
Tutorial Introdutório ao Kiwi

Gustavo Sverzut Barbieri

Julho, 2006

barbieri@gmail.com

Resumo

A plataforma Kiwi é desenvolvida, principalmente, pela empresa Async (<http://www.async.com.br>), baseada em Python e GTK, ela visa facilitar o desenvolvimento de aplicativos que se relacionam com Banco de Dados.

<http://www.async.com.br/projects/kiwi/>

Este tutorial pode ser obtido em:

<http://www.gustavobarbieri.com.br/kiwi-howto/>

<http://code.gustavobarbieri.com.br/playground/kiwi-howto/> (SVN)

1 Requisitos

- **GTK+ 2.6.x:** <ftp://ftp.gtk.org/pub/gtk/v2.6/>
- **PyGTK 2.6.x:** <ftp://ftp.gtk.org/pub/gtk/python/v2.6/>
- **Python > 2.3:** <http://www.python.org/download/>
- **Gazpacho 0.6.x:** <http://ftp.gnome.org/pub/GNOME/sources/gazpacho/0.6/> mas se possível utilize a versão do SVN:

```
svn co http://svn.sicem.biz/gazpacho/trunk gazpacho
```

2 Sobre este Texto

Este texto é baseado no texto do Kiko (Christian Reis) disponível em <http://www.async.com.br/projects/kiwi/howto/> o qual descreve a versão 1 da plataforma. Desde lá algumas coisas mudaram, porém os conceitos ainda são válidos. Este texto visa ser uma abordagem mais superficial, porém rápida e direta, com os exemplos funcionando na versão mais nova do software.

Durante a leitura e entendimento deste texto, você deverá consultar a referência da interface de programação do Kiwi, em:

<http://async.com.br/projects/kiwi/api/>

3 Obtendo o Kiwi

Utilize o software SVN:

```
svn co svn://svn.async.com.br/kiwi/trunk kiwi
```

4 Instalação

Após instalar as dependências e baixar o código do Kiwi, instale-o no seu sistema, entre no diretório do kiwi e como super-usuário (root) faça:

```
python setup.py install
```

5 Primeiros Passos com Kiwi e MVC

O Kiwi utiliza o modelo de desenvolvimento Modelo-Visão-Controlador (*Model-View-Controller*, MVC).

Nosso primeiro exemplo vai utilizar o modelo de pessoa definido por Pessoa, com os atributos: nome, endereço e telefone. Este modelo pode ser qualquer classe Python.

Nossa “Visão” vai utilizar um arquivo XML gerado no Gazpacho. É um arquivo glade convencional, a única diferença é que os campos do modelo que se quer ligar automaticamente à visão devem ser baseados nos tipos em `kiwi.ui.widgets`, como `kiwi.ui.widgets.entry.Entry`. No arquivo XML ele vai aparecer como `class=“kiwi+ui+widgets+entry+Entry”`. O nome do objeto e a propriedade `model_attribute` devem ter os nomes indicados no modelo. Outra peculiaridade é que o nome do arquivo “.glade” deve ser igual ao do componente (widget) principal, caso contrário é necessário especificar o parâmetro “gladename” para o construtor das classes de visão.

O controlador neste caso é feito automaticamente pelo Kiwi.

Os códigos estão presentes no diretório `example-01`, divididos em 4 diretórios:

- **simple:** o mínimo necessário. O modelo é um objeto python convencional.
- **pickle:** uma versão que persiste os dados utilizando o formato de serialização pickle.
- **sqlobject:** uma versão que persiste os dados utilizando sqlobject.
- **kiwi-model:** uma versão que utiliza `kiwi.models.Model`. Esta versão mostra que ao utilizá-la, a interface é notificada de modificações no modelo.

