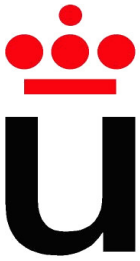


Universidad
Rey Juan Carlos

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Curso Académico 2009/2010

Proyecto de Fin de Carrera

Plugin de Optsicom para Eclipse

Autor/a: Rocío González García

Tutor: Francisco Gortázar Bellas

Resumen

Optsicom para Eclipse es una herramienta para facilitar el manejo de la librería Optsicom, utilizada en el desarrollo de procedimientos metaheurísticos sobre problemas de optimización y que ha sido desarrollada en el grupo de investigación Gavab de la Universidad Rey Juan Carlos.

Con este Plugin para el IDE de Java Eclipse se podrán definir Experimentos, gestionar las Instancias y Algoritmos asociados a los mismos, así como detectar Algoritmos desarrollados por el usuario y añadirlos al Experimento, es decir, todo el ciclo de definición y ejecución de experimentos que es una de las partes a la que se dedica más tiempo en el desarrollo de Metaheurísticas.

Índice

Resumen **3**

1. Introducción **9**

2. Objetivos **11**

3. Ejemplo de Uso **13**

3.1. Instalación 13

3.2. Creación, configuración y ejecución 15

4. Aspectos del Desarrollo **31**

4.1. Metodología de Desarrollo 31

4.2. Herramientas Utilizadas 33

4.3. Tecnologías Aplicadas 34

4.3.1. Java 34

4.3.2. Eclipse Plugins 34

4.3.3. SWT-JFACE 35

4.3.4. Eclipse Forms 35

4.3.5. XML 35

4.3.6. JAXB 36

4.3.7. JDT 36

4.3.8. Launching Framework 37

4.3.9. Preferences Page 37

5. Descripción Informática **39**

5.1. Especificación 39

5.2. Diseño 46

5.3. Implementación 54

5.3.1. Reunión de presentación del proyecto (8/10/09) . . . 55

5.3.2. 2ª Reunión (27/10/09) 55

5.3.3. 3ª Reunión (01/12/2009) 60

5.3.4. 4ª Reunión (02/02/2010) 66

5.3.5. 5ª Reunión (13/04/2010) 73

5.3.6. 6ª Reunión (29/04/2010) 77

5.3.7. 7ª Reunión (11/05/2010) 82

5.3.8. 8ª Reunión (20/05/2010) 85

5.3.9. 9ª Reunión (01/06/2010) 90

5.4. Estadísticas 92

5.4.1. *COCOMO* Básico 93

5.4.2. *COCOMO* Intermedio 94

6. Conclusiones y Trabajos Futuros **97**

Bibliografía **99**

Apéndices **103**

Bugs 103

Código Fuente 114

A mi padre,
que allá donde esté
sé que estará
orgulloso de mí.

Capítulo 1

Introducción

Una de las líneas de investigación del grupo Gavab, denominada Optsicom, es la resolución de problemas de optimización mediante algoritmos aproximados o metaheurísticas. Para ello utilizan la librería de Optsicom, desarrollada también en el grupo. Una vez diseñados e implementados, se realizan experimentos en los que los algoritmos son ejecutados y comparados con otros propuestos en la literatura. Dichos experimentos están formados por ejecuciones de estos algoritmos sobre instancias de datos.

La experimentación es una parte de la investigación en la que se emplea una importante cantidad de tiempo, por lo que una gestión adecuada de los experimentos permitiría ahorrar tiempo a los investigadores, así como facilitar el trabajo con la ingente cantidad de experimentos que pueden llegar a realizarse por problema.

Inicialmente la ejecución de estos experimentos se realizaba de forma local, lo que implicaba un largo tiempo de espera para que finalizara la ejecución. Actualmente utilizan un Cluster de ordenadores para realizar las ejecuciones, lo que disminuye considerablemente el tiempo de ejecución de los experimentos. El problema es que tanto la definición de los experimentos, donde se definen los algoritmos e instancias que se desean ejecutar, como el lanzamiento de la ejecución, se realizan de forma manual, lo que sigue

resultando tedioso y propenso a numerosos errores.

Debido a estos problemas, era necesaria la implementación de una aplicación intuitiva, visual, simple y de fácil aprendizaje, que simplificara el trabajo de la creación, configuración y ejecución de los experimentos metaheurísticos.

Capítulo 2

Objetivos

El propósito principal de este proyecto de fin de carrera es la creación del Plugin de Optsicom para el IDE de Java Eclipse con el que facilitar la tarea del desarrollador de Experimentos Optsicom, automatizando la gestión de estos experimentos metaheurísticos. De este modo se simplifica la edición y ejecución de experimentos.

Se decidió la creación de este Plugin porque es la mejor alternativa debido a la predilección en el uso del entorno Eclipse de desarrollo de Java por parte de los miembros del departamento. Además, con el propósito de aumentar sus prestaciones, así como investigar cómo está implementado el IDE y las múltiples posibilidades de ampliación que tiene.

Así pues, inicialmente el Plugin deberá permitir:

- El correcto uso de la librería de Optsicom para implementación de algoritmos (también llamados métodos).
- La configuración de los experimentos a ejecutar, pudiendo decidir:
    - Qué métodos se ejecutarán.
    - Sobre qué instancias de datos se ejecutarán los métodos.

  - Y la definición de los problemas asociados a los métodos e instancias seleccionados.

- La ejecución local o remota de los experimentos, al permitir la comunicación con el Plugin *Controlador del Cluster* del grupo Gavab que gestiona las tareas que se ejecutan en el Cluster y las máquinas que lo componen, y que está siendo desarrollado en otro PFC.

Capítulo 3

Ejemplo de Uso

Para exponer de una forma más gráfica el objetivo y utilidad de este Plugin, a continuación se recoge un ejemplo ilustrativo, con explicaciones sobre el uso del Plugin.

3.1. Instalación

Antes de comenzar, debe instalarse el Plugin en nuestro Eclipse. Para lo cual, vamos al menú *Help - Install New Software*, pulsamos sobre el botón *Add...*, introducimos la localización de **Sidelab** donde está alojado el plugin, `http://www.sidelab.es/files/optsicomeplugin/es.optsicom.eclipse.update-site`, y pulsamos *Ok*.

Se buscarán los plugins disponibles en dicha localización y nos aparecerá el Plugin de Optsicom. Pulsaremos el botón *Next* y aceptaremos los términos de licencia cuando sea oportuno.

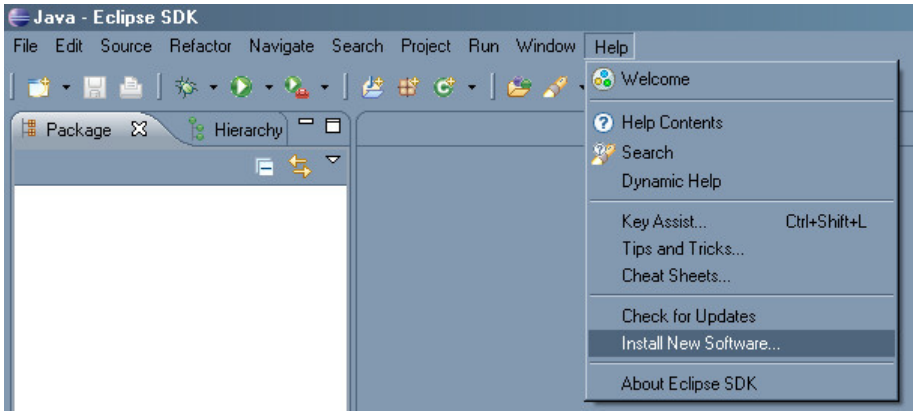


Figura 3.1: *Help - Install New Software...*

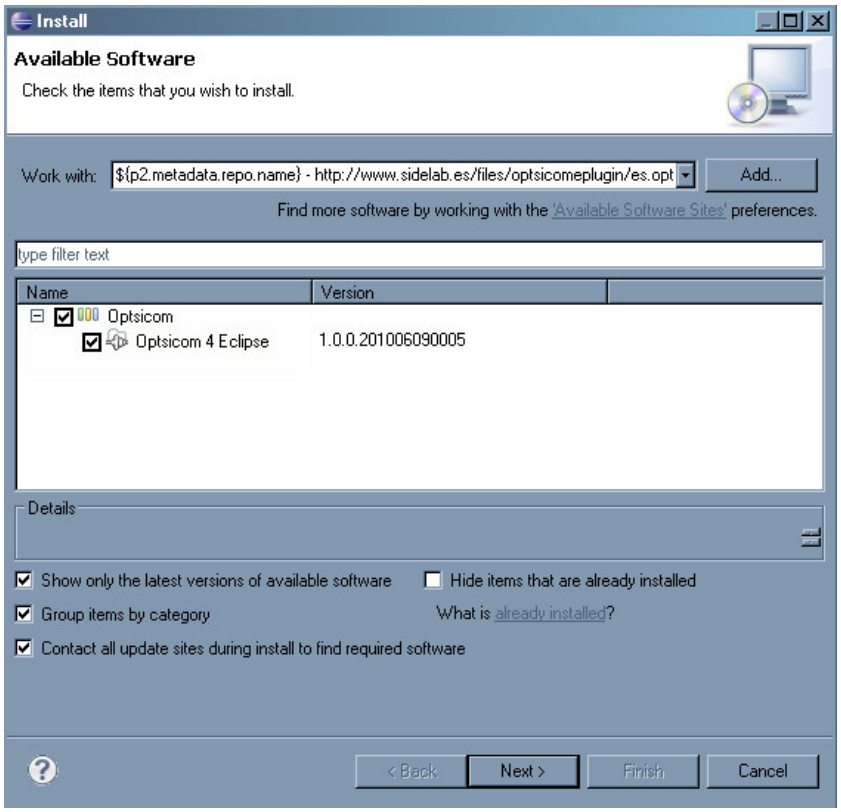


Figura 3.2: Búsqueda del plugin en el Site de Sidelab.

Cuando salgan nuevas versiones del Plugin, para instalarlas sólo será nece-

sario pulsar sobre *Help - Check for Updates* para que Eclipse las busque y proceder a instalarlas.

3.2. Creación, configuración y ejecución de Experimentos

Una vez instalado el Plugin, ya podemos proceder a su utilización para crear Experimentos Optsicom.

Sobre un Proyecto Java, creamos un nuevo Experimento con *File - New - Other...*

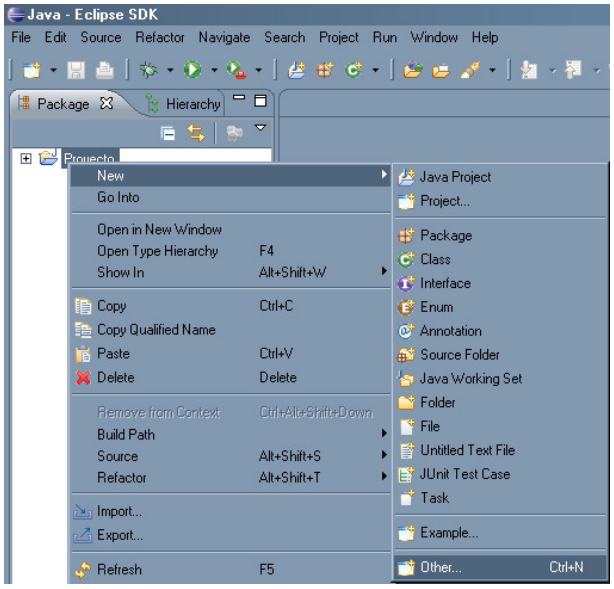
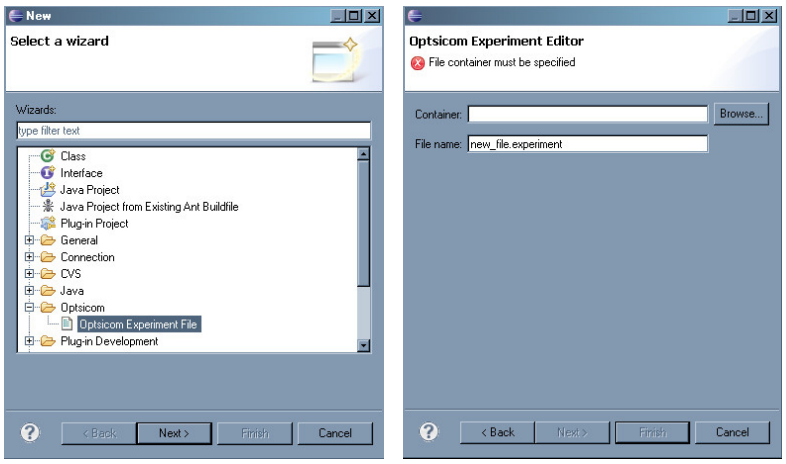


Figura 3.3: Crear un nuevo fichero.

Se abrirá entonces la ventana de selección de Wizard, donde tendremos que buscar el grupo **Optsicom**, dentro seleccionar el **Optsicom Experiment File** y pulsar *Next*.



(a) Selección del Wizard.

(b) Es necesario especificar el contenedor del fichero.

Figura 3.4: Paso 1 para la crear un nuevo Experimento Optsicom.

La siguiente ventana muestra los datos del fichero que queremos crear, de los cuales falta el Container, el paquete que contendrá al fichero. Debe especificarse ese contenedor pulsando el botón *Browse...*.

Tras seleccionar el contenedor donde queremos que se aloje el fichero, podremos comprobar y cambiar el nombre del fichero, aunque el Plugin ya le pone por defecto `new_file.experiment`.

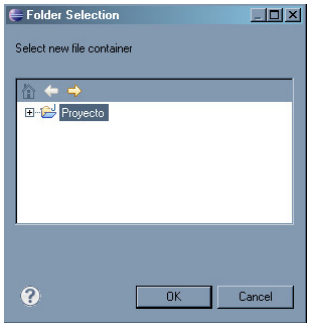


Figura 3.5: Seleccionar el contenedor del fichero.

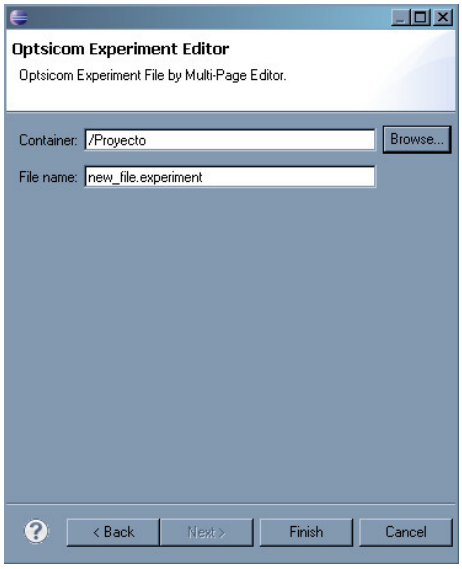


Figura 3.6: Datos del fichero a crear.

A continuación, se pulsa el botón *Finish* y se crearán en la carpeta del Proyecto seleccionado como contenedor los siguientes recursos:

- Fichero **.experiment**: es la representación del Experimento en XML.

- Carpeta **Experiments**: esta carpeta se utilizará por defecto para guardar los resultados de la ejecución del Experimento.

- Clase **MiExperimento.java**: es la clase main que se ejecuta cuando se quiere ejecutar el Experimento. (En realidad tendrá el nombre del fichero .experiment, pero acabado con `Experiment.java`, pero para simplificar su denominación en este ejemplo nos referiremos a esta clase como MiExperimento).

Además, se han creado las referencias a las librerías de Optsicom en el *Build Path* del proyecto contenedor.

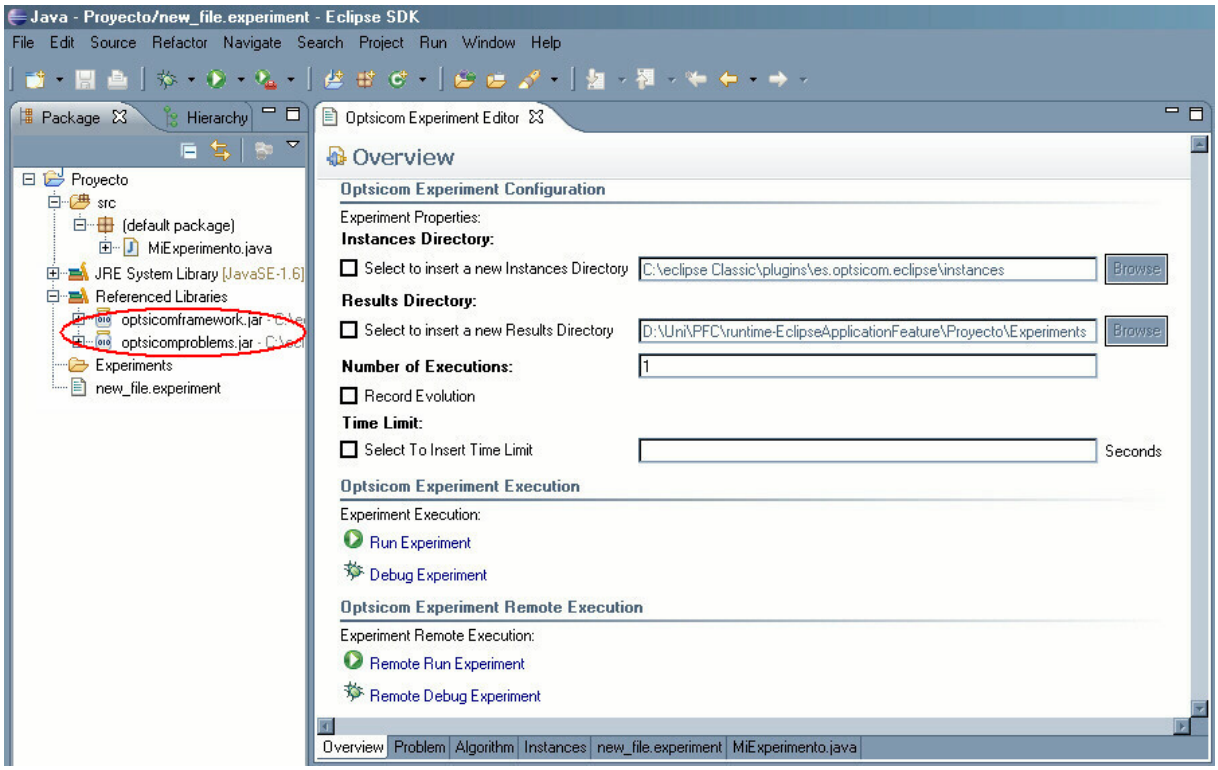


Figura 3.7: Recursos creados y vista de la página Overview del Plugin.

La primera pestaña visible del Plugin es la Overview, en la que se pueden configurar los siguientes parámetros de un Experimento:

- **Instances Directory**: Es el directorio en el que se alojan las instancias del Problema sobre el que se ejecutará el Experimento. Por defecto es el directorio de instancias que contiene el plugin, pero puede modificarse al pulsar sobre el CheckBox, lo que habilitará el botón *Browse* para buscar un nuevo directorio que contenga instancias. Al realizar este cambio ese directorio se cambiará de forma persistente para el Problema seleccionado. Si se vuelve a clicar sobre el CheckBox este directorio pasará a ser el de por defecto de nuevo.

- **Results Directory**: como se mencionó antes, por defecto es la carpeta *Experiments* creada dentro del proyecto contenedor. También

puede ser modificada, habilitando el botón *Browse* clicando sobre el CheckBox. Y como en el caso de la *Instances Directory*, si se vuelve a clicar el CheckBox se volverá al directorio por defecto.

- **Number of Executions**: el número de veces que se ejecutará cada algoritmo sobre cada instancia. Cuando los algoritmos están parcialmente aleatorizados, realizar múltiples ejecuciones permite conocer la robustez del mismo.

- **Record Evolution**: CheckBox que indica si se debe guardar la evolución de los algoritmos del Experimento: en lugar de guardar sólo el resultado final de la ejecución, se guardan también los resultados intermedios, para después poder ver la evolución temporal de los algoritmos.

- **Time Limit**: límite máximo de tiempo empleado en la ejecución, en segundos. En ocasiones el tiempo está limitado, o se quiere comprobar cómo se comporta un algoritmo con un determinado horizonte temporal.

También en la página Overview se encuentran los botones de ejecución *Run* y *Debug* para ejecución local y remota del Experimento.

La segunda pestaña es la de Problems, en la que se encuentra una lista de los Problemas contenidos en el Modelo, a la que se le pueden añadir nuevos problemas pulsando el botón *Add New Problem*. Como los Problemas siguen una jerarquía, estos se muestran en forma de árbol para su visualización. Cuando se pulsa sobre uno de los problemas, aparecen a la derecha sus características más importantes. Para crear el Experimento sobre uno de los Problemas, se debe seleccionar el deseado y pulsar sobre el botón *Load* para cargar sus Algoritmos e Instancias asociados y que éstos puedan ser seleccionados.

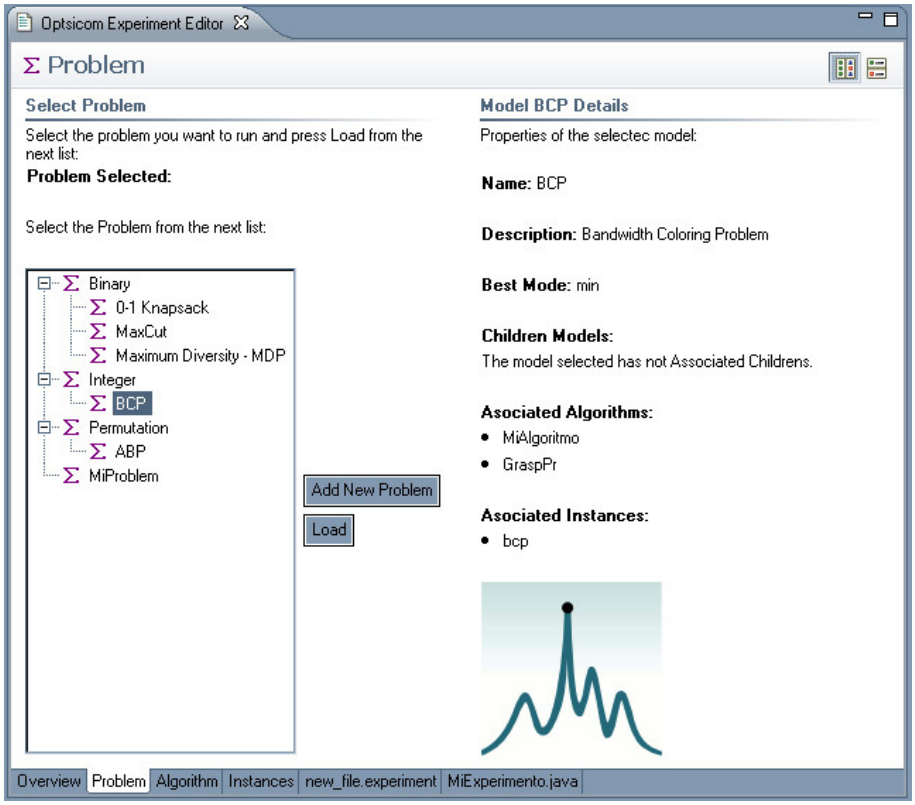


Figura 3.8: Vista de la pestaña Problem.

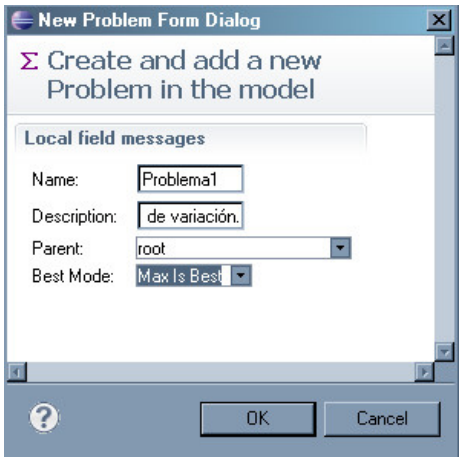


Figura 3.9: Crear un nuevo Problema en el Modelo.

A continuación tenemos la pestaña Algorithm, en la que, una vez haya sido *cargado* el Problema deseado, se pueden añadir Algoritmos al Experimento, para que sean instanciados y ejecutados, pulsando el botón *Add...*.

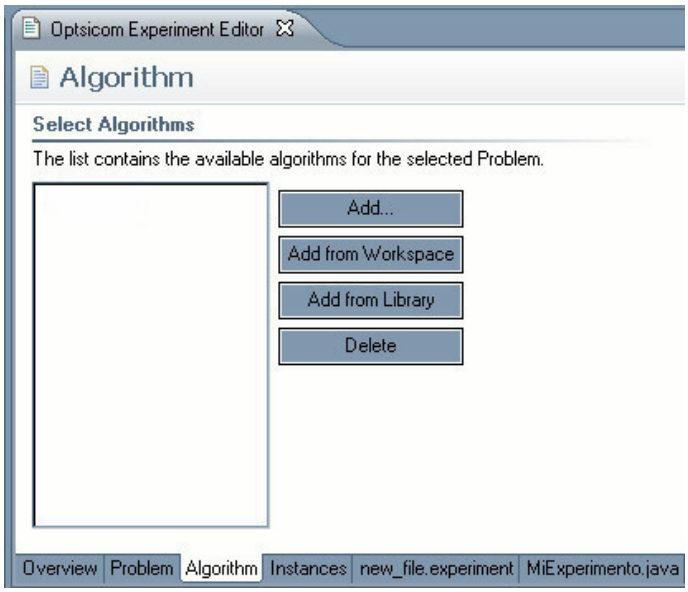


Figura 3.10: Vista de la página Algorithm.

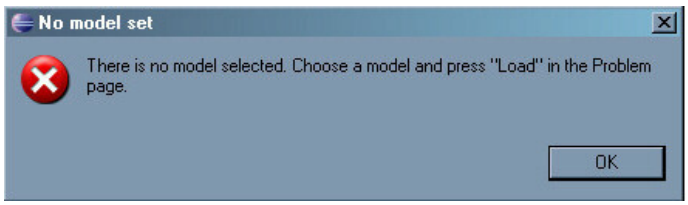


Figura 3.11: Si no se ha cargado el Problema previamente, al pulsar el botón Add... aparecerá este mensaje.

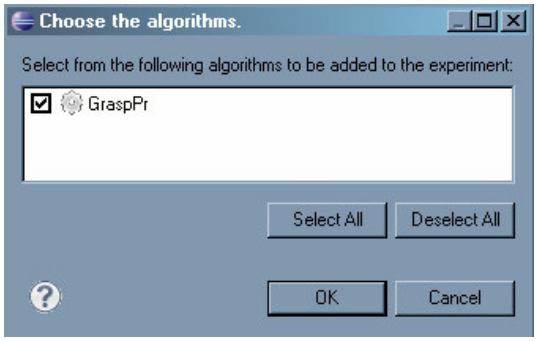


Figura 3.12: Añadir Algoritmos al Experimento con Add...

Después de seleccionar el algoritmo, éste es añadido a la lista de la página Algorithms y al pulsar sobre él se pueden ver sus características y configurar sus parámetros (si tuviera).

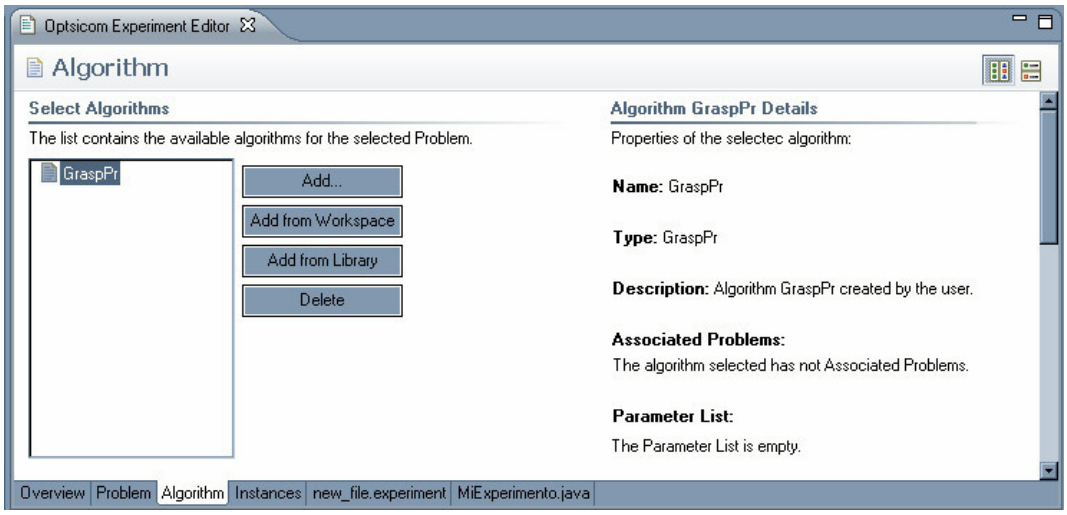


Figura 3.13: Vista de la página Algorithm, mostrando las características de un algoritmo de la lista.

El usuario puede crear nuevos algoritmos en su proyecto, creando clases que hereden de la clase ApproxMethod. Se debe tener en cuenta que los algoritmos deben tener un constructor sin parámetros, sino no se podrán añadir al Experimento.

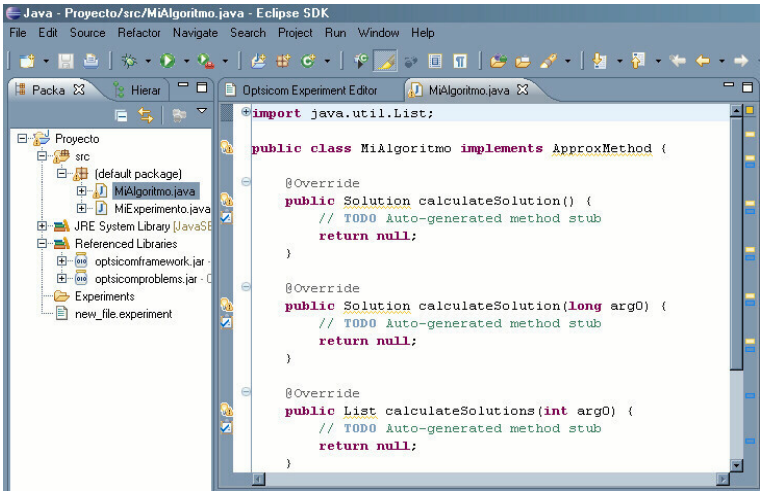
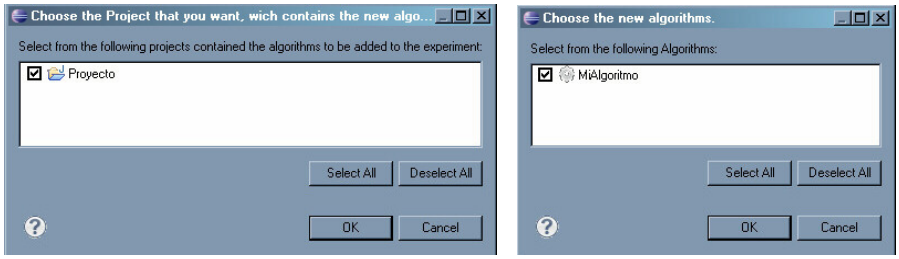


Figura 3.14: Crear un nuevo Algoritmo.

