Desenvolvimento Multiplataforma de Interfaces Gráficas

RENATO DE SOUZA GOMES[1]
BRUNO DE OLIVEIRA SCHNEIDER[2]
JOAQUIM QUINTEIRO UCHÔA[2]

[1]BCC – Bacharelado em Ciência da Computação
[2]DCC – Depto. Ciência da Computação
UFLA – Universidade Federal de Lavras
Cx. Postal 37 – CEP 37.200-000 – Lavras (MG)
{rsgomes,bruno,joukim}@comp.ufla.br

Resumo: o presente artigo tem como objetivo apresentar idéias sobre desenvolvimento multiplataforma de interfaces gráficas e demonstrar comparativamente quatro *kits* de desenvolvimento multiplataforma.

Palavras Chaves: PIGUI, GUI, desenvolvimento multiplataforma.

1 Introdução

A substituição da interface em linha de comando por interface gráfica facilitou em muito o aprendizado de programas por parte dos usuários. Porém, essa transformação acarretou em mais um problema para os programadores ao desenvolverem uma aplicação, uma vez que agora além de se preocupar com o objetivo do programa (como os cálculos matemáticos, por exemplo) é necessário que ele desenvolva uma interface com objetos que interajam com o usuário (como caixas de texto ou botões).

Além disso, o mercado de sistemas operacionais também está sofrendo mudanças, uma vez que novos sistemas operacionais estão aumentando significativamente a concorrência contra o MS-Windows9X© (atual padrão de mercado), como é o caso do Linux©, por exemplo. Logo, surge um novo desafio ao programador: portar seus programas de modo que possam ser compilados em várias plataformas.

Para resolver o primeiro problema apresentado podem ser utilizadas ferramentas apropriadas para o desenvolvimento de interfaces, conhecidas como IDEs (*Integrated Development Enviroment*), as quais são de ótimo desempenho e qualidade. Porém essas ferramentas geralmente destinam-se a uma única plataforma e, às vezes, podem custar bastante caro.

Como o mercado atual requer de um programa que este seja portável e a principal dificuldade encontrada para dar essa característica de portabilidade a um código de programa é justamente no desenvolvimento de interfaces gráficas, surgiram os *kits* PIGUI (*Plataform-Independent Grafical User Interface*) como um meio de se esquivar de tal problema.

Um *kit* PIGUI é uma biblioteca de software que um programador usa para produzir códigos de interfaces gráficas para diferentes sistemas computacionais. O *kit* apresenta funções e/ou objetos que são independentes de qual interface gráfica o programador tem como objetivo. O *kit* não necessariamente fornece quaisquer características adicionais de portabilidade [1].

A idéia lançada é que debates sobre onde o Macintosh© é mais fácil de usar que o MS-Windows, ou os méritos do Unix© sobre o Windows NT©, ou qual será o futuro do OS/2©, irão continuar. Mas se um programador usa um *kit* multiplataforma para construir suas aplicações, ele não terá que ficar apostando sua experiência nos resultados desses debates [2].

1.1 Soluções e problemas com PIGUIs

Quando um programador julgar melhor utilizar um *kit* PIGUI para desenvolver seu programa para que este possa ser compilado em várias plataformas, ele não terá (teoricamente) que mudar qualquer linha de seu código. Com tal *kit*, quando ele decidir colocar um menu na tela, ele chamará a função 'PIGUI_menu' do *kit*. Quando ele compilar seu código em um Macintosh, a biblioteca PIGUI colocará um menu de Macintosh na tela em resposta à chamada PIGUI_menu. Quando compilar com o *flag* configurado para Motif, a chamada faz com que

a biblioteca ponha na tela um menu no estilo Motif. Se ele tomar cuidado em fazer a outra parte do código portável, ele terá um único programa (com um único código fonte) que funciona em várias plataformas.

Porém, o programador deve fazer algumas considerações antes de decidir quando usar um PIGUI:

- muitos (e talvez todos) os PIGUIs irão tornar a execução do código mais lenta;
- o desenvolvimento está limitado ao conjunto de características fornecidas pelo *kit* a menos que ele modifique o código deste *kit* (mas, mesmo assim, por que você iria adquirir um PIGUI se você vai ter que escrever código para ele?);
- *bugs*, em qualquer conjunto de ferramentas (PIGUI ou qualquer outro), diminuem a produção de código;
- poucas pessoas sabem como fazer código de qualquer PIGUI específico que faça uma interface gráfica de uma plataforma específica (Ex.: MS-Windows), então a ajuda *wizard* será limitada;
- a maioria dos PIGUIs só manipulam os aspectos da interface gráfica do programa, as outras características de portabilidade serão por conta do programador;
- se o distribuidor do *kit* adotado sair do negócio, o programador poderia ficar desprovido de suporte para futuras atualizações de Sistemas Operacionais (código fonte pode diminuir, mas não eliminar, a dor de o vendedor fechar suas portas).

