

Desarrollo de aplicaciones con Kiwi2 y Gazpacho

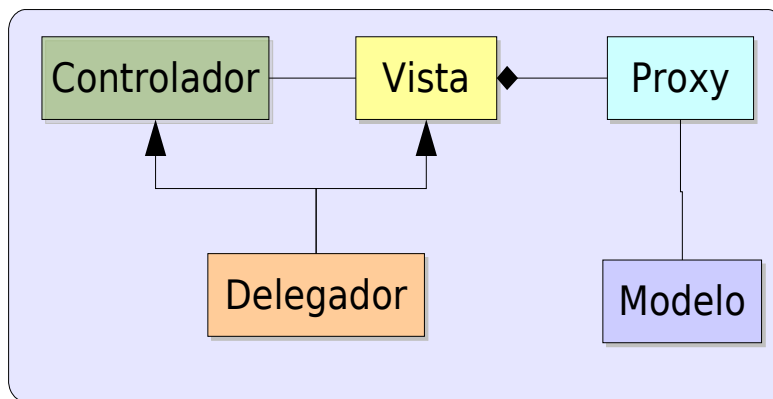
Lorenzo Gil Sánchez
lgs@sicem.biz

SICEm. Software Integrado para el Control de la Empresa.
Jorge Guillén, 19, Ático, 18220 Albolote (Granada)

Resumen: Kiwi2 es un marco de trabajo que facilita el desarrollo rápido de aplicaciones gráficas y que utiliza el toolkit GTK+. Ofrece conceptos de alto nivel de abstracción basados en la arquitectura MVC (Modelo-Vista-Controlador) con algunas mejoras. También contiene un conjunto de controles gráficos potentes y cómodos para el desarrollador y el usuario. Gazpacho es un diseñador de interfaces de usuario que hace hincapié en la usabilidad y soporte de las últimas tecnologías de GTK+. Ambas herramientas están escritas en el lenguaje de programación Python. En este artículo se describen los principales conceptos de Kiwi2 y también las ventajas de Gazpacho respecto a otros sistemas de diseño de interfaces. Se facilitan ejemplos sencillos para mejorar la comprensión final.

1. El marco de trabajo Kiwi2

1.1 Introducción



Kiwi2 es un paquete para Python que facilita enormemente el desarrollo de aplicaciones gráficas en este lenguaje. Kiwi2 funciona sobre GTK+ 2.4 o posterior y es la evolución de Kiwi1 (o simplemente Kiwi) que funciona sobre GTK+ 1.2. Kiwi2

no es simplemente la adaptación de Kiwi1 para GTK+ 2 sino que en su desarrollo se han rediseñado sus componentes internos manteniendo casi por completo la interfaz externa o API.

Kiwi es software libre (LGPL) desarrollado por la compañía brasileña Async¹. Esta segunda versión de Kiwi se ha desarrollado en colaboración con la empresa española Sicem².

Kiwi es una librería que ha surgido como un proceso natural de abstracción de un código que venía siendo utilizado una y otra vez en aplicaciones empresariales desarrolladas por Async. Con el objetivo principal de la reutilización de código se hizo un estudio de las similitudes y relaciones del código que se repetía en dichas aplicaciones a nivel de interfaz de usuario. De esta manera se formalizaron los conceptos e ideas que dan forma a Kiwi: Vistas, Controladores, Delegadores, Proxys y Modelos.

Un proyecto similar es Bakery, de Murray Cumming, que pretende alcanzar objetivos similares de reutilización de componentes de interfaces gráficas para programas escritos en C++.

A continuación se explican estos conceptos, básicos para entender el funcionamiento de una aplicación que utilice Kiwi.

1.2 Vistas (Views)

Una Vista es una parte de la interfaz de la aplicación que ve y manipula el usuario final. En Kiwi2 hay dos tipos de vistas, vistas normales (BaseView) y vistas *esclavas* (SlaveView). Una vista normal contiene una ventana y, habitualmente, controles dentro de ella. Una vista esclava es un conjunto de controles agrupados según un criterio común, normalmente debido a que son controles que manipulan atributos del mismo objeto en nuestra aplicación.

