AbsTK: uma biblioteca para desenvolvimento unificado de aplicações em modo texto e gráfico

Hisham H. Muhammad André Detsch

[1] Projeto GoboLinux
http://www.gobolinux.org
{lode,detsch}@gobolinux.org

***Resumo**: Manter diferentes versões para uma mesma aplicação em modos gráfico e texto é custoso e problemático. Este artigo apresenta um toolkit livre para construção de interface que abstrai os elementos visuais e permite desenvolver aplicações que executam tanto em ambiente texto como gráfico.*

***Abstract**: Maintaning different versions for an application in text and graphic modes is costly and problematic. This paper presents a free toolkit for interface building, which abstracts away the visual elements and allows one to develop applications which execute both in textual and graphical environments.*

1 Introdução

A disponibilização de interfaces adequadas para configuração e instalação de sistemas ainda é um problema bastante comum em plataformas livres. Um número cada vez maior de usuários de software livre carecem de conhecimentos profundos sobre estes sistemas e suas aplicações, necessitando interfaces mais avançadas e amigáveis.

O desenvolvimento de aplicações amigáveis, entretanto, demanda o uso de bibliotecas de *widgets* gráficos, que são ferramentas complexas e demandam considerável dedicação por parte do desenvolvedor, tornando o desenvolvimento de interfaces para configuração e instalação pouco atrativo para o programador, embora necessário para os usuários. Um problema que agrava este quadro é o fato de que no mundo do software livre diferentes usuários utilizam diferentes interfaces gráficas como KDE e GNOME, ou mesmo o console de texto.

Cada uma destes diferentes ambientes é baseado em bibliotecas visuais diferentes (Qt para KDE, GTK+ para GNOME, e assim por diante), levando o programador a optar entre escolher uma biblioteca ou manter múltiplas versões de sua ferramenta. Ambas as opções trazem consideráveis desvantagens. No primeiro caso, o esforço dispendido pelo programador para produzir uma interface amigável para sua ferramenta acaba por cobrir apenas uma fração do público-alvo. No segundo caso, mesmo que várias versões da interface sejam desenvolvidas, a manutenção de todas em paralelo, e especialmente a sua sincronização de desenvolvimento são um problema.

Visando resolver este problema, foi desenvolvida uma nova biblioteca de *widgets* para a linguagem Python denominada AbsTK (Abstract Tool Kit), que, como o nome indica, é baseada num nível conceitualmente mais abstrato. A programação realizada sobre esta API é baseada nos aspectos semânticos dos elementos, sem necessidade de definição explícita dos aspectos de exibição. Em tempo de execução, dependendo do ambiente onde for executada, a aplicação exibirá então uma interface em modo texto ou gráfico. Através desta API foi realizada a programação do instalador da distribuição GoboLinux [2], utilizado a partir da versão 010.

2 AbsTK

A biblioteca AbsTK surgiu no contexto do projeto GoboLinux. Objetivava-se desenvolver um instalador amigável, que atendesse tanto aos usuários do ambiente gráfico provido pelo CD de instalação como aos usuários que necessitem utilizar o modo texto devido a incompatibilidades de hardware no processo de instalação (ou prefiram o uso do console). A primeira opção seria desenvolver dois instaladores, um para o ambiente gráfico, outro para o ambiente de texto. A manutenção em paralelo de dois programas que deveriam prover funcionalidades idênticas seria algo trabalhoso e problemático.

Decidiu-se então desenvolver uma camada subjacente única que isolasse a semântica do instalador da interface. Diferentemente da abordagem de *front-end/back-end*, tipicamente empregada em programas que provêm múltiplas interfaces, optou-se por isolar a aplicação da interface por meio de uma camada genérica em nível de *toolkit*, que fosse capaz de atender aos seguintes requisitos: (1) simplicidade de prototipação; (2) independência de toolkit de exibição; (3) utilização de campos arbitrários nas janelas; (4) definição simples de interação entre campos.

