DESARROLLO DE NAVEGADOR WEB CON HASKELL[1]

Carlos Gomez

Universidad Mayor de San Simón
Facultad de Ciencias y Tecnología
Carrera de Licenciatura en Informática

11 de Julio, 2011

[1] Defensa de Proyecto de Grado, Presentado para Optar al Diploma Académico de Licenciatura en Informática

Contenido

Introducción

Navegadores Web

- Programa informático del lado del cliente, que se encarga de renderizar documentos.
- Trabaja con varias tecnologías y estándares Web:
  - HTML/XHTML/XML
  - CSS
  - DOM
  - JavaScript
  - ..., etc.

Navegadores Web Actuales

Fueron desarrollados con lenguajes de programación imperativos.

Motor de Rend.	**Leng. Prog.**	**Navegadores Web**
Gecko	C++	Firefox, Camino, Flock, Sea-Monkey, K-Melen, Netscape 9, Lenascape, Epiphany
Presto	C++	Opera
WebKit	C++	Safari, Google Chrome

Fuente: Wikipedia

Lenguajes Imperativos vs Funcionales

Los programas gigantes (Navegadores Web) exigen que el lenguaje de programación permita un alto nivel de modularidad, código compacto, fácil de comprender , etc, con el objetivo de reducir el costo de desarrollo del programa.

	Prog. Imperativa	**Prog. Funcional**
Navegadores Web Actuales		WWWBrowser 

Los lenguajes funcionales, en muchos casos, permiten aprovechar estas caracteristicas al maximo y reducir el costo de desarrollo. Pero no existe un Navegador Web actual implementado en un lenguaje de programación funcional.

Haskell

Haskell

Haskell es un lenguaje de programación funcional, con evaluación perezosa y de propósito general que incorpora muchas de las innovaciones recientes del diseño de lenguajes de programación.

– *(Peyton Jones, Haskell 98 Report, 2002)*

Motivación del Proyecto

Motivación del Proyecto

Experimentar las capacidades/características, librerías y herramientas de Haskell en el Desarrollo de un Navegador Web.

Objetivos del Proyecto

Objetivo General

Desarrollar un Navegador Web con el lenguaje de programación funcional Haskell.

Objetivos Específicos

- Desarrollar el módulo de comunicación entre el Navegador Web y los protocolos HTTP y modelo TCP/IP.
- Desarrollar un intérprete de la información HTML.
- Desarrollar algoritmos que permitan mostrar la información en la pantalla del Navegador Web.
- Desarrollar la Interfaz Gráfica de Usuario (GUI).
- Desarrollar un módulo que dé soporte a CSS (Cascading Style Sheet).

Desarrollo del Proyecto

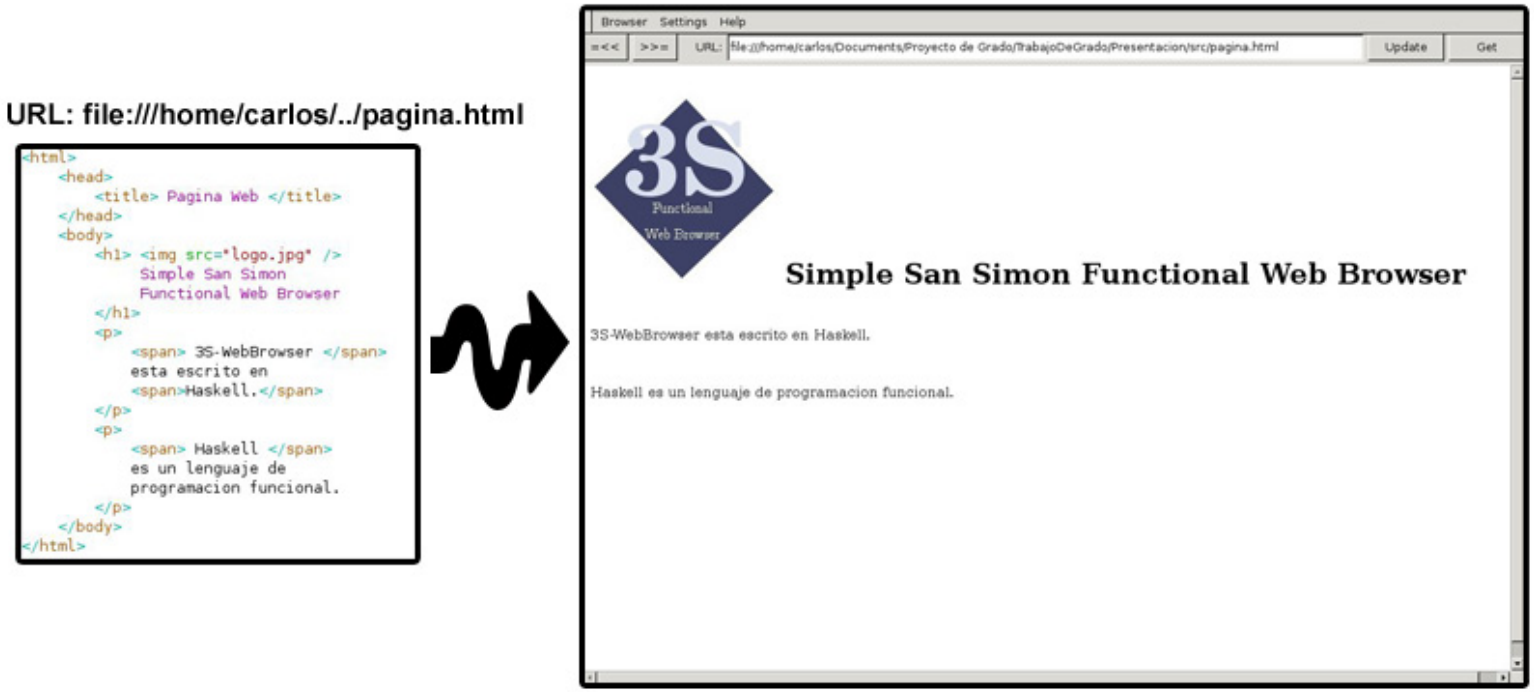
Principales Herramientas y Librerías

Principales Herramientas y Librerías que se utilizaron en el proyecto:

- Librería uu-parsinglib
- Herramienta UUAGC
- Librería wxHaskell
- Librería libcurl
- Librería Map

Renderización

La renderización consiste en convertir un documento de texto (descrito por un lenguaje de marcado) a un formato visual en la pantalla.



Renderización

La renderización es realizada aplicando un conjunto de transformaciones a la entrada hasta obtener un formato visual para la pantalla.

