Modelo de Desarrollo de Interfaces en Lenguaje Funcional

María Claudia Albornoz, Claudia Mónica Necco, Germán Antonio Montejano
Facultad de Ciencias Físico Matemáticas y Naturales, Universidad Nacional de San Luis, Ejercito de los Andes 950, D5700HHW - San Luis - Argentina
Tel: +54 (2652) 424027 int. 251 - Fax: +54 (2652) 430059
e-mail:{albornoz, ayesha, gmonte}@unsl.edu.ar

RESUMEN

Desde no hace muchos años la Ingeniería del Software le ha dado una mayor importancia a la Ingeniería de Interfaces Gráficas de Usuario, siendo ésta una parte fundamental de cualquier aplicación, ya que es lo primero que ve el usuario. Hoy en día existen lenguajes de diseño de modelado de Interfaces, los cuales proponen distintos modelos y técnicas de modelado según el enfoque al que pertenecen.

El objetivo de este trabajo es proponer un modelo de desarrollo de Interfaces que integre el vocabulario de las herramientas modernas de desarrollo de Interfaces de Usuario Gráficas (GUIs) (frames, windows, botones, menúes, etc.) en el paradigma de la programación funcional, en particular en lenguaje Haskell.

El modelo será construido sobre dos librerías monádicas de nivel medio existentes: Gtk2Hs [1], basada en la herramienta multiplataforma Gtk+ [2], y wxHaskell [3], nativa y portable para lenguaje Haskell, construida sobre wxWidgets [4].

Este trabajo tambien contempla la implementación de una interfaz concreta en el marco del modelo propuesto a los fines de validar las afirmaciones realizadas sobre el mismo.

Palabras Claves: Interfaz Gráfica de Usuario (Graphic User Interfaz: GUI), Programación funcional, Lenguaje Funcional Haskell, Librerías Gtk2hs y wxHaskell.

CONTEXTO

Una definición de Ingeniería de Software pergeñada por Ghezzi (1991) establece que "la ingeniería de software se refiere a la construcción por parte de muchas personas, de múltiples versiones de software" [5]. La visión más popular de la ingeniería de software se concentra en la primera parte de esta definición, manejar equipos de personas para la producción de software a gran escala. La mayor parte de la gente adhiere a esta visión debido a que ven la ingeniería de software como una disciplina que promueve herramientas y técnicas de niveles artesanales a posiciones que permiten a los diseñadores manejarlas de manera eficiente y reproducible para completar grandes proyectos. Pero de igual manera es importante la segunda parte de la definición: identificar las partes específicas de un producto, para que puedan ser diseñadas por expertos, y producidas en línea, libres de dependencias de ambientes y lenguajes.

Las Interfaces de Usuarios Gráficas son un ejemplo de tales especializaciones y como tal pueden ser exitosamente diseñadas, producidas y manufacturadas, no en múltiples versiones (reminiscencia de "línea de producción"), sino de manera tal que sean portables entre múltiples plataformas y dispositivos.

La interfaz del usuario es la parte de un programa que maneja el flujo de datos producido durante el proceso de interacción entre el usuario y la computadora. El resto del programa es llamado la aplicación. Debido al hecho que la aplicación debería

brindar al usuario diferentes maneras de visualizar, concebir y manipular complejas estructuras de datos, es muy importante establecer un marco que permita enfrentar y manejar tal complejidad de manera coherente.

La programación de GUIs en lenguajes de programación funcional puros, se ha desarrollado en dos líneas: la aproximación funcional pura, evitando el monad IO imperativo, (Fudgets, FrankTk y, mas recientemente Fruit) o una aproximación más convencional, que utiliza el monad IO, y que ha permitido el desarrollo de bibliotecas de funciones para el desarrollo de GUIs que reflejan el estilo de programación orientado a objetos tradicional. En esta última línea podemos citar Gtk2Hs y wxHaskell como las librerias mas usadas.

Estas dos aproximaciones comparten un objetivo común: hacer disponibles los elementos que conforman el desarrollo de GUIs de una manera declarativa, de alto nivel dentro del contexto de un lenguaje funcional puro. Esto es, integrar el vocabulario de las herramientas modernas de desarrollo de GUIs (frames, windows, botones, menues etc) en el estilo de la programación funcional.