Baseados nestes requisitos, a definição do AbsTK se baseia em três tipos de entidades: **campos**, entidades de entradas de dados que representam informações de diversos tipos com o qual o usuário pode ou não interagir; **telas**, que são conjuntos de campos; e um ***container***, que engloba uma ou mais telas e representa a interface como um todo. Para a aplicação do instalador, o container é um objeto do tipo `Wizard`, que modela uma interface do tipo "assistente", onde cada uma das telas é um passo no processo de instalação, composto de campos que representam os dados e opções entrados pelo usuário. Primeiramente, o programador cria uma instância da classe `Wizard`. Cada tela do *wizard* corresponde a um objeto da classe `Screen`. Esta classe possui métodos que permitem a adição de campos. A distribuição dos campos na tela é dada pela ordem de inserção, e o layout é definido automaticamente pelo próprio *toolkit*, de acordo com o ambiente onde o programa é executado.

Alguns tipos de campos, como labels, são somente-leitura, e apenas exibem informações para o usuário. Outros têm como objetivo requisitar informações do usuário, e são descritos pela sua representação semântica, independente de como serão visualmente exibidos. Por exemplo, cada campo pode conter uma informação de ajuda adicional. Em um ambiente gráfico, essa informação pode aparecer como um *tooltip* flutuante, e em modo texto, como uma barra de status no canto inferior da tela. Graças a esta abstração, interfaces automaticamente adaptáveis se tornam possíveis: um campo `List`, que permite a escolha de um elemento dentre uma lista, pode se exibido como um *radio group*, se contiver poucos elementos, ou uma *list box* com barra lateral de scroll caso possua muitos elementos. Isto facilita a manutenção da aplicação, uma vez que os elementos de interface são substituídos automaticamente à medida que o desenvolvimento da interface evolui.

Campos podem ter seus valores acessados e alterados através dos métodos `setValue` e `getValue`, onde os campos são referenciados através do identificador definido quando da sua inserção na tela. Assim, basta listar a seqüência de dados a serem requisitados, seguindo uma ordem de telas, combiná-las em um *container*, e a interface do programa está construida, plenamente funcional tanto em ambiente gráfico como de texto. Além da exibição e coleta de dados, foi implementado um sistema de *callbacks* que permite funcionalidades bastante avançadas de programabilidade, embora sem perder a simplicidade do sistema. Cada campo pode ter um *callback* associado, que é acionado sempre que alguma ação for realizada sobre o campo. Por exemplo, é possível alterar o valor de um campo sempre que outro for alterado.

A dinâmica de execução do programa é bastante simples: após a definição do *container*, que é, no caso do instalador, um *wizard* (assistente), basta acionar o metodo `Wizard.start()`. O *wizard* exibirá suas telas para o usuário à medida que este for prosseguindo pelo assistente, seja em modo gráfico pressionando os botões "Próximo" e "Anterior" ou em modo texto, usando as teclas de movimentação indicadas na interface. Assim que o usuário solicitar o término (via botão "Concluir" graficamente, ou pela tecla de término em modo texto), o fluxo de execução volta para o script, que pode acessar todas as informações entradas pelo usuário.

Adicionalmente, AbsTK oferece ainda recursos de internacionalização, permitindo que a interface se traduza automaticamente em tempo de execução, de acordo com as configurações de idioma do ambiente onde ela executa. O arquivo de traduções é em formato ASCII, permitindo fácil edição e contribuição de traduções por parte da comunidade de usuários. Existe também um conversor que permite usar a interface gráfica do Qt Linguist [5] para gerenciar as traduções.

