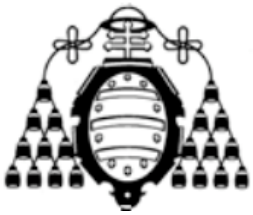
UNIVERSIDAD DE OVIEDO



ESCUELA UNIVERSITARIA DE INGENIERÍA TÉCNICA
EN INFORMÁTICA DE OVIEDO

PROYECTO FIN DE CARRERA

"ENTORNO PARA LA PROGRAMACIÓN COLABORATIVA EN EQUIPOS VIRTUALES"

VºBº del Director del Proyecto

DIRECTOR: Juan Ramón Pérez Pérez

AUTOR: Luis Fernández Álvarez

RESUMEN

El fin del proyecto es ayudar a los desarrolladores en la programación por parejas. Se pretende conseguir acercar este método a programadores que trabajan en lugares alejados geográficamente, de tal forma que no sea necesario que ambos miembros de la pareja de programación estén en el mismo lugar.

Este método de programación, conocido como programación por parejas distribuida, está siendo objeto de muchos estudios y desarrollo. En la actualidad no existen herramientas especializadas para ello, se utilizan programas de mensajería instantánea y escritorio remoto. Por ello resulta muy positivo crear una herramienta que se especialice en ese campo.

El enfoque en su implementación ha sido la integración como un módulo del entorno de programación Eclipse. El módulo ofrece a la pareja de programación un entorno que satisface las necesidades de la programación por parejas tradicional, permitiendo controlar lo que hace el compañero, comunicarse con él, hacer observaciones oportunas sobre el trabajo y editar el código de manera conjunta. Para ello emplea las técnicas de trabajo habituales en el entorno Eclipse, como pueden ser: perspectivas de trabajo especializadas, vistas para controlar de una manera práctica y rápida las funciones del entorno o asistentes para la creación de los proyectos del sistema.

PALABRAS CLAVE

Programación Colaborativa

Groupware

Equipos virtuales

CSCW

XP

Eclipse

PlugIn

Programación por parejas

Programación por parejas distribuida

ABSTRACT

The purpose of the project is to help to developers in the practice of pair programming. It tries to approach developers that works in remote places this programming method. By this way it's not necessary for the both members of the pair to stay in the same place.

This practice is known as distributed pair programming and it's being the object of many studies and development. Nowadays tools specialized for it do not exist, applications like Messenger and remote desktops are used. Because of that, to develop a specific tool is really positive for this knowledge area.

The tool has been implemented with the model of a plugin for Eclipse. The module offers to the pair of programming an environment that satisfy the necessities with the traditional pair programming, it allows to control what the companion does, to communicate with him, to make opportune observations on the work and to change the code simultaneously. To do that, it uses the habitual techniques of work in the Eclipse platform, for example, it has views to control of a practical and fast way the functions of the environment and wizards for the creation of the projects of the system.

KEYWORDS

Colaborative Programming

Groupware

Virtual Teaming

CSCW

XP

Eclipse

PlugIn

Pair Programming

Distributed Pair Programming

Me gustaría agradecer el trabajo de mi director Juan Ramón Pérez Pérez, así como la labor de los usuarios que participaron en las pruebas del sistema. Finalmente, doy las gracias a mis compañeros y amigos, con su interés y opinión han inspirado parte de este proyecto.

TABLA DE CONTENIDOS

1 INTRODUCCIÓN

1.1 Necesidad de la programación colaborativa.

Con el auge de Internet y en general de las redes de alta velocidad se ha producido un mayor incremento de la comunicación entre profesionales y gente en general, por otro lado este desarrollo también ha causado que las actividades profesionales se distancien en cuanto al emplazamiento donde son realizadas. En este marco es donde aparece una creciente necesidad de colaboración en los distintos campos que se apoyan en las nuevas tecnologías.

Un ámbito en el que la colaboración se hace claramente necesaria es la programación y desarrollo de software. El desarrollo de software es un campo en el cual los proyectos son cada día más grandes y trabaja más gente, en la mayoría de las ocasiones en diferentes departamentos o ciudades, por ello es claramente necesario que se estudien y mejoren las técnicas y herramientas de las que se dispone para realizar estas tareas. Es un paso inevitable que se ha dado en el campo de la programación y que ha de ser potenciado en los próximos años.

Pero la programación colaborativa (y el trabajo colaborativo en general) necesita de una coordinación y estructura bien definida para que resulte una actividad productiva, es aquí donde nace la necesidad de la creación de una herramienta específica (o conjunto de ellas) que se encargue de gestionar esta actividad tanto a nivel de aplicación como a nivel humano. La actividad en la programación colaborativa está sujeta a muchas decisiones individuales de cada miembro del grupo y es importante integrar todas estas opiniones y decisiones para todo el conjunto del grupo y así este permanezca coordinado evitando, por ejemplo, el trabajo redundante.

1.2 Modelos de programación colaborativa y comunicación

La programación colaborativa tiene una necesidad esencial: los mecanismos de comunicación. En este punto encontramos la comunicación directa entre los participantes y el empleo de "artefactos compartidos" [SCHLICHTER].

En el segundo caso los participantes pueden observar el empleo y manipulación de estos artefactos compartidos, constituyendo por tanto una vía de comunicación indirecta entre ellos. Estos dos mecanismos pueden ser complementarios, puesto que la comunicación directa toma, en muchos casos, la manipulación de artefactos compartidos como punto de referencia.

En estos medios de comunicación se definen diferentes clasificaciones. Así podemos encontrar en la comunicación directa, una vía asíncrona y otra síncrona que poseerán sus ventajas y particularidades:

- **Síncrona**

  En este modelo de comunicación se establece un enlace de comunicación y la información se intercambia en tiempo real entre los trabajadores. Para ello se pueden constituir varios flujos, un flujo de datos podría contener el documento compartido entre los miembros del grupo y otro flujo coordenadas de los diferentes miembros del grupo.

  Hay que tener atención en la información enviada en relación al ancho de banda disponible. No es lo mismo trabajar en una LAN que a través de Internet, aunque

hoy por hoy este aspecto pasa a ser trivial por la implantación que goza la banda ancha.

- **Asíncrona**

  Este es el modelo de comunicación alternativo, no hay interacción en tiempo real y los requerimientos son menos exigentes que el modelo síncrono.

Uno de los mayores problemas que surgen cuando se realiza programación colaborativa trabajando sobre recursos compartidos es la concurrencia. Este es un punto importante puesto que la información es accedida por varios usuarios a la vez y hay que garantizar en todo momento que la consistencia de la misma es la adecuada. Ante este problema surgen dos enfoques:

- **Visión pesimista**: en este caso es la aplicación de ciertos protocolos técnicos lo que se encarga de controlar la concurrencia en los accesos y garantizar la consistencia de los datos almacenados en el espacio de trabajo compartido.
- **Visión optimista**: protocolos "sociales" para asegurar la consistencia de la información. En este caso los trabajadores son conscientes del problema de la consistencia de la información y saben informar al resto de trabajadores que van a realizar ciertas modificaciones o que van a utilizar cierto recurso y así evitar cualquier conflicto con el resto.

1.2.1 Modelos de programación

1.2.1.1 Equipos de trabajo tradicionales

En este caso se trata de un equipo de trabajo tradicional que trabajan conjuntamente pero sin herramientas especializadas en la colaboración.

La comunicación en muchos casos es precaria y no especializada en el trabajo en equipo. Existen problemas de redundancia de trabajo y problemas en la resolución de problemas y en la puesta de acuerdo en la toma de decisiones.

1.2.1.2 Equipos virtuales (virtual teaming)

Es un término genérico que engloba a un grupo de gente trabajando de manera conjunta para conseguir un mismo objetivo independientemente del tiempo, la distancia, cultura…

Los miembros de un equipo virtual pueden estar en diferentes lugares de trabajo, o ser trabajadores con mucha movilidad y necesitan mantener una buena coordinación y colaboración en su trabajo. Este es el fruto de entornos cada vez más globales que se dan en las empresas, aunque también se aplican en ámbitos educativos con el aprendizaje distribuido o la educación a distancia.

1.2.1.3 Programación en parejas:

Es una técnica de programación colaborativa cada vez más usada. En ella colaboran dos programadores de manera síncrona, en este caso la sincronía se debe a que está pensada para realizarse cara a cara. Se enmarca en el modelo de programación extrema.

En este enfoque las dos personas trabajan sobre el mismo computador con un teclado y ratón solamente.

Algunas ventajas que tiene este sistema en particular sobre la programación por parejas distribuida que se comenta a continuación son:

- Los usuarios pueden señalar sobre el trabajo de su pareja lo que acelera la corrección e fallos, en el caso de la programación por parejas distribuida se necesita hacer referencia a líneas y secciones de código.
- Se puede controlar las expresiones de la pareja y esto ayuda a mantener la relación, en la programación distribuida aunque exista el empleo de webcams no se obtiene el mismo resultado.
- El hecho de estar físicamente cerca facilita las explicaciones y aclaraciones de aspectos del proyecto.

1.2.1.4 Programación en parejas distribuida:

La programación por parejas distribuida (distributed pair programming, dPP) [STOTTS 2003] significa que dos miembros del equipo colaboran de manera síncrona en el mismo diseño o código desde diferentes lugares. Esto significa que los dos estarán viendo una copia de la misma pantalla y al menos uno e ellos debería tener la capacidad para realizar cambios en el contenido.

Existen algunas ventajas de la programación por parejas distribuida, entre ellas destaca la mejora de la visibilidad de los miembros que disponen cada uno de una pantalla, esto conlleva que el observador pueda hacer consultas en la web sobre el trabajo sin entorpecer la labor del programador principal, las conversaciones se centran más sobre el trabajo que sobre otros temas, facilita mantener copias de las ideas que vayan surgiendo al emplear medios electrónicos…

1.3 Herramientas de edición y programación colaborativa existentes.

1.3.1 RECIPE

RECIPE [SHEN 2000, PEREZ2006] es un entorno de programación colaborativa para el diseño, codificación, prueba, depuración y documentación de un mismo programa.

Se trata de un sistema en tiempo real que se asienta sobre Internet.

Es un prototipo que emplea una arquitectura centralizada. La entrada de todos los participantes es unificada y enviada a una instancia de la aplicación compartida y cualquier salida es devuelta a todos los participantes.

La aplicación centralizada es un servidor java y los participantes se componen de applets java descargados del servidor. Sus principales características son:

- La colaboración se realiza de forma transparente. Las aplicaciones habituales para un solo usuario no necesitan modificación y la adaptación a un campo colaborativo es llevada a cabo por el sistema.
- Control de acceso flexible. Los usuarios que se unen a la sesión colaborativa reciben una serie de roles o permisos (tales como ver, participar, etc..).
- El sistema es extensible, pudiendo adaptarse a nuevas herramientas de desarrollo.

- La colaboración se realiza de manera jerárquica. Existen cuatro tipos de sesiones en el prototipo: sesión de shell de unix, sesión de edición, sesión e compilación y sesión de depuración.

El sistema para la edición en tiempo real que viene incluido en el sistema tiene las siguientes características:

- Concurrencia en la edición de cualquier texto en cualquier momento.
- Deshacer cualquier operación en cualquier momento de manera concurrente.
- Control de concurrencia optimista.

1.3.2 COLLEGE

COLLEGE [BRAVO 2004] es un entorno orientado a la realización de problemas de programación de estudiantes. Sus características las podemos estructurar en varios ámbitos del sistema:

- **Protocolo de colaboración**

Se compone de varios espacios de trabajo: 1) Edición / Revisión, en este espacio un usuario edita mientras que los demás revisan el código de manera síncrona y pueden hacer sugerencias o cambiar el usuario que esta codificando. 2) Compilación, compila el programa editado y revisado por los participantes mostrando los errores de compilación a todos. 3) Ejecución, en este punto los participantes se ponen de acuerdo en cuanto a los parámetros de ejecución.

En este modelo se observa claramente el enfoque de programación por parejas, alejándose de un editor totalmente síncrono. Por ello el trabajo es mono usuario con varios observadores, pudiendo cambiarse el rol de editor de manera cómoda.

Para cambiar entre los espacios de trabajo un participante del grupo propone el cambio y el resto ha de aprobar dicho cambio.

- **Soporte para las tareas del dominio.**

Se basa en el modelo objeto-acción. En la edición los objetos dependiendo de la granularidad podríamos tener desde las clases, métodos,… hasta los caracteres que se manipulan en el código fuente. En el resto de espacios no hay objetos ni acciones destacados.

- **Soporte de comunicación.**

La comunicación se basa en el empleo de un Chat estructurado, esto es, un Chat con un conjunto preestablecido de actos de comunicación. El sistema clasifica los mensajes de acuerdo a tres criterios: 1) Tipo de mensaje: aserciones, preguntas y respuestas. 2) Lugar del diálogo: mensajes iniciales y reactivos. 3) Tipo de adaptación: mensajes adaptables (información e las tareas del dominio) y no adaptables.

Así encontramos por ejemplo: "*Pienso que…*", "*Falta el/la (clase | método | atributo | palabra reservada | variable) <objeto>*", "*Veo un fallo en…*". También se disponen de mensajes libres que no siempre estarán disponibles para forzar al participante a usar las combinaciones preestablecidas.

- **Técnicas de awareness**

La información el entorno y awareness se ofrecen mediante los siguientes elementos: panel de sesión, muestra los participantes de la sesión con una foto de cada uno y un color para resaltar e identificar las interacciones de cada uno. También muestra el estado de cada usuario (editando, observando, ejecutando…). Por otro lado están los telepunteros: se utilizan para mantener un puntero de edición para el usuario el turno, así poder ver que zona se esta editando o que carácter se está manipulando. También posee una lista de interacciones referida a las áreas de coordinación: cambio de turno, propuesta de compilación y propuesta de ejecución, y con ellas los mensajes de propuesta y de posible acuerdo o desacuerdo de cada uno. Finalmente los aspectos de awareness en la comunicación mantenida en el Chat con los emisores de cada mensaje.

1.3.3 COLLDEV

COLLDEV [PEREZ2006] es un sistema CSCL vía web cuyo objetivo es facilitar la coordinación y colaboración de grupos en el desarrollo de proyectos software.

Proporciona un entorno en el cual los desarrolladores comparten los archivos fuente trabajando de forma paralela sobre los mismos.

En el aspecto de comunicación se ha creado un sistema de envío de mensajes asíncronos que facilite el planteamiento de tareas, resolución de dudas, etc… entre el supervisor y los miembros del proyecto. Estos mensajes pueden ser del tipo preguntas y respuestas, comentarios o tareas. Todos estos mensajes pueden ser respondidos por el destinatario mediante otras preguntas o respuestas, matizando comentarios o dando soluciones, mensajes de aceptación o rechazo respectivamente.

En cuanto al trabajo en paralelo, el sistema permite a los desarrolladores modificar de este modo los archivos y será el propio entorno el encargado de juntar automáticamente y de manera asíncrona las modificaciones. En el caso de posibles conflictos en las modificaciones serán los propios desarrolladores los que manualmente los solucionen.

El sistema también dispone de una historia activa que permite el seguimiento del proyecto, ésta se forma con las diferentes versiones del código fuente que se van generando a medida que los desarrolladores realizan las tareas que se le plantean.

Para la toma de decisiones en el grupo de desarrollo se pone a disposición de los participantes un sistema de encuestas.

El prototipo que plantea COLLDEV también se encarga de la gestión de roles de usuarios como puedan ser desarrolladores, supervisores o administradores.

Cada usuario tendrá acceso al sistema mediante una clave y nombre de usuario. Dentro de cada rol comentado se disponen tareas específicas, así los supervisores se encargan de revisar el código fuente de sus desarrolladores, y los administradores se encargarán de tareas como aceptar nuevos usuarios al sistema y finalmente los desarrolladores se permitirán tareas como edición y modificación de los código fuente del proyecto.

COLLDEV también incorpora el concepto de Grupo e Trabajo en el cual los usuarios se darán de alta para participar y colaborar. Estos grupos e trabajo son configurables en cuanto a miembros que lo conforman o la pertenencia o no e un supervisor.

Cada uno de estos grupos tiene un espacio de trabajo colaborativo asociado. En él se almacenan los archivos correspondientes al trabajo que están realizando. Aun así el trabajo personal de cada desarrollador se realiza sobre su propio espacio personal, al solicitar la modificación de un archivo este es copiada a su espacio personal.

1.3.4 IRIS

IRIS[IRISWEB,SCHLICHTER,KOCH95b] es un entorno de edición en grupo. Se compone de un núcleo formado por componentes replicados que se comunican entre si para asegurar la consistencia del documento y calculan la información de awareness, este núcleo se denomina "almacenamiento y servicio de awareness".

- Acceso al documento y un historial de sus cambios.
- Se producen eventos de notificación (Notification events) fruto de la interacción del usuario con los datos del documento, y atributos de estado son distribuidos.
- Atributos de estado: nombre y valor.
- El servicio de awareness almacena atributos para todos los documentos, usuarios y para todos los hosts participantes.
    - Información de los documentos:
- Lista de las áreas de trabajo de lectura y de escritura. El área de trabajo guarda información sobre las posiciones que los diferentes usuarios han tenido en el documento en el tiempo.
- Lista de hosts que almacenan replicas del documento.
- Lista de usuarios que han accedido al documento.
- Momento el último cambio.
    - Información de usuario:
- El sistema mantiene un atributo de estado para los que están o han estado trabajando (valores del tipo 'inactive', 'active-on-host', 'idle-in-groupware-application',…).
- Lista de hosts y documentos en los que ha estado trabajando el usuario.
- Momento de los últimos cambios.
- Estados definidos por el usuario como 'do not disturb', 'in meeting' que 'supersedes' el estado calculado automáticamente. Debe ir acompañado por un tiempo de validez.

Aparte de estos atributos fruto de la interacción existen otros que han de permitir ser definidos por el usuario como atributos estándar del documento, también existen reservas que son controles de concurrencia optimistas que pueden ser establecidos en partes del documento.

Los documentos se presentan como una estructura de árbol con las diferentes partes. Cada nodo es coloreado de acuerdo a su estado, si está reservado o no. Una vez seleccionado un nodo se muestra un historial del mismo.

En el prototipo también se muestra una lista de usuarios que están trabajando en el documento.

1.4 Carencias de las aplicaciones existentes de programación colaborativa.

Con todo lo mostrado en los apartados anteriores vemos como las herramientas y prototipos existentes y observados carecen en muchos de ellos de técnicas awareness especializadas en el campo de la programación. Al igual que los medios de comunicación son insuficientes y poco adaptados al ámbito de la programación, aun así existen proyectos como College que se acerca más a este punto presentando un Chat compuesto de frases comúnmente usadas en la programación, lo cual sin duda agiliza el trabajo en grupo.

Otro problema que se observa es la falta de información sobre el trabajo general del grupo, es decir, información con la que se pueda obtener una valoración relativa del trabajo de cada uno o un seguimiento del rendimiento y productividad que se está obteniendo en el trabajo realizado. Así aspectos como errores en los programas, modificaciones realizadas por los miembros del equipo, etc…

Por otro lado, a veces los entornos hace uso excesivo de la necesidad de que los usuarios estén presentes en el mismo momento, así es recomendable que los sistemas proporcionen métodos que no exijan a los miembros del grupo estar en el mismo instante trabajando, de este modo sería adecuado que un usuario pudiera saber quien fue el último en modificar una determinada función y dejarle un comentario acerca de ello sin necesidad de estar en el mismo momento trabajando conjuntamente.

Por último, en muchos casos los entornos vistos son prototipos aislados y se ve necesaria la integración con las herramientas habituales de programación.

1.5 Experiencias en la aplicación de modelos de programación colaborativa.

Existen numerosos estudios comparando los diferentes modelos de aplicación de programación colaborativa. Actualmente estos estudios se orientan a comparar y analizar las ventajas y deficiencias de la programación por parejas, un caso particular de programación colaborativa que ha demostrado ser una buena estructuración de equipos de desarrollo. A continuación se muestran algunos de ellos:

Uno de los estudios analizados [STOTTS 2003] se llevó a cabo entre estudiantes de Carolina del Norte, en ella se enfrentó el modelo de equipos virtuales tradicional frente a los equipos usando programación por parejas distribuida. El primero de los grupos realizaba el código independientemente y se lo enviaban por email mientras que los estudiantes con programación por parejas emplearon herramientas del tipo de NetMeeting o pcAnywhere.

El código desarrollado en Java fue testeado con JUnit.

Al final del estudio se obtuvieron los siguientes resultados:

- Los equipos virtuales tradicionales emplearon mucho tiempo en integrar su código.
- Los equipos dPP realizaban citas para sesiones virtuales y coordinarse.
- El hecho de trabajar por parejas añadía más responsabilidad a los estudiantes, ya existía cierta presión entre los estudiantes.
- Los test de unidad realizados por los equipos dPP fueron un 70% más que en el resto de equipos no por parejas, al trabajar de manera síncrona existía una ventaja a la hora de añadir test y dirigir las pruebas, con ello aumenta la calidad del software, al menos en cuanto a sus pruebas.

Las conclusiones que se obtienen en el estudio se muestran a continuación:

- Necesidad de encuentros cara a cara al menos para conocerse y hacer un brainstorm inicial del sistema.
- Los programadores por parejas deben mantener una comunicación continua para explicar que es lo que están haciendo
- En los equipos dPP la duplicación de ficheros se reduce al trabajar de manera síncrona sobre uno de los hosts.
- Los programas realizados en los equipos dPP fueron mejores que los realizados en los equipos que no empleaban parejas distribuidas.
- Se puede desarrollar software de manera colaborativa fácilmente con las herramientas existentes.
- Las parejas de programación adoptan mejor trabajo en equipo y comunicación.
- Los equipos dPP no pierden la ventaja de la programación por parejas no distribuida en cuanto a presión de las parejas, aprendizaje entre los miembros de la pareja y el hecho de estar trabajando dos cabezas en vez de una.

Otras experiencias de aplicación de la programación colaborativa la podemos encontrar en [WILLIAMS2001], en ella se realiza un estudio de las ventajas que tiene el pair programming, sus resultados los enfoca en varios caminos, a continuación se muestran los resultados y conclusiones de sus estudios:

- **Aspectos de economicidad**: Según los estudios hay un 15% de reducción en los defectos y un incremento del tiempo de desarrollo con la segunda persona. Los casos de prueba pasados por los equipos de programación por pareja fue superior a todos los desarrolladores individuales.
- **Satisfacción**: Los estudiantes que han trabajado en programación por equipos se muestran menos frustrados. Se encuentran más a gusto teniendo un compañero en quien apoyarse y del cual obtener ayuda. Las soluciones obtenidas también resultan ser más creativas y eficientes.
- **Revisiones continuas**: Proporciona los mejores resultados en la eliminación de defectos al trabajar conjuntamente. Este punto sin duda es ventajoso ya que es más barato encontrar los errores en las etapas de diseño y codificación en las que se realiza la programación por parejas que en etapas posteriores.
- **Resolución de problemas:** En este aspecto también se ha demostrado un excelente rendimiento, según se ha estudiado los miembros de la pareja complementan sus conocimientos para resolver los problemas rápidamente.
- **Aprendizaje**: Los miembros de la pareja de programación han manifestado que la actividad les proporciona mayor conocimiento. Esta mejora se produce en varias formas, por un lado se aprenden conceptos, se aprenden hábitos para aprender y se mejoran la reflexión y observación de uno mismo.
- **Construcción de equipos y comunicación**: Las parejas en este tipo de programación aprenden a resolver problemas de manera conjunta y mejora el trabajo en grupo. Esto conlleva que la comunicación se haga fluida y frecuente entre los miembros de la pareja de programación.

1.6 Conclusiones

A la vista de todo lo anterior surge la necesidad de crear una herramienta que facilite la programación colaborativa en equipos de desarrollo. Dicha herramienta proporcionará un entorno que permita al desarrollador y al proyecto:

- Estar informado del proyecto mediante sistema de awareness, se especializará en dos ámbitos, el proyecto, los documentos o archivos de código fuente y los usuarios del sistema.
- Mantener comunicación con los miembros del grupo de desarrollo.
- Resolver problemas de redundancia de versiones y ficheros.
- Establecer roles de los miembros del proyecto.
- Trabajo síncrono.
- Un medio de comunicación especializado para la programación.

Se centrará especialmente en el trabajo síncrono de los miembros del grupo sobre el mismo código fuente y conjunto de archivos intentando reducir el problema de la redundancia de archivos y versiones, centralizando todos ellos y creando una vía común para su modificación. Se facilitará aun así (mediante el sistema de awareness y comunicación) el trabajo de los desarrolladores para que no requiera la presencia de los miembros del grupo en el mismo instante.

También se pretende una herramienta integrable y funcional con las herramientas habituales de desarrollo.

2 ANÁLISIS

2.1 Análisis previo[1]

El análisis previo realizado sobre el proyecto nos ha permitido realizar una clasificación de los requisitos que se estiman para el proyecto. Esta clasificación abarca cuatro conjunto principales y un quinto conjunto de requisitos generales. A continuación se muestran los requisitos

2.1.1 Aspectos generales

R1.1. El sistema debe ser adaptable a un entorno de programación habitual.

R1.2. El sistema deber establecer varios roles de usuario, básicamente un usuario principal que controle la versión maestra del proyecto y otro de cliente.

R1.3. El sistema almacenará para todos los usuarios: nombre completo, dirección de correo electrónica y un campo de información adicional.

R1.4. El sistema almacenará adicionalmente para los usuarios principales: nombre corto.

R1.5. El sistema almacenará adicionalmente para los usuarios cliente: su login de acceso al proyecto remoto, su contraseña y la dirección del host remoto.

2.1.2 Sesión

R2.1. La iniciativa para iniciar y finalizar una sesión de trabajo sobre un proyecto la tiene el usuario principal.

R2.2. Los clientes requerirán un login y contraseña para conectarse a los proyectos fuente y participar en el trabajo de colaboración.

R2.3. Los clientes deben poder tener el proyecto accesible a través de la red.

2.1.3 Comunicación

R3.1. Debe existir un medio de comunicación, a modo de chat, entre ambos participantes del proyecto que les permita enviarse mensajes textuales rápidamente. Los mensajes que se envíen mediante este medio deberán mostrar un identificador para saber cuales el origen del mensaje.

R3.2. El sistema debe mostrar información detallada al usuario sobre el programador con quien está colaborando.

R3.3. La información básica que ha de mostrar será: la información referida en R1.3, versión de la JDK que tiene instalada y la versión del sistema operativo sobre el que se está trabajando.

R3.4. El sistema deberá informar a los participantes sobre que recursos pueden editar en colaboración.

---

[1] En el apartado 3.1 se puede encontrar una relación de las restricciones aplicadas sobre estos requisitos iniciales.

2.1.4 Recursos

R4.1. Tanto el usuario cliente como el usuario principal dispondrán de una carpeta de trabajo.

R4.2. El sistema permitirá al usuario principal crear los recursos dentro del proyecto fuente (su carpeta de trabajo) que se irán propagando al resto de clientes.

R4.3. Los usuarios cliente no podrán crear recursos que se propaguen al proyecto fuente. Estos se irán creando de acuerdo a la manipulación del usuario principal.

R4.4. El sistema permitirá a los participantes crear anotaciones sobre los recursos que permita dejar información de manera persistente.

R4.5. Las anotaciones tendrán un campo de mensaje y otro de prioridad.

R4.6. Todos los usuarios pueden crear anotaciones pero solo el usuario principal puede eliminarlas.

2.1.5 Coedición

R5.1. El sistema ha de ser capaz de permitir la edición del código fuente de manera síncrona por parte de varios usuarios del sistema, de tal forma que todos los participantes puedan ver en tiempo real el estado del proyecto y lo que hace cada uno.

R5.2. El sistema debe informar al usuario sobre quien esta modificando un recurso.

2.2 Casos de uso

En este proyecto se han encontrado dos actores en el análisis de los casos de uso. Por un lado tenemos “usuario fuente”, es el usuario principal y quien tendrá la iniciativa en el trabajo de colaboración. El segundo actor que encontramos es “usuario cliente” y será el que dependa de “usuario fuente” para las labores principales del sistema.

Los casos de uso encontrados han sido cinco para cada uno de los actores. En las figuras 1 y 2 se pueden observar los diagramas de casos de uso asociados a cada uno de los actores del sistema.

Figura. 2-1. Casos de uso del actor 'Pareja Fuente'

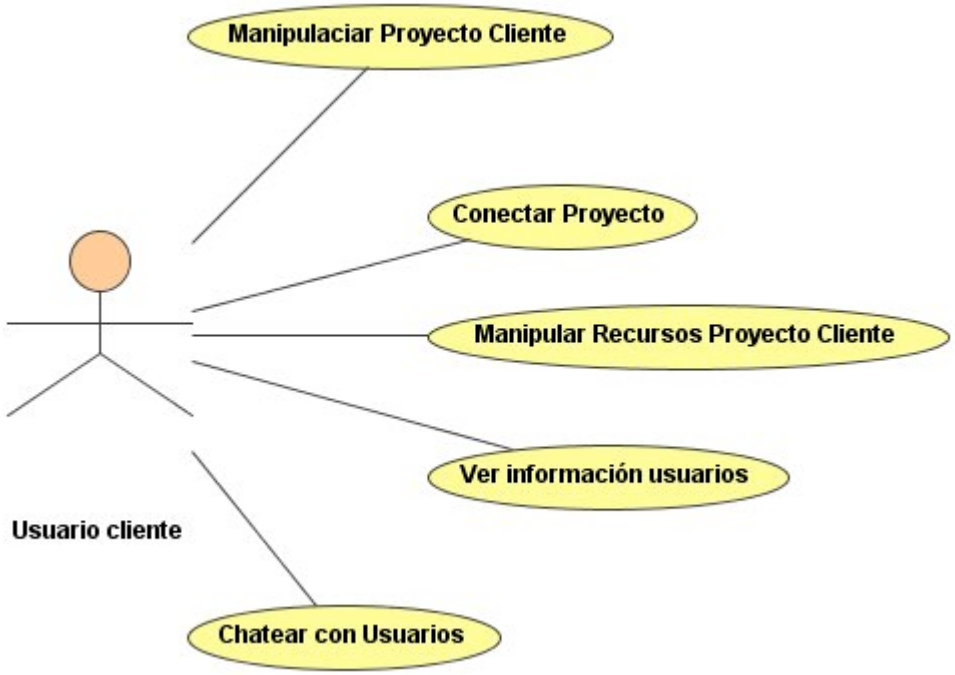


Figura. 2-2 Casos de uso del actor 'Pareja Cliente'

Caso de Uso: Manipular Proyecto Fuente

El usuario fuente que trabaje con el sistema podrá gestionar los proyectos fuente del entorno. Para la creación de un nuevo proyecto fuente el usuario deberá introducir la información de acceso del cliente, además de la información personal del usuario que actúa de usuario fuente. Esta información podrá ser modificada después de que el proyecto haya sido creado.

Caso de Uso: Lanzar Proyecto

El usuario fuente tendrá el control para lanzar un proyecto fuente y así permitir la conexión de los clientes al proyecto y dar comienzo a una sesión de trabajo de colaboración.

Caso de Uso: Manipular Recursos Proyecto Servidor

Para la gestión de los recursos en el proyecto servidor el usuario fuente tendrá total libertad para su manejo, teniendo en cuenta que sus modificaciones son las que establecerán la estructura del proyecto. Podrá crear nuevos archivos o carpetas en la estructura del proyecto sobre los que se hará el trabajo de colaboración. El usuario fuente podrá por tanto, realizar modificaciones en el contenido de los archivos.

También se podrá hacer anotaciones sobre los archivos de manera persistente.

Caso de Uso: Ver información usuario

El usuario tendrá siempre disponible la opción de ver la información personal de los usuarios con quien está programando.

Caso de Uso: Chatear

Los usuarios podrán enviar mensajes a todos los participantes en el trabajo de colaboración.

Caso de Uso: Manipular Proyecto Cliente

El usuario cliente que trabaje con el sistema podrá gestionar los proyectos cliente del entorno. Para la creación de un nuevo proyecto cliente el usuario deberá introducir la información de acceso al proyecto fuente, además de la información personal del usuario que actúa de usuario fuente. Esta información podrá ser modificada después de que el proyecto haya sido creado.

Caso de Uso: Conectar Proyecto

El usuario cliente tendrá el control del momento en el que se conecta con el proyecto fuente y empieza a trabajar con el usuario fuente remoto.

Caso de Uso: Manipular Recursos Proyecto Cliente

El usuario cliente no tiene el control sobre la creación o eliminación de recursos en el proyecto. Esto depende de las decisiones del usuario fuente. El usuario cliente sólo podrá realizar anotaciones sobre los recursos que se encuentren en el proyecto y modificar su contenido.

2.3 Escenarios

Caso de Uso 1: Manipular Proyecto Fuente

Identificador: 1.1

Título: Creación de un Proyecto Fuente

Poscondiciones: Se crea un nuevo directorio al espacio de trabajo del usuario con la estructura básica de un proyecto fuente.

Descripción: Para crear un nuevo proyecto fuente el usuario deberá introducir un nombre para el proyecto, la información de acceso de la pareja (login y contraseña) y su propia información personal (nombre corto, nombre, correo electrónico y un campo de información adicional).

Caso de Uso 2: Lanzar Proyecto

Identificador: 2.1

Título: Publicación de proyecto

Precondiciones: Tener un proyecto fuente correctamente creado.

Poscondiciones: El proyecto fuente deberá ser accesible remotamente. La información personal del usuario que lanza el proyecto tiene todos los campos completos.

Descripción: El usuario fuente lanzará el proyecto previamente creado para permitir el acceso remoto al mismo y el trabajo en colaboración. Durante el proceso la información personal del usuario deberá ser completada en cuanto a JDK y sistema utilizado. Como esta información puede ser variante no será persistente, sino que tendrá la duración del trabajo en colaboración.

Caso de Uso 3: Gestión de Recursos de Proyecto Fuente

Identificador: 3.1

Título: Creación de nuevo recurso

Precondiciones: Tener un proyecto fuente correctamente creado.

Poscondiciones: El nuevo recurso creado deberá estar presente en la estructura del proyecto fuente. En el caso de que el proyecto esté publicado deberá estar creado el mismo recurso en la estructura de proyecto de los usuarios cliente que se encuentren en ese momento conectados al proyecto.

Descripción: El usuario creará un nuevo recurso para el proyecto con un nombre determinado y se creará en la estructura del proyecto. En el caso de que se este trabajando en colaboración sobre el proyecto, una vez creado el recurso se informará a los clientes del proyecto sobre ello y todos ellos deberán crear un recurso homologo en sus proyectos.

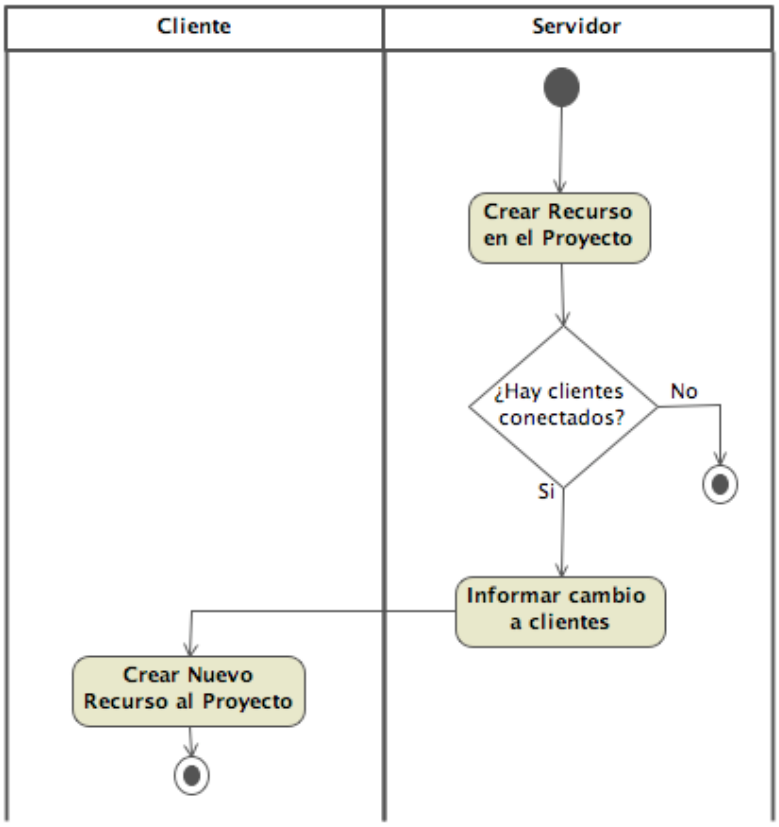


Figura. 2-3. Diagrama de actividad que representa la creación de un proyecto

Identificador: 3.2

Título: Eliminación de un recurso

Precondiciones: Tener un proyecto fuente correctamente creado.

Poscondiciones: El recurso seleccionado habrá sido eliminado de la estructura del proyecto fuente y, en el caso de que el proyecto esté en publicación, de la estructura de proyecto de los clientes que estén conectados al proyecto en ese momento.

Descripción: Para eliminar un recurso del proyecto fuente el usuario fuente deberá seleccionar y confirmar el borrado del recurso. Además, si el proyecto se encuentra en publicación, se deberá notificar a todos los clientes que se encuentren en ese momento conectados acerca de la modificación, y así la lleven a cabo cada uno de ellos.

Identificador: 3.3

Título: Creación de anotación en un recurso

Precondiciones: Tener un proyecto fuente en publicación

Poscondiciones: La anotación estará almacenada de manera persistente en la estructura del proyecto fuente y los clientes estarán notificados de la creación de la anotación.

Descripción: Para crear una anotación sobre un recurso el usuario fuente deberá seleccionar el recurso adecuado e introducir todos los campos de la información requerida para las anotaciones: mensaje y prioridad. La anotación será

almacenada de manera persistente en el proyecto y será notificada dicha creación a los clientes que se encuentre conectados en ese momento al proyecto.

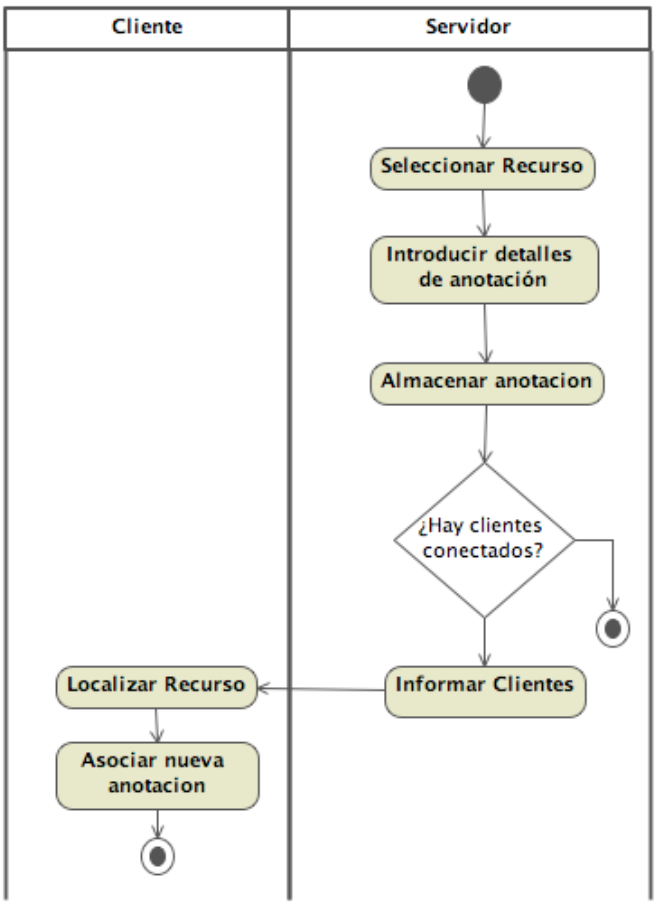


Figura. 2-4. Diagrama de actividad para la creación de anotaciones

Identificador: 3.4

Título: Edición en colaboración de un recurso

Precondiciones: Tener un proyecto fuente en publicación

Poscondiciones: Las modificaciones realizadas se manifestarán igualmente en todos los clientes del proyecto fuente.

Descripción: Para empezar la edición en colaboración de un recurso el usuario fuente deberá abrir el recurso deseado, esto notificará al resto de clientes que se va a empezar la edición del recurso. En este momento el usuario fuente estará en disposición de empezar a modificar el contenido del recurso.

A partir de este momento toda la secuencia de modificaciones deberá ser enviada en tiempo real a todos los clientes conectados al sistema para permitir una edición síncrona del recurso.

Para terminar la edición del recurso el usuario fuente podrá cerrar el archivo en cuestión, en este momento se deberá notificar a los clientes tal evento para que sepan que se ha terminado el trabajo en colaboración.

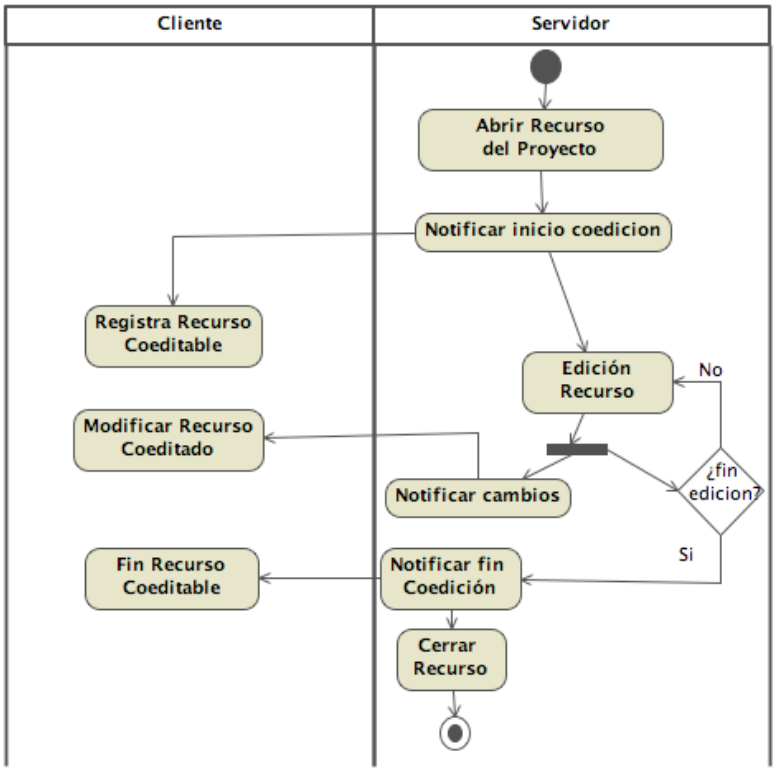


Figura. 2-5. Diagrama de actividad en la coedición por parte del servidor

Caso de Uso 4: Ver información usuario

Identificador: 4.1

Título: Información sobre usuario

Precondiciones: En el caso del usuario fuente deberá tener publicado su proyecto. En el caso del usuario cliente deberá tener su proyecto conectado.

Descripción: El usuario podrá ver la información personal de los usuarios de la misma sesión de trabajo de colaboración en la que está participando. Para cada uno de ellos podrá ver su nombre, dirección de correo, JDK y versión del sistema operativo junto con un campo de información adicional.

Caso de Uso 5: Chatear

Identificador: 5.1

Título: Enviar mensaje a usuarios

Precondiciones: Tener iniciada una sesión de trabajo. En el caso de un usuario fuente habiendo publicado el proyecto, y en el caso de usuario cliente habiendo conectado un proyecto.

Poscondiciones: El mensaje es recibido en todos los participantes de la sesión y es identificable su origen.

Descripción: Los usuarios podrán enviar mensajes a todos los participantes de la sesión de trabajo. Para ello tendrán que introducir el texto del mensaje que desean enviar a los usuarios. El sistema tomará el nombre corto del usuario (en el caso de los usuarios clientes coincide con su login) y este será el identificador que se dará como origen al mensaje enviado.

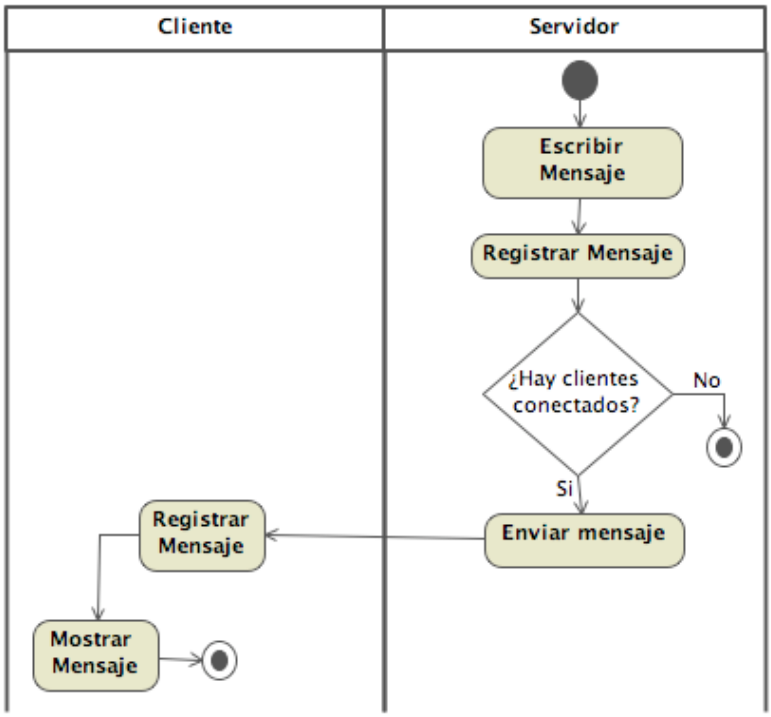


Figura. 2-6. Diagrama de actividad para el envío de un mensaje (válido en ambos sentidos)

Caso de Uso 6: Manipular Proyecto Cliente

Identificador: 6.1

Título: Creación de un Proyecto Cliente

Poscondiciones: Se crea un nuevo directorio al espacio de trabajo del usuario con la estructura básica de un proyecto cliente.

Descripción: Para crear un nuevo proyecto fuente el usuario deberá introducir un nombre para el proyecto, la información de acceso al proyecto fuente (host, login y contraseña) y su propia información personal (nombre, correo electrónico y un campo de información adicional).

Caso de Uso 7: Conectar Proyecto

Identificador: 7.1

Título: Conexión de proyecto cliente

Precondiciones: Tener un proyecto cliente correctamente creado y el proyecto fuente publicado.

Poscondiciones: La estructura del proyecto cliente deberá ser igual a la estructura del proyecto fuente.

Descripción: Para conectar el proyecto cliente el usuario deberá elegir la opción oportuna del sistema que tomará los datos previamente almacenados al crear el proyecto para realizar la conexión. El host fuente comprobará sus credenciales y si son correctas le permitirá el acceso al sistema. En ese momento la estructura actual del proyecto será cargada al cliente y se podrá empezar a trabajar en colaboración.

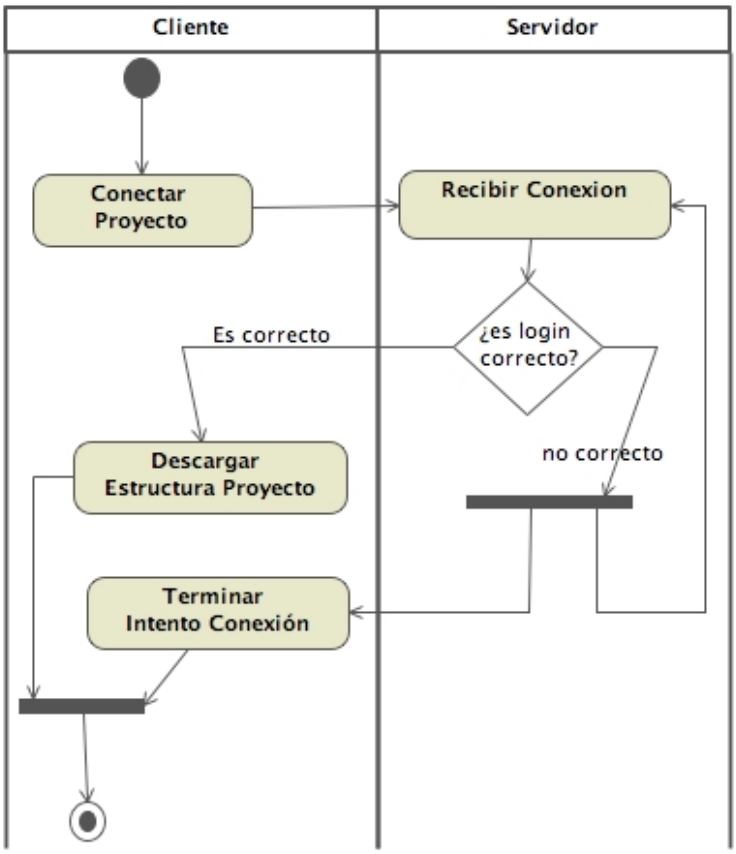


Figura. 2-7. Diagrama de actividad para representar la conexión

Identificador: 7.2

Título: Desconexión de proyecto cliente

Precondiciones: Tener un proyecto cliente conectado

Poscondiciones: Todos los participantes del proyecto deberán estar notificados de la desconexión del participante.

Descripción: El usuario cliente solicitará la desconexión de la fuente del proyecto. La fuente registra su desconexión y libera los recursos asociados para no intentar volver a informar a ese usuario.

Caso de Uso 8: Manipular Recursos Proyecto Cliente

Identificador: 8.1

Título: Creación de una anotación

Precondiciones: Tener un proyecto cliente conectado

Poscondiciones: La anotación habrá sido enviada a la fuente del proyecto y estará de persistirá en el cliente durante la sesión de trabajo.

Descripción: Para crear una anotación sobre un recurso el usuario cliente deberá seleccionar el recurso adecuado e introducir todos los campos de la información requerida para las anotaciones: mensaje y prioridad. La anotación será almacenada de en el proyecto y será notificada dicha creación a la fuente del proyecto a la que se encuentre conectado el proyecto.

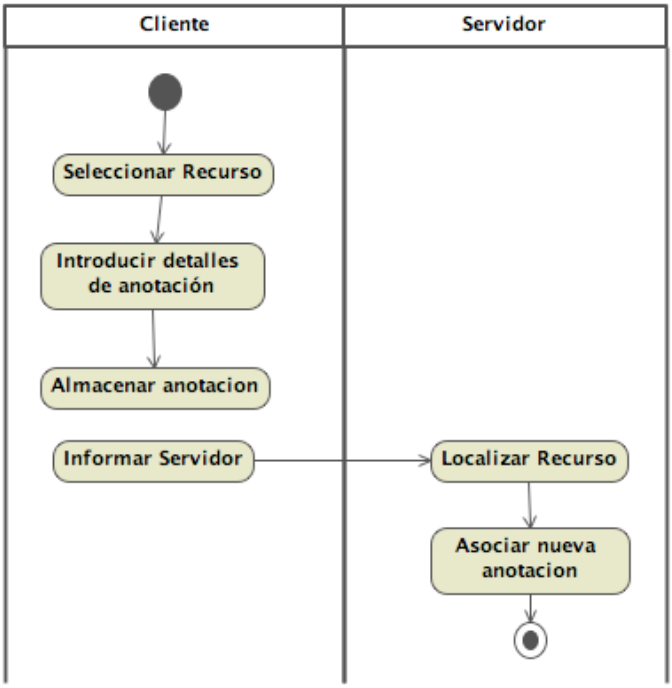


Figura. 2-8. Diagrama de la creación de una anotación por el cliente

Identificador: 8.2

Título: Edición en colaboración de un recurso

Precondiciones: Tener el proyecto asociado al recurso conectado y el usuario fuente tiene que haber iniciado la edición en colaboración del recurso.

Poscondiciones: Los contenidos del recurso habrán de ser los mismos entre todos los participantes de la sesión de colaboración.

Descripción: Para empezar a participar en la edición en colaboración de un recurso el usuario cliente deberá abrir el recurso deseado, y que previamente ha sido iniciado por la fuente. En este momento el usuario cliente estará en disposición de empezar a participar en la modificación del contenido del recurso.

A partir de este momento toda la secuencia de modificaciones deberá ser enviada en tiempo real a la fuente del proyecto, quien tiene el almacenamiento centralizado del proyecto.

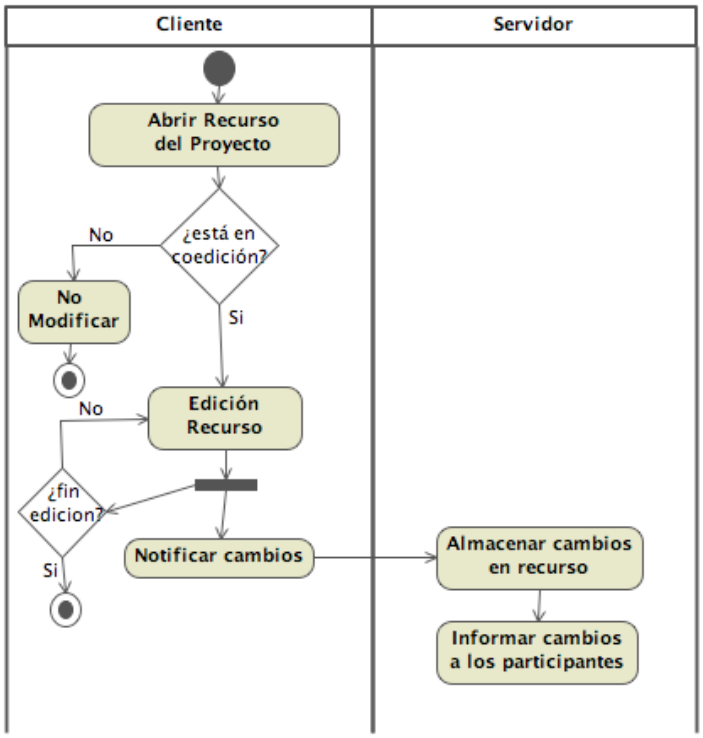


Figura. 2-9. Diagrama de actividad para la modificación en el cliente

3 DISEÑO

3.1 Restricciones en el diseño

En este apartado se exponen los errores conocidos en la versión actual del prototipo y restricciones a la hora de diseñar la aplicación. Estas restricciones han surgido del estudio de la plataforma en la que se va implementar la aplicación, así como de valorar la funcionalidad frente al coste de implementación.

Pretende ayudar a la utilización del mismo por parte del usuario, de forma que conozca las limitaciones del sistema, los errores que pueden aparece y otros detalles de manejo de la misma.

- La herramienta 'deshacer' en el trabajo colaborativo no ha sido adaptada. Por tanto, la operación deshace los últimos cambios realizados en el documento, ya sean con origen local o remoto. Por tanto se recomienda utilizar con cuidado esta funcionalidad del editor.
- Hay operaciones no soportadas sobre los recursos, por ejemplo, mover un recurso (o renombrar) en el proyecto fuente no hará que se manifieste los cambios en los recursos del cliente.
- No se toma como iniciado un recurso abierto en el momento de publicar un proyecto. Para su coedición se tendrá que reabrir.
- Se ha limitado a dos participantes el trabajo colaborativo. La idea inicial sugerida en un inicio de varios participantes se ha descartado debido, principalmente, al coste de diseño e implementación que conllevaba respecto a la funcionalidad real que podría dar.
- Por la misma razón que la anterior, no se ha implementado una edición síncrona del documento completa. Los participantes deben ponerse de acuerdo en quien edita en cada momento el documento.

3.2 Diagramas de interacción

A continuación se muestran los diagramas de secuencia realizados para el diseño del proyecto. Se han organizado en varios apartados, en primer lugar en función de si se trata de la fuente del proyecto o un cliente y dentro de cada apartado en relación a la funcionalidad que definen. Puesto que es un proyecto grande y va a formar parte de una plataforma existente, no se ha entrado en un detalle minucioso, puesto que contribuiría a una perdida de legibilidad y claridad en los diagramas.

3.2.1 Diagramas de la fuente del proyecto

3.2.1.1 Publicación de un proyecto

En la siguiente serie de diagramas se muestra cuales son las tareas que se realizan durante la publicación de un proyecto por parte del usuario fuente. Se ha contemplado una publicación correcta, sin contemplar las posibles situaciones de error o de excepción.

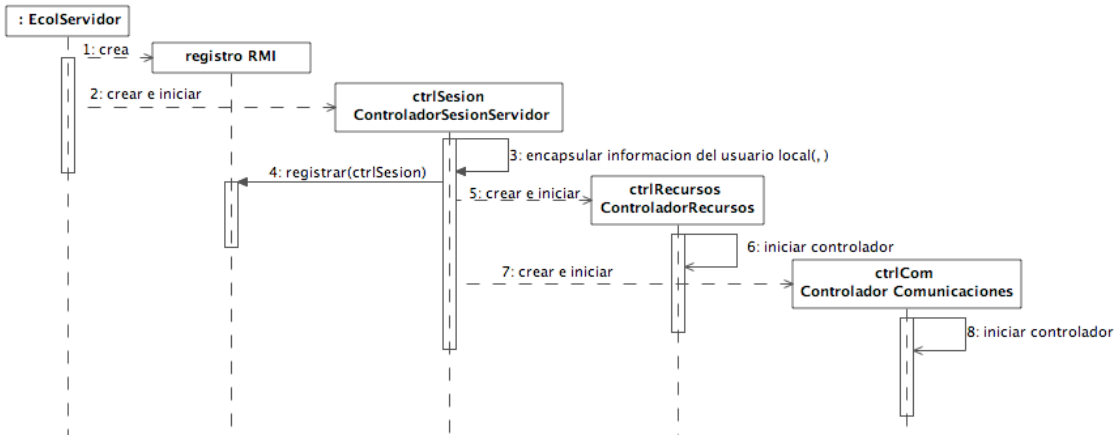


Figura 3–1. Diagrama principal de la publicación

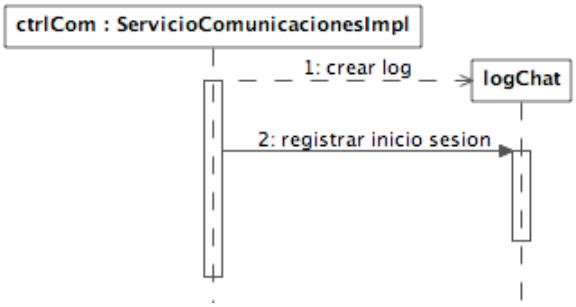


Figura 3–2. Iniciación del control de comunicaciones

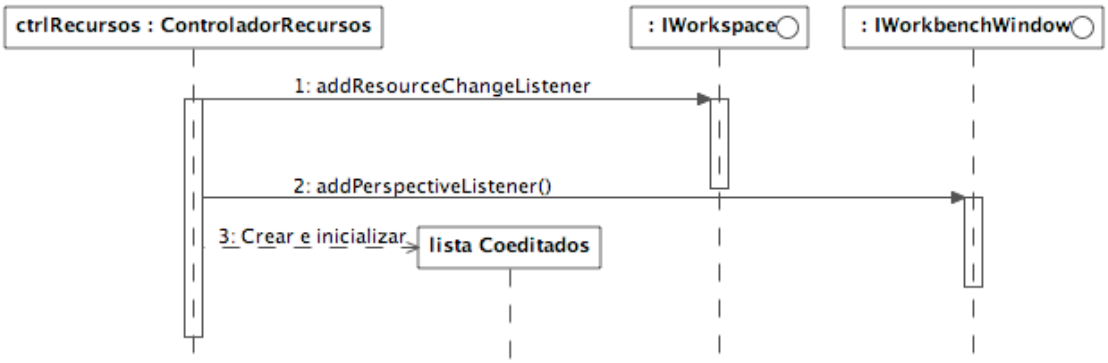


Figura 3–3. Inicialización del control de recursos

3.2.1.2 Modificación de recursos y coedición

En este apartado se muestran los diagramas asociados con la manipulación de los recursos en el proyecto fuente, como se manifiestan al cliente y como se realiza la coedición de los mismos.

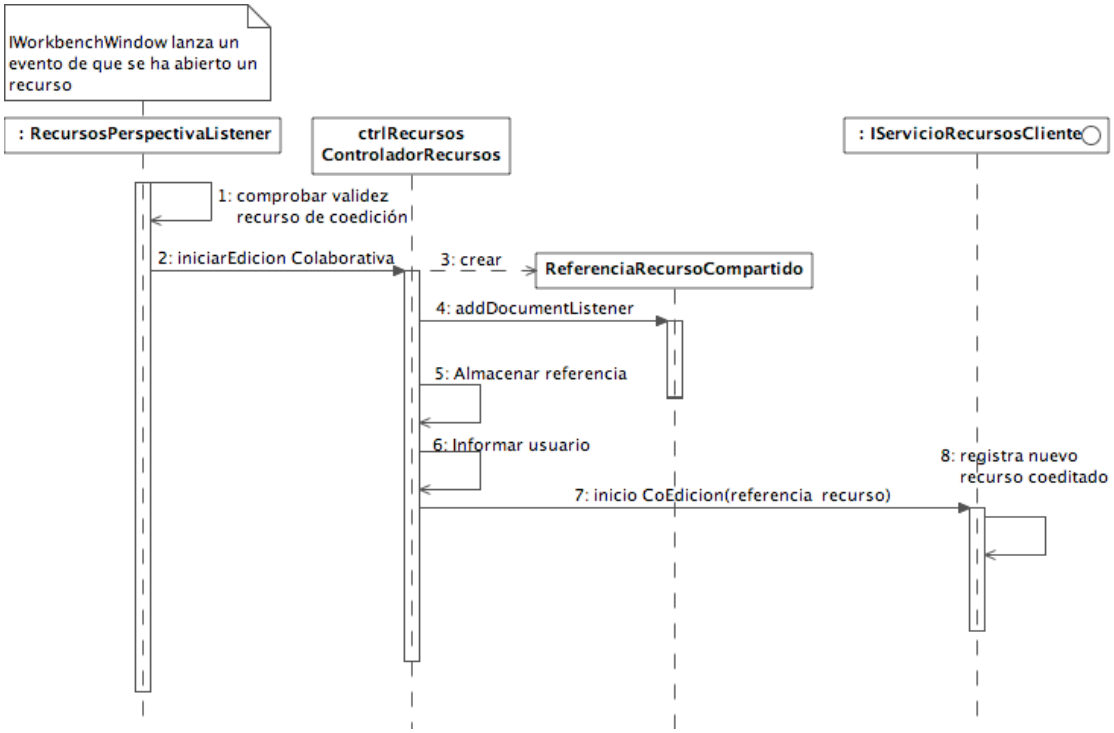


Figura 3–4. Inicio de colaboración en un recurso del proyecto

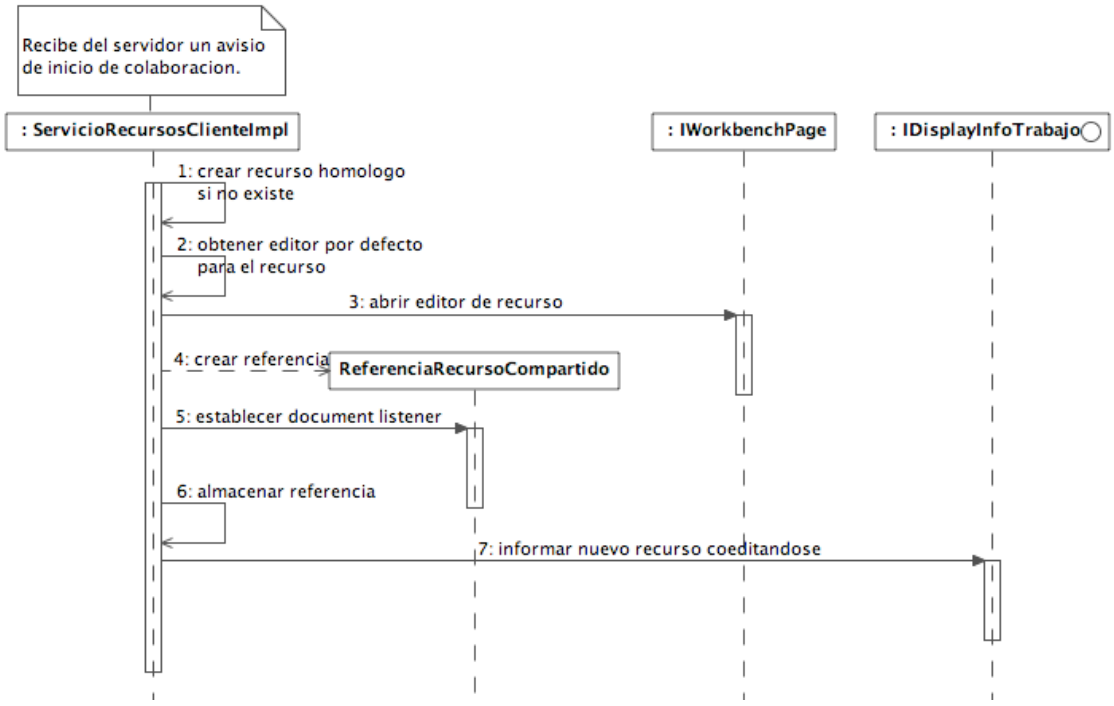


Figura 3–5. Secuencia en el cliente al ser informado del recurso a coeditar

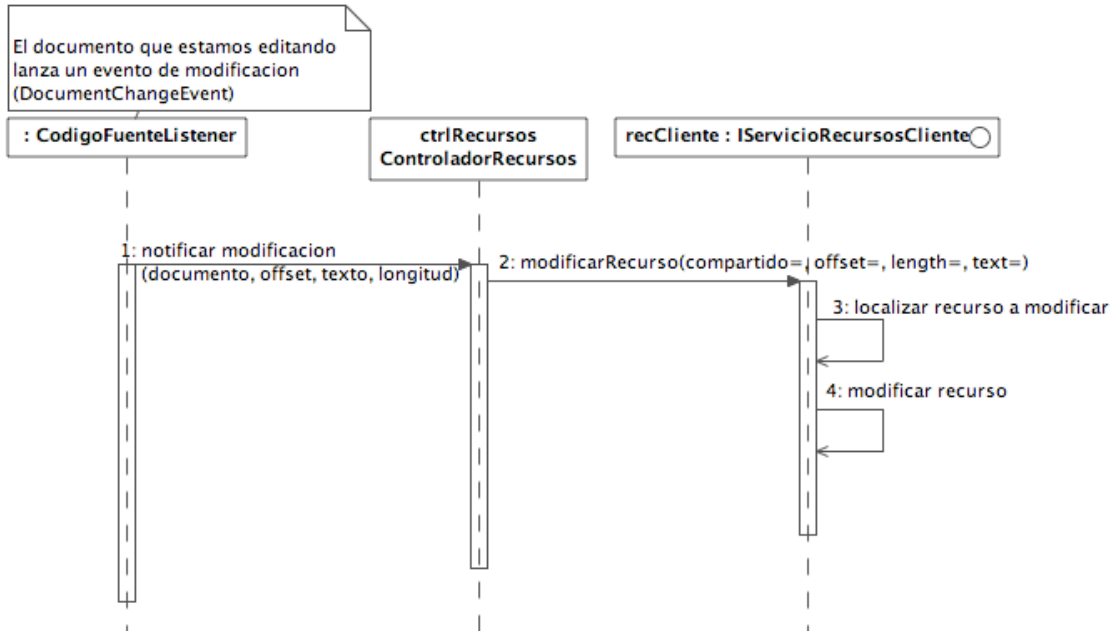


Figura 3–6. Secuencia de modificación de un recurso en coedición

Nuevo recurso

Los recursos creados en el entorno Eclipse se abren automáticamente, en ese momento se registrará con el listener y se iniciará la coedición. Como se puede ver en el diagrama asociado al inicio de la coedición, el cliente creará el recurso automáticamente si no lo tiene.

Anotacion

A continuación se muestra el diagrama de la secuencia para la creación de una nueva anotación.

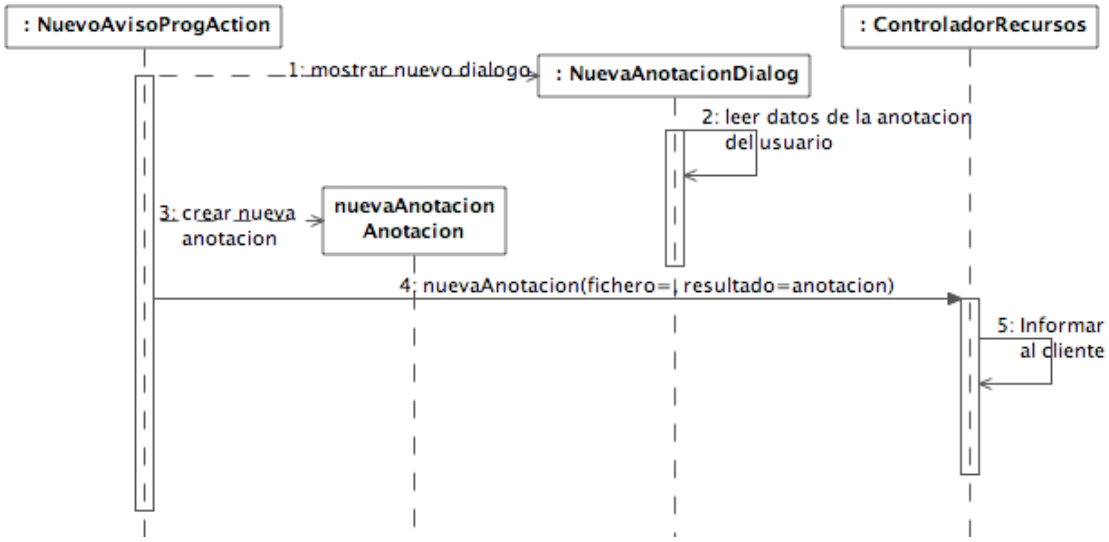


Figura 3–7. Diagrama de secuencia en la creación de una nueva anotación

Borrar recurso

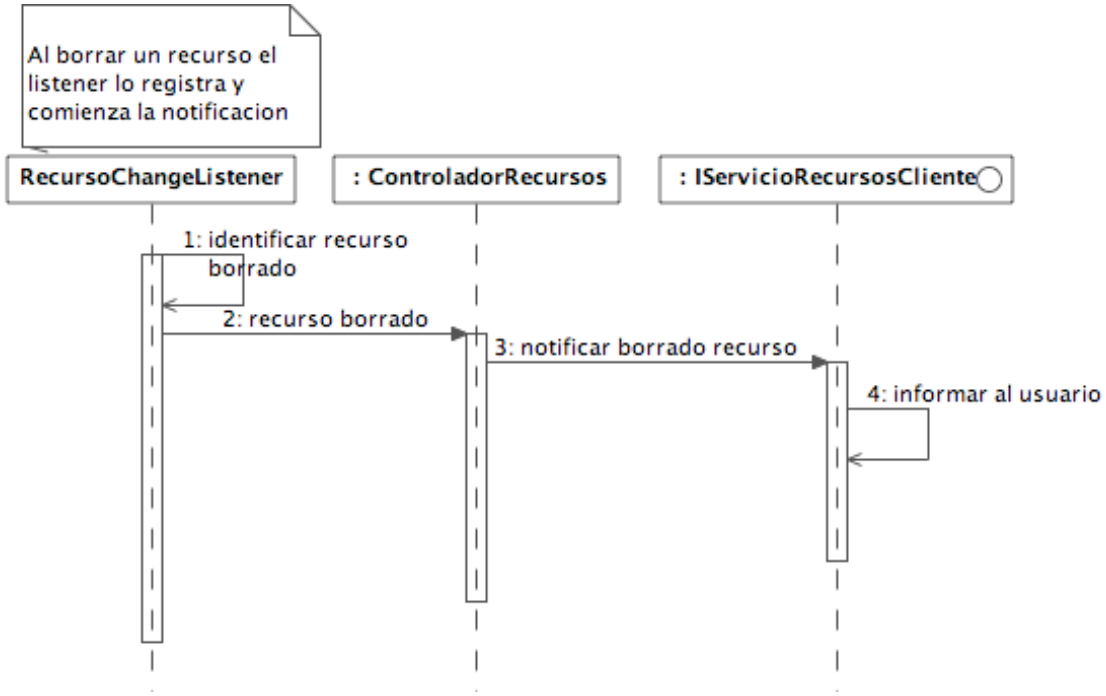


Figura 3–8. Diagrama de secuencia del borrado de un recurso

3.2.1.3 Envío de mensajes

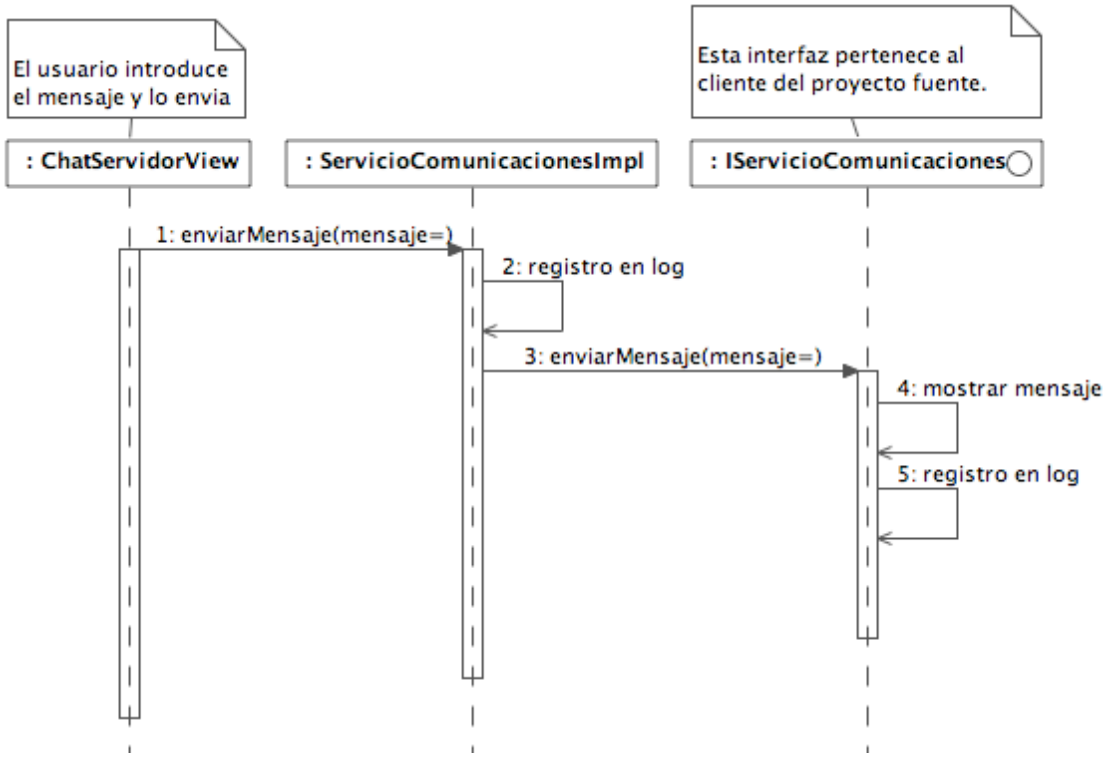


Figura 3–9. Diagrama de secuencia del envío de un mensaje

3.2.2 Diagramas del cliente

3.2.2.1 Conexión de un proyecto

En este apartado se muestran una serie de diagramas que representan las tareas realizadas en al conexión del cliente al proyecto fuente. Se han desglosado en varios

diagramas para facilitar la legibilidad y se ha contemplado el escenario de una conexión correcta a la fuente del proyecto.