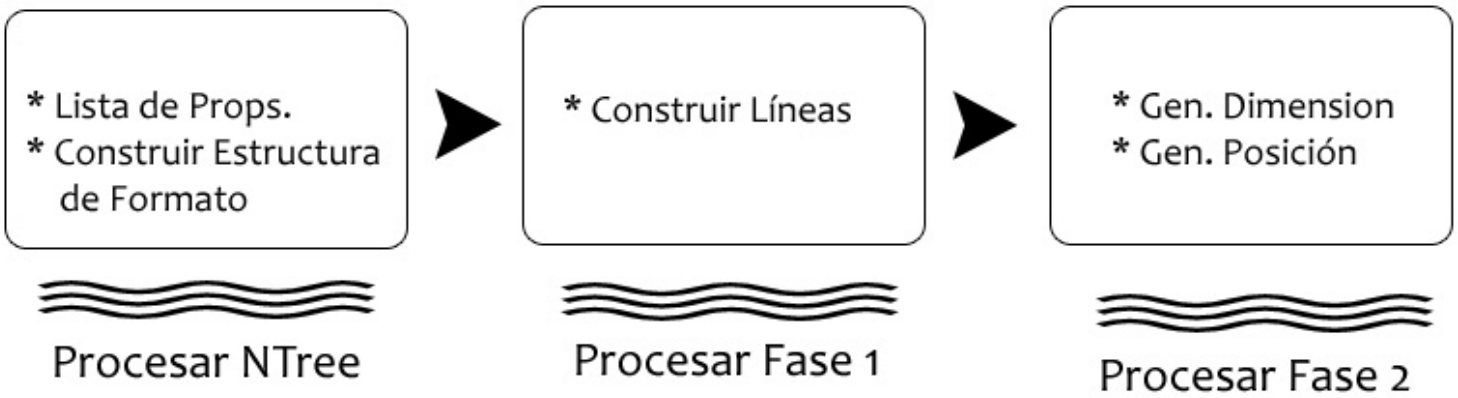


Diagrama de Renderización

Estado inicial + GUI con wxHaskell

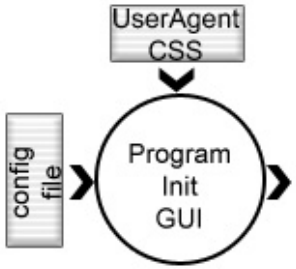


Diagrama de Renderización

Obtener el doc. y recursos con libcurl

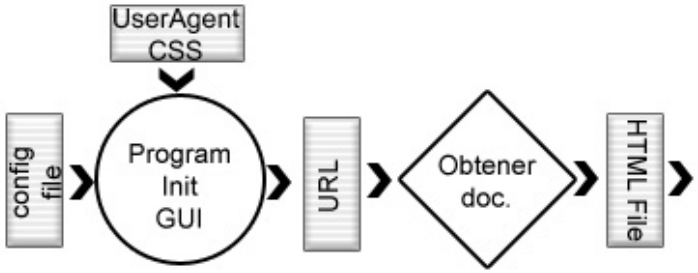


Diagrama de Renderización