Outra solução que o programador poderia escolher seria desenvolver seu sistema partido em objetos GUI e não-GUI e implementar os objetos GUI na API (Applications Programming Interface) nativa. Então, quando estiver portando, somente os objetos GUI precisam ser reescritos para a nova plataforma. Há alguns desenvolvedores que recomendam essa resolução por ela gerar uma melhor forma em cada plataforma e elimina os overheads freqüentemente associados aos kits PIGUI. Obviamente, isto significaria mais esforço do programador, em ambos desenvolvimento inicial e futuras manutenções. Também significa que deve aprender como fazer código para toda plataforma escolhida. Geralmente não é uma tarefa trivial, daí o mercado para os kits PIGUI. Porém, particionar o desenvolvimento em objetos GUI e não-GUI é uma boa resolução de qualquer forma [1].

1.2 Escolha de linguagem

Há kits PIGUI para linguagens incluindo C, C++, Smalltalk, Java, Ada, Tcl e Python, dentre outras. Como as funções GUI são idealmente recomendadas pelo uso de Orientação a Objetos (especialmente herança/reuso), a maioria dos novos *kits* PIGUIs está sendo desenvolvida em linguagens Orientadas a Objeto. A grande maioria dos kits existentes estão em C++ (com muitas ainda em C mas suportando C++), principalmente devido à sua popularidade.

Muitos programadores de C podem encarar a aquisição de uma biblioteca PIGUI como uma oportunidade de migrar para o C++. Se a biblioteca tira total proveito do C++, o programador terá que usar metodologias de C++ (não somente um compilador C++ com sintaxe de C) para usá-la. Quando alguém porta um programa em C para essa biblioteca, deve-se investir uma quantidade de esforço significante para aprender classes, herança e construtores para a mudança de código [1].

1.3 Outras características a se considerar

A grande barreira na programação de GUIs é quando se deseja que o código seja multiplataforma, porém não é a única barreira. Outras características que freqüentemente atrapalham os *kits* PIGUI incluem (mas podem não estar limitadas a) [1]:

- Alinhamento de byte (little-endian X big-endian) de tipos de dados em arquivos e sobre conecções de redes;
- Formatos de dados (Ex.: ponto flutuante que não seja IEEE);
- Gerenciamento de memória (Ex.: limitações de segmentos de 64k);
- Fim de linha em arquivos de texto;
- Navegação de diretórios/pastas e serviços de gerenciamento de arquivos;
- Suporte a multi-threading;
- Comunicações entre processos;
- Funções a nível de sistema (SO específico).

Quando desenvolvendo uma aplicação que possa ser portada para mais de uma plataforma (mesmo se as plataformas forem as versões de 16 e 32 bits do MS-Windows) deve-se ser cauteloso com as diferenças entre as plataformas antes que o desenvolvimento comece.

2 Testes realizados

Foram realizados testes com algumas bibliotecas disponíveis na página em [3].

Os critérios para escolha das bibliotecas foram:

- portável para, pelo menos, MS-Windows e Linux;
- utilizasse C ou C++ como linguagem de programação;
- fosse gratuita.

O compilador utilizado foi o mingw32 [4] para MS-Windows9X e o gcc/g++ [5] presente no Linux, pelas suas características de serem de qualidade reconhecida e serem gratuitos.

Os testes visaram comparar principalmente as condições de orientação a objetos, velocidades e tamanhos dos executáveis gerados.

2.1 Bibliotecas selecionadas

As bibliotecas selecionadas foram:

- FLTK (*Fast Light Tool Kit*) [6];
- WxWindows [7];
- GTK (*GIMP Tool Kit*) [8];
- Fox [9].

2.2 Resultados Obtidos

Com relação à orientação a objetos foi possível fazer as observações demonstradas na tabela1.

Tabela1: Análise da OO das bibliotecas

Biblioteca	Avaliação
FLTK	Razoável
WxWindows	Ótima
GTK*	Ruim
Fox	Boa

* Há uma biblioteca chamada GTK-- para C++, porém é desconhecida uma versão para Windows.

2.3 Vantagens e Desvantagens

A seguir expõe-se uma lista com as possíveis vantagens e desvantagens de cada biblioteca estudada:

- FLTK:
  - Gera executáveis pequenos;
  - Bastante veloz tanto em Linux como em MS-Windows;
  - Difícil instalação em MS-Windows;
  - Orientação a objetos complicada.
- wxWindows:
  - Orientação a objetos muito boa;
  - Ótima documentação on-line;
  - Fácil instalação;
  - Gera executáveis grandes;
  - Lenta em Linux.
- GTK:
  - Gera executáveis pequenos;
  - Bastante veloz tanto em Linux como em MS-Windows;
  - Orientação a objetos fraca;
  - Versão para MS-Windows ainda em desenvolvimento;
- Fox:
  - Orientação a objetos muito boa;
  - makefiles de difícil compreensão;
  - Ainda não funciona em MS-Windows9X com compiladores GNU gratuitos (mingw32/ cygwin).