INTRODUCCIÓN

El modelado de Interfaz de Usuario es una técnica de desarrollo. La Interfaz es la carta de presentación del producto o aplicación, y desempeña un papel fundamental en la usabilidad del producto. Si la Interfaz está bien diseñada el usuario podrá encontrar respuesta a sus acciones. Por lo tanto el diseño y desarrollo de la Interfaz requiere de un proceso de desarrollo, que incluye elaboración de modelos visuales y notación estándar.

Algunos principios o características a tener en cuenta en una Interfaz son:
Anticipación: la aplicación se debe adelantar a las necesidades del usuario.
Autonomía: el ambiente debe ser 'flexible', así el usuario puede aprender rápida y fácilmente a usar la aplicación.
Percepción del color: si bien la creatividad juega un papel importante en el diseño, se debe tener en cuenta la armonía y la estética; pero además se deben incluir mecanismos para aquellos usuarios con problemas en la visualización del color.
Valores por defecto: la aplicación debe contar con valores inteligentes si el usuario no sabe responder en ese punto.
Consistencia: aquí se hace referencia a una gran variedad de temas de los cuales mencionaremos algunos. Por ejemplo, debemos ser consistentes en el uso de íconos. Un determinado ícono debe comportarse igual en toda la aplicación. Consistencia de diseño, la aplicación debe verse como una sola unidad y no una agrupación de componentes sin relación entre ellas. La forma de verificar la consistencia es a través del testeo.
Eficiencia: los mensajes de ayuda deben ser sencillos y proveer respuestas a los problemas (Help). Los menúes y etiquetas de botones deberían tener las palabras claves del proceso (ok, cancel, etc).

El modelo de una Interfaz de Usuario intenta mostrar cómo interactuará el usuario con la aplicación, y cómo responderá ésta.

Las técnicas de modelado de las interacciones pueden describir objetos, tareas y diálogos. El modelado de los casos de uso pueden ayudar a captar los requerimientos del usuario, evitar usar prematuramente diseños específicos y widgets, y hacer explícitas las relaciones entre las diferentes partes de la Interfaz.

Existen distintos lenguajes para el modelado de interfaces:

- UML: diversos aspectos del la Interfaz de Usuario pueden modelarse con éste lenguaje, pero UML no está específicamente destinado a éste tipo de aplicaciones.

- UMLi: es una extensión de UML, al que se le ha añadido soporte para la representación de Interfaces de Usuario. Dado que los modelos de aplicación de UML describen pocos aspectos de la Interfaz de Usuario, y que los ambientes de desarrollo de modelos basados en Interfaz de Usuario (MB-UIDE) carecen de habilidad para el modelado de aplicaciones, La Universisdad de Manchester inició el proyecto de Investigación UMLi en 1998. El fin de UMLi es abordar el problema de diseñar e implementar Interfaces de Usuario utilizando una combinación de UML y MB-UIDE.
- DiaMODL: combina un lenguaje orientado al flujo de datos con diagramas de estado de UML. Se puede modelar el flujo de datos como, asi también, el comportamiento de los objetos de interacción. Se lo puede utilizar para documentar la funcionalidad y la estructura de Interfaces de Usuario.
- Himalia: combina Modelos Hipermedia con el paradigma control/compuesto. Es un lenguaje completo de Interfaz de Usuario, él cual no sólo se lo puede utilizar para la especificación sino también para la ejecución.

Los distintos aspectos de la interfaz de Usuario requieren distintos tipos de modelos:

- Modelo del Dominio: incluyendo el modelo de datos (define los objetos que el usuario puede ver, acceder y manipular).
- Modelo de Navegación: define cómo navegan los objetos que el usuario ve.
- Modelo de tareas: describe las tareas que realizará el usuario final y dictamina la capacidad de interacción que debe ser diseñada.
- Modelo de usuario: representa las diversas características del usuario final y los roles que desempeña.
- Modelo de plataforma: usado para modelar los dispositivos físicos en donde se desarrolla la aplicación y cómo interactúan entre ellos.
- Modelo de diálogo: muestra cómo el usuario puede interactuar con los objetos (botones, comandos, ventanas, etc.), con los medios de interacción (entrada de voz, pantalla táctil, etc.) y las reacciones que la Interfaz comunica a través de ésos objetos.
- Modelo de presentación: modela la apariencia de la aplicación, representa los elementos visuales, táctiles y auditivos que la Interfaz ofrece a los usuarios.
- Modelo de aplicación: representa los comandos y datos que la aplicación provee.