Reconocer la entrada con uu-parsinglib

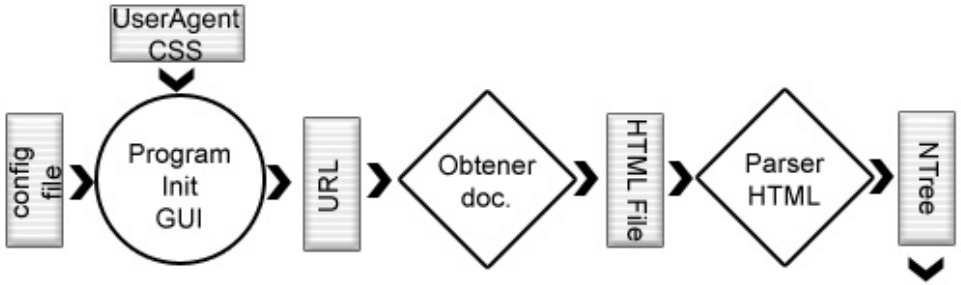


Diagrama de Renderización

Procesar el NTree con UUAGC

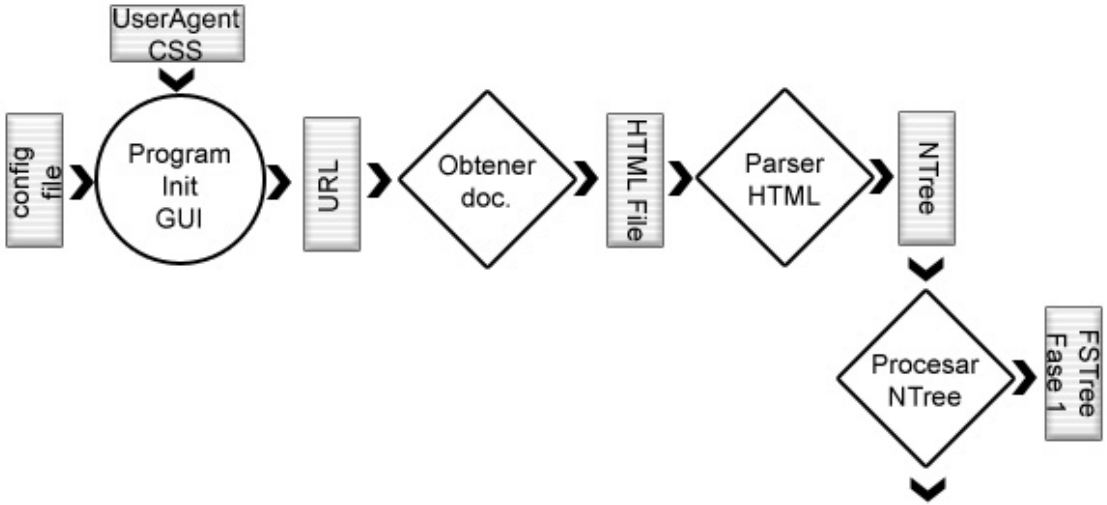


Diagrama de Renderización

Procesar el FSTreeFase1 con UUAGC

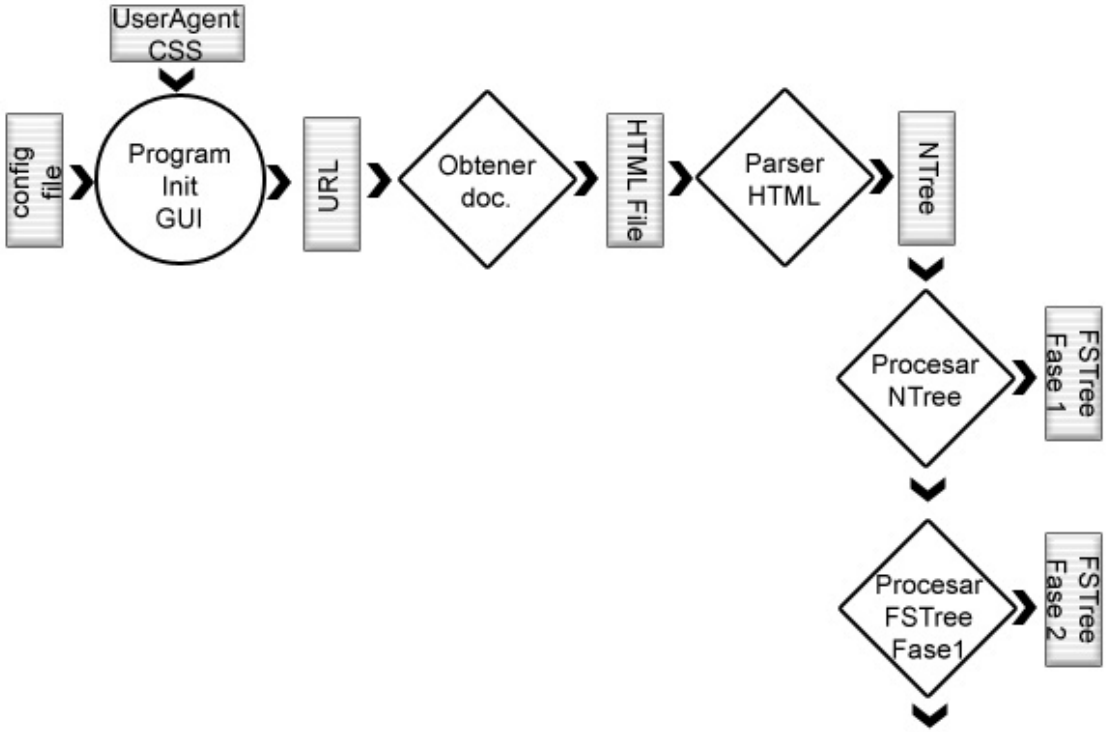


