**Programación Bajo Entornos Avanzados, Diciembre 2016**

**PROYECTO ECLIPSE**

**Resumen**

En este documento se pretende presentar una introducción a la plataforma de desarrollo integrado Eclipse.

La primera parte del trabajo está dedicada a establecer una idea general sobre el proyecto, sus orígenes y objetivos.

La segunda parte explica la arquitectura tanto de la plataforma en sí como de su estructura extensible mediante *plug-ins,* y una introducción a su desarrollo acompañada de un sencillo ejemplo.

**Tabla de Contenidos**

**1. Introducción a la Plataforma Eclipse** 1

**2. ¿Qué es Eclipse?** 1

**2.1. Orígenes del proyecto**  3

**2.2. Objetivos de la Plataforma Eclipse**  4

**2.3. Obtener e instalar el IDE de Eclipse**  4

**3. Arquitectura de la Plataforma Eclipse** 5

**3.1. Plataforma de Ejecución (*Platform Runtime*)** 5

**3.2. *Workspace*** 6

**3.3. *Workbench***  7

**3.3.1. *Standard Widget Toolkit* (SWT)**  8

**3.3.2. JFace** 8

**3.4. *Team Support***  9

**3.5. *Help*** 9

**3.6. *Debug*** 10

**4. Arquitectura basada en *plug-ins*** 11

**4.1. ¿Qué es un *plug-in*?** 11

**4.2. Arquitectura y *plug-in manifest*** 11

**4.2.1. Activación de los *plug-ins*** 13

**4.3. Obtener e instalar *plug-ins*** 13

**4.4. Ejemplo de creación** 14

**4.5. Algunos *plug-ins* interesantes** 16

**6. Conclusiones**  21

**ii Programación Bajo Entornos Avanzados**

**6.1. Ventajas del Proyecto Eclipse** 21

**6.2. Inconvenientes del Proyecto Eclipse** 23

**Programación Bajo Entornos Avanzados iii**

**1. Introducción a la Plataforma Eclipse**

En este documento se intentarán exponer las ventajas que ofrece el Proyecto Eclipse respecto a otros IDEs y cómo éstas influyen en la programación orientada a objetos.

En un primer vistazo se introducirá el Proyecto, sus orígenes y objetivos, para enmarcarlo como un importante IDE para el desarrollo de aplicaciones tanto libres como comerciales.

Más adelante se enfocará desde el punto de vista de la arquitectura, valorando su calidad como entorno de desarrollo y las ventajas que puede ofrecer tanto para la programación orientada a objetos como para la creación de todo tipo de aplicaciones, bien sean en equipo o desarrollos individuales. Un punto fuerte de la arquitectura de Eclipse es la extensibilidad mediante *plug-ins*, ofreciendo un amplio abanico de herramientas para el desarrollo de nuevas aplicaciones e incluso el de nuevas herramientas que pueden a su vez extender la funcionalidad de la Plataforma.

En última instancia se expondrá el *Standard Widget Toolkit* (SWT), otro de los puntos fuertes de Eclipse con respecto a las bibliotecas gráficas multiplataforma existentes para Java. Esta biblioteca ofrece dos características únicas: es nativa y portable a la vez. Además dota a las aplicaciones escritas con ella de una mayor velocidad en su ejecución y un menor consumo de recursos respecto a las aplicaciones escritas con Swing o AWT.

**2. ¿Qué es Eclipse?**

Como es definida en la página Web oficial [(www.eclipse.org):](http://www.eclipse.org) “*The Eclipse Platform is an IDE for everything and nothing in particular*” (la Plataforma Eclipse es un IDE para todo y nada en particular), una poderosa herramienta que permite integrar diferentes aplicaciones para construir entornos de desarrollo integrado (IDEs) que pueden ser utilizados para la construcción de aplicaciones Web, JavaTM, C/C++…entre otras, dando a los desarrolladores la libertad de elegir en un entorno multilenguaje y multiplataforma. Es un proyecto de desarrollo de software *open- source*, que está dividido en tres partes:

• *The Eclipse Project:* es un proyecto de desarrollo de software libre destinado a proporcionar una plataforma de desarrollo de herramientas integradas robusta, completa y comercial. Se subdivide, a su vez, en tres subproyectos:

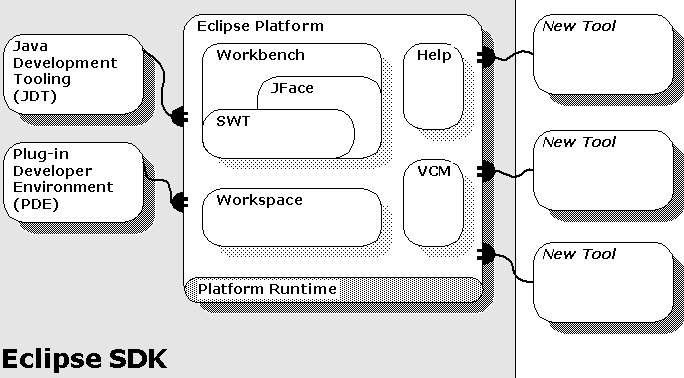
• la propia plataforma, que contiene las herramientas Eclipse.

• JDT (*Java Development Toolkit*): añade a la plataforma un IDE de Java completamente equipado, incluyendo: editor, refactoría (permite preservar la semántica de un programa), compilador y depurador.

**- 1 -**

**Proyecto Eclipse**

• PDE (*Plug-in Development Environment*): es un conjunto de herramientas diseñadas para ayudar al desarrollador de Eclipse en las tareas de desarrollo, prueba, depuración, construcción y distribución de *plug-ins*.



