Instituto Tecnológico de Costa Rica

Sede Central

Algoritmos y estructuras de datos I

Segundo proyecto programado

Flow Plugin

Profesor:

Isaac Ramírez Herrera

Estudiantes:

Jasson Rodríguez

Marco Herrera

2017

Tabla de contenidos

Introducción

La plataforma de Eclipse provee un sistema extensible para la construcción de complementos y aplicaciones de una forma modular (Blewitt, 2014). Eclipse Flow Plugin es un complemento para el entorno de desarrollo Eclipse 4 el cuál permite la generación de diagramas de flujo interactivos a partir del código fuente del proyecto.

Según Lucid Software (2017) un diagrama de flujo es un diagrama que describe un proceso, sistema o algoritmo informático. Se usan ampliamente en numerosos campos para documentar, estudiar, planificar, mejorar y comunicar procesos que suelen ser complejos en diagramas claros y fáciles de comprender. Los diagramas de flujo emplean rectángulos, óvalos, diamantes y otras numerosas figuras para definir el tipo de paso, junto con flechas conectoras que establecen el flujo y la secuencia.

Por otra parte es importante definir el concepto de plug-in o componente en Eclipse pues facilita el desarrollo del presente proyecto, según Arthorne J. (2011) plug-in, tal vez no era el término más apropiado para los componentes que crean una aplicación en Eclipse. El término implica la existencia de un socket, una máquina o red monolítica en la que se está conectando. En Eclipse, este no es el caso. Un plug-in se conecta con un universo de otros plug-ins para formar una aplicación en ejecución. La mejor analogía de software compara un plug-in a un objeto en la programación orientada a objetos. Un plug-in, como un objeto, es una encapsulación de comportamiento y/o datos, que interactúa con otros plug-ins para formar un programa en ejecución.

Descripción del problema

El problema consiste en el diseño, desarrollo y verificación de un complemento o plugin para el entorno de desarrollo Eclipse, dicho complemento debe encargarse de generar un diagrama de flujo de los métodos de la clase abierta.

El diagrama de flujo creado deberá cumplir con las normas establecidas en el UML (Unified Modeling Language) para obtener de esta forma un diagrama universal fácil de entender.

Además, el complemento deberá sincronizarse con el debugger de Eclipse para mostrar en tiempo real el proceso de depurado del código de un manera más gráfica, permitiendo al usuario observar el código que se está ejecutando.

Bitácora de actividades de Jasson Rodríguez

8/10/2017 8:00 a 9/10/2017 1:00 Investigación Commands, Handlers y Menu

9/10/2017 11:00 a 12:00 Investigación toolbars

14/10/2017 11:00 a 15/10/2017 3:00 Creación de gráficos

15/10/2017 11:00 a 16/10/2017 1:00 Creación de gráficos

16/10/2017 1:30 a 2:00 Documentación interna

16/10/2017 5:15 a 7:15 Barras de desplazamiento de la ventana principal

18/10/2017 5:30 a 8:00 Vista para el menú de métodos y reordenamiento de la ventana principal.

19/10/2017 9:30 a 10:00 Documentación

22/10/2017 11:00 a 23/10/2017 1:00 Documentación y manual

24/10/2017 7:45 a 11:30 Diseño de la aplicación

25/10/2017 7:00 a 2:00 Conversion del ASTStorage a gráficos

25/10/2017 7:15 a 11:00 Documentación y manual

Bitácora de actividades de Marco Herrera

03/10/2017 - 1h - *"Eclipse RCP Tutorial"* - Vogella

05/10/2017 - 1.5h - *"Eclipse plugin Tutorial"* - Vogella, *"Extension and Extension Points"* - Vogella

08/10/2017 - 2h - *"Eclipse JDT - Abstract Syntax Tree (AST) and the java model Tutorial"* - Vogella

10/10/2017 - 1.5h - Intento fallido de leer el AST.