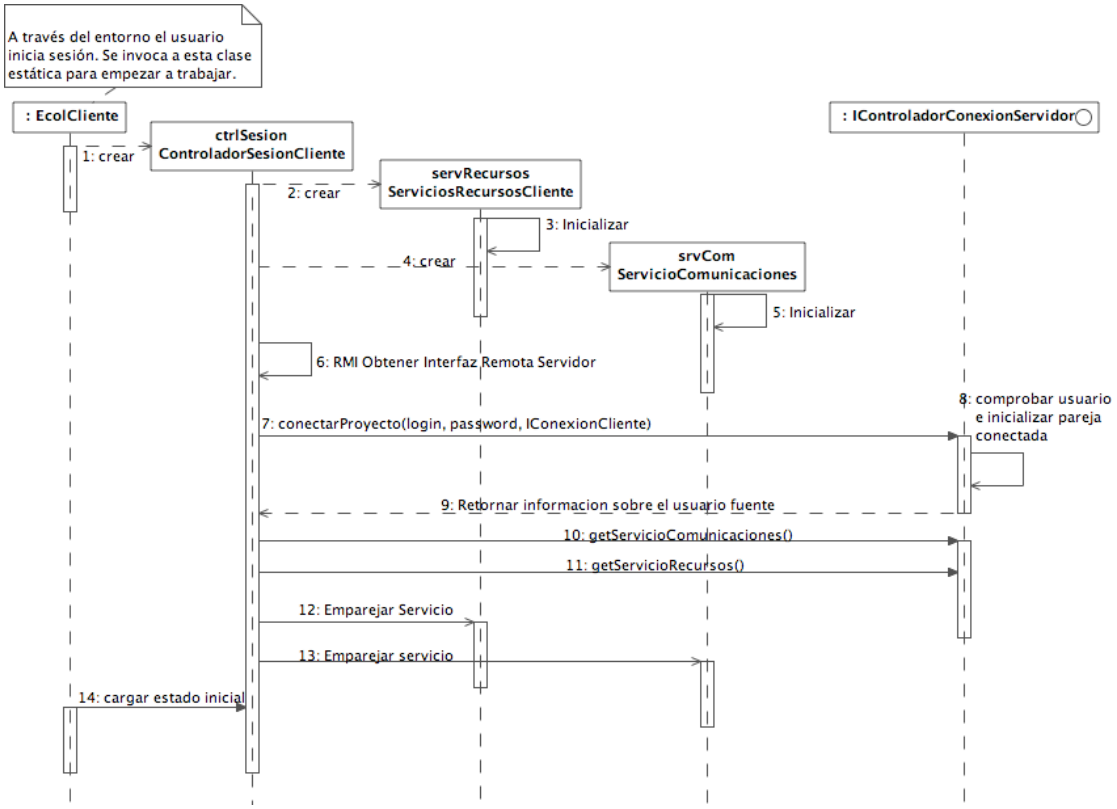


Figura 3–10. Diagrama principal de la conexión de un proyecto

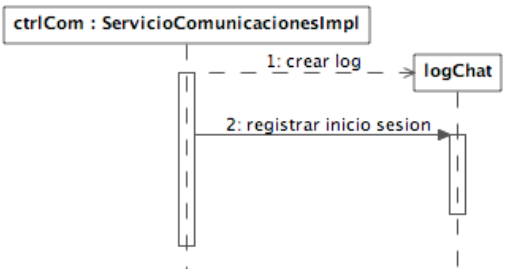


Figura 3–11. Inicialización del servicio de comunicaciones

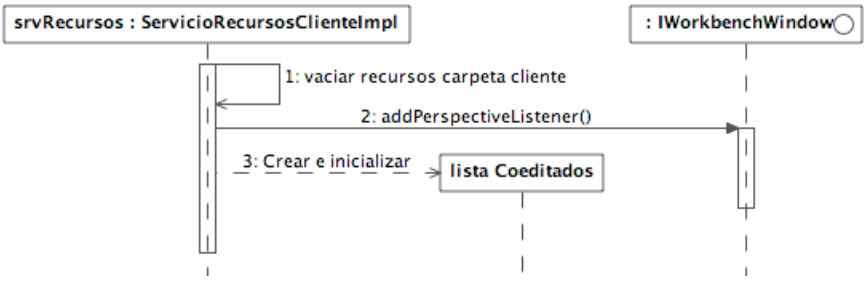


Figura 3–12. Inicialización del servicio de recursos

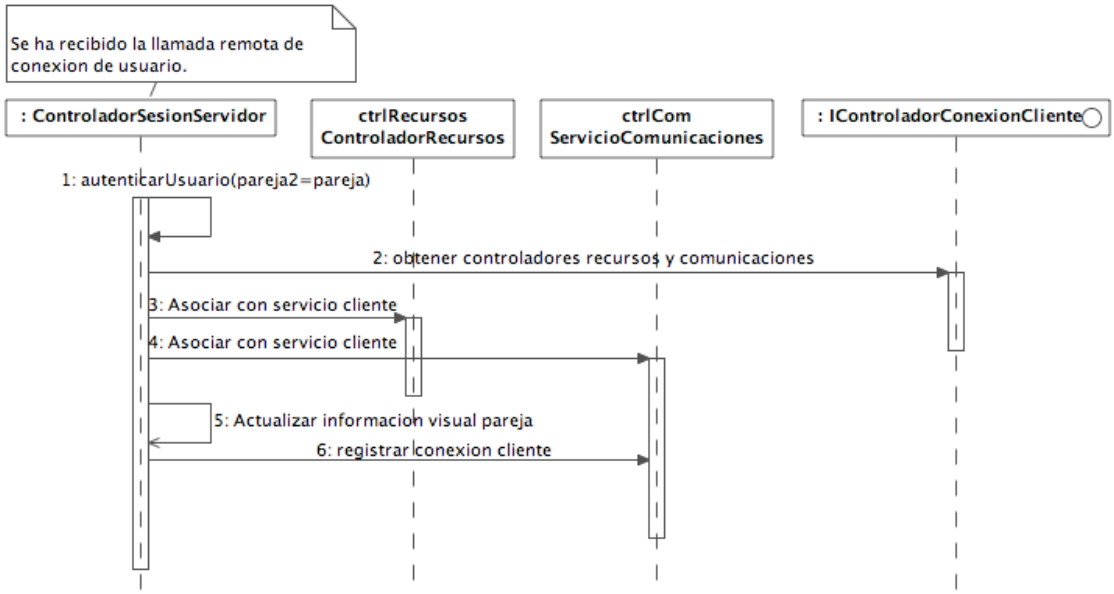


Figura 3–13. Establecimiento de la pareja en la fuente del Proyecto

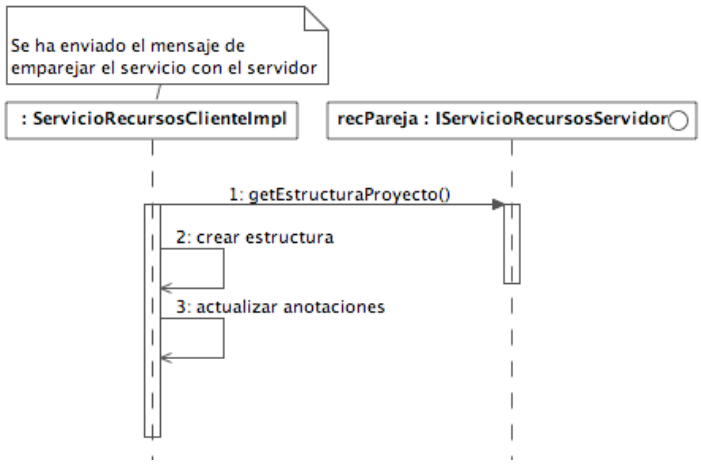


Figura 3–14. Diagrama de secuencia al emparejar el servicio de recursos en el cliente

3.2.2.2 Desconexión de un proyecto

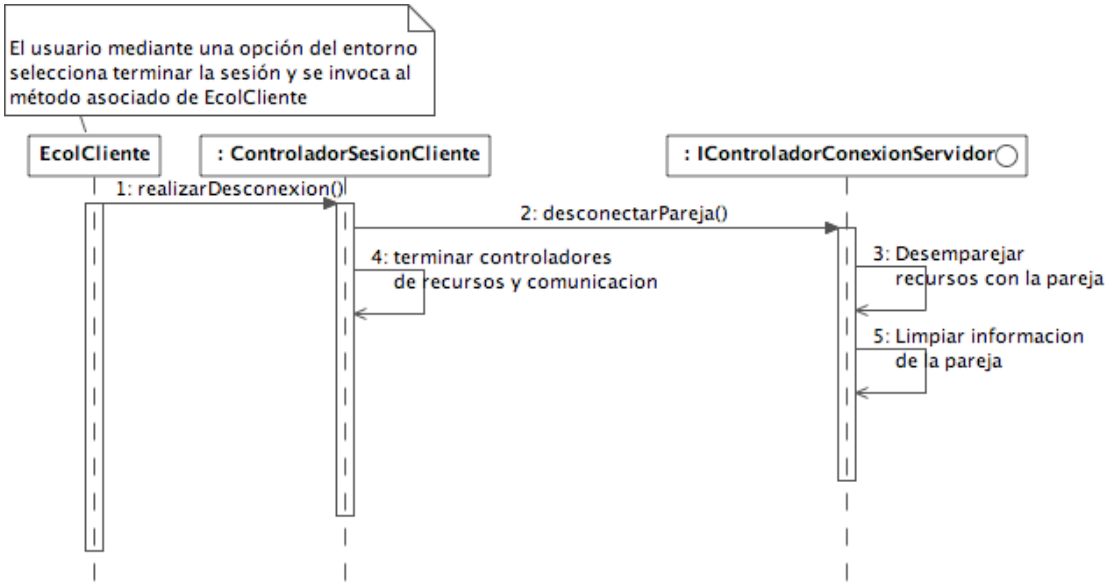


Figura 3–15. Diagrama de la secuencia básica en la desconexión de un cliente.

3.2.2.3 Modificación y coedición de recursos

Recibir inicio coedición del servidor

En la “Figura 3–5. Secuencia en el cliente al ser informado del recurso a coeditar” se puede ver el diagrama de secuencia asociado al inicio de colaboración cuando se notifica.

Coeditar un recurso

En este caso el diagrama representa la iniciativa del cliente para participar en la coedición, por ejemplo puede darse cuando se inicia la coedición, el cliente cierra el editor y vuelve a abrirlo (sin que el servidor haya cerrado el suyo y por tanto siga activo el recurso).

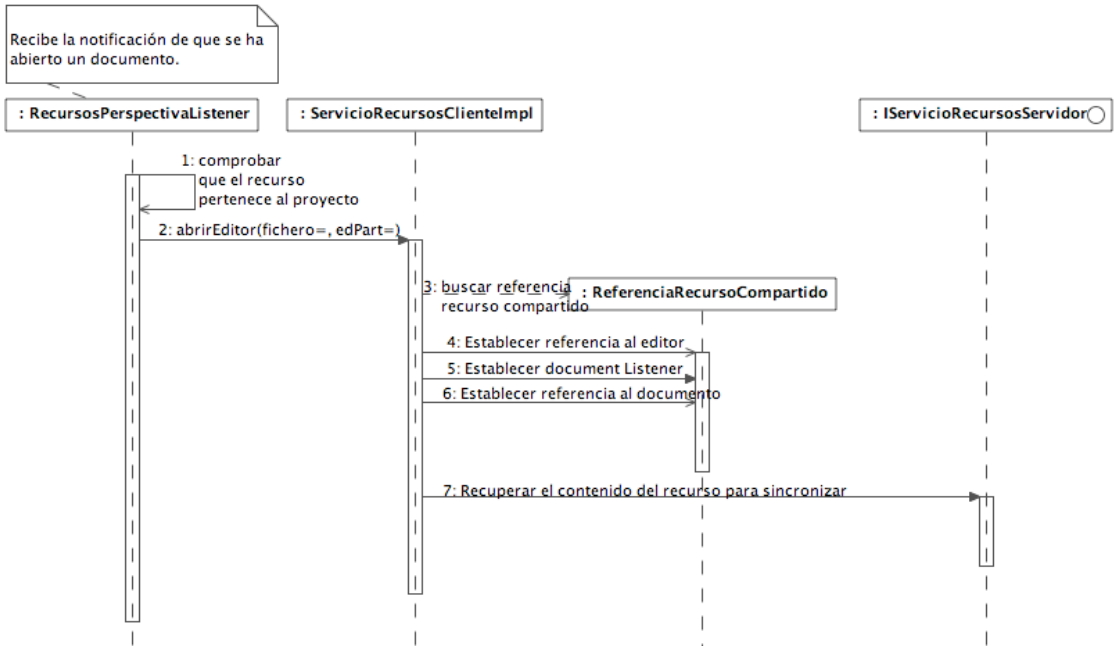


Figura 3–16. Secuencia al abrir un editor de un recurso en coedición

Modificar un recurso en coedición

Para la modificación de un recurso por parte del cliente se lleva a cabo la siguiente serie de pasos entre los diferentes controladores de los recursos de cliente y servidor.

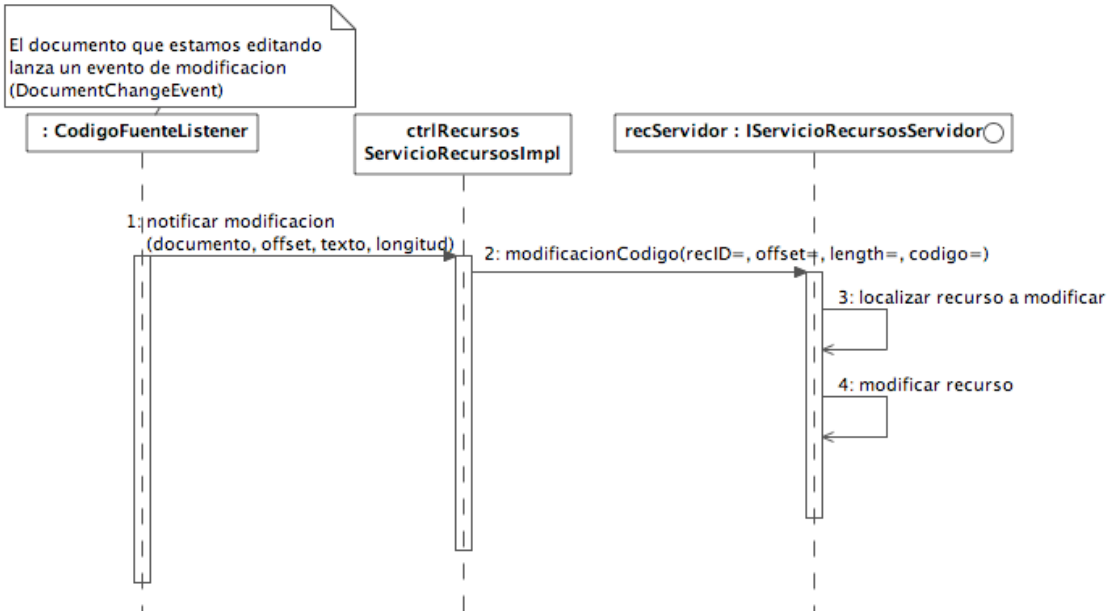


Figura 3–17. Diagrama de la modificación en colaboración de un recurso en el cliente.

Crear una anotación en un recurso

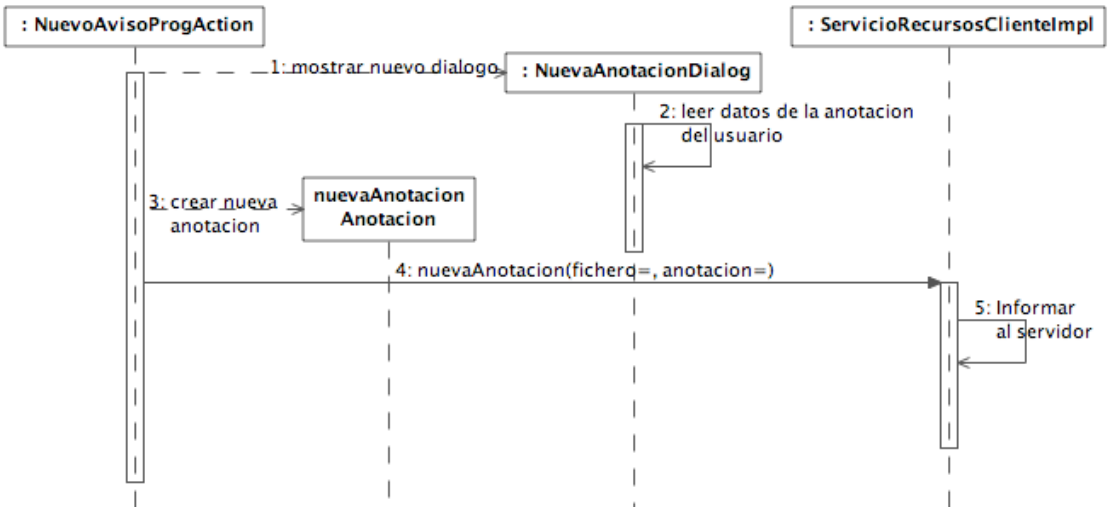


Figura 3–18. Diagrama de secuencia para crear una anotación.

3.2.2.4 Envío de mensajes

El diagrama de secuencia para el envío de mensajes en el cliente es muy similar al ya representado en el bloque del servidor, la única diferencia es el origen y destino de los mensajes, que en este caso parten del cliente.

3.3 Diagrama de clases

A continuación se muestran los diagramas de clases que han resultado del diseño del proyecto. Puesto que se trata de un proyecto con muchas clases se ha decidido clasificar el diagrama de clases en varios conjuntos para facilitar su lectura y entendimiento. Junto con cada diagrama se muestra un comentario explicativo.

Otro de los aspectos del proyecto que condiciona la representación del diagrama de clases es el tipo de proyecto, al tratarse de un plugin muchas de las clases se relacionan con clases propias del entorno, lo cual implicaría un diagrama demasiado complejo, en esas clases se hará una representación aproximada que simplifique y ayude a entender la organización del proyecto.

La representación del diagrama es la habitual salvo en el aspecto de al representación de las interfaces. La herramienta utilizada para la representación de los diagramas no usa la habitual etiqueta del estereotipo <<*interface*>>, en su lugar las representa con un círculo.

3.3.1 Fuente Proyecto

A continuación se muestra la representación del diagrama de clases en la parte de la fuente del proyecto.

El primer diagrama representa la organización principal del plugin, en él quedan reflejadas las clases principales que entran en juego para el trabajo en colaboración, listeners de perspectivas y recursos, controladores de recursos, comunicaciones, etc…

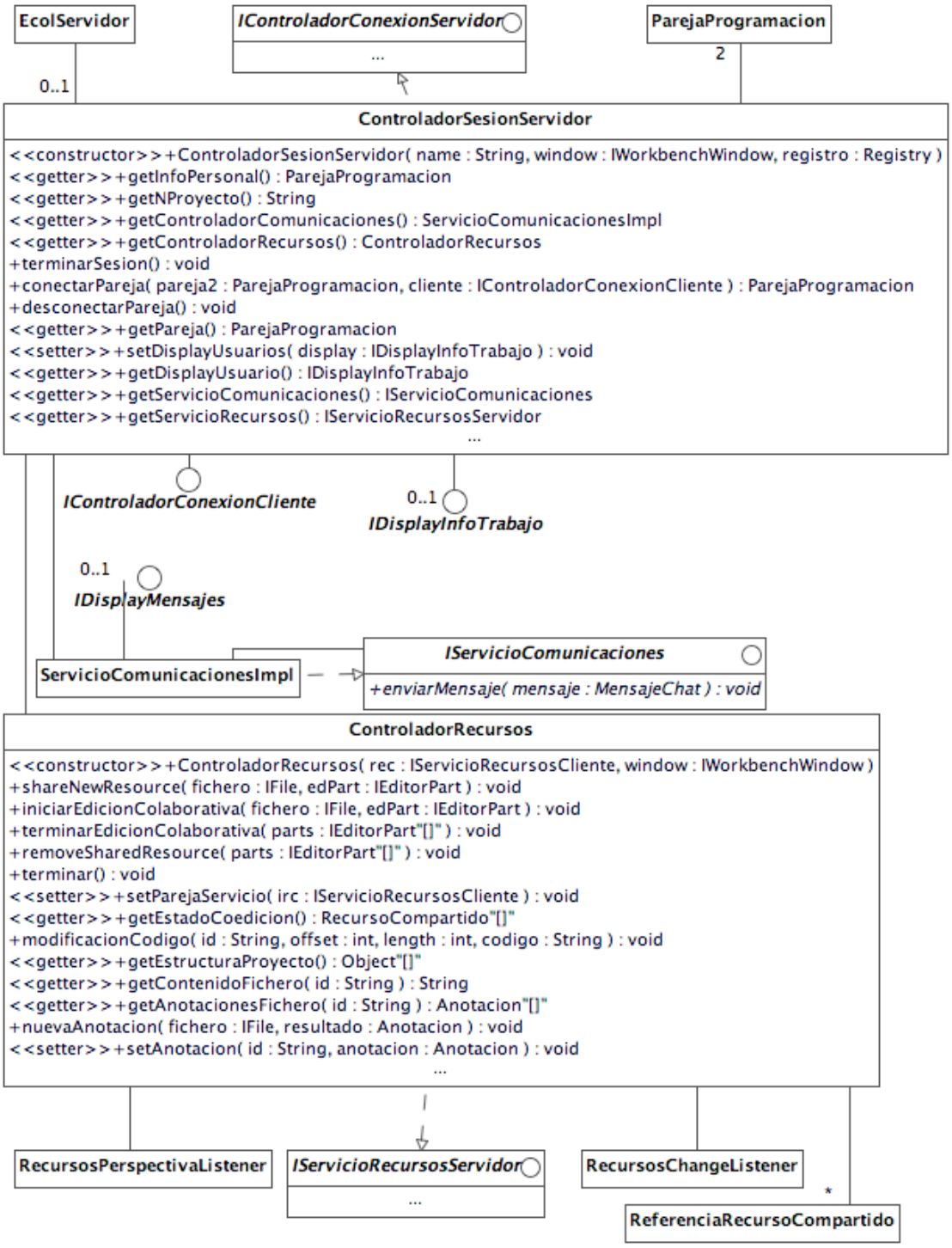


Figura 3–19 . Diagrama de clases general en la parte del servidor fuente

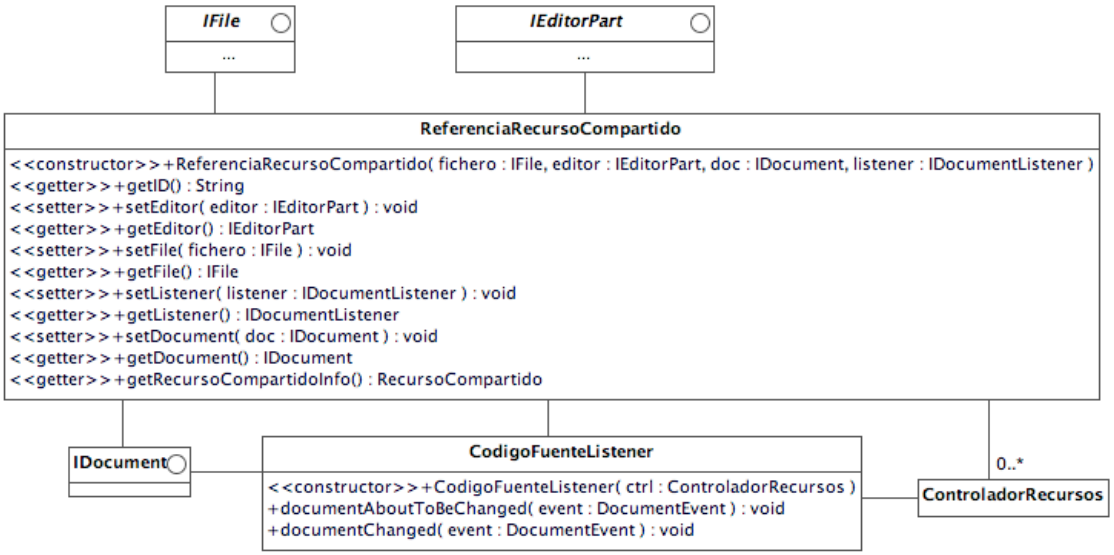


Figura 3–20. Diagrama en detalle de las referencias para la coedición

En el siguiente diagrama se representa la relación entre las clases que intervienen al iniciar la acción por parte de un usuario de lanzar el proyecto. También sirve para representar el termino de una sesión, puesto que serán las clase clases encargadas de dar al usuario.

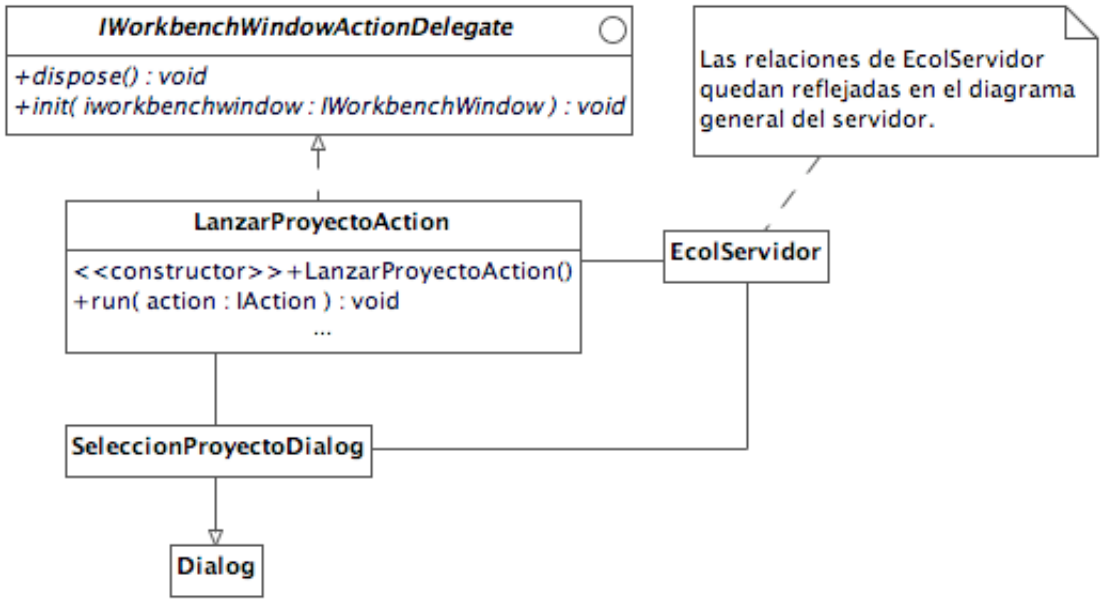


Figura 3–21. Diagrama de clases relacionadas con lanzar un proyecto

El diagrama que se muestra a continuación representa las clases asociadas a la creación de un nuevo proyecto.

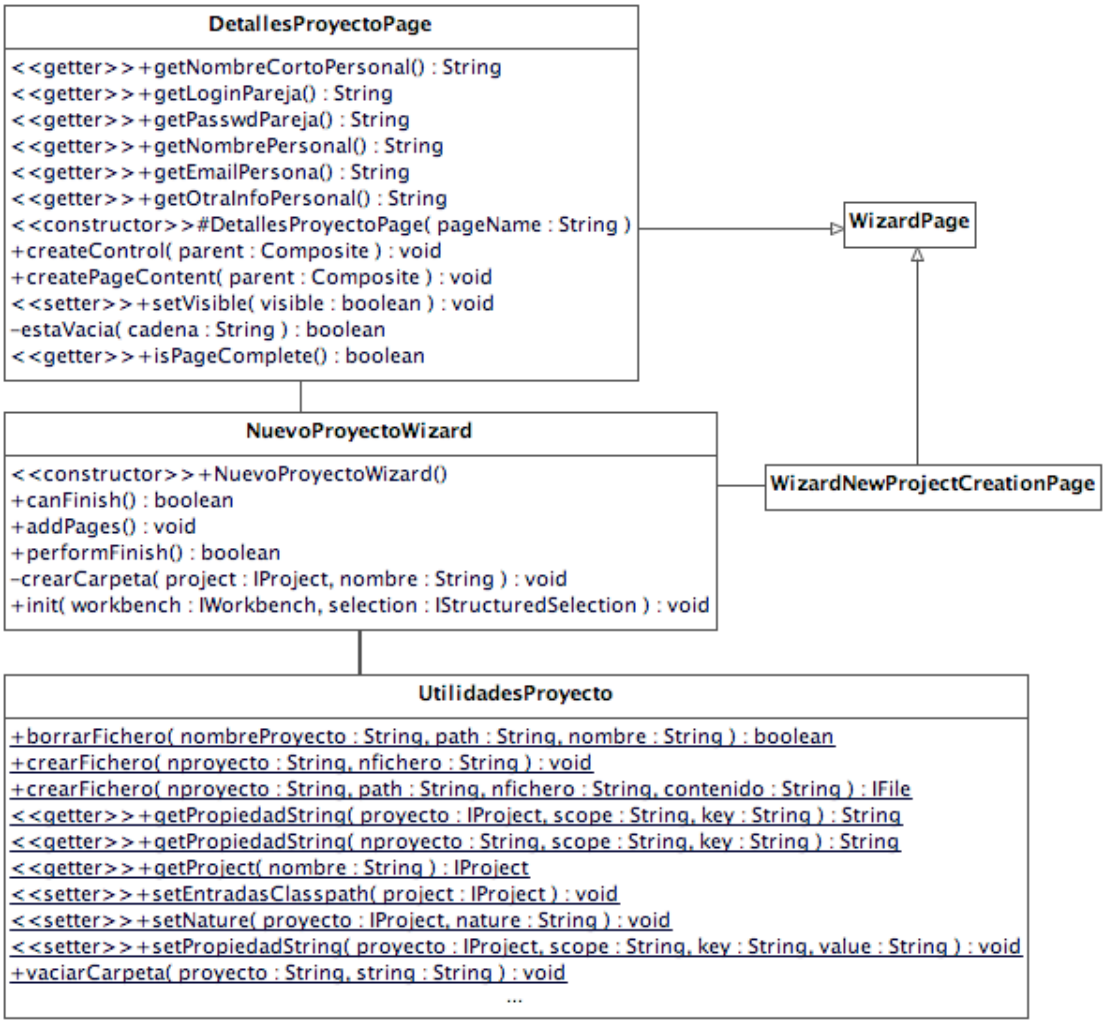


Figura 3–22. Diagrama de clases para la creación de un nuevo proyecto

Para la creación de anotaciones en los recursos entran en juego las siguientes clases de la acción, dialogo y control de recursos.

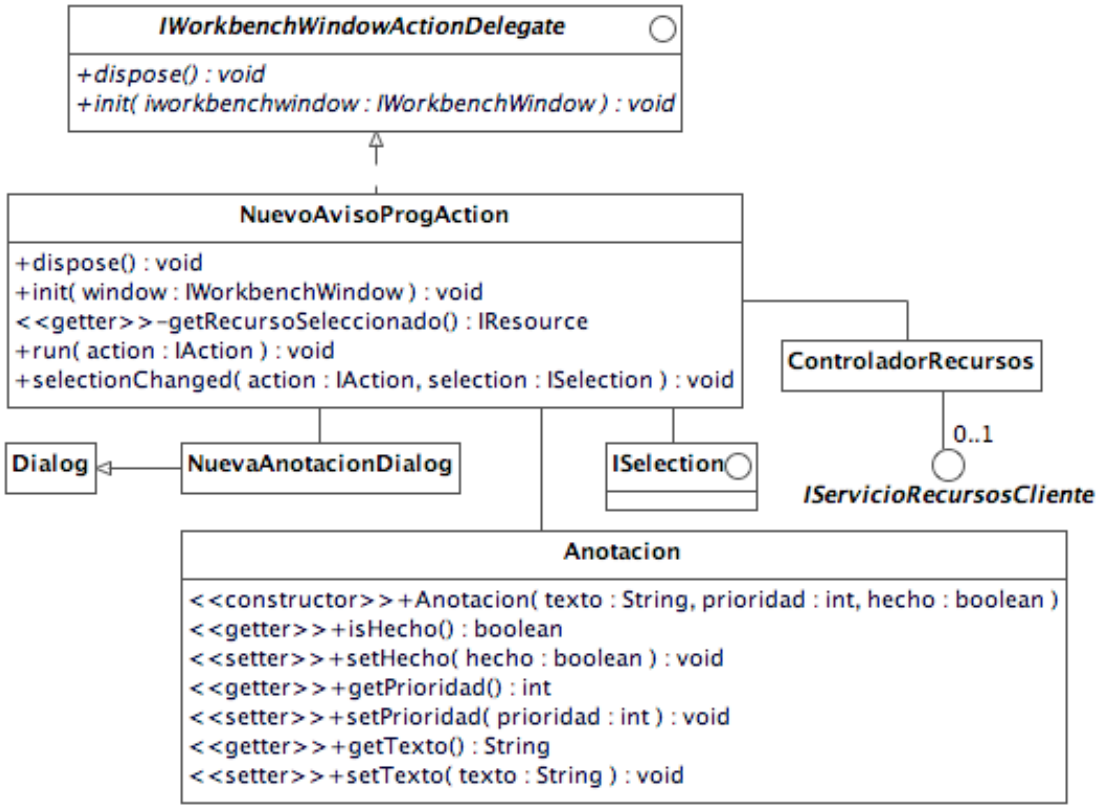


Figura 3–23.Diagrama de clases para la creación de anotaciones

3.3.2 Cliente Proyecto

En la figura Figura 3–24 podemos ver la representación general del proyecto en la parte del cliente.

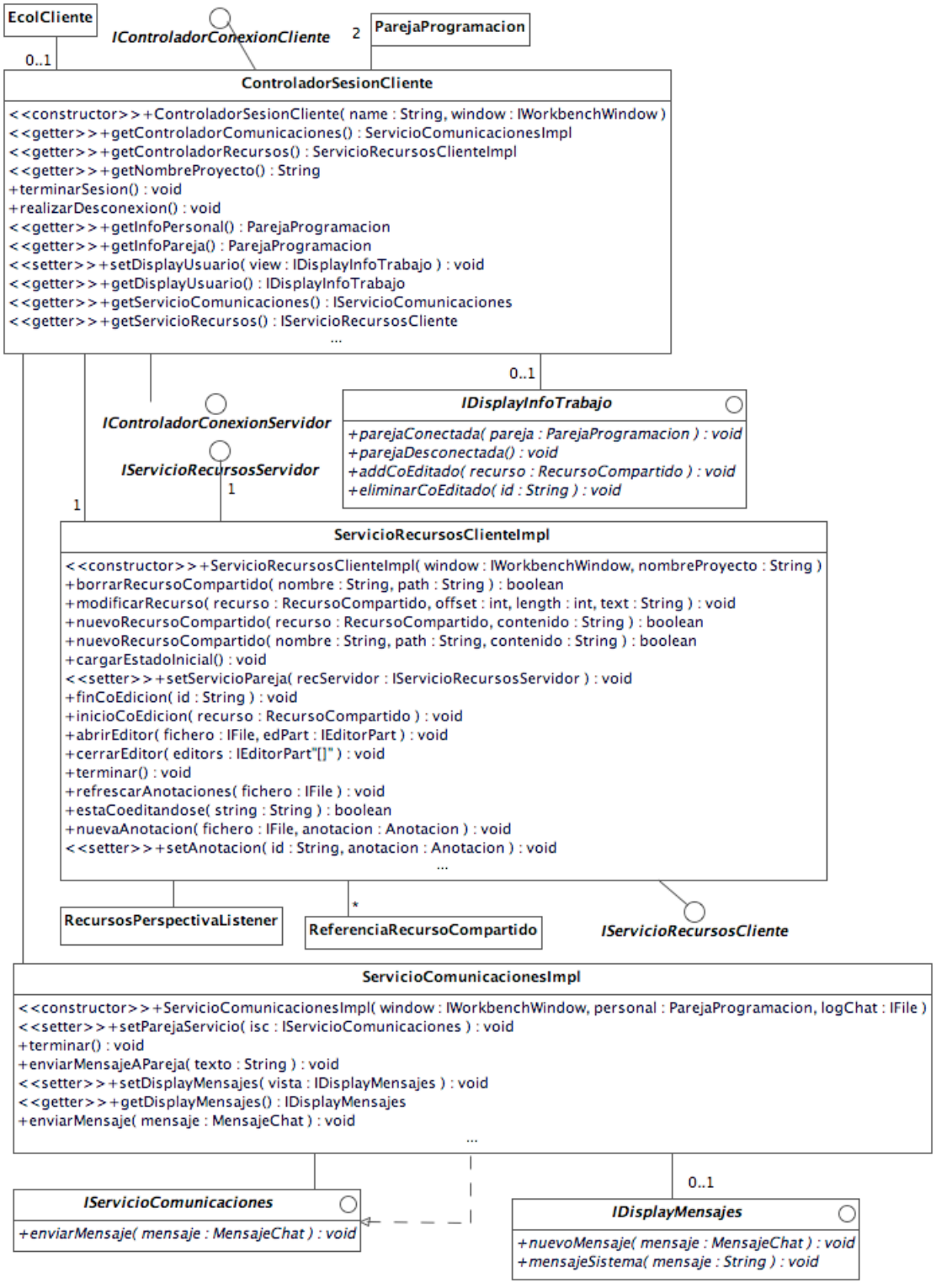


Figura 3–24. Diagrama de clases general del cliente

Las referencias a los recursos compartidos y sus detalles tienen ciertas diferencias en cuanto a la cardinalidad, puesto que puede haber registrado un recurso compartido pero que no se tenga abierto en cierto momento.

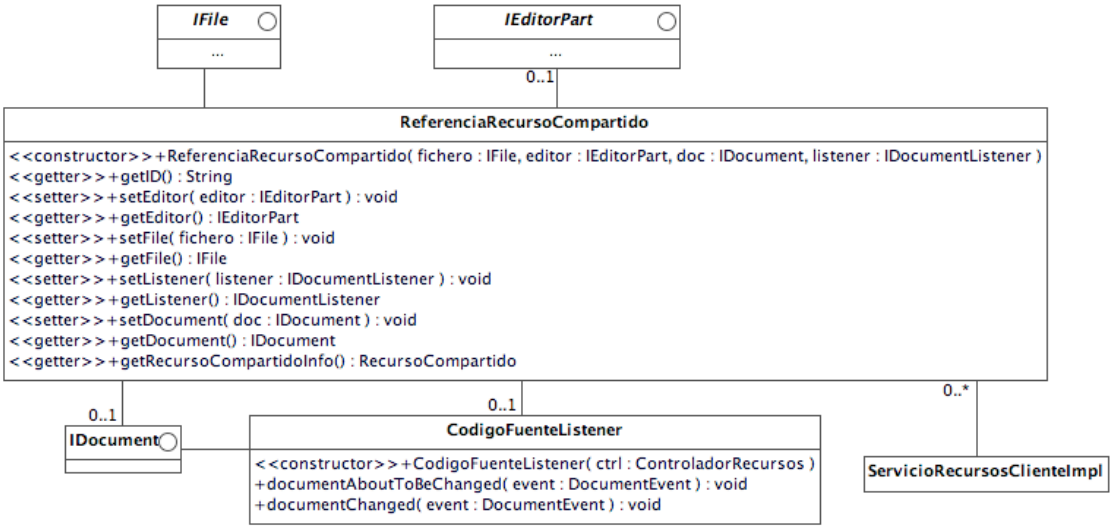


Figura 3–25. Diagrama de la organizacion de la referencia a un recurso

Para conectar un proyecto las clases diseñadas quedan representadas en el siguiente diagrama de clases, refleja esencialmente la acción propiamente, el resto de implementación se refleja en las asociaciones de `EcolCliente`.

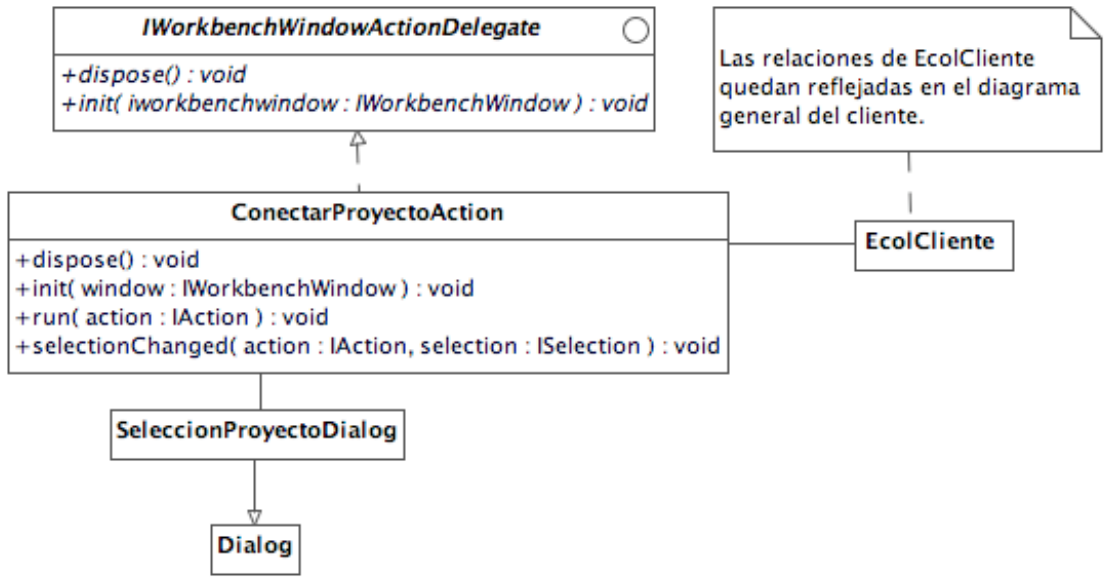


Figura 3–26. Diagrama de clases de la acción de conectar un proyecto

La creación de un nuevo proyecto queda determinada por la siguiente relación de clases:

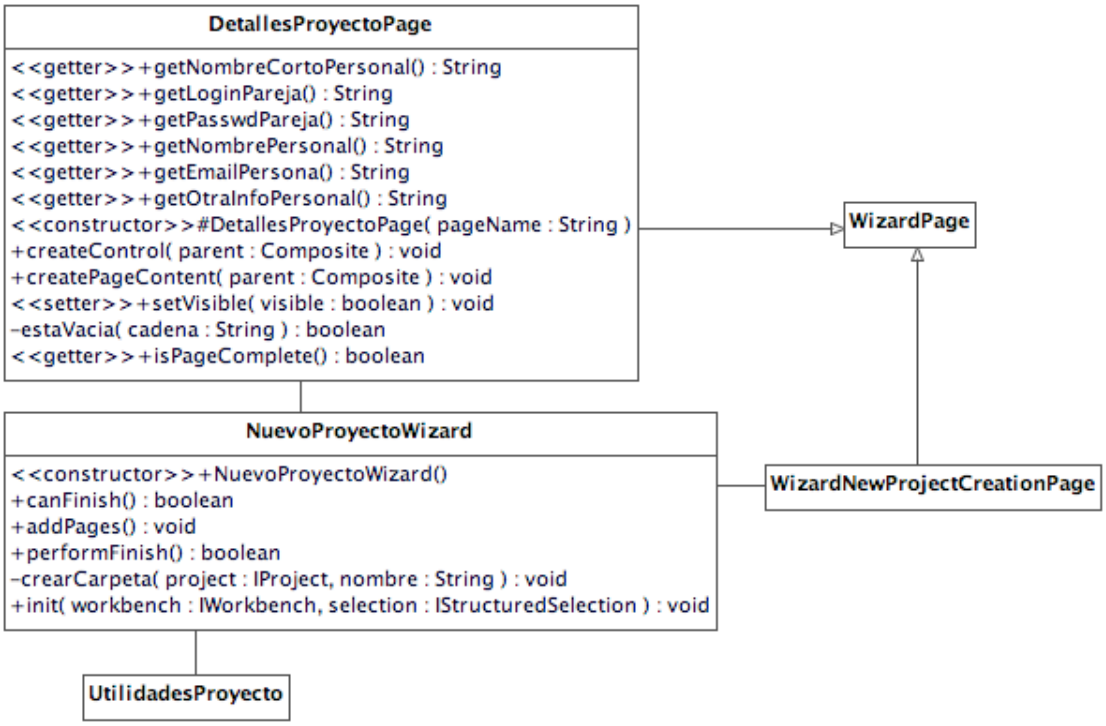


Figura 3–27. Diagrama de clases para la creación de un nuevo proyecto en el cliente

Para la creación de anotaciones en la parte del cliente, el siguiente diagrama de clases muestra las clases principales que intervienen.

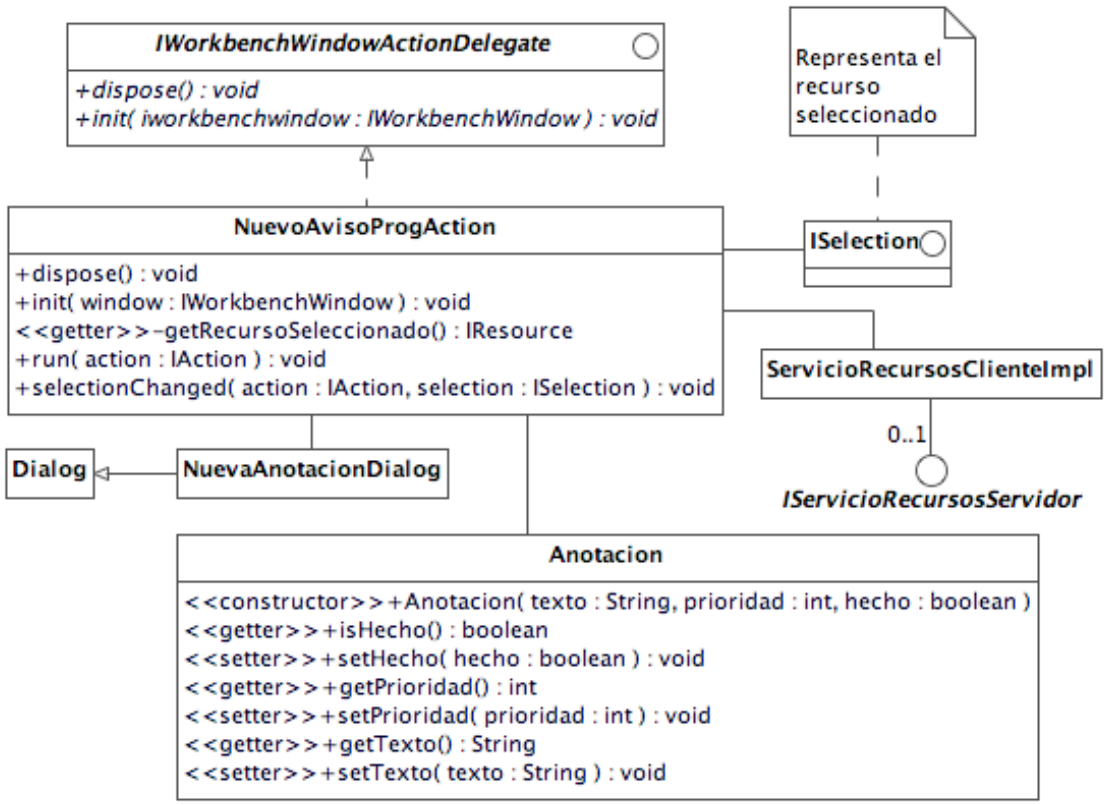


Figura 3–28. Diagrama de clases implicadas en la creación de una anotación

3.4 Diseño de la arquitectura de la aplicación

A continuación se comenta la arquitectura elegida para el diseño de la aplicación.

Se trata de un plugin para la plataforma de desarrollo de aplicaciones Eclipse. La comunicación entre los plugin del usuario fuente y cliente se hace siguiente una arquitectura de aplicación RMI. Finalmente, el almacenamiento maestro de los proyectos sobre los que se trabaja es mantenido por la fuente de proyecto.

En el siguiente diagrama se muestra una arquitectura aproximada que tendrá la aplicación a desarrollar:

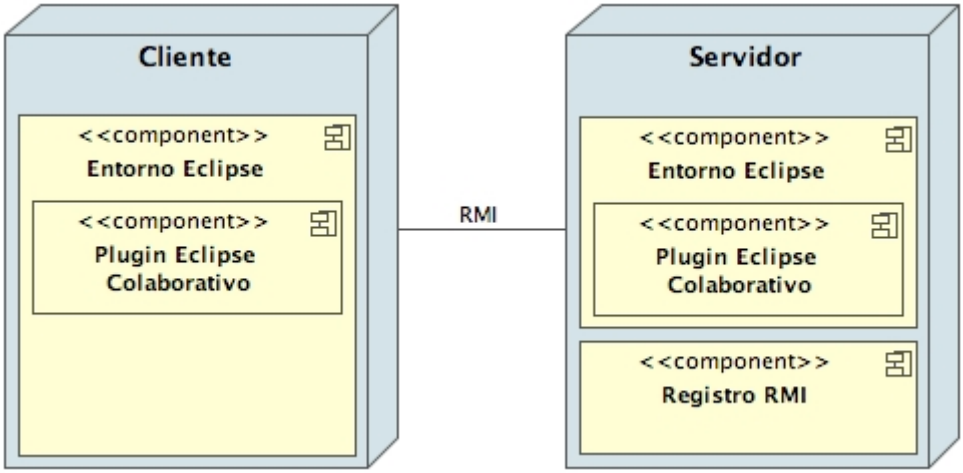


Figura 3–29. Diagrama de la arquitectura

Detalles a destacar de la arquitectura

La comunicación entre los dos extremos se realiza mediante RMI, como se comenta en el punto 4.2.3.1 la arquitectura RMI habitual dispone un servidor web para descargar los stub de los objetos remotos. Aun así, existe un diseño alternativo y más simple, consiste en distribuir con la propia aplicación todos los archivos de las clases necesarias, como en nuestro caso no se ven implicadas gran cantidad de clases en la arquitectura RMI esta solución es tangible y no se pierde en eficiencia. De esta manera cuando la aplicación requiera una determinada clase lo encontrará en el classpath local.

3.5 Diseño de la interfaz

Para el diseño de la interfaz del plugin se ha intentado seguir la estética y comportamiento que Eclipse presenta en su plataforma.

En primer lugar se ha definido una perspectiva de trabajo, tanto para el cliente como para el servidor de proyecto. Estas perspectivas (cuya implementación se comenta más detalladamente en 4.2.2.1.5), permiten organizar una serie de vistas y acciones del entorno para facilitar el trabajo del usuario en una tarea determinada.

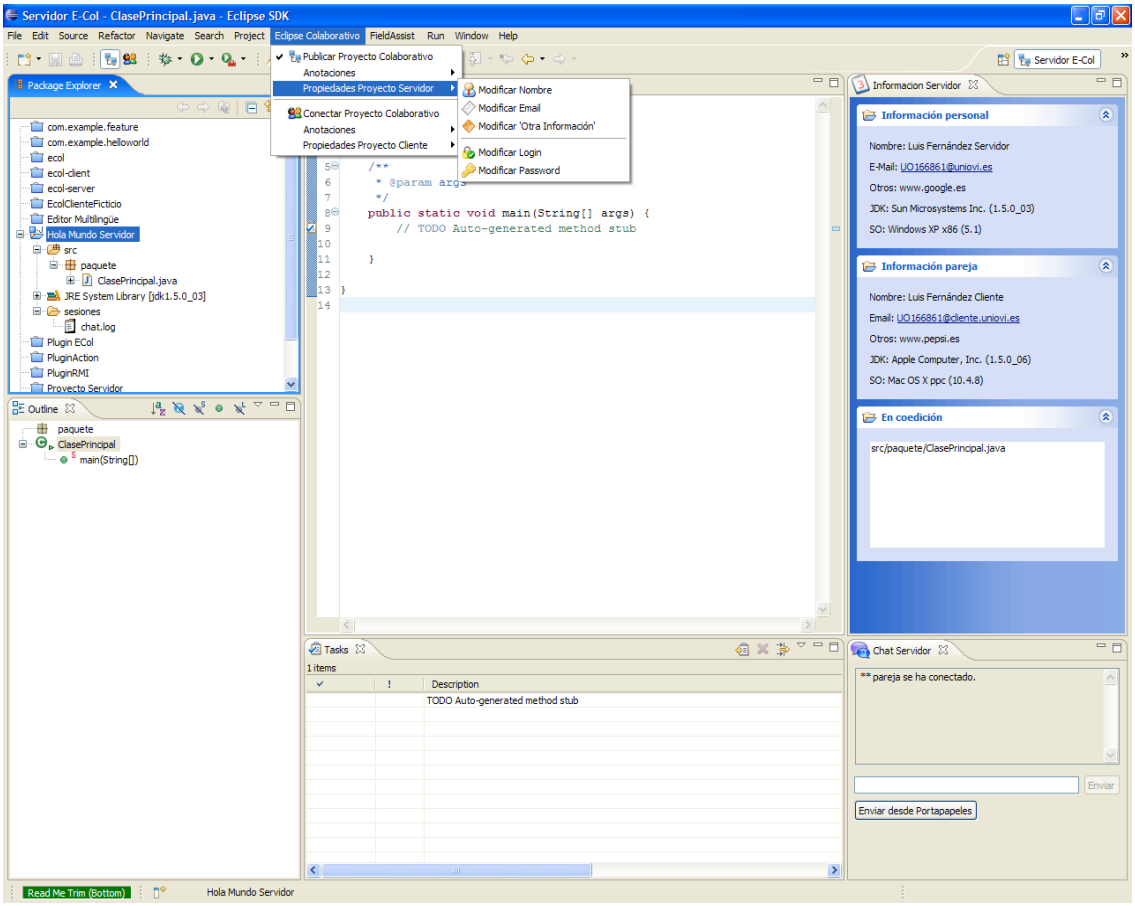


Figura 3–30 . Diseño del entorno

En la figura podemos observar las diferentes partes de las que se compone la perspectiva creada para el trabajo colaborativo de nuestro plugin.

- **Parte izquierda**:
  - Mitad superior: en esta parte se dispone un navegador del espacio de trabajo donde el usuario puede observar los proyectos con los que está trabajando. Los proyectos del proyecto, tanto cliente como servidor tienen un marcador gráfico para facilitar la visualización.
  - Mitad inferior: se muestra la organización jerárquica del elemento seleccionado, por ejemplo los métodos, atributos, etc… del archivo en edición.
- **Parte derecha**:
  - Vista superior: en este apartado se ha creado una vista especial para mostrar información sobre el trabajo en la sesión. Se muestra datos sobre

los participantes y una lista con los recursos que están activos para ser coeditados por los participantes.

- Vista inferior: en este panel se ha dispuesto un chat textual en el que el usuario se puede comunicar con los participantes del proyecto.

- **Parte centrar:**
  - En la parte central se dispone el panel de edición donde el usuario podrá editar los recursos del proyecto.
- **Parte inferior:**
  - En esta área dispondremos de los paneles de información referidos a tareas pendientes, problemas registrados en el proyecto o el panel de consola al ejecutar los proyectos.

Por otro lado, dejando a parte las perspectivas, se ha incluido en todas las opciones del menú principal del plugin, así como en las vistas creadas, iconos que faciliten la asociación de las opciones.

4 IMPLEMENTACIÓN

4.1 Convenios seguidos en la codificación

En este apartado se enumeran una serie de convenios que se han seguido en la implementación de la aplicación. Es recomendable tenerlos en cuenta para entender mejor la estructura del código implementado y la organización de los diferentes elementos.

Nombrado de las clases e interfaces

A continuación se muestran los convenios seguidos para el nombrado de las clases del proyecto:

- Todas las interfaces comienzan con la letra I. Ejemplo: `IControladorConexionCliente`.
- Todas las clases que representan elementos propios del entorno Eclipse, como vistas, acciones, etc.… se nombran añadiendo el sufijo del elemento que representan en inglés. Ejemplos: `ChatClienteView`, `TrabajoClientePerspective`, etc.
- Las clases que representan listeners del entorno se nombran añadiendo el sufijo Listener al nombre de la clase. Ejemplo: `RecursosPerspectivaListener`.

4.2 Tecnologías y herramientas

4.2.1 Introducción a Eclipse

Eclipse es una plataforma diseñada para desarrollar herramientas de desarrollo de aplicaciones. Lo más importante de la plataforma es su enfoque en un modelo de plugin que permite y facilita el desarrollo e integración de nuevas características a la plataforma.

4.2.1.1 Arquitectura básica de la plataforma

La arquitectura de la plataforma Eclipse está orientada al concepto de plugin. Estos plugin proveen una funcionalidad añadida a la plataforma en la que se ensamblan y a su vez establecen una serie de puntos de extensión, estos son lugares donde otros nuevos plugin pueden añadir nueva funcionalidad, esto convierte la estructura de la plataforma ampliamente extensible.

En la figura 3 se representa la plataforma básica y una serie de herramientas que extienden la funcionalidad de la misma.

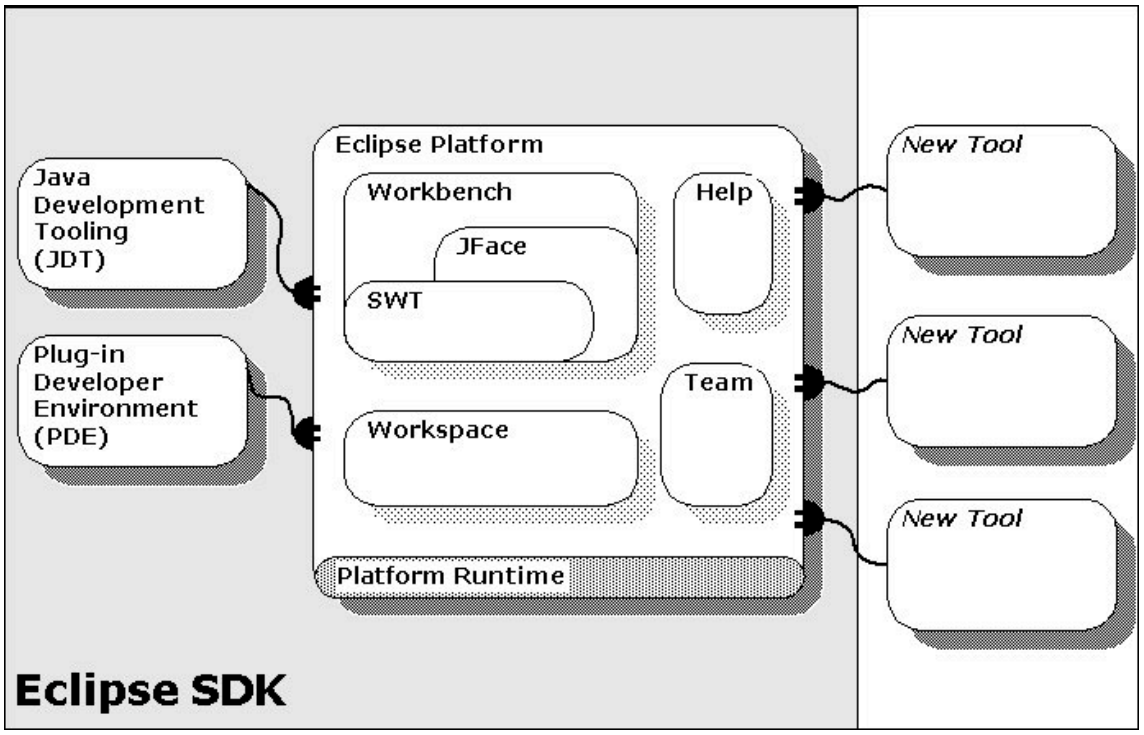


Figura. 4-1. Estructura de la Plataforma Eclipse

Runtime core

Es el componente que representa el núcleo de la plataforma. Este componente implementa el motor que se encarga de arrancar Eclipse y descubrir y lanzar dinámicamente los plugin.

La estructura de los plugin que el núcleo lanza vienen definidos por dos elementos básicos, un archivo de manifiesto OSGi y otro manifiesto en formato XML.

Una ventaja de esta implementación es que no se malgasta memoria ni rendimiento del sistema al no instalarse aquellos plugin que no son utilizados, es decir, los plugin son

registrados pero solamente serán activados cuando sea solicitada una funcionalidad del plugin.

Resource Management

Este componente define un modelo de gestión de los recursos común para todos los plugin que se vayan añadiendo al sistema. Establece tres elementos básicos para organizar y almacenar la información: proyectos, carpetas y ficheros.

Workbench UI

Se trata del plugin encargado básicamente de establecer unos puntos de extensión que permitan a nuevos plugin ampliar la funcionalidad en cuanto a menús, barras de herramientas, acciones, diálogos, asistentes, editores, etc.…

También se definen unos frameworks para desarrollar la interfaz de usuario, lo que facilita el desarrollo de la misma además de proporcionar un *look & feel* común en todo el entorno. Estos frameworks son:

- Standard Widget Toolkit (SWT): es un toolkit de bajo nivel independiente del sistema operativo.
- JFace UI: se encuentra en un nivel superior al anterior, se encarga de facilitar construcciones de diálogos, asistentes, etc.…

Team Support

Este plugin proporciona funcionalidades a otros plugin para implementar programación en equipos, acceso a repositorios y gestión de versiones.

Help System

Se trata de una plataforma basada en un servidor Web para proporcionar ayuda, facilitando así la integración de los documentos. El servidor permite a los plugin estructurar las referencias de ficheros mediante una estructura lógica de URL basada en los plugin y no en el sistema de ficheros.

Java Development Tools (JDT)

El JDT es uno de los plugin de la plataforma que especializa las características del Workbench en cuanto a edición, compilación, ejecución, etc.… de código Java.

Plug-in Development Environment (PDE)

El PDE es un plugin que proporciona, al igual que el anterior, una especialización del entorno. En este caso proporciona herramientas para automatizar la creación, manipulación, depuración y despliegue de nuevos plugin.

4.2.2 Desarrollar un Plugin en Eclipse

El plugin es el componente en el cual la plataforma Eclipse deposita toda su extensibilidad y potencia. Se tratan de unos programas que necesitan de las funcionalidades que ofrece la plataforma en la cual se incluyen y que a su vez amplían la funcionalidad de la misma mediante nuevas características.

Las posibilidades que ofrece el sistema de plugins, en el caso de Eclipse, son muy numerosas lo que ha contribuido sin duda a elegirlo como base para el desarrollo de nuestro proyecto.

La estructura que ha de seguir cualquier plugin dentro de la plataforma de Eclipse requiere básicamente tres archivos: plugin.xml, MANIFEST.MF y la clase java que sirve de activador. A continuación se hace un recorrido por estos tres elementos.

Plugin.xml

Este archivo define los plugins en la plataforma Eclipse. Se trata de un archivo XML que empieza con el elemento <plugin>, este elemento define el cuerpo del manifiesto. En el cuerpo del documento podemos encontrar declaraciones de nuevos puntos de extensión introducidos por el plugin, así como la configuración de extensiones ya definidas por otros plugins del sistema o el propio paquete.

Se puede encontrar más información acerca del formato de este archivo y como se declaran nuevos puntos de extensión y se configuran otros existentes en el documento “Eclipse platform plug-in manifest”.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension
      point="org.eclipse.ui.newWizards">
    <category
        id="ecol.comun.categorias.nuevoProyecto"
        name="Proyectos Colaborativos (E-dPP)"/>
    <wizard
        category="ecol.comun.categorias.nuevoProyecto"
        class="ecol.servidor.wizards.NuevoProyectoWizard"
        icon="icons/client.png"
        id="ecol.servidor.wizards.nuevoProyecto"
        name="Nuevo Proyecto de Fuente de Proyecto"
        project="true"/>
  </extension>
  <extension
      point="org.eclipse.ui.perspectives">
    <perspective
        class="ecol.cliente.perspectivas.PerspectivaTrabajoCliente"
        fixed="false"
        icon="icons/crow.gif"
        id="ecol.perspectivas.cliente"
        name="Eclipse dPP Cliente"/>
  </extension>
</plugin>
```

MANIFEST.MF

Este archivo de manifiesto incluido en el directorio META-INF almacena información descriptiva acerca del paquete en el que se incluye. Las cabeceras que se pueden establecer en este archivo vienen determinadas por la especificación del Framework OSGi R4, aun así existen servicios opcionales que no son incluidas con la implementación que hace Eclipse del framework. De igual manera la implementación de Eclipse añade nuevas opciones que no forman parte del estándar OSGi

Una de estas opciones es *Eclipse-LazyStart*. Esta cabecera es usada para especificar si el paquete debe iniciarse automáticamente antes de que la primera clase o recursos sea accedida en el paquete. Esto modelo permite empezar a Eclipse con los elementos necesarios.

Otra opción útil es *Eclipse-PlatformFilter* que nos permite un filtro para la plataforma en la que integramos el paquete. Se puede encontrar más información y más detallada en el documento “OSGi Bundle Manifest Headers” dentro de la documentación de Eclipse SDK.

Ejemplo:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Plugin Eclipse Colaborativo
Bundle-SymbolicName: ecol;singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: ecol.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.ui.ide,
 org.eclipse.jdt.core,
 org.eclipse.jdt.ui,
 org.eclipse.ui.editors,
 org.eclipse.ui.workbench.texteditor,
 org.eclipse.jface.text
Eclipse-LazyStart: true
```

Activador Java

El activador es la clase encargada de controlar el ciclo de vida de un plugin dentro de la plataforma Eclipse.

Su nombre se decide al desarrollar el paquete y se especifica en el fichero *MANIFEST.MF* visto anteriormente con la directiva *Bundle-Activator*.

Esta clase Java debe heredar de la clase `org.eclipse.ui.plugin.AbstractUIPlugin`. Esta clase especifica métodos especificos para iniciar y parar el plugin. Esta clase no necesita de una implentación especial a la hora de desarrollar un plugin.

Ejemplo:

```
import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

public class Activator extends AbstractUIPlugin {

	public static final String PLUGIN_ID = "ecol";

	private static Activator plugin;

	public Activator() {
		plugin = this;
	}

	public void start(BundleContext context) throws Exception {
		super.start(context);
	}

	public void stop(BundleContext context) throws Exception {
		plugin = null;
		super.stop(context);
	}

	public static Activator getDefault() {
		return plugin;
	}
}
```

Puntos de extensión

Para ofrecer esta nueva funcionalidad los plugin establecen una serie de puntos de extensión. Este es un concepto importante puesto que toda la funcionalidad de la aplicación se sustentará en saber utilizar de manera adecuada y eficiente todas estas extensiones.

Los puntos de extensión se componen de diferentes elementos, todos ellos se definen mediante la especificación de un DTD, este DTD es el encargado de definir la estructura, y los tipos de datos que se admiten en cada uno de los campos del punto de extensión. No se entra en detalle en el formato de los DTD pero en las sucesivas explicaciones se dará por supuesto un conocimiento básico de los mismos.

A continuación se muestra una especificación DTD de un elemento de un punto de extensión a modo de ejemplo y la definición en el archivo plugin.xml por parte de un plugin que emplea dicho elemento dentro del punto de extensión al que pertenece.

```
<!ELEMENT extension (category | view | stickyView)*>
<!ATTLIST extension
point              CDATA #REQUIRED
id                 CDATA #IMPLIED
name               CDATA #IMPLIED>

<!ELEMENT      view     (description?)     >
<!ATTLIST      view
Id                      CDATA #REQUIRED
Name                    CDATA #REQUIRED
Category                CDATA #IMPLIED
Class                   CDATA #REQUIRED
Icon                    CDATA #IMPLIED
fastViewWidthRatio      CDATA #IMPLIED
allowMultiple           (true | false)      >
```

```
<extension
point="org.eclipse.ui.views">
<view
category="ecol.vistas.VistasServidor"
class="ecol.servidor.vistas.ChatServidorView"
icon="icons/chat.png"
id="ecol.vistas.servidor.ChatServidorView"
name="Chat Servidor"/>
</extension>
```

En los siguientes apartados se muestra detalladamente todos los puntos de extensión que se han empleado en el desarrollo del proyecto, comentando en todos ellos sus detalles de implementación, los problemas que han aparecido en cada punto y las soluciones encontradas.

4.2.2.1 Puntos de Extensión empleados

4.2.2.1.1 Vistas

Las vistas son partes del workbench en las que se muestra al usuario información dispuesta jerárquicamente (cómo puede ser el navegador de recursos de un proyecto) o mostrar propiedades de objetos. Sólo una instancia de cada vista es abierta en una página del workbench.

El punto de extensión que en el cual un plugin ha de registrarse en su archivo *plugin.xml* para extender esta funcionalidad con nuevas vistas es: `org.eclipse.ui.views`. La especificación DTD de la vista propiamente es:

```
<!ELEMENT extension (category | view | stickyView)*>

<!ATTLIST extension
point     CDATA #REQUIRED
id        CDATA #IMPLIED
name      CDATA #IMPLIED>

<!ELEMENT view (description?)>
```

```
<!ATTLIST view
id                      CDATA #REQUIRED
name                    CDATA #REQUIRED
category                CDATA #IMPLIED
class                   CDATA #REQUIRED
icon                    CDATA #IMPLIED
fastViewWidthRatio      CDATA #IMPLIED
allowMultiple           (true | false) >
```

Los campos principales para la definición de una vista son su:

- Identificador (id): un nombre único que será el identificativo para referirse a esta vista en el entorno. Ejemplo: `plugin.vistas.VistaInformacion`.
- Nombre (name): un nombre que será el utilizado en el entorno de usuario para referirse a la vista. Deberá ser por tanto un nombre legible y entendible por las personas. Ejemplo: Vista de Información.
- Clase (class): en este apartado se especifica la clase que se encargará de implementar la nueva vista. Lo habitual es heredar de la clase `org.eclipse.ui.part.ViewPart` que implementa la funcionalidad por defecto de las vistas, ahorrándonos el trabajo de tener que implementarlo nosotros si usásemos la interfaz `org.eclipse.ui.IViewPart`.
- Icon (icon): es una ruta relativa dentro de la estructura del proyecto al icono que representará a la vista.
- Categoría (category): Un identificador compuesto por los ID de las categorías de la vista separados por el símbolo '/'. Las referencias a las categorías deben existir.

Para mejorar la organización de los plugin, podemos organizar las vistas que creemos en nuestros proyectos mediante categorías. Estas categorías son definidas desde el mismo punto de extensión que las vistas. Su especificación DTD se muestra a continuación.

```
<!ELEMENT category EMPTY>
<!ATTLIST category
id                      CDATA #REQUIRED
name                    CDATA #REQUIRED
parentCategory          CDATA #IMPLIED>
```

Los campos son bastante simples, destacando parentCategory que permite la creación de jerarquías de categorías, para ello este campo debe ser rellenado por identificadores de categorías separados por el símbolo '/'. El resto son el id, identificador único de la categoría y el name, nombre legible y representativo de la categoría que será por el que se muestre en el entorno.

4.2.2.1.2 ActionSets

Este punto de extensión (definido en `org.eclipse.ui.actionSets`) es utilizado para añadir menús, elementos de menus y elementos de barras de herramientas al workbench.

Para mejorar el orden en el entorno todos estos elementos son agrupados en conjuntos de acciones (action sets). Estos conjuntos de acciones serán accesibles desde el dialogo disponible en *Window > Customize Perspective* (un análisis más detallado de las perspectivas está en el apartado 4.2.2.1.5).

Las acciones definidas deben implementar, generalmente, la clase `org.eclipse.ui.IWorkbenchWindowActionDelegate`, aunque esto dependerá del tipo

que se establezca al definir la acción. La especificación DTD el primer elemento que deberemos definir a la hora de extender nueva funcionalidad a los action sets es:

```
<!ELEMENT extension (actionSet+)>
<!ATTLIST extension
point     CDATA #REQUIRED
id        CDATA #IMPLIED
name      CDATA #IMPLIED>


<!ELEMENT actionSet (menu* , action*)>
<!ATTLIST actionSet
id                CDATA #REQUIRED
label             CDATA #REQUIRED
visible           (true | false)
description       CDATA #IMPLIED>
```

Como vemos en la especificación este elemento se compone de varios elementos menú y action. Los campos son bastante intuitivos, el identificador único para designar al set de acciones, la etiqueta label para darle un nombre legible y representativo en el entorno y el campo descripción para describir el conjunto de acciones. El único campo un poco particular es visible, se encarga de especificar si este conjunto de acciones estará visible o no por defecto en una perspectiva que no haya sido personalizada.

Para definir el primero de los elementos (*action*) se especifica la siguiente plantilla DTD:

```
<!ELEMENT action (selection* | enablement?)>
<!ATTLIST action
id                          CDATA #REQUIRED
label                       CDATA #REQUIRED
accelerator                 CDATA #IMPLIED
definitionId                CDATA #IMPLIED
menubarPath                 CDATA #IMPLIED
toolbarPath                 CDATA #IMPLIED
icon                        CDATA #IMPLIED
disabledIcon                CDATA #IMPLIED
hoverIcon                   CDATA #IMPLIED
tooltip                     CDATA #IMPLIED
helpContextId               CDATA #IMPLIED
style                       (push|radio|toggle|pulldown) "push"
state                       (true | false)
pulldown                    (true | false)
class                       CDATA #IMPLIED
retarget                    (true | false)
allowLabelUpdate            (true | false)
enablesFor                  CDATA #IMPLIED>
```

Los campos esenciales para crear una nueva acción son:

- *id*: es el identificador único de la acción por el cual se referirá a ella la plataforma.
- *label*: es el nombre representativo y legible que se mostrará en el entorno al usuario.
- *menubarPath*: es una secuencia de identificadores de menú separados por el elemento '/'. El último elemento ha de ser el grupo dentro del menú donde será añadida la acción (más información a continuación en la definición de menús).
- *toolbarPath*: es una ruta delimitada por el símbolo '/'. El primer token especifica el identificador de la barra de herramientas, el segundo token es el nombre del grupo de la barra de herramientas donde la acción será añadida.
- *style*: este atributo define el estilo del interfaz de usuario para la acción. Los valores posibles son los especificados en la DTD.

- *class*: define el nombre de la clase que implementa la funcionalidad de la acción. Esta clase debe implementar `org.eclipse.ui.IWorkbenchWindowActionDelegate` o, en el caso de que el estilo definido sea pulldown, implementar `org.eclipse.ui.IWorkbenchWindowPulldownDelegate`.
- *enablesFor*: establece el número de elementos seleccionados en el entorno para que se habilite esta opción. Esto es bastante útil, puesto que las acciones reciben los elementos seleccionados del entorno, y si nuestra acción solo trabaja sobre un elemento aplicando esta restricción nos facilitará la implementación y la usabilidad. Los formatos del atributo son: **!** (0 elementos), **?** (0 ó 1 elementos), + (1 ó más), **multiple**, **2+** (2 ó más), **n** (n elementos), ***** (cualquier número de elementos).

Finalmente, para definir los menus en los que se disponen las acciones que se hayan definido deberemos añadir la información de acuerdo a la siguiente plantilla DTD:

```
<!ELEMENT menu (separator+ , groupMarker*)>
<!ATTLIST menu
id      CDATA #REQUIRED
label   CDATA #REQUIRED
path    CDATA #IMPLIED>
```

La especificación de los menús muestra la existencia de los elementos separador, estos elementos funcionan a modo de 'slots' dentro de los menús, en ellos podremos disponer acciones de una manera ordenada, estos separadores se definen simplemente con un nombre. Los campos del elemento menú son muy básicos, limitándose a los habituales identificador y etiqueta, y al atributo *path* que define la ruta delimitada por el símbolo '/', esta ruta establece donde estará dispuesto el menú y se compone de una serie de identificadores de menú, siendo el último una referencia a un separador.

4.2.2.1.3 Project Natures

Las naturalezas de proyecto permiten a los plugin etiquetar los proyectos con tipo específico, por ejemplo los proyectos java tienen su propia etiqueta "Java nature".

El punto de extensión asociado a las naturalezas de proyecto es: `org.eclipse.core.resources.natures`. Para implementar funcionalidad específica se debe implementar la clase `org.eclipse.core.resources.IProjectNature`, sus detalles se comentará en apartados posteriores.

Los proyectos pueden tener varias naturalezas y se permite la definición de restricciones sobre dichas naturalezas de proyecto:

- **One-of-nature**: establece que la naturaleza es una de un conjunto determinado, en dicho conjunto las naturalezas son excluyentes. De esta manera sólo una de las naturalezas del conjunto es aplicado a un proyecto.
- **Requires-nature**: mediante esta restricción establece que la naturaleza requiere de otra para ser establecida en un proyecto.

La configuración DTD para este punto de extensión se muestra a continuación:

```
<!ELEMENT extension (runtime , (one-of-nature | requires-nature | builder | content-type)* , options?)>
<!ATTLIST extension
point   CDATA #REQUIRED
id      CDATA #REQUIRED
name    CDATA #IMPLIED>
```

En este caso la especificación se realiza en el primer elemento del punto de extensión, es decir cuando especificamos que vamos a utilizarlo, en el resto de casos lo habitual es

que el elemento a definir sea un subelemento y podamos definir varios, en este caso si deseamos definir varias naturalezas deberemos especificar varias veces que vamos a usar el punto de extensión.

El campo identificador es un identificador único que nos permitirá referirnos a la naturaleza, por ejemplo al atribuir una naturaleza a un proyecto deberemos usar este identificador. El nombre es un campo opcional para nombrar la naturaleza.

En lo que respecta a los subelementos de la especificación, destacamos las restricciones comentadas, estas son fácilmente definibles puesto que solo tienen un campo cuyo valor será el identificador de la naturaleza requerida o el nombre del conjunto de naturalezas.

El otro elemento importante e imprescindible para la definición de la naturaleza es el runtime, como vemos ha de ser definido obligatoriamente.

```
<!ELEMENT runtime (run)>
<!ELEMENT run (parameter*)>
<!ATTLIST run
class     CDATA #REQUIRED>
```

El atributo class hace referencia al nombre de la clase que implementa `org.eclipse.core.resources.IProjectNature`.

4.2.2.1.4 Wizards

Los wizards son usados para guiar al usuario en una serie de tareas para facilitarle la labor. Los plugin pueden extender los wizards que trae la plataforma Eclipse, pudiendo emplear para ello diferentes puntos de extensión dependiendo el tipo de wizard que se pretenda desarrollar:

- `org.eclipse.ui.newWizards`: este punto de extensión se utiliza para la creación de wizards que crean nuevos recursos en el entorno. Este tipo de asistentes pueden ser localizados en el menú *File* > *New*.
- `org.eclipse.ui.importWizards`: este es el punto de extensión empleado cuando se desea importar recursos. Estos asistentes son accesibles desde el menú *File* > *Import*.
- `org.eclipse.ui.exportWizards`: cuando se desee crear un wizard para exportar recursos se ha de usar este punto de extensión. Para acceder a estos asistentes se utiliza el menú *File* > *Export*.

El punto de extensión que se ha utilizado en el desarrollo del proyecto ha sido `org.eclipse.ui.newWizards`. Antes de comentar los campos de su DTD conviene estudiar la estructura de los wizards en Eclipse, en la siguiente figura se muestra el esquema general que siguen.

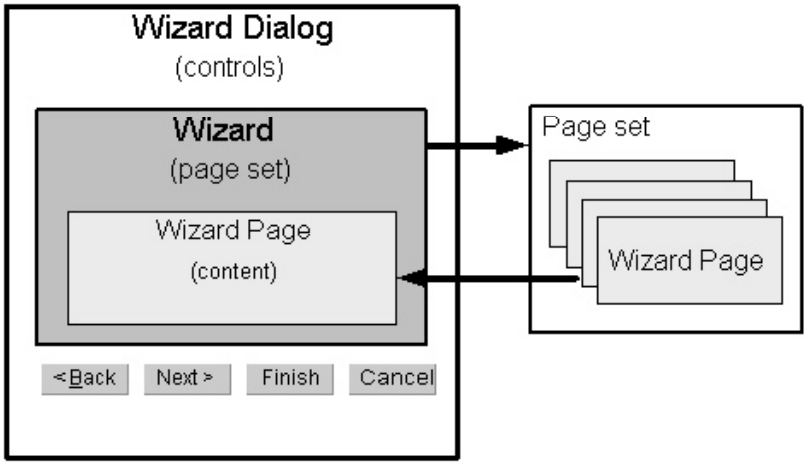


Figura. 4-2 Composición de los Wizards en Eclipse

El **Wizard Dialog** (`org.eclipse.jface.wizard.WizardDialog`), se encarga de definir los botones principales y gestionar el conjunto de paginas que le han sido asociadas. Habilita los botones de Siguiente, Anterior y Finalizar de los asistentes en función de la información recogida de las páginas del wizard. Este componente no ha de ser implementado por parte del plugin que añade nuevos wizards.

El siguiente componente que encontramos en la figura 4-2 es **Wizard**, este se encarga del comportamiento y apariencia general del asistente, como texto de la barra de título, imagen o la disponibilidad de un botón de ayuda. Para su implementación, se recomienda extender la clase `org.eclipse.jface.wizard.Wizard`, que implementa con un funcionamiento general la interfaz `org.eclipse.jface.wizard.IWizard`. La funcionalidad específica que se debe añadir a la implementación, se refiere básicamente a la creación y adición de las paginas del wizard y la implementación del comportamiento cuando el usuario pulsa el botón Finalizar.

Las páginas del asistente (**Wizard Page**) suponen el contenido del propio asistente. Para su implementación se recomienda la extensión de `org.eclipse.jface.wizard.WizardPage`, esta clase implementa la interfaz `org.eclipse.jface.wizard.IWizardPage` con un funcionamiento general. El comportamiento específico que debemos añadirle es lo referido a:

- Crear los controles SWT que compondrán la página para el usuario.
- Determinar cuando una página ha sido completada para permitir al usuario moverse a la siguiente página.