**Figura 1: *The Eclipse Project***

• *The Eclipse Tools Project:* su misión es fomentar la creación de una gran variedad de herramientas, proporcionando un punto de coordinación entre los desarrolladores para minimizar la duplicación y promover la interoperabilidad entre los diversos tipos de herramientas.

• *The Eclipse Technology Project:* su misión es proporcionar nuevos canales de comunicación a desarrolladores, profesores y educadores para participar en la evolución de Eclipse. Está organizado en tres proyectos relacionados:

• *Research:* investigación en dominios relacionados con Eclipse, tales como lenguajes de programación, herramientas y entornos de desarrollo.

• *Incubators:* son pequeños proyectos que añaden capacidades al software base de Eclipse.

• *Education:* estos proyectos se centran en el desarrollo de material de ayuda.

**Programación Bajo Entornos Avanzados 2**

**Instructor: Goicoechea, Lucas**

***2.1 Orígenes del proyecto***

Gran parte de la programación de Eclipse la desarrollaron OTI y equipos de IBM, responsables de las distintas versiones de VisualAge (construidas usando Smalltalk en un entorno de desarrollo llamado Envy), antes de la creación del proyecto como tal. Con la aparición de Java™ en la década de los 90, IBM creó una máquina virtual dual válida tanto para Smalltalk como para Java. La rápida expansión y sus ventajas respecto a Smalltalk obligaron a IBM a replantearse su máquina virtual dual y la construcción de una plataforma basada en Java desde el principio. El producto resultante fue Eclipse, cuyo coste había supuesto unos 40 millones de dólares a IBM en el año 2001.

En noviembre de 2001, IBM dona el código base de Eclipse y lo pone en manos de un consorcio (*Eclipse Consortium*) de empresas fabricantes de herramientas software. En un principio, la Junta Directiva del consorcio incluía a *Borland*, IBM, MERANT, QNX *Software Systems*, *Rational Software*, *Red Hat*, *SuSE*, *TogetherSoft* y *Webgain*. En marzo de 2002, la Junta Directiva aprobó la entrada de *Serena*, *Sybase* y *Fujitsu* a los que se unieron en junio del mismo año *Hitachi*, *Instantiations*, *MontaVista*, *Scapa Technologies* y *Telelogic*. En septiembre ETRI, HP, MKS y *SlickEdit* fueron elegidos unánimemente para formar parte de la Junta Directiva y se aprobó la entrada de *Oracle*. *Catalyst Systems*, *Flashline*, *Parasoft*, SAP, *teamstudio* y *TimeSys* se convirtieron en miembros del consorcio en diciembre de 2002. En marzo de 2003 el consorcio dio la bienvenida como miembro asociado al *Fraunhofer Institute*, y a representantes de *Ericsson*, *LogicLibrary*, M7 *Corpororation*, QA *Systems, SilverMark, Inc*., *Advanced Systems Concepts*, *Genuitec* y INNOOPRACT *Informationssysteme GmbH*. En mayo de 2003 se unieron como miembros *CanyonBlue*, *Ensemble Systems*, *Intel*, *Micro Focus*, *Tensilica* y *Wasabi*. El *Object Management Group* (OMG) también forma parte de la Junta Directiva como miembro asociado.

El 2 de febrero de 2004, la Junta Directiva anunció la reorganización de Eclipse en una corporación no lucrativa. Gracias a este cambio, se estableció una dirección permanente en el consorcio, lo que permitió establecer relaciones con desarrolladores comerciales y consumidores e instituciones académicas y de investigación, además de coordinar los diferentes proyectos *open-source*. Para supervisar y dirigir esta nueva organización, se estableció una Junta Directiva dividida en cuatro clases de miembros: *Strategic Developers*, *Strategic Consumers*, *Add-in Providers* y *Open Source Project Leaders*.

La última versión estable de la Plataforma Eclipse es la 3.0.2, lanzada en marzo de 2005, la cual añade mejoras en la flexibilidad, escalabilidad e interoperabilidad. Está disponible para las siguientes plataformas: Windows (XP/ME/NT/2000/98/Server 2003), Linux (x86/Motif), Linux

**3 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

(x86/GTK 2), Linux (AMD 64/GTK 2), Linux (IA 64/GTK 2), Solaris 8 (SPARC/Motif), Solaris 8 (SPARC/GTK 2), AIX (PPC/Motif), HP-UX (HP9000/Motif), HP-UX (IA 64/Motif), Mac OSX (Mac/Carbon).

***2.2 Objetivos de la Plataforma Eclipse***

La plataforma Eclipse está diseñada para afrontar las siguientes necesidades:

• Soportar la construcción de gran variedad de herramientas de desarrollo.

• Soportar herramientas proporcionadas por diferentes fabricantes de software independientes (ISVs)

• Soportar herramientas que permitan manipular diferentes contenidos (HTML, Java, C/C++, JSP, XML...).

• Facilitar una integración transparente entre todas las herramientas y tipos de contenidos sin tener en cuenta al proveedor.

• Proporcionar entornos de desarrollo gráficos (GUI) y no gráficos (non-GUI).

• Compatibilidad con una gran variedad de sistemas operativos, incluyendo Windows®

y Linux™.

• Fomentar el uso de Java para el desarrollo de nuevos *plug-ins*.

El principal objetivo de la Plataforma Eclipse es proporcionar mecanismos y reglas que permitan a los desarrolladores integrar de manera transparente sus herramientas. Estos mecanismos son proporcionados a través de APIs.

***2.3 Obtener e instalar el IDE de Eclipse***

El IDE de Eclipse se puede obtener directamente desde el sitio web oficial del **Proyecto Eclipse**

– [www.eclipse.org](http://www.eclipse.org) – o de cualquier “*mirror*” autorizado. Existen versiones para cualquier plataforma que soporte la biblioteca SWT, descargas que incluyen el código fuente y los *plug- ins* más habituales. Desde este sitio también se pueden obtener de forma independiente el SDK, JDT, PDE o SWT.

