Embeleze seu desktop KDE com o SuperKaramba e Phyton

Python pra Karamba

O SuperKaramba lhe ajuda transformar o background de seu desktop KDE em algo mais útil. Agora, com uma ajudinha da linguagem Python, você pode definir exatamente como vai ser aquele recurso extra de que você precisa.

POR HAGEN HÖPFNER

Uma vez que você começa a usar o SuperKaramba [2], para adicionar um relógio ou um MP3 Player ao background de seu desktop KDE [1], provavelmente você vai começar a querer mais. A boa notícia é que você pode usar a linguagem de programação Python para adicionar uma ampla gama de recursos interativos aos temas do Karamba. Por exemplo, você pode criar menus, modificar textos dinamicamente e adicionar suporte a drag & drop aos seus temas.

O conceito é o seguinte: Quando um evento do KDE ocorre (por exemplo. ao mover o mouse, iniciar um programa, ou selecionar uma opção em um menu), o programa responsável por tais eventos emite um sinal, que é recebido e interpretado por uma função. Naturalmente, o SuperKaramba pode interpretar eventos deste tipo. A API Python do SuperKaramba [5], permite que você especifique o que deve ocorrer.

Encantando a serpente

Se nós quisermos usar Python e adicionar mais funcionalidades aos temas [1], nós não precisamos modificar o código existente. Em vez disso, basta adicionar um outro arquivo ao diretório do tema. Ele deve ter o mesmo nome que o tema, e é reconhecido pela extensão *.py*. Os desenvolvedores do Super Karamba recomendam que você começe com um modelo [6], e a partir daí modifique-o. Veja um link para um exemplo no item [4], no quadro *Info* ao final deste artigo. Descompacte-o com o comando:

```
tar  - C ~/.superkaramba  - ➥
xjvf U small_text_xmms.tar.bz2
```

no diretório *~/.superkaramba* (talvez você precise criar o diretório antes). Isto vai criar um diretório chamado *small_text_xmms* contendo o tema. Agora copie o arquivo modelo, *template.py*, para este diretório. Certifique-se de mudar o nome do arquivo para o mesmo nome do tema:

```
cp template.py ➥
~/.superkaramba/➥
small_text_xmms/small_text_xmms➥
 .py
```

Agora abra o arquivo do tema, *small_text_xmms.theme*, selecionando Open... na janela principal do SuperKaramba, para traduzir o código fonte Python em ***bytecode*** Python (veja o Quadro 1). Se você não consegue ver o diretório oculto *.superkaramba*, na caixa de diálogo de seleção do tema, aperte [F8]. Como o compilador mostra mensagens de erro e debug na saída padrão (stdout), faz sentido iniciar o SuperKaramba manualmente digitando *superkaramba* (talvez você tenha que especificar o caminho completo até o binário) em um terminal como o *konsole*. Por exemplo, uma mensagem como "Minha extensão em Python foi carregada!," é gerada pelo seguinte comando:

```
print "Minha extensão em Python➥
 foi carregada!"
```

na última linha do modelo. Isto permite que eu me assegure de que a compilação do arquivo Python foi feita corretamente. Infelizmente, não foi isto o que aconteceu em minha instalação do Suse Linux 9, pois a distribuição configura incorretamente algumas das variáveis de ambiente usadas pelo Python. Neste caso, para resolver o problema digite os comandos abaixo no console antes de executar o SuperKaramba:

```
 export PYTHONPATH=/usr/lib➥
 /python2.3/
 export PYTHONHOME=/usr/lib/➥
python2.3/
```

Arrastar & Soltar

Nossa primeira extensão adiciona suporte a drag & drop de arquivos mp3, que

permite que os usuários arrastem um arquivo do Konqueror para o tocador de MP3 integrado. Por enquanto, queremos que o XMMS toque os arquivos.

A Listagem 1 mostra nossa solução Se você não quiser desenvolver o arquivo modelo, simplesmente sobrescreva o arquivo *small_text_xmms.py* com o conteúdo da listagem.

