Depuración Orientada a Código Fuente de Programas Prolog basada en Eclipse

Juan Angel Vanrell Claudio Vaucheret
Departamento de Ciencias de La Computación
Facultad de Economía
Universidad Nacional del Comahue
email: {javanrell,vaucheret}@gmail.com

Resumen

En este trabajo presentamos la implementación de componentes para un plug-in de Eclipse que permite la visualización del Modelo de Byrd de depuración de programas lógicos sobre el código fuente de los programas. La depuración en el código fuente es una característica esencial en un entorno de programación moderno para el lenguaje Prolog. Este trabajo contribuye en la creación de un entorno de programación visual de código abierto para Prolog.

Abstract

In this paper we present the implementation of components for an Eclipse plug-in that allows to visualize the Byrd Model for logic programs source debugging. Source debugging is an important feature in moderns enviroments of programming for Prolog languages. This work contributes with the creation of an open source visual enviroment for Prolog.

1. Introducción

Este trabajo expone la construcción de un esquema de depuración de código fuente[1] para Ciao Prolog en la plataforma Eclipse (ver [8] y [7]).

Se busca con el mismo proveer un entorno amigable tanto para principiantes como para usuarios expertos para el uso de Ciao de forma que el usuario sólo necesite aprender las características del lenguaje.

[1] En ingles: source debugger

Antes de este trabajo las formas de utilización del ambiente Ciao Prolog se limitaban a un shell en modo texto y al entorno de programación Emacs. Éste último posee un rico entorno con herramientas de ayuda, depuración fuente y preprocesador, además de ser, al igual que el Ciao, distribuido bajo licencia GNU junto con su código fuente para que sea modificado por cualquier persona.

El problema principal del Emacs es su entorno poco amigable, básicamente se trabaja en modo texto, con muy poco uso de las bondades de los entornos gráficos actuales. Al mismo tiempo posee una definición propia del uso de elementos de tal forma que es necesario aprender a utilizar la plataforma antes de poder trabajar cómodamente con Ciao.

A diferencia del Emacs, Eclipse posee un rico entorno gráfico que se asemeja a la mayoría de las plataformas actuales, utiliza conceptos estandarizados para la realización de acciones (como pueden ser las formas de copiar y pegar texto, navegación de recursos, etc) y al igual que Emacs y Ciao es distribuido de forma libre y gratuita como un proyecto de código abierto.

Si bien actualmente no puede ser reemplazo del Emacs (ya que el proyecto se encuentra en la etapa inicial) es de esperar que adquiera la funcionalidad del mismo con el correr del tiempo.

En la figura 1 puede verse la opción de source debugger que hemos implementado funcionando bajo la plataforma Eclipse.

En ella pueden observarse los tres componentes básicos del entorno, a la izquierda el Navegador de Recursos (desde donde se puede interactuar con los archivos, carpetas y proyectos), a la derecha, en la parte superior, se encuentra el editor con características como el resaltado de sintaxis y debajo del mismo la consola de ejecución sobre la cual podemos ingresar consultas y recibir respuestas de ejecución y errores.

Puede verse en la consola una secuencia de ejecución de un programa prolog junto con la información de salida para el source debugger que es provista por el shell de Ciao y el coloreo del éxito de la llamada al predicado *antecesor(juan,ana)*.

1.1. Ciao

Ciao[2] es un ambiente de programación multiparadigma de próxima generación de dominio público con las siguientes características:

- Ofrece un sistema Prolog completo, soportando ISO-Prolog, pero con un nuevo diseño modular que permite restringir o extender el lenguaje en cada módulo de programa individualmente permitiendo que distintas extensiones coexistan en una misma aplicación para distintos módulos.