La descarga del SDK de Eclipse incluye: la Plataforma Eclipse, *Java Development Toolkit* (JDT) y *Plug-in Development Environment* (PDE), además del código fuente y la documentación para el programador y el usuario. Al estar escrito en Java™, Eclipse necesita

**Programación Bajo Entornos Avanzados 4**

**Instructor: Goicoechea, Lucas**

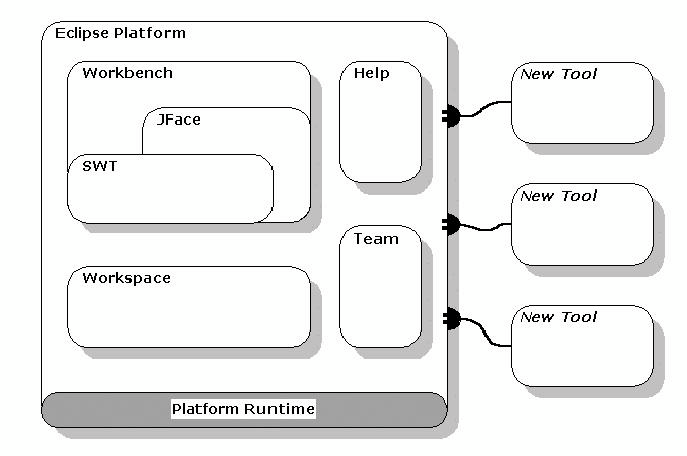
para su ejecución que exista una versión 1.4.1 o superior de *Java Runtime Environment* (JRE) o

*Java Development Kit* (JDK) previamente instalada en el sistema.

La instalación de la Plataforma Eclipse es tan sencilla como descomprimir el archivo descargado en el directorio deseado.

**3. Arquitectura de la Plataforma Eclipse**

Considerándola desde términos de diseño, la Plataforma Eclipse no ofrece gran funcionalidad por sí sola, sino que su valor real yace en el modelo de *plug-ins* (unidades mínimas de funcionalidad, explicadas en el apartado 3). Con lo cual, Eclipse está estructurada como un conjunto de subsistemas los cuales son implementados en uno o más *plug-ins* que corren sobre una pequeña plataforma de ejecución (**figura 2**). Dichos subsistemas definen puntos de extensión para permitir agregar funcionalidad a la plataforma. A continuación se describen los principales componentes de la plataforma.



**Figura 2: Arquitectura de la Plataforma Eclipse**

***3.1 Plataforma de Ejecución (Platform Runtime)***

Se trata del único componente de Eclipse que no es un *plug-in*. Al iniciar la Plataforma de Ejecución se descubren de manera dinámica el conjunto de *plug-ins* disponibles, se leen sus archivos de manifiesto, y se construye en memoria un registro de *plug-ins*, que está disponible a

**5 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

través de la API de la Plataforma. La plataforma mantiene el registro de aquellos *plug-ins* instalados así como de las funcionalidades que proveen. No podrán ser añadidos nuevos *plug- ins* después del inicio.

Para agregar nuevas funciones al sistema se usa un modelo de extensión común. Los “**puntos de extensión**” son lugares bien definidos dentro del sistema que permiten ser extendidos por *plug-ins*. Cuando una herramienta contribuye con una implementación para determinado punto de extensión se dice que agrega una “extensión” a la plataforma. A su vez, cada *plug-in* puede definir sus propios puntos de extensión de tal forma que puedan ser extendidos por otros. Este mecanismo de extensión es la única manera de agregar funcionalidad a la plataforma. Las extensiones están típicamente escritas en Java utilizando la API de la plataforma. Sin embargo, algunos puntos de extensión tienen asociadas extensiones provistas de ejecutables o incluso extensiones que han sido desarrolladas mediante lenguajes *script*. De forma general, solo un subconjunto del total de la funcionalidad de la plataforma se encuentra disponible para extensiones no realizadas con Java.

Un objetivo muy importante del *Runtime* es que usuarios finales no sufran desventajas a causa del uso de memoria por aquellos *plug-ins* que, si bien están instalados, no están siendo usados. De esta manera un *plug-in* puede ser instalado y agregado al registro pero el mismo no será activado a menos que se requiera mediante la actividad del usuario.

***3.2 Workspace***

Se trata del bloque central, o espacio de trabajo, para los archivos regulares que son específicos de cada usuario, y sobre los que actúan las diferentes herramientas instaladas en la Plataforma.

El espacio de trabajo del usuario consta de uno o más proyectos donde cada uno se mapea en un directorio especificado por el usuario en el sistema de archivos. Cada proyecto contiene los archivos que son creados y manipulados por el usuario. Todos los archivos en el espacio de trabajo son directamente accesibles por programas estándar y herramientas del sistema operativo.

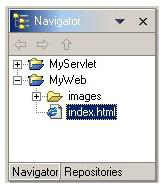
El conjunto de proyectos, archivos y carpetas que son generados por herramientas y almacenados en el sistema de archivos constituyen los **recursos**. Los recursos del *workspace* están organizados en una estructura de árbol, con los proyectos arriba y los archivos y carpetas por debajo. Existe un recurso especial, el *workspace root*, que funciona como raíz del árbol de recursos. Éste es creado internamente cuando el *workspace* es creado y existirá mientras éste

exista.

**Programación Bajo Entornos Avanzados 6**

**Instructor: Goicoechea, Lucas**

Los nombres de los recursos son cadenas de caracteres arbitrarias y cualquier cadena es admitida excepto *“.metadata”* ya que es usada internamente.



