EDG: uma ferramenta para criação de interfaces gráficas interativas

WALDEMAR CELES FILHO, LUIZ HENRIQUE DE FIGUEIREDO, MARCELO GATTASS

TeCGraf-Grupo de Tecnologia em Computação Gráfica, PUC-Rio
Rua Marquês de São Vicente 255, 22453-900 Rio de Janeiro, RJ, Brasil
`celes,lhf,gattass@icad.puc-rio.br`

Abstract. We describe a tool for developing highly abstract, interactive graphical applications. Simple, high level programmable access is provided to both conventional interface objects and active graphical objects that are specific to the application domain. This allows occasional programmers to quickly develop their own sophisticated interactive tools, such as front-ends to legacy batch programs.

Introdução

A crescente demanda por *software* nas diversas áreas do conhecimento humano implica num maior número de pessoas diretamente envolvidas com o processo de desenvolvimento de programas. Em geral, esses programadores recém-chegados são profissionais qualificados nas áreas específicas em que atuam, mas com poucos conhecimentos de computação. No entanto, com a difusão de computadores pessoais poderosos, invariavelmente eles necessitam criar interfaces adequadas para acessar dados e métodos que implementam tecnologias específicas de suas áreas. A programação de interfaces gráficas do tipo WIMP (*windows*, *icons*, *menus*, *pointers*), utilizando bibliotecas gráficas e *toolkits* de interface convencionais, é muito complexa. Para possibilitar o trabalho desses *programadores ocasionais*, são necessárias *meta-ferramentas* que permitam o desenvolvimento de ferramentas específicas com interfaces adequadas para acessar tecnologias já implementadas. As meta-ferramentas devem apresentar elevado grau de abstração. Não se pode exigir que esses programas específicos sejam desenvolvidos através dos princípios da engenharia de *software* (Cowan et al., 1992), pois a prioridade é implementar as funcionalidades que são requeridas. Cabe aos profissionais em sistemas de computação prover meios para que esses programadores possam desenvolver seus próprios programas de uma maneira simples e eficiente.

Programadores ocasionais vêem com receio a tarefa de programação. Para auxiliá-los, diversas pesquisas têm sido feitas na área de programação visual (Chang, 1990). Recursos visuais tornam as tarefas de programação mais naturais. Entretanto, apesar dos avanços conquistados, sistemas com programação visual ainda não têm a expressividade suficiente para atender às diferentes tarefas de programação (Souza e Ferreira, 1994).

Neste trabalho, apresentamos o sistema EDG, uma evolução do sistema ED (Figueiredo et al., 1992), criado para dar suporte ao desenvolvimento de programas com interfaces gráficas interativas com alto grau de abstração. O sistema permite acesso facilitado tanto a objetos de interface tradicionais quanto a objetos gráficos, possibilitando que programadores ocasionais desenvolvam rapidamente programas com sofisticado grau de interação. Através dos objetos gráficos, os programadores podem criar seus próprios objetos de interface, atribuindo-lhes funcionalidades próprias, direcionadas para o domínio da aplicação.

Requisitos para meta-ferramentas

Meta-ferramentas são programas que auxiliam no desenvolvimento de ferramentas específicas. Neste trabalho, interessa-nos o desenvolvimento de uma meta-ferramenta para auxiliar na criação de interfaces gráficas simples. Existem requisitos que devem ser atendidos para assegurar a utilidade da meta-ferramenta para programadores ocasionais. A seguir, discutem-se três aspectos relacionados com esses requisitos: características das aplicações alvo, características dos usuários programadores, e características da linguagem de programação.

Características das aplicações alvo

No contexto deste trabalho, interfaces gráficas interativas representam acessos a tecnologias específicas implementadas em sistemas maiores. Assim, os programas desenvolvidos são *transformadores* de dados. Suas funcionalidades principais baseiam-se na captura e transformação de dados para serem interpretados por outros sistemas; ou na aquisição e transformação de dados para serem interpretados pelos próprios usuários.

Por representarem interfaces, esses programas necessitam de recursos gráficos para serem implementados. É necessário prover acesso facilitado a objetos de interface tradicionais para composição de diálogos e mecanismos para criação de objetos de interfaces específicos, projetados para atender às funcionalidades específicas da aplicação. Para possibilitar a transformação dos dados, também são necessários recursos para manipulação e estruturação de informações. Construções estruturadas, bibliotecas de suporte e operações de entrada e saída devem estar disponíveis.

Um domínio típico para desenvolvimento de interfaces gráficas é encontrado nos simuladores numéricos da área de engenharia. Diversos programas de simulação são desenvolvidos sem uma interface adequada para entrada dos dados, dificultando, ou mesmo inviabilizando, sua utilização. Interfaces gráficas são apropriadas para esse domínio pois é comum existir representações gráficas que caracterizam a natureza dos dados a serem especificados. A Figura 1 ilustra um modelo de um vaso de pressão, com seus respectivos bocais. Para a captura dos dados que definem as características desse vaso, é conveniente que uma representação gráfica do vaso seja usada como objeto de interface, através do qual o usuário informa os parâmetros necessários à simulação.

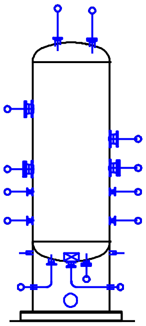


Figura 1: Representação de um vaso de pressão.

Características dos usuários do sistema

Busca-se auxiliar profissionais não especializados em sistemas de computação no desenvolvimento de programas gráficos. Conforme discutido, os programas desenvolvidos geralmente representam interfaces para acessar tecnologias específicas das áreas em que esses profissionais atuam. Nesse aspecto, estendemos o paradigma de programação por usuários finais (*end-user programming*) para programação por profissionais não especializados em computação (ou programadores ocasionais). Usuários e programadores ocasionais têm muitas características em comum, pois ambos necessitam de elevado nível de abstração para o desenvolvimento de programas. No entanto, eles têm objetivos distintos. Conforme caracterizam Cowan et al. (1992), na grande maioria dos casos, o usuário final configura a *sua* aplicação para melhor atender seu uso pessoal. Assim, o próprio usuário é o "programador" do *seu* sistema. O usuário não cria *software* para distribuição, apenas configura para uma possível distribuição informal. Na prática, os programas são *sempre* protótipos, dispensando documentação.

O objetivo deste trabalho é atender não só usuários finais, mas sim um público mais abrangente: além de usuários que queiram desenvolver tarefas individualizadas, deseja-se dar suporte ao desenvolvimento de *programas* de uso mais geral. Programadores ocasionais visam acessar tecnologias específicas criando ferramentas para atender diversos usuários. Portanto, o programador tem o compromisso de desenvolver programas com interfaces adequadas para viabilizar seu uso e permitir sua distribuição, ainda que a um pequeno grupo de usuários.

Características da linguagem de programação

Idealmente, os sistemas para programadores ocasionais deveriam dispensar as linguagens de programação tradicionais e permitir a programação via recursos mais abstratos. Nesse sentido, muitas pesquisas têm sido feitas em programação visual. No entanto, apesar dos contínuos progressos, as linguagens de programação visual ainda não são suficientes para atender todas as tarefas de programação (Souza e Ferreira, 1994). Na prática, apenas a construção de interfaces tem sido plenamente atendida através de recursos visuais. O êxito da programação visual nessa etapa se deve principalmente a dois aspectos: primeiro, interfaces são compostas por um número pequeno de objetos (menus, botões, listas, etc.); segundo, interfaces são naturalmente visuais (Cowan et al., 1992). Para a implementação de procedimentos mais gerais, ainda precisamos de uma linguagem de programação convencional.

Programadores ocasionais têm pouco conhecimento formal de princípios de programação. A própria tarefa de aprender uma linguagem de programação é árdua e pouco animadora. Eles não objetivam tornarem-se profissionais em programação; apenas necessitam construir programas para solucionar problemas específicos. Portanto, a linguagem oferecida a esses profissionais deve ser simples e fácil de aprender. A linguagem também deve *proteger* o usuário de possíveis, e freqüentes, erros de programação (Cowan et al., 1992). Não é possível exigir desses programadores conhecimentos para controlar ponteiros e gerenciamento de memória, para citar apenas um exemplo básico; portanto, qualquer meta-ferramenta para programação ocasional deve ter coleta automática de lixo. Programadores ocasionais devem trabalhar com linguagens que provêem os recursos de estruturação necessários, mas com elevado grau de abstração.

Características do sistema EDG

O sistema EDG é uma ferramenta para desenvolvimento de interfaces gráficas interativas. Nesse sentido, ele oferece acesso programável a objetos de interface e a objetos gráficos. O programador pode construir diálogos através da composição de diversos objetos de interface, além de desenhar e interagir com representações gráficas instanciadas na tela.

No sistema EDG, utilizamos a linguagem de programação Lua (Figueiredo et al., 1994; Ierusalimschy et. al, 1995). Lua é uma linguagem procedural para extensão de aplicações com um poderoso mecanismo para descrição de dados. Lua incorpora as facilidades usuais de uma linguagem procedural – controle de fluxo, atribuições, definição funções e operadores – e permite a configuração da linguagem para um domínio específico qualquer, através da criação de tipos e operadores com semântica própria. O programador final utiliza os recursos incorporados à linguagem através desta configuração, e também acessa os serviços inerentes à própria linguagem. Assim, o programador trabalha com uma linguagem abstrata direcionada para um determinado domínio tendo acesso aos recursos usuais de uma linguagem procedural.

Lua tem sido usada em diversas aplicações industriais e projetos de pesquisas desenvolvidas no TeCGraf. O fato de as aplicações usarem uma mesma linguagem de extensão facilita o aprendizado por parte do usuário e possibilita a comunicação entre aplicações (Cowan e Veitch, 1992).

A Figura 2 ilustra a arquitetura do sistema EDG. O código do EDG representa basicamente uma especialização da linguagem Lua, provendo ao programador acesso a objetos de interface tradicionais e objetos gráficos. Os objetos de interface representam uma abstração do sistema de interface com o usuário IUP (Levy, 1993). Os objetos gráficos são abstrações de primitivas gráficas gerenciadas por uma biblioteca chamada GLB.

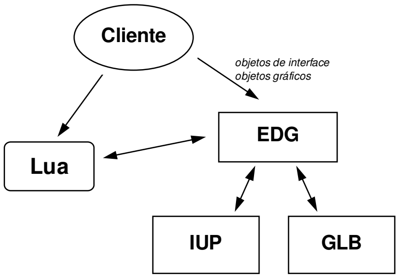


Figura 2: Arquitetura do sistema EDG.

O programador instancia os objetos através de uma codificação descritiva. Por exemplo, o trecho de código abaixo descreve e instancia um objeto gráfico "linha":

```
reta = line {
         x = {0.0, 1.0},                  coordenadas X
         y = {1.0, 1.0},                  coordenadas Y
         color = EDG_YELLOW,               cor da linha
         width = 2.0                          espessura
       }
```

O nome `reta` representa o objeto criado e serve de referência para esse objeto. Os objetos gráficos também podem ser construídos com o programa para edição de desenhos TeCDraw, desenvolvido no TeCGraf. Assim, em vez de codificar a descrição de uma linha, o programador pode desenhá-la diretamente no editor gráfico. No TecDraw, as construções também podem ser identificadas com nomes que servem de referência ao objeto. Esses nomes são a ligação entre os objetos gráficos e as rotinas dos usuários que implementam seus comportamentos.

Processo de desenvolvimento de aplicações EDG

A Figura 3 ilustra as etapas de trabalho no desenvolvimento de aplicações utilizando o sistema EDG. O programador da aplicação utiliza o programa TeCDraw para construir os desenhos que são necessários. Através de um editor de texto, o programador escreve o código do programa usando a linguagem Lua. Nesse código, o programador pode instanciar diálogos acessando os objetos de interface, bem como codificar a manipulação dos objetos gráficos. Usando as funções básicas de Lua, o programador implementa mecanismos para salvar, carregar e transformar os dados fornecidos pelo usuário.

Durante a execução, o sistema EDG interpreta o código gerado pelo programador e o *metafile* gerado pelo TeCDraw. Na aplicação final, o usuário pode interagir com o programa e efetuar operações de entrada e saída de dados.

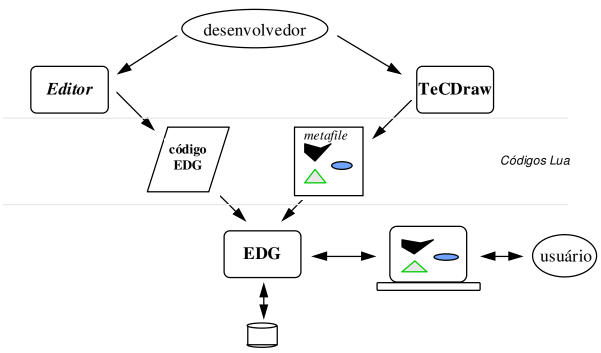


Figura 3: Desenvolvimento de aplicações no EDG.

O sistema EDG também pode ser acoplado a aplicações escritas em C. Existe uma biblioteca, também chamada EDG, que pode ser ligada a outras aplicações. O programador da aplicação inicializa a biblioteca e tem ao seu dispor todos os serviços oferecidos pelo sistema. O EDG tem sido usado dessa forma em várias aplicações para dar suporte a programação da interface e serviços de configuração.

Acesso a objetos de interface

Para possibilitar a construção de diálogos, deve-se prover acesso a objetos de interface. Nesse contexto, salientamos a distinção entre serviços de configuração de *layout* de diálogos e acesso programável aos objetos para construção de diálogos. O primeiro serviço (configuração de *layout*) é oferecido por diversos sistemas de interface. O sistema IUP, por exemplo, possui uma linguagem de especificação de diálogos – LED – que é interpretada em tempo de execução da aplicação. Através de LED, o *layout* e a composição dos diálogos das aplicações podem ser configurados pelo usuário. É possível, por exemplo, desabilitar determinadas funções, redefinir teclas associadas a ações da aplicação, alterar a composição dos diálogos e configurar cores. Esses serviços de configuração não possibilitam, no entanto, a criação de novas funcionalidades.

Os objetivos do sistema EDG são mais abrangentes. O programador deve ter meios de criar diálogos associados às suas próprias funções, validando dados e transformando-os para serem processados por outros programas. O programador deve trabalhar num ambiente com alto grau de abstração, mas a ferramenta deve ser versátil o suficiente para atender às tarefas de programação requeridas. O equilíbrio entre o nível de abstração e a versatilidade caracteriza a funcionalidade da ferramenta que atua como base do ambiente para desenvolvimento de interfaces: de nada adianta elevado grau de abstração sem poder de expressão; por outro lado, não nos servem ambientes versáteis com alto grau de complexidade, pois neste caso poderíamos disponibilizar para o programador as linguagens convencionais de programação (tais como C ou Modula 2), o que está longe do objetivo que buscamos alcançar.

O sistema EDG oferece acesso a objetos de interface que representam abstrações dos objetos de interface do sistema IUP. Nessas abstrações, objetiva-se principalmente facilitar a captura e manipulação de dados. Para exemplificar, considera-se o tratamento dado aos *valores* associados aos objetos de interface. Os objetos que têm valores associados (campos de edição, lista de opções, etc.) apresentam no EDG duas formas correspondentes de expressão desses valores: um valor cadeia de caracteres (*string*) e um valor numérico. Esta estratégia tem trazido muitas facilidades na programação de captura de dados, pois livra o programador da tarefa de conversão de valores. Por exemplo, os valores associados a uma lista de opções são a cadeia de caracteres da opção selecionada e o número dessa opção. Assim, o programador utiliza o valor que lhe for mais adequado melhor: a cadeia de caracteres ou a forma numérica.

Para exemplificar com mais detalhes, analisemos o objeto *field* que possibilita a captura de uma cadeia de caracteres (campo de edição). A abstração *field* caracteriza um objeto para captura de cadeias de caracteres, mas é estendida para captura de valores numéricos e expressões avaliadas em tempo de execução. Se o usuário entrar com uma cadeia de caracteres que pode ser convertida para um valor numérico equivalente, ambos os valores (cadeia de caracteres e valor numérico) ficam automaticamente disponíveis no ambiente de programação. Além disso, se o usuário preceder a cadeia de caracteres com um sinal de igual ‘=‘, o restante da cadeia é avaliado como uma expressão, ficando mais uma vez tanto a cadeia de caracteres da expressão quanto o valor avaliado disponíveis.

Além do objeto *field*, o sistema permite a instanciação de diversos outros objetos, entre eles listas, botões, menus e os elementos de composição do IUP: caixa horizontal e caixa vertical. O trecho de código abaixo ilustra a construção do diálogo mostrado na Figura 4.

```
nome = field { prompt = “Nome:”}
sexo = vradio { options = {“Masc”, “Fem”} }
dial = dialog { title = “Dados Pessoais”,
         object = hbox { nome, sexo } }
       }
```

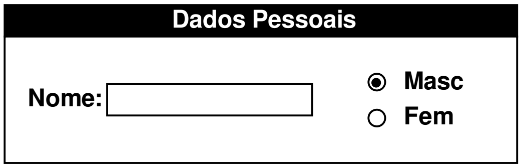


Figura 4: Exemplo de um diálogo simples.

Pode-se associar aos objetos de interface ações que são *callbacks* às interações do usuário. Os diálogos podem ser instanciados em forma modal ou não modal. O sistema permite ainda a associação de funções para validação dos dados capturados.

Acesso e manipulação de objetos gráficos

A principal característica do sistema EDG é prover fácil acesso a instanciação de objetos gráficos e permitir que sejam manipulados. A tarefa de visualizar representações gráficas através de sistemas gráficos convencionais não é trivial. Por exemplo, a construção da representação gráfica ilustrada na Figura 1, com todos seus detalhes gráficos, pode ser muito trabalhosa: deve-se especificar as coordenadas de todas as primitivas gráficas que compõem o desenho.

EDG permite instanciar e manipular representações gráficas (complexas ou não) de maneira muito simples. Como no sistema ED original (Figueiredo et al., 1992), o programador pode criar suas representações gráficas através de um programa de edição de desenhos, no caso o TeCDraw. Esse programa possui os recursos usuais encontrados em editores de desenho. Com simples operações de seleção com o *mouse*, o usuário pode criar diversos tipos de desenhos independentes de dispositivos. Como trabalha-se com objetos gráficos, esses podem ser livremente editados e transformados. Permite-se ainda importar desenhos gerados por outros sistemas.

A particularidade do TecDraw é permitir que as primitivas gráficas que compõem o desenho sejam agrupadas e identificadas como *entidades*, para serem manipuladas no ambiente de programação do EDG. A idéia é permitir que o programador desenhe livremente as representações gráficas que deseja instanciar. Atribuindo-lhes um *nome* simbólico, ainda dentro do desenhador, essas representações tornam-se entidades que podem ser manipuladas na codificação do programa EDG/Lua. Dentro desse código, os nomes atribuídos às entidades representam objetos que podem ser processados. Existem diversos métodos associados aos objetos gráficos que podem ser executados. É possível desenhar e apagar a entidade da área de desenho; mover, escalar ou rodar a entidade; atribuir-lhe novas cores e formas, etc.

A comunicação entre o TeCDraw e o EDG é possível porque utilizamos a sintaxe da própria linguagem Lua para escrever o *metafile* gerado pelo desenhador. Isto é, o arquivo gerado pelo desenhador é um código Lua que descreve cada objeto gráfico que compõe a imagem. Portanto, para o sistema EDG, carregar uma imagem é executar o código Lua correspondente. Como o programador também codifica em Lua, ele naturalmente tem acesso aos objetos gráficos contidos nos *metafiles*.

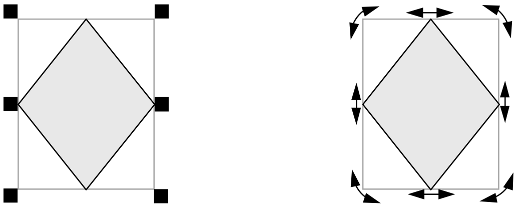
A geração de *metafiles* com as descrições dos objetos gráficos em Lua apresenta diversas vantagens: pode-se usar a própria linguagem para interpretar o arquivo; é fácil editar o arquivo com editores de texto convencionais; possibilita a criação de objetos procedurais (é fácil descrever as coordenadas de uma linha por funções matemáticas). Como desvantagem, ainda existem problemas de eficiência na carga de imagens complexas.

Tratamento de eventos sobre os objetos gráficos

Para permitir a construção de programas gráficos interativos, o sistema EDG permite o tratamento de eventos de *mouse* sobre os objetos gráficos instanciados. O sistema filtra os detalhes da programação por eventos convencionais – tais como o gerenciamento de *button down*, *button up* e *mouse motion* – e passa ao programador a possibilidade de programar sobre dois eventos abstratos: objeto apontado (*click*) e objeto arrastado (*drag*). Sempre que um objeto gráfico for apontado ou arrastado, o sistema executa o método associado ao evento, que pode ser criado pelo programador.

O tratamento do evento de *click* permite criar aplicações para entrada de dados através de formulários gráficos (Figueiredo et al., 1992): os objetos de uma imagem podem representar dados que o usuário deve especificar. Quando o usuário aponta um determinado objeto, o programa apresenta um diálogo com os dados que devem ser capturados. Terminada a entrada de dados, o programa pode fornecer um *feedback* gráfico adequado para indicar que os dados referentes àquele objeto já foram fornecidos. A utilização de formulários gráficos é adequada para entrada de dados na área de engenharia onde é comum existir representações gráficas que caracterizam a natureza dos dados a serem especificados.

A combinação dos eventos *click* e *drag* permite a programação de tarefas de interação mais sofisticadas, impossíveis no sistema ED original. Considere-se dois exemplos típicos: especificação de transformações sobre objetos, e edição de objetos. A especificação de transformações de objetos pode ser feita através da exposição de marcas de controle nas bordas do retângulo que circunscreve o objeto. O usuário especifica a transformação manipulando essas marcas. Assim, a configuração ilustrada na Figura 6a permite que o usuário *arraste* as marcas para aplicar escala sobre o objeto; e a configuração da Figura 6b permite que o usuário *arraste* as marcas para aplicar rotação e cisalhamento sobre o objeto.



(a) Marcas para escalamento. (b) Marcas para rotação e cisalhamento.

Figura 5: Marcas para transformação de objetos.

Para implementar essas funcionalidades com o EDG, pode-se programar o evento de *click* sobre o objeto para criar os objetos que representam as marcas de controle. A cada uma dessas marcas, associa-se uma função que transforma o objeto original, através do evento *drag*. Observa-se que este procedimento pode ser aplicado a qualquer objeto. As marcas podem assumir a forma desejada e suas ações associadas ao *drag* são implementadas independentemente. Portanto, pode-se criar diversas formas de manipulação dos objetos dentro de uma estrutura conceitual simples.

A edição de objetos (*re-shape*) também pode ser implementada com essa estratégia. Para ilustrar, considera-se a ação de editar uma linha poligonal. Para esta tarefa, cria-se marcas de controle em cada vértice da poligonal, e permite-se que o usuário re-posicione os vértices, movendo essas marcas (Figura 6).

No EDG, pode-se programar a ação de *click* sobre a poligonal para mostrar os objetos que representam as marcas de controle. A cada um desses objetos, programa-se o evento de *drag* para permitir o reposicionamento do vértice da poligonal. O trecho de código abaixo ilustra a implementação dessa estratégia.

```
POLY = {}                                         define classe POLY
MARCA = {}                                       define classe MARCA

function POLY:click ()                        trata evento de “click”
 local i = 1
 local n = self:getn()                núm. de ptos. da poligonal
 while i<=n do                      para cada vértice da poligonal
  local m = mark{x=self:getx(i),                    cria uma marca
                 y=self:gety(i),
                 filled=1,
                 poly=self,
                 index=i
                }
  m.godparent = MARCA                 faz objeto herdar de MARCA
  m:draw()
  i = i+1
 end
end

function MARCA:drag (x0, y0, x1, y1)  trata “drag”
 local i = self.index
 self.poly:clear()                               apaga poligonal
 self:clear()                                        apaga marca
 self.poly.x[i] = self.poly.x[i] + (x1-x0)
 self.poly.y[i] = self.poly.y[i] + (y1-y0)
 self.x = self.x + (x1-x0)
 self.x = self.x + (y1-y0)
 self.poly:change("x", self.poly.x)
 self.poly:change("y", self.poly.y)
 self:move(x1-x0,y1-y0)                               move marca
 self.poly:draw()                             redesenha poligonal
 self:draw()                                     redesenha marca
end
```

O conceito de edição de poligonais pode ser estendido para edição de objetos gráficos quaisquer. O programador projeta a forma de edição de um objeto e implementa a criação das marcas de controle necessárias para o usuário realizar a edição. Assim, cada objeto pode ter sua forma própria de edição.

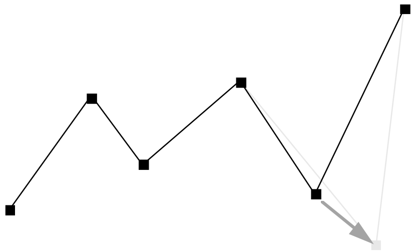


Figura 6: Edição de uma linha poligonal.

Os recursos do sistema EDG também possibilitam a criação de objetos de interface próprios. É muito fácil construir novos objetos de interface e atribuir-lhes funcionalidades próprias. Para tanto, basta que o usuário desenhe o novo objeto de interface e depois associe funções que respondam às ações do usuário. Considere-se o exemplo de construir um objeto de interface que representa um potenciômetro. Supõe-se que queremos um potenciômetro que funcione com um ponteiro marcador, e que o usuário especifique o valor manipulando diretamente este ponteiro. A primeira etapa para criação do novo objeto de interface é desenhá-lo no programa TeCDraw. A Figura 7 ilustra um desenho para representar o potenciômetro. Para o usuário poder manipular o ponteiro, deve-se identificar as primitivas gráficas que representam o desenho do ponteiro como sendo uma *entidade*, conferindo-lhe um nome simbólico; por exemplo, *ponteiro*.

A codificação do funcionamento do potenciômetro é simples: basta escrever uma função associada à ação “arrastar a entidade *ponteiro*”. A função associada ao evento de *drag* recebe como parâmetros as posições inicial e final do *mouse*. Dadas essas posições, pode-se apagar o ponteiro, calcular sua nova posição, posicioná-lo corretamente e então desenhá-lo. O *valor* do potenciômetro deve ser armazenado para permitir futuras consultas. O código abaixo implementa essa funcionalidade.

```
CONTROLE = {}                              define classe CONTROLE

function ponteiro:drag (x0,y0,x1,y1)  trata “drag”
 local a0 = atan2(y0,x0)                        ângulo inicial
 local a1 = atan2(y1,x1)                          ângulo final
 ponteiro:clear()                              apaga ponteiro
 ponteiro:rotate(0, 0, a1-a0) transforma ponteiro
 ponteiro:draw()                           redesenha ponteiro
 angulo = angulo - (a1-a0)                    atualiza angulo
 valor = angulo*100/180                calcula valor associado
end

angulo = 0                                   inicializa angulo
valor = 0                                     inicializa valor
ponteiro.parent = CONTROLE   faz ponteiro herdar de controle
```

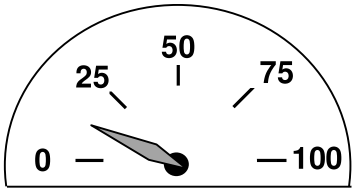


Figura 7: Exemplo de um potenciômetro.

Nos eventos de *drag*, pode-se ainda programar *feedback* dos objetos sendo arrastados. A cada movimento do *mouse*, o sistema executa o método *feedback* que pode ser programado para mostrar as posições intermediárias do objeto.

Transferência de objetos entre aplicações

A programação de um novo objeto de interface como ilustrado na seção anterior permite sua instanciação em qualquer aplicação EDG: basta que o arquivo com a codificação da funcionalidade e o arquivo *metafile* com a descrição da imagem sejam carregados pela aplicação.

Num aspecto mais abrangente, observa-se que o uso do EDG permite a transferência de objetos gráficos entre aplicações. A linguagem Lua tem funções próprias que permitem programar persistência de objetos. Assim, como os objetos podem ser totalmente descritos, incluindo métodos e dados, qualquer aplicação pode instanciá-los e manipulá-los. Os objetos transportam para outras aplicações inclusive seus estados correntes. Portanto, um mesmo objeto pode ser parcialmente editado numa aplicação e concluído em outra. A Figura 8 esquematiza essa transferência de objetos entre aplicações.

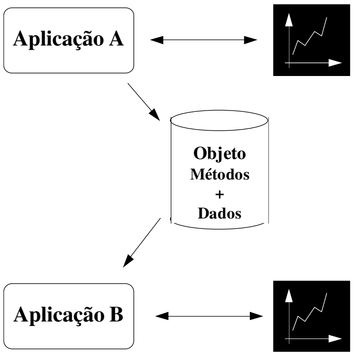


Figura 8: Transferência de objetos entre aplicações EDG.

Comparação com sistemas correlatos

Existem diversos outros sistemas que oferecem suporte para o desenvolvimento de programas gráficos interativos. Nesta seção, comparamos três desses sistemas – Visual Basic, Tcl/Tk e ezd – com o EDG.

Visual Basic tem sido amplamente utilizado para o desenvolvimento de aplicações gráficas interativas. Esse sistema permite a programação dos objetos de interface que compõem as aplicações via recursos visuais. As ações associadas aos eventos sobre os objetos de interface são codificadas em Basic. Para instanciação de representação gráficas, Visual Basic oferece duas formas: através dos *controladores gráficos*, primitivas (linha, imagem ou formas geométricas pré-definidas) que podem ser instanciadas visualmente em tempo de programação; e através de *funções* que permitem programar o traçado de primitivas gráficas. Os controladores gráficos são tratados como objetos de interface sobre os quais podem ser programadas ações associadas aos eventos sobre eles incidentes (Microsoft, 1993).

Visual Basic oferece um ambiente de programação amigável onde programadores ocasionais não encontram dificuldades de projetar suas interfaces. O tratamento das representação gráficas só tem suporte adequado para as primitivas instanciadas visualmente em tempo de programação. As representações gráficas não podem ser adequadamente estruturadas, não permitindo a criação de abstrações de objetos gráficos. Os gráficos desenhados via *funções* têm que ser codificados (e visualizados apenas em tempo de execução) e não há tratamento de eventos sobre eles.

Tcl/Tk formam um sistema para desenvolvimento de aplicações provendo acesso a construção de interfaces gráficas *Motif-like* através da linguagem Tcl (Ousterhout, 1994). Tk implementa um *tookit* completo de objetos de interface. O sistema também permite a instanciação e manipulação de primitivas gráficas. Essas primitivas podem ser identificadas por *tags* (nomes simbólicos) para serem posteriormente referenciadas. O mecanismo de *tags* permite agrupar primitivas especificando o mesmo *tag* para diversas primitivas. O sistema permite aplicar transformações geométricas e programar ações (com *scripts* na linguagem Tcl) em respostas a eventos (*enter*, *leave*, *mouse motion*, etc.) ocorridos sobre as primitivas.

Diversas extensões têm sido feitas no sistema Tcl/Tk e diversas aplicações estão sendo desenvolvidas sobre essa base. A programação em Tcl não apresenta uma sintaxe simples e clara, mas Tk provê acesso a um sofisticado *toolkit* de objetos de interface. A instanciação de representações gráficas tem que ser codificada. O uso de *tags* não é suficiente para uma estruturação adequada dessas representações.

O sistema ezd é um servidor gráfico situado entre o programa de aplicação e o servidor X, provendo acesso facilitado a objetos gráficos estruturados (Bartlett, 1991). Seu principal objetivo é ajudar o programador de aplicação no tratamento e instanciação de objetos gráficos e no gerenciamento dos eventos do X. O ezd estrutura os objetos gráficos de forma análoga ao EDG, permitindo agrupamentos e associação de nomes simbólicos para futuras referências. O sistema filtra os eventos do X que interessam à aplicação. No entanto, o ezd também só permite instanciação das representações gráficas via codificação (em scheme).

