

# $\lambda Page$

*Fernando Benavides*

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

17 de julio de 2010

## *El Orador*

- ▶ Fernando Benavides

## *El camino recorrido*

- ▶ Alumno de Computación desde 2001
- ▶ Programador desde hace más de 10 años
- ▶ Programador *Funcional* desde hace 2 años

## *La idea*

- ▶ Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

## *El Orador*

- Fernando Benavides

## *El camino recorrido*

- Alumno de Computación desde 2001
- Programador desde hace más de 10 años
- Programador *Funcional* desde hace 2 años

## *La idea*

- Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

## *El Orador*

- ▶ Fernando Benavides

## *El camino recorrido*

- ▶ Alumno de Computación desde 2001
- ▶ Programador desde hace más de 10 años
- ▶ Programador *Funcional* desde hace 2 años

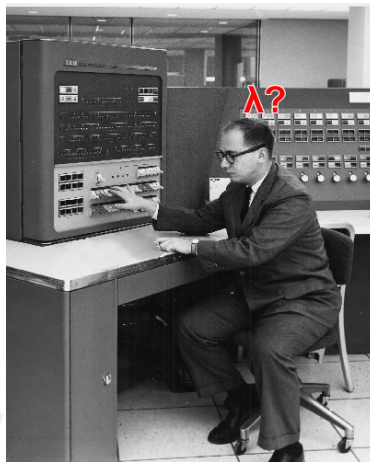
## *La idea*

- ▶ Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

# Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

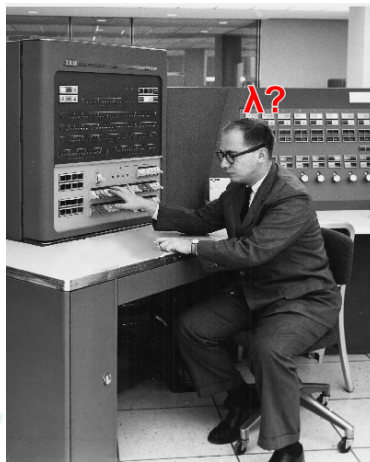
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



# Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

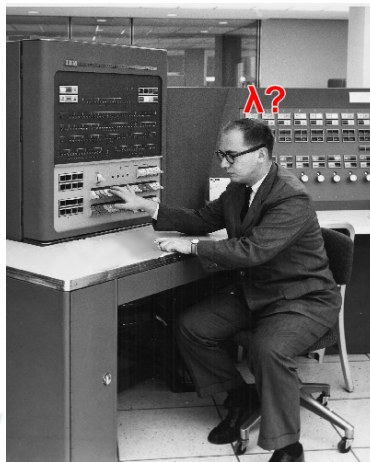
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



# Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

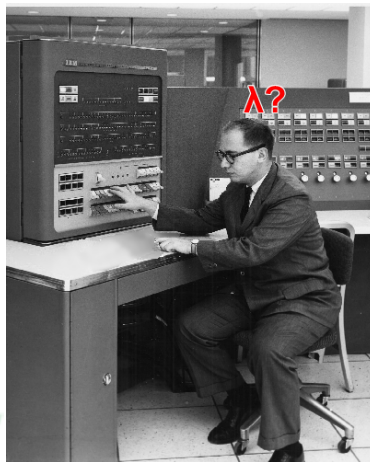
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



# Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*





# Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Herramientas de *refactoring*
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

# Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Herramientas de *refactoring*
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

# Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Herramientas de *refactoring*
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

# Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Herramientas de *refactoring*
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

# Trabajando con Lenguajes Orientados a Objetos



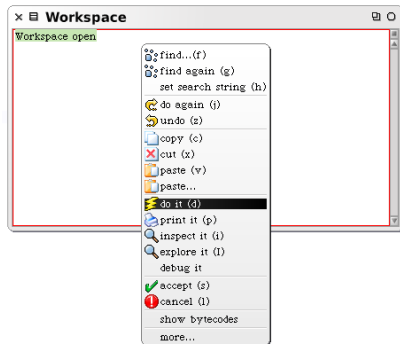
En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Herramientas de *refactoring*
- ▶ Debugger integrado
- ▶ Herramientas para “*micro-testing*”

## “Micro-testing”

El *Workspace* de *Smalltalk* permite:

- ▶ introducir código para ejecutarlo, inspeccionarlo y analizar los resultados obtenidos
- ▶ administrar varias paginas de texto
- ▶ crear objetos y utilizarlos



# *Conociendo $\lambda$ Page*

$\lambda$ Page es similar al *Workspace* de *Smalltalk* pues permite al usuario

- ▶ Evaluar expresiones
- ▶ Detectar excepciones
- ▶ Administrar páginas de texto libre
- ▶ Intercalar expresiones y definiciones