UML puede ser usado para varios de los modelos mencionados con un variable grado de éxito, pero carece de soporte para el Modelo de usuario, Modelo de plataforma y Modelo de presentación.

También se debemos mencionar los distintos enfoques en el modelado de Interfaces de Usuario:

- Diseño centrado en el Uso: desde ésta perspectiva, la tarea de modelado es mostrar cómo ocurre la interacción entre el usuario y la presentación de un sistema planificado. Posiblemente éste sea el método más elogiado, y es el que sido utilizado con mas éxito en una gran variedad de proyectos.
- Modelos de Contenido: éste tipo de modelos muestran el contenido de una Interfaz de Usuario y sus componentes. No incluyen detalles estéticos ni de comportamiento.

Hasta aquí se han mencionado conceptos básicos respecto de Interfaces, del modelo de Interfaz de Usuario en general, de los lenguajes actuales para el modelado y los distintos submodelos que comprende un modelo. A continuación se presentarán las distintas librerías gráficas con las que cuenta el lenguaje funcional Haskell.

Existe un gran número de librerías de Interfaz Gráfica de Usuario para Haskell, aunque ninguna de ellas es estándar y todas son algo incompletas. Se las puede clasificar como:

- Alto nivel: Fruit, wxFriut.
- Nivel medio: Gtk2Hs,Htk,wxHaskell.
- Bajo nivel: GLFW, TclHaskell.
- No categorizadas y sin soporte: Gtk+Hs, iHaskell.

Por otra parte, existen dos estilos de librerías:

- Declarativa: es la aproximación funcional pura, evitando el uso de Monad IO imperativo (estilo point-free).
- Monádica: que refleja el estilo de programación orientada a objetos tradicional.

En éste último estilo se enmarcan las dos librerías para la generación de Interfaces Gráficas de Usuario de nivel medio que se abordarán en el presente trabajo:

- librería *Gtk2Hs*, basada en la herramienta multiplataforma Gtk+ , y
- librería *wxHaskell*, nativa y portable para lenguaje Haskell, construida sobre wxWidgets.

Gtk2Hs: Gtk+ es un conjunto de herramientas (toolkit) extenso, maduro y multi-plataforma. Gtk+ es fundamentalmente una biblioteca de componentes llamados 'widgets', que se organizan en una jerarquía orientada a objetos, es decir en clases. En Gtk2Hs esto se hace dando a cada widget tanto un tipo como una clase de tipo (type y type class) de Haskell.

- Misceláneas: botones, sliders, …
- Agregados: selector de archivos, calendario, …
- Contenedores: ventanas, grids,….
- Activos: menúes, barras de herramientas.

De este modo el sistema de tipos de Haskell está rigurosamente preservado, y el programador tiene todas las ventajas de la comprobación de tipos del compilador y del intérprete de Haskell de Glasgow (ghc). Gtk2Hs tiene también muchas funciones para el casting de tipos.

Los widgets tienen atributos que controlan su comportamiento, apariencia y contenido

Todo lo que sucede en una aplicación en Gtk2Hs está determinado por las acciones del usuario, como pulsaciones de ratón, presionado de botones, selección de botones de radio y demás. A esto se le llama Programación por Eventos y resultan en señales emitidas. La información sobre los eventos, por ejemplo cuando el usuario pulsa el botón derecho o izquierdo del ratón, se captura en campos de datos que pueden ser consultados por el programador de Gtk2Hs. Normalmente las señales se gestionan por funciones específicas de cada tipo de widget. Se listan en la sección **señales (*Signals*)** de la documentación de la API de ese widget. La mayoría toman una función de tipo `IO ()` como parámetro, pero algunas tienen algún resultado. El programador de Gtk2Hs, por supuesto, debe escribir esas funciones que determinan la respuesta del sistema a la acción del usuario.