Pulsando sobre el botón *Add from Workspace* de la página Algorithm, se pueden buscar los nuevos algoritmos creados en los proyectos del *Workspace* para incluirlos en el Problema. Se abrirá una ventana en la que se mostrarán todos los proyectos que hay en el *Workspace* para seleccionar el deseado (o deseados).

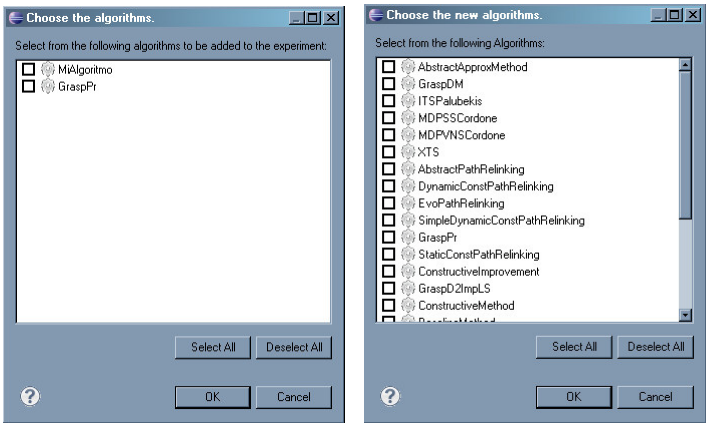


(a) Seleccionar el proyecto donde buscar nuevos algoritmos.

(b) Seleccionar los algoritmos encontrados.

El Plugin realizará una búsqueda sobre esos proyectos seleccionados, mostrando finalmente los algoritmos encontrados en ellos para que el usuario seleccione los que quiere añadir al Experimento. Además se añadirán al Problema de forma persistente, para que en el futuro el usuario pueda

añadirlos pulsando simplemente el botón *Add...*.



(c) Algoritmos para añadir al Experimento tras pulsar el botón *Add...*.

(d) Algoritmos encontrados tras pulsar *Add from Library*.

En la página Algorithm también tenemos un botón *Add from Library*, que buscará todos los algoritmos que haya en las librerías de Optsicom y en el *Workspace*, para añadirlos al Problema y al Experimento.

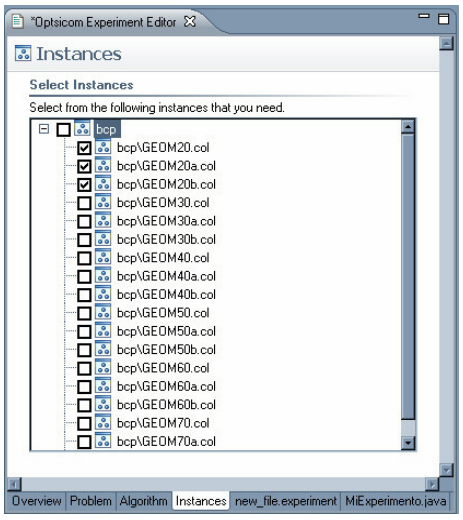


Figura 3.15: Vista de la página Instances.

La siguiente pestaña es la página Instances, en la que se muestran las instancias asociadas al Problema cargado, para que el usuario seleccione las que quiere que utilice el Experimento. Estas instancias se encuentran en el directorio que indique el campo *Instances Directory* de la página Overview.

Como se comentó anteriormente, este directorio puede ser modificado, cambiándose además las instancias asociadas al Problema cuando en la página Overview se introduce un nuevo directorio de instancias.

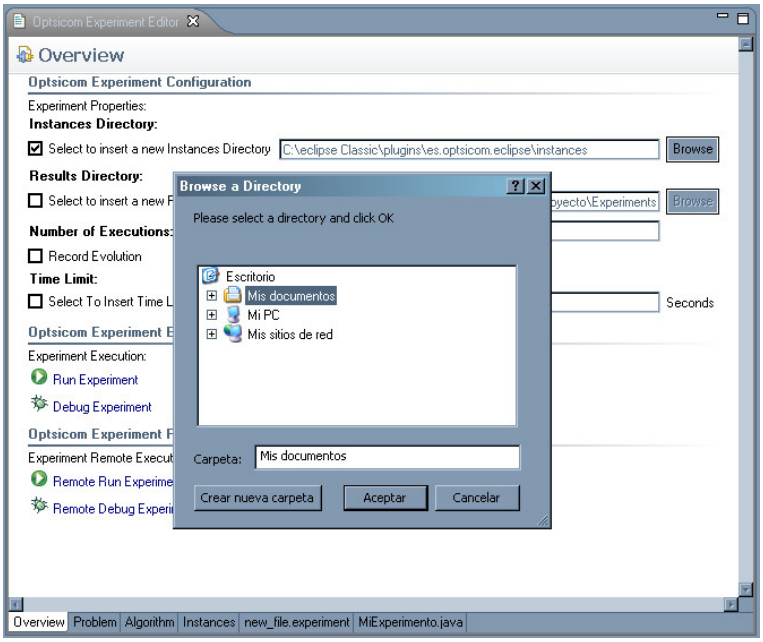


Figura 3.16: Al clicar sobre el CheckBox se habilita la búsqueda del directorio nuevo para la *Instances Directory*.

Figura 3.17: El nuevo directorio de instancias.

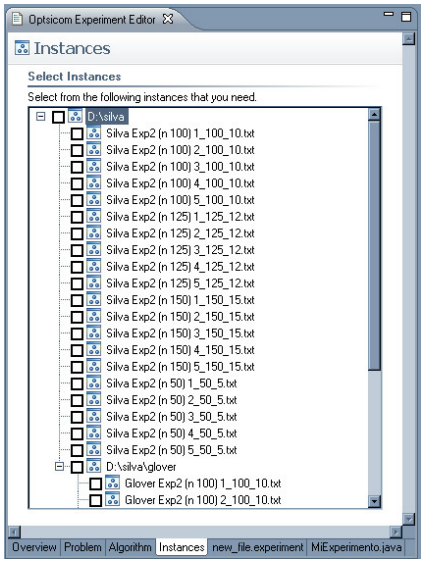


Figura 3.18: Las instancias han cambiado en la página Instances.

Seguidamente, se encuentra la pestaña que muestra el contenido del fichero .experiment en XML, mostrando con colores los distintos componentes del esquema XML.

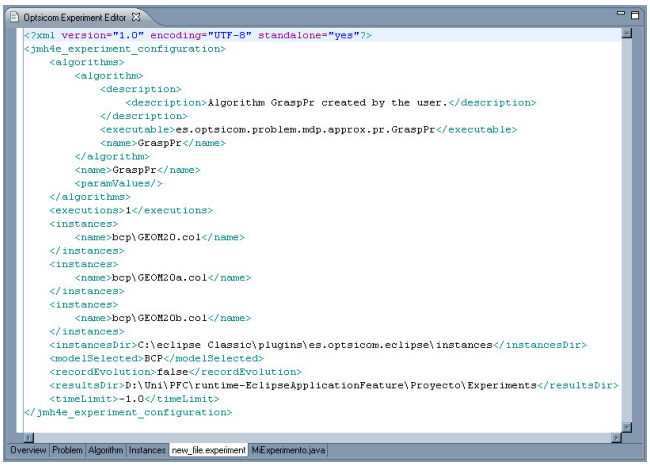


Figura 3.19: Vista XML del contenido del fichero .experiment.

Y por último, tenemos la pestaña que muestra el contenido de la clase Java main MiExperimento, que ejecutará el Experimento cuando se pulsen uno de los botones de *Run* o *Debug* de la página Overview.

En el método main de esta clase, que hereda de ApproxExperimentConf, se instancian los algoritmos e instancias seleccionados en las anteriores páginas, además se instancia un objeto de la clase para añadirle los algoritmos, instancias y atributos del Experimento, y se llama a su método run, que es el que realmente realiza la ejecución del Experimento.

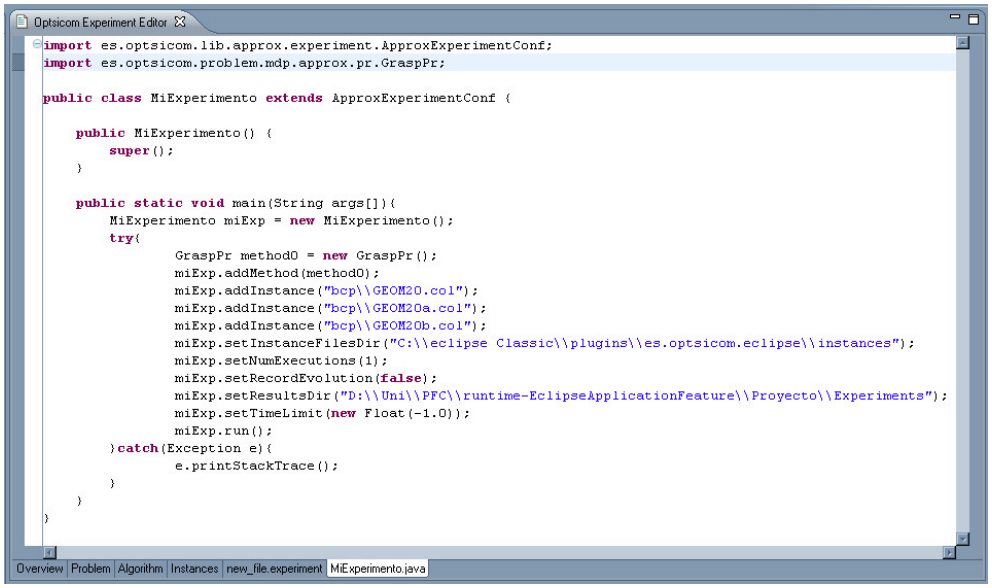


Figura 3.20: Vista de la pestaña que muestra el contenido de la clase Java MiExperimento.

Se debe tener en cuenta, cuando vamos a añadir algoritmos al Experimento, ya sean asociados al Problema, nuevos creados por el usuario o pertenecientes a la librería, que pueden tener *Parámetros* configurables. Éstos parámetros son detectados por el Plugin gracias a una *Anotación* de Java creada en la librería de Optsicom. Esta anotación, @Id, se debe colocar en los métodos get de los parámetros para que éstos sean detectados, como se puede ver en la siguiente imagen. Además, con la anotación se puede

determinar el valor por defecto del parámetro, así como sus valores mínimo y máximo (aunque ninguno de ellos es obligatorio).

Hay que considerar, que los parámetros también deben tener un método set asociado para poder ser asignado su valor en el algoritmo. Por tanto, si un parámetro no tiene los dos métodos get y set y su get no tiene la anotación @Id, este parámetros será ignorado por el Plugin.

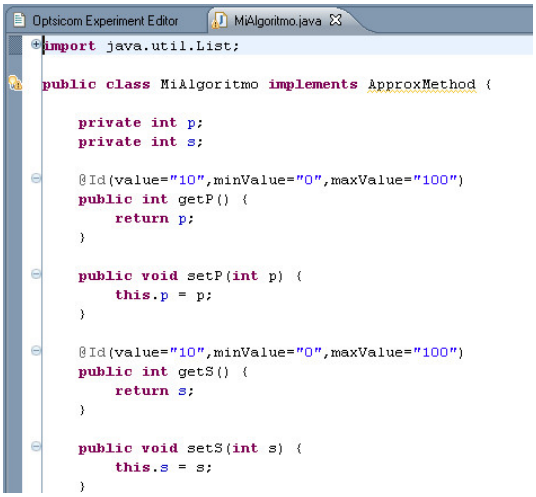


Figura 3.21: Clase MiAlgoritmo con dos parámetros: p y s.

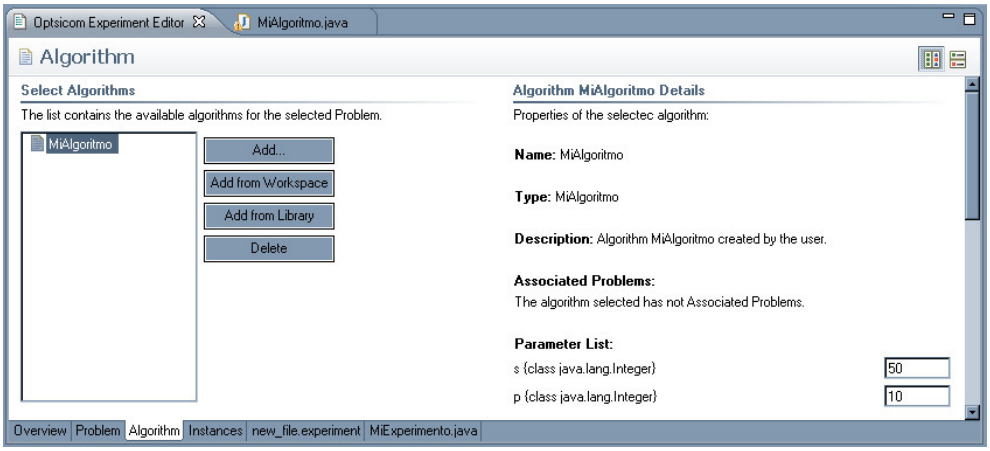


Figura 3.22: Vista de la página Algorithms mostrando los parámetros de un algoritmo.

Por lo tanto cuando se pulsa sobre el algoritmo en la página Algorithms, en la parte derecha podemos ver sus características y asignarle valores a sus parámetros.

Así pues, en la clase MiExperimento se puede apreciar cómo se asignan los valores a los parámetros del algoritmo después de su instanciación.

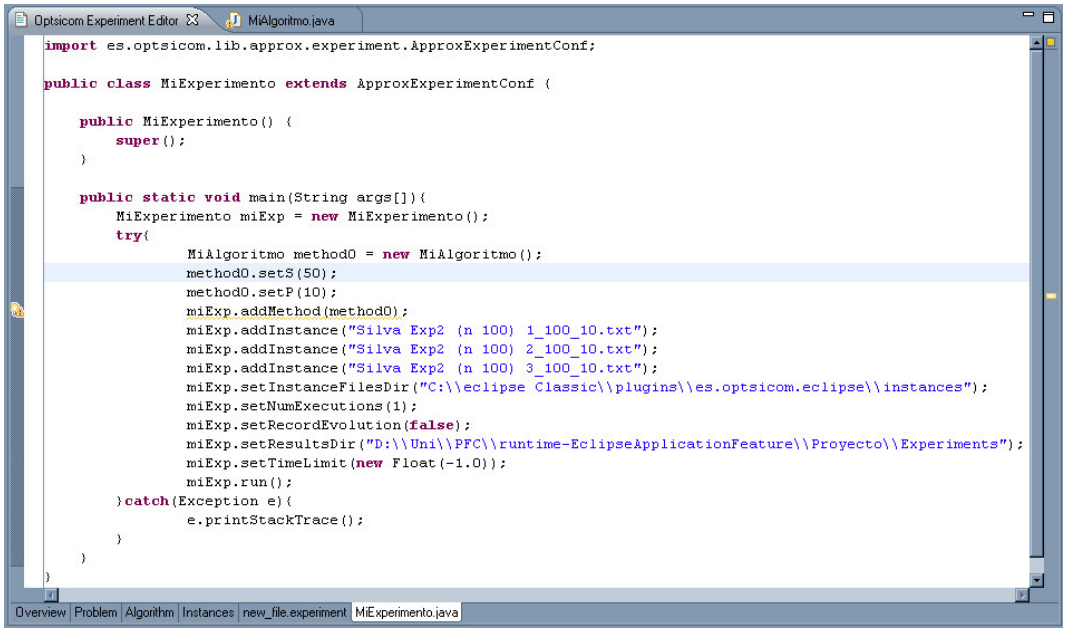


Figura 3.23: Asignación de los valores a los parámetros en la clase MiExperimento.

Finalmente, tras configurar el Experimento se puede proceder a la ejecución. En la página Overview tenemos los botones para crear un *Launch Configuration*, con el que ejecutar en modo *Run* o *Debug* el Experimento, de forma local o remota.

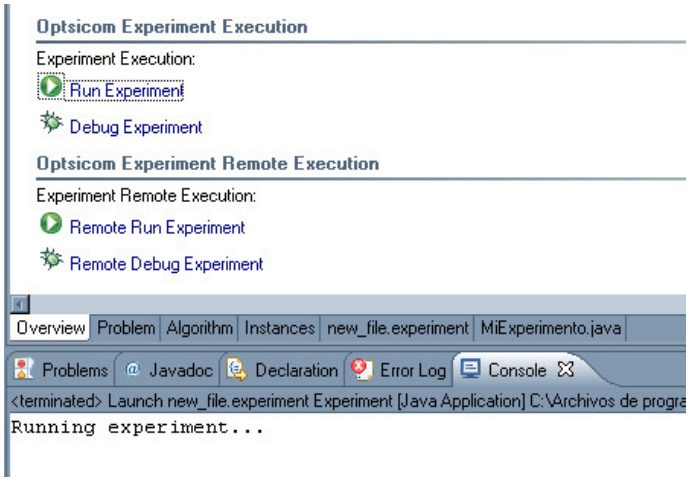


Figura 3.24: Ejecución del Experimento.

Después de la ejecución, si pulsamos sobre el menú *Run* - *Run Configurations*, podemos ver que se ha creado la configuración Launch de tipo Java del Experimento.

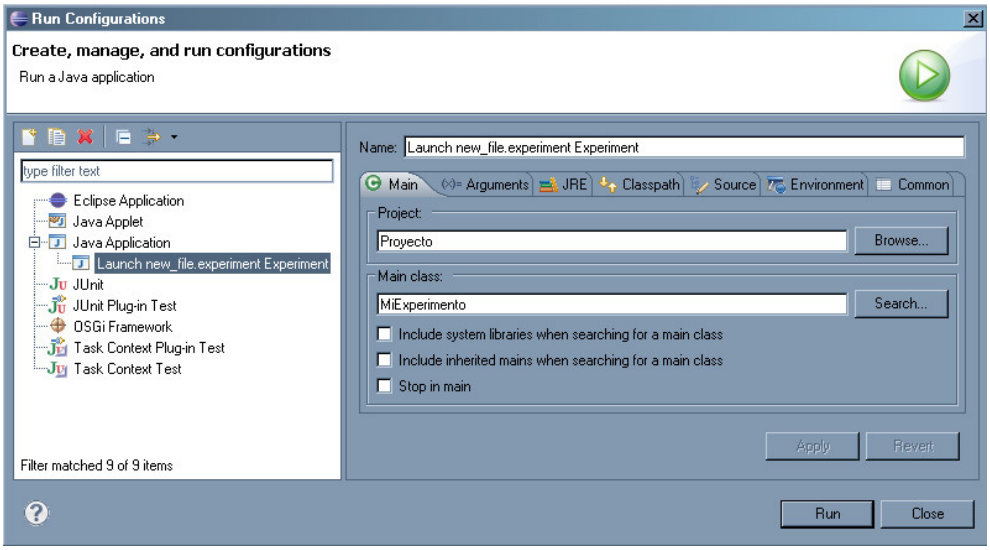


Figura 3.25: *Launch Configuration.*

Capítulo 4

Aspectos del Desarrollo

4.1. Metodología de Desarrollo

La Metodología de Desarrollo utilizada en el proyecto ha sido un *Ciclo de Vida en Espiral*, en el que en cada iteración del ciclo se han llevado a cabo las tareas necesarias para definir, diseñar, implementar y probar un prototipo del proyecto que ha ido evolucionando a lo largo de todo el desarrollo hasta su estado final.

Las tareas de cada iteración han sido las siguientes:

- **Obtención y Análisis de Requisitos**: se determinan los productos a obtener y se definen los pasos a seguir para añadirle al prototipo hecho hasta el momento (si procede) los nuevos requisitos que incrementarán su funcionalidad, a partir de lo ya obtenido en el ciclo anterior.

- **Diseño**: se define cómo se deben realizar los requisitos indicados en la anterior fase, lo que implicará búsqueda de documentación, ejemplos y manuales, con los que guiarse para llevar a cabo la implementación.

- **Implementación**: codificación en lenguaje Java.

- **Pruebas**: una vez codificado, se llevan a cabo los pertinentes casos de prueba para la detección y corrección de errores y fallos en el funcionamiento.

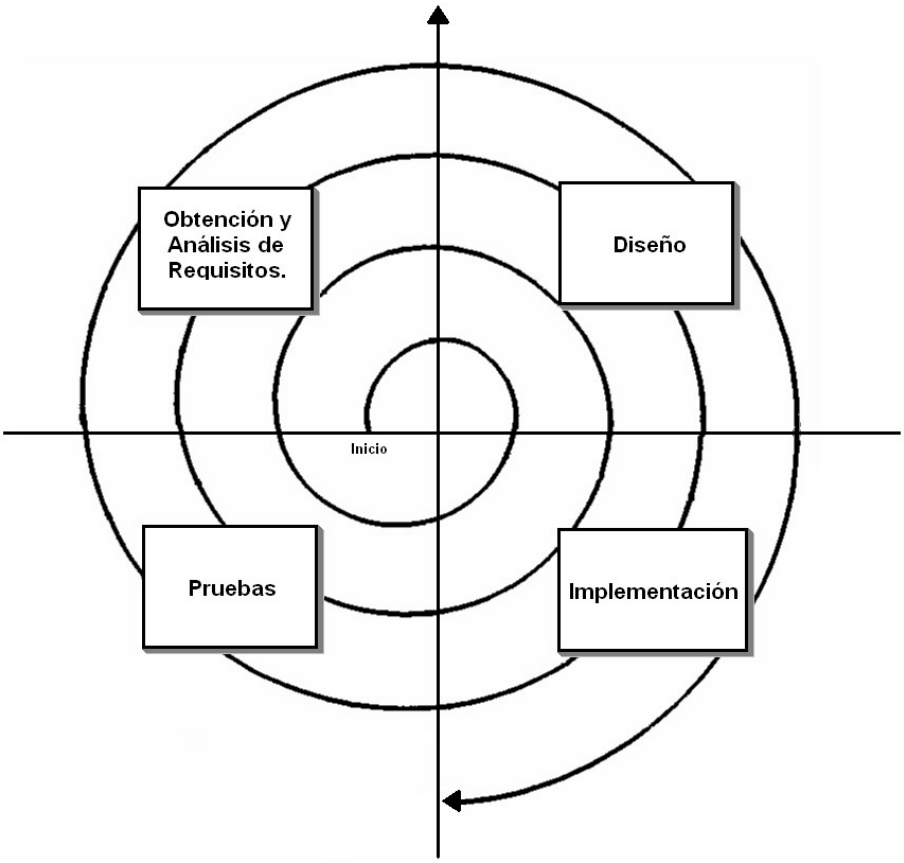


Figura 4.1: Ciclo de Vida en Espiral

De este modo se va obteniendo un prototipo funcional en cada fase final del ciclo, con el que realizar los casos de prueba y definir los siguientes requisitos a cumplir para la siguiente fase.

Se ha decidido utilizar esta metodología porque en un principio no se tenían claros todos los aspectos a desarrollar, así como las características relevantes que debía cumplir el proyecto, es decir, se tenía una idea global,

pero faltaba depurarla hasta el último detalle. Por ese motivo se inició el trabajo creando un prototipo inicial con muy baja funcionalidad, que fue incrementando a medida que surgían nuevas necesidades y aspectos a tener en cuenta.

Como apoyo a la metodología se ha utilizado el Sistema de Control de Versiones **Subversion** para mantener las diferentes versiones del código, una página Web **Wiki** para documentar los progresos realizados y las reuniones mantenidas con el cliente (tutor) y un **Bugzilla** para documentar los errores reportados tanto por el desarrollador (alumno), como por el cliente.

4.2. Herramientas Utilizadas

Para el desarrollo de plugins en Eclipse es necesario utilizar el plugin **PDE** (*Plugin Development Environment*). Además, es muy útil disponer del código fuente del propio Eclipse a modo de librería de consulta. Gracias al plugin **Plug-in Spy**, accesible mediante la combinación de teclas `Alt+Shift+F1`, se simplifica la tarea de búsqueda del código fuente en Eclipse.

El paquete más recomendable de Eclipse para el desarrollo de plugins es el denominado *Classic* que incluye el PDE y el código fuente, disponible en `http://www.eclipse.org/downloads/`.

Por último, hay que asegurarse de que está instalado el Plugin **Subversive** para la conexión con repositorios de Subversion, utilizado para mantener un control de versiones y *backups* del proyecto, además de facilitar la compartición del código.

Como se comentó anteriormente, para el seguimiento del desarrollo del proyecto se ha hecho uso del sistema **Wiki** del grupo Gavab (`http://www.gavab.es/wiki/Main/`), manteniéndose una página Web a modo de documentación detallada del desarrollo realizado (`http://www.`

gavab.es/wiki/jmh4e/), actualizándose durante todo el proceso, como se puede ver en el *histórico* de la página: http://www.gavab.es/wiki/jmh4e/#History.

Además, también se ha utilizado el repositorio de código **Subversion** (https://labs.gavab.es/svn/jmh4e/) y la herramienta de seguimiento de errores **BugZilla** (http://labs.gavab.es/cgi-bin/bugzilla3/index.cgi). Al final de este documento, en **Apéndices** se recogen todos aquellos *Bugs* abiertos en **BugZilla** durante el desarrollo del proyecto. Destacar que a fecha de fin del proyecto todos los *Bugs* introducidos han sido resueltos y cerrados.

4.3. Tecnologías Aplicadas

En esta sección se enumeran las herramientas y librerías utilizadas para la implementación del Plugin, además de incluir su descripción y usos.

4.3.1. Java

Java es un lenguaje de Programación Orientado a Objetos desarrollado por Sun Microsystems, que tiene como objetivo la codificación de programas software portables e independientes del sistemas operativos [1, 2].

Se ha elegido este lenguaje porque Eclipse en su totalidad está implementado en Java. La versión utilizada ha sido la 6.0 y está disponible en la página Web oficial de Sun Microsystems [9].

4.3.2. Eclipse Plugins

Eclipse es un potente y ampliable entorno de desarrollo de aplicaciones integrado de código abierto y multiplataforma, con entornos de desarrollo integrados, como el entorno para Java JDT (*Java Development Toolkit*).

Su arquitectura puede extenderse mediante el desarrollo e inclusión de Plugins, está desarrollado totalmente en Java y utiliza las librerías de SWT y JFace para su entorno gráfico. [3, 4, 5, 10, 11].

Además, Eclipse tiene un entorno adaptado al desarrollo de plugins y ejemplos prácticos. [12, 13, 14].

4.3.3. SWT-JFACE

Las librerías SWT y JFace son dos de las librerías para generar entornos gráficos de Java, que utilizan concretamente Eclipse y sus Plugins [6, 7].

En Eclipse los componentes visuales se llaman *widgets* y están implementados con los *widgets* de la librería SWT (*Standard Widget Toolkit*). JFace es la capa intermedia que gestiona la interfaz gráfica de usuario y que simplifica la construcción de aplicaciones basadas en SWT.

4.3.4. Eclipse Forms

Los **Forms** en Eclipse, son editores con formato de formulario Web, que incluyen, desde un punto de vista básico, campos de texto, etiquetas y botones. Estos componentes se extienden creando componentes complejos como *CheckBox*, *ComboBox*, cuadros de texto, *RadioButtons*, etc. [8, 15, 16].

Estos formularios se utilizan principalmente para definir el contenido, mostrarlo o configurar un recurso, que puede ser un fichero o un objeto Java. En este caso concreto se utiliza un FormEditor para editar un fichero, que es la representación XML de un objeto Java que representa el Experimento Metaheurístico.

4.3.5. XML

El *Extensible Markup Language*, XML, es un metalenguaje extensible de etiquetas desarrollado por el W3C (*World Wide Consortium*), que además de otros múltiples usos, permite la compatibilidad entre sistemas para compartir información de manera segura, fiable y fácil.

Concretamente para este proyecto, XML se utiliza como representación de objetos Java que deben tener una persistencia temporal, para su reutilización en posteriores ejecuciones.

4.3.6. JAXB

JAXB (*Java Architecture for XML Binding*) proporciona de una manera rápida la serialización a XML de objetos Java y su posterior recuperación. Dado un esquema, que especifica la estructura de los datos XML, el compilador JAXB genera un conjunto de clases de Java que contienen todo el código para analizar los documentos XML basados en el esquema. Una aplicación que utilice las clases generadas puede construir un árbol de objetos Java que representa un documento XML, manipular el contenido del árbol, y regenerar los documentos del árbol, todo ello en XML sin requerir que el desarrollador escriba código de análisis y de proceso complejo.

En el proyecto, se utiliza JAXB para crear el fichero XML que representa al objeto Java Experimento. [19, 20, 21].

Para la correcta serialización de los objetos a XML es necesario incluir en las clases Java anotaciones del paquete javax.xml.bind.annotation, de modo que JAXB pueda identificar los atributos del elemento, así como serializar correctamente aquellos que tengan conflictos. [22, 23].

4.3.7. JDT

JDT (*Java Development Tools*) es un plugin de Eclipse especializado en las características del *Workbench* de Java, de modo que representa en forma de árbol todos los componentes del *Workspace* de Eclipse como elementos del Modelo Java. De este modo podemos navegar por el árbol recorriendo el *Workspace*, los Proyectos que contiene, así como las librerías, paquetes, clases, atributos de clase y métodos que se puedan contener. Con el paquete JDT podemos crear y editar recursos, compilarlos, ejecutarlos y ver sus características.[27, 28, 29]

En el proyecto se hace uso de las clases de JDT, con las que se descubre la jerarquía de herencia de una clase, se añaden referencias a librerías y proyectos en el *Build Path* de un proyecto Java, se lanza la ejecución de una clase Java con método *main* y se crea un editor de tipo JavaEditor.