Pero, a su vez, por estar hecho para *Haskell*, toma en cuenta sus características:

- ▶ *Lazy evaluation*
- ▶ Acciones con efectos colaterales
- ▶ Administración de módulos

Pero, a su vez, por estar hecho para *Haskell*, toma en cuenta sus características:

- ▶ *Lazy evaluation*
- ▶ Acciones con efectos colaterales
- ▶ Administración de módulos

Pero, a su vez, por estar hecho para *Haskell*, toma en cuenta sus características:

- ▶ *Lazy evaluation*
- ▶ Acciones con efectos colaterales
- ▶ Administración de módulos

# Lazy Evaluation

# Efectos Colaterales

# Administración de Módulos

Utilizando  $\lambda$ Page, el usuario puede

- ▶ Importar módulos
- ▶ Cargar módulos
- ▶ Recargar módulos

Todo esto *sin perder las expresiones que ya tiene definidas*

# $\lambda$ Page *por Dentro*

# Conceptos Centrales

$\lambda$ Page está desarrollado en *Haskell*

En gran parte está desarrollado utilizando  $\lambda$ Page

Se conecta con *GHC* a través de su API

Su interfaz gráfica fue creada usando *wxHaskell*



# Conceptos Centrales

$\lambda$ Page está desarrollado en *Haskell*

En gran parte está desarrollado utilizando  $\lambda$ Page

Se conecta con *GHC* a través de su API

Su interfaz gráfica fué creada usando *wxHaskell*

# Conceptos Centrales

$\lambda$ Page está desarrollado en *Haskell*

En gran parte está desarrollado utilizando  $\lambda$ Page

Se conecta con *GHC* a través de su API

Su interfaz gráfica fué creada usando *wxHaskell*

# Conceptos Centrales

$\lambda$ Page está desarrollado en *Haskell*

En gran parte está desarrollado utilizando  $\lambda$ Page

Se conecta con *GHC* a través de su API

Su interfaz gráfica fué creada usando *wxHaskell*

# Paralelismo

Muchas cosas suceden simultáneamente en  $\lambda$ Page

- ▶ Manejo de Páginas (crear, abrir, modificar, guardar, etc.)
- ▶ Interpretación de Expresiones
- ▶ Evaluación de Acciones de Entrada/Salida

Para lograrlo, creamos *eprocess*:

- ▶ Basada conceptualmente en *Erlang*
- ▶ Construída utilizando *Threads*, *Channels* y *MVars*

# Paralelismo

Muchas cosas suceden simultáneamente en  $\lambda$ Page

- ▶ Manejo de Páginas (crear, abrir, modificar, guardar, etc.)
- ▶ Interpretación de Expresiones
- ▶ Evaluación de Acciones de Entrada/Salida

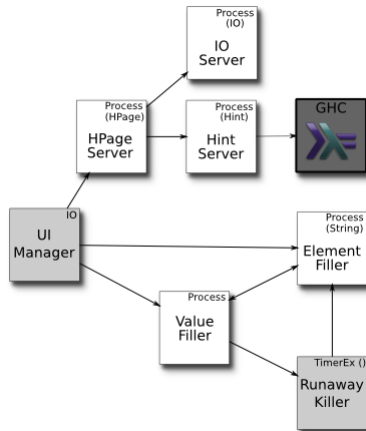
Para lograrlo, creamos *eprocess*:

- ▶ Basada conceptualmente en *Erlang*
- ▶ Construída utilizando *Threads*, *Channels* y *MVars*

# Arquitectura

## Principales Requerimientos:

- Conexión con GHC
- Paralelismo
- Errores Controlados
- Presentación de Resultados



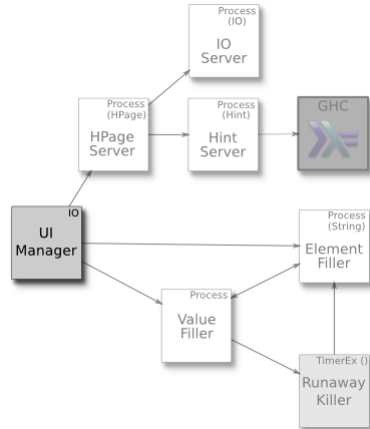
## Ejemplo de Interacción

Veremos cómo interactúan estos componentes para evaluar la siguiente expresión:

```
readFile "hpage.cabal" >>=  
  return . length . head . lines
```