**Figura 3: Árbol de recursos del *Workspace***

La **figura 3** representa la típica jerarquía de recursos en un *workspace*. La raíz implícita del árbol es el *workspace root* y cada uno de los proyectos es inmediatamente hijo suyo. Los proyectos pueden contener archivos y carpetas, pero no otros proyectos. Loa proyectos y carpetas son funcionan como directorios de un sistema de archivos, es decir, si se borra un proyecto o una carpeta también serán borrados todos los recursos que contengan.

***3.3 Workbench***

Implementa el aspecto visual que permite al usuario navegar por los recursos y utilizar las herramientas integradas. El *workbench* es simplemente un *frame* donde se presentan varias partes visuales. Estas partes se pueden dividir en dos categorías mayores: editores y vistas.

Los **editores** permiten al usuario abrir, editar y guardar objetos. El *worbench* incluye un editor sencillo de texto. A partir de puntos de extensión para contribución de nuevos editores se pueden añadir en modo de *plug-ins* editores para Java, HTML...

Las **vistas** proporcionan información sobre aquellos objetos con los que el usuario está trabajando en el *workbench*. Con lo cual, las vistas cambiarán su contenido al cambiar el objeto que seleccione el usuario.

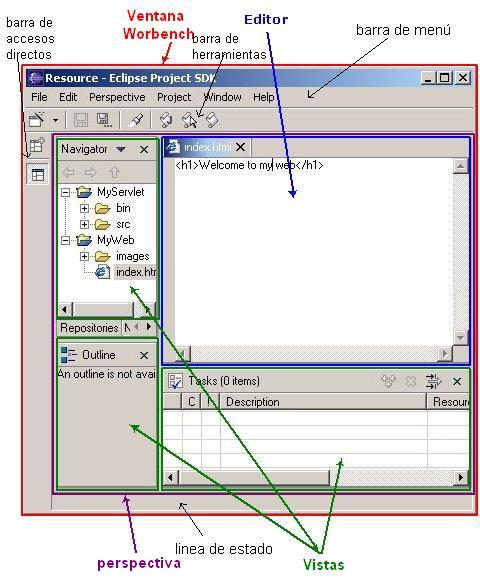
Hasta ahora se ha descrito el *workbench* como: “esa ventana que se abre cuando inicias la Plataforma”, pero en la **figura 4** se presenta una descripción un poco más concisa. A parte de las vistas y editores generales se encuentra el término **perspectiva**. En un *workbench* puede haber una o más perspectivas. Cada una consiste en una agrupación de vistas y editores que se

presentan juntos en pantalla para que, desde el punto de vista del usuario, sea una ventana de

**7 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

*workbench* sencilla. Si bien cada proyecto tiene distintas perspectivas solamente se podrá visualizar una a la vez.



**Figura 4: Estructura de la ventana del *Workbench***

***3.3.1 Standard Widget Toolkit* (SWT)**

El *Standard Widget Toolkit* (SWT) es una biblioteca de bajo nivel de componentes gráficos y utilidades. Esta biblioteca fue diseñada como alternativa a AWT y a Swing, añadiendo la ventaja de que se integra muy bien con el sistema operativo.

***3.3.2* JFace**

Interfaz gráfico implementado sobre SWT que simplifica las tareas de programación poniendo orden en el desarrollo con SWT. JFace es un “UI *toolkit”*, es decir, conjunto de herramientas para el desarrollo de interfaces (*widgets*) que, proporcionando un conjunto de clases, permiten

simplificar el desarrollo de características gráficas difíciles de implementar.

**Programación Bajo Entornos Avanzados 8**

**Instructor: Goicoechea, Lucas**

***3.4 Team Support***

El componente de equipo (*Team* o *Team support*) define un modelo de programación en equipo para crear y mantener un registro de las versiones de las aplicaciones que se desarrollen. Este componente permite que diferentes *plug-ins* de repositorios convivan dentro de la plataforma. Un **repositorio** es, en esencia, una base de datos que almacena código y material complementario. Cuando se usa un repositorio el código fuente con el que se trabaja se almacena en la base de datos, no en ficheros, y se extrae de él cuando se necesita. Este sistema de almacenamiento ofrece ventajas para los lenguajes orientados a objetos, pues resulta más fácil seguir la pista a las clases, con sus relaciones de herencia, en una base de datos.

Asimismo, el componente *team,* añade las vistas que el usuario necesite para interaccionar con cualquier sistema de control de versiones (si hay alguno) que se esté usando.

Eclipse incluye de forma estándar un *plug-in* **CVS** (*Concurrent Version System*), uno de los sistemas de control de código fuente más usados y muy útil para llevar el control de las versiones con las que se trabaja y conocer en todo momento sus diferencias, pero pueden añadirse repositorios como *ChangeMan* (*Serena*), *ClearCase* (*Rational*), *CM Synergy* (*Telelogic*), *PVCS* (*Merant*) y *StarTeam* (*Starbase*). Independientemente del sistema de control de versión que se use, la interfaz de usuario no cambia.

***3.5 Help***

El componente de ayuda (*Help*) permite a los *plug-ins* proporcionar documentación HTML que pueda ser presentada contextualmente por el *Workbench*. Las facilidades para configurar este contenido HTML en algo estructurado están expresadas en archivos XML externos. Esta separación permite que documentación HTML preexistente pueda ser incorporada directamente en los libros online sin necesidad de ser editada.

Define puntos de extensión que los *plug-ins* pueden utilizar para contribuir con ayudas u otro tipo de documentación.

