

Sistemas Distribuidos

Sistema básico de almacenamiento en la nube usando Java RMI

Práctica

Alumno: Buenaventura Salcedo Santos-Olmo
DNI: 5XXXXXX-B
Centro Asociado: 018000
Teléfono de contacto: 666xxxxxx
Email: bsalcedo8@alumnos.uned.es

Índice

1.- Presentación.....	3
2.- Introducción.....	4
3.- Algunas decisiones tomadas.....	5
4.- Esquemas y diagramas de la operativa y clases.....	8
5.- Desarrollos comentados: las clases e interfaces.....	17
6.- Ejemplos de prueba.....	25
7.- Conclusiones y mejoras.....	41

1.- Presentación.-

La práctica consiste en la creación de un entorno de nube de almacenaje de ficheros para la asignatura de Sistemas Distribuidos. Se desarrollará en Java, usando Java RMI para invocación de objetos remotos, RPC. El entorno de desarrollo ha sido Eclipse Neón, usando como sistema operativo Windows 10 , aunque para conocer su compatibilidad con Linux algunas clases se implementaron y probaron con la distribución Lubuntu 16.04 y Eclipse igualmente con resultados completamente satisfactorios. Todo funciona bajo localhost.

El orden correcto de ejecución sería primero el servidor, después el repositorio y después lanzar varios clientes, de no ser así podría haber errores en la ejecución, en la medida de lo posible se han tratado, si lanzamos clientes o repos sin ejecutar el servidor, mostrara una excepción personalizada finalizando los programas después de pulsar enter para continuar.

Estructura de la práctica: los ficheros con el código fuente se adjuntan en la carpeta SRC, las clases en la carpeta CLASS y los ficheros jar y bat en la carpeta JAR, en esta ultima carpeta se incluyen 3 ficheros de texto, fichero1.txt, fichero2.txt y fichero3.txt con los que hemos usado para pruebas. También se adjunta otra carpeta workscape, con la estructura generada en Eclipse, usando una estructura similar a la usada por Fermín en el vídeo, destacando la carpeta common, en la cual se incluyen las interfaces que se hacen publicas y algunas clases que son asadas en todo el proyecto.

La versión que se va a enviar esta dotada de persistencia implementada en los registros de clientes, repos, y ficheros del servicio de datos, una estructura lógica de ficheros, las sesiones quedan fuera, ahora bien, como no se exige en el enunciado me aventuraré a entregarla, en caso de alguna excepción no controlada, podría entorpecer la persistencia, y que los datos no fueran consistente. Creo que dotar el trabajo de persistencia facilitaría la corrección de la práctica, al no tener que por lo menos registrar clientes, repos y subida de ficheros, con los que trabajar. Pido que no se me penalice por los errores de la versión que entrego si se producen errores que pudiesen derivarse de la incoherencia en algún datos por un error producido por la persistencia. En caso de errores fatales, seria necesario borrar las carpetas con nombre “persistencia”, tanto en el servidor como en los repositorios.

También se ha dotado de un registro de actividad física, no solo se muestra lo que ocurre por la consola en el servidor, sino que cada día de ejecución se generaría un fichero de log con la actividad llamado logAAAAMMDD.txt acompañada cada linea de la hora. Estos ficheros de log se generan en la carpeta de ejecución del servidor al igual que las carpetas con los almacenes de datos persistentes comentados antes. Para ver los ficheros logs formateados usar editores como wordpad o notepad++ o cualquier editor de linux. Si usamos el bloc de notas de windows se vera mal sin saltos de linea.

2.- Introducción.-

En primera instancia, vamos a describir que hemos entendido del esquema de la operativa propuesta del sistema. Intentaremos dejar clara la responsabilidad de cada componente y sus restricciones. Después se enumerarán algunas decisiones tomadas, que también serán desarrolladas en su correspondiente sección. Es importante, desarrollar como hemos entendido el enunciado de la práctica, ya que de ahí parten las decisiones tomadas en el desarrollo.

El sistema parte de la idea básica en que el servidor cuelga unos servicios, mediante sus interfaces, es decir, una serie de métodos permitidos en forma de objetos remotos. Los clientes o repositorios deben buscar esos servicios, para recabar unos datos y colgar los suyos propios para establecer ambos una comunicación, consistente en el envío y recepción de los ficheros.

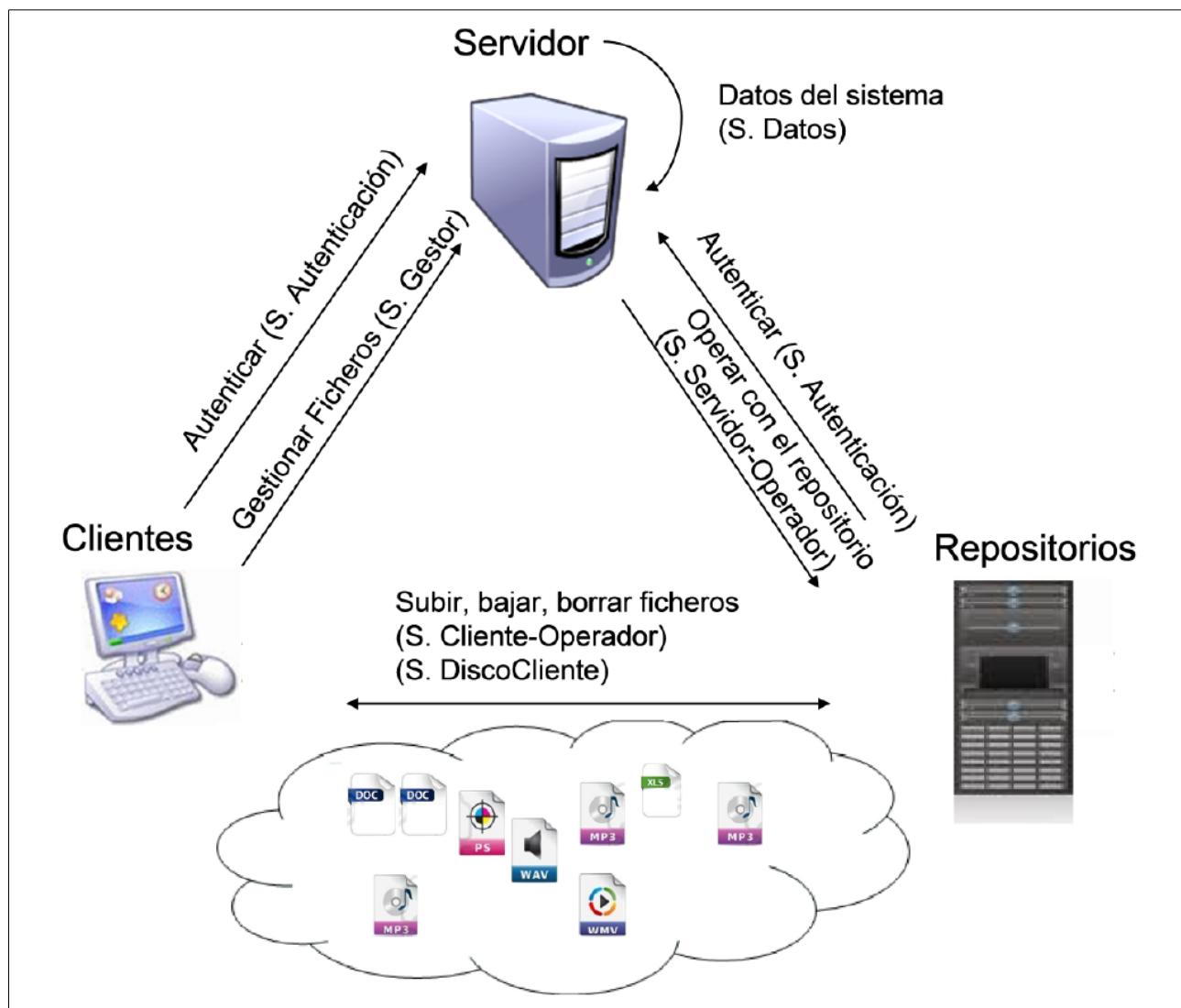


Imagen 1.- Esquema de la operativa del sistema.

Imaginemos 3 máquinas, aunque para el desarrollo de la práctica están todos en la misma máquina, localhost, como fue comentado en el foro de la asignatura.

El servidor cuelga tres servicios datos, autenticar y gestor.

El cliente solo puede usar el servicio autenticar y el gestor del servidor, NO PUEDE USAR el servicio de datos directamente.

Cuando un cliente se intenta registrar busca el servicio autenticar del servidor, autenticar usa el servicio de datos, para completar la operación. Aclarar que registrar y autenticar un cliente son dos operaciones distintas, un cliente no se puede autenticar si primero no se registra, y cuando un cliente se registra no termina el proceso autenticado, sino que después de registrarse debe autenticarse. La operación de intentar autenticarse hace uso del servicio autenticar del servidor.

Los repositorios solo pueden usar el servicio autenticar del servidor, NO PUEDE USAR ni el gestor ni el servicio de datos, este uso supuestamente está prohibido en la arquitectura.

Los repositorios levantan un servicio servidor-operador para escuchar la bajada de ficheros o la creación de una carpeta cuando se registra el cliente.

Los repositorios levantan otro servicio cliente-operador para escuchar la subida o el borrado de un fichero, la url de este servicio se la da el gestor cuando el cliente inicia esa operación. Para ello el gestor busca ese servicio una vez que el repositorio esta registrado y autenticado.

El cliente levanta un servicio discocliente para escuchar la recepción (bajada) de un fichero que él mismo inicia mediante el gestor, el propio gestor envía la url del discocliente mediante el servicio servidor-operador que levanto el repositorio.

El cliente si quiere subir o borrar, un fichero busca el servicio gestor y realiza la petición, el cual contesta al cliente con la url del servicio cliente-operador, que como esta levantado siempre va a estar esperando esa operación, con la url el cliente busca el servicio cliente-operador del repositorio para completar la operación.

3.- Algunas decisiones tomadas.-

Si el cliente quiere listar datos o clientes como lo hace no queda claro en el enunciado, pero lo más evidente es que lo haga a través del gestor, y que el gestor devuelva los datos en esa invocación con unos datos en forma de String o bien objetos que implementen serializable SINO ES ASÍ DA ERROR. Recordemos que el cliente no puede acceder al servidor de datos, esto debe de quedar claro.

El repositorio debe mantener una pequeña lista de sus clientes, es decir, sus carpetas, cuyo nombre sera el id único que obtiene al registrarse ese cliente, o un mecanismo similar y quizás tener un listado de los ficheros de cada carpeta, pero esto ultimo simplemente con listar lo que hay en la carpeta cumpliría el requisito del enunciado de la práctica y del menú de repositorio. Por

simplicidad esa lista fue una List declarada como static que pueda accederse desde el Servicio-SrOperador del repositorio ya que pertenecen al mismo paquete.

Si el cliente esta autenticado SIGNIFICA QUE SU REPO ESTA ACTIVA Y/O AUTENTICADA, no solo registrada, como se ha dicho registrarse no implica autenticación, por si hubiese varias repositorios, por lo tanto el cliente no necesita informar del repositorio que le corresponde en esta implementación, el repositorio se supone una vez autenticada que siempre esta funcionando y es único funcionando, si este repositorio no fuera el del cliente no nos habrían autenticado para el repositorio que nos asignaron al registrarnos, esto también esta comentado en el foro de la asignatura. Pero en la última versión que he realizado he propuesto que pudiese haber varios repositorios online, algo que nos da flexibilidad y escalabilidad al añadir más supuestas máquinas. Por tanto cada repositorio levanta dos servicios el cl-oeprador y el sr-operador añadiendo el id sesión del propietario. Esto dara seguridad a las sesiones del repositorio, porque??? sencillo, si fuese el caso que estuviesen sniffando tráfico para aprovechar sesiones abiertas estas no tendrían el mismo id sesión en caso de caída o desconexión y reconectado del cada repositorio, con lo cual serian sesiones distintas y dificultaría el aprovechamiento de esta propiedad, es decir no es lo mismo levantar los servicios con id unicos de repo que con los de sesión, y como el servidor conoce los id sesión de los repositorios siempre el gestor devolverá las url correctas al cliente para subir o borrar ficheros. Creo que es así como debe hacerse. Si vemos un ejemplo real, un banco online, por ejemplo, ofrece un id sesión al cliente y puede ocurrir que el cliente cierre la ventana sin pulsar el botón de desconexión, salida o logout. El servidor no sería informado de este hecho, facilitando un tiempo hasta que por inactividad el servidor de por concluida la sesión, ese tiempo sigue teniendo las credenciales de sesión validas, que si fueron sniffadas, pueden seguirse usando por un atacante al no haber todavía cerrado la sesión, pero esas credenciales no serán validas en la siguiente conexión. Sin embargo si en vez de id sesión fuesen id únicas de cliente, las credenciales SIEMPRE serían válidas en cualquier sesión, lo que no es deseable.