Los dos tipos de vistas pueden contener a su vez subvistas esclavas. De esta forma se introduce el concepto de reutilización de partes de la interfaz de usuario. Por ejemplo, en una aplicación de gestión habitualmente es necesario visualizar los datos asociados a un cliente. Se puede diseñar un formulario con dichos datos y utilizar una vista esclava a partir de él. Una vez hecho esto, esta vista se puede utilizar sin problemas desde cualquier parte de la aplicación, incrustada en otras vistas más específicas.

Las vistas de Kiwi pueden crear sus controles manualmente (escribiendo la definición de los controles en el código del programa) o a partir de un fichero Glade que puede ser generado por Glade-2 o bien por Gazpacho. Más adelante se explicarán las ventajas de diseñar las interfaces con Gazpacho.

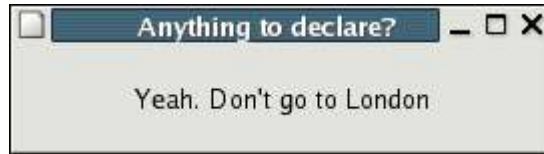
A continuación se muestra un ejemplo del uso de vistas y archivos glade:

```
01 from Kiwi2 import Views
02 from Kiwi2.initgtk import gtk, quit_if_last
03
04 app = Views.BaseView(gladefile="hey", delete_handler=quit_if_last)
05 app.show()
```

1 <http://www.async.com.br>

2 <http://www.sicem.biz>

```
06 gtk.main()
```



En este ejemplo se está cargando la interfaz de usuario de un archivo denominado `hey.glade` y como se puede ver sólo consiste en una ventana con una etiqueta de texto.

1.3 Controladores (Controllers)

Para que una vista sea útil es necesario asociar comportamiento a los distintos eventos que el usuario genera mediante su interacción con la interfaz. Por ejemplo, normalmente se guardan los datos que el usuario introduce en los formularios de la aplicación o se realizan cálculos cuando se pincha en los botones. En la terminología de aplicaciones gráficas esto se consigue asociando funciones a dichos eventos. Estas funciones o retrollamadas de eventos (event callbacks) serán ejecutadas cuando se produzcan los eventos correspondientes. En GTK+ a los eventos se les suele denominar señales y aunque no son exactamente lo mismo, en este documento no es necesario resaltar sus diferencias.

En Kiwi el programador escribe sus retrollamadas como métodos de una subclase de la clase `Controller`. Esta clase, junto con la clase `View` permiten asociar eventos de los controles con métodos de una forma muy sencilla. La asociación se realiza en base al nombre del método siguiendo la siguiente sintaxis:

```
[on|after]_nombre_del_control__nombre_de_la_señal
```

Por ejemplo, si el método `on_button1__clicked` existe, cuando el usuario haga click en el botón “button1” el método será ejecutado automáticamente. Esta forma de asociar señales con código ejecutable tiene las siguientes ventajas:

- Uniformidad en el estilo de las retrollamadas
- Legibilidad de dichas retrollamadas. Solo viendo el nombre del método nos hacemos una idea de cuándo será llamado
- Facilidad de programación. No es necesario hacer la conexión entre los controles, las señales y las retrollamadas ya que Kiwi lo hace internamente.