A continuación se muestra la especificación del punto de extensión comentado anteriormente para la añadir nuevos asistentes encargados de añadir nuevos recursos al espacio de trabajo de la plataforma.

```
<!ELEMENT extension (category | wizard | primaryWizard)*>
<!ATTLIST extension
point     CDATA #REQUIRED
id        CDATA #IMPLIED
name      CDATA #IMPLIED>

<!ELEMENT wizard (description? , selection*)>
<!ATTLIST wizard
id                      CDATA #REQUIRED
name                    CDATA #REQUIRED
icon                    CDATA #IMPLIED
category                CDATA #IMPLIED
class                   CDATA #REQUIRED
```

```
project                 (true | false)
finalPerspective        CDATA #IMPLIED
preferredPerspectives   CDATA #IMPLIED
helpHref                CDATA #IMPLIED
descriptionImage        CDATA #IMPLIED
canFinishEarly          (true | false)
hasPages                (true | false) >
```

Los apartados más importantes de la especificación para la creación de un nuevo asistente son los siguientes:

- *id*: un nombre único que sirve para identificar al wizard en la plataforma.
- *name*: Un nombre legible y representativo por el que se mostrará al asistente en la ventana de dialogo *File > New*.
- *category*: es una composición de identificadores de categoría separados por el símbolo '/'. Cada identificador debe estar definido en el entorno, de no ser así el asistente será añadido a la categoría "Others".
- *class*: nombre de la clase Java que implementa `org.eclipse.ui.INewWizard`.
- *project*: es un atributo opcional que nos permite especificar si el asistente va a crear un nuevo recurso de tipo proyecto. Esto es útil para que nuestro nuevo asistente aparezca en el diálogo de nuevo proyecto.

Para mejorar la organización, podemos organizar, como ya hemos visto en la creación de vistas, los asistentes mediante categorías. Estas categorías son definidas desde el mismo punto de extensión que los propios asistentes. Su especificación DTD se muestra a continuación.

```
<!ELEMENT category EMPTY>
<!ATTLIST category
id                      CDATA #REQUIRED
name                    CDATA #REQUIRED
parentCategory          CDATA #IMPLIED>
```

Los campos son bastante simples, destacando *parentCategory*, en él se ha de especificar la ruta a otra categoría si la categoría que se está definiendo debe ser añadida como un hijo. El resto son el *id*, identificador único de la categoría y el *name*, nombre legible y representativo de la categoría que será por el que se muestre en el entorno.

4.2.2.1.5 Perspectives

Las perspectivas proporcionan una capa de organización adicional dentro de la ventana del workbench. Las perspectivas definen un conjunto de vistas, su organización y los conjuntos de acciones visibles por defecto.

El punto de extensión en el que se definen nuevas perspectivas es `org.eclipse.ui.perspectives`.

La definición del punto extensión es bastante sencilla:

```
<!ELEMENT extension (perspective*)>
<!ATTLIST extension
point   CDATA #REQUIRED
id      CDATA #IMPLIED
name    CDATA #IMPLIED>

<!ELEMENT perspective (description?)>
<!ATTLIST perspective
id      CDATA #REQUIRED
name    CDATA #REQUIRED
class   CDATA #REQUIRED
icon    CDATA #IMPLIED
fixed   (true | false) >
```

El plugin deberá especificar el identificador único del elemento perspectiva junto con un nombre representativo que será utilizado en la interfaz de usuario para referirse a ella. La clase asociada a la perspectiva debe implementar `org.eclipse.ui.IPerspectiveFactory`, sus detalles de implementación se mostrarán en siguientes apartados.

4.2.2.1.6 Perspective Extensions

Esta extensión se emplea para ampliar las perspectivas ya definidas en el entorno por otros plugin. Los plugins pueden contribuir a perspectivas ya definidas con nuevas vistas, acciones, accesos directos a asistentes, etc….

Para servirnos de esta funcionalidad tenemos que hacer uso del punto de extensión `org.eclipse.ui.perspectiveExtensions`. La especificación que se ha de cumplir para utilizar esta extensión viene definida de la siguiente manera:

```
<!ELEMENT extension (perspectiveExtension*)>
<!ATTLIST extension
point     CDATA #REQUIRED
id        CDATA #IMPLIED
name      CDATA #IMPLIED>

<!ELEMENT perspectiveExtension (actionSet | viewShortcut | perspectiveShortcut | newWizardShortcut |
view | showInPart)*>
<!ATTLIST perspectiveExtension
targetID CDATA #REQUIRED>
```

El campo *targetID* es el identificador de la perspectiva existente en la plataforma sobre la que añadiremos nuevos componentes. Los elemento posibles que se ven en la especificación requieren simplemente el identificador adecuado en cada caso, el único campo con más atributos es la adición de nuevas vistas, en las que se debe especificar aspectos como disposición de la nueva vista, orientación en relación a otras vistas, visibilidad, etc…

4.2.2.1.7 Decorators

Este punto de extensión (`org.eclipse.ui.decorators`) nos permite gestionar los decoradores y de las imágenes asociadas, con ello podremos decorar los recursos y elementos de las vistas en función de las condiciones que establezcamos.

Existe dos maneras de establecer los decoradores. La manera más simple es estableciendo un decorador ligero declarativo (declarative lightweight decorador), de esta manera toda la información se establece en el *plugin.xml*. La primera opción es útil cuando se requiere establecer nada más una imagen, cuando se requiere más flexibilidad o si los iconos son determinados dinámicamente se puede usar la clase `org.eclipse.jface.viewers.ILightweightLabelDecorator`.

A continuación se muestra la configuración requerida para especificar los decoradores.

```
<!ELEMENT extension (decorator*)>

<!ATTLIST extension
point     CDATA #REQUIRED
id        CDATA #IMPLIED
name      CDATA #IMPLIED>

<!ELEMENT decorator (description? , enablement?)>
<!ATTLIST decorator
id                CDATA #REQUIRED
label             CDATA #REQUIRED
class             CDATA #IMPLIED
objectClass       CDATA #IMPLIED
```

```
adaptable       (true | false)
state           (true | false)
lightweight     (true|false)
icon            CDATA #IMPLIED
location        (TOP_LEFT|TOP_RIGHT|BOTTOM_LEFT|BOTTOM_RIGHT|UNDERLAY) >
```

Los datos más importantes y empleados del elemento decorador son el

- Identificador único y su etiqueta representativa.
- Clase, dependiendo si se establece como ligero o no esta clase será `org.eclipse.jface.viewers.ILightweightLabelDecorator` ó `org.eclipse.jface.viewers.ILabelDecorator` respectivamente. Si no se establece ninguna clase se toma como un decorador declarativo.
- Lightweight, determina si el decorador es declarativo o implementa `org.eclipse.jface.viewers.ILightweightLabelDecorator` .
- Icon y location, si el decorador es declarativo aquí especificaremos la disposición del mismo en el elemento y el icono a mostrar.

Un subelemento de los decoradores que resulta muy útil en su uso es *enablement*, este elemento nos permite determinar para que elementos del entorno este decorador se activará.

```
<!ELEMENT enablement (and | or | not | objectClass | objectState | pluginState | systemProperty)>
```

Dentro de las diferentes posibilidades uno de los elementos más útiles es *objectClass* que nos permite especificar la clase de los elementos sobre los que queremos aplicar el decorador (por ejemplo sólo sobre `IProject`). Las combinaciones lógicas de todos ellos son posibles ofreciendo muchas posibilidades.

4.2.2.1.8 Markers

Los markers o marcadores son un tipo de metainformación que el entorno puede emplear para etiquetar los recursos, estos marcadores pueden ser almacenados de manera persistente.

Los marcadores son de diferente tipo y en función del tipo pueden tener diferentes atributos. Este punto de extensión permite la creación de nuevos marcadores con sus propios atributos o crear nuevos marcadores a partir de otros ya existentes en la plataforma, heredando así sus atributos.

Las especificaciones para los diferentes elementos que componen el punto de extensión se comentan a continuación:

```
<!ELEMENT extension (super* , persistent? , attribute*)>
<!ATTLIST extension
point CDATA #REQUIRED
id    CDATA #IMPLIED
name  CDATA #IMPLIED>

<!ELEMENT super EMPTY>
<!ATTLIST super
type CDATA #REQUIRED>

<!ELEMENT persistent EMPTY>
<!ATTLIST persistent
value (true | false) >

<!ELEMENT attribute EMPTY>
<!ATTLIST attribute
name CDATA #REQUIRED>
```

Como vemos en este punto de extensión sólo se puede definir un marcador de cada vez. En el definiremos su identificador único, su nombre y los posibles subelementos. En cuanto a *super*, identifica el marcador del que heredaremos los atributos (puede existir herencia múltiple), *persistent*, determina si el marcador será almacenado de manera persistente y finalmente *attribute*, empleado para definir la serie de atributos que posee el marcador.

4.2.3 Introducción a Java Remote Method Invocation

Java Remote Method Invocation (RMI) es una tecnología desarrollada por Sun para facilitar el desarrollo de aplicaciones Java distribuidas. Esta tecnología permite que los métodos de objetos Java remotos sean invocados desde otra máquina virtual y que pueda estar en otro host diferente. Para todo ello RMI fundamenta su funcionamiento en la serialización de los objetos para pasar los parámetros.

Algunas de las ventajas que se destacan a la hora de desarrollar aplicaciones basadas en esta tecnología son:

- Soporta la invocación remota en objetos sobre diferentes máquinas virtuales.
- Mantiene la mayor parte de la semántica del lenguaje Java para el desarrollo de aplicaciones distribuidas lo que facilita la programación.
- Permite diferentes tipos de referencias para objetos remotos, persistentes, no persistentes o 'lazy activation'.
- Mantiene la seguridad del entorno java en cuanto a gestores de seguridad y cargadores de clases.

4.2.3.1 Arquitectura

En la figura 4 se muestra un diseño habitual en el desarrollo de aplicaciones distribuidas mediante el uso de Java RMI.

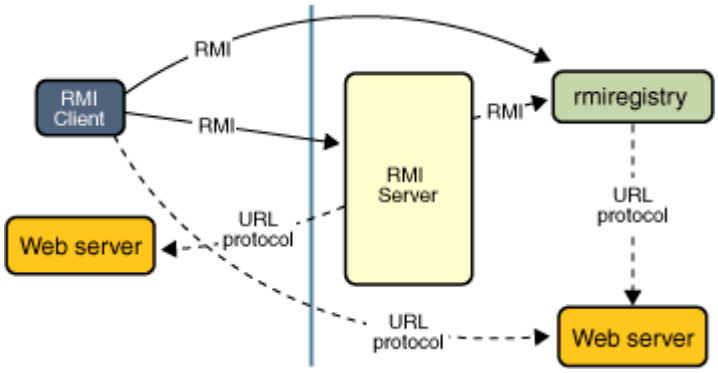


Figura. 4-3. Arquitectura de una aplicación RMI

Este esquema permite a las aplicaciones realizar las funciones básicas:

- Localizar objetos remotos
- Comunicarse con los objetos remotos.
- Cargar los bytecodes de las clases que son pasadas como parámetros o valores de retorno.

El servidor RMI registra en el registro RMI los objetos que exporta de forma que los clientes puedan acceder a ellos a través de un formato de URL. Una vez que el cliente

tiene la referencia al objeto remoto, ya puede trabajar con él como se trabaja con cualquier objeto en java.

El sistema RMI permite además utilizar servidores web para descargar las clases entre los diferentes extremos de la comunicación RMI, esto permite versatilidad para comunicar a los clientes cambios en la implementación.

4.2.3.2 Desarrollo

A continuación se muestran las etapas y detalles en la implementación de una aplicación básica con una arquitectura RMI.

Crear el *rmiregistry*

Existen varias formas de iniciar el registro RMI, la más práctica a la hora de programar es utilizando el método `createRegistry` de la clase `java.rmi.registry.LocateRegistry`. A continuación se muestra una porción de código a modo de ejemplo:

```
try{
      Registry registro=LocateRegistry.createRegistry(Registry.REGISTRY_PORT);
      }catch(RemoteException ex)
      {

      }
```

El método invocado inicializa el registro RMI en el puerto por defecto (1099).

Creando la interfaz y objeto remoto

Los objetos que se deseen utilizar como objetos remotos en una aplicación RMI deben implementar una interfaz que cumpla los siguientes principios:

- La interfaz debe extender `java.rmi.Remote`.
- Todos los métodos de la interfaz que vayan a ser invocados remotamente deben declarar que lanzan la excepción `java.rmi.RemoteException`.

En el constructor de la clase que implementa la interfaz remota se debe invocar al método `exportObject` de `java.rmi.server.UnicastRemoteObject`.

A continuación se pone un ejemplo de interfaz remota y su posible implementación.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public interface InterfazRemota extends Remote {
        public String getMensaje() throws RemoteException;
}

public class ClaseRemotaImpl implements InterfazRemota {

private String mensaje;

public ClaseRemotaImpl(String mensaje) {
        UnicastRemoteObject.exportObject(this);
        this.mensaje=mensaje;
}

public String getMensaje() throws RemoteException{
        return mensaje;
}

}
```

Registrar el objeto remoto

Para registrar el objeto remoto con un nombre determinado en el registro RMI,y así hacer el objeto accesible por un nuevo cliente, deberemos utilizar la referencia al registro creado. Para registrar el objeto deberemos utilizar los métodos `bind` o `rebind`, dependiendo si deseamos sobreescribir un objeto ya registrado con el mismo nombre o no.

La siguiente porción de código ejemplifica el registro de un objeto en el registro:

```
registro.rebind("ObjetoRemoto", new ClaseRemotaImpl());
```

Acceder a objetos remotos desde el cliente

Para acceder desde el cliente a los objetos que el servidor ha dispuesto en el registro RMI deberemos hacer uso del método `getRegistry` de `java.rmi.registry.LocateRegistry`. Una vez obtenida la referencia al registra podremos obtener la referencia remota y ejecutar los métodos remotos. A continuación se muestra una porción de código a modo de ejemplo:

```
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Cliente {

public Cliente(){
        try{
        Registry registro=LocateRegistry.getRegistry(host);
        InterfazRemota objRemoto = (InterfazRemota) registro.lookup("ObjetoRemoto");
        System.out.println(objRemoto.getMensaje());
        }catch(RemoteException ex) {}
        catch(NotBoundException ex) {}
}
```

Compilando el código

Para construir las aplicaciones RMI deberemos compilar las clases de manera habitual con *javac* y posteriormente usar el comando *rmic* con la clase que implementa la interfaz remota para generar el Stub de la clase, este elemento servirá para que pueda establecerse la comunicación RMI.

4.2.4 Implementación

4.2.4.1 Creación de Proyectos

En este apartado se tratará lo referido a la creación de un proyecto. Puesto que la implementación no difiere mucho entre los proyectos fuente y los proyectos cliente, trataremos la implementación de ambos en conjunto.

Asistentes

En la creación de un proyecto se ha implementando un asistente siguiendo la especificación del punto de extensión comentada en el punto 4.2.2.1.4.

Por un lado se ha creado la clase `NuevoProyectoWizard`, esta clase implementa `INewWizard`. Esta clase es la encargada de añadir las páginas al asistente, determinar cuando se puede finalizar el asistente y realizar las tareas de creación del proyecto y almacenar la información asociada. El criterio seguido para determinar que se puede finalizar el asistente es consultar que todos los métodos `isPageComplete()` de las páginas del asistente devuelve verdadero.

El asistente se compone de dos paginas, una primera página en la que se pide el nombre del proyecto y una segunda página para detalles específicos de nuestro tipo de proyecto.

En la primera página del asistente que pide el nombre del proyecto, para ello se ha utilizado la clase `org.eclipse.ui.dialogs.WizardNewProjectCreationPage`, esta clase esta especialmente diseñada para solicitar nombres de proyectos y su localización. Ha sido de gran utilidad puesto que ella misma se encarga de evitar que el nombre de proyecto ya exista y ofrece una serie de métodos para obtener el manejador del proyecto.

La segunda página del asistente `DetallesProyectoPage` extiende `WizardPage` para heredar la funcionalidad básica de una página de asistente, en ella hemos creado toda una serie de componentes SWT para recoger los detalles del proyecto: nombre, email, login, contraseña, host, nombre corto, etc… (dependiendo de si estamos creando un proyecto cliente o fuente). Un punto importante en este apartado ha sido refrescar los botones del asistente para poder continuar, el criterio estimado para poder hacer eso ha sido que estén todos los campos completos, para ello hemos añadido a todos los componentes `org.eclipse.swt.widgets.Text` un listener para sus modificaciones e implementar el método `isPageComplete()` para que devuelva verdadero cuando todos los campos estén rellenados. La clase del listener comentado se muestra a continuación, se ha establecido como clase privada dentro de la propia `DetallesProyectoPage`.

```
private class DetallesProyectoListener implements ModifyListener {
public void modifyText(ModifyEvent e) {
        getWizard().getContainer().updateButtons();
}
}
```

Crear la estructura del proyecto

Una vez que se finaliza el asistente se ejecuta el método `perfomFinish()` de la clase `NuevoProyectoWizard`. Este método se encargará de crear el nuevo proyecto y establecer su estructura y características.

En primer lugar, una vez creado el proyecto vacío en el espacio de trabajo, establece la naturaleza del proyecto (para más información ir al apartado 4.2.2.1.3 donde se comentan las naturalezas y su definición). Nuestros proyectos tendrán dos naturalezas, la propia de los proyectos Java *org.eclipse.jdt.core.javanature* y la propia del proyecto dependiendo si es un proyecto fuente (*ecol.comun.natures.servernature*) o cliente (*ecol.comun.natures.clientnature*). Para no repetir código en los asistentes de cliente y servidor se ha creado una clase auxiliar con varios métodos estáticos `ecol.comun.UtilidadesProyecto`, el código asociado al establecimiento de la naturaleza de un proyecto se muestra a continuación:

```
void setNature(IProject proyecto, String nature)  throws CoreException
{
IProjectDescription description = proyecto.getDescription();
String[] natures = description.getNatureIds();
String[] newNatures = new String[natures.length + 1];
System.arraycopy(natures, 0, newNatures, 0, natures.length);
newNatures[natures.length] = nature;
description.setNatureIds(newNatures);
proyecto.setDescription(description, null);
}
```

En segundo lugar el asistente crea la estructura de carpetas del proyecto. Este punto es bastante simple, se crean las carpetas *sesiones* y *src*.

Finalmente se establecen las entradas del classpath, este apartado se facilita bastante al haber especificado la naturaleza de java al proyecto. En nuestro caso deberemos incluir

al classpath la entrada propia de la JRE y la referida a nuestra carpeta de código *src*. Para ello la JDT nos ofrece la clase `JavaCore` con una serie de métodos que nos generan los objetos asociados a las entradas del classpath. A continuación se muestra el fragmento de código que se encarga de generar las entradas del classpath en un proyecto vacío:

```
Try
{
IJavaProject ijp = JavaCore.create(project);
IClasspathEntry cpath[] = new IClasspathEntry[2];
cpath[0] = JavaCore.newSourceEntry(new Path("/" + project.getName() + "/src"));
cpath[1] = JavaCore.newContainerEntry(new Path( "org.eclipse.jdt.launching.JRE_CONTAINER"), false);
ijp.setRawClasspath(cpath, null);
}catch(JavaModelException ex) {}
```

Con estos pasos ya tenemos creada la estructura de nuestros proyectos, el último paso es almacenar de manera persistente la información suministrada en la pagina de detalles del proyecto.

Almacenar información

Para almacenar la información suministrada por el usuario se ha empleado una herramienta que proporciona el entorno Eclipse para el manejo de recursos.

Los proyectos pueden tener almacenadas en su estructura una serie de preferencias. Estas preferencias se organizan en diferentes nodos dentro de diferentes scopes en el propio proyecto. En dichos nodos se puede almacenar la información fácilmente, incluso especificando el tipo básico que se está almacenando. Las clases principales implicadas son `org.eclipse.core.runtime.preferences.IEclipsePreferences` y `org.eclipse.core.runtime.preferences.IScopeContext`. Para nuestros proyectos se ha establecido dos scopes, *ecol.cliente* y *ecol.servidor*. También se ha establecido una serie de identificadores para los valores que se almacenan como preferencias. Todos ellos se pueden referenciar como constantes de la clase `UtilidadesProyecto`.

```
void setPropiedadString(IProject proyecto, String scope, String key, String value) throws
BackingStoreException
{
IScopeContext projectScope = new ProjectScope(proyecto);
IEclipsePreferences projectNode = projectScope.getNode(scope);
projectNode.put(key, value);
projectScope.getNode(scope).flush();
}
```

De esta manera se hacen persistentes todos los detalles del proyecto que el usuario ha introducido en el asistente. Estas preferencias son almacenadas en el subdirectorio *.settings* del proyecto y en el archivo *nombrescope.prefs* (por ejemplo *ecol.cliente.prefs*).

4.2.4.2 Inicio de sesión de trabajo

A continuación se comenta la implementación referida al establecimiento de una sesión de trabajo por parte del servidor y del cliente.

4.2.4.2.1 Lado del servidor

Para iniciar la sesión de trabajo se ha implementado una acción tal y como se describe en el punto 4.2.2.1.2. La implementación de dicha acción lanza un nuevo dialogo `SeleciccionProyectoDialog` en el que se muestra los proyectos abiertos con naturaleza de servidor de proyecto. Esta clase extiende `org.eclipse.jface.dialogs.Dialog` para el funcionamiento básico de un diálogo.

Será la clase con métodos estáticos `EcolServidor` el que propiamente se encargue de crear el controlador de la sesión y el registro RMI donde se publicará.

Crear controlador de recursos

Al crear el controlador de recursos en el servidor se inicializa una `java.util.LinkedList` que mantendrá una lista con todos las referencias a los recursos que se estén coeditando en un momento determinado.

Para controlar el cierre y apertura de recursos se establecen dos listeners, uno sobre los recursos y otro sobre la perspectiva. Para poder hacer esto el entorno nos proporciona, para el caso de los recursos la clase `ResourcesPlugin`, esta nos permite con `getWorkspace()` obtener una referencia al espacio de trabajo y con el método `addResourceChangeListener` podremos mantener un control de los sucesos de los recursos. En el caso de controlar la perspectiva, podemos utilizar la referencia a `IWorkbenchWindow` que se almacena para referirnos a la ventana sobre la que estamos trabajando para invocar el método `addPerspectiveListener`, este listener nos será útil para determinar cuando abrimos los editores o los cerramos, estos eventos serán los que determinen cuando se inicia la coedición de un elemento y cuando se terminan.

Crear controlador de comunicaciones

La creación del control de comunicaciones es muy básica, el punto más importante es inicializar el archivo de log en la carpeta *sesiones* de la estructura del proyecto en el que estamos trabajando.

En el caso de que el fichero no existiese debimos crear uno vacío, para crear los ficheros utilizando la API de Eclipse se le debe suministrar un stream de entrada al fichero a crear, en este caso como el fichero estaría vacío y no habría stream de entrada se utilizó un `ByteArrayInputStream` con un cadena vacía.

El registro del inicio de sesión se hace mediante `java.util.Date` y su método `toGMTString`.

Lanzar la perspectiva de trabajo del servidor

Si todo ha ido bien y no ha se ha producido ninguna excepción al lanzar los servicios o con el registro RMI se abre la perspectiva de trabajo definida para el servidor.

```
window.getActivePage().setPerspective(window.getWorkbench().getPerspectiveRegistry().findPerspectiv
eWithId("ecol.perspectivas.servidor"));
```

El identificador *ecol.perspectivas.servidor* es el definido en *plugin.xml* para la perspectiva del servidor. Para más información sobre la definición de nuevas perspectivas puede consultar el punto 4.2.2.1.5.

Su implementación se genera mediante la implementación de la interfaz `org.eclipse.ui.IPerspectiveFactory`. Se trata de una factoría que genera el formato de la pagina y las acciones y vistas visibles en la página. Por defecto una perspectiva consta sólo de un editor, es por ello que para añadir nuevas vistas se utiliza la posición del editor como referencia.

```
String editorArea = layout.getEditorArea();
IFolderLayout arribaDerecha =
layout.createFolder("arribaDerecha",IPageLayout.RIGHT,0.75f,editorArea);
arribaDerecha.addView("ecol.vistas.servidor.InformacionParejaView");
```

La perspectiva de trabajo del servidor consta de dos vistas particulares y específicas de la aplicación, estas son la vista de Chat y la vista de información de sesión. Los detalles de cómo funcionan y se inicializan vendrán en el apartado 4.2.4.4.

Recibir conexión de cliente

Cuando un usuario intenta conectarse al proyecto enviándole el objeto `ParejaProgramacion` con su información personal, el sistema comprueba que no esté otro usuario conectado, en tal caso lanza una excepción `ParejaEstablecidaException`, por otro lado comprueba la información de login y password cotejándola con los datos almacenados en las preferencias del proyecto, tal y como se explicó en el apartado de la creación de un proyecto. Si la autenticación falla se lanza una excepción `AutenticacionInvalidaException`.

Una vez realizadas las dos primeras comprobaciones el servidor se comunica con la interfaz de conexión del cliente (`IControladorConexionCliente`) y mediante `getServicioRecursos()` y `getServicioComunicaciones`() obtiene ambos controladores para emparejar los servicios locales. De esta manera ya se sabrá el destino al que se tiene que enviar las modificaciones y mensajes de trabajo.

Si todo ha ido bien se registra y almacena el objeto `ParejaProgramacion` y se pasa, por un lado al display de mensajes, si es que se ha asociado uno y que implementará la interfaz `IDisplayInfoTrabajo` y por otro lado al controlador de comunicaciones para que informe de la conexión y la registre.

4.2.4.2.2 Lado del cliente

Al igual que pasaba en el lado del servidor se ha implementado una acción en el entorno Eclipse que lanza una ventana de dialogo de la misma manera.

Esta acción se comunicará con los métodos estáticos de la clase `ecol.cliente.EcolCliente`, ejecutando `public static boolean conectarProyecto(String name, IWorkbenchWindow window)` para iniciar el inicio de los controladores del proyecto.

El primero que se crea es el controlador de sesión (`ecol.cliente.ControladorSesionCliente`). Este se encarga de cargar el objeto `ParejaProgramacion` referente a la información personal del usuario del proyecto cliente. Esta información será la que se suministre al servidor para, por un lado obtener la información de acceso y por otro obtener la información personal que se usará en el entorno para informar.

Este controlador de sesión también es el encargado de obtener la referencia al registro remoto RMI (tal y como se explica en el apartado de desarrollo de aplicaciones RMI, apartado 4.2.3.2) y hacerse exportabl remotamente a si mismo mediante la invocación al método `exportObject` de `UnicastRemoteObject`, puesto que al conectarse con el servidor, este necesitará invocar los métodos remotos de este controlador de sesión.

Los pasos siguientes corresponden al inicio de los controladores de recurso locales, la conexión con el servidor y el emparejamiento de los controladores de recursos. A continuación se detallan estos puntos.

Lanzar controladores de recursos y comunicaciones

En el caso del controlador de comunicaciones, su inicialización es idéntica a la del lado del servidor puesto que se emplea la misma clase.

Al crear el controlador de recursos en el cliente se inicializa una `java.util.LinkedList` que mantendrá una lista con todos las referencias a los recursos que se estén coeditando en un momento determinado. En este caso esta lista

mantiene las referencias a los documentos coeditándose no sólo a los que el usuario tenga abiertos en ese momento.

Al igual que el controlador de recursos en el servidor el sistema cliente registra un listener sobre la ventana de trabajo (`ecol.cliente.recursos.RecursosPerspectivaListener`), para ello se usa la referencia almacenada a `IWorkbenchWindow`.

Conectar con proyecto fuente

Para comenzar la conexión con el servidor propiamente, el cliente invoca el método privado del controlador de sesión: `private ParejaProgramacion conectarProyecto()`. Este método se encarga de, usando la referencia al registro obtenida anteriormente, ejecutar el método remoto de conexión (los detalles del funcionamiento en el lado del servidor han quedado descritos en el apartado anterior referente a como recibe el servidor uan conexión del cliente).

El resultado de esta operación, si no se han recibido alguna de las excepciones posibles: `RemoteException`, `NotBoundException`, `ConexionParejaException`, `AutenticacioInvalidaException` ó `ParejaEstablecidaException`, es un objeto `ParejaProgramacion`, este objeto será utilizado para informar

Finalmente obtenemos los objetos remotos para emparejar los servicios de recursos y comunicaciones, esto les permitirá saber a quien enviar todos los eventos y mensajes del sistema. Para ello se disponen los métodos `setServicioPareja()` de cada uno de ellos.

Cargar la estructura del proyecto

Una de las particularidades al establecer las parejas de servicio ocurre en el controlador de recursos del cliente. Este debe cargar la estructura inicial del proyecto en la carpeta *src* del proyecto.

Puesto que se trata de una tarea que puede llevar tiempo y no se debería permitir la manipulación mientras se carga la estructura se ha creado un `ProgressMonitorDialog`, este se encarga de mostrar una barra de progreso, en este caso indeterminada, al usuario mientras se realiza la tarea.

Para la carga de la estructura se invoca al controlador de recursos del servidor que nos devolverá un array de objetos `RecursoCompartido`, estos poseen un path, nombre y contenido. También se obtiene del servidor un array de `Anotaciones` para establecerlas posteriormente a los recursos del proyecto.

El servidor recorre todo el directorio src del `IProject`. En este caso al obtener los contenidos de los ficheros hay dos posibilidades, que estén siendo editados por el servidor o que estén cerrados. Para soportar esta situación se empleó `FileBuffers.getTextFileBufferManager()`, este método estático nos retorna un manager sobre el que podemos invocar el método `getTextFileBuffer()`, pasándole como parámetro el path completo del fichero que queremos. Si este método retorna null, entonces quiere decir que el fichero no está siendo actualmente en edición por parte del servidor, en este caso se abre el fichero de manera habitual y se lee completo.

Para la obtención de todas las anotaciones del proyecto, el servidor actúa de igual manera, consulta sobre todos los recursos de la carpeta src sobre el tipo de anotación creado para nuestro sistema.

Al finalizar toda la consulta, el cliente tendrá disponible todos los recursos del proyecto sobre su carpeta *src* y podrá empezar a trabajar.

Cargar el estado inicial de coedición

El único problema del apartado anterior es que cuando se reciben los recursos no se indica cual se está coeditando o cual no, por ello el cliente no sabría cuales son las referencias activas para trabajar en colaboración.

Para ello realiza una consulta al servidor para que le devuelva las referencias sobre los recursos del proyecto que están siendo coeditados. El cliente en ese momento inicializará los objetos `ReferenciaRecursoCompartido` oportunos y los almacenará en el controlador de recursos. El resto de detalles para la coedición se comenta en el apartado siguiente.

Lanzar la perspectiva de trabajo del cliente

Al igual que pasa en el lado del servidor, cuando el cliente ha realizado una conexión satisfactoria y no recibe ningún tipo de excepción de las anteriormente comentadas, la perspectiva del entorno cambia para establecer *ecol.perspectivas.cliente*. El modo de activar la perspectiva y manejarla es igual que en el caso del lado del servidor.

Los elementos más importantes en este caso son las vistas de Chat y de información que serán comentadas más en profundidad en el apartado Comunicación e información.

4.2.4.3 Manipulación y coedición de recursos

Para la manipulación de los recursos entran en juego, tanto en el cliente como en el servidor una serie de listeners que nos permiten estar informados de las modificaciones del entorno y actuar en consecuencia. Por ello trataremos este apartado en conjunto, explicando el funcionamiento en general de cada uno de esos listeners y demás elementos que entran en juego para dar soporte al sistema.

Listener de la perspectiva

Cuando se empieza a trabajar, tanto en el cliente como en el servidor, se registra un listener que implementa `org.eclipse.ui.IPerspectiveListener` en la ventana de trabajo. Este listener se encargará de mantener un control sobre los editores que se abren y sobre los que se cierran.

Cuando un editor es abierto, el listener recibe un evento que es manejado por el método `perspectiveChanged(IWorkbenchPage page, IPerspectiveDescriptor perspective, String changeId)`. Para identificar el tipo utilizamos el campo *changeId*, en nuestro caso nos interesará cuando se tomen los valores definidos en las constantes de la clase `IWorkbenchPage` (`CHANGE_EDITOR_OPEN` y `CHANGE_EDITOR_CLOSE`).

En el caso de que un editor sea abierto, tendremos que determinar si se trata de un editor de un recurso que nos interesa. Para ello obtendremos el editor activo, puesto que al ser lanzado el evento será el que esté abierto. A continuación se muestra una porción de código empleado:

```
IEditorPart edPart = page.getActiveEditor();
IEditorInput input = edPart.getEditorInput();
if (!(input instanceof IFileEditorInput) ){
        return;
}
IFile fichero=((IFileEditorInput)input).getFile();
```

En el caso de que se haya cerrado un editor, no podremos mantener una referencia a cual fue el recurso cerrado. Para tratar este caso se obtendrán todas las referencias a los editores abiertos, y así podremos comparar que editores se han cerrado y cuales siguen activos de acuerdo a las referencias almacenadas en las listas de coeditados del controlador de recursos.

Listener de recursos

Este listener se empleará nada más en el lado del servidor, su principal tarea es controlar cuando un recurso ha sido eliminado del servidor, puesto que las creaciones de recursos se controlan al abrirse automáticamente su editor asociado.

Para crear un listener sobre los recursos necesitamos implementar la interfaz `org.eclipse.core.resources.IResourceChangeListener`. Una vez hecho esto, registraremos el plugin en el espacio de trabajo, para ello disponemos de los métodos estáticos de la clase `ResourcesPlugin` que nos permite obtener una referencia al espacio de trabajo y asociarle el listener.

De esta manera cuando un recurso cambie se lanzará un evento `IResourceChangeEvent`, que será recibido por el método `resourceChanged` del listener. En este momento deberemos consultar los cambios producidos para comprobar en que recurso se produjo y si el tipo de evento era el que nosotros estabamos esperando.

Para ello deberemos invocar el método `geDelta()` del evento para obtener la variación en los recursos del sistema. Para consultar las modificaciones de los recursos lo más adecuado es crear una clase que implemente la interfaz `IResourceDeltaVisitor`. Una vez implementado podremos consultar la variación mediante:

```
event.getDelta().accept(new DeltaVisitor());
```

En el visitor deberemos implementar el método `boolean visit(IResourceDelta delta)`. En el podremos distinguir los diferentes tipos de variaciones. En nuestro caso, el siguiente fragmento de código representa la manera de consultar nada más los recursos borrados.

```
public boolean visit(IResourceDelta delta)
{
	IResource res = delta.getResource();
	switch (delta.getKind()) {
	case IResourceDelta.REMOVED:
		if( (res instanceof IFile) && (EcolServidor.getControladorSesion().getNProyecto().
			compareTo(res.getProject().getName())==0))
		{
			IFile fichero=(IFile)res;
			if(fichero.getName().endsWith("java"))controlador.recursoBorrado(fichero);
		}
	break;
	}
	return true;
}
```

En nuestro caso al encontrar un recurso java borrado que pertenezca al proyecto con el que estamos trabajando lo informaremos al controlador de recursos para que lo notifique al cliente.

Listener de los documentos

El último de los listener que entran en juego en el sistema de coedición de documentos. Este será el que nos permita controlar las modificaciones que se realizan sobre los documentos y así poder enviarlas al otro miembro de la pareja.

Para establecer un listener de este tipo deberemos implementar la intefaz `org.eclipse.jface.text.IDocumentListener`, tener una referencia al `IDocument` del documento sobre el que estamos trabajando y emplear el método `addDocumentListener`.

La referencia al `IDocument` se puede obtener a partir de la referencia al editor abierto, en el siguiente fragmento de código se puede ver como obtener dicha referencia:

```
ITextEditor editor = (ITextEditor) edPart;
IDocumentProvider dp = editor.getDocumentProvider();
IDocument doc = dp.getDocument(editor.getEditorInput());
CodigoFuenteListener cfListener = new CodigoFuenteListener(this);
doc.addDocumentListener(cfListener);
```

De esta manera ya tendremos disponibles un medio para controlar los eventos de modificación. Estos eventos se representan con la clase `DocumentEvent`, en ella encontraremos el texto modificado, el desplazamiento, la longitud y una referencia al propio documento.

4.2.4.4 Comunicación e información

Como ya hemos visto, cuando se inicia una sesión correctamente, tanto cliente como servidor inician una perspectiva de trabajo especializada en la programación por parejas. Estas perspectivas son prácticamente idénticas, comparten los elementos comunes de vista de información y vista de comunicación.

A continuación se detalla el funcionamiento de estas dos vistas y como se comunican con los controladores.

4.2.4.4.1 Vista de información

Esta vista es la encargada de informar al usuario sobre su información personal, la información de la pareja de programación y los recursos en coedición.

Cuando se crea esta vista, se registra en el controlador de sesión del sistema. Dependiendo si es la vista de información de cliente o de servidor obtendrá este controlador a través de la clase `EcolCliente` o `EcolServidor`.

El método invocado para registrarse es `public void setDisplayUsuarios(IDisplayInfoTrabajo display)`, este método avisa al controlador de sesión de que el objeto que se pasa como parámetro será a partir de ese momento el destino de los mensajes referidos a la información de usuarios y de trabajo.

Al cerrar la vista, esta se debe encargar de desregistrarse como panel de información en el mismo controlador, para ello se pasará como parámetro null al método anteriormente comentado.

La interfaz definida que deben cumplir los objetos que actúen de vista de información se muestra a continuación:

```
package ecol.comun;

public interface IDisplayInfoTrabajo {
public void parejaConectada(ParejaProgramacion pareja);
public void parejaDesconectada();
public void addCoEditado(RecursoCompartido recurso);
public void eliminarCoEditado(String id);
}
```

El nombre de los métodos son bastante representativos en cuanto a de que se encarga cada uno de ellos. Los controladores de sesión informan a esta interfaz sobre si se ha conectado la pareja, desconectado o si hay nuevos recursos en coedición.

Uno de los problemas que surgen en este apartado (y en otros comentados anteriormente) es el referido al origen de las llamadas en algunos casos. Por ejemplo, cuando un usuario se conecta al servidor se inicia una pila de llamadas que nace en el cliente y finaliza en el hilo del proceso servidor, esto generalmente no implica ningún problema y es el funcionamiento esperado. Pero en algunos casos esta estructura genera problemas como es el caso de SWT.

El problema de "org.eclipse.swt.SWTException: Invalid thread access"

En SWT el hilo que crea el entorno de usuario. Las implementaciones de SWT utilizan un hilo simple para el UI, en este modelo solo el hilo del UI puede invocar operaciones sobre el interfaz de usuario. Por tanto si se intenta acceder a objetos SWT desde fuera del propio hilo UI se obtendrá la excepción `org.eclipse.swt.SWTException: Invalid thread access`. Este es un problema muy a tener en cuenta en nuestra aplicación puesto que el origen de gran cantidad de llamadas se realizan desde otros procesos e hilos.

Para solucionar este problema se disponen una serie de métodos `syncExec(Runnable runnable)` and `asyncExec(Runnable runnable)`. Estos son los únicos métodos de SWT que pueden ser invocados desde cualquier hilo. Esto permite la ejecución de un objeto `Runnable` por parte del hilo del UI, ya sea de manera asíncrona o síncrona.

```
display.syncExec(
  new Runnable() {
    public void run(){
      label.setText(text);
    }
  });
```

Para más información consultar la FAQ de Eclipse SWT ([FAQSWT]).

4.2.4.4.2 Vista de comunicación

Otra de las vistas especializadas que se crean en la perspectiva de trabajo, tanto de cliente como de servidor es la ventana de comunicación (Chat).

Cuando se crea la vista se registra en el controlador de comunicaciones del sistema mediante el método `setDisplayMensajes(IDisplayMensajes display)`. Para ello se consulta al método `getControladorComunicaciones()` del controlador de sesión correspondiente.

Al cerrar la vista, esta se debe encargar de desregistrarse como panel de comunicación en el mismo controlador, para ello se pasará como parámetro null al método anteriormente comentado.

Para poder registrarse un objeto como vista de comunicación debe implementar la interfaz `ecol.comun.comunicaciones.IDisplayMensajes`:

```
package ecol.comun.comunicaciones;

public interface IDisplayMensajes {
        void nuevoMensaje(MensajeChat mensaje);
        void mensajeSistema(String mensaje);
}
```

El objeto `MensajeChat` contiene el mensaje propiamente y el origen del mismo. Por otro lado también se encarga de manejar mensajes del sistema. Estos mensajes del sistema básicamente se refieren a lo concerniente a conexiones y desconexiones de usuarios del sistema.

Un problema en la implementación de esta vista era lo concerniente al empleo del portapapeles. Se trataba de una funcionalidad importante, puesto que en una comunicación entre programadores pasarse porciones de código es algo muy habitual. Finalmente se encontró la clase `org.eclipse.swt.dnd.Clipboard`. Esta clase se construye pasándole como parámetro la ventana sobre la que se está trabajando. Para obtener el contenido del portapapeles se apoya en una serie de clases adicionales, a continuación se muestra un ejemplo:

```
TextTransfer transfer = TextTransfer.getInstance();
String datos = (String) clip.getContents(transfer);
if (datos != null) {
        //Enviar datos
}
```

5 PRUEBAS

5.1 Pruebas unitarias y de integración

Las pruebas del entorno se han realizado desplegando el paquete JAR del plugin en el entorno Eclipse.

Se han prescindido de herramientas como JUnit puesto que muchas de las operaciones a probar no encajaban con el modelo propuesto por JUnit, tal es el caso de operaciones cuyos resultados se manifiestan en la aplicación en el equipo remoto, o tareas que acaban en nuevas tareas asíncronas.

Todas las pruebas han sido realizadas siguien el manual de usuario y teniendo en consideración las limitaciones conocidas de la aplicación.

A continuación se muestran las pruebas realizadas y los resultados obtenidos en cada una de ellas.

5.1.1 Pruebas generales de sesión

En este apartado se tratarán las pruebas generales realizadas al entorno de trabajo.

5.1.1.1 Creación de un proyecto fuente

- Resultado esperado: Un nuevo proyecto creado en la estructura de trabajo del tipo servidor con todos los parámetros almacenados.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.2 Creación de un proyecto cliente

- Resultado esperado: Un nuevo proyecto creado en la estructura de trabajo del tipo cliente con todos los parámetros almacenados.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.3 Publicación de un proyecto fuente por primera vez

- Resultado esperado: Interfaz de proyecto fuente accesible a través de RMI en el puerto 1099. Perspectiva de trabajo servidor.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.4 Publicación de un proyecto fuente después de una desconexión

- Resultado esperado: El mismo que publicar un proyecto por primera vez.
- Resultado de la prueba: *Satisfactorio*

5.1.1.5 Conexión válida de un proyecto cliente por primera vez.

- Resultado esperado: Estructura del proyecto cliente equivalente a la del fuente en ese instante. Información de la fuente en el panel de trabajo. Perspectiva de trabajo establecida en Cliente Ecol.
- Resultado de la prueba: *Satisfactorio*

5.1.1.6 Reconexión válida de un proyecto cliente después de una desconexión.

- Resultado esperado: El mismo que al conectar el proyecto por primera vez.
- Resultado de la prueba: *Satisfactorio*

5.1.1.7 Conexión de cliente a un proyecto fuente con una pareja establecida

- Resultado esperado: se informa al usuario cliente que existe una pareja establecida.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.8 Conexión de cliente a un proyecto fuente con información inválida

- Resultado esperado: El cliente es informado de la invalidez de los datos.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.9 Cambiar propiedad del proyecto fuente en publicación

- Resultado esperado: el nuevo valor de la propiedad se persiste en el proyecto.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.10 Cambiar propiedad de un proyecto fuente diferente al del trabajo en curso.

- Resultado esperado: el nuevo valor se almacena adecuadamente.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.11 Cambiar propiedad de un proyecto cliente conectado

- Resultado esperado: el nuevo valor de la propiedad se persiste en el proyecto
- Resultado de la prueba: *Satisfactorio*.

5.1.1.12 Cambiar propiedad de un proyecto cliente diferente al del trabajo en curso

- Resultado esperado: el nuevo valor de la propiedad se persiste en el proyecto
- Resultado de la prueba: *Satisfactorio*.

5.1.1.13 Cambiar propiedad de un proyecto diferente

- Resultado esperado: la opción del menú se muestra deshabilitada
- Resultado de la prueba: *Satisfactorio*.

5.1.1.14 Finalizar sesión de un proyecto fuente

- Resultado esperado: Se avisa a la pareja del fin de la sesión. Se liberan los listeners de los recursos y se quita la perspectiva de trabajo si es la activa de la ventana. Se habilita el sistema para iniciar una nueva sesión.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.15 Finalizar sesión de un proyecto cliente

- Resultado esperado:
- En el cliente se liberan los controladores par no informar a la fuente más. El sistema queda habilitado para una nueva conexión. El servidor elimina la información del cliente.
- Resultado de la prueba: *Satisfactorio*.

5.1.1.16 Finalizar sesión en el cliente después del servidor

- Resultado esperado: La sesión es finalizada tal como una sesión normal, liberando recursos y terminando la perspectiva de la ventana activa.
- Resultado de la prueba: *Insatisfactorio*. Se comprobó que en algunos casos el listener de recursos de perspectiva no se finalizaba. La solución ha sido controlar en el propio listener que hay una sesión iniciada y si no, no actuar ante los eventos.

5.1.2 Pruebas específicas de manejo de recursos

En este apartado se tratarán las pruebas específicas del manejo de recursos dentro de una sesión de trabajo colaborativo entre cliente y servidor.

5.1.2.1 Creación de un nuevo recurso en un proyecto vacío

- Resultado esperado: Un nuevo recurso aparece en la estructura de la fuente y en la pareja conectada.
- Resultado de la prueba: *Satisfactorio*.

5.1.2.2 Creación de un nuevo recurso en un proyecto no vacío

- Resultado esperado: Un nuevo recurso aparece en la estructura local de la fuente y en la del cliente.
- Resultado de la prueba: *Satisfactorio*.

5.1.2.3 Eliminación de un recurso en un proyecto

- Resultado esperado: El recurso queda eliminado localmente y el cliente recibe un informe del borrado.
- Resultado de la prueba: *Satisfactorio*.

5.1.2.4 Inicio de coedición por parte del usuario fuente

- Resultado esperado: El recurso abierto se abre automáticamente en el cliente. Se actualizan las listas de coedición.
- Resultado de la prueba: Satisfactorio.

5.1.2.5 Finalización de la coedición de un recurso.

- Resultado esperado: Se cierra el recurso. Se elimina de la lista de coeditados en cliente y servidor. Las modificaciones que a partir de ahora haga el cliente no se manifiestan en el recurso fuente y viceversa.
- Resultado de la prueba: *Satisfactorio*.

5.1.2.6 Apertura por parte del usuario cliente de un recurso que se esta coeditando.

- Resultado esperado: El recurso se abre con los contenidos que en ese momento tenga.
- Resultado de la prueba: *Satisfactorio*

5.1.2.7 Modificación por parte del usuario fuente de un documento en coedición

- Resultado esperado: Las modificaciones se manifiestan igualmente en cliente y servidor.
- Resultado de la prueba: *Satisfactorio* (se tiene en cuenta la limitación de turnos de edición).

5.1.2.8 Modificación por parte del usuario cliente de un documento en coedición

- Resultado esperado: Las modificaciones se manifiestan igualmente en cliente y servidor.
- Resultado de la prueba: *Satisfactorio* (se tiene en cuenta la limitación de turnos de edición).

5.1.2.9 Creación de una anotación sobre un recurso por parte del usuario fuente

- Resultado esperado: La anotación se manifiesta tanto en cliente como servidor en el mismo recurso.
- Resultado de la prueba: *Satisfactorio*

5.1.2.10 Creación de una anotación sobre un recurso por parte del usuario cliente

- Resultado esperado: La anotación se manifiesta en el usuario fuente y en local.
- Resultado de la prueba: *satisfactorio*.

5.1.2.11 Actualización de las anotaciones por parte del usuario cliente

- Resultado esperado: Recibe las anotaciones actualizadas para un recurso concreto.
- Resultado de la prueba: *Satisfactorio*.

5.1.3 Pruebas específicas de comunicación e información

5.1.3.1 Notificación en el usuario fuente de la conexión de una pareja

- Resultado esperado: Se muestra un mensaje en la ventana de Chat y se mete en el log.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.2 Actualización en el usuario fuente de la información de la pareja conectada

- Resultado esperado: Se muestran los datos personales en la pestaña de pareja en la vista de trabajo.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.3 Actualización en el usuario fuente de la información de la pareja desconectada

- Resultado esperado: Se elimina la información personal cuando el usuario pareja se desconecta.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.4 Envío de un mensaje al usuario cliente desde la ventana de Chat

- Resultado esperado: Se manifiesta el mensaje en la ventana de Chat en ambos extremos.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.5 Actualización de la lista de recursos coeditados al iniciarse la coedición.

- Resultado esperado: Se añade una entrada a la lista con el identificador el recurso abierto.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.6 Actualización de la información de la pareja en el usuario cliente

- Resultado esperado: La información personal del usuario fuente se muestra al conectarse a un proyecto.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.7 Envío de un mensaje al usuario fuente desde la ventana de Chat

- Resultado esperado: El mensaje se muestra en ambas vistas de Chat y queda registrado en el log.
- Resultado de la prueba: *Satisfactorio*.

5.1.3.8 Actualización de la lista de recursos coeditados al iniciarse la sesión.

- Resultado esperado: La lista en el cliente se muestra actualizada con los recursos en coedición.
- Resultado de la prueba: *Insatisfactorio*. Sólo se actualizan los recursos a partir del instante de conexión. La solución es reabrir la vista de información una vez iniciada la sesión para actualizarla.

5.2 Pruebas de usabilidad

En este apartado se van a mostrar las pruebas de usabilidad que se hicieron.

5.2.1 Elaboración de las pruebas

Para la realización del módulo se sigue un diseño basado en el usuario. Las pruebas se basan en los modelos propuestos siguientes:

1) SUS- A "quick and dirty" usability scale de Jonh Brooke. Este modelo fue desarrollado en 1986 como parte de un estudio de Introducción a la Ingeniería de la Usabilidad para la integración de programas de gestión. Su objetivo es desarrollar un test de objetivos completo y de medición sencilla para realizar comparaciones entre productos. Es muy simple y de uso extendido. Emplea un número reducido de preguntas para verificar si el usuario se encuentra cómodo utilizando el sistema.
2) Computer System Usability Questionnaire de Lewis, J.R (1995). Se basa en un test con una serie de preguntas sobre la interfaz y pide que se valore dentro de una escala si esa característica resulta positiva o no.
3) How to Conduct a Heuristic Evaluation (Nielsen y Molich 1990). Da una serie de recomendaciones para realizar una prueba de usabilidad, como son el número de usuarios evaluados, la definición de unos heurísticos o puntos que pueden ser motivos de problemas a la hora de lograr que un software se de fácil uso.

El resultado es la elaboración de un test de fácil realización en el que se intentan medir tanto aspecto visuales, de contenido y estéticos como si el usuario ha comprendido el modo de funcionamiento del sistema.

Otra técnica que se utilizó para medir la usabilidad fue "Pensando en Voz Alta". A medida que el usuario navega por la aplicación se puede escuchar lo que opina. Estas opiniones se recogen junto al test y son tan útiles para las conclusiones finales como el resultado del propio test.

5.2.1.1 Número y Características de los Usuarios

El número de usuarios que realizan el test es 3 a 5, siguiendo la directiva propuesta por Jakob Nielsen [NIEL2001c].. Las características de los usuarios, son las siguientes:

Usuario 1:

Usuario con experiencia en el ámbito de la programación. Con experiencia en la plataforma Eclipse.

Usuario 2:

Usuario con experiencia en el ámbito de la programación. Poca experiencia en la plataforma Eclipse.

Usuario 3:

Usuario con experiencia en el ámbito de la programación. No conoce la plataforma de desarrollo Eclipse.

5.2.2 Contenido de las pruebas

Se proponen los siguientes escenarios para el test que deberán ser realizados antes de rellenar el cuestionario:

- **Primer test:**
    - **Rol usuario fuente**: crea un proyecto fuente, inicia sesión de trabajo y espera a que el sistema le informe de que su pareja se ha conectado.
    - **Rol usuario cliente:** crea un proyecto cliente, inicia sesión de trabajo y saluda al usuario fuente.
- **Segundo test:**
    - **Rol usuario fuente**: con una sesión ya iniciada, espera a que su pareja se conecte, crear una clase con un método main que imprima por pantalla un saludo. Deberá emplear el entorno para decirle a la pareja que escriba dicho saludo.
    - **Rol usuario cliente:** con un proyecto ya creado, iniciar sesión de trabajo, ver lo que el usuario fuente programa y atender a lo que dice.
- **Tercer test:**
    - **Rol usuario fuente:** con una sesión iniciada. Esperar a que la pareja se conecte. Pregunte a la pareja que recurso quiere abrir y ábralo.
    - **Rol usuario cliente**: inicie sesión con el proyecto creado. Espere instrucciones del usuario fuente. Cuando abran un recurso tome la iniciativa para crear un método main. Al terminar deberán probar ambos que la clase funciona correctamente.
- **Cuarto test:**
    - **Rol usuario fuente:** con una sesión iniciada y un recurso abierto. Espere a que el usuario cliente haga una anotación. Márquela como realizada y coméntele este cambio al cliente.
    - **Rol usuario cliente:** con una sesión iniciada, haga una anotación sobre el recurso que está en coedición. Cuando el usuario fuente le comente que la ha realizado, refresque las anotaciones.

5.2.3 Resultados de los cuestionarios

Las pruebas realizadas a los usuarios reflejan varios aspectos importantes a tener en cuenta.

Por un lado, en cuanto a la usabilidad de la aplicación. Los usuarios echan en falta la integración de un sistema de ayuda dentro del propio prototipo, este aspecto se tiene en cuenta para futuras ampliaciones. Por otro lado la claridad de los asistentes en algunos casos es baja, dudando, por ejemplo, en el campo de información personal si esa información era la propia o la del usuario remoto. Se ha valorado positivamente el diseño y la estructura de vistas propuestas para trabajar.

En cuanto a la valoración de la aplicación y la programación por parejas con una herramienta de este estilo, se ha visto como una técnica útil y se ha visto positivamente la creación de una herramienta que potencie esta forma de trabajo. Sólo un usuario ha considerado que una herramienta así sólo sería útil en entornos educativos. Por otro lado se echa de menos el soporte para múltiples usuarios, con el fin de un trabajo en equipo más unido y potente.

A continuación se muestran los test realizados a los usuarios después de las pruebas anteriormente comentadas.

5.2.3.1 Usuario 1

TEST de USABILIDAD y VALORACION del PROTOTIPO

1. Calidad del entorno visual **a) aspectos gráficos.**	**Excesivo/a**	**Suficiente**	**Insuficiente**	**Nulo/a**
La iconicidad (uso de imágenes visuales) es:		**X**		
El número de textos incluidos es:		**X**		

b) Diseño de pantallas.	**SI**	**NO**	**A VECES**
¿Son claros los asistentes?			**X**
¿Tienen las vistas de trabajo un diseño claro?	**X**		
¿Es atractivo el diseño de las vistas del entorno?	**X**		
¿Son claros los cuadros de dialogo?			**X**

2. Valoración del prototipo:	**Bien estructurada**		**Mal estructurada**	
La información está ...	**X**			
Los contenidos están ...	**X**			
Facilidad de uso:	**Siempre**	**Casi Siempre**	**A veces**	**Nunca**
¿En todo momento se sabe dónde nos encontramos ?		**X**		
¿Existe un sistema de ayuda para resolver dudas?				**x**

Opinión	**Si**	**No**	**Observaciones**
¿Considera útil la programación por parejas distribuida?:	**X**		
¿Le parece un buen enfoque emplear un entorno de desarrollo existente?:	**X**		
¿Cree que la línea de diseño del prototipo es la adecuada?:	**X**		
Como programador, ¿utilizaría una versión final de una herramienta así?	**X**		
¿Valoraría una la ampliación del prototipo para que participen más de dos usuarios?	**x**		**Sería lo ideal**

5.2.3.2 Usuario 2

TEST de USABILIDAD y VALORACION del PROTOTIPO

1. Calidad del entorno visual **a) aspectos gráficos.**	**Excesivo/a**	**Suficiente**	**Insuficiente**	**Nulo/a**
La iconicidad (uso de imágenes visuales) es:		**X**		
El número de textos incluidos es:		**X**		

b) Diseño de pantallas.	**SI**	**NO**	**A VECES**
¿Son claros los asistentes?	**X**		
¿Tienen las vistas de trabajo un diseño claro?	**X**		
¿Es atractivo el diseño de las vistas del entorno?	**X**		
¿Son claros los cuadros de dialogo?			**X**

2. Valoración del prototipo:	Bien estructurada		Mal estructurada	
La información está ...	X			
Los contenidos están ...	X			
Facilidad de uso:	**Siempre**	**Casi Siempre**	**A veces**	**Nunca**
¿En todo momento se sabe dónde nos encontramos ?	X			
¿Existe un sistema de ayuda para resolver dudas?			X	

Opinión	**Si**	**No**	**Observaciones**
¿Considera útil la programación por parejas distribuida?:	X		
¿Le parece un buen enfoque emplear un entorno de desarrollo existente?:	X		
¿Cree que la línea de diseño del prototipo es la adecuada?:	X		
Como programador, ¿utilizaría una versión final de una herramienta así?	X		
¿Valoraría una la ampliación del prototipo para que participen más de dos usuarios?	x		

5.2.3.3 Usuario 3

TEST de USABILIDAD y VALORACION del PROTOTIPO

1. Calidad del entorno visual **a) aspectos gráficos.**	Excesivo/a	Suficiente	Insuficiente	Nulo/a
La iconicidad (uso de imágenes visuales) es:		X		
El número de textos incluidos es:		X		

b) Diseño de pantallas.	**SI**	**NO**	**A VECES**
¿Son claros los asistentes?	X		
¿Tienen las vistas de trabajo un diseño claro?	X		
¿Es atractivo el diseño de las vistas del entorno?	X		
¿Son claros los cuadros de dialogo?			X

2. Valoración del prototipo:	Bien estructurada		Mal estructurada	
La información está ...	X			
Los contenidos están ...	X			
Facilidad de uso:	**Siempre**	**Casi Siempre**	**A veces**	**Nunca**
¿En todo momento se sabe dónde nos encontramos ?	X			
¿Existe un sistema de ayuda para resolver dudas?			X	

Opinión	**Si**	**No**	**Observaciones**
¿Considera útil la programación por parejas distribuida?:	X		
¿Le parece un buen enfoque emplear un entorno de desarrollo existente?:	X		
¿Cree que la línea de diseño del prototipo es la adecuada?:	X		
Como programador, ¿utilizaría una versión final de una herramienta así?	X		**Sólo en entornos educativos**
¿Valoraría una la ampliación del prototipo para que	x		

participen más de dos usuarios?			

6 MANUALES

6.1 Manual de usuario

En el siguiente manual se recogerá los pasos a seguir por el usuario de la aplicación. Para ello se abarcarán todas las funciones de la aplicación, explicando los pasos oportunos y acompañando los mismos con capturas de pantalla. El manual se organizará en diferentes secciones correspondientes a los diferentes conjuntos de funcionalidades que ofrece el sistema al usuario, así mismo se hará una diferenciación entre el manual para el usuario que actúa de cliente y el usuario que actúa de servidor.

El manual del usuario está orientado a un usuario experto, alguien habituado a la programación y a los entornos de desarrollo habituales como es Eclipse. Por otro lado también debe tener conocimiento básico sobre lo que supone la programación por parejas. Esto ayudará sin duda a conocer los objetivos de la aplicación y facilitar y agilizar su manejo.

6.1.1 Inicio de la aplicación

Cuando se inicia la aplicación Eclipse con el plugin instalado, tal y como se explica en la sección del manual de instalación, se nos presentará una interfaz parecida a la que se muestra en la siguiente captura:

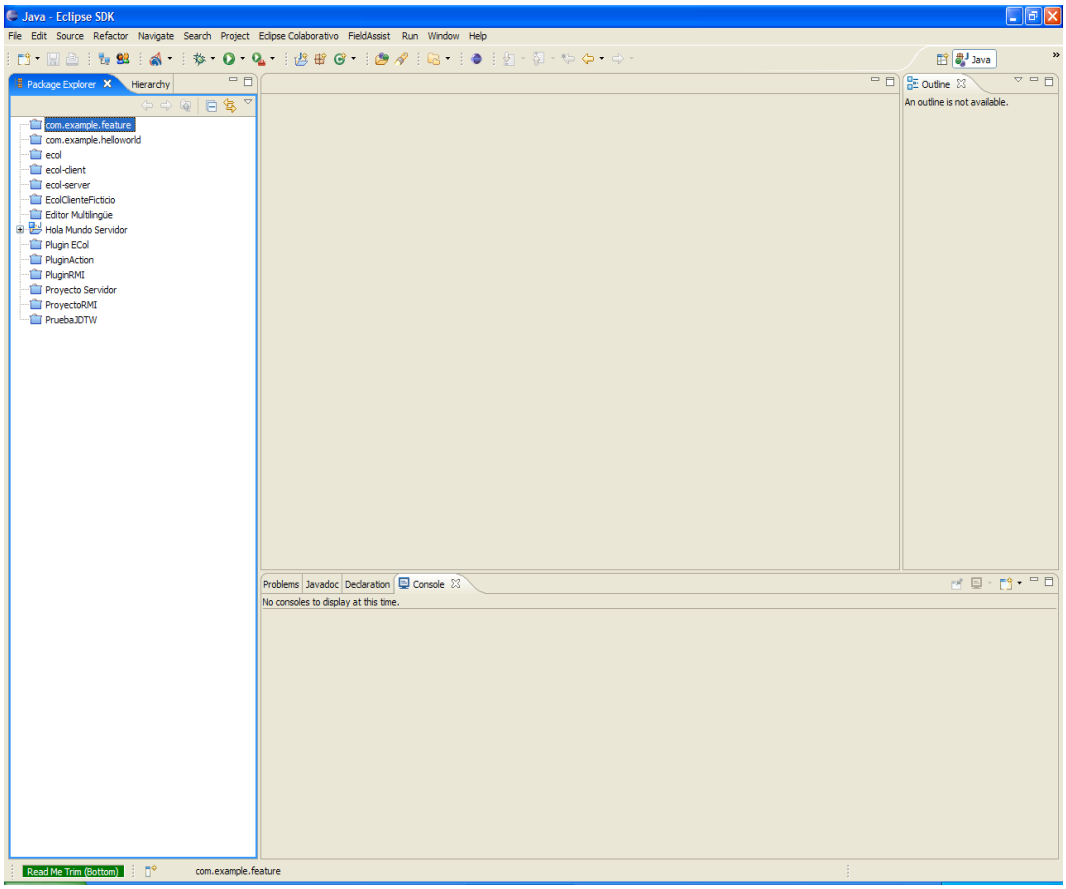


Figura. 6-1. Inicio de la aplicación

Si tenemos activada la perspectiva Java veremos el menú del plugin "Eclipse Colaborativo" y en la barra de herramientas veremos dos accesos a las funcionalidades de lanzar un proyecto fuente y conectar un proyecto cliente.

En el caso de que no apareciesen estas acciones ni el menú es posible que se deba a que no tenga la perspectiva Java activada o a que ya personalizó previamente la perspectiva de trabajo Java y, por ello, sus preferencias prevalecen sobre la configuración interna del sistema. En este caso tiene dos opciones para solucionarlo: puede resetear la perspectiva a sus valores por defecto mediante el menú *Window > Reset Perspective* (si está en la perspectiva Java), o puede añadir el conjunto de acciones a la perspectiva mediante la opción *Window > Customize Perspective…* (esta es la opción más recomendable), esta opción abrirá un diálogo al usuario en el que se puede elegir el conjunto de acciones y acceso directo a funciones de las que se compondrá la perspectiva actual.

Una vez que ya tenemos el conjunto de acciones activado podemos empezar a trabajar con el entorno.

6.1.2 Manual del usuario servidor

6.1.2.1 Crear un nuevo proyecto fuente

La primera tarea que se debe realizar al empezar a trabajar con el sistema es la creación de un nuevo proyecto como fuente de proyecto.

Para esta tarea se ha puesto a disposición del usuario un asistente que facilita la tarea. El asistente es accesible desde la opción de menú File > New > Project, esta opción abrirá un dialogo que le permite al usuario elegir el tipo de proyecto a crear, tal y como se muestra en la siguiente figura:

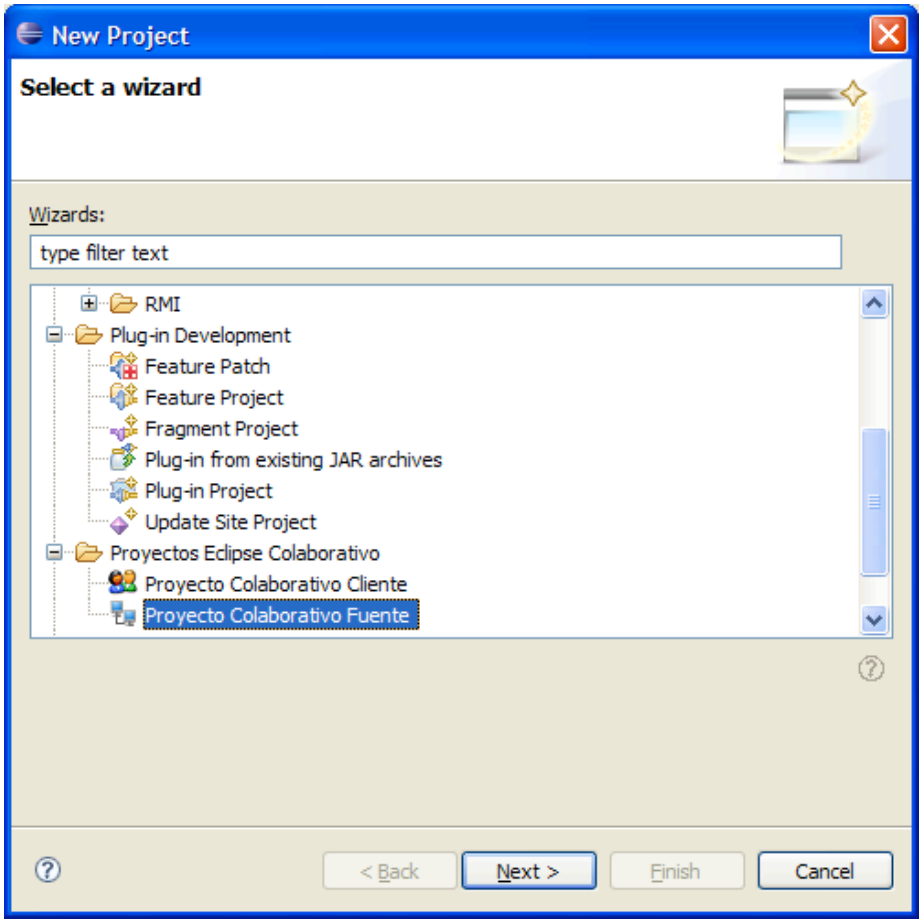


Figura. 6-2 Inicio Asistente Creación de Proyecto

En nuestro caso deberemos movernos hasta la carpeta “Proyectos Eclipse Colaborativo”, dentro de esta opción elegiremos “Proyecto Colaborativo Fuente” y pulsaremos sobre “Next”.

En este punto entramos propiamente en la creación del proyecto, el primer paso es establecer el nombre para el proyecto, para ello se ha habilitado el dialogo que se muestra en la figura. Se trata de un campo obligatorio y por tanto, no se podrá continuar hasta que se haya introducido un nombre válido y que no exista previamente. Una vez concluido pulsaremos sobre el botón “Next” que se habilitará.

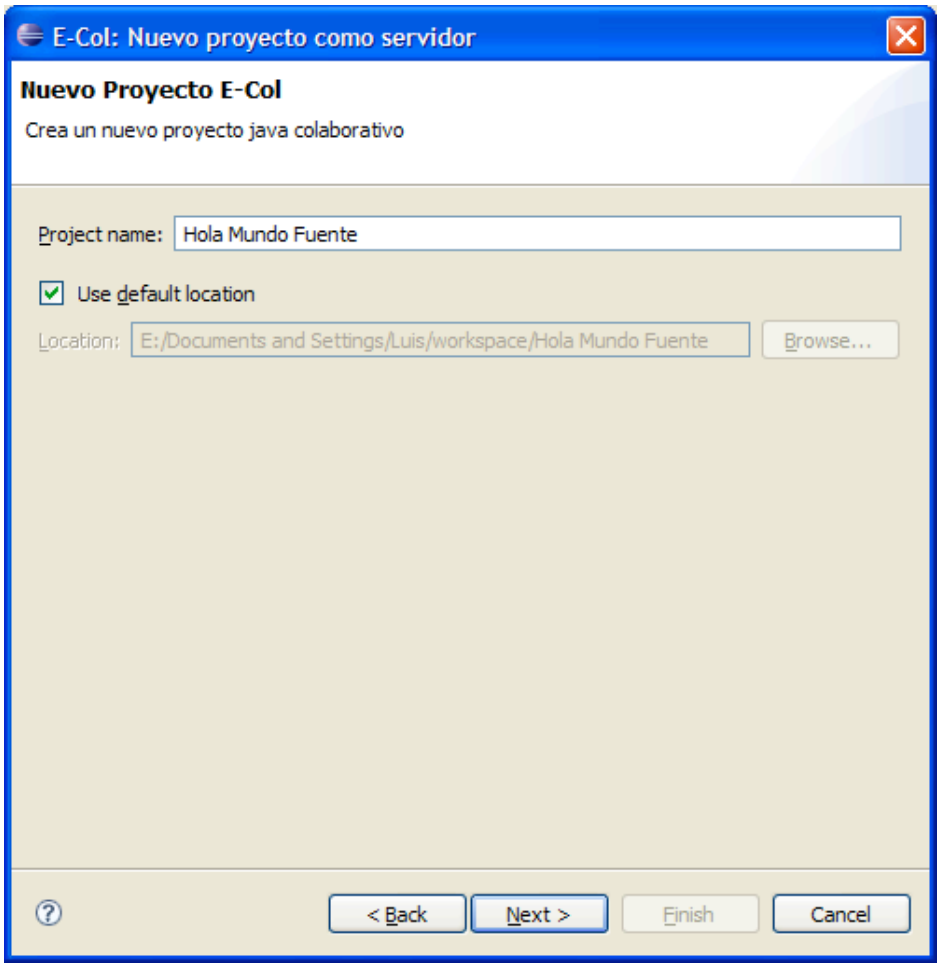


Figura. 6-3 Definición de nombre de proyecto

Finalmente, el último paso para la creación del proyecto es establecer los detalles del mismo. Se deben cumplimentar todos los apartados del cuadro de dialogo del asistente tal y como se muestra en la figura. Aun así, dichos campos se podrán modificar posteriormente desde el menú del plugin.

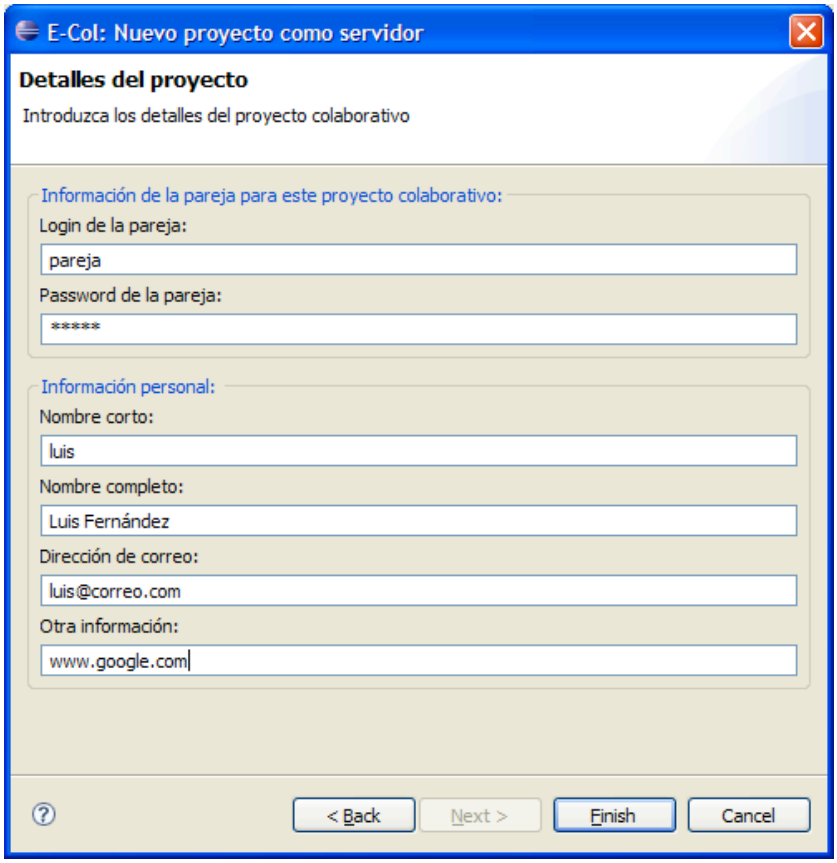


Figura. 6-4. Pagina detalles de proyecto

Una vez hecho esto y pulsando en “Finish" veremos como se crea en el espacio de trabajo un nuevo proyecto con el nombre que hemos seleccionado. Para facilitar la diferenciación con el resto de proyectos se marca con un pequeño cuadrado azul en la esquina superior izquierda del icono.

La estructura de un proyecto fuente consta de los siguientes elementos:

- Carpeta *src*: esta carpeta no debe ser eliminada ni cambiado su nombre. En ella será donde cree los recursos del proyecto. Sólo los recursos que se dispongan en esta carpeta se tendrán en cuenta para el trabajo de colaboración del sistema.
- Carpeta *sesiones*: en esta carpeta se mantendrá un log de las sesiones. Este log se limitará prácticamente a las sesiones de Chat y a las conexiones y desconexiones de la pareja de programación.
- *JRE System Library*: también se asocia en el proyecto a la JRE por defecto del entorno, será la que se emplee para compilar el proyecto que se esté desarrollando.

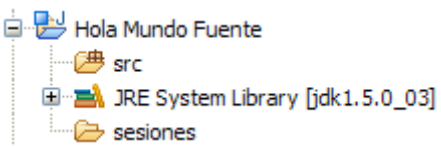


Figura. 6-5. Detalle de la estructura creada y decoradores

6.1.2.2 Modificar los detalles de un proyecto fuente

Para modificar las propiedades de un proyecto fuente que ya hemos creado previamente se deberán llevar a cabo los pasos que se comentan a continuación.

En primer lugar deberá seleccionar la carpeta raíz del proyecto que desea modificar en el entorno Eclipse, es necesario que el proyecto esté abierto para permitirle realizar la modificación.

Una vez seleccionado el elemento vaya al menú “Eclipse Colaborativo” y seleccione dentro del menú Propiedades Proyecto Servidor la opción que prefiera. Existe una opción por cada campo modificable del proyecto.

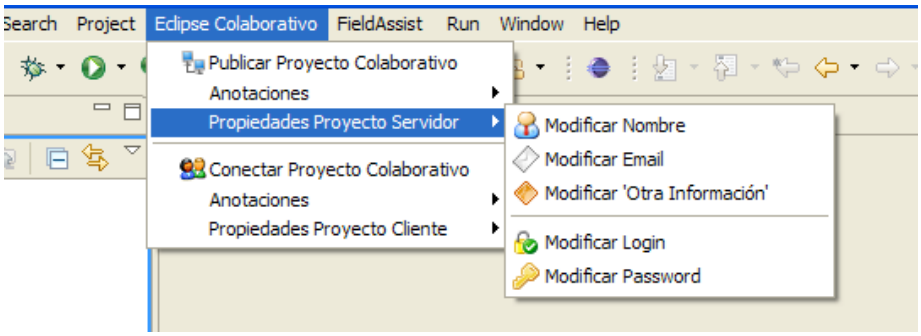


Figura. 6-6. Menú de modificación de proyecto

Todas las opciones siguen el mismo patrón para la modificación. Tomando como ejemplo la modificación del nombre asociado al usuario del proyecto, al elegir la opción correspondiente en el menú se nos abrirá el siguiente cuadro de dialogo:

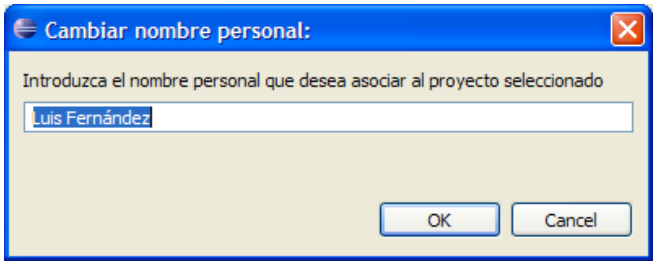


Figura. 6-7. Detalles de la modificación de campos

En el se nos muestra el campo del nombre que estaba establecido hasta el momento. En este momento podemos realizar la modificación que queramos sobre el mismo y una vez terminado pulsaremos “OK”.

Para el resto de opciones de modificación el modo de operar es el mismo.

Es **muy importante** para realizar este tipo de acciones realizar correctamente la selección del proyecto. Se hace especial hincapié en este aspecto debido a que las acciones del entorno trabajan sobre el elemento ‘activo’, es decir, el último elemento seleccionado antes de elegir la acción. De esta manera, si usted tiene como elemento activo la vista de tareas pendientes, aunque tenga seleccionado el proyecto adecuado no podrá realizar la acción.

6.1.2.3 Iniciar una sesión de colaboración

En el momento que tenga un nuevo proyecto fuente creado, está en disposición de empezar una sesión de trabajo de colaboración con su pareja de programación.

Para ello el único paso que debe realizar es acceder a la opción “Publicar Proyecto Colaborativo” dentro del menú “Eclipse Colaborativo”.

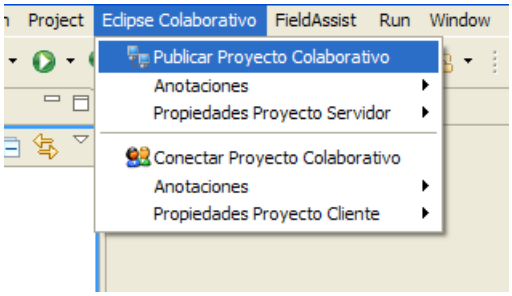


Figura. 6-8. Menú de publicación de proyecto

Una vez haya seleccionado la opción se abrirá una nueva ventana de dialogo, en ella se muestran todos los proyectos disponibles y válidos para trabajar con el plugin que existen en el espacio de trabajo. Se elige el proyecto deseado y se pulsa “OK”.

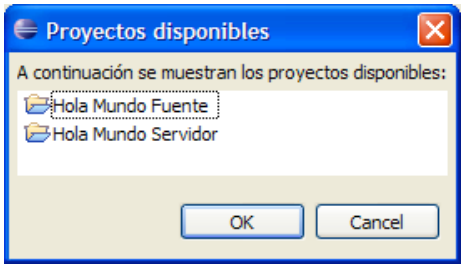


Figura. 6-9 Dialogo de selección de proyecto servidor

Después de esto el proyecto se publica, poniéndose a disposición de la pareja remota de programación. En el caso de que haya ocurrido algún error se notificará al usuario mediante una ventana emergente.

Si la publicación ha sido correcta la perspectiva de trabajo de la ventana actual cambiará a la perspectiva de trabajo “Servidor E-Col” y la acción de Publicar Proyecto Colaborativo quedará seleccionada tanto en el menú como en la barra de herramientas. En la perspectiva activada se disponen los elementos que el plugin ofrece al usuario para trabajar con el entorno: una vista de información, otra de comunicaciones y las habituales vistas para trabajar con los proyectos de programación. Los detalles de las vistas propias del plugin se comentarán en sus respectivos apartados.

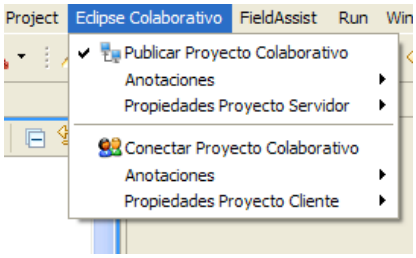


Figura. 6-10 Detalle de proyecto publicado

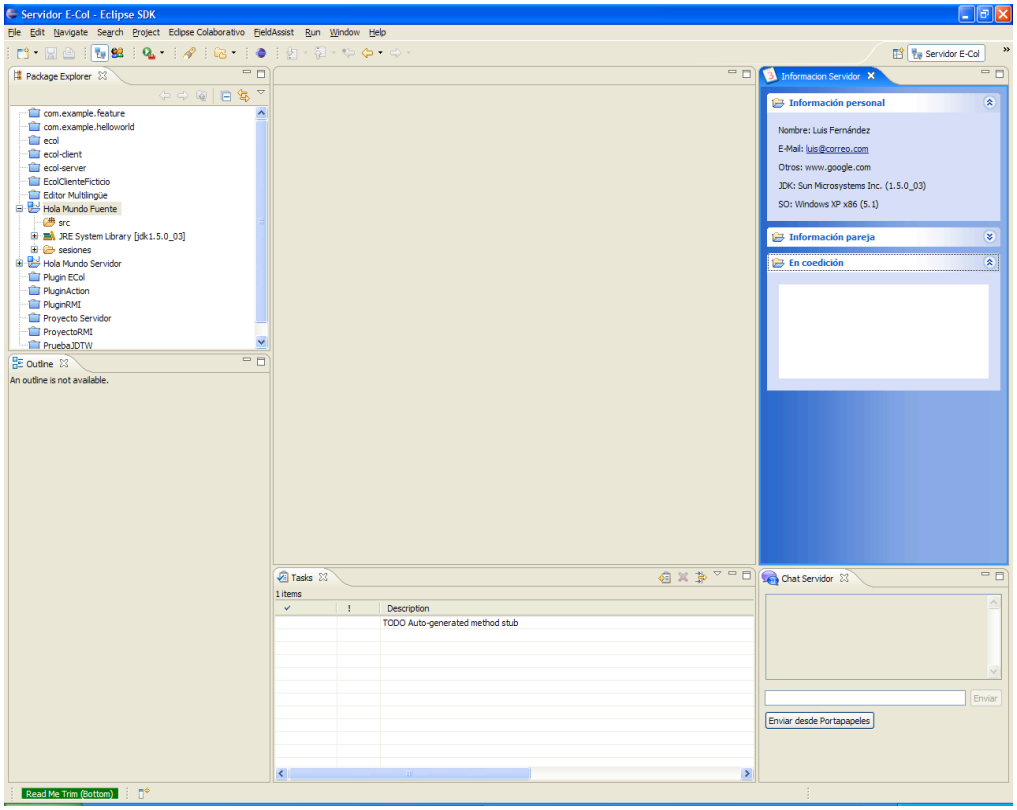


Figura. 6-11 Detalles de la perspectiva de trabajo Servidor

Se recomienda no tener ningún editor abierto sobre los recursos del proyecto fuente antes de iniciar la sesión, de lo contrario no se mantendrá un control de los cambios realizados en dicho editor para enviárselos al cliente, es decir, no estará almacenado como recurso en coedición.

Es **muy importante** tener en cuenta, como se comenta en los requisitos de la aplicación, que el sistema utiliza el puerto 1099 para las comunicaciones RMI entre cliente y servidor. Por ello debe revisar la configuración del firewall y demás elementos de seguridad del sistema en el que esté trabajando.

6.1.2.4 Recibir la conexión de un cliente

Una vez iniciada la sesión de trabajo, estamos en disposición de recibir la conexión de una pareja de programación. El entorno de usuario notifica al usuario servidor mediante el panel de información.

Este panel se muestra por defecto en la perspectiva de trabajo e inicialmente solo muestra la información personal del usuario. Cuando una nueva pareja se conecta, la información asociada se actualiza conforme a los datos personales del usuario conectado.

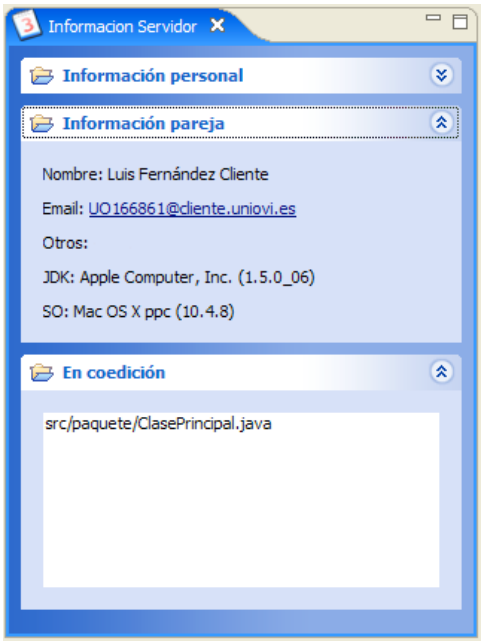


Figura. 6-12 Panel de información de usuarios

6.1.2.5 Modificar un recurso

Cuando hemos iniciado una sesión de trabajo podemos empezar a editar recursos colaborativamente. Para ello deberemos elegir el recurso en el explorador de paquetes laterales y abrirlo.

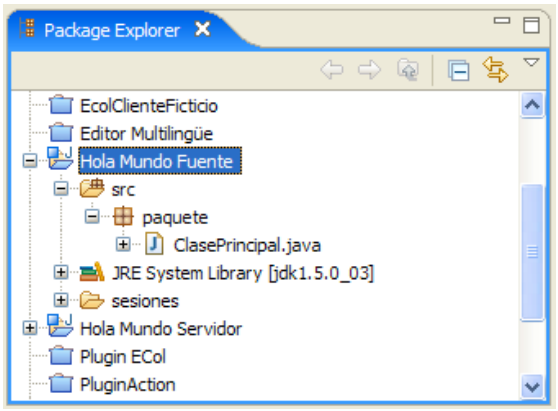


Figura. 6-13. Panel de exploración de recursos

En este momento estaremos en posición de empezar a modificar el documento como de costumbre con un código fuente.

Es **muy importante**, debido a las limitaciones del prototipo no editar ambos miembros de la pareja el mismo recurso a la vez, para ello utilizar el panel de Chat para ponerse de acuerdo en quien edita en cada momento.

6.1.2.6 Realizar anotación

El sistema nos permite añadir anotaciones a los recursos, estas anotaciones representan tareas pendientes o comentarios. Tienen un mensaje, una prioridad y un atributo que indica si la tarea ya se ha completado.

Para realizar una anotación sobre un recurso de la estructura del proyecto deberemos seleccionar el recurso que queremos (ha de ser necesariamente un fichero) e ir a la

opción del menú *Eclipse Colaborativo > Anotaciones (en la parte del servidor) > Nueva Anotacion*.

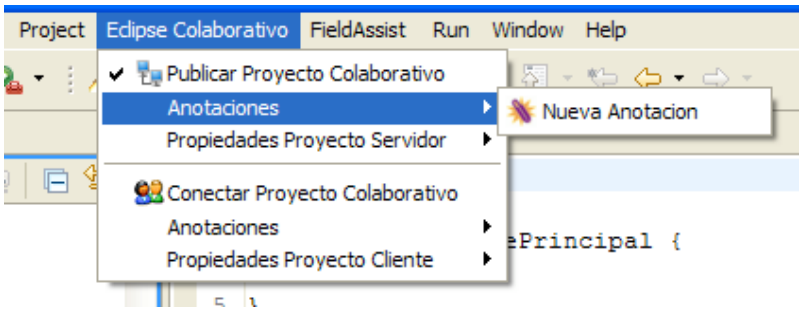


Figura. 6-14 Disposición el menú de anotaciones de servidor

Una vez seleccionada la opción (se recomienda recordar el comentario del apartado 6.1.2.2 sobre la selección adecuada de los elementos), se abrirá una ventana de dialogo en el que se pide que se introduzcan los datos de la anotación.

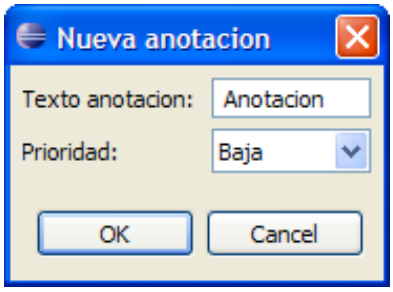


Figura. 6-15 Dialogo de nueva anotación

En caso de que el archivo seleccionado no sea válido, ya sea porque no pertenece al proyecto en curso o no se ha iniciado sesión, se informará al usuario con una ventana de error.

Finalmente, una vez terminada la anotación pulsamos OK y quedará almacenada. Estas anotaciones pueden ser visualizadas desde el panel de tareas en la parte inferior.

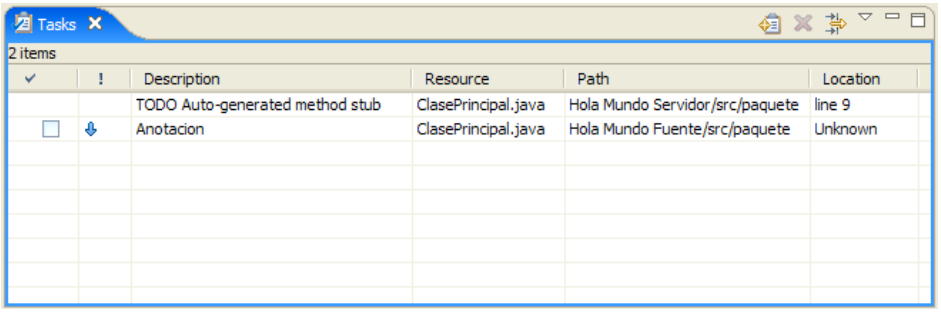


Figura. 6-16 Tabla de anotaciones y tareas

6.1.2.7 Chatear con usuario

Otra de las funcionalidades que el entorno pone a disposición del usuario es la comunicación con la pareja mediante un Chat en modo textual.

Este Chat le informará de los mensajes que el usuario le envíe con un identificador del nombre del usuario y el contenido del mensaje. También le notificará cuando un usuario se haya conectado o desconectado al entorno de manera complementaria al panel de información.

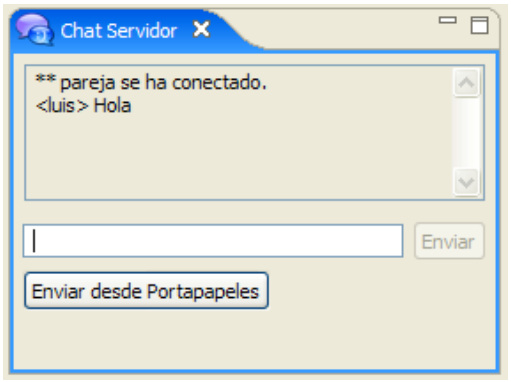


Figura. 6-17 Ventana de chat

Las opciones del Chat son bastante básicas como se puede observar en la figura. Por un lado podemos introducir un mensaje de texto, cuando comencemos a escribir el botón de enviar se habilitará (también se puede enviar el mensaje pulsando Enter). Cuando deseemos enviar un contenido almacenado en el portapapeles usaremos el botón asociado a ello, esta funcionalidad se ha añadido debido a que los mensajes normales sólo se pueden escribir en una sola línea, y es habitual que durante la programación se envíen fragmentos de código.

Se almacena un log de las conversaciones en la carpeta sesiones de la estructura del proyecto.

6.1.2.8 Terminar una sesión de trabajo

Cuando decidamos terminar la sesión de trabajo deberemos ponernos de acuerdo verbalmente con nuestra pareja de programación. Esto es debido a que el sistema no realiza ningún control sobre si el cliente del proyecto está trabajando o no, simplemente se le avisa de que la sesión ha terminado y no puede seguir modificando los recursos del servidor.

Teniendo en cuenta el punto anterior, para terminar la sesión de trabajo deberá volver a seleccionar la opción de lanzar proyecto que previamente estaba activada.

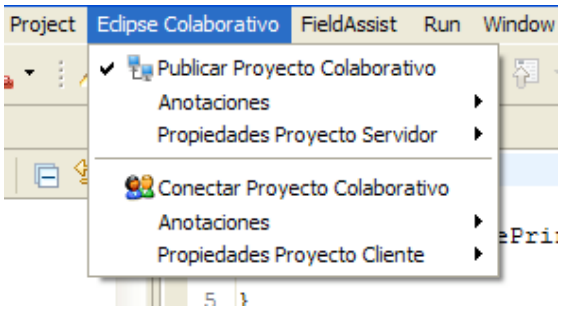


Figura. 6-18 Opción de desconexión

Al seleccionar esta opción de nuevo el sistema detectará de que se está trabajando sobre un proyecto de programación y le preguntará si desea terminar la sesión de colaboración con la pareja.

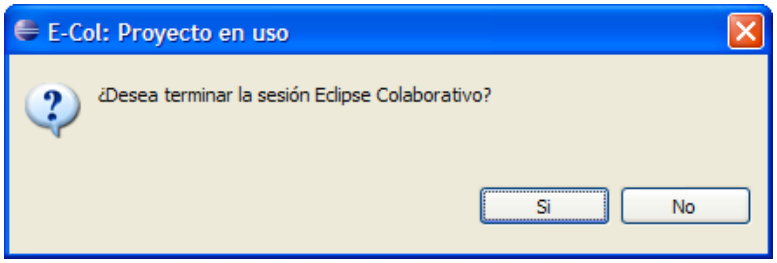


Figura. 6-19 Diálogo de confirmación de finalizar sesión

El resultado de la operación será el cierre de la perspectiva de trabajo y la habilitación de nuevo de la opción de publicar un proyecto.

6.1.3 Manual del usuario cliente

6.1.3.1 Creación de un proyecto cliente

Para la creación de un nuevo proyecto cliente se ha puesto a disposición del usuario un asistente que facilita la tarea. El asistente es accesible desde la opción de menú *File* > *New* > *Project*, esta opción abrirá un dialogo que le permite al usuario elegir el tipo de proyecto a crear:

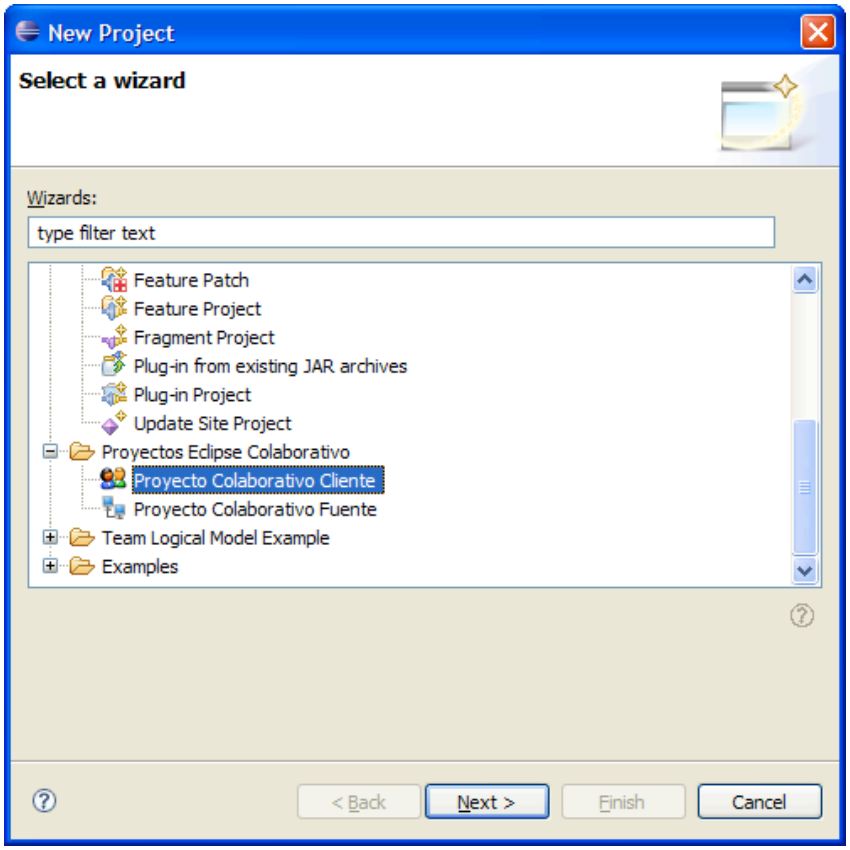


Figura. 6-20 Opción de asistente para proyecto cliente

En nuestro caso deberemos movernos hasta la carpeta "Proyectos Eclipse Colaborativo", dentro de esta opción elegiremos "Proyecto Colaborativo Cliente" y pulsaremos sobre "Next".

En este punto entramos propiamente en la creación del proyecto, el primer paso es establecer el nombre para el proyecto, para ello se ha habilitado el dialogo que se muestra en la figura. Se trata de un campo obligatorio y por tanto, no se podrá continuar

hasta que se haya introducido un nombre válido y que no exista previamente. Una vez concluido pulsaremos sobre el botón “Next” que se habilitará.

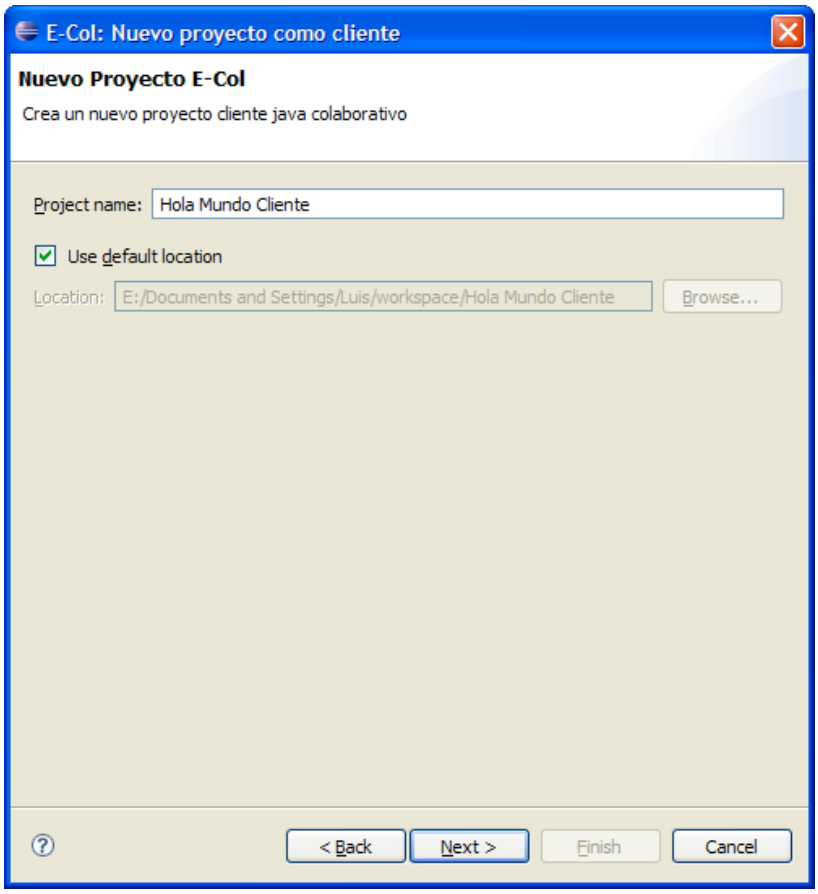


Figura. 6-21 Nombre Proyecto cliente

Finalmente, el último paso para la creación del proyecto es establecer los detalles del mismo. Se deben cumplimentar todos los apartados del cuadro de dialogo del asistente tal y como se muestra en la figura. Aun así, dichos campos se podrán modificar posteriormente desde el menú del plugin.

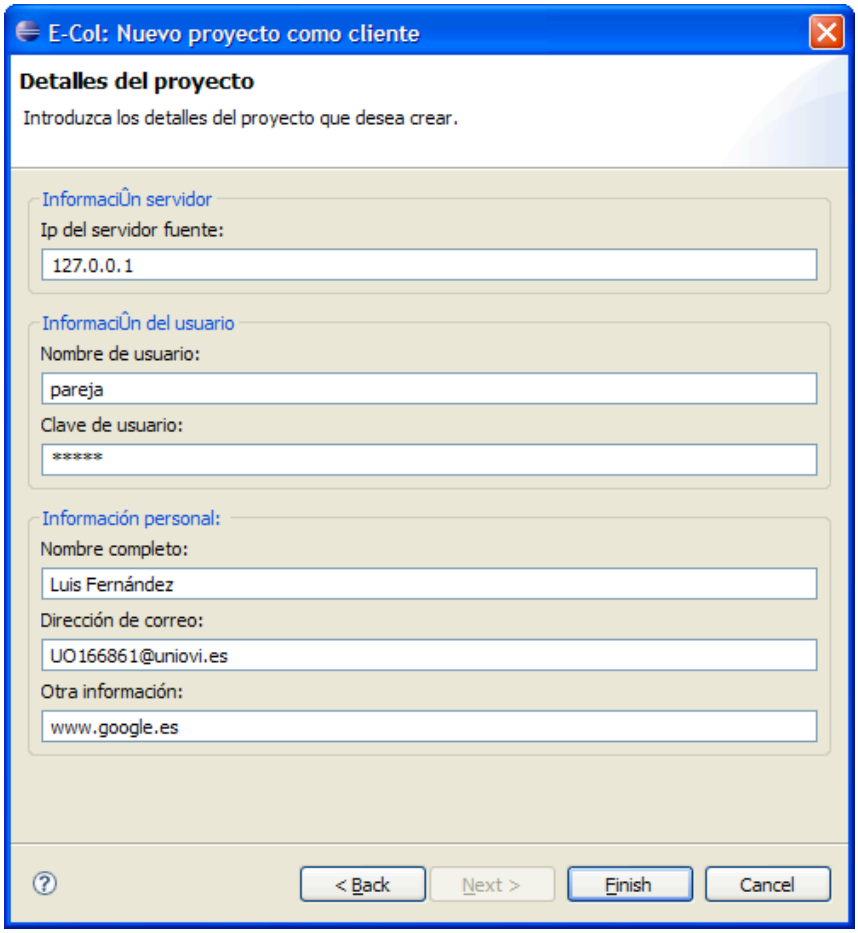


Figura. 6-22 Destalles de proyecto cliente

Una vez hecho esto y pulsando en “Finish" veremos como se crea en el espacio de trabajo un nuevo proyecto con el nombre que hemos seleccionado. Para facilitar la diferenciación con el resto de proyectos se marca con un pequeño lápiz en la esquina superior izquierda del icono.

La estructura de un proyecto fuente consta de los siguientes elementos:

- Carpeta *src*: esta carpeta no debe ser eliminada ni cambiado su nombre. En ella será donde cree los recursos del proyecto. Sólo los recursos que se dispongan en esta carpeta se tendrán en cuenta para el trabajo de colaboración del sistema.
- Carpeta *sesiones*: en esta carpeta se mantendrá un log de las sesiones. Este log se limitará prácticamente a las sesiones de Chat y a las conexiones y desconexiones de la pareja de programación.
- *JRE System Library*: también se asocia en el proyecto a la JRE por defecto del entorno, será la que se emplee para compilar el proyecto que se esté desarrollando.

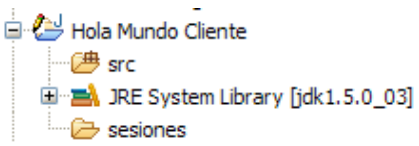


Figura. 6-23 Estructura de un proyecto cliente y su decorador

6.1.3.2 Modificar las propiedades de un proyecto

Una vez creado un proyecto cliente podremos modificar sus detalles establecidos si así se requiere. Para ello el modo de operar es muy similar al establecido en el punto 6.1.2.2.

En primer lugar deberemos seleccionar la carpeta raíz del proyecto cliente cuyos detalles deseamos modificar. Es importante que el proyecto esté abierto, de otra manera el sistema no habilitará la acción correspondiente.

Las opciones modificables varían respecto a las opciones que se pueden modificar en el usuario fuente, en este caso encontramos un nuevo campo de host.

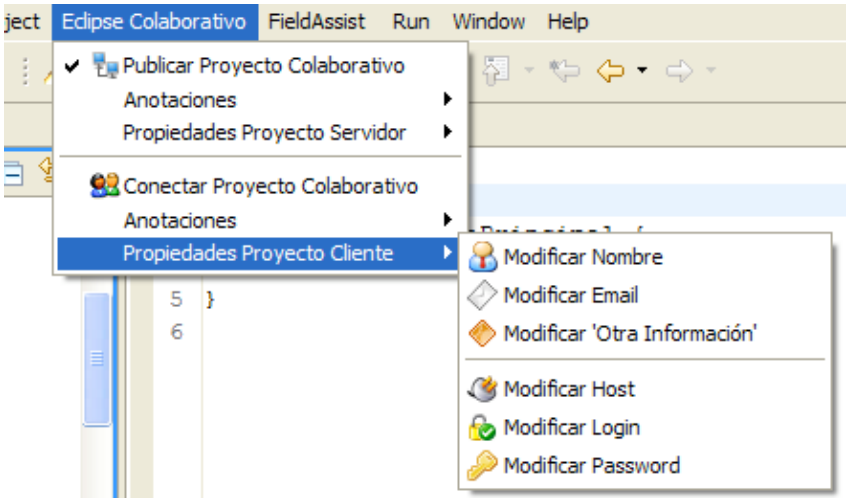


Figura. 6-24 Menú de propiedades de proyecto cliente

La forma de modificar cada una de las opciones es la misma que en el caso de modificar los detalles de un proyecto fuente. Elegimos la opción deseada y se nos abrirá un panel para introducir los cambios, una vez finalizada la modificación seleccionamos Ok.

De nuevo hay que **tener en cuenta el comentario** realizado en el punto 6.1.2.2 acerca de la forma adecuada de seleccionar un elemento del entorno.

6.1.3.3 Conectar con un proyecto fuente

Una vez que hemos creado un proyecto cliente con la información oportuna para conectarnos al proyecto fuente, podremos iniciar la sesión de trabajo.

El primer paso es seleccionar la opción de Conectar Proyecto colaborativo dentro del menú principal Eclipse Colaborativo

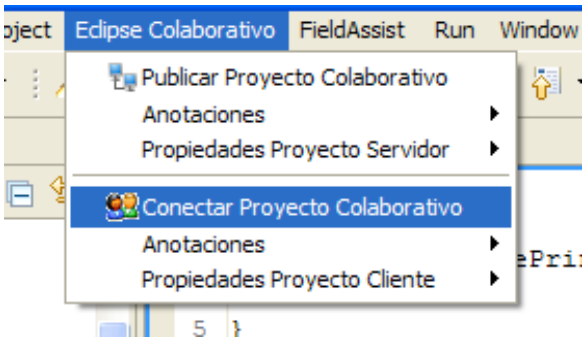


Figura. 6-25 Conexión proyecto

Una vez elegida la opción se comenzará la conexión. Si todos los datos son correctos y el servidor está activo se comenzará la carga de la estructura del proyecto en la estructura local. Durante la realización de esta operación, que puede llevar tiempo, se mostrará una ventana con una barra de progreso al usuario.

Una vez terminada la descarga de la estructura y finalizada la conexión, se cambiará la perspectiva de trabajo del usuario a la de trabajo colaborativo como cliente.

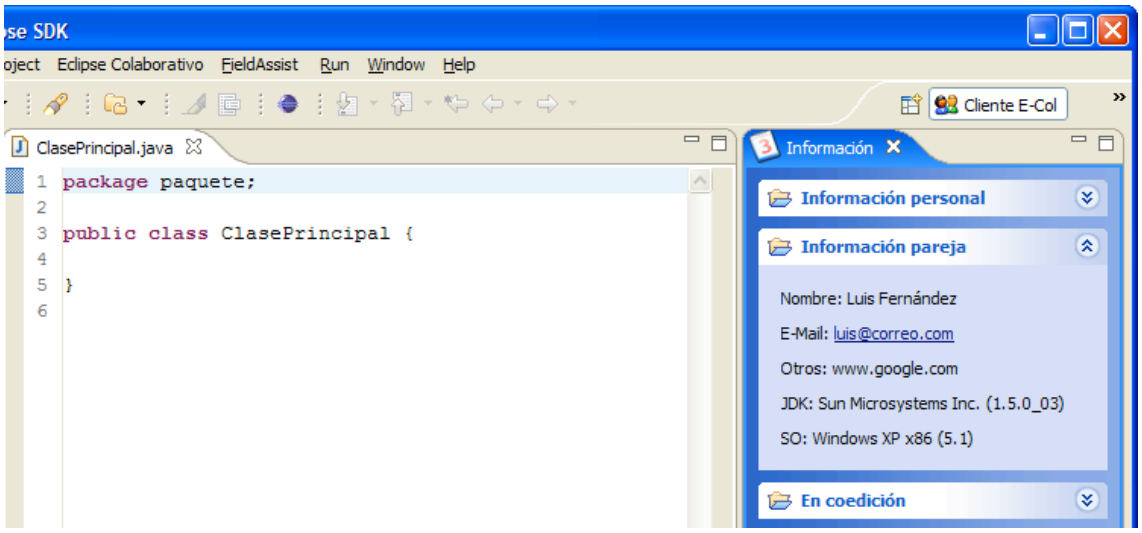


Figura. 6-26 Detalle de inición de sesión cliente

Como vemos en la figura, la perspectiva abierta muestra un panel de información con detalles sobre la pareja con la que estamos trabajando. Toda su información personal, así como datos específicos del sistema que está utilizando en la sesión actual de trabajo.

6.1.3.4 Modificar un recurso

Cuando hemos iniciado una sesión de trabajo podremos empezar a modificar recursos cuando estén activos para coedición. Para ello los recursos deberán figurar en el panel de coedición.

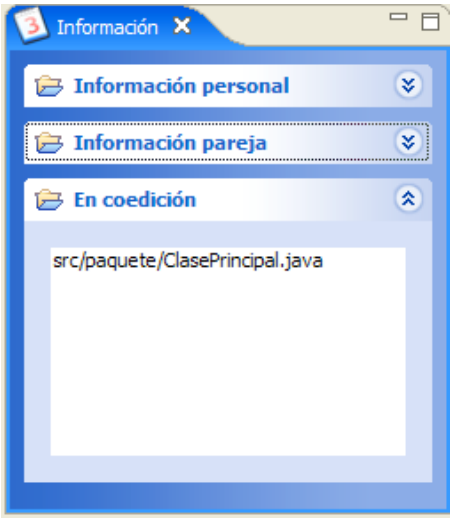


Figura. 6-27 Listado de recursos en coedición

Como funcionamiento normal, cuando el servidor abre un recurso para coeditar y estamos trabajando en la sesión se abre automáticamente en nuestra sesión. De todas maneras podemos abrir el recurso apropiado en caso de que no tengamos el editor

asociado abierto, para ello simplemente seleccionaremos el recurso en el explorador de paquetes lateral.

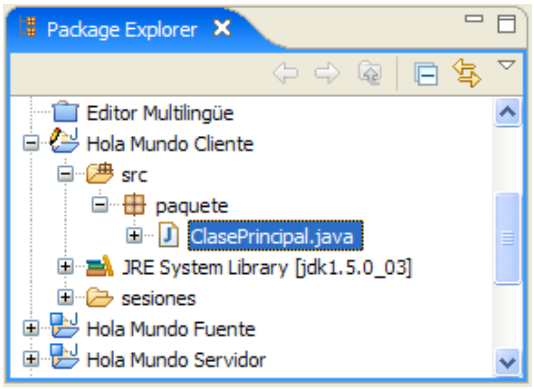


Figura. 6-28 Panel de recursos en un cliente

Se recomienda no modificar los recursos del proyecto cliente que no figuren en el panel de coedición. De esta forma se evitan irregularidades entre servidor y cliente. También hay que tener en cuenta que cuando se empieza a coeditar un recurso su contenido inicial se sobreescribe para sincronizarlo con el servidor, por tanto cualquier modificación que haya realizado sin que se notifique al servidor se pierde.

Es **muy importante**, debido a las limitaciones del prototipo no editar ambos miembros de la pareja el mismo recurso a la vez, para ello utilizar el panel de Chat para ponerse de acuerdo en quien edita en cada momento.

6.1.3.5 Manejar anotaciones

El sistema también permite al usuario cliente crear anotaciones sobre los recursos del proyecto. El sistema sólo le permite visualizarlos y crearlos, si elimina alguno de ellos no se manifestará en el proyecto fuente, sólo en local.

Para crear una anotación deberemos seleccionar el recurso de nuestro proyecto sobre el que estamos trabajando y posteriormente ir al menú *Eclipse Colaborativo* > *Anotaciones* > *Nueva Anotación* (En este caso el menú Anotaciones es el referido al bloque del cliente).

Con ello se nos abrirá una ventana de dialogo en la que estableceremos los detalles de la anotación. Una vez finalizada la nota se envía al servidor fuente para que la almacene.

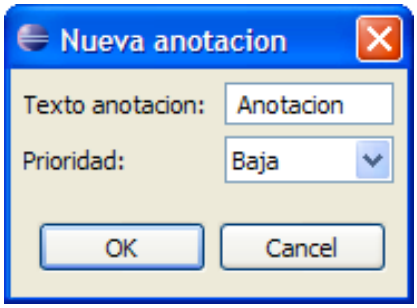


Figura. 6-29 Nueva anotación

Como no tenemos permiso para modificar o eliminar las anotaciones, deberemos actualizarlas eligiendo la opción del menú *Eclipse Colaborativo* > *Anotaciones* > *Refrescar anotaciones*. Esta operación actualizará las anotaciones, marcará como realizadas las que se hayan realizado y borrará las que ya no existan en el proyecto fuente.

Las anotaciones pueden ser vistas en el panel de tareas en la parte inferior de la perspectiva de trabajo.

6.1.3.6 Chatear con usuario fuente

El sistema permite al usuario cliente mantener un Chat textual con el usuario fuente del proyecto. Este Chat le permite enviarse mensajes de texto y fragmentos de código desde el portapapeles.

El uso es muy básico y queda explicado en el punto 6.1.2.7 del manual del usuario fuente.

Todas las conversaciones son almacenadas en el archivo Chat.log dentro de la carpeta sesiones, dentro de la estructura del proyecto cliente sobre el que estamos trabajando en ese momento.

6.1.3.7 Terminar sesión de trabajo

La finalización de la sesión de trabajo en el cliente puede ser solicitada por el propio usuario o a raíz de la desconexión del usuario fuente.

En el primer caso, el cliente deberá escoger la opción del menú *Eclipse Colaborativo* > *Conectar Proyecto Colaborativo* que en este momento estará marcada. El sistema preguntará al usuario si desea terminar la sesión.

En el segundo caso, el usuario fuente terminará la sesión y se nos avisará mediante una ventana emergente sobre ello.

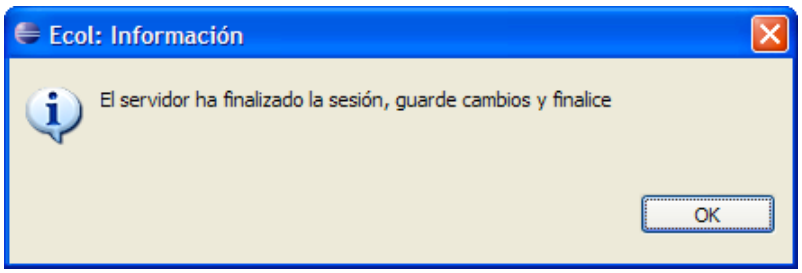


Figura. 6-30 Informe de fin de sesion del servidor

En este caso deberemos tener en cuenta que los cambios que hagamos a partir de aquí no se manifestarán en el servidor fuente. Después de ser avisados de que el servidor ha terminado, procederemos como en el primer caso a terminar la sesión. Cuando el servidor termina la sesión parte de los recursos son liberados, pero deberemos terminar nosotros la sesión para que todo el entorno quede correctamente.

La estructura del proyecto se mantendrá intacta hasta la próxima vez que inicie la sesión de trabajo, momento en el que se actualizará la estructura con la situación actual del proyecto. Por ello es recomendable que se guarde una copia de los recursos si lo desea.

6.2 Manual de instalación

A continuación se muestran los pasos necesarios para realizar la instalación del plugin dentro de la plataforma Eclipse. También se muestran los requisitos básicos que ha de cumplir el sistema destino para el plugin.

6.2.1 Requisitos

Los requisitos que el sistema ha de cumplir para poder utilizar el plugin se citan a continuación:

- Tener el entorno Eclipse instalado en el sistema. La versión en la que ha sido probado el plugin es la 3.2. Es la versión mínima necesaria puesto que algunos elementos utilizados no existen en versiones anteriores.
- La maquina virtual Java requerida es la 1.5. Esta ha sido la versión en la que se ha probado y desarrollado el proyecto, aunque la versión 1.4.2 debería ser compatible se recomienda utilizar la 1.5.
- El sistema servidor debe tener accesible remotamente el puerto del registro RMI (1099) para permitir el acceso a los clientes remotos.

6.2.2 Instalación del plugin

Para instalar el plugin en el entorno Eclipse es necesario copiar el archivo JAR proporcionado en la carpeta *plugins* dentro del directorio de la aplicación Eclipse. Una vez hecho esto el sistema al iniciar cargará automáticamente el plugin y lo hará disponible al usuario para comenzar con su funcionalidad.

6.3 Manual del programador

6.3.1 Organización general del plugin

La organización interna del proyecto se compone de tres paquetes principales, estos son:

- `ecol.servidor`: este paquete contiene todo el código referente al manejo de la sesión de trabajo del usuario fuente. A continuación se comenta su organización interna.
- `ecol.cliente`: en este paquete se disponen los elementos referentes al usuario cliente de la aplicación. A continuación se comenta cual es su organización.
- `ecol.comun`: por último, este paquete contiene elementos comunes a ambos tipos de usuario. También se incluyen clases de apoyo con métodos estáticos, estas sirven para realizar tareas frecuentes en la aplicación y no repetir innecesariamente código.
    - `ecol.comun.comunicaciones`: en este paquete se encuentran todos los elementos referidos al controlador de comunicaciones y los paneles de comunicaciones, en la versión actual, solamente el panel de Chat.
    - `ecol.comun.dialogos`: los diálogos comunes a ambos paquetes de cliente y servidor se disponen en este paquete.
    - `ecol.comun.excepciones`: todas las excepciones del sistema se almacenan en este paquete.
    - `ecol.comun.natures`: las naturalezas establecidas para los proyectos se definen en las clases incluidas en este paquete.

Los dos primeros paquetes de servidor y cliente, se dividen a su vez en varios subpaquetes, a continuación se enumeran todos ellos:

- `acciones`: en este paquete se almacenan todas las definiciones de las acciones que se definen en el entorno Eclipse.
- `dialogos`: este paquete almacena todos los cuadros de dialogo que se emplean en la aplicación.
- `perspectivas`: las clases que implementan la definición de las perspectivas se deben disponer en este paquete.
- `recursos`: para el control de recursos del entorno se ha creado este paquete especialmente dedicado.
- `sesion`: en este paquete van todas las clases referidas al control de la sesión de trabajo, conexión, etc…
- `vistas`: las vistas de las que se componen las perspectivas se almacenan en este paquete.
- `wizards`: en este paquete se disponen los asistentes creados para el entorno.

6.3.2 Organización de los proyectos:

Los proyectos que se emplean para el trabajo colaborativo tienen la siguiente estructura de directorios y elementos:

- Carpeta *src*: esta carpeta tiene como objetivo almacenar todos los recursos del proyecto, ya sea como fuente de proyecto o como cliente. Actualmente su nombre es obligatoriamente src, y no debería ser cambiado. Sólo los recursos dispuestos en esta carpeta de trabajo son controlados por el sistema.
- Carpeta *sesiones*: esta carpeta se ha creado con el fin de guardar un registro de las sesiones de trabajo realizadas. La versión actual emplea esta carpeta para almacenar un log del Chat de la sesión. Se almacena en el archivo Chat.log.
- *JRE System Library*: la versión actual del proyecto sólo trabaja con proyectos de tipo Java, por ello se ha incluido, en la creación de los proyectos una referencia al contenedor de la JRE por defecto del sistema en el que se lanza la aplicación.

La naturaleza que se establecen para los proyectos cliente y servidor son, *ecol.nature.clientnature* y *ecol.nature.servernature* respectivamente.

Referenciar recursos del proyecto

En la implementación del proyecto ha sido importante identificar los recursos de una manera única.

Su nombre no podría haber cumplido el criterio de unicidad, puesto que pueden haber varios documentos con el mismo nombre en diferentes paquetes del proyecto. También hay que tener en cuenta que los nombres de los proyectos en cada uno de los participantes es diferente por tanto una ruta completa desde la raiz del proyecto no resulta válida.

La manera para establecer un identificador único a los recursos es su ruta completa exceptuando el primer segmento referente al nombre del proyecto. Es decir todos los identificadores de los recursos empezarán con el identificador *src* referente a la carpeta que los contiene y que hemos convenido no es modificable y deberá existir. A continuación se muestra una forma práctica de obtener dicho ID a partir de un recurso `IFile` del proyecto:

```
IFile fichero=[…];
fichero.getFullPath().removeFirstSegments(1).toString();
```

Ampliar las funcionalidades de los tipos de proyecto

Los diferentes tipos de proyecto que tiene la aplicación (proyectos fuente y cliente) están determinados por las naturalezas, comentadas en el apartado de implementación.

Cuando un proyecto se crea y se establece una determinada naturaleza, el sistema llama al método `configure()` de la clase `IProjectNature` asociada a la naturaleza.

Por tanto, la implementación de dicho método es un buen lugar para realizar las operaciones oportunas para ampliar la estructura en particular de un tipo de proyecto, su semántica o ciertos aspectos funcionales.

7 CONCLUSIONES Y AMPLIACIONES

7.1 Conclusiones

Uno de los objetivos de la realización del prototipo ha sido evaluar la capacidad del entorno de desarrollo Eclipse para la integración de una herramienta de desarrollo colaborativo.

Los resultados que se han obtenido durante la realización del proyecto han demostrado que la plataforma elegida tiene potencial para emplearla en la realización de una aplicación útil y productiva para un desarrollador. El entorno permite a los desarrolladores de plugins abstraerse de partes de la implementación propias de una herramienta de desarrollo de este estilo. Apoyarse en toda la funcionalidad existente y bien estructurada de Eclipse permite obtener resultados funcionales y prácticos.

Aunque no todo han sido ventajas durante el desarrollo, la complejidad de la API ha provocado que algunos de los requisitos iniciales no se hayan podido llevar a cabo. Una de las principales causas de dificultad en el desarrollo ha sido el emplear, como modelo de aplicación, un plugin dentro de un entorno tan amplio y con tanta funcionalidad como Eclipse. Esto ha requerido emplear mucho tiempo en documentarse sobre la API de programación de la plataforma, además de comprender la organización interna de los diferentes componentes y la 'filosofía' de desarrollo seguida por los plugins del entorno.

Otro de los problemas esenciales ha sido el propio objetivo de la herramienta. El soporte general y que se potencia en los entornos actuales es el trabajo asíncrono (empleo de CVS, etc…), y en la API de Eclipse la documentación referente al módulo de trabajo en equipo se centra en ese tipo de trabajo. Por tanto, ha sido difícil decidir la línea de diseño 'ideal' de la aplicación en ciertos aspectos, incluidas las restricciones sobre el análisis inicial.

Finalmente, el empleo de RMI como medio de comunicación entre los participantes del proyecto también ha requerido de un tiempo de documentación y de comprensión de la estructura de las aplicaciones de este estilo. En muchas casos se perdió mucho tiempo en problemas de esta parte del desarrollo, para su solución se emplearon foros de desarrolladores especializados en RMI.

A pesar de todos estos inconvenientes, la realización del proyecto ha resultado muy interesante desde dos puntos de vista. Por un lado, como desarrollador, ha sido muy interesante afrontar el desarrollo de una herramienta de este estilo, donde nos tenemos que ceñir a una especificación y ciertos límites de desarrollo. Por otro, la documentación previa realizada sobre el trabajo colaborativo, la programación por parejas distribuida y en general, el trabajo en equipos síncrono, me ha resultado muy interesante, la considero una línea de investigación y desarrollo útil y necesaria para potenciar y mejorar la tarea de los desarrolladores.

7.2 Ampliaciones

Como ya hemos visto, se ha tenido que poner ciertas restricciones al diseño de la aplicación, fruto de la integración de la herramienta en un entorno como Eclipse.

A continuación se proponen una serie de ampliaciones y mejoras que deberán marcar la línea de desarrollo futuro de la herramienta:

Aspectos generales

- Un sistema de ayuda completo integrado en la aplicación.
- Soporte para más de dos usuarios en el entorno y mejorar el sistema de edición colaborativa, haciéndolo más funcional. En este punto se deben mejorar aspectos como la utilidad de deshacer cambios ó subsanar problemas en la edición en tiempo real de varios usuarios.
- Creación de varios roles de usuarios y sus restricciones. Este punto aportaría mayor funcionalidad a la aplicación en cuanto a tareas de permisos de los usuarios (quién puede crear recursos, borrarlos, etc…).

Gestión de recursos

En la versión actual, la gestión de recursos no es del todo flexible. Tiene ciertas limitaciones como ya se ha visto.

Las ampliaciones referentes a la gestión de recursos deberían combinarse con los roles de usuarios. Completar toda la funcionalidad sobre recursos, en cuanto a desplazamiento de los mismos, cambiar nombres, etc…

Diseño de la arquitectura

- Hacer totalmente la implementación del servidor del proyecto de los participantes. Este es un punto importante, conseguir desarrollar un servidor del proyecto con una especificación de comunicación con los clientes bien definida y funcional. Si se diseña correctamente podría derivar en la interoperabilidad de diferentes aplicaciones cliente, es decir, podría haber clientes Eclipse, aplicaciones web Java, módulos integrados en otros entornos, etc…

Información del desarrollo

En el prototipo actual existe una cierta carencia de información sobre el trabajo que los usuarios están realizando.

Se debería ampliar esta funcionalidad para que el sistema permita informar sobre que está haciendo cada usuario, donde está editando, etc… También debería mantenerse un cierto control estadístico de todas las tareas del desarrollo.

APÉNDICE A. BIBLIOGRAFÍA

[BRAVO2004]: Bravo, Crescencio, Redondo, M.A., Ortega, M.; *(2004) Aprendizaje en grupo de la programación mediante técnicas de colaboración distribuida en tiempo real*. Actas del V Congreso Interacción Persona Ordenador, Lleida.

[ECLIPSE32]: http://help.eclipse.org/help32/index.jsp**.**

[FAQSWT]: http://www.eclipse.org/swt/faq.php.

[KOCH95b]: *The Collaborative Multi-User Editor Project Iris*. Michael Koch. TUM-I9524, Inst. für Informatik. Technische Univ. München, Aug. 1995

[IRISWEB]: http://www11.in.tum.de/proj/iris/.

[NIEL2001c]: Nielsen , J. *Beyond Accessibility: Treating Users with Disabilities as People*. Revised Three Years Later. Alertbox, On-line Journal: http://www.useit.com/alertbox/20011111.html

[SCHLICHTER]: *Workspace Awareness for Distributed Teams*, Johann Schilchter, Michael Koch, and Martin Bürger.

[SHEN2000]: Shen, H. & Sun, C.; (2000) *RECIPE: a prototype for Internet-based real-time collaborative programming*. Proceedings of the 2nd Annual International Workshop on Collaborative Editing Systems in conjunction with ACM CSCW Conference, December 2-6, Philadelphia, Pennsylvania, USA.

[STOTTS 2003]: Stotts, D., Williams L., Nagappan, N., Baheti P., Jen, D., Jackson A., *Virtual Teaming: Experiments and Experiences with Distributed Pair Programming*.

[PÉREZ 2006]: *Clasificación de Usuarios Basada en la Detección de Errores Usando Técnicas de Procesadores de Lenguaje*. Juan Ramón Pérez Pérez.

[WILLIAMS2001]: Williams, L. & Upchurch, R.L. (2001*) In Support of Student Pair-Programming*. ACM SIGCSE Conference for Computer Science Educators, February.

APÉNDICE B. DESCRIPCIÓN DETALLADA DE CLASES

En este apartado se hará una descripción detallada de las clases que se han implementado en el proyecto. La información ha sido extraida de la documentación JavaDoc del proyecto.

ecol

Class Activator

La clase Activator se encarga de controlar el ciclo de vida del plugin.

---

ecol.cliente

Class EcolCliente

Esta clase sirve al plugin para controlar el modelo de trabajo, desde aquí se solicita el comienzo de una sesión de trabajo sobre un proyecto determinado y su terminación.

Detalles de Métodos

estaConectadoProyecto

```
public static boolean estaConectadoProyecto()
```

Nos permite determinar si se ha iniciado sesión de trabajo con algún proyecto.

Retorno:

Devuelve verdadero o falso dependiendo si está conectado o no.

conectarProyecto

```
public static boolean conectarProyecto(String name,
IWorkbenchWindow window) throws ConnectIOException,
RemoteException, NotBoundException, ConexionParejaException,
AutenticacioInvalidaException, ParejaEstablecidaException
```

Este método estático se encarga de iniciar la conexión a un proyecto fuente a partir de la información del proyecto pasado como parámetro y que se tomará como proyecto local.

Parámetros:

name - Nombre del proyecto del espacio de trabajo sobre el que se trabajará.

window - Ventana que se tomará como la ventana de trabajo para el plugin.

Retorno:

Devuelve verdadero o falso dependiendo si se realizó bien la conexión o no.

Throws:

`java.rmi.ConnectIOException` - Excepción lanzada ante un error de conexión con la fuente.

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

`java.rmi.NotBoundException` - Excepción lanzada cuando no se ha logrado obtener un objeto remoto con un nombre dado.

`ConexionParejaException` - Excepción lanzada cuando el servidor obtiene errores al contactar con la pareja recién conectada.

`AutenticacioInvalidaException` - Excepción lanzada cuando los datos de acceso no son válidos.

`ParejaEstablecidaException` - Excepción lanzada cuando ya existe una pareja conectada al proyecto fuente.

setControladorSesion

```
public static void setControladorSesion(ControladorSesionCliente
ctrlSesion)
```

Establece el controlador de la sesión de trabajo.

Parámetros:

ctrlSesion - Nuevo controlador de la sesión.

getControladorSesion

```
public static ControladorSesionCliente getControladorSesion()
```

Obtiene el controlador de sesión de trabajo actual.

Retorno:

Retorna el controlador de la sesión

terminarSesionPareja

```
public static void terminarSesionPareja() throws RemoteException
```

Método invocado cuando se solicita una desconexión por parte del cliente.

Throws:

`java.rmi.RemoteException` - Excepción lanzada por un fallo en la comunicación RMI.

getVentanaTrabajo

```
public static IWorkbenchWindow getVentanaTrabajo()
```

Obtiene la referencia a la ventana de trabajo

Retorno:

Devuelve una referencia a `IWorkbenchWindow` que representa la ventana de trabajo.

limpiarSesion

```
public static void limpiarSesion()
```

Método invocado para limpiar todos los recursos de la sesión. Útil cuando se produce algún error de conexión y queremos liberarlos

abortarSesion

```
public static void abortarSesion(java.lang.String razon)
```

Método invocado cuando ha habido un error importante en la comunicación

Parámetros:

`String` - razón por la cual se aborta.

---

ecol.cliente.acciones

Class CambiarEmailAction

Clase asociada a la acción para cambiar el campo de e-mail en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class CambiarHostAction

Clase asociada a la acción para cambiar el campo de host en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class CambiarLoginAction

Clase asociada a la acción para cambiar el campo de login de acceso en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class CambiarNombreAction

Clase asociada a la acción para cambiar el campo de nombre personal en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class CambiarOtraInfoAction

Clase asociada a la acción para cambiar el campo de información adicional en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class CambiarPasswordAction

Clase asociada a la acción para cambiar el campo de password de acceso en un proyecto Eclipse Colaborativo Cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class ConectarProyectoAction

Implementación de la acción asociada a conectar un proyecto cliente con el proyecto fuente central.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class NuevoAvisoProgAction

Clase encargada de la implementación de la acción de crear una nueva anotación en los recursos del proyecto cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.acciones

Class RefrescarAnotacionAction

Esta clase representa la implementación de la acción de refrescar las anotaciones de los recursos incluidos en el proyecto cliente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.cliente.dialogos

Class SeleccionProyectoDialog

Clase que representa el cuadro de dialogo que se le muestra al usuario cuando selecciona el proyecto que desea para iniciar la sesión de trabajo de colaboración con la fuente de proyecto.

Detalles de Métodos

SeleccionProyectoDialog

```
public SeleccionProyectoDialog(org.eclipse.swt.widgets.Shell parentShell, org.eclipse.ui.IWorkbenchWindow window)
```

Constructor para el cuadro de dialogo de selección de proyecto.

Parámetros:

parentShell - Shell padre en la que se dispondrá dialogo.

window - `IWorkbenchWindow` sobre la que se lanza el dialogo.

createDialogArea

```
protected org.eclipse.swt.widgets.Control createDialogArea(Composite parent)
```

Crea la parte superior del cuadro de dialogo (por encima de la barra de botones).

Parámetros:

parent - Composite que representa el area donde se contendrá el cuadro de dialogo.

Retorno:

Referencia al controlador del area de dialogo.

configureShell

```
protected void configureShell(org.eclipse.swt.widgets.Shell newShell)
```

Configura el shell en el que se dispondrá el cuadro de dialogo. principalmente se inicializa el título y demás parámetros de aspecto.

Parámetros:

newShell - Shell en el que se dispondrá el cuadro de dialogo.

okPressed

```
protected void okPressed()
```

Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo creado.

isSesionIniciada

```
public boolean isSesionIniciada()
```

Permite saber si se ha iniciado sesión o no al conectar el proyecto cliente al servidor.

Retorno:

Verdadero si se ha iniciado la sesión correctamente o falso en caso contrario.

---

ecol.cliente.perspectivas

Class TrabajoClientePerspectiva

Clase encarga de componer la vista de trabajo del usuario cliente del proyecto.

Detalles de Métodos

createInitialLayout

```
public void createInitialLayout(org.eclipse.ui.IPageLayout
layout)
```

Crea la composición inicial de la perspectiva (vistas, acciones,...). Esta podrá ser modificada por el usuario desde las opciones del menú Window.

Parámetros:

layout - Representa la composición de la perspectiva

---

ecol.cliente.recursos

Class IServicioRecursosCliente

Interfaz que será mediante la cual el usuario fuente se comunique con los recursos del usuario cliente.

Detalles de Métodos

borrarRecurso

```
public void borrarRecurso(java.lang.String id) throws
RemoteException
```

Método que permite al usuario fuente avisar al usuario cliente cuando un recurso ha sido borrado de la estructura del proyecto.

Parámetros:

id - String que representa el identificador del recurso que ha sido borrado en el proyecto fuente.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

cargarEstadoInicial

```
public void cargarEstadoInicial() throws RemoteException
```

Método encargado de cargar el estado inicial en lo que refiere a los recursos que se están coeditando.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

finCoEdicion

```
public void finCoEdicion(java.lang.String id)
throws java.rmi.RemoteException
```

Informa al usuario cliente de que ha terminado la coedición del recurso cuyo identificador viene determinado por el parámetro id.

Parámetros:

id - String que representa el identificador del recurso que ha finalizado su coedición.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

inicioCoEdicion

```
public void inicioCoEdicion(RecursoCompartido recurso) throws RemoteException
```

Informa al usuario cliente de que se ha iniciado la coedición del recurso pasado como parámetro. El cliente debe sincronizar los contenidos antes de empezar a trabajar sobre él.

Parámetros:

recurso - `RecursoCompartido` con información del recurso que se va a empezar a coeditar.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

modificarRecurso

```
public void modificarRecurso(RecursoCompartido recurso, int offset, int length, java.lang.String text) throws java.rmi.RemoteException
```

Método que permite a la fuente del proyecto enviar las modificaciones sobre los recursos al usuario cliente para que los almacene en su estructura de proyecto.

Parámetros:

recurso - Objeto que encapsula información sobre el recurso que se está modificando.

offset - Desplazamiento en el documento de la modificación.

length - Longitud del area modificada.

text - Texto a sustituir en el area modificada.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

setAnotacion

```
public void setAnotacion(java.lang.String id, Anotacion anotacion) throws RemoteException
```

Método que permite al usuario fuente mandar nuevas anotaciones para los recursos del proyecto del usuario cliente.

Parámetros:

id - String que representa el identificador del recurso que ha recibido una nueva anotación.

anotacion - Objeto `Anotacion` que encapsula la información de la nueva anotación.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

---

ecol.cliente.recursos

Class CodigoFuenteListener

Clase que implementa el `IDocumentListener`, encargada de mantener el control de las modificaciones de los documentos sobre los que trabaja.

Detalles de Métodos

CodigoFuenteListener

```
public CodigoFuenteListener()
```

Crea un nuevo listener de código fuente. Este listener se asocia con el controlador de recursos del cliente, destino al que enviará todas las modificaciones que reciba.

documentChanged

```
public void documentChanged(org.eclipse.jface.text.DocumentEvent event)
```

Método invocado cuando un documento ha sido modificado. Se utilizará para mantener el control de las modificaciones en los documentos.

Parámetros:

event - `DocumentEvent` que encapsula la modificación realizada en el método.

---

ecol.cliente.recursos

Class RecursosPerspectivaListener

Clase que se encarga de escuchar en la perspectiva que se asocia sobre nuevos editores abiertos o cerrados para mantener un control actualizado de los recursos sobre los que está trabajando el cliente.

Detalles de Métodos

perspectiveActivated

```
public void perspectiveActivated(org.eclipse.ui.IWorkbenchPage page, org.eclipse.ui.IPerspectiveDescriptor perspective)
```

Notifica al listener de que una perspectiva en la pagina ha sido activada.

Parámetros:

page - pagina que contiene la perspectiva activada.

perspective - el descriptor de la perspectiva que ha sido activada.

perspectiveChanged

```
public void perspectiveChanged(org.eclipse.ui.IWorkbenchPage page,org.eclipse.ui.IPerspectiveDescriptor perspective, java.lang.String changeId)
```

Notifica al listener de que una perspectiva ha cambiado de alguna manera. (un editor ha sido abierto, una vista, etc...)

Parámetros:

page - la página que contiene la perspectiva afectada.

perspective - el descriptor de la perspectiva.

changeId - constante CHANGE_* presente en `IWorkbenchPage`

---

ecol.cliente.recursos

Class ServicioRecursosClienteImpl

Esta clase se encarga de gestionar todo lo referido a los recursos del proyecto cliente de colaboración.

Detalles de Métodos

ServicioRecursosClienteImpl

```
public ServicioRecursosClienteImpl(IWorkbenchWindow window, String nombreProyecto) throws RemoteException
```

Constructor del servicio de recursos del cliente. Se encarga de inicializar la estructura del proyecto cliente, añadir los listeners correspondientes y demás estructuras de datos.

Parámetros:

window - Referencia al `IWorkbenchWindow` sobre la que se está trabajando.

nombreProyecto - Nombre del proyecto sobre el que se está trabajando.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

abrirEditor

```
public void abrirEditor(org.eclipse.core.resources.IFile fichero, org.eclipse.ui.IEditorPart edPart)
```

Método invocado por el listener de la perspectiva de trabajo cuando un editor es abierto. Este método debe determinar si el recurso abierto está o no en coedición. De ser así debe sincronizar los contenidos y asociar los listeners adecuados. Así como registrar información de su apertura.

Parámetros:

fichero - IFile que representa el recurso que ha sido abierto.

edPart - Referencia al editor donde ha sido abierto el recurso.

borrarRecurso

```
public void borrarRecurso(java.lang.String id) throws RemoteException
```

Método que permite al usuario fuente avisar al usuario cliente cuando un recurso ha sido borrado de la estructura del proyecto.

Parámetros:

id - String que representa el identificador del recurso que ha sido borrado en el proyecto fuente.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

cargaInicialProyecto

```
private void cargaInicialProyecto()
```

Hace la carga inicial de la esructura del proyecto, para ello invocar‡ un mŽtodo de la fuente del proyecto que le suministrar‡ un array de objetos que contienen las referencias y contenidos iniciales de los recursos.

cargarEstadoInicial

```
public void cargarEstadoInicial() throws RemoteException
```

Método encargado de cargar el estado inicial en lo que refiere a los recursos que se están coeditando.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

cerrarEditor

```
public void cerrarEditor(org.eclipse.ui.IEditorPart[] editors)
```

Método invocado por el listener de la perspectiva cuando se han cerrado editores en el entorno de trabajo. Se encarga de determinar si se ha cerrado alguno de los editores de recursos en coedición para liberar sus recursos asociados de listeners.

Parámetros:

editors - Referencia a los editores que se encuentran activos actualmente en el sistema.

enviarModificacionCodigo

```
protected void enviarModificacionCodigo(IDocument document, int
offset, int length, String text)
```

Se encarga de enviar las modificaciones del código fuente al servidor del proyecto con el cual el sistema se encuentra conectado actualmente.

Parameters:

document - `IDocument` que hace referencia al documento que está siendo modificado.

offset - Desplazamiento de la modificación realizada sobre el documento.

length - Longitud del area modificada.

text - Texto que modificará el area establecida por la longitud y el offset.

estaCoeditandose

```
public boolean estaCoeditandose(java.lang.String id)
```

Permite determinar si un determinado recurso se está coeditando.

Parámetros:

id - String que representa el identificador único del recurso.

Retorno:

Verdadero si se está coeditando o falso en caso contrario.

finCoEdicion

```
public void finCoEdicion(java.lang.String id)
throws java.rmi.RemoteException
```

Informa al usuario cliente de que ha terminado la coedición del recurso cuyo identificador viene determinado por el parámetro id.

Parámetros:

id - String que representa el identificador del recurso que ha finalizado su coedición.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

inicioCoEdicion

```
public void inicioCoEdicion(RecursoCompartido recurso) throws
RemoteException
```

Informa al usuario cliente de que se ha iniciado la coedición del recurso pasado como parámetro. El cliente debe sincronizar los contenidos antes de empezar a trabajar sobre él.

Parámetros:

recurso - `RecursoCompartido` con información del recurso que se va a empezar a coeditar.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

modificarRecurso

```
public void modificarRecurso(RecursoCompartido recurso, int offset, int length, java.lang.String text) throws java.rmi.RemoteException
```

Método que permite a la fuente del proyecto enviar las modificaciones sobre los recursos al usuario cliente para que los almacene en su estructura de proyecto.

Parámetros:

recurso - Objeto que encapsula información sobre el recurso que se está modificando.

offset - Desplazamiento en el documento de la modificación.

length - Longitud del area modificada.

text - Texto a sustituir en el area modificada.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

nuevaAnotacion

```
public void nuevaAnotacion(org.eclipse.core.resources.IFile fichero,Anotacion anotacion) throws org.eclipse.core.runtime.CoreException, java.rmi.RemoteException
```

Método encargado de crear una nueva anotación y notificarlo al servicio de recursos de la fuente del proyecto.

Parámetros:

fichero - IFile que representa el recurso sobre el que hacemos la anotación.

anotacion - Objeto Anotación que almacena los detalles de la anotación.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada cuando hay problemas estableciendo la anotación al recurso.

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

obtenerRecursoCompartido

```
private ReferenciaRecursoCompartido obtenerRecursoCompartido( org.eclipse.jface.text.IDocument document)
```

Busca en la lista de documentos en proceso de coedición uno concreto a partir de la referencia al `IDocument` correspondiente.

Parámetros:

document - `IDocument` que sirve de referencia para buscar.

Retorno:

`ReferenciaRecursoCompartido` que apunta al recurso en coedición buscado.

obtenerRecursoCompartido

```
private ReferenciaRecursoCompartido obtenerRecursoCompartido(
String IDaBuscar)
```

Busca en la lista de documentos en proceso de coedición uno concreto a partir de la referencia al identificador correspondiente.

Parámetros:

IDaBuscar - Identificador del recurso que se est‡ buscando

Retorno:

`ReferenciaRecursoCompartido` que apunta al recurso en coedición buscado.

refrescarAnotaciones

```
public void refrescarAnotaciones(IFile fichero)
throws RemoteException, CoreException
```

Nos permite actualizar las anotaciones para un determinado recurso del sistema.

Parámetros:

fichero - IFile que representa el recurso sobre el que hacemos la actualización de anotaciones.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada cuando hay problemas estableciendo la anotación al recurso.

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

setAnotacion

```
private void setAnotacion(org.eclipse.core.resources.IFile
fichero, Anotacion anot) throws CoreException
```

Es el encargado de almacenar propiamente la anotación en el recurso.

Parámetros:

fichero - IFile que representa el recurso sobre el que hacemos la anotación.

anot - Objeto Anotacion que almacena los detalles de la anotación.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada cuando hay problemas estableciendo la anotación al recurso.

setAnotacion

```
public void setAnotacion(java.lang.String id, Anotacion
anotacion) throws RemoteException
```

Método que permite al usuario fuente mandar nuevas anotaciones para los recursos del proyecto del usuario cliente.

Parámetros:

id - String que representa el identificador del recurso que ha recibido una nueva anotación.

anotacion - Objeto `Anotacion` que encapsula la información de la nueva anotación.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

setServicioPareja

```
public void setServicioPareja( IServicioRecursosServidor recServidor)
```

Empareja el servidor de recursos locales con el servicio de recursos que la fuente del proyecto nos ofrece.

Parámetros:

recServidor - Referencia al objeto remoto del servidor para el control de recursos.

terminar

```
public void terminar()
```

Método invocado cuando se desea que el controlador de recursos deje de ofrecer su funcionalidad al sistema.

getIdCoeditados

```
public RecursoCompartido[] getIdCoeditados()
```

Obtiene las cadenas identificadores de los elementos en coedición.

Retorno:

String[] con los identificadores.

---

ecol.cliente.sesion

Class IControladorConexionCliente

Interfaz que sirve de medio de comunicación para que el usuario fuente se comunique con el usuario cliente en cuanto a tareas generales de conexión.

Detalles de Métodos

getServicioComunicaciones

```
public IServicioComunicaciones getServicioComunicaciones()
throws java.rmi.RemoteException
```

Retorna la interfaz de comunicaciones del cliente exportada al usuario fuente para mantener la comunicación.

Retorno:

referencia a la interfaz remota exportada para el control de comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

getServicioRecursos

```
public IServicioRecursosCliente getServicioRecursos()
throws java.rmi.RemoteException
```

Retorna la interfaz de recursos del cliente exportada al usuario fuente para el manejo de los mismos.

Retorno:

referencia a la interfaz remota exportada para el control de recursos.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

terminarSesion

```
public void terminarSesion() throws java.rmi.RemoteException
```

Método invocado por el usuario fuente del proyecto cuando se debe notificar que ha terminado la sesión de trabajo de colaboración.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

---

ecol.cliente.sesion

Class ControladorSesionCliente

Esta clase se encarga de controlar toda la sesión de trabajo en la parte del cliente.

Detalles de Métodos

ControladorSesionCliente

```
public ControladorSesionCliente(String name,
org.eclipse.ui.IWorkbenchWindow window) throws
java.rmi.RemoteException, java.rmi.NotBoundException,
ConexionParejaException, AutenticacioInvalidaException,
ParejaEstablecidaException
```

Constructor de un nuevo controlador de sesión.

Parameters:

name - Es el nombre del proyecto sobre el que se va a trabajar.

window - Ventana de trabajo desde la que se va a realizar el trabajo colaborativo.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

`java.rmi.NotBoundException` – Excepción lanzada cuando no se ha logrado obtener un objeto remoto con un nombre dado.

`ConexionParejaException` - Excepción lanzada cuando el servidor obtiene errores al contactar con la pareja recién conectada.

`AutenticacioInvalidaException` – Excepción lanzada cuando los datos de acceso no son válidos.

`ParejaEstablecidaException` - Excepción lanzada cuando ya existe una pareja conectada al proyecto fuente.

limpiarRecursos

```
public void limpiarRecursos()
```

Se encarga de liberar los recursos asociados a los servicios del usuario cliente, desactivar listeners, etc...

encapsularUsuario

```
private ParejaProgramacion encapsularUsuario(String proyecto)
```

Método encargado de obtener los detalles del usuario almacenados en el proyecto de trabajo.

Parámetros:

proyecto - Nombre del proyecto de donde se extraerá la información personal del usuario.

Retorno:

Devuelve un objeto `ParejaProgramacion` con la información personal encapsulada.

emparejarServicios

```
private void emparejarServicios()
throws java.rmi.RemoteException
```

Método encargado de asociar a cada uno de los controladores (sesión y comunicaciones) con su homólogo en la parte del servidor.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si existen errores en la comunicación RMI.

lanzarServicios

```
private void lanzarServicios() throws RemoteException
```

Método encargado de inicializar los controladores de recursos y comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si existen errores en la comunicación RMI.

getControladorComunicaciones

```
public ServicioComunicacionesImpl getControladorComunicaciones()
```

Obtiene el controlador de comunicaciones local del cliente.

Retorno:

`ServicioComunicacionesImpl` eferencia al controlador de comunicaciones del cliente.

getControladorRecursos

```
public ServicioRecursosClienteImpl getControladorRecursos()
```

Obtiene el controlador de recursos del cliente.

Retorno:

`ServicioRecursosClienteImpl` Referencia al controlador de recursos del cliente.

getNombreProyecto

```
public java.lang.String getNombreProyecto()
```

Obtiene el nombre del proyecto sobre el que se está trabajando.

Retorno:

String Nombre del proyecto sobre el que se trabaja.

conectarProyecto

```
private ParejaProgramacion conectarProyecto() throws
ConnectIOException, RemoteException, NotBoundException,
ConexionParejaException, AutenticacioInvalidaException,
ParejaEstablecidaException
```

Se encarga de negociar la conexión con el usuario fuente del proyecto.

Returns:

Un objeto `ParejaProgramacion` que representa al usuario remoto, con su información personal, o null si algo ha ido mal.

Throws:

`java.rmi.ConnectIOException` - Excepción lanzada si se produce un error de comunicación.

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

`java.rmi.NotBoundException` - Excepción lanzada cuando no se ha logrado obtener un objeto remoto con un nombre dado.

`ConexionParejaException` - Excepción lanzada cuando el servidor obtiene errores al contactar con la pareja recién conectada.

`AutenticacioInvalidaException` - Excepción lanzada cuando los datos de acceso no son válidos.

`ParejaEstablecidaException` - Excepción lanzada cuando ya existe una pareja conectada al proyecto fuente.

terminarSesion

```
public void terminarSesion() throws java.rmi.RemoteException
```

Método invocado por el usuario fuente del proyecto cuando se debe notificar que ha terminado la sesión de trabajo de colaboración.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

realizarDesconexion

```
public void realizarDesconexion()
throws java.rmi.RemoteException
```

Realiza la terminación de la sesión de trabajo con el servidor fuente del proyecto.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en un fallo RMI.

getInfoPersonal

```
public ParejaProgramacion getInfoPersonal()
```

Obtiene el objeto que representa la información personal del usuario local.

Retorno:

ParejaProgramacion Informacion del usuario local.

getInfoPareja

```
public ParejaProgramacion getInfoPareja()
```

Obtiene el objeto que representa la información personal del usuario remoto.

Retorno:

ParejaProgramacion Informacion del usuario remoto.

setDisplayUsuario

```
public void setDisplayUsuario(IDisplayInfoTrabajo view)
```

Establece el elemento que servirá de visor para la información de trabajo de la sesión colaborativa.

Parámetros:

view - Elemento que implemente la interfaz IDisplayInfoTrabajo. O null si no se desea ningún visor.

getDisplayUsuario

```
public IDisplayInfoTrabajo getDisplayUsuario()
```

Obtiene una referencia al elemento que sirve de visor para la información de la sesión de trabajo colaborativo.

Retorno:

IDisplayInfoTrabajo Referencia al elemento visor de información. Null si no existe.

getServicioComunicaciones

```
public IServicioComunicaciones getServicioComunicaciones()
throws java.rmi.RemoteException
```

Retorna la interfaz de comunicaciones del cliente exportada al usuario fuente para mantener la comunicación.

Retorno:

referencia a la interfaz remota exportada para el control de comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

getServicioRecursos

```
public IServicioRecursosCliente getServicioRecursos()
throws java.rmi.RemoteException
```

Retorna la interfaz de recursos del cliente exportada al usuario fuente para el manejo de los mismos.

Retorno:

referencia a la interfaz remota exportada para el control de recursos.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

abortar

```
public void abortar()
```

Método invocado cuando se va a acabar la sesión bruscamente.

---

ecol.cliente.vistas

Class ChatClienteView

Clase que se encarga de crear la vista para la comunicación mediante un chat textual con el otro usuario que participa en la sesión de comunicación.

Detalles de Métodos

createPartControl

```
public void createPartControl(org.eclipse.swt.widgets.Composite
parent)
```

Este método es llamado y nos permite crear la vista e inicializarla.

Parámetros:

parent - Composite sobre el que empezaremos a añadir elementos.

dispose

```
public void dispose()
```

Método invocado cuando se cierra la vista. Útil para notificar que ya no hay un visor de mensajes al entorno.

setFocus

```
public void setFocus()
```

Establece el foco en el control de la lista de mensajes.

nuevoMensaje

```
public void nuevoMensaje(MensajeChat mensaje)
```

Método invocado cuando llega un nuevo mensaje de chat al sistema.

Parámetros:

mensaje - Mensaje de chat encapsulado en un objeto `MensajeChat`.

mensajeSistema

```
public void mensajeSistema(java.lang.String mensaje)
```

Método invocado cuando llega un mensaje del sistema

Parámetros:

mensaje - String que representa el mensaje.

---

ecol.cliente.vistas

Class InformacionParejaView

Clase que implementa un visor de información de trabajo particular para el entorno de trabajo del usuario cliente.

Detalles de Métodos

InformacionParejaView

```
public InformacionParejaView()
```

createPartControl

```
public void createPartControl(org.eclipse.swt.widgets.Composite
parent)
```

Este método es llamado y nos permite crear la vista e inicializarla.

Parámetros:

parent - Composite sobre el que empezaremos a añadir elementos.

setFocus

```
public void setFocus()
```

Establece el foco en el control de la lista de mensajes.

dispose

```
public void dispose()
```

Método invocado cuando se cierra la vista. Útil para notificar que ya no hay un visor de información al entorno.

addCoEditado

```
public void addCoEditado(RecursoCompartido recurso)
```

Método invocado cuando se añade un nuevo recurso para coedición al sistema. El visor debe facilitar al usuario su control

Parámetros:

recurso - Referencia al recurso compartido que se empieza a coeditar.

eliminarCoEditado

```
public void eliminarCoEditado(java.lang.String id)
```

Método invocado cuando se termina de coeditar un recurso determinado.

Parámetros:

id - Identificador del recurso que ha dejado de ser coeditable.

parejaConectada

```
public void parejaConectada(ParejaProgramacion pareja)
```

Método invocado por el sistema cuando una nueva pareja se conecta a la sesión de trabajo.

Parámetros:

pareja - Objeto `ParejaProgramacion` que encapsula la información a mostrar.

parejaDesconectada

```
public void parejaDesconectada()
```

Método invocado cuando la pareja de programación se ha desconectado del entorno. Se debe notificar de ello al usuario.

---

ecol.cliente.wizards

Class DetallesProyectoPage

Clase que implementa la página del asistente para los detalles del proyecto cliente que se va ha crear.

Detalles de Métodos

DetallesProyectoPage

```
protected DetallesProyectoPage(java.lang.String pageName)
```

Constructor de la página del asistente encargada de tomar los detalles del proyecto creado.

Parámetros:

pageName - Nombre de la página

getUsuario

```
public java.lang.String getUsuario()
```

Devuelve el valor introducido por el usuario en la casilla de login de acceso.

Retorno:

`String` que representa el login introducido para acceder al proyecto.

getPasswdord

```
public java.lang.String getPasswdord()
```

Devuelve el valor introducido por el usuario en la casilla de contraseña de acceso.

Retorno:

`String` que representa el password introducido para acceder al proyecto.

getEmailPersona

```
public java.lang.String getEmailPersona()
```

Obtiene el e-mail personal introducido.

Retorno:

`String` con el e-mail personal.

getIp

```
public java.lang.String getIp()
```

Devuelve el valor introducido por el usario en la casilla de host.

Retorno:

String que representa el host introducido

getOtraInfoPersonal

```
public java.lang.String getOtraInfoPersonal()
```

Obtiene el campo de información adicional introducido.

Retorno:

`String` con el campo de información adicional

createControl

```
public void createControl(org.eclipse.swt.widgets.Composite
parent)
```

Método invocado para la creación visual de la página del asistente.

Parámetros:

parent - Composite donde iremos creando los elementos.

createPageContent

```
public void createPageContent(org.eclipse.swt.widgets.Composite
parent)
```

Crea el contenido de la página del asistente.

Parámetros:

parent - Composite donde crearemos los elementos del asistente.

setVisible

```
public void setVisible(boolean visible)
```

Método invocado cuando la página va a ser visualizada.

estaVacia

```
private boolean estaVacia(java.lang.String cadena)
```

Determina si una cadena está vacía.

Parámetros:

cadena - String a comprobar.

Retorno:

Verdadero si esta vacía y falso en el caso contrario.

isPageComplete

```
public boolean isPageComplete()
```

Determina si la página ha sido completada adecuadamente

---

ecol.cliente.wizards

Class NuevoProyectoWizard

Clase que representa el asistente para la creación de un nuevo proyecto cliente.

Detalles de Métodos

NuevoProyectoWizard