Ambos exemplos são bem simples, os 3 primeiros diferindo apenas no tipo de modelo utilizado. Já `kiwi-model` utiliza-se de duas visões para o mesmo modelo, sendo assim possível notar que se este for derivado de `kiwi.models.Model`, a interface é notificada de mudanças no modelo, a seqüência de acontecimentos é:

```
Usuário modifica Visão 1 ⇒  
Gera evento para o Controlador 1 ⇒  
Modifica o Model ⇒  
Notifica os proxies 1 e 2 ⇒  
Visão 1 já está correta, fica igual. Visão 2 será atualizada.
```

Os exemplos são compostos da seguinte forma:

1. Importamos as classes e funções dos módulos que utilizaremos.
2. Definimos nosso modelo.
3. Instanciamos o modelo.
4. Criamos a visão, passando como parâmetro a função a ser executada quando a janela for fechada (`delete_handler`), os nomes dos componentes gráficos do XML/Glade os quais queremos ter acesso como atributos do nosso objeto de visão (widgets) e o nome do arquivo XML (`gladefile`). Note que o componente principal deve ter o mesmo nome do arquivo, senão será necessário especificar o parâmetro adicional “`glade-name`”.
5. Adicionamos um “proxy” para manter a sincronia entre modelo e visão.
6. Configuramos a interface para focalizar o componente do topo e ainda mostrar todos os componentes existentes.
7. Por fim nós entregamos a execução do programa para o GTK, que entrará no seu laço para tratamento de eventos, entendendo assim entrada de teclado, mouse e outros.

6 Componentes Básicos do Kiwi

O Kiwi é composto por um conjunto de classes que podem ser agrupadas em:

- **Visão (*Views*):** representação gráfica, com componentes de interface de usuário: botões, entrada de texto, rótulos, figuras e outros.
- **Controladores (*Controllers*):** trata os sinais e eventos gerados pelo usuário que interage com os componentes de Visão: entradas de teclado e mouse, eventos temporais e outros.
- **Delegadores (*Delegates*):** classes que combinam Visão e Controladores para facilitar e agilizar o desenvolvimento.
- **Modelos (*Models*):** são objetos que representam/modelam seus dados.
- **Proxies:** facilitador que mantém sincronizados os dados do modelo e visão, toda mudança na visão será aplicada ao modelo. Caso o modelo seja derivado de `kiwi.models.Model`, o contrário também é verdade: mudanças no modelo serão aplicadas na visão. São utilizados para a maioria dos formulários.

6.1 Visão (*Views*)

O Kiwi provê duas classes para Visão, localizadas em `kiwi.ui.view`:

- **`SlaveView`:** é considerado um objeto do GTK e pode ser embutido em outros componentes que aceitem um `GObject`. Várias `SlaveView` podem ser combinadas para formar uma interface completa a partir de componentes reusáveis. Pode ser baseada em um arquivo XML do `gaspacho/glade`.
- **`BaseView`:** derivado de `SlaveView`, implementa algumas funcionalidades extras necessárias para uma janela principal (*oplevel*). O parâmetro do construtor `delete_handler` é um objeto “chamável” (*callable*) que será chamado quando a janela for removida (por exemplo o usuário clicou no botão de fechar a janela).

Composição de Interfaces de Usuário

A fim de reusar os componentes da interface de usuário, podemos combinar `SlaveViews`, compondo, assim, uma interface de usuário mais rica.

Os passos para fazer a composição são extremamente simples:

1. **Gazpacho:** Desenhe os seus componentes em arquivos Glade/XML separados.
2. **Python:** Crie uma instância de `SlaveView` para cada componente criado.
3. **Gazpacho:** Na tela principal, na qual os objetos serão embutidos, crie componentes do tipo `EventBox` com nomes apropriados. Este elemento será removido pelo Kiwi e seu componente será incluído no lugar.
4. **Python:** Crie uma instância de `BaseView` para a tela principal.
5. **Python:** Utilize o método `attach_slave()` de `BaseView` com os parâmetro sendo o nome do `EventBox` adicionado e a instância de `SlaveView` equivalente.

Vide os arquivos em `example-02/`. Os componentes separados são `entry_editor` e `list_entries` que podem ser visualizados nos arquivos XML/Glade correspondentes. A tela principal, também chamada de casca (ou em inglês, “shell”) está em `addressbook.glade`. O arquivo `addressbook.py` lê os três componentes e então anexa os componentes `entry_editor` e `list_entries` ao componente principal `addressbook`. Note, porém, que por enquanto não temos nenhum modelo associado.

6.2 Controladores (Controllers)

Para que a interface de usuário responda aos eventos, seja o pressionar de um botão, teclas de atalho, preenchimento de um campo, etc precisamos ligá-la a um controlador.

O kiwi provê uma classe chamada `BaseController` para implementarmos os nossos controladores. Esta classe tem um recurso que torna extraordinariamente fácil criarmos as ações, apenas temos que criar métodos que tenham a nomenclatura:

```
<quando>_<nome_do_componente>__<senal_do_componente>
```

<quando> pode assumir 2 valores: “**on**” e “**after**”. O primeiro é usado para ações no instante que elas aconteceram, geralmente utilizadas para “*clicks*” de botão, ativação de ações, etc. Já o segundo é usado para ações após o “on” ter sido executado, geralmente utilizadas para ações como mudança de um campo de texto.

Por exemplo, se queremos que ao (quando=“on”) pressionar (senal_do_componente=“clicked”) o botão cancelar (nome_do_componente=“cancelar”) seja chamado nosso método, então usamos:

```
def on_cancelar__clicked( self, button, *args ):
    codigo
```

Caso o desejado seja uma ação após (quando=“after”) o conteúdo de uma caixa de texto chamada nome (nome_do_componente=“nome”) tenha mudado (senal_do_componente=“content_changed”), poderíamos usar:

```
def after_nome__content_changed( self, entry, *args ):
    codigo
```

Para associar teclas de atalho, apenas crie um mapeamento (`dict`) que contenha as teclas (ex: `gtk.keysyms.a`) como chave e as funções de tratamento como valor.

Vide os arquivos em `example-03/`. Este exemplo define um controlador `ExampleController` com 3 ações, sendo uma para cada botão e outra para a caixa de texto. Os botões, quando clicados, irão modificar o texto do rótulo “`label_last_button`”. A caixa de texto, ao ser modificada, irá ter seu conteúdo copiado para o rótulo “`label_name`”. Note que todos os componentes utilizados devem ter sido declarados com o parâmetro “`widgets`” de `BaseView`.

6.3 Delegadores (Delegates)

Dos exemplos anteriores com Visão e Controlador você já deve imaginar que deveria existir algo para automatizar a tarefa. E existe: os delegadores (`Delegates`).

Delegadores são nada mais que a junção de Visão e Controlador, feita usando herança múltipla, com o objetivo de facilitar a programação.

Existem também o `SlaveDelegate`, que é a junção de `SlaveView` com `Controller`, e se comunica com o *slave*.

Vide os arquivos em `example-04/`, sendo o sub-diretório `simple/` um amostra de um delegador simples e o sub-diretório `composite/` um exemplo mais avançado, com `SlaveDelegate` para compor uma tela utilizando componentes menores.

O exemplo simples (`example-04/simple`) é o `example-03` utilizando `Delegates`. Já o exemplo composto (`example-04/composite`) é o `example-02` utilizando `Delegates` e `SlaveDelegates`. Note que `SlaveViews`, e portanto `BaseView`, têm disponível o sinal “`result`” que deve ser utilizado para passar dados entre várias visões e, portanto, delegadores. Isto está exemplificado em `example-04/composite` no qual `ListEntries` emite o sinal quando a seleção da tabela mudou e `Addressbook` trata este sinal com `entry_selected()`, perceba que a conexão do sinal tem que ser feito utilizando o método convencional do GTK ao invés da facilidade do controlador.

6.4 Modelos (Models), Tipos de Dados e Validadores

A plataforma Kiwi já faz a conferência de tipos de dados baseada no valor do campo “`data_type`” do arquivo XML/Glade. Além disso, ela confere os campos obrigatórios baseada no valor do campo “`mandatory`” deste mesmo arquivo.

Após ter sido ligado o modelo à visão e controle (ou delegador), o Kiwi mostra um ícone ao lado dos campos obrigatórios e ainda não preenchidos, indicando que o mesmo deve conter um valor. Os campos que não conferem com o tipo de dado, por exemplo um texto no lugar de um inteiro, ficarão marcados em vermelho e um ícone aparecerá ao lado que ainda fornecerá uma dica “`tool tip`” dizendo qual o tipo de dado esperado.

Podemos também registrar uma função que será chamada quando o estado do formulário mudar de válido para inválido, podendo, por exemplo, desabilitar o botão de “`Ok`”. Esta função é `register_validate_function()`.

Vide `example-05/simple`.

Porém podemos adicionar nosso próprio validador aos campos, utilizando métodos do controlador com o sinal “`validate`” e a qual deve retornar `ValidationError` quando a validação falhar, por exemplo:

```
def on_caixa_de_texto__validate( self, entry, data ):
    if not check_data( data ):
        return ValidationError( "Message to be displayed in tool tip" )
```

Vide `example-05/advanced`.