Diagrama de Renderización

Procesar el FSTreeFase2 con UUAGC

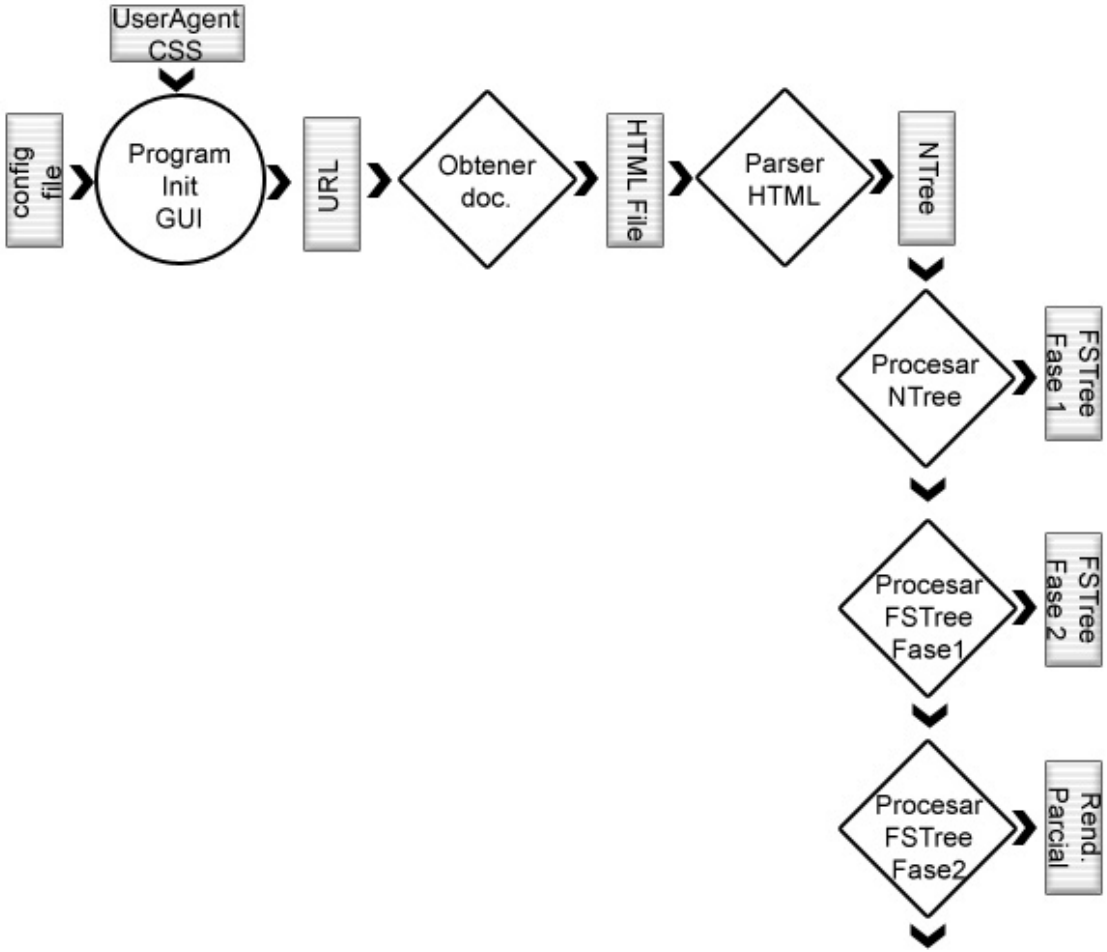


Diagrama de Renderización

Renderizar el documento y Props con wxHaskell

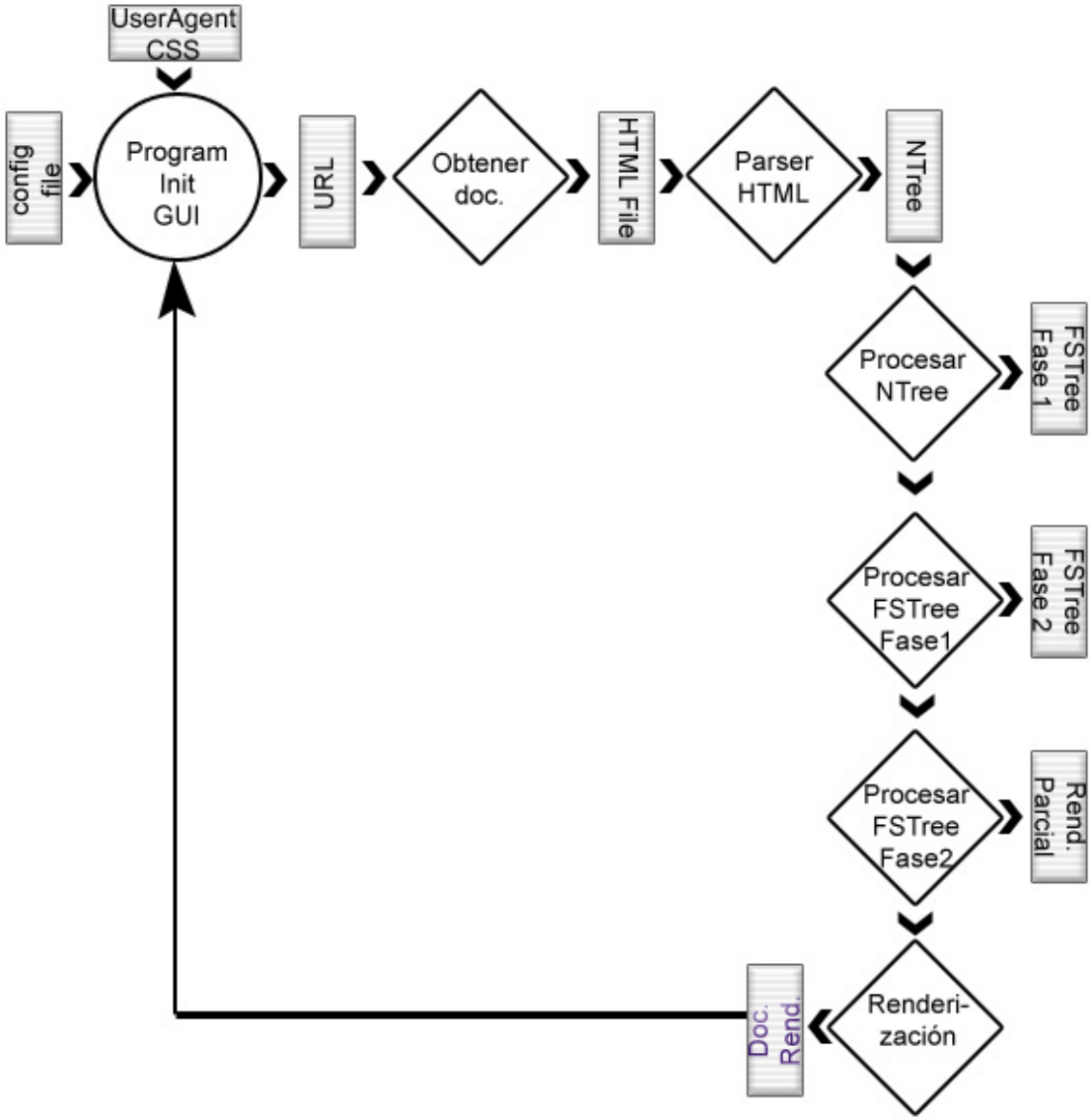
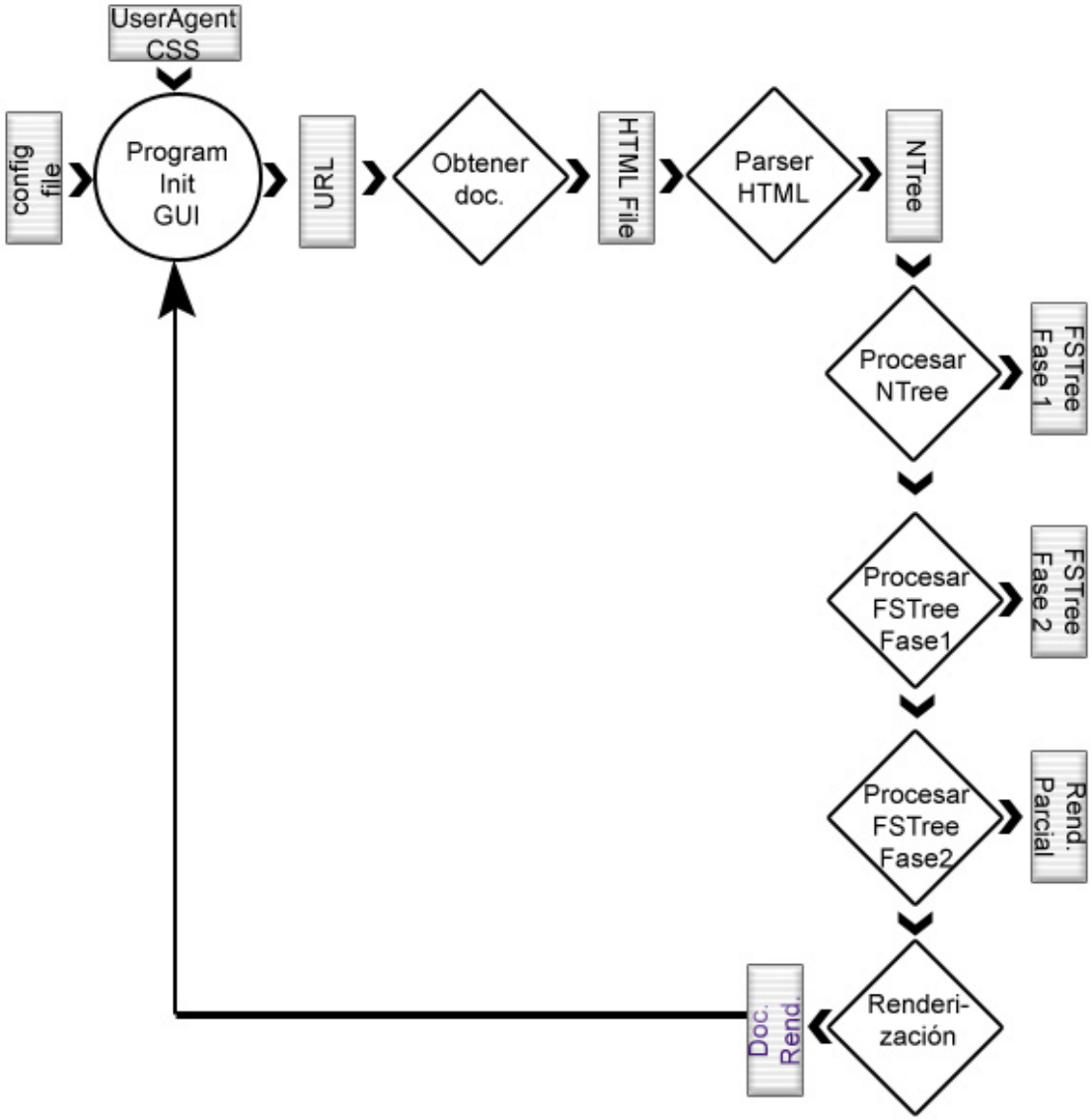
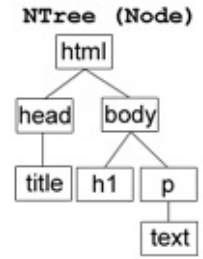


Diagrama de Renderización

Transformaciones





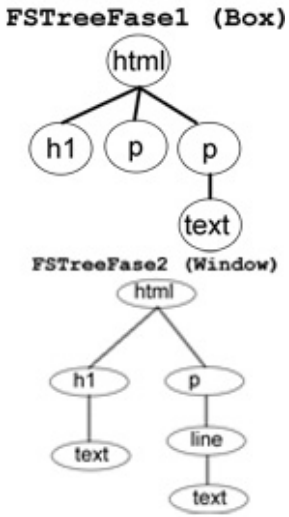


Diagrama de Renderización

Procesar NTree en detalle

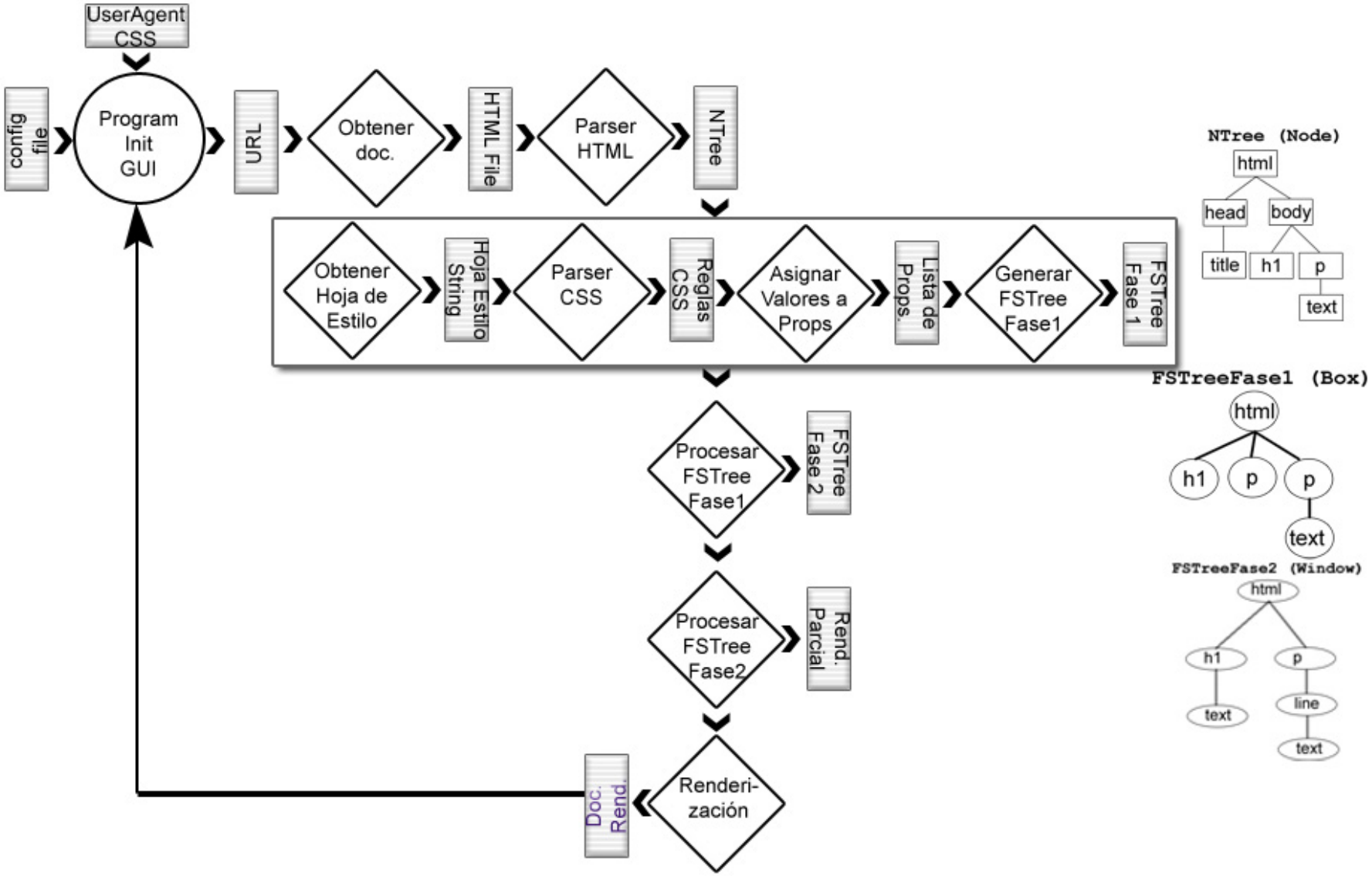


Diagrama de Renderización

Procesar FSTreeFase1 en detalle

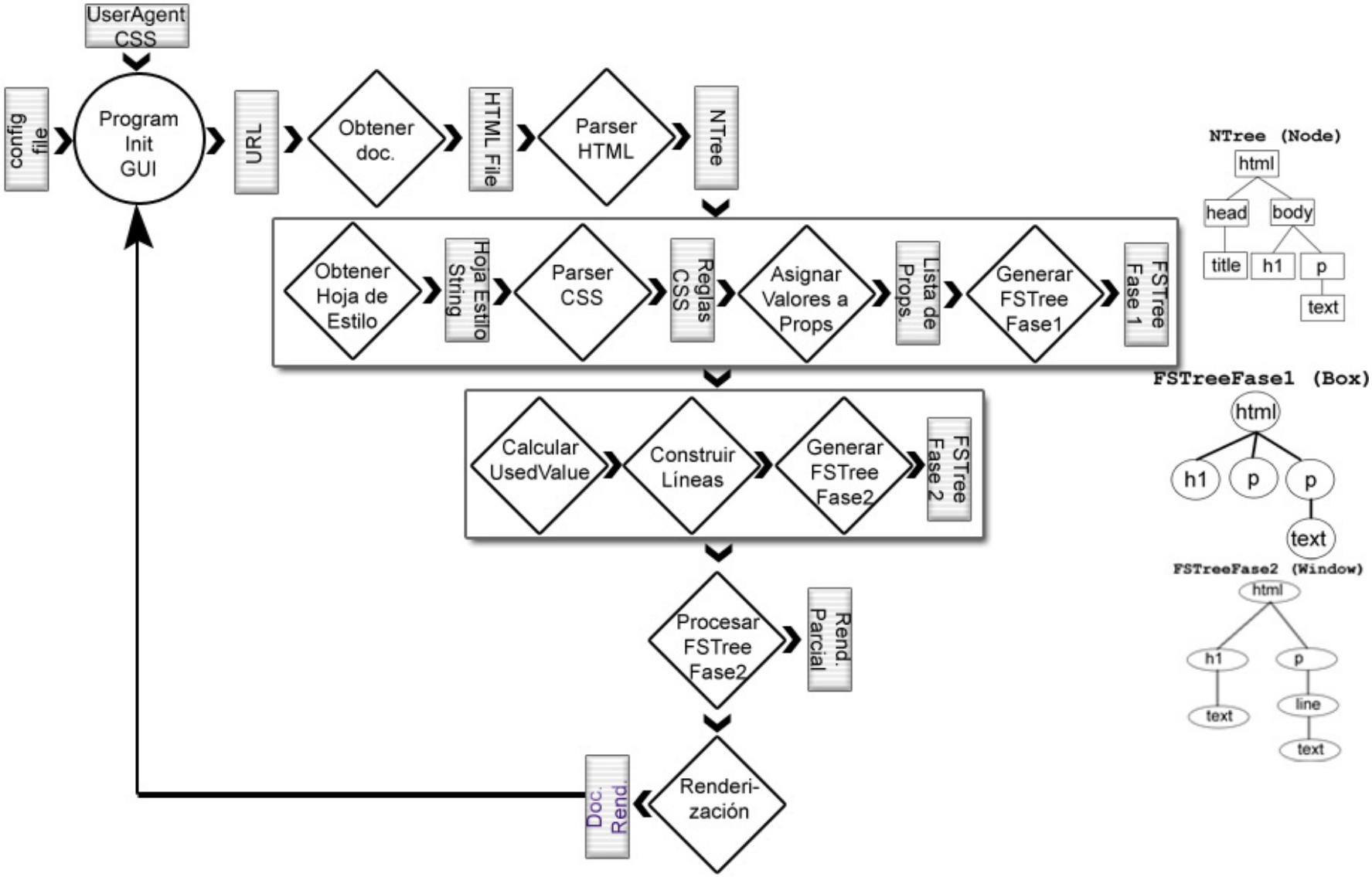


Diagrama de Renderización

Procesar FSTreeFase2 en detalle

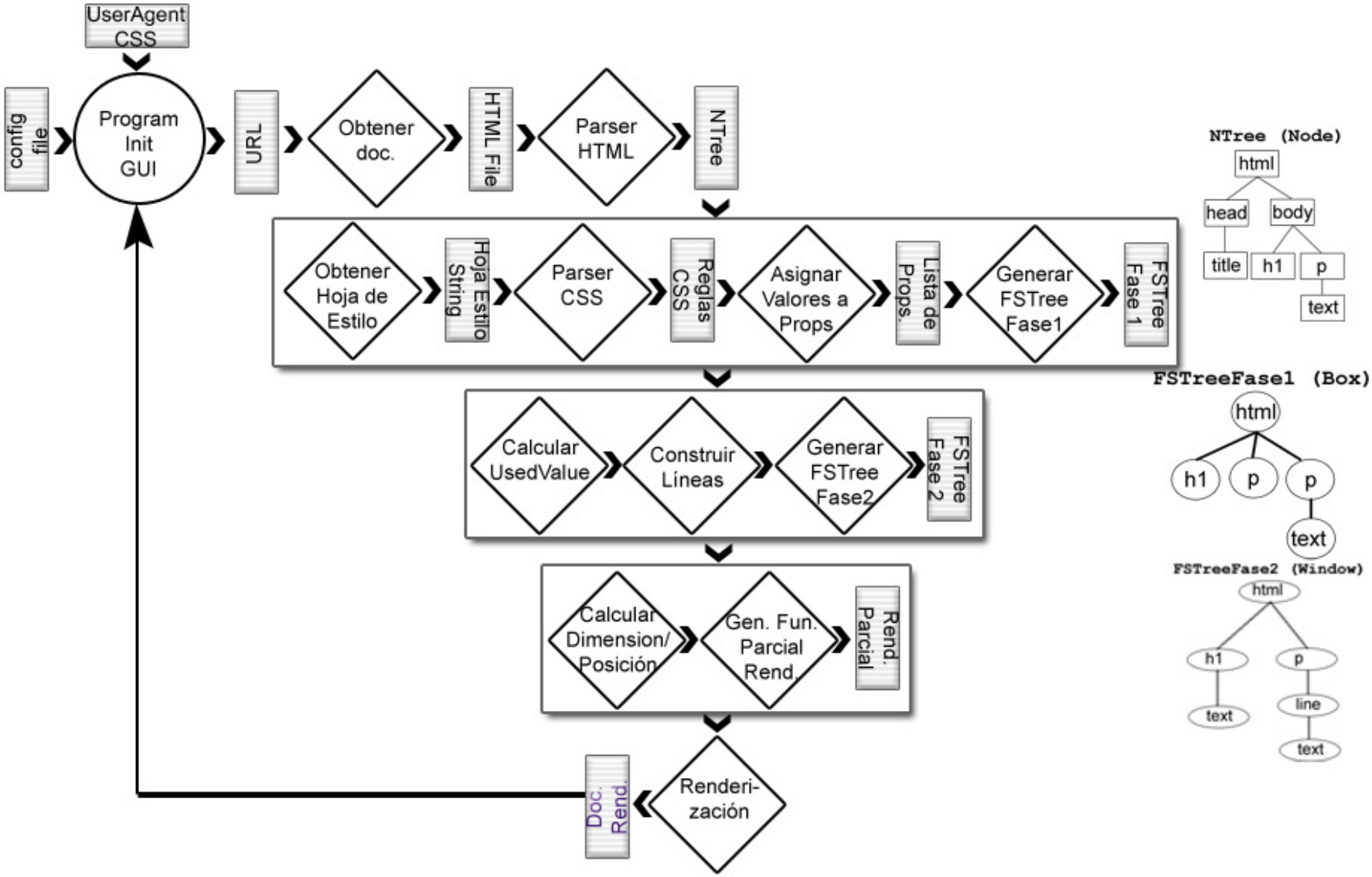
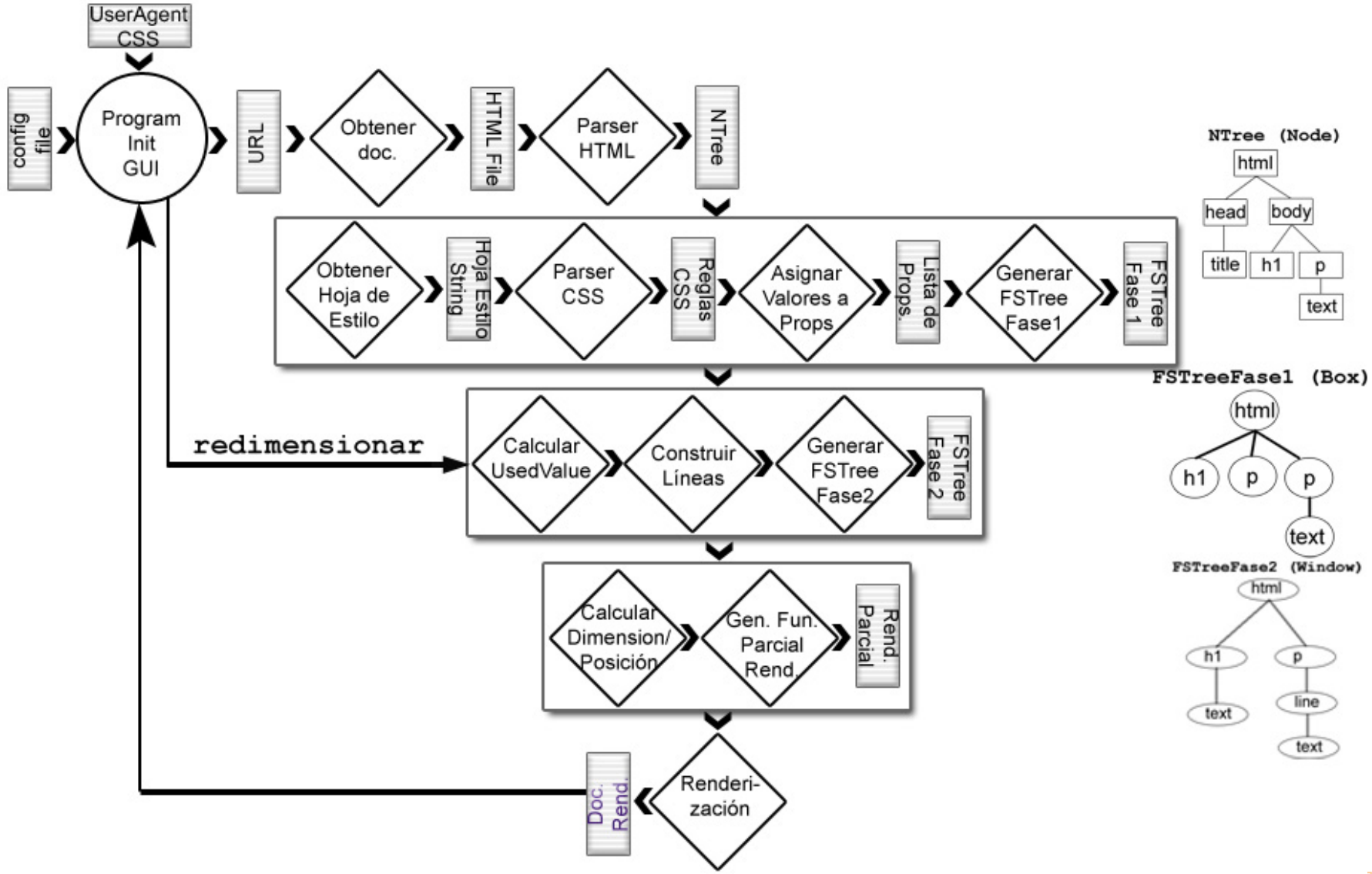


Diagrama de Renderización

Redimensionamiento de paginas



NTree

Estructura Rosadelfa (NTree)


```
DATA NTree
  | NTree Node ntrees: NTrees

TYPE NTrees = [NTree]

DATA Node
  | NTag   name     : String
           replaced : Bool
           atribs   : {Map.Map String String}
  | NText  text     : String
```

FSTreeFase1

Estructura de formato de fase 1


```
DATA BoxTree
  | BoxContainer name : String
                 fcnxt: {FormattingContext}
                 props: {Map.Map String Property}
                 attrs: {Map.Map String String}
                 bRepl: Bool
                 boxes: Boxes
  | BoxText name : String
            props: {Map.Map String Property}
            attrs: {Map.Map String String}
            text : String
  | BoxItemContainer props: {Map.Map String Property}
                     attrs: {Map.Map String String}
                     boxes: Boxes

TYPE Boxes = [BoxTree]

DATA FormattingContext
  | InlineContext | BlockContext | NoContext
```

FSTreeFase2

Estructura de formato de fase 2


```
DATA WindowTree
   | WindowContainer name : String
                     fcnxt: {FormattingContext}
                     props: {Map.Map String Property}
                     attrs: {Map.Map String String}
                     tCont: {TypeContinuation}
                     bRepl: Bool
                     elem : Element
   | WindowText name : String
                props: {Map.Map String Property}
                attrs: {Map.Map String String}
                tCont: {TypeContinuation}
                text : String
   | WindowItemContainer marker     : {ListMarker}
                         sizeMarker: {(Int,Int)}
                         elem      : Element
                         props     : {Map.Map String Property}
                         attrs     : {Map.Map String String}

DATA TypeContinuation
   | Full   | Init   | Medium | End
```

FSTreeFase2

Estructura de formato de fase 2 (Continuación)


```
DATA Element
   | EWinds winds: WindowTrees
   | ELines lines: Lines
   | ENothing

TYPE WindowTrees = [WindowTree]

TYPE Lines       = [Line]

DATA Line
   | Line winds: WindowTrees

DATA ListMarker
   | Glyph       name : String
   | Numering    value: String
   | Alphabetic  value: String
   | NoMarker
```

CSS

Cascading Style Sheet (CSS)

La especificación de CSS provee:

1. Modelos para renderización
2. Lenguaje para Hojas de Estilos
3. Propiedades de CSS

Modelo en Cascada

- 3 Usuarios/Origen de donde provienen las hojas de estilo:

```
DATA Origen
  | UserAgent | User | Author
```

- 3 Tipos de Hojas de estilo
  - **Externa**, elemento *link* de HTML.
  - **Interna**, elemento *style* de HTML.
  - **Atributos**, atributo *style* de HTML.

```
DATA Tipo
  | HojaExterna | HojaInterna | EstiloAtributo
```

- El usuario *Author* es el único que puede definir estilos en los 3 tipos. Los demás sólo pueden definir estilos del tipo externo.

Lenguaje para Hojas de Estilos

Su representación es:

```
TYPE HojaEstilo = [Regla]

TYPE Regla = (Tipo, Origen, Selector, Declaraciones)

DATA Selector
  | SimpSelector SSelector
  | CompSelector SSelector
                 operador: String
                 Selector

DATA SSelector
  | TypeSelector nombre: String
                 Atributos
                 MaybePseudo
  | UnivSelector Atributos MaybePseudo
```

Lenguaje para Hojas de Estilos


```
TYPE Atributos = [Atributo]

DATA Atributo
   | AtribID      id: String
   | AtribNombre nombre: String
   | AtribTipoOp nombre, op, valor : String

TYPE MaybePseudo = MAYBE PseudoElemento

DATA PseudoElemento
   | PseudoBefore
   | PseudoAfter

TYPE Declaraciones = [Declaracion]

DATA Declaracion
   | Declaracion nombre: String Value importancia: Bool
```

Propiedades de CSS

Existe como 80 propiedades para la pantalla, que se encuentran en la especificación de CSS. Haskell permite representarlos casi de forma plana y directa.

```
data Property
  = Property { name             :: String
             , inherited        :: Bool
             , initial          :: Value
             , value            :: Parser Value
             , propertyValue    :: PropertyValue
             , fnComputedValue  :: FunctionComputed
             , fnUsedValue      :: FunctionUsed }

data PropertyValue
  = PropertyValue { specifiedValue :: Value
                  , computedValue  :: Value
                  , usedValue      :: Value
                  , actualValue    :: Value }
```

Propiedades de CSS (Continuación)