API de programação. As classes disponibilizadas ao usuário do AbsTK são `Screen` e

```
tamanhos = {'KDE':250, 'Python':30, 'Qt':50, 'PyQt':15 }
nomes_pacotes = tamanhos.keys()
wizard = Wizard('Exemplo')
tela = Screen('Pacotes')
def habilita_desabilita_lista() :
  wizard.setEnabled('pacotes', wizard.getValue('habilita_selecao'))
def soma_lista_e_atualiza_campo() :
  selecionados = wizard.getValue('pacotes')[1]
  s = 0
  for cada_um in selecionados :
    s = s + tamanhos[cada_um]
  wizard.setValue('soma', str(s))
tela.addBoolean('habilita_selecao', 'Seleção manual', True, '',
              habilita_desabilita_lista)
tela.addCheckList('pacotes', 'Selecione', (nomes_pacotes,[]), '',
                soma_lista_e_atualiza_campo)
tela.addLabel('soma', '0')
wizard.addScreen(tela)
wizard.start()
```

Figura 1: Exemplo de uso do AbsTK: uma lista de seleção de pacotes

`Wizard`. A classe `Screen` possui diversos métodos para adição de campos, como `addBoolean()`, `addButton()`, `addCheckList()`, `addLabel()`, `addLineEdit()`, `addList()` e `addMultiLineEdit()`. Ao criar um campo, o programador pode especificar, respectivamente, um nome para o campo (para que possa obter os dados posteriormente), um *label* (em toolkits tradicionais, o label é usualmente um *widget* distinto, que deve ser gerenciado separadamente), um valor *default* para o campo, uma mensagem de ajuda e uma função de *callback*. A semântica, o nome e a ordem dos parâmetros são iguais para todos os métodos de adição de campo. Cabe resaltar que todos os parâmetros são opcionais, podendo-se passar apenas os parâmetros considerados necessários pelo programador.

A classe `Wizard`, que define um container do tipo "assistente", permite adicionar telas (através do método `addScreen()`) e requisitar ou ajustar valores de campos que estejam inseridos em alguma de suas telas (métodos `getValue()` e `setValue()`). De forma similar, é possível habilitar ou desabilitar campos através do método `setEnabled()`.

Implementação Qt. A implementação do AbsTK para uso em modo gráfico utiliza os bindings Qt [5] para Python (biblioteca PyQt [4]). Grande parte das funcionalidades exigidas para exibição e funcionamento dos campos oferecidos por AbsTK já estão presentes em PyQt, cabendo à implementação de AbsTK apenas oferecer *wrappers* adequados.

Os campos inseridos são enquadrados em um `QGridLayout`, classe para layout automático disponibilizada pelo Qt. Com isso, as telas podem ser redimendionadas durante o uso sem qualquer problema de organização dos campos, ao mesmo tempo em que o programador não precisa lidar explicitamente com este quesito. A funcionalidade de *callbacks* é implementada através do uso do mecanismo de *sinais* e *slots* oferecido pelo Qt.

Implementação NCurses. A implementação do AbsTK para operação em console baseia-se no *wrapper* Python para a biblioteca NCurses [3]. Esta implementação tem a vantagem de estar disponível na grande maioria dos ambientes Linux, uma vez que a biblioteca NCurses é um item básico da maior parte das distribuições e o *wrapper* Python vem incluído no pacote Python padrão.

Enquanto a biblioteca Qt disponibiliza objetos gráficos de interface completos, de modo que AbsTK limita-se a realizar o mapeamento entre estes objetos e os seus campos, a biblioteca ncurses disponibiliza apenas primitivas para exibição de caracteres. Assim, na implementação NCurses, os *widgets* visuais de modo texto (botões, listas, *check boxes*, etc.) são implementados por AbsTK internamente. O mecanismo de *callbacks* no modo NCurses é implementado conjuntamente com a implementação interna dos *widgets* textuais. Assim, o funcionamento dele na API externa é idêntico