El sistema comentado en el párrafo anterior es igualmente usado con la autenticación de los clientes. La misma motivación nos ha facilitado la toma de el uso de id sesión al levantar servicios en vez de los id únicos de cliente que se usan al registrarse.

En el proceso de registro , servicio de autenticación genera un id que es de sesión pero que sera único y será almacenado por el servicio de datos en los almacenes, siendo su id único en toda su vida ligado a su nombre.

En el proceso de autenticación, el servicio de autenticación genera un id de sesión, que será usado mientras este activa esa sesión. El servicio de autenticación se comunica con el servicio de datos enviado el nombre y ese id de sesión. Ese id de sesión no es el mismo que el id único de registro. El cual se guarda en una hashMap volátil de sesiones, relacionado con su nombre de usuario, aunque podríamos haberlo relacionado directamente con su id único, que por otro lado se guarda en el almacén.

Usar id sesiones en vez de id únicos cuando se autentica un cliente o un repositorio, pienso que da seguridad, ya que esos id irán cambiando en cada sesión, impidiendo que alguna persona maliciosa quisiese usar ese id en sucesivas sesiones si por ejemplo no cerrásemos la sesión, de ahí la importancia real de siempre cerrar las sesiones por ejemplo las bancarias o de redes sociales. Si usásemos los id únicos con los que nos registramos esas sesiones serían siempre las mismas al

tener siempre el mismo id, no queremos eso. Por contra el uso de sesión y teniendo cuenta la arquitectura y no poder el id al realizar un envío o borrado de ficheros, hace que el cliente no pueda enviar el id único de su carpeta, esto es porque en el enunciado no se especifica claramente este hecho o no lo he entendido correctamente, pero bueno, con la url se envío este dato, dando lugar a una ñapa en toda regla como explicaremos en la sección de comentarios del código fuente.

Por otro lado, se han incluido la operación de desconexión tanto de los clientes como de los repositorios, de esta forma, podremos aplicar lo comentado anteriormente. El programa cliente cierra la sesión al salir con el número 7, pero nos permite autenticar a otro cliente distinto, no se cierra el programa, volviendo a levantar su propio discocliente, que también lo hemos ligado a su sesión que acaba de activar, por el mismo motivo de seguridad, es decir, en vez de levantar un objeto discocliente genérico para todos, cada cliente registrado levanta su propio "discocliente/XXXXXXX" siendo XXXXXXXX su id sesión de la sesión actual. Esto se verá en su clase correspondiente.

No se permiten duplicar los nombres de los clientes ni de los repositorios en los registro ni en las autenticaciones, esto lo notificara por pantalla el servidor y cada programa que lo haga. Igual sucede con las autenticaciones, podríamos haber optado por cerrar la sesión anterior o no tomar ninguna medida sino la de informar, por sencillez se tomo la segunda. Si el repositorio que se le asigno al cliente en el registro, no esta online, se denegará el acceso al sistema, si no hay repositorios online en el momento del registro, también se denegará la operación de registro, también se notificará este hecho.

Las salidas por pantalla de las colecciones o resultados, por simplicidad se expondrán los formatos de impresión pro defecto que se generan con eclipse para el método `toString()` o formatos lineales similares, **entre corchetes para cada elemento de la colección**. Estas colecciones usadas en la implementación ha sido `List` (o `ArrayList`) y `Map` (o `HashMap`), tal como se recomienda en el enunciado.

Si en el sistema hay dos ficheros con el mismo nombre, en principio no debe pasar nada, ya que cada registro del fichero (cada metadato) tiene un id único, el mismo que se usará en las peticiones de datos, para las operaciones, normalmente se presentaran como X.- nombre fichero, siendo la X el id, este hecho se informara en cada petición de datos.

Cada vez que un cliente descargue un fichero, se añadirá al final del nombre de ese fichero el id de sesión de ese cliente que corresponde con el discocliente de esa sesión obviamente. Es lo mismo que se uso en el ejemplo del foro de la asignatura con el uso de fichero. Así sabremos que de quien es cada fichero en caso de ejecución de dos clientes al mismo tiempo en la misma carpeta.

El borrado de un fichero implica la eliminación del metadatos en el servicio datos y en la lista que trabaja como almacén de la lista de ficheros que puede operar cada cliente cliente, Así mismo se elimina físicamente del repositorio.

Se ha implementado la operación de **compartir ficheros** aunque era opcional, pero que realmente tiene un calado en la propuesta bastante interesante. Este hecho ha llevado ligado una

modificación del código que estaba ya realizado, por no querer implementar esta característica en el sistema al ser opcional. Se añadió al metadatos un campo en forma de lista con los clientes con los que se ha compartido ese fichero y por otro lado otro almacén en el cual se almacena cada cliente los ficheros que puede descargar incluidos los compartidos. Ahora bien no se permite que un cliente al que le han compartido un fichero NO PUEDA BORRAR ese fichero, esto se informará en el servidor y en el cliente. Si el cliente tiene un fichero que comparte con otros usuarios y desea borrarlos, se eliminará de todas las colecciones y entradas de los usuarios con quien se compartió ese fichero. Podríamos haber negado esta operación al estar compartido, pero al fin y al cabo el fichero es del cliente que lo compartió puede hacer con él lo que quiera.

Algunas decisiones tomadas, se comentarán en las explicaciones del código, incluso también en los comentarios del propio código fuente.

4.- Esquemas y diagramas de la operativa y clases.-

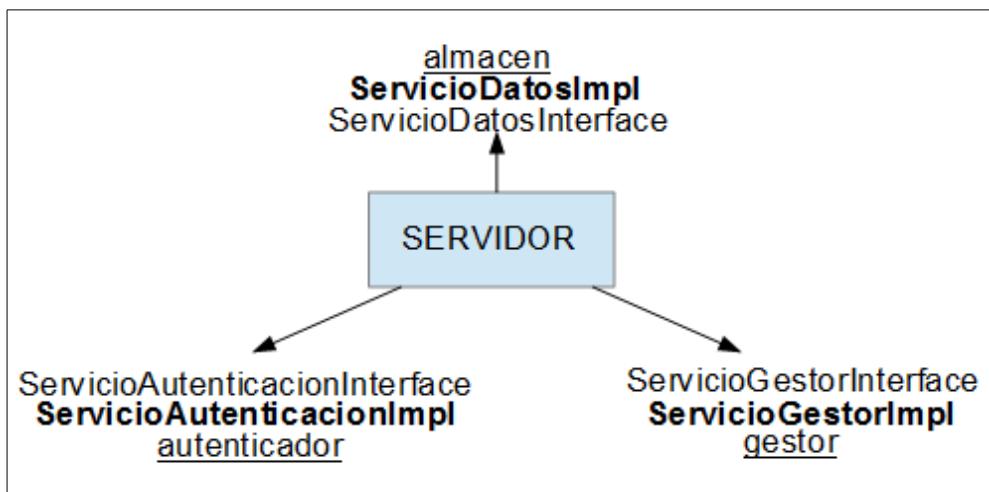


Imagen 2.- Servicios, interfaces y cadenas que levanta el servidor.

El servidor levanta tres servicios a través de las tres interfaces, ServicioDatosInterface, ServicioAutenticacionInterface y ServicioGestorInterface, que son implementados con ServicioDatosImpl, ServicioAutenticacionImpl, ServicioGestorImpl, los nombres de los objetos en el programa principal son objetoDatos, objetoAutenticador y objetoGestor, que usan las cadenas **almacen**, **autenticador** y **gestor**.



Imagen 3.- Operativa de autenticación del repositorio.

Cuando el repositorio se registra o autentica usando el servicio de Autenticación, buscando

el autenticador. Este servicio de autenticación busca el servicio de Datos, el almacen, del propio servidor en este caso, pero podría estar porque no en otra máquina distinta. El registro genera un id sesión, que será el id único para esa repo. Cuando se autentica el repositorio que, previamente, tiene que ser registrado, recibe un id sesión, el cual lo genera el autenticador, y se lo pasa al servicio de Datos. Sin una repo activa, no podrían después registrarse o autenticarse los clientes.

Una vez autenticado el repositorio levanta 2 servicios Sr-Operador y Cl-Operador, que veremos enseguida.

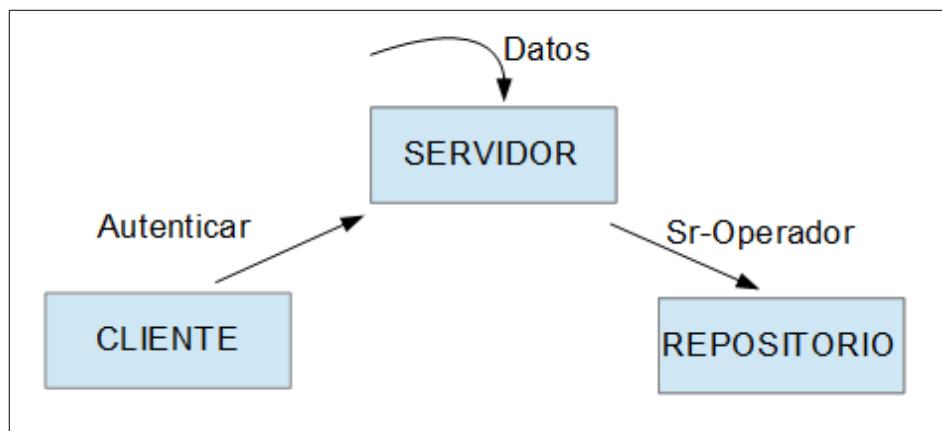


Imagen 4.- Operativa de autenticación del cliente 1.

Cuando el cliente quiere registrarse o autenticarse lo hace buscando el servicio de Autenticacion del servidor, generando un id sesión, que en el caso del registro se convertirá en el id único para ese cliente, si la operación es de autenticación ese id sesión solo será valido para la sesión activa. El autenticador busca el servicio de Datos y pasa ese id sesión, que obra en consecuencia si es para registro o autenticación, almacenándolo en las tablas junto con el nombre del cliente.

En el caso de un cliente registrándose, se pide un repositorio si no hay ninguno disponible se deniega la operación de registro, y si lo hay se busca el servicio SR-Operador para que se cree una carpeta con el Id único del cliente. Este par cliente-repositorio es almacenado por el servicio de Datos.

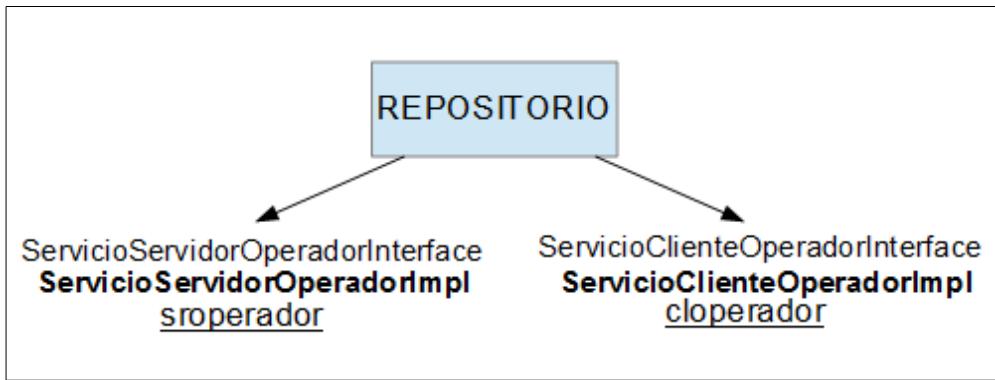


Imagen 5.- Servicios, interfaces y cadenas que levanta el Repositorio al autenticarse

Cuando el repositorio se autentica publica dos servicios mediante las dos interfaces, **ServicioServidorOperadorInterface**, que se encarga de crear las carpetas de cliente que le indica el servidor y de bajar ficheros, y **ServicioClienteOperadorInterface**. Estas interfaces son implementadas por el Repositorio mediante las clases **ServicioServidorOperadorImpl** y **ServicioClienteOperadorImpl**, los objetos son colgados mediante las cadenas **sroperador** y **cloperador**. El servicio Cl-Operador realiza dos operaciones la de subir y borrar fichero.