- A través de sus extensiones permite programar con funciones, alto nivel (con abstracción de predicados), restricciones y objetos, así como con características como registros, persistencia, distintas reglas de control (breadth-first search, iterative

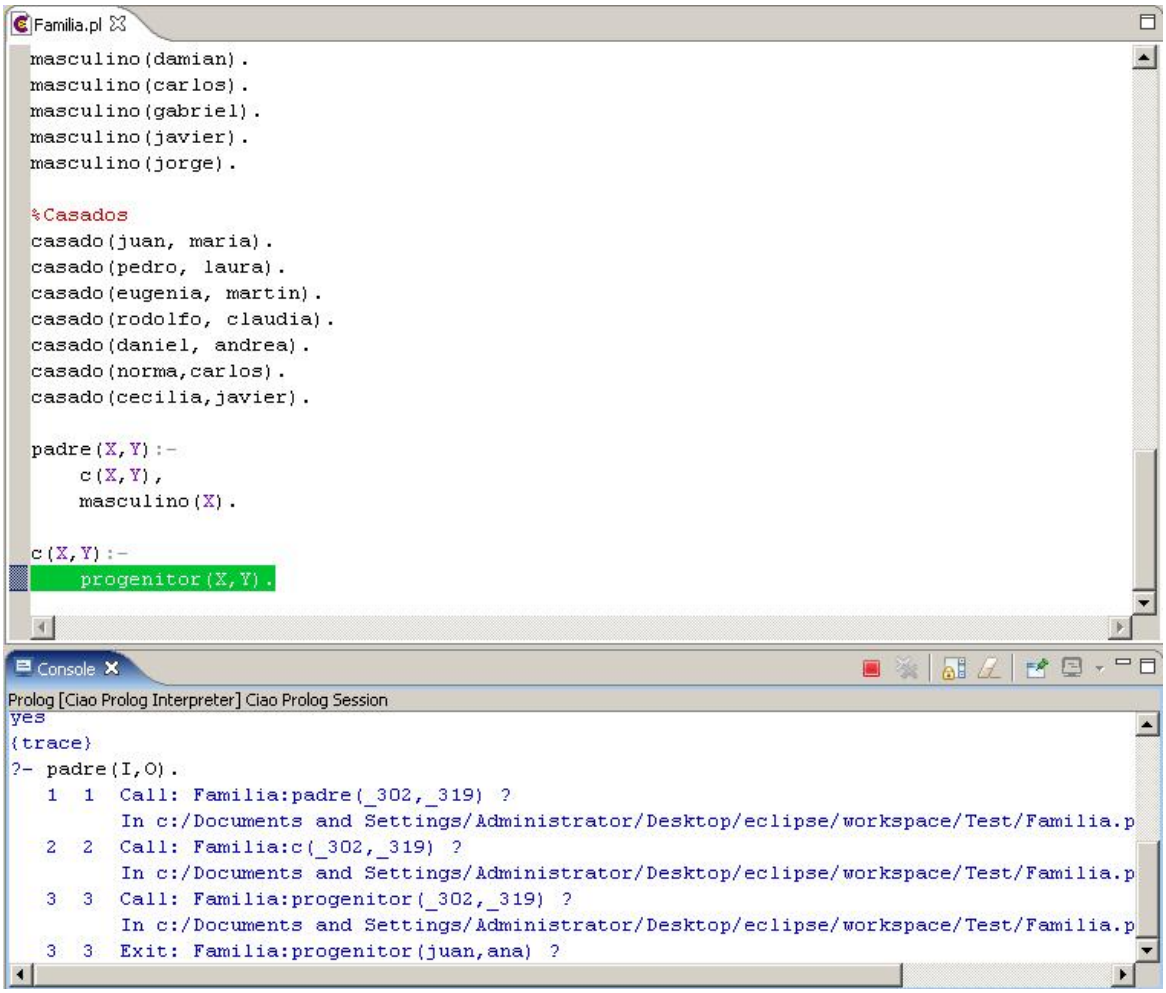

Figura 1: Source Debugger en Eclipse

deepening, etc), concurrencia, una buena base para ejecución distribuida (agentes), y ejecución paralela. Las librerías también soportan programación WWW, socket, interfaces externas (C, Java, TclTk, bases de datos relacionales, etc).

- Ofrece soporte para grandes programas a través de un sistema modular de compilación incremental separada.
- También permite la creación de pequeños ejecutables y soporte para scripts.
- Es distribuido bajo GNU Library General Public License (LGPL).

1.2. Eclipse

Eclipse es una plataforma universal para la integración de herramientas de desarrollo. Es de arquitectura abierta y extensible basada en componentes añadidos (plug-ins). Inicialmente desarrollado por OTI e IBM evolucionó en un proyecto de código abierto. La plataforma Eclipse está diseñada para afrontar las siguientes necesidades[4]:

- Soportar la construcción de gran variedad de herramientas de desarrollo.
- Soportar las herramientas proporcionadas por diferentes fabricantes de software independientes (ISV's).
- Soportar herramientas que permitan manipular diferentes contenidos (HTML, Java, C, JSP, EJB, XML, GIF, etc).
- Facilitar una integración transparente entre todas las herramientas y tipos de contenidos sin tener en cuenta al proveedor.
- Proporcionar entornos de desarrollo gráfico (GUI) o no gráficos.
- Ejecutarse en una gran variedad de sistemas operativos, incluyendo Windows y Linux.
- Hacer hincapié en que el lenguaje de programación sea Java para la construcción de nuevos plug-ins.

El articulo se organiza de la siguiente manera en la sección 2 se resume las características de Eclipse; en la sección 3 se explica el modelo de depuración en Prolog; en la sección siguiente se muestra nuestra implementación y por ultimo en la sección 5 se presentan conclusiones y trabajo futuro.

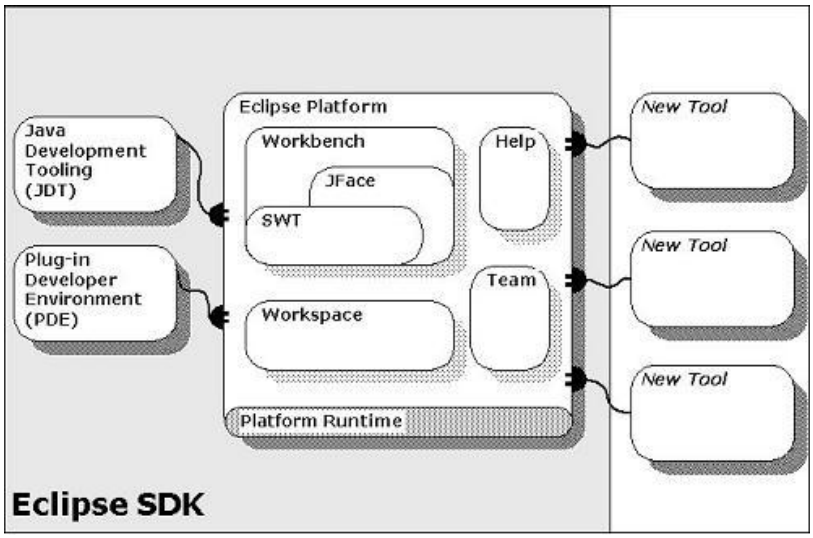

Figura 2: Plataforma Eclipse y sus plug-ins

2. Plataforma de Desarrollo de Software Eclipse

La plataforma de desarrollo Eclipse no posee en si misma funcionalidad alguna. Es a través de los plug-ins, que le incorporan funcionalidades para el desarrollo de tareas específicas, que obtiene la funcionalidad adecuada para cada usuario.
Como se puede ver en la figura 2 existen conectores básicos que dan funcionalidad a la plataforma definiendo puntos de extensión con los cuales un plug-in puede contribuir. Estos conectores son Help (que aporta funcionalidad común para la introducción de ayudas en la plataforma (tanto estática como dinámica y emergente)), Workbench (que aporta funcionalidad para el uso del entorno gráfico de trabajo), Debug (que aporta la funcionalidad común para el lanzamiento y depuración de programas), etc. Para mayor información o detalle vea [3].
Nuestro plug-in declarará el uso de algunos de los puntos de extensión, provistos por estos conectores en el archivo *plugin.xml*, denominado manifiesto, que define las propiedades del plug-in, los puntos de extensión que provee o implementa, datos del creador, librerías que utiliza, etc. Además proveerá la implementación de los puntos de extensión que declare, entre los cuales se encuentran:

- **org.eclipse.debug.ui.ConsoleColorProviders** que define mecanismos para un esquema de coloreado de un documento de consola asociada a un proceso.
- **org.eclipse.debug.ui.ConsoleLineTrackers** que provee mecanismos para la escucha de eventos de inserción en una consola asociada a un proceso.

ambos serán utilizados para lograr el source debugger.

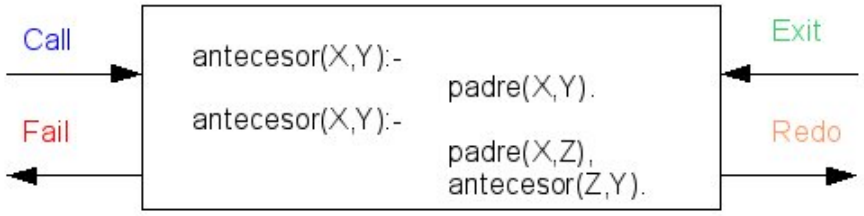

Figura 3: Modelo de Byrd

3. Modelo de Byrd e implementación de Ciao Prolog

A diferencia de los lenguajes imperativos en el que una ejecución siempre posee una correspondencia única con cada parte del código fuente, la ejecución de un predicado en los lenguajes declarativos como Ciao pueden resultar en cuatro tipos distintos de resultado. Como puede verse en [5], cuando utilizamos el source debugger debemos estar atentos a cuatro tipos distintos de eventos correspondientes a los cuatro puertos que el Modelo de Byrd (figura 3) asocia a cada uno de los literales que componen un programa fuente, estos son:

- Call: llamada a un predicado
- Exit: éxito de un predicado
- Fail: falla de un predicado
- Redo: backtraking

El top-level de Ciao Prolog implementa este modelo.
En cada una de las detenciones provee la siguiente salida, estando en modo *traza*:

```
   In /home/jvanrell/ciao/familia.pl (5-7) antecesor-1
S 13 7 Call: T user:antecesor(roberto,_123) ?
```

La primer línea indica el punto del programa en donde se está ejecutando y sus campos indican la dirección completa del archivo fuente que se está depurando, los números de líneas de inicio y fin de un rango en el cual el literal puede ser encontrado dentro de ese archivo, la cadena de caracteres a ser localizada y el número de ocurrencia de la misma. Con esta información podemos obtener el número de línea que debe ser coloreado para el seguimiento de la ejecución.

La segunda línea indica lo que se está ejecutando. De esta línea sólo nos interesa (a fines de la implementación del source debugger) el tipo de puerto que la genera

(indicado como cuarto valor) que nos permite configurar un color distinto para cada una de las cuatro opciones de información vistas anteriormente.

Dicha información permite el coloreo de código fuente de forma que pueda seguirse la ejecución de un programa. Para ello coloreamos aquél predicado que se está utilizando en cada momento con un color distintivo según el puerto por el cual se está obteniendo la información. Por defecto estos colores son azul para una llamada, verde para un éxito, rojo para una falla y naranja para un redo.

4. Implementación

Entre otras herramientas Ciao incluye un shell interactivo (top-level) llamado **ciao** o **ciaosh** que contiene herramientas para la construcción incremental de programas, debugging y ejecución así como la modificación de los mismos sin necesidad de iniciar desde cero. Al iniciar el shell recibimos como se muestra a continuación un mensaje de bienvenida y un prompt (?-) indicando que está listo para recibir instrucciones.

```
Ciao-Prolog 1.10 #5: Fri Aug 6 19:01:54 2004
?-
```

Una vez inicializado el shell podemos invocar distintos predicados nativos. Para el debugger de programas utilizaremos los siguientes:

- use_module/1: `?- use_module(Module).` carga en el top-level el módulo definido por Module, importando todos los predicados que el exporta con lo cual se encuentran disponibles para ser invocados. Module es el nombre completo del archivo que declara el módulo a cargar.
- debug_module_source/1: `?- debug_module_source(Module).` carga en el top-level el módulo definido por Module en modo debug de forma que pueda ser depurado. Es importante destacar que sólo los módulos que se carguen con este predicado van a ser depurados, por lo que sólo cargaremos aquellos que nos interese revisar y pasando por alto los restantes que serán ejecutados en modo interpretado (de forma mucho más veloz). Module es el nombre del módulo.
- trace/0: `?- trace.` inicializa la traza habilitando el debugger. El interprete va a detenerse en cada salida de los puertos de los predicados que sean marcados para debug, imprimiendo un mensaje en cada punto de detención y esperando una entrada del usuario.

Eclipse provee algunas herramientas útiles al momento de generar plug-ins a través del PDE (Plug-in Developement Enviroment) que provee un conjunto de extensiones de la plataforma (vistas, editores, perspectivas, etc) que permite la generación de plug-ins

dentro del mismo workbench de Eclipse. PDE está basado en la plataforma y en Java Developement Tooling (JDT). Para mas información vea [1] o [3].

Junto con los puntos de extensión Eclipse define una API que permite la implementación de las características más comunes que puede necesitar nuestro plug-in.

En la figura 4 podemos ver un esquema de la forma en que nuestro plug-in contribuye con Eclipse.

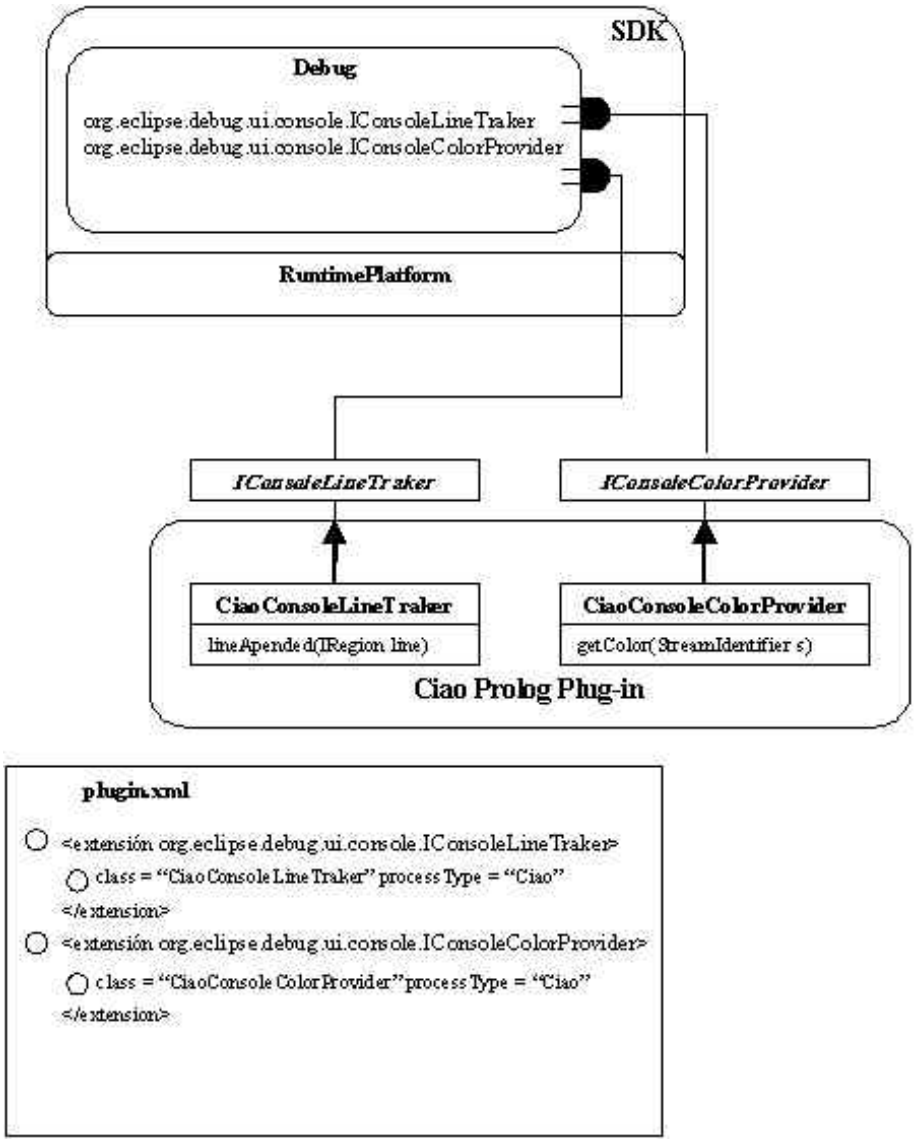

Figura 4: Esquema de conexión con Eclipse

El plug-in Debug provee distintos puntos de extensión para lanzamiento y depuración de programas. Entre éstos se encuentran **org.eclipse.debug.ui.console.IConsoleLineTraker** y **org.eclipse.debug.ui.console.IConsole ColorProvider** que pro-

veen comportamiento a la consola que Eclipse asocia a cada proceso lanzado. Eclipse identifica a cada tipo de proceso obteniendo el atributo `ATTR_PROCESS_TYPE` que puede ser seteado cuando se lanza el proceso [6].

Para proveer el comportamiento deseado declaramos en el archivo *plugin.xml* (el manifiesto) la extensión de los puntos antes mencionados, declarando el tipo de proceso al cual van a proveer el comportamiento y las clases que implementan las interfaces que declara cada punto de extensión (etiquetas **processType** y **class**). Las interfaces a implementar son **IConsoleColorProvider** para el coloreo distintivo de streams del proceso en la consola (input, output y error), e **IConsoleLineTraker** para la captura de líneas que van a ser agregadas en la consola (de los tres streams del proceso).

El framework de edición de texto sigue los mismos principios arquitectónicos del resto de la plataforma Eclipse. Se divide en cuatro capas: Model (core), View (SWT), Controller (JFace) y Contexto de presentación (usualmente el workbench) Tanto el model como el view son piezas autocontenidas que no conocen nada acerca del resto de las piezas del framework. En el caso de la edición de texto el core es **org.eclipse.jface.text.IDocument** que no tiene dependencia con ninguna de las piezas del resto del modelo UI. El view es **org.eclipse.swt.custom.StyledText**. La capa básica de control es provista por **org.eclipse.jface.text.ITextViewer** que es extendida por **org.eclipse.jface.text.ISourceViewer** para proveer comportamiento particular de los editores de los lenguajes de programación. El contexto para la presentación de editores de texto en una parte del workbench es provisto por **org.eclipse.ui.texteditor.ITextEditor**.

En la figura 5 podemos ver como colaboran los elementos de la plataforma con las clases definidas por nuestro plug-in y los puntos de extensión definidos.

El primer paso es definir el coloreo de la consola, para esto debemos implementar la interfase *IConsoleColorProvider* definiendo el método *getColor*.

El siguiente paso es reconocer las salidas del top-level para lo cual implementamos la interfase *IConsoleLineTracker* que nos da acceso a cada una de las líneas que se van a agregar a la consola. Como el top-level coloca en la salida más de una línea simultáneamente debemos agregar un listener que obtenga toda las líneas de entrada, de forma que nos permita poseer la información completa para el coloreo. Esto se hace en el método *init(IConsole c)* que obtiene el *IDocument* asociado a la consola y le agrega el *CiaoDocumentListener* al conjunto de listeners del mismo.

El método principal de este listener es *documentAboutToBeChanged(DocumentEvent event)* que obtiene el stream a ser colocado en la consola. Sabiendo esto filtramos la información en base a los patrones de caracteres “Call:”, “Exit:”, “Redo” y “Fail:” para obtener el color con el que debe ser coloreada la línea en base al puerto del Modelo de Byrd al cual corresponde la información.

Cada vez que se agrega una línea a la consola se genera un evento que es capturado por el método *lineAppended(IRegion line)* de esta forma podemos revisar si es una línea de depuración y actuar en consecuencia.

Como Eclipse posee threads separados para UI, todas las ejecuciones UI deben ser realizadas en threads-UI, ya que de no ser así pueden provocar errores de ejecución. Por

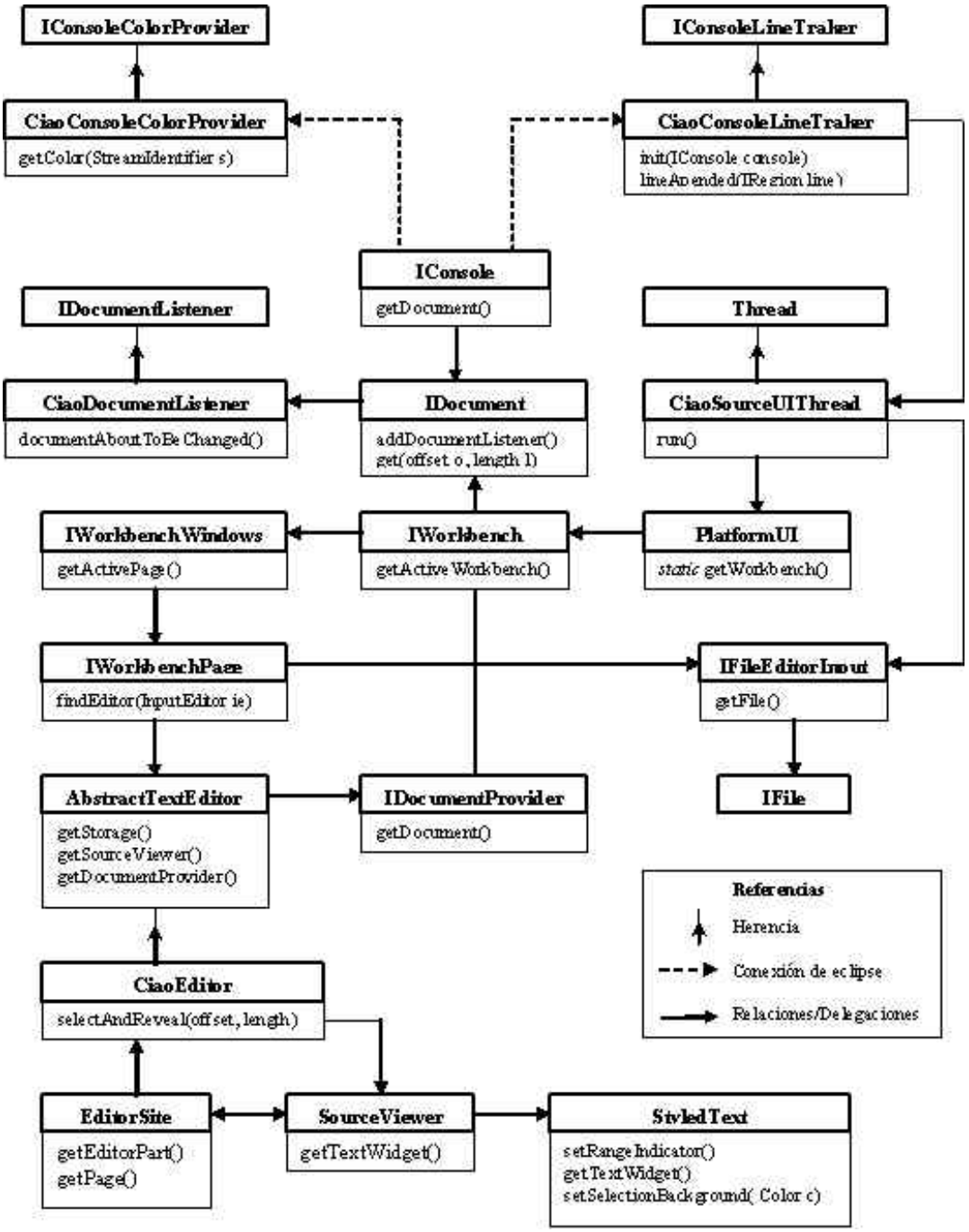

Figura 5: Diagrama de clases relacionadas a la implementación del coloreo de código fuente

lo tanto, generamos un nuevo thread (*CiaoSourceManagerUIThread*) que será ejecutado en forma sincrónica con nuestro procesamiento de entrada y que será el encargado de obtener el editor y colorear sus líneas.

Este thread obtiene el archivo sobre el cual vamos a colorear la línea, abre el editor predeterminado para este tipo de archivo (si es que no se encuentra abierto) y coloca como entrada del editor el archivo obtenido. Luego obtiene el número de línea que debe colorearse, activa el editor para que sea visible, obtiene su *SourceViewer* y colorea la línea. Luego finaliza y retorna el control al *CiaoLineTracker* para que pueda obtener la próxima línea a colorear.

El editor es obtenido utilizando el path del archivo en el cual se esta depurando (provisto por Ciao en la información de depuración) y el acceso estático provisto por la clase *PlatformUI* que permite manejar todos los elementos GUI del entorno.

5. Conclusiones

Hemos logrado obtener un plug-in para Eclipse utilizable tanto para edición de código Prolog como para lanzamiento y depuración fuente del mismo como puede verse en la figura 1. El plug-in será liberado con las mismas características del Ciao en cuanto se posea una versión lo suficientemente testeada.

En trabajos posteriores se incorporarán distintos elementos que permitan una mayor integración del entorno Eclipse con las herramientas provistas por Ciao.

Referencias

[1] Kent Beck and Erich Gamma. *Contributing to Eclipse*. 2003.

[2] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López, and G. Puebla. *The Ciao Prolog System - A Next Generation Multi-Paradigm Programming Environment*. Grupo Clip, http://www.ciaohome.org/, the ciao system documentation series edition, August 2004.

[3] IBM. *Eclipse- Platform Plug-In Developer Guide*. IBM, included documentation eclipse platform edition, 2003.

[4] Object Technology International (OTI). *Eclipse Platform Technical Overview*. OTI, http://www.eclipse.org/whitepapers/eclipse-overview.pdf edition, 2003.

[5] Proc. of the Workshop on Logic Programming. *Understanding the Control Flow of Compiled Prolog Clauses*, Debrecen, 1980.

[6] J. Szurszewsky. We have lift-off: The launching framework in eclipse. http://www.eclipse.org/articles, January 2003.

[7] J. A. Vanrell and C. Vaucheret. Implementando ayuda dinámica en la pltaforma eclipse. Anales del VIII Workshop de Investigadores en Ciencias de la Computación, Junio 2006.

[8] J. A. Vanrell and C. Vaucheret. Plug-in de eclipse paa un sistema prolog de código abierto. Anales del VIII Workshop de Investigadores en Ciencias de la Computación, Junio 2006.