Además la plataforma Eclipse nos proporciona una “Guía de usuarios del *Workbench*” y una “Guía de desarrollo de *plug-ins*”, y también una guía para cada uno de lo *plug-ins*. Aparte tanto JDT como PDE contribuyen con sus propias ayudas.

**9 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

***3.6 Debug***

El componente de depuración (*Debug*) proporciona un modelo genérico de depuración potente, sencillo y muy cómodo de utilizar, junto a una interfaz gráfica genérica de depuración. Cualquier *plug-in* puede aprovechar los mecanismos de depuración que proporciona este componente.

Cuando el depurador entra en acción, de forma automática, se abre la **Perspectiva Depuración***,* en la que se muestra toda la información relativa al programa que se está depurando. Cuando la Perspectiva *Debug* está activa, el menú contextual del Editor cambia para mostrar opciones de depuración, por ejemplo, ejecutar el programa hasta la línea que tiene el cursor, inspeccionar una variable (o una expresión seleccionada), etc.

Una característica de Eclipse como IDE es el soporte que ofrece Eclipse a *HotSwap* (**cambio en caliente**). Esta propiedad, incluida en la *Java Platform Debugger Architecture* (Arquitectura del Depurador de la Plataforma Java) de la versión 1.4, permite sustituir código o modificarlo mientras se está ejecutando un programa, sin necesidad de pararlo. Podemos establecer puntos de interrupción en el código, ejecutarlo, comprobar el estado de las variables en los puntos de interrupción cuando éstos se alcancen, modificar el código o introducir código nuevo, grabar los cambios y continuar la ejecución del programa. Los fallos pueden localizarse y analizarse *al vuelo*, sin vernos obligados a salir de la aplicación, cambiar el código, recompilar y comenzar una nueva sesión de depuración.

Para sacar partido de esta característica basta con iniciar una sesión de depuración, sustituir o cambiar el código generador de errores, grabar el archivo y apretar el botón ***Resume*** de la barra de herramientas ***Debug***. Automáticamente, el programa sometido a depuración proseguirá su ejecución, teniendo en cuenta los cambios efectuados. Utilizando *HotSwap* con Eclipse, se reducen los tiempos destinados a depuración, en particular cuando se trabaja con aplicaciones que tardan mucho tiempo en arrancar - como aplicaciones web de cierta complejidad o aplicaciones que accedan a bases de datos remotas- o en las que se tarda mucho en alcanzar los “puntos calientes” o problemáticos.

**Programación Bajo Entornos Avanzados 10**

**Instructor: Goicoechea, Lucas**

**4. Arquitectura basada en *plug-ins***

***4.1 ¿Qué es un plug-in?***

Un *plug-in* es la unidad mínima de funcionalidad de Eclipse que puede ser distribuida de manera separada. Herramientas pequeñas se escriben como un único *plug-in*, mientras que en las complejas la funcionalidad está en varios *plug-ins*. Excepto un pequeño núcleo de la plataforma Eclipse, el *Platform Runtime*, el resto de la funcionalidad de la plataforma Eclipse está implementada como *plug-ins*.

Los *plug-ins*, por lo general, están escritos en Java. Cada uno está formado por código Java, ficheros de lectura y otros recursos como imágenes, catálogos de mensajes, bibliotecas de código nativo, etc.

Existen algunos *plug-ins* que no contienen nada de código, por ejemplo: el *plug-in* que nos proporciona ayuda en forma de páginas HTML. Todos los recursos que componen el *plug-in* se encuentran en un directorio del sistema de archivos del sistema operativo, o en una URL de un servidor. Existe un mecanismo que permite que un *plug-in* pueda ser sintetizado a partir de distintos fragmentos, cada uno en su propio directorio o en su propia URL. Este mecanismo es utilizado para distribuir diferentes paquetes de idiomas para un *plug-in* que soporte diversos idiomas (**internacionalización**).

Para añadir *plug-ins* a la plataforma existe un **único modo**, los puntos de extensión. En conformidad con la orientación a objetos, un punto de extensión no deja de ser una interfaz que puede ser implementada por algún desarrollador dispuesto a extender la plataforma.

***4.2 Arquitectura y plug-in manifest***

Cada *plug-in* tiene un fichero denominado de manifiesto (*manifest*) en el cual se declaran sus interconexiones con otros *plug-ins*. La interconexión sigue un modelo muy simple: un *plug-in* declara un número de los denominados puntos de extensión, y un número de extensiones para uno o más puntos de extensión de otros *plug-ins*.

Al iniciar la Plataforma de Ejecución (*Platform Runtime*) se descubren de manera dinámica el conjunto de *plug-ins* disponibles, se leen sus archivos de manifiesto, y se construye en memoria un registro de *plug-ins*. La Plataforma enlaza cada extensión por el nombre con sus declaraciones de puntos de extensión. No pueden ser añadidos nuevos *plug-ins* después del

inicio.

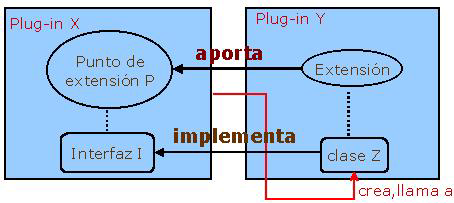
**11 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

Los archivos de manifiesto de los *plug-ins* contienen XML. Un punto de extensión puede declarar tipos de elementos XML adicionales para ser utilizados en las extensiones. Esto permite el paso de datos entre *plug-ins.* Además, la información del archivo de manifiesto está disponible desde el registro de *plug-ins* sin haber activado los *plug-ins* o haber cargado algo de su código. Esto es fundamental para soportar un gran número de *plug-ins*, a pesar de que sólo un número muy pequeño de ellos sean utilizados por el usuario, además de marcar una diferencia notable, en cuanto a consumo de recursos, entre los IDEs comerciales y Eclipse. Hasta que el código del *plug-in* no es cargado, el efecto que tiene sobre el entorno de ejecución es nulo.