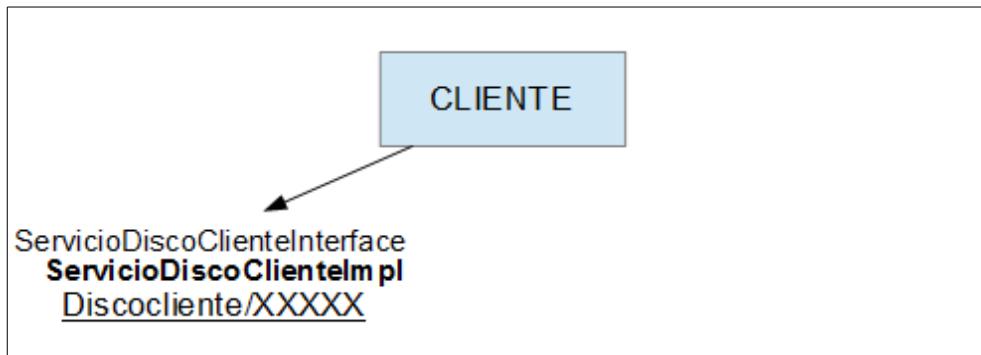


Imagen 6.- Servicio, interfaz y cadena que levanta el Cliente al autenticarse.

Cada cliente que se autentica cuelga en cada sesión el servicio DiscoCliente para descargar ficheros sin la cargar al servidor con esta operación ya que se realiza entre cliente y repositorio directamente. El servidor tan solo facilita la comunicación entre ambos, validando y registrando la operación, como se verá enseguida. En cada sesión la URL del servicio DiscoCliente de cada Cliente cambia usando el **idsesión(XXX....X)** en vez del **idUnico**, dando seguridad entre sesiones, de cada cliente, y permitiendo distinguir cada DiscoCliente de cada cliente autenticado si hay varios en ese instante autenticados.

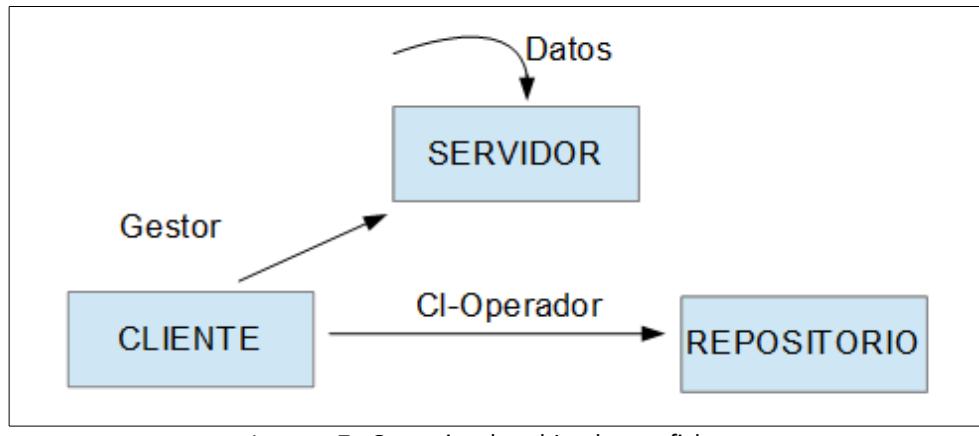


Imagen 7.- Operativa de subir o borrar fichero

Cuando un cliente necesita subir un fichero o borrar un fichero, lo hace buscando el servicio Gestor del servidor y este le devuelve la URL del servicio Cliente Operador del repositorio. El Gestor pide al servicio de Datos que guarde la operación devolviéndole el idUnico del cliente después de haber actualizado esa operación en las tablas de Datos e informar por pantalla de este hecho en la consola del servidor (podríamos haber, en ese punto, guardo un log en fichero, pero verlo en pantalla es interesante así no hay que abrir ficheros de log y se va viendo fácilmente). El repositorio debe guardar.

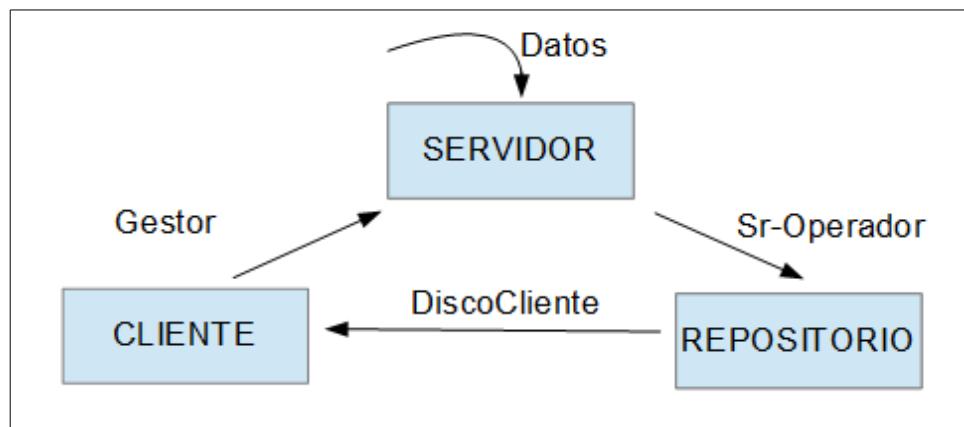


Imagen 8.- Operativa de bajar un fichero.

Para bajar un fichero del repositorio, el cliente busca el servicio Gestor de nuevo como antes, pero esta vez el Gestor busca el servicio Sr-Operador del Repositorio y le pasa la URL con el idsesión que corresponde del DiscoCliente adecuado, Sr-Operador es quien inicia la transferencia real del fichero hacia el DiscoCliente, haciendo eso mismo buscando ese servicio del cliente, librando de esta carga, igual que hizo el CI-Operador del apartado anterior, al servidor en dichas transferencias, realizando solamente el servidor la función de comunicación entre ambas entidades al facilitar los datos para que ellos se comuniquen.

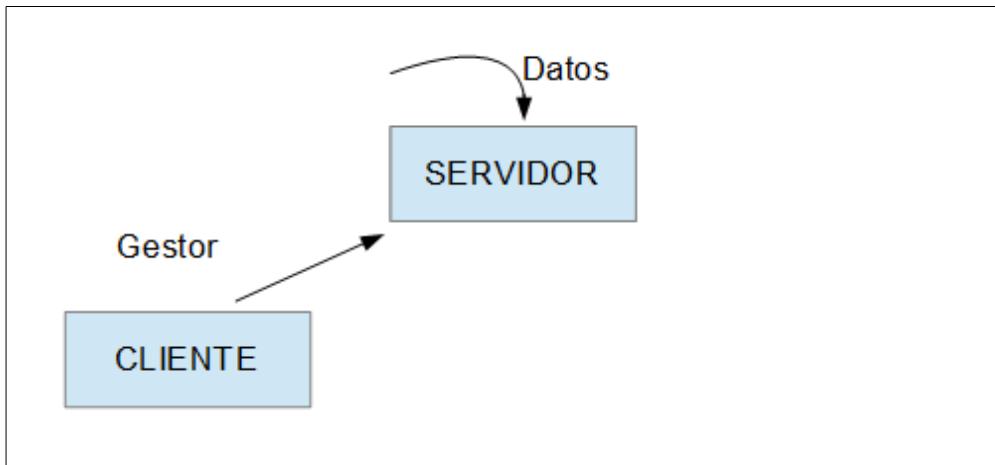


Imagen 9.- Listados y operaciones del cliente.

El cliente puede pedir listados o compartir ficheros, en el enunciado no me quedo claro donde ubicar estar operaciones, pero el sitio mas conveniente es que el cliente busque el servicio Gestor para que le facilite listados de ficheros y clientes, permitir la operación de compartir ficheros. Por tanto el gestor buscara en el servicio de Datos, esta información y operación. El servicio de Datos en sus tablas mantendrá toda esta información.

A continuación vamos a ver los atributos que poseen algunas clases, comenzaremos por la que mantiene los ids, los ficheros en forma de metadatos, las sesiones, los ficheros que le comparten a cada cliente, en definitiva el almacen de datos que le corresponde al servicio de Datos implementando la interfaz mediante ServicioDatosImpl.

```
//Estructuras que mantiene el almacen de Clientes y Repositorios registrados PERSISTENTES
private Map<Integer, String> almacenIdCliente = new HashMap<Integer, String>();
private Map<String, Integer> almacenClienteId = new HashMap<String, Integer>();
private Map<Integer, String> almacenIdRepositorio = new HashMap<Integer, String>();
private Map<String, Integer> almacenRepositorioId = new HashMap<String, Integer>();
private Map<Integer, Integer> almacenClienteRepositorio = new HashMap<Integer, Integer>();
```

Imagen 10.- Tablas de ids únicos de clientes, repositorios y pares cliente-repositorio.

El atributo **almacenIdCliente** y **almacenClienteId**, declarado como Map mantiene el id y el nombre de cada cliente y su inverso, de tal forma que podamos recuperar la información con el método get(key) del Map, e incluso podamos comprobar su existencia con la devolución del booleano procedente de containsKey(key) o containsValue(valor). Recordemos en este punto que cuando un cliente se registra en ese momento el servicio de autenticación genera un id de sesión que será asignado como id único a ese cliente. Este id y el nombre del cliente, debe ser introducido de una forma atómica en ambos maps. Ahora bien si no hay repositorio ONLINE estas tablas no serán actualizadas.

El atributo **almacenIdRepositorio** y **almacenRepositorioId**, igual que los anteriores son declarados como Map y funcionan de la misma forma para el registro del repositorio.

El **almacenClienteRepositorio**, también es un Map que tiene las parejas de id únicos de cada cliente a que repositorio pertenece. Este atributo debe ser actualizado cada vez que se registra

correctamente a un cliente.

```
//en los metadatos esta a quien se le ha compartido cada fichero.  
private List<Metadatos> almacenFicheros = new ArrayList<Metadatos>();  
//necesitamos tambien a cada cliente quien le ha compartido  
private Map<Integer,List<Integer>> almacenClienteFicheros = new HashMap<Integer,List<Integer>>();
```

Imagen 11.- Lista de ficheros y tabla de los ficheros que le han compartido a cada cliente.

Metadatos es una clase que contiene el id único de cada fichero, solucionando así el problema que podría darse si dos ficheros tuviese el mismo nombre.

```
private static int cont;//contador de instancias, para tener ids unicos de los metadatos  
private int id; //id de la colección  
  
private int idCliente; //identificador del propietario original del fichero  
private int idRepositorio;  
private String nombreFichero; //nombre del fichero  
private long peso; //peso del fichero en bytes  
private long checksum; //suma de chequeo de los bytes del fichero  
private List<Integer> compartidoCon;
```

Imagen 12.- Atributos de la clase Metadatos

cont es el contador de instancias para generar id únicos correlativos, cada vez que se llama al constructor se incrementa este contador y se asigna al **id** que es el id único de cada fichero que hay en los repositorios.

idCliente indica el id único del propietario del fichero

idRepositorio indica el id único del repositorio en el que esta guardado el fichero

nombreFichero contiene el nombre del fichero en cuestión

peso y **checksum** son el tamaño en bytes y la suma de comprobación de los bytes del fichero.

compartidoCon es una lista de los clientes con los que esta compartido ese fichero, el propietario no esta incluido en la lista.

Todos los atributos tiene sus métodos get y set.

Volviendo a la lista de Metadatos, que corresponde al atributo **almacenFicheros**, pues mantiene la lista de todos los ficheros que están en todos los repositorios.

Por otro lado el **almacenClienteFicheros**, es un Map que cuando llamamos al método **get(idCliente)** de la colección nos devuelve la lista de los ficheros que le han compartido a ese cliente.

Ambas colecciones que se ven en la imagen 11 son actualizadas con cada fichero que subimos o borramos, la operación es un tanto más compleja cuando se trata de la compartición de un fichero o el borrado por parte del propietario de un fichero que ha sido compartido.

```
//Estructuras que mantienen las autenticaciones VOLATILES
private Map<Integer, String> sesionCliente = new HashMap<Integer, String>();
private Map<String, Integer> clienteSesion = new HashMap<String, Integer>();
private Map<Integer, String> sesionRepositorio = new HashMap<Integer, String>();
private Map<String, Integer> repositorioSesion = new HashMap<String, Integer>();
```

Imagen 13.- Atributos usado para sesiones.

Las tablas **sesiónCliente** y **clientesesión**, mantiene la sesiones activas, es decir, guardan los pares de forma atómica el idsesión y el nombre del cliente y su inversa, podríamos haber usado el id único en vez del nombre del cliente. No ha habido nada especial en la decisión.

De igual forma trabajan **sesiónRepositorio** y **repositoriosesión** con los id de sesión y los nombres de los repositorios.

Bien, en combinación las tablas de sesiones y la tablas almacén de la imagen 10, podemos elaborar los listados de clientes y repositorios, ofreciendo la información de todos los clientes (y repositorios) que hay registrados y cuales de ellos están ONLINE o desconectados OFFLINE, algo interesante por ejemplo para quien maneja el servidor, visualizar todos los clientes y cuales están conectados. Para las presentaciones de datos por pantalla se ha optado por el formato básico que nos ofrece Eclipse cuando pedimos que autogenere el método `toString()` o devolviendo cadenas con un formato sencillo y similar, así no complicamos el código y facilitamos la corrección.

