

# **Interfaces gráficas en Haskell**

# Índice

- 1. WxHaskell
  - 1.1. Introducción a WxHaskell
  - ¿Por qué usar wxHaskell?
  - 1.2. Visión genérica de wxHaskell.
  - 1.3. Ejemplos
- 2. Ap.GUI
  - 2.1. Introducción a Ap.GUI
  - 2.2. Ejemplos

# Introducción a WxHaskell

- Es una librería de interfaces gráficas para Haskell de código abierto
- Construída sobre la librería wxWidgets(plataforma GUI desarrollada para C++). Soporta el 75% de su funcionalidad.
- Comenzó su desarrollo en el año 1992 declarada código abierto en el año 2009.
- Es una librería derivada de wxEiffel.
- Ofrece un alto nivel de abstracción, polimorfismo paramétrico

# ¿Por qué usar WxHaskell?

Creación rápida de prototipos, sobre los que seguir desarrollando.

Aplicaciones comerciales

Una librería multiplataforma que usa la filosofía look-feel.

Se integra con código Haskell

Porque es sencillo su programación

Ofrece seguridad en gestión de memoria

# Seguridad en WxHaskell

Ofrece un chequeador de tipos que impide realizar operaciones ilegales de widgets.

La gestión de la memoria es totalmente automática.

En lugar de provocar segmentaciones de memoria se lanzarán excepciones.

El estricto orden jerárquico permite hacer una gestión de la memoria de forma explícita.

# Visión genérica de WxHaskell

Paquetes

Control

Tipos y herencia.

Eventos

Atributos y propiedades.

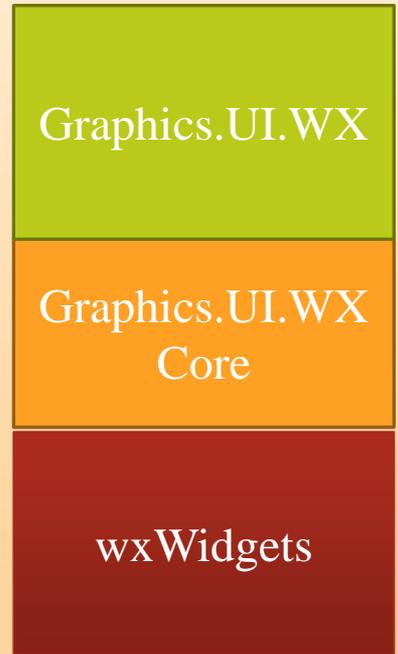
# Visión genérica. Paquetes WxHaskell

## Graphics.UI.WXCore

- Interfaz de bajo nivel con la librería wxWidgets
- Relación uno-a-uno entre C++ y Haskell

## Graphics.UI.WX

- Construido en la cima de WXCore
- Proporciona buena abstracción funcional (atributos, diseño, etc)



# Controles

```
p <- panel []
```

```
txt <- textEntry p AlignLeft [text := "your name here"]
```

```
cb <- comboBox p true ["NSW", "ACT", "VIC", "WA"] []
```

```
rd <- radioButton p Horizontal ["one", "two"]  
  [on select := logSelection]
```

Otras: Choice, ListBox, Slider, Toolbar

# Tipos y herencia en WxHaskell

- Codifica relación de herencia entre tipos diferentes usando TAD

Object (Ptr)

|- ..

|- Window

|- Frame

|- Control

|- Button

|- RadioBox

Button a == Ptr (... (CWindow (CControl (CButton a))) ...)

# Visión genérica. Atributos en WxHaskell

Podemos controlar los diversos atributos de los widgets, por ejemplo, título, color, fuente, tamaño, etc

Pero, ¿qué atributos se pueden utilizar con qué widget?

- Los atributos se organizan en clases de Haskell
- Tipos de reproductores instancias de clases correspondientes.
- Heredar definiciones ejemplo, de tipo "padre"

# Visión genérica. Atributos en WxHaskell

Type Frame a = Window (CFrame a)

Frame es instancia de HasImage, Form, Closable, y de cualquier otra instancia de Window

Window es una instancia Textual, Literate, Dimensions, ...

La clase HasImage define el atributo 'image',

La clase Textual define the atributo 'text'.