4.3.8. Launching Framework

Es una API de alto nivel proporcionada por el Plugin *Java Launching* que los desarrolladores pueden aprovechar para el lanzamiento de aplicaciones Java locales. De este modo se pueden crear, guardar y ejecutar *Launch Configurations* mediante código Java, indicándoles la clase *main*, los argumentos del programa y los argumentos de la *Máquina Virtual* de Java. [24, 25].

4.3.9. Preferences Page

Las preferencias de un plugin permiten configurar determinados aspectos del comportamiento del plugin [30], y son necesarios para que ese plugin pueda lanzarse. Se configuran mediante la ventana *Preferences*, accesible desde *Window - Preferences* en Eclipse.

Capítulo 5

Descripción Informática

5.1. Especificación

A continuación se detalla la Especificación de Requisitos del proyecto.

R.1: *Plugin de Optsicom para Eclipse.*

- **Tipo:** Funcional.
- **Descripción:** Crear un plugin para Eclipse con el que manejar los Experimentos Optsicom.
- **Entrada:** IDE Eclipse.
- **Salida:** IDE Eclipse con un nuevo plugin.

R.1.1: *FormEditor.*

- **Tipo:** Funcional.
- **Descripción:** La creación del Experimento se hará mediante un FormEditor, en el que se irán rellenando los campos necesarios para crear un Experimento.
- **Entrada:** Los atributos del Experimento.
- **Salida:** Un formulario tipo FormEditor para Eclipse que represente el Experimento.

R.1.1.1: *FormEditor dividido en Pestañas.*

- **Tipo:** Funcional.
- **Descripción:** Para facilitar la visualización de las tareas a completar para crear un Experimento, se dividirá el FormEditor en varias pestañas (Pages) significativas con las tareas de creación de Experimento, que serán: Overview, Problems, Algorithms e Instances. Además, se deben incluir dos pestañas más en las que se muestre el contenido de los dos ficheros generados por el Plugin: el fichero .experiment y la clase Java MiExperimento.java.
- **Entrada:** El FormEditor.
- **Salida:** El FormEditor dividido en pestañas.

R.1.1.2: *Página Overview.*

- **Tipo:** Funcional.
- **Descripción:** El FormEditor contendrá una pestaña Overview, similar a la de mismo nombre de los ficheros Manifest de los Plugin Projects, desde la que se configurarán los parámetros referentes a la ejecución del Experimento y se lanzará la ejecución en modo *Run* o *Debug* del Experimento Optsicom creado.
- **Entrada:** El FormEditor.
- **Salida:** El FormEditor con la pestaña Overview.

R.1.1.3: *Página Problems.*

- **Tipo:** Funcional.
- **Descripción:** El *Modelo de Problemas* que tiene Optsicom es una jerarquía, por lo que deben mostrarse como una estructura en forma de árbol en la página Problems.

- **Entrada:** Modelo de Problemas.
- **Salida:** Vista de los Problemas en forma de árbol en la página Problems.

R.1.1.4: *Página Algorithms.*

- **Tipo:** Funcional.
- **Descripción:** Los algoritmos que ejecutará el Experimento se seleccionarán en la página Algorithms, de entre los asociados al Problema seleccionado en la página Problems.
- **Entrada:** Lista de algoritmos asociadas a un problema.
- **Salida:** Acceso a la lista de algoritmos y vista de los seleccionados en la página Algorithms.

R.1.1.5: *Página Instances.*

- **Tipo:** Funcional.
- **Descripción:** Las instancias asociadas a un problema, pueden pertenecer a una jerarquía por lo que se deberán mostrar en forma de árbol en la página Instances.
- **Entrada:** Lista de instancias asociadas a un problema.
- **Salida:** Vista de las instancias en forma de árbol.

R.1.1.6: *XMLEditor.*

- **Tipo:** Funcional.
- **Descripción:** *Page* en la que se muestra el contenido XML del fichero .experiment.
- **Entrada:** El .experiment.
- **Salida:** La vista XMLEditor mostrando el contenido del fichero.

R.1.1.7: *JavaEditor.*

- **Tipo:** Funcional.
- **Descripción:** *Page* en la que se muestra el código Java de la clase MiExperimento en el FormEditor.
- **Entrada:** La clase MiExperimento.
- **Salida:** El JavaEditor mostrando el contenido de la clase.

R.1.2: *Serialización del Experimento a XML.*

- **Tipo:** Funcional.
- **Descripción:** El Experimento se serializará a código XML para que sea persistente.
- **Entrada:** Objeto Java Experimento.
- **Salida:** Representación del objeto serializado en XML.

R.1.3: *Fichero .experiment.*

- **Tipo:** Funcional.
- **Descripción:** El documento XML que representa el objeto Experimento serializado, se guardará como un fichero .experiment.
- **Entrada:** El documento XML.
- **Salida:** Fichero .experiment creado dentro del Proyecto donde se crea el Experimento.

R.1.4: *Clase main.*

- **Tipo:** Funcional.
- **Descripción:** El plugin debe crear una clase MiExperimento que represente al Experimento Optsicom y que tenga un método main para poder ejecutarla, en el que se creen los objetos de los Algoritmos e Instancias a ejecutar.
- **Entrada:** El Objeto Experimento.

- **Salida:** Clase MiExperimento.

R.1.5: *Librerías de Optsicom.*

- **Tipo:** Funcional.
- **Descripción:** Cuando se crea un Experimento dentro de un Proyecto Java en Eclipse, se deben incluir las librerías de Optsicom en el *Build Path* del Proyecto.
- **Entrada:** El Proyecto donde se crea el Experimento.
- **Salida:** El Proyecto con las librerías de Optsicom en el *Build Path*.

R.1.6: *Nuevos Algoritmos.*

- **Tipo:** Funcional.
- **Descripción:** El plugin debe permitir añadir nuevos algoritmos a los problemas, creados por el usuario en su proyecto. Estos nuevos algoritmos se buscarán como clases hijas de ApproxMethod con la clase JavaCore del paquete JDT.
- **Entrada:** El Modelo de Problemas.
- **Salida:** Nuevos algoritmos agregados al Modelo.

R.1.7: *Modelo de Problemas serializado.*

- **Tipo:** Funcional.
- **Descripción:** Serializar el Modelo de Problemas a un fichero XML con JAXB de modo que tengan persistencia los cambios introducidos en el Modelo.
- **Entrada:** El Modelo de Problemas.
- **Salida:** Fichero modelo.xml.

R.1.8: *Nuevos Problemas en el Modelo.*

- **Tipo:** Funcional.
- **Descripción:** Se pueden añadir nuevos Problemas al Modelo.

- **Entrada:** El Modelo.
- **Salida:** El Modelo con nuevos Problemas.

R.1.9: *Carpeta Experiments.*

- **Tipo:** Funcional.
- **Descripción:** Al crear el Experimento se debe crear una carpeta llamada *Experiments*, en la que por defecto se guardarán los resultados de las ejecuciones del Experimento. Esta carpeta *Experiments* puede ser cambiada en la página Overview.
- **Entrada:** El Proyecto donde se crea el Experimento.
- **Salida:** La carpeta *Experiments* dentro del Proyecto donde se crea el Experimento.

R.1.10: *Nuevas Instancias.*

- **Tipo:** Funcional.
- **Descripción:** En la pestaña Overview se mostrará el directorio *instances* por defecto, que contiene las instancias por defecto de los Problemas del Modelo. Este directorio se puede cambiar, cambiando las instancias por defecto por las nuevas.
- **Entrada:** El Modelo de Problemas.
- **Salida:** Instancias cambiadas en el Modelo de Problemas.

R.1.11: *Log.*

- **Tipo:** Funcional.
- **Descripción:** Todo error que se produzca en el Plugin en tiempo de ejecución debe mostrarse en la pestaña *Error Log* de Eclipse.
- **Entrada:** El Plugin.
- **Salida:** El Plugin con la salida de errores redireccionada al *Error Log*.

R.1.12: *Preferencias.*

- **Tipo:** Funcional.
- **Descripción:** Todo parámetro del que dependa la ejecución del Plugin se debe mostrar y configurar a través de la Preferences Page de Eclipse.
- **Entrada:** El Plugin.
- **Salida:** El Plugin Preferences Page.

R.2: *Update-Site.*

- **Tipo:** No Funcional.
- **Descripción:** El Plugin se debe instalar mediante un *Update Site.*
- **Entrada:** Eclipse.
- **Salida:** Eclipse con el Plugin instalado.

R.2.1: *Recursos del Plugin.*

- **Tipo:** No Funcional.
- **Descripción:** El Plugin, una vez que esté instalado, debe tener accesible los ficheros de las librerías de Optsicom, la carpeta *instances*, el fichero *modelo.xml* y la carpeta *icons.*
- **Entrada:** El Plugin.
- **Salida:** Cuando el Plugin sea instalado en la carpeta *plugins* de Eclipse, debe contener los recursos desempaquetados.

R.2.2: *Feature*

- **Tipo:** No Funcional.
- **Descripción:** Se creará una *Feature* que represente el Plugin para poder crear el Update-Site sobre ella.

- **Entrada:** El Plugin.
- **Salida:** La *Feature*.

R.2.3: *Update-Site*

- **Tipo:** No Funcional.
- **Descripción:** Sobre la *Feature* del Plugin, se crea un Update-Site, que permite la instalación del Plugin en Eclipse mediante *Help - Install New Software*.
- **Entrada:** La *Feature*.
- **Salida:** El *Update-Site*.

R.2.4: *Repositorio Público*

- **Tipo:** No Funcional.
- **Descripción:** El Update-Site se subirá a un repositorio público para que sea accesible de forma remota.
- **Entrada:** El *Update-Site*.
- **Salida:** El *Update-Site* subido a un repositorio público.

R.3: *WIKI.*

- **Tipo:** No Funcional.
- **Descripción:** Durante el desarrollo del proyecto se debe mantener una página Web alojada en la *Wiki* del grupo Gavab, a modo de documentación: `http://www.gavab.es/wiki/jmh4e/`.
- **Entrada:** El proceso de desarrollo.
- **Salida:** Una página Web.

5.2. Diseño

A partir de los Requisitos identificados en la fase de Especificación, se decide implementar el Plugin utilizando las dependencias siguientes:

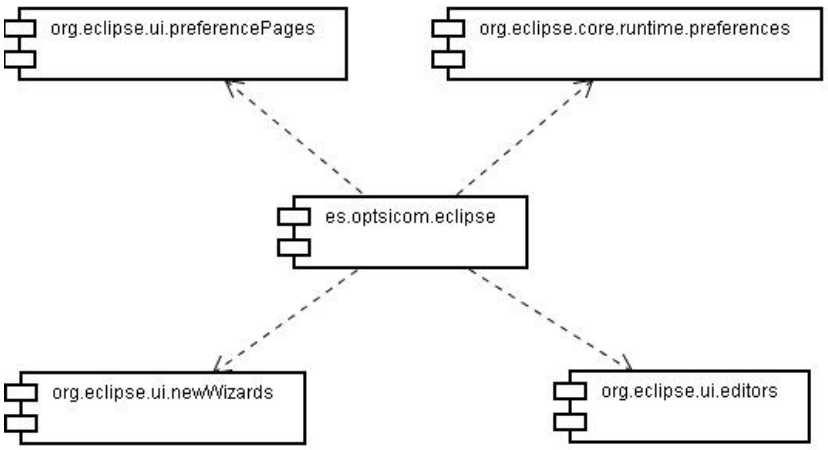


Figura 5.1: Diagrama de dependencias del Plugin con otros Plugins de la plataforma Eclipse.

Como muestra la figura, el Plugin de Optsicom hace uso de los plugin de Eclipse para:

- Crear una Preferences Page para Optsicom: que se mostrará en las Preferencias de Eclipse, accesible mediante el menú *Window - Preferences*.

- Gestionar las Preferencias en tiempo de ejecución: los valores que contienen las Preferencias, ya sean los valores de por defecto o hayan sido modificados por el usuario, serán guardados y gestionados en tiempo de ejecución por Eclipse.

- Crear nuevos *Wizards*: un *Wizard* es un asistente que ayuda al usuario a crear un recurso, que en nuestro caso es el Experimento Optsicom representado por un fichero XML de extensión .experiment, y que tiene asociado un formulario de tipo FormEditor.

- Utilizar editores para mostrar el contenido de ficheros: en este caso se incluyen tanto el FormEditor como el XMLEditor y el JavaEditor, que muestran el contenido de los ficheros .experiment y MiExperimento.java creados por el Plugin.

Más detalladamente, se pueden ver en la siguiente figura los paquetes del entorno org.eclipse que utiliza el Plugin en su implementación.

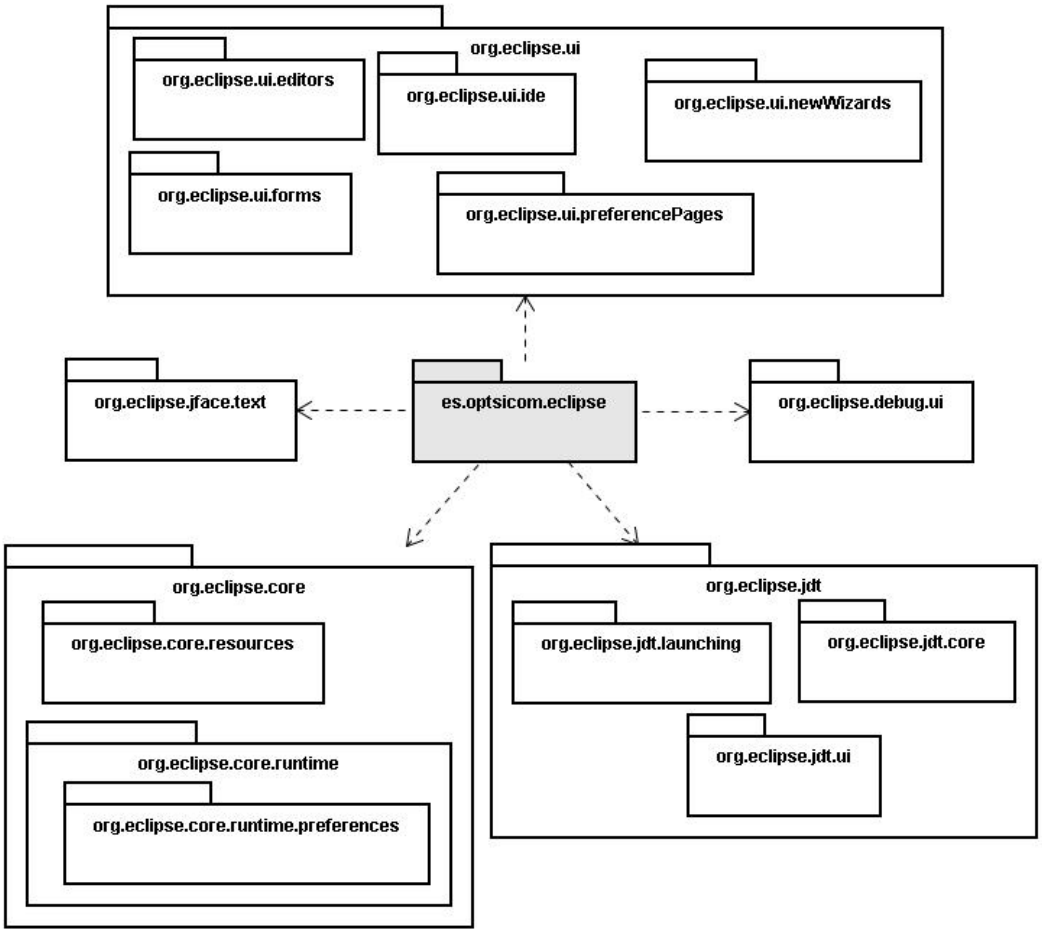


Figura 5.2: Diagrama detallado de dependencias del Plugin con las librerías de plugins de Eclipse.

A continuación se detalla el uso de cada una de las librerías utilizadas:

- **org.eclipse.core**: API de Eclipse.
  - **org.eclipse.core.resources**: se utilizan sus clases e interfaces para la gestión de los recursos contenidos en el Proyecto donde se crea un Experimento Optsicom. Se utilizará para crear ficheros,

directorios, gestionarlos, navegar entre ellos, cambiar su contenido, sus propiedades, etc.

- **org.eclipse.core.runtime**: principalmente se utilizará para crear un Status que se mostrará en el *Error Log* como información de errores, avisos (*warnings*) o mensajes procedentes del plugin. Además, se utilizará para crear Monitores de seguimiento sobre ficheros (en concreto el fichero .experiment), y devolver la ruta en tiempo de ejecución de recursos.
  - **org.eclipse.core.runtime.preferences**: inicialización y gestión de las preferencias del plugin.

- **org.eclipse.debug.ui**: contiene las clases pertenecientes al *Launch Framework*, con las que crear un Launch Configuration, guardarlo y ejecutarlo.

- **org.eclipse.jdt**: Librería *Java Development Toolkit*.
  - **org.eclipse.jdt.launching**: este paquete se utiliza la interfaz IJavaLaunchConfigurationConstants, con la que se obtienen los nombres de los atributos configurables de un Launch Configuration para un *Launch* de tipo *Java Application*.
  - **org.eclipse.jdt.core**: es el paquete que contiene la clase JavaCore y todas las demás interfaces con las que referenciar recursos Java. Con este paquete se obtiene la jerarquía de clases de una clase, el *Build Path* de un Proyecto Java, los métodos y atributos de una clase, sus anotaciones, etc.
  - **org.eclipse.jdt.ui**: contiene la clase JavaEditor de la que extenderá el JavaEditor que se utilizará para el Plugin.

- **org.eclipse.jface.text**: librería utilizada en el XMLEditor para la identificación de tokens XML, identificando palabras reservadas para su coloreado.

- **org.eclipse.ui**: Librería para el desarrollo de Interfaces de Usuario en Eclipse.
    - **org.eclipse.ui.editors**: librería base para crear Editors. En nuestro caso se utiliza para crear el XMLEditor en base a un TextEditor.
    - **org.eclipse.ui.forms**: librería para crear formularios de Eclipse. Es el paquete que contiene los *widgets* de SWT.
    - **org.eclipse.ui.ide**: de esta librería se utiliza la clase ResourceUtil, que es una clase para manipular recursos y determinar su correspondencia con los objetos del *Workbench*.
    - **org.eclipse.ui.preferencePages**: Página de configuración de las Preferencias del Plugin, accesible mediante *Window - Preferences*.
    - **org.eclipse.ui.newWizards**: Creación de nuevos recursos mediante el menú *File - New*.

A continuación se muestra la división en paquetes que se ha realizado del plugin desarrollado en este proyecto.

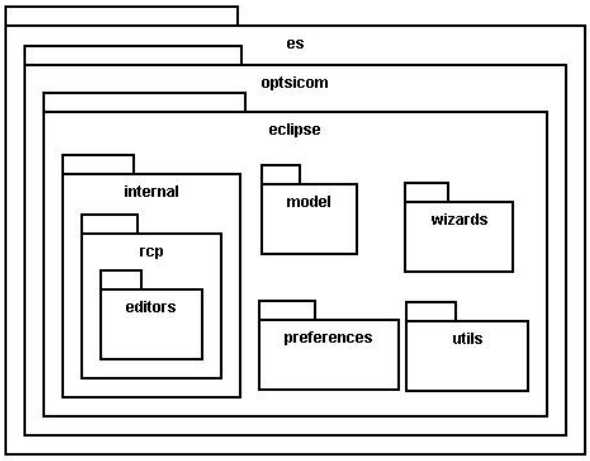


Figura 5.3: Diagrama de paquetes.

Para la implementación del Plugin se han codificado 47 clases y ha sido necesario dividirlas en paquetes funcionales para una mayor claridad a la hora de codificar.

En el paquete *model* tenemos las clases que implementan el modelo de datos que utilizará el plugin, estando representados el Experimento con la clase Jmh4eExperimentConfiguration, los Algoritmos con la clase Algorithm y las Instancias con Jmh4eInstance. El resto de clases se utiliza para dar formato a los demás elementos, por ejemplo para definir los Parámetros de los Algoritmos se utiliza Parameter, toda Descripción es un objeto de la clase Description y las Instancias se agrupan en InstanceGroup.

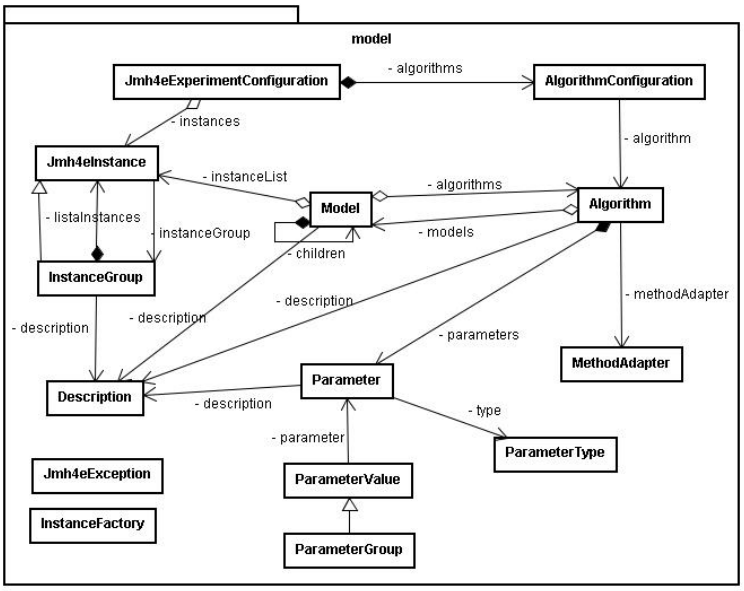


Figura 5.4: Diagrama de clases del paquete *model*.

En el paquete *utils* se encuentran las clases Hierarchy, que se apoya en la API de JDT para gestionar los recursos que se encuentran en el *Workspace* de trabajo del proyecto, y la clase Marshalling, que haciendo uso de JAXB serializa objetos Java a XML y la operación inversa.

Con las clases del paquete *wizards* se añade el nuevo *Wizard* asociado al experimento a la lista de *Wizards* de Eclipse y su página de configuración,

con los que se creará el fichero .experiments, el directorio *Experiments* y la clase main con la que se ejecutará el Experimento.

Y con el paquete *preferences* se crea la página de Preferencias de Optsicom en el *Preference Dialog* de Eclipse, y se establecen los valores iniciales de las preferencias del Plugin.

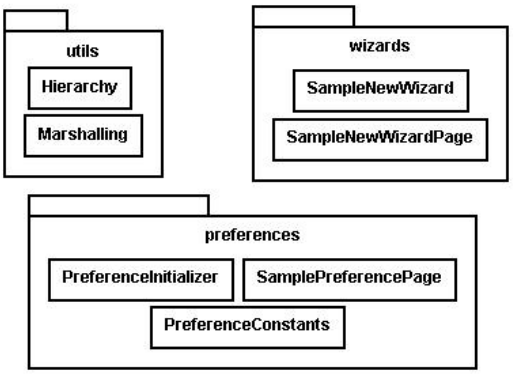


Figura 5.5: Diagrama de clases de los paquetes *utils*, *wizards* y *preferences*.

En el paquete *editors*, que se encuentra dentro del paquete *internal* se ubican las clases JavaEditor y XMLEditor, necesarias para implementar los Editores. XMLEditor hace uso del resto de clases del paquete para implementar la coloración de los elementos del esquema de un documento XML.

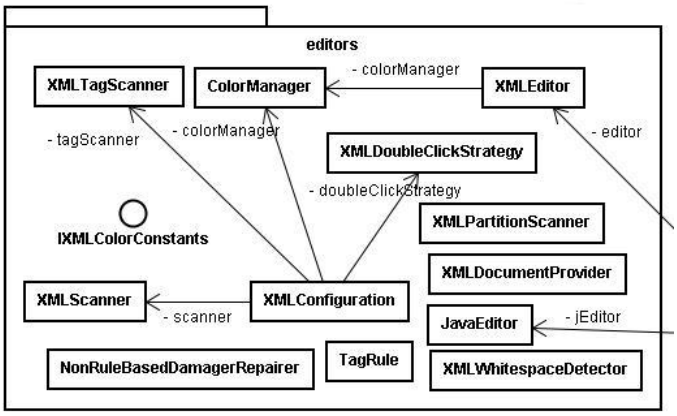


Figura 5.6: Diagrama de clases del paquete *editors*.

Además, en el paquete *internal* tenemos la clase Activator, que es la clase de arranque del Plugin, la clase FormEditorInput y el paquete *rcp* donde se encuentran las clases necesarias para implementar el Form y las distintas pestañas que lo componen, con los que se crearán los Experimentos.

La clase SimpleFormEditor es el *MultiForm* de este Plugin, con el que se configurará el Experimento. Está compuesto por tres pestañas de tipo Page y dos editores que muestran el contenido de los dos ficheros generados por el Plugin. El Wizard crea los recursos, pero es el SimpleFormEditor quien establece su contenido.

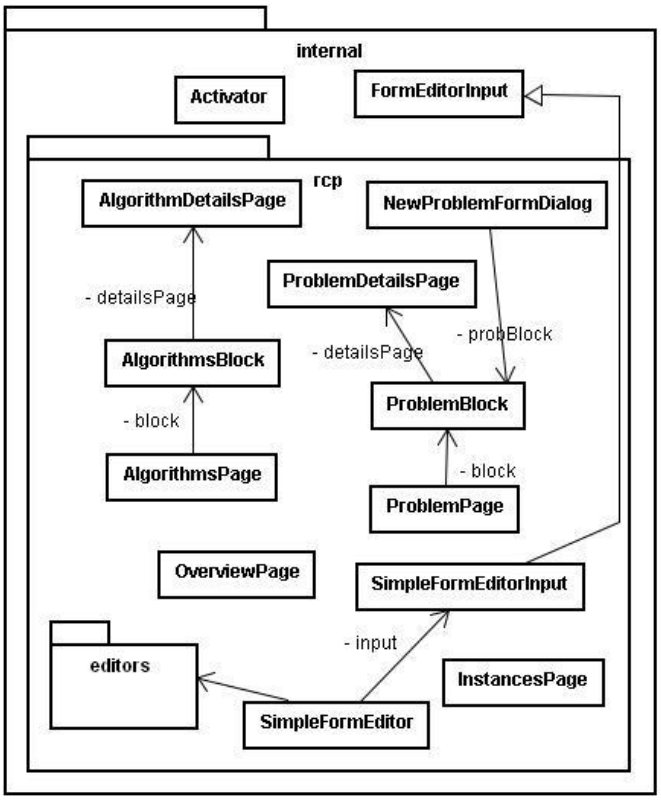


Figura 5.7: Diagrama de clases de los paquetes *internal* y *rcp*.

Como se puede ver en el Diagrama de Secuencia siguiente, cuando se realiza un cambio sobre algún campo configurable del Form, éste se pone en

estado *dirty*, habilitándose la opción de *Guardar*. Entonces cuando el usuario realiza la acción de *Guardar* los cambios, se llama al método doSave del SimpleFormEditor, en el que se realiza la serialización con la clase Marshalling del Experimento, representado por un objeto de la clase Jmh4eExperimentConfiguration, y crea el contenido de la clase Java MiExperimento.

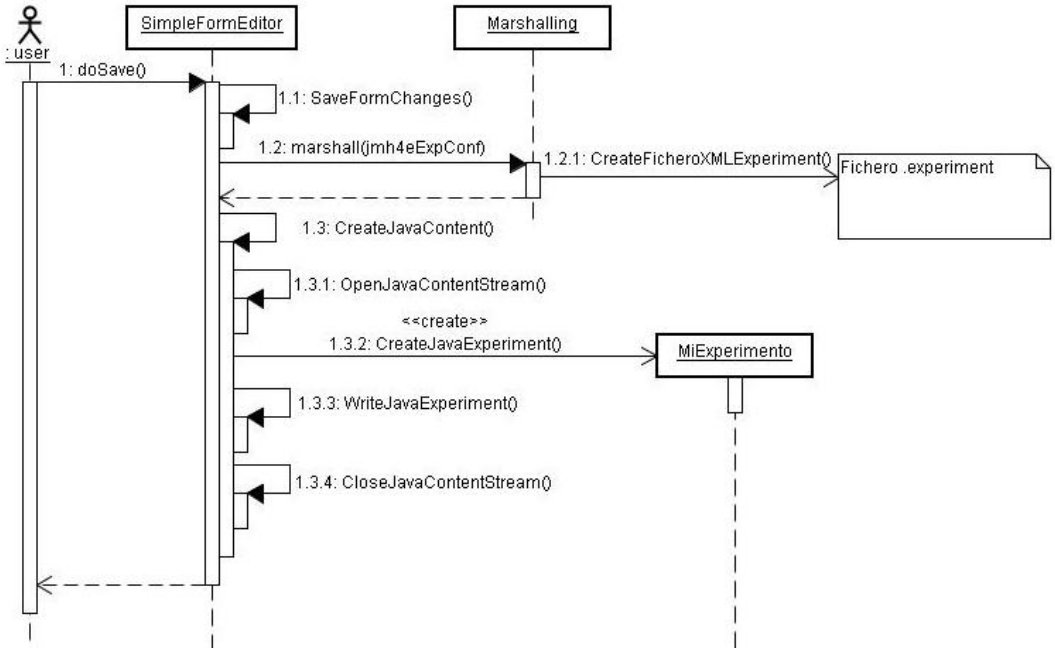


Figura 5.8: Diagrama de Secuencia del doSave del SimpleFormEditor.

5.3. Implementación

Como se explica en el Capítulo 4, se ha seguido una Implementación Incremental, en la que en cada reunión con los tutores se definían los nuevos requisitos a cumplir por el proyecto, en el proceso de desarrollo se implementaban generando así un prototipo, sobre el que realizar las pruebas.