Menciono de paso en este apartado que si quisésemos tener persistencia en los datos, sería conveniente guardar en ficheros físicos los almacenes, las sesiones podríamos descartarlas obligando a autenticarse a los clientes que hubiese conectados en alguna caída del sistema.

Vayamos ahora a ver un atributo de la clase Repositorio, que también debería ser persistente y que se actualiza con cada cliente nuevo.

```
//lista de carpeta que se van a ir creando en la repo, hace falta agregar persistencia
//la visibilidad es label paquete
static List<String> listaCarpetas = new ArrayList<String>();
```

Imagen 14.- Atributo del repositorio para mantener la lista de Carpetas de ese repositorio.

La lista **listaCarpetas**, es una lista con los id de las carpetas que hay en el repositorio, la cual se devuelve con la opción de listar del menú de el repositorio. No se han mantenido, por ejemplo, los nombres de los ficheros, ya que simplemente viendo el contenido de cada id único de listaCarpetas con el método `list` de File, pasándole al constructor la ruta de la carpeta, seria suficiente para ver ese listado.

```

//atributos para buscar el servicio de autenticacion del servidor
private static int miSesion=0;
private static int puerto = 7791;
private static ServicioAutenticacionInterface servidor;
private static String direccion = "localhost";

//atributos para levantar los servicios Servidor-Operador y Cliente-Operador
private static int puertoServicio = 7792;
private static Registry registryServicio;
private static String direccionServicio = "localhost";

```

Imagen 15.- Atributos de los servicios de las clases similares en servidor,cliente y repositorio.

La Imagen 15 nos muestra los atributos para conectar y levantar con los servicios, los cuales se los podríamos pasar como argumentos en las llamadas al jar del servidor, a clientes o repositorios, o solicitándolos desde dentro del main() o porque no leyendo de un fichero de configuración que acompañase al jar.

Veamos a continuación relaciones entre clases e interfaces con sus atributos y métodos de los paquetes, se ha usado ObjectAid UML Diagram para Eclipse. Se han añadido las interfaces del paquete common para ver la interacción entre las interfaces publicadas. Será necesario usar el ZOOM.

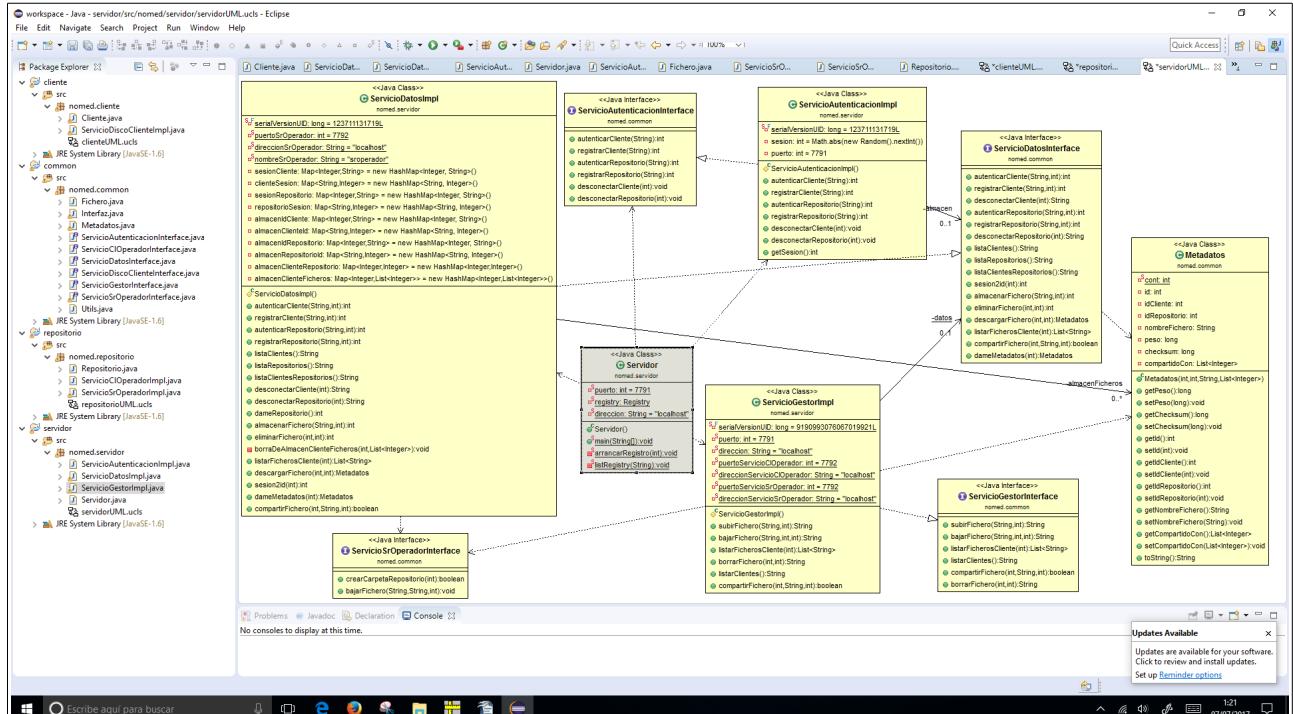


Imagen 16.- Clases, interfaces, atributos y métodos del paquete servidor

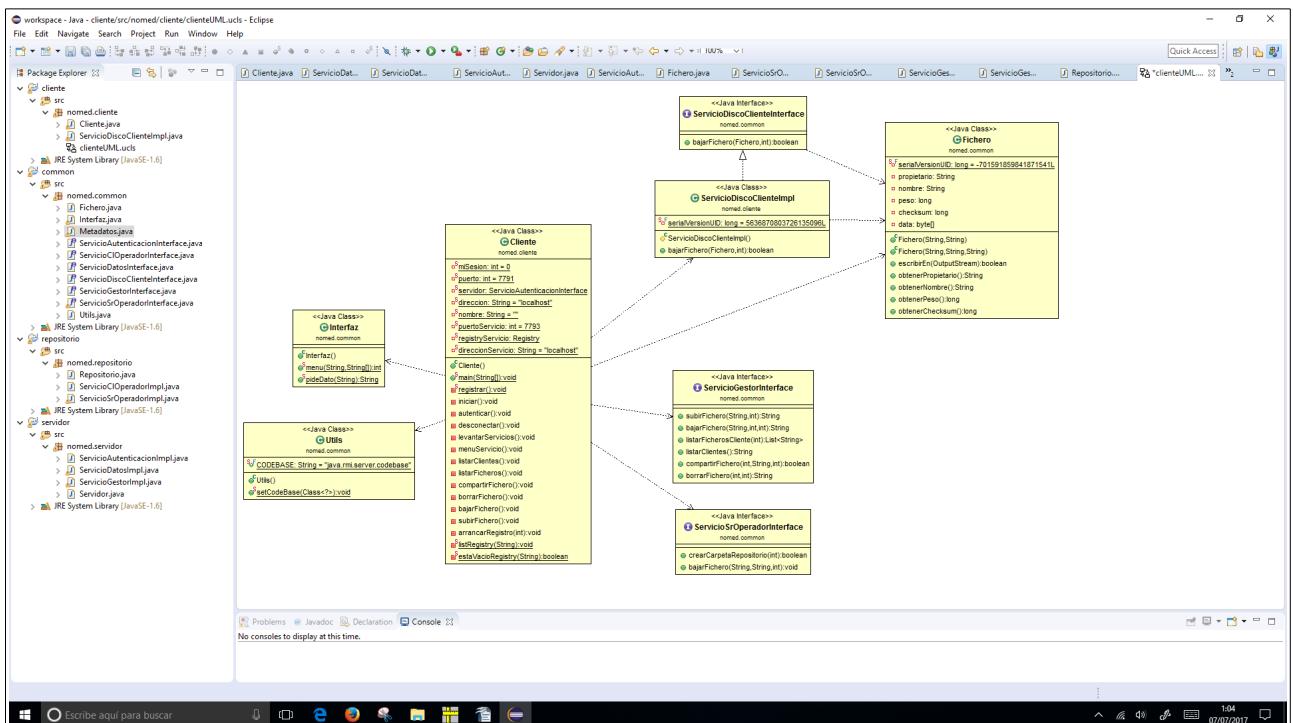


Imagen 17.- Clases, interfaces, atributos y métodos del paquete cliente.

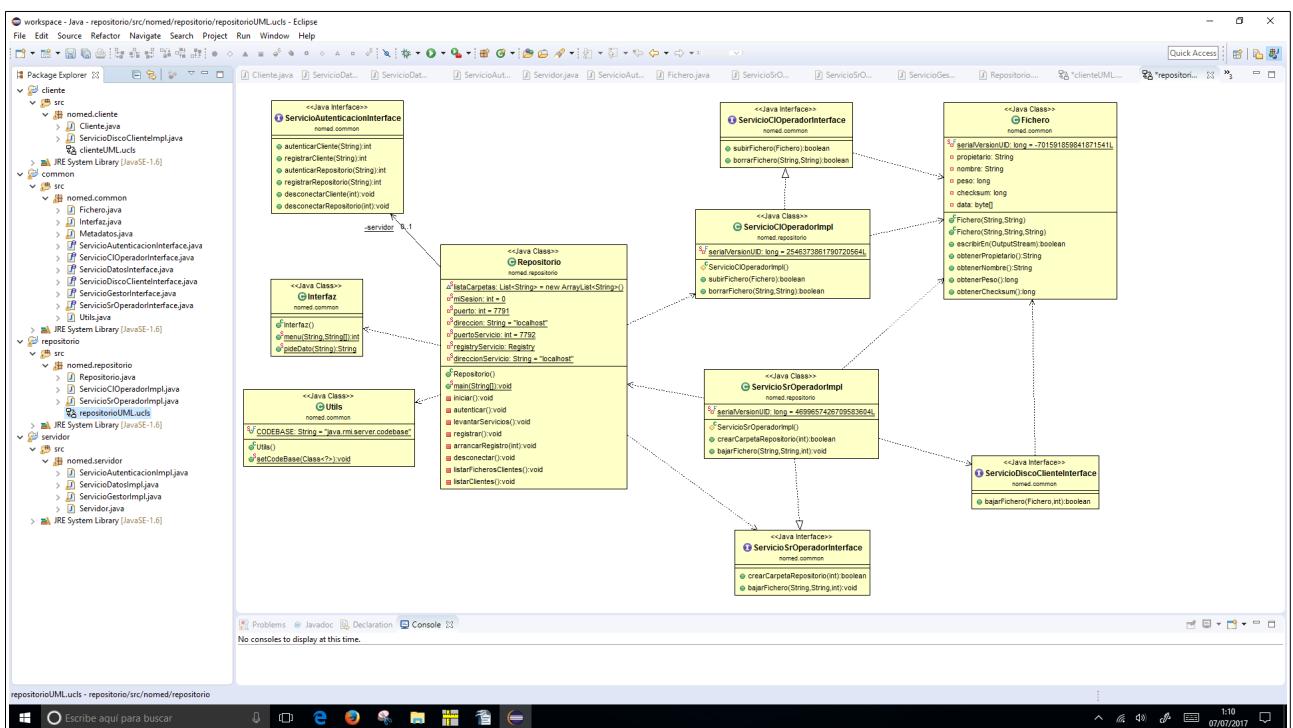


Imagen 18.- Clases, interfaces, atributos y métodos del paquete repositorio.

Nota.- No se ha incluido en los diagramas la clase Persistencia, ya que fue incluida al finalizar la práctica como mejora.

5.- Desarrollos comentados: las clases e interfaces.-

Servidor.java

Lo primero que hace el servidor es arrancar el Registry intentando buscarlo en el puerto asignado y si no lo encuentra lo crea, para ello RemoteException es controlada. Ubicamos la ruta de las interfaces para ser accesible en el resto del programa con Utils.setCodeBase() y después con Naming.rebind colgamos las cadenas de los servicios almacen, autenticador y gestor. Usando el formato usado en el texto complementario y como se recomienda en el enunciado.

```
URL_nombre="rmi://" + ip + ":" + rmiport + "/" + nombre_servicio + "/" + identificador_unico;  
Naming.rebind(URL_nombre, objExportado);
```

Después de colgar los servicios lo confirmamos imprimiendo con listRegistry() (método usado en el ejemplo de uso de la clase Fichero del foro de la asignatura) los servicios colgados, para después entrar en el menu del servidor que nos permitirá listar los clientes que hay registrados indicando cuales estan online, lo mismo con los repositorios y las parejas cliente-repositorio . Al colgar los tres servicios, creamos tres objetos de las clases de las interfaces publicas y que podemos usar aunque esten colgados. En este caso usaremos el objetoDatos del servicio de Datos.