Entonces, podremos hacer lo siguiente:

```
f <- frame []
```

```
set f [text := "Window Title", image := "/some/image.ico"]
```

# Eventos en WxHaskell

- Se organizan en clases.
- Un widget que instancia de una clase evento significa que puede recibir eventos de esa clase.
- Los manejadores de eventos pueden ser definidos en un atributo usando la función «on»:

```
paint :: (Paint w) => Event w (DC () -> Rect -> IO ())
```

Window es una instancia de Paint, nos permite definir nuestra propia rutina paint para todo tipo de ventanas (incluyendo botones y cajas de texto).

```
set f [on paint := drawObjects]
```

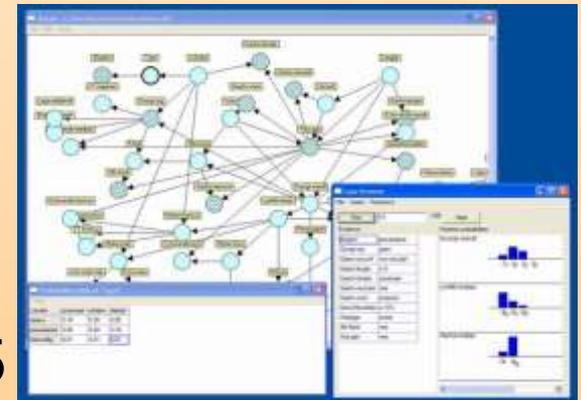
# Visión genérica. Diseño (Layout)

- Administra el posicionamiento y dimensionamiento de los artilugios de un contenedor widget.
- wxHaskell utiliza combinadores layout que permite un estilo más declarativo para especificar la disposición  
El tipo de retorno de un combinador de layout es siempre Layout
- Permite un control preciso del comportamiento cuando la ventana cambia de tamaño (o para evitar el cambio de tamaño)
- Los tipos de combinadores de layout son: diseños (widgets, contenedores, pegamento, separadores) y transformadores (estiramiento, ampliar márgenes, alineación)



# Aplicaciones importantes de WxHaskell

- Dazzle: Editor gráfico para redes bayesianas. Desarrollado por la universidad de Utrecht.
- Dazzle usa la librería Smile y algoritmos en C++.
- Presenta datos estadísticos multidimensionales.
- Última versión del 15 de octubre de 2005



# Aplicaciones importantes de WxHaskell

- Simulador de colonias de hormigas. Usado en la resolución de pistas para la resolución de pistas.
- La interfaz se desarrolló en un solo día. Ganaron el concurso ICFP Programming Contest 2003 y 2004
- Usa fragmentos de código C para programar sobre wxWidgets.

# Ejemplo

```
main = start gui  
gui :: IO ()  
gui =
```

```
do f ← frame [text := "Example"]  
    lab ← label f [text := "Hello wxHaskell"]  
    ok button f [text := "Ok"]  
    can ← button f [text := "Cancel"]  
    set ok [on command := close f]  
    set can [on command := set lab [text := "Goodbye?"]]  
    set f [layout := column 5 [floatCenter (widget lab)  
                              ,floatCenter $  
                                row 5 [widget ok,widget can]]]
```



# Ap.Gui

- Fue realizada por Pepe Gallardo, profesor de la universidad de Málaga.
- Conserva la misma filosofía que wxHaskell y HToolkit
- Usada en la docencia.
- Trabaja sobre GHC
- Programación más intuitiva.
- No trabaja sobre winhugs

# Atributos

- Cada uno de los atributos se separan por comas.
- Las asignaciones de cada atributo se hace usando el operador =:

```
button [ text =: "Button" ] w
```

Se suprime el uso de la directiva get y set

Las rutinas se especifican igualmente con «on»:

```
b4 ← button [ text =: "Quit", on action =: quit
```

# Layout

- El layout se declara de una forma más intuitiva
- Para tener a un widget que aparezca con alineamiento horizontal de otro se declara de la siguiente forma:

➤ `w!:layout =: b1 <.< b2 <.< b3 <.< b4`

Para tener a un widget que aparezca con alineamiento vertical de otro se declara de la siguiente forma:

➤ `w!:layout =: b1 ^.^ b2 ^.^ b3 ^.^ b4`

Es posible combinar ambos alineamientos:

`w!:layout =: b1 <.< b2 ^.^ b3 <.< b4`

# Layout

- Cada widget puede ser dimensionado de forma individual usando size:
  - `c ← canvas [ size =: 400 400, bgColor =: white ] w`
- Cada widget tiene su propio contenedor, salvo la ventana:
  - `w ← window [ title =: "window" ]`
    - `; b1 ← button [ text =: "Button"`
      - `, bgColor =: blue] w`

# Comparativa

Ventajas	Desventajas
wxHaskell: Muy difundido en ámbito docente y comercial	Ap.GUI: solo usado en el ámbito docente
Ap.GUI: tiene una curva de aprendizaje más suave	wxHaskell: Es una librería menos intuitiva
	wxHaskell: Funciona sobre una librería ajena y de gran tamaño
wxHaskell: Trabaja sobre cualquier compilador	Ap.GUI: solo trabaja con el compilador ghci