A continuación se describen las reuniones realizadas y las tareas llevadas a cabo para implementar los requisitos propuestos en cada reunión. Se adjuntan capturas del prototipo sobre el que se van realizando las prue-

bas pertinentes, los errores y problemas encontrados a medida que se iba desarrollando y las referencias a fragmentos de código relevantes de la implementación, que se encuentran en los Apéndices.

5.3.1. Reunión de presentación del proyecto (8/10/09)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres de
1	**1ª Reunión: Buscar información y ejemplos**	**17,44 días**	**jue 08/10/09**	**mar 27/10/09**		**Rocío**
2	Sobre Plugins en Eclipse	4 días	jue 08/10/09	dom 11/10/09		Rocío
3	Sobre la Librería SWT y J-Face	4 días	lun 12/10/09	vie 16/10/09	2	Rocío
4	Sobre el uso de FormEditor en Plugins de Eclipse	4 días	sáb 17/10/09	mié 21/10/09	3	Rocío
5	Crear un ejemplo de Plugin del tipo FormEditor	5 días	jue 22/10/09	mar 27/10/09	4	Rocío

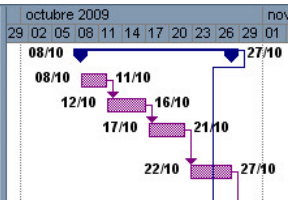


Figura 5.9: Diagrama de Gantt: 1ª Reunión.

En la primera reunión del proyecto, se presentaron el problema a resolver y los objetivos a desarrollar, proponiéndose para iniciar el proyecto la búsqueda de información sobre las herramientas a utilizar, así como las tecnologías más importantes a emplear, descritas en el capítulo sobre Aspectos del Desarrollo.

5.3.2. 2ª Reunión (27/10/09)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres de
6	**2ª Reunión: Desarrollo de la primera versión del FormEditor**	**28 días**	**mar 27/10/09**	**mar 01/12/09**	**1**	**Rocío**
7	Revisar el Modelo de datos de Optsicom y adaptarlo para reutilizarlo.	4 días	mar 27/10/09	lun 02/11/09	5	Rocío
8	Buscar documentación sobre JAXB2	6 días	lun 02/11/09	lun 09/11/09	7	Rocío
9	**Creación del FormEditor**	**18 días**	**lun 09/11/09**	**mar 01/12/09**	**8**	**Rocío**
10	Crear pestaña Overview: clase OverviewPage	6 días	lun 09/11/09	mar 17/11/09	8	Rocío
11	Crear pestaña Algorithm: clase AlgorithmsPage	6 días	mar 17/11/09	mar 24/11/09	10	Rocío
12	Crear pestaña Instances: clase InstancesPage	6 días	mar 24/11/09	mar 01/12/09	11	Rocío

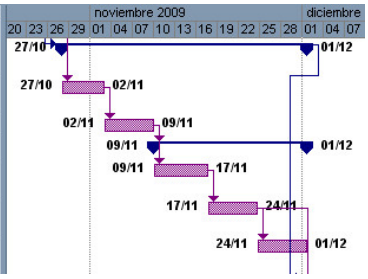


Figura 5.10: Diagrama de Gantt: 2ª Reunión.

Al iniciar el trabajo se centró la atención en la parte de Definición de Experimentos.

En un principio, tenemos la Librería de Optsicom para implementar los algoritmos en Java y luego tenemos Proyectos Problema, en Java también, que están compuestos por varios algoritmos y conjuntos de instancias (o agrupaciones de instancias), que estarán definidos como una Estructura de Datos en memoria y que serán gestionadas por el Editor de Experimentos. El Modelo inicial sobre el que se basará el formulario, estará compuesto básicamente por una jerarquía como la siguiente:

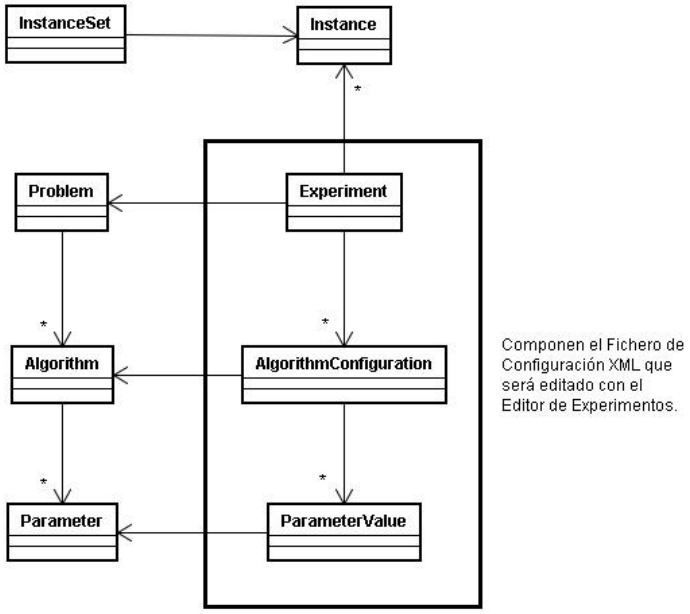


Figura 5.11: Modelo de datos.

El Editor de Experimentos será un FormEditor [8], que modificará las instancias de la estructura de los experimentos y las guardará como un fichero XML (con extensión .experiment), para lo cual además haremos uso de JAXB [19] con la que se puede transformar un objeto java en un fichero XML (como el *Marshalling* que realiza el *Middleware Corba*).

Como ejemplo del objetivo a desarrollar tenemos *Optsicom*, que es un proyecto en desarrollo como *Rich Client Aplication* de Eclipse para configu-

rar y ejecutar también Experimentos, que utiliza un Modelo de Estructura de Datos que nos puede ser útil y que podremos reutilizar, así como partes clave de la interfaz que sirvan de ejemplo.

El FormEditor a desarrollar contendrá en principio tres Pages (pestañas):

- Overview: Donde se elegirá el Problema (o Modelo) que se va a ejecutar de los que hay definidos en el entorno, utilizando unos criterios de búsqueda. Tras hacer una selección de los Problemas más adecuados con los criterios introducidos, estos deberán ser mostrados y el usuario seleccionará de una lista el que necesite y procederá a cargarlo.
  La otra opción que se baraja para la página Overview sería simplemente seleccionar de la lista completa de Modelos existentes el que queremos ejecutar y cargarlo:

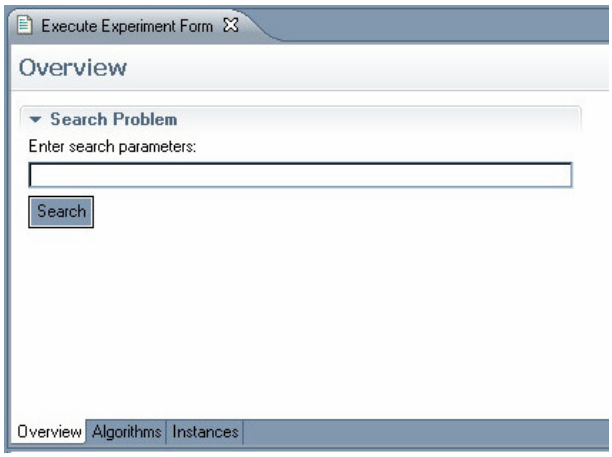


Figura 5.12: Página Overview con Búsqueda del Problema.

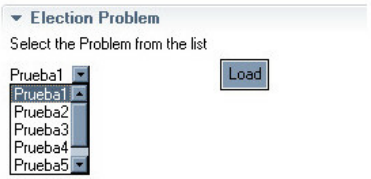


Figura 5.13: Elección del Problema desde una lista en un ComboBox.

- Algorithm: Basada en la opción de selección y configuración de algoritmos de *Optsicom*. En el cuadro de la página aparece la lista de los algoritmos que pertenecen al Problema seleccionado (las que se muestran son instancias de prueba).

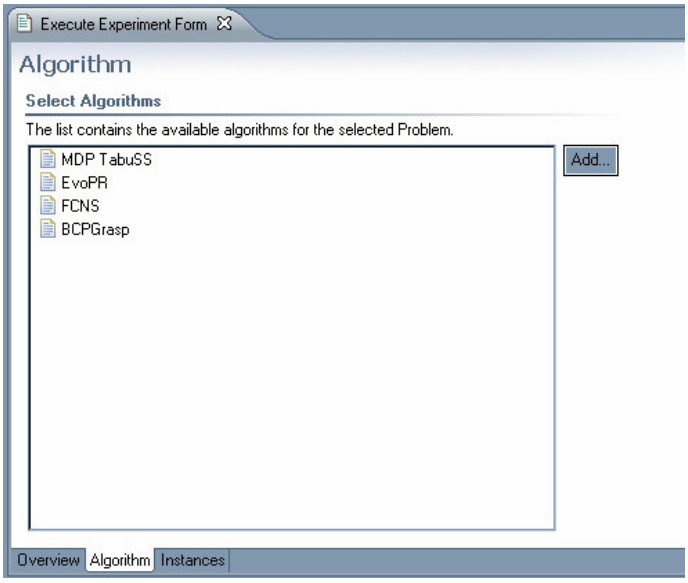


Figura 5.14: Página Algorithm.

Cuando se pulsa sobre uno de los elementos de la lista, sus propiedades se muestran en el lado de la derecha, donde además se podrán cambiar los valores de los parámetros que contiene. Cuando se pulse el botón Add se abrirá una ventana donde aparecerán todos los algoritmos pertenecientes al problema seleccionado y en la que seleccionando un algoritmo determinado este será añadido a la lista de la ventana Algorithm para su posterior visualización y configuración.

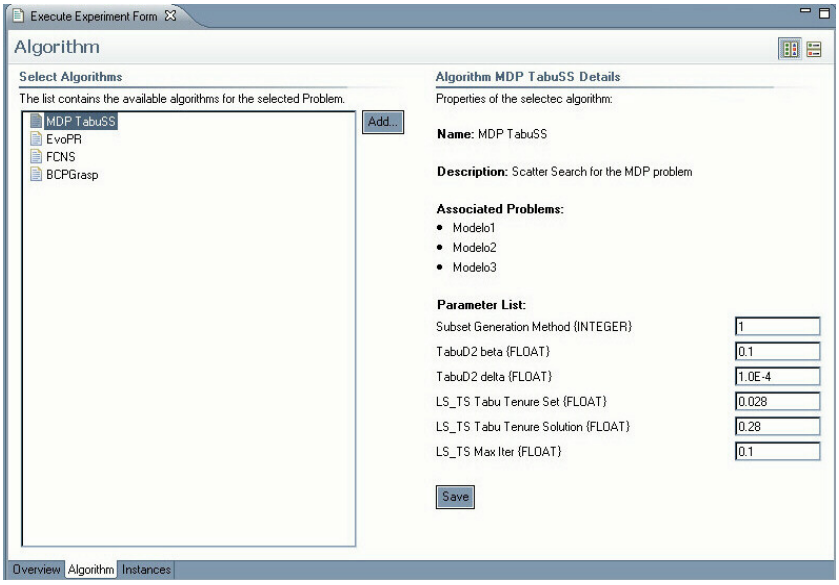


Figura 5.15: Página Algorithm con los detalles del Algoritmo seleccionado mostrándose a la derecha.

- Instances: Al igual que el anterior, basada en la selección de Instancias, mostrándolas como un árbol como realiza *Optsicom*.

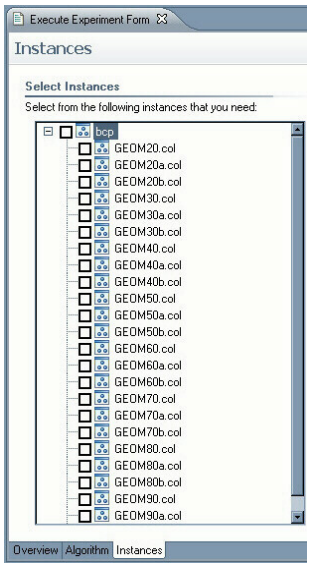
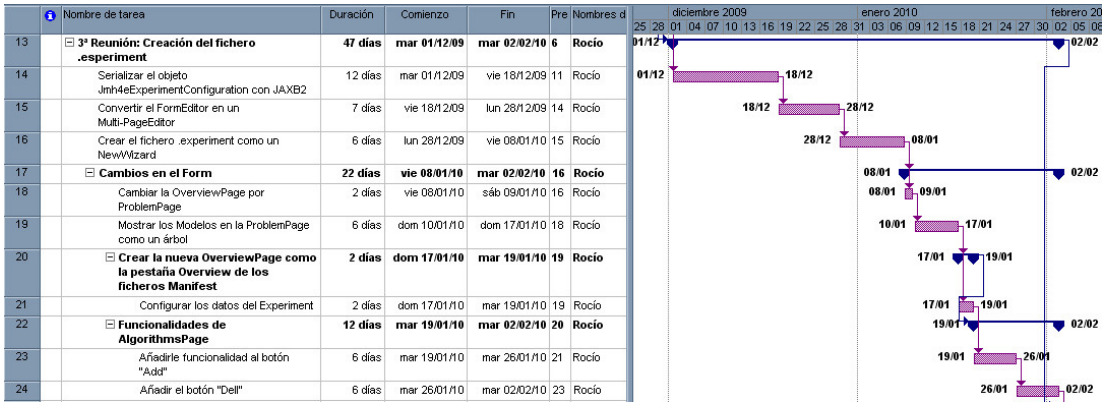


Figura 5.16: Vista de la InstancesPage.

Por lo tanto, el trabajo inicial consiste en iniciar la implementación del FormEditor y sus Pages, e implementar el Modelo de Estructura de Datos a utilizar, utilizando como referencia base el que utiliza *Optsicom*.

5.3.3. 3ª Reunión (01/12/2009)



	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
13	**3ª Reunión: Creación del fichero .esperiment**	**47 días**	**mar 01/12/09**	**mar 02/02/10**	**6**	**Rocío**
14	Serializar el objeto Jmh4eExperimentConfiguration con JAXB2	12 días	mar 01/12/09	vie 18/12/09	11	Rocío
15	Convertir el FormEditor en un Multi-PageEditor	7 días	vie 18/12/09	lun 28/12/09	14	Rocío
16	Crear el fichero .experiment como un NewWizard	6 días	lun 28/12/09	vie 08/01/10	15	Rocío
17	**Cambios en el Form**	**22 días**	**vie 08/01/10**	**mar 02/02/10**	**16**	**Rocío**
18	Cambiar la OverviewPage por ProblemPage	2 días	vie 08/01/10	sáb 09/01/10	16	Rocío
19	Mostrar los Modelos en la ProblemPage como un árbol	6 días	dom 10/01/10	dom 17/01/10	18	Rocío
20	**Crear la nueva OverviewPage como la pestaña Overview de los ficheros Manifest**	**2 días**	**dom 17/01/10**	**mar 19/01/10**	**19**	**Rocío**
21	Configurar los datos del Experiment	2 días	dom 17/01/10	mar 19/01/10	19	Rocío
22	**Funcionalidades de AlgorithmsPage**	**12 días**	**mar 19/01/10**	**mar 02/02/10**	**20**	**Rocío**
23	Añadirle funcionalidad al botón "Add"	6 días	mar 19/01/10	mar 26/01/10	21	Rocío
24	Añadir el botón "Dell"	6 días	mar 26/01/10	mar 02/02/10	23	Rocío

Figura 5.17: Diagrama de Gantt: 3ª Reunión.

Los siguientes pasos a realizar serán:

- Serialización de configuración del Experimento a XML con JAXB2 y de los objetos Model, con el método marshall() de la clase es.optsicom.eclipse.utils.Marshalling [ver el código fuente 11 del apéndice].
- Cambiar la clase Jmh4eExperiment a Jmh4eExperimentConfiguration, ya que una clase Experiment formará parte de la librería Optsicom y será quien ejecute el Experimento en base a la configuración del Jmh4eExperimentConfiguration.
- Afinar el diseño del formulario, para que permita la creación del fichero .experiment de configuración del Experimento, que será la representación XML de un objeto de la clase Jmh4eExperimentConfiguration. Por ese motivo hay que convertir el formulario realizado en un Multi-PageEditor con el que crear, editar y guardar el .experiment.

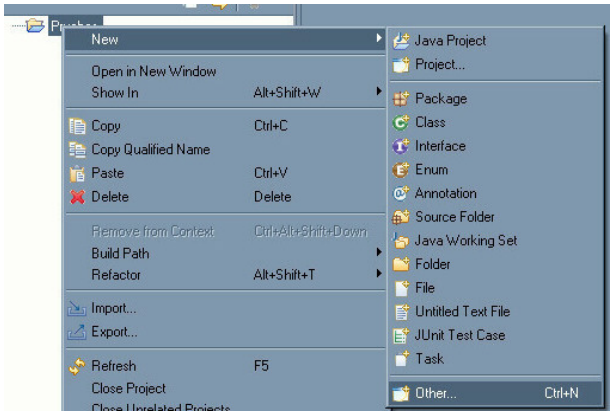


Figura 5.18: Creación del fichero .experiment como un NewWizard.

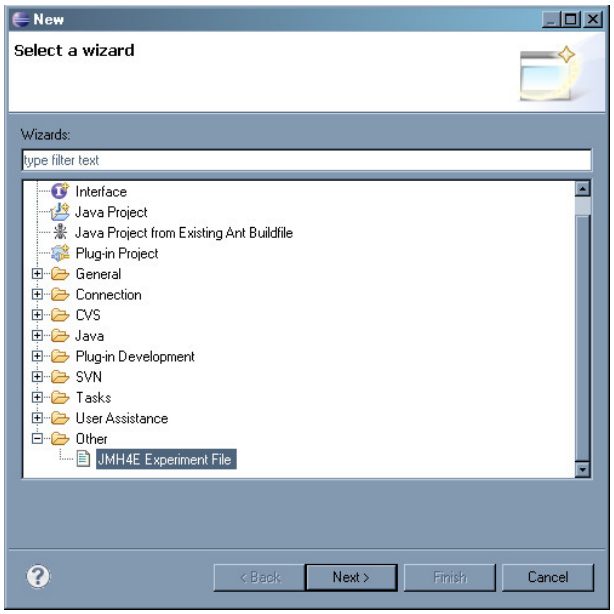


Figura 5.19: Selección del fichero **Optsicom**.

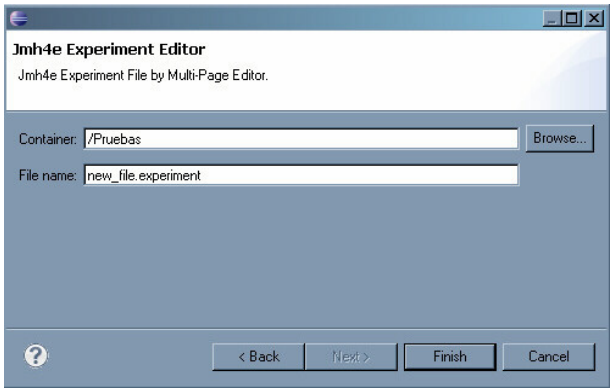


Figura 5.20: Vista de la NewWizardPage.

- Cambios en la vista de las Pages: tras el primer prototipo, ya se puede tener una idea aproximada de cómo se encaminará el Plugin y qué es realmente necesario que se complete en el formulario. Por ello se realizan los siguientes cambios y mejoras para la siguiente versión del prototipo, de modo que tras estos cambios ya se aprecia un inicio de la versión final del Plugin.

  - Cambiar el nombre de la página Overview por Problem.

  - Cambiar la vista de la página Problem para que aparezcan todos los Problemas como un árbol (similar al árbol de la tabla de la página Instances) y donde se seleccione el problema deseado para cargarlo.
    Cuando se seleccione uno de los problemas aparecerán al lado sus atributos (como aparecen al pinchar sobre uno de los algoritmos de la página Algorithms) con formato de página Web.

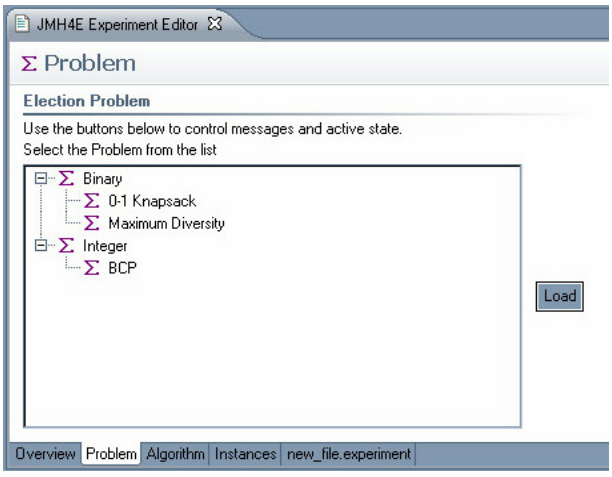


Figura 5.21: Vista de la nueva ProblemPage.

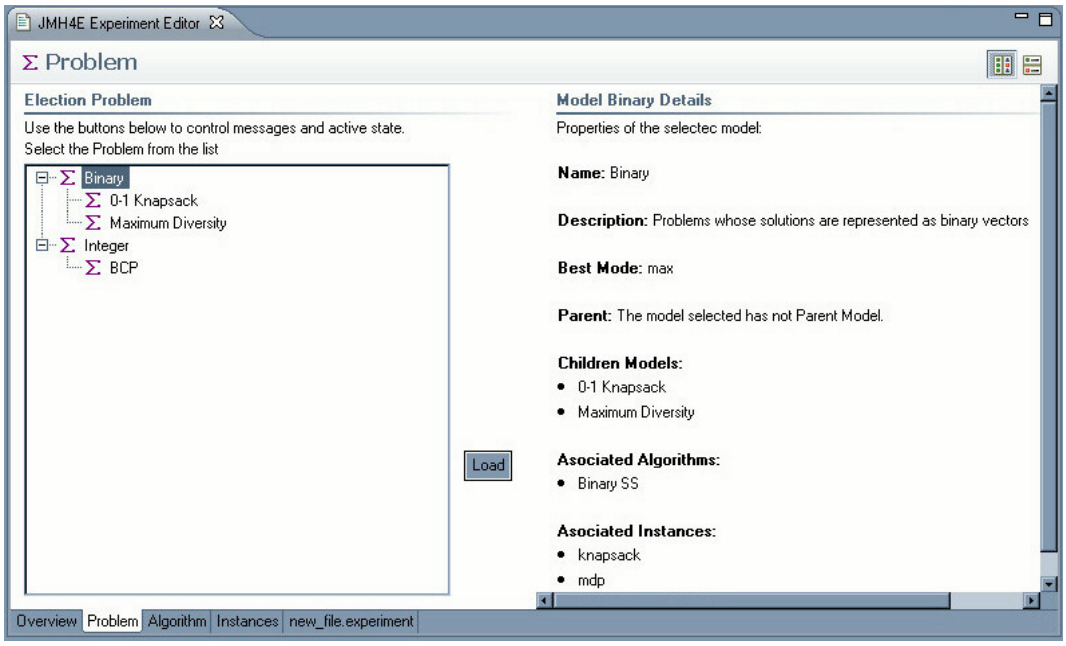


Figura 5.22: Vista de la nueva ProblemPage con los datos del Problema seleccionado a la derecha.

- Crear una nueva página Overview que tendrá una vista similar a la pestaña Overview de los ficheros Manifest de configuración de proyectos de desarrollo de *Plugins*, en la que se configurarán los parámetros deseados de la clase Jmh4eExperimentConfiguration.

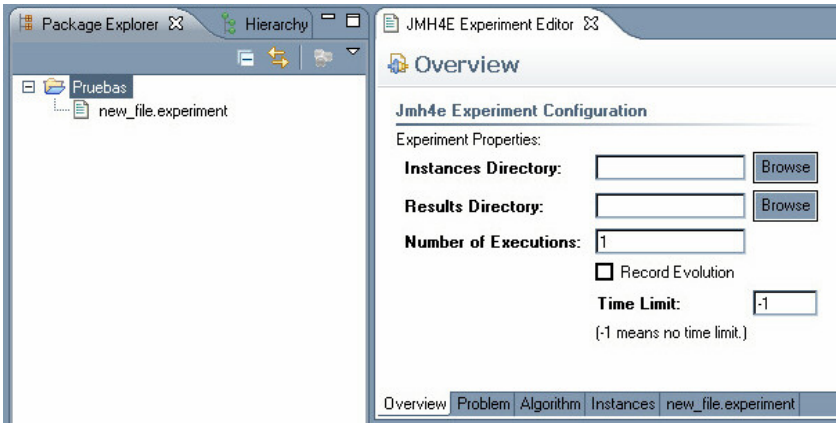


Figura 5.23: Nueva vista de la OverviewPage.

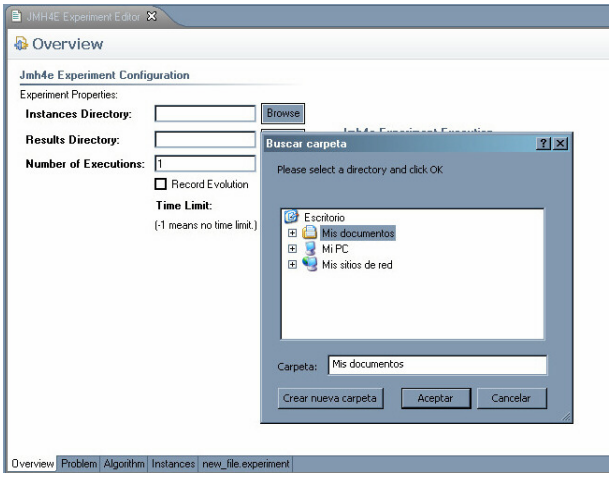


Figura 5.24: Al pulsar un botón Browse se abre una ventana de búsqueda de directorios. Tras seleccionarlo y pulsar Aceptar, su path se guarda en el campo de texto correspondiente de la página OverviewPage.

- Completar la implementación de la página Algorithms añadiéndole la funcionalidad al botón Add... para que se abra una ventana donde aparezcan todos los algoritmos asociados al problema, se seleccione el deseado y se añada una nueva instancia de la clase AlgorithmConfiguration de dicho algoritmo a la tabla de la página Algorithms (que hasta ahora contenía instancias de prueba de la clase Algorithm).

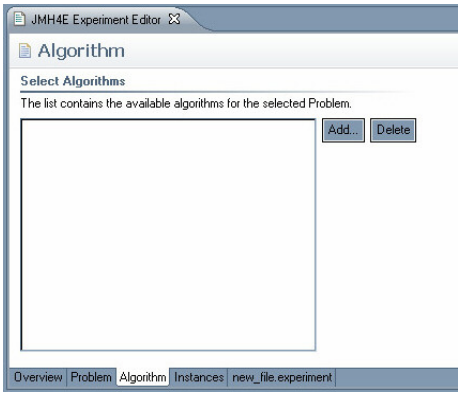


Figura 5.25: Nueva vista de AlgorithmsPage.

Además, se le debe añadir la funcionalidad de poder cambiar los valores por defecto de los parámetros y que esos cambios se guarden correctamente.

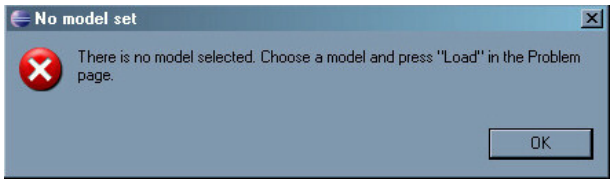


Figura 5.26: En la AlgorithmsPage, si se pulsa *Add* sin haber cargado el Problema aparecerá un mensaje de Error.

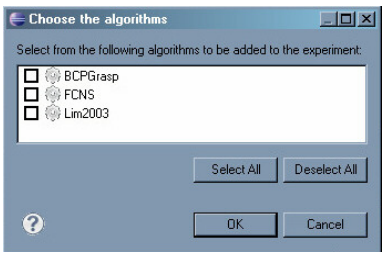


Figura 5.27: Diálogo para agregar un Algoritmo a la lista.

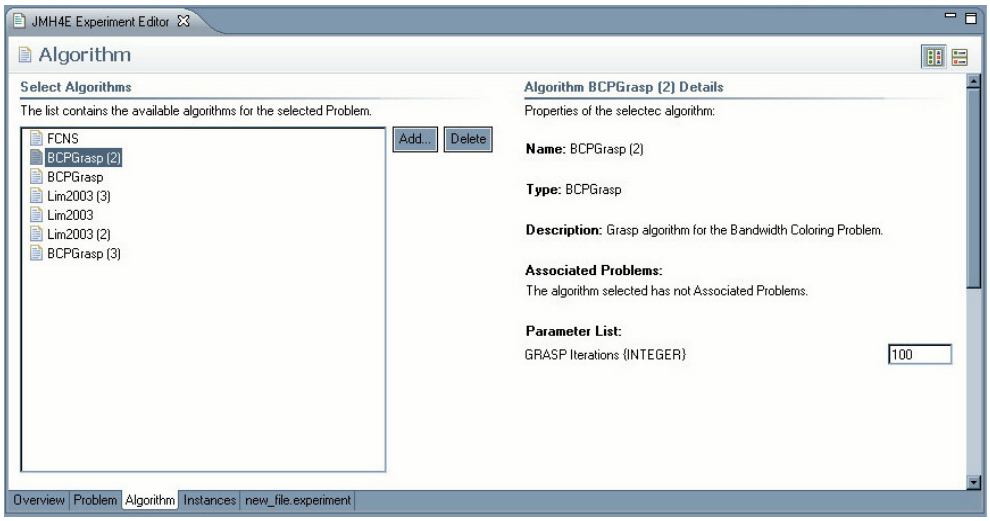


Figura 5.28: Nueva vista de AlgorithmsPage, mostrando las características del algoritmo seleccionado a la derecha.

- Añadir una nueva pestaña de tipo TextEditor donde se mostrará el contenido del fichero .experiment.