Cuando se decide dejar de usar del programa saliendo con la opción 4, el programa realiza los Naming.unbind() correspondientes, e intenta quitar el objeto remoto con UnicastRemoteObject.unexportObject() tratando la excepción que pudiese provocar al intentar cerrarlo si no tuviese éxito.

ServicioDatosInterface , ServicioGestorInterface y ServicioAutenticarInterface

Son las interfaces que publica el servidor a continuacion veremos como son implementadas en el paquete del servidor.

ServicioDatosImpl

Veamos primero esta clase que implementa ServicioDatosInterface, por ser la que guarda los datos. Los atributos ya se comentaron anteriormente, vayamos a los métodos.

autenticarCliente() recibe un nombre y un idsesión e intenta realizar la autenticación, devolviendo 0 si ya esta registrado, -1 si su repositorio no esta online, -2 si no esta registrado y el idsesión en otro caso. Actualiza las tablas, sesiónCliente y clientesesión, para que se mantengan las sesiones que serán posteriormente consultadas.

registrarCliente() recibe un nombre para registrar al cliente y un idsesión, que sera usado como id único, pide un repositorio al método dameRepositorio() si este repositorio es 0 se cancela el registro al no haber repositorios disponibles, no basta con que haya repositorios registrados sino que deben estar online, para poder atender los servicios. Los datos se guardan en el almacenIdCliente, almacenClienteld, en almacenClienteRepositorio y se crea una entrada en

`almacenClienteFicheros` para saber que ficheros le han compartido, que de momento tiene la lista vacía. Tambien busca el servicio Servidor-Operador del Repositorio para que cree una carpeta con el id único del cliente el cual contesta devolviendo un booleano a true en caso de éxito.

`autenticarRepositorio()` recibe un nombre y un idsesión e intenta autenticarlo, devolviendo 0 si ya esta autenticado, -1 si la repo no esta registrada y el idsesión si todo es correcto, añadiendo a las tablas, sesiónRepositorio y repositoriosesión para que se puedan mantener las sisiones de los repositorios y poder hacer consultas.

`registrarRepositorio()` recibe un nombre para registra el repositorio y un idsesión, que será usado como id único, comprueba si el nombre existe en el `almacenRepositoriold` y si no esta lo crea añadiendo la entrada a `almacenRepositoriold` y en `almacenIdRepositorio`. Este metodo no autentica la repo, solo la registra como nueva repo.

`listaClientes()` devuelve un string en formato simple de la lista de clientes registrado indicando cuales de ellos estan ONLINE u OFFLINE usando la tabla de sesiónCliente, su nombre e idUnico.

`listRepositorio()` devuelve un string en formato simple de la lista de repositorios registrados indicando cuales de ellos esta ONLINE u OFFLINE usando la tabla de sesiónRepositorio.

`listaClientesRepositorios()` devuelve un string en formato simple de los emparejamientos de la tabla `almacenClienteRepositorio` con el nombre y los id únicos de ambos.

`desconectarCliente()` y `desconectarRepositorio()`, desconectan a ambas entidades eliminando las entradas de cada tabla de sesión y de su inversa y devuelve el nombre que ha sido desconectado.

`dameRepositorio()` devuelve el id único de la primera repo que encuentre que este online, 0 sino hay ninguna online.

`almacenarFichero()` recibe el nombre de un fichero y un idsesión del cliente, crea una nueva entrada en la colección de **Metadatos, que es donde se almacenan los datos de cada fichero, con su nombre, id del cliente, id del repositorio y una lista de id únicos de ficheros con los que es compartido, la cual de momento está vacia. Cada fichero tiene un id único que corresponde con el numero de instancias que se ha creado.**

`eliminarFichero()` borra el id del fichero pero hay que pasarle la idsesión para que el método compruebe si el fichero es de ese cliente. Si es suyo borra las entradas de `almacenClienteFicheros` en caso de que la lista de ese fichero contenga clientes con los que haya sido compartido. Si un fichero esta compartido y quien intenta borrarlo no es propietario devuelve -1 indicando que esta intentado borrar un fichero que no puede borrar, solo el propietario puede borrar un fichero compartido.

`borraDeAlmacenClienteFicheros()` le pasamos un id de fichero y una lista de clientes con quien ha sido compatido y recorre la tabla `almacenClienteFichero` seleccionando a cada cliente de la lista y eliminando los id de fichero de su lista de ficheros que le han compartido. Este método privado es usado por `eliminarFichero()`.

listarFicherosCliente() recibe un id de cliente y devuelve un List<String> con el id de cada fichero, el nombre y en caso de ser un fichero compartido lo indica diciendo quien se lo comparto.

descargarFichero() devuelve un objeto Metadatos que es serializable con los datos del fichero, despues de comprobar si el fichero es el cliente o le ha sido compartido.

dameMetadatos() devulve un objeto Metadatos sin realizar ninguna comprobación de propiedad.

compartirFichero() comparte el fichero con el id que le pasamos al cliente con el nombre que le enviamos y comprobamos con el idsesión que ese fichero es del propietario , si todo es correcto la llamada devolvera true y false en caso contrario. La compartición del fichero requiere modificar la lista de la tabla compartidoCon de los Metadatos del almacenFicheros, además de añadir un entrada en la lista del ficheros compartido del cliente en la tabla alamcenClienteFicheros.

ServicioAutenticacionImpl

Esta clase implementa la interface ServicioAutenticacionInterface, esta clase busca en su constructor el servicio de Datos e intenta registrar, autenticar y desconectar a los repositorios, también devuelve en la consola del servidor los mensajes de error que pudiera ocasionar tales procesos. **Cada método genera un id de sesión que se pasa al servicio de Datos en cada llamada excepto en el proceso de desconexion que es el cliente o repo quien suministra el id de sesión.**

registrarCliente() le pasamos el nombre del cliente y devuelve al cliente el -1 si no hay repositorios online para hacer el registro, 0 si el cliente ya esta registrado y el idCliente si ya el proceso se ha completado con éxito. Para ello usa el objeto almacen del servicio de Datos. Solo registra el cliente no lo deja autenticado.

autenticarCliente() le pasamos el nombre del cliente y devuelve -2 si no esta registrado, -1 si su repo no esta online, 0 si ya tiene sesión abierta y el id de sesión si se ha autenticado correctamente, igualmente informa en la consola del servidor. Podríamos haber devuelto el id único desde el servicio de Datos y este al cliente ese id único, pero por motivos de seguridad en las sesiones es mas interesante devolver este id, aunque esto entraña un pequeño problema que se comentara en el servicio Gestor y en las conclusiones de esta memoria. Si este supuesto no es correcto me gustaria saberlo.

registrarRepository() le pasamos el nombre del repositorio y devuelve 0 si se intenta registrar otra repo con el mismo nombre y el id en caso contrario, para ello hace uso del Servicio de Datos.

autenticarRepository() le pasamos el nombre del repositorio y devuelve 0 si el repositorio ya esta autenticado. Se supone que solamente hay un repositorio trabajando, pero bueno, ya que estamos podríamos usar varios on unas modificaciones minimas, levantando los servicios del repositorio, Servidor-Operador y Cliente-Operador, con el id de sesión del repositorio de la misma forma que se ha hecho con el DiscoCliente de cada cliente.

desconectarCliente() y desconectarRepository() reciben el id de sesión de cada cliente o repo y

usan el objeto almacen nuevamente para desconectarse del sistema informando del éxito o no al consola del servidor.

getsesión() devuelve el siguiente id de sesión para repo o cliente incrementando en una unidad el atributo de clase sesión que se generaba con un aleatorio entero positivo.

ServicioGestorImpl

Implementa la interface que publico el servidor con ServicioGestorInterface. Se encarga de las peticiones de subir, bajar, borrar, compartir y de solicitar listados de clientes y de ficheros. Para ello busca el servicio de Datos y ademas debe conocer las direcciones en formato rmi de los servicios del Repositorio y del DiscoCliente. Hemos metido en esta clase las peticiones de los listados, ya que en el enunciado no se menciona quien debe hacerlo y como ya hemos dicho en alguna ocasión, el esquema de la arquitectura, no permite el acceso al servicio de Datos directamente del cliente.

subirFichero() recibe el nombre del fichero y el idsesión, y devuelve la URL del repositorio, para que el cliente pueda buscar el servicio Cliente-Operador. Busca el servicio de Datos y le pasa los datos confirmando que se ha agregado.

bajarFichero() recibe la URL del servicio DiscoCliente, el id único del fichero y el id sesión. Busca el servicio de Datos y envia el id del fichero y el id sesión mediante el método descargarFichero, si todo va bien recibimos los metadatos para componer el nombre y la carpeta del fichero junto con la URL del cliente, que pasaremos al metodo bajarFichero después de buscar el servicio ServidorOperador, para que se inicie la transferencia directa entre el Repositorio y el DiscoCliente.

listarFicheros() recibe el id sesión del cliente, busca el servicio de Datos y para solicitar la lista de ficheros cliente a quien pertenece ese id de sesión, la cual devuelve al cliente. Recordemos que recibimos la lista con los ficheros que le han compartido a ese cliente.

listarClientes() nos devuelve la lista de los clientes después de buscar de nuevo el servicio de Datos y usar el método listarClientes(), que recordemos nos devuelve una cadena con la lista de clientes registrado indicando cuales estan online y cuales no.

compartirFichero() recibe el id del fichero, el nombre del cliente a quien compartir el fichero y el id sesión del cliente, buscando el servicio de Datos pide los Metadatos del fichero para componer el mensaje de confirmacion por consola y hace la peticion de particion con el metodo compartirFichero, para que compruebe si todo esta bien, es decir, si el fichero que se comparte es del cliente y si el cliente final existe, el cual devulve un booleando si lo ha podido compartir.

Repositorio.java

Cuando se lanza el programa busca el servicio de Autenticacion, para poder registrar o autenticar el repositorio. Cuando se autentica arranca el Registry y cuelga los dos servicios mediante la interfaces publicas, ServicioServidorOperadorInterface y ServicioClienteOperadorInterface, que son implementados por ServicioServidorOperadorImpl y el ServicioClienteOperadorImpl, despues muestra un menu pequeño menu con las 2 operaciones de lsitar carpetas y los ficheros de la carpeta de un cliente, identificadas con el id único del cliente. Cuando salimos, elimina las cadenas de los servicios biddenados anteriormente.

iniciar() busca el servicio de Autenticación del servidor y muestra un menu para registrar o autenticar un repositorio. Este metodo iniciar() lo lanza el constructor del repositorio.

registrar() intenta el registro del repositorio enviando el nombre, recibira 0 si la repo no puede registrarse porque ya existe ese nombre, el id sesión y unico si todo a ido bien. Recordemos que el registro no deja autenticado el repositorio.

autenticar() envia el nombre del repositorio al servicio de Autenticacion mediante autenticarRepositorio() y recibe -1 si el nombre del repositorio no existe, 0 si ya esta autenticado y el id sesión si todo ha ido bien, en cuyo caso, ejercuta el metodo levantarServicios().

levantarServicios() levanta el Registry y cuelga los servicios mediante las interfaces publicas ServicioServidorOperadorInterface y ServicioClienteOperadorInterface, usando las cadenas, sroperador y cloperador. A continuaucion muestra el menu del repositorio para listar las carpetas y los fichero de la carpeta que elijamos. Cuando queremos terminar la sesión del repositorio enviamos al Servicio de datos la peticion de desconesion para informar el fin a traves del servidio de Autenticacion que este as u vez lo hace notificandolo al servicio de Datos. Despues elimina la cadenas de los Servicos y elimina el Registry y vuelve al menu principal en espera de registrar o autenticar otro repositorio. Aunque según lo comentado en el foro de la asignatura, solo habria un repositorio online siempre.

arrancarRegistro() funciona igual que en el servidor.java.

desconectar() hace del servidor de Autenticacion para pedir la desconexion del sistema.

listarFicherosClientes() comprueba la lista de carpetas indexadas si esta vacia lo notifica sino pide el nombre de la carpeta a mostrar e intenta leer los ficheros fisicos de la carpeta, aunque los ficheros los podríamos **haber indexado en una pequeña colección, lo cual seria más rapido que tener que leer los ficheros físicos**. Realmente hubiera sido una buena opcion. O incluso podríamos haber colgado otro servicio para consultar estos datos. Nos conforramos con la lista de las carpetas de la repo.

listarClientes() simplemente imprime esa lista de carpetas que el un atributo de clase de Repositorio, declarada como public static para que sea único y accesible dentro del paquete del repositorio.

ServicioSrOperadorInterface.java y ServicioClOperadorInterface.java

Son la interfaces publicas para colgar el objeto remoto. Las implementan las siguientes clases del repositorio.