## Ejemplo de Interacción

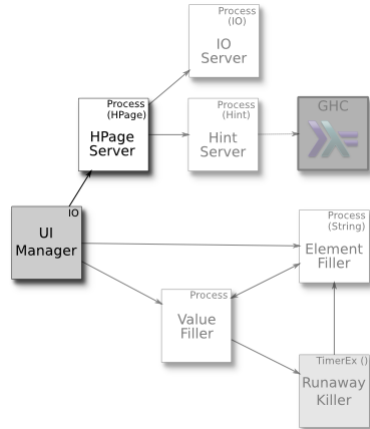
El usuario indica al UI Manager que desea interpretar la expresión





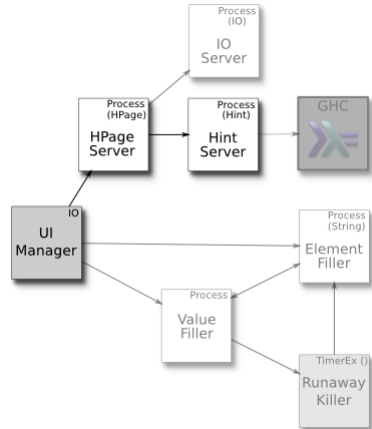
## Ejemplo de Interacción

El UI Manager solicita la  
evaluación del texto al  
HPage Server



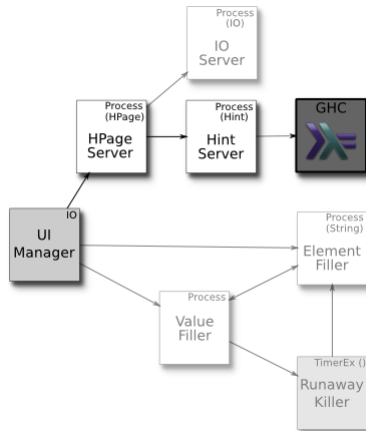
## Ejemplo de Interacción

El HPage Server envía al Hint Server la expresión para conocer su valor y su tipo



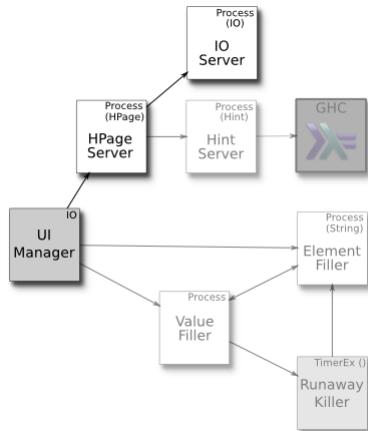
## Ejemplo de Interacción

El Hint Server se comunica con GHC utilizando *hint* y obtiene los valores solicitados, que luego informa al HPage Server



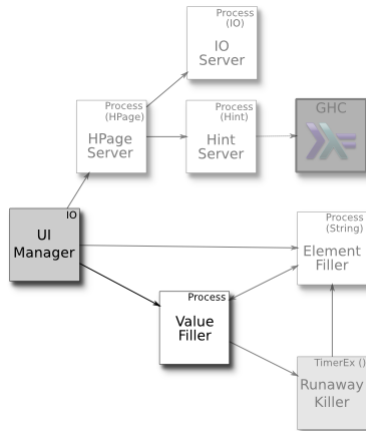
## Ejemplo de Interacción

El HPage Server, al detectar que se trata de una acción de la mónada IO, crea una MVar y envía un mensaje al IO Server para que ejecute la acción y llene la MVar con su resultado. Luego informa al UI Manager el tipo de la expresión y la MVar en la que se volcará su resultado una vez ejecutada



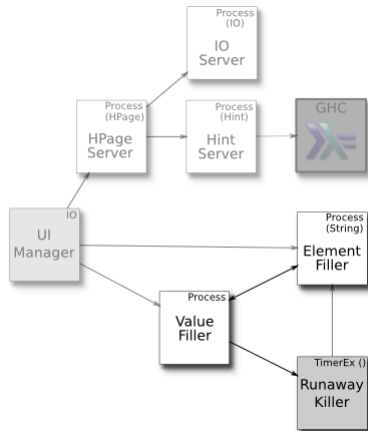
## Ejemplo de Interacción

El UI Manager muestra el tipo al usuario y vacía el cuadro de texto donde se encuentra el resultado. Luego, informándole la MVar recibida, solicita al Value Filler que espere el resultado y llene el cuadro de texto con él.



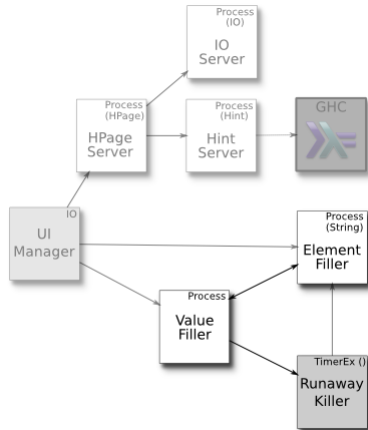
## Ejemplo de Interacción

El Value Filler se bloquea a la espera de que un valor sea depositado en la MVar recibida. Una vez conseguido ese valor, que es una cadena de caracteres ("11"), toma el primero y lo envía al Element Filler al tiempo que inicia el Runaway Killer. Luego, se bloquea esperando recibir un mensaje de alguno de ellos.