```
public NuevoProyectoWizard()
```

Constructor del asistente para la creación de un proyecto colaborativo como servidor.

canFinish

```
public boolean canFinish()
```

Determina si se puede activar la finalización el asistente de creación de proyecto.

Retorno:

Verdadero si se puede finalizar o falso en caso contrario.

addPages

public void addPages()

Añade páginas al asistente de creación de un nuevo proyecto.

performFinish

```
public boolean performFinish()
```

Se encarga de realizar las tareas del asistente una vez el usuario lo ha terminado.

Retorno:

Verdadero si ha terminado bien o falso en caso contrario.

crearCarpeta

```
private void crearCarpeta(org.eclipse.core.resources.IProject
project, java.lang.String nombre) throws
org.eclipse.core.runtime.CoreException
```

Método utilizado para crear una carpeta bajo el directorio raíz del proyecto.

Parámetros:

project - Referencia al IProject donde se requiere hacer al carpeta.

nombre - `String` nombre de la carpeta a crear.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si algo ha ido mal en la creación del recurso.

init

```
public void init(org.eclipse.ui.IWorkbench workbench,
org.eclipse.jface.viewers.IStructuredSelection selection)
```

Inicializa el asistente con el workbench sobre el que se trabaja as’ como los elementos seleccionados.

Parámetros:

workbench - Representa el objeto IWorkbench.

selection - Representa los elementos seleccionados.

setEntradasClasspath

```
private void setEntradasClasspath(IProject project, String
directorio) throws org.eclipse.jdt.core.JavaModelException
```

Sobreescribe el Classpath de un proyecto con una entrada del contenedor de la JRE, además de una entada de código fuente para el directorio pasado como parámetro.

Parámetros:

project - Referencia al IProject cuyo classpath será alterado.

directorio - Directorio a ser añadido al classpath.

Throws:

org.eclipse.jdt.core.JavaModelException - Excepción lanzada al establecer el classpath si algo fue mal.

---

ecol.comun

Class IDisplayInfoTrabajo

Interfaz que deben implementar todos los objetos que deseen ser usados como visor de información del sistema.

Detalles de Métodos

parejaConectada

```
void parejaConectada(ParejaProgramacion pareja)
```

Método invocado por el sistema cuando una nueva pareja se conecta a la sesi—n de trabajo.

Parámetros:

pareja - Objeto `ParejaProgramacion` que encapsula la información a mostrar.

parejaDesconectada

```
void parejaDesconectada()
```

MŽtodo invocado cuando la pareja de programaci—n se ha desconectado del entorno. Se debe notificar de ello al usuario.

addCoEditado

```
void addCoEditado(RecursoCompartido recurso)
```

Método invocado cuando se añade un nuevo recurso para coedición al sistema. El visor debe facilitar al usuario su control

Parámetros:

recurso - Referencia al recurso compartido que se empieza a coeditar.

eliminarCoEditado

```
void eliminarCoEditado(java.lang.String id)
```

Método invocado cuando se termina de coeditar un recurso determinado.

Parámetros:

id - Identificador del recurso que ha dejado de ser coeditable.

---

ecol.comun

Class Anotacion

Clase que encapsula la información de las anotaciones creadas por los usuarios.

Detalles de Métodos

Anotacion

```
public Anotacion(java.lang.String texto, int prioridad, boolean
hecho)
```

Crea una nueva anotación con todos sus campos.

Parámetros:

texto - String que representa el texto de la anotaci—n.

prioridad - Valor entero que representa su prioridad. Son constantes definidas en IMarker.PRIORITY_*

hecho – Determina si esta hecha o no la anotación.

isHecho

```
public boolean isHecho()
```

Retorna si la anotación está hecha o no.

Retorno:

Verdadero si está hecha, falso en caso contrario.

setHecho

```
public void setHecho(boolean hecho)
```

Establece si una anotación está hecha o no.

Parámetros:

hecho - Valor boolean que representa si está o no hecha la anotación.

getPrioridad

```
public int getPrioridad()
```

Retorna la prioridad de la anotación.

Retorno:

valor entero definido en IMarker.PRIORITY_*

setPrioridad

```
public void setPrioridad(int prioridad)
```

Establece la prioridad de la anotación.

Parámetros:

prioridad - Debe ser un entero valido definido en `IMarker.PRIORITY_*`

getTexto

```
public java.lang.String getTexto()
```

Retornar el texto de la anotación.

Retorno:

String que representa el texto de la anotación.

setTexto

```
public void setTexto(java.lang.String texto)
```

Establece el texto de la anotación

Parámetros:

texto - String que representa el nuevo texto de la anotación

---

ecol.comun

Class ParejaProgramacion

Clase que encapsula los datos personales y de sesión del usuario programador.

Detalles de Métodos

ParejaProgramacion

```
public ParejaProgramacion(java.lang.String login,
java.lang.String passwd, java.lang.String nombre,
java.lang.String email, java.lang.String otraInfo)
```

Construye un objeto con toda la información. La información personal es pasada como parámetro, mientras que el resto de información de la sesión se obtiene del entorno.

Parámetros:

login - String que representa el login (ó nombre corto) del usuario.

passwd - String que representa la clave del usuario (si procede).

nombre - String que representa el nombre completo del usuario.

email - String que representa el email del usuario.

otraInfo - String que representa información adicional del usuario.

getEmail

```
public java.lang.String getEmail()
```

Retorna el e-mail del usuario

Retorno:

String que representa el email del usuario.

getOtraInformacion

```
public java.lang.String getOtraInformacion()
```

Retorna información adicional del usuario

Retorno:

String que representa información adicional del usuario.

getLogin

```
public java.lang.String getLogin()
```

Retorna el login (o nombre corto) del usuario

Retorno:

String que representa el login (o nombre corto segœn proceda) del usuario.

getNombre

```
public java.lang.String getNombre()
```

Retorna el nombre completo del usuario

Retorno:

String que representa el nombre completo del usuario.

getOS

```
public java.lang.String getOS()
```

Retorna una cadena que representa la versión del sistema operativo

Returns:

String con la versión del sistema en el que se está trabajando.

getJDK

```
public java.lang.String getJDK()
```

Retorna una versión del vendedor de la JDK instalada y su versi—n.

Retorno:

String con el nombre del vendedor de la JDK instalada y su versi—n.

getPasswd

```
public java.lang.String getPasswd()
```

Retorna la clave establecida para el usuario que encapsula este objeto.

Retorno:

String con el password establecido.

toString

```
public java.lang.String toString()
```

Muestra una cadena resumen de la información personal del usuario.

Retorno:

String con los datos del usuario.

---

ecol.comun

Class RecursoCompartido

Clase que encapsula los datos básicos de un recurso compartido.

Detalles de Métodos

RecursoCompartido

```
public RecursoCompartido(java.lang.String ID, java.lang.String name, java.lang.String path)
```

Constructor de un recurso compartido.

Parámetros:

ID - Identificador del recurso compartido.

name - Nombre del recurso compartido.

path - Path relativo dentro del proyecto al recurso.

RecursoCompartido

```
public RecursoCompartido(java.lang.String ID, java.lang.String name, java.lang.String path, java.lang.String contenido)
```

Constructor de un recurso compartido con contenido inicial, esto es œtil para sincronizar contenidos.

Parámetros:

ID - Identificador del recurso compartido.

name - Nombre del recurso compartido.

path - Path relativo dentro del proyecto al recurso.

contenido - String con el contenido inicial del recurso.

getName

```
public java.lang.String getName()
```

Retorna el nombre del recurso.

Retorno:

String que representa el nombre del recurso.

getPath

```
public java.lang.String getPath()
```

Retorna el path del recurso.

Retorno:

String que representa el path del recurso.

getID

```
public java.lang.String getID()
```

Retorna el Identificador del recurso.

Retorno:

String que representa el identificador del recurso.

getContenido

```
public java.lang.String getContenido()
```

Retorna el contenido del recurso.

Retorno:

String que representa el contenido del recurso.

setAnotaciones

```
public void setAnotaciones(Anotacion[] anotaciones)
```

Establece las anotaciones asociadas al recurso en particular.

Parámetros:

anotaciones - Array de anotaciones asociadas al recurso.

getAnotaciones

```
public Anotacion[] getAnotaciones()
```

Retorna el conjunto de anotaciones asociadas al recurso.

Retorno:

Array de anotaciones asociadas al recurso.

---

ecol.comun

Class ReferenciaRecursoCompartido

Clase empleada para mantener un control de los recursos en coedición en la sesión de trabajo local de cada usuario.

Detalles de Métodos

ReferenciaRecursoCompartido

```
public ReferenciaRecursoCompartido(IFile fichero, IEditorPart
editor, IDocument doc, IDocumentListener listener)
```

Crea una nueva referencia a un recurso compartido en coedición.

Parámetros:

fichero - Referencia al recurso IFile asociado.

editor - Referencia al editor con el que se esta editando el recurso.

doc - Referencia al IDocument que representa la estructura del documento.

listener - REferencia al listener asociado al recurso en coedici—n.

getID

```
public java.lang.String getID(
```

Retorna el identificador del recurso compartido.

Retorno:

String que representa de manera única al recurso.

setEditor

```
public void setEditor(org.eclipse.ui.IEditorPart editor)
```

Establece la referencia al editor del recurso compartido.

Parámetros:

editor - Referencia al editor del documento o null si se quiere marcar que no est‡ siendo editado

getEditor

```
public org.eclipse.ui.IEditorPart getEditor()
```

Obtiene el editor del recurso representado por el objeto.

Returns:

Una referencia a IEditorPart, o null si no se est‡ editando.

setFile

```
public void setFile(org.eclipse.core.resources.IFile fichero)
```

Establece el IFile del recurso asociado al recurso compartido.

Parámetros:

fichero - Referencia a IFile.

getFile

```
public org.eclipse.core.resources.IFile getFile()
```

Obtiene el recurso IFile asociado al recurso compartido.

Retorno:

Referencia al IFile del recurso.

setListener

```
public void setListener(org.eclipse.jface.text.IDocumentListener
listener)
```

Establece el listener del código del documento.

Parámetros:

listener - Referencia al IDocumentListener del código del documento, o null se quiere asociar.

getListener

```
public org.eclipse.jface.text.IDocumentListener getListener()
```

Retorna el listener asociado al documento compartido.

Retorno:

Referencia al `IDocumentListener` o null si no tiene.

setDocument

```
public void setDocument(org.eclipse.jface.text.IDocument doc)
```

Establece el IDocument asociado al recurso cuando está siendo editado.

Parámetros:

doc - Referencia al `IDocument` asociado al recurso compartido o null si no está siendo editado.

getDocument

```
public org.eclipse.jface.text.IDocument getDocument()
```

Retorna la referencia al IDocument que está estructurando el recurso en coedición.

Retorno:

Referencia a `IDocument` o null si no está siendo editado.

getRecursoCompartidoInfo

```
public RecursoCompartido getRecursoCompartidoInfo()
```

Retorna la información básica del recurso compartido e intercambiable entre servidor y cliente.

Retorno:

referencia al objeto `RecursoCompartido` creado.

---

ecol.comun

Class UtilidadesProyecto

Esta clase contiene campos constantes utilizados en el funcionamiento interno del sistema, así como métodos estáticos para tareas habituales relacionadas con el manejo de proyectos.

Detalles de atributos

SCOPE_CLIENTE

```
public static final java.lang.String SCOPE_CLIENTE
```

Cadena que representa el scope del cliente para almacenar preferencias.

NATURE_SERVIDOR

```
public static final java.lang.String NATURE_SERVIDOR
```

Cadena que representa el identificador de la naturaleza de los proyectos servidor.

NATURE_CLIENTE

```
public static final java.lang.String NATURE_CLIENTE
```

Cadena que representa el identificador de la naturaleza de los proyectos cliente.

SCOPE_SERVIDOR

```
public static final java.lang.String SCOPE_SERVIDOR
```

Cadena que representa el scope del servidor para almacenar preferencias.

KEY_*

```
public static final java.lang.String KEY_*
```

Cadena que representan identificadores de valores de almacenamiento.

Detalles de Métodos

borrarFichero

```
public static boolean borrarFichero( String nombreProyecto,
String path, String nombre) throws
org.eclipse.core.runtime.CoreException
```

Borra un determinado fichero de la estructura del proyecto.

Parámetros:

nombreProyecto - Nombre del proyecto donde queremos borrar.

path - Ruta del recurso que deseamos borrar.

nombre - Nombre del recurso que se requiere borrar.

Retorno:

Verdadero o falso dependiendo si ha ido bien o no

Throws:

org.eclipse.core.runtime.CoreException - Excepción lanzada si ha habido algún error.

comprobarPath

```
private static void
comprobarPath(org.eclipse.core.resources.IProject proyecto,
String path) throws org.eclipse.core.runtime.CoreException
```

Se encarga de comprobar la validez de un path, y caso de no ser válido crearlo.

Parámetros:

proyecto - Proyecto donde queremos comprobar el path.

path - Path que deseamos comprobar.

Throws:

org.eclipse.core.runtime.CoreException - Excepción lanzada si hubo algœn problema al crear los segmentos del path.

crearFichero

```
public static void crearFichero(java.lang.String nproyecto,
java.lang.String nfichero) throws
org.eclipse.core.runtime.CoreException
```

Crea un fichero en un proyecto.

Parámetros:

nproyecto - Nombre del proyecto donde deseamos crearlo.

nfichero - Nombre completo del fichero.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si algo ha ido mal.

crearFichero

```
public static org.eclipse.core.resources.IFile
crearFichero(java.lang.String nproyecto, java.lang.String path,
java.lang.String nfichero, java.lang.String contenido) throws
org.eclipse.core.runtime.CoreException
```

Crea un fichero.

Parámetros:

nproyecto - Nombre del proyecto donde queremos crear el recurso.

path - Ruta del fichero que deseamos crear.

nfichero - Nombre del ficheor que queremos crar.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si hubo algún problema en al creación.

getPropiedadString

```
public static java.lang.String getPropiedadString(
org.eclipse.core.resources.IProject proyecto, java.lang.String
scope, java.lang.String key)
```

Obtiene una preferencia almacenada en un proyecto.

Parámetros:

proyecto - Referencia al proyecto donde queremos obtener la preferencia.

scope - Scope de almacenaje de la preferencia del proyecto.

key - Identificador válido de una propiedad. Están definidos en UtilidadesProyecto.KEY_*

Retorno:

String que representa la propiedad buscada o null si no exist'a o no estaba establecida.

getPropiedadString

```
public static String getPropiedadString(String nproyecto, String
scope, java.lang.String key)
```

Obtiene una preferencia almacenada en un proyecto.

Parámetros:

nproyecto - Nombre del proyecto donde queremos obtener la preferencia.

scope - Scope de almacenaje de la preferencia del proyecto.

key - Identificador válido de una propiedad. Están definidos en UtilidadesProyecto.KEY_*

Retorno:

String que representa la propiedad buscada o null si no existía o no estaba establecida.

getProject

```
public static org.eclipse.core.resources.IProject
getProject(java.lang.String nombre)
```

Retorna el IProject asociado a un nombre de proyecto.

Parámetros:

nombre - String que contiene el nombre de proyecto a localizar.

Retorno:

Referencia al IProject del proyecto buscado.

setEntradasClasspath

```
public static void setEntradasClasspath(IProject project) throws
org.eclipse.jdt.core.JavaModelException
```

Sobreescribe el Classpath de un proyecto con una entrada del contenedor de la JRE, además de una entada de código fuente para el directorio src del proyecto

Parámetros:

project - Proyecto que deseamos modificar su classpath.

Throws:

org.eclipse.jdt.core.JavaModelException - Excepción lanzada si huvo algún problema al establecer el classpath.

setNature

```
public static void setNature(org.eclipse.core.resources.IProject
proyecto, java.lang.String nature) throws
org.eclipse.core.runtime.CoreException
```

Establece la naturaleza en la descripci—n de un proyecto. No sobreescribe las naturalezas ya establecidas, simplemente se van agregando.

Parámetros:

proyecto - Referencia al IProject que deseamos modificar.

nature - Identificador de la naturaleza a agregar. Para los propios del entorno colaborativo están definidos en UtilidadesProyecto.NATURE_*

Throws:

org.eclipse.core.runtime.CoreException

setPropiedadString

```
public static void setPropiedadString(IProject proyecto,
String scope, String key, String value) throws
org.osgi.service.prefs.BackingStoreException
```

Establece una preferencia concreta en un proyecto según el mecanismo de Eclipse.

Parámetros:

proyecto - Referencia IProject del proyecto donde queremos establecer la preferencia.

scope - Scope de almacenaje de la preferencia del proyecto.

key - Identificador válido de una propiedad. Están definidos en UtilidadesProyecto.KEY_*

value - String que representa el valor a almacenar.

Throws:

`org.osgi.service.prefs.BackingStoreException` - Excepción lanzada si se produce algún error de almacenamiento.

vaciarCarpeta

```
public static void vaciarCarpeta(java.lang.String proyecto,
java.lang.String string)
```

Método encargado de vaciar una determinada carpeta de un proyecto dado.

Parámetros:

proyecto - Nombre del proyecto donde queremos vaciar la carpeta

string - Nombre de la carpeta a vaciar.

---

ecol.comun

Class UtilidadesSesion

Esta clase contiene campos constantes utilizados en el funcionamiento interno del sistema, así como métodos estáticos para tareas habituales relacionadas con la sesión de trabajo.

Detalles de atributos

INT_SERVIDOR

```
public static final java.lang.String INT_SERVIDOR
```

Nombre usado al registrar el interfaz de conexión de la fuente.

Detalles de Métodos

getProyectoSeleccionado

```
public static org.eclipse.core.resources.IProject
getProyectoSeleccionado(org.eclipse.jface.viewers.ISelection
seleccion)
```

Obtiene el IProject, si se trata de un recursos del espacio de trabajo, de la selección pasada como parámetro.

Parámetros:

seleccion - Rerencia a la selección de la que queremos extrae el IProject.

Retorno:

IProject asociado al recurso seleccionado, o null si no tiene IProject.

getRecursoSeleccionado

```
public static org.eclipse.core.resources.IResource
getRecursoSeleccionado(org.eclipse.jface.viewers.ISelection
seleccion)
```

Obtiene el IResource, si se trata de un recursos del espacio de trabajo, de la selección pasada como parámetro.

Parámetros:

seleccion - Rerencia a la selección de la que queremos extrae el IResource.

Retorno

IResource asociado al recurso que se extrae de la selección.

---

ecol.comun.comunicaciones

Class IDisplayMensajes

Interfaz que deben implementar todos los objetos que deseen ser usados como visor de mensajes.

Detalles de Métodos

nuevoMensaje

```
void nuevoMensaje(MensajeChat mensaje)
```

Método invocado cuando llega un nuevo mensaje de chat al sistema.

Parámetros:

mensaje - Mensaje de chat encapsulado en un objeto MensajeChat.

mensajeSistema

```
void mensajeSistema(java.lang.String mensaje)
```

Método invocado cuando llega un mensaje del sistema (Por ejemplo la conexión de un usuario.

Parámetros:

mensaje - String que representa el mensaje.

---

ecol.comun.comunicaciones

Class IServicioComunicaciones

Interfaz que será mediante la cual los usuarios se comuniquen entre si.

Detalles de Métodos

enviarMensaje

```
public void enviarMensaje(MensajeChat mensaje)
throws java.rmi.RemoteException
```

Método invocado por el usuario remoto para enviarnos un mensaje.

Parámetros:

mensaje - Objeto que encapsula el mensaje.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI.

---

ecol.comun.comunicaciones

Class MensajeChat

Clase que encapsula un mensaje de chat entre los usuarios.

Detalles de Métodos

MensajeChat

```
public MensajeChat(java.lang.String mensaje,
java.lang.String usuario)
```

Construye un nuevo mensaje de chat con un texto y un identificador del usuario que lo envía.

Parámetros:

mensaje – mensaje a enviar

usuario - usuario que lo envía

getMensaje

```
public java.lang.String getMensaje()
```

Retorna el texto del mensaje enviado.

Retorno:

String del mensaje encapsulado.

getUsuario

```
public java.lang.String getUsuario()
```

Retorna el usuario que envió el mensaje.

Retorno:

String que contiene el nombre del usuario.

toString

```
public java.lang.String toString()
```

Método que genera el mensaje de una manera legible y formateada.

Retorno:

String con el mensaje y el usuario.

---

ecol.comun.comunicaciones

Class ServicioComunicacionesImpl

Clase que implementa la interfaz de comunicaciones para dar soporte al sistema del intercambio de mensajes entre usuarios y la creación de un registro de los mismos.

Detalles de Métodos

ServicioComunicacionesImpl

```
public
ServicioComunicacionesImpl(org.eclipse.ui.IWorkbenchWindow
window, ParejaProgramacion personal,
java.lang.String nombreProyecto)throws java.rmi.RemoteException
```

Crea el controlador de comunicaciones del sistema.

Parámetros:

window - Ventana sobre la qeu trabajará el controlador de comunicaciones.

personal - Información del usuario local del sistema.

nombreProyecto - Nombre del proyecto sobre el que se está trabajando.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas RMI.

mensajeSistema

```
public void mensajeSistema(java.lang.String mensaje)
```

Método invocado cuando se quiere anunciar un mensaje importante para el sistema.

Parámetros:

mensaje - String que representa el mensaje del sistema.

log

```
public void log(java.lang.String cadena)
```

Se encarga de almacenar en el log los mensajes y dem‡s enventos del sistema.

Parámetros:

cadena - String que representa la cadena a almacenar.

setParejaServicio

```
public void setParejaServicio(IServicioComunicaciones isc)
```

Establece la pareja correspondiente a las comunicaciones en el lado del otro usuario.

Parámetros:

isc - Objeto que implementa la interfaz de comunicaciones.

terminar

```
public void terminar()
```

Método invocado cuando se desea liberar los recursos de comunicaci—n asociados.

enviarMensajeAPareja

```
public void enviarMensajeAPareja(java.lang.String texto)
```

Método invocado por el visor de mensajes para enviar un mensaje al otro miembro del trabajo colaborativo.

Parámetros:

texto - Texto a enviar.

setDisplayMensajes

```
public void setDisplayMensajes(IDisplayMensajes vista)
```

Establece el visor de mensajes del sistema.

Parámetros:

vista - Referencia al visor que implementa IDisplayMensajes

getDisplayMensajes

```
public IDisplayMensajes getDisplayMensajes()
```

Obtiene el objeto que está funcionando de visor de mensajes.

Retorno:

IDisplayMensajes o null si no hay ninguno establecido.

enviarMensaje

```
public void enviarMensaje(MensajeChat mensaje)
throws java.rmi.RemoteException
```

Implementación del método invocado por el usuario remoto para enviarnos un mensaje.

Parámetros:

mensaje - Objeto que encapsula el mensaje.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI.

---

ecol.comun.dialogos

Class NuevaAnotacionDialog

Clase que compone e inicializa el dialogo de usuario encargado de tomar los datos para las nuevas anotaciones del sistema.

Detalles de Métodos

NuevaAnotacionDialog

```
public NuevaAnotacionDialog(org.eclipse.swt.widgets.Shell
parentShell)
```

Crea un nuevo dialogo de crear anotación.

Parámetros:

parentShell - Shell padre del dialogo.

createDialogArea

```
protected org.eclipse.swt.widgets.Control
createDialogArea(Composite parent)
```

Crea la parte superior del cuadro de dialogo (por encima de la barra de botones).

Parámetros:

parent - Composite que representa el area donde se contendrá el cuadro de dialogo.

Retorno:

Referencia al controlador del area de dialogo.

configureShell

```
protected void configureShell(org.eclipse.swt.widgets.Shell newShell)
```

Configura el shell en el que se dispondrá el cuadro de dialogo. principalmente se inicializa el título y demás parámetros de aspecto.

Parámetros:

newShell - Shell en el que se dispondrá el cuadro de dialogo.

okPressed

```
protected void okPressed()
```

Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo creado.

getAnotacion

```
public Anotacion getAnotacion()
```

Retorna la anotación creada en el dialogo.

Retorno:

Anotación que encapsula la anotación creada por el usuario.

---

ecol.comun.excepciones

Class AutenticacionInvalidaException

Excepción lanzada por parte del usuario fuente al usuario cliente cuando los datos de validación del usuario no son correctos

---

ecol.comun.excepciones

Class ConexionParejaException

Excepción lanzada por parte del usuario fuente al usuario cliente cuando el servidor no puede establecer una conexión de vuelta con el cliente al iniciar la sesión

---

ecol.comun.excepciones

Class ParejaEstablecidaException

Excepción lanzada por parte del usuario fuente al usuario cliente cuando ya hay una pareja establecida en el sistema

ecol.comun.natures

Class ClienteEcolNature

Clase que implementa el comportamiento de la naturaleza de los proyectos colaborativos cliente.

Detalles de Métodos

configure

```
public void configure() throws
org.eclipse.core.runtime.CoreException
```

Configura esta naturaleza para su proyecto. Este método es llamado cuando se establece una naturaleza a un proyecto.

Throws:

`org.eclipse.core.runtime.CoreException`

deconfigure

```
public void deconfigure() throws
org.eclipse.core.runtime.CoreException
```

Elimina la configuración de esta naturaleza para el proyecto. Este método es llamado cuando se elimina una naturaleza a un proyecto.

Throws:

`org.eclipse.core.runtime.CoreException`

getProject

```
public org.eclipse.core.resources.IProject getProject()
```

Retorna el proyecto al cual esta naturaleza se aplica.

setProject

```
public void setProject(org.eclipse.core.resources.IProject
project)
```

Establece el proyecto en el cual se aplicará esta naturaleza.

Parámetros:

project - referencia al `IProject` sobre el que se aplica la naturaleza

---

ecol.comun.natures

Class ServidorEcolNature

Clase que implementa el comportamiento de la naturaleza de los proyectos colaborativos fuente.

Detalles de Métodos

configure

```
public void configure() throws
org.eclipse.core.runtime.CoreException
```

Configura esta naturaleza para su proyecto. Este método es llamado cuando se establece una naturaleza a un proyecto.

Throws:

`org.eclipse.core.runtime.CoreException`

deconfigure

```
public void deconfigure() throws
org.eclipse.core.runtime.CoreException
```

Elimina la configuración de esta naturaleza para el proyecto. Este método es llamado cuando se elimina una naturaleza a un proyecto.

Throws:

`org.eclipse.core.runtime.CoreException`

getProject

```
public org.eclipse.core.resources.IProject getProject()
```

Retorna el proyecto al cual esta naturaleza se aplica.

setProject

```
public void setProject(org.eclipse.core.resources.IProject
project)
```

Establece el proyecto en el cual se aplicará esta naturaleza.

Parámetros:

project - referencia al `IProject` sobre el que se aplica la naturaleza

---

ecol.servidor

Class EcolServidor

Esta clase sirve al plugin para controlar el modelo de trabajo, desde aquí se solicita el comienzo de una sesión de trabajo sobre un proyecto determinado y su terminación.

Detalles de Métodos

getVentanaTrabajo

```
public static org.eclipse.ui.IWorkbenchWindow
getVentanaTrabajo()
```

Obtiene la referencia a la ventana de trabajo

Retorno:

Devuelve una referencia a IWorkbenchWindow que reprsesenta la ventana de trabajo.

setWorkingProject

```
public static void setWorkingProject(java.lang.String name,
org.eclipse.ui.IWorkbenchWindow window) throws
java.rmi.RemoteException
```

Comienza una sesión de trabajo colaborativo.

Parámetros:

name - Nombre del proyecto sobre el que queremos trabajar.

window - Ventana de trabajo sobre la que estamos estableciendo la sesión

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas RMI

getControladorSesion

```
public static ControladorSesionServidor getControladorSesion()
```

Obtiene el controlador de sesión de trabajo actual.

Retorno:

Retorna el controlador de la sesión

estaIniciadaSesion

```
public static boolean estaIniciadaSesion()
```

Nos permite determinar si se ha iniciado sesión de trabajo con algún proyecto.

Returns:

Devuelve verdadero o falso dependiendo si está iniciado o no.

terminarSesion

```
public static void terminarSesion()
```

Método encargado de finalizar la sesión de trabajo actual.

abortarSesion

```
public static void abortarSesion(java.lang.String mensaje)
```

Método invocado cuando ha habido un error irrecuperable en la comunicación con el cliente. Útil ante fallos RMI.

Parámetros:

mensaje - Mensaje del error insalvable.

---

ecol.servidor.acciones

Class CambiarEmailAction

Clase asociada a la acción para cambiar el campo de e-mail en un proyecto Eclipse Colaborativo Fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.servidor.acciones

Class CambiarLoginAction

Clase asociada a la acción para cambiar el campo de login de acceso en un proyecto Eclipse Colaborativo Fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

ecol.servidor.acciones

Class CambiarNombreAction

Clase asociada a la acción para cambiar el campo de nombre personal en un proyecto Eclipse Colaborativo Fuente.

ecol.servidor.acciones

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

Class CambiarOtraInfoAction

Clase asociada a la acción para cambiar el campo de información adicional en un proyecto Eclipse Colaborativo Fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.servidor.acciones

Class CambiarPasswordAction

Clase asociada a la acción para cambiar el campo de password de acceso en un proyecto Eclipse Colaborativo Fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.servidor.acciones

Class LanzarProyectoAction

Implementación de la acción asociada a publicar un proyecto fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.servidor.acciones

Class NuevoAvisoProgAction

Clase encargada de la implementación de la acción de crear una nueva anotación en los recursos del proyecto fuente.

Detalles de Métodos

init

```
public void init(org.eclipse.ui.IWorkbenchWindow window)
```

Inicializa esta acción con la ventana del workbench en la que trabajará.

Parámetros:

window - IWorkbenchWindow

run

```
public void run(org.eclipse.jface.action.IAction action)
```

Realiza la acción. Este método se invoca cuando la acción es lanzada.

Parámetros:

action - IAction manejador encargado de la parte de presentación de la acción.

selectionChanged

```
public void selectionChanged(org.eclipse.jface.action.IAction
action, org.eclipse.jface.viewers.ISelection selection)
```

Notifica a esta acción que la selección del workbench ha cambiado. Esta notificación se puede emplear para cambiar la disponibilidad de la acción o para modificar las propiedades de presentación de la acción.

Parámetros:

action - IAction que representa la parte de la acción referente a su presentación.

selection - ISelection que representa la selección actual o null si no hay.

---

ecol.servidor.dialogos

Class SeleccionProyectoDialog

Clase que representa el cuadro de dialogo que se le muestra al usuario cuando selecciona el proyecto que desea para iniciar la sesión de trabajo de colaboración.

Detalles de Métodos

SeleccionProyectoDialog

```
public SeleccionProyectoDialog(org.eclipse.swt.widgets.Shell
parentShell, org.eclipse.ui.IWorkbenchWindow window)
```

Constructor para el cuadro de dialogo de selección de proyecto.

Parámetros:

parentShell - Shell padre en la que se dispondrá dialogo.

window - `IWorkbenchWindow` sobre la que se lanza el dialogo.

createDialogArea

```
protected org.eclipse.swt.widgets.Control
createDialogArea(Composite parent)
```

Crea la parte superior del cuadro de dialogo (por encima de la barra de botones).

Parámetros:

parent - Composite que representa el area donde se contendrá el cuadro de dialogo.

Retorno:

Referencia al controlador del area de dialogo.

configureShell

```
protected void configureShell(org.eclipse.swt.widgets.Shell
newShell)
```

Configura el shell en el que se dispondrá el cuadro de dialogo. principalmente se inicializa el título y demás parámetros de aspecto.

Parámetros:

newShell - Shell en el que se dispondrá el cuadro de dialogo.

okPressed

```
protected void okPressed()
```

Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo creado.

isSesionIniciada

```
public boolean isSesionIniciada()
```

Determina si se ha iniciado correctamente la sesión por parte del servidor.

Retorno:

Verdadero si se ha iniciado la sesión correctamente o falso en caso contrario.

---

ecol.servidor.perspectivas

Class TrabajoServidorPerspective

Clase encarga de componer la vista de trabajo del usuario fuente del proyecto.

Detalles de Métodos

createInitialLayout

```
public void createInitialLayout(org.eclipse.ui.IPageLayout
layout)
```

Crea la composición inicial de la perspectiva (vistas, acciones,...). Esta podrá ser modificada por el usuario desde las opciones del menú Window.

Parámetros:

layout - Representa la composición de la perspectiva.

---

ecol.servidor.recursos

Class IServicioRecursosServidor

Interfaz que el servidor exporta al usuario cliente para el manejo de los recursos.

Detalles de Métodos

getEstadoCoedicion

```
public RecursoCompartido[] getEstadoCoedicion() throws
java.rmi.RemoteException
```

Método invocado para conocer el estado de coedición de los recursos

Retorno:

Referencia de los recursos que están siendo coeditados

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

modificacionCodigo

```
public void modificacionCodigo(java.lang.String id, int offset, int length, java.lang.String codigo) throws java.rmi.RemoteException
```

Método invocado para notificar los cambios en los recursos del servidor

Parámetros:

id - Identificador del recurso que se va modificar

offset - Desplazamiento en el documento de la modificación.

length - Longitud del area modificada.

codigo - Texto a sustituir en el área modificada.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getEstructuraProyecto

```
public java.lang.Object[] getEstructuraProyecto()
throws java.rmi.RemoteException
```

Método por el cliente invocado para obtener la estructura del proyecto y su contenido

Retorno:

Array de objetos con los contenidos del proyecto.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getContenidoFichero

```
public java.lang.String getContenidoFichero(java.lang.String id) throws java.rmi.RemoteException
```

Obtiene el contenido de un determinado recurso.

Parámetros:

id - Identificador del recurso que queremos consultar

Retorno:

`String` con el contenido del recurso.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getAnotacionesFichero

```
public Anotacion[] getAnotacionesFichero(java.lang.String id)
```

`throws java.rmi.RemoteException`

Método invocado para obtener las anotaciones de un recurso

Parámetros:

id - Identificador del recurso

Retorno:

Array con las anotaciones encapsuladas en el objeto `Anotacion`.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

setAnotacion

```
public void setAnotacion(java.lang.String id, Anotacion anotacion) throws java.rmi.RemoteException
```

Establece una anotación en un recurso

Parámetros:

id - Identificador del recurso

anotacion - Anotacion para establecer en el recurso

Throws:

`java.rmi.RemoteException` - Exepción lanzada ante problemas de comunicación RMI

---

ecol.servidor.recursos

Class CodigoFuenteListener

Clase que implementa el `IDocumentListener`, encargada de mantener el control de las modificaciones de los documentos sobre los que trabaja.

Detalles de Métodos

CodigoFuenteListener

```
public CodigoFuenteListener(ControladorRecursos ctrl)
```

Crea un nuevo listener de código fuente para posteriormente ser asociado a un documento de trabajo.

Parámetros:

ctrl - `ControladorRecursos` que será el destino de las modificaciones recibidas por el listener.

documentChanged

```
public void documentChanged(org.eclipse.jface.text.DocumentEvent event)
```

Método invocado cuando un documento ha sido modificado. Se utilizará para mantener el control de las modificaciones en los documentos.

Parámetros:

event - `DocumentEvent` que encapsula la modificación realizada en el método.

---

ecol.servidor.recursos

Class ControladorRecursos

Esta clase se encarga de gestionar todo lo referido a los recursos del proyecto fuente de colaboración.

Detalles de Métodos

ControladorRecursos

```
public ControladorRecursos(IServicioRecursosCliente rec,
org.eclipse.ui.IWorkbenchWindow window ) throws
java.rmi.RemoteException
```

Crea el controlador de recursos para el proyecto fuente de la sesión de trabajo.

Parámetros

rec - Referencia a los recursos de la pareja cliente.

window - Ventana de trabajo de la sesión de trabajo.

Throws:

`java.rmi.RemoteException` - Se lanza cuando hay errores RMI

recursoBorrado

```
public void recursoBorrado(org.eclipse.core.resources.IFile
fichero)
```

Método invocado cuando un recurso es borrado.

Parámetros:

fichero - Referencia al fichero borrado.

getContenido

```
private java.lang.String getContenido(
org.eclipse.core.resources.IFile fichero)
```

Obtiene el contenido actualizado de un determinado recurso.

Parámetros:

fichero - Recurso del cual se quiere obtener el contenido actualizado.

Retorno:

`String` que contiene el código fuente del recurso.

getArchivoEncapsulado

```
private RecursoCompartido getArchivoEncapsulado(IFile fichero)
```

Obtiene el recurso pasado como un objeto comunicable al usuario cliente.

Parámetros:

fichero - Fichero que se desea encapsular

Retorno:

Contenido e información del recurso encapsulado.

getEstructuraProyecto

```
private Object[] getEstructuraProyecto(String nproyecto)
```

Recorre la estructura del proyecto y retorna todos los recursos de la misma.

Parámetros:

nproyecto - Nombre del proyecto a recorrer.

Retorno:

Array con todos los recursos encapsulados.

recorrerPaquete

```
private void recorrerPaquete(org.eclipse.core.resources.IFolder
paquete, java.util.LinkedList lista)
```

Recorre el contenido de un determinado paquete (carpeta).

Parámetros:

paquete - carpeta a recorrer del proyecto.

lista - lista donde ir almacenando los objetos encapsulados.

obtenerRecursoCompartido

```
private ReferenciaRecursoCompartido
obtenerRecursoCompartido(IDocument document)
```

Localiza la referencia a recurso compartido en coedición de la lista de coeditados.

Parámetros:

document - `IDocument` que se toma como criterio de búsqueda.

Retorno:

El objeto `ReferenciaRecursoCompartido` al `IDocument` buscado.

modificarRecurso

```
protected void modificarRecurso(org.eclipse.jface.text.IDocument
document, int offset, int length, java.lang.String text)
```

Método invocado cuando se detecta una modificación en un recurso por parte de su listener asociado.

Parámetros:

document - `IDocument` que referencia al documento modificado.

offset - Desplazamiento en el documento de la modificación.

length - Longitud del area modificada.

text - Texto a sustituir en el area modificada.

localizarRecurso

```
private ReferenciaRecursoCompartido localizarRecurso(
java.lang.String IDaBuscar)
```

Localiza la referencia a recurso compartido en coedición de la lista de coeditados.

Parámetros:

IDaBuscar - String que se toma como criterio de bœsqueda.

Retorno:

El objeto ReferenciaRecursoCompartido al String identificador buscado.

existeRecurso

```
private boolean existeRecurso(ReferenciaRecursoCompartido nuevo)
```

Determina si una referencia a un recurso en coedición existe ya o no, para ello tiene en cuenta el valor del método `getID()` del mismo.

Parámetros:

nuevo - Referencia sobre el cual se desea consultar.

Retorno:

Devuelve verdadero o falso dependiendo si existe o no.

estaEditorActivo

```
private boolean estaEditorActivo(org.eclipse.ui.IEditorPart
editor, org.eclipse.ui.IEditorPart[] parts)
```

Determina si un editor asociado a una referencia de un recurso compartido está entre los editores del entorno. Este método es útil para saber si un editor de un recurso se ha cerrado.

Parámetros:

editor - Editor que queremos comprobar si todavía está activo.

parts - conjunto de editores actuales.

Retorno:

Verdadero si está activo, falso en caso contrario.

iniciarEdicionColaborativa

```
public void iniciarEdicionColaborativa(IFile fichero,
org.eclipse.ui.IEditorPart edPart)
```

Método empleado para avisar a la pareja de programación de que se está empezando a coeditarse

Parámetros:

fichero - Recurso sobre el que se trabajará colaborativamente.

edPart - Editor que se a abierto para el recurso.

terminarEdicionColaborativa

```
public void terminarEdicionColaborativa( IEditorPart[] parts)
```

Este método se encarga de recorrer los editores activos en la ventana de trabajo pasados como parámetro y compararlos con la lista de archivos en co edición y así saber cual de ellos fue cerrado.

Parámetros:

parts - Conjunto de editores activos en el entorno.

terminar

```
public void terminar()
```

Se encarga de terminar y liberar los recursos y listeners asociados al controlador de recursos del proyecto fuente.

setParejaServicio

```
public void setParejaServicio(IServicioRecursosCliente irc)
```

Establece el servicio parejo de manejo de recursos perteneciente al cliente del proyecto fuente.

Parámetros:

irc - Interfaz de recursos del cliente del proyecto

getEstadoCoedicion

```
public RecursoCompartido[] getEstadoCoedicion() throws
java.rmi.RemoteException
```

Método invocado para conocer el estado de coedición de los recursos

Retorno:

Referencia de los recursos que están siendo coeditados

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

modificacionCodigo

```
public void modificacionCodigo(java.lang.String id, int offset,
int length, java.lang.String codigo) throws
java.rmi.RemoteException
```

Método invocado para notificar los cambios en los recursos del servidor

Parámetros:

id - Identificador del recurso que se va modificar

offset - Desplazamiento en el documento de la modificación.

length - Longitud del area modificada.

codigo - Texto a sustituir en el área modificada.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getEstructuraProyecto

```
public java.lang.Object[] getEstructuraProyecto()
throws java.rmi.RemoteException
```

Método por el cliente invocado para obtener la estructura del proyecto y su contenido

Retorno:

Array de objetos con los contenidos del proyecto.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getContenidoFichero

```
public java.lang.String getContenidoFichero(java.lang.String id)
throws java.rmi.RemoteException
```

Obtiene el contenido de un determinado recurso.

Parámetros:

id - Identificador del recurso que queremos consultar

Retorno:

`String` con el contenido del recurso.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getAnotaciones

```
private Anotacion[] getAnotaciones(IFile fichero)
```

Obtiene las anotaciones para un recurso determinado.

Parámetros:

fichero - Recurso a consultar.

Retorno:

Array de Anotaciones.

getAnotaciones

```
private Anotacion[] getAnotaciones(java.lang.String id)
```

Obtiene las anotaciones de un recurso

Parámetros:

id - Identificador del recurso

Retorno:

Array con las anotaciones encapsuladas en el objeto `Anotacion`.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

getAnotacionesFichero

```
public Anotacion[] getAnotacionesFichero(java.lang.String id)
throws java.rmi.RemoteException
```

Método invocado para obtener las anotaciones de un recurso

Parámetros:

id - Identificador del recurso

Retorno:

Array con las anotaciones encapsuladas en el objeto `Anotacion`.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas de comunicación RMI

asignarAnotacion

```
private void asignarAnotacion(org.eclipse.core.resources.IFile fichero, Anotacion anotacion) throws org.eclipse.core.runtime.CoreException
```

Se encarga de almacenar propiamente la anotación en el recurso.

Parámetros:

fichero - Fichero sobre el que almacenamos la anotación.

anotacion - Anotación que queremos almacenar en el recurso.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si hay algœn problema estableciendo la anotación.

nuevaAnotacion

```
public void nuevaAnotacion(org.eclipse.core.resources.IFile fichero, Anotacion resultado) throws org.eclipse.core.runtime.CoreException, java.rmi.RemoteException
```

Método que se encarga de crear una nueva anotación y notificarlo al cliente.

Parámetros:

fichero - Recursos sobre el que se ha creado la anotación

resultado - Anotación a almacenar

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si hay algun problema estableciendo la anotación

`java.rmi.RemoteException` - Excepción lanzada si hay problemas en la comunicación RMI

setAnotacion

```
public void setAnotacion(java.lang.String id, Anotacion anotacion) throws java.rmi.RemoteException
```

Establece una anotación en un recurso

Parámetros:

id - Identificador del recurso

anotacion - Anotacion para establecer en el recurso

Throws:

`java.rmi.RemoteException` - Exepción lanzada ante problemas de comunicación RMI

---

ecol.servidor.recursos

Class RecursosChangeListener

Esta clase se encarga de hacer de listener sobre los recursos del espacio de trabajo para mantener un controlo de los recursos borrados en el mismo.

Detalles de Métodos

RecursosChangeListener

```
public RecursosChangeListener(ControladorRecursos recursos)
```

Crea un listener de los cambios de los recursos.

Parámetros:

recursos - Controlador de recursos del servidor donde notificar.

resourceChanged

```
public void resourceChanged(IResourceChangeEvent event)
```

Método invocado cuando hay un cambio en los recursos.

Parámetros:

event – Evento que informa de la modificación

---

ecol.servidor.recursos

Class RecursosPerspectivaListener

Clase que se encarga de escuchar en la perspectiva que se asocia sobre nuevos editores abiertos o cerrados para mantener un control actualizado de los recursos sobre los que está trabajando el usuario fuente, y así saber cuando se empieza a coeditar un recurso o cuando se termina.

Detalles de Métodos

RecursosPerspectivaListener

```
public RecursosPerspectivaListener(ControladorRecursos
recursosManager)
```

Crea un nuevo listener para una perspectiva de trabajo de servidor fuente.

Parámetros:

recursosManager - Referencia al `ControladorRecursos` del proyecto fuente.

perspectiveActivated

```
public void perspectiveActivated(org.eclipse.ui.IWorkbenchPage
page, org.eclipse.ui.IPerspectiveDescriptor perspective)
```

Notifica al listener de que una perspectiva en la pagina ha sido activada.

Parámetros:

page - pagina que contiene la perspectiva activada.

perspective - el descriptor de la perspectiva que ha sido activada.

perspectiveChanged

```
public void perspectiveChanged(org.eclipse.ui.IWorkbenchPage
page,org.eclipse.ui.IPerspectiveDescriptor perspective,
java.lang.String changeId)
```

Notifica al listener de que una perspectiva ha cambiado de alguna manera. (un editor ha sido abierto, una vista, etc...)

Parámetros:

page - la página que contiene la perspectiva afectada.

perspective - el descriptor de la perspectiva.

changeId - constante CHANGE_* presente en `IWorkbenchPage`

---

ecol.servidor.sesion

Class IControladorConexionServidor

Interfaz que sirve de medio de comunicación para que el usuario cliente se comunique con el usuario fuente en cuanto a tareas generales de conexión.

Detalles de Métodos

conectarPareja

```
ParejaProgramacion conectarPareja(ParejaProgramacion pareja,
IControladorConexionCliente conexion) throws
java.rmi.RemoteException, ConexionParejaException,
AutenticacioInvalidaException, ParejaEstablecidaException
```

Método invocado por el cliente para iniciar una validación en la sesión de trabajo.

Parámetros:

pareja - Objeto que encapsula toda la información del usuario.

conexion - Referencia al controlador de conexión del usuario.

Retorno:

`ParejaProgramacion` con la información local del usuario fuente.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

`ConexionParejaException` - Excepción lanzada cuando el servidor obtiene errores al contactar con la pareja recién conectada.

`AutenticacioInvalidaException` - Excepción lanzada cuando los datos de acceso no son válidos.

`ParejaEstablecidaException` - Excepción lanzada cuando ya existe una pareja conectada al proyecto fuente.

desconectarPareja

```
void desconectarPareja() throws java.rmi.RemoteException
```

Método invocado por la pareja de programación para informar que ha terminado de trabajar en la sesión.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

getServicioRecursos

```
IServicioRecursosServidor getServicioRecursos()
throws java.rmi.RemoteException
```

Retorna la interfaz de recursos del usuario fuente exportada al usuario cliente para el manejo de los mismos.

Retorno:

Referencia a la interfaz remota exportada para el control de recursos.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

getServicioComunicaciones

```
IServicioComunicaciones getServicioComunicaciones()
throws java.rmi.RemoteException
```

Retorna la interfaz de comunicaciones del usuario fuente exportada al usuario cliente para mantener la comunicación.

Retorno:

Referencia a la interfaz remota exportada para el control de comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

---

ecol.servidor.sesion

Class ControladorSesionServidor

Esta clase se encarga de controlar toda la sesión de trabajo en la parte del servidor.

Detalles de Métodos

ControladorSesionServidor

```
public ControladorSesionServidor(java.lang.String name,
org.eclipse.ui.IWorkbenchWindow window,
java.rmi.registry.Registry registro) throws
java.rmi.RemoteException
```

Crea un nuevo controlador de sesión.

Parámetros:

name - Nombre del proyecto bajo el que estamos trabajando.

window - Ventana de trabajo del proyecto.

registro - Registroy RMI donde disponer las interfaces

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante problemas RMI.

getInfoPersonal

```
public ParejaProgramacion getInfoPersonal()
```

encapsularUsuario

```
private ParejaProgramacion encapsularUsuario(java.lang.String
proyecto)
```

Crea el objeto referente a la información personal del usuario local.

Parámetros:

proyecto - Proyecto sobre el que está almacenada la información.

Retorno:

El objeto `ParejaProgramacion` que encapsula la información.

informar

```
private void informar(java.lang.String mensaje)
```

Informa al sistema de comunicaciones con un mensaje del sistema.

Parámetros:

mensaje - `String` mensaje a notificar

iniciarServicios

```
private void iniciarServicios()
throws java.rmi.RemoteException
```

Inicia los servicios de conexión, recursos y comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada ante errores RMI.

getNProyecto

```
public java.lang.String getNProyecto()
```

Obtiene el nobmre del proyecto.

Retorno:

`String` que contiene el nombre del proyecto.

getControladorComunicaciones

```
public ServicioComunicacionesImpl getControladorComunicaciones()
```

Obtiene el controlador de las comunicaciones de la sesion de trabajo.

Retorno:

Objeto controlador de las comunicaciones.

getControladorRecursos

```
public ControladorRecursos getControladorRecursos()
```

Obtiene el controlador de recursos de la sesión de trabajo.

Retorno:

objeto controlador de recursos de la sesión.

autenticarUsuario

```
private boolean autenticarUsuario(ParejaProgramacion pareja2)
```

Realiza la validación del usuario.

Parámetros:

pareja2 - Objeto con toda la información del usuario que se va a validar.

Retorno:

Verdadero o falso dependiendo si se autenticó bien o mal.

conectarServicios

```
private void conectarServicios(ParejaProgramacion pareja)
throws java.rmi.RemoteException
```

Conecta los servicios con los de la pareja.

Parámetros:

pareja - objeto que representa a la pareja de programacion remota.

Throws:

`java.rmi.RemoteException` - Excepción lanzada cuando hay problemas con la comunicación RMI.

desconectarServicios

```
private void desconectarServicios()
```

Desconecta y libera los recursos creados para la sesión de trabajo colaborativo.

limpiarEnlaceServicios

```
private void limpiarEnlaceServicios()
```

Limpia la información referente al emparejamiento de recursos con el cliente.

setPareja

```
private void setPareja(ParejaProgramacion pareja,
IControladorConexionCliente cliente2)
throws ConexionParejaException
```

Establece la pareja de programación de la sesión.

Parámetros:

pareja - Objeto que encapsula a la pareja de programación.

cliente2 - Interfaz de conexión con el cliente.

Throws:

`ConexionParejaException` - Excepción lanzada si hay errores al comunicarse con el cliente.

terminarSesion

```
public void terminarSesion()
```

Se encarga de los procedimientos para terminar la sesión de trabajo, avisar al cliente y terminar recursos.

conectarPareja

```
public ParejaProgramacion conectarPareja(ParejaProgramacion
pareja2,
      IControladorConexionCliente cliente)
      throws java.rmi.RemoteException,
      ConexionParejaException,
      AutenticacioInvalidaException,
      ParejaEstablecidaException
```

Método invocado por el cliente para iniciar una validación en la sesión de trabajo.

Parámetros:

pareja2 - Objeto que encapsula toda la información del usuario.

cliente - Referencia al controlador de conexión del usuario.

Retorno:

`ParejaProgramacion` con la información local del usuario fuente.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

`ConexionParejaException` - Excepción lanzada cuando el servidor obtiene errores al contactar con la pareja recién conectada.

`AutenticacioInvalidaException` - Excepción lanzada cuando los datos de acceso no son vólidos.

`ParejaEstablecidaException` - Excepción lanzada cuando ya existe una pareja conectada al proyecto fuente.

desconectarPareja

```
public void desconectarPareja()
throws java.rmi.RemoteException
```

Método invocado por la pareja de programación para informar que ha terminado de trabajar en la sesión.

Throws:

`java.rmi.RemoteException` - Excepción lanzada en la comunicación RMI.

getPareja

```
public ParejaProgramacion getPareja()
```

Obtiene el objeto que representa la información personal del usuario remoto.

Retorno:

`ParejaProgramacion` Información del usuario remoto.

setDisplayUsuarios

```
public void setDisplayUsuarios(IDisplayInfoTrabajo display)
```

Establece el elemento que servirá de visor para la información de trabajo de la sesión colaborativa.

Parámetros:

display - Elemento que implemente la interfaz IDisplayInfoTrabajo. O null si no se desea ningœn visor.

getDisplayUsuario

```
public IDisplayInfoTrabajo getDisplayUsuario()
```

Retorna el elemento que sirve de visor de información de la sesión

Retorno:

Referencia al objeto que implementa `IDisplayInfoTrabajo` y sirve de visor.

getServicioComunicaciones

```
public IServicioComunicaciones getServicioComunicaciones()
throws java.rmi.RemoteException
```

Retorna la interfaz de comunicaciones del usuario fuente exportada al usuario cliente para mantener la comunicación.

Retorno:

Referencia a la interfaz remota exportada para el control de comunicaciones.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

getServicioRecursos

```
public IServicioRecursosServidor getServicioRecursos()
      throws java.rmi.RemoteException
```

Retorna la interfaz de recursos del usuario fuente exportada al usuario cliente para el manejo de los mismos.

Retorno:

Referencia a la interfaz remota exportada para el control de recursos.

Throws:

`java.rmi.RemoteException` - Excepción lanzada si hay algún error con la comunicación RMI.

abortarSesion

```
public void abortarSesion()
```

Termina todos los servicios y recursos asociados a la sesión de trabajo.

---

ecol.servidor.vistas

Class ChatServidorView

Clase que se encarga de crear la vista para la comunicación mediante un chat textual con el otro usuario que participa en la sesión de comunicación.

Detalles de Métodos

ChatServidorView

```
public ChatServidorView()
```

El constructor.

createPartControl

```
public void createPartControl(org.eclipse.swt.widgets.Composite
parent)
```

Este método es llamado y nos permite crear la vista e inicializarla.

Parámetros:

parent - Composite sobre el que empezaremos a añadir elementos.

dispose

```
public void dispose()
```

Método invocado cuando se cierra la vista. Útil para notificar que ya no hay un visor de mensajes al entorno.

setFocus

```
public void setFocus()
```

Establece el foco en el control de la lista de mensajes.

nuevoMensaje

```
public void nuevoMensaje(MensajeChat mensaje)
```

Método invocado cuando llega un nuevo mensaje de chat al sistema.

Parámetros:

mensaje - Mensaje de chat encapsulado en un objeto `MensajeChat`.

mensajeSistema

```
public void mensajeSistema(java.lang.String mensaje)
```

Método invocado cuando llega un mensaje del sistema (Por ejemplo la conexión de un usuario.

Parámetros:

mensaje - String que representa el mensaje.

---

ecol.servidor.vistas

Class InformacionParejaView

Clase que implementa un visor de información de trabajo particular para el entorno de trabajo del usuario fuente.

Detalles de Métodos

InformacionParejaView

```
public InformacionParejaView()
```

createPartControl

```
public void createPartControl(org.eclipse.swt.widgets.Composite
parent)
```

Este método es llamado y nos permite crear la vista e inicializarla.

Parámetros:

parent - Composite sobre el que empezaremos a añadir elementos.

setFocus

```
public void setFocus()
```

Establece el foco en el control de la lista de mensajes.

dispose

```
public void dispose()
```

Método invocado cuando se cierra la vista. Útil para notificar que ya no hay un visor de información al entorno.

addCoEditado

```
public void addCoEditado(RecursoCompartido recurso)
```

Método invocado cuando se añade un nuevo recurso para coedición al sistema. El visor debe facilitar al usuario su control

Parámetros:

recurso - Referencia al recurso compartido que se empieza a coeditar.

eliminarCoEditado

```
public void eliminarCoEditado(java.lang.String id)
```

Método invocado cuando se termina de coeditar un recurso determinado.

Parámetros:

id - Identificador del recurso que ha dejado de ser coeditable.

parejaConectada

```
public void parejaConectada(ParejaProgramacion pareja)
```

Método invocado por el sistema cuando una nueva pareja se conecta a la sesión de trabajo.

Parámetros:

pareja - Objeto `ParejaProgramacion` que encapsula la información a mostrar.

editarInfoPareja

```
private void editarInfoPareja(ParejaProgramacion pareja)
```

Edita la información relacionada con la pareja.

Parámetros:

pareja - Objeto que almacena la información de la pareja que se actualizará al sistema.

vaciarInfoPareja

```
private void vaciarInfoPareja()
```

Vacía la información de la pareja.

parejaDesconectada

```
public void parejaDesconectada()
```

Método invocado cuando la pareja de programación se ha desconectado del entorno. Se debe notificar de ello al usuario.

---

ecol.servidor.wizards

Class DetallesProyectoPage

Clase que implementa la página del asistente para los detalles del proyecto fuente que se va ha crear.

Detalles de Métodos

DetallesProyectoPage

```
protected DetallesProyectoPage(java.lang.String pageName)
```

Constructor de la página del asistente encargada de tomar los detalles del proyecto creado.

Parámetros:

pageName - Nombre de la página

getNombreCortoPersonal

```
public java.lang.String getNombreCortoPersonal()
```

Devuelve el valor introducido por el usuario en la casilla de nombre corto personal.

Retorno:

`String` que representa el nombre corto personal del usuario fuente.

getLoginPareja

```
public java.lang.String getLoginPareja()
```

Devuelve el valor introducido por el usuario en la casilla de login de acceso.

Retorno:

`String` que representa el login introducido para acceder al proyecto.

getPasswdPareja

```
public java.lang.String getPasswdPareja()
```

Devuelve el valor introducido por el usuario en la casilla de contraseña de acceso.

Retorno:

`String` que representa el password introducido para acceder al proyecto.

getNombrePersonal

```
public java.lang.String getNombrePersonal()
```

Obtiene el nombre introducido.

Retorno:

`String` con el nombre personal.

getEmailPersona

```
public java.lang.String getEmailPersona()
```

Obtiene el e-mail personal introducido.

Retorno:

`String` con el e-mail personal.

getOtraInfoPersonal

```
public java.lang.String getOtraInfoPersonal()
```

Obtiene el campo de información adicional introducido.

Retorno:

`String` con el campo de información adicional

createControl

```
public void createControl(org.eclipse.swt.widgets.Composite
parent)
```

Método invocado para la creación visual de la página del asistente.

Parámetros:

parent - Composite donde iremos creando los elementos.

createPageContent

```
public void createPageContent(org.eclipse.swt.widgets.Composite
parent)
```

Crea el contenido de la página del asistente.

Parámetros:

parent - Composite donde crearemos los elementos del asistente.

setVisible

```
public void setVisible(boolean visible)
```

Método invocado cuando la página va a ser visualizada.

estaVacia

```
private boolean estaVacia(java.lang.String cadena)
```

Determina si una cadena está vacía.

Parámetros:

cadena - String a comprobar.

Retorno:

Verdadero si esta vac'a y falso en el caso contrario.

isPageComplete

```
public boolean isPageComplete()
```

Determina si la página ha sido completada adecuadamente.

---

ecol.servidor.wizards

Class NuevoProyectoWizard

Clase que representa el asistente para la creación de un nuevo proyecto fuente. Esta clase se encarga de añadir las páginas al asistente, así como finalizar la tarea persistiendo los cambios.

Detalles de Métodos

NuevoProyectoWizard

```
public NuevoProyectoWizard()
```

Constructor del asistente para la creación de un proyecto colaborativo como servidor.

canFinish

```
public boolean canFinish()
```

Determina si se puede activar la finalización el asistente de creación de proyecto.

Retorno:

Verdadero si se puede finalizar o falso en caso contrario.

addPages

public void addPages()

Añade páginas al asistente de creación de un nuevo proyecto.

performFinish

```
public boolean performFinish()
```

Se encarga de realizar las tareas del asistente una vez el usuario lo ha terminado.

Retorno:

Verdadero si ha terminado bien o falso en caso contrario.

crearCarpeta

```
private void crearCarpeta(org.eclipse.core.resources.IProject
project, java.lang.String nombre) throws
org.eclipse.core.runtime.CoreException
```

Método utilizado para crear una carpeta bajo el directorio raíz del proyecto.

Parámetros:

project - Referencia al IProject donde se requiere hacer al carpeta.

nombre - `String` nombre de la carpeta a crear.

Throws:

`org.eclipse.core.runtime.CoreException` - Excepción lanzada si algo ha ido mal en la creación del recurso.

init

```
public void init(org.eclipse.ui.IWorkbench workbench,
org.eclipse.jface.viewers.IStructuredSelection selection)
```

Inicializa el asistente con el workbench sobre el que se trabaja as’ como los elementos seleccionados.

Parámetros:

workbench - Repersenta el objeto IWorkbench.

selection - Representa los elementos seleccionados.

APÉNDICE C. CÓDIGO FUENTE

Archivos de manifiesto del plugin:

MANIFEST.MF:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Plugin Eclipse Colaborativo
Bundle-SymbolicName: ecol;singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: ecol.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime,
 org.eclipse.core.resources,
 org.eclipse.ui.ide,
 org.eclipse.jdt.core,
 org.eclipse.jdt.ui,
 org.eclipse.ui.editors,
 org.eclipse.ui.workbench.texteditor,
 org.eclipse.jface.text
Eclipse-LazyStart: true
```

Plugin.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension
        point="org.eclipse.ui.views">
     <view
           category="ecol.vistas.VistasServidor"
           class="ecol.servidor.vistas.ChatServidorView"
           icon="icons/chat.png"
           id="ecol.vistas.servidor.ChatServidorView"
           name="Chat Servidor"/>
     <view
           category="ecol.vistas.VistasCliente"
           class="ecol.cliente.vistas.ChatClienteView"
           icon="icons/chat.png"
           id="ecol.vistas.cliente.ChatClienteView"
           name="Chat Cliente"/>
     <view
           category="ecol.vistas.VistasServidor"
           class="ecol.servidor.vistas.InformacionParejaView"
           icon="icons/date.png"
           id="ecol.vistas.servidor.InformacionParejaView"
           name="Informacion Servidor"/>
     <category
           id="ecol.vistas.VistasCliente"
           name="Vistas eCol como Cliente"
           />
     <category
           id="ecol.vistas.VistasServidor"
           name="Vistas eCol como Servidor"
           />
     <category
           id="ecol"
           name="ecol.vistas.VistasServidor"/>
     <view
           category="ecol"
           class="ecol.servidor.vistas.SampleView"
           icon="icons/sample.gif"
           id="ecol.servidor.vistas.SampleView"
           name="Sample View"/>
     <view
           category="ecol.vistas.VistasCliente"
           class="ecol.cliente.vistas.InformacionParejaView"
           icon="icons/date.png"
           id="ecol.vistas.cliente.InformacionParejaView"
```

```
        name="Informaci√≥n"/>
  </extension>
  <extension
      point="org.eclipse.ui.actionSets">
    <actionSet
        id="ecol.acciones"
        label="Acciones de Eclipse Colaborativo">
      <action
          class="ecol.cliente.acciones.ConectarProyectoAction"
          icon="icons/crow.gif"
          id="ecol.acciones.conectarProyectoCliente"
          label="Conectar Proyecto Colaborativo"
          menubarPath="ecol.menuPrincipal/seccionCliente"
          style="toggle"
          toolbarPath="seccionComun"/>
      <menu
          id="ecol.menuPrincipal"
          label="Eclipse Colaborativo">
        <separator name="seccionServidor"/>
        <separator name="seccionCliente"/>
        <separator name="seccionComun"/>
      </menu>
      <menu
          id="ecol.menuServidor"
          label="Propiedades Proyecto Servidor"
          path="ecol.menuPrincipal/seccionServidor">
        <separator name="seccionUnicaServidor"/>
        <separator name="seccionParticularServidor"/>
      </menu>
      <action
          class="ecol.servidor.acciones.LanzarProyectoAction"
          icon="icons/client.png"
          id="ecol.acciones.servidor.LanzarProyecto"
          label="Publicar Proyecto Colaborativo"
          menubarPath="ecol.menuPrincipal/seccionServidor"
          style="toggle"
          toolbarPath="seccionComun"/>
      <action
          class="ecol.servidor.acciones.NuevoAvisoProgAction"
          enablesFor="1"
          icon="icons/clip.png"
          id="ecol.acciones.servidor.AvisoProgramacion"
          label="Nueva Anotacion"

menubarPath="ecol.menuPrincipal/ecol.menuAnotacionesServidor/seccionAnotacionesServidor"
          style="push">
      </action>

                  <action
                      class="ecol.servidor.acciones.CambiarPasswordAction"
                      enablesFor="1"
                      icon="icons/password.png"
                      id="ecol.servidor.acciones.CambiarPasswordAction"
                      label="Modificar Password"
                      menubarPath="ecol.menuPrincipal/ecol.menuServidor/seccionParticularServidor"
                      style="push">
        <enablement>
          <objectState
              name="nature"
              value="ecol.comun.natures.servernature"/>
        </enablement>
      </action>
            <action
                class="ecol.servidor.acciones.CambiarLoginAction"
                enablesFor="1"
                icon="icons/login.png"
                id="ecol.servidor.acciones.CambiarLoginAction"
                label="Modificar Login"
                menubarPath="ecol.menuPrincipal/ecol.menuServidor/seccionParticularServidor"
                style="push">
        <enablement>
```

```
        <objectState
            name="nature"
            value="ecol.comun.natures.servernature"/>
      </enablement>
    </action>


    <action
        class="ecol.servidor.acciones.CambiarOtraInfoAction"
        enablesFor="1"
        icon="icons/other.png"
        id="ecol.acciones.servidor.CambiarOtraInfoAction"
        label="Modificar &apos;Otra Informaci√≥n&apos;"
        menubarPath="ecol.menuPrincipal/ecol.menuServidor/seccionUnicaServidor"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.servernature"/>
      </enablement>
    </action>
    <action
        class="ecol.servidor.acciones.CambiarEmailAction"
        enablesFor="1"
        icon="icons/arroba.png"
        id="ecol.acciones.servidor.CambiarEmailAction"
        label="Modificar Email"
        menubarPath="ecol.menuPrincipal/ecol.menuServidor/seccionUnicaServidor"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.servernature"/>
      </enablement>
    </action>
    <action
        class="ecol.servidor.acciones.CambiarNombreAction"
        enablesFor="1"
        icon="icons/nombre.png"
        id="ecol.acciones.servidor.CambiarNombreAction"
        label="Modificar Nombre"
        menubarPath="ecol.menuPrincipal/ecol.menuServidor/seccionUnicaServidor"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.servernature"/>
      </enablement>
    </action>
                <action
                    class="ecol.cliente.acciones.CambiarPasswordAction"
                    enablesFor="1"
                    icon="icons/password.png"
                    id="ecol.acciones.CambiarPasswordAction"
                    label="Modificar Password"
                    menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionInfoParticular"
                    style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
          <action
              class="ecol.cliente.acciones.CambiarLoginAction"
              enablesFor="1"
              icon="icons/login.png"
              id="ecol.acciones.CambiarLoginAction"
              label="Modificar Login"
              menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionInfoParticular"
              style="push">
```

```
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
    <action
        class="ecol.cliente.acciones.CambiarHostAction"
        enablesFor="1"
        icon="icons/host.png"
        id="ecol.acciones.CambiarHostAction"
        label="Modificar Host"
        menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionInfoParticular"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
    <action
        class="ecol.cliente.acciones.CambiarOtraInfoAction"
        enablesFor="1"
        icon="icons/other.png"
        id="ecol.acciones.CambiarOtraInfoAction"
        label="Modificar &amp;apos;Otra Informaci√≥n&amp;apos;"
        menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionUnicaCliente"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
    <action
        class="ecol.cliente.acciones.CambiarEmailAction"
        enablesFor="1"
        icon="icons/arroba.png"
        id="ecol.acciones.CambiarEmailAction"
        label="Modificar Email"
        menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionUnicaCliente"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
    <action
        class="ecol.cliente.acciones.CambiarNombreAction"
        enablesFor="1"
        icon="icons/nombre.png"
        id="ecol.acciones.CambiarNombreAction"
        label="Modificar Nombre"
        menubarPath="ecol.menuPrincipal/ecol.menuCliente/seccionUnicaCliente"
        style="push">
      <enablement>
        <objectState
            name="nature"
            value="ecol.comun.natures.clientnature"/>
      </enablement>
    </action>
    <menu
        id="ecol.menuCliente"
        label="Propiedades Proyecto Cliente"
        path="ecol.menuPrincipal/seccionCliente">
      <separator name="seccionUnicaCliente"/>
      <separator name="seccionInfoParticular"/>
    </menu>
    <menu
        id="ecol.menuAnotacionesCliente"
```

```
            label="Anotaciones"
            path="ecol.menuPrincipal/seccionCliente">
          <separator name="seccionAnotacionesCliente"/>
        </menu>
        <menu
            id="ecol.menuAnotacionesServidor"
            label="Anotaciones"
            path="ecol.menuPrincipal/seccionServidor">
          <separator name="seccionAnotacionesServidor"/>
        </menu>
        <action
            class="ecol.cliente.acciones.NuevoAvisoProgAction"
            enablesFor="1"
            icon="icons/clip.png"
            id="ecol.acciones.cliente.AvisoProgramacion"
            label="Nueva Anotacion"
            menubarPath="ecol.menuPrincipal/ecol.menuAnotacionesCliente/seccionAnotacionesCliente"
            style="push"/>
        <action
            class="ecol.cliente.acciones.RefrescarAnotacionesAction"
            enablesFor="1"
            icon="icons/refresh.png"
            id="ecol.acciones.cliente.RefrescarAvisos"
            label="Refrescar Anotaciones"
            menubarPath="ecol.menuPrincipal/ecol.menuAnotacionesCliente/seccionAnotacionesCliente"
            style="push"
            tooltip="Actualiza las anotaciones sobre el recurso solicitado"/>
      </actionSet>
    </extension>

    <extension
        id="ecol.comun.natures.clientnature"
        name="Ecol Client Nature"
        point="org.eclipse.core.resources.natures">
      <runtime>
        <run class="ecol.comun.natures.ClienteEcolNature"/>
      </runtime>
    </extension>
  <extension
        id="ecol.comun.natures.servernature"
        name="Ecol Server Nature"
        point="org.eclipse.core.resources.natures">
      <runtime>
        <run class="ecol.comun.natures.ServidorEcolNature"/>
      </runtime>
    </extension>
    <extension
        point="org.eclipse.ui.newWizards">
      <category
          id="ecol.comun.categorias.nuevoProyecto"
          name="Proyectos Eclipse Colaborativo"/>
      <wizard
          category="ecol.comun.categorias.nuevoProyecto"
          class="ecol.servidor.wizards.NuevoProyectoWizard"
          icon="icons/client.png"
          id="ecol.servidor.wizards.nuevoProyecto"
          name="Proyecto Colaborativo Fuente"
          project="true"/>
      <wizard
          category="ecol.comun.categorias.nuevoProyecto"
          class="ecol.cliente.wizards.NuevoProyectoWizard"
          icon="icons/crow.gif"
          id="ecol.cliente.wizards.nuevoProyecto"
          name="Proyecto Colaborativo Cliente"
          project="true"/>
    </extension>
    <extension
        point="org.eclipse.ui.perspectives">
      <perspective
          class="ecol.cliente.perspectivas.TrabajoClientePerspective"
          fixed="false"
```

```
        icon="icons/crow.gif"
        id="ecol.perspectivas.cliente"
        name="Cliente E-Col"/>
    <perspective
        class="ecol.servidor.perspectivas.TrabajoServidorPerspective"
        fixed="false"
        icon="icons/client.png"
        id="ecol.perspectivas.servidor"
        name="Servidor E-Col"/>
  </extension>
  <extension
      point="org.eclipse.ui.perspectiveExtensions">
    <perspectiveExtension targetID="ecol.perspectivas.servidor">
      <actionSet id="ecol.acciones"/>
      <newWizardShortcut id="ecol.servidor.wizards.nuevoProyecto"/>
      <newWizardShortcut id="ecol.cliente.wizards.nuevoProyecto"/>
    </perspectiveExtension>
    <perspectiveExtension targetID="ecol.perspectivas.cliente">
      <actionSet id="ecol.acciones"/>
      <newWizardShortcut id="ecol.servidor.wizards.nuevoProyecto"/>
      <newWizardShortcut id="ecol.cliente.wizards.nuevoProyecto"/>
    </perspectiveExtension>
    <perspectiveExtension targetID="org.eclipse.jdt.ui.JavaPerspective">
      <actionSet id="ecol.acciones"/>
      <newWizardShortcut id="ecol.servidor.wizards.nuevoProyecto"/>
      <newWizardShortcut id="ecol.cliente.wizards.nuevoProyecto"/>
    </perspectiveExtension>
  </extension>
  <extension
      point="org.eclipse.ui.decorators">
    <decorator
        adaptable="true"
        icon="icons/bluecuad.gif"
        id="ecol.decoradores.servidor.ProyectosServidor"
        label="DecoradorProyectos"
        lightweight="true"
        location="TOP_LEFT"
        state="true">
      <enablement>
        <and>
          <objectClass name="org.eclipse.core.resources.IProject"/>
          <objectState
              name="projectNature"
              value="ecol.comun.natures.servernature"/>
        </and>
      </enablement>
    </decorator>
    <decorator
        adaptable="true"
        icon="icons/edit.png"
        id="ecol.decoradores.cliente.ProyectosServidor"
        label="DecoradorProyectos"
        lightweight="true"
        location="TOP_LEFT"
        state="true">
      <enablement>
        <and>
          <objectClass name="org.eclipse.core.resources.IProject"/>
          <objectState
              name="nature"
              value="ecol.comun.natures.clientnature"/>
        </and>
      </enablement>
    </decorator>
  </extension>
  <extension
      id="ecol.marcador.AvisoProgramacion"
      name="Avisos E-Col"
      point="org.eclipse.core.resources.markers">
    <super type="org.eclipse.core.resources.taskmarker"/>
    <persistent value="true"/>
```

```
   </extension>
</plugin>
```

Paquete `ecol`**:**

Clase activadora del Plugin: `Activator.java`**:**

```
package ecol;

import org.eclipse.ui.plugin.AbstractUIPlugin;
import org.osgi.framework.BundleContext;

/**
* La clase Activator se encarga de controlar el ciclo de vida del plugin.
*/
public class Activator extends AbstractUIPlugin {

// The plug-in ID
public static final String PLUGIN_ID = "ecol";

// The shared instance
private static Activator plugin;

/**
 * Constructor del activador del plugin.
 */
public Activator() {
          plugin = this;
}

/*
 * (non-Javadoc)
 * @see org.eclipse.ui.plugin.AbstractUIPlugin#start(org.osgi.framework.BundleContext)
 */
public void start(BundleContext context) throws Exception {
          super.start(context);
}

/*
 * (non-Javadoc)
 * @see org.eclipse.ui.plugin.AbstractUIPlugin#stop(org.osgi.framework.BundleContext)
 */
public void stop(BundleContext context) throws Exception {
          plugin = null;
          super.stop(context);
}

/**
 * Devuelve la instancia compartida.
 *
 * @return la instancia compartida
 */
public static Activator getDefault() {
          return plugin;
}
}
```

Paquete `ecol.cliente`**:**

Clase `EcolCliente.java`:

```
package ecol.cliente;

 import java.rmi.ConnectIOException;
 import java.rmi.NotBoundException;
 import java.rmi.RemoteException;

 import org.eclipse.jface.dialogs.MessageDialog;
 import org.eclipse.ui.IWorkbenchWindow;
 import ecol.cliente.sesion.ControladorSesionCliente;
 import ecol.comun.excepciones.AutenticacioInvalidaException;
 import ecol.comun.excepciones.ConexionParejaException;
```

```
import ecol.comun.excepciones.ParejaEstablecidaException;

/**
 * Esta clase sirve al plugin para controlar el modelo de trabajo, desde aquí se
 * solicita el comienzo de una sesión de trabajo sobre un proyecto determinado y
 * su terminación.
 * @author Luis Fernández Álvarez
 */
public class EcolCliente {
	private static ControladorSesionCliente ctrlSesion;

	private static IWorkbenchWindow window;

	/**
	 * Nos permite determinar si se ha iniciado sesión de trabajo con algun
	 * proyecto.
	 *
	 * @return devuelve verdadero o falso dependiendo si está conectado o no.
	 */
	public static boolean estaConectadoProyecto() {
		if (ctrlSesion == null || ctrlSesion.getNombreProyecto() == null)
			return false;
		else
			return true;
	}

	/**
	 * Este método estático se encarga de iniciar la conexión a un proyecto fuente a partir
	 * de la información del proyecto pasado como parámetro y que se tomará como proyecto local.
	 * @param name Nombre del proyecto del espacio de trabajo sobre el que se trabajará.
	 * @param window Ventana que se tomará como la ventana de trabajo para el plugin.
	 * @return Devuelve verdadero o falso dependiendo si se realizó bien la conexión o no.
	 * @throws ConnectIOException Excepción lanzada ante un error de conexión con la fuente.
	 * @throws RemoteException Excepción lanzada en la comunicación RMI.
	 * @throws NotBoundException Excepción lanzada cuando no se ha logrado obtener un objeto
remoto con un nombre dado.
	 * @throws ConexionParejaException Excepción lanzada cuando el servidor obtiene errores al
contactar con la pareja recien conectada.
	 * @throws AutenticacioInvalidaException Excepción lanzada cuando los datos de acceso no son
válidos.
	 * @throws ParejaEstablecidaException Excepción lanzada cuando ya existe una pareja
conectada al proyecto fuente.
	 */
	public static boolean conectarProyecto(String name, IWorkbenchWindow window)
			throws ConnectIOException, RemoteException, NotBoundException,
			ConexionParejaException,
			AutenticacioInvalidaException, ParejaEstablecidaException {
		EcolCliente.window = window;

		ControladorSesionCliente ctrlTemp = new ControladorSesionCliente(name,
				window);
		ctrlSesion = ctrlTemp;

			ctrlSesion.getControladorRecursos().cargarEstadoInicial();

		return true;
	}

	/**
	 * Establece el controlador de la sesión de trabajo.
	 * @param ctrlSesion Nuevo controlador de la sesión.
	 */
	public static void setControladorSesion(ControladorSesionCliente ctrlSesion) {
		EcolCliente.ctrlSesion = ctrlSesion;
	}

	/**
	 * Obtiene el controlador de sesión de trabajo actual.
	 * @return Retorna el controlador de la sesión
	 */
	public static ControladorSesionCliente getControladorSesion() {
```