ServicioSrOperadorImpl.java

Esta clase implementa ServicioSrOperadorInterface, que es la interfaz publica del servicio sroperador. Tiene dos metodos para crear la carpeta del cliente en el repositorio y bajar un fichero del repositorio.

crearCarpeta() recibe el id único del cliente y crea una carpeta en disco. Además index esta carpeta en un lista static del Repositorio. Devuelve un booleano si todo ha ido bien. Además imprime el mensaje en la consola del Repositorio con el path completo.

bajarFichero() recibe la URL del DiscoCliente, el nombre del fichero y el id único del cliente. Busca el servicio Disco del Cleinte del cliente en cuestion para enviarle el fichero sin que intervenga el servidor.

ServicioClOperadorImpl.java

Esta clase implementa ServicioClOperatorInterface, Tiene dos metodos subirFichero y borrarFichero.

SubirFichero() recibe un objeto Fchihero con el fichero y usa los metodos de la clase para conseguir la ruta del propietario y el nombre del fichero y colocarlo en al carpeta correcta, devulve booleano para indicar que todo ha ido bien. La transferencia se realiza entre cliente y repositorio sin intervencion del servidor, para no sobre cargar.

BorrarFichero() recibe el nombre del fichero y la carpeta donde ubicado apra borrar el fichero, devuelve booleano para indicar que todo ha ido bien.

Cliente.java

Lo que hace es buscar el servicio de Autenticacion y despues lanza un menu para registrar o autenticar al cliente. Una vez autenticado muestra un menu con las operaciones de subir, bajar, borrar y compartir un fichero, ademas tambien lista a los clientes y muestra sus ficheros, los que le han compartido tambien los muestra. Una vez autenticado cuelga su propio servicio DiscoCliente con su id sesion. Creo que usar este id sesion en vez del unico aumenta la seguridad ya que no siempre seran los id iguales.

Nota: Cada vez que se pide un id de fichero o nombre de cliente por teclado, se muestra, la lista de ficheros posibles y la lista de clientes. El id en el listado va en la forma X.- nombfichero, siendo X el id que se está solicitando.

Iniciar() es quien busca el servicio de Autenticacion del servidor, y muestra el menu de registrar y autenticar al cliente.

Registrar() pide el nombre de un cliente, y usa el objeto remoto para crear al usuario devolviendo, -1 si no hay repos disponibles online, 0 si el cliente ya esta registrado y el id sesion que sera unico ya que estamos registran al usaurio en otro caso.

Autenticar() pide el nombre de un cliente y manda su nombre a traves del servicio de autenticación, el cual contesta con el id de sesion menor que cero si ha habido algun problema, y ejecuta el metodo levantarServicio().

LevantarServicio() levanta el servicio DiscoCliente acabado con el id de sesion actual, y muestra un menu, cuando el cliente desea abandonar se debe eliminar el servicio discocliente, pero hay que revisar si hay mas DiscoCliente colgados de otro clientes para no provocar excepciones al borrar el Registry, por tanto el ultimo cliente que quede abierto lo cierra.

ListarCientes() busca el servicio Gestor y le pide la lista de los clientes, lo recibe como String para imprimir directamente en la consola.

ListarFicheros() busca de nuevo el servicio Gestor para pedir la lista de ficheros (compartidos incluidos) de ese cliente a partir del id sesion del cliente, almacenado en la variable misesion.

CompartirFichero() pide los datos por teclado del id del fichero y el nombre de usuario con quien quiere compartirlo. Busca el servicio Gestor y lo usa para hacer la peticion de comparticion, enviando los datos anteriores y el id de sesion actual. El gestor contesta devolviendo un booleano si todo es correcto, es decir, si el fichero, por ejemplo, era nuestro para poder compartirlo.

BorrarFichero() como el anterior pide por teclado el id del fichero. Busca el servicio Gestor y lo usa para pedir el borrado del fichero pasando el id de sesion del cliente para que compruebe el fichero a borrar es suyo. Recordemos que no podemos borrar fichero que nos han sido compartido, ahora bien si nosotros somos los dueños de un fichero que ha sido compartido con otros, se borrarán todas las entradas en el servidor y como no, el fichero fisicamente del repositorio, para lo cual buscamos el servicio Cl-Operador del repositorio con la URL que nos proporciona el lector. Cuando

el borrado se realiza con éxito nos devuelve true la llamada del método del servicio.

BajarFichero() de nuevo pedimos por teclado el id del fichero, buscamos el servicio Gestor y le pasamos la URL del DiscoCliente, id del fichero y el id sesión al método bajarFichero() del Gestor, este debe informar al servicio Cl-Operador y este iniciar la carga con el DiscoCliente.

SubirFichero() pide por teclado el nombre del fichero, comprueba si existe y busca el servicio Gestor para pedir la URL del servicio Cl-Operador y cargar por este camino el fichero, eliminando de la ecuación al Servidor, en el proceso descarga. El problema que tenemos es como conseguir el id único de cliente. Ahora bien he usado una forma muy fea y poco ortodoxa de componer la llamada al servicio SrOperador, soy consciente de ello y es la forma realmente más complicada y menos viable de hacerlo, pero como ya esta hecha vamos a explicarla y ver los fallos que se han cometido y las posibles soluciones que hubiésemos tenido mucho más elegantes por supuesto.

Decir que la idea parte de los siguientes supuestos, en ningún momento el cliente conoce su id único que sabemos que es el nombre de la carpeta en el repositorio. Siempre hemos trabajado con el idsesión para mantener la seguridad de posibles intrusos o no cerrar las sesiones debidamente. Esto nos obliga a que no podemos componer el nombre de la carpeta en la llamada. Para solventar este supuesto podríamos haber preguntado por ejemplo al servicio autenticador cual es nuestro id único, pero... en ningún momento en el enunciado de la práctica se nos permite hacer esto, el servicio Autenticador, solo registra, autentica y desconecta. Podríamos haber colgado método para pedir el id, hubiese sido fantástico, pero... se hubiese vulnerado la seguridad???

Bien por otro lado podríamos haber pasado la sesión y que el repositorio consultase al servicio Gestor cual es la carpeta que le corresponde a ese id sesión, pero claro volvemos a hacer intervenir al Servidor, pero en este caso no estaríamos vulnerando la seguridad, ya que un intruso podría estar recavando información de la comunicación entre el servidor y el cliente, pero no entre el repositorio y el servidor, suponiendo que el repositorio estuviese en otra máquina distinta, no se puede estar en dos sitios al mismo tiempo.

Otro supuesto es dejar usar al cliente el servicio de Datos y conseguir el id único de cliente, algo que no veo coherente dada la arquitectura y saltaríamos por tanto estas restricciones en ella.

Por otro lado podríamos haber obrado de otra forma, quizás la más interesante, a la hora de construirlos repositorios, la idea es la misma que la forma de funcionar los DiscoCliente, sabemos que cada cliente tiene un id sesión, de igual forma que cada repositorio tiene un id sesión. Sabemos que el cliente cuelga su propio DiscoCliente con su id sesión, igual podríamos haber hecho con cada repositorio cuando se autentica, colgar los servicios SrOperador y ClOperador con "/" y su idsesión. Yendo un poco más alla, podríamos haber colgado las cadenas de las sesiones correspondiente a cada cliente de la forma:

cloperador/idsesióncliente1
cloperador/idsesióncliente2
...
clooperador/idsesiónclienteN

Lo cual requeriría que el Autenticador informase al repositorio de este id sesión y a que id único de cliente (carpeta) corresponde. Pero claro no tengo la suficiente experiencia como saber hasta que punto esto sería coherente. O bien quizás solamente colgar los nombres de las carpetas.

Volviendo al tema que no lleva es que la solución utilizada a sido: extraer de la URL que nos devuelve el Gestor, el id único del propietario del cliente que corresponde a la carpeta donde se guardará en la repo, suponiendo que todos los repositorios guardan en la misma carpeta.

```
String URLcompleta = servicioGestor.subirFichero(nombreFichero,miSesion);  
//extraemos la carpeta donde se va a guardar el fichero, que sera el id, del cliente  
//que sera al mismo tiempo el propietario del Fichero, segundo parametro  
  
int indice = URLcompleta.lastIndexOf('/');  
String URLservicioClOperador = URLcompleta.substring(0,indice);  
String propietario = URLcompleta.substring(indice + 1);  
//HAY QUE COMPROBAR SI EXISTE EL FICHERO en la carpeta actual  
Fichero fichero= new Fichero(nombreFichero,propietario);  
ServicioClOperadorInterface servicioClOperador =(ServicioClOperadorInterface)Naming.lookup(URLservicioClOperador);
```

ServicioDiscoClienteImpl.java

Implementa la interface ServicioDiscoClienteInterface la cual publica el método `bajarFichero()` que se encarga de recibir el fichero solicitado al repositorio que envia a través del servicio Sr-Operador.

6.- Ejemplos de prueba

1º prueba

A continuación se van a mostrar una serie de capturas de pantalla de la puesta en marcha del sistema. Se lanza el servidor, seguidamente del repositorio y se lanzan 2 consolas de cliente. Se realizan algunas operaciones de subida, bajada, borrado y compartir ficheros. Al finalizar se sale del repositorio, seguidamente los clientes y por último el servidor.