Los *plug-ins* no sólo extienden o amplían la plataforma base, también pueden extender, a su vez, otros *plug-ins* que hayan definido sus propios puntos de extensión. Es decir, un *plug-in* puede hacer públicos interfaces que otros *plug-ins* pueden implementar. Las implementaciones de las interfaces (llamadas extensiones) mostradas por los puntos de extensión se realizan típicamente en Java, aunque algunos puntos de extensión pueden acomodar extensiones proporcionadas por ficheros ejecutables nativos o componentes ActiveX; incluso pueden programarse en lenguajes de *script*. El principal obstáculo con el cual se enfrentan las extensiones no realizadas en Java es la falta de acceso a la funcionalidad completa de la

plataforma Eclipse.



**Figura 5: Esquema de la extensión de un *plug-in* mediante otro *plug-in***

Como representa la **figura 5** la arquitectura de *plug-ins* de Eclipse viene marcada por los puntos de extensión y las aportaciones de los *plug-ins* a la plataforma así como entre ellos. En el ejemplo de la **figura 5**, el *plug-in* X declara el punto de extensión P y la interfaz I asociada a P. Mientras que el *plug-in* Y implementa la interfaz I con su clase Z y por tanto, aporta la clase Z al punto de extensión P. Con lo cual, el *plug-in* X será el que instancia la clase Z y el encargado

de llamar a sus I-métodos.

**Programación Bajo Entornos Avanzados 12**

**Instructor: Goicoechea, Lucas**

**4.2.1 Activación de los *plug-ins***

La Plataforma se ejecuta en una única invocación de la máquina virtual Java (JVM). A cada *plug-in* se le asigna su propio cargador de clases (Java *classloader*), usado para cargar las clases del *plug-in* así como los recursos del mismo.

Un *plug-in* es activado cuando su código realmente necesita ser ejecutado. Una vez activado, un *plug-in* utiliza el registro de *plug-ins* para descubrir y acceder a las extensiones que contribuyen a sus puntos de extensión. Por ejemplo, el *plug-in* que declara el punto de acceso de preferencias de usuario puede descubrir todas las preferencias de usuario contribuidoras y mostrarlas para construir un cuadro de diálogo con todas ellas. Esto puede ser realizado, sin necesidad de cargar el código de esos *plug-ins*, simplemente consultando el registro de *plug-ins*. El *plug-in* contribuidor será activado solo cuando el usuario seleccione la preferencia de la lista. Esto nos permite eliminar largos tiempos de arranque y es una solución de gran escalabilidad.

Es posible saber, qué *plug-ins* están activos en un momento determinado, para ello, es necesario lanzar la ayuda de Eclipse (desde el menú principal, seleccionado la entrada “*Help About Eclipse Platform*”). Desde la ventana que se despliega, se puede ver la lista de *plug-ins.*

Una vez activado, un *plug-in* permanece activo hasta que la Plataforma finaliza. Cada *plug- in* tiene su propio subdirectorio donde poder almacenar datos específicos sobre él; lo que permite mantener datos de sesión entre ejecuciones.

***4.3 Obtener e instalar plug-ins***

La descarga básica del entorno Eclipse incluye algunos de los *plug-ins* más esenciales, pero siempre es deseable obtener alguna funcionalidad extra. Para ello, es necesario instalar nuevos *plug-ins*. En el apartado *Community* de la web oficial de Eclipse [(www.eclipse.or](http://www.eclipse.org)g) se pueden encontrar enlaces a cientos de *plug-ins*. Para añadir un nuevo *plug-in*, basta con descomprimir el archivo descargado en el subdirectorio “*Plugins*” de la carpeta donde está instalado Eclipse. La próxima vez que se ejecute Eclipse, automáticamente, se reconocerán y añadirán los nuevos *plug-ins* instalados.

Existen dos formas de instalar los nuevos *plug-ins* en Eclipse. En la mayoría de los casos sólo hay que descomprimir el archivo descargado en el subdirectorio “*Plugins*” de la carpeta donde está instalado Eclipse. Pero existen casos donde la herramienta provee algún programa de instalación.

**13 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

Al comenzar Eclipse, el *Runtime* descubre cuáles son los *plug-ins* que se encuentran disponibles, lee su información desde sus propios archivos de manifiesto, y construye lo que se denomina *plug-in-registry* (registro de *plug-ins*).

***4.4 Ejemplo de creación***

Para construir un *plug-in* los pasos básicos a realizar son:

• Decidir cómo va a estar integrado el *plug-in* en la plataforma.

• Identificar a qué punto de extensión hay que contribuir para integrar el *plug-in* en la plataforma.

• Implementar la extensión de acuerdo con las especificaciones del punto de extensión.

• Proveer de un archivo *manifest* (*plugin.xml*) que describa la extensión con la que se está colaborando y el paquete con el código.