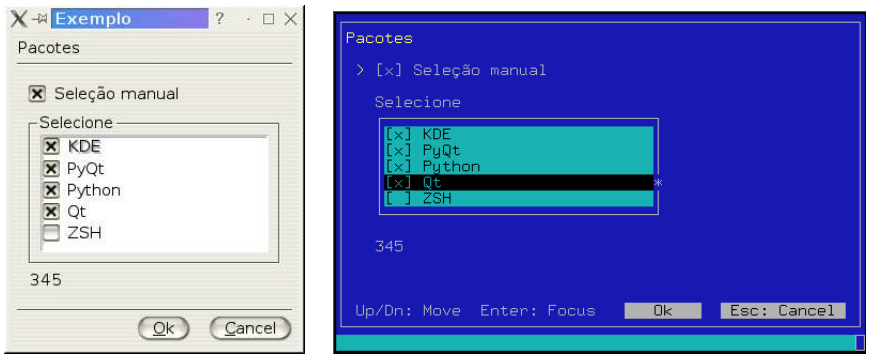

Figura 2: O mesmo programa, rodando nos modos Qt e NCurses.

em ambas as plataformas.

3 Exemplo de uso

Na Figura 1 temos um breve exemplo de uso do AbsTK: uma simplificação da tela de seleção de pacotes do instalador. Após instanciar o *wizard* e a tela, são definidas as rotinas de *callback* utilizadas adiante. Na tela são inseridos um campo booleano (que define se a seleção manual de pacotes deve ser habilitada) uma *check list* (que permite a seleção dos pacotes disponíveis), e um *label* (que exibe a soma dos tamanhos dos pacotes selecionados). Através do ultimo parâmetro passado em `addBoolean()`, define-se que mudanças neste campo devem acionar a função 'habilita_desabilita_lista'. Esta função consulta o valor do campo e define se a *check list* de seleção de pacotes deve estar habilitada para alteração por parte do usuário. De forma similar, utiliza-se a função 'soma_lista_e_atualiza_campo' como *callback* para alterações no campo 'pacotes'. Esta função identifica os pacotes selecionados e soma os seus tamanhos, ajustando o valor do campo 'soma'. Note que o valor de um campo *check list* é dado por uma tupla, onde o primeiro elemento é a lista dos valores disponíveis, e o segundo (acessado pelo índice [1]) a lista dos itens selecionados. Ao final, é acionado o método `start()` do *wizard*, que procede com a exibição das telas e aguarda a finalização pelo usuário.

A Figura 2 apresenta o resultado visual da execução do exemplo utilizando a implementação Qt e NCurses. Um aspecto a ser ressaltado é a adaptação que o próprio AbsTK faz em relação aos botões apresentados na parte inferior da interface. Por haver apenas uma tela no *wizard*, são apresentados os botões 'Cancel' e 'Ok', no lugar dos botões, 'Anterior' e 'Próximo'/'Concluir', presentes em um *wizard* tradicional, com mais de uma tela.

4 Considerações finais

Este artigo apresentou o AbsTK, um toolkit que permite a fácil geração de wizards através da linguagem Python. A biblioteca é distribuída segundo a GNU General Public License, e está disponível para download em `http://www.gobolinux.org/abstk`, onde a API completa é documentada. A biblioteca está em um estágio maduro de desenvolvimento, sendo atualmente usada em ambiente de produção. Como possibilidades de trabalhos futuros temos a implementação de novos *back-ends*, em especial GTK+, e o desenvolvimento de novos tipos de campos e containers, flexibilizando ainda mais a biblioteca para uso em outros tipos de ferramenta.

Referências

[1] AbsTK. `http://www.gobolinux.org/abstk`

[2] MUHAMMAD, H. H., DETSCH, André. "Uma nova proposta para a árvore de diretórios UNIX" In: III Workshop Software Livre, 2002, Porto Alegre. *Anais do III Workshop Software Livre.* , 2002.

[3] *NCurses.* `http://dickey.his.com/ncurses/ncurses.html`

[4] *PyQt.* Riverbank Computing. `http://www.riverbankcomputing.co.uk/pyqt/`

[5] *Qt Overview.* Trolltech. `http://www.trolltech.com/products/qt/`