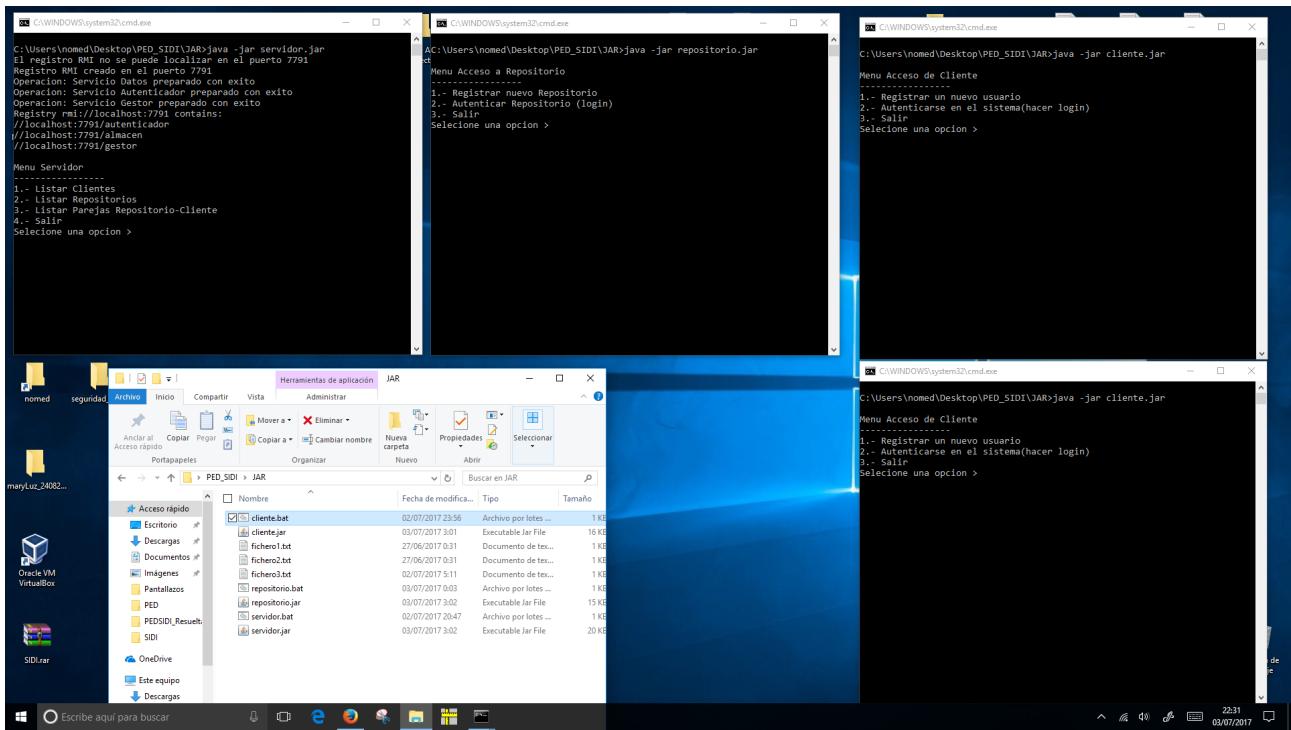


Imagen 19.- lanzamos los el servidor, el repositorio y 2 clientes

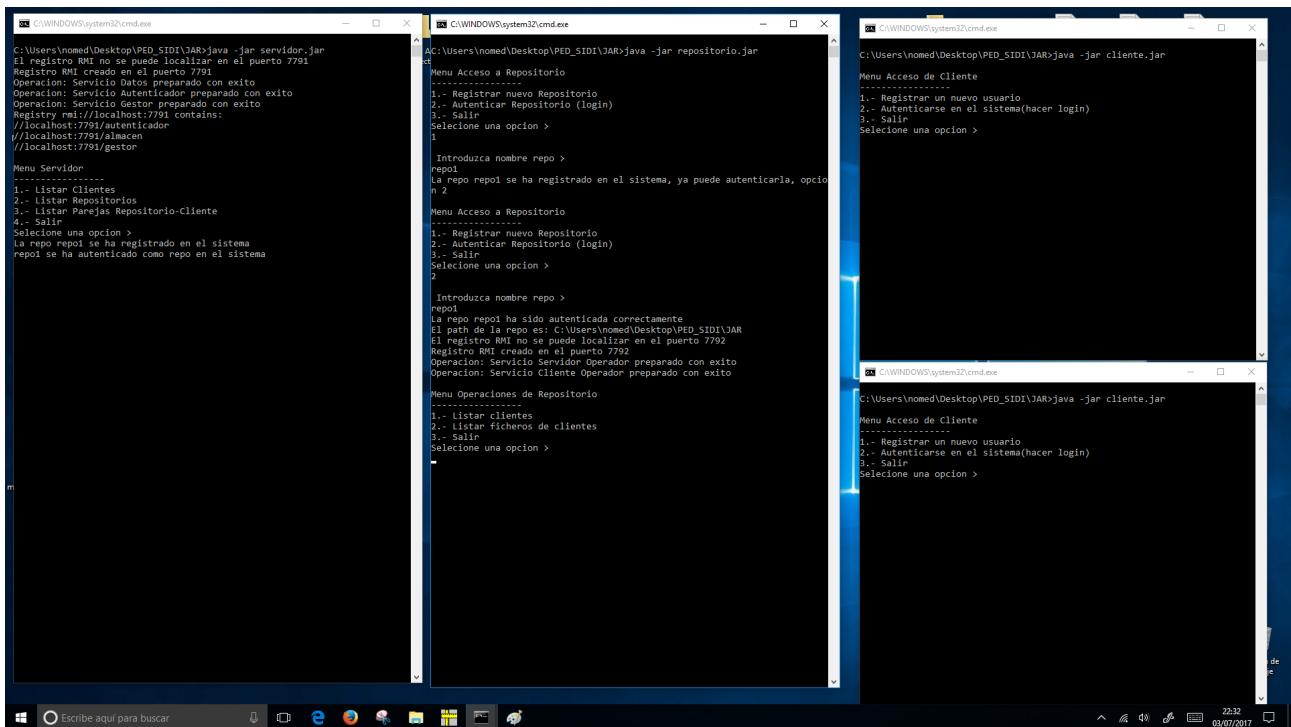


Imagen 20.- Registro y autenticación del repositorio

```

C:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar servidor.jar
El registro RMI no se puede localizar en el puerto 7791
Registro RMI creado en el puerto 7791
Operacion: Servicio Datos preparado con exito
Operacion: Servicio Autenticador preparado con exito
Operacion: Servicio Gestor preparado con exito
Registro rm://localhost:7791 contains:
//localhost:7791/autenticador
//localhost:7791/almacen
//localhost:7791/gestor

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
la repo rep01 se ha registrado en el sistema
repo1 se ha autenticado como repo en el sistema
El cliente user1 se ha registrado en el sistema
El cliente user2 se ha registrado en el sistema
El cliente user2 se ha registrado en el sistema

AC:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar repositorio.jar
ct
Menu Acceso a Repositorio
-----
1.- Registrar nuevo Repositorio
2.- Autenticar Repositorio (login)
3.- Salir
Seleccione una opcion >
1
Introduzca nombre repo >
repo1
La repo rep01 se ha registrado en el sistema, ya puede autenticarla, opcion n >
2

Menu Acceso a Repositorio
-----
1.- Registrar nuevo Repositorio
2.- Autenticar Repositorio (login)
3.- Salir
Seleccione una opcion >
2
Introduzca nombre repo >
repo1
La repo rep01 ha sido autenticada correctamente
El path de la repo es: C:\Users\nomed\Desktop\PED_SIDI\JAR
El registro RMI no se puede localizar en el puerto 7792
Registro RMI creado en el puerto 7792
Operacion: Servicio Servidor Operador preparado con exito
Operacion: Servicio Cliente Operador preparado con exito

Menu Operaciones de Repositorio
-----
1.- Listar clientes
2.- Listar ficheros de clientes
3.- Salir
Seleccione una opcion >
Se ha creado la carpeta: 1348429947 en el path: C:\Users\nomed\Desktop\PED_SIDI\JAR
Se ha creado la carpeta: 1348429948 en el path: C:\Users\nomed\Desktop\PED_SIDI\JAR

C:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar cliente.jar
Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >
1
Introduzca nombre cliente >
user1
user1 se ha registrado en el sistema, ahora puede autenticarse con opcion 2

Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >
2
Introduzca nombre cliente >
user2
user2 se ha registrado en el sistema, ahora puede autenticarse con opcion 2

Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >

```

Imagen 21.- registro de dos clientes

```

C:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar servidor.jar
El registro RMI no se puede localizar en el puerto 7791
Registro RMI creado en el puerto 7791
Operacion: Servicio Datos preparado con exito
Operacion: Servicio Autenticador preparado con exito
Operacion: Servicio Gestor preparado con exito
Registro rm://localhost:7791 contains:
//localhost:7791/autenticador
//localhost:7791/almacen
//localhost:7791/gestor

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
la repo rep01 se ha registrado en el sistema
repo1 se ha autenticado como repo en el sistema
El cliente user1 se ha registrado en el sistema
El cliente user2 se ha registrado en el sistema
El cliente user1 se ha autenticado como cliente en el sistema
1
Cliente [[OFFLINE] id=1348429948, nombre=user2] Cliente [[ ONLINE] id=1348429947, nombre=user1]

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >

AC:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar repositorio.jar
ct
Menu Acceso a Repositorio
-----
1.- Registrar nuevo Repositorio
2.- Autenticar Repositorio (login)
3.- Salir
Seleccione una opcion >
1
Introduzca nombre repo >
repo1
La repo rep01 se ha registrado en el sistema, ya puede autenticarla, opcion n >
2

Menu Acceso a Repositorio
-----
1.- Registrar nuevo Repositorio
2.- Autenticar Repositorio (login)
3.- Salir
Seleccione una opcion >
2
Introduzca nombre repo >
repo1
La repo rep01 ha sido autenticada correctamente
El path de la repo es: C:\Users\nomed\Desktop\PED_SIDI\JAR
El registro RMI no se puede localizar en el puerto 7792
Registro RMI creado en el puerto 7792
Operacion: Servicio Servidor Operador preparado con exito
Operacion: Servicio Cliente Operador preparado con exito

Menu Operaciones de Repositorio
-----
1.- Listar clientes
2.- Listar ficheros de clientes
3.- Salir
Seleccione una opcion >
Se ha creado la carpeta: 1348429947 en el path: C:\Users\nomed\Desktop\PED_SIDI\JAR
Se ha creado la carpeta: 1348429948 en el path: C:\Users\nomed\Desktop\PED_SIDI\JAR

C:\Users\nomed\Desktop\PED_SIDI\JAR>java -jar cliente.jar
Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >
1
Introduzca nombre cliente >
user1
user1 se ha registrado en el sistema, ahora puede autenticarse con opcion 2

Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >
2
Introduzca nombre cliente >
user2
user2 se ha registrado en el sistema, ahora puede autenticarse con opcion 2

Menu Acceso de Cliente
-----
1.- Registrarse un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >

```

Imagen 22.- listado de clientes en el servidor, el cliente user1 esta autenticado (online)

The screenshot shows two separate command-line windows on a Windows desktop. Both windows are titled 'Seleccionar C:\WINDOWS\system32\cmd.exe'.
 The left window shows the output of running 'java -jar servidor.jar' and 'java -jar repositorio.jar'. It displays the creation of an RMI registry, successful service preparation, and the creation of two client accounts ('user1' and 'user2'). It also shows the registration of a repository named 'repo1' and its status as 'online'.
 The right window shows the output of running 'java -jar repositorio.jar'. It displays the creation of an RMI registry, successful service preparation, and the creation of a new folder ('user2') at the path 'C:\Users\nomed\Desktop\PED_SIDI\JAR'.
 The taskbar at the bottom shows several icons, including a search bar and the date/time '03/07/2017 22:37'.

Imagen 23.- los dos clientes están autenticados, y la repo aparece online con las 2 carpetas

The screenshot shows three Windows command-line windows on a Windows desktop. All three windows are titled 'Seleccionar C:\WINDOWS\system32\cmd.exe'.
 The leftmost window shows the same initial setup as in Image 23, including the creation of the 'repo1' repository and its status as 'online'.
 The middle window shows the output of running 'java -jar operaciones.jar'. It displays the creation of an RMI registry, successful service preparation, and the creation of a file 'ficher01.txt' in the current directory. The file contents are shown as 'Fichero: ficher01.txt enviado'.
 The rightmost window shows the output of running 'java -jar operaciones.jar'. It displays the creation of an RMI registry, successful service preparation, and the creation of a file 'ficher01.txt' in the current directory. The file contents are shown as 'Fichero 1348429947\ficher01.txt recibido y guardado'.
 The taskbar at the bottom shows several icons, including a search bar and the date/time '03/07/2017 22:38'.