Como ejemplo se verá el desarrollo de un *plug-in* mínimo: “*Hello World*”. El objetivo es extender un punto de extensión del *workbench* con una vista que incluya la etiqueta *(label*) “*hello world*”. En el paquete *org.eclipse.ui* están incluidas las interfaces públicas que definen la interfaz de usuario del *workbench*. Algunas de esas interfaces tienen clases que las implementan por defecto y que pueden se extendidas para introducir cambios. La interfaz que se necesita para el ejemplo es *IViewPart*, siendo la clase que la implementa por defecto *ViewPart*. Aunque las vistas del *workbench* suelen cambiar su contenido dependiendo del elemento que el usuario señale, en este caso, por simplificar, la vista permanecerá estática.

package org.eclipse.examples.helloworld;

import org.eclipse.swt.widgets.Composite; import org.eclipse.swt.widgets.Label; import org.eclipse.swt.SWT;

import org.eclipse.ui.part.ViewPart;

public class HelloWorldView extends ViewPart { Label label;

public HelloWorldView() {

}

public void createPartControl(Composite parent) { label = new Label(parent, SWT.WRAP); label.setText("Hello World");

}

public void setFocus() {

// coloca el foco en el widget. Para una etiqueta no

**Programación Bajo Entornos Avanzados 14**

**Instructor: Goicoechea, Lucas**

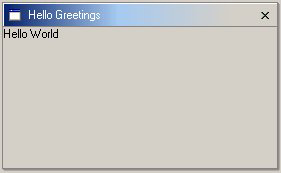
//tiene mucho sentido pero para estructuras más complejas es

//necesario elegir cual widget tiene el foco.

}

}

El método *createPartControl* crea una etiqueta (*label*) de la biblioteca SWT y escribe el texto “*Hello World*” en ella. En este momento se dispone del código del *plug-in* de cuya ejecución se obtendrá la **figura 6**.



**Figura 6: Vista producida por la ejecución del *plug-in “Hello World”***

En este momento se debe informar a la plataforma de que se va a contribuir con una vista. Esto se hará contribuyendo al punto de extensión *org.eclipse.ui.views*. Para registrar la extensión hay que incluir el archivo de manifiesto “***plugin.xml****”*, que describa el *plug-in*, incluyendo dónde está el código y cuál será la extensión.

<?xml version="1.0" ?>

<plugin

name="Hello World Example"

id="org.eclipse.examples.helloworld" version="1.0"

provider-name="Object Technology International, Inc.">

<requires>

<import plugin="org.eclipse.ui" />

</requires>

<runtime>

<library name="helloworld.jar" />

</runtime>

<extension point="org.eclipse.ui.views">

<category id="org.eclipse.examples.helloworld.hello" name="Hello" />

<view

id="org.eclipse.examples.helloworld.helloworldview" name="Hello Greetings" category="org.eclipse.examples.helloworld.hello"

class="org.eclipse.examples.helloworld.HelloWorldView" />

**15 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

</extension>

</plugin>

El *plug-in* es identificado con los parámetros *name*, *id*, *version* y *provider-name*. En este fichero también se debe incluir los *plug-ins* que son requeridos, que en este caso como se ha utilizado SWT, se ha incluido *org.eclipse.ui*. También se especifica dónde está localizado el código, en el ejemplo *helloworld.jar*. Finalmente, se indica a qué punto de extensión se está contribuyendo además de los parámetros configurados.

***4.5 Algunos plug-ins interesantes***

Para terminar con los *plug-ins*, se comentan algunos de los más populares de la plataforma

Eclipse.

**CDT** es el equivalente, para los lenguajes C y C++, al *plug-in* JDT. Las diferencias más importantes con el *plug-in* JDT están en la gestión de la documentación Javadoc (que es específica de la plataforma Java) y en el uso de bibliotecas JUnit (también específicas de Java). Otra diferencia importante es que CDT no puede compilar automáticamente el código, es necesario indicar un fichero *Makefile* para ello.

Hasta hace un tiempo, una de las mayores pegas que se le achacaba al IDE Eclipse era que no disponía de un editor gráfico de interfaces de usuario. Pues bien, El Proyecto Eclipse dispone de un *plug-in*, llamado  ***Visual Editor***, que permite, de forma sencilla y completamente visual, diseñar interfaces gráficos de usuario.

Los *plug-ins* disponibles para Eclipse no se tienen por qué limitar a la programación. Existen *plug-ins* que permiten integrar otras partes del proceso de desarrollo de aplicaciones como, por ejemplo, el diseño. ***Omondo*** es el *plug-in* UML permite unificar diseño e implementación en una sola herramienta. Cualquier actualización realizada sobre el modelo UML, se verá reflejada en el código fuente de todas las clases a las que dicha modificación afecte, y a la inversa, cualquier cambio en el código se verá reflejado en los diagramas de clases UML.

**5. Conclusiones**

En resumen, la Plataforma Eclipse proporciona un núcleo de componentes de construcción básico y APIs como el *workspace* y el *workbench,* y un conjunto de puntos de extensión a través de los cuales podremos integrar nuevas funcionalidades. A través de estos puntos de extensión, nuevas herramientas escritas como *plug-ins* pueden extender la funcionalidad de la Plataforma Eclipse. Al usuario se le muestra un IDE especializado con los *plug-ins* que tenga instalados.

***5.1 Ventajas del Proyecto Eclipse***

Eclipse se distribuye actualmente bajo la licencia **CPL** (*Common Public License*) versión 1.0 de

IBM, aprobada por la organización *Open Source Initiative* (OSI). A diferencia de otros proyectos *open source* (o, más exactamente, *free software*) que no permiten que se deriven de



**21 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

ellos trabajos propietarios o cerrados, Eclipse puede extenderse, por estar bajo esta licencia CPL, mediante la inclusión de *plug-ins* propietarios o ser usado como base para la creación de nuevas herramientas y, tras volverse a empaquetar y compilar el código resultante, el producto final puede venderse de forma comercial, manteniéndose público el código de Eclipse utilizado o modificado, pero sin la obligación de poner a disposición del público el nuevo código añadido.

Como es bien sabido, el software propietario o cerrado se caracteriza porque su redistribución y modificación está prohibida o requiere autorización previa; la mayor parte del software comercial es propietario, pero no cabe identificar ambos tipos de software: se pueden obtener beneficios económicos de Eclipse (al igual que de cualquier otro proyecto de código fuente abierto o de software libre).

En consecuencia, cualquier programa bajo licencia CPL puede compilarse (aunque no se haya efectuado ninguna modificación sobre el código original) y venderse el resultado de modo comercial sin requerir el pago de *royalties* u otras formas de compensación, de acuerdo con los términos de CPL; lo cual implica, aparte de otras obligaciones, poner a disposición del público el código fuente. Si una aplicación tiene una parte licenciada bajo CPL y el resto no (propietaria), la parte bajo CPL debe cumplir con esta licencia y, en consecuencia, el código de esa parte debe estar a disposición del público. El código fuente de la parte propietaria no tiene por qué licenciarse bajo CPL ni estar disponible al público.

Cualquier desarrollador puede modificar el código *open source* de Eclipse, redistribuirlo, comercializarlo crear trabajos derivados…, sin pagar *royalties* a IBM, pero no puede eliminar o modificar el *copyright* de IBM.

Otro de los aspectos que contribuye a la vertiginosa expansión de Eclipse es que poca gente (comparativamente hablando) tiene conocimientos de programación de sistemas operativos; sin embargo, cualquier desarrollador usuario de Eclipse, y hay muchos más desarrolladores que expertos en sistemas operativos, es un potencial colaborador del proyecto Eclipse.

Se podría pensar que Eclipse no deja de ser otra herramienta de desarrollo para Java, similar a herramientas como *JBuilder (Borland), JDeveloper (Oracle)* ó *NetBeans (Sun)*. Sin embargo, no es así: Eclipse presenta cuatro características conjuntas muy importantes que la diferencian:

• Eclipse se beneficia de la capacidad de aceptar *plug-ins open source* o propietarios, escritos por los propios desarrolladores Java, que pueden extender la plataforma y, a su vez, otros *plug-ins*.

**Programación Bajo Entornos Avanzados 22**

**Instructor: Goicoechea, Lucas**

• Eclipse cuenta con el respaldo de un consorcio de empresas muy importantes, ya detalladas en el primer apartado.

• Eclipse es neutral con respecto a la plataforma y el lenguaje (aunque en su mayor parte esté escrito en Java).

• Eclipse permite realizar íntegramente el proceso de desarrollo de software tal y como se entiende en la actualidad, desde el análisis inicial de requerimientos hasta la distribución final y el mantenimiento.

La primera característica no es del todo nueva, pues la plataforma *NetBeans* de Java (también una iniciativa *open source*) sigue una estrategia similar, pero no cuenta con el respaldo de empresas tan importantes como las citadas. En relación con la última característica, casi todas las herramientas de desarrollo en Java proporcionan algún tipo de asistencia para el modelado y el diseño, pero no de forma tan detallada y continua, de principio a fin, como el que puede proporcionar Eclipse mediante *plug-ins*.

***6.2 Inconvenientes del Proyecto Eclipse***

Pero en el Proyecto Eclipse no todo son ventajas, también encontramos algunos inconvenientes. Si bien Eclipse es multiplataforma, los *plug-ins* no tienen por qué serlo. Existen *plug-ins* que sólo corren en una plataforma, o que aún no han sido desarrollados para más de una. También hay que tener en cuenta que al ser una herramienta *open source*, se desarrollan *plug-ins* que no tienen todas las funcionalidades que tienen en otras herramientas comerciales, como ser IBM *Websphere*.

Otro punto a destacar entre los inconvenientes es que al estar desarrollado en Java, aunque facilite su uso en cualquier máquina, se convierte en un programa lento de ejecución y con un consumo excesivo de memoria RAM. Por otro lado, aunque la decisión de incluir los aspectos más básicos de un entorno es acertada, la instalación de demasiados complementos no es recomendable a los usuarios más noveles por su excesiva dificultad.

**7. Bibliografía**

• **Página Oficial del Proyecto Eclipse**: <http://www.eclipse.org>

• **Eclipse Platform Technical Overview**, Object Technology International, Inc., Febrero

2003: <http://www.eclipse.org/whitepapers/eclipse-overview.pdf>

• **Platform Plug-In Developer Guide**, IBM, 2000-2001:

<http://www.eclipse.org/documentation/html/plugins/org.eclipse.platform.doc.isv/>

• **Eclipse Project and Proposal**, <http://www.eclipse.org/projects/index.html>

**23 Programación Bajo Entornos Avanzados**

**Proyecto Eclipse**

• **BULMA: Eclipse, el framework para entornos de desarrollo**, Paco Ros, Enero 2003:

<http://bulma.net/pdf.phtml>?nIdNoticia=1645

• **Proyecto Eclipse**, Marzo 2004: <http://crl.iic.uam.es/descargas_web/cursos_verano/20040801/Jesus_Montero/Entornos_> de\_programacion.pdf

• **Mi primera hora con Eclipse**, Ángel Retamar:

<http://www.asturlinux.org/archivos/jornadas2004/ponencias/eclipse_ide.pdf>

• **El archipiélago Eclipse**, Miguel Ángel Abián, 2005:

<http://www.javahispano.org/articles.article.action>?id=75

• **Desarrollo de aplicaciones multiplataforma en Linux**, Franco M. Catrin L.:

<http://linux.ubiobio.cl/documentacion/franco_catrin/linux-multiplatform.pdf>

• **Plataforma Eclipse 2.1.3 Desarrollo de Plug-ins**, Proyecto Pasantía LEAD, Julio

2004: <http://www.fing.edu.uy/inco/grupos/coal/investigacion/proyectos/lead/docs/DesarrolloD> ePlugins.pdf

• **Eclipse o un entorno de desarrollo “para todo”**, Jorge López:

<http://www.iberprensa.com>

**Programación Bajo Entornos Avanzados 24**