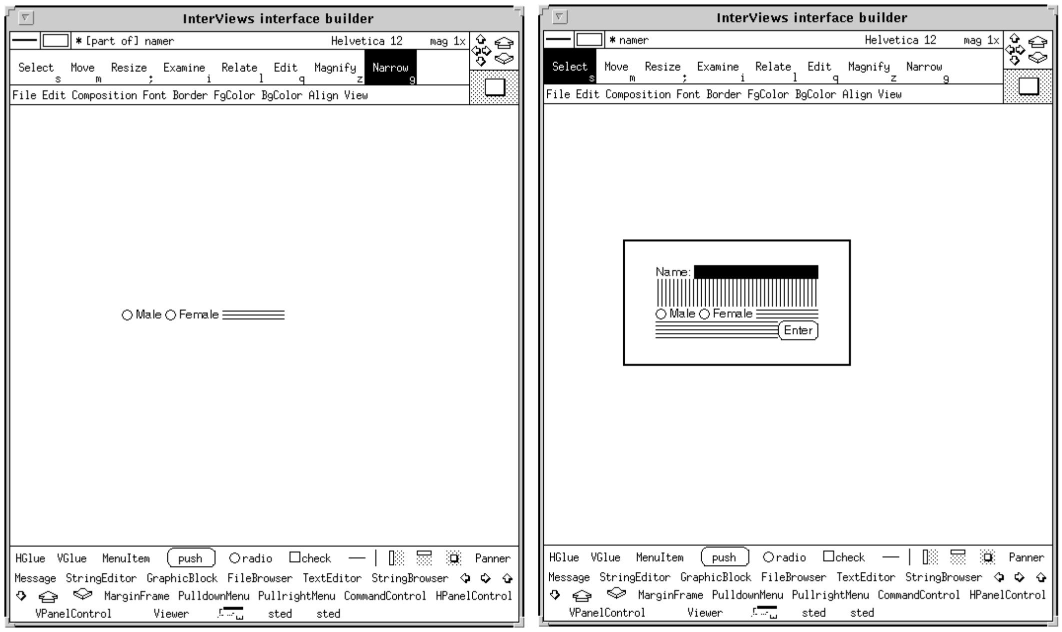

Figura 5.4: Vista com edição da hierarquia.

imediata: quando se tem vários níveis de profundidade, e portanto várias bordas, pode ser difícil distinguir a borda do elemento desejado das bordas dos outros elementos.

O FormsEdit permite que se faça edição em profundidade nos grupos. O usuário pode então selecionar, criar, apagar, inserir, retirar, substituir e trocar atributos tanto dos grupos como dos filhos de um grupo. Para selecionar um elemento, basta pressionar o botão do *mouse* sobre o elemento desejado. O elemento selecionado é o mais profundo na hierarquia naquela posição. É possível ainda selecionar um elemento pai sem selecionar seus filhos, ou selecionar mais de um elemento de uma vez.

Um elemento pode ser inserido em qualquer nível da hierarquia. Para inserir um elemento entre dois elementos já existentes, basta posicioná-lo sobre o espaço entre os outros dois. Como alguns elementos podem suportar apenas um número restrito de filhos, a inserção só é efetuada se ela for válida. Se o novo elemento for posicionado sobre um elemento já existente, o elemento já existente será substituído pelo novo. O *feedback* da operação sendo executada é indicada pela forma do cursor.

Para se agrupar elementos, cria-se uma `hbox` ou `vbox` sobre um ou mais elementos. Estes elementos passam a ser filhos da `hbox` ou `vbox`. A ordem dos elementos no grupo já está definida, pois é a mesma ordem dos elementos no diálogo.

A vista resultante possui o diálogo com seu *look-and-feel* final. Na verdade, este diálogo pode ter algumas diferenças do diálogo resultante, pois alguns elementos só são definidos em tempo de execução (por exemplo, elementos de *browsers*). O usuário pode testar como o diálogo reage a modificações de tamanho ou à interação com seus elementos. No entanto,

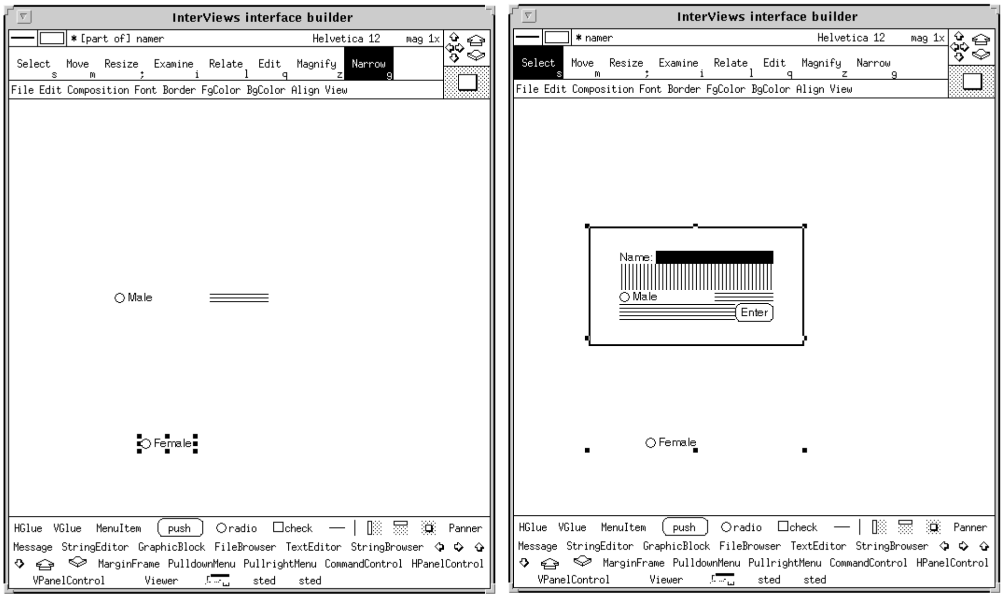

Figura 5.5: Vista com edição da hierarquia e vista original incoerente.

ao interagir com o diálogo na vista resultante, podem aparecer algumas mensagens de erro que não apareceriam em tempo de execução, uma vez que a tecnologia da aplicação não está ativa.

Como já foi dito, o usuário pode trabalhar com as três vistas simultaneamente, alternando a edição entre as vistas textual e gráfica conforme lhe pareça mais adequado. No entanto, é importante ressaltar que a vista textual é considerada o editor primário. Existem, portanto, algumas operações, como carregar, salvar ou trocar o nome de arquivos, que não podem ser feitas no editor gráfico, mas apenas no editor textual. Além disso, ao iniciar o FormsEdit, apenas o editor textual é mostrado; as outras duas vistas apenas aparecem após um `reparse`.

5.6 Análise crítica

Nesta seção, fazemos uma comparação entre os editores gráficos de interface estudados e discutidos neste capítulo.

Os editores gráficos de interface estudados podem ser classificados de acordo com o paradigma de *layout* da interface: concreto ou abstrato. O Guide, o Visual Basic e o OPUS adotam o *layout* concreto, enquanto que o ibuild e o FormsEdit adotam o *layout* abstrato. Na verdade, o OPUS pode ser classificado em uma classe intermediária, pois, apesar de usar *layout* concreto, ele permite a abstração do *layout* através dos elementos abstratos que ele oferece (`frame`, linhas de referência e restrições).

Todos os editores estudados permitem a construção da interface por manipulação direta, independente do tipo de *layout* adotado. O FormsEdit, além de permitir a edição por manipulação direta, permite também a edição da descrição textual da interface. Combinando as duas formas de edição, o FormsEdit dá mais liberdade de trabalho ao usuário. O uso de múltiplas vistas editáveis foi iniciado com o editor de documentos Lilac [Bro91].

A seleção de elementos nos editores pode ser de um ou mais elementos simultaneamente. Quando uma operação é requisitada, ela incide sobre todos os elementos selecionados. Nos editores de *layout* concreto Guide e OPUS, o usuário pode criar grupos para facilitar a manipulação e operações sobre os elementos. O Visual Basic não fornece a operação de agrupamento. O OPUS permite que o usuário faça edição em profundidade no grupo, enquanto que o Guide sempre trata um grupo como sendo uma única primitiva.

A operação de agrupamento é fundamental nos editores de interface de *layout* abstrato, pois é a base da composição dos diálogos. Tanto o ibuild quanto o FormsEdit permitem a edição na hierarquia. No entanto, a política de edição em profundidade do FormsEdit é bem mais simples e intuitiva que a do ibuild. Além disso, a política de edição em profundidade do ibuild permite que se tenha diálogos incoerentes, o que não acontece no FormsEdit.

Como os diálogos são compostos por agrupamento, é importante que o usuário possa visualizar a hierarquia do diálogo. O FormsEdit representa todos os elementos de interface, mesmo os que não são visíveis no diálogo; logo, sua hierarquia é visível. Como `hbox`'s e `vbox`'s não são visíveis no ibuild, a hierarquia não é explicitamente representada.

O OPUS possui duas estratégias para o cálculo do tamanho final da interface: *inside-out* e *outside-in*. No ibuild e no FormsEdit, a estratégia utilizada normalmente é uma combinação das duas. Os elementos primitivos têm seus tamanhos naturais; os grupos têm seus tamanhos calculados a partir dos tamanhos dos seus filhos. As únicas exceções são os elementos expansíveis, cujos tamanhos são especificados de acordo com o tamanho do diálogo e podem ser alterados dinamicamente. Nos editores de *layout* concreto, os tamanhos dos elementos e do diálogo são independentes entre si, e seus tamanhos são especificados pelo usuário e não calculados pelo sistema.

Todos os editores estudados possuem uma forma de prototipação rápida da interface. Esta característica é muito importante, pois só assim o usuário conseguirá visualizar o diálogo que ele está construindo. O OPUS e o FormsEdit têm uma vista que mostra o *look-and-feel* final da interface, ou seja, eles possuem um protótipo ativo durante toda a execução do editor. No Guide, Visual Basic e ibuild, o usuário deve requisitar a prototipação, e o editor sai do modo de edição e passa para o modo de prototipação. Nestes editores, os protótipos gerados têm exatamente o mesmo *look-and-feel* do diálogo. Por outro lado, o OPUS e o FormsEdit apresentam um protótipo bem aproximado do diálogo resultante, mas não idêntico. No OPUS, os elementos de interface abstratos são visíveis no protótipo; no FormsEdit, alguns elementos só são definidos em tempo de execução.

A partir deste estudo, identificamos os problemas existentes nos editores gráficos de interface e as vantagens e desvantagens das técnicas usadas em cada um dos editores estudados. Projetamos então a interface e a interação do Visual LED, tentando resolver os problemas existentes e combinar as vantagens dos diversos editores.

No próximo capítulo, apresentaremos e discutiremos o Visual LED.

5.7 Taxonomia

Nesta seção, sugerimos uma taxonomia para editores gráficos de interface, baseada no estudo apresentado.

Para compararmos os diversos editores, selecionamos algumas características que consideramos relevantes para o desenvolvimento do Visual LED. Apresentamos, a seguir, os ítens selecionados e as opções para cada um deles.

A primeira característica a ser considerada é o paradigma de *layout* adotado, que pode ser abstrato ou concreto. Consideramos o *layout* abstrato melhor que o concreto, uma vez que é mais simples fazer modificações na interface e a criação dos diálogos também é simples (ver seção 2.1).

Para avaliar a interação do editor com o usuário, comparamos a forma de manipulação dos elementos, a existência ou não de agrupamento, e no caso da sua existência, a política de agrupamento adotada, e, finalmente, a possibilidade de percorrer e visualizar a hierarquia.

A manipulação dos elementos pode ser feita textualmente ou por manipulação direta. A manipulação direta é mais simples e intuitiva para o usuário, sendo então preferida à forma textual.

É desejável que o editor tenha agrupamento. Além de permitir a criação de objetos compostos, esta operação ainda facilita a manipulação de objetos. Quando o agrupamento está disponível, avaliamos a política de agrupamento adotada. A visualização da hierarquia criada e a possibilidade de percorrê-la são características muito desejáveis e foram consideradas.

Ao construir uma inteface, é importante que o usuário possa visualizar a interface com a sua aparência final e saber como ela vai funcionar. Comparamos então a facilidade de se gerar um protótipo da interface e a sua semelhança com o *look-and-feel* final do diálogo.

Os outros itens usados na comparação dos editores foram a possibilidade de se programar as funções de *callback* dos elementos de interface no próprio editor e a portabilidade dos diálogos gerados.

A Tabela 5.1 resume a taxonomia que estamos sugerindo, contendo os ítens de classificação e as opções de cada um deles. A Tabela 5.2 contém a classificação de cada um dos editores estudados de acordo com esta taxonomia.

Esta taxonomia foi construída originalmente tendo para o desenvolvimento do Visual LED. No entanto, a taxonomia é geral e pode ser usada para classificar e comparar outros editores gráficos de interface.

Item de comparação	Opções
paradigma de *layout* adotado	concreto ou abstrato
posicionamento de elementos	manipulação direta ou textual
existência de agrupamento	sim ou não
percorrimento da hierarquia	possível ou não
visualização da hierarquia	possível ou não
política de agrupamento	política simples ou outra
protótipo	precisa ser compilado ou ativo
look-and-feel do protótipo	fiel ou aproximado
funções de *callback*	existente ou não
portabilidade dos diálogos	sim ou não

Tabela 5.1: Taxonomia para editores gráficos de interface.

Item de comparação	Guide	Visual Basic	OPUS	Ibuild	FormsEdit
paradigma de *layout*	concreto	concreto	intermediário	abstrato	abstrato
posicionamento de elementos	direta	direta	direta	direta	ambos
existência de agrupamento	sim	não	sim	sim	sim
percorrimento da hierarquia	não	—	possível	possível	possível
visualização da hierarquia	—	—	possível	não	possível
política de agrupamento	simples	—	outra	simples	hierárquica
protótipo	ativo	ativo	ativo	compilado	ativo
look-and-feel do protótipo	fiel	aprox.	aprox.	fiel	aprox.
funções de *callback*	não	existente	não	não	não
portabilidade dos diálogos	não	não	não	não	não

Tabela 5.2: Classificação dos editores gráficos de interface na taxonomia.

Capítulo 6

Visual LED

Neste capítulo, apresentamos Visual LED, um editor gráfico de interfaces com *layout* abstrato que lê e gera arquivos em LED. O Visual LED foi projetado com os seguintes objetivos:

- facilitar a geração de uma descrição em LED para uma interface;
- permitir a visualização da hierarquia da interface pelo projetista durante o processo de construção da mesma;
- permitir a visualização da interface pelo projetista durante o processo de construção da mesma;
- combinar os métodos de edição de interface textual e gráfico para tornar mais eficiente a criação das interfaces;
- fornecer uma interação simples e intuitiva ao usuário.

O principal objetivo de Visual LED é facilitar a geração de uma descrição em LED de uma interface. Para isso, o Visual LED deve ser tão portátil quanto o sistema IUP/LED, permitindo assim que o usuário trabalhe no ambiente no qual ele já está acostumado. Como o IUP/LED é multi-plataforma e o Visual LED é um aplicativo do sistema IUP/LED, o Visual LED também é multi-plataforma. A arquitetura do Visual LED pode ser vista na Figura 6.1.

Como a principal dificuldade encontrada no sistema IUP/LED atacada aqui é a transferência da interface imaginada para sua descrição em LED, o Visual LED tem como um de seus principais objetivos permitir que a descrição seja feita visualmente, por manipulação direta dos elementos de interface. Simultaneamente a esta edição dos elementos do diálogo, o usuário pode testar o *look-and-feel* real deste diálogo. Para dar maior liberdade ao usuário e facilitar ainda mais a geração da descrição LED, o Visual LED combina a edição gráfica por manipulação direta com a edição da descrição textual. O Visual LED é então um editor de interfaces LED composto por três vistas: uma vista textual, contendo a descrição em LED

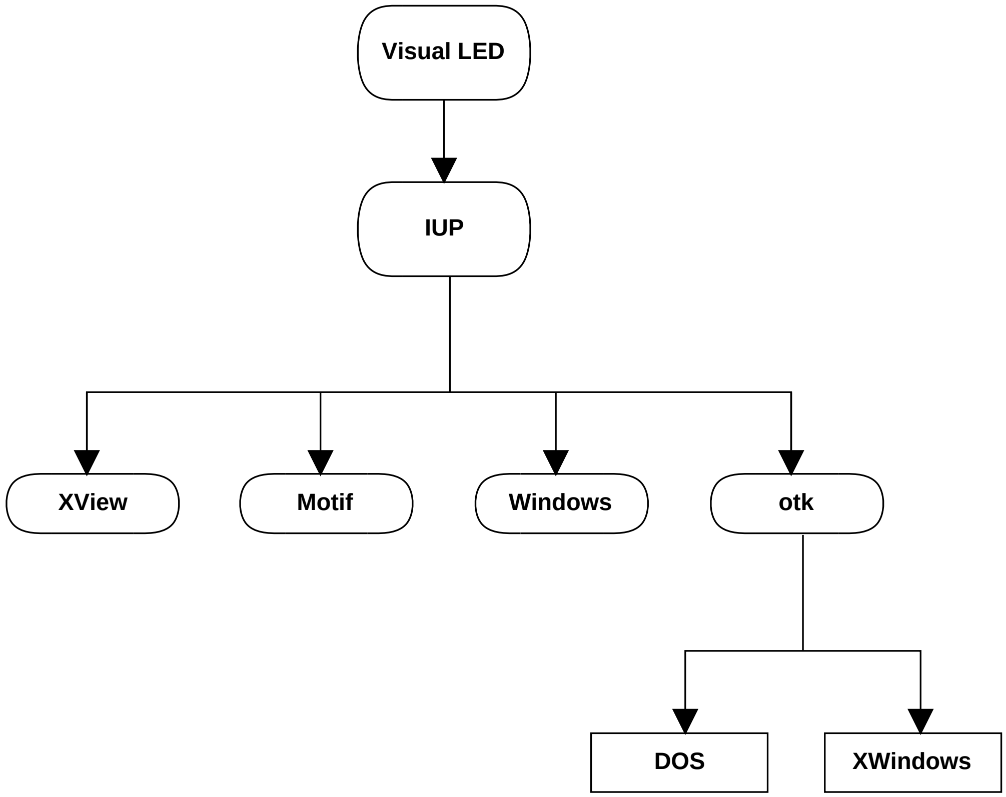

Figura 6.1: Arquitetura do Visual LED.

do diálogo sendo editado; uma vista contendo o *layout* abstrato, ou seja, uma representação gráfica editável do diálogo; e finalmente uma vista contendo o resultado final ou o *layout* concreto do diálogo. A idéia de se ter várias vistas e representações não é inovadora e já foi utilizada tanto em editores gráficos de inteface (exemplo: FormsEdit) [ABB89], quanto em editores de documentos [Bro91] e sistemas operacionais [BHC94].

Nesta capítulo, não vamos entrar em detalhes sobre o modo como foi feita a implementação do Visual LED, vamos apenas explicar algumas decisões globais que foram tomadas,

A estrutura de dados utilizada é a do sistema IUP/LED. Como foi visto na seção 3.5, cada diálogo no sistema IUP/LED é representado por uma árvore. O IUP/LED armazena então um conjunto de árvores. Todas as operações feitas em Visual LED são feitas nestas árvores.

Para evitar a duplicação do código e evitar que o Visual LED acessasse os *drivers* (Figura 6.3), alguns cálculos necessários para o Visual LED e para o IUP/LED são feitos pelo IUP/LED e apenas consultados pelo Visual LED. Mais especificamente, o cálculo do tamanho dos elementos de interface e a sua aparência com decoração é de responsabilidade do IUP/LED.

Ao implementar o Visual LED, buscamos criar uma interação simples e intuitiva que facilitasse ao máximo o trabalho do usuário. Um ponto crítico na interação, como já foi mencionado, é como permitir uma edição em profundidade.

Na seção 4.2, discutimos o problema de edição em profundidade e apresentamos a solução da árvore para os editores gráficos genéricos e de interfaces de *layout* concreto. No entanto, concluímos que esta solução não era adequada para os editores gráficos de interfaces de *layout* abstrato. Em particular, no caso do Visual LED, o único benefício desta solução seria a visualização da topologia da árvore, pois a informação contida na árvore seria a mesma contida no arquivo LED apresentado na vista textual, e, logo, redundante. No entanto, como a árvore já está representada textual e geometricamente no Visual LED, a visualização da sua topologia não é fundamental para o trabalho do usuário.

Buscando uma solução apropriada para o Visual LED, estudamos a edição em hierarquia, não só em editores gráficos genéricos, mas também em editores gráficos de interface. Criamos então a **política hierárquica**, que será apresentada na seção 6.3.4.

6.1 As três vistas

O Visual LED é composto pelas três vistas apresentadas acima: vista textual, vista com *layout* abstrato, vista com *layout* concreto.

Cada uma destas vistas tem um objetivo diferente. Inicialmente, as três vistas são apresentadas simultaneamente, mas o usuário não precisa trabalhar com todas elas se não desejar. O Visual LED fornece as três vistas para dar liberdade de trabalho para o usuário, e ao mesmo tempo permitir que ele veja e teste o *look-and-feel* do diálogo sendo criado.

Quando as duas vistas de edição estão abertas simultaneamente, o usuário pode trabalhar na vista que ele preferir a cada momento. Qualquer modificação feita na vista de *layout* abstrato, ou seja, na vista contendo o editor gráfico, é automaticamente atualizada nas outras duas vistas. As modificações feitas na vista textual porém, só refletem nas outras duas vistas sob demanda. Esta diferença entre a atualização das duas vistas acontece porque, durante a edição, a vista textual passa por estados no qual a descrição LED não é válida (por exemplo, por que os parênteses não estão corretamente casados). A vista do editor gráfico nunca passa por uma situação inválida, pois todo o elemento pertence a um diálogo a partir do momento em que é criado. Uma vez terminada uma etapa da edição na descrição LED do diálogo, o usuário deve requisitar a atualização das demais vistas.

A terceira vista contém o *layout* concreto do diálogo, ou seja, seu resultado. Esta é a única vista que é passiva, o que significa que ela não pode ser editada; entretanto, o diálogo contido nesta vista pode ser testado.

A seguir apresentamos, detalhadamente, cada uma das três vistas.

6.2 A vista textual

A vista textual contém a descrição em LED dos diálogos sendo criados. Esta descrição pode ser editada a qualquer momento. Para isso, são oferecidos ao usuário os comandos geralmente encontrados em um editor de texto.

Os objetivos de se apresentar a descrição textual do diálogo são:

- possibilitar modificações no diálogo via LED;
- facilitar o entendimento do *layout* do diálogo por parte do usuário.

Possibilitando que modificações no diálogo sejam feitas através da sua descrição em LED, permitimos que o usuário trabalhe na linguagem toda vez que esta opção lhe parecer mais simples ou natural. Por exemplo, pode ser mais fácil editar os atributos na vista textual que na vista gráfica. Assim, o Visual LED não apresenta nenhuma desvantagem em relação a trabalhar direto em LED, o que ocorreria caso o Visual LED tivesse apenas o editor gráfico e o resultado.

Apresentando a descrição LED, o Visual LED permite ainda que o usuário que saiba LED tire suas dúvidas em relação ao comportamento do diálogo. Quando o usuário faz uma modificação na vista do editor gráfico, e o diálogo não se comporta da forma esperada, ele pode recorrer à descrição em LED para descobrir exatamente qual modificação foi feita. Por outro lado, um usuário que não sabe LED pode aprendê-la mais facilmente trabalhando no Visual LED. Ele pode criar diálogos simples na vista do editor gráfico e ver suas respectivas descrições, ou criar descrições em LED e ver sua representação gráfica e resultado.

6.3 A vista do editor gráfico

A vista contendo o editor gráfico apresenta uma representação gráfica do *layout* abstrato do diálogo. Nesta vista, *todos* os elementos são representados, inclusive aqueles que não são visíveis na interface final, como `hbox`, `vbox`, `radio` e `fill`. A vista do editor gráfico pode ser classificada como sendo intermediária entre a vista textual e a do resultado, pois, nesta vista, todos os elementos existentes em LED estão representados, e a aparência do diálogo é próxima à aparência do diálogo na vista do resultado. Podemos dizer que a vista gráfica é uma representação bidimensional de uma descrição unidimensional.

6.3.1 Justificativa da existência da vista gráfica

Uma vez que a vista do editor gráfico é intermediária, surge a questão de existir a necessidade de apresentarmos esta vista. Não seria suficiente apresentar apenas a vista textual e a vista do resultado, sendo esta última atualizada automaticamente a cada modificação válida na descrição LED? Não seriam estas duas vistas suficientes para o usuário visualizar a interface sendo construída? De fato, já é possível visualizar a interface com a descrição em LED e o resultado sendo mostrados simultaneamente. No entanto, transferir a interface que se tem em mente para uma descrição em LED continua sendo a principal dificuldade. Com um editor gráfico por manipulação direta, esta dificuldade é bem amenizada. O projetista não precisa transformar a interface que tem em mente em uma descrição textual: basta montá-la utilizando diretamente os elementos oferecidos.

6.3.2 Representação dos elementos de interface

Os diversos tipos de elementos de LED têm representações gráficas distintas. No entanto, todas estas representações são sem decoração. Os elementos primitivos `button`, `toggle`, `label`, `list` e `text` são representados por caixas de borda preta e fundo cinza claro. O `canvas` é uma caixa de borda preta também, mas de fundo cinza escuro. Esta distinção é feita porque o `canvas` é um elemento expansível e os outros não. O `fill` também é diferenciado, pois, além de expansível, ele nem sempre tem representação na interface final. Optamos por representar `fill` por uma caixa hachurada na diagonal, uma vez que ele pode se expandir tanto na horizontal quanto na vertical. Todos estes elementos têm um título, composto pelo tipo do elemento e um número que representa a sua ordem de criação. Este título é na verdade o atributo `TITLE` do IUP; no entanto, este atributo não é identificado e interpretado pelo sistema IUP/LED para todos os elementos de interface. Assim, o título mostrado na vista gráfica só aparece na interface final para os elementos para os quais o IUP/LED reconhece o título como atributo.

Os elementos que possuem filhos (`hbox`, `vbox`, `frame` e `radio`) são representados por bordas em torno de seus filhos. Os `hbox`'s são representados por bordas azuis; os `vbox`'s por bordas verdes. Estes elementos de agrupamento possuem na sua borda setas (opcionais) indicando a sua direção (vertical ou horizontal). Os `frame`'s são representados por bordas pretas; os `radio`'s, por bordas pretas e pontilhadas.

Uma distinção importante a ser feita é que mesmo os elementos de tamanho natural zero no *layout* concreto devem ter um tamanho mínimo na representação do *layout* abstrato, para torná-los passíveis de manipulação direta.

O único elemento que não é representado explicitamente no *layout* abstrato é o diálogo. Como o diálogo sempre existe e só tem um filho, a representação deste filho também o representa. Cada grupo de elementos conexos apresentados na vista gráfica é um diálogo.

Na Figura 6.2, é mostrada a representação dos elementos do diálogo mostrado na Figura 3.3 na vista gráfica.

Não seria melhor representar os elementos com decoração, ao invés de representá-los desta forma mais simples? Apesar de os usuários já serem familiarizados com a representação decorada dos elementos, alguns motivos nos levaram a optar por essa representação mais simples. Primeiramente, como nem todos os elementos representados aqui têm uma representação no resultado, não existe uma representação decorada para todos. Seria necessário então criar alguma representação para aqueles elementos que ainda não tivessem uma. Teríamos então,

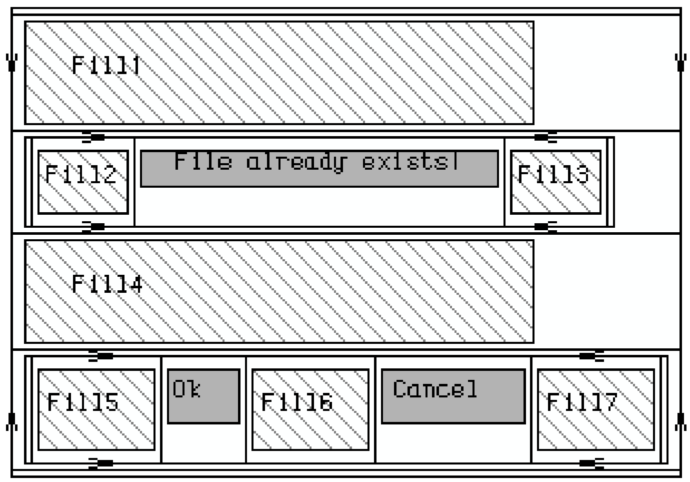

Figura 6.2: Representação de alguns elementos de interface na vista gráfica.

em uma mesma vista, elementos com uma representação decorada igual à sua representação final, enquanto outros teriam uma representação na vista gráfica que não teria qualquer correspondência na representação final, o que poderia ser confuso para o usuário. Além da dificuldade da representação de alguns elementos, para apresentarmos os objetos com decoração seria necessário que o Visual LED fizesse chamadas diretas ao *driver* e sua estrutura passaria da arquitetura mostrada anteriormente (Figura 6.1), com chamadas apenas para o IUP, para uma nova arquitetura que acessaria o driver (Figura 6.3). Com isso, o Visual LED perderia sua portabilidade.

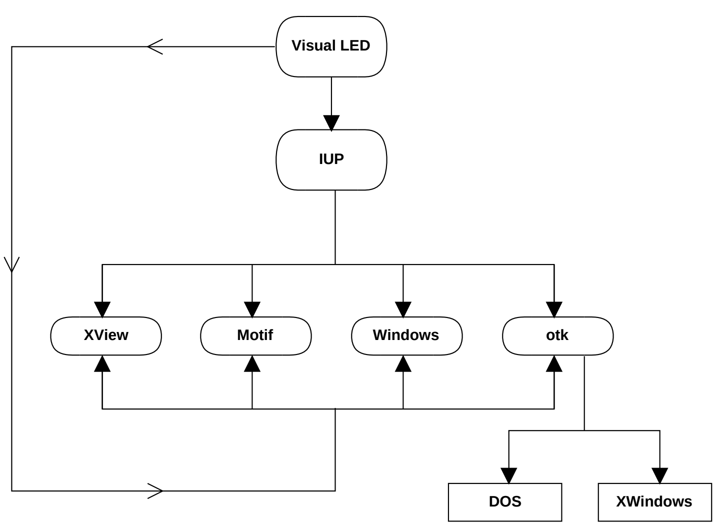

Figura 6.3: Arquitetura do Visual LED acessando *drivers*.

A solução seria então utilizar o *driver* `otk` que é portátil e manteria o *look-and-feel* fixo do *layout* abstrato. No entanto, é importante lembrar que o usuário pode optar por ter um *look-and-feel* fixo ou nativo nas diversas plataformas. Se optássemos pela decoração fixa do `otk`, a vista resultante sempre seria mostrada com o *look-and-feel* fixo. Isto poderia ser ainda mais confuso quando o usuário desejasse usar o *look-and-feel* nativo na sua aplicação. Neste caso, o diálogo resultante apresentado pelo Visual LED teria uma aparência completamente diferente deste mesmo diálogo durante a execução da aplicação.

6.3.3 Representação gráfica da hierarquia da árvore

É importante ressaltar que toda a informação contida na descrição LED também está contida na vista gráfica. A árvore montada em LED é visível na vista do *layout* abstrato, pois a hierarquia da árvore sempre corresponde à geometria do diálogo. Sendo assim, todas as relações que observamos entre os nós da árvore têm uma correspondente no *layout* abstrato e no concreto, como mostra a Tabela 6.1.

Árvore	*Layout* abstrato e concreto
A é irmão da esquerda de B	A está à esquerda ou acima de B
A é irmão da direita de B	A está à direita ou abaixo de B
A é pai de B	A contém B
A tem n filhos	A contém n elementos
a árvore tem x níveis de profundidade	o diálogo possui pelo menos um elemento que está contido em $(x-1)$ elementos.

Tabela 6.1: Correspondência entre a árvore e os *layouts* abstrato e concreto.

Todos os casos citados na Tabela 6.1 são facilmente visíveis no *layout* abstrato, uma vez que *todos* os elementos da árvore, e não somente os elementos primitivos, estão representados. No *layout* concreto, nem sempre é tão fácil ver estas correspondências, já que alguns elementos não têm representação, como `hbox`, `vbox` e `radio`. A Figura 6.4 mostra uma árvore e os *layouts* abstrato e concreto correspondentes. Observe as seguintes correspondências nesta figura: o `Botão A` é irmão da direita do `Botão B` e, logo, o `Botão B` é irmão da esquerda do `Botão A`; `vbox` possui dois filhos, `Botão A` e `Botão B`; `frame` é pai de `vbox`; e finalmente a árvore possui quatro níveis de profundidade.

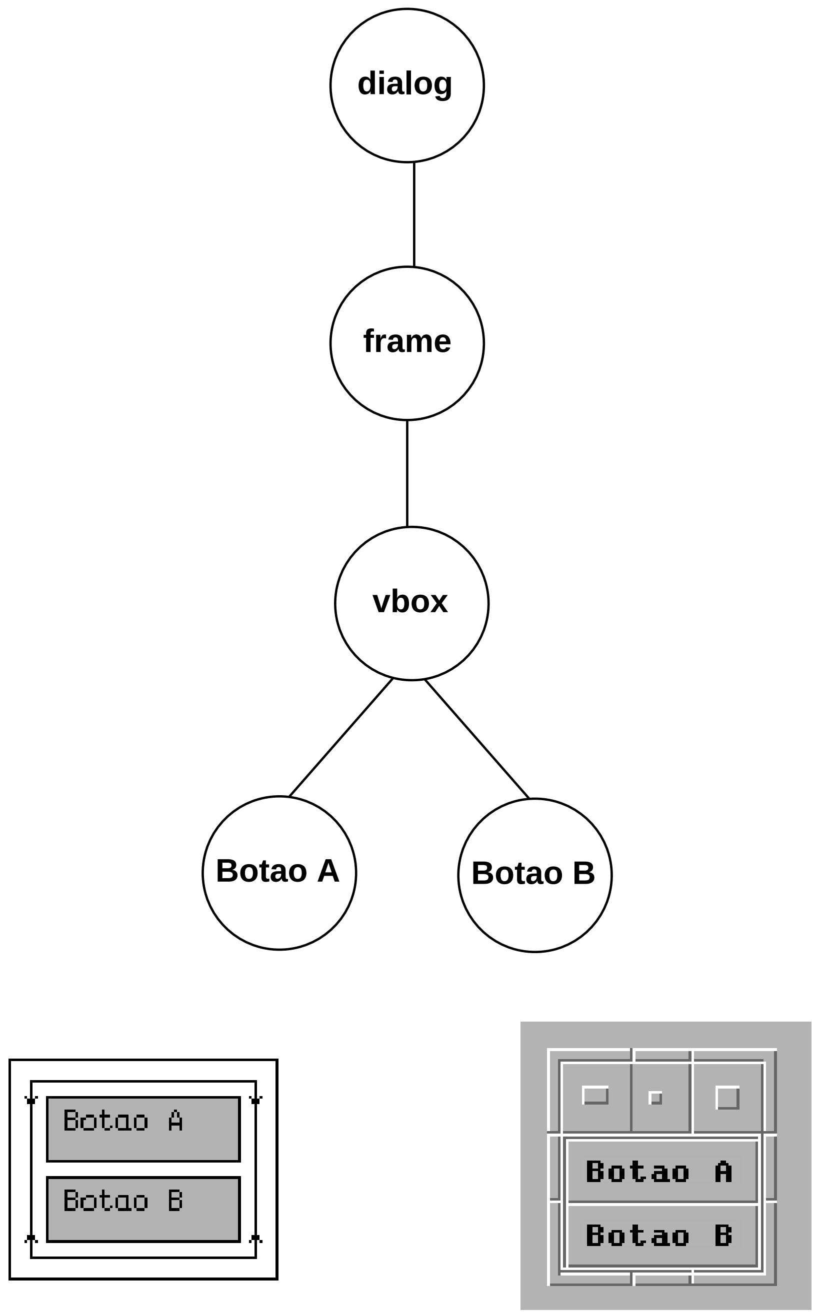

Figura 6.4: Árvore e os *layouts* abstrato e concreto correspondentes.

6.3.4 Política hierárquica

Nesta seção, apresentamos a **política hierárquica**, que criamos para possibilitar a edição em profundidade pelo usuário. Esta política permite que o usuário acesse e trabalhe em qualquer nível da hierarquia da árvore.

Seleção

Quando o usuário requisita uma operação, ela incide sobre os elementos que estão selecionados no momento. O Visual LED permite que o usuário selecione um ou mais elementos. Para selecionar um elemento, basta que o usuário entre no modo de seleção e pressione o botão da esquerda do *mouse*, quando o cursor estiver sobre o elemento desejado. No entanto, pode-se ter vários elementos em uma mesma posição: um elemento primitivo e todos os seus ancestrais diretos na hierarquia. O Visual LED possui duas formas distintas para decidir qual dos elementos em uma determinada posição será selecionado: a **seleção descendente** e a **seleção ascendente**. Na seleção descendente, ao se pressionar o botão do *mouse* em uma posição, se nenhum elemento estiver selecionado, o elemento mais alto na hierarquia, naquela posição, é selecionado. Desta forma, é possível se percorrer a hierarquia a partir da raiz até se chegar aos elementos primitivos. Quando na posição selecionada já existe um elemento selecionado, se ele tiver um filho nesta posição, o elemento é de-selecionado e seu filho é selecionado. Para se selecionar o irmão de um elemento já selecionado, não é necessário se percorrer novamente a hierarquia. Na seleção ascendente, ao se pressionar o botão do *mouse*, se nenhum elemento está selecionado, o elemento mais profundo na hierarquia naquela posição é selecionado. Se o elemento mais profundo naquela posição já está selecionado, então seu pai é selecionado.

A seleção descendente tem a desvantagem de ser necessário pressionar o botão do *mouse* diversas vezes para atingir elementos muito profundos na hierarquia. Por outro lado, na seleção ascendente, quando a área de um elemento é pequena, pode ser difícil posicionar o cursor sobre ela e, conseqüentemente, selecionar o elemento desejado. Como o Visual LED permite as duas formas de seleção, o usuário pode usar a que for mais conveniente ou eficiente a cada momento.

Visual LED fornece duas opções para selecionar vários elementos de uma só vez. Na primeira, o usuário pressiona a tecla ⟨`Shift`⟩ ao selecionar o elemento. O elemento é então incluído na lista dos elementos selecionados. Caso se selecione com ⟨`Shift`⟩ um elemento já selecionado, ele é de-selecionado. A segunda maneira é selecionar todos os elementos desejados usando uma *fence* retangular. Pressiona-se o botão de seleção em uma posição que não tem elementos e move-se o *mouse* mantendo-se o botão pressionado. Surge então uma área retangular, que é a *fence*. Quando o botão é solto, são selecionados todos os elementos que estão no primeiro nível da hierarquia, e que estão totalmente dentro da *fence*. Só é possível selecionar de uma só vez elementos que estejam no primeiro nível da hierarquia ou que sejam irmãos. Deste modo, garantimos que apenas operações válidas sejam efetuadas. Caso contrário, o usuário poderia tentar agrupar componentes de grupos distintos.

Operações sobre elementos

Descrevemos, a seguir, como funciona cada uma das operações possíveis sobre os elementos na vista gráfica.

Criação: Os elementos primitivos sempre são criados no primeiro nível da hierarquia. Para criar um elemento primitivo, primeiro entra-se no modo de criação; em seguida, seleciona-se no menu o tipo de elemento desejado; ao se pressionar o botão da esquerda do *mouse* em uma posição qualquer da tela, o elemento é criado com seu canto superior esquerdo nesta posição. Para criar os elementos `radio` e `frame`, os elementos que deverão ser seus filhos são selecionados e então seleciona-se a opção `radio` ou `frame` no menu. É criado um `radio` ou `frame` para cada um dos elementos selecionados.

Quando um elemento é criado, também um diálogo, contendo este elemento como filho, é criado. Isto é feito porque o sistema IUP/LED só permite a criação de uma árvore que tenha um elemento `dialog` como raiz. Como o sistema IUP/LED é responsável pelo cálculo do tamanho dos elementos de interface e pela sua aparência com decoração, é preciso que o elemento pertença à árvore para que ele possa ser mostrado pelo Visual LED na vista do editor gráfico e pelo IUP/LED na vista do resultado.

Destruição: A operação de destruição de um elemento consiste apenas em selecionar os elementos selecionados e requisitar a operação de destruição. Os elementos que possuem apenas um filho (`dialog`, `frame`, e `radio`) são destruídos quando seu filho é destruído.

Agrupamento: O Visual LED fornece duas formas de se agrupar objetos. A primeira delas, baseada na política simples, segue a estratégia *bottom-up* sugerida por LED. Para agrupar seguindo esta estratégia, seleciona-se os elementos desejados na ordem desejada e requisita-se o agrupamento horizontal ou vertical. Um grupo será criado contendo os elementos selecionados dispostos na ordem selecionada como filhos. Por outro lado, é possível usar uma estratégia *top-down*. Neste caso, um grupo é criado como uma caixa vazia. O usuário então introduz nesta caixa os elementos desejados, na ordem desejada e no lugar desejado (ver o item *Inserção*).

Desagrupamento: Para desagrupar, basta selecionar os grupos desejados e requisitar um desagrupamento. Se o grupo em questão estiver no primeiro nível de hierarquia, então é criado um diálogo contendo cada filho do grupo. Caso contrário, seus filhos passam a ser filhos do pai do grupo, subindo então um nível na hierarquia. As Figuras 6.5 e 6.6 mostram o funcionamento do desagrupamento na árvore para um grupo no primeiro nível da hierarquia e para outro grupo mais profundo, respectivamente.

Inserção: Para inserir um elemento em uma posição de um grupo, é necessário selecionar o elemento a ser inserido e a posição no grupo onde este elemento deve ser inserido. Um elemento pode ser inserido entre dois elementos do grupo ou entre um elemento e a borda do grupo. Bastaria então colocar o elemento sobre a posição desejada e então inseri-lo nesta posição. No entanto, dependendo do tamanho do elemento sendo inserido, pode não ser

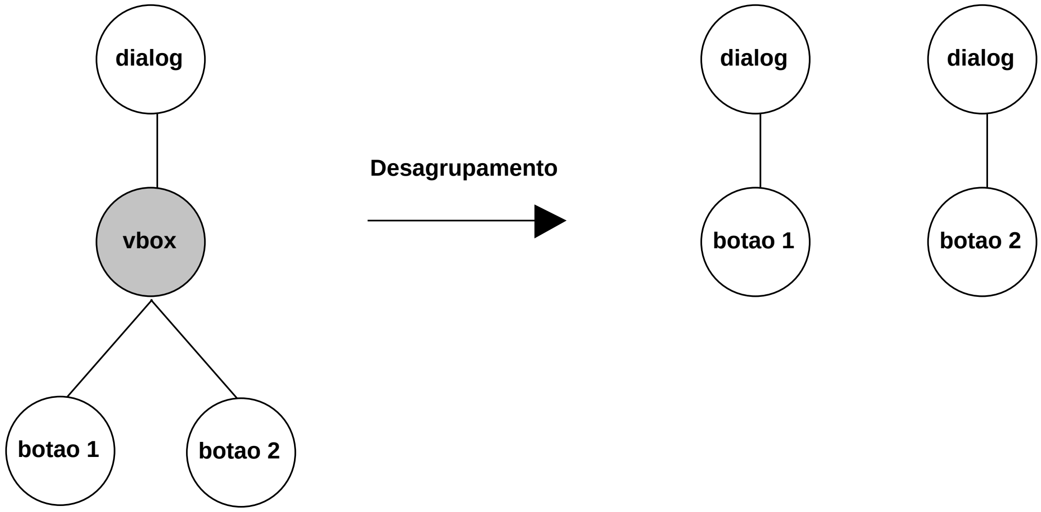

Figura 6.5: Desagrupamento de um elemento no primeiro nível da hierarquia.

possível colocar um elemento sobre uma única posição. Neste caso, seria necessário permitir ao usuário indicar em qual das posições sobrepostas pelo elemento ele deveria ser inserido.

Para indicar ao usuário a posição selecionada para a inserção em um grupo, criamos no Visual LED as **antenas**, arestas que aparecem no início e no fim do grupo, e entre os seus elementos. Quando um elemento é colocado sobre uma antena, a antena é ativada indicando a posição selecionada. Quando uma antena é ativada, ela troca de cor até que seja novamente desativada.

As duas próximas figuras mostram um exemplo de inserção usando antenas: a Figura 6.7 mostra o elemento `Novo Botão` fora do `hbox` e o `hbox` com suas antenas; a Figura 6.8 mostra o grupo após a inserção do `Novo Botão` na posição desejada.

A antena selecionada como ponto de inserção é aquela interceptada pelo elemento de interface a ser inserido. Se mais de uma antena interceptar o elemento de interface, então a antena mais próxima do seu canto superior esquerdo será selecionada. No caso de haver mais de um elemento de interface a ser inserido, a seleção da antena de inserção será feita dentre as antenas que interceptarem o elemento de maior prioridade interceptado, onde a prioridade é definida pela ordem de seleção (o primeiro elemento selecionado é o de maior prioridade). Entretanto, o usuário não está normalmente consciente destes detalhes: ele simplesmente "passeia" com o elemento a ser inserido sobre as antenas até que a antena correspondente à posição desejada seja ativada.

Portanto, no Visual LED, é possível inserir um novo elemento de interface em qualquer grupo em qualquer nível da hierarquia, desde que o grupo esteja **ativo**. Um grupo está ativo quando ele possui antenas. Quando se tem vários elementos selecionados, todos eles são sempre inseridos na mesma posição. Se um elemento do primeiro nível da hierarquia é inserido em um grupo, então o diálogo que o continha é destruído.

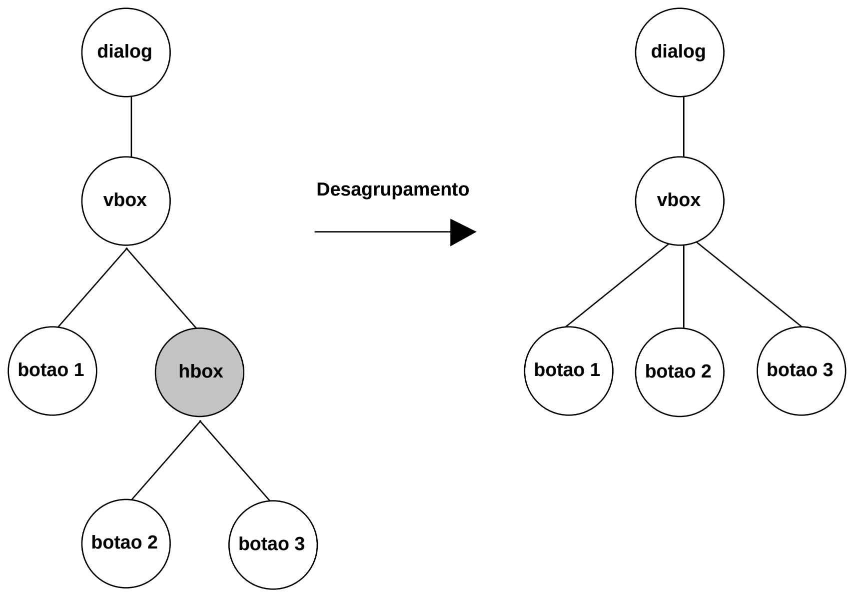

Figura 6.6: Desagrupamento de um elemento que não está no primeiro nível da hierarquia.

Remoção: Para retirar um elemento de um grupo, basta movê-lo para fora do grupo. Na verdade, o elemento não só é retirado do grupo, mas também do diálogo que contém o grupo, passando então a ser filho de um novo diálogo. Caso todos os filhos de um elemento sejam retirados, este elemento é destruído. Neste caso, a destruição se propaga recursivamente na árvore até que o pai de um elemento destruído tenha mais de um filho.

Translação: Para mover um elemento, entra-se no modo de mover, pressiona-se o *mouse* sobre um dos elementos selecionados e, com o botão pressionado, faz-se sua translação. Todos os elementos que estiverem selecionados serão movidos. Esta é a única operação na qual os elementos não precisam estar selecionados *a priori*. Se o botão do *mouse* for pressionado sobre um elemento não selecionado, então o elemento mais alto na hierarquia naquele ponto será selecionado e movido. Se outros elementos estiverem selecionados, eles serão de-selecionados.

Troca de atributos: A última operação oferecida é a troca de atributos de um elemento. O usuário pode fornecer os atributos que quiser; aqueles que forem relevantes para o tipo do elemento no sistema IUP/LED serão imediatamente redefinidos; os que não forem, não serão interpretados. Se vários elementos estiverem selecionados, todos eles terão os atributos definidos atualizados. Como o Visual LED é uma ferramenta do IUP/LED, tem-se o mecanismo de herança: se o atributo está definido para o pai, assim o está para seus filhos, a menos que os filhos o tenham declarado explicitamente. Quando esta operação é executada, a vista do *layout* abstrato não é automaticamente redesenhada, pois nem todos os atributos de um elemento modificam a sua aparência. A atualização da vista gráfica após uma troca de atributos deve ser requisitada pelo usuário.

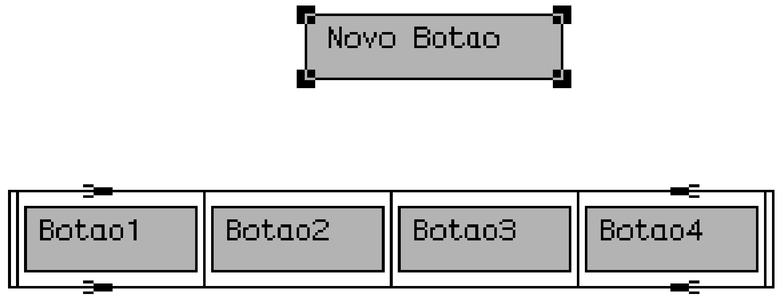

Figura 6.7: `Novo Botão` antes da sua inserção no `hbox`.

Figura 6.8: `Novo Botão` após a sua inserção.

6.4 A vista do resultado

Esta vista apresenta para o usuário a interface final, ou seja, a interface com o *look-and-feel* exatamente igual ao que ela terá na aplicação, quando a aplicação estiver sendo executada na mesma plataforma do Visual LED. Além de permitir que o usuário visualize a aparência final do diálogo sendo criado, esta vista permite também que o usuário teste a sua interface. Com este teste, é possível analisar o comportamento do diálogo quando seu tamanho é modificado ou quando se pressiona um botão.

A vista do resultado, entretanto, não é editável, a menos do tamanho do diálogo. Se o tamanho do diálogo for alterado nesta vista, ele também é alterado em todas as outras. Já vimos que o diálogo só está representado na vista do editor gráfico pelo seu filho, logo uma alteração no seu tamanho não modifica a sua representação nesta vista, a menos que ele contenha elementos expansíveis como o `fill` ou o `canvas`. Neste caso, a alteração no *layout* abstrato vai depender dos elementos contidos no diálogo, isto é, se ele contém ou não elementos expansíveis.

A vista resultante de um diálogo não está sempre ativa; ela pode ser ativada e desativada pelo usuário para cada diálogo conforme desejado. Esta opção é fornecida para o usuário porque ele pode não querer ter diversos diálogos sendo representados pela sua interface final, pois, inicialmente, cada elemento é um diálogo. Enquanto o usuário constrói o diálogo desejado, podem existir vários diálogos temporários e a apresentação do *layout* concreto de todos eles pode confundir mais do que ajudar. Ainda assim, é útil poder testar partes de diálogos.

No caso de o *layout* concreto de um diálogo estar ativo, ele é automaticamente atualizado quando ocorrem mudanças no seu *layout* abstrato. Entretanto, quando estas mudanças são feitas na sua descrição em LED, ele só é atualizado sob demanda (veja seção 6.2).

6.5 Comparação com outros editores de interface

Nesta seção, comparamos o Visual LED com os outros editores gráficos de interface estudados. Os editores gráficos de interfaces de *layout* abstrato, ibuild e FormsEdit, tiveram maior influência sobre o projeto de Visual LED que os de *layout* concreto, uma vez que adotam não só a mesma filosofia de *layout* do Visual LED, mas também o mesmo modelo TEX.

Agrupamento

Como nos editores gráficos de *layout* abstrato os diálogos são construídos por composição de elementos, o agrupamento é uma operação de extrema importância. As políticas de agrupamento adotadas pelo ibuild, FormsEdit e Visual LED são bem distintas. No ibuild, o usuário seleciona os elementos desejados, na ordem desejada, e requisita um agrupamento. No FormsEdit, o agrupamento é feito por manipulação direta sem requisição explícita; o grupo é colocado em torno dos seus elementos. Ambas as políticas de agrupamento citadas são *bottom-up*. No Visual LED, a política de agrupamento pode ser tanto *bottom-up* quanto *top-down*. A política *bottom-up* é idêntica à do ibuild. Na política *top-down*, o usuário cria um grupo vazio e então insere, por manipulação direta, seus elementos. Quando o usuário já sabe os elementos primitivos que utilizará, acreditamos que a política *bottom-up* seja mais apropriada. No entanto, nos casos em que o projetista está apenas começando a delinear sua interface, ou ainda não definiu a disposição dos seus elementos, a política *top-down* pode ser mais agradável de se trabalhar.

Visualização e edição da hierarquia

A visualização da hierarquia do diálogo sendo construído é essencial para o usuário. No ibuild, os grupos não têm nenhuma representação geométrica; logo, a hierarquia não pode ser visualizada. O FormsEdit, por sua vez, soluciona este problema desenhando os grupos sempre ao redor de seus filhos. O Visual LED adotou esta solução, mas acrescentou uma melhoria: as bordas dos diversos elementos que podem ter filhos são diferenciados por cor ou estilo de linha. Esta diferenciação permite que os elementos de composição sejam rapidamente distinguidos uns dos outros, o que não acontece no FormsEdit.

Para fornecer ao usuário um editor de interfaces de *layout* abstrato no qual o usuário possa trabalhar com naturalidade, não basta permitir que ele visualize a hierarquia do diálogo; também é necessário que ele possa editá-la. Nos diversos editores gráficos de interface estudados e no Visual LED, a seleção de primitivas no primeiro nível de hierarquia é bem semelhante. A seleção em profundidade dos elementos é que varia, nos editores que a permitem. No ibuild, a seleção em profundidade é possível, mas não é intuitiva: não é possível selecionar o elemento desejado diretamente no diálogo. O FormsEdit oferece a política ascendente, cuja interação é mais simples e natural. O Visual LED facilita ainda mais a seleção em profundidade, fornecendo, além da política ascendente, a política descendente. Cada uma destas políticas pode ser usada nos casos em que ela seja mais apropriada, evitando assim os piores casos de cada uma.

Inserção

Com a seleção em profundidade, já é possível efetuar operações sobre os elementos que não estão no primeiro nível da hierarquia. Na operação de inserção, porém, além dos elementos a serem inseridos, deve-se selecionar a posição onde estes elementos serão inseridos. O ibuild permite que elementos sejam criados e apagados em qualquer nível da hierarquia, mas a inserção e a remoção de elementos existentes só é possível no primeiro nível da hierarquia. Tanto no FormsEdit quanto no Visual LED, é possível inserir e retirar elementos em qualquer nível da hierarquia: basta posicionar o elemento selecionado na posição em que se deseja inseri-lo. O FormsEdit porém não indica ao usuário a posição selecionada para a inserção. No Visual LED, as antenas indicam, além das possíveis posições de inserção, qual delas está ativa a cada momento. Uma antena é ativada quando ela é interceptada pelo elemento sendo inserido. Para o usuário inserir um elemento, basta "passear" com o elemento sobre as antenas, até que a antena que indica a posição onde ele deseja inserir o elemento seja ativada.

Vista textual

Até agora, comparamos a vista gráfica do Visual LED com os outros editores. No entanto, assim como o FormsEdit, o Visual LED tem também uma vista que permite a edição textual do diálogo. A vista textual do FormsEdit tem prioridade sobre a vista gráfica: algumas operações só podem ser feitas na vista textual. Esta característica força o usuário a sempre utilizar a vista textual. No Visual LED, as vistas textual e gráfica têm a mesma prioridade; cabe ao usuário decidir em qual delas quer trabalhar a cada momento, podendo até mesmo não utilizar uma delas em momento algum.

Vista do resultado

O FormsEdit e o Visual LED têm ainda a vista do resultado. Esta vista constitui um protótipo da interface que está ativo e é atualizado durante a construção do diálogo. No entanto, o protótipo do FormsEdit pode ter algumas diferenças do diálogo final, enquanto o Visual LED apresenta um protótipo que sempre corresponde ao diálogo final. Nos demais editores (OPUS, ibuild, Visual Basic e Guide) é necessário requisitar a visualização do protótipo. Além disso, os protótipos do OPUS e do ibuild são apenas aproximações dos diálogos resultantes.

Outras características

Os editores Guide e ibuild geram um arquivo contendo os *stubs* das funções de *callback* dos elementos de interface. Ao modificar a interface, os usuários devem ficar atentos para não perder as modificações feitas neste arquivo. O Visual LED, o FormsEdit, o Visual Basic e o OPUS não têm este problema.

No Visual LED, assim como no ibuild e no FormsEdit, a estratégia de cálculo de tamanho dos diálogos é basicamente *inside-out*. Os tamanhos dos elementos pais são calculados a partir dos tamanhos dos seus filhos. A estratégia *outside-in* só é utilizada para elementos extensíveis (`fill` e `canvas`), cujos tamanhos dependem do tamanho final do diálogo.

O Visual LED tem ainda uma característica que não é encontrada em nenhum dos outros editores gráficos de interface estudados: a portabilidade. Esta portabilidade, tanto do Visual LED quanto dos diálogos gerados por ele, é, na verdade, advinda do uso do sistema IUP/LED. De qualquer forma, esta portabilidade permite ao usuário uma flexibilidade de trabalho muito grande e é uma grande vantagem.

O estudo dos outros editores gráficos de interface se mostrou enriquecedor, na medida em que tentamos aproveitar no Visual LED as vantagens encontradas em cada um e solucionar os problemas.

Taxonomia

Resumindo a comparação feita do Visual LED com os outros editores gráficos de interface, apresentamos a classificação do Visual LED de acordo com a taxonomia apresentada na seção 5.7.

Item de comparação	Guide	V. Basic	OPUS	Ibuild	FormsEdit	Visual LED
paradigma de *layout*	concreto	concreto	interm.	abstrato	abstrato	abstrato
posicionamento de elementos	direta	direta	direta	direta	ambos	ambos
agrupamento	sim	não	sim	sim	sim	sim
percorrimento da hierarquia	não	—	possível	possível	possível	possível
visualização da hierarquia	—	—	possível	não	possível	possível
política de agrupamento	simples	—	outra	simples	hierárquica	hierárquica
protótipo	ativo	ativo	ativo	comp.	ativo	ativo
look-and-feel do protótipo	fiel	aprox.	aprox.	fiel	aprox.	fiel
funções de *callback*	não	existente	não	não	não	não
portabilidade dos diálogos	não	não	não	não	não	sim

Tabela 6.2: Classificação do Visual LED na taxonomia.

Capítulo 7

Conclusão

7.1 Contribuições do Visual LED

Neste trabalho, descrevemos o projeto e o desenvolvimento de uma ferramenta interativa para a geração de interfaces gráficas em LED, o Visual LED. Esta ferramenta apresenta ao usuário três vistas da interface LED sendo construída: uma vista textual, uma vista do *layout* abstrato e a vista do *layout* concreto.

Vista gráfica

As três vistas apresentam representações diferentes de um mesmo diálogo. O programador LED já estava habituado à vista textual, que contém a descrição LED, e à vista do *layout* concreto, que contém o diálogo resultante. No entanto, ele não podia trabalhar com as duas vistas simultaneamente. Além de permitir a integração da descrição em LED com o *look-and-feel* final do diálogo, o Visual LED apresenta uma nova vista gráfica.

A vista gráfica é uma vista intermediária entre a vista textual e a vista do resultado (seção 6.3). Apresentando geometricamente, na vista gráfica, a hierarquia da árvore montada pelo sistema IUP/LED, o Visual LED permite que o usuário concretize imagens mentais que possui do diálogo. Esta é uma grande vantagem do Visual LED, pois facilita o processo de criação de interface mostrado na seção 3.7. Outra vantagem da vista gráfica é que ela permite não só a visualização dos elementos de interface e da sua hierarquia, mas também a manipulação direta destes elementos.

Edição em profundidade

Para chegarmos a uma interação que fosse intuitiva e agradável, estudamos editores gráficos genéricos e de interfaces. Deparamo-nos com o problema de edição em profundidade, que é comum a todos os editores gráficos estudados, mas agravado nos editores gráficos de interfaces de *layout* abstrato, devido à importância dos agrupamentos no *layout*. Apresentamos a solução da árvore para os editores gráficos genéricos e para os editores gráficos de interfaces de *layout* concreto (capítulo 4). No entanto, esta solução não resolvia o problema nos editores gráficos de *layout* abstrato.

Para solucionar o problema de edição em profundidade nos editores gráficos de *layout* abstrato, criamos a política hierárquica, que permite que o usuário acesse qualquer elemento de hierarquia. Para chegar ao elemento desejado, o Visual LED fornece ao usuário o modo descendente e o modo ascendente.

Antenas

Uma outra contribuição do Visual LED na interação com o usuário foi a criação das antenas. As antenas permitem que o usuário insira elementos em qualquer nível da hierarquia. Além disso, as antenas permitem que o usuário identifique imediatamente, e sem ambigüidade, a posição em que está inserindo o elemento.

As vistas textual e gráfica estão integradas, podendo o usuário optar em qual delas deseja trabalhar a cada momento. O usuário tem grande liberdade de trabalho, pois pode passar de uma vista para a outra a qualquer momento. Esta característica do Visual LED faz com que o Visual LED tenha todas as vantagens apresentadas acima, mas nenhuma desvantagem em relação à criação de diálogos através de sua descrição LED. Afinal, a edição da descrição LED é uma opção do Visual LED, porém não é mais a única opção existente.

7.2 Trabalhos futuros

Como trabalhos futuros sugerimos algumas melhorias e extensões ao Visual LED.

Uma característica interessante do ibuild, que não foi implementada no Visual LED, é a possibilidade de customização da interface em tempo de execução do sistema. O usuário pode retirar da interface os elementos que não interessam à sua aplicação. Além disso, o usuário pode criar diálogos e acrescentá-los à interface como se fossem elementos. Isto permite ao usuário a criação de elementos compostos que são relevantes e de uso freqüente na sua aplicação.

Seria ainda mais interessante permitir que o usuário não só customizasse a interface e criasse elementos compostos de interface, mas que pudesse criar elementos genéricos de interface. Com isso, o usuário poderia criar novos elementos de interface que seriam específicos para uma determinada aplicação. Esta é a linha de pesquisa "objetos gráficos ativos" do TeCGraf.

Finalmente, sugerimos que seja implementado o Visual Lua baseado no Visual LED. Lua é uma linguagem desenvolvida no TeCGraf que permite a implementação de programas configuráveis [IFC94]. O Visual Lua seria então um sistema completo de programação visual que utilizaria o sistema IUP/LED e a linguagem de programação Lua.

Bibliografia

[ABB89] G. Avrahami, K. P. Brooks, e M. H. Brown. A two-view approach to constructing user interfaces. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):137–146, July 1989.

[BHC94] E. Bos, C. Huls, e W. Claassen. Edward: full integration of language and action in a multimodal user interface. *Int. J. Human - Computer Studies*, 40:473–495, 1994.

[Bro91] K. P. Brooks. Lilac: A two-view document editor. *Computer*, 24(6):7–19, June 1991.

[DEC88] Digital Equipment Corporation. *Guide to the X Toolkit Widgets: C Language Binding*, 1988.

[DEC89] Digital Equipment Corporation. *FormsVBT Reference Manual*, 1989.

[FGIF94] L. H. Figueiredo, M. Gattass, R. Ierusalimschy, e W. Celes Filho. Estratégias de reuso de *software* no TeCGraf. Em *VIII SBES*, 1994. (a ser publicado).

[FGL93] L. H. Figueiredo, M. Gattass, e C. H. Levy. Uma estratégia de portabilidade para aplicações gráficas interativas. Em *VI SIBGRAPI*, páginas 203–211, 1993.

[Fis91] R. Fischer. GeneSys: Sistema híbrido para modelagem de sólidos. Dissertação de Mestrado, Dep. Informática, PUC-Rio, Rio de Janeiro, Brasil, 1991.

[Hel90] D. Heller. *XView Programming Manual: An OPEN LOOK Toolkit for X11*, 1990.

[HM90] S. E. Hudson e S. P. Mohamed. Interactive specification of flexible user interface displays. *ACM Transactions on Information Systems*, 8(3):269–288, July 1990.

[IFC94] R. Ierusalimschy, L. Figueiredo, e W. Celes. Reference manual of the programming language Lua. Monografias em Ciência da Computação 3/94, PUC-Rio, Rio de Janeiro, Brazil, 1994.

[IV91] InterViews. *Ibuild User's Guide*, 1991.

[Knu84] D. E. Knuth. *The TEXbook*. Addison-Wesley, 1984.

[Lev93] C. H. Levy. IUP/LED: Uma ferramenta portátil de interface com usuário. Dissertação de Mestrado, Dep. Informática, PUC-Rio, Rio de Janeiro, Brasil, 1993.

[Mar92] A. Marcus. *Graphic Design for Electronic Documents and User Interfaces*. Addison-Wesley, 1992.

[MS92] Microsoft. *Programmer's Guide: Visual Basic Programming System for Windows*, 1992.

[NM90] F. Neelamkavil e O. Mullarney. Separating graphics from applications in the design of user interfaces. *The Computer Journal*, páginas 437–443, 1990.

[OSF91a] Open Software Foundation. *OSF/Motif Programmer's Guide*, 1991.

[OSF91b] Open Software Foundation. *User Environment Volume*, 1991.

[SUN90] Sun Microsystems. *OpenWindows Developer's Guide 1.0 User's Manual*, 1990.

[TeC94] TeCGraf, PUC-Rio, Rio de Janeiro, Brasil. *TeCDraw - Manual do Usu'ario versão 1.0*, 1994.

Visual LED: uma ferramenta interativa para geração de interfaces gráficas

Raquel Oliveira Prates

Dissertação apresentada ao Departamento de Informática da PUC-Rio, no dia 15 de agosto de 1994, como parte dos requisitos para a obtenção do título de Mestre em Ciências em Informática, tendo sido aprovada pela Banca Examinadora, da qual participaram os seguintes professores:

Prof. Marcelo Gattass — orientador
PUC-Rio

Prof. Luiz Henrique de Figueiredo — co-orientador
PUC-Rio

Prof. Clarisse Sieckenius de Souza
PUC-Rio

Prof. Carlos José Pereira de Lucena
PUC-Rio

Eng. Ênio E. R. Russo
CENPES/PETROBRÁS

Visto e permitida a impressão.
Rio de Janeiro, de de 1994.

Coordenador dos Programas de Pós-Graduação do Centro Técnico Científico