A continuación se muestra un ejemplo del uso de un controlador:

```
01 from Kiwi2 import Views, Controllers
02 from Kiwi2.initgtk import gtk, quit_if_last
03
04 class FarenControl(Controllers.BaseController):
05     def __init__(self, view):
06         Controllers.BaseController.__init__(self, view)
07
08     def on_quitbutton__clicked(self, *args):
09         self.view.hide_and_quit()
10
11     def after_temperature__changed(self, entry, *args):
```

```

12     try:
13         temp = float(entry.get_text())
14     except ValueError:
15         temp = 0
16     celsius = (temp - 32) * 5/9.0
17     fahrenheit = (temp * 9/5.0) + 32
18     self.view.celsius.set_text("%.2f" % celsius)
19     self.view.fahrenheit.set_text("%.2f" % fahrenheit)
20
21 widgets = ["quitbutton", "temperature", "celsius", "fahrenheit"]
22 view = Views.BaseView(gladefile="faren", delete_handler=quit_if_last,
23                      widgets=widgets)
24 ctl = FarenControl(view)
25 view.show()
26 gtk.main()

```



En las líneas 21-22 se puede que se están nombrando explícitamente una serie de controles y dicha lista se le pasa al constructor de la Vista. En el archivo glade se han definido esos nombres para algunos controles de la interfaz y el objetivo es poder acceder a dichos controles como si fueran miembros de nuestra clase vista.

En las líneas 8 y 10 se definen dos métodos con el formato de nombre que se ha explicado anteriormente. Es fácil intuir cuándo serán llamados y el efecto que producen.

1.4 Delegadores (Delegates)

Uno de los problemas de la arquitectura MVC es que la separación entre la Vista y el Controlador dificulta la programación de ambas clases y, habitualmente, no supone una ventaja en cuanto a la reutilización de componentes ya que las retrollamadas en el Controlador suelen estar muy ligadas a la Vista particular con la que trabaja. En otras palabras, normalmente es necesario que la Vista ofrezca una API para sus controles que el Controlador utiliza y para casos no triviales supone un exceso de trabajo sin ventajas prácticas.

Para resolver este problema en Kiwi existe la clase Delegate que no es más que una clase que hereda tanto de View como de Controller. En una aplicación que utiliza Kiwi, la mayoría de las clases son subclases de Delegate.

En la clase Delegate se colocan los controles (View) y el código asociado a ellos

(Controller) y si bien esto parece ser contraproducente en cuanto la separación que aboga la arquitectura MVC, lo cierto es que, en la práctica, se consiguen componentes más reutilizables con esta filosofía.

Veamos el ejemplo anterior de conversor de temperatura utilizando un delegador:

```
01 from Kiwi2 import Delegates
02 from Kiwi2.initgtk import gtk, quit_if_last
03
04 class Farenheit(Delegates.Delegate):
05     widgets = ["quitbutton", "temperature", "celsius", "farenheit",
06               "celsius_label" , "farenheit_label",
07               "temperature_label"]
08     def __init__(self):
09         Delegates.Delegate.__init__(self, gladefile="faren",
10                                     delete_handler=quit_if_last)
11
12     def convert_temperature(self, temp):
13         farenheit = (temp * 9/5.0) + 32
14         celsius = (temp - 32) * 5/9.0
15         return farenheit, celsius
16
17     def clear_temperature(self):
18         self.farenheit.set_text("")
19         self.celsius.set_text("")
20
21     # Signal handlers
22     def on_quitbutton__clicked(self, *args):
23         self.hide_and_quit()
24
25     def after_temperature__changed(self, entry, *args):
26         temp = entry.get_text().strip() or None
27         if temp is None:
28             self.clear_temperature()
29         else:
30             farenheit, celsius = self.convert_temperature(float
31                                 (temp))
32             self.farenheit.set_text("%.2f" % farenheit)
33             self.celsius.set_text("%.2f" % celsius)
34
35 delegate = Farenheit()
36 delegate.show()
37 gtk.main()
```

Se puede ver que aunque el hay un poco más de código el diseño es mucho más limpio y legible ya que hemos agrupado la vista y el controlador en una sola clase. De esta forma conseguimos que se pueda reutilizar esta clase mucho más cómodamente ya que constituye un componente cerrado sin dependencias externas a otras clases como vistas o controladores externos.