Imagen 24.- el cliente user1 sube el fichero1.txt

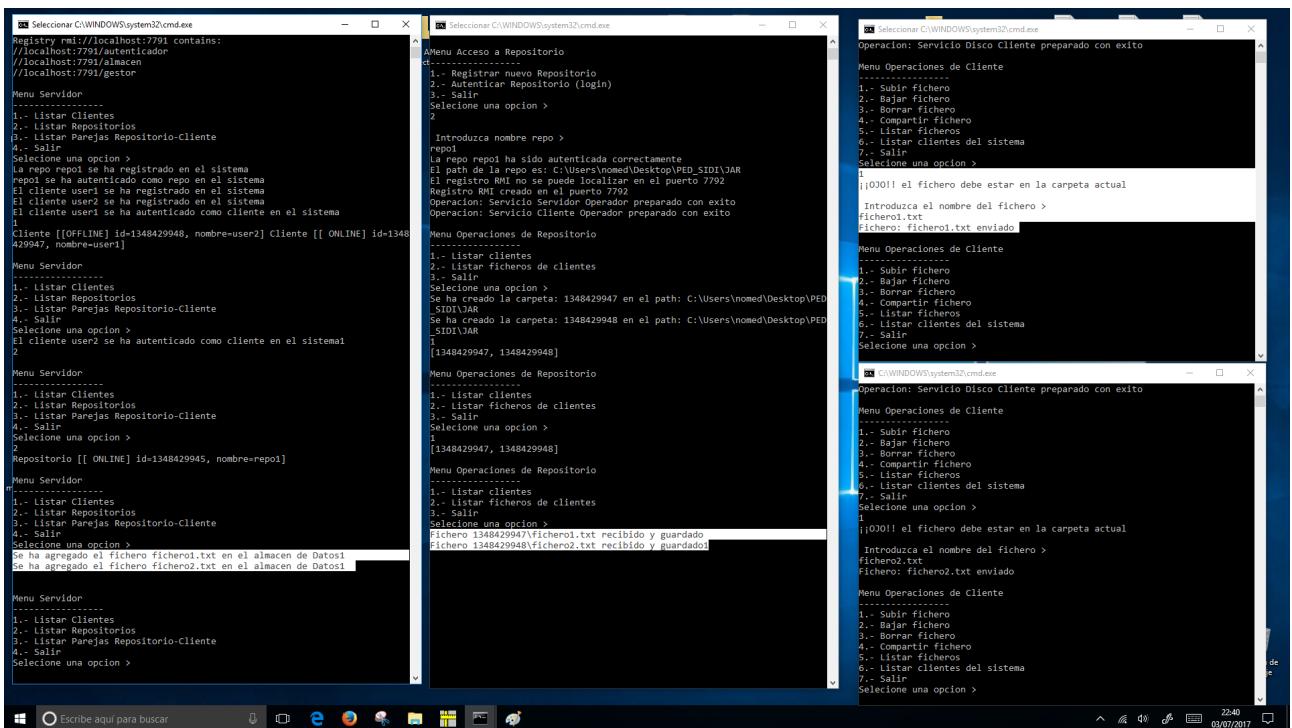


Imagen 25.- el cliente user2 ha subido el fichero2.txt

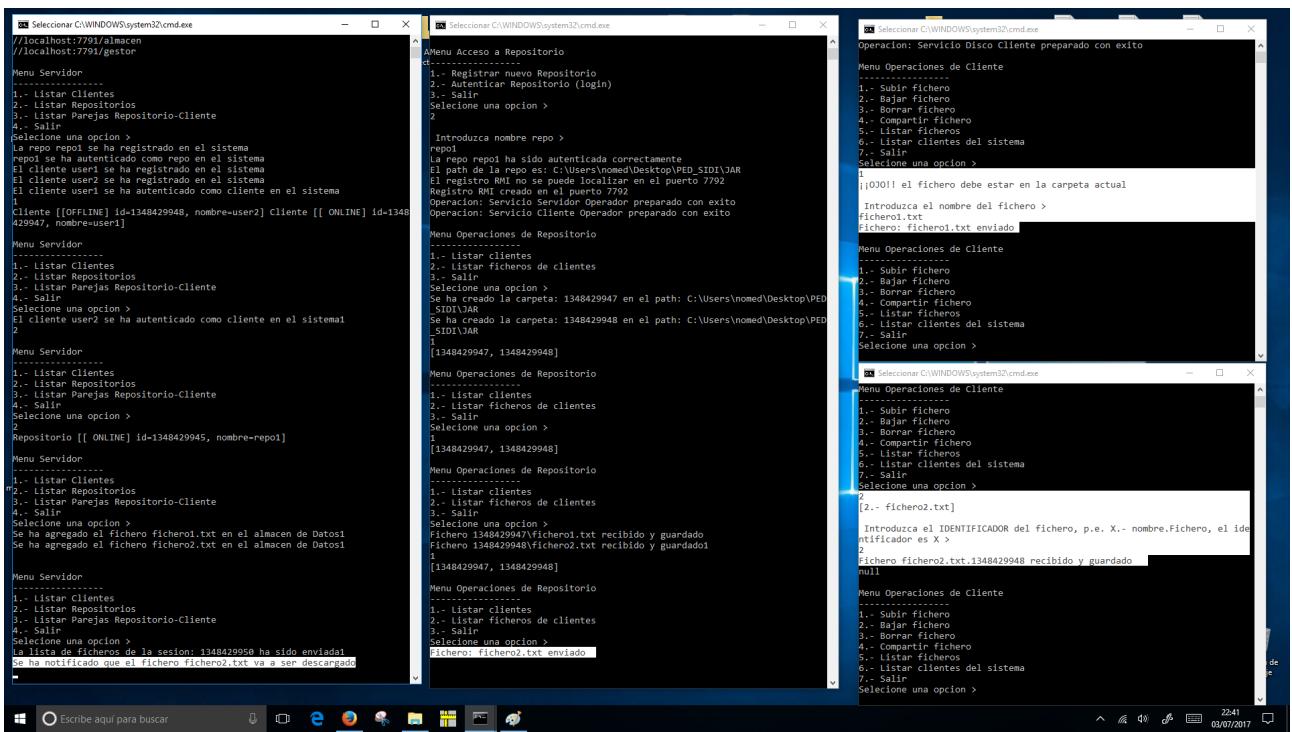


Imagen 26.- el cliente user2 baja el fichero2.txt

```

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
El cliente user2 se ha autenticado como cliente en el sistema
2

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
2
Repositorio [[ ONLINE] id=1348429945, nombre=repo1]
Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
Se ha agregado el fichero fichero1.txt en el almacen de Datos1
Se ha agregado el fichero fichero2.txt en el almacen de Datos1

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
La lista de ficheros de la sesion: 1348429950 ha sido enviada
Se ha notificado que el fichero fichero2.txt va a ser descargado
1

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
1
Cliente [[ ONLINE] id=1348429948, nombre=user2] Cliente [[ ONLINE] id=1348429947, nombre=user1]
Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
La lista de ficheros de la sesion: 1348429949 ha sido enviada
El Fichero fichero1.txt se ha compartido correctamente
La lista de ficheros de la sesion: 1348429950 ha sido enviada
[2] Seleccionar C:\WINDOWS\system32\cmd.exe
A

```

Imagen 27.- el cliente user1 comparte el fichero1.txt con el user2.

```

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
Seleccione una opcion >
El cliente user2 se ha autenticado como cliente en el sistema
2

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
2
Repositorio [[ ONLINE] id=1348429945, nombre=repo1]
Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
1
Cliente [[ ONLINE] id=1348429948, nombre=user2] Cliente [[ ONLINE] id=1348429947, nombre=user1]
Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
La lista de ficheros de la sesion: 1348429950 ha sido enviada
Se ha notificado que el fichero fichero2.txt va a ser descargado
1

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
1
Cliente [[ ONLINE] id=1348429948, nombre=user2] Cliente [[ ONLINE] id=1348429947, nombre=user1]
Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
La lista de ficheros de la sesion: 1348429949 ha sido enviada
El Fichero fichero1.txt se ha compartido correctamente
La lista de ficheros de la sesion: 1348429950 ha sido enviada
La lista de ficheros de la sesion: 1348429958 ha sido enviada
Se ha notificado que el fichero fichero1.txt va a ser descargado
[2] Seleccionar C:\WINDOWS\system32\cmd.exe
A

```

Imagen 28.- el cliente user2 descarga el fichero compartido fichero1.txt

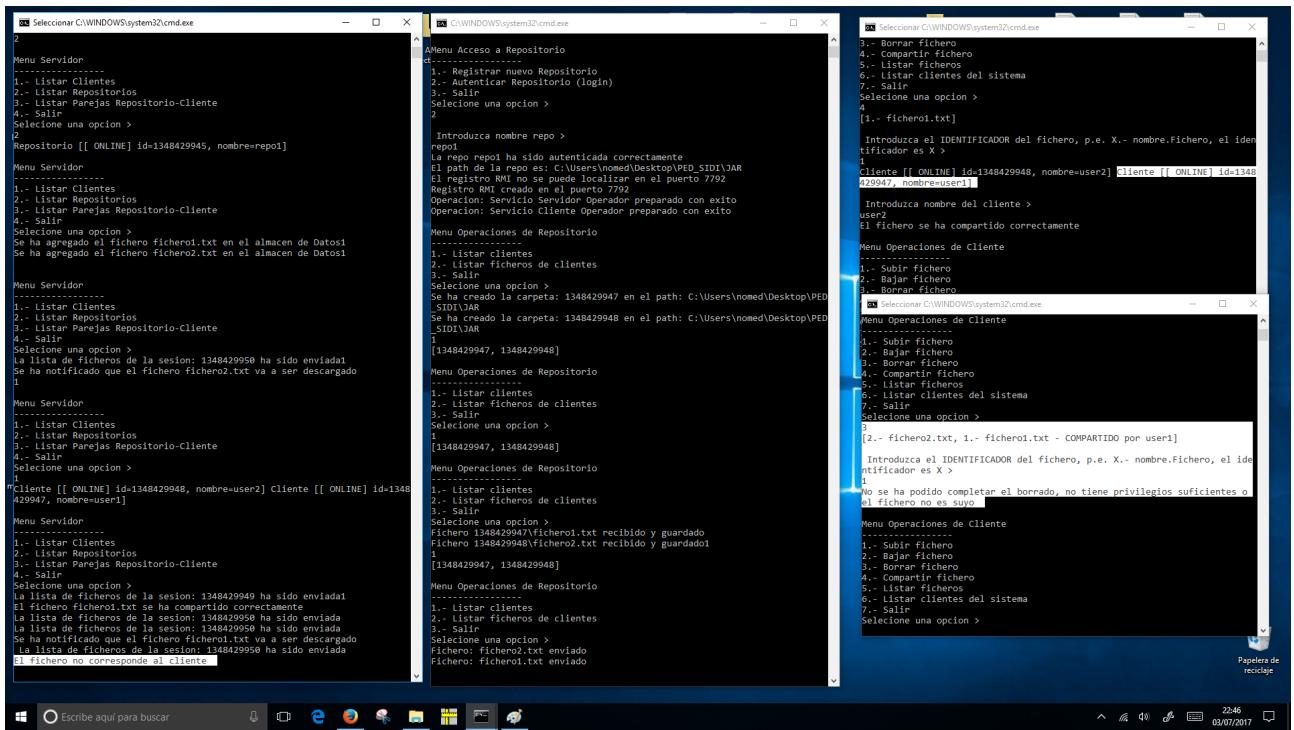


Imagen 29.- el cliente user2 intenta borrar el fichero compartido, no puede borrarlo no es suyo es de user1

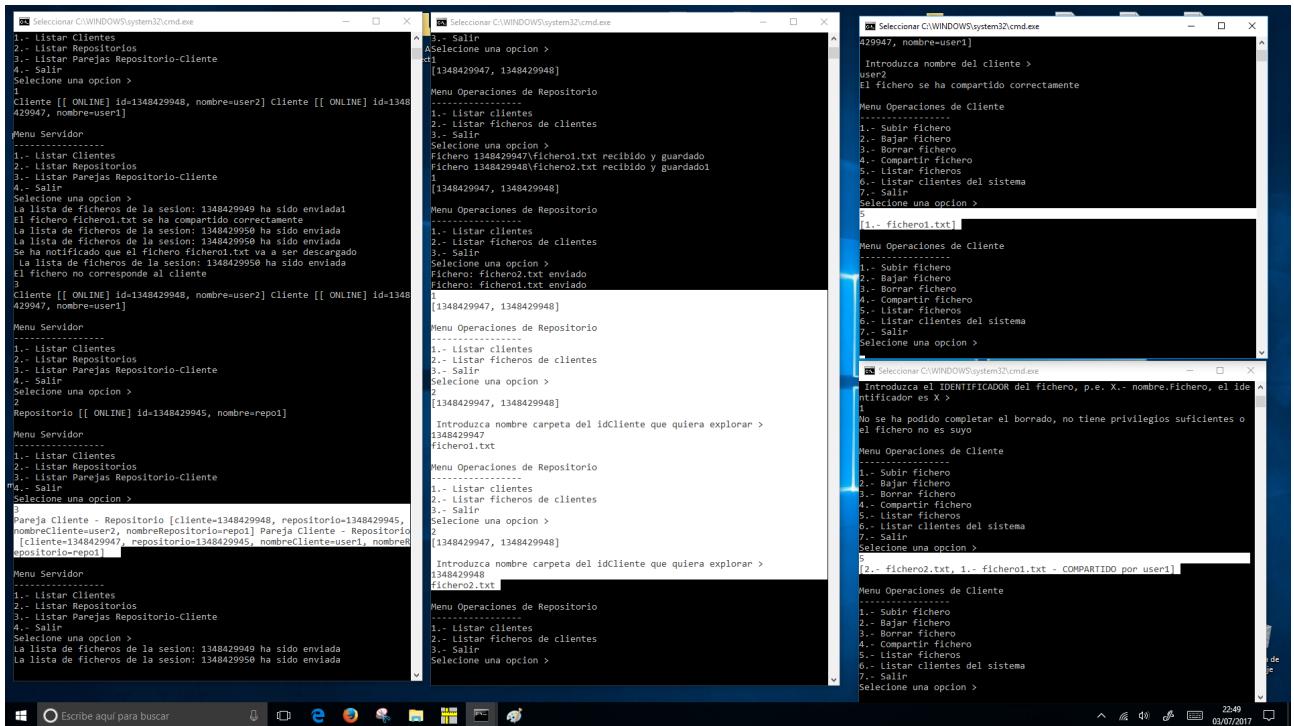


Imagen 30.- listados de ficheros y parejas cliente - repositorio

```

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
[2] Seleccionar C:\WINDOWS\system32\cmd.exe
[3] Seleccionar C:\WINDOWS\system32\cmd.exe

```

Client [[ONLINE] id=1348429948, nombre=user2] Cliente [[ONLINE] id=1348429947, nombre=user1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >

La lista de ficheros de la sesion: 1348429949 ha sido enviada
El fichero fichero1.txt se ha compartido correctamente
La lista de ficheros de la sesion: 1348429950 ha sido enviada
La lista de ficheros de la sesion: 1348429959 ha sido enviada
Se ha notificado que el fichero fichero1.txt va a ser descargado
La lista de ficheros de la sesion: 1348429950 ha sido enviada
El fichero no corresponde al cliente

Client [[ONLINE] id=1348429948, nombre=user2] Cliente [[ONLINE] id=1348429947, nombre=user1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 2

Repositorio [[ONLINE] id=1348429945, nombre=repo1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 3

Pareja Cliente - Repositorio [cliente=1348429948, repositorio=1348429945, nombreCliente=user2, nombreRepositorio=repo1] Pareja Cliente - Repositorio [cliente=1348429947, repositorio=1348429945, nombreCliente=user1, nombreRepositorio=repo1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 3

La lista de ficheros de la sesion: 1348429949 ha sido enviada
La lista de ficheros de la sesion: 1348429950 ha sido enviada
La lista de ficheros de la sesion: 1348429948 ha sido enviada

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
Fichero1.txt

Menu Operaciones de Repositorio
1.- Listar clientes
2.- Listar ficheros de clientes
3.- Salir
Seleccione una opcion >

Fichero 1348429947\fichero1.txt recibido y guardado
Fichero 1348429948\fichero2.txt recibido y guardado

[1] Seleccionar C:\WINDOWS\system32\cmd.exe

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[1.- fichero1.txