λ*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*

Trabajando con Lenguajes Orientados a Objetos

Workspace

Lazy Evaluation

Efectos Colaterales

Manejo de Módulos

Utilizando λ*Page*, el usuario puede

- Importar módulos
- Cargar módulos
- Recargar módulos

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

λ*Page por Dentro*

λ*Page* está desarrollado en *Haskell*

Se conecta con *GHC* a través de su API

La interfaz gráfica está desarrollada usando *wxHaskell*

λ*Page* está desarrollado en *Haskell*

Se conecta con *GHC* a través de su API

La interfaz gráfica está desarrollada usando *wxHaskell*

λ*Page* está desarrollado en *Haskell*

Se conecta con *GHC* a través de su API

La interfaz gráfica está desarrollada usando *wxHaskell*

Paralelismo

Muchas cosas suceden simultáneamente en *λPage*

- Manejo de Páginas (crear, abrir, modificar, guardar, cerrar, 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 *λPage*

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

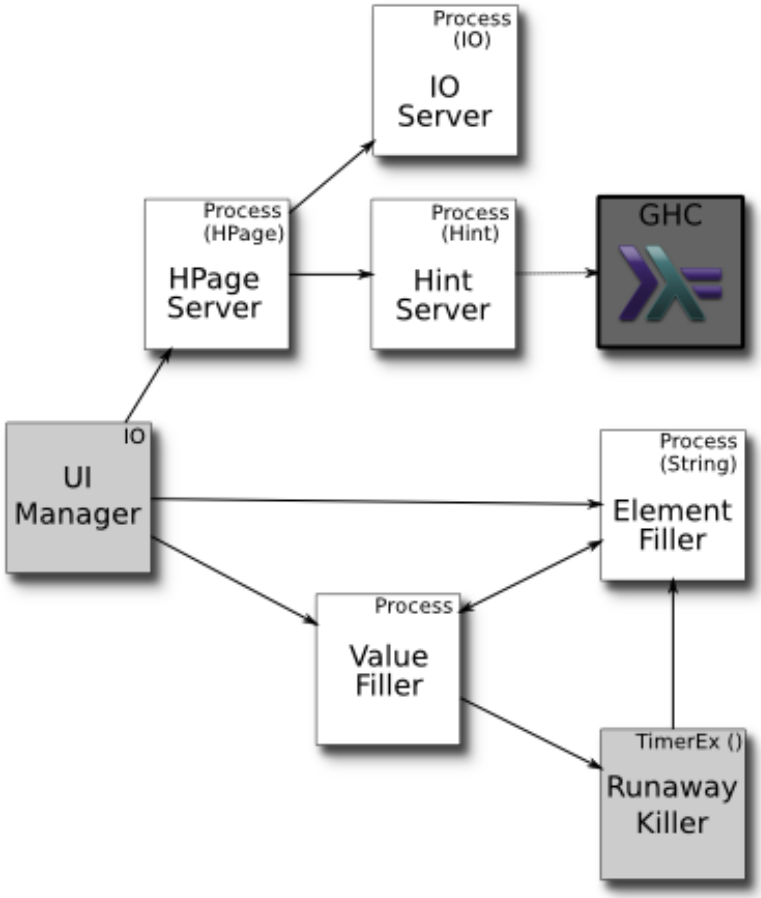
Para lograrlo, creamos *eprocess*:

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

Arquitectura

Principales Requerimientos:

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



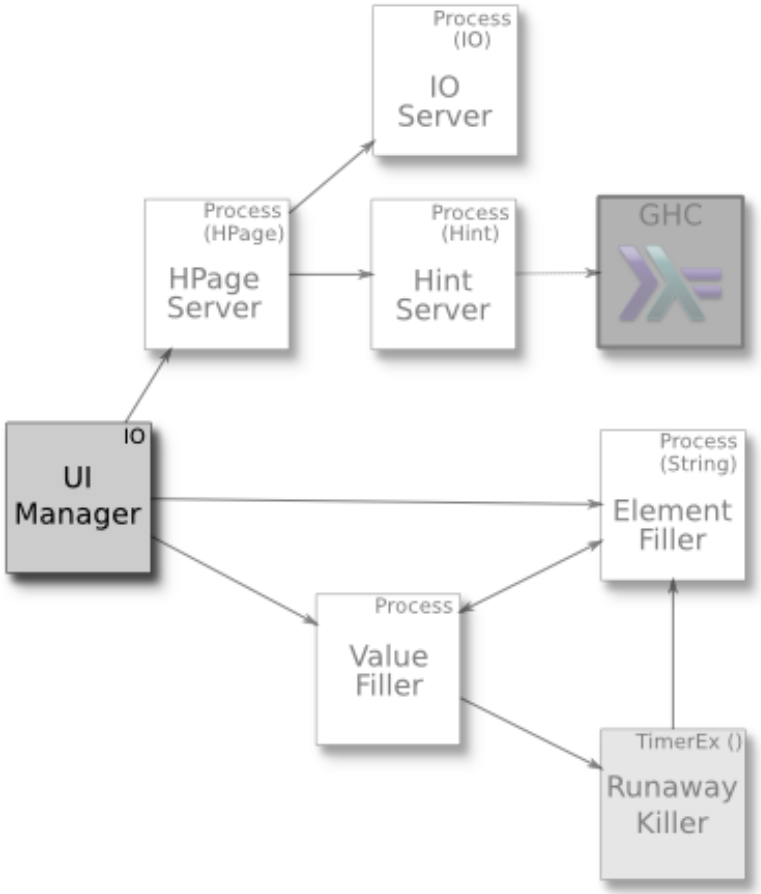
Ejemplo de Interacción

Veremos cómo interactúan estos componentes para evaluar la siguiente expresión:

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

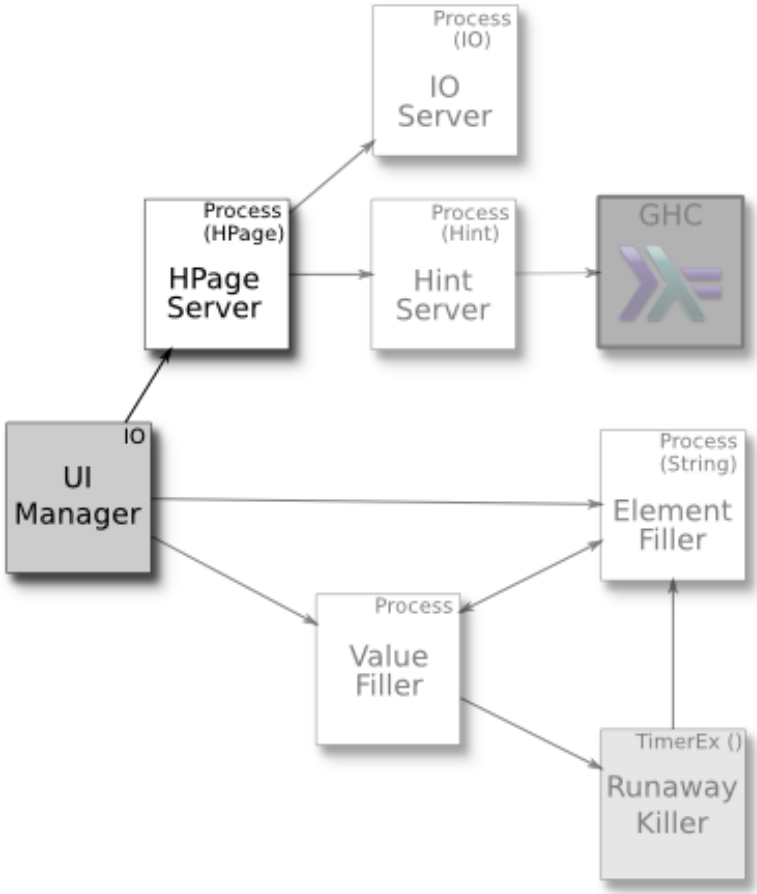
Ejemplo de Interacción

El usuario indica al `UI Manager` que desea interpretar la expresión



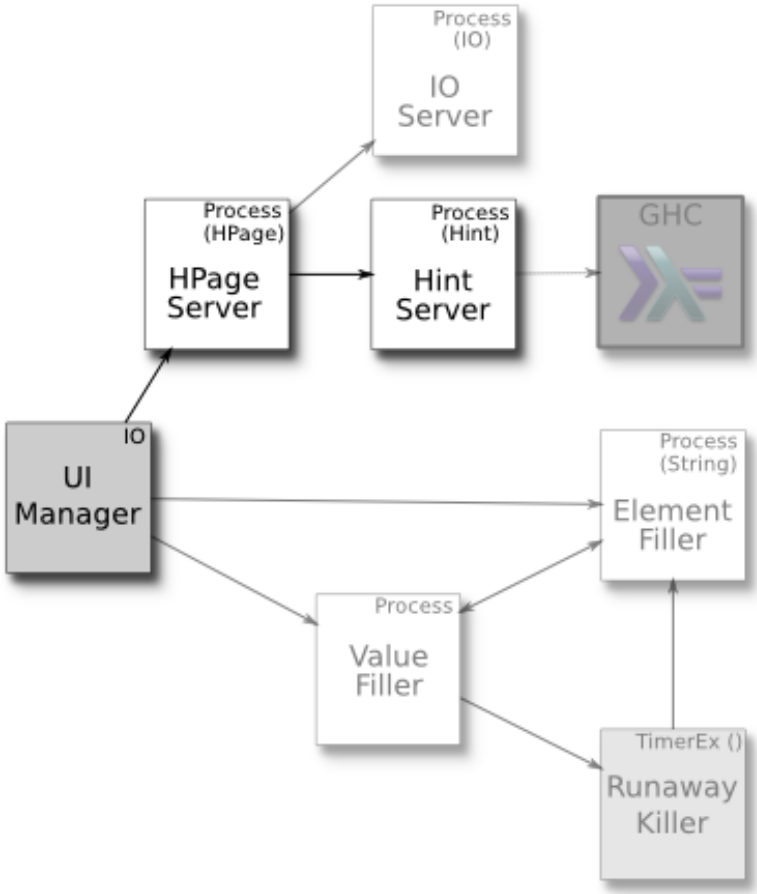
Ejemplo de Interacción

El `UI Manager` solicita la evaluación del texto al `HPage Server`



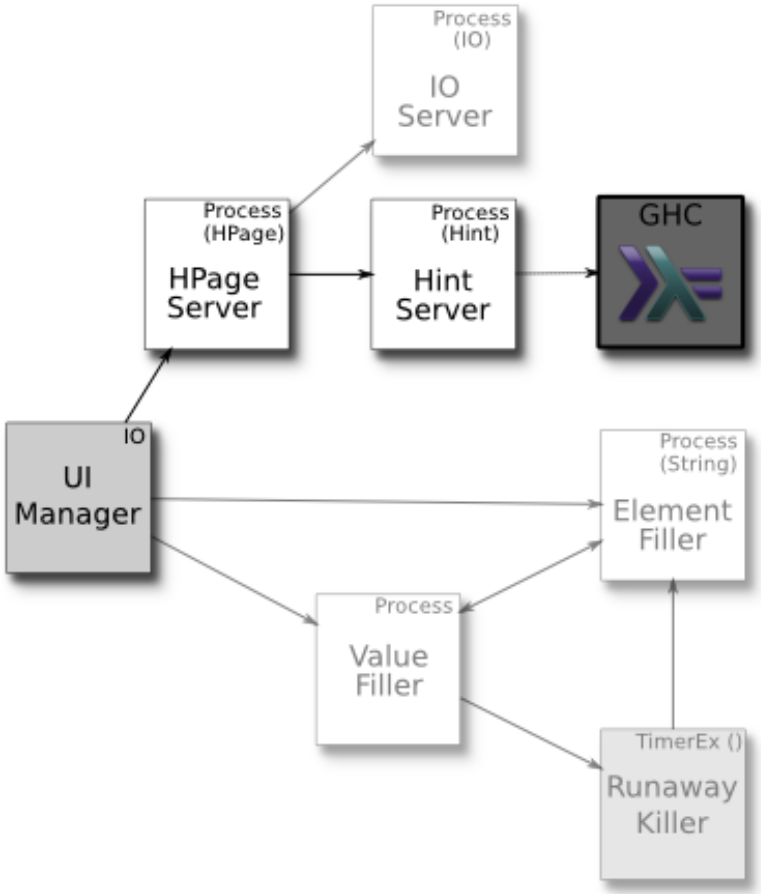
Ejemplo de Interacción

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



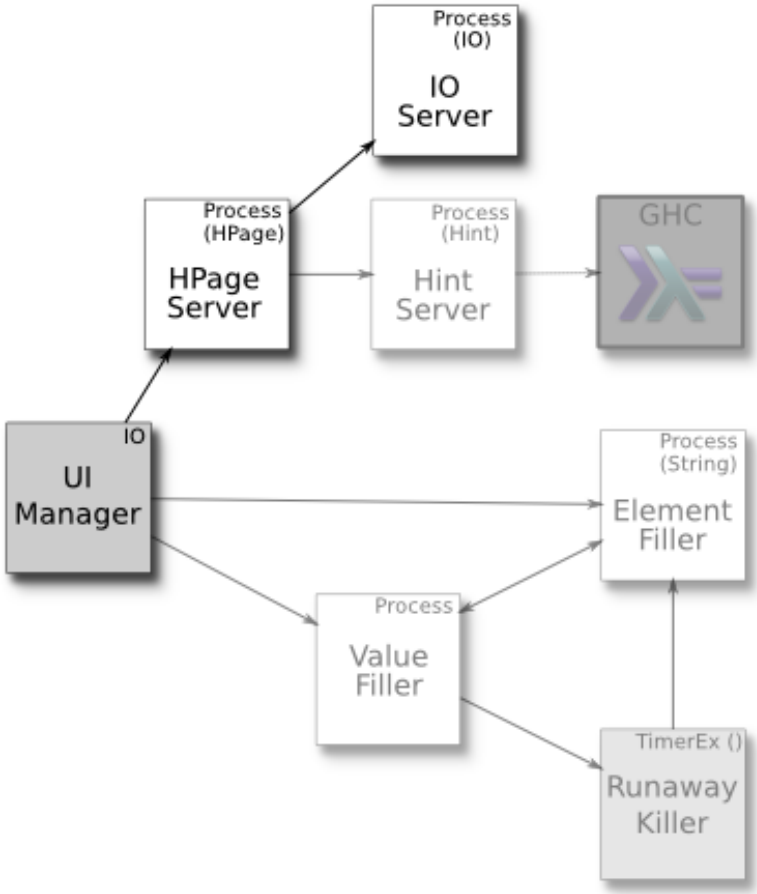
Ejemplo de Interacción

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



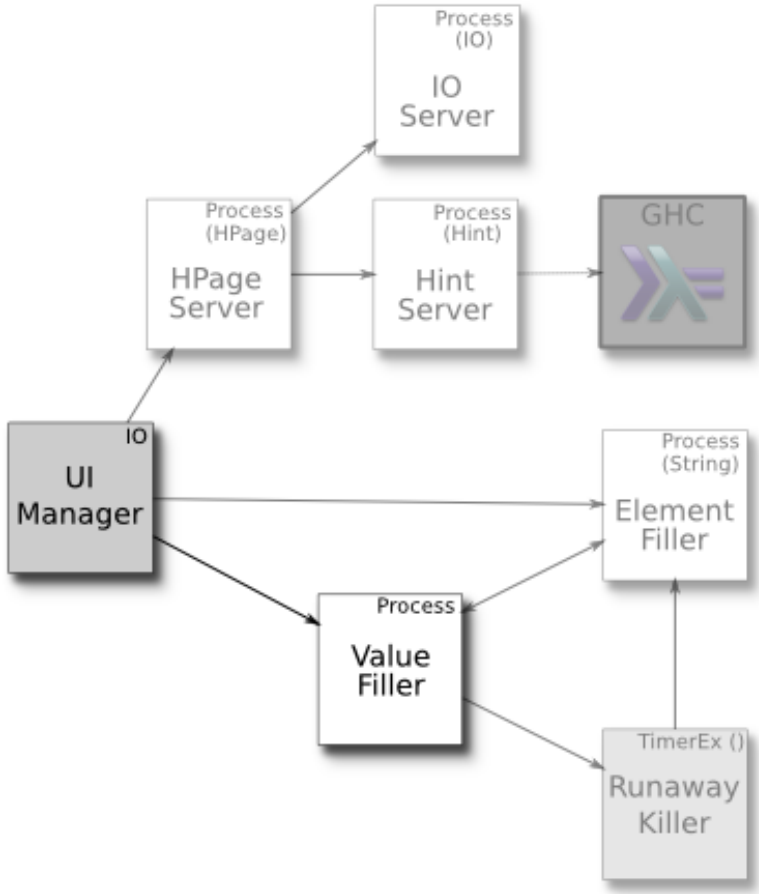
Ejemplo de Interacción

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



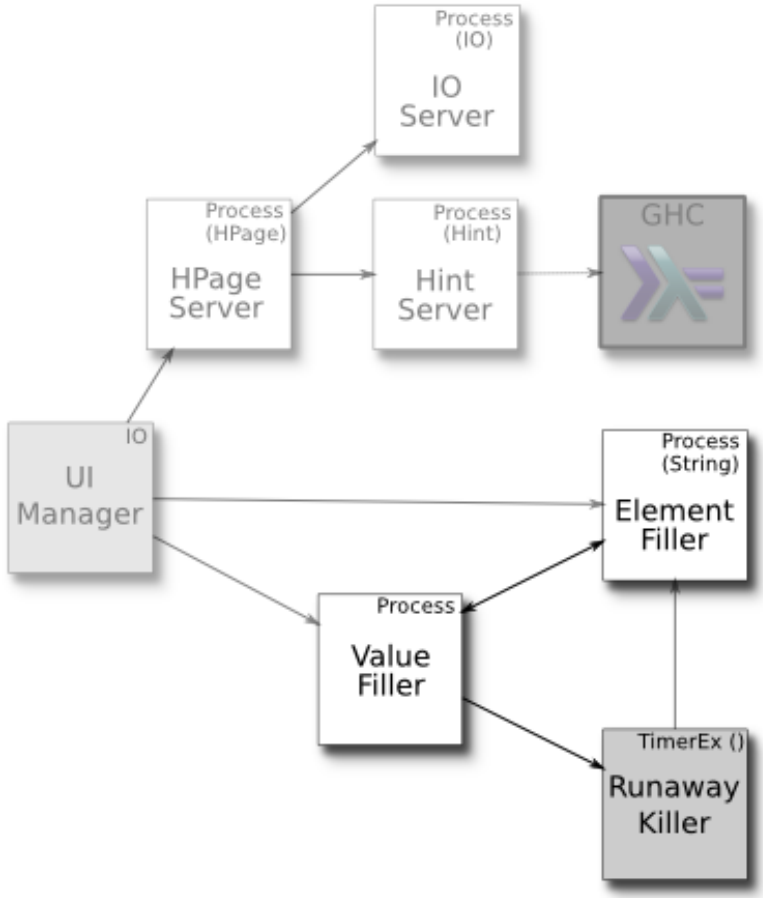
Ejemplo de Interacción

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.



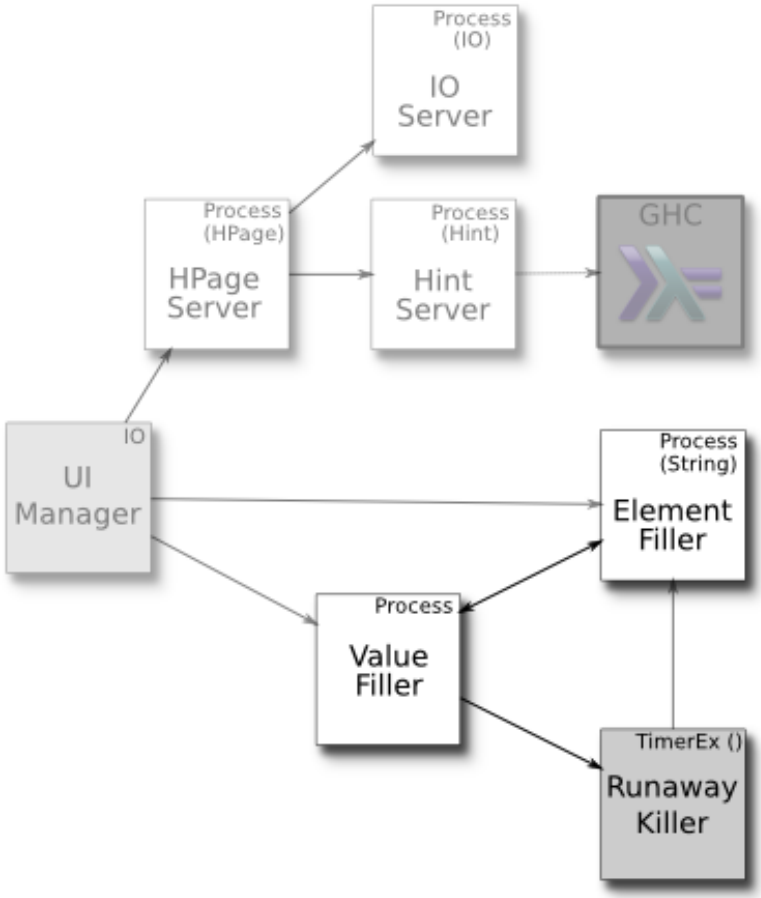
Ejemplo de Interacción

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.



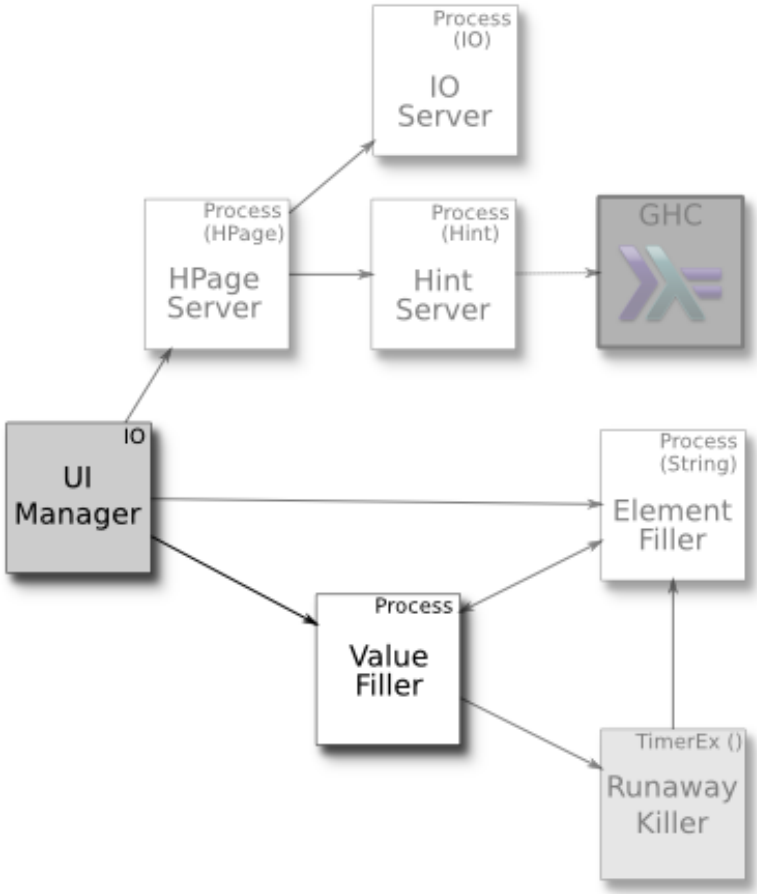
Ejemplo de Interacción

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.



Limitaciones

¿Qué falta?

- Nuevas visualizaciones
- Distintos tipos a tratar de modo particular
- Composición

¿Qué se puede hacer?

- Clase `Presentable`

Limitaciones

¿Qué falta?

- Nuevas visualizaciones
- Distintos tipos a tratar de modo particular
- Composición

¿Qué se puede hacer?

- Clase `Presentable`

Otras Tareas a Realizar

- Otras herramientas
  - Soporte para TDD
  - Refactoring
  - Análisis de Terminación
  - Debugging
- Otros lenguajes
  - Erlang

Otras Tareas a Realizar

- Otras herramientas
  - Soporte para TDD
  - Refactoring
  - Análisis de Terminación
  - Debugging
- Otros lenguajes
  - Erlang

¡Gracias a todos!

- *Sitio Web de λPage*:
  - http://hpage.haskell.com
- *λPage en Github*
  - http://github.com/elbrujohalcon/hPage
- *Fernando Benavides en la Internet*
  - http://profiles.google.com/greenmellon