Para el uso de la biblioteca Gtk2Hs, es necesario que se instale GHC, que es el compilador de Haskell creado por la Universidad de Glasgow.

WxHaskell: es un enlace Haskell a la biblioteca GUI WxWidget. La biblioteca Graphics.UI.WX es utilizada para escribir Interfaces Gráficas de Usuario y está construida sobre Graphics.UI.WXCore, núcleo de la Interfaz de WxWidget. WxWidget es una biblioteca escrita en C++ que es portable a través de las principales plataformas GUI (GTK, Windows, MacOSX). Soporta un amplio rango de widgets.

WxHaskell no soporta la progremación multihebra (estilo de ejecución en el que se conmuta entre distintas partes del código de un mismo programa durante la ejecución. A cada una de estas partes individuales y atómicas(no divisibles) se les dá el nombre de Threads (traducidos como hilos,hebras)

WxHaskell permite crear botones, cajas de texto, menús, etc. y especificar la disposición física de los componentes utilizando espacios de trabajo al estilo del lenguaje Java (éstos espacios sirven para organizar los Widgets. También trabaja con Programación por Eventos.

Ésta librería utiliza características importantes de Haskell, como el polimorfismo paramétrico, funciones de orden superior y evaluación perezosa. Los programas resultantes tienden a ser más cortos y más claros que sus homólogos en C++. Se puede interactuar con código Haskell.

LÍNEAS DE INVESTIGACIÓN Y DESARROLLO

El presente trabajo tiene como objetivo proponer un modelo para el desarrollo de Interfaces Gráficas de Usuario para construcción de sistemas en el paradigma de la programación funcional.

Para tal fín se necesitará hacer:

- un estudio "riguroso" de la Ingeniería de Interfaces, del lenguaje funcional Haskell y sus operaciones de entrada-salida, el uso de Monads, las bibliotecas Gtk+, Glade, wxWidgets y librerías Gtk2Hs, wxHaskell.
- desarrollo de aplicaciones concretas usando las librerias escogidas a los fines analizar las ventajas y desventajas de las mismas.
- la propuesta de un modelo de desarrollo que, en la medida de lo posible, incorpore las ventajas y subsane las desventajas y funcionalidades que no hayan sido tenidas en cuenta en las librerias analizadas.
- validación del modelo propuesto mediante el desarrollo de una interface concreta construida en el marco del mismo.

RECURSOS HUMANOS

El presente trabajo es la descripción de la tesis de Maestría en Ingeniería del Software de la Universidad Nacional de San Luis de la Prof. María Claudia Albornoz, dirigida por la Mg. Cs. Claudia Monica Necco y el Mg. Ing. Germán Montejano.

REFERENCIAS

[1] A GUI library for Haskell based on Gtk, http://haskell.org/gtk2hs/

[2] The GTK+ Project, http://www.gtk.org/

[3] WxHaskell. (Categories: User interfaces | Libraries | WxHaskell | Packages), http://haskell.org/haskellwiki/WxHaskell

[4] Cross-Platform GUI Library, http://www.wxwidgets.org/

[5] C. Ghezzi, M. Jazayeri, and D. Mandrioli, Fundamentals of Software Engineering, Prentice Hall, 1991.

[5] Judith Bishop, Nigel Horspool, "Cross-Platform Development: Software that Lasts", *Computer*, vol. 39, no. 10, pp. 26-35, October, 2006.

Bibliografía consultada:

http://sebastiangomez.sytes.net/papers/DIU.pdf

http://en.wikipedia.org/wiki/User_Interface_Modeling

http://trust.utep.edu/umli/index.html

http://www.haskell.org/haskellwiki/Applications_and_libraries/GUI_

http://ldc.usb.ve/~emhn/cursos/ci4251/201004/17-gui.pdf

http://www.muitovar.com/gtk2hs/es-chap1.html

http://osl2.uca.es/wikihaskell/index.php/Biblioteca_WxHaskell

(Última visita a los sitios web: Abril de 2011)