2.4 Exemplos

A seguir expõe-se alguns programas desenvolvidos com os *kits* analisados:

- wxWindows (figura1):
  - Função: Protótipo de compilador para Pascal.
  - Autor: Renato de Souza Gomes.

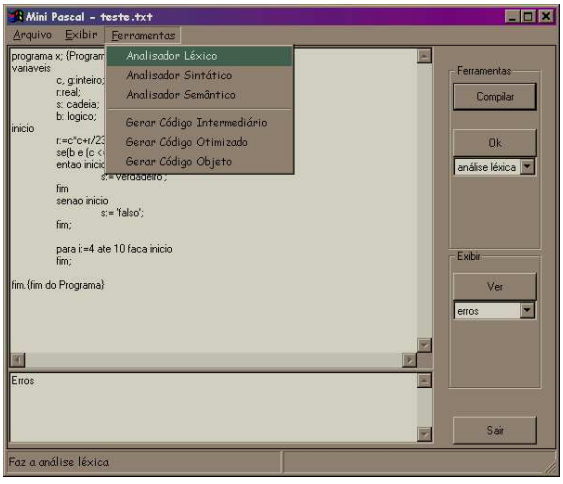

Figura1: Protótipo de compilador para Pascal (wxWindows)

- FLTK (figura2) - exemplo da própria biblioteca:
  - Função: Editor de Texto.
  - Autor: Bill Spitzak.

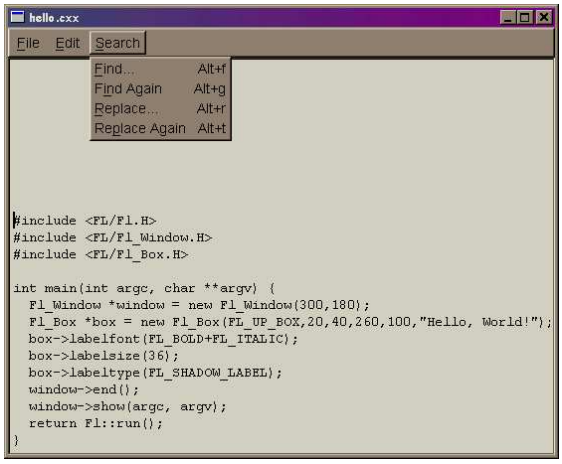

Figura2: Editor de Textos (FLTK)

3 Conclusões

Com os estudos desses *kits* foi possível observar que não há um que se destaque como o melhor de todos. Porém, pode-se obter as qualidades e defeitos de cada um deles proporcionando então uma capacidade de se indicar qual o mais indicado para a implementação de um determinado programa.

A grosso modo pode-se fazer as seguintes observações:

- Desejando-se um aplicativo pequeno e rápido, é recomendável a utilização da FLTK ou GTK;
- Desejando-se um código fonte mais legível, com menos linhas de código recomenda-se a wxWindows ou a Fox;
- Os testes sobre a GTK ainda não estão completos, dado sua portagem para o ambiente Windows encontra-se em processo – assim esta biblioteca merecerá maior atenção no futuro.

Além disso, há ainda uma infinidade de outros *kits* disponíveis que deverão ser analisados no decorrer da pesquisa.

4 Referências

1. McKay, Ross. *Plataform Independent FAQ*. 1997. url: http://www.zeta.org.au/~rosko/pigui.htm.
2. Apiki, Steve. *Paths to Plataform Independence*. 1994. url: http://www.byte.com/art/9401/sec9/art1.htm.
3. Tai, Li-Cheng. The GUI Toolkit, Framework Page. 2000. url: http://www.geocities.com/SiliconValley/Vista/7184/guitool.html.
4. Khan, Mummit. MinGW: Minimalist GNU for Windows. 2000. http://www.mingw.org/.
5. Free Software Foundation. *GCC Home Page*. 2000. url: http://www.gnu.org/software/gcc/gcc.html.
6. Bill Spitzak, et al. *The Fast Light Toolkit Home Page*. 2000. http://www.fltk.org.
7. Roebling, Robert. *WxWindows, Cross-Plataform Development for Unix/Windows/MacOS*. 2000. url: http://wxwindows.org.
8. Amundson, S.T., et al. *The GIMP Toolkit*. 2000. url: http://www.gtk.org.
9. Zijp, Jeroen van der. *Welcome to Fox*. 2000. url: http:// cyberia.cfdrc.com/FOX/fox.html.