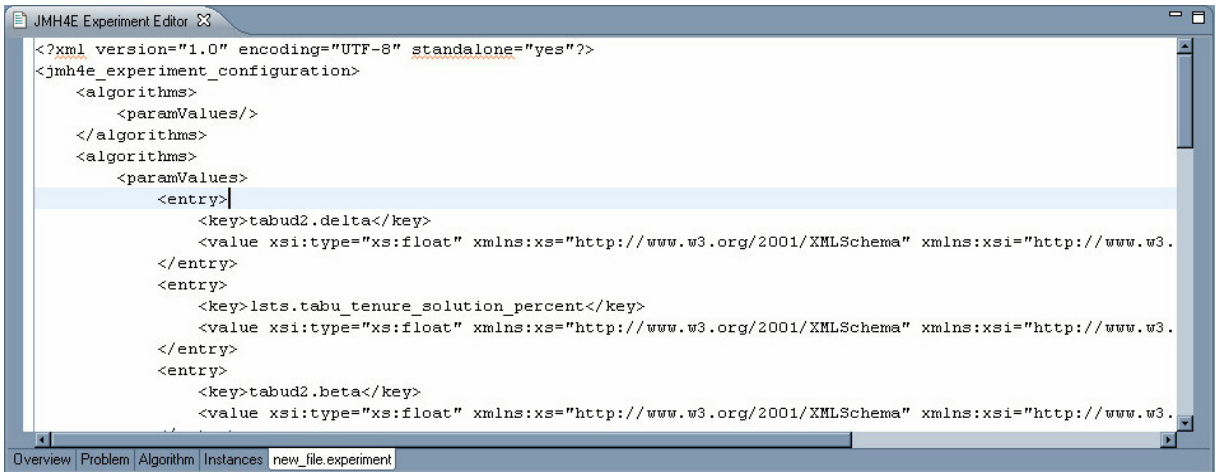
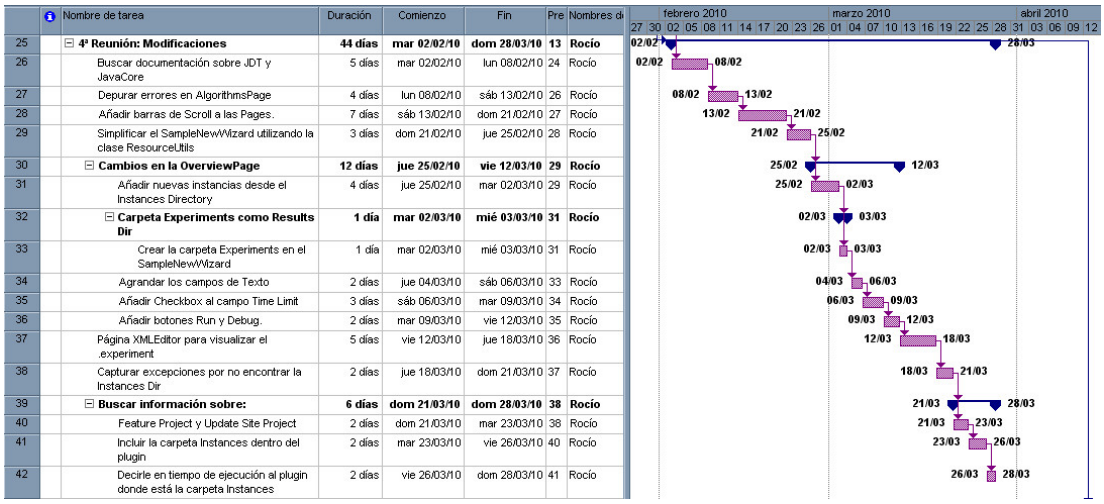


Figura 5.29: Vista del TextEditor que muestra el contenido del fichero .experiment.

Para llevar a cabo estos cambios y poder probarlos, será necesario también que se implemente un Modelo de Pruebas más extenso y completo. Además, para poder realizar la serialización a XML del Jmh4eExperimentConfiguration se deben repasar las anotaciones @XML del modelo de datos para una correcta serialización [22].

5.3.4. 4ª Reunión (02/02/2010)

Para esta reunión quedaba pendiente el análisis de código en busca de métodos (clases hijas de ApproxMethod) y definir el siguiente paso a implementar.



	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
25	**4ª Reunión: Modificaciones**	**44 días**	**mar 02/02/10**	**dom 28/03/10**	**13**	**Rocío**
26	Buscar documentación sobre JDT y JavaCore	5 días	mar 02/02/10	lun 08/02/10	24	Rocío
27	Depurar errores en AlgorithmsPage	4 días	lun 08/02/10	sáb 13/02/10	26	Rocío
28	Añadir barras de Scroll a las Pages.	7 días	sáb 13/02/10	dom 21/02/10	27	Rocío
29	Simplificar el SampleNewWizard utilizando la clase ResourceUtils	3 días	dom 21/02/10	jue 25/02/10	28	Rocío
30	**Cambios en la OverviewPage**	**12 días**	**jue 25/02/10**	**vie 12/03/10**	**29**	**Rocío**
31	Añadir nuevas instancias desde el Instances Directory	4 días	jue 25/02/10	mar 02/03/10	29	Rocío
32	**Carpeta Experiments como Results Dir**	**1 día**	**mar 02/03/10**	**mié 03/03/10**	**31**	**Rocío**
33	Crear la carpeta Experiments en el SampleNewWizard	1 día	mar 02/03/10	mié 03/03/10	31	Rocío
34	Agrandar los campos de Texto	2 días	jue 04/03/10	sáb 06/03/10	33	Rocío
35	Añadir Checkbox al campo Time Limit	3 días	sáb 06/03/10	mar 09/03/10	34	Rocío
36	Añadir botones Run y Debug.	2 días	mar 09/03/10	vie 12/03/10	35	Rocío
37	Página XMLEditor para visualizar el .experiment	5 días	vie 12/03/10	jue 18/03/10	36	Rocío
38	Capturar excepciones por no encontrar la Instances Dir	2 días	jue 18/03/10	dom 21/03/10	37	Rocío
39	**Buscar información sobre:**	**6 días**	**dom 21/03/10**	**dom 28/03/10**	**38**	**Rocío**
40	Feature Project y Update Site Project	2 días	dom 21/03/10	mar 23/03/10	38	Rocío
41	Incluir la carpeta Instances dentro del plugin	2 días	mar 23/03/10	vie 26/03/10	40	Rocío
42	Decirle en tiempo de ejecución al plugin donde está la carpeta Instances	2 días	vie 26/03/10	dom 28/03/10	41	Rocío

Figura 5.30: Diagrama de Gantt: 4ª Reunión.

Tras la implementación del FormEditor y la creación del fichero .experiment, el siguiente paso consiste en la ejecución del experimento configurado. Para ello es necesario conocer los algoritmos que se deben ejecutar, ya que la jerarquía de la clase padre Algorithm sólo es un *recubrimiento* con el que se conocen los parámetros y características de cada algoritmo, pero no contiene el algoritmo en sí que se debe ejecutar.
Para profundizar en la materia, se debe investigar sobre la clase JavaCore, perteneciente al plugin JDT y incluir una dependencia a los plugin org.eclipse.jdt y org.eclipse.jdt.core. JavaCore se utiliza para obtener la jerarquía de herencia de una clase, en este caso de la clase ApproxMethod, que es la clase padre de los algoritmos ejecutables.

Además de lo mencionado, en el Form creado deben hacerse las siguientes modificaciones:

- Revisar la página AlgorithmsPage para depurar errores cuando se realizan modificaciones.
- Añadir barras de *Scroll* a las Pages, revisando los tamaños de los componentes para que se añadan barras de *Scroll* cuando la ventana

se haga más pequeña.

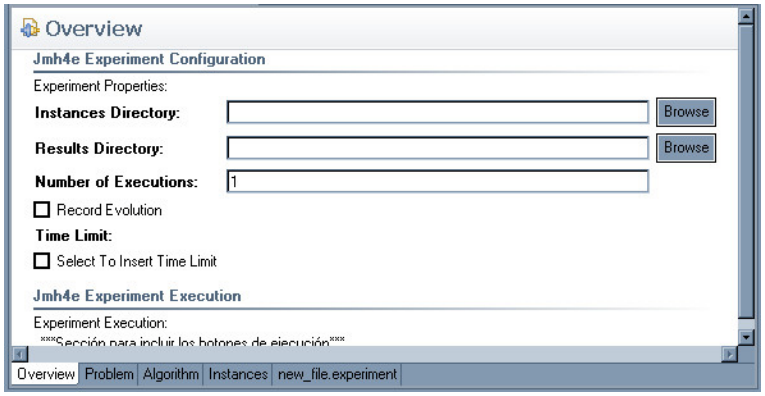


Figura 5.31: Nueva vista de la OverviewPage con barras de *Scroll*.

- En lugar de utilizar dos atributos estáticos en el NewSimpleWizard para obtener el path absoluto del fichero .experiment que se crea, utilizar la clase ResourceUtil del siguiente modo, para obtener el IFile en el SimpleFormEditor [ver el código fuente 12 del apéndice].

- En la página Overview se deben realizar los siguientes cambios y mejoras:

  - Agrandar los campos de texto donde se añadirán la ruta de los Directorios, para que muestren más caracteres.

  - Se debe tener en cuenta que el *Instances Directory* de la clase Jmh4eExperimentConfiguration que se indica en esta página contendrá más instancias de datos que deben ser también incluidas en el árbol de instancias de la InstancesPage, para que el usuario las añada al Experimento si quiere. estas nuevas instancias no han sido añadidas al problema por haber sido creadas por el usuario, u obtenidas de otros proyectos, por lo que se incluyen en el directorio dicho.

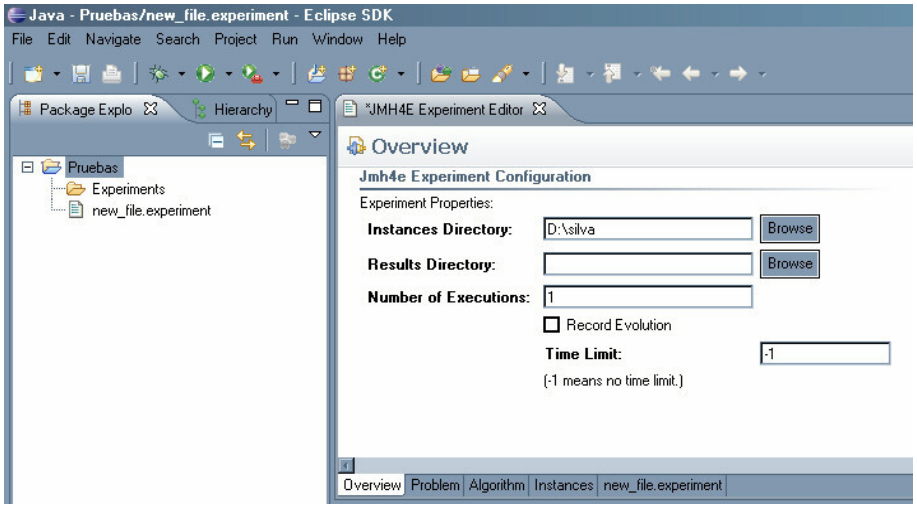


Figura 5.32: Ejemplo que muestra como añadir nuevas instancias.

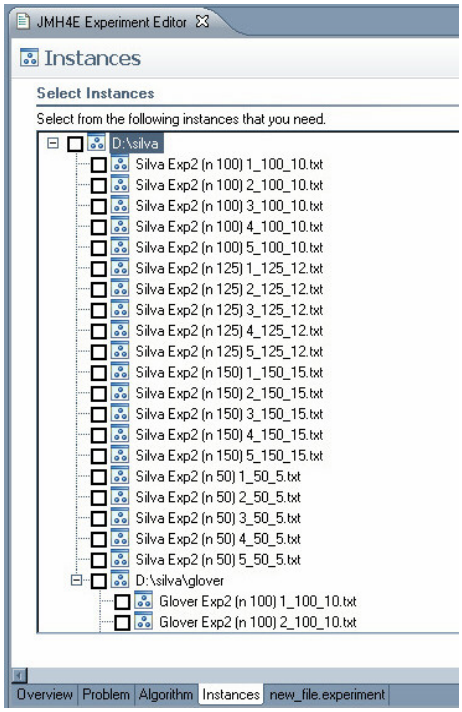


Figura 5.33: Nuevas instancias añadidas.

- El *Results Dir* debe ser por defecto una carpeta llamada *Experiments* que se creará dentro del proyecto cuando se cree el fichero

.experiment. En dicha carpeta se añadirán los resultados de las ejecuciones, de modo que cada ejecución tenga su resultado en una carpeta identificada con el Nombre, la Fecha y la Hora de la ejecución. De este modo no habrá colisiones entre los datos de los resultados de las ejecuciones y será totalmente identificable cada resultado.

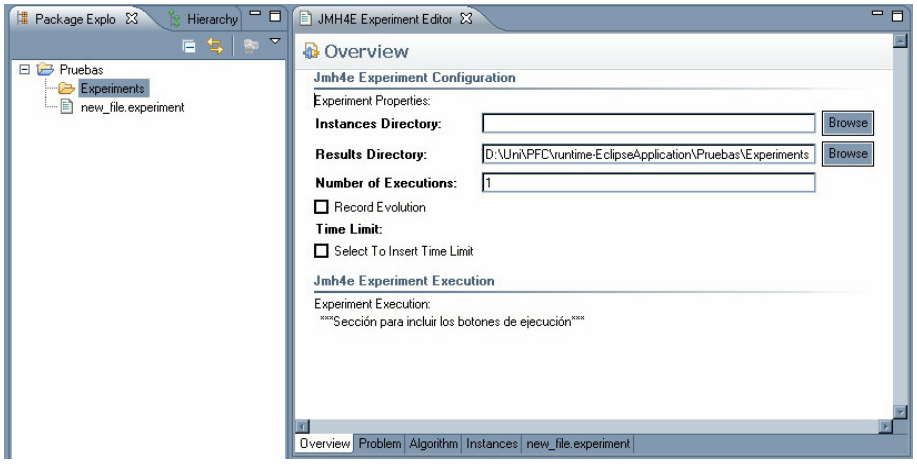


Figura 5.34: Carpeta *Experiments*.

- Cambiar el estilo del campo *Time Limit*: poner un CheckBox de modo que cuando esté a *true* (clicado) se debe habilitar el cuadro de texto (*Time Limit* > 0), y si está a *false* (no clicado) se deshabilite (*Time Limit* $= -1$). Añadir también la unidad de medida al lado, Segundos. Posteriormente esos segundos se deben multiplicar por 1000 ya que los algoritmos internamente utilizan milisegundos.

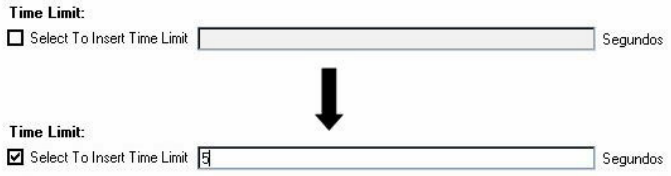


Figura 5.35: Campo *Time Limit*.

- La última página del Form, que es un TextEditor, debe ser sustituida por un XMLEditor, para que las palabras reservadas del XML se vean en colores y resaltadas.

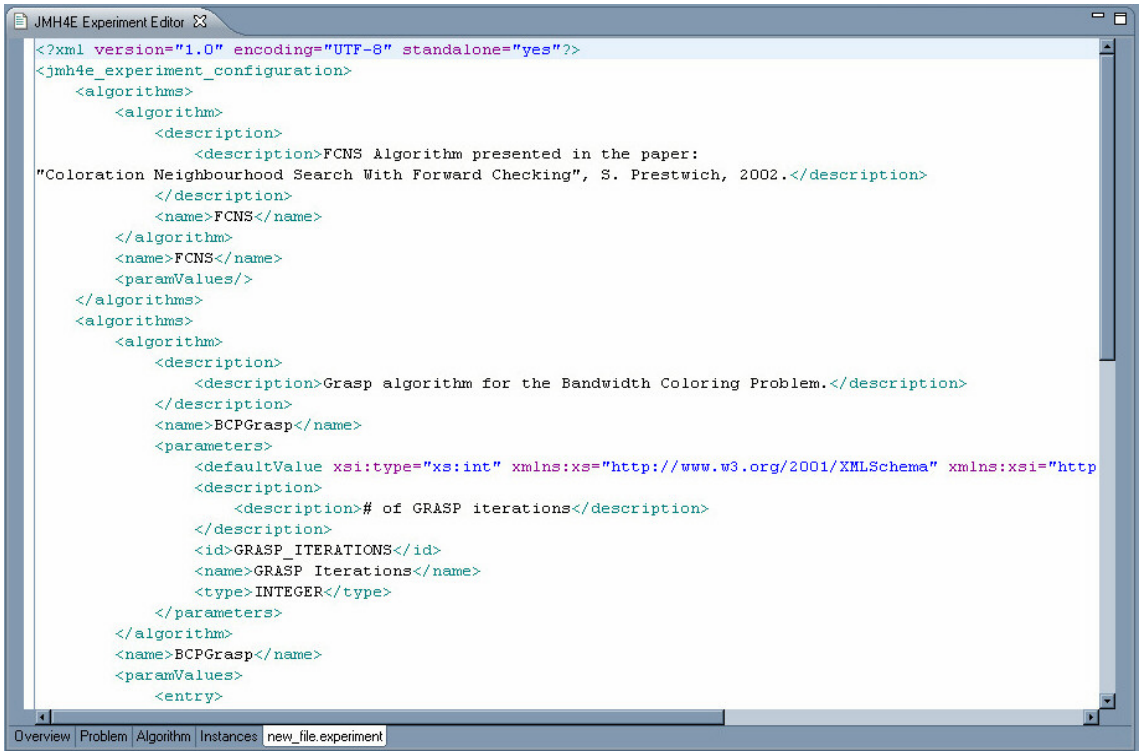


Figura 5.36: XMLEditor.

- Se deben capturar las excepciones que pueden salir cuando al iniciar el Form no se encuentran la carpeta de las instancias originales en la dirección provista, y en dicho caso, permitir buscar y seleccionar la carpeta correcta donde se encuentran las instancias.

- Poner los botones de Run y Debug, para que se pueda crear una configuración de Menú Launch (de tipo Java, LaunchConfiguration(JAVA)). El objetivo es añadir una nueva configuración al Run Configurations (*ShortCut - Botón derecho - Run As*) donde el Project es el proyecto donde se crea el .experiment y la clase Main que se ejecutará podrá implementarse según dos opciones posibles:

- Que se genere una clase Main.
- A partir del XML del .experiment, tener una clase que acceda a él y lo interprete.

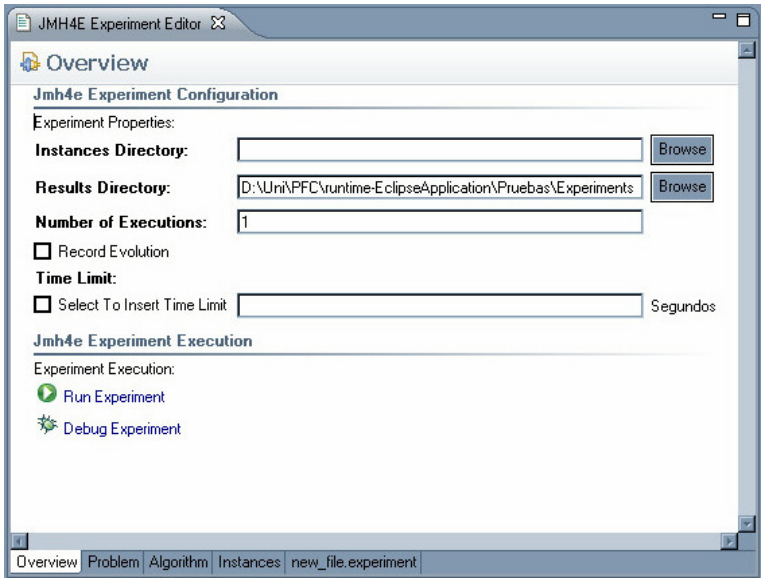


Figura 5.37: Vista de la OverviewPage.

Otras notas complementarias:

- Mirar qué son el Feature Project y el Update Site Project: se utilizan como una URL dónde buscar el Plugin, para poder obtenerlo desde un servidor y que sea accesible para su instalación en Eclipse desde *Help - Install New Software...*, y en el futuro poder actualizar la versión del plugin en Eclipse con el menú *Help - Check for Updates*.

- Incluir la carpeta Instances dentro del Plugin, para que no haya que copiarla a mano y se incluya dentro de la carpeta de instalación de Plugins en Eclipse y así pueda ser accedida desde el Plugin. El objetivo es que se cree una carpeta es.optsicom.eclipse dentro de la carpeta

plugins de Eclipse cuando se instale el Plugin, y que contendrá la carpeta *instances*.

- ¿Cómo decirle en tiempo de ejecución al plugin donde está la carpeta *instances* tras la instalación? En primer lugar se debe instalar el Plugin de forma desempaquetada en la carpeta es.optsicom.eclipse de la carpeta *plugins* de Eclipse, para tener así la carpeta *instances*. En segundo lugar, para acceder a la carpeta de instalación en tiempo de ejecución se utilizará el método Platform.getInstallLocation() que tiene el método Activator.getInstallLocation(). Hay que tener en cuenta que el Plugin se instalará con número de versión, por lo que el nombre de la carpeta desempaquetada del Plugin no será exactamente es.optsicom.eclipse, sino que tendrá la versión tras el carácter "_", [ver el código fuente 13 del apéndice].

5.3.5. 5ª Reunión (13/04/2010)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
43	**5ª Reunión: Descubrimiento de ApproxMethods**	**13 días**	**mar 13/04/10**	**mié 28/04/10**	**25**	**Rocío**
44	Obtener las hijas de una clase con JavaCore y JDT.	3 días	mar 13/04/10	vie 16/04/10		Rocío
45	Añadir nuevas clases hijas de ApproxMethod a la AlgorithmsPage.	2 días	vie 16/04/10	dom 18/04/10	44	Rocío
46	Deserializar el Experiment con unmarshall	3 días	dom 18/04/10	jue 22/04/10	45	Rocío
47	Añadirle Preferencias al Plugin	1 día	jue 22/04/10	vie 23/04/10	46	Rocío
48	Crear la clase main MiExperimento.	1 día	sáb 24/04/10	sáb 24/04/10	47	Rocío
49	Launche del Experimento mediante la clase MiExperimento	1 día	sáb 24/04/10	dom 25/04/10	48	Rocío
50	Inicio de la memoria	2 días	dom 25/04/10	mié 28/04/10	49	Rocío

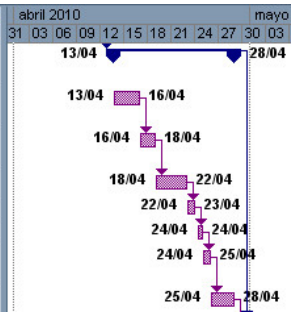


Figura 5.38: Diagrama de Gantt: 5ª Reunión.

Restricciones y nuevos requisitos:

- Para obtener las clases hijas de una clase en tiempo de ejecución se utiliza la clase JavaCore del paquete JDT. En la clase Hierachy se ha

implementado el método getSubClasses() que utiliza esta librería [ver el código fuente 14 del apéndice].

- Todas las clases deben encontrarse en el mismo proyecto, o bien como código fuente o como librería referenciada en el Build Path. [ver el código fuente 15 del apéndice].

- En el AlgorithmBlock de la AlgorithmPage se debe utilizar como lista base de algoritmos de la que se seleccionan los que se desea incluir en el experimento, la lista de las clases hijas de ApproxMethod, obtenida con JDT. Dicha lista se debería filtrar para sacar las clases que pertenezcan al problema seleccionado, pero esa parte se verá más adelante.

- Utilizar el método unmarshall() de la clase Marshall para deserializar el xml del Jmh4eExperimentConfiguration y así recomponer el contenido del Form cuando se cierra y se vuelve a abrir. La única manera de poder realizar el unmarshall de la clase abstracta Algorithm es añadiéndole la anotación @XmlSeeAlso dónde se le indican las clases hijas que tiene, por lo que en tiempo de ejecución no pueden incrementar las hijas de Algorithm, [ver el código fuente 16 del apéndice].

- Añadirle Preferencias al Plugin: añadirle al Plugin como preferencia el nombre de la clase de la que heredan los algoritmos. En el método start() de la clase Activator se añade la instrucción correspondiente para obtener el valor de una preferencia del plugin, [ver el código fuente 17 del apéndice].

- Contribuir a la ventana del menú en Eclipse *Window - Preferences* con una vista para editar las preferencias del Plugin, [ver el código fuente

18 del apéndice]. Para ello se añade la extensión PreferencePages al *plugin.xml*, [ver el código fuente 19 del apéndice].

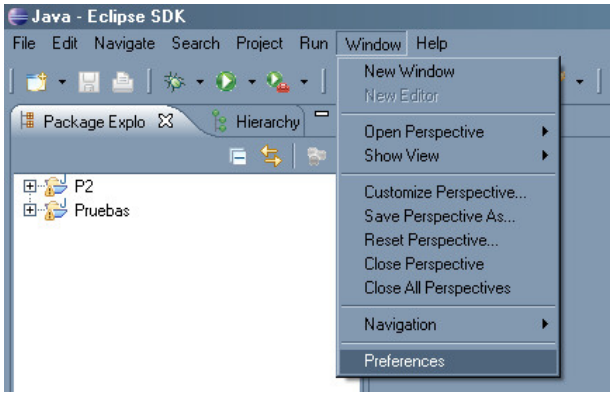


Figura 5.39: Acceso a la *Preferences Page de Eclipse*.

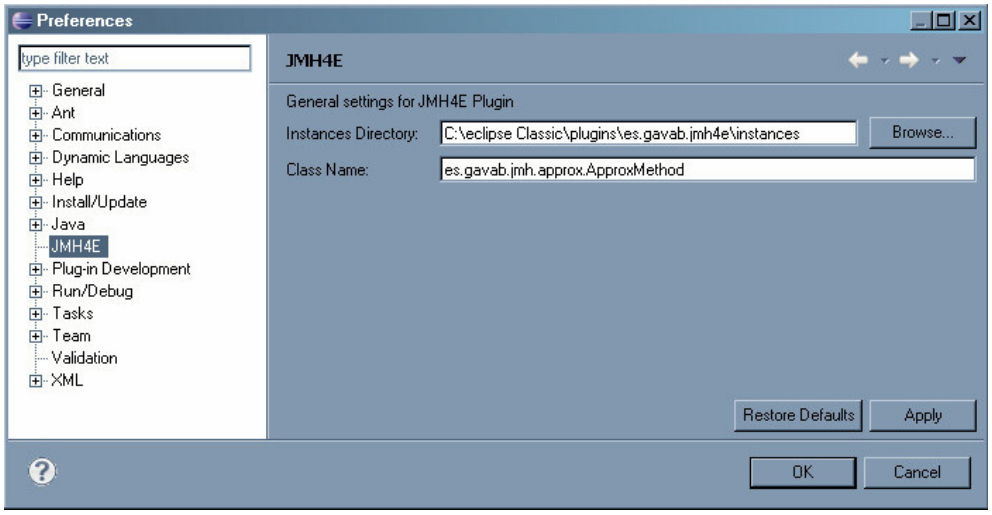


Figura 5.40: PreferencePage de Optsicom.

- Launch: para ejecutar el Experimento el Plugin debe crear una clase MiExperimento con un método main, que a partir del .experiment construya el Experimento y lo pueda ejecutar. Esta clase se debe crear dentro de la carpeta *src* del proyecto donde se ha creado el .experiment, y mostrar en el MultiForm como la última Page con un JavaEditor.

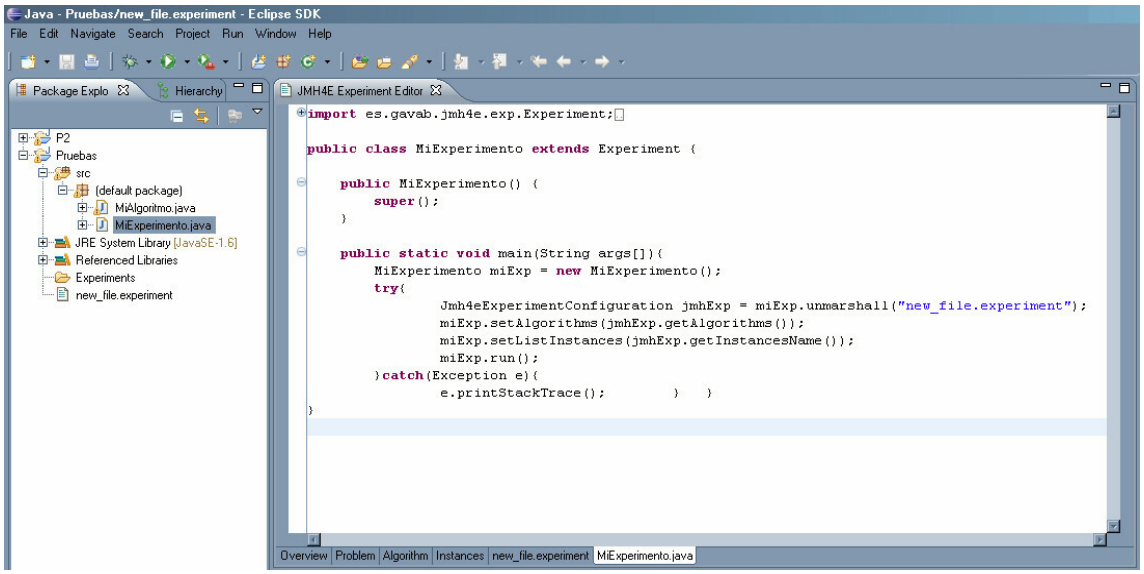


Figura 5.41: Vista de la pestaña JavaEditor.

- Para que la clase MiExperimento tenga las referencias a las librerías de *Optsicom*, y poder acceder a sus clases se deben añadir las librerías al proyecto donde se ejecuta el Plugin. Primero se deben tener estas librerías en la carpeta *lib* de la carpeta de instalación del Plugin *es.optsicom.eclipse*, dentro de la carpeta *plugins* de Eclipse y en tiempo de ejecución se añadirán al proyecto, [ver el código fuente 20 del apéndice].

Figura 5.42: Detalle de las librerías de Optsicom en el Proyecto donde se crea el .experiment.