1.5 Proxys (Proxies)

En Kiwi, un proxy es un componente que se utiliza para sincronizar el estado de la interfaz de usuario (View) con el estado de los objetos de dominio que manipula la aplicación. Un Proxy es el responsable de que cuando el usuario modifica los controles en un formulario estos cambios se propaguen al modelo automáticamente y sin necesidad de escribir una sola línea de código. Esta propagación también se

realiza en la dirección opuesta, es decir, cuando la aplicación modifica los objetos de dominio en procedimientos internos, la interfaz se actualiza automáticamente. Es fácil comprender que los proxys son el concepto más potente de Kiwi.

En Kiwi2 para poder utilizar proxys es necesario utilizar los controles específicos de Kiwi. Con controles nativos de GTK+ un Proxy no funcionará pero esto no es un problema porque existen controles en Kiwi para casi todos los controles GTK+ que se refieren a la manipulación de datos como cajas de texto, botones de activación, listas, listas desplegadas, etc. Más adelante se explican otras ventajas asociadas al uso de controles específicos de Kiwi.

Al actualizar el modelo, se actualiza la interfaz, pero al actualizar la interfaz también se actualiza el modelo. Para evitar bucles infinitos debido a esta actualización en cadena, es necesario que el programador utilice la API de Kiwi en su código. Afortunadamente esta restricción no supone ningún esfuerzo extra ya que dicha API es clara y está bien documentada.

1.6 Modelos (Models)

Los modelos son los objetos que contienen los datos específicos del dominio de nuestra aplicación. Junto con los Proxys, son los responsables de la actualización automática y bidireccional de las aplicaciones que utilizan Kiwi. Otra gran ventaja de su uso es que pueden ofrecer persistencia automática para los datos. Por ejemplo, en Kiwi hay una subclase de Model, PickledModel que salva los datos en disco automáticamente utilizando el módulo pickle de Python. Es muy sencillo escribir otras clases que hagan lo mismo en archivos XML o CSV. Async ha desarrollado aplicaciones que utilizan modelos que guardan los datos en bases de datos ZODB. Es fácil imaginar modelos que utilicen bases de datos relacionales como Mysql o Postgresql bien directamente o bien utilizando librerías más abstractas como libgda, sqlobject o Modeling.

A continuación se muestra un ejemplo de un formulario para editar noticias. En el ejemplo, una noticia tiene un título, un autor, una url y un tamaño.

```
01 from Kiwi2 import Views
02 from Kiwi2.initgtk import gtk, quit_if_last
03
04 class NewsItem:
05     """An instance representing an item of news.
06     Attributes: title, author, url, size"""
07     pass
08
09 item = NewsItem()
10 my_widgets = ["title", "author", "url", "size"]
11 view = Views.BaseView(gladefile="newsform", widgets=my_widgets,
12                      delete_handler=quit_if_last)
13 view.add_proxy(item, my_widgets)
14 view.show()
15 gtk.main()
16
17 print 'Item: "%s" (%s) %s %d' % (item.title, item.author, item.url,
18                               item.size)
```

En las líneas 4-7 se define un modelo extremadamente simple pero que servirá para

ilustrar el funcionamiento del Proxy. En la línea 13 hacemos la conexión entre interfaz y modelo añadiendo un proxy a la vista. Esto significa que a partir de ese momento cualquier cambio que se produzca en la interfaz será reflejado en el modelo y viceversa. Por eso, cuando la aplicación termina y se imprimen los valores de la noticia se ven los mismos datos que el usuario ha introducido en la interfaz.



2. Los controles de Kiwi2

Los controles específicos de Kiwi entienden el protocolo utilizado por los Proxys para leer y escribir datos en la interfaz y es por ello que son necesarios para que funcionen el mecanismo de actualización automática de Kiwi.

Todos los controles de Kiwi tienen una propiedad llamada 'model-attribute' y 'data-type' que son utilizados por el Proxy para saber qué atributo del modelo el control está representando.