Para permitir que um tema do Super Karamba suporte drag & drop, o tema precisa ser inicializado. A melhor maneira de se fazer isto é criar um *widget* do SuperKaramba (veja Glossário nesta página). Isto emite um sinal que será recebido por um dos "receptores de eventos" do SuperKaramba, a função *callback initWidget()*. Podemos digitar:

```
def initWidget(widget):
```

para definir na linha seguinte a reação quando usarmos o tema como fundo:

```
karamba.acceptDrops(widget)
```

Neste caso, queremos que o widget aceite que itens sejam soltos sobre ele. O SuperKaramba usa o *itemDropped()* para processar o sinal. A função espera receber duas informações, como indicado na Tabela 1: um apontador para o widget e - no caso de arquivos - uma lista de nome de arquivo em múltiplas linhas na variável *dropText*, onde cada linha representa um arquivo.

Infelizmente, cada nome de arquivo é precedido pela palavra chave *file:*. Como o XMMS não sabe como lidar com URLs do tipo *file:/caminho/para/arquivo.mp3*, mas espera um nome de arquivo no formato */caminho/para/arquivo.mp3*, temos que remover os dados supérfluos das linhas armazenadas em *dropText* antes de passá-los ao XMMS para que os nomes de arquivo sejam reconhecidos.

Perdendo peso

Nós vamos precisar das funções de manipulação de strings [7] acessíveis através de, *import string*. A próxima etapa é definir uma variável vazia chamada *xmms_filename = ""* onde iremos armazenar a lista "limpa" de nomes de arquivos. Nós podemos então chamar *string.split()* para dividir a lista de URLs de *dropText* em seus componentes:

```
string.split(string.➥
rstrip(str (dropText)),"\n")
```

Estamos usando o caractere *\n* (nova linha) como separador. A função mais interna na expressão acima, *str()*, permite que conteúdo de *dropText* seja manipulado como uma string, enquanto *string.rstrip()* remove todos os caracteres não padronizados à direita.

Agora que temos as URLs individuais, podemos usar um loop *for* para remover o prefixo *file:*. Vamos escreverer as URLs na variável *filename* uma por uma, com:

```
string.split(nomedoarquivo, ➥
"file:",  1)
```

Para ser mais preciso, dividimos o conteúdo de *filename* em duas partes na primeira (1) instância do separador *file:*. Se não fizéssemos isso quaisquer arquivos MP3 que contivessem a string *file:* seria cortado em pedaços. O resultado é uma lista com duas entradas, que armazenamos em uma outra variável, *temp*. O primeiros elemento, *temp[0]*, contêm apenas uma string vazia depois da primeira operação de divisão.

Para cada iteração do loop, adicionamos o nome do arquivo ao conteúdo da variável *xmms_filename*. Como o ele é o segundo elemento da lista *temp*, podemos acessá-lo como *temp[1]*:

```
xmms_filename=xmms_➥
filename+ " \" "+ string.➥
rstrip(U temp[1 ]) +" \ ""
```

Tabela 1 - Receptores de sinais do SuperKaramba

Função	Chamada quando...
initWidget(widget)	Um widget SuperKaramba é criado.
widgetUpdated(widget)	O tema é atualizado. O intervalo de atualização é definido no arquivo *.theme*.
widgetClicked(widget, x, y, button)	Um clique do mouse ocorre dentro do tema. *x* e *y* indicam as coordenadas (relativas ao tema) *button* indica o botão do mouse que foi pressionado.
widgetMouseMoved(widget, x, y, button)	O mouse é movido dentro do tema. *x* e *y* indicam as coordenadas atuais (relativas ao tema); *button* indica o botão do mouse que foi acionado, e seu status (pressionado ou não).
menuItemClicked(widget, menu, id)	Um item de menu é selecionado. Isto informa o manipulador do menu (veja o texto) e o item de menu que foi clicado (*id*).
menuOptionChanged(widget, key, value)	Um item no menu de configuração do tema é selecionado. *key* indica o manipulador do item do menu, e *value* o novo valor (verdadeiro ou falso).
meterClicked(widget, meter, button)	Um mostrador é clicado. *meter* indica o manipulador, *button* o botão do mouse que foi clicado.
commandOutput(widget, pid, output)	Um programa é chamado por *executeInteractive()*, desde que isto gere saída em **stdout**. pid é o *process ID* do programa, *output* contém o texto enviado a **stdout**.
itemDropped(widget, dropText)	Objetos (por exemplo ícones) são soltos sobre o tema através de uma operação de arrastar e soltar. *dropText* contém o texto do objeto (por exemplo, sua URL, veja a seção **Arrastar & Soltar**).
StartupAdded(widget, startup)	O KDE executa um novo aplicativo. Quando a inicialização do programa é completada, um sinal *startupRemoved()* é emitido, seguido por *taskAdded()*.
startupRemoved(widget, task)	Veja *startupAdded()*.
taskAdded(widget, task)	Veja *startupAdded()*.
taskRemoved(widget, task)	O programa foi encerrado.
activeTaskChanged(widget, task)	Um outro aplicativo é movido para o primeiro plano.