El tipo de dato Value:

```
DATA Value
   | PixelNumber    Float
   | PointNumber    Float
   | EmNumber       Float
   | Percentage     Float
   | KeyValue       String
   | KeyColor       rgb: {(Int, Int, Int)}
   | StringValue    String
   | ListValue      values: {[Value]}
   ...
   | NotSpecified
```

Propiedades de CSS (Continuación)


```
type FunctionComputed
    =  Bool                              -- soy el root?
    -> Map.Map String Property           -- father props
    -> Map.Map String Property           -- local  props
    -> Maybe Bool                        -- soy replaced ?
    -> Bool                              -- revisar pseudo?
    -> String                            -- Nombre
    -> PropertyValue                     -- PropertyValue
    -> Value

type FunctionUsed
    =  Bool                              -- soy el root?
    -> (Float, Float)                    -- dim root
    -> Map.Map String Property           -- father props
    -> Map.Map String Property           -- local props
    -> Map.Map String String             -- atributos
    -> Bool                              -- replaced?
    -> String                            -- Nombre
    -> PropertyValue                     -- PropertyValue
    -> Value
```

Ejemplo

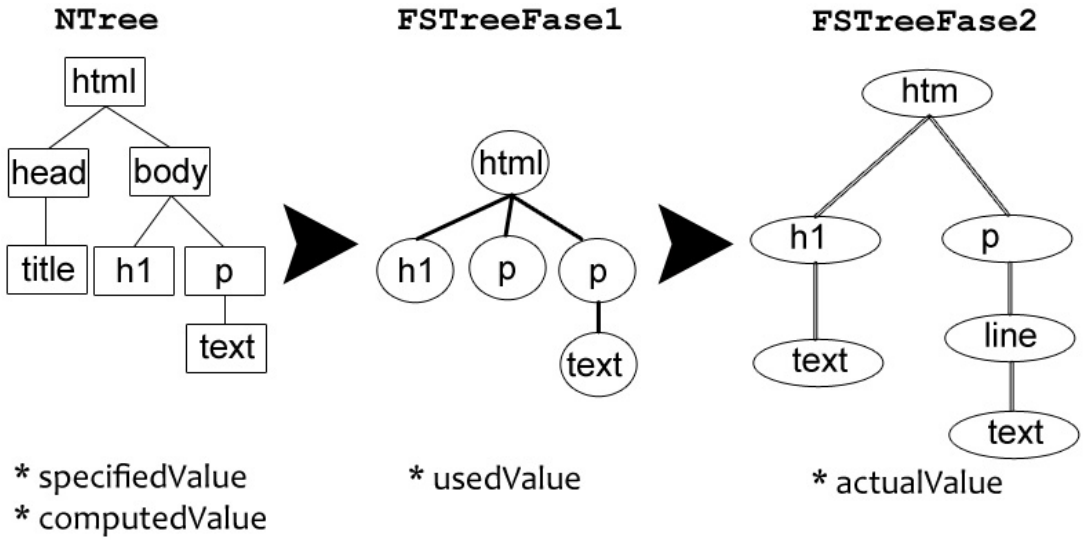
Definición de la propiedad 'border-color'


```
mkProp :: ( String, Bool, Value, Parser Value
  , FunctionComputed, FunctionUsed) -> Property

bc = mkProp ( "border-top-color"
            , False
            , NotSpecified
            , pBorderColor
            , computed_border_color
            , used_asComputed )
pBorderColor
    = pColor <|> pKeyValues ["inherit"]
computed_border_color
  iamtheroot fatherProps locProps iamreplaced iamPseudo nm prop
    = case specifiedValue prop of
        NotSpecified
            -> specifiedValue (locProps `get` "color")
        KeyColor (r,g,b)
            -> KeyColor (r,g,b)
```

Renderización y Propiedades CSS

Las propiedades de CSS guían la renderización de un documento.



- *specifiedValue*: es calculado utilizando el algoritmo cascada de CSS.
- *computedValue*, *usedValue*: es calculado utilizando el comportamiento definido en cada propiedad.

Resultados del Proyecto

Resultados del Proyecto

Como resultado, se ha desarrollado un Navegador Web con Haskell:

Simple San Simon Funcional Web Browser

Características:

- Trabaja con los protocolos HTTP y File
- Soporte para un subconjunto de la gramática de XHTML y CSS
- Soporte para 48 propiedades de CSS, que incluye:
  - Modificar la dimensión, posición, tipo y estilo de un elemento
  - Modificar el estilo de un texto
  - Listas
  - Generación de contenidos

Conclusiones

Conclusiones

Se encontró que el lenguaje de programación funcional Haskell fue apropiado y maduro para el desarrollo de un Navegador Web.

- Apropiado , porque varias partes de la implementación fueron expresadas de mejor manera utilizando mecanismos de la programación funcional.
- Maduro , porque en muchos casos, simplemente se utilizó las librerías ya existentes para Haskell.

Sin embargo, también se ha tenido varias dificultades en la implementación de algunos algoritmos, y algunas limitaciones de algunas librerías.

Beneficios del lenguaje

Al utilizar Haskell como lenguaje de desarrollo, el proyecto se ha beneficiado con varias de sus características.

- Forma rica de definir los tipos de datos.
- Amplia Biblioteca de funciones.
- Varias formas de definir una función.
- Aplicación parcial de funciones.
- Definición de módulos.

Beneficios de las Herramientas y Librerías

Las Herramientas y Librerías utilizadas han jugado un rol importante en la simplificación de la complejidad en el desarrollo del proyecto.

UUAGC

- Permite enfocarse en la resolución del problema.
- Código compacto (reglas de copiado de UUAGC).
- Generar código Haskell sólo de las partes que se necesita.

uu-parsingLib

- Permite que la gramática implementada sea robusta.
- Implementar todo en Haskell.

wxHaskell

- Las variables de wxHaskell permitieron implementar el diagrama de renderización.

Limitaciones del Proyecto

Limitaciones del proyecto

Este proyecto corresponde simplemente a una pequeña parte de la amplia y compleja área de los Navegadores Web.

- Funcionalidad
  No se dio soporte a:
  - Formularios, Tablas y Frames de HTML.
  - Posicionamiento flotante, absoluto y fijo de CSS.
- Limitaciones de la librería wxHaskell
  - Color Transparente
  - Cambio de la fuente de un texto.

Recomendaciones para Trabajos Futuros

Recomendaciones para Trabajos Futuros

- Parser de HTML/XHTML + DTD
- Extender el soporte para la especificación de CSS.
  - Tablas
  - Posicionamiento flotante, absoluto y fijo
  - Más propiedades de CSS (sólo se implemento 48 de las 80 propiedades para la pantalla)
- Delegar la tarea de dimensionamiento y posicionamiento a la librería gráfica.
- Optimizar los algoritmos de renderización del proyecto.
- Implementar JavaScript.

Fin de la Presentación

Carlos Gomez
carliros.g@gmail.com
Desarrollo De Navegador Web Con Haskell
http://hsbrowser.wordpress.com