5.3.6. 6ª Reunión (29/04/2010)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
51	**6ª Reunión: Solución de Bugs y Errores**	**11 días**	**jue 29/04/10**	**mar 11/05/10**	**43**	**Rocío**
52	Solución de Bugs	2 días	jue 29/04/10	sáb 01/05/10	50	Rocío
53	Implementación de la funcionalidad de recarga del plugin.	1 día	sáb 01/05/10	sáb 01/05/10	52	Rocío
54	Reimplementación de la clase Algorithm	1 día	dom 02/05/10	dom 02/05/10	53	Rocío
55	**Variaciones en la AlgorithmsPage**	**0 días**	**dom 02/05/10**	**dom 02/05/10**	**54**	**Rocío**
56	Añadir botón "Add from Workspace"	0 días	dom 02/05/10	dom 02/05/10	54	Rocío
57	Crear el Modelo de pruebas serializado: modelo.xml	1 día	dom 02/05/10	mar 04/05/10	56	Rocío
58	Añadirle la funcionalidad de log del AbstractUIPlugin para ver los errores.	0 días	mar 04/05/10	mar 04/05/10	57	Rocío
59	Añadir el fichero modelo.xml en una Preferences.	0 días	mar 04/05/10	mar 04/05/10	58	Rocío
60	Cambiar la clase MiExperimento y Experiment para que utilicen ApproxMethod	0 días	mar 04/05/10	mar 04/05/10	59	Rocío
61	Lanzar el experimento: implementarle funcionalidad a los botones Run y Debug	1 día	mar 04/05/10	jue 06/05/10	60	Rocío
62	Depurar la clase Activator	1 día	jue 06/05/10	vie 07/05/10	61	Rocío
63	Depurar la clase JavaEditor y XMLEditor	1 día	vie 07/05/10	sáb 08/05/10	62	Rocío
64	Solución de errores al serializar con JAXB	1 día	sáb 08/05/10	dom 09/05/10	63	Rocío
65	Pruebas de Instalación del Plugin	1 día	dom 09/05/10	dom 09/05/10	64	Rocío
66	Memoria: modificaciones estructurales.	1 día	dom 09/05/10	mar 11/05/10	65	Rocío

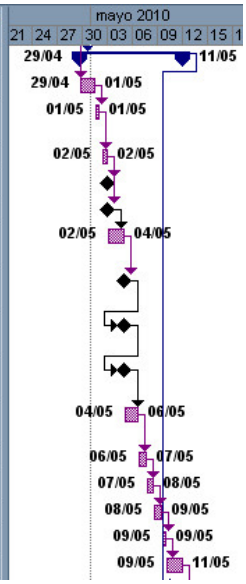


Figura 5.43: Diagrama de Gantt: 6ª Reunión.

A partir de esta reunión se decide cambiar el nombre del Plugin de **JMH4E** a **Optsicom 4 Eclipse**, por lo que se modifican los nombres de los recursos visibles en el Plugin implementados en el proyecto.

Sobre el problema del Unmarshall de la clase abstracta Algorithm: al ser una clase abstracta, cuando JAXB iba a deserializar su representación XML necesitaba saber qué clases eran sus hijas y la única manera de hacerlo era mediante la anotación @XmlSeeAlso. Pero esta solución en tiempo de ejecución no nos servía, ya que la jerarquía de herencia de la clase Algorithm podía aumentar en tiempo de ejecución.

Solución: la clase Algorithm deja de ser abstracta, ya no tendrá una jerarquía de herencia, ni el método getJMHMethod, y se utilizará un atributo executable necesario para sustituir su forma abstracta, que contendrá el nombre del ApproxMethod con el que se corresponde. De este modo se solucionará también el problema a la hora de crear nuevos algoritmos Approx-

Method, ya que se debe crear un Algorithm que lo represente para su uso con el MultiForm.

De este modo también, desaparecerá el paquete *problems* del modelo de objetos del plugin, ya que es innecesario.

Solucionado este problema, se debe terminar de implementar la funcionalidad de *Recarga* cuando el Form se cierra y se vuelve a abrir, de modo que se inicializan todos los parámetros modificables del FormEditor con el valor contenido en el fichero .experiment. De otro modo, se perderían los valores ya introducidos y el .experiment se crearía de nuevo vacío.

Se debe tener en cuenta que lo que se va guardando en el Experimento son AlgorithmConfiguration, que tienen instancias de Algorithm. Para descubrir nuevos algoritmos (nuevas clases hijas de ApproxMethod) que ha creado el usuario, se debe añadir un nuevo botón a la página AlgorithmsPage (al AlgorithmBlock concretamente), que se llame *Add from Workspace*, y en el que se busquen nuevos algoritmos para añadirlos al experimento.

Se buscan nuevas clases hijas de ApproxMethod con JDT, y se muestran para que el usuario las seleccione. Cuando seleccione una, se creará un objeto de la clase Algorithm que represente a ese algoritmo, y un objeto de la clase AlgorithmConfiguration que lo contenga para mostrarlo en la vista. Esa configuración se deberá guardar en el Modelo para tener persistencia y que el usuario no tenga q volver a añadir ese algoritmo como nuevo, sino con el *Add...* normal.

Por lo tanto se debe variar la AlgorithmsPage, añadiéndole el botón *Add from Workspace*, de modo que al pulsarlo se abrirá una ventana en la que se mostrarán los proyectos que hay en el *Workspace*, se seleccionará uno/o varios de esos proyectos y se buscarán las clases hijas de ApproxMethod nuevas implementadas por el usuario y que haya en los proyectos seleccionados. Estas clases se mostrarán en la siguiente ventana, donde el usuario po-

drá seleccionar las que desee añadir al Problema seleccionado.

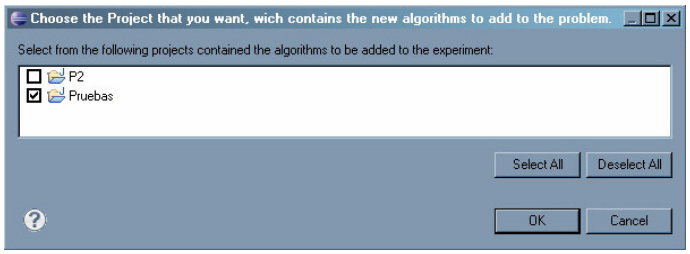


Figura 5.44: Vista de la AlgorithmPage y cómo se añaden nuevos algoritmos al problema.

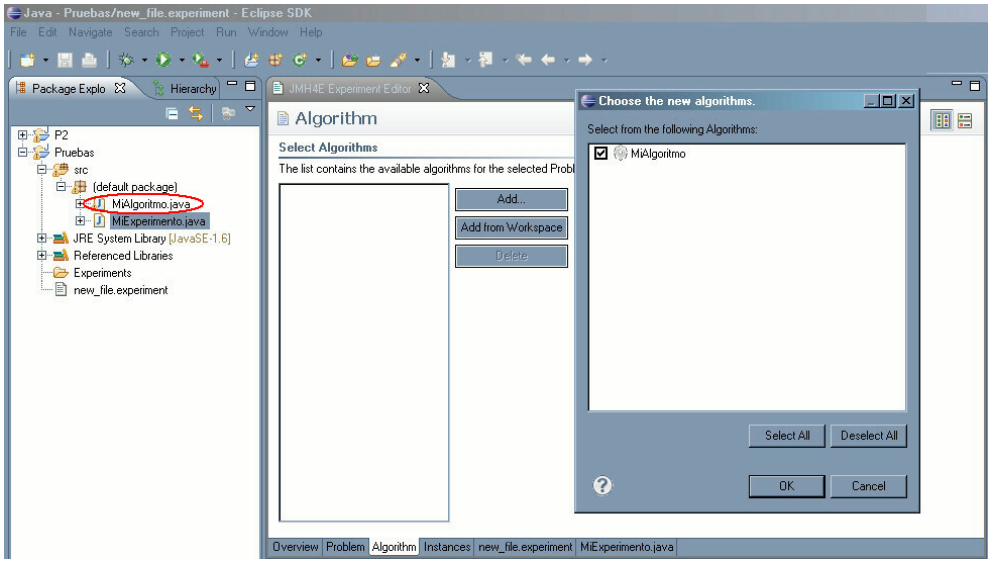


Figura 5.45: Nuevos ApproxMethod.

Hay que tener en cuenta que si se selecciona una ApproxMethod que se encuentra en otro proyecto, éste debe ser referenciado en el *Build Path* del proyecto donde se crea el Experimento, para que dicha clase sea accesible para la compilación y posterior ejecución del experimento [ver el código

fuente 21 del apéndice].

Además, se deben realizar las siguientes tareas:

- Crear el Modelo de pruebas serializado en un fichero *.xml* para su uso por el Plugin, en lugar de crear el DummyModel cada vez que se lanza el Form, ya que si se van añadiendo nuevos algoritmos implementados por el usuario, éstos se deben añadir al modelo y estar disponibles en futuros usos. Además añadir una Preferencia a la PreferencePage de Optsicom con la ruta para encontrar el *modelo.xml*.

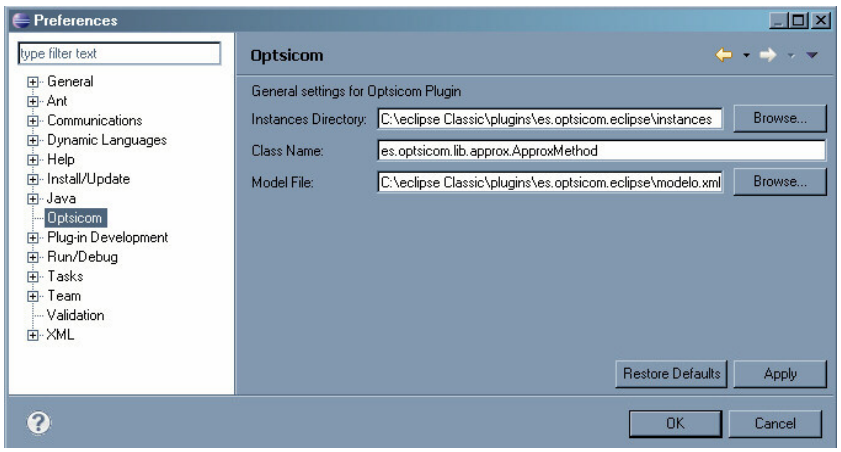


Figura 5.46: Preferencepage de Optsicom.

- Las clases Experiment y MiExperimento deben cambiar para que utilicen una lista de ApproxMethod en lugar de AlgorithmConfiguration, e instancien los ApproxMethod según la información del Algorithm que contienen los AlgorithmConfiguration. [ver el código fuente 22 del apéndice].

- Lanzar la ejecución del experimento: Implementar la funcionalidad de los botones Run y Debug de la página Overview, ejecutando una aplicación programáticamente, [ver el código fuente 23 del apéndice].

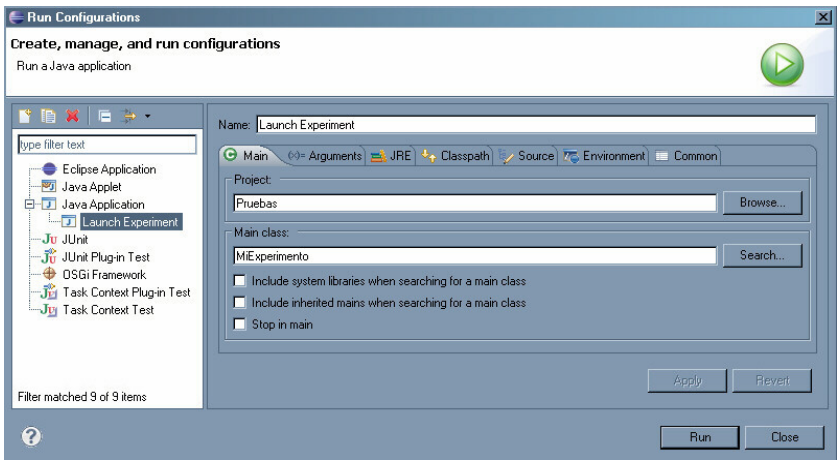


Figura 5.47: Launch.

- Detección y solución de error al serializar listas con JAXB2: se deben tener en cuenta las estructuras List y Map, ya que si se crean con las interfaces Collections y Arrays se crean estáticos y no se pueden serializar, porque serían estructuras a las que no se le pueden añadir ni borrar elementos, y para serializar con JAXB2 necesitamos que las estructuras sean dinámicas.

- Arreglar la clase Activator (depurar y limpiar métodos) ya que al utilizar ahora las Preferences, muchos métodos se simplifican o eliminan, como el método para el cambio de *Instances Location*, ya que ahora tendremos en la página de las preferencias esa funcionalidad, liberando al SimpleNewWizardPage de ello: eliminación del botón para cambiar el directorio de *instances*. [ver el código fuente 24 del apéndice].

- En las pages JavaEditor y XMLEditor, se verifica si se puede cambiar el contenido de dichas páginas y si se guarda: si se puede escribir, pero no se guardan los cambios.

- Mirar la compatibilidad del Plugin en Linux, ya que existen fallos a la hora de indicar la ruta de directorios y otros problemas gráficos como el que no se vea el borde de los cuadros de texto.

5.3.7. 7ª Reunión (11/05/2010)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
67	**7ª Reunión: Remodelación de la librería**	**7 días**	**mar 11/05/10**	**jue 20/05/10**	**51**	**Rocío**
68	Remodelación de las librerías de Optsicom	6 días	mar 11/05/10	mar 18/05/10		Patxi y Mic
69	Depurar errores en el botón "Add from Workspace"	1 día	mar 11/05/10	jue 13/05/10	66	Rocío
70	Filtrar las clases hijas de ApproxMethod si no tiene Constructor sin parámetros	2 días	jue 13/05/10	sáb 15/05/10	69	Rocío
71	Averiguar los parámetros de un ApproxMethod: clase IType	1 día	sáb 15/05/10	dom 16/05/10	70	Rocío
72	**Cambios en la OverviewPage**	**3 días**	**dom 16/05/10**	**jue 20/05/10**	**71**	**Rocío**
73	Secods por Segundos	0 días	dom 16/05/10	dom 16/05/10	71	Rocío
74	Poner CheckBox en los campos Instances Directory y Results Directory	1 día	dom 16/05/10	dom 16/05/10	73	Rocío
75	Al cambiar el Instances Directory se borrarán las instancias del model seleccionado y se sustituyen por las nuevas	2 días	dom 16/05/10	jue 20/05/10	74	Rocío

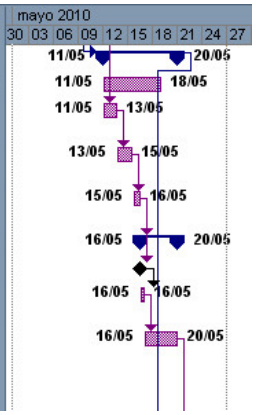


Figura 5.48: Diagrama de Gantt: 7ª Reunión.

A partir de este punto, se debe remodelar la librería de Optsicom para que las hijas de ApproxMethod tengan un constructor sin parámetros para facilitar su instanciación con la clase Class en las clases Experiment y MiExperimento. Además se debe tener en cuenta el hecho de que hay algoritmos que necesitan clases Constructive y/o Improvement, etc... y a veces más de una, por lo que hay que encontrar la forma de averiguar cuántos y qué objetos necesita cada ApproxMethod, porque cada una necesita unos diferentes.

Para esta tarea los tutores prepararán dos librerías: optsicomproblems.jar y optsicomframework.jar que:

- Tengan Experiment como clase base de los experimentos
- Los algoritmos tengan un constructor sin parámetros y se configuren con métodos set.

Modificaciones y Solución de Errores:

- Averiguar los parámetros del ApproxMethod para su inicialización en la clase Experiment: con la interfaz IAnnotable de JDT (implementada

por IType, IMethod, etc...), con la que se sabe si un elemento Java (una clase, un método, un atributo,...) tiene una anotación. En este caso buscamos la anotación @Id implementada en la librería OptsicomFramework, que se añade en los métodos *get* de los atributos de las clases hijas de ApproxMethod para indicar que representa un parámetro de del algoritmo, así como su tipo y su valor por defecto. También se deberá tener en cuenta las anotaciones de las clases *padre*, ya que una clase hereda los atributos y métodos de su clase padre.

- Se deberán filtrar los parámetros que no sean tipos básicos: int, float o boolean.
- También se filtrarán los ApproxMethod que no tengan método *set* en alguno de sus parámetros. [ver el código fuente 28 del apéndice].

- Cambios en la OverviewPage:
  - Cambiar la palabra *Segundos* por *Seconds*.
  - Para el campo *Instances Directory* y *Results Directory*: poner un Checkbox como en el caso del *Time Limit*.
  - Al cambiar el *Instances Directory* se borrarán las instancias del Problem seleccionado y se pondrán en su lugar en el Modelo las nuevas encontradas.

- El botón *Add from Workspace* de la AlgorithmsPage no funciona bien: coge hijas de ApproxMethod de todo el *Workspace* y no sólo del Proyecto seleccionado.

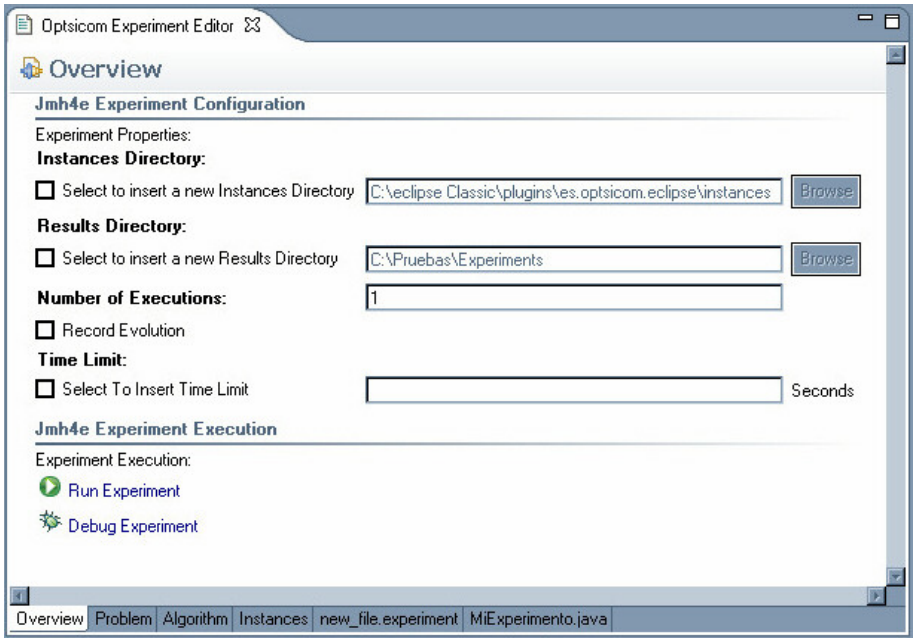


Figura 5.49: Nueva vista de la OverviewPage.

- Filtrar las clases hijas de ApproxMethod: si no tienen constructor sin parámetros la ignoro, en el método getSubClasses() de la clase Hierarchy. Para comprobarlo, el método tieneConstructorSinParametros() utiliza la interfaz IMethod del paquete JDT, [ver el código fuente 25 del apéndice][ver el código fuente 26 del apéndice].

- Utilizar el Log del AbstractUIPlugin para mostrar los errores en la *Error Log* de Eclipse, [ver el código fuente 27 del apéndice]

- Comprobar si se guardan en el Problem seleccionado del Modelo, las instancias que se agreguen desde la OverviewPage.

- Cuando se borra el fichero .experiment se debe saber en el Form: para ello controlar que haya cambios en el *Workspace* para saber si se ha borrado el .experiment.

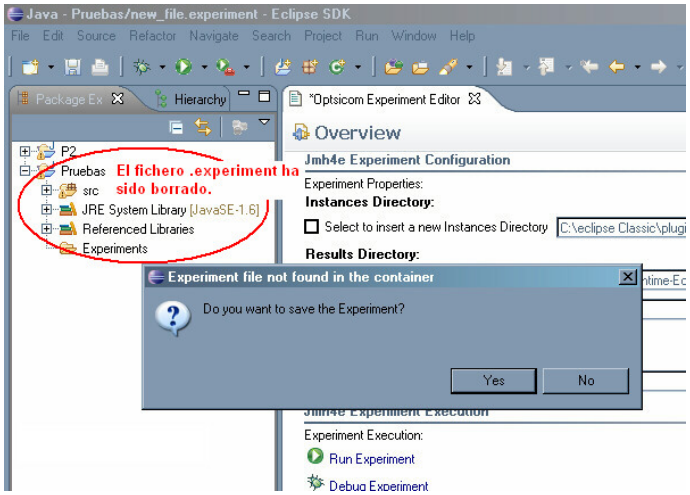


Figura 5.50: Cuando se borra el fichero .experiment se pregunta si guardar los cambios o descartarlos.

5.3.8. 8ª Reunión (20/05/2010)

	Nombre de tarea	Duración	Comienzo	Fin	Pre	Nombres d
76	**8ª Reunión: Últimos detalles**	**11 días**	**jue 20/05/10**	**mar 01/06/10**	**67**	**Rocío**
77	El nombre de las instancias debe contener el directorio padre	0 días	jue 20/05/10	jue 20/05/10	75	Rocío
78	Saber el directorio del plugin en tiempo de ejecución	1 día	jue 20/05/10	vie 21/05/10	77	Rocío
79	**Cambiar la clase MiExperimento**	**1 día**	**vie 21/05/10**	**sáb 22/05/10**	**78**	**Rocío**
80	Debe heredar de ApproxExperimentConf	0 días	vie 21/05/10	vie 21/05/10	78	Rocío
81	Instanciar cada approxMethod en el main	1 día	vie 21/05/10	sáb 22/05/10	80	Rocío
82	Generarse en el SimpleFormEditor cada vez que se llama a doSave	0 días	sáb 22/05/10	sáb 22/05/10	81	Rocío
83	Remodelar el modelo.xml inicial, para que contenga ApproxMethods con constructor sin parámetros	2 días	sáb 22/05/10	lun 24/05/10	82	Rocío
84	Añadir un nuevo botón "Add From Library" a la AlgorithmsPage para añadir ApproxMethod de la librería	1 día	lun 24/05/10	mar 25/05/10	83	Rocío
85	Añadir un botón a la ProblemPage para añadir nuevos Problems al Modelo.	1 día	mar 25/05/10	jue 27/05/10	84	
86	Quitar librerías que sobran	0 días	jue 27/05/10	jue 27/05/10	85	Rocío
87	Update-Site: en Sidelab.es, e instalación del plugin para pruebas	1 día	jue 27/05/10	vie 28/05/10		Rocío
88	Fase de Pruebas y Detección de Errores	11 días	jue 20/05/10	mar 01/06/10		Rocío y Pat
89	Memoria	11 días	jue 20/05/10	mar 01/06/10		Rocío

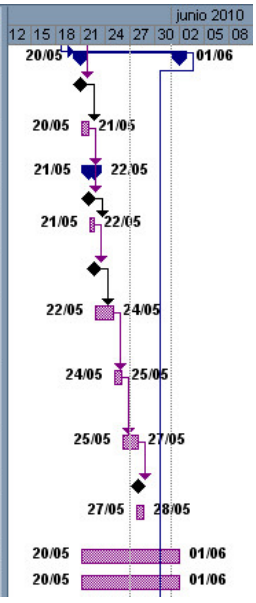


Figura 5.51: Diagrama de Gantt: 8ª Reunión.

Para finalizar el proyecto, se debe crear el *Update-Site* con la última versión del Plugin y subirlo al servidor `http://www.sidelab.es`, perteneciente al grupo Gavab, para que los tutores se lo puedan instalar en Eclipse y realizar pruebas.

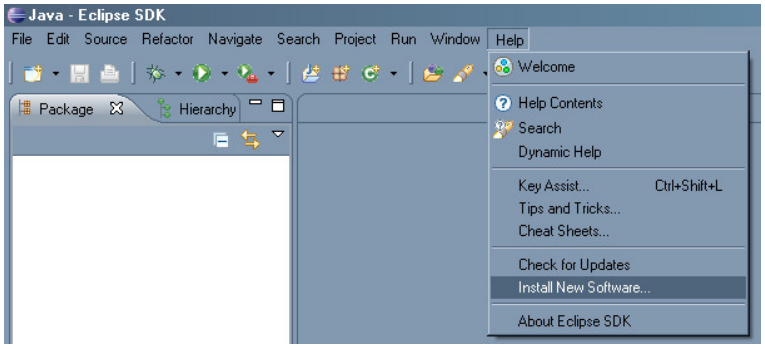


Figura 5.52: Cómo instalar nuevos plugins en Eclipse.

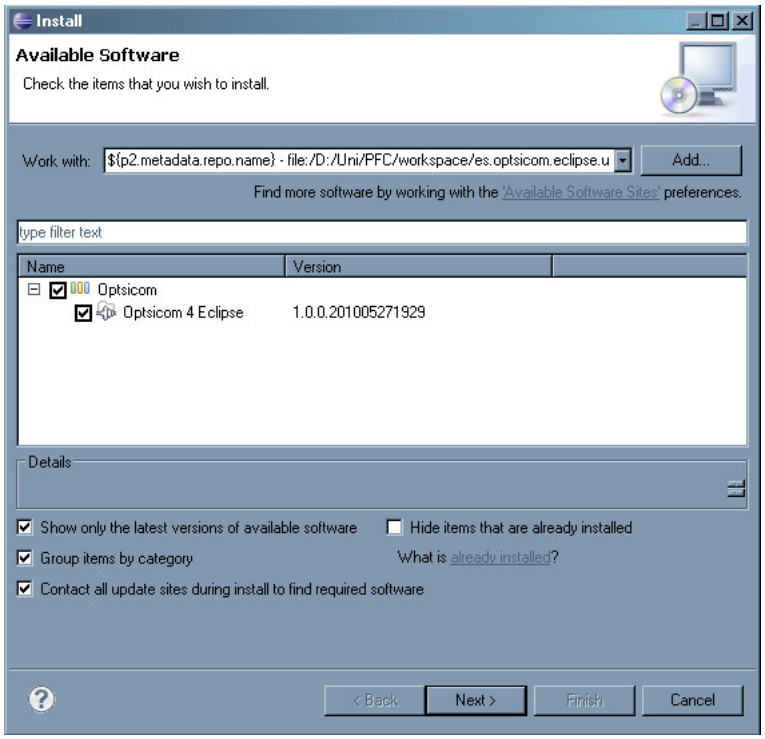


Figura 5.53: Instalar el Plugin desde su *Update-Site*.

Modificaciones y nuevas implementaciones:

- La clase MiExperimento no debe heredar de la clase Experiment del paquete es.optsicom.eclipse, ya que así evitamos que el proyecto donde se crea el Experimento tenga referencias a librerías del plugin. Por tanto ahora el método main a ejecutar como Experimento Optsicom se debe crear cada vez que se haga doSave() en el SimpleFormEditor. La clase MiExperimento hereda ahora de ApproxExperimentConf, y debe contener la creación de los ApproxMethod y sus set, los set de la clase ApproxExperimentConf en el main, así como la llamada al método run().

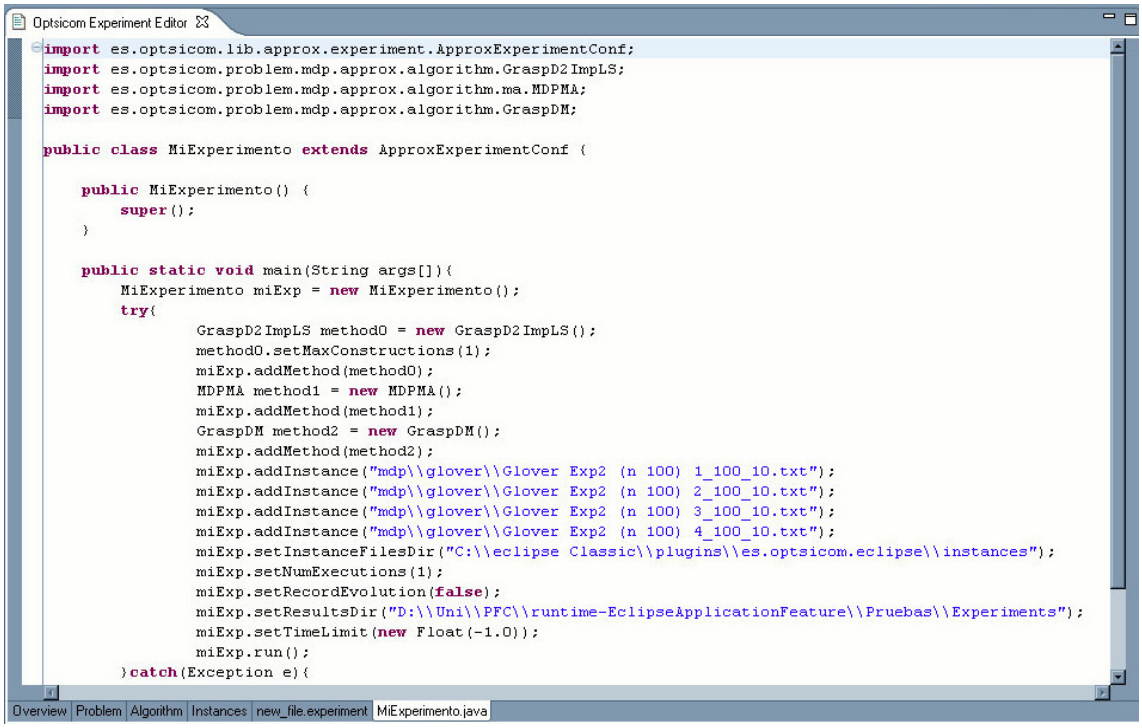


Figura 5.54: Nueva estructura de la clase MiExperimento.

- Añadir un nuevo botón *Add from Library* con el que añadir ApproxMethod desde la librería de Optsicom a los problemas del modelo.