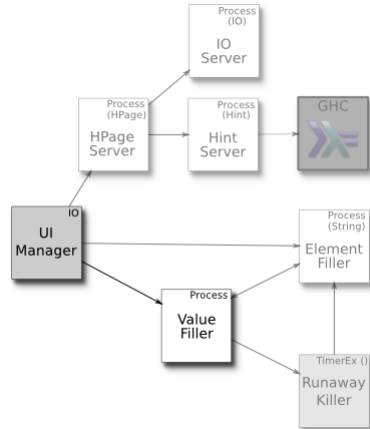
## Ejemplo de Interacción

Antes de transcurrido un segundo, el Element Filler computa el caracter 1 y envía un mensaje al Value Filler con el mismo. El Value Filler lo agrega al cuadro de texto del resultado y detiene el Runaway Killer. Luego repite el paso anterior y éste para el segundo caracter.



## Ejemplo de Interacción

Finalmente, el Value Filler informa al UI Manager que ha presentado el resultado con éxito, para que éste se lo informe al usuario.





# *Próximos Pasos*

## ¿Qué falta?

- ▶ Nuevas visualizaciones
  - ▶ Más que un cuadro de texto
- ▶ Más tipos *especiales*
  - ▶ Tuplas
  - ▶ Either
  - ▶ Maybe
- ▶ Composición
  - ▶ Listas de listas
  - ▶ Acciones que generen listas
  - ▶ Listas de acciones

## ¿Qué se puede hacer?

- ▶ Clase Presentable

## ¿Qué falta?

- ▶ Nuevas visualizaciones
  - ▶ Más que un cuadro de texto
- ▶ Más tipos *especiales*
  - ▶ Tuplas
  - ▶ Either
  - ▶ Maybe
- ▶ Composición
  - ▶ Listas de listas
  - ▶ Acciones que generen listas
  - ▶ Listas de acciones

## ¿Qué se puede hacer?

- ▶ Clase Presentable

## ¿Qué falta?

- ▶ Nuevas visualizaciones
  - ▶ Más que un cuadro de texto
- ▶ Más tipos *especiales*
  - ▶ Tuplas
  - ▶ Either
  - ▶ Maybe
- ▶ Composición
  - ▶ Listas de listas
  - ▶ Acciones que generen listas
  - ▶ Listas de acciones

## ¿Qué se puede hacer?

- ▶ Clase *Presentable*

¿Qué falta?

- ▶ Nuevas visualizaciones
  - ▶ Más que un cuadro de texto
- ▶ Más tipos *especiales*
  - ▶ Tuplas
  - ▶ Either
  - ▶ Maybe
- ▶ Composición
  - ▶ Listas de listas
  - ▶ Acciones que generen listas
  - ▶ Listas de acciones

¿Qué se puede hacer?

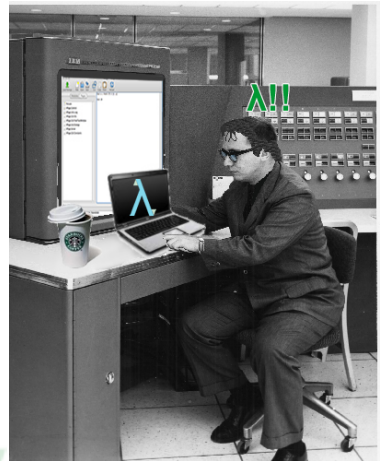
- ▶ Clase Presentable

Con  $\lambda$ Page hemos acercado al desarrollador *Haskell* sólo **una** de muchas herramientas:

- ▶ Soporte para TDD
- ▶ Refactoring
- ▶ Análisis de Terminación
- ▶ Debugging

Con  $\lambda$ Page hemos acercado al desarrollador *Haskell* sólo **una** de muchas herramientas:

- Soporte para TDD
- Refactoring
- Análisis de Terminación
- Debugging



# ¡Gracias a todos!

- ▶ *Sitio Web de  $\lambda$ Page:*
  - ▶ <http://hpage.haskell.com>
- ▶  *$\lambda$ Page en Github*
  - ▶ <http://github.com/elbrujohalcon/hPage>
- ▶ *Fernando Benavides en la Internet*
  - ▶ <http://profiles.google.com/greenmellon>