Los controles de Kiwi tienen otra razón de ser: suelen ser mucho más fáciles de usar tanto por el programador como por el usuario. Para el programador, controles como GtkTreeView, GtkComboBox o GtkTextView son extremadamente complejos de utilizar. Dicha complejidad se debe a que son controles muy potentes y flexibles, pero en la vida real, el 90% de las veces que se utilizan es para un caso de uso concreto que se repite y que no es fácil de implementar para un programador sin mucha experiencia. Kiwi ofrece controles basados en estos controles de GTK+ pero sin este problema de dificultad de programación.

Para el usuario final los controles de Kiwi son fáciles de usar porque se ha hecho especial énfasis en la usabilidad. Por ejemplo, el control de caja de texto de Kiwi contiene funcionalidad para validación automática según el tipo de dato que manipula. El control de lista ofrece funcionalidad para ordenar los datos según una columna concreta y también puede mostrar u ocultar columnas fácilmente, entre otras cosas. Como este tipo de funcionalidad está dentro de los controles de Kiwi el programador se ahorra gran cantidad de trabajo y la calidad de las aplicaciones es mayor.

Algunos de estos controles son:

- Lista (List). Esta basada en GtkTreeView pero es mucho más sencilla de programar ya que lo único que hay que especificar es las columnas que contiene.

Para cada columna se especifica el atributo de los objetos que contendrá la lista que esta columna visualiza y varias opciones más. Se pueden añadir, borrar y modificar objetos en la lista con una sintaxis similar a las listas de Python.

- Lista desplegable (ComboBox). Basada en GtkComboBox pero con facilidades de programación similares a las de la Lista.
- Caja de Texto (Entry). Está basada en GtkEntry y le añade un marco de validación potente y flexible. También permite recuperar el valor anterior que contenía la caja con una simple pulsación de tecla.
- Texto (Text). Basado en GtkTextView pero simplificado en gran parte sesgando gran parte de las características del widget de GTK+ en aras de la facilidad de uso.
- Botón biestado (CheckBox), Grupos de botones biestado (RadioButtons), Etiqueta (Label), etc.

3. Diseño de interfaces con Gazpacho

Gazpacho es un diseñador de interfaces de usuario que intenta solucionar los problemas de Glade. Dichos problemas se agrupan en dos grandes grupos principalmente:

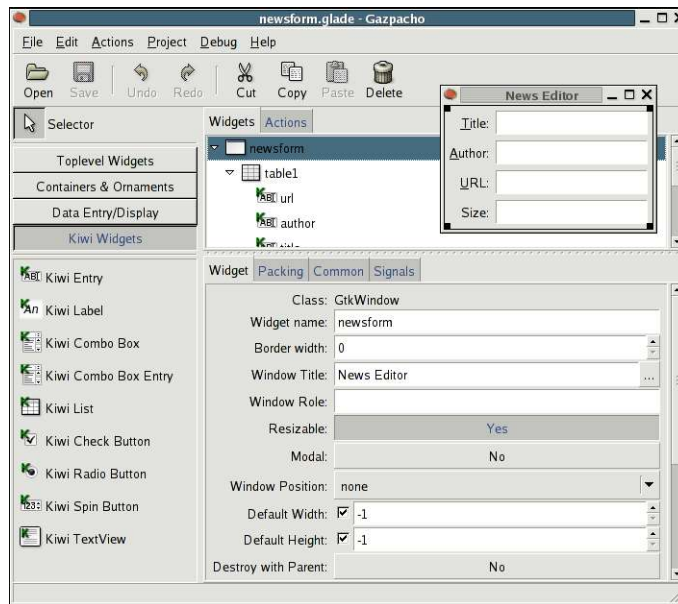
- Problemas de usabilidad: La disposición en múltiples ventanas de Glade hace que sea complicado de utilizar, los botones de la paleta no reflejan con claridad su cometido, no se pueden deshacer acciones, ...
- Problemas de soporte de GTK+: Las últimas mejoras en GTK+ no están soportadas en Glade, por lo que un programador que desee utilizar Glade tendrá que optar por a) no usar estas mejoras o b) escribir parte de su interfaz en el código. Algunas de estas mejoras son los nuevos menús y barras de herramientas centrados en el concepto de Acción, la agrupación de controles en SizeGroup para obtener una disposición más homogénea, la nueva lista desplegable basada en el mismo modelo que el GtkTreeView, ... Es justo decir que parte de estos problemas se deben a las limitaciones de la librería libglade.