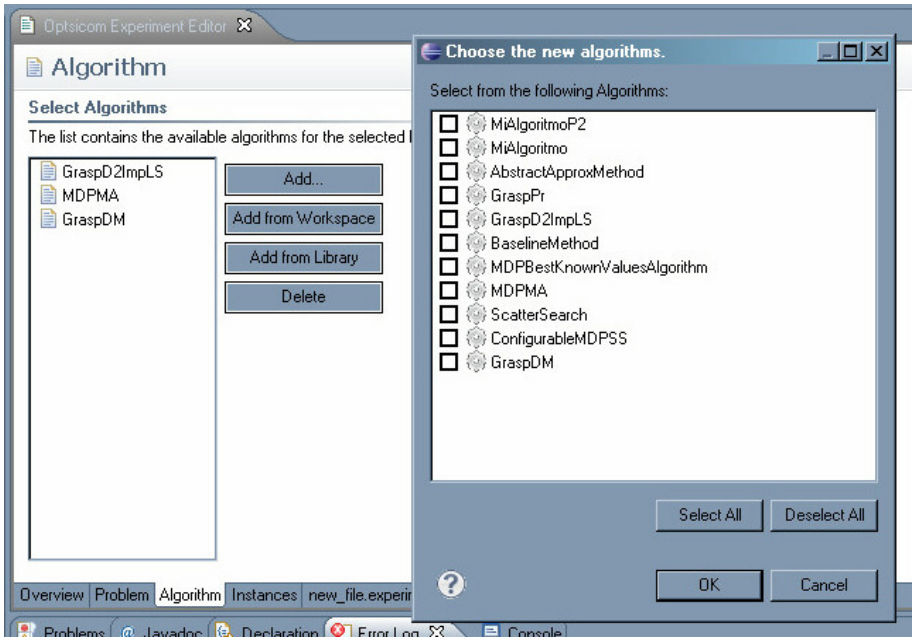


Figura 5.55: Vista de la AlgorithmsPage con el nuevo botón.

- El nombre de las instancias debe llevar el directorio que las contiene, por lo menos hasta llegar al *Instances Dir*.

- Remodelar el *modelo.xml* inicial, para que contenta sólo ApproxMethod con constructor sin parámetros.

- Quitar librerías que sobran: las librerías añadidas para JAXB no son necesarias ya que están incluidas en la JDK 6.0 de Java.

- Se pueden añadir nuevos Problemas al *modelo.xml*, para lo que hay que añadir un nuevo botón *Add New* en la ProblemPage, con el que se abra un FormDialog con el que rellenar los datos del nuevo Problema.

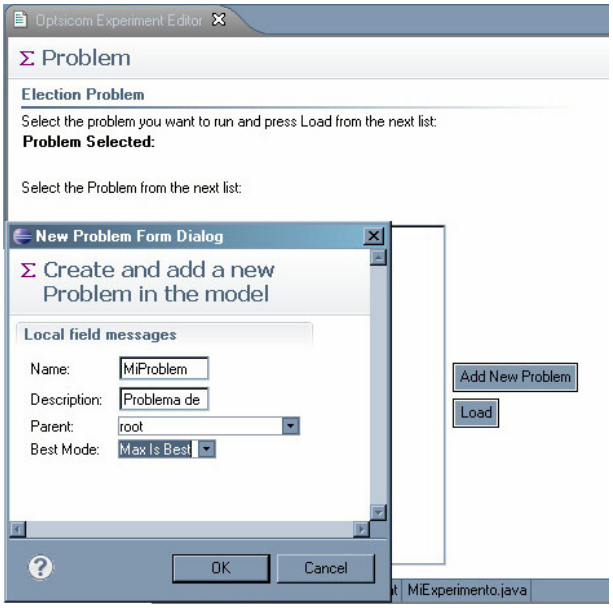


Figura 5.56: Vista del NewProblemFormDialog.

- Ponerle el nombre al grupo al que pertenece el NewWizard del fichero .experiment.

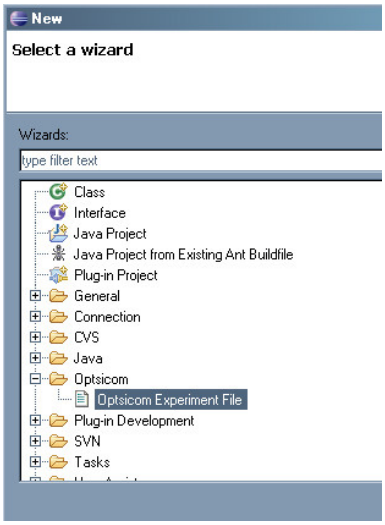


Figura 5.57: Nombre del grupo del NewWizard.

- Fase de Pruebas y Detección de Errores.

5.3.9. 9ª Reunión (01/06/2010)

	Nombre de tarea	Duración	Comienzo	Fin	Pred	Nombres
90	**9ª Reunión: Fase de Pruebas y Finalización de la Memoria**	**8 días**	**mar 01/06/10**	**vie 11/06/10**	**76**	**Rocío**
91	**Solución de Fallos encontrados en la fase de Pruebas:**	**4 días**	**mar 01/06/10**	**dom 06/06/10**		**Rocío**
92	El Plugin no incluye bien las librerias en Linux	0 días	mar 01/06/10	mar 01/06/10		Rocío
93	No se realiza correctamente el cambio de Instancias	3 días	mar 01/06/10	sáb 05/06/10	92	Rocío
94	No se instancian los algoritmos en la clase MiExperimento	1 día	sáb 05/06/10	dom 06/06/10	93	Rocío
95	No se deben incluir los InstanceGroup en MiExperimento	0 días	dom 06/06/10	dom 06/06/10	94	Rocío
96	Incluir la búsqueda del constructor sin parámetros y los parametros en las clases	1 día	dom 06/06/10	lun 07/06/10	95	Rocío
97	Incluir los botones para Ejecución Remota	0 días	lun 07/06/10	lun 07/06/10	96	Rocío
98	Cambiar el nombre de la clase MiExperimento, para que se llame igual que el fichero .experiment	1 día	mar 08/06/10	mié 09/06/10	97	Rocío
99	Cambiar el nombre de la pestaña del Form, para que se puedan diferenciar cuando se crean varios Experimentos.	0 días	mié 09/06/10	mié 09/06/10	98	Rocío
100	Filtrar las clases abstractas que se encuentran cuando se pulsa el botón Add from Library.	0 días	mié 09/06/10	mié 09/06/10	99	Rocío
101	Memoria	7 días	mar 01/06/10	jue 10/06/10		Rocío
102	JavaDoc	2 días	mié 09/06/10	vie 11/06/10	100	

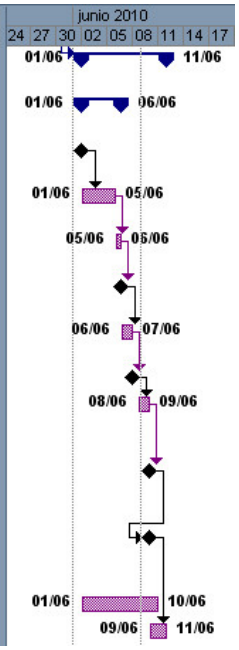


Figura 5.58: Diagrama de Gantt: 9ª Reunión.

Tras la fase de Pruebas realizada por parte del tutor, se han detectado los siguientes errores y aspectos que no se han tenido en cuenta y que deben incluirse:

- El Plugin no incluye correctamente las librerías cuando se ejecuta en el Sistema Operativo Linux. **Motivo y Solución:** Al crear la referencia al fichero *.jar* de la librería se utilizaba el carácter "\" como delimitador del Path, que no es correcto para sistemas operativos basados en Unix, por lo que la mejor solución es utilizar el separador que determine la clase File. [ver el código fuente 29 del apéndice].

- No se realiza correctamente el cambio de las instancias asociadas al Problema al cambiar el *Instances Dir* de la página Overview. **Motivo**

y Solución: El código que cambia las instancias estaba cambiando la estructura que no debía.

- En el código de la clase MiExperimento no se realiza correctamente la instanciación de los ApproxMethod que se incluyen en el Experimento. **Motivo y Solución:** Se estaba instanciando los ApproxMethod con la clase Class para obtener sus métodos y atributos, y se elevaba un NoClassDefFoundException, no continuando con la escritura de ese ApproxMethod en la clase MiExperimento. Se debe utilizar la clase JavaCore para obtener un IType del ApproxMethod y utilizar éste para obtener sus métodos y atributos. [ver el código fuente 30 del apéndice].

- Falta incluir cuando se buscan los parámetros de un IType en la clase Hierarchy, que considere también buscar en la clase *padre*.[ver el código fuente 31 del apéndice].

- Cuando se añaden instancias al Experimento, señalando un directorio de instancias, que puede tener a su vez subdirectorios (que son objetos de la clase InstanceGroup), en la clase MiExperiment se añaden como instancias también los InstanceGroup y sólo deberían añadirse los ficheros. **Motivo y Solución:** como ya se ha comentado, los directorios son objetos InstanceGroup, que heredan de la clase Jmh4eInstance y al ser una lista de Jmh4eInstance lo que se añaden al Experimento, se añaden todos los objetos. Lo que se debe hacer es evitar añadir a la clase MiExperimento los objetos que sean de tipo InstanceGroup.[ver el código fuente 32 del apéndice]

- Cambiarle el nombre a la clase Java MiExperimento, para que se llame como el experimento, ya que si se crean más de un fichero .experiment en el proyecto, se machacaría esta clase. Por ejemplo: si el Experimento se llama `binaryss.experiment` que la clase se llame BinarySSExperiment.java. Se han creado dos métodos estáticos en la clase SampleNewWizard con los que se obtiene el nombre de la clase y su

nombre de fichero en función del nombre del fichero .experiment.[ver el código fuente 33 del apéndice]

- El nombre de la pestaña del formulario debería ser el nombre del fichero .experiment, ya que cuando se tienen varios abiertos es difícil distinguirlos.[ver el código fuente 34 del apéndice]

- Al seleccionar el botón Add from Library se muestran clases que son abstractas (como AbstractApproxMethod o AbstractPathRelinking) que no deberían aparecer, y aparecen clases duplicadas. Para ello se implementan dos métodos en la clase Hierarchy que comprueban los dos aspectos cuando se va a añadir la clase a la lista de clases hijas de ApproxMethod. [ver el código fuente 35 del apéndice]

- Después de crear un Experimento en el proyecto, si se actualiza el Plugin y se procede a crear otro Experimento, se añaden las librerías por duplicado. **Motivo y Solución:** Después de haber actualizado la versión del Plugin, el directorio de instalación del Plugin ha cambiado, ya que contiene el número de versión, y las librerías entonces se agregan por duplicado, al considerarse librerías distintas al estar en diferentes directorios. Se deben borrar las librerías que tengan igual nombre de fichero *.jar* cuando se van a agregar al *Build Path*. [ver el código fuente 36 del apéndice]

- Incluir los botones para la ejecución Remota en la página Overview.

5.4. Estadísticas

En esta sección vamos a realizar un estudio estadístico y estimatorio del proyecto final, utilizando el modelo *COCOMO* [32, 33] para ello.

En primer lugar, señalar que *COCOMO* es un **Modelo matemático de Estimación del Esfuerzo** desarrollado por B. W. Boehm a finales de

los 70 y comienzos de los 80, para calcular la estimación de costes del desarrollo de software.

Para comenzar, se señalan los datos finales del proyecto desarrollado. Éstos son:

- Número de Paquetes: 8.
- Número de Clases: 47.
- Número total aproximado de líneas de código: 7000.

El modelo *COCOMO* determina que el Esfuerzo de Desarrollar un proyecto Software se puede calcular a partir del número de líneas de código implementadas, dos factores a y b, que dependen del tipo de desarrollo empleado en el proyecto, y un multiplicador que depende de 15 atributos.

$$Esfuerzo = a \cdot Miles\ de\ Líneas\ de\ Código^{b} \cdot m(X)$$

Tras considerarlo adecuadamente se determina que el tipo del desarrollo del proyecto empleado es el tipo **Semilibre**, ya que se trata de un esquema intermedio entre el tipo *Orgánico* y el *Rígido*, al tener unas restricciones técnicas y funcionales, pero no de tipo rígido inflexible, ser un problema a resolver único y en el que sí hay experiencia previa.

A continuación, se calculan los modelos Básico e Intermedio de *COCOMO*:

5.4.1. *COCOMO* Básico

Se va a calcular una primera aproximación rápida del esfuerzo. A continuación se muestran los valores de los parámetros necesarios para estos cálculos, para el tipo de desarrollo **Semilibre**.

	Variables			
Tipo	a	b	c	d
Semilibre	3.00	1.12	2.5	0.35

Tabla 5.1: Variables y valores para el tipo **Semilibre** en el *COCOMO* Básico

El factor $\mathcal{K}$ es el número de líneas de código en miles, por lo que en nuestro caso $\mathcal{K} = 7{,}00$.

Cálculos:[1]

- Esfuerzo personal por mes para llevar adelante el proyecto: $\mathcal{MM} = a \cdot \mathcal{K}^b = 3{,}00 \cdot 7{,}00^{1{,}12} = 26{,}52$.
- Tiempo de Desarrollo del Proyecto: $\mathcal{TDEV} = c \cdot \mathcal{MM}^d = 2{,}5 \cdot 26{,}52^{0{,}35} = 7{,}87$ meses.
- Personas necesarias para realizar el proyecto: $CosteH = \frac{\mathcal{MM}}{\mathcal{TDEV}} = \frac{26{,}52}{7{,}87} = 3{,}36$ personas.
- Coste total del proyecto: $CosteM = CosteH \cdot Salario\ medio = 3{,}36 \cdot 1100 = 3696$€ al mes.

5.4.2. *COCOMO* Intermedio

A la estimación del esfuerzo obtenida en el *COCOMO* Básico, se le añaden los 15 atributos cuantificadores del proyecto, que establecer el multiplicador del Esfuerzo.

[1] Éstos son unos cálculos teóricos, que nada tienen que ver con la realidad, ya que el proyecto ha sido desarrollado por una sola persona, durante un tiempo de unos 9 meses aproximadamente y sin remuneración.

	Variables	
Tipo	a	b
Semilibre	3.00	1.12

Tabla 5.2: Variables y valores para el tipo **Semilibre** en el *COCOMO* Intermedio

Cuantificadores del proyecto:

- De Software:
  - RELY: Fiabilidad → Muy bajo: 0,75
  - DATA: Tamaño de la Base de Datos → Bajo: 0,94
  - CPLX: Complejidad → Nominal: 1,00
- De Hardware:
  - TIME: Restricciones de tiempo de ejecución → Nominal: 1,00
  - STOR: Restricciones de memoria virtual → Nominal: 1,00
  - VIRT: Restricciones de la máquina virtual → Nominal: 1,00
  - TURN: Tiempo de respuesta → Nominal: 1,00
- De Personal:
  - ACAP: Capacidad de análisis → Alta: 0,86
  - AEXP: Experiencia en la aplicación → Baja: 1,13
  - PCAP: Calidad de los programadores → Alta: 0,86
  - VEXP: Experiencia en la máquina virtual → Alta: 0,90
  - LEXP: Experiencia en el lenguaje → Alta: 0,95
- De Proyecto:
  - MODP: Técnicas actualizadas de programación → Alta: 0,91

- TOOL: Utilización de herramientas de software → Alta: 0,91
- SCED: Restricciones de tiempo de desarrollo → Alta: 1,04

Por lo tanto ahora el Esfuerzo resulta ser:

$$Esfuerzo = a \cdot \mathcal{K}^b \cdot m(X) = 3{,}00 \cdot 7{,}00^{1{,}12} \cdot (0{,}75 \cdot 0{,}94 \cdot 1{,}00 \cdot 1{,}00 \cdot 1{,}00 \cdot 1{,}00 \cdot 1{,}00 \cdot 0{,}86 \cdot 1{,}13 \cdot 0{,}86 \cdot 0{,}90 \cdot 0{,}95 \cdot 0{,}91 \cdot 0{,}91 \cdot 1{,}04) = 11{,}50.$$

Capítulo 6

Conclusiones y Trabajos Futuros

Al finalizar el desarrollo del proyecto se han alcanzado todos los objetivos propuestos, obteniendo como **Producto Final** un Plugin para el IDE Eclipse con el que se crean Experimentos basados en las librerías de Optsicom. Entrando más en detalle, para cada Experimento creado se asocia un Problema del Modelo de Optsicom, así como unos Algoritmos que ejecutar y unas Instancias de Datos sobre las que se ejecutan los mismo.

Para poder efectuar la ejecución del Experimento se crea una clase Java con método main, en la que se instancian los Algoritmos y las Instancias de Datos, se configuran las características del Experimento con sus métodos set y se lanza su ejecución llamando a su método run. La ejecución local de los Experimentos se realiza creando un *Launch Configuration* de tipo Java.

Por otro lado, para mantener una persistencia y reutilidad de los datos introducidos se serializan los Experimentos a XML, como igualmente ocurre con el Modelo de Problemas.

Además, se permite la creación de nuevos Problemas, añadiéndolos al Modelo de Problemas para futuros usos. También se pueden añadir nuevos Algoritmos creados por el usuario en su proyecto a los problemas del Modelo.

Para la elaboración de este Proyecto, además de completar mis conocimientos sobre el lenguaje de programación Java, he adquirido nuevos conocimientos sobre la arquitectura del IDE Eclipse, así como sobre el API de las librerías de org.eclipse. He aprendido a crear plugins de Eclipse y a contribuir a la arquitectura del IDE. Y también, he aprendido a utilizar el API de JAXB. Todos estos conceptos me han servido para aumentar mis conocimientos y formación, y me serán muy útiles en mi desarrollo profesional.

Como **Trabajos Futuros**, se está valorando el planteamiento de los siguientes proyectos:

- La integración del Plugin **Optsicom para Eclipse** con el **Controlador del Cluster** del grupo Gavab realizado en otro proyecto de fin de carrera, y que permitiría la ejecución Remota de los Experimentos.
- Se puede aumentar la funcionalidad del Plugin al integrarle un módulo de **Análisis de Resultados** de las ejecuciones de los Experimentos.
- Como objetivo futuro, se baraja permitir la creación de proyectos de tipo **Optsicom Projects**, donde sea más sencilla e intuitiva la creación de Experimentos y Algoritmos de Optsicom. Éste tipo de proyecto se crearía como un nuevo Wizard para Eclipse, y estaría basado en un proyecto Java que incluye las librerías de Optsicom y las funcionalidades descritas.

Bibliografía

[1] Javier García de Jalón y otros. — *Aprenda Java como si estuviera en primero* — Universidad de Navarra - Tecnun, 2000.

[2] James Gosling, Bill Joy, Guy Steele and Gilad Bracha — *The Java Language Specification, Third Edition* — Addison-Wesley, 2005. ISBN: 0321246780.

[3] Eric Clayberg and Dan Rubel — *Eclipse Plug-Ins, Third Edition* — Addison-Wesley, 2009. ISBN: 0321553462.

[4] Berthold Daum — *Professional Eclipse 3, for Java Developers* — Wiley, 2004. ISBN: 0470020059.

[5] Erich Gamma and Kent Beck — *Contributing to Eclipse: Principles, Patterns, and Plug-Ins* — Addison-Wesley, 2004. ISBN: 0321205758.

[6] Robert Harris and Rob Warner — *The Definitive Guide to SWT and JFace* — Apress, 2004. ISBN:1590593251.

[7] Matthew Scarpino, Stephen Holder, Stanford Ng and Laurent Mihalkovic — *SWT/JFace In Action: How to design graphical applications with Eclipse 3.0* — Manning, 2005. ISBN: 1932394273.

[8] Jim DÁnjou, Scott Fairbrother and Dan Kehn — *The Java Developer's Guide to Eclipse* — Addison-Wesley, 2003. ISBN: 0321305027.

[9] Página Oficial en Español — *Sun Microsystems* — `http://es.sun.com/`.

[10] Lars Vogel — *Eclipse Extension Points and Extensions Tutorial* — http://www.vogella.de/articles/EclipseExtensionPoint/article.html.

[11] Lars Vogel — *Eclipse Plugins Development Tutorial* — http://www.vogella.de/articles/EclipsePlugIn/article.html.

[12] Wassim Melhem and Dejan Glozic, at IBM Canada Ltd — *PDE Does Plug-ins* — http://www.eclipse.org/articles/Article-PDE-does-plugins/PDE-intro.html.

[13] Chris Aniszczyk, Software Engineer from IBM — *Plug-in development 101* — http://www.ibm.com/developerworks/library/os-eclipse-plugindev1/index.html?ca=dgr-eclipse-1, https://www.ibm.com/developerworks/opensource/library/os-eclipse-plugindev2/?ca=dgr-eclipse-1.

[14] Laurie Williams, Dright Ho and Sarah Smith — *Plug-in Development Environment (PDE) in Eclipse Tutorial* — http://agile.csc.ncsu.edu/SEMaterials/tutorials/plugin_dev/.

[15] Dejan Glozic, at IBM Canada Ltd. — *Eclipse Forms: Rich UI for the Rich Client* — http://www.eclipse.org/articles/Article-Forms/article.html.

[16] Chris Aniszczyk, Software Engineer from IBM — *Revitalize your applications with Eclipse Forms* — http://www.ibm.com/developerworks/opensource/library/os-eclipse-forms/.

[17] Eclipse Plugin Development — *Eclipse Editors Plugin Tutorial* — http://www.eclipsepluginsite.com/editors.html.

[18] Chris Grindstaff, at Applied Reasoning — *How to use the JFace Tree Viewer* — http://www.eclipse.org/articles/Article-TreeViewer/TreeViewerArticle.htm.

[19] Página Oficial de JAXB — *JAXB Reference Implementation* — `https://jaxb.dev.java.net/`.

[20] Ricard Lou Torrijos, at "Programación en castellano" — *El API de JAXB* — `http://www.programacion.com/articulo/el_api_jaxb_115`.

[21] Enrique Rodríguez Lasterra, at "Lasterra Blog" — *Java y XML con JAXB2* — `http://Weblogs.javahispano.org/lasterra/es/entry/java_y_xml_con_jaxb2`.

[22] Using JAXB Data Binding — *Customizing Java-to-XML Schema Mapping Using JAXB Annotations* — `http://download.oracle.com/docs/cd/E12840_01/wls/docs103/Webserv/data_types.html#wp218659`.

[23] Java Platform SE 6 — *Hierarchy For Package javax.xml.bind.annotation* — `http://java.sun.com/javase/6/docs/api/javax/xml/bind/annotation/package-tree.html`.

[24] Darin Wright, at IBM OTI Labs — *Launching Java Applications Programmatically* — `http://www.eclipse.org/articles/Article-Java-launch/launching-java.html`.

[25] Joe Szurszewski, at IBM OTI Labs — *We Have Lift-off: The Launching Framework in Eclipse* — `http://www.eclipse.org/articles/Article-Launch-Framework/launch.html`.

[26] Pawel Leszek — *XML development with Eclipse* — `http://www.ibm.com/developerworks/library/os-ecxml/`.

[27] Lars Vogel — *Eclipse JDT - Abstract Syntax Tree (AST) and the Java Model - Tutorial* — `http://www.vogella.de/articles/EclipseJDT/article.html`.

[28] Vaadin-trac — *JavaCore Code Example* — http://dev.vaadin.com/browser/integration/eclipse/plugins/com.vaadin.integration.eclipse/src/com/vaadin/integration/eclipse/util/VaadinPluginUtil.java?rev=8890.

[29] Eclipse Doc — *Hierarchy For Package org.eclipse.jdt.core* — http://mobius.inria.fr/eclipse-doc/org/eclipse/jdt/core/package-tree.html.

[30] Eclipse Plugin Development — *Getting Started With Preference Pages* — http://eclipsepluginsite.com/preference-pages.html.

[31] Eclipse Community — *FAQ How do I enable The Save and Revert actions?* — http://wiki.eclipse.org/FAQ_How_do_I_enable_the_Save_and_Revert_actions%3F

[32] Wikipedia, la enciclopedia libre — *COCOMO* — http://es.wikipedia.org/wiki/COCOMO

[33] José Javier Dolado Cosín — *El Modelo COCOMO* — http://www.sc.ehu.es/jiwdocoj/mmis/cocomo.htm

Apéndices

Bugs

A continuación se describen los *Bugs* introducidos en la herramienta de seguimiento de errores **BugZilla** a lo largo del desarrollo del proyecto, indicando también su solución.

- **Número de Bug:** 83
  - **Título:** Errores con la clase Algorithm e hijas.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=83`
  - **Fecha:** 13/11/2009.
  - **Descripción:** Para poder probar la página Algorithms del Form, se ha querido implementar un Modelo de prueba con 5 algoritmos con datos de prueba. Se han probado varios tipos de Algoritmos (incluidos en el proyecto de Optsicom) y con ninguno se consigue instanciar. Por ejemplo:

Código Fuente 1 : Instanciar un objeto de la clase Algorithm

```
1   Algorithm a = new MDPTabuSSAlgorithm();
```

No eleva ninguna excepción, sólo que cuando se llama a super("MDP TabuSS"); en el constructor de la clase (en este ejemplo la clase MDPTabuSSAlgorithm), no se realiza dicha llamada

a la clase padre Algorithm, porque al poner trazas éstas no se imprimen.

- **Solucionado:** Sí (14/11/2009).
- **Solución aceptada:** Modificaciones en el fichero *Manifest*:
  - ⋆ En la pestaña *Runtime* se especifican las librerías jmh-problems.jar y jmh4e.jar en la lista *Classpath*. Esto hará que el Plugin no se genere como un jar con jmh-problems.jar dentro, sino que se exporta como una carpeta con los dos jar al mismo nivel (el jar del plugin: jmh4e.jar y el de *problems*).
  - ⋆ En la pestaña *Build* se cambia en *Runtime Information* el nombre de la librería "." por jmh4e.jar y la carpeta a compilar es src. De esta forma se fuerza a generar un jar con el código del plugin denominado jmh4e.jar.
  - ⋆ Además en *Binary Build*, hay que marcar la casilla *lib/jmh-problems.jar*.
  - ⋆ Al final el MANIFEST.MF debe contener la siguiente línea: Bundle-ClassPath: lib/jmh-problems.jar, jmh4e.jar.

- **Número de Bug:** 91
  - **Título:** Problemas con la clase org.eclipse.ui.forms.DetailsPart.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=91`
  - **Fecha:** 18/11/2009.
  - **Descripción:** En la página Algorithms se utiliza la clase AlgorithmsBlock que hereda de MasterDetailsBlock, en la que se crea el cuadro que contiene la lista de algoritmos que se irán agregando, y para cada tipo de objeto de la lista se tiene la página AlgorithmDetailsPage. Esta página debe ser registrada en el Block en el método registerPages(DetailsPart detailsPart), de modo que cada tipo de elemento debe tener su página registrada. En resumen,

la página que se registra es igual para todos las clases de Algoritmos, por la jerarquía de herencia, pero el problema aparece con el registro de esas páginas, porque se debe registrar cada tipo de Algoritmo en este método, y si se crean algoritmos nuevos habría que modificar este método para añadir su página.

Código Fuente 2 : Método registerPages() en la clase AlgorithmsBlock:

```
1  protected void registerPages(DetailsPart detailsPart) {
2          detailsPart.registerPage(MDPTabuSSAlgorithm.class, new
3           AlgorithmDetailsPage());
4          detailsPart.registerPage(BCPEvoPR.class, new
5           AlgorithmDetailsPage());
6          detailsPart.registerPage(BCPFCNS.class, new
7           AlgorithmDetailsPage());
8          detailsPart.registerPage(BCPGraspAlgorithm.class, new
9           AlgorithmDetailsPage());
10 }
```

Cada clase que se muestra en el cuadro, hay que registrarla en este método, y si añado nuevas clases hay que modificarlo, pero eso ocurriría en tiempo de ejecución.

- **Solucionado:** Sí (19/11/2009).
- **Solución intermedia:** Se ha creado un Set de Objetos en la clase Algorithm:

Código Fuente 3 : Set de las clases Hijas de Algorithm:

```
1  private static Set<Object> hijas = new LinkedHashSet<Object>();
```

, donde a medida q se creen las instancias de las clases hijas se guardará su Objeto, para luego en la clase AlgorithmsBlock hacer el registro de su DetailsPage:

Código Fuente 4 : Solución del método registerPages de la clase AlgorithmsBlock:

```
1  protected void registerPages(DetailsPart detailsPart){
2     for (Object o : Algorithm.getHijas()){
```

```
3         detailsPart.registerPage(o, new AlgorithmDetailsPage());
4     }
5 }
```

De este modo se pueden registrar las páginas en tiempo de ejecución con sus correspondientes .class sin problemas.

- **Solución Final:** Finalmente no se utiliza el Set de las clases hijas de Algorithm, ya que lo que se muestra en la AlgorithmBlock son instancias de la clase AlgorithmConfiguration, y no se tiene el problema expuesto en este *Bug*.

- **Número de Bug:** 92

  - **Título:** Problemas con el CheckboxTreeViewer de la Instaces-Page.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=92`
  - **Fecha:** 23/11/2009.
  - **Descripción:** En la pestaña de las instancias, se ha reutilizado el código correspondiente de Optsicom para realizar un árbol que debe mostrar las instancias de prueba de la clase Model pero luego cuando se ejecuta el Plugin no las muestra en la tabla de la InstancePage.
  - **Solucionado:** Sí (24/11/2009).
  - **Solución aceptada:** El problema es que se utilizaba un objeto Table y se añadía al client (que es un Composite) y luego se creaba un CheckboxTreeViewer y se enganchaba al Table, pero el CheckboxTreeViewer ya tiene su propio Table y él se encarga de gestionarlo.
    Sólo hay que quitar la parte del Table y enganchar el viewer al Composite client.

Código Fuente 5 : Creación del CheckboxTreeViewer

```
1  //  La parte de creación del Table no es necesaria:
2  //  Table t = toolkit.createTable(client, SWT.NULL);
3  //  GridData gd = new GridData(GridData.FILL_BOTH);
4  //  gd.heightHint = 400;
5  //  gd.widthHint = 400;
6  //  t.setLayoutData(gd);
7      toolkit.paintBordersFor(client);
8      section.setClient(client);
9      final SectionPart spart = new SectionPart(section);
10     managedForm.addPart(spart);
11     this.viewer = new CheckboxTreeViewer(client);
```

- **Número de Bug:** 107
  - **Título:** Errores con la clase Algorithm e hijas.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=107`
  - **Fecha:** 11/01/2010.
  - **Descripción:** Tres problemas:
    - ⋆ Se ignore cómo hacer que cuando se modifica algún parámetro en las *pages* (además de guardarse los cambios en el objeto Jmh4eExperimentConfiguration contenido en SimpleFormEditorInput) se ponga el asterisco "*.[en] la cabecera del Form, habilitando la opción de guardar (que habilita la invocación del método doSave del SimpleFormEditor).
    - ⋆ Cuando se invoque al método doSave (y al doSaveAs) del SimpleFormEditor, se debe llamar al método de la clase Marshalling.marshall(...), para que se serialice el objeto Jmh4eExperimentConfiguration al fichero .experiment. Por otro lado, se dispone de una página en el Form que es un org.eclipse.ui.editors.text.TextEditor para mostrar el contenido del fichero .experiment en XML. ¿Cómo se le indica que se actualice para que se vea el contenido nuevo del .experiment?