12/10/2017 - 5h - Función de parseo del AST con ayuda del ASTView

14/10/2017 - 2h - Crear estructura alterna que guarda los componentes del AST

16/10/2017 - 3 - Investigar sobre el Debugger de Eclipse

17/10/2017 - 4.5h - Leer y probar diferentes opciones de elementos en documentacion de eclipse.

19/10/2017 - 2h - Implementar la clase para escuchar el breakpoint del debugger, "*BreakPointListener*"

19/10/2017 - 3h - Investigar como escuchar el debugger no solo cuando choca con un breakpoint

19/10/2017 - 1h - Documentación externa

23/10/2017 - 5h - Utilizar el thread del debugger para activarlo y mantener referencia a la linea que se está ejecutando

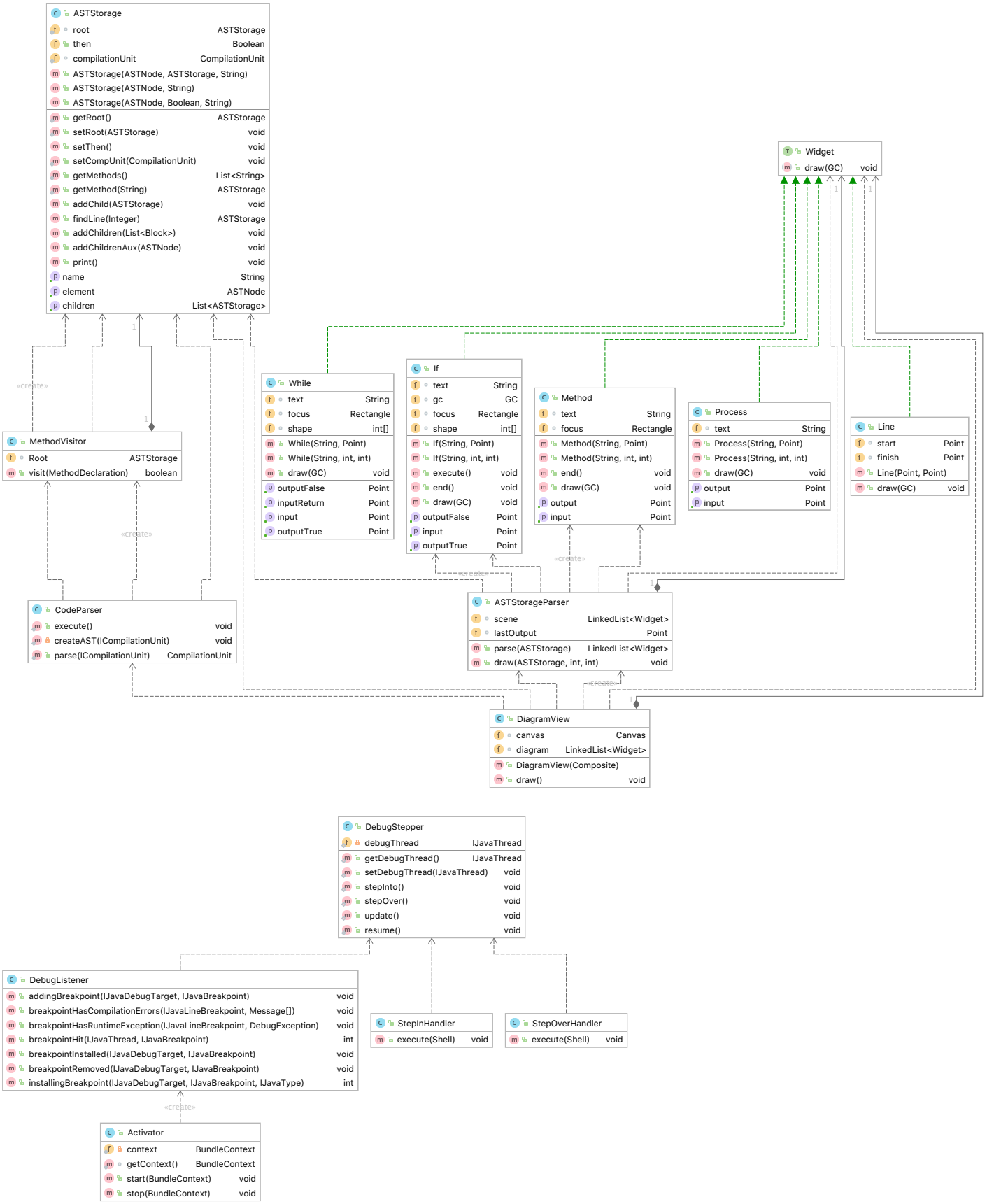
24/10/2017 - 4h - Reunión Grupal

25/10/2017 - 2h - Documentación externa

Función	Tiempo (h)	
	Marco Herrera	Jasson Rodríguez
Análisis de requerimientos	1	1
Diseño de la aplicación	4.5	3.75
Programación	27	16.5
Documentación interna	0.75	0.5
Documentación de usuario	0.5	2
Documentación técnica	3.75	4.25
Total	37.5	28

Diseño

Diagrama de clases



Implementación

Descripción de las bibliotecas utilizadas

SWT: Es una suite de herramientas de widgets de código libre para Java. Está diseñada para proveer un acceso portátil y eficiente a la interfaz gráfica de los sistemas operativos sobre los cuales está implementada.

AST: Esta biblioteca permite acceder al árbol de sintaxis abstracta (Abstract Syntax Tree) del código de Java. El árbol es la representación que se le da a todos los archivos de código para poder ser comprendidos y compilados.

Descripción de las estructuras de datos utilizadas

Se desarrolló una estructura jerárquica con el fin de almacenar de forma temporal los distintos componentes sintácticos del código que se desea parsear, esta estructura es una representación más sencilla del Abstract Syntax Tree implementado por eclipse, de manera que mediante un atributo llamado Children se puede acceder a los componenetes que se encuentran dentro de otro.

Descripción detallada de los algoritmos desarrollados

El algoritmo encargado de crear los diagramas de flujo consiste en analizar de manera recursiva el árbol y crear una instancia de la clase gráfica correspondiente. Durante este proceso es necesario mantener variables para almacenar la ultima conexión y el ancho máximo del diagrama con el propósito de conectar todos los dibujos mediante líneas y además, ajustar el tamaño del canvas correctamente.

Problemas encontrados

Uno de los problemas que se presentaron fue la forma en que se actualiza un widget de SWT al que se le asignó un PaintListener. De acuerdo con el funcionamiento de Eclipse la ventana se actualiza cada vez que una parte de ella debe ser dibujada nuevamente, por ejemplo, al cambiar su tamaño. Esto ocasionó que la idea inicial de convertir el árbol de AST a gráficos proporcionara un rendimiento pobre al actualizarse frecuentemente. La solución ideada para este problema fue crear una lista con todos los elementos gráficos que se deben dibujar durante cada actualización para así no

tener que convertir todo el árbol tan frecuentemente y que únicamente ocurra cuando se cambia la clase o método seleccionado.

Otro problema encontrado fue la interferencia que ocurre en el "thread" del debugger cuando este se encuentra haciendo "stepping" lo cual se solucionó con un ciclo que revisa constantemente el estado del "thread"

Repositorio de versiones:

https://github.com/JassonRM/Flow

Conclusión

Se comprendió que los complementos son de gran importancia para proporcionar funcionalidades adicionales a paquetes de software, especialmente aquellos que funcionan bajo la metodología open source, ya que permiten a diferentes desarrolladores contribuir con sus ideas y experiencia.

Bibliografía

Arthorne, J. (2011). FAQ What is a plug-in?. Recuperado el 24 de Octubre de 2017, de Eclipse: https://wiki.eclipse.org/FAQ_What_is_a_plug-in%3F

Blewitt, D. A. (2014). Mastering Eclipse Plug-in Development. Packt Publishing. Birmingham, UK: Packt Publishing Ltd.

Lucid Software. (27 de Septiembre de 2017). *Qué es un diagrama de flujo*. Recuperado el 23 de Octubre de 2017, de Lucidchart: https://www.lucidchart.com/pages/es/qué-es-un-diagrama-de-flujo