La otra gran razón para usar Gazpacho es que permite utilizar controles escritos en Python nativamente ya que Gazpacho esta escrito en este lenguaje. También soporta controles escritos en C gracias a la colaboración de Imendio³.

Otras características que están en desarrollo son el soporte de Plantillas de controles y de plugins que permitan cosas tan interesantes como la creación automática de interfaces basándose en fuentes de datos externas como el esquema de una base de datos.

Actualmente Gazpacho goza de una estupenda actividad con 15 personas contribuyendo código al proyecto, multitud de fallos resueltos cada semana y nuevas versiones prácticamente cada dos semanas.

3 <http://www.imendio.com>



En el futuro se planea utilizar Gazpacho como diseñador de Informes escribiendo una serie de componentes adicionales:

- Una librería de controles que representen los elementos comunes en un informe: páginas, etiquetas, imágenes, párrafos, encabezados, pies de página, listas de agrupación, etc.
- Una librería que acepte un fichero glade con la definición del informe y una fuente de datos para rellenar dicho informe. Esta forma de trabajo es exactamente igual a cómo funcionan los sistemas tradicionales de plantillas como Cheetah (Python), Smarty (PHP) o Velocity (Java). A partir del fichero glade y los datos se obtendría una representación lista para imprimir (PDF, Postscript, ...)

4. Integración de Kiwi2 y Gazpacho

Gracias al soporte de controles escritos en Python de Gazpacho, es posible utilizar los controles de Kiwi directamente en Gazpacho como si fueran otros controles más de GTK+. Aparecen en su propia pestaña en la paleta de controles y pueden mezclarse con los demás controles en la interfaz que se está diseñando.

Actualmente Kiwi utiliza un módulo de Gazpacho para cargar los ficheros Glade aunque se espera que en el futuro esto no sea necesario ya que se está trabajando en incluir una librería que haga exactamente eso dentro de GTK+.

5. Conclusión

Kiwi y Gazpacho ofrecen numerosas ventajas al programador que le permiten desarrollar sus aplicaciones en un tiempo récord y con gran calidad:

- Estructuración del código de la aplicación de una forma clara y lógica.
- Sincronización automática entre la interfaz y los objetos del dominio.
- Fácil integración con distintos backends para el almacenamiento de los datos.
- Controles avanzados sencillos de utilizar y de programar.
- Reutilización de componentes no sólo a nivel de código sino también a nivel de interfaz de usuario.

Todo esto unido a la potencia y sencillez del lenguaje de programación Python hacen de este entorno una elección ideal para el desarrollo rápido de aplicaciones.

El principal inconveniente de Kiwi y Gazpacho es la falta de madurez de estos proyectos ya que aún no han recibido la masa crítica necesaria para que dichas tecnologías adquieran un alto grado de estabilidad. Además, al estar siendo desarrolladas en la actualidad muchos elementos se rediseñan a diario.

6. Referencias

[1] Christian Reis. Developing applications with Kiwi. 2002.
<http://www.async.com.br/projects/kiwi/howto>

[2] Allen Holub. Building user interfaces for object-oriented systems. 1999.
<http://www.javaworld.com/javaworld/jw-07-1999/jw-07-toolbox.html>

[3] Proyecto Glade. <http://glade.gnome.org>

[4] Documentación técnica de GTK+. <http://www.gtk.org/api>

[5] Murray Cumming. Bakery. <http://bakery.sourceforge.net>