- ⋆ En la página Overview todavía no están las modificaciones controladas, pero en la página Algorithm sí, y no se guardan los cambios, siendo estos en un HashMap contenido en el SimpleFormEditorInput.

- **Solucionado:** Sí (17/01/2010).
- **Solución aceptada:**
  - ⋆ Debería haber un atributo dirty en el FormEditorInput. Poniéndolo a *true* indicas que hay cambios sin guardar. [31]

Código Fuente 6 : Atributo dirty en el SimpleFormEditor

```
1  public class SimpleFormEditor extends FormEditor{
2     ...
3     protected boolean dirty = false;
4     ...
5      public void setDirty(boolean value) {
6         this.dirty = value;
7         firePropertyChange(PROP_DIRTY);
8      }
9     ...
10 }
```

  - ⋆ Los editores detectan cuándo ha cambiado el fichero físico del disco y se actualizan automáticamente.
  - ⋆ El problema era que mostraba la lista algConfiguration.getAlgorithm().getDefaultParameters() y modificaba la algConfiguration.getParamValues() (siendo algConfiguration un objeto de la clase AlgorithmConfiguration). Entonces, obviamente no se visualizaban los cambios. Se debe mostrar y modificar siempre sobre algConfiguration.getParamValues().

- **Número de Bug:** 111
  - **Título:** Problemas en la Serialización.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=111`

- **Fecha:** 19/01/2010.
- **Descripción:** Se tienen problemas al serializar el objeto de la clase Jmh4eExperimentConfiguration debido a que:
  - La lista private List<AlgorithmConfiguration> algorithms; no se serializa correctamente, omitiéndose los atributos de cada elemento de la clase AlgorithmConfiguration contenido en la lista (excepto el Map<String, Object> que contiene que se corresponde con un Map<NombreParametro,Valor>), así como el elemento Algorithm que contiene cada AlgorithmConfiguration no se serializa.
  - El atributo private List<Jmh4eInstance> instances; directamente no aparece serializado en el fichero.
- **Solucionado:** Sí (01/02/2010).
- **Solución aceptada:** Se deben repasar las anotaciones @Xml para la correcta serialización de los objetos, añadiendo la anotación @XmlTransient donde corresponda para facilitar a JAXB la serialización de las objetos más complejos. Además, se deben crear las estructuras de datos a serializar (List, Map) de modo dinámico, sin utilizar las Interfaces Collections, ni Arrays.

- **Número de Bug:** 149
  - **Título:** Unmarshall con Jaxb.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=149`
  - **Fecha:** 21/03/2010.
  - **Descripción:** Se detectan problemas al deserializar un objeto de la clase Algorithm con JAXB, al realizar el unmarshalling de un objeto Experiment que contiene clases hijas de la clase Algorithm. El problema es que al realizar el unmarshall del código XML a objetos Java, al ser Algorithm una clase abstracta, no se puede

deserializar un objeto de dicha clase sin saber las clases hijas que tiene. Existe una anotación @XmlSeeAlso donde indicarle todas las clases hijas, pero si aumentan en tiempo de ejecución no sirve.

- **Solucionado:** Sí (03/05/2010).
- **Solución aceptada:** La solución es quitarle la característica abstracta a la clase Algorithm: ya no tendrá hijos, ni el método abstracto getJMHMethod, y se le añadirán los atributos necesarios para sustituir su forma abstracta, como el nombre del ApproxMethod con el que se corresponde.
  De este modo se solucionará también el problema a la hora de crear nuevos algoritmos ApproxMethod, ya que se debe crear un Algorithm que lo represente para su uso con el Form.
- De este modo desaparecerán todas las clases del paquete problems, pero antes se debe crear el Modelo de Pruebas y serializarlo en un fichero .xml para su uso por el plugin, en lugar de crear el DummyModel. Además, se debe añadir para el Modelo serializado una preference en el Plugin en la que se especifique el fichero que contiene el modelo.

- **Número de Bug:** 150
  - **Título:** JavaCore.
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=150`
  - **Fecha:** 30/04/2010.
  - **Descripción:** Se han detectado problemas al calcular la jerarquía de herencia con JavaCore de la clase ApproxMethod con el siguiente código:

Código Fuente 7 : Obtener la jerarquía de herencia de una clase.

```
1 try{
```

```
2            IProject project = ResourcesPlugin.getWorkspace()
3            .getRoot().getProject();
4            IType [] clases = Hierarchy.getSubClasses(project,
5                    "es.gavab.jmh.approx.ApproxMethod",true,new
6               NullProgressMonitor());
7            for (IType t : clases){
8                System.out.println(t.getClass().toString()+
9               "␣-␣"+t.toString());
10           }
11       }catch(Exception e){
12           e.printStackTrace();
13       }
```

Donde Hierachy.getSubClasses, es el método que calcula todas las clases hijas. ¿Cuándo o Dónde se debe ejecutar dicho código en el Plugin para obtener la referencia al proyecto y a la clase es.gavab.jmh.approx.ApproxMethod? Se obtiene un error al ejecutar un método main en la clase Hierachy:

Código Fuente 8 : Excepción al ejecutar el main de la clase Hierarchy

```
1  java.lang.IllegalStateException: Workspace is closed.
2     at org.eclipse.core.resources.ResourcesPlugin
3           .getWorkspace(ResourcesPlugin.java:340)
4     at es.gavab.jmh4e.jaxb.Hierarchy.main(Hierarchy.java:29)}
5     \ldots
```

Y si se pone el mismo código en, por ejemplo, el doSave de la clase SimpleFormEditor, se obtiene un NullPointerException ya que el parámetro project es null (no lo obtiene correctamente), y por tanto la lista de clases hijas la devuelve vacía.

- **Solucionado:** Sí (09/04/2010).
- **Solución aceptada:** No se pueden utilizar las clases del *Framework* de Eclipse si no hay una instancia de Eclipse arrancada, eso es lo que quiere decir el error. Por tanto no se puede hacer ese main. Se debería utilizar JDT en el momento en que se va a mostrar el cuadro de diálogo para seleccionar los algoritmos, concretamente en la clase AlgorithmsBlock, en el addSelectionLis-

tener del botón add:

Código Fuente 9 : Obtener la jerarquía de herencia de una clase del modo idóneo.

```
1  add.addSelectionListener(new SelectionAdapter() {
2      /*
3      * Con este botón, se abrirá una ventana donde
4     * aparecerá la lista completa de
5      * algoritmos que contiene el modelo,
6     * y cuando seleccione uno y le dé a un botón OK
7      * se añadirá este a la lista del
8     * "TableViewer viewer" de abajo...
9      */
10     public void widgetSelected(SelectionEvent e) {
11     IFile file = ResourceUtil.getFile(page.getEditorInput());
12     IProject project = file.getProject();
13     try {
14       IType[] subclassesFound = Hierarchy.getSubClasses(project,
15          "es.gavab.jmh.ApproxMethod", true,
16          new NullProgressMonitor());
17        for(IType subclass : subclassesFound) {
18          System.out.println(subclass.getFullyQualifiedName());
19        }
20     } catch (JavaModelException e1) {
21       e1.printStackTrace();
22     }
23     ...
```

Donde el EditorInput de la page el que es el fichero .xml. El EditorInput que representa el código XML está enganchado con el fichero, y a partir del fichero se puede obtener el proyecto.
Hay que comprobar que saca la jerarquía utilizando también los proyectos referenciados, es decir, si se crea el fichero de configuración del experimento en el ProyectoB y éste depende del ProyectoA (y por tanto esta dependencia está en el Build Path de alguna manera) se debe comprobar que si tengo una clase A en ProyectoA que hereda de ApproxMethod, ésta también se devuelve al consultar la jerarquía.
También habría que probar qué pasa si ApproxMethod está dentro de un jar referenciado por el proyecto: ¿se sigue resolviendo

la jerarquía de herencia?

- **Número de Bug:** 156
  - **Título:** Dudas sobre ApproxMethod, JMH, Algorithm,...
  - **Url:** `http://labs.gavab.es/cgi-bin/bugzilla3/show_bug.cgi?id=156`
  - **Fecha:** 03/05/2010.
  - **Descripción:** Haciendo el Modelo Inicial utilizando la clase Algorithm como no abstracta (no usando objetos de sus clases hijas) y poniéndole como atributos el nombre de la clase ApproxMethod, se detecta una complejidad elevada al tener clases como BinarySSAlgorithm o BCPEvoPR, que tienen una creación de ApproxMethod complicada. Por ejemplo, para la clase es.gavab.jmh4e.problem.BCPEvoPR, tenemos que su ApproxMethod se instancia así:

Código Fuente 10 : Ejemplo de instanciación de un ApproxMethod

```
1  EvoPathRelinking<BMCPSolution, BMCPInstance> evoPR =
2     new EvoPathRelinking<BMCPSolution,BMCPInstance>(
3     (Integer)parameterValues.get(GLOBAL_ITER),
4     (Integer)parameterValues.get(LOCAL_ITER),
5     new FinalSimpleGraspBandwidthMarti(),
6     new ColoringOrderInsertLS(0.5, true),
7     new GreedyPR(),
8     new ColoringOrderDistanceCalc(),
9     (Integer)parameterValues.get(THRESHOLD));
```

Partiendo desde la base de que la instanciación del ApproxMethod sería sin parámetros pero, por ejemplo en este caso, se debe saber qué clases necesita de los paquetes es.gavab.jmh.problem.bcp.approx-.constructive, es.gavab.jmh.problem.bcp.approx.improvement, etc... necesita, y cuántas clases, porque cada ApproxMethod necesita unas diferentes...
¿Ese tipo de aspectos se van a implementar en la librería de Optsicom o lo debe tener en cuenta el plugin? En ese caso, ¿cómo se

podría implementar utilizando sólo la clase Algorithm e instanciando ahora en MiExperimento.java (la clase main de ejecución) los ApproxMethod?

- **Solucionado:** Sí (04/05/2010).
- **Solución aceptada:** Este tipo de aspectos se tendrán en cuenta en la librería de Optsicom. En el futuro los algoritmos no se configurarán a través del constructor, sino que el constructor estará vacío, y se configurarán mediante métodos set.
  En cuanto a las clases de los parámetros, no se van a considerar estos casos, más complicados, en los que los algoritmos se configuran con clases que heredan de Constructive, Improvement, etc. Sólo se van a considerar los tipos de parámetros básicos.
  Lo ideal sería que si algoritmo depende de una clase Constructive, el plugin te ofreciera todas las clases hijas de Constructive y el usuario escogiera la que quiere utilizar con ese algoritmo. Pero esto no lo vamos a contemplar.

Código Fuente

En esta sección se incluyen los Código Fuente más relevantes de la aplicación.

Marshalling.marshall()

Código Fuente 11 : Método para serializar un objeto Java a XML.

```
1 public static void marshall(Object obj, IFile file)
2       throws JAXBException, CoreException, IOException{
3    StringWriter sw = new StringWriter();
4    JAXBContext context = JAXBContext.newInstance(obj.getClass());
5    Marshaller m = context.createMarshaller();
6    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
7     m.marshal(obj, sw);
8     InputStream stream = new ByteArrayInputStream(sw.getBuffer()
```

```
9       .toString().getBytes());
10     if (file.exists()) {
11       file.setContents(stream, true, true, null);
12     } else {
13       file.create(stream, true, null);
14     }
15     stream.close();
16 }
```

Obtener el IFile

Código Fuente 12 : Obtener el IFile del SimpleFormEditor con ResourceUtil

```
1 IFile file = ResourceUtil.getFile(this.getEditorInput());
```

Activator.getPluginLocation()

Código Fuente 13 : Obtener la carpeta de instalación del plugin.

```
1  public static String getPluginLocation(){
2     File f = getPluginsFolder();
3     String [] files = f.list();
4     ArrayList<String> plugins = new ArrayList<String>();
5     for(String s : files){
6        if (s.contains(PLUGIN_ID)){
7           plugins.add(s);
8        }
9     }
10    String plugin="";
11    if(plugins.size()>1){
12       long version=-1;
13       for(String s : plugins){
14          try{
15             StringTokenizer st = new StringTokenizer(s,"_");
16             String token="";
17             while(st.hasMoreTokens()){
18                token = st.nextToken();
19             }
20             st = new StringTokenizer(token,".");
21             while(st.hasMoreTokens()){
22                token = st.nextToken();
23             }
24             long v =Long.parseLong(token);
25             if(v>version){
```

```
26                version=v;
27                plugin=s;
28             }
29          }catch(Exception e){
30             Activator.log(new Status(IStatus.ERROR,Activator.PLUGIN_ID,
31                "Excepción calculando la ruta del Plugin",e));
32          }
33       }
34    }else{
35       plugin=plugins.get(0);
36    }
37    f = new File(f,plugin);
38    return f.getAbsolutePath();
39 }
```

Hierachy.getSubClasses()

Código Fuente 14 : Obtener la jerarquía de herencia de una clase.

```
1  public static List<IType> getSubClasses(IProject project,
2     String superClass,boolean allSubTypes, boolean projTypes,
3     IProgressMonitor monitor) throws JavaModelException {
4          IJavaProject javaProject = JavaCore.create(project);
5          if (javaProject == null) {
6              return Collections.emptyList();
7          }
8          IType superType = javaProject.findType(superClass);
9          ITypeHierarchy newTypeHierarchy = superType.newTypeHierarchy(
10          monitor);
11
12         IType[] subTypes;
13         if (allSubTypes) {
14             subTypes = newTypeHierarchy.getAllSubtypes(superType);
15         } else {
16             subTypes = newTypeHierarchy.getSubclasses(superType);
17         }
18         List<IType> types = new ArrayList<IType>();
19
20         for (IType subclass : subTypes) {
21          if(!projTypes){
22             types.add(subclass);
23          }else{
24             if (subclass.isResolved() && !subclass.isBinary()) {
25                if(project.contains(subclass.getSchedulingRule())){
26                   types.add(subclass);
27                }
```

```
28              }
29          }
30         }
31         return types;
32 }
```

Obtener las clases hijas de **ApproxMethod**

Código Fuente 15 : Cómo llamar al método getSubClasses.

```
1    IFile file = ResourceUtil.getFile(page.getEditor().getEditorInput());
2    IProject project = file.getProject();
3    // Calcular las clases hijas de ApproxMethod:
4    List<IType> clases = null;
5    try{
6       clases = Hierarchy.getSubClasses(project,
7          Activator.getDefault().getPreferenceStore().
8          getString(PreferenceConstants.P_CLASSNAME),
9          true,false,new NullProgressMonitor());
10   }catch(Exception exc){
11      MessageDialog.openWarning(page.getSite().getShell(),
12         "No Algorithms founded","New algorithms not founded.");
13      Activator.log(new Status(IStatus.ERROR,Activator.PLUGIN_ID,
14         "Unexpected Error",exc));
15   }
```

Anotación @XmlSeeAlso

Código Fuente 16 : Anotación @XmlSeeAlso.

```
1  @XmlSeeAlso({MDPTabuSSAlgorithm.class, BCPAlgorithm.class,
2     BCPEvoPR.class, BCPFCNS.class,
3     BCPGraspAlgorithm.class,
4     BCPLim2003.class,
5     BinarySSAlgorithm.class,
6      KLDLSAlgorithm.class,
7      KLDv2LSAlgorithm.class})
8  public abstract class Algorithm {
9           ...
10 }
```

Obtener el valor de una preferencia del plugin

Código Fuente 17 : Obtener el valor de una preferencia.

```
1 this.getPreferenceStore().getString("className");
```

Establecer el valor por defecto de una preferencia del plugin

Código Fuente 18 : Establecer el valor por defecto de una preferencia.

```
1 Activator.getPreferenceStore().setDefault(PreferenceConstants.P_CLASSNAME,
2    "es.optsicom.lib.approx.ApproxMethod");
3 Activator.getPreferenceStore().setValue(PreferenceConstants.P_CLASSNAME,
4    "es.optsicom.lib.approx.ApproxMethod");
```

Añadir el *Extension Point* de la Preference Page

Código Fuente 19 : Contribuir a la *Preferences Page de Eclipse*.

```
1    <page class="es.optsicom.eclipse.preferences.SamplePreferencePage"
2       id="es.optsicom.eclipse.preferences.SamplePreferencePage"
3       name="Optsicom">
4    </page>
5 </extension>
6 <extension point="org.eclipse.core.runtime.preferences">
7    <initializer
8       class="es.optsicom.eclipse.preferences.PreferenceInitializer">
9    </initializer>
10 </extension>
```

Añadir librerías a un proyecto programáticamente

Código Fuente 20 : Añadir librerias a un proyecto en tiempo de ejecución.

```
1 try{
2    IProject project = file.getProject();
3    IJavaProject jproject = JavaCore.create(project);
4    IClasspathEntry[] classpath = jproject.getRawClasspath();
5    List<IClasspathEntry> entries = new ArrayList<IClasspathEntry>();
6    for(IClasspathEntry e : classpath){
```

```
7       entries.add(e);
8    }
9    IClasspathEntry jmh = JavaCore.newLibraryEntry(new Path(Activator
10      .getLibLocation()+"\\optsicomframework.jar"),null,null,true);
11   if(!entries.contains(jmh))
12      entries.add(jmh);
13   IClasspathEntry jmh2 = JavaCore.newLibraryEntry(new Path(Activator
14      .getLibLocation()+"\\optsicomproblems.jar"),null,null,true);
15   if(!entries.contains(jmh2))
16      entries.add(jmh2);
17   jproject.setRawClasspath(entries.toArray(new IClasspathEntry[entries
18      .size()]), monitor);
19}catch(Exception e){
20   e.printStackTrace();
21   Activator.log(new Status(IStatus.ERROR,Activator.PLUGIN_ID,
22      "El␣Proyecto␣Contenedor␣seleccionado␣no␣es␣un␣proyecto␣Java",e));
23}
```

Añadir la referencia de un proyecto al *Build Path* de otro

Código Fuente 21 : Añadir la referencia a un Proyecto en el *Build Path*.

```
1    /*
2    * Si se referencian clases de otro proyecto, éste se
3    * debe añadir al Build Path.
4    */
5    IJavaProject jproject = JavaCore.create(project);
6    IClasspathEntry[] classpath = jproject.getRawClasspath();
7    List<IClasspathEntry> entries = new ArrayList<IClasspathEntry>();
8    for(IClasspathEntry entry : classpath){
9       entries.add(entry);
10   }
11   IClasspathEntry otherProj = JavaCore.newProjectEntry(((IProject)proj)
12      .getFullPath());
13   if(!entries.contains(otherProj))
14      entries.add(otherProj);
15   jproject.setRawClasspath(entries.toArray(new IClasspathEntry[
16      entries.size()]),new NullProgressMonitor());
17}
```

Instanciar un objeto **ApproxMethod** con la clase **Class**

Código Fuente 22 : Instanciar un objeto de ApproxMethod.

```
ApproxMethod alg = (ApproxMethod) clase.newInstance();
```

Launch

Código Fuente 23 : Lanzar la ejecución del experimento.

```
ILaunchManager manager = DebugPlugin.getDefault().getLaunchManager();
ILaunchConfigurationType type = manager.getLaunchConfigurationType(
   IJavaLaunchConfigurationConstants.ID_JAVA_APPLICATION);
ILaunchConfiguration[] configurations = manager
      .getLaunchConfigurations(type);
for (int i = 0; i < configurations.length; i++) {
   ILaunchConfiguration configuration = configurations[i];
   if (configuration.getName().equals("Launch Experiment")) {
      configuration.delete();
      break;
   }
}
ILaunchConfigurationWorkingCopy workingCopy = type.newInstance(null,
   "Launch Experiment");
workingCopy.setAttribute("attributeName","value");
workingCopy.setAttribute(IJavaLaunchConfigurationConstants
   .ATTR_PROJECT_NAME, project.getName());
workingCopy.setAttribute(IJavaLaunchConfigurationConstants
   .ATTR_WORKING_DIRECTORY, (String) null);
workingCopy.setAttribute(IJavaLaunchConfigurationConstants
   .ATTR_MAIN_TYPE_NAME,"MiExperimento");
ILaunchConfiguration config = workingCopy.doSave();
config.launch(ILaunchManager.RUN_MODE,null);
```

Utilizar las Preferencias en el Activator

Código Fuente 24 : Ejemplo de utilización de las Preferences en la clase Activator.

```
public static File getInstancesLocation() {
   File model = new File(plugin.getPreferenceStore().getString(
      PreferenceConstants.P_INSTANCES_PATH));
   return model;
}
```

Comprobar si un **IType** tiene Constructor sin Parámetros en la clase **Hierarchy**

Código Fuente 25 : Cambios en el método getSubClasses() de la clase Hierarchy.

```
for (IType subclass : subTypes) {
        if(!projTypes){
           if(tieneConstructorSinParametros(subclass)){
              types.add(subclass);
           }
        }else{
           /*
            * Filtra las clases encontradas:
            * Si pertenecen al projecto seleccionado y si
            * tienen un constructor sin parámetros.
            */
           if (subclass.isResolved() && !subclass.isBinary()) {
              if(project.contains(subclass.getSchedulingRule())
                    && tieneConstructorSinParametros(subclass)){
                 types.add(subclass);
              }
           }
     }
}
```

Código Fuente 26 : Método que busca el constructor sin parámetros utilizando la interfaz IMethod.

```
public static boolean tieneConstructorSinParametros(IType type){
   try{
      IMethod [] members = type.getMethods();
      boolean tiene = false;
      for(IMethod member : members){
         if(member.isConstructor()){
            String [] parameters = member.getParameterNames();
            if(parameters.length==0){
               tiene = true;
            }
         }
      }
      return tiene;
   }catch(JavaModelException jme){
      return false;
   }
}
```

Uso del Log de Activator

Código Fuente 27 : Uso del log en la clase Activator.

```
1 public static void log(IStatus status){
2    getDefault().getLog().log(status);
3 }
```

Buscar los parámetros de un ApproxMethod

Código Fuente 28 : Búsqueda de los parámetros de un IType en la clase Hierarchy.

```
1  public static ArrayList<Parameter> buscaParametros(IType type){
2     try{
3        ArrayList<Parameter> parameters = new ArrayList<Parameter>();
4        IMethod [] methods = type.getMethods();
5        for(int i=0;i<methods.length;i++){
6           IAnnotation annotation = methods[i].getAnnotation("Id");
7           //Si es un "get" y tiene anotación @Id:
8           if(methods[i].getElementName().startsWith("get") &&
9              annotation!=null){
10             String paramName = methods[i].getElementName()
11                .substring(3);
12             String setName = "set".concat(paramName);
13             //Si hay un "set" que se corresponda con este "get":
14             if(contains(methods,setName)){
15                String ini = paramName.substring(0,1);
16                ini = ini.toLowerCase(Locale.getDefault());
17                paramName = ini.concat(paramName.substring(1));
18
19                String tipoName = methods[i].getReturnType();
20                ParameterType tipo = ParameterType.toParameterType(
21                   tipoName);
22                //Si es un tipo básico:
23                if(tipo!=null){
24                   IMemberValuePair [] pairs = annotation
25                      .getMemberValuePairs();
26                   Object value = getValue(pairs,"value",tipo);
27                   //Si el valor es del tipo correcto.
28                   if(value!=null){
29                      try{
30                         Parameter param = new Parameter(
31                            paramName,paramName,tipo,
32                            new Description(paramName+
33                            "parameter."),value);
```

```
                        param.setMinValue(getValue(pairs,
                           "minValue", tipo));
                        param.setMaxValue(getValue(pairs,
                           "maxValue", tipo));
                        parameters.add(param);
                     }catch(RuntimeException re){}
                  }
               }
            }
         }
      }
      return parameters;
   }catch(JavaModelException jme){
      return null;
   }catch(Exception e){
      return null;
   }

}

public static Object getValue(IMemberValuePair [] pairs,
      String memberName,ParameterType tipo){
   for(int j=0;j<pairs.length;j++){
      if(pairs[j].getMemberName().equals(memberName)){
         return tipo.toObjectType((String)pairs[j].getValue());
      }
   }
   return null;
}

public static boolean contains(IMethod [] methods,String methodName){
   for(int i=0;i<methods.length;i++){
      if(methods[i].equals(methodName))
         return true;
   }
   return false;
}
```

Uso del atributo estático **File.separator**

Código Fuente 29 : Utilización del separador que determine la clase File.

```
IClasspathEntry jmh = JavaCore.newLibraryEntry(new Path(
   Activator.getLibLocation()+File.separator+
   "optsicomframework.jar"),null,null,true);
```

Búsqueda de métodos y atributos de una clase con IType

Código Fuente 30 : Uso de IType para buscar los métodos y atributos de una clase.

```
...
IJavaProject javaProject = JavaCore.create(container.getProject());
IType type = javaProject.findType(lista.get(i).getAlgorithm()
      .getExecutable());
Map<String,Object> parametros = lista.get(i).getParamValues();
for(String s : parametros.keySet()){
   Object value = (Object) parametros.get(s);
   IMethod [] methods = type.getMethods();
...
```

Incluir la búsqueda de parámetros en las clases padre

Código Fuente 31 : Buscar los parámetros de una clase también en sus clases padres.

```
public static ArrayList<Parameter> getParametros(IType type) {
   try{
      ArrayList<Parameter> params = new ArrayList<Parameter>();
      ArrayList<Parameter> typeParams = buscaParametros(type);
      params.addAll(typeParams);
      ITypeHierarchy newTypeHierarchy = type.newTypeHierarchy(
         new NullProgressMonitor());
      IType [] padres = newTypeHierarchy.getAllSupertypes(type);
      for(IType padre : padres){
         ArrayList<Parameter> padreParams = buscaParametros(padre);
         params.addAll(padreParams);
      }
      return params;
   }catch(JavaModelException e ){
      return null;
   }
}
```

Evitar los InstanceGroup al crear la clase MiExperimento

Código Fuente 32 : Insertar las instancias en la clase MiExperimento.

```
List<Jmh4eInstance> listInstances = ((SimpleFormEditorInput)
   this.getInput()).getExperiment().getInstances();
for(Jmh4eInstance inst : listInstances){
   if(!(inst instanceof InstanceGroup)){
      sb.append("          miExp.addInstance(\""+
         convert(inst.getName())+"\");\n");
   }
}
```

Definir el nuevo nombre de la clase **MiExperimento** en base al nombre del fichero **.experiment**

Código Fuente 33 : Obtener el nombre de la clase Java main.

```
   public static String claseNameFile(String fileName){
      int pos = fileName.indexOf(".experiment");
      String name = fileName.substring(0,pos);
      String primer = name.substring(0,1).toUpperCase();
      name=primer+name.substring(1)+"Experiment.java";
      return name;
   }

   public static String claseName(String fileName){
      int pos = fileName.indexOf(".experiment");
      String name = fileName.substring(0,pos);
      String primer = name.substring(0,1).toUpperCase();
      name=primer+name.substring(1)+"Experiment";
      return name;
   }
```

Cambiar el nombre a la pestaña del **Form**

Código Fuente 34 : Cambiar el nombre de la pestaña del Form en tiempo de ejecución.

```
   this.setPartName(getEditorInput().getName());
```

Para no incluir clases hijas de **ApproxMethod** repetidas, ni abstractas

Código Fuente 35 : Métodos de la clase Hierarchy para comprobar que no se encuentra la clase en la lista y que si es abstracta.

```
1   public static boolean contains(List<IType> lista, IType type){
2      for (IType t : lista){
3         if(type.getElementName().equals(t.getElementName()))
4            return true;
5      }
6      return false;
7   }
8
9   public static boolean isAbstract(IType type){
10     try{
11        IMethod[] methods = type.getMethods();
12        for(IMethod m : methods){
13           if (Flags.isAbstract(m.getFlags()))
14              return true;
15        }
16        return Flags.isAbstract(type.getFlags()) || type.isInterface();
17     }catch(JavaModelException e){
18        return false;
19     }
20  }
```

Añadir las librerías de Optsicom al proyecto evitando repeticiones

Código Fuente 36 : Control al añadir las librerías al proyecto para evitar repeticiones.

```
1  ...
2     IClasspathEntry jmh = JavaCore.newLibraryEntry(new Path(Activator
3        .getLibLocation()+File.separator+"optsicomframework.jar"),null,
4        null,true);
5     if(contains(entries,jmh)){
6        borrar(entries,jmh);
7     }
8     entries.add(jmh);
9  ...
10
11 private boolean contains(List<IClasspathEntry> entries,IClasspathEntry lib){
12    StringTokenizer st = new StringTokenizer(lib.getPath().toOSString(),
13       File.separator);
14    String nameLib="";
15    while(st.hasMoreTokens()){
```

```
16          nameLib=st.nextToken();
17       }
18       for(IClasspathEntry entry : entries){
19          st = new StringTokenizer(entry.getPath().toOSString(),File.separator);
20          String nameEntry = "";
21          while(st.hasMoreTokens()){
22             nameEntry = st.nextToken();
23          }
24          if(nameEntry.equals(nameLib))
25             return true;
26       }
27       return false;
28    }
29
30    private void borrar(List<IClasspathEntry> entries,IClasspathEntry lib){
31       StringTokenizer st = new StringTokenizer(lib.getPath().toOSString(),
32          File.separator);
33       String nameLib="";
34       while(st.hasMoreTokens()){
35          nameLib=st.nextToken();
36       }
37       int borrar = -1;
38       for(int i=0;i<entries.size();i++){
39          st = new StringTokenizer(entries.get(i).getPath().toOSString(),
40             File.separator);
41          String nameEntry = "";
42          while(st.hasMoreTokens()){
43             nameEntry = st.nextToken();
44          }
45          if(nameEntry.equals(nameLib))
46             borrar=i;
47       }
48       if(borrar!=-1){
49          entries.remove(borrar);
50       }
51    }
```