```
                return ctrlSesion;
        }

        /**
         * Método invocado cuando se solicita una desconexión por parte del cliente.
         * @throws RemoteException Excepción lanzada por un fallo en la comunicación RMI.
         */
        public static void terminarSesionPareja() throws RemoteException {
                ControladorSesionCliente aux = ctrlSesion;
                ctrlSesion = null;
                aux.realizarDesconexion();
        }

        /**
         * Obtiene la referencia a la ventana de trabajo
         * @return Devuelve una referencia a IWorkbenchWindow que reprsesenta la ventana de
trabajo.
         */
        public static IWorkbenchWindow getVentanaTrabajo() {
                return window;
        }

        /**
         * Método invocado para limpiar todos los recursos
         * de la sesión. Útil cuando se produce algún error de conexión y queremos
         * liberarlos
         *
         */
        public static void limpiarSesion() {
                if(ctrlSesion!=null)
                ctrlSesion.limpiarRecursos();
                ctrlSesion=null;
        }

        /**
         * Método invocado cuando ha habido un error importante en la comunicación
         * @param String razón por la cual se aborta.
         */
        public static void abortarSesion(String razon){
                MessageDialog.openError(window.getShell(), "Eclipse Colaborativo: Error Fatal", "Se va
a terminar la sesión bruscamente por problemas en la comunicación con el servidor:
\nRazón:\n"+razon);
                ctrlSesion.abortar();
                ctrlSesion=null;
                if (window.getActivePage().getPerspective().getId().compareTo(
                                "ecol.perspectivas.cliente") == 0)
                        window.getActivePage().closePerspective(
                                        window.getActivePage().getPerspective(), true,
                                        false);
        }
 }
```

Paquete ecol.cliente.acciones:

Clase `CambiarEmailAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
```

```
 * Clase asociada a la acción para cambiar el campo de email en un proyecto
 * Eclipse Colaborativo.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarEmailAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;

	private ISelection seleccion;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_CLIENTE,
				UtilidadesProyecto.KEY_EMAIL_USUARIO);
		InputDialog dialogo = new InputDialog(window.getShell(),
				"Cambiar informacion de proyecto cliente",
				"Introduzca la modificación deseada sobre el E-Mail del proyecto <"
						+ proyecto.getName() + ">", value, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_CLIENTE,
						UtilidadesProyecto.KEY_EMAIL_USUARIO, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"E-Col Cliente: Informe de Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
```

```
		 */
		public void selectionChanged(IAction action, ISelection selection) {
			seleccion = selection;
		}

}
```

Clase `CambiarHostAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;


/**
 * Clase asociada a la acción para cambiar el campo de host en un proyecto
 * Eclipse Colaborativo.
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarHostAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;

	private ISelection seleccion;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
    * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 * @param window IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 * @param action IAction manejador encargado de la parte de presentación de la  acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_CLIENTE,
				UtilidadesProyecto.KEY_HOST_FUENTE);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar informacion de proyecto cliente",
				"Introduzca la modificación deseada sobre el Host Fuente del
proyecto <"+proyecto.getName()+">",
				value, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_CLIENTE,
						UtilidadesProyecto.KEY_HOST_FUENTE, dialogo
```

```
                                                                .getValue());
                        } catch (BackingStoreException e) {
                                MessageDialog.openError(window.getShell(),
                                                "E-Col Cliente: Informe de Error",
                                                "Error almacenando los cambios en el proyecto:\n"
                                                                + e.getMessage());
                        }
                }
        }

        /**
         * Natifica a esta acción que la selección del workbench ha cambiado. Esta notificación
         * se puede emplear para cambiar la disponibilidad de la acción o para modificar
         * las propiedades de presentación de la acción.
         * @param action IAction que representa la parte de la acción referente a su presentación.
         * @param selection ISelection que representa la selección actual o null si no hay.
         */
        public void selectionChanged(IAction action, ISelection selection) {
                seleccion =  selection;
        }

}
```

Clase `CambiarLoginAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de login en un proyecto
 * Eclipse Colaborativo.
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarLoginAction implements IWorkbenchWindowActionDelegate {
        private IWorkbenchWindow window;

        private ISelection seleccion;

        /**
         * Dispose this action delegate
         */
        public void dispose() {

        }

        /**
    * Inicializa esta acción con la ventana del workbench en la que trabajará.
         * @param window IWorkbenchWindow
         */
        public void init(IWorkbenchWindow window) {
                this.window = window;
        }


        /**
         * Realiza la acción. Este método se invoca cuando la acción es lanzada.
         * @param action IAction manejador encargado de la parte de presentación de la  acción.
         */
        public void run(IAction action) {
```

```
                IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
                String value = UtilidadesProyecto.getPropiedadString(proyecto,
                                UtilidadesProyecto.SCOPE_CLIENTE,
                                UtilidadesProyecto.KEY_LOGIN_PAREJA);
                InputDialog dialogo = new InputDialog(
                                window.getShell(),
                                "Cambiar informacion de proyecto cliente",
                                "Introduzca la modificación deseada sobre el Login establecido en el
proyecto <"+proyecto.getName()+">",
                                value, null);
                int retorno = dialogo.open();
                if (retorno == Window.OK) {
                        try {
                                UtilidadesProyecto.setPropiedadString(proyecto,
                                                UtilidadesProyecto.SCOPE_CLIENTE,
                                                UtilidadesProyecto.KEY_LOGIN_PAREJA, dialogo
                                                                .getValue());
                        } catch (BackingStoreException e) {
                                MessageDialog.openError(window.getShell(),
                                                "E-Col Cliente: Informe de Error",
                                                "Error almacenando los cambios en el proyecto:\n"
                                                                + e.getMessage());
                        }
                }
        }

        /**
         * Natifica a esta acción que la selección del workbench ha cambiado. Esta notificación
         * se puede emplear para cambiar la disponibilidad de la acción o para modificar
         * las propiedades de presentación de la acción.
         * @param action IAction que representa la parte de la acción referente a su presentación.
         * @param selection ISelection que representa la selección actual o null si no hay.
         */
        public void selectionChanged(IAction action, ISelection selection) {
                seleccion =  selection;
        }
}
```

Clase `CambiarNombreAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de nombre en un proyecto
 * Eclipse Colaborativo.
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarNombreAction implements IWorkbenchWindowActionDelegate {
        private IWorkbenchWindow window;

        private ISelection seleccion;

        /**
         * Dispose this action delegate
         */
        public void dispose() {

        }
```

```
	/**
    * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 * @param window IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 * @param action IAction manejador encargado de la parte de presentación de la  acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String nombre = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_CLIENTE,
				UtilidadesProyecto.KEY_NOMBRE_USUARIO);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar nombre personal: ",
				"Introduzca el nombre personal que desea asociar al proyecto
seleccionado",
				nombre, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_CLIENTE,
						UtilidadesProyecto.KEY_NOMBRE_USUARIO, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"E-Col Cliente: Informe de Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta notificación
	 * se puede emplear para cambiar la disponibilidad de la acción o para modificar
	 * las propiedades de presentación de la acción.
	 * @param action IAction que representa la parte de la acción referente a su presentación.
	 * @param selection ISelection que representa la selección actual o null si no hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion =  selection;
	}

}
```

Clase `CambiarOtraInfoAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de otra información en un proyecto
```

```
 * Eclipse Colaborativo.
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarOtraInfoAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;

	private ISelection seleccion;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
    * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 * @param window IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 * @param action IAction manejador encargado de la parte de presentación de la  acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String nombre = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_CLIENTE,
				UtilidadesProyecto.KEY_OTRAINFO_USUARIO);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar propiedades proyecto ",
				"Introduzca la modificación deseada sobre el campo 'Otra
Información' del proyecto <"+proyecto.getName()+">",
				nombre, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_CLIENTE,
						UtilidadesProyecto.KEY_OTRAINFO_USUARIO,
dialogo
							.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"E-Col Cliente: Informe de Error",
						"Error almacenando los cambios en el proyecto:\n"
							+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta notificación
	 * se puede emplear para cambiar la disponibilidad de la acción o para modificar
	 * las propiedades de presentación de la acción.
	 * @param action IAction que representa la parte de la acción referente a su presentación.
	 * @param selection ISelection que representa la selección actual o null si no hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion =  selection;
	}

}
```

Clase `CambiarPasswordAction.java`:

```
package ecol.cliente.acciones;
```

```
import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de password en un proyecto
 * Eclipse Colaborativo.
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarPasswordAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;

	private ISelection seleccion;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
    * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 * @param window IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 * @param action IAction manejador encargado de la parte de presentación de la  acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_CLIENTE,
				UtilidadesProyecto.KEY_PASS_PAREJA);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar informacion de proyecto cliente",
				"Introduzca la nueva password asociada al proyecto
<"+proyecto.getName()+">",
				value, null);

		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_CLIENTE,
						UtilidadesProyecto.KEY_PASS_PAREJA, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"E-Col Cliente: Informe de Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
			}
		}
	}
```

```
	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta notificación
	 * se puede emplear para cambiar la disponibilidad de la acción o para modificar
	 * las propiedades de presentación de la acción.
	 * @param action IAction que representa la parte de la acción referente a su presentación.
	 * @param selection ISelection que representa la selección actual o null si no hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion =  selection;
	}

}
```

Clase `ConectarProyectoAction.java`:

```
package ecol.cliente.acciones;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import ecol.cliente.EcolCliente;
import ecol.cliente.dialogos.SeleccionProyectoDialog;

/**
 * Implementación de la acción asociada a conectar un proyecto cliente con el
 * proyecto fuente central.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ConectarProyectoAction implements IWorkbenchWindowActionDelegate {

	private IWorkbenchWindow window;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *		IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *		IAction manejador encargado de la parte de presentación de la
	 *		acción.
	 */
	public void run(IAction action) {
		if (!EcolCliente.estaConectadoProyecto()) {
			SeleccionProyectoDialog dialog = new SeleccionProyectoDialog(window
					.getShell(), window);
			dialog.open();
			if (dialog.isSesionIniciada()) {
				action.setChecked(true);
				window.getActivePage().setPerspective(
						window.getWorkbench().getPerspectiveRegistry()
								.findPerspectiveWithId(
```

```
		"ecol.perspectivas.cliente"));
				} else {
					action.setChecked(false);
				}
			} else {
				String botones[] = { "Si", "No" };
				MessageDialog dialogo = new MessageDialog(window.getShell(),
						"E-Col Cliente: Proyecto activo", null,
						"¿Desea terminar la sesión de trabajo colaborativo?",
MessageDialog.QUESTION,
						botones, 0);
				if (dialogo.open() == 0) {
					// Desconectamos todos los recursos.
					try {
						EcolCliente.terminarSesionPareja();
					} catch (Exception ex) {
						MessageDialog.openError(window.getShell(), "E-col Cliente:
Informe de Error",
								"La sesión ha finalizado pero con errores");
					}
					action.setChecked(false);
					if (window.getActivePage().getPerspective().getId().compareTo(
							"ecol.perspectivas.cliente") == 0)
						window.getActivePage().closePerspective(
								window.getActivePage().getPerspective(),
true,
								false);
				} else {
					action.setChecked(true);
				}
			}
		}

		/**
		 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
		 * notificación se puede emplear para cambiar la disponibilidad de la acción
		 * o para modificar las propiedades de presentación de la acción.
		 *
		 * @param action
		 *          IAction que representa la parte de la acción referente a su
		 *          presentación.
		 * @param selection
		 *          ISelection que representa la selección actual o null si no
		 *          hay.
		 */
		public void selectionChanged(IAction action, ISelection selection) {
			if (EcolCliente.estaConectadoProyecto())
				action.setChecked(true);
			else
				action.setChecked(false);
		}

}
```

Clase `NuevoAvisoProgAction.java`:

```
package ecol.cliente.acciones;

import java.rmi.RemoteException;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import ecol.cliente.EcolCliente;
```

```
import ecol.comun.Anotacion;
import ecol.comun.UtilidadesSesion;
import ecol.comun.dialogos.NuevaAnotacionDialog;

/**
 * Clase encargada de la implementación de la acción de crear una nueva
 * anotación en los recursos del proyecto cliente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class NuevoAvisoProgAction implements IWorkbenchWindowActionDelegate {
	private ISelection selection;

	private IWorkbenchWindow window;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		if (!EcolCliente.estaConectadoProyecto())
			return;

		IResource recurso = UtilidadesSesion.getRecursoSeleccionado(selection);
		if (recurso instanceof IFile) {
			IFile fichero = (IFile) recurso;
			// Tenemos el fichero, comprobamos que está en el proyecto con ele
			// que estamos trabajando.
			boolean esP = false;
			String nombreProyecto = recurso.getProject().getName();
			String enTrabajo = EcolCliente.getControladorSesion()
					.getNombreProyecto();
			if (nombreProyecto.compareTo(enTrabajo) == 0)
				esP = true;
			if (esP) {
				try {
					NuevaAnotacionDialog dialogo = new NuevaAnotacionDialog(
							window.getShell());
					dialogo.open();
					Anotacion resultado = dialogo.getAnotacion();
					if (resultado != null) {
						EcolCliente.getControladorSesion()

	.getControladorRecursos().nuevaAnotacion(
	                                                                fichero,
resultado);
					}
				} catch (CoreException e) {
					MessageDialog.openError(window.getShell(),
							"Error asignando anotación",
```

```
                                                        "Esta asignando el marcador al fichero: "
                                                                + e.getMessage());
                                } catch (RemoteException e) {
                                        //TODO NUEVO.
        EcolCliente.abortarSesion(                                                      "Esta
Anotacion puede no haber sido notificada a la fuente: "
                                        + e.getMessage());                      }
                                } else {
                                        MessageDialog
                                                        .openError(
                                                                window.getShell(),
                                                                "Error asignando anotación",
                                                                "Esta acción de antación sólo es
válida sobre el proyecto Eclipse Colaborativo (Cliente) sobre el que se está trabajando.");
                                }
                        }
                }

                /**
                 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
                 * notificación se puede emplear para cambiar la disponibilidad de la acción
                 * o para modificar las propiedades de presentación de la acción.
                 *
                 * @param action
                 *          IAction que representa la parte de la acción referente a su
                 *          presentación.
                 * @param selection
                 *          ISelection que representa la selección actual o null si no
                 *          hay.
                 */
                public void selectionChanged(IAction action, ISelection selection) {
                        this.selection = selection;
                        IResource recurso = UtilidadesSesion
                                        .getRecursoSeleccionado(this.selection);
                        if ((recurso != null) && (recurso instanceof IFile)
                                        && (EcolCliente.estaConectadoProyecto())) {
                                action.setEnabled(true);
                        } else {
                                action.setEnabled(false);
                        }
                }

}
```

Clase `RefrescarAnotacionesAction.java`:

```
package ecol.cliente.acciones;

import java.rmi.RemoteException;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import ecol.cliente.EcolCliente;
import ecol.comun.UtilidadesSesion;

/**
 * Esta clase representa la implementación de la acción de refrescar las
 * anotaciones de los recursos incluidos en el proyecto cliente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class RefrescarAnotacionesAction implements
                IWorkbenchWindowActionDelegate {
```

```
	private IWorkbenchWindow window;

	private ISelection selection;

	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		if (!EcolCliente.estaConectadoProyecto())
			return;
		IResource recurso = UtilidadesSesion.getRecursoSeleccionado(selection);
		if (recurso instanceof IFile) {
			IFile fichero = (IFile) recurso;
			// Tenemos el fichero, comprobamos que está en el proyecto con ele
			// que estamos trabajando.
			boolean esP = false;
			String nombreProyecto = recurso.getProject().getName();
			String enTrabajo = EcolCliente.getControladorSesion()
					.getNombreProyecto();
			if (nombreProyecto.compareTo(enTrabajo) == 0)
				esP = true;
			if (esP) {
				try {
					EcolCliente.getControladorSesion().getControladorRecursos()
							.refrescarAnotaciones(fichero);
				} catch (CoreException e) {
					MessageDialog.openError(window.getShell(),
							"Error refrescando anotación",
							"La actualización de anotaciones ha
generado errores: "
									+ e.getMessage());
				} catch (RemoteException e) {
					//TODO NUEVO.
						EcolCliente.abortarSesion(
						"La comunicación con el servidor ha generado
errores: "
							+ e.getMessage());
				}
			} else {
				MessageDialog
						.openError(
								window.getShell(),
								"Error refrescando anotación",
								"Esta acción sólo es válida sobre el
proyecto Eclipse Colaborativo (Cliente) sobre el que se está trabajando.");
			}
		}

	}
```

```
	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		this.selection = selection;
		IResource recurso = UtilidadesSesion
				.getRecursoSeleccionado(this.selection);
		if ((recurso != null) && (recurso instanceof IFile)
				&& (EcolCliente.estaConectadoProyecto())) {
			action.setEnabled(true);
		} else {
			action.setEnabled(false);
		}
	}

}
```

Paquete ecol.cliente.dialogos:

Clase `SeleccionProyectoDialog.java`:

```
package ecol.cliente.dialogos;

import java.net.ConnectException;
import java.rmi.AlreadyBoundException;
import java.rmi.ConnectIOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.LinkedList;

import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.TableItem;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PlatformUI;

import ecol.cliente.EcolCliente;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.excepciones.AutenticacioInvalidaException;
import ecol.comun.excepciones.ConexionParejaException;
import ecol.comun.excepciones.ParejaEstablecidaException;

/**
 * Clase que representa el cuadro de dialogo que se le muestra al usuario cuando
 * selecciona el proyecto que desea para iniciar la sesión de trabajo de
```

```
 * colaboración con la fuente de proyecto.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class SeleccionProyectoDialog extends Dialog {
	private TableViewer viewer;

	private boolean sesionIniciada;

	private IWorkbenchWindow window;

	/**
	 * Clase que se encarga de llevar el control del contenido de la tabla de
	 * proyectos disponibles en el entorno de trabajo.
	 *
	 * @author Luis Fernández Álvarez
	 *
	 */
	class ViewContentProvider implements IStructuredContentProvider {
		private LinkedList listaProyectos = new LinkedList();

		/**
		 * Este método es llamado por el visor de elementos cuando se le envía
		 * el mensaje dispose.
		 */
		public void dispose() {
		}

		/**
		 * Devuelve los elementos que serán visualizados en el visor asociado.
		 *
		 * @param parent
		 *          elemento que sirve de entrada.
		 * @return Object[] Array de elementos para ser visualizados.
		 */
		public Object[] getElements(Object parent) {
			IProject proyectos[] = ResourcesPlugin.getWorkspace().getRoot()
					.getProjects();

			for (int i = 0; i < proyectos.length; i++) {
				try {
					if (proyectos[i]

	.hasNature(UtilidadesProyecto.NATURE_CLIENTE)) {
						listaProyectos.add(proyectos[i].getName());
					}
				} catch (CoreException e) {

				}
			}
			return listaProyectos.toArray();
		}

		/**
		 * Notifica a este proveedor de contenido de que la entrada del visor ha
		 * sido cambiada a un elemento diferente.
		 *
		 * @param viewer
		 *          visor en el que está este proveedor de contenido.
		 * @param oldInput
		 *          Objeto que representa la entrada anterior.
		 * @param newInput
		 *          Objeto que representa la nueva entrada.
		 */
		public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {

		}
	}

	/**
```

```
 * Esta clase se encarga de proporcionar las etiquetas e identificadores
 * para los elementos de las tablas y listas de los visores de contenido.
 *
 * @author Luis Fernández Álvarez
 *
 */
class ViewLabelProvider extends LabelProvider implements
                ITableLabelProvider {

        /**
         * Retorna la etiqueta para la columna solicitada del elemento.
         *
         * @param obj
         *          es el objeto que representa la fila, o null.
         * @param index
         *          Índice (partiendo de 0) que indica la columna en la que la
         *          etiqueta aparecerá.
         * @return String o null si no hay etiqueta para la columna dada.
         */
        public String getColumnText(Object obj, int index) {
                return getText(obj);
        }

        /**
         * Retorna la imagen para la columna solicitada del elemento.
         *
         * @param obj
         *          es el objeto que representa la fila, o null.
         * @param index
         *          Índice (partiendo de 0) que indica la columna en la que la
         *          etiqueta aparecerá.
         * @return Image o null si no hay imagen para la columna dada.
         */
        public Image getColumnImage(Object obj, int index) {
                return PlatformUI.getWorkbench().getSharedImages().getImage(
                                ISharedImages.IMG_OBJ_PROJECT);
        }
}

/**
 * Constructor para el cuadro de dialogo de selección de proyecto.
 *
 * @param parentShell
 *          Shell padre en la que se dispondrá dialogo.
 * @param window
 *          IWorkbenchWindow sobre la que se lanza el dialogo.
 */
public SeleccionProyectoDialog(Shell parentShell, IWorkbenchWindow window) {
        super(parentShell);
        sesionIniciada = false;
        this.window = window;
}

/**
 * Crea la parte superior del cuadro de dialogo (por encima de la barra de
 * botones).
 *
 * @param parent
 *          Composite que representa el area donde se contendrá el cuadro
 *          de dialogo.
 * @return Referencia al controlador del area de dialogo.
 */
protected Control createDialogArea(Composite parent) {

        Composite composite = (Composite) super.createDialogArea(parent);
        GridLayout layout = new GridLayout(1, true);
        GridData data = new GridData();
        data.horizontalAlignment = GridData.FILL;
        data.grabExcessHorizontalSpace = true;
        composite.setLayout(layout);
        Label enunciado = new Label(composite, SWT.NONE);
```

```
                enunciado
                                .setText("A continuación se muestran los proyectos disponibles:");
                viewer = new TableViewer(composite, SWT.SINGLE | SWT.V_SCROLL);
                viewer.setLabelProvider(new ViewLabelProvider());
                viewer.setContentProvider(new ViewContentProvider());
                viewer.setInput(composite);
                viewer.getTable().setLayoutData(data);
                return composite;
        }

        /**
         * Configura el shell en el que se dispondrá el cuadro de dialogo.
         * principalmente se inicializa el título y demás parametros de aspecto.
         *
         * @param newShell
         *          Shell en el que se dispondrá el cuadro de dialogo.
         */
        protected void configureShell(Shell newShell) {
                super.configureShell(newShell);
                newShell.setText("Proyectos disponibles");
        }

        /**
         * Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo
         * creado.
         */
        protected void okPressed() {
                TableItem seleccionados[] = viewer.getTable().getSelection();
                if (seleccionados != null) {
                        try {
                                if (EcolCliente.conectarProyecto(seleccionados[0].getText(),
                                                window)) {
                                        sesionIniciada = true;
                                } else {
                                        // Devolvio false, mal mal.
                                        MessageDialog
                                                        .openError(window.getShell(),
                                                                        "E-Col: Informe de Error",
                                                                        "Error conectando,
compruebe su login o requisitos de la config de red.");
                                        EcolCliente.limpiarSesion();
                                        sesionIniciada=false;
                                }
                        } catch (ConnectIOException ex) {
                                MessageDialog.openError(window.getShell(), "E-Col: error",
                                                "El destino servidor no puede ser alcanzado");
                                EcolCliente.limpiarSesion();
                                sesionIniciada=false;

                        } catch (RemoteException e) {
                                if (e.getCause() instanceof ConnectException)
                                        MessageDialog
                                                        .openError(window.getShell(),
                                                                        "E-Col: Error de
conexión",
                                                                        "Conexión al host
rechazada, revise la configuración");
                                else
                                        MessageDialog.openError(window.getShell(),
                                                        "E-Col: Error remoto", "Error en la
conexión: "
                                                                        + e.getMessage());
                                sesionIniciada = false;
                                EcolCliente.limpiarSesion();
                        } catch (NotBoundException e) {
                                MessageDialog.openError(window.getShell(),
                                                "E-Col: Error de conexión",
                                                "No se ha encontrado las referencias en el registro
RMI: \n"
                                                                + e.getMessage());
                                sesionIniciada = false;
```

```
                                        EcolCliente.limpiarSesion();
                                } catch (ConexionParejaException e) {
                                        MessageDialog
                                                        .openError(
                                                                        window.getShell(),
                                                                        "E-Col: Error iniciando",
                                                                        "La fuente del proyecto ha tenido
el siguiente error conectando a este equipo, revise la configuración e intentelo de nuevo: "
                                                                                        +
e.getMessage());
                                        sesionIniciada = false;
                                        EcolCliente.limpiarSesion();
                                } catch (AutenticacioInvalidaException e) {
                                        MessageDialog
                                                        .openError(window.getShell(),
                                                                        "E-Col: Error conectando",
                                                                        "La información de autenticación
suministrada es inválida");
                                        sesionIniciada = false;
                                        EcolCliente.limpiarSesion();
                                } catch (ParejaEstablecidaException e) {
                                        MessageDialog
                                                        .openError(window.getShell(),
                                                                        "E-Col: Error conectando",
                                                                        "El proyecto al que intenta
conectarse ya tiene establecida una pareja");
                                        sesionIniciada = false;
                                        EcolCliente.limpiarSesion();
                                } catch (Exception ex){
                                        MessageDialog
                                        .openError(window.getShell(),
                                                        "E-Col: Error conectando",
                                                        "Error desconocido al conectar:
"+ex.getMessage());
                                        EcolCliente.limpiarSesion();
                                        sesionIniciada=false;

                                }
                        }
                        this.close();
                }

                /**
                 * Permite saber si se ha iniciado sesión o no al conectar el proyecto
                 * cliente al servidor.
                 *
                 * @return Verdadero si está iniciada la sesión o falso en caso contrario.
                 */
                public boolean isSesionIniciada() {
                        return sesionIniciada;
                }
        }
```

Paquete `ecol.cliente.perspectivas`:

Clase `TrabajoClientePerspective.java`:

```
package ecol.cliente.perspectivas;

import org.eclipse.jdt.ui.JavaUI;
import org.eclipse.ui.IFolderLayout;
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

/**
 * Clase encarga de componer la vista de trabajo del usuario cliente del
 * proyecto.
 *
 * @author Luis Fernández Álvarez
 *
 */
```

```
public class TrabajoClientePerspective implements IPerspectiveFactory {

	/**
	 * Crea la composición inicial de la perspectiva (vistas, acciones,...).
	 * Esta podrá ser modificada por el usuario desde las opciones del menú
	 * Window.
	 *
	 * @param layout
	 *          Representa la composición de la perspectiva.
	 */
	public void createInitialLayout(IPageLayout layout) {
		// Obtenemos el area ocupada por el editor, sobre
		// esa referencia es sobre la que trabajaremos.

		String editorArea = layout.getEditorArea();

		/*
		 * Parte derecha, Información personal y chat.
		 */
		IFolderLayout arribaDerecha = layout.createFolder("arribaDerecha",
				IPageLayout.RIGHT, 0.75f, editorArea);
		arribaDerecha.addView("ecol.vistas.cliente.InformacionParejaView");
		layout.addView("ecol.vistas.cliente.ChatClienteView",
				IPageLayout.BOTTOM, 0.7f,
				"ecol.vistas.cliente.InformacionParejaView");

		/*
		 * Parte izquierda, Paquete y Outline.
		 */

		IFolderLayout arribaIzquierda = layout.createFolder("arribaIzquierda",
				IPageLayout.LEFT, 0.35f, editorArea);

		arribaIzquierda.addView(JavaUI.ID_PACKAGES);
		layout.addView(IPageLayout.ID_OUTLINE, IPageLayout.BOTTOM, 0.4f,
				JavaUI.ID_PACKAGES);

		/*
		 * Parte abajo.
		 */
		layout.addView(IPageLayout.ID_TASK_LIST, IPageLayout.BOTTOM, 0.7f,
				editorArea);

	}
}
```

Paquete `ecol.cliente.recursos`:

Clase `CodigoFuenteListener.java`:

```
package ecol.cliente.recursos;

import org.eclipse.jface.text.DocumentEvent;
import org.eclipse.jface.text.IDocumentListener;
import ecol.cliente.EcolCliente;

/**
 * Clase que implementa el IDocumentListener, encargada de mantener el control
 * de las modificaciones de los documentos sobre los que trabaja.
 *
 * @author Luis Fernández Álavarez.
 *
 */
public class CodigoFuenteListener implements IDocumentListener {
	private ServicioRecursosClienteImpl recursos;

	/**
	 * Crea un nuevo listener de código fuente. Este listener se asocia con el
	 * controlador de recursos del cliente, destino al que enviará todas las
	 * modificaciones que reciba.
	 */
```

```
	public CodigoFuenteListener() {
		recursos = EcolCliente.getControladorSesion().getControladorRecursos();
	}

	/**
	 * The manipulation described by the document event will be performed.
	 *
	 * @param event
	 *		the document event describing the document change
	 */
	public void documentAboutToBeChanged(DocumentEvent event) {
	}

	/**
	 * Método invocado cuando un documento ha sido modificado. Se utilizará para
	 * mantener el control de las modificaciones en los documentos.
	 *
	 * @param event
	 *		DocumentEvent que encapsula la modificación realizada en el
	 *		método.
	 */
	public void documentChanged(DocumentEvent event) {
		recursos.enviarModificacionCodigo(event.getDocument(), event.fOffset,
				event.fLength, event.fText);
	}

}
```

Interfaz `IServicioRecursosCliente.java`:

```
package ecol.cliente.recursos;

import java.rmi.Remote;
import java.rmi.RemoteException;

import ecol.comun.Anotacion;
import ecol.comun.RecursoCompartido;

/**
 * Interfaz que será mediante la cual el usuario fuente se comunique con los
 * recursos del usuario cliente. Extiende java.rmi.Remote para permitir ser
 * implementada como objeto remoto.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IServicioRecursosCliente extends Remote {

	/**
	 * Método que permite a la fuente del proyecto enviar las modificaciones
	 * sobre los recursos al usuario cliente para que los almacene en su
	 * estructura de proyecto.
	 *
	 * @param compartido
	 *		Objeto que encapsula información sobre el recurso que se está
	 *		modificando.
	 * @param offset
	 *		Desplazamiento en el documento de la modificación.
	 * @param length
	 *		Longitud del area modificada.
	 * @param text
	 *		Texto a sustituir en el area modificada.
	 * @throws RemoteException
	 *		 Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.comun.RecursoCompartido
	 */
	public void modificarRecurso(RecursoCompartido compartido, int offset,
			int length, String text) throws RemoteException;

	/**
	 * Informa al usuario cliente de que ha terminado la coedición del recurso
```

```
         * cuyo identificador viene determinado por el parámetro id.
         *
         * @param id
         *          String que representa el identificador del recurso que ha
         *          finalizado su coedición.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         */
        public void finCoEdicion(String id) throws RemoteException;

        /**
         * Informa al usuario cliente de que se ha iniciado la coedición del recurso
         * pasado como parámetro. El cliente debe sincronizar los contenidos antes
         * de empezar a trabajar sobre él.
         *
         * @param recurso
         *          RecursoCompartido con información del recurso que se va a
         *          empezar a coeditar.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         */
        public void inicioCoEdicion(RecursoCompartido recurso)
                        throws RemoteException;

        /**
         * Método que permite al usuario fuente mandar nuevas anotaciones para los
         * recursos del proyecto del usuario cliente.
         *
         * @param id
         *          String que representa el identificador del recurso que ha
         *          recibido una nueva anotación.
         * @param anotacion
         *          Objeto Anotacion que encapsula la información de la nueva
         *          anotación.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         */
        public void setAnotacion(String id, Anotacion anotacion)
                        throws RemoteException;


        /**
         * Método que permite al usuario fuente avisar al usuario cliente cuando un
         * recurso ha sido borrado de la estructura del proyecto.
         *
         * @param id
         *          String que representa el identificador del recurso que ha sido
         *          borrado en el proyecto fuente.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         */
        public void borrarRecurso(String id) throws RemoteException;
}
```

Clase `RecursosPerspectivaListener.java`:

```
package ecol.cliente.recursos;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorPart;
import org.eclipse.ui.IFileEditorInput;
import org.eclipse.ui.IPerspectiveDescriptor;
import org.eclipse.ui.IPerspectiveListener;
import org.eclipse.ui.IWorkbenchPage;

import ecol.cliente.EcolCliente;

/**
 * Clase que se encarga de escuchar en la perspectiva que se asocia sobre nuevos
```

```
 * editores abiertos o cerrados para mantener un control actualizado de los recursos
 * sobre los que está trabajando el cliente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class RecursosPerspectivaListener implements IPerspectiveListener {

	/**
	 * Notifica al listener de que una perspectiva en la pagina ha sido
	 * activada.
	 *
	 * @param page
	 *          pagina que contiene la perspectiva activada.
	 * @param perspective
	 *          el descriptor de la perspectiva que ha sido activada.
	 * @see IWorkbenchPage#setPerspective
	 */
	public void perspectiveActivated(IWorkbenchPage page,
			IPerspectiveDescriptor perspective) {

	}

	/**
	 * Notifica al listener de que una perspectiva ha cambiado de alguna manera.
	 * (un editor ha sido abierto, una vista, etc...)
	 *
	 * @param page
	 *          la página que contiene la perspectiva a fectada.
	 * @param perspective
	 *          el descriptor de la perspectiva.
	 * @param changeId
	 *          constante CHANGE_* presente en IWorkbenchPage
	 */
	public void perspectiveChanged(IWorkbenchPage page,
			IPerspectiveDescriptor perspective, String changeId) {
		If(!EcolCliente.estaConectadoProyecto())return;
		if (changeId.compareTo(IWorkbenchPage.CHANGE_EDITOR_OPEN) == 0) {
			/*
			 * Un nuevo editor ha sido abierto, comprobemos que pertenece a
			 * nuestro proyecto y que nos interesa.
			 */
			IEditorPart edPart = page.getActiveEditor();
			IEditorInput input = edPart.getEditorInput();
			if (!(input instanceof IFileEditorInput)) {
				return;
			}
			IFile fichero = ((IFileEditorInput) input).getFile();
			IProject proyecto = fichero.getProject();

			if ((proyecto.getName().compareTo(
					EcolCliente.getControladorSesion().getNombreProyecto()) !=
0)
					|| !(fichero.getName().endsWith("java"))) {
				return;
			}
			EcolCliente.getControladorSesion().getControladorRecursos()
					.abrirEditor(fichero, edPart);
		}
		if (changeId.compareTo(IWorkbenchPage.CHANGE_EDITOR_CLOSE) == 0) {
			EcolCliente.getControladorSesion().getControladorRecursos()
					.cerrarEditor(

	EcolCliente.getVentanaTrabajo().getActivePage()
			.getEditors());
		}
	}

}
```

Clase `ServicioRecursosClienteImpl.java`:

```
package ecol.cliente.recursos;


import java.lang.reflect.InvocationTargetException;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.LinkedList;
import java.util.ListIterator;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.dialogs.ProgressMonitorDialog;
import org.eclipse.jface.operation.IRunnableWithProgress;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorPart;
import org.eclipse.ui.IFileEditorInput;
import org.eclipse.ui.IPerspectiveListener;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PartInitException;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.FileEditorInput;
import org.eclipse.ui.texteditor.AbstractTextEditor;
import org.eclipse.ui.texteditor.ITextEditor;


import ecol.cliente.EcolCliente;
import ecol.comun.Anotacion;
import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.RecursoCompartido;
import ecol.comun.ReferenciaRecursoCompartido;
import ecol.comun.UtilidadesProyecto;
import ecol.servidor.recursos.IServicioRecursosServidor;

/**
 * Esta clase se encarga de gestionar todo lo referido a los recursos del
 * proyecto cliente de colaboración. En esta clase se controla las
 * modificaciones en los recursos para notificarlas al servidor. Se controla las
 * actuaciones del usuario en cuanto a editores que ha abierto o cerrado y se
 * controla las notificaciones que el servidor nos hace para mantener los
 * recursos sincronizados.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ServicioRecursosClienteImpl implements IServicioRecursosCliente {

	private IWorkbenchWindow window;

	private LinkedList listaCoEditados;


	private IServicioRecursosServidor recPareja;

	private String nombreProyecto;

	private IPerspectiveListener listenerPerspectiva;


	/**
	 * Constructor del servicio de recursos del cliente. Se encarga de
	 * inicializar la estructura del proyecto cliente, añadir los listeners
	 * correspondientes y demás estructuras de datos.
	 *
```

```
	 * @param window
	 *          Referencia al IWorkbenchWindow sobre la que se está
	 *          trabajando.
	 * @param nombreProyecto
	 *          Nombre del proyecto sobre el que se está trabajando.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public ServicioRecursosClienteImpl(IWorkbenchWindow window,
			String nombreProyecto) throws RemoteException {
		this.nombreProyecto = nombreProyecto;


		UtilidadesProyecto.vaciarCarpeta(nombreProyecto, "src");
		UnicastRemoteObject.exportObject(this);
		this.window = window;
		listaCoEditados = new LinkedList();
		listenerPerspectiva = new RecursosPerspectivaListener();
		window.addPerspectiveListener(listenerPerspectiva);
	}


	/**
	 * Método invocado por el listener de la perspectiva de trabajo cuando un
	 * editor es abierto. Este método debe determinar si el recurso abierto está
	 * o no en coedición. De ser así debe sincronizar los contenidos y asociar
	 * los listenersr adecuados. Así como registrar información de su apertura.
	 *
	 * @param fichero
	 *          IFile que representa el recurso que ha sido abierto.
	 * @param edPart
	 *          Referencia al editor donde ha sido abierto el recurso.
	 */

	public void abrirEditor(IFile fichero, IEditorPart edPart) {
		String idFichero = fichero.getFullPath().removeFirstSegments(1)

				.toString();
		ReferenciaRecursoCompartido asociado = obtenerRecursoCompartido(idFichero);
		String cadena = null;

		try {
			cadena = recPareja.getContenidoFichero(idFichero);
		} catch (RemoteException e) {
			EcolCliente.abortarSesion("Error fatal obteniendo contenido de fichero en el
servidor."+e.getMessage());
		}

		if (asociado == null) {
			// Estamos abriendo un fichero en el que no se está coeditando.
			ITextEditor editor = (ITextEditor) edPart;
			IDocument doc = editor.getDocumentProvider().getDocument(

					editor.getEditorInput());
			if (cadena != null)
				doc.set(cadena);
		} else {
			ITextEditor editor = (ITextEditor) edPart;

			IDocument doc = editor.getDocumentProvider().getDocument(
					editor.getEditorInput());

			if (cadena != null)
				doc.set(cadena);

			CodigoFuenteListener tempListener = new CodigoFuenteListener();
			doc.addDocumentListener(tempListener);
			asociado.setDocument(doc);
			asociado.setEditor(edPart);
			asociado.setListener(tempListener);
		}
```

```
	}

	/**
	 * Método que permite al usuario fuente avisar al usuario cliente cuando un
	 * recurso ha sido borrado de la estructura del proyecto.
	 *
	 * @param id
	 *          String que representa el identificador del recurso que ha sido
	 *          borrado en el proyecto fuente.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void borrarRecurso(final String id) throws RemoteException {

		window.getShell().getDisplay().asyncExec(new Runnable() {


			public void run() {
				MessageDialog
						.openWarning(
								window.getShell(),
								"ECol Cliente. Informe Recurso
borrado",
								"El recurso: "
										+ id
										+ " ha sido
borrado en la fuente del proyecto. Guarde una copia si lo desea y bórrelo.");
			}
		});

	}

	/**
	 * Hace la carga inicial de la esructura del proyecto, para ello invocará un
	 * método de la fuente del proyecto que le suministrará un array de objetos
	 * que contienen las referencias y contenidos iniciales de los recursos.
	 */
	private void cargaInicialProyecto() {
		try {
			Object[] recursos = recPareja.getEstructuraProyecto();
			for (int i = 0; i < recursos.length; i++) {
				RecursoCompartido recurso = (RecursoCompartido) recursos[i];
				IFile creado = UtilidadesProyecto.crearFichero(nombreProyecto,
						recurso.getPath(), recurso.getName(), recurso
								.getContenido());
				Anotacion[] anotaciones = recurso.getAnotaciones();
				if (anotaciones != null) {
					for (int j = 0; j < anotaciones.length; j++) {
						IMarker marker = creado

	.createMarker("ecol.marcador.AvisoProgramacion");
						marker.setAttribute(IMarker.MESSAGE,
anotaciones[j]
								.getTexto());
						marker.setAttribute(IMarker.PRIORITY,
anotaciones[j]
								.getPrioridad());
						marker.setAttribute(IMarker.DONE, anotaciones[j]
								.isHecho());
					}
				}
			}
		} catch (RemoteException e) {
			EcolCliente.abortarSesion("Algún error cargando la estructura del proyecto:
"+e.getMessage());
		} catch (CoreException ex) {
			EcolCliente.abortarSesion("Algún error cargando los
	ficheros."+ex.getMessage());
			}
	}
```

```
	/**
	 * Método encargado de cargar el estado inicial en lo que refiere a los
	 * recursos que se están coeditando.
	 *
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void cargarEstadoInicial() throws RemoteException {

		RecursoCompartido[] recursos = recPareja.getEstadoCoedicion();
		if (recursos == null)
			return;
		for (int i = 0; i < recursos.length; i++) {
			inicioCoEdicion(recursos[i]);
		}

	}


	/**
	 * Método invocado por el listener de la perspectiva cuando se han cerrado
	 * editores en el entorno de trabajo. Se encarga de determinar si se ha
	 * cerrado alguno de los editores de recursos en coedición para liberar sus
	 * recursos asociados de listeners.
	 *
	 * @param editors
	 *          Referencia a los editores que se encuentran activos
	 *          actualmetne en el sistema.
	 */
	public void cerrarEditor(IEditorPart[] editors) {
		// Un editor ha sido cerrado.
		// Recorremos la lista de coeditados para ver si está entre los editores
		// activos
		ListIterator iterador = listaCoEditados.listIterator();
		while (iterador.hasNext()) {
			boolean encontrado = false;
			ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
					.next();
			for (int i = 0; i < editors.length; i++) {
				// Buscamos este editor en la lista de editores
				IEditorInput input = editors[i].getEditorInput();
				if (!(input instanceof IFileEditorInput)) {
					continue;
				}
				IFile fichero = ((IFileEditorInput) input).getFile();
				if (recurso.getID()
						.compareTo(

	fichero.getFullPath().removeFirstSegments(1)
	.toString()) ==
0) {
					encontrado = true;
				}
			}

			if (!encontrado) {
				// Lo encontramos, el que no esta siendo editado, hacemos las
				// cosas y terminamos.
				recurso.setDocument(null);
				recurso.setEditor(null);
				recurso.setListener(null);
				return;
			}
		}

	}

	/**
```

```
	 * Se encarga de enviar las modificaciones del código fuente al servidor del
	 * proyecto con el cual el sistema se encuentra conectado actualmente.
	 *
	 * @param document
	 *          IDocument que hace referencia al documento que está siendo
	 *          modificado.
	 * @param offset
	 *          Desplazamiento de la modificación realizada sobre el
	 *          documento.
	 * @param length
	 *          Longitud del area modificada.
	 * @param text
	 *          Texto que modificará el area establecida por la longitud y el
	 *          offset.
	 */
	protected void enviarModificacionCodigo(IDocument document, int offset,

				int length, String text) {

		ReferenciaRecursoCompartido recurso = obtenerRecursoCompartido(document);

		if (recurso == null) {
			return;
		}
		try {
			recPareja.modificacionCodigo(recurso.getID(), offset, length, text);
		} catch (RemoteException e) {
			//TODO Nuevo.
			EcolCliente.abortarSesion(
			"No se ha podido enviar la modificación de código al servidor. Revise el estado
de la sesión y reiniciela si es necesario.\nERROR: "
+ e.getMessage());
		}
	}

	/**
	 * Permite determinar si un determinado recurso se está coeditando.
	 *
	 * @param id
	 *          String que representa el identificador único del recurso.
	 * @return Verdadero si se está coeditando o falso en caso contrario.
	 */
	public boolean estaCoeditandose(String id) {
		if (obtenerRecursoCompartido(id) != null)
			return true;
		return false;
	}



	/**
	 * Informa al usuario cliente de que ha terminado la coedicón del recurso
	 * cuyo identificador viene determinado por el parámetro id.
	 *
	 * @param id
	 *          String que representa el identificador del recurso que ha
	 *          finalizado su coedición.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */

	public void finCoEdicion(String id) throws RemoteException {

		ListIterator iterador = listaCoEditados.listIterator();
		while (iterador.hasNext()) {
			ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
					.next();
			if (recurso.getID().compareTo(id) == 0) {
				// Hemos encontrado el que nos mandan borrar.
				// Le quitamos los listener.
```

```
                                IDocument docACerrar = recurso.getDocument();
                                // Puede ser null cuando no está abierto localmente.
                                if (docACerrar != null)
                                        recurso.getDocument().removeDocumentListener(
                                                        recurso.getListener());
                                // Lo eliminamos de la lista de coeditados.
                                iterador.remove();
                                IDisplayInfoTrabajo display = EcolCliente
                                                .getControladorSesion().getDisplayUsuario();
                                if (display != null) {
                                        display.eliminarCoEditado(recurso.getID());
                                }
                                // Terminamos, nos ahorramos el resto de editores.
                                return;
                        }
                }
        }

        /**
         * Informa al usuario cliente de que se ha iniciado la coedición del recurso
         * pasado como parámetro. El cliente debe sincronizar los contenidos antes
         * de empezar a trabajar sobre él.
         *
         * @param recurso
         *          RecursoCompartido con información del recurso que se va a
         *          empezar a coeditar.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         */
        public void inicioCoEdicion(RecursoCompartido recurso)
                        throws RemoteException {
                IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
                                .getProject(

        EcolCliente.getControladorSesion().getNombreProyecto());
                final IFile fichero = proyecto.getFolder(recurso.getPath()).getFile(
                                recurso.getName());
                if (!fichero.exists()) {
                        // Si no existìa el fichero, entonces lo creamos.
                        try {
                                UtilidadesProyecto.crearFichero(EcolCliente
                                                .getControladorSesion().getNombreProyecto(),
recurso
                                                .getPath(), recurso.getName(),
recurso.getContenido());
                        } catch (Exception e) {
                                MessageDialog
                                                .openError(
                                                                window.getShell(),
                                                                "ECol cliente: Informe de error",
                                                                "No se ha podido crear el recurso
nuevo recibido: "
                                                                                +
recurso.getName()
                                                                                + ". Revise la
configuración de permisos del proyecto y comuniquelo a la fuente de proyecto para reiniciar la
coedición.");
                        }
                }
                final String descrip = PlatformUI.getWorkbench().getEditorRegistry()
                                .getDefaultEditor(fichero.getName()).getId();

                PlatformUI.getWorkbench().getDisplay().asyncExec(new Runnable() {

                        public void run() {

                                try {
                                        window.removePerspectiveListener(listenerPerspectiva);
                                        IEditorPart edP = window.getActivePage().openEditor(
                                                        new FileEditorInput(fichero), descrip,
false);
```

```
                                        window.addPerspectiveListener(listenerPerspectiva);
                                        if (!(edP instanceof AbstractTextEditor))
                                                return;
                                        ITextEditor editor = (ITextEditor) edP;

                                        IDocument doc =
editor.getDocumentProvider().getDocument(
                                                        editor.getEditorInput());
                                        CodigoFuenteListener tempListener = new
CodigoFuenteListener();
                                        doc.addDocumentListener(tempListener);
                                        listaCoEditados.add(new ReferenciaRecursoCompartido(
                                                        fichero, edP, doc, tempListener));
                                } catch (PartInitException e) {
                                        MessageDialog.openConfirm(window.getShell(),
                                                        "Error creando editor", "UO:" +
e.getMessage());
                                }
                        }
                });
                IDisplayInfoTrabajo display = EcolCliente.getControladorSesion()
                                .getDisplayUsuario();
                if (display != null) {
                        display.addCoEditado(recurso);
                }

        }


        /**
         * Método que permite a la fuente del proyecto enviar las modificaciones
         * sobre los recursos al usuario cliente para que los almacene en su
         * estructura de proyecto.
         *
         * @param recurso
         *          Objeto que encapsula información sobre el recurso que se está
         *          modificando.
         * @param offset
         *          Desplazamiento en el documento de la modificación.
         * @param length
         *          Longitud del area modificada.
         * @param text
         *          Texto a sustituir en el area modificada.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         * @see ecol.comun.RecursoCompartido
         */
        public void modificarRecurso(RecursoCompartido recurso, final int offset,
                        final int length, final String text) throws RemoteException {
                // TODO HACER ESTO YA!!!
                final ReferenciaRecursoCompartido ref = obtenerRecursoCompartido(recurso
                                .getID());

                // Puede ser queno tengas la referencia (null)
                // O que la tengamosy tengamos el editor cerrado para ese documento
                // lo que implicarÌa lo segndo.
                if ((ref == null) || (ref.getDocument() == null))
                        return; // No queremos la modificacion, la descartamos.

                window.getWorkbench().getDisplay().asyncExec(new Runnable() {
                        public void run() {
                                try {

        ref.getDocument().removeDocumentListener(ref.getListener());
                                        ref.getDocument().replace(offset, length, text);
                                        ref.getEditor().doSave(null);
                                        ref.getDocument().addDocumentListener(ref.getListener());
                                } catch (BadLocationException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
```

```
                }
        });
}


/**
 * Método encargado de crear una nueva anotación y notificarlo al servicio
 * de recursos de la fuente del proyecto.
 *
 * @param fichero
 *          IFile que representa el recurso sobre el que hacemos la
 *          anotación.
 * @param anotacion
 *          Objeto Anotación que almacena los detalles de la anotación.
 * @throws CoreException
 *           Excepción lanzada cuando hay problemas estableciendo la
 *           anotación al recurso.
 * @throws RemoteException
 *           Excepción lanzada si hay algún error con la comunicación RMI.
 */
public void nuevaAnotacion(IFile fichero, Anotacion anotacion)
                throws CoreException, RemoteException {
        setAnotacion(fichero, anotacion);
        recPareja.setAnotacion(fichero.getFullPath().removeFirstSegments(1)
                        .toString(), anotacion);
}


/**
 * Busca en la lista de documentos en proceso de coedición uno concreto a
 * partir de la referencia al IDocument correspondiente.
 *
 * @param document
 *          IDocument que sirve de referencia para buscar.
 * @return ReferenciaRecursoCompartido que apunta al recurso en coedición
 *       buscado.
 */
private ReferenciaRecursoCompartido obtenerRecursoCompartido(
                IDocument document) {
        ListIterator iterador = listaCoEditados.listIterator();
        while (iterador.hasNext()) {
        ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido) iterador
                                .next();
                IDocument doc = recurso.getDocument();
                if (doc == null)
                        continue;
                if (doc.equals(document)) {
                        return recurso;
                }
        }
        return null;
}

/**
 * Busca en la lista de documentos en proceso de coedición uno concreto a
 * partir de la referencia al identificador correspondiente.
 *
 * @param IDaBuscar
 *          Identificador del recurso que se está buscando
 * @return ReferenciaRecursoCompartido que apunta al recurso en coedición
 *       buscado.
 */
private ReferenciaRecursoCompartido obtenerRecursoCompartido(
                String IDaBuscar) {
        ListIterator iter = listaCoEditados.listIterator();
        while (iter.hasNext()) {
                ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido) iter
                                .next();
                String idRecurso = recurso.getID();
                if (idRecurso.compareTo(IDaBuscar) == 0)
                        return recurso;
```

```
		}
		return null;
	}


	/**
	 * Nos permite actualizar las anotaciones para un determinado recurso del
	 * sistema.
	 *
	 * @param fichero
	 *          IFile que representa el recurso sobre el que hacemos la
	 *          actualización de anotaciones.
	 * @throws CoreException
	 *           Excepción lanzada cuando hay problemas estableciendo la
	 *           anotación al recurso.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void refrescarAnotaciones(IFile fichero) throws RemoteException,
			CoreException {
		Anotacion[] anotaciones = recPareja.getAnotacionesFichero(fichero
				.getFullPath().removeFirstSegments(1).toString());
		fichero.deleteMarkers("ecol.marcador.AvisoProgramacion", false,
				IResource.DEPTH_ZERO);
		if (anotaciones != null) {
			for (int i = 0; i < anotaciones.length; i++) {
				setAnotacion(fichero, anotaciones[i]);
			}

		}
	}



	/**

	 * Es el encargado de almacenar propiamente la anotación en el recurso.
	 *
	 * @param fichero
	 *          IFile que representa el recurso sobre el que hacemos la
	 *          anotación.
	 * @param anot
	 *          Objeto Anotación que almacena los detalles de la anotación.
	 * @throws CoreException
	 *           Excepción lanzada cuando hay problemas estableciendo la
	 *           anotación al recurso.
	 */
	private void setAnotacion(IFile fichero, Anotacion anot)
			throws CoreException {
		IMarker marker = fichero
				.createMarker("ecol.marcador.AvisoProgramacion");
		marker.setAttribute(IMarker.MESSAGE, anot.getTexto());
		marker.setAttribute(IMarker.PRIORITY, anot.getPrioridad());
		marker.setAttribute(IMarker.DONE, anot.isHecho());
	}


	/**
	 * Método que permite al usuario fuente mandar nuevas anotaciones para los
	 * recursos del proyecto del usuario cliente.
	 *
	 * @param id
	 *          String que representa el identificador del recurso que ha
	 *          recibido una nueva anotación.
	 * @param anotacion
	 *          Objeto Anotacion que encapsula la información de la nueva
	 *          anotación.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void setAnotacion(String id, Anotacion anotacion)
```

```
                throws RemoteException {

                try {
                        setAnotacion(UtilidadesProyecto.getProject(nombreProyecto).getFile(
                                        id), anotacion);
                } catch (CoreException e) {
                        throw new RemoteException("Error asignando la anotacion: "
                                        + e.getMessage());
                }
        }


        /**
         * Empareja el servidor de recursos locales con el servicio de recursos que
         * la fuente del proyecto nos ofrece.
         *
         * @param recServidor
         *          Referencia al objeto remoto del servidor para el control de
         *          recursos.
         */
        public void setServicioPareja(IServicioRecursosServidor recServidor) {

                recPareja = recServidor;
                ProgressMonitorDialog dialogo = new ProgressMonitorDialog(window
                                .getShell());
                try {
                        dialogo.run(false, false, new IRunnableWithProgress() {
                                public void run(IProgressMonitor monitor)
                                                throws InvocationTargetException,
InterruptedException {
                                        // TODO Auto-generated method stub
                                        monitor.beginTask("Cargando Proyecto Eclipse
Colaborativo",

                                                        IProgressMonitor.UNKNOWN);

                                        cargaInicialProyecto();
                                        monitor.done();
                                }


                        });
                } catch (InvocationTargetException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }


        /**
         * Método invocado cuando se desea que el controlador de recursos deje de
         * ofrecer su funcionalidad al sistema.
         *
         */
        public void terminar() {
                window.removePerspectiveListener(listenerPerspectiva);
                ListIterator iter=listaCoEditados.listIterator();
                //Quitamos los listener de los documentos abiertos.
        while(iter.hasNext()){
                        ReferenciaRecursoCompartido
recurso=(ReferenciaRecursoCompartido)iter.next();
                        recurso.getDocument().removeDocumentListener(recurso.getListener());
        }
        /**
         * Obtiene las cadenas identificadoras de los elementos en coedición.
         * @return String[] con los identificadores.
         */
```

```
	public RecursoCompartido[] getIdCoeditados(){
		if(listaCoEditados.isEmpty())return null;
		RecursoCompartido[] array=new RecursoCompartido[listaCoEditados.size()];
		ListIterator iterador=listaCoEditados.listIterator();
		int i=0;
		while(iterador.hasNext()){
			ReferenciaRecursoCompartido
ref=(ReferenciaRecursoCompartido)iterador.next();
			array[i]=ref.getRecursoCompartidoInfo();
				i++;
		}
		return array;
	}
}
```

Paquete `ecol.cliente.sesion`:

Clase `ControladorSesionCliente.java`:

```
package ecol.cliente.sesion;


import java.rmi.ConnectIOException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.ui.IWorkbenchWindow;


import ecol.cliente.recursos.IServicioRecursosCliente;
import ecol.cliente.recursos.ServicioRecursosClienteImpl;
import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.ParejaProgramacion;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;
import ecol.comun.comunicaciones.IServicioComunicaciones;
import ecol.comun.comunicaciones.ServicioComunicacionesImpl;
import ecol.comun.excepciones.AutenticacioInvalidaException;
import ecol.comun.excepciones.ConexionParejaException;
import ecol.comun.excepciones.ParejaEstablecidaException;
import ecol.servidor.recursos.IServicioRecursosServidor;
import ecol.servidor.sesion.IControladorConexionServidor;


/**
 * Esta clase se encarga de controlar toda la sesión de trabajo en la parte del
 * cliente. Mantiene los controladores de recursos, de comunicaciones, así como
 * información sobre los desarrolladores que están participando.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ControladorSesionCliente implements IControladorConexionCliente {

	// Ventana del workbench sobre la cual se está trabajando
	// y que nos limitaremos a controlar.

	private IWorkbenchWindow window;


	private IDisplayInfoTrabajo display;

	private String nombreProyecto;

	private String host;

	private ServicioRecursosClienteImpl srvRecursos;
```

```
private ServicioComunicacionesImpl srvComunic;

private ParejaProgramacion infoUsuario;

// Cosas del servidor.

private Registry regRMI;

private IControladorConexionServidor cnxServidor;

private ParejaProgramacion pareja;


/**
 * Constructor de un nuevo controlador de sesión.
 *
 * @param name
 *          Es el nombre del proyecto sobre el que se va a trabajar.
 * @param window
 *          Ventana de trabajo desde la que se va a realizar el trabajo
 *          colaborativo.
 * @throws RemoteException
 *           Excepción lanzada en la comunicación RMI.
 * @throws NotBoundException
 *           Excepción lanzada cuando no se ha logrado obtener un objeto
 *           remoto con un nombre dado.
 * @throws ConexionParejaException
 *           Excepción lanzada cuando el servidor obtiene errores al
 *           contactar con la pareja recien conectada.
 * @throws AutenticacioInvalidaException
 *           Excepción lanzada cuando los datos de acceso no son válidos.
 * @throws ParejaEstablecidaException
 *           Excepción lanzada cuando ya existe una pareja conectada al
 *           proyecto fuente.
 */
public ControladorSesionCliente(String name, IWorkbenchWindow window)
                    throws RemoteException, NotBoundException, ConexionParejaException,
                    AutenticacioInvalidaException, ParejaEstablecidaException {

          UnicastRemoteObject.exportObject(this);
          nombreProyecto = name;
          this.window = window;
          infoUsuario = encapsularUsuario(name);

          host = UtilidadesProyecto.getPropiedadString(name,
                              UtilidadesProyecto.SCOPE_CLIENTE,
                              UtilidadesProyecto.KEY_HOST_FUENTE);
          regRMI = LocateRegistry.getRegistry(host);

          lanzarServicios();

          pareja = conectarProyecto();
          emparejarServicios();
}


/**
 * Se encarga de liberar los recursos asociados a los servicios del usuario
 * cliente, desactivar listeners, etc...
 *
 */
private void limpiarRecursos() {
          if (srvComunic != null) {

                    srvComunic.terminar();
                    srvComunic = null;

          }

          if (srvRecursos != null) {
```

```
                        srvRecursos.terminar();
                        srvRecursos = null;
                }
                if (cnxServidor != null) {
                        try {
                                cnxServidor.desconectarPareja();
                        } catch (RemoteException e) {
                                // No nos importa que haya terminado mal.
                        }
                        cnxServidor = null;
                }
        }



        /**
         * Método encargado de obtener los detalles del usuario almacenados en el
         * proyecto de trabajo.
         *
         * @param proyecto
         *          Nombre del proyecto de donde se estraerá la información
         *          eprsonal del usuario.
         * @return Devuelve un objeto ParejaProgramacion con la informacion personal
         *       encapsulada.
         * @see ecol.comun.ParejaProgramacion
         */
        private ParejaProgramacion encapsularUsuario(String proyecto) {
                ParejaProgramacion yo = new ParejaProgramacion(UtilidadesProyecto
                                .getPropiedadString(proyecto, UtilidadesProyecto.SCOPE_CLIENTE,
                                                UtilidadesProyecto.KEY_LOGIN_PAREJA),
                                UtilidadesProyecto.getPropiedadString(proyecto,
                                                UtilidadesProyecto.SCOPE_CLIENTE,
                                                UtilidadesProyecto.KEY_PASS_PAREJA),
UtilidadesProyecto
                                                .getPropiedadString(proyecto,

        UtilidadesProyecto.SCOPE_CLIENTE,

        UtilidadesProyecto.KEY_NOMBRE_USUARIO),
                                UtilidadesProyecto.getPropiedadString(proyecto,
                                                UtilidadesProyecto.SCOPE_CLIENTE,
                                                UtilidadesProyecto.KEY_EMAIL_USUARIO),
                                UtilidadesProyecto.getPropiedadString(proyecto,
                                                UtilidadesProyecto.SCOPE_CLIENTE,
                                                UtilidadesProyecto.KEY_OTRAINFO_USUARIO));
                return yo;
        }


        /**
         * Método encargado de asociar a cada uno de los controladores (sesion y
         * comunicaciones) con su homólogo en la parte del servidor.
         *
         * @throws RemoteException
         *           Excepción lanzada si existen errores en la comunicación RMI.
         */
        private void emparejarServicios() throws RemoteException {

                /*
                 * Emparejamos el serivico de recursos.
                 */
                IServicioRecursosServidor recServidor = cnxServidor
                                .getServicioRecursos();
                srvRecursos.setServicioPareja(recServidor);


                /*
                 * Emparejamos el serivico de comunicaciones.
                 */
                IServicioComunicaciones comServidor = cnxServidor
                                .getServicioComunicaciones();
```

```
            srvComunic.setParejaServicio(comServidor);

        }

        /**
         * Método encargado de inicializar los controladores de recursos y
         * comunicaciones.
         *
         * @throws RemoteException
         *             Excepción lanzada si existen errores en la comunicación RMI.
         */
        private void lanzarServicios() throws RemoteException {
            // Empezamos lanzando la gestiÛn de recursos.
            srvRecursos = new ServicioRecursosClienteImpl(window, nombreProyecto);
            // Seguimos con las cosas de comunicacion.
            srvComunic = new ServicioComunicacionesImpl(window, infoUsuario,
                                nombreProyecto);

        }


        /**
         * Obtiene el controlador de comunicaciones local del cliente.
         *
         * @return ServicioComunicacionesImpl eferencia al controlador de
         *         comunicaciones del cliente.
         * @see ecol.comun.comunicaciones.ServicioComunicacionesImpl
         */
        public ServicioComunicacionesImpl getControladorComunicaciones() {

            return srvComunic;

        }


        /**
         * Obtiene el controlador de recursos del cliente.
         *
         * @return ServicioRecursosClienteImpl Referencia al controlador de recursos
         *         del cliente.
         * @see ecol.cliente.recursos.ServicioRecursosClienteImpl
         */
        public ServicioRecursosClienteImpl getControladorRecursos() {

            return srvRecursos;

        }


        /**
         * Obtiene el nombre del proyecto sobre el que se está trabajando.
         *
         * @return String Nombre del proyecto sobre el que se trabaja.
         */
        public String getNombreProyecto() {
            return nombreProyecto;
        }

        /**
         * Se encarga de negociar la conexión con el usuario fuente del proyecto.
         *
         * @return Un objeto ParejaProgramacion que representa al usuario remoto,
         *         con su información personal, o null si algo ha ido mal.
         * @throws ConnectIOException
         *             Excepción lanzada si se produce un error de comunicación.
         * @throws RemoteException
         *             Excepción lanzada en la comunicación RMI.
         * @throws NotBoundException
         *             Excepción lanzada cuando no se ha logrado obtener un objeto
         *             remoto con un nombre dado.
         * @throws ConexionParejaException
```

```
	 *		Excepción lanzada cuando el servidor obtiene errores al
	 *		contactar con la pareja recien conectada.
	 * @throws AutenticacioInvalidaException
	 *		Excepción lanzada cuando los datos de acceso no son válidos.
	 * @throws ParejaEstablecidaException
	 *		Excepción lanzada cuando ya existe una pareja conectada al
	 *		proyecto fuente.
	 */
	private ParejaProgramacion conectarProyecto() throws ConnectIOException,
			RemoteException, NotBoundException, ConexionParejaException,
			AutenticacioInvalidaException, ParejaEstablecidaException {
		cnxServidor = (IControladorConexionServidor) regRMI
				.lookup(UtilidadesSesion.INT_SERVIDOR);
		return cnxServidor.conectarPareja(infoUsuario, this);
	}

	/**
	 * Método invocado por el usuario fuente del proyecto cuando se debee
	 * notificar que ha terminado la sesión de trabajo de colaboración.
	 *
	 * @throws RemoteException
	 *		Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void terminarSesion() throws RemoteException {
		cnxServidor = null;
		window.getWorkbench().getDisplay().asyncExec(new Runnable() {
			public void run() {
				MessageDialog
					.openInformation(window.getShell(),
						"Ecol: Información",
						"El servidor ha finalizado la sesión,
guarde cambios y finalice");
			}
		});
	}

	/**
	 * Realiza la terminación de la sesión de trabajo con el servidor fuente del
	 * proyecto.
	 *
	 * @throws RemoteException
	 *		Excepción lanzada en un fallo RMI.
	 */
	public void realizarDesconexion() throws RemoteException {

		srvRecursos.terminar();
		srvComunic.terminar();
		srvRecursos = null;
		srvComunic = null;
		// Comprobamos esto, porque si era null es que nos llamó el servidor
		// antes para desconectar.
		if (cnxServidor != null)
			cnxServidor.desconectarPareja();


	}


	/**
	 * Obtiene el objeto que representa la información personal del usuario
	 * local.
	 *
	 * @return ParejaProgramacion Informacion del usuario local.
	 * @see ecol.comun.ParejaProgramacion
	 */
	public ParejaProgramacion getInfoPersonal() {

		return infoUsuario;
	}
```

```
/**
 * Obtiene el objeto que representa la información personal del usuario
 * remoto.
 *
 * @return ParejaProgramacion Informacion del usuario remoto.
 * @see ecol.comun.ParejaProgramacion
 */
public ParejaProgramacion getInfoPareja() {

        return pareja;

}



/**
 * Establece el elemento que servirá de visor para la información de trabajo
 * de la sesión colaborativa.
 *
 * @param view
 *          Elemento que implemente la interfaz IDisplayInfoTrabajo. O
 *          null si no se desea ningún visor.
 * @see ecol.comun.IDisplayInfoTrabajo
 */
public void setDisplayUsuario(IDisplayInfoTrabajo view) {

        display = view;
}

/**
 * Obtiene una referencia al elemento que sirve de visor para la información
 * de la sesión detrabajo colaborativo.
 *
 * @return IDisplayInfoTrabajo Referencia al elemento visor de información.
 *       Null si no existe.
 * @see ecol.comun.IDisplayInfoTrabajo
 */
public IDisplayInfoTrabajo getDisplayUsuario() {

        return display;

}



/**
 * Retorna la interfaz de comunicaciones del cliente exportada al usuario
 * fuente para mantener la comunicacion.
 *
 * @return referencia a la interfaz remota exportada para el control de
 *       comunicaciones.
 * @throws RemoteException
 *           Excepción lanzada si hay algún error con la comunicación RMI.
 * @see ecol.comun.comunicaciones.IServicioComunicaciones
 */
public IServicioComunicaciones getServicioComunicaciones()
                throws RemoteException {
        return srvComunic;
}


/**
 * Retorna la interfaz de recursos del cliente exportada al usuario fuente
 * para el manejo de los mismos.
 *
 * @return referencia a la interfaz remota exportada para el control de
 *       recursos.
 * @throws RemoteException
 *           Excepción lanzada si hay algún error con la comunicación RMI.
 * @see ecol.cliente.recursos.IServicioRecursosCliente
 */
```

```
	public IServicioRecursosCliente getServicioRecursos()
				throws RemoteException {
		return srvRecursos;
	}

	/**
	 * Método invocado cuando se va a acabar la sesión bruscamente.
	 *
	 */
	public void abortar() {
		try {
			cnxServidor.desconectarPareja();
	} catch (RemoteException e) {
		// Estamos abortando la conexión, le decimos al servidor que terminamos
		// 'por cortesía' ya que quiza ya no esté accesible. No nos importa
		// si no llega el mensaje.
	}
		srvRecursos.terminar();
		srvComunic.terminar();
		try {
			//TODO Desconfiar
			UnicastRemoteObject.unexportObject(this, true);
			UnicastRemoteObject.unexportObject(srvRecursos, true);
			UnicastRemoteObject.unexportObject(srvComunic, true);
		} catch (NoSuchObjectException e) {
			// No nos interesa mucho esta excepción, nos estamos haciendo no
			// exportables a nosotros.
		}
	}
}
```

Clase `IControladorConexionCliente.java`:

```
package ecol.cliente.sesion;

import java.rmi.Remote;
import java.rmi.RemoteException;

import ecol.cliente.recursos.IServicioRecursosCliente;
import ecol.comun.comunicaciones.IServicioComunicaciones;

/**
 * Interfaz que sirve de medio de comunicación para que el usuario fuente se
 * comunique con el usuario cliente en cuanto a tareas generales de conexión.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IControladorConexionCliente extends Remote {

	/**
	 * Método invocado por el usuario fuente del proyecto cuando se debee notificar
	 * que ha terminado la sesión de trabajo de colaboración.
	 * @throws RemoteException
	 *		Excepción lanzada si hay algún error con la comunicación RMI.
	 */
	public void terminarSesion() throws RemoteException;

	/**
	 * Retorna la interfaz de recursos del cliente exportada al usuario fuente para
	 * el manejo de los mismos.
	 * @return referencia a la interfaz remota exportada para el control de recursos.
	 * @throws RemoteException
	 *		Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.cliente.recursos.IServicioRecursosCliente
	 */
	public IServicioRecursosCliente getServicioRecursos()
				throws RemoteException;

	/**
```

```
         * Retorna la interfaz de comunicaciones del cliente exportada al usuario fuente para
         * mantener la comunicacion.
         * @return referencia a la interfaz remota exportada para el control de comunicaciones.
         * @throws RemoteException
         *           Excepción lanzada si hay algún error con la comunicación RMI.
         * @see ecol.comun.comunicaciones.IServicioComunicaciones
         */
        public IServicioComunicaciones getServicioComunicaciones()
                        throws RemoteException;
}
```

Paquete `ecol.cliente.vistas`:

Clase `ChatClienteView.java`:

```
package ecol.cliente.vistas;

import org.eclipse.swt.SWT;
import org.eclipse.swt.dnd.Clipboard;
import org.eclipse.swt.dnd.TextTransfer;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.part.ViewPart;

import ecol.cliente.EcolCliente;
import ecol.comun.comunicaciones.IDisplayMensajes;
import ecol.comun.comunicaciones.MensajeChat;

/**
 * Clase que se encarga de crear la vista para la comunicación mediante un chat
 * textual con el otro usuario que participa en la sesión de comunicación.
 * Permite el envio de mensajes textuales introducidos por el usuario y que
 * estén en el portapapeles.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ChatClienteView extends ViewPart implements IDisplayMensajes {
        private Button btEnviar;

        private Button btEnviarPaste;

        private Text txtAEnviar;

        private Text listaMensajes;

        /**
         * El constructor.
         */
        public ChatClienteView() {
        }

        /**
         * Este método es llamado y nos permite crear la vista e inicializarla.
         *
         * @param parent
         *           Composite sobre el que empezaremos a añadir elementos.
         */
        public void createPartControl(Composite parent) {

                if (!EcolCliente.estaConectadoProyecto()) {
                        Label noIniciado = new Label(parent, SWT.NONE);
```

```
                        noIniciado.setText("No ha iniciado ninguna sesión de trabajo.");
                        return;
                }

                final Clipboard clip = new Clipboard(parent.getDisplay());

                FillLayout fillLayout = new FillLayout();
                fillLayout.type = SWT.VERTICAL;
                parent.setLayout(fillLayout);

                Composite panelMensajes = new Composite(parent, SWT.NONE);
                GridLayout layout = new GridLayout();
                layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
1;
                layout.verticalSpacing = 10;
                panelMensajes.setLayout(layout);

                GridData data = new GridData();
                data.horizontalAlignment = SWT.FILL;
                data.grabExcessHorizontalSpace = true;
                data.widthHint = 100;
                data.verticalAlignment = SWT.FILL;
                data.grabExcessVerticalSpace = true;
                data.heightHint = 100;

                listaMensajes = new Text(panelMensajes, SWT.BORDER | SWT.V_SCROLL
                                | SWT.WRAP | SWT.READ_ONLY);
                listaMensajes.setLayoutData(data);

                Composite panelEnvio = new Composite(parent, SWT.NONE);
                GridLayout gridLayout = new GridLayout();
                gridLayout.numColumns = 2;
                panelEnvio.setLayout(gridLayout);

                txtAEnviar = new Text(panelEnvio, SWT.SINGLE | SWT.BORDER);
                txtAEnviar.setText("");
                data = new GridData();
                data.horizontalAlignment = GridData.FILL;
                data.grabExcessHorizontalSpace = true;
                txtAEnviar.setLayoutData(data);
                txtAEnviar.addModifyListener(new TextoAEnviarListener());
                btEnviar = new Button(panelEnvio, SWT.PUSH);
                btEnviar.setText("Enviar");
                btEnviar.setEnabled(false);
                btEnviar.addSelectionListener(new SelectionAdapter() {
                        public void widgetSelected(SelectionEvent e) {
                                String texto = txtAEnviar.getText();
                                txtAEnviar.setText("");
                                EcolCliente.getControladorSesion()

        .getControladorComunicaciones().enviarMensajeAPareja(
                                                                texto);
                        }

                });
                btEnviarPaste = new Button(panelEnvio, SWT.PUSH);
                btEnviarPaste.setText("Enviar desde Portapapeles");
                btEnviarPaste.addSelectionListener(new SelectionAdapter() {
                        public void widgetSelected(SelectionEvent e) {
                                TextTransfer transfer = TextTransfer.getInstance();
                                String datos = (String) clip.getContents(transfer);
                                if (datos != null) {
                                        EcolCliente.getControladorSesion()
                                                        .getControladorComunicaciones()
                                                        .enviarMensajeAPareja(datos);
                                }
                        }

                });

                EcolCliente.getControladorSesion().getControladorComunicaciones()
```

```
                                        .setDisplayMensajes(this);
                parent.getShell().setDefaultButton(btEnviar);
        }

        /**
         * Método invocado cuando se cierra la vista. Útil para notificar que ya no
         * hay un visor de mensajes al entorno.
         */
        public void dispose() {
                if (EcolCliente.estaConectadoProyecto())
                        EcolCliente.getControladorSesion().getControladorComunicaciones()
                                        .setDisplayMensajes(null);
        }

        /**
         * Establece el foco en el control de la lista de mensajes.
         */
        public void setFocus() {
                listaMensajes.setFocus();
        }

        /**
         * Clase que se encarga de controlar los eventos de modificación producidos
         * en el cuadro de texto que representa el mensaje del usuario. Su objetivo
         * principal es actualizar el botón de enviar.
         *
         * @author Luis Fernández Álvarez
         *
         */
        private class TextoAEnviarListener implements ModifyListener {

                /**
                 * Método invocado cuando se produce el evento de modificación.
                 * @param e Evento de modificación recibido.
                 */
                public void modifyText(ModifyEvent e) {
                        btEnviar.setEnabled(txtAEnviar.getText().compareTo("") != 0);
                }
        }

        /**
         * Método invocado cuando llega un nuevo mensaje de chat al sistema.
         *
         * @param mensaje
         *          Mensaje de chat encapsulado en un objeto MensajeChat.
         */
        public void nuevoMensaje(MensajeChat mensaje) {
                listaMensajes.append(mensaje.toString() + "\n");
        }

        /**
         * Método invocado cuando llega un mensaje del sistema (Por ejemplo la
         * conexión de un usuario.
         *
         * @param mensaje
         *          String que representa el mensaje.
         */
        public void mensajeSistema(String mensaje) {
                listaMensajes.append("** " + mensaje + "\n");
        }
}
```

Clase `InformacionParejaView.java`:

```
package ecol.cliente.vistas;

import org.eclipse.jface.viewers.ListViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.ExpandBar;
```

```
import org.eclipse.swt.widgets.ExpandItem;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Link;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;

import ecol.cliente.EcolCliente;
import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.ParejaProgramacion;
import ecol.comun.RecursoCompartido;

/**
 * Clase que implementa un visor de información de trabajo particular
 * para el entorno de trabajo del usuario cliente.
 * Se trata de una vista.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class InformacionParejaView extends ViewPart implements
                IDisplayInfoTrabajo {
        private Composite parent;

        private ListViewer viewer;

        private Label labelNombreRemoto;

        private Link labelEmailRemoto;

        private Label labelOtraInfoRemota;

        private Label labelJDKRemota;

        private Label labelOSRemoto;

        /**
         * Este método es llamado y nos permite crear la vista e inicializarla.
         *
         * @param parent
         *          Composite sobre el que empezaremos a añadir elementos.
         */
        public void createPartControl(Composite parent) {

                this.parent = parent;
                if (!EcolCliente.estaConectadoProyecto()) {
                        Label noIniciado = new Label(parent, SWT.NONE);
                        noIniciado.setText("No ha iniciado ninguna sesión de trabajo.");
                        return;
                }
                ParejaProgramacion infoLocal = EcolCliente.getControladorSesion()
                                .getInfoPersonal();
                ParejaProgramacion infoRemota = EcolCliente.getControladorSesion()
                                .getInfoPareja();
                // Con Expand bar.
                ExpandBar bar = new ExpandBar(parent, SWT.V_SCROLL);

                /*
                 * Creamos el último group para los documentos compartidos.
                 */
                Composite composite = new Composite(bar, SWT.NONE);
                GridLayout layout = new GridLayout();
                layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
                layout.verticalSpacing = 10;
                composite.setLayout(layout);

                GridData data = new GridData();
                data.horizontalAlignment = SWT.FILL;
                data.grabExcessHorizontalSpace = true;
                data.widthHint = 100;
```

```
                data.verticalAlignment = SWT.FILL;
                data.grabExcessVerticalSpace = true;
                data.heightHint = 100;
                // Bounds: Rectangle {9, 21, 228, 397}

                viewer = new ListViewer(composite, SWT.SINGLE | SWT.V_SCROLL
                                | SWT.H_SCROLL);
                viewer.getList().computeSize(SWT.DEFAULT, SWT.DEFAULT);
                viewer.getList().setLayoutData(data);

                ExpandItem item2 = new ExpandItem(bar, SWT.NONE, 0);
                item2.setText("En coedición");
                item2.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
                item2.setControl(composite);
                item2.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
                                ISharedImages.IMG_OBJ_PROJECT));

                /*
                 * Creamos el Composite inferior, que será la información de nuestra
                 * pareja remota de programación.
                 */
                composite = new Composite(bar, SWT.NONE);
                layout = new GridLayout();
                layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
                layout.verticalSpacing = 10;
                composite.setLayout(layout);

                labelNombreRemoto = new Label(composite, SWT.NONE);
                labelNombreRemoto.setText("Nombre: " + infoRemota.getNombre());
                labelEmailRemoto = new Link(composite, SWT.NONE);

                labelEmailRemoto
                                .setText("E-Mail: <A>" + infoRemota.getEmail() + "</A>");
                labelOtraInfoRemota = new Label(composite, SWT.NONE);
                labelOtraInfoRemota
                                .setText("Otros: " + infoRemota.getOtraInformacion());
                labelJDKRemota = new Label(composite, SWT.NONE);
                labelJDKRemota.setText("JDK: " + infoRemota.getJDK());
                labelOSRemoto = new Label(composite, SWT.NONE);
                labelOSRemoto.setText("SO: " + infoRemota.getOS());

                ExpandItem item1 = new ExpandItem(bar, SWT.NONE, 0);
                item1.setText("Información pareja");
                item1.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
                item1.setControl(composite);
                item1.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
                                ISharedImages.IMG_OBJ_PROJECT));

                /*
                 * Creamos el Composite superior, que será la információn personal de la
                 * persona local.
                 */
                composite = new Composite(bar, SWT.NONE);
                layout = new GridLayout();
                layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
                layout.verticalSpacing = 10;
                composite.setLayout(layout);

                Label labelNombreLocal = new Label(composite, SWT.NONE);
                labelNombreLocal.setText("Nombre: " + infoLocal.getNombre());
                Link labelEmailLocal = new Link(composite, SWT.NONE);
                labelEmailLocal.setText("E-Mail: <A>" + infoLocal.getEmail() + "</A>");
                Label labelOtraInfoLocal = new Label(composite, SWT.NONE);
                labelOtraInfoLocal.setText("Otros: " + infoLocal.getOtraInformacion());
                Label labelJDKLocal = new Label(composite, SWT.NONE);
                labelJDKLocal.setText("JDK: " + infoLocal.getJDK());
                Label labelOSLocal = new Label(composite, SWT.NONE);
                labelOSLocal.setText("SO: " + infoLocal.getOS());
```

```
		ExpandItem item0 = new ExpandItem(bar, SWT.NONE, 0);
		item0.setText("Información personal");
		item0.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
		item0.setControl(composite);
		item0.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
				ISharedImages.IMG_OBJ_PROJECT));

		bar.setSpacing(8);

		EcolCliente.getControladorSesion().setDisplayUsuario(this);

	}

	/**
	 * Establece el foco en el control de la lista de mensajes.
	 */
	public void setFocus() {
		viewer.getControl();
	}

	/**
	 * Método invocado cuando se cierra la vista. Útil para notificar que ya no
	 * hay un visor de información al entorno.
	 */
	public void dispose() {
		if (EcolCliente.estaConectadoProyecto())
			EcolCliente.getControladorSesion().setDisplayUsuario(null);

	}

	/**
	 * Método invocado cuando se añade un nuevo recurso para coedición
	 * al sistema. El visor debe facilitar al usuario su control
	 *
	 * @param recurso Referencia al recurso compartido que se empieza a coeditar.
	 */
	public void addCoEditado(final RecursoCompartido recurso) {
		parent.getDisplay().asyncExec(new Runnable() {
			public void run() {
				viewer.getList().add(recurso.getID());
			}

		});
	}

	/**
	 * Método invocado cuando se termina de coeditar un recurso determinado.
	 *
	 * @param id Identificador del recurso que ha dejado de ser coeditable.
	 */
	public void eliminarCoEditado(final String id) {
		parent.getDisplay().asyncExec(new Runnable() {
			public void run() {
				viewer.getList().remove(id);
			}

		});

	}

	/**
	 * Método invocado por el sistema cuando una nueva pareja se conecta a la
	 * sesión de trabajo.
	 *
	 * @param pareja
	 *          Objeto ParejaProgramacion que encapsula la información a
	 *          mostrar.
	 */
	public void parejaConectada(ParejaProgramacion pareja) {
		// No es útil en el cliente con la implementación actual
	}
```

```
	/**
	 * Método invocado cuando la pareja de programación se ha desconectado del
	 * entorno. Se debe notificar de ello al usuario.
	 *
	 */
	public void parejaDesconectada() {
		// No es útil en el cliente con la implementación actual
	}

}
```

Paquete `ecol.cliente.wizards`:

Clase `DetallesProyectoPage.java`:

```
package ecol.cliente.wizards;

import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

/**
 * Clase que implementa la página del asistente para los detalles del proyecto
 * cliente que se va ha crear.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class DetallesProyectoPage extends WizardPage {

	/**
	 * Clase que se encarga de controlar los eventos de modificación producidos
	 * en el cuadro de texto que representan los diferentes campos. Su objetivo
	 * principal es actualizar los botones del asistente.
	 *
	 * @author Luis Fernández Álvarez
	 *
	 */
	private class DetallesProyectoListener implements ModifyListener {

		/**
		 * Método invocado cuando se produce el evento de modificación.
		 * @param e Evento de modificación recibido.
		 */
		public void modifyText(ModifyEvent e) {
			getWizard().getContainer().updateButtons();
		}
	}

	private Text textoIp;

	private Text textoUsuario;

	private Text textoPassword;

	private Text txtNombrePersonal;

	private Text txtEmailPersonal;

	private Text txtOtraInfoPersonal;

	/**
	 * Constructor de la página del asistente encargada de tomar los detalles
```

```
	 * del proyecto creado.
	 *
	 * @param pageName Nombre de la página
	 */
	protected DetallesProyectoPage(String pageName) {
		super(pageName);
		this.setTitle("Detalles del proyecto");
		this
				.setDescription("Introduzca los detalles del proyecto que desea
crear.");
	}

	/**
	 * Método invocado para la creación visual de la página del asistente.
	 *
	 * @param parent
	 *          Composite donde iremos creando los elementos.
	 */
	public void createControl(Composite parent) {
		// create main composite & set layout
		Composite mainComposite = new Composite(parent, SWT.NONE);
		GridLayout gridLayout = new GridLayout();
		gridLayout.numColumns = 1;
		gridLayout.verticalSpacing = 10;
		mainComposite.setLayout(gridLayout);

		// create contents
		createPageContent(mainComposite);

		// page setting
		setControl(mainComposite);
	}

	/**
	 * Crea el contenido de la página del asistente.
	 *
	 * @param parent Composite donde crearemos los elementos del asistente.
	 */
	public void createPageContent(Composite parent) {
		// Listener de las casillas de los detalles del proyecto.
		DetallesProyectoListener textListener = new DetallesProyectoListener();

		// PANEL SERVIDOR
		Group groupIp = new Group(parent, SWT.SHADOW_ETCHED_IN);
		groupIp.setText("InformaciÛn servidor");
		GridLayout gridLayout = new GridLayout();
		gridLayout.numColumns = 1;
		groupIp.setLayout(gridLayout);
		GridData gridData = new GridData();
		gridData.horizontalAlignment = GridData.FILL;
		gridData.grabExcessHorizontalSpace = true;

		groupIp.setLayoutData(gridData);
		Label labelIp = new Label(groupIp, SWT.LEFT);
		labelIp.setText("Ip del servidor fuente: ");

		textoIp = new Text(groupIp, SWT.SINGLE | SWT.BORDER | SWT.LEFT);
		textoIp.addModifyListener(textListener);
		textoIp.setLayoutData(gridData);

		// PANEL DEL USUARIO
		Group groupUsuario = new Group(parent, SWT.SHADOW_ETCHED_IN);
		groupUsuario.setText("InformaciÛn del usuario");
		groupUsuario.setLayout(gridLayout);
		GridData gridData2 = new GridData();
		gridData2.horizontalAlignment = GridData.FILL;
		gridData2.grabExcessHorizontalSpace = true;

		groupUsuario.setLayoutData(gridData2);

		Label labelUsuario = new Label(groupUsuario, SWT.LEFT);
```

```
		labelUsuario.setText("Nombre de usuario: ");

		textoUsuario = new Text(groupUsuario, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		textoUsuario.addModifyListener(textListener);
		textoUsuario.setLayoutData(gridData2);

		Label labelPassword = new Label(groupUsuario, SWT.LEFT);
		labelPassword.setText("Clave de usuario: ");

		textoPassword = new Text(groupUsuario, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		textoPassword.addModifyListener(textListener);
		textoPassword.setLayoutData(gridData2);
		textoPassword.setEchoChar('*');
		/*
		 * Segundo panel: InformaciÛn personal.
		 */
		Group groupPersonal = new Group(parent, SWT.SHADOW_ETCHED_IN);
		groupPersonal.setText("Información personal: ");
		GridLayout layoutPersonal = new GridLayout();
		layoutPersonal.numColumns = 1;
		groupPersonal.setLayout(layoutPersonal);
		GridData dataPersonal = new GridData();
		dataPersonal.horizontalAlignment = GridData.FILL;
		dataPersonal.grabExcessHorizontalSpace = true;
		groupPersonal.setLayoutData(dataPersonal);

		// Primera entrada: nombre completo.
		Label labelNombrePersonal = new Label(groupPersonal, SWT.LEFT);
		labelNombrePersonal.setText("Nombre completo: ");
		txtNombrePersonal = new Text(groupPersonal, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		txtNombrePersonal.addModifyListener(textListener);
		txtNombrePersonal.setLayoutData(dataPersonal);

		// Segunda entrada: E-Mail
		Label labelEmailPersonal = new Label(groupPersonal, SWT.LEFT);
		labelEmailPersonal.setText("Dirección de correo: ");
		txtEmailPersonal = new Text(groupPersonal, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		txtEmailPersonal.addModifyListener(textListener);
		txtEmailPersonal.setLayoutData(dataPersonal);

		// Tercera entrada: Otra información.
		Label labelOtraInfoPersonal = new Label(groupPersonal, SWT.LEFT);
		labelOtraInfoPersonal.setText("Otra información: ");
		txtOtraInfoPersonal = new Text(groupPersonal, SWT.MULTI | SWT.BORDER
				| SWT.LEFT);
		txtOtraInfoPersonal.addModifyListener(textListener);
		txtOtraInfoPersonal.setLayoutData(dataPersonal);
	}

	/**
	 * Determina si una cadena está vacía.
	 * @param cadena String a comprobar.
	 * @return Verdadero si esta vacía y falso en el caso contrario.
	 */
	private boolean estaVacia(String cadena) {
		return cadena.length() == 0;
	}

	/**
	 * Obtiene el email personal introducido.
	 * @return String con el email personal.
	 */
	public String getEmailPersona() {
		return txtEmailPersonal.getText();
	}

	/**
```

```
	 * Devuelve el valor introducido por el usario en la casilla de host.
	 *
	 * @return String que representa el host introducido
	 */
	public String getIp() {
		return textoIp.getText();
	}

	/**
	 * Obtiene el nombre introducido.
	 * @return String con el nombre personal.
	 */
	public String getNombrePersonal() {
		return txtNombrePersonal.getText();
	}

	/**
	 * Obtiene el campo de información adicional introducido.
	 * @return String con el campo de información adicional
	 */
	public String getOtraInfoPersonal() {
		return txtOtraInfoPersonal.getText();
	}

	/**
	 * Devuelve el valor introducido por el usario en la casilla de contraseÒa.
	 *
	 * @return String que representa el password introducido
	 */
	public String getPassword() {
		return textoPassword.getText();
	}

	/**
	 * Devuelve el valor introducido por el usario en la casilla de nombre de
	 * usuario.
	 * @return String que representa el login introducido
	 */
	public String getUsuario() {
		return textoUsuario.getText();
	}

	/**
	 * Determina si la página ha sido completada adecuadamente.
	 */
	public boolean isPageComplete() {
		return !estaVacia(getIp()) && !estaVacia(getUsuario())
				&& !estaVacia(getPassword()) && !estaVacia(getNombrePersonal())
				&& !estaVacia(getEmailPersona())
				&& !estaVacia(getOtraInfoPersonal());

	}

	/**
	 * Método invocado cuando la página va a ser visualizada.
	 */
	public void setVisible(boolean visible) {
		super.setVisible(visible);
	}
}
```

Clase `NuevoProyectoWizard.java`:

```
package ecol.cliente.wizards;

 import org.eclipse.core.resources.IFolder;
 import org.eclipse.core.resources.IProject;
 import org.eclipse.core.runtime.CoreException;
 import org.eclipse.core.runtime.Path;
 import org.eclipse.jdt.core.IClasspathEntry;
 import org.eclipse.jdt.core.IJavaProject;
```

```
import org.eclipse.jdt.core.JavaCore;
import org.eclipse.jdt.core.JavaModelException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.wizard.Wizard;
import org.eclipse.ui.INewWizard;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.dialogs.WizardNewProjectCreationPage;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;

/**
 * Clase que representa el asistente para la creación de un nuevo proyecto
 * cliente. Esta clase se encarga de añadir las páginas al asistente, así como
 * finalizar la tarea persistiendo los cambios.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class NuevoProyectoWizard extends Wizard implements INewWizard {

	private WizardNewProjectCreationPage paginaProyecto;

	private DetallesProyectoPage detallesProyecto;

	private IWorkbench workbench;

	/**
	 * Constructor del asistente para la creación de un proyecto colaborativo
	 * como cliente.
	 */
	public NuevoProyectoWizard() {
		super();
		setWindowTitle("E-Col: Nuevo proyecto como cliente");
	}

	/**
	 * Inicializa el asistente con el workbench sobre el que se trabaja así como
	 * los elementos seleccionados.
	 *
	 * @param workbench
	 *          Repersenta el objeto IWorkbench.
	 * @param selection
	 *          Representa los elementos seleccionados.
	 */
	public void init(IWorkbench workbench, IStructuredSelection selection) {
		this.workbench = workbench;
	}

	/**
	 * Determina si se puede activar la finalización el asistente de creación de
	 * proyecto.
	 *
	 * @return Verdadero si se puede finalizar o falso en caso contrario.
	 */
	public boolean canFinish() {
		return paginaProyecto.isPageComplete()
				&& detallesProyecto.isPageComplete();
	}

	/**
	 * Añade páginas al asistente de creación de un nuevo proyecto.
	 */
	public void addPages() {
		paginaProyecto = new WizardNewProjectCreationPage("Nuevo Proyecto");
		paginaProyecto.setTitle("Nuevo Proyecto E-Col");
		paginaProyecto
				.setDescription("Crea un nuevo proyecto cliente java colaborativo");
		detallesProyecto = new DetallesProyectoPage("Detalles de la fuente");
		this.addPage(paginaProyecto);
```

```
                this.addPage(detallesProyecto);
        }

        /**
         * Se encarga de realizar las tareas del asistente una vez el usuario lo ha
         * terminado.
         *
         * @return Verdadero si ha terminado bien o falso en caso contrario.
         */
        public boolean performFinish() {
                {
                        // Manejador del proyecto creado.
                        IProject proyecto = paginaProyecto.getProjectHandle();
                        try {
                                proyecto.create(null);
                                // Lo abrimos puesto que se crea cerrado.
                                proyecto.open(null);
                                // Establecemos su estructura propia como proyecto E-Col de
                                // Servidor.
                                UtilidadesProyecto.setNature(proyecto,
                                                "org.eclipse.jdt.core.javanature");
                                UtilidadesProyecto.setNature(proyecto,
                                                UtilidadesProyecto.NATURE_CLIENTE);

                                // En este caso la jerarquía de directorios.
                                crearCarpeta(proyecto, "src");
                                crearCarpeta(proyecto, "sesiones");
                                setEntradasClasspath(proyecto, "src");
                                try {
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,

        UtilidadesProyecto.KEY_NOMBRE_USUARIO,
                                                        detallesProyecto.getNombrePersonal());
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,
                                                        UtilidadesProyecto.KEY_EMAIL_USUARIO,
                                                        detallesProyecto.getEmailPersona());
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,

        UtilidadesProyecto.KEY_OTRAINFO_USUARIO,
                                                        detallesProyecto.getOtraInfoPersonal());
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,
                                                        UtilidadesProyecto.KEY_HOST_FUENTE,
                                                        detallesProyecto.getIp());
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,
                                                        UtilidadesProyecto.KEY_LOGIN_PAREJA,
                                                        detallesProyecto.getUsuario());
                                        UtilidadesProyecto.setPropiedadString(proyecto,
                                                        UtilidadesProyecto.SCOPE_CLIENTE,
                                                        UtilidadesProyecto.KEY_PASS_PAREJA,
                                                        detallesProyecto.getPassword());
                                } catch (BackingStoreException e) {
                                        MessageDialog
                                                        .openError(

        workbench.getActiveWorkbenchWindow()

        .getShell(),
                                                                        "Ecol Plug-In: Proyecto
cliente.",
                                                                        "Se producido errores
almacenando los detalles del proyecto, revise dichos parámetros.");
                                }
                        } catch (CoreException e) {
                                MessageDialog.openError(workbench.getActiveWorkbenchWindow()
                                                .getShell(), "Ecol Plug-In: Proyecto cliente.",
```

```
                                                    "Debido a errores, el proyecto no se ha creado
correctamente.\n"
                                                                    + e.getMessage());
                                    return false;
                            }
                    }
                    return true;
            }

            /**
             * Método utilizado para crear una carpeta bajo el directorio raíz del
             * proyecto
             *
             * @param project
             *          Referencia al IProject donde se requiere hacer al carpeta.
             * @param nombre
             *          String nombre de la carpeta a crear.
             * @throws CoreException
             *           Excepción lanzada si algo ha ido mal en la creación del
             *           recurso.
             */
            private void crearCarpeta(IProject project, String nombre)
                            throws CoreException {
                    IFolder folder = project.getFolder(new Path(nombre));
                    if(!folder.exists())
                    folder.create(true, true, null);
            }

            /**
             * Sobreescribe el Classpath de un proyecto con una entrada del contenedor
             * de la JRE, además de una entada de código fuente para el directorio
             * pasado como parámetro.
             *
             * @param project
             *          Referencia al IProject cuyo classpath será alterado.
             * @param directorio
             *          Directorio a ser añadido al classpath.
             * @throws JavaModelException
             *           Excepción lanzada al establecer el classpath si algo fue mal.
             */
            private void setEntradasClasspath(IProject project, String directorio)
                            throws JavaModelException {
                    IJavaProject ijp = JavaCore.create(project);
                    IClasspathEntry cpath[] = new IClasspathEntry[2];
                    cpath[0] = JavaCore.newSourceEntry(new Path("/" + project.getName()
                                    + "/" + directorio));
                    cpath[1] = JavaCore.newContainerEntry(new Path(
                                    "org.eclipse.jdt.launching.JRE_CONTAINER"), false);
                    ijp.setRawClasspath(cpath, null);

            }
    }
```

Paquete `ecol.comun`:

Clase `Anotacion.java`:

```
package ecol.comun;


import java.io.Serializable;

/**
 * Clase que encapsula la información de las anotaciones creadas por los
 * usuarios. Implementa Serializable para poder ser enviado mediante RMI.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class Anotacion implements Serializable {
```

```
	private String texto;

	private int prioridad;

	private boolean hecho;

	/**
	 * Crea una nueva anotacion con todos sus campos.
	 *
	 * @param texto
	 *          String que representa el texto de la anotación.
	 * @param prioridad
	 *          Valor entero que representa su prioridad. Son constantes
	 *          definidas en IMarker.PRIORITY_*
	 * @param hecho
	 */
	public Anotacion(String texto, int prioridad, boolean hecho) {

		this.texto = texto;
		this.prioridad = prioridad;
		this.hecho = hecho;

	}

	/**
	 * Retorna si la anotación está hecha o no.
	 *
	 * @return Verdadero si está hecha, falso en caso contrario.
	 */
	public boolean isHecho() {
		return hecho;
	}


	/**
	 * Establece si una anotación está hecha o no.
	 *
	 * @param hecho
	 *          Valor boolean que representa si está o no hecha la anotación.
	 */
	public void setHecho(boolean hecho) {

		this.hecho = hecho;

	}


	/**
	 * Retorna la prioridad de la anotación.
	 *
	 * @return valor entero definido en IMarker.PRIORITY_*
	 */
	public int getPrioridad() {

		return prioridad;

	}


	/**
	 * Establece la prioridad de la anotación.
	 *
	 * @param prioridad
	 *          Debe ser un entero valido definido en IMarker.PRIORITY_*
	 */
	public void setPrioridad(int prioridad) {
		this.prioridad = prioridad;
	}


	/**
```

```
         * Retornar el texto de la anotación.
         *
         * @return String que representa el texto de la anotación.
         */
        public String getTexto() {

                return texto;

        }


        /**
         * Establece el texto de la anotación
         *
         * @param texto
         *          String que representa el nuevo texto de la anotación.
         */
        public void setTexto(String texto) {

                this.texto = texto;

        }

}
```

Clase `IDisplayInfoTrabajo.java`:

```
package ecol.comun;

/**
 * Interfaz que deben implementar todos los objetos que deseen ser usados como
 * visor de información del sistema.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IDisplayInfoTrabajo {

        /**
         * Método invocado por el sistema cuando una nueva pareja se
         * conecta a la sesión de trabajo.
         *
         * @param pareja Objeto ParejaProgramacion que encapsula la información a mostrar.
         */
        public void parejaConectada(ParejaProgramacion pareja);

        /**
         * Método invocado cuando la pareja de programación se ha
         * desconectado del entorno. Se debe notificar de ello al usuario.
         *
         */
        public void parejaDesconectada();

        /**
         * Método invocado cuando se añade un nuevo recurso para coedición
         * al sistema. El visor debe facilitar al usuario su control
         *
         * @param recurso Referencia al recurso compartido que se empieza a coeditar.
         */
        public void addCoEditado(RecursoCompartido recurso);

        /**
         * Método invocado cuando se termina de coeditar un recurso determinado.
         *
         * @param id Identificador del recurso que ha dejado de ser coeditable.
         */
        public void eliminarCoEditado(String id);
}
```

Clase `ParejaProgramacion.java`:

```
package ecol.comun;

import java.io.Serializable;


/**
 * Clase que encapsula los datos personales y de sesión del usuario programador.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ParejaProgramacion implements Serializable {

	private static final long serialVersionUID = 1L;

	private String login;

	private String nombre;

	private String passwd;

	private String email;

	private String otraInfo;

	private String ssOO;

	private String jdk;


	/**
	 * Construye un objeto con toda la información. La información personal es
	 * pasada como parámetro, mientras que el resto de información de la sesión
	 * se obtiene del entorno.
	 *
	 * @param login
	 *		String que representa el login (ó nombre corto) del usuario.
	 * @param passwd
	 *		String que representa la clave del usuario (si procede).
	 * @param nombre
	 *		String que representa el nombre completo del usuario.
	 * @param email
	 *		String que representa el email del usuario.
	 * @param otraInfo
	 *		String que representa información adicional del usuario.
	 */
	public ParejaProgramacion(String login, String passwd, String nombre,
			String email, String otraInfo) {
		ssOO = System.getProperty("os.name") + " "
				+ System.getProperty("os.arch") + " ("
				+ System.getProperty("os.version") + ")";
		jdk = System.getProperty("java.vendor") + " ("
				+ System.getProperty("java.version") + ")";
		this.login = login;
		this.nombre = nombre;
		this.passwd = passwd;
		this.email = email;
		this.otraInfo = otraInfo;
	}


	/**
	 * Retorna el email del usuario
	 *
	 * @return String que representa el email del usuario.
	 */
	public String getEmail() {
		return email;
	}
```

```
/**
 * Retorna información adicional del usuario
 *
 * @return String que representa información adicional del usuario.
 */
public String getOtraInformacion() {
        return otraInfo;
}


/**
 * Retorna el login (o nombre corto) del usuario
 *
 * @return String que representa el login (o nombre corto según proceda) del
 *       usuario.
 */
public String getLogin() {
        return login;
}


/**
 * Retorna el nombre completo del usuario
 *
 * @return String que representa el nombre completo del usuario.
 */
public String getNombre() {
        return nombre;
}

/**
 * Retorna una cadena que representa la versión del sistema operativo
 *
 * @return String con la versión del sistema en el que se está trabajadno.
 */
public String getOS() {
        return ssOO;
}


/**
 * Retorna una versión del vendedor de la JDK instalada y su versión.
 *
 * @return String con el nombre del vendedor de la JDK instalada y su
 *       versión.
 */
public String getJDK() {
        return jdk;
}

/**
 * Retorna la clave establecida para el usuario que encapsula este objeto.
 *
 * @return String con el password establecido.
 */
public String getPasswd() {
        return passwd;
}


/**
 * Muestra una cadena resumen de la información personal del usuario.
 *
 * @return String con los datos del usuario.
 */
public String toString() {
        return "Login: " + login + "\n" + "Nombre: " + nombre + "\n"
                        + "E-Mail: " + email + "" + "Otra Info: " + otraInfo;
}
```

```
}
```

Clase `RecursoCompartido.java`:

```
package ecol.comun;


import java.io.Serializable;


/**
 * Clase que encapsula los datos básicos de un recurso compartido.
 * Los identificadores establecidos para los recursos son su path completo sin el nombre de proyecto.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class RecursoCompartido implements Serializable {

	private String name;

	private String path;

	private String ID;

	private String contenido;

	private Anotacion[] anotaciones;


	/**
	 * Constructor de un recurso compartido.
	 * @param ID Identificador del recurso compartido.
	 * @param name Nombre del recurso compartido.
	 * @param path Path relativo dentro del proyecto al recurso.
	 */
	public RecursoCompartido(String ID, String name, String path){
		this.name=name;
		this.ID=ID;
		this.path=path;
	}


	/**
	 * Constructor de un recurso compartido con contenido inicial, esto es útil para sincronizar
contenidos.
	 * @param ID Identificador del recurso compartido.
	 * @param name Nombre del recurso compartido.
	 * @param path Path relativo dentro del proyecto al recurso.
	 * @param contenido String con el contenido inicial del recurso.
	 */
	public RecursoCompartido(String ID, String name, String path, String contenido){

		this(ID,name,path);
		this.contenido=contenido;

	}


	/**
	 * Retorna el nombre del recurso.
	 * @return String que representa el nombre del recurso.
	 */
	public String getName() {
		return name;
	}


	/**
	 * Retorna el path del recurso.
```

```
         * @return String que representa el path del recurso.
         */
        public String getPath(){

                return path;

        }

        /**
         * Retorna el Identificador del recurso.
         * @return String que representa el identificador del recurso.
         */
        public String getID(){

                return ID;
        }


        /**
         * Retorna el contenido del recurso.
         * @return String que representa el contenido del recurso.
         */
        public String getContenido(){

                return contenido;

        }


        /**
         * Establece las anotaciones asociadas al recurso en particular.
         * @param anotaciones Array de anotaciones asociadas al recurso.
         */
        public void setAnotaciones(Anotacion[] anotaciones){

                this.anotaciones=anotaciones;

        }


        /**
         * Retorna el conjunto de anotaciones asociadas al recurso.
         * @return Array de anotaciones asociadas al recurso.
         */
        public Anotacion[] getAnotaciones(){

                return anotaciones;

        }
}
```

Clase `ReferenciaRecursoCompartido.java`:

```
package ecol.comun;

 import org.eclipse.core.resources.IFile;
 import org.eclipse.jface.text.IDocument;
 import org.eclipse.jface.text.IDocumentListener;
 import org.eclipse.ui.IEditorPart;

 /**
  * Clase empleada para mantener un control de los recursos en coedición en la
  * sesión de trabajo local de cada usuario.
  *
  * @author Luis Fernández Álvarez
  *
  */
 public class ReferenciaRecursoCompartido {
        private IFile fichero;
```

```
	private IDocument doc;

	private IEditorPart editor;

	private IDocumentListener listener;

	/**
	 * Crea una nueva referencia a un recurso compartido en coedición.
	 *
	 * @param fichero
	 *          Referencia al recurso IFile asociado.
	 * @param editor
	 *          Referencia al editor con el que se esta editando el recurso.
	 * @param doc
	 *          Referencia al IDocument que representa la estructura del
	 *          documento.
	 * @param listener
	 *          REferencia al listener asociado al recurso en coedición.
	 */
	public ReferenciaRecursoCompartido(IFile fichero, IEditorPart editor,
			IDocument doc, IDocumentListener listener) {
		this.listener = listener;
		this.fichero = fichero;
		this.editor = editor;
		this.doc = doc;
	}

	/**
	 * Retorna el identificador del recurso compartido.
	 *
	 * @return String que representa de manera única al recurso.
	 */
	public String getID() {
		return fichero.getFullPath().removeFirstSegments(1).toString();
	}

	/**
	 * Establece la referencia al editor del recurso compartido.
	 *
	 * @param editor
	 *          Referencia al editor del documento o null si se quiere marcar
	 *          que no está siendo editado
	 */
	public void setEditor(IEditorPart editor) {
		this.editor = editor;
	}

	/**
	 * Obtiene el editor del recurso representado por el objeto.
	 *
	 * @return Una referencia a IEditorPart, o null si no se está editando.
	 */
	public IEditorPart getEditor() {
		return editor;
	}

	/**
	 * Establece el IFile del recurso asociado al recurso compartido.
	 *
	 * @param fichero
	 *          Referencia a IFile.
	 */
	public void setFile(IFile fichero) {
		this.fichero = fichero;
	}

	/**
	 * Obtiene el recurso IFile asociado al recurso compartido.
	 *
	 * @return Referencia al IFile del recurso.
	 */
```

```
	public IFile getFile() {
		return fichero;
	}

	/**
	 * Establece el listener del código del documento.
	 *
	 * @param listener
	 *          Referencia al IDocumentListener del código del documento, o
	 *          null se quiere asociar.
	 */
	public void setListener(IDocumentListener listener) {
		this.listener = listener;
	}

	/**
	 * Retorna el listener asociado al documento compartido.
	 *
	 * @return Referencia al IDocumentListener o null si no tiene.
	 */
	public IDocumentListener getListener() {
		return listener;
	}

	/**
	 * Establece el IDocument asociado al recurso cuandno está siendo editado.
	 *
	 * @param doc
	 *          Referencia al IDocument asociado al recurso compartido o null
	 *          si no está siendo editado.
	 */
	public void setDocument(IDocument doc) {
		this.doc = doc;
	}

	/**
	 * Retorna la referencia al IDocument que está estructurando el recurso en
	 * coedición.
	 *
	 * @return Referencia a IDocument o null si no está siendo editado.
	 */
	public IDocument getDocument() {
		return doc;
	}

	/**
	 * Retorna la información básica del recurso compartido eintercambiable
	 * entre servidor y cliente.
	 *
	 * @return referencia al objeto RecursoCompartido creado.
	 */
	public RecursoCompartido getRecursoCompartidoInfo() {
		return new RecursoCompartido(getID(), fichero.getName(), fichero
				.getFullPath().removeFirstSegments(1).removeLastSegments(1)
				.toString(), doc.get());
	}

}
```

Clase `UtilidadesProyecto.java`:

```
package ecol.comun;

import java.io.ByteArrayInputStream;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IProjectDescription;
import org.eclipse.core.resources.ProjectScope;
import org.eclipse.core.resources.ResourcesPlugin;
```

```
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.preferences.IEclipsePreferences;
import org.eclipse.core.runtime.preferences.IScopeContext;
import org.eclipse.jdt.core.IClasspathEntry;
import org.eclipse.jdt.core.IJavaProject;
import org.eclipse.jdt.core.JavaCore;
import org.eclipse.jdt.core.JavaModelException;
import org.osgi.service.prefs.BackingStoreException;

public class UtilidadesProyecto {

	/**
	 * Cadena que representa el scope del cliente para almacenar preferencias.
	 */
	public static final String SCOPE_CLIENTE = "ecol.cliente";

	/**
	 * Cadena que representa el identificador de la naturaleza de los proyectos
	 * servidor.
	 */
	public static final String NATURE_SERVIDOR = "ecol.comun.natures.servernature";

	/**
	 * Cadena que representa el identificador de la naturaleza de los proyectos
	 * cliente.
	 */
	public static final String NATURE_CLIENTE = "ecol.comun.natures.clientnature";

	/**
	 * Cadena que representa el scope del servidor para almacenar preferencias.
	 */
	public static final String SCOPE_SERVIDOR = "ecol.servidor";

	/**
	 * Cadena que representa el identificador del valor nombre corto de usuario
	 */
	public static final String KEY_NOMBRECORTO_USUARIO = "nombreCorto";

	/**
	 * Cadena que representa el identificador del valor nombre personal de
	 * usuario
	 */
	public static final String KEY_NOMBRE_USUARIO = "nombre";

	/**
	 * Cadena que representa el identificador del valor email del usuario.
	 */
	public static final String KEY_EMAIL_USUARIO = "email";

	/**
	 * Cadena que representa el identificador del valor información adicional
	 * del usuario.
	 */
	public static final String KEY_OTRAINFO_USUARIO = "otrainfo";

	/**
	 * Cadena que representa el identificador del valor login de conexión del
	 * usuario o proyecto.
	 */
	public static final String KEY_LOGIN_PAREJA = "loginPareja";

	/**
	 * Cadena que representa el identificador del valor password de conexión del
	 * usuario o proyecto.
	 */
	public static final String KEY_PASS_PAREJA = "passPareja";

	/**
	 * Cadena que representa el identificador del valor host de conexión del
	 * proyecto.
```

```
	 */
	public static final String KEY_HOST_FUENTE = "hostFuente";

	/**
	 * Borra un determinado fichero de la estructura del proyecto.
	 *
	 * @param nombreProyecto
	 *          Nombre del proyecto donde queremos borrar.
	 * @param path
	 *          Ruta del recurso que deseamos borrar.
	 * @param nombre
	 *          Nombre del recurso que se requiere borrar.
	 * @return Verdadero o falso dependiendo si ha ido bien o no
	 * @throws CoreException
	 *           Excepción lanzada si ha habido algún error.
	 */
	public static boolean borrarFichero(String nombreProyecto, String path,
			String nombre) throws CoreException {

		IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
				.getProject(nombreProyecto);

		IFile fichero = proyecto.getFolder(path).getFile(nombre);
		if (!fichero.exists())
			return true;
		else {
			fichero.delete(true, null);
			return true;
		}
	}

	/**
	 * Se encarga de comprobar la validez de un path, y caso de no ser válido
	 * crearlo.
	 *
	 * @param proyecto
	 *          Proyecto donde queremos comprobar el path.
	 * @param path
	 *          Path que deseamos comprobar.
	 * @throws CoreException
	 *           Excepción lanzada si huvo algún problema al crear los
	 *           segmentos del path.
	 */
	private static void comprobarPath(IProject proyecto, String path)
			throws CoreException {
		String segmentos[] = path.split("/");
		String pathComprobado = "";
		IFolder folder = proyecto.getFolder(segmentos[0]);
		if (!folder.exists())
			folder.create(false, true, null);
		pathComprobado += segmentos[0] + "/";
		for (int i = 1; i < segmentos.length; i++) {
			folder = proyecto.getFolder(pathComprobado + segmentos[i]);
			if (!folder.exists())
				folder.create(false, true, null);
			pathComprobado += segmentos[i] + "/";
		}
	}

	/**
	 * Crea un fichero en un proyecto.
	 *
	 * @param nproyecto
	 *          Nombre del proyecto donde deseamos crearlo.
	 * @param nfichero
	 *          Nombre completo del fichero.
	 * @throws CoreException
	 *           Excepción lanzada si algo ha ido mal.
	 */
	public static void crearFichero(String nproyecto, String nfichero)
			throws CoreException {
```

```
		IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
				.getProject(nproyecto);
		IFile fichero = proyecto.getFile(nfichero);
		fichero.create(null, false, null);
	}

	/**
	 * Crea un fichero.
	 *
	 * @param nproyecto
	 *          Nombre del proyecto donde queremos crear el recurso.
	 * @param path
	 *          Ruta del fichero que deseamos crear.
	 * @param nfichero
	 *          Nombre del ficheor que queremos crar.
	 * @throws CoreException
	 *           Excepción lanzada si huvo algún problema en al creación.
	 */
	public static IFile crearFichero(String nproyecto, String path,
			String nfichero, String contenido) throws CoreException {
		IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
				.getProject(nproyecto);
		comprobarPath(proyecto, path);
		IFile fichero = proyecto.getFolder(path).getFile(nfichero);
		fichero.create(new ByteArrayInputStream(contenido.getBytes()), false,
				null);
		return fichero;
	}

	/**
	 * Obtiene una preferencia almacenada en un proyecto.
	 *
	 * @param proyecto
	 *          Referencia al proyecto donde queremos obtener la preferencia.
	 * @param scope
	 *          Scope de almacenaje de la preferencia del proyecto.
	 * @param key
	 *          Identificador válido de una propiedad. Están definidos en
	 *          UtilidadesProyecto.KEY_*
	 * @return String que representa la propiedad buscada o null si no existía o
	 *        no estaba establecida.
	 */
	public static String getPropiedadString(IProject proyecto, String scope,
			String key) {
		IScopeContext scopeProject = new ProjectScope(proyecto);
		IEclipsePreferences pluginNode = scopeProject.getNode(scope);
		return pluginNode.get(key, null);
	}

	/**
	 * Obtiene una preferencia almacenada en un proyecto.
	 *
	 * @param nproyecto
	 *          Nombre del proyecto donde queremos obtener la preferencia.
	 * @param scope
	 *          Scope de almacenaje de la preferencia del proyecto.
	 * @param key
	 *          Identificador válido de una propiedad. Están definidos en
	 *          UtilidadesProyecto.KEY_*
	 * @return String que representa la propiedad buscada o null si no existía o
	 *        no estaba establecida.
	 */
	public static String getPropiedadString(String nproyecto, String scope,
			String key) {
		IProject proyecto = getProject(nproyecto);
		return getPropiedadString(proyecto, scope, key);
	}

	/**
	 * Retorna el IProject asociado a un nombre de proyecto.
	 *
```

```
	 * @param nombre
	 *          String que contiene el nombre de proyecto a localizar.
	 * @return Referencia al IProject del proyecto buscado.
	 */
	public static IProject getProject(String nombre) {
		return ResourcesPlugin.getWorkspace().getRoot().getProject(nombre);
	}

	/**
	 * Sobreescribe el Classpath de un proyecto con una entrada del contenedor
	 * de la JRE, además de una entada de código fuente para el directorio src
	 * del proyecto
	 *
	 * @param project
	 *          Proyecto que deseamos modificar su classpath.
	 * @throws JavaModelException
	 *           Excepción lanzada si huvo algún problema al establecer el
	 *           classpath.
	 */
	public static void setEntradasClasspath(IProject project)
			throws JavaModelException {
		IJavaProject ijp = JavaCore.create(project);
		IClasspathEntry cpath[] = new IClasspathEntry[2];
		cpath[0] = JavaCore.newSourceEntry(new Path("/" + project.getName()
				+ "/src"));
		cpath[1] = JavaCore.newContainerEntry(new Path(
				"org.eclipse.jdt.launching.JRE_CONTAINER"), false);
		ijp.setRawClasspath(cpath, null);
	}

	/**
	 * Establece la naturaleza en la descripción de un proyecto. No sobreescribe
	 * las naturalezas ya establecidas, simplemente se van agregando.
	 *
	 * @param proyecto
	 *          Referencia al IProject que deseamos modificar.
	 * @param nature
	 *          Identificador de la naturaleza a agregar. Para los propios del
	 *          entorno colaborativo están definidos en
	 *          UtilidadesProyecto.NATURE_*
	 * @throws CoreException
	 */
	public static void setNature(IProject proyecto, String nature)
			throws CoreException {
		IProjectDescription description = proyecto.getDescription();
		// Clonamos los actuales y hacemos sitio para uno más.
		String[] natures = description.getNatureIds();
		String[] newNatures = new String[natures.length + 1];
		System.arraycopy(natures, 0, newNatures, 0, natures.length);
		newNatures[natures.length] = nature; // Aquí asignamos la nature
		// pasada.
		description.setNatureIds(newNatures);
		proyecto.setDescription(description, null);
	}

	/**
	 * Establece una preferencia concreta en un proyecto según el mecanismo de
	 * Eclipse.
	 *
	 * @param proyecto
	 *          Referencia IProject del proyecto donde queremos establecer la
	 *          preferencia.
	 * @param scope
	 *          Scope de almacenaje de la preferencia del proyecto.
	 * @param key
	 *          Identificador válido de una propiedad. Están definidos en
	 *          UtilidadesProyecto.KEY_*
	 * @param value
	 *          String que representa el valor a almacenar.
	 * @throws BackingStoreException
	 *           Excepción lanzada si se produce algún error de
```

```
	 *			almacenamiento.
	 */
	public static void setPropiedadString(IProject proyecto, String scope,
				String key, String value) throws BackingStoreException {
		IScopeContext projectScope = new ProjectScope(proyecto);
		IEclipsePreferences projectNode = projectScope.getNode(scope);
		projectNode.put(key, value);
		projectScope.getNode(scope).flush();
	}

	/**
	 * Método encargado de vaciar una determinada carpeta de un proyecto dado.
	 *
	 * @param proyecto
	 *			Nombre del proyecto donde queremos vaciar la carpeta
	 * @param string
	 *			Nombre de la carpeta a vaciar.
	 */
	public static void vaciarCarpeta(String proyecto, String string) {
		IProject mProyecto = getProject(proyecto);
		try {
			mProyecto.getFolder(string).delete(true, null);
		} catch (CoreException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
		try {
			mProyecto.getFolder(string).create(true, true, null);
		} catch (CoreException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

}
```

Clase `UtilidadesSesion.java`:

```
package ecol.comun;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;

/**
 * Esta clase contiene campos constantes utilizados en el funcionamiento interno
 * del sistema, así como métodos estáticos para tareas habituales.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class UtilidadesSesion {

	/**
	 * Prefijo usado al registra el interfaz de conexión de cliente.
	 */
	public static final String PREFIJO_INT_CONEXION = "Pareja";

	/**
	 * Prefijo usado al registra el interfaz de recursos de cliente.
	 */
	public static final String PREFIJO_INT_RECURSOS = "Recursos";

	/**
	 * Prefijo usado al registra el interfaz de mensajes de cliente.
	 */
	public static final String PREFIJO_INT_MENSAJES = "Mensajes";
```

```
	/**
	 * Nombre usado al registra el interfaz de conexión de la fuente.
	 */
	public static final String INT_SERVIDOR = "FuenteProyecto";

	/**
	 * Nombre usado al registra el interfaz de recursos de la fuente.
	 */
	public static final String INT_RECURSOS_SERVIDOR = "RecursosProyecto";

	/**
	 * Nombre usado al registra el interfaz de mensajes de la fuente.
	 */
	public static final String INT_MENSAJES_SERVIDOR = "MensajesProyecto";

	/**
	 * Obtiene el IProject, si se trata de un recursos del espacio de trabajo,
	 * de la selección pasada como parámetro.
	 *
	 * @param seleccion
	 *          Rerencia a la selección de la que queremos extrae el IProject.
	 * @return IProject asociado al recurso seleccionado, o null si no tiene
	 *        IProject.
	 */
	public static IProject getProyectoSeleccionado(ISelection seleccion) {
		if (!(seleccion instanceof IStructuredSelection))
			return null;
		IStructuredSelection ss = (IStructuredSelection) seleccion;
		Object element = ss.getFirstElement();
		if (element instanceof IResource)
			return ((IResource) element).getProject();
		if (!(element instanceof IAdaptable))
			return null;
		IAdaptable adaptable = (IAdaptable) element;
		Object adapter = adaptable.getAdapter(IResource.class);
		IResource recurso = (IResource) adapter;
		return recurso.getProject();
	}

	/**
	 * Obtiene el IResource, si se trata de un recursos del espacio de trabajo,
	 * de la selección pasada como parámetro.
	 *
	 * @param seleccion
	 *          Rerencia a la selección de la que queremos extrae el
	 *          IResource.
	 * @return IResource asociado al recurso que se extrae de la selección.
	 */
	public static IResource getRecursoSeleccionado(ISelection seleccion) {
		if (!(seleccion instanceof IStructuredSelection))
			return null;
		IStructuredSelection ss = (IStructuredSelection) seleccion;
		Object element = ss.getFirstElement();
		if (element instanceof IFile)
			return ((IResource) element);
		if (!(element instanceof IAdaptable))
			return null;
		IAdaptable adaptable = (IAdaptable) element;
		Object adapter = adaptable.getAdapter(IResource.class);
		IResource recurso = (IResource) adapter;
		return recurso;
	}
}
```

Paquete `ecol.comun.comunicaciones`:

Interfaz `IDisplayMensajes.java`:

```
package ecol.comun.comunicaciones;
```

```
/**
 * Interfaz que deben implementar todos los objetos que deseen ser usados como
 * visor de mensajes.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IDisplayMensajes {

	/**
	 * Método invocado cuando llega un nuevo mensaje de chat al sistema.
	 *
	 * @param mensaje
	 *		Mensaje de chat encapsulado en un objeto MensajeChat.
	 */
	void nuevoMensaje(MensajeChat mensaje);


	/**
	 * Método invocado cuando llega un mensaje del sistema (Por ejemplo la
	 * conexión de un usuario.
	 *
	 * @param mensaje
	 *		String que representa el mensaje.
	 */
	void mensajeSistema(String mensaje);

}
```

Interfaz `IServicioComunicaciones.java`:

```
package ecol.comun.comunicaciones;


import java.rmi.Remote;
import java.rmi.RemoteException;


/**
 * Interfaz que será mediante la cual los usuarios se comuniquen entre si.
 * Extiende java.rmi.Remote para permitir ser implementada como objeto remoto.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IServicioComunicaciones extends Remote {


	/**
	 * Método invocado por el usuario remoto para enviarnos un mensaje.
	 *
	 * @param mensaje Objeto que encapsula el mensaje.
	 * @throws RemoteException Excepción lanzada ante problemas de comunicación RMI.
	 */
	public void enviarMensaje(MensajeChat mensaje) throws RemoteException;

}
```

Clase `MensajeChat.java`:

```
package ecol.comun.comunicaciones;


import java.io.Serializable;

/**
 * Clase que encapsula un mensaje de chat entre los usuarios.
 * @author Luis Fernández Álvarez
 *
 */
```

```
public class MensajeChat implements Serializable{

	private String mensaje;

	private String usuario;


	/**
	 * Construye un nuevo mensaje de chat con un texto y un identificador del usuario que lo envía.
	 * @param mensaje
	 * @param usuario
	 */
	public MensajeChat(String mensaje, String usuario) {
		this.mensaje = mensaje;
		this.usuario = usuario;
	}


	/**
	 * Retorna el texto del mensaje enviado.
	 * @return String del mensaje encapsulado.
	 */
	public String getMensaje() {
		return mensaje;
	}


	/**
	 * Retorna el usuario que envió el mensaje.
	 * @return String que contiene el nombre del usuario.
	 */
	public String getUsuario() {

		return usuario;

	}


	/**
	 * Método que genera el mensaje de una manera legible y formateada.
	 * @return String con el mensaje y el usuario.
	 */
	public String toString() {
		return "<"+usuario+"> "+mensaje;
	}

}
```

Clase `ServicioComunicacionesImpl.java`:

```
package ecol.comun.comunicaciones;

import java.io.ByteArrayInputStream;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.Date;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.ui.IWorkbenchWindow;

import ecol.comun.ParejaProgramacion;

/**
 * Clase que implementa la interfaz de comunicaciones para dar soporte al
 * sistema del intercambio de mensajes entre usuarios y la creación de un
 * registro de los mismos.
 *
 * @author Luis Fernández Álvarez
```

```
 *
 */
public class ServicioComunicacionesImpl implements IServicioComunicaciones {
	private IServicioComunicaciones pareja;

	private IFile logChat;

	private IDisplayMensajes displayMensajes;

	private IWorkbenchWindow window;

	private ParejaProgramacion personal;

	/**
	 * Crea el controlador de comunicaciones del sistema.
	 *
	 * @param window
	 *          Ventana sobre la qeu trabajará el controlador de
	 *          comunicaciones.
	 * @param personal
	 *          Información del usuario local del sistema.
	 * @param nombreProyecto
	 *          Nombre del proyecto sobre el que se está trabajando.
	 * @throws RemoteException
	 *           Excepción lanzada ante problemas RMI.
	 */
	public ServicioComunicacionesImpl(IWorkbenchWindow window,
			ParejaProgramacion personal, String nombreProyecto)
			throws RemoteException {

		logChat = ResourcesPlugin.getWorkspace().getRoot().getProject(
				nombreProyecto).getFile("sesiones/chat.log");
		if (!logChat.exists())
			try {
				logChat.create(new ByteArrayInputStream("".getBytes()), false,
						null);
			} catch (CoreException e) {
				logChat = null;
			}

		if (logChat != null)
			log("Sesion iniciada: " + new Date().toGMTString() + "\n");
		UnicastRemoteObject.exportObject(this);
		this.personal = personal;

		this.window = window;
	}

	/**
	 * Método invocado cuando se quiere anunciar un mensaje importante para el
	 * sistema.
	 *
	 * @param mensaje
	 *          String que representa el mensaje del sistema.
	 */
	public void mensajeSistema(final String mensaje) {
		if (logChat != null)
			log("**" + mensaje + "\n");
		if (getDisplayMensajes() != null) {
			window.getWorkbench().getDisplay().asyncExec(new Runnable() {
				public void run() {
					getDisplayMensajes().mensajeSistema(mensaje);
				}
			});
		}
	}

	/**
	 * Se encarga de almacenar en el log los mensajes y demás enventos del
	 * sistema.
	 *
```

```
	 * @param cadena
	 *          String que representa la cadena a almacenar.
	 */
	public void log(String cadena) {
		try {
			logChat.appendContents(new ByteArrayInputStream(cadena.getBytes()),
					false, false, null);
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	/**
	 * Establece la pareja correspondiente a las comunicaciones en el lado del
	 * otro usuario.
	 *
	 * @param isc
	 *          Objeto que implementa la interfaz de comunicaciones.
	 */
	public void setParejaServicio(IServicioComunicaciones isc) {
		pareja = isc;
	}

	/**
	 * Método invocado cuando se desea liberar los recursos de comunicación
	 * asociados.
	 *
	 */
	public void terminar() {

	}

	/**
	 * Método invocado por el visor de mensajes para enviar un mensaje al otro
	 * miembro del trabajo colaborativo.
	 *
	 * @param texto
	 *          Texto a enviar.
	 */
	public void enviarMensajeAPareja(String texto) {
		MensajeChat mensaje = new MensajeChat(texto, personal.getLogin());
		if (logChat != null)
			log(mensaje.toString() + "\n");

		if (displayMensajes != null)
			displayMensajes.nuevoMensaje(mensaje);

		if (pareja != null) {
			try {
				pareja.enviarMensaje(mensaje);
			} catch (RemoteException e) {
				// TODO Mejorar esta excepción, abortar quizá
				e.printStackTrace();
			}
		}
	}

	/**
	 * Establece el visor de mensajes del sistema.
	 *
	 * @param vista
	 *          Referencia al visor que implementa IDisplayMensajes
	 */
	public void setDisplayMensajes(IDisplayMensajes vista) {
		displayMensajes = vista;
	}

	/**
	 * Obtiene el objeto que está funcionando de visor de mensajes.
	 *
	 * @return IDisplayMensajes o null si no hay ninguno establecido.
```

```
	 */
	public IDisplayMensajes getDisplayMensajes() {
		return displayMensajes;
	}

	/**
	 * Implementación del método invocado por el usuario remoto para enviarnos
	 * un mensaje.
	 *
	 * @param mensaje
	 *          Objeto que encapsula el mensaje.
	 * @throws RemoteException
	 *           Excepción lanzada ante problemas de comunicación RMI.
	 */
	public void enviarMensaje(final MensajeChat mensaje) throws RemoteException {
		if (logChat != null)
			log(mensaje.toString() + "\n");

		if (displayMensajes != null) {
			window.getShell().getDisplay().asyncExec(new Runnable() {

				public void run() {
					displayMensajes.nuevoMensaje(mensaje);
				}
			});
		}
	}

}
```

Paquete `ecol.comun.dialogos`:

Clase `NuevaAnotacionDialog.java`:

```
package ecol.comun.dialogos;

import org.eclipse.core.resources.IMarker;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Combo;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.Text;

import ecol.comun.Anotacion;

/**
 * Clase que compone e inicializa el dialogo de usuario encargado
 * de tomar los datos para las nuevas anotaciones del sistema.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class NuevaAnotacionDialog extends Dialog{
	private Text texto;
	private Combo prioridad;
	private Anotacion anotacion;

	/**
	 * Crea un nuevo dialogo de crear anotación.
	 * @param parentShell Shell padre del dialogo.
	 */
	public NuevaAnotacionDialog(Shell parentShell) {
		super(parentShell);
	}

	/**
```

```
	 * Crea la parte superior del cuadro de dialogo (por encima de la barra de
	 * botones).
	 *
	 * @param parent
	 *          Composite que representa el area donde se contendrá el cuadro
	 *          de dialogo.
	 * @return Referencia al controlador del area de dialogo.
	 */
	protected Control createDialogArea(Composite parent) {
		anotacion=null;
		Composite composite = (Composite) super.createDialogArea(parent);
		GridLayout layout=new GridLayout();
		layout.numColumns=2;
		layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
1;
		layout.verticalSpacing = 5;
		composite.setLayout(layout);

		Label lbTexto=new Label(composite, SWT.NONE);
		lbTexto.setText("Texto anotacion: ");
		texto = new Text(composite, SWT.SINGLE | SWT.BORDER);
		texto.setText("");
		GridData data = new GridData();
		data.horizontalAlignment = GridData.FILL;
		data.grabExcessHorizontalSpace = true;
		texto.setLayoutData(data);

		lbTexto=new Label(composite, SWT.NONE);
		lbTexto.setText("Prioridad: ");
		prioridad=new Combo(composite,SWT.DROP_DOWN|SWT.READ_ONLY);
		prioridad.add("Baja", IMarker.PRIORITY_LOW);
		prioridad.add("Normal",IMarker.PRIORITY_NORMAL);
		prioridad.add("Alta",IMarker.PRIORITY_HIGH);
		prioridad.select(IMarker.PRIORITY_LOW);
		data = new GridData();
		data.horizontalAlignment = GridData.FILL;
		data.grabExcessHorizontalSpace = true;
		prioridad.setLayoutData(data);


		return composite;
	}

	/**
	 * Configura el shell en el que se dispondrá el cuadro de dialogo.
	 * principalmente se inicializa el título y demás parametros de aspecto.
	 *
	 * @param newShell
	 *          Shell en el que se dispondrá el cuadro de dialogo.
	 */
	protected void configureShell(Shell newShell) {
		super.configureShell(newShell);
		newShell.setText("Nueva anotacion");
	}

	/**
	 * Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo
	 * creado.
	 */
	protected void okPressed() {
		anotacion=new Anotacion(texto.getText(),prioridad.getSelectionIndex(),false);
		this.close();
	}

	/**
	 * retorna la anotación creada en el dialogo.
	 * @return Anotación que encapsula la anotación creada por el usuario.
	 */
	public Anotacion getAnotacion(){
		return anotacion;
	}
```

```
}
```

Paquete ecol.comun.excepciones:

Clase `AutenticacioInvalidaException.java`:

```
package ecol.comun.excepciones;

/**
 * Excepción lanzada por parte del usuario fuente al usuario cliente cuando los
 * datos de validación del usuario no son correctos
 *
 * @author Luis Fernández Álvarez
 *
 */
public class AutenticacioInvalidaException extends Exception {

	/**
	 * Crea una nueva excepción
	 *
	 * @param mensaje
	 *		Mensaje asociado a la excepción.
	 */
	public AutenticacioInvalidaException(String mensaje) {

		super(mensaje);

	}

}
```

Clase `ConexionParejaException.java`:

```
package ecol.comun.excepciones;

/**
 * Excepción lanzada por parte del usuario fuente al usuario cliente cuando
 * el servidor no puede establecer una conexión de vuelta con el cliente al
 * iniciar la sesión.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ConexionParejaException extends Exception{

	/**
	 * Crea una nueva excepción
	 *
	 * @param string
	 *		Mensaje asociado a la excepción.
	 */
	public ConexionParejaException(String string) {
		super(string);
	}


}
```

Clase `ParejaEstablecidaException.java`:

```
package ecol.comun.excepciones;


/**
 * Excepción lanzada por parte del usuario fuente al usuario cliente cuando ya
 * hay una pareja establecida en el sistema
 *
 * @author Luis Fernández Álvarez
 *
```

```
 */
public class ParejaEstablecidaException extends Exception {


	/**
	 * Crea una nueva excepción
	 *
	 * @param mensaje
	 *          Mensaje asociado a la excepción.
	 */
	public ParejaEstablecidaException(String mensaje) {
		super(mensaje);
	}

	/**
	 * Retorna el mensaje de la excepción.
	 * @return String que contiene el mensaje de la excepción.
	 */
	public String getMensaje() {

		return getMessage();

	}

}
```

Paquete **`ecol.comun.natures:`**

Clase `ClienteEcolNature.java`:

```
package ecol.comun.natures;

import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IProjectNature;
import org.eclipse.core.runtime.CoreException;

/**
 * Clase que implementa el comportamiento de la naturaleza de los
 * proyectos colaborativos cliente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ClienteEcolNature implements IProjectNature {

	/**
	 * Configura esta naturaleza para su proyecto.
	 * Este método es llamado cuando se establece una naturaleza
	 * a un proyecto.
	 */
	public void configure() throws CoreException {
	}

	/**
	 * Elimina la configuración de esta naturaleza para el proyecto.
	 * Este método es llamado cuando se elimina una naturaleza
	 * a un proyecto.
	 */
	public void deconfigure() throws CoreException {

	}

	/**
	 * Retorna el proyecto al cual esta naturaleza se aplica.
	 */
	public IProject getProject() {
		return null;
	}

	/**
```

```
	 * Establece el proyecto en el cual se aplicará esta naturaleza.
	 * @param project referencia al IProject sobre el que se aplica la naturaleza
	 */
	public void setProject(IProject project) {

	}

}
```

Clase `ServidorEcolNature.java`:

```
package ecol.comun.natures;

import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IProjectNature;
import org.eclipse.core.runtime.CoreException;

/**
 * Clase que implementa el comportamiento de la naturaleza de los
 * proyectos colaborativos fuente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ServidorEcolNature implements IProjectNature {

	/**
	 * Configura esta naturaleza para su proyecto.
	 * Este método es llamado cuando se establece una naturaleza
	 * a un proyecto.
	 */
	public void configure() throws CoreException {
	}

	/**
	 * Elimina la configuración de esta naturaleza para el proyecto.
	 * Este método es llamado cuando se elimina una naturaleza
	 * a un proyecto.
	 */
	public void deconfigure() throws CoreException {

	}

	/**
	 * Retorna el proyecto al cual esta naturaleza se aplica.
	 */
	public IProject getProject() {
		return null;
	}

	/**
	 * Establece el proyecto en el cual se aplicará esta naturaleza.
	 * @param project referencia al IProject sobre el que se aplica la naturaleza
	 */
	public void setProject(IProject project) {

	}

}
```

Paquete `ecol.servidor`:

Clase `EcolServidor.java`:

```
package ecol.servidor;

import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import org.eclipse.jface.dialogs.MessageDialog;
```

```
import org.eclipse.ui.IWorkbenchWindow;

import ecol.servidor.sesion.ControladorSesionServidor;

/**
 * Esta clase sirve al plugin para controlar el modelo de trabajo, desde aquí se
 * solicita el comienzo de una sesión de trabajo sobre un proyecto determinado y
 * su terminación.
 * @author Luis Fernández Álvarez
 */
public class EcolServidor {
	private static ControladorSesionServidor ctrlSesion;
	private static Registry regRMI=null;
	private static IWorkbenchWindow window;

	/**
	 * Obtiene la referencia a la ventana de trabajo
	 * @return Devuelve una referencia a IWorkbenchWindow que reprsesenta la ventana de
trabajo.
	 */
	public static IWorkbenchWindow getVentanaTrabajo() {
		return window;
	}

	/**
	 * Comienza una sesión de trabajo colaborativo.
	 * @param name Nombre del proyecto sobre el que queremos trabajar.
	 * @param window Ventana de trabajo sobre la que stamos estableciendo la sesión
	 * @throws RemoteException Excepción lanzada ante problemas RMI
	 */
	public static void setWorkingProject(String name, IWorkbenchWindow window)
			throws RemoteException
	{

		if(regRMI==null)
			regRMI = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);

		ControladorSesionServidor ctrlTemp = new ControladorSesionServidor(
				name, window, regRMI);
		ctrlSesion = ctrlTemp;
		EcolServidor.window = window;

	}

	/**
	 * Obtiene el controlador de sesión de trabajo actual.
	 * @return Retorna el controlador de la sesión
	 */
	public static ControladorSesionServidor getControladorSesion() {
		return ctrlSesion;
	}

	/**
	 * Nos permite determinar si se ha iniciado sesión de trabajo con algun
	 * proyecto.
	 *
	 * @return devuelve verdadero o falso dependiendo si está iniciado o no.
	 */
	public static boolean estaIniciadaSesion() {
		if (ctrlSesion == null)
			return false;
		else
			return true;
	}

	/**
	 * Método encargado de finalizar la sesión de trabajo actual.
	 *
	 */
	public static void terminarSesion() {
		ctrlSesion.terminarSesion();
```

```
                ctrlSesion = null;
                EcolServidor.window = null;
        }
    /**
     * Método invocado cuando ha habido un error irrecuperable
     * en la comunicación con el cliente. Util ante fallos RMI.
     * @param mensaje Mensaje del error insalvable.
     */
    public static void abortarSesion(String mensaje){
        MessageDialog.openError(window.getShell(),"Eclipse Colaborativo: Error Fatal","Error
irrecuperable. Se terminará la sesión de trabajo.\nERROR:\n"+mensaje);
        if(ctrlSesion!=null)
        ctrlSesion.abortarSesion();
        ctrlSesion=null;
        if(window.getActivePage().getPerspective().getId().compareTo("ecol.perspectivas.servidor")==0)
            window.getActivePage().closePerspective(window.getActivePage().getPerspective(), true,
false);
        EcolServidor.window=null;
    }
 }
```

Paquete `ecol.servidor.acciones`:

Clase `CambiarEmailAction.java`:

```
package ecol.servidor.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de email en un proyecto
 * Eclipse Colaborativo Fuente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarEmailAction implements IWorkbenchWindowActionDelegate {
        private IWorkbenchWindow window;

        private ISelection seleccion;


        /**
         * Dispose this action delegate
         */
        public void dispose() {

        }

        /**
         * Inicializa esta acción con la ventana del workbench en la que trabajará.
         *
         * @param window
         *          IWorkbenchWindow
         */
        public void init(IWorkbenchWindow window) {
                this.window = window;
        }

        /**
```

```
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_EMAIL_USUARIO);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar informacion de proyecto cliente",
				"Introduzca la modificación deseada sobre el E-Mail del proyecto
<"+proyecto.getName()+">",
				value, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
					UtilidadesProyecto.SCOPE_SERVIDOR,
					UtilidadesProyecto.KEY_EMAIL_USUARIO, dialogo
						.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
					"ECol Servidor: Error",
					"Error almacenando los cambios en el proyecto:\n"
						+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion = selection;
	}

}
```

Clase `CambiarLoginAction.java`:

```
package ecol.servidor.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

public class CambiarLoginAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;
```

```
	private ISelection seleccion;


	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_LOGIN_PAREJA);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar informacion de proyecto servidor",
				"Introduzca la modificación deseada sobre el Login de la pareja en el
proyecto <"+proyecto.getName()+">",
				value, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_LOGIN_PAREJA, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"ECol Servidor: Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion = selection;
	}

}
```

Clase `CambiarNombreAction.java`:

```
package ecol.servidor.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

public class CambiarNombreAction implements IWorkbenchWindowActionDelegate {
	private IWorkbenchWindow window;

	private ISelection seleccion;


	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String nombre = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_NOMBRE_USUARIO);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar nombre personal: ",
				"Introduzca el nombre personal que desea asociar al proyecto
seleccionado",
				nombre, null);
		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_NOMBRE_USUARIO, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"ECol Servidor: Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
```

```
                        }
                }
        }

        /**
         * Natifica a esta acción que la selección del workbench ha cambiado. Esta
         * notificación se puede emplear para cambiar la disponibilidad de la acción
         * o para modificar las propiedades de presentación de la acción.
         *
         * @param action
         *          IAction que representa la parte de la acción referente a su
         *          presentación.
         * @param selection
         *          ISelection que representa la selección actual o null si no
         *          hay.
         */
        public void selectionChanged(IAction action, ISelection selection) {
                seleccion = selection;
        }

}
```

Clase `CambiarOtraInfoAction.java`:

```
package ecol.servidor.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

/**
 * Clase asociada a la acción para cambiar el campo de información adicional en
 * un proyecto Eclipse Colaborativo Fuente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class CambiarOtraInfoAction implements IWorkbenchWindowActionDelegate {
        private IWorkbenchWindow window;

        private ISelection seleccion;

        /**
         * Dispose this action delegate
         */
        public void dispose() {

        }

        /**
         * Inicializa esta acción con la ventana del workbench en la que trabajará.
         *
         * @param window
         *          IWorkbenchWindow
         */
        public void init(IWorkbenchWindow window) {
                this.window = window;
        }

        /**
         * Realiza la acción. Este método se invoca cuando la acción es lanzada.
         *
```

```
		 * @param action
		 *          IAction manejador encargado de la parte de presentación de la
		 *          acción.
		 */
		public void run(IAction action) {
			IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
			String nombre = UtilidadesProyecto.getPropiedadString(proyecto,
					UtilidadesProyecto.SCOPE_SERVIDOR,
					UtilidadesProyecto.KEY_OTRAINFO_USUARIO);
			InputDialog dialogo = new InputDialog(
					window.getShell(),
					"Cambiar propiedades proyecto ",
					"Introduzca la modificación deseada sobre el campo 'Otra
Información' del proyecto <"
							+ proyecto.getName() + ">", nombre, null);
			int retorno = dialogo.open();
			if (retorno == Window.OK) {
				try {
					UtilidadesProyecto.setPropiedadString(proyecto,
							UtilidadesProyecto.SCOPE_SERVIDOR,
							UtilidadesProyecto.KEY_OTRAINFO_USUARIO,
dialogo
									.getValue());
				} catch (BackingStoreException e) {
					MessageDialog.openError(window.getShell(),
							"ECol Servidor: Error",
							"Error almacenando los cambios en el proyecto:\n"
									+ e.getMessage());
				}
			}
		}

		/**
		 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
		 * notificación se puede emplear para cambiar la disponibilidad de la acción
		 * o para modificar las propiedades de presentación de la acción.
		 *
		 * @param action
		 *          IAction que representa la parte de la acción referente a su
		 *          presentación.
		 * @param selection
		 *          ISelection que representa la selección actual o null si no
		 *          hay.
		 */
		public void selectionChanged(IAction action, ISelection selection) {
			seleccion = selection;
		}

}
```

Clase `CambiarPasswordAction.java`:

```
package ecol.servidor.acciones;

import org.eclipse.core.resources.IProject;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.InputDialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.window.Window;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;

public class CambiarPasswordAction implements IWorkbenchWindowActionDelegate {
		private IWorkbenchWindow window;

		private ISelection seleccion;
```

```
	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}

	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		IProject proyecto = UtilidadesSesion.getProyectoSeleccionado(seleccion);
		String value = UtilidadesProyecto.getPropiedadString(proyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_PASS_PAREJA);
		InputDialog dialogo = new InputDialog(
				window.getShell(),
				"Cambiar información de proyecto servidor",
				"Introduzca la nueva password asociada a la pareja del proyecto
<"+proyecto.getName()+">",
				value, null);

		int retorno = dialogo.open();
		if (retorno == Window.OK) {
			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_PASS_PAREJA, dialogo
								.getValue());
			} catch (BackingStoreException e) {
				MessageDialog.openError(window.getShell(),
						"ECol Servidor: Error",
						"Error almacenando los cambios en el proyecto:\n"
								+ e.getMessage());
			}
		}
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
	 */
	public void selectionChanged(IAction action, ISelection selection) {
		seleccion = selection;
	}

}
```

Clase `LanzarProyectoAction.java`:

```
package ecol.servidor.acciones;

 import org.eclipse.jface.action.IAction;
 import org.eclipse.jface.dialogs.MessageDialog;
 import org.eclipse.jface.viewers.ISelection;
 import org.eclipse.ui.IWorkbenchWindow;
 import org.eclipse.ui.IWorkbenchWindowActionDelegate;

 import ecol.servidor.EcolServidor;
 import ecol.servidor.dialogos.SeleccionProyectoDialog;

 /**
  * Implementación de la acción asociada a publicar un proyecto fuente.
  *
  * @author Luis Fernández Álvarez
  *
  */
 public class LanzarProyectoAction implements IWorkbenchWindowActionDelegate {
          private IWorkbenchWindow window;


          /**
           * Dispose this action delegate
           */
          public void dispose() {

          }

          /**
           * Inicializa esta acción con la ventana del workbench en la que trabajará.
           *
           * @param window
           *          IWorkbenchWindow
           */
          public void init(IWorkbenchWindow window) {
                  this.window = window;
          }

          /**
           * Realiza la acción. Este método se invoca cuando la acción es lanzada.
           *
           * @param action
           *          IAction manejador encargado de la parte de presentación de la
           *          acción.
           */
          public void run(IAction action) {

                  if (!EcolServidor.estaIniciadaSesion()) {
                          SeleccionProyectoDialog dialog = new SeleccionProyectoDialog(
                                          window.getShell(),window);
                          dialog.open();
                          if(dialog.isSesionIniciada()){
                                  action.setChecked(true);
                                  //Activamos la perspectiva.

          window.getActivePage().setPerspective(window.getWorkbench().getPerspectiveRegistry().findPe
rspectiveWithId("ecol.perspectivas.servidor"));
                          }else{
                                  action.setChecked(false);
                          }
                  } else{
                          String botones[]={"Si","No"};
                          MessageDialog dialogo=new MessageDialog(window.getShell(),"E-Col:
Proyecto en uso",null,"¿Desea terminar la sesión Eclipse
Colaborativo?",MessageDialog.QUESTION,botones,0);
                          if(dialogo.open()==0){
                                  //Desconectamos todos los recursos.
                                  EcolServidor.terminarSesion();
                                  action.setChecked(false);
```

```
		if(window.getActivePage().getPerspective().getId().compareTo("ecol.perspectivas.servidor")==
0)

		window.getActivePage().closePerspective(window.getActivePage().getPerspective(), true,
false);
				}else{
					action.setChecked(true);
				}
				}
		}

		/**
		 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
		 * notificación se puede emplear para cambiar la disponibilidad de la acción
		 * o para modificar las propiedades de presentación de la acción.
		 *
		 * @param action
		 *		IAction que representa la parte de la acción referente a su
		 *		presentación.
		 * @param selection
		 *		ISelection que representa la selección actual o null si no
		 *		hay.
		 */
		public void selectionChanged(IAction action, ISelection selection) {
			if(EcolServidor.estaIniciadaSesion())action.setChecked(true);
			else action.setChecked(false);
		}

}
```

Clase `NuevoAvisoProgAction.java`:

```
package ecol.servidor.acciones;

import java.rmi.RemoteException;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IAdaptable;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;

import ecol.comun.Anotacion;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;
import ecol.comun.dialogos.NuevaAnotacionDialog;
import ecol.servidor.EcolServidor;

/**
 * Clase encargada de la implementación de la acción de crear una nueva
 * anotación en los recursos del proyecto fuente.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class NuevoAvisoProgAction implements IWorkbenchWindowActionDelegate{
	private ISelection selection;
	private IWorkbenchWindow window;
	/**
	 * Dispose this action delegate
	 */
	public void dispose() {

	}
```

```
	/**
	 * Inicializa esta acción con la ventana del workbench en la que trabajará.
	 *
	 * @param window
	 *          IWorkbenchWindow
	 */
	public void init(IWorkbenchWindow window) {
		this.window = window;
	}

	/**
	 * Realiza la acción. Este método se invoca cuando la acción es lanzada.
	 *
	 * @param action
	 *          IAction manejador encargado de la parte de presentación de la
	 *          acción.
	 */
	public void run(IAction action) {
		if(!EcolServidor.estaIniciadaSesion())return;

		IResource recurso = UtilidadesSesion.getRecursoSeleccionado(selection);
		if(recurso instanceof IFile){
			IFile fichero=(IFile)recurso;
			//Tenemos el fichero, comprobamos que está en el proyecto con ele que
estamos trabajando.
			boolean esPServidor=false;
			String nombreProyecto=recurso.getProject().getName();
			String enTrabajo=EcolServidor.getControladorSesion().getNProyecto();
			if(nombreProyecto.compareTo(enTrabajo)==0)esPServidor=true;
			if(esPServidor){
				try {
					NuevaAnotacionDialog dialogo=new
NuevaAnotacionDialog(window.getShell());
					dialogo.open();
					Anotacion resultado=dialogo.getAnotacion();
					if(resultado!=null){


EcolServidor.getControladorSesion().getControladorRecursos().nuevaAnotacion(fichero,resultado);
					}


				} catch (CoreException e) {
					MessageDialog.openError(window.getShell(), "Error
asignando anotaciÛn", "Esta asignando el marcador al fichero: "+e.getMessage());
				} catch (RemoteException e) {
					EcolServidor.abortarSesion("Se ha producido un error al
establece la anotacion en los recursos de la pareja: "+e.getMessage());

				}
			}else{
				MessageDialog.openError(window.getShell(), "Error asignando
anotación", "Esta acción de anotación sólo es válida sobre el proyecto Servidor Eclipse dPP sobre el que
se est· trabajando.");
			}
		}//Else... no se debería venir puesto que en el plugin.xml hacemos un enablement solo
a IFile's
	}

	/**
	 * Natifica a esta acción que la selección del workbench ha cambiado. Esta
	 * notificación se puede emplear para cambiar la disponibilidad de la acción
	 * o para modificar las propiedades de presentación de la acción.
	 *
	 * @param action
	 *          IAction que representa la parte de la acción referente a su
	 *          presentación.
	 * @param selection
	 *          ISelection que representa la selección actual o null si no
	 *          hay.
```

```
		 */
		public void selectionChanged(IAction action, ISelection selection) {
			this.selection=selection;
			IResource recurso = UtilidadesSesion
			.getRecursoSeleccionado(this.selection);
			if((recurso!=null)&&(recurso instanceof IFile)&&(EcolServidor.estaIniciadaSesion())){
				action.setEnabled(true);
			}else{
				action.setEnabled(false);
			}
		}

}
```

Paquete `ecol.servidor.dialogos`:

Clase `SeleccionProyectoDialog.java`:

```
package ecol.servidor.dialogos;

import java.rmi.RemoteException;
import java.util.LinkedList;

import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.dialogs.Dialog;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.IStructuredContentProvider;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.TableViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Shell;
import org.eclipse.swt.widgets.TableItem;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PlatformUI;

import ecol.servidor.EcolServidor;

/**
 * Clase que representa el cuadro de dialogo que se le muestra al usuario cuando
 * selecciona el proyecto que desea para iniciar la sesión de trabajo de
 * colaboración.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class SeleccionProyectoDialog extends Dialog {

	private IWorkbenchWindow window;

	private TableViewer viewer;

	private boolean sesionIniciada;

	/**
	 * Clase que se encarga de llevar el control del contenido de la tabla de
	 * proyectos disponibles en el entorno de trabajo.
	 *
	 * @author Luis Fernández Álvarez
	 *
	 */
```

```
class ViewContentProvider implements IStructuredContentProvider {
	private LinkedList listaProyectos = new LinkedList();

	/**
	 * Notifica a este proveedor de contenido de que la entrada del visor ha
	 * sido cambiada a un elemento diferente.
	 *
	 * @param v
	 *          visor en el que está este proveedor de contenido.
	 * @param oldInput
	 *          Objeto que representa la entrada anterior.
	 * @param newInput
	 *          Objeto que representa la nueva entrada.
	 */
	public void inputChanged(Viewer v, Object oldInput, Object newInput) {
	}

	/**
	 * Este método es llamado por el visor de elementos cuando se le envía
	 * el mensaje dispose.
	 */
	public void dispose() {
	}

	/**
	 * Devuelve los elementos que serán visualizados en el visor asociado.
	 *
	 * @param parent
	 *          elemento que sirve de entrada.
	 * @return Object[] Array de elementos para ser visualizados.
	 */
	public Object[] getElements(Object parent) {
		IProject proyectos[] = ResourcesPlugin.getWorkspace().getRoot()
				.getProjects();

		for (int i = 0; i < proyectos.length; i++) {
			try {
				if (proyectos[i]

.hasNature("ecol.comun.natures.servernature")) {
					listaProyectos.add(proyectos[i].getName());
				}
			} catch (CoreException e) {

			}
		}
		return listaProyectos.toArray();
	}
}

/**
 * Esta clase se encarga de proporcionar las etiquetas e identificadores
 * para los elementos de las tablas y listas de los visores de contenido.
 *
 * @author Luis Fernández Álvarez
 *
 */
class ViewLabelProvider extends LabelProvider implements
		ITableLabelProvider {

	/**
	 * Retorna la etiqueta para la columna solicitada del elemento.
	 *
	 * @param obj
	 *          es el objeto que representa la fila, o null.
	 * @param index
	 *          Índice (partiendo de 0) que indica la columna en la que la
	 *          etiqueta aparecerá.
	 * @return String o null si no hay etiqueta para la columna dada.
	 */
	public String getColumnText(Object obj, int index) {
```

```
                        return getText(obj);
                }

                /**
                 * Retorna la imagen para la columna solicitada del elemento.
                 *
                 * @param obj
                 *          es el objeto que representa la fila, o null.
                 * @param index
                 *          Índice (partiendo de 0) que indica la columna en la que la
                 *          etiqueta aparecerá.
                 * @return Image o null si no hay imagen para la columna dada.
                 */
                public Image getColumnImage(Object obj, int index) {
                        return PlatformUI.getWorkbench().getSharedImages().getImage(
                                        ISharedImages.IMG_OBJ_PROJECT);
                }
        }

        /**
         * Constructor para el cuadro de dialogo de selección de proyecto.
         *
         * @param parentShell
         *          Shell padre en la que se dispondrá dialogo.
         * @param window
         *          IWorkbenchWindow sobre la que se lanza el dialogo.
         */
        public SeleccionProyectoDialog(Shell parentShell, IWorkbenchWindow window) {
                super(parentShell);
                sesionIniciada = false;
                this.window = window;
        }

        /**
         * Crea la parte superior del cuadro de dialogo (por encima de la barra de
         * botones).
         *
         * @param parent
         *          Composite que representa el area donde se contendrá el cuadro
         *          de dialogo.
         * @return Referencia al controlador del area de dialogo.
         */
        protected Control createDialogArea(Composite parent) {

                Composite composite = (Composite) super.createDialogArea(parent);
                GridLayout layout = new GridLayout(1, true);
                GridData data = new GridData();
                data.horizontalAlignment = GridData.FILL;
                data.grabExcessHorizontalSpace = true;
                composite.setLayout(layout);
                Label enunciado = new Label(composite, SWT.NONE);
                enunciado
                                .setText("A continuación se muestran los proyectos disponibles:");
                viewer = new TableViewer(composite, SWT.SINGLE | SWT.V_SCROLL);
                viewer.setLabelProvider(new ViewLabelProvider());
                viewer.setContentProvider(new ViewContentProvider());
                viewer.setInput(composite);
                viewer.getTable().setLayoutData(data);
                return composite;
        }

        /**
         * Configura el shell en el que se dispondrá el cuadro de dialogo.
         * principalmente se inicializa el título y demás parametros de aspecto.
         *
         * @param newShell
         *          Shell en el que se dispondrá el cuadro de dialogo.
         */
        protected void configureShell(Shell newShell) {
                super.configureShell(newShell);
                newShell.setText("Proyectos disponibles");
```

```
	}

	/**
	 * Método invocado cuando se pulsa el botón de OK en el cuadro de diálogo
	 * creado.
	 */
	protected void okPressed() {
		TableItem seleccionados[] = viewer.getTable().getSelection();
		if (seleccionados != null) {
			try {
				EcolServidor.setWorkingProject(seleccionados[0].getText(),
						window);
				sesionIniciada = true;
			} catch (RemoteException e) {
				MessageDialog.openError(window.getShell(),
						"Ecol Servidor: Informe de Error",
						"Error publicando el proyecto: \n " +
e.getMessage());
				sesionIniciada = false;
			}
		}
		this.close();
	}

	/**
	 * Determina si se ha iniciado correctamente la sesión por parte del
	 * servidor.
	 *
	 * @return Verdadero si se ha iniciado la sesión correctamente o falso en
	 *	caso contrario.
	 */
	public boolean isSesionIniciada() {
		return sesionIniciada;
	}
}
```