GLOSSÁRIO

API: *Uma Application Programming Interface (Interface de Programação de Aplicativos), define um número de funções dentro de um software que outros programas escritos em uma linguagem específica podem chamar. Em nosso caso, aplicativos em Python podem usar as funções implementadas no SuperKaramba.*

stdout: *A saída padrão especifica para onde os aplicativos Linux devem enviar o texto de saída.Tipicamente é a tela, mas você pode redirecionar esta saída para um arquivo ou uma impressora. Além do stdout, o Linux também possui uma saída chamada stderror (tipicamente também o monitor), para onde todas as mensagens de erro são enviadas, e stdin, a entrada de dados padrão, que geralmente é o teclado.*

Widget: *Termo genérico para os elementos de controle de uma interface gráfica. Estes incluem janelas, botões, menus, checkboxes, abas, barras de rolagem, etc.*

Para permitir que os XMMS toque mp3 com espaços nos nomes dos arquivos, coloque-os entre aspas. Como o Python também usa aspas para separar strings, temos que isolar o caractere usando "\"", um procedimento razoavelmente complexo. Se *xmms_filename* realmente incluir aspas, isolá-las com \ diz ao XMMS que elas devem ser consideradas literalmente. Após armazenar a lista com os nomes dos arquivos em *xmms_filename*, podemos usar:

```
karamba.execute("xmms"+ ➥
xmms_filename)
```

para dizer ao XMMS para abrir o arquivo MP3 que foi solto sobre nosso widget.

O menu, por favor

Adicionar outro menu para complementar o tradicional menu acionado com o botão direito do mouse é outra forma interessante de incrementar um tema. Pode ser um menu para lançamento rápido de aplicativos que é acessado quando você dá um duplo-clique com o botão esquerdo do mouse, ou um clique com o botão do meio. Seria deselegante definir um número infinito de áreas clicáveis no arquivo *.theme*, então vamos usar a API Python para fazer isto.

Para experimentar, substitua o conteúdo do arquivo *~/.superkaramba/small_text_xmms/small_text_xmms.py* com o conteúdo da Listagem 2. Minhas desculpas aos desenvolvedores profissionais pela gambiarra no uso das variáveis globais *select_menu* e *my_menu*, usadas apenas para simplificar as coisas.

select_menu tem inicialmente valor *0* (zero) e representará mais tarde o *object ID* do menu. *my_menu [[],[]]* cria um "contâiner" para armazenamento de uma lista com dois elementos. Esta lista será usada mais tarde pelo programa para mapear os itens do menu ao IDs atribuídos a eles.

Em resposta à abertura da janela do tema, *initWidget()* adiciona um menu:

```
selectmenu= karamba.create➥
Menu(widget)
```

O número interno atribuído pelo sistema é armazenado na variável global *select menu*. Isto é chamado de "índice", porque o número é usado para identificar internamente o menu. A, um tanto quanto longa, expressão a seguir armazena um menu com dois itens em *my_menu*. O primeiros deles...

```
karamba.addMenuItem(widget, ➥
selectmenu , "konqueror", ➥
"konqueror.png")
```

...cria uma entrada para o navegador konqueror no *selectmenu*. O ícone *konqueror.png*, que também tem que ser armazenado no mesmo diretório do tema, completa o item do menu. O segundo *addMenuItem()* faz a mesma coisa, adicionando uma entrada para o navegador Opera:

```
karamba.addMenuItem(widget, ➥
selectmenu , "opera", ➥
"opera.png")
```

Naturalmente, no momento ainda falta a funcionalidade. Para permitir que adicionemos funções aos itens do menu, precisamos de um meio para distinguir os itens individuais. Podemos armazenar as palavras *konqueror* ou *opera* no identificador criado por *addMenuItem()* em *my_menu*. O conteúdo da variável se parece com isto:

```
(['konqueror', -22], ['opera',➥
-23])
```

Agora temos um menu. Naturalmente ele não será exibido imediatamente, apenas quando você der um duplo-clique com o botão esquerdo do mouse sobre o tema do SuperKaramba, isto é, sempre que um sinal *widgetClicked(widget, x, y, button)* é gerados.