A possibilidade do programador usar um editor de desenhos para criar as representações gráficas é uma característica única do EDG. As facilidades de estruturação dessas representações, e a criação de abstrações sobre os objetos gráficos, possibilitam uma programação adequada para aplicações que manipulam desenhos, sem necessidade de codificação de detalhes gráficos.

Deve-se considerar ainda a questão da portabilidade. Tanto Tcl/Tk quanto ezd foram desenvolvidos para sistemas Unix com X. O sistema Visual Basic é para programação em MS-Windows. O sistema EDG tem portabilidade assegurada pelo sistema de interface IUP (Figueiredo et al., 1993). Atualmente, o sistema pode ser usado para programas desenvolvidos em ambientes X, MS-Windows, e DOS.

Conclusão

Criar desenhos com um editor gráfico é mais rápido, mais criativo e menos sujeito a erros do que programá-los usando bibliotecas gráficas convencionais. Além disso, a possibilidade de associar nomes a grupos de primitivas de um desenho permite a criação de objetos gráficos “ativos”; a linguagem Lua fornece mecanismos simples para programar o comportamento desses objetos.

A programação de interfaces gráficas usando eventos abstratos, como *click* e *drag*, parece ser adequada a programadores ocasionais na criação de objetos de interface específicos ao seu domínio de interesse.

O sistema EDG tem sido usado no desenvolvimento de interfaces para acessar tecnologias específicas já existentes em diversas aplicações industriais. Com essas interfaces, cria-se ambientes adequados para entrada de dados e interpretação de resultados. O sistema também está sendo utilizado, acoplado a aplicações, para possibilitar a construção de mecanismos de configuração por usuários e para facilitar a programação da interface pelo próprio programador da aplicação.

Como extensão da implementação atual, prevê-se oferecer acesso a bancos de dados (via SQL) e criar um ambiente integrado onde o programador possa instanciar graficamente objetos de interfaces tradicionais e objetos gráficos, tendo acesso a programação procedural.

Agradecimentos

Agradecemos a Carlos Henrique Levy pelas valiosas sugestões para estruturação e manipulação dos objetos gráficos. A motivação para o sistema ED original teve origem em solicitações de Pedro Paulo Matos, no âmbito do convênio TeCGraf–PETROBRÁS/CENPES. O desenho da Figura 1 foi extraído do programa Vasos, desenvolvido pelo TeCGraf para a Divisão de Projetos Mecânicos do CENPES Os autores são parcialmente financiados com bolsas de formação da CAPES e pesquisa e desenvolvimento do CNPq.

Referências

J. F. Bartlett, “Don’t Fidget with Widgets, Draw!”, *WRL research Report 91/6*, DEC/WRL, 1991.

S. Chang, *Principles of Visual Programming Systems*, Prentice-Hall, 1990.

D. D. Cowan, R. Ierusalimschy e T. M. Stepien, “Programming Environments for End-Users*”, Anais do 12th World Computer Congress*, vol. A-14, p. 54–60, 1992.

D. D. Cowan e R. G. Veitch, “Composing Applications from Reuseable and Configurable Components”, *pre-print,* Computer Systems Group, University of Waterloo, 1992.

L. H. de Figueiredo, C. S. de Souza, M. Gattass e L. C. G. Coelho, "Geração de interfaces para captura de dados sobre desenhos", *Anais do V SIBGRAPI*, 169–175, 1992.

L. H. de Figueiredo, M. Gattass, C. H. Levy, "Uma estratégia de portabilidade para aplicações gráficas interativas", *Anais do VI SIBGRAPI*, 203–211, 1993.

L. H. de Figueiredo, R. Ierusalimschy e W. Celes Filho, "The design and implementation of a language for extending applications", *Anais do XXI Semish*, 273–283, 1994.

R. Ierusalimschy, L. H. de Figueiredo e W. Celes Filho, “Reference manual of the programming language Lua 2.1”, *Monografias em Ciência da Computação*, Dep. de Informática, PUC-Rio 1995.

C. H. Levy, "*IUP/LED: uma ferramenta portátil de interface com usuário*", Dissertação de mestrado, Dep. de Informática, PUC-Rio, 1993.

Microsoft Corporation, *Visual Basic Programmer’s Guide*, version 3.0, 1993.

J. K. Ousterhout, *Tcl and the Tk toolkit*, Addison-Wesley, 1994.

C. S. de Souza e D. J. Ferreira, “Especificações Formais para Linguagens Visuais de Programação”, *Anais do SIBGRAPI VII*, 181–188, 1994.