Paquete `ecol.servidor.perspectivas`:

Clase `TrabajoServidorPerspective.java`:

```
package ecol.servidor.perspectivas;

import org.eclipse.jdt.ui.JavaUI;
import org.eclipse.ui.IFolderLayout;
import org.eclipse.ui.IPageLayout;
import org.eclipse.ui.IPerspectiveFactory;

/**
 * Clase encarga de componer la vista de trabajo del usuario fuente del
 * proyecto.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class TrabajoServidorPerspective implements IPerspectiveFactory {

	/**
	 * Crea la composición inicial de la perspectiva (vistas, acciones,...).
	 * Esta podrá ser modificada por el usuario desde las opciones del menú
	 * Window.
	 *
	 * @param layout
	 *		Representa la composición de la perspectiva.
	 */
	public void createInitialLayout(IPageLayout layout) {

		// Obtenemos el area ocupada por el editor, sobre
		// esa referencia es sobre la que trabajaremos.

		String editorArea = layout.getEditorArea();
```

```
		/*
		 * Parte derecha, Información personal y chat.
		 */
		IFolderLayout arribaDerecha = layout.createFolder("arribaDerecha",
				IPageLayout.RIGHT, 0.75f, editorArea);
		arribaDerecha.addView("ecol.vistas.servidor.InformacionParejaView");
		layout.addView("ecol.vistas.servidor.ChatServidorView",
				IPageLayout.BOTTOM, 0.7f,
				"ecol.vistas.servidor.InformacionParejaView");

		/*
		 * Parte izquierda, Paquete y Outline.
		 */

		IFolderLayout arribaIzquierda = layout.createFolder("arribaIzquierda",
				IPageLayout.LEFT, 0.35f, editorArea);

		arribaIzquierda.addView(JavaUI.ID_PACKAGES);
		layout.addView(IPageLayout.ID_OUTLINE, IPageLayout.BOTTOM, 0.4f,
				JavaUI.ID_PACKAGES);

		/*
		 * Parte abajo.
		 */
		layout.addView(IPageLayout.ID_TASK_LIST, IPageLayout.BOTTOM, 0.7f,
				editorArea);

	}

}
```

Paquete `ecol.servidor.recursos`:

Clase `CodigoFuenteListener.java`:

```
package ecol.servidor.recursos;

import org.eclipse.jface.text.DocumentEvent;
import org.eclipse.jface.text.IDocumentListener;

/**
 * Clase que implementa el IDocumentListener, encargada de mantener el control
 * de las modificaciones de los documentos sobre los que trabaja.
 *
 * @author Luis Fernández Álavarez.
 *
 */
public class CodigoFuenteListener implements IDocumentListener {

	private ControladorRecursos ctrlRecursos;

	/**
	 * Crea un nuevo listener de código fuente para posteriormente ser asociado
	 * a un documento de trabajo.
	 *
	 * @param ctrl
	 *          ControladorRecursos que será el destino de las modificaciones
	 *          recibidas por el listener.
	 */
	public CodigoFuenteListener(ControladorRecursos ctrl) {
		ctrlRecursos = ctrl;
	}

	/**
	 * The manipulation described by the document event will be performed.
	 *
	 * @param event
	 *          the document event describing the document change
	 */
	public void documentAboutToBeChanged(DocumentEvent event) {
```

```
        }

        /**
         * Método invocado cuando un documento ha sido modificado. Se utilizará para
         * mantener el control de las modificaciones en los documentos.
         *
         * @param event
         *          DocumentEvent que encapsula la modificación realizada en el
         *          método.
         */
        public void documentChanged(DocumentEvent event) {
                ctrlRecursos.modificarRecurso(event.getDocument(), event.fOffset,
                                event.fLength, event.fText);
        }

}
```

Clase `ControladorRecursos.java`:

```
package ecol.servidor.recursos;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.util.LinkedList;
import java.util.ListIterator;

import org.eclipse.core.filebuffers.FileBuffers;
import org.eclipse.core.filebuffers.ITextFileBuffer;
import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IResourceChangeEvent;
import org.eclipse.core.resources.ResourcesPlugin;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.IEditorPart;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.part.FileEditorInput;
import org.eclipse.ui.texteditor.AbstractTextEditor;
import org.eclipse.ui.texteditor.IDocumentProvider;
import org.eclipse.ui.texteditor.ITextEditor;

import ecol.cliente.recursos.IServicioRecursosCliente;
import ecol.comun.Anotacion;
import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.RecursoCompartido;
import ecol.comun.ReferenciaRecursoCompartido;
import ecol.comun.UtilidadesProyecto;
import ecol.servidor.EcolServidor;

/**
 * Esta clase se encarga de gestionar todo lo referido a los recursos del
 * proyecto fuente de colaboración. En esta clase se controla las modificaciones
 * en los recursos para notificarlas al cliente. Se controla las actuaciones del
 * usuario en cuanto a editores que ha abierto o cerrado, recursos borrados,
 * etc...
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ControladorRecursos implements IServicioRecursosServidor {
        private LinkedList listaCoEditados;

        private IServicioRecursosCliente recCliente;
```

```
	private RecursosPerspectivaListener listener;

	private IWorkbenchWindow window;

	private String contenidoTemporal;

	private RecursosChangeListener resourceListener;

	/**
	 * Crea el controlador de recursos para el proyecto fuente de la sesión de
	 * trabajo.
	 *
	 * @param rec
	 *          Referencia a los recursos de la pareja cliente.
	 * @param window
	 *          Ventana de trabajo de la sesión de trabajo.
	 * @throws RemoteException
	 *           Se lanza cuando hay errores RMI
	 */
	public ControladorRecursos(IServicioRecursosCliente rec,
			IWorkbenchWindow window) throws RemoteException {
		UnicastRemoteObject.exportObject(this);
		this.window = window;
		recCliente = rec;
		listaCoEditados = new LinkedList();
		resourceListener = new RecursosChangeListener(this);
		ResourcesPlugin.getWorkspace().addResourceChangeListener(
				resourceListener, IResourceChangeEvent.POST_CHANGE);
		listener = new RecursosPerspectivaListener(this);

		window.addPerspectiveListener(listener);
	}

	/**
	 * Método invocado cuando un recurso es borrado.
	 *
	 * @param fichero
	 *          Referencia al fichero borrado.
	 */
	public void recursoBorrado(IFile fichero) {
		try {
			recCliente.borrarRecurso(fichero.getFullPath().removeFirstSegments(
					1).toString());
		} catch (RemoteException e) {
			EcolServidor.abortarSesion("Error comunicando un borrado al usuario:
"+e.getMessage());
		}
	}

	/**
	 * Obtiene el contenido actualizado de un determinado recurso.
	 *
	 * @param fichero
	 *          Recurso del cual se quiere obtener el contenido acutalizado.
	 * @return String que contiene el código fuente del recurso.
	 */
	private String getContenido(final IFile fichero) {
		ITextFileBuffer filebuffer = FileBuffers.getTextFileBufferManager()
				.getTextFileBuffer(fichero.getFullPath());
		if (filebuffer == null) {
			StringBuffer contenido = new StringBuffer("");
			FileEditorInput input = new FileEditorInput(fichero);
			try {
				InputStreamReader stream = new InputStreamReader(fichero
						.getContents());
				BufferedReader reader = new BufferedReader(stream);
				try {
					while (reader.ready()) {
						contenido.append((char) reader.read());
					}
```

```
                                        reader.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        } catch (CoreException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        return contenido.toString();
                }
                return filebuffer.getDocument().get();
        }

        /**
         * Obtiene el recurso pasado como un objeto comunicable al usuario cliente.
         *
         * @param fichero
         *          Fichero que se desesa encapsualr
         * @return Contenido e información del recurso encapsulado.
         */
        private RecursoCompartido getArchivoEncapsulado(IFile fichero) {
                RecursoCompartido recurso = new RecursoCompartido(fichero.getFullPath()
                                .removeFirstSegments(1).toString(), fichero.getName(), fichero
                                .getFullPath().removeFirstSegments(1).removeLastSegments(1)
                                .toString(), getContenido(fichero));
                recurso.setAnotaciones(getAnotaciones(fichero));
                return recurso;
        }

        /**
         * Recorre la estructura del proyecto y retorna todos los recursos de la
         * misma.
         *
         * @param nproyecto
         *          Nombre del proyecto a recorrer.
         * @return Array con todos los recursos encapsulados.
         */
        private Object[] getEstructuraProyecto(String nproyecto) {
                LinkedList listaRecursos = new LinkedList();
                IProject proyecto = UtilidadesProyecto.getProject(nproyecto);
                IFolder src = proyecto.getFolder("src");
                try {
                        IResource recursos[] = src.members();
                        for (int i = 0; i < recursos.length; i++) {
                                if (recursos[i] instanceof IFile) {
                                        listaRecursos
                                                        .add(getArchivoEncapsulado((IFile)
recursos[i]));
                                } else {
                                        IFolder folder = (IFolder) recursos[i];
                                        recorrerPaquete(folder, listaRecursos);
                                }
                        }
                } catch (CoreException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                return listaRecursos.toArray();
        }

        /**
         * Recorre el contenido de un determinado paquete (carpeta).
         *
         * @param paquete
         *          carpeta a recorrer del proyecto.
         * @param lista
         *          lista donde ir almacenando los objetos encapsulados.
         */
        private void recorrerPaquete(IFolder paquete, LinkedList lista) {
                try {
```

```
                        IResource recursos[] = paquete.members();
                        for (int i = 0; i < recursos.length; i++) {
                                if (recursos[i] instanceof IFile) {
                                        lista.add(getArchivoEncapsulado((IFile) recursos[i]));
                                } else {
                                        IFolder folder2 = (IFolder) recursos[i];
                                        recorrerPaquete(folder2, lista);
                                }
                        }
                } catch (CoreException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

        /**
         * Localiza la referencia a recurso compartido en coedición de la lista de
         * coeditados.
         *
         * @param document
         *          IDocument que se toma como criterio de búsqueda.
         * @return El objeto ReferenciaRecursoCompartido al IDocument buscado.
         */
        private ReferenciaRecursoCompartido obtenerRecursoCompartido(
                        IDocument document) {
                ListIterator iterador = listaCoEditados.listIterator();
                while (iterador.hasNext()) {
                        ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
                                        .next();
                        IDocument doc = recurso.getDocument();
                        if (doc.equals(document)) {
                                return recurso;
                        }
                }
                return null;
        }

        /**
         * Método invocado cuando se detecta una modificación en un recurso por
         * parte de su listener asociado.
         *
         * @param document
         *          IDocument que referencia al documento modificado.
         * @param offset
         *          Desplazamiento en el documento de la modificación.
         * @param length
         *          Longitud del area modificada.
         * @param text
         *          Texto a sustituir en el area modificada.
         */
        protected void modificarRecurso(IDocument document, int offset, int length,
                        String text) {
                if (recCliente == null)
                        return;
                ReferenciaRecursoCompartido recurso = obtenerRecursoCompartido(document);
                if (recurso == null)
                        return;

                try {

                        recCliente.modificarRecurso(new RecursoCompartido(recurso.getID(),
                                        recurso.getFile().getName(), recurso.getFile()
                                                        .getFullPath().removeFirstSegments(1)
                                                        .removeLastSegments(1).toString()),
offset, length,
                                        text);
                } catch (RemoteException e) {
                        EcolServidor.abortarSesion(
                                                        "Error RMI al enviar la modificación al
cliente. Si los problemas persisten, revise la configuracion y reinicie la sesión de trabajo.");
```

```
                }
        }

        /**
         * Localiza la referencia a recurso compartido en coedición de la lista de
         * coeditados.
         *
         * @param IDaBuscar
         *          String que se toma como criterio de búsqueda.
         * @return El objeto ReferenciaRecursoCompartido al String identificador
         *        buscado.
         */
        private ReferenciaRecursoCompartido localizarRecurso(String IDaBuscar) {
                ListIterator iter = listaCoEditados.listIterator();
                while (iter.hasNext()) {
                        ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido) iter
                                        .next();
                        String idRecurso = recurso.getID();
                        if (idRecurso.compareTo(IDaBuscar) == 0)
                                return recurso;
                }
                return null;
        }

        /**
         * Determina si una referencia a un recurso en coediciÛn existe ya o no,
         * para ello tiene en cuenta el valor del método <code>getID()</code> del
         * mismo.
         *
         * @param nuevo
         *          Referencia sobre el cual se deseae consultar.
         * @return Devuelve verdadero o falso dependiendo si existe o no.
         */
        private boolean existeRecurso(ReferenciaRecursoCompartido nuevo) {
                ListIterator iterador = listaCoEditados.listIterator();
                while (iterador.hasNext()) {
                        ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
                                        .next();
                        String idRecurso = recurso.getID();
                        if (idRecurso.compareTo(nuevo.getID()) == 0)
                                return true;
                }
                return false;
        }

        /**
         * Determina si un editor asociado a una referencia de un recurso compartido
         * está entre los editores del entorno. Este método es útil para saber si un
         * editor de un recurso se ha cerrado.
         *
         * @param editor
         *          Editor que queremos comprobar si todavía está activo.
         * @param parts
         *          conjunto de editores actuales.
         * @return Verdadero si está activo, falso en caso contrario.
         */
        private boolean estaEditorActivo(IEditorPart editor, IEditorPart[] parts) {
                for (int i = 0; i < parts.length; i++) {
                        if (editor.equals(parts[i]))
                                return true;
                }
                return false;
        }

        /**
         * Método empleado para avisar a la pareja de programación de que se está
         * empezando a
         *
         * @param fichero
         *          REcurso sobre el que se trabajará colaborativamente.
```

```
	 * @param edPart
	 *          Editor que se a abierto para el recurso.
	 */
	public void iniciarEdicionColaborativa(IFile fichero, IEditorPart edPart) {
		if (!(edPart instanceof AbstractTextEditor))
			return;
		ITextEditor editor = (ITextEditor) edPart;
		IDocumentProvider dp = editor.getDocumentProvider();
		IDocument doc = dp.getDocument(editor.getEditorInput());
		CodigoFuenteListener cfListener = new CodigoFuenteListener(this);
		doc.addDocumentListener(cfListener);
		try {
			ReferenciaRecursoCompartido nuevo = new ReferenciaRecursoCompartido(
					fichero, edPart, doc, cfListener);
			if (!existeRecurso(nuevo)) {
				RecursoCompartido aEnviar = nuevo.getRecursoCompartidoInfo();
				listaCoEditados.add(nuevo);

				IDisplayInfoTrabajo display = EcolServidor
						.getControladorSesion().getDisplayUsuario();
				if (display != null) {
					display.addCoEditado(aEnviar);
				}

				if (recCliente != null)
					recCliente.inicioCoEdicion(aEnviar);
			}
		} catch (RemoteException e) {

			EcolServidor.abortarSesion("Error comunicando un inicio de edicion al usuario:
"+e.getMessage());
		}
	}

	/**
	 * Este método se encarga de recorrer los editores activos en la ventana de
	 * trabajo pasados como parametro y compararlos con la lista de archivos en
	 * co edición y así saber cual de ellos fue cerrado.
	 *
	 * @param parts
	 *          Conjunto de editores activos en el entorno.
	 */
	public void terminarEdicionColaborativa(IEditorPart[] parts) {
		ListIterator iterador = listaCoEditados.listIterator();
		while (iterador.hasNext()) {
			ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
					.next();
			if (!estaEditorActivo(recurso.getEditor(), parts)) {
				try {
					// Avisamos al cliente si es necesario.
					if (recCliente != null)
						recCliente.finCoEdicion(recurso.getID());
					iterador.remove();
					IDisplayInfoTrabajo display = EcolServidor

	.getControladorSesion().getDisplayUsuario();
					if (display != null) {
						display.eliminarCoEditado(recurso.getID());
					}
				} catch (RemoteException e) {

					EcolServidor.abortarSesion("Error terminando edicion
colaborativa: "+e.getMessage());

				}


			}
		}
	}
```

```
	/**
	 * Se encarga de terminar y liberar los recursos y listeners asociados al
	 * controlador de recursos del proyecto fuente.
	 *
	 */
	public void terminar() {
		// 1∫ Eliminamos el Listener de la p·gina.
		ResourcesPlugin.getWorkspace().removeResourceChangeListener(
				resourceListener);
		window.removePerspectiveListener(listener);
		listener = null;
		listaCoEditados = null;
		recCliente = null;
		window = null;
	}

	/**
	 * Establece el servicio parejo de manejo de recursos perteneciente al
	 * cliente del proyecto fuente.
	 *
	 * @param irc
	 *          Interfaz de recursos del cliente del proyecto
	 */
	public void setParejaServicio(IServicioRecursosCliente irc) {
		this.recCliente = irc;
	}

	/**
	 * Método invocado para conocer el estado de coedición de los recursos
	 *
	 * @return Referencia de los recursos que están siendo coeditados
	 * @throws RemoteException
	 *           Exepción lanzada ante problemas de comunicacion RMI
	 */
	public RecursoCompartido[] getEstadoCoedicion() throws RemoteException {
		if (listaCoEditados.isEmpty())
			return null;
		RecursoCompartido[] recursos = new RecursoCompartido[listaCoEditados
				.size()];
		int i = 0;
		ListIterator iterador = listaCoEditados.listIterator();
		while (iterador.hasNext()) {
			ReferenciaRecursoCompartido recurso = (ReferenciaRecursoCompartido)
iterador
					.next();
			recursos[i] = recurso.getRecursoCompartidoInfo();
			i++;
		}
		return recursos;
	}

	/**
	 * Método invocado para notificar los cambios en los recursos del servidor
	 *
	 * @param id
	 *          Identificador del recurso que se va modificar
	 * @param offset
	 *          Desplazamiento en el documento de la modificación.
	 * @param length
	 *          Longitud del area modificada.
	 * @param codigo
	 *          Texto a sustituir en el area modificada.
	 * @throws RemoteException
	 *           Exepción lanzada ante problemas de comunicacion RMI
	 */
	public void modificacionCodigo(String id, final int offset,
			final int length, final String codigo) throws RemoteException {

		/*
		 * METODO CLIENTE A SERVIDOR.
```

```
		 */

		// TODO Auto-generated method stub
		final ReferenciaRecursoCompartido ref = localizarRecurso(id);

		window.getWorkbench().getDisplay().asyncExec(new Runnable() {
			public void run() {
				// TODO Auto-generated method stub
				try {

ref.getDocument().removeDocumentListener(ref.getListener());
					ref.getDocument().replace(offset, length, codigo);
					ref.getEditor().doSave(null);
					ref.getDocument().addDocumentListener(ref.getListener());
				} catch (BadLocationException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
			}
		});
	}

	/**
	 * Método por el cliente invocado para obtener la estructura del proyecto y
	 * su contenido
	 *
	 * @return Array de objetos con los contenidos del proyecto.
	 * @throws RemoteException
	 *           Exepción lanzada ante problemas de comunicacion RMI
	 */
	public Object[] getEstructuraProyecto() throws RemoteException {
		return getEstructuraProyecto(EcolServidor.getControladorSesion()
				.getNProyecto());
	}

	/**
	 * OBtiene el contenido de un determinado recurso.
	 *
	 * @param id
	 *          Identificador del recurso que queremos consultar
	 * @return String con el contenido del recurso.
	 * @throws RemoteException
	 *           Exepción lanzada ante problemas de comunicacion RMI
	 */
	public String getContenidoFichero(String id) throws RemoteException {
		IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
				.getProject(EcolServidor.getControladorSesion().getNProyecto());
		return getContenido(proyecto.getFile(id));
	}

	/**
	 * Obtiene las anotaciones para un recurso determinado.
	 *
	 * @param fichero
	 *          Recurso a consultar.
	 * @return Array de Anotaciones.
	 */
	private Anotacion[] getAnotaciones(IFile fichero) {
		Anotacion[] anotaciones = null;
		try {
			IMarker[] marcadores = fichero.findMarkers(
					"ecol.marcador.AvisoProgramacion", false,
					IResource.DEPTH_ZERO);
			anotaciones = new Anotacion[marcadores.length];
			for (int i = 0; i < anotaciones.length; i++) {
				anotaciones[i] = new Anotacion(marcadores[i].getAttribute(
						IMarker.MESSAGE, ""), marcadores[i].getAttribute(
						IMarker.PRIORITY, 0), marcadores[i].getAttribute(
						IMarker.DONE, false));
			}
		} catch (CoreException e) {
```

```
                    // TODO Establecer.
                    e.printStackTrace();
            }
            return anotaciones;
    }

    /**
     * Obtiener las anotaciones de un recurso
     *
     * @param id
     *          Identificador del recurso
     * @return Array con las anotaciones encapsuladas en el objeto Anotacion.
     * @throws RemoteException
     *           Exepción lanzada ante problemas de comunicacion RMI
     */
    private Anotacion[] getAnotaciones(String id) {
            IProject proyecto = ResourcesPlugin.getWorkspace().getRoot()
                            .getProject(EcolServidor.getControladorSesion().getNProyecto());
            IFile fichero = proyecto.getFile(id);
            return getAnotaciones(fichero);
    }

    /**
     * Método invocado para obtener las anotaciones de un recurso
     *
     * @param id
     *          Identificador del recurso
     * @return Array con las anotaciones encapsuladas en el objeto Anotacion.
     * @throws RemoteException
     *           Exepción lanzada ante problemas de comunicacion RMI
     */
    public Anotacion[] getAnotacionesFichero(String id) throws RemoteException {
            return getAnotaciones(id);
    }

    /**
     * Se encarga de almacenar propiamente la anotación en el recurso.
     *
     * @param fichero
     *          Fichero sobre el que almacenamos la anotación.
     * @param anotacion
     *          Anotación que queremos almacenar en el recurso.
     * @throws CoreException
     *           Excepción lanzada si hay algún problema estableciendo la
     *           anotación.
     */
    private void asignarAnotacion(IFile fichero, Anotacion anotacion)
                    throws CoreException {
            IMarker marker = fichero
                            .createMarker("ecol.marcador.AvisoProgramacion");
            marker.setAttribute(IMarker.MESSAGE, anotacion.getTexto());
            marker.setAttribute(IMarker.PRIORITY, anotacion.getPrioridad());
            marker.setAttribute(IMarker.DONE, false);
    }

    /**
     * Método que se encarga de crear una nueva anotación y notificarlo al
     * cliente.
     *
     * @param fichero
     *          Recursos sobre el que se ha creado la anotación
     * @param resultado
     *          Anotación a almacenar
     * @throws CoreException
     *           Excepción lanzada si hay algun problema estableciendo la
     *           anotación
     * @throws RemoteException
     *           Excepción lanzada si hay problemas en la comunicación RMI
     */
    public void nuevaAnotacion(IFile fichero, Anotacion resultado)
                    throws CoreException, RemoteException {
```

```
                // Lo creamos localmente.
                asignarAnotacion(fichero, resultado);
                recCliente.setAnotacion(fichero.getFullPath().removeFirstSegments(1)
                                .toString(), resultado);
        }

        /**
         * Establece una anotación en un recurso
         *
         * @param id
         *          Identificador del recurso
         * @param anotacion
         *          Anotacion para establecer en el recurso
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public void setAnotacion(String id, Anotacion anotacion)
                        throws RemoteException {
                try {
                        asignarAnotacion(UtilidadesProyecto.getProject(
                                        EcolServidor.getControladorSesion().getNProyecto())
                                        .getFile(id), anotacion);
                } catch (CoreException e) {
                        throw new RemoteException(
                                        "Error asignando la anotacion en lafuente: "
                                                        + e.getMessage());
                }

        }

}
```

Interfaz `IServicioRecursosServidor.java`:

```
package ecol.servidor.recursos;

import java.rmi.Remote;
import java.rmi.RemoteException;

import ecol.comun.Anotacion;
import ecol.comun.RecursoCompartido;

/**
 * Interfaz que el servidor exporta al usaurio cliente para el manejo de los
 * recursos
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IServicioRecursosServidor extends Remote {

        /**
         * Método invocado para notificar los cambios en los recursos del servidor
         *
         * @param recID
         *          Identificador del recurso que se va modificar
         * @param offset
         *          Desplazamiento en el documento de la modificación.
         * @param length
         *          Longitud del area modificada.
         * @param codigo
         *          Texto a sustituir en el area modificada.
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public void modificacionCodigo(String recID, int offset, int length,
                        String codigo) throws RemoteException;

        /**
         * Método por el cliente invocado para obtener la estructura del proyecto y
```

```
         * su contenido
         *
         * @return Array de objetos con los contenidos del proyecto.
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public Object[] getEstructuraProyecto() throws RemoteException;

        /**
         * OBtiene el contenido de un determinado recurso.
         * @param id Identificador del recurso que queremos consultar
         * @return String con el contenido del recurso.
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public String getContenidoFichero(String id) throws RemoteException;

        /**
         * Método invocado para obtener las anotaciones de un recurso
         * @param id Identificador del recurso
         * @return Array con las anotaciones encapsuladas en el objeto Anotacion.
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public Anotacion[] getAnotacionesFichero(String id) throws RemoteException;

        /**
         * Establece una anotación en un recurso
         * @param id Identificador del recurso
         * @param anotacion Anotacion para establecer en el recurso
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public void setAnotacion(String id, Anotacion anotacion)
                        throws RemoteException;

        /**
         * Método invocado para conocer el estado de coedición de los recursos
         * @return Referencia de los recursos que están siendo coeditados
         * @throws RemoteException
         *           Exepción lanzada ante problemas de comunicacion RMI
         */
        public RecursoCompartido[] getEstadoCoedicion() throws RemoteException;
}
```

Clase `RecursosChangeListener.java`:

```
package ecol.servidor.recursos;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.resources.IResourceChangeEvent;
import org.eclipse.core.resources.IResourceChangeListener;
import org.eclipse.core.resources.IResourceDelta;
import org.eclipse.core.resources.IResourceDeltaVisitor;
import org.eclipse.core.runtime.CoreException;

import ecol.servidor.EcolServidor;

/**
 * Esta clase se encarga de hacer de listener sobre los recursos
 * del espacio de trabajo para mantener un controlo de los
 * recursos borrados en el mismo.
 *
 * @author Luis Fernández Álvarez
 *
 */
public    class RecursosChangeListener implements IResourceChangeListener{
        private ControladorRecursos controlador;

        /**
```

```
	 * Crea un listener de los cambios de los reucrsos.
	 * @param recursos Controlador de recursos del servidor donde notificar.
	 */
	public RecursosChangeListener(ControladorRecursos recursos) {
		controlador=recursos;
	}

	/**
	 * Método invocado cuando hay un cambio en los recursos.
	 * @param event
	 */
	public void resourceChanged(IResourceChangeEvent event) {
		try {
			event.getDelta().accept(new DeltaPrinter());
		} catch (CoreException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	/**
	 * Clase que implementa IResourceDeltaVisitor que permite recorrer un objeto
	 * IResourceDelta en busca de los elementos que han sido modificados.
	 *
	 * @author Luis Fernández Álvarez
	 *
	 */
	  class DeltaPrinter implements IResourceDeltaVisitor {

		/**
		 * Método que visita un determinado IResourceDelta, en busca de elementos
		 * borrados para su notificación.
		 */
		  public boolean visit(IResourceDelta delta) {
		    IResource res = delta.getResource();
		    switch (delta.getKind()) {
		      case IResourceDelta.REMOVED:
		        if((res instanceof
IFile)&&(EcolServidor.getControladorSesion().getNProyecto().compareTo(res.getProject().getName())==
0)){
			//Es fichero y pertenece al proyecto en el que estamos trabajando.
			 IFile fichero=(IFile)res;
			 if(fichero.getName().endsWith("java"))controlador.recursoBorrado(fichero);
			 }
			 break;
		    }
		    return true; // visit the children
		  }
		}
}
```

Clase `RecursosPerspectivaListener.java`:

```
package ecol.servidor.recursos;

 import org.eclipse.core.resources.IFile;
 import org.eclipse.core.resources.IProject;
 import org.eclipse.ui.IEditorInput;
 import org.eclipse.ui.IEditorPart;
 import org.eclipse.ui.IFileEditorInput;
 import org.eclipse.ui.IPerspectiveDescriptor;
 import org.eclipse.ui.IPerspectiveListener;
 import org.eclipse.ui.IWorkbenchPage;

 import ecol.servidor.EcolServidor;


 /**
  * Clase que se encarga de escuchar en la perspectiva que se asocia sobre nuevos
  * editores abiertos o cerrados para mantener un control actualizado de los recursos
  * sobre los que está trabajando el usuario fuente, y así saber cuando se empieza
```

```
 * a coeditar un recurso o cuando se termina.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class RecursosPerspectivaListener implements IPerspectiveListener {

	private ControladorRecursos recursosManager;

	/**
	 * Crea un nuevo listener para una perspectiva de trabajo de servidor fuente.
	 * @param recursosManager Referencia al ControladorRecursos del proyecto fuente.
	 */
	public RecursosPerspectivaListener(ControladorRecursos recursosManager){
		this.recursosManager=recursosManager;
	}

	/**
	 * Notifica al listener de que una perspectiva en la pagina ha sido
	 * activada.
	 *
	 * @param page
	 *		pagina que contiene la perspectiva activada.
	 * @param perspective
	 *		el descriptor de la perspectiva que ha sido activada.
	 * @see IWorkbenchPage#setPerspective
	 */
	public void perspectiveActivated(IWorkbenchPage page,
			IPerspectiveDescriptor perspective) {

	}

	/**
	 * Notifica al listener de que una perspectiva ha cambiado de alguna manera.
	 * (un editor ha sido abierto, una vista, etc...)
	 *
	 * @param page
	 *		la página que contiene la perspectiva a fectada.
	 * @param perspective
	 *		el descriptor de la perspectiva.
	 * @param changeId
	 *		constante CHANGE_* presente en IWorkbenchPage
	 */
	public void perspectiveChanged(IWorkbenchPage page,
			IPerspectiveDescriptor perspective, String changeId) {
		if(!EcolServidor.estaIniciadaSesion())return;
		if(changeId.compareTo(IWorkbenchPage.CHANGE_EDITOR_OPEN)==0){
			/*
			 * Un nuevo editor ha sido abierto, comprobemos que pertenece a nuestro
proyecto y que nos interesa.
			 */
			 IEditorPart edPart = page.getActiveEditor();
		    IEditorInput input = edPart.getEditorInput();
		    if (!(input instanceof IFileEditorInput) ){
			 //No estabamos editando un fichero que controlemos.
			 return;
		    }
			IFile fichero=((IFileEditorInput)input).getFile();
		      IProject proyecto=fichero.getProject();


if((proyecto.getName().compareTo(EcolServidor.getControladorSesion().getNProyecto())!=0)||!(fichero.
getName().endsWith("java"))){
			 //No nos atañe que se abra este fichero.
			 return;
		    }
		    //Es algo que necesitamos, le decimos que lo comparta. Tal fichero y su editor
asociado.
		    recursosManager.iniciarEdicionColaborativa(fichero, edPart);
		}
```

```
                if(changeId.compareTo(IWorkbenchPage.CHANGE_EDITOR_CLOSE)==0){
                        /*
                         * Detectamos que se ha cerrado un editor pero no sabemos cual,
                         * asÌ que pasamos todos los editores y que se comparen las referencias o
similar.
                         */
                        recursosManager.terminarEdicionColaborativa(page.getEditors());
                }
        }
}
```

Paquete `ecol.servidor.sesion`:

Clase `ControladorSesionServidor.java`:

```
package ecol.servidor.sesion;

import java.rmi.AccessException;
import java.rmi.NoSuchObjectException;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;

import org.eclipse.ui.IWorkbenchWindow;


import ecol.cliente.recursos.IServicioRecursosCliente;
import ecol.cliente.sesion.IControladorConexionCliente;
import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.ParejaProgramacion;
import ecol.comun.UtilidadesProyecto;
import ecol.comun.UtilidadesSesion;
import ecol.comun.comunicaciones.IServicioComunicaciones;
import ecol.comun.comunicaciones.ServicioComunicacionesImpl;
import ecol.comun.excepciones.AutenticacioInvalidaException;
import ecol.comun.excepciones.ConexionParejaException;
import ecol.comun.excepciones.ParejaEstablecidaException;
import ecol.servidor.recursos.ControladorRecursos;
import ecol.servidor.recursos.IServicioRecursosServidor;

/**
 * Esta clase se encarga de controlar toda la sesión de trabajo en la parte del
 * servidor. Mantiene los controladores de recursos, de comunicaciones, así como
 * información sobre los desarrolladores que están participando.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ControladorSesionServidor implements IControladorConexionServidor {

        private IWorkbenchWindow window;

        private String nProyecto;

        private Registry regRMI;

        private ControladorRecursos ctrlRecursos;

        private IControladorConexionCliente cliente;

        private ParejaProgramacion pareja;

        private ParejaProgramacion infoPersonal;

        private ServicioComunicacionesImpl ctrlCom;

        private IDisplayInfoTrabajo display;

        /**
         * Crea un nuevo controlador de sesión.
```

```
	 * @param name Nombre del proyecto bajo el que estamos trabajando.
	 * @param window Ventana de trabajo del proyecto.
	 * @param registro Registroy RMI donde disponer las interfaces
	 * @throws RemoteException Excepción lanzada ante problemas RMI.
	 */
	public ControladorSesionServidor(String name, IWorkbenchWindow window,
			Registry registro) throws RemoteException {
		UnicastRemoteObject.exportObject(this);
		this.window = window;
		nProyecto = name;
		infoPersonal = encapsularUsuario(name);
		regRMI = registro;
		iniciarServicios();
	}

	public ParejaProgramacion getInfoPersonal() {
		return infoPersonal;
	}

	/**
	 * Crea el objeto referente a la información personal del usuario local.
	 * @param proyecto Proyecto sobre el que está almacenada la información.
	 * @return El objeto ParejaProgramacion que encapsula la información.
	 */
	private ParejaProgramacion encapsularUsuario(String proyecto) {
		ParejaProgramacion yo = new ParejaProgramacion(UtilidadesProyecto
				.getPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_NOMBRECORTO_USUARIO),
null,
				UtilidadesProyecto.getPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_NOMBRE_USUARIO),
				UtilidadesProyecto.getPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_EMAIL_USUARIO),
				UtilidadesProyecto.getPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_OTRAINFO_USUARIO));

		return yo;
	}

	/**
	 * Informa al sistema de comunicaciones con un mensaje del sistema.
	 * @param mensaje String mensaje a notificar
	 */
	private void informar(final String mensaje){
		ctrlCom.mensajeSistema(mensaje);
	}

	/**
	 * Inicia los servicios de conexion, recursos y comunicaciones.
	 * @throws RemoteException Excepción lanzada ante errores RMI.
	 */
	private void iniciarServicios() throws RemoteException {
		regRMI.rebind(UtilidadesSesion.INT_SERVIDOR, this);

		// Creamos el controlador de recursos
		ctrlRecursos = new ControladorRecursos(null, window);

		// Creamos el controlador de comunicaciones.
		ctrlCom = new ServicioComunicacionesImpl(window, infoPersonal,nProyecto);

	}

	/**
	 * Obtiene el nobmre del proyecto.
	 * @return String que contiene el nombre del proyecto
	 */
	public String getNProyecto() {
```

```
		return nProyecto;
	}

	/**
	 * Obtiene el controlador de las comunicaciones de la sesion de trabajo.
	 * @return Objeto controlador de las comunicaciones.
	 */
	public ServicioComunicacionesImpl getControladorComunicaciones() {
		return ctrlCom;
	}

	/**
	 * Obtiene el controlador de recursos de la sesión de trabajo.
	 * @return objeto controlador de recursos de la sesión.
	 */
	public ControladorRecursos getControladorRecursos(){
		return ctrlRecursos;
	}

	/**
	 * Realiza la validación del usuario.
	 * @param pareja2 Objeto con toda la información del usuario que se va a validar.
	 * @return Verdadero o falso dependiendo si se autenticó bien om al.
	 */
	private boolean autenticarUsuario(ParejaProgramacion pareja2) {

		String nomValido = UtilidadesProyecto.getPropiedadString(nProyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_LOGIN_PAREJA);
		String passValida = UtilidadesProyecto.getPropiedadString(nProyecto,
				UtilidadesProyecto.SCOPE_SERVIDOR,
				UtilidadesProyecto.KEY_PASS_PAREJA);

		if ((pareja2.getLogin().compareTo(nomValido) == 0)
				&& (pareja2.getPasswd().compareTo(passValida) == 0))
			return true;
		else
			return false;
	}

	/**
	 * Conecta los servicios con los de la pareja.
	 * @param pareja objeto que representa a la pareja de programacion remota.
	 * @throws RemoteException Excepción lanzada cuando hay problemas con la comunicacion
RMI.
	 */
	private void conectarServicios(ParejaProgramacion pareja) throws RemoteException{
		/*
		 * Conectamos la pareja para manejo de recursos.
		 */
		IServicioRecursosCliente irc = cliente.getServicioRecursos();
		ctrlRecursos.setParejaServicio(irc);

		/*
		 * Conctamos la pareja para manejo de comunicacion.
		 */
		IServicioComunicaciones isc = cliente.getServicioComunicaciones();
		ctrlCom.setParejaServicio(isc);

	}

	/**
	 * Desconecta y libera los recursos creados para la sesión
	 * de trabajo colaborativo.
	 *
	 */
	private void desconectarServicios() {

		/*
		 * Desconectamos la pareja de recursos.
		 */
```

```
                ctrlRecursos.terminar();
                ctrlRecursos = null;

                /*
                 * Desconctamos la pareja para manejo de comunicacion.
                 */
                ctrlCom.terminar();
                ctrlCom = null;

        }

        /**
         * Limpia la información referente al emparejamiento de recursos
         * con el cliente.
         *
         */
        private void limpiarEnlaceServicios() {
                ctrlRecursos.setParejaServicio(null);
                ctrlCom.setParejaServicio(null);
                cliente = null;
        }

        /**
         * Establece la pareja de programación de la sesión.
         *
         * @param pareja Objeto que encapsula a la parejade programacion.
         * @param cliente2 Interfaz de conexión con el cliente.
         * @throws ConexionParejaException Excepción lanzada si hay errores al comunicarse con el
cliente.
         */
        private void setPareja(ParejaProgramacion pareja, IControladorConexionCliente cliente2)
                        throws ConexionParejaException {
                boolean error=false;
                String cadena="";
                try {
                        this.cliente = cliente2;
                        // Comenzariamos a conectar los servicios.
                        conectarServicios(pareja);
                        // Todo estuvo bien, asignamos la pareja.
                        this.pareja = pareja;
                        if (display != null)
                                display.parejaConectada(this.pareja);
                } catch (RemoteException e) {
                        error=true;
                        cadena="Error en la comunicación: "
                                        + e.getMessage();
                }
                if(error){
                        limpiarEnlaceServicios();
                        throw new ConexionParejaException(cadena);
                }
        }

        /**
         * Se encarga de los procedimientos para terminar la sesión de trabajo,
         * avisar al cliente y terminar recursos.
         *
         */
        public void terminarSesion() {
                if (cliente != null) {
                        // Alguien se conectÛ=> Hay servicios aparte de la interfaz de
                        // conexiÛn
                        try {
                                cliente.terminarSesion();
                        } catch (RemoteException e1) {
                                // TODO Auto-generated catch block
                                // No me importa mucho si la acabo bien o no, yo aquÌ la acabo.
                                e1.printStackTrace();
                        }
                        cliente = null;
                }
```

```
		// En este punto el cliente tendrÌa que estar totalmente
		// desinteresado en nuestros servicios.
		// Por tanto, los vamos terminando.

			desconectarServicios();


		// Ya podemos unbind la Int del proyecto.
		try {
			regRMI.unbind(UtilidadesSesion.INT_SERVIDOR);
		} catch (AccessException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (RemoteException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NotBoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		regRMI = null;
		nProyecto = null;
		try {
			UnicastRemoteObject.unexportObject(this, true);
		} catch (NoSuchObjectException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	/**
	 * Método invocado por el cliente para inciar una validación en la
	 * sesión de trabajo.
	 *
	 * @param pareja2 Objeto que encapsula toda la información del usuario.
	 * @param cliente Referencia al controlador de conexión del usuario.
	 * @return ParejaProgramacion con la informacion local del usuario fuente.
	 * @throws RemoteException
	 *		Excepción lanzada en la comunicación RMI.
	 * @throws ConexionParejaException
	 *		Excepción lanzada cuando el servidor obtiene errores al
	 *		contactar con la pareja recien conectada.
	 * @throws AutenticacioInvalidaException
	 *		Excepción lanzada cuando los datos de acceso no son válidos.
	 * @throws ParejaEstablecidaException
	 *		Excepción lanzada cuando ya existe una pareja conectada al
	 *		proyecto fuente.
	 */
	public ParejaProgramacion conectarPareja(ParejaProgramacion pareja2,
IControladorConexionCliente cliente)
			throws RemoteException, ConexionParejaException,
			AutenticacioInvalidaException, ParejaEstablecidaException {

		if (pareja == null) {
			if (autenticarUsuario(pareja2)) {
				setPareja(pareja2,cliente);
				informar(pareja2.getLogin()+" se ha conectado.");
				return infoPersonal;
			} else
				throw new AutenticacioInvalidaException(
						"Imposible autenticar el usuario: "
							+ pareja2.getLogin() + ".");
		} else {
			throw new ParejaEstablecidaException(
					"Ya existe una pareja establecida en este proyecto.");
		}
	}

	/**
```

```
	 * Método invocado por la pareja de programación para informar
	 * que ha terminado de trabajar en la sesión.
	 *
	 * @throws RemoteException
	 *           Excepción lanzada en la comunicación RMI.
	 */
	public void desconectarPareja() throws RemoteException {
		ctrlRecursos.setParejaServicio(null);
		ctrlCom.setParejaServicio(null);
		informar(pareja.getLogin()+" se ha desconectado.");
		pareja = null;
		if (display != null)
			display.parejaDesconectada();
		cliente = null;
	}

	/**
	 * Obtiene el objeto que representa la información personal del usuario
	 * remoto.
	 *
	 * @return ParejaProgramacion Informacion del usuario remoto.
	 * @see ecol.comun.ParejaProgramacion
	 */
	public ParejaProgramacion getPareja() {
		return pareja;
	}

	/**
	 * Establece el elemento que servirá de visor para la información de trabajo
	 * de la sesión colaborativa.
	 *
	 * @param display
	 *          Elemento que implemente la interfaz IDisplayInfoTrabajo. O
	 *          null si no se desea ningún visor.
	 * @see ecol.comun.IDisplayInfoTrabajo
	 */
	public void setDisplayUsuarios(IDisplayInfoTrabajo display) {
		this.display = display;
	}

	/**
	 * Retorna el elemento que sirve de visor de información de la sesión
	 * @return Referencia al objeto que implementa IDisplayInfoTrabajo y sirve de visor.
	 */
	public IDisplayInfoTrabajo getDisplayUsuario() {
		return display;
	}

	/**
	 * Retorna la interfaz de comunicaciones del usuario fuente exportada al usuario cliente para
	 * mantener la comunicacion.
	 * @return referencia a la interfaz remota exportada para el control de comunicaciones.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.comun.comunicaciones.IServicioComunicaciones
	 */
	public IServicioComunicaciones getServicioComunicaciones() throws RemoteException {
		return ctrlCom;
	}

	/**
	 * Retorna la interfaz de recursos del usuario fuente exportada al usuario cliente para
	 * el manejo de los mismos.
	 * @return referencia a la interfaz remota exportada para el control de recursos.
	 * @throws RemoteException
	 *           Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.servidor.recursos.IServicioRecursosServidor
	 */
	public IServicioRecursosServidor getServicioRecursos() throws RemoteException {
		return ctrlRecursos;
	}
```

```
   /**
    * Termina todos los servicios y recursos asociados a la
    * sesión de trabajo.
    *
    */
   public void abortarSesion() {
      ctrlCom.terminar();
      ctrlRecursos.terminar();
      try {
         cliente.terminarSesion();
      } catch (RemoteException e) {
         // NO NOS IMPORTA SI FUE MAL; ESTAMOS
         // ABORTANDO.
      }
      cliente=null;
      try {
                        UnicastRemoteObject.unexportObject (this,true);
            UnicastRemoteObject.unexportObject(ctrlCom, true);
            UnicastRemoteObject.unexportObject(ctrlRecursos, true);
               } catch (NoSuchObjectException e) {
                        // NO DEBERÍA SALTAR Y SI SALTA NO IMPORTA; ESTAMOS
                        //ABORTANDO.
               }

   }
}
```

Interfaz `IControladorConexionServidor.java`:

```
package ecol.servidor.sesion;

import java.rmi.Remote;
import java.rmi.RemoteException;

import ecol.cliente.sesion.IControladorConexionCliente;
import ecol.comun.ParejaProgramacion;
import ecol.comun.comunicaciones.IServicioComunicaciones;
import ecol.comun.excepciones.AutenticacioInvalidaException;
import ecol.comun.excepciones.ConexionParejaException;
import ecol.comun.excepciones.ParejaEstablecidaException;
import ecol.servidor.recursos.IServicioRecursosServidor;

/**
 * Interfaz que sirve de medio de comunicación para que el usuario cliente se
 * comunique con el usuario fuente en cuanto a tareas generales de conexión.
 *
 * @author Luis Fernández Álvarez
 *
 */
public interface IControladorConexionServidor extends Remote {

        /**
         * Método invocado por el cliente para inciar una validación en la
         * sesión de trabajo.
         *
         * @param pareja Objeto que encapsula toda la información del usuario.
         * @param conexion Referencia al controlador de conexión del usuario.
         * @return ParejaProgramacion con la informacion local del usuario fuente.
         * @throws RemoteException
         *           Excepción lanzada en la comunicación RMI.
         * @throws ConexionParejaException
         *           Excepción lanzada cuando el servidor obtiene errores al
         *           contactar con la pareja recien conectada.
         * @throws AutenticacioInvalidaException
         *           Excepción lanzada cuando los datos de acceso no son válidos.
         * @throws ParejaEstablecidaException
         *           Excepción lanzada cuando ya existe una pareja conectada al
         *           proyecto fuente.
         */
```

```
	public ParejaProgramacion conectarPareja(ParejaProgramacion pareja,
IControladorConexionCliente conexion) throws RemoteException, ConexionParejaException,
AutenticacioInvalidaException, ParejaEstablecidaException;

	/**
	 * Método invocado por la pareja de programación para informar
	 * que ha terminado de trabajar en la sesión.
	 *
	 * @throws RemoteException
	 *		Excepción lanzada en la comunicación RMI.
	 */
	public void desconectarPareja()throws RemoteException;

	/**
	 * Retorna la interfaz de recursos del usuario fuente exportada al usuario cliente para
	 * el manejo de los mismos.
	 * @return referencia a la interfaz remota exportada para el control de recursos.
	 * @throws RemoteException
	 *		Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.servidor.recursos.IServicioRecursosServidor
	 */
	public IServicioRecursosServidor getServicioRecursos() throws RemoteException;

	/**
	 * Retorna la interfaz de comunicaciones del usuario fuente exportada al usuario cliente para
	 * mantener la comunicacion.
	 * @return referencia a la interfaz remota exportada para el control de comunicaciones.
	 * @throws RemoteException
	 *		Excepción lanzada si hay algún error con la comunicación RMI.
	 * @see ecol.comun.comunicaciones.IServicioComunicaciones
	 */
	public IServicioComunicaciones getServicioComunicaciones() throws RemoteException;

}
```

Paquete `ecol.servidor.vistas`:

Clase `ChatServidorView.java`:

```
package ecol.servidor.vistas;

import org.eclipse.swt.SWT;
import org.eclipse.swt.dnd.Clipboard;
import org.eclipse.swt.dnd.TextTransfer;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.FillLayout;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.part.ViewPart;

import ecol.comun.comunicaciones.IDisplayMensajes;
import ecol.comun.comunicaciones.MensajeChat;
import ecol.servidor.EcolServidor;

/**
 * Clase que se encarga de crear la vista para la comunicación mediante un chat
 * textual con el otro usuario que participa en la sesión de comunicación.
 * Permite el envio de mensajes textuales introducidos por el usuario y que
 * estén en el portapapeles.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class ChatServidorView extends ViewPart implements IDisplayMensajes {
```

```
	private Button btEnviar;

	private Button btEnviarPaste;

	private Text txtAEnviar;

	private Text listaMensajes;

	/**
	 * El constructor.
	 */
	public ChatServidorView() {

	}

	/**
	 * Este método es llamado y nos permite crear la vista e inicializarla.
	 *
	 * @param parent
	 *          Composite sobre el que empezaremos a añadir elementos.
	 */
	public void createPartControl(Composite parent) {
		if (!EcolServidor.estaIniciadaSesion()) {
			Label noIniciado = new Label(parent, SWT.NONE);
			noIniciado.setText("No ha iniciado ninguna sesiÛn de trabajo.");
			return;
		}
		final Clipboard clip = new Clipboard(parent.getDisplay());

		FillLayout fillLayout = new FillLayout();
		fillLayout.type = SWT.VERTICAL;
		parent.setLayout(fillLayout);

		Composite panelMensajes = new Composite(parent, SWT.NONE);
		GridLayout layout = new GridLayout();
		layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
1;
		layout.verticalSpacing = 5;
		panelMensajes.setLayout(layout);

		GridData data = new GridData();
		data.horizontalAlignment = SWT.FILL;
		data.grabExcessHorizontalSpace = true;
		data.widthHint = 100;
		data.verticalAlignment = SWT.FILL;
		data.grabExcessVerticalSpace = true;
		data.heightHint = 100;

		listaMensajes = new Text(panelMensajes, SWT.BORDER | SWT.V_SCROLL
				| SWT.WRAP | SWT.READ_ONLY);
		listaMensajes.setLayoutData(data);


		Composite panelEnvio = new Composite(parent, SWT.NONE);

		GridLayout gridLayout = new GridLayout();
		gridLayout.numColumns = 2;
		panelEnvio.setLayout(gridLayout);

		txtAEnviar = new Text(panelEnvio, SWT.SINGLE | SWT.BORDER);
		txtAEnviar.setText("");
		data = new GridData();
		data.horizontalAlignment = GridData.FILL;
		data.grabExcessHorizontalSpace = true;
		txtAEnviar.setLayoutData(data);
		txtAEnviar.addModifyListener(new TextoAEnviarListener());
		btEnviar = new Button(panelEnvio, SWT.PUSH);
		btEnviar.setText("Enviar");
		btEnviar.setEnabled(false);
		btEnviar.addSelectionListener(new SelectionAdapter() {
			public void widgetSelected(SelectionEvent e) {
```

```
                                String texto = txtAEnviar.getText();
                                txtAEnviar.setText("");
                                EcolServidor.getControladorSesion()

        .getControladorComunicaciones().enviarMensajeAPareja(
                                                                texto);
                        }
                });
                btEnviarPaste = new Button(panelEnvio, SWT.PUSH);
                btEnviarPaste.setText("Enviar desde Portapapeles");
                btEnviarPaste.addSelectionListener(new SelectionAdapter() {
                        public void widgetSelected(SelectionEvent e) {
                                TextTransfer transfer = TextTransfer.getInstance();
                                String datos = (String) clip.getContents(transfer);
                                if (datos != null) {
                                        EcolServidor.getControladorSesion()
                                                        .getControladorComunicaciones()
                                                        .enviarMensajeAPareja(datos);
                                }
                        }

                });
                EcolServidor.getControladorSesion().getControladorComunicaciones()
                .setDisplayMensajes(this);
                parent.getShell().setDefaultButton(btEnviar);

        }

        /**
         * Método invocado cuando se cierra la vista. Útil para notificar que ya no
         * hay un visor de mensajes al entorno.
         */
        public void dispose() {
                if (EcolServidor.estaIniciadaSesion())
                        EcolServidor.getControladorSesion().getControladorComunicaciones()
                                        .setDisplayMensajes(null);
        }

        /**
         * Establece el foco en el control de la lista de mensajes.
         */
        public void setFocus() {
                listaMensajes.setFocus();
        }

        /**
         * Clase que se encarga de controlar los eventos de modificación producidos
         * en el cuadro de texto que representa el mensaje del usuario. Su objetivo
         * principal es actualizar el botón de enviar.
         *
         * @author Luis Fernández Álvarez
         *
         */
        private class TextoAEnviarListener implements ModifyListener {

                /**
                 * Método invocado cuando se produce el evento de modificación.
                 * @param e Evento de modificación recibido.
                 */
                public void modifyText(ModifyEvent e) {
                        btEnviar.setEnabled(txtAEnviar.getText().compareTo("") != 0);
                }
        }

        /**
         * Método invocado cuando llega un nuevo mensaje de chat al sistema.
         *
         * @param mensaje
         *          Mensaje de chat encapsulado en un objeto MensajeChat.
         */
        public void nuevoMensaje(MensajeChat mensaje) {
```

```
                listaMensajes.append(mensaje.toString() + "\n");
        }

        /**
         * Método invocado cuando llega un mensaje del sistema (Por ejemplo la
         * conexión de un usuario.
         *
         * @param mensaje
         *          String que representa el mensaje.
         */
        public void mensajeSistema(String mensaje) {
                listaMensajes.append("** "+mensaje + "\n");
        }
}
```

Clase `InformacionParejaView.java`:

```
package ecol.servidor.vistas;

import org.eclipse.jface.viewers.ListViewer;
import org.eclipse.swt.SWT;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.ExpandBar;
import org.eclipse.swt.widgets.ExpandItem;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Link;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;

import ecol.comun.IDisplayInfoTrabajo;
import ecol.comun.ParejaProgramacion;
import ecol.comun.RecursoCompartido;
import ecol.servidor.EcolServidor;

/**
 * Clase que implementa un visor de información de trabajo particular para el
 * entorno de trabajo del usuario fuente. Se trata de una vista.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class InformacionParejaView extends ViewPart implements
                IDisplayInfoTrabajo {
        private Composite parent;

        private ListViewer viewer;

        private Label labelNombreRemoto;

        private Link labelEmailRemoto;

        private Label labelOtraInfoRemota;

        private Label labelJDKRemota;

        private Label labelOSRemoto;

        private ExpandBar bar;

        private ExpandItem item1;

        /**
         * Este método es llamado y nos permite crear la vista e inicializarla.
         *
         * @param parent
         *          Composite sobre el que empezaremos a añadir elementos.
         */
        public void createPartControl(Composite parent) {
```

```
		this.parent = parent;
		if (!EcolServidor.estaIniciadaSesion()) {
			Label noIniciado = new Label(parent, SWT.NONE);
			noIniciado.setText("No ha iniciado ninguna sesiÛn de trabajo.");
			return;
		}
		ParejaProgramacion infoLocal = EcolServidor.getControladorSesion()
				.getInfoPersonal();

		bar = new ExpandBar(parent, SWT.V_SCROLL);

		/*
		 * Creamos el último group para los documentos compartidos.
		 */
		Composite composite = new Composite(bar, SWT.NONE);
		GridLayout layout = new GridLayout();
		layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
		layout.verticalSpacing = 10;
		composite.setLayout(layout);

		GridData data = new GridData();
		data.horizontalAlignment = SWT.FILL;
		data.grabExcessHorizontalSpace = true;
		data.widthHint = 100;
		data.verticalAlignment = SWT.FILL;
		data.grabExcessVerticalSpace = true;
		data.heightHint = 100;

		viewer = new ListViewer(composite, SWT.SINGLE | SWT.V_SCROLL
				| SWT.H_SCROLL);

		viewer.getList().computeSize(SWT.DEFAULT, SWT.DEFAULT);
		viewer.getList().setLayoutData(data);

		ExpandItem item2 = new ExpandItem(bar, SWT.NONE, 0);
		item2.setText("En coedición");
		item2.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
		item2.setControl(composite);
		item2.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
				ISharedImages.IMG_OBJ_PROJECT));
		/*
		 * Creamos el Composite inferior, que será la información de nuestra
		 * pareja remota de programación.
		 */
		composite = new Composite(bar, SWT.NONE);
		layout = new GridLayout();
		layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
		layout.verticalSpacing = 10;
		composite.setLayout(layout);

		data = new GridData();
		data.horizontalAlignment = SWT.FILL;
		data.grabExcessHorizontalSpace = true;
		data.widthHint = 100;
		labelNombreRemoto = new Label(composite, SWT.NONE);
		labelNombreRemoto.setLayoutData(data);

		data = new GridData();
		data.horizontalAlignment = SWT.FILL;
		data.grabExcessHorizontalSpace = true;
		data.widthHint = 100;
		labelEmailRemoto = new Link(composite, SWT.NONE);
		labelEmailRemoto.setLayoutData(data);

		data = new GridData();
		data.horizontalAlignment = SWT.FILL;
		data.grabExcessHorizontalSpace = true;
		data.widthHint = 100;
		labelOtraInfoRemota = new Label(composite, SWT.NONE);
```

```
                labelOtraInfoRemota.setLayoutData(data);

                data = new GridData();
                data.horizontalAlignment = SWT.FILL;
                data.grabExcessHorizontalSpace = true;
                data.widthHint = 100;
                labelJDKRemota = new Label(composite, SWT.NONE);
                labelJDKRemota.setLayoutData(data);

                data = new GridData();
                data.horizontalAlignment = SWT.FILL;
                data.grabExcessHorizontalSpace = true;
                data.widthHint = 100;
                labelOSRemoto = new Label(composite, SWT.NONE);
                labelOSRemoto.setLayoutData(data);

                ParejaProgramacion remoto = EcolServidor.getControladorSesion()
                                .getPareja();
                if (remoto == null)
                        vaciarInfoPareja();
                else
                        editarInfoPareja(remoto);

                item1 = new ExpandItem(bar, SWT.NONE, 0);
                item1.setText("InformaciÛn pareja");
                item1.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
                item1.setControl(composite);
                item1.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
                                ISharedImages.IMG_OBJ_PROJECT));

                /*
                 * Creamos el Composite superior, que será la información personal de la
                 * persona local.
                 */
                composite = new Composite(bar, SWT.NONE);
                layout = new GridLayout();
                layout.marginLeft = layout.marginTop = layout.marginRight = layout.marginBottom =
10;
                layout.verticalSpacing = 10;
                composite.setLayout(layout);

                Label labelNombreLocal = new Label(composite, SWT.NONE);
                labelNombreLocal.setText("Nombre: " + infoLocal.getNombre());

                Link labelEmailLocal = new Link(composite, SWT.NONE);
                labelEmailLocal.setText("E-Mail: <A>" + infoLocal.getEmail() + "</A>");

                Label labelOtraInfoLocal = new Label(composite, SWT.NONE);
                labelOtraInfoLocal.setText("Otros: " + infoLocal.getOtraInformacion());

                Label labelJDKLocal = new Label(composite, SWT.NONE);
                labelJDKLocal.setText("JDK: " + infoLocal.getJDK());

                Label labelOSLocal = new Label(composite, SWT.NONE);
                labelOSLocal.setText("SO: " + infoLocal.getOS());

                ExpandItem item0 = new ExpandItem(bar, SWT.NONE, 0);
                item0.setText("InformaciÛn personal");
                item0.setHeight(composite.computeSize(SWT.DEFAULT, SWT.DEFAULT).y);
                item0.setControl(composite);
                item0.setImage(PlatformUI.getWorkbench().getSharedImages().getImage(
                                ISharedImages.IMG_OBJ_PROJECT));

                bar.setSpacing(8);

                EcolServidor.getControladorSesion().setDisplayUsuarios(this);
        }

        /**
         * Establece el foco en el control de la lista de mensajes.
         */
```

```
	public void setFocus() {
		viewer.getControl();
	}

	/**
	 * Método invocado cuando se cierra la vista. Útil para notificar que ya no
	 * hay un visor de información al entorno.
	 */
	public void dispose() {
		if (EcolServidor.estaIniciadaSesion())
			EcolServidor.getControladorSesion().setDisplayUsuarios(null);
	}

	/**
	 * Método invocado cuando se añade un nuevo recurso para coedición al
	 * sistema. El visor debe facilitar al usuario su control
	 *
	 * @param recurso
	 *          Referencia al recurso compartido que se empieza a coeditar.
	 */
	public void addCoEditado(RecursoCompartido recurso) {
		viewer.getList().add(recurso.getID());
	}

	/**
	 * Método invocado cuando se termina de coeditar un recurso determinado.
	 *
	 * @param id
	 *          Identificador del recurso que ha dejado de ser coeditable.
	 */
	public void eliminarCoEditado(String id) {
		viewer.getList().remove(id);
	}

	/**
	 * Método invocado por el sistema cuando una nueva pareja se conecta a la
	 * sesión de trabajo.
	 *
	 * @param pareja
	 *          Objeto ParejaProgramacion que encapsula la información a
	 *          mostrar.
	 */
	public void parejaConectada(final ParejaProgramacion pareja) {
		EcolServidor.getVentanaTrabajo().getWorkbench().getDisplay().asyncExec(
				new Runnable() {
					public void run() {
						editarInfoPareja(pareja);
						item1.getControl().redraw();
						item1.getControl().update();
					}
				});

	}

	/**
	 * Edita la información relacionada con la pareja.
	 *
	 * @param pareja
	 *          Objeto que almacena la información de la pareja que se
	 *          actualizará al sistema.
	 */
	private void editarInfoPareja(ParejaProgramacion pareja) {
		labelEmailRemoto.setText("Email: <A>" + pareja.getEmail() + "</A>");
		labelOtraInfoRemota.setText("Otros: " + pareja.getOtraInformacion());
		labelNombreRemoto.setText("Nombre: " + pareja.getNombre());
		labelJDKRemota.setText("JDK: " + pareja.getJDK());
		labelOSRemoto.setText("SO: " + pareja.getOS());
	}

	/**
	 * Vacía la información de la pareja.
```

```
		 *
		 */
		private void vaciarInfoPareja() {
			labelEmailRemoto.setText("Email:");
			labelOtraInfoRemota.setText("Otros:");
			labelNombreRemoto.setText("Nombre:");
			labelJDKRemota.setText("JDK: ");
			labelOSRemoto.setText("SO: ");

		}

		/**
		 * Método invocado cuando la pareja de programación se ha desconectado del
		 * entorno. Se debe notificar de ello al usuario.
		 *
		 */
		public void parejaDesconectada() {
			parent.getDisplay().asyncExec(new Runnable() {
				public void run() {
					vaciarInfoPareja();
				}
			});

		}

}
```

Paquete `ecol.servidor.wizards`:

Clase `DetallesProyectoPage.java`:

```
package ecol.servidor.wizards;

import org.eclipse.jface.wizard.WizardPage;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Group;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.Text;

/**
 * Clase que implementa la página del asistente para los detalles del proyecto
 * fuente que se va ha crear.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class DetallesProyectoPage extends WizardPage {

	private Text txtLoginPareja;

	private Text txtPasswdPareja;

	private Text txtNombrePersonal;

	private Text txtNombreCortoPersonal;

	private Text txtEmailPersonal;

	private Text txtOtraInfoPersonal;

	/**
	 * Devuelve el valor introducido por el usario en la casilla de nombre corto personal.
	 *
	 * @return String que representa el nombre corto personal del usuario fuente.
	 */
	public String getNombreCortoPersonal() {
```

```
		return txtNombreCortoPersonal.getText();
	}

	/**
	 * Devuelve el valor introducido por el usario en la casilla de login de acceso.
	 *
	 * @return String que representa el login introducido para acceder al proyecto.
	 */
	public String getLoginPareja() {
		return txtLoginPareja.getText();
	}

	/**
	 * Devuelve el valor introducido por el usario en la casilla de contraseña de acceso.
	 *
	 * @return String que representa el password introducido para acceder al proyecto.
	 */
	public String getPasswdPareja() {
		return txtPasswdPareja.getText();
	}

	/**
	 * Obtiene el nombre introducido.
	 * @return String con el nombre personal.
	 */
	public String getNombrePersonal() {
		return txtNombrePersonal.getText();
	}

	/**
	 * Obtiene el email personal introducido.
	 * @return String con el email personal.
	 */
	public String getEmailPersona() {
		return txtEmailPersonal.getText();
	}

	/**
	 * Obtiene el campo de información adicional introducido.
	 * @return String con el campo de información adicional
	 */
	public String getOtraInfoPersonal() {
		return txtOtraInfoPersonal.getText();
	}

	/**
	 * Constructor de la página del asistente encargada de tomar los detalles
	 * del proyecto creado.
	 *
	 * @param pageName Nombre de la página
	 */
	protected DetallesProyectoPage(String pageName) {
		super(pageName);
		this.setTitle("Detalles del proyecto");
		this
				.setDescription("Introduzca los detalles del proyecto colaborativo");
	}

	/**
	 * Método invocado para la creación visual de la página del asistente.
	 *
	 * @param parent
	 *          Composite donde iremos creando los elementos.
	 */
	public void createControl(Composite parent) {
		// Crea el Composite principal y le establecemos el layout.
		Composite mainComposite = new Composite(parent, SWT.NONE);
		GridLayout gridLayout = new GridLayout();
		gridLayout.numColumns = 1;
		gridLayout.verticalSpacing = 10;
		mainComposite.setLayout(gridLayout);
```

```
                // Crea los contenidos de la página.
                createPageContent(mainComposite);

                // page setting
                setControl(mainComposite);
        }

        /**
         * Crea el contenido de la página del asistente.
         *
         * @param parent
         *          Composite donde crearemos los elementos del asistente.
         */
        public void createPageContent(Composite parent) {
                // Listener de las casillas de los detalles del proyecto.
                DetallesProyectoListener textListener = new DetallesProyectoListener();

                /*
                 * Primer panel: Información de la pareja.
                 */
                Group groupPareja = new Group(parent, SWT.SHADOW_ETCHED_IN);
                groupPareja
                                .setText("Información de la pareja para este proyecto
colaborativo:");
                GridLayout gridLayout = new GridLayout();
                gridLayout.numColumns = 1;
                groupPareja.setLayout(gridLayout);
                GridData gridData = new GridData();
                gridData.horizontalAlignment = GridData.FILL;
                gridData.grabExcessHorizontalSpace = true;
                groupPareja.setLayoutData(gridData);

                // Primera entrada: nombre de usuario de la pareja.
                Label labelNombre = new Label(groupPareja, SWT.LEFT);
                labelNombre.setText("Login de la pareja: ");
                txtLoginPareja = new Text(groupPareja, SWT.SINGLE | SWT.BORDER
                                | SWT.LEFT);
                txtLoginPareja.addModifyListener(textListener);
                txtLoginPareja.setLayoutData(gridData);

                // Segunda entrada: password de la pareja.
                Label labelPasswd = new Label(groupPareja, SWT.LEFT);
                labelPasswd.setText("Password de la pareja: ");
                txtPasswdPareja = new Text(groupPareja, SWT.SINGLE | SWT.BORDER
                                | SWT.LEFT);
                txtPasswdPareja.addModifyListener(textListener);
                txtPasswdPareja.setLayoutData(gridData);
                txtPasswdPareja.setEchoChar('*');
                /*
                 * Segundo panel: InformaciÛn personal.
                 */
                Group groupPersonal = new Group(parent, SWT.SHADOW_ETCHED_IN);
                groupPersonal.setText("Información personal: ");
                GridLayout layoutPersonal = new GridLayout();
                layoutPersonal.numColumns = 1;
                groupPersonal.setLayout(layoutPersonal);
                GridData dataPersonal = new GridData();
                dataPersonal.horizontalAlignment = GridData.FILL;
                dataPersonal.grabExcessHorizontalSpace = true;
                groupPersonal.setLayoutData(dataPersonal);

                // Entrada: nombre corto
                Label labelNombreCortoPersonal = new Label(groupPersonal, SWT.LEFT);
                labelNombreCortoPersonal.setText("Nombre corto: ");
                txtNombreCortoPersonal = new Text(groupPersonal, SWT.SINGLE
                                | SWT.BORDER | SWT.LEFT);
                txtNombreCortoPersonal.addModifyListener(textListener);
                txtNombreCortoPersonal.setLayoutData(dataPersonal);

                // Primera entrada: nombre completo.
```

```
		Label labelNombrePersonal = new Label(groupPersonal, SWT.LEFT);
		labelNombrePersonal.setText("Nombre completo: ");
		txtNombrePersonal = new Text(groupPersonal, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		txtNombrePersonal.addModifyListener(textListener);
		txtNombrePersonal.setLayoutData(dataPersonal);

		// Segunda entrada: E-Mail
		Label labelEmailPersonal = new Label(groupPersonal, SWT.LEFT);
		labelEmailPersonal.setText("Dirección de correo: ");
		txtEmailPersonal = new Text(groupPersonal, SWT.SINGLE | SWT.BORDER
				| SWT.LEFT);
		txtEmailPersonal.addModifyListener(textListener);
		txtEmailPersonal.setLayoutData(dataPersonal);

		// Tercera entrada: Otra información.
		Label labelOtraInfoPersonal = new Label(groupPersonal, SWT.LEFT);
		labelOtraInfoPersonal.setText("Otra información: ");
		txtOtraInfoPersonal = new Text(groupPersonal, SWT.MULTI | SWT.BORDER
				| SWT.LEFT);
		txtOtraInfoPersonal.addModifyListener(textListener);
		txtOtraInfoPersonal.setLayoutData(dataPersonal);
	}

	/**
	 * Método invocado cuando la página va a ser visualizada.
	 */
	public void setVisible(boolean visible) {
		super.setVisible(visible);
	}

	/**
	 * Determina si una cadena está vacía.
	 *
	 * @param cadena
	 *          String a comprobar.
	 * @return Verdadero si esta vacía y falso en el caso contrario.
	 */
	private boolean estaVacia(String cadena) {
		return cadena.length() == 0;
	}

	/**
	 * Determina si la página ha sido completada adecuadamente.
	 */
	public boolean isPageComplete() {
		return !estaVacia(getLoginPareja()) && !estaVacia(getPasswdPareja())
				&& !estaVacia(getNombrePersonal())
				&& !estaVacia(getEmailPersona())
				&& !estaVacia(getOtraInfoPersonal())
				&& !estaVacia(getNombreCortoPersonal());

	}

	/**
	 * Clase que se encarga de controlar los eventos de modificación producidos
	 * en el cuadro de texto que representan los diferentes campos. Su objetivo
	 * principal es actualizar los botones del asistente.
	 *
	 * @author Luis Fernández Álvarez
	 *
	 */
	private class DetallesProyectoListener implements ModifyListener {

		/**
		 * Método invocado cuando se produce el evento de modificación.
		 *
		 * @param e
		 *          Evento de modificación recibido.
		 */
		public void modifyText(ModifyEvent e) {
```

```
                        getWizard().getContainer().updateButtons();
                }
        }
}
```

Clase `NuevoProyectoWizard.java`:

```
package ecol.servidor.wizards;

import org.eclipse.core.resources.IFolder;
import org.eclipse.core.resources.IProject;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.Path;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.wizard.Wizard;
import org.eclipse.ui.INewWizard;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.dialogs.WizardNewProjectCreationPage;
import org.osgi.service.prefs.BackingStoreException;

import ecol.comun.UtilidadesProyecto;

/**
 * Clase que representa el asistente para la creación de un nuevo proyecto
 * fuente. Esta clase se encarga de añadir las páginas al asistente, así como
 * finalizar la tarea persistiendo los cambios.
 *
 * @author Luis Fernández Álvarez
 *
 */
public class NuevoProyectoWizard extends Wizard implements INewWizard {
        private DetallesProyectoPage detallesProyecto;

        private WizardNewProjectCreationPage nombreProyecto;

        private IWorkbench workbench;

        /**
         * Constructor del asistente para la creación de un proyecto colaborativo
         * como servidor.
         */
        public NuevoProyectoWizard() {
                super();
                setWindowTitle("E-Col: Nuevo proyecto como servidor");
        }

        /**
         * Determina si se puede activar la finalización el asistente de creación de
         * proyecto.
         *
         * @return Verdadero si se puede finalizar o falso en caso contrario.
         */
        public boolean canFinish() {
                return nombreProyecto.isPageComplete()
                                && detallesProyecto.isPageComplete();
        }

        /**
         * Añade páginas al asistente de creación de un nuevo proyecto.
         */
        public void addPages() {
                nombreProyecto = new WizardNewProjectCreationPage("Nuevo Proyecto");
                nombreProyecto.setTitle("Nuevo Proyecto E-Col");
                nombreProyecto
                                .setDescription("Crea un nuevo proyecto java colaborativo");
                detallesProyecto = new DetallesProyectoPage("Detalles Proyecto");
                this.addPage(nombreProyecto);
                this.addPage(detallesProyecto);
        }
```

```
	/**
	 * Se encarga de realizar las tareas del asistente una vez el usuario lo ha
	 * terminado.
	 *
	 * @return Verdadero si ha terminado bien o falso en caso contrario.
	 */
	public boolean performFinish() {
		// Tomamos el manejador del proyecto.
		IProject proyecto = nombreProyecto.getProjectHandle();

		try {
			proyecto.create(null);
			// Lo abrimos puesto que se crea cerrado.
			proyecto.open(null);
			// Establecemos su estructura propia como proyecto E-Col de
			// Servidor.
			UtilidadesProyecto.setNature(proyecto,"org.eclipse.jdt.core.javanature");
			UtilidadesProyecto.setNature(proyecto,
UtilidadesProyecto.NATURE_SERVIDOR);

			// En este caso la jerarquía de directorios.
			crearCarpeta(proyecto, "src");
			crearCarpeta(proyecto, "sesiones");
			UtilidadesProyecto.setEntradasClasspath(proyecto);

			try {
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_NOMBRECORTO_USUARIO,
detallesProyecto
							.getNombreCortoPersonal());
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_NOMBRE_USUARIO,
detallesProyecto
							.getNombrePersonal());
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_EMAIL_USUARIO,
detallesProyecto
							.getEmailPersona());
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_OTRAINFO_USUARIO,
detallesProyecto
							.getOtraInfoPersonal());
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_LOGIN_PAREJA,
detallesProyecto
							.getLoginPareja());
				UtilidadesProyecto.setPropiedadString(proyecto,
						UtilidadesProyecto.SCOPE_SERVIDOR,
						UtilidadesProyecto.KEY_PASS_PAREJA,
detallesProyecto
							.getPasswdPareja());
			} catch (BackingStoreException e) {
				MessageDialog
				.openError(
						workbench.getActiveWorkbenchWindow()
							.getShell(),
						"Ecol Plug-In: Proyecto fuente.",
						"Se producido errores almacenando los detalles del
proyecto, revise dichos parámetros.");
		}
		} catch (CoreException e) {
			MessageDialog.openError(workbench.getActiveWorkbenchWindow()
					.getShell(), "Ecol Plug-In: Server Project",
					"Debido a errores, el proyecto no se ha creado
correctamente.\n"
							+ e.getMessage());
```

```
                        return false;
                }

                return true;
        }

        /**
         * Método utilizado para crear una carpeta bajo el directorio raíz del
         * proyecto
         *
         * @param project
         *          Referencia al IProject donde se requiere hacer al carpeta.
         * @param nombre
         *          String nombre de la carpeta a crear.
         * @throws CoreException
         *           Excepción lanzada si algo ha ido mal en la creación del
         *           recurso.
         */
        private void crearCarpeta(IProject project, String nombre)
                        throws CoreException {
                IFolder folder = project.getFolder(new Path(nombre));
                if(!folder.exists())
                        folder.create(true, true, null);
        }

        /**
         * Inicializa el asistente con el workbench sobre el que se trabaja así como
         * los elementos seleccionados.
         *
         * @param workbench
         *          Repersenta el objeto IWorkbench.
         * @param selection
         *          Representa los elementos seleccionados.
         */
        public void init(IWorkbench workbench, IStructuredSelection selection) {
                this.workbench = workbench;
        }
}
```

APÉNDICE D. CONTENIDO DE CD-ROM

A continuación se muestra el contenido del cd-rom proporcionado junto a la documentación del proyecto.

Directorio	Contenido
./	Directorio raíz del CD. Contiene un fichero leeme.txt explicando la estructura del CD.
./Plugin Ecol	Contiene toda la estructura de directorios del proyecto para desarrollo.
./instalacion	Plugin instalable en la plataforma Eclipse.
./documentacion	Contiene toda la documentación asociada al proyecto en formato Word y PDF.
./documentacion/img	Directorio que contiene las imágenes utilizadas en la documentación.
./documentacion/uml	Directorio que contiene los diagramas del proyecto.
./presentacion	Directorio que contiene la presentación en Powerpoint.
./herram	Contiene los ficheros de instalación de las herramientas utilizadas para el desarrollo: Eclipse 3.2 y Java Development Kit 1.5.0