Nós não vamos avaliar as coordenadas x-y e o número do botão do mouse pressionado, mas se fizéssemos, poderíamos tratar áreas diferentes dentro de tema de forma distinta, ou distinguir entre os botões esquerdo, direito e central do mouse. Poderíamos até mesmo mapear um menu ao botão direito do mouse, mas como um clique com este botão

Listagem 1 : Tema com suporte a drag & drop

```
01 import karamba
02 import string
03
04 def initWidget(widget):
05     karamba.acceptDrops(widget)
06
07 def itemDropped(widget, dropText):
08     xmms_filename=""
09     for filename in string.split(string.rstrip(str(dropText)),"\n"):
10
11             temp=string.split(filename,"file:", 1)
12             xmms_filename=xmms_filename + " \"" + ➥
                       string.rstrip(temp[1]) + "\""
14     karamba.execute("xmms" + xmms_filename)
15
16 print "Minha extensão Python foi carregada!"
```

GLOSSÁRIO

Variável global: *O paradigma de programação orientada a objetos tende a encapsular funções e variáveis em objetos. Por exemplo, definir em uma variável a cor de um botão facilita a tarefa de criar botões de várias cores, já que a variável é armazenada no objeto "botão". Em contraste, variáveis globais são visíveis em todo o programa. Então, se você atribuir a cor verde à variável, todos os botões que você criar serão verdes.*

Multithreaded: *Normalmente programas não são capazes de executar mais de uma tarefa por vez. Por exemplo, enquanto o programa espera pela abertura de um arquivo , não pode executar mais nenhumas outra tarefa. Threads permitem que um programa execute mmúltiplas tarefas simultaneamente. Em vez de sentar-se e esperar até o arquivo abrir, um programa aproveitaria este tempo em uma outra thread.*

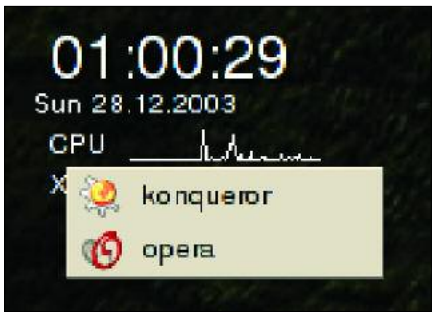


Figura 1: Menus criados com Python.

sobre um tema abre o menu do Super Karamba, isto não faz sentido na prática.

Note que cliques do mouse sempre executam a ação definida no arquivo *.theme*. Em nosso caso, um clique no relógio abre as preferências do relógio e nosso menu (Figura 1).

Para mostrar o menu na tela, precisamos do identificador, que é armazenado na variável *selectmenu*. Podemos usar *global selectmenu* para recuperar esta informação do espaço global de dados. A função *karamba.popupMenu (widget, selectmenu, x, y)* permite que coloquemos o menu nas mesmas coordenadas (relativas ao tema) onde ocorreu o clique do mouse. Se estas coordenadas não forem conhecidas, o menu surge no canto superior esquerdo (coordenadas 0,0) do tema.

Quando o usuário seleciona um item do menu, o SuperKaramba reage a este sinal chamando a função *menuItemClicked(widget, menu, id)*. Esta função usa *id* para fornecer um identificador para o item do menu, e *menu* para nome do item propriamente dito. Como o sistema determina os identificadores de forma arbitrária, não há uma maneira de fácil de adivinhá-los, mas podemos nos livrar rapidamente do problema se os armazenarmos nas variáveis globais *my_menu*, para os itens do menu, e *selectmenu*, para o menu propriamente dito. Como nós temos apenas um menu, não precisamos nos preocupar com esta diferença. Ou seja, a única coisa de que precisamos é da variável *main_ menu*, armazenada no espaço global de dados.

O laço *for index in my_menu:* (linha 20) nos permite iterar através de todos os elementos da variável *my_menu* e mapear as listas nela contidas à variável *index*, uma por uma. Durante cada iteração, precisamos verificar se o identificador do item do menu que foi fornecido pelo sinal confere com o número armazenado no segundo item de *index*:

```
If index[1]==id;
```

Após encontrarmos o par em *my_menu* que corresponde ao item clicado, temos que verificar qual dos navegadores ele representa. Se for o Konqueror...

```
if index[0] == 'konqueror':
```

...o chamamos com *karamba.execute ("konqueror");* caso contrário usamos:

```
elif index[0]=='opera':
```

para ver se o Opera foi o selecionado.

Temas pra Karamba

Se você quer apenas usar o SuperKaramba, sem se aventurar pelo mundo da programação, temas não faltam. Desde itens úteis, como o conjunto de utilitários *TDE* (*The Desktop Enhancements*) que engloba calendário, indicador de espaço em disco, monitor de logs, bloco de notas, estatísticas sobre a máquina, lista de usuários "logados" e estado da sua conexão à rede, até inutilidades absolutas, como o *Station V*, que mostra a estação espacial do filme "2001: Uma Odisséia no Espaço" girando no seu desktop. O ponto de partida para encontrar novos temas é o já conhecido *KDE-Look.org* [11], lotado de ícones, papéis de parede, temas e outros itens para o seu desktop KDE. Já os desenvolvedores do SuperKaramba estão trabalhando no *SuperKaramba Theme Archive* [12], que pretende ser o repositório definitivo de temas para o programa. Apesar de ainda estar em construção e contar com poucos temas, o site merece uma visita.

Listagem 2: Interagindo com menus

```
01 import karamba
02
03 selectmenu=0
04 my_menu=[[],[]]
05
06 def initWidget(widget):
07     global selectmenu
08     global my_menu
09     selectmenu=karamba.createMenu(widget)
10     my_menu=[
11     "konqueror", karamba.addMenuItem(widget, selectmenu, "konqueror", ➥
   "konqueror.png")],
12     ["opera", karamba.addMenuItem(widget, selectmenu, "opera", ➥
   "opera.png")]
13
14 def widgetClicked(widget, x, y, button):
15     global selectmenu
16     karamba.popupMenu(widget, selectmenu, x, y)
17
18 def menuItemClicked(widget, menu, id):
19     global my_menu
20     for index in my_menu:
21         if index[1] == id:
22             if index[0] == 'konqueror':
23                 karamba.execute("konqueror")
24             elif index[0] == 'opera':
25                 karamba.execute("opera")
26
27 print "Minha extensão Python foi carregada!"
```

Se quisermos juntar o conteúdo das Listagens 1 e 2 em um único arquivo, podemos eliminar algumas linhas. Por exemplo, precisamos apenas de uma única instância do comando *import karamba* e um único comando *print* no final. Também precisaremos juntar o conteúdo dos blocos *def initWidget(widget):* (linha 4 na listagem 1, linha 6 na listagem 2). em uma única função. Se uma destas funções estiver faltando após você abrir o tema, basta apagar o arquivo *.pyc*, que é criado automaticamente no seu diretório de temas.

Nos limites do reino das cobras

Embora seja incrivelmente simples realizar tarefas complexas como drag & drop no SuperKaramba, e embora a API Python forneça um grande número de funções adicionais, as quais são documentadas (veja o item [5] no box INFO ao final do artigo), existem alguns poucos problemas com a API. O maior deles é que ela não pode lidar com sensores dinâmicos (veja [1]). Se também quiséssemos adicionar ao nosso tema a capacidade de trocar entre múltiplos sensores, conseguiríamos clicar para atualizar o texto, porém seríamos incapazes de mudá-los automaticamente.

Há outro problema. Python só pode acessar os elementos criados por ele mesmo. Caso você pretenda utilizar um elemento de texto para interagir com uma função, você precisa criá-lo com a função *createText()* do Python. Textos definidos no arquivo *.theme* não podem ser manipulados com a API Python. ■

Figura 2: A TuxBar é um dos temas mais populares

INFORMAÇÕES

[1] Introdução ao SuperKaramba: Hagen Höpfner, "Karamba on the Desktop", Linux Magazine, Edição 41

[2] SuperKaramba: *http://netdragon.sourceforge.net/*

[3] Primeiros passos com Python: *http://www.python.org/doc/Intros.html*

[4] Exemplo de tema: *http://wwwiti.cs.uni-magdeburg.de/~hoepfner/download.html*

[5] Documentação da API Python do SuperKaramba: *http://netdragon.sourceforge.net/api.html*

[6] Template em Python para o SuperKaramba: *http://netdragon.sourceforge.net/template.py*

[7] Funções de manipulação de strings em Python: *http://www.python.org/doc/2.3.3/lib/module-string.html*

[8] Loops e condições em Python: *http://www.python.org/doc/2.3.3/ref/compound.html*

[9] Biblioteca de referência sobre Python: *http://www.python.org/doc/2.3.3/lib/lib.html*

[10] Página oficial do SuperKaramba: *http://nedtdragon.sf.net*

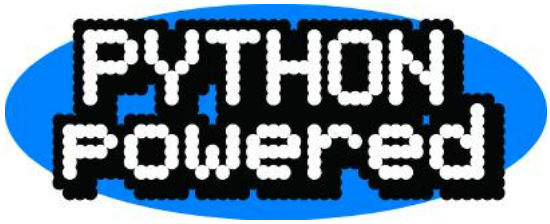
[11] KDE Look: *http://www.kde-look.org/*

[12] SuperKaramba Theme Archive: *http://www.superkaramba.com/*

[13] Python: *http://www.python.org/*

[14] GDesklets: *gdesklets.gnomedesktop.org/*

O que é Python?



Se você já conhece outra linguagem de programação, talvez não fique muito entusiasmado ao descobrir que deverá aprender uma nova linguagem só para fazer o Super Karamba feliz. Mas para os iniciantes em programação, Python tem algumas vantagens, sendo uma linguagem universal. Um dos principais fatores é que Python combina os três principais paradigmas de programação atuais em um só. Python pode ser utilizado como uma linguagem orientada a objeto, como C++ ou Java, ou como uma linguagem orientada a procedimentos, como Pascal ou C. Ao mesmo tempo, é uma linguagem de script como PHP ou Perl, e seus programas sequer precisam ser compilados pelo usuário. Também suporta múltipla herança, coneta-se a bancos de dados e pode usar vários protocolos de comunicação.

Assim como Java, o código-fonte de um programa em Python é convertido em *bytecode* pelo compilador. Este código também é executado por uma máquina virtual, mas Python é bem mais rápido que Java.

Se você está migrando de Python para outra linguagem de programação, há uma coisa sobre a qual você precisa saber: como funciona a definição de blocos de código. Blocos são utilizados para definir trechos de código que serão executados se uma condição por preenchida. C utiliza chaves ({ }) para isto, e nos Shell Scripts para o Bash um bloco iniciado com o comando *if_then* termina com o comando *fi*. Pascal utiliza as palavras chave *Begin* e *End*. Python não utiliza nenhum dos métodos citados acima: Ao invés de uma marcação específica, você simplesmente indenta o código.

Assim, duas linhas sucessivas com o mesmo nível de indentação formam um bloco de código. Se uma segunda linha começa mais à esquerda que a primeira, ela não pertence ao bloco. Um sinal de dois pontos ao final de uma linha indica um novo nível, como pode ser visto nas listagens de código deste artigo. Após uma linha com a palavra-chave *def*, linhas consecutivas com o mesmo nível de indentação são consideradas como parte da definição da função.

A exigência de indentação garante que código em Python será sempre bem estruturado e fácil de ler, o que nem sempre acontece em outras linguagens. Mas isto pode distrair os iniciantes e levá-los a cometer erros de indentação, pois uma função com nível errado fica simplesmente fora do contexto do bloco onde reside. Fique atento ao criar suas próprias extensões!

SOBRE O AUTOR

Hagen Höpfner é formado em Ciências da Computação, e membro da equipe científica da Universidade Otto-von-Guericke em Magdeburg, na Alemanha. Em seu tempo livre ele gosta de fuçar na sua coleção de computadores com Linux, mas também dirige sua energia creativa para a composição de letras e melodias. Hagen também toca guitarra em uma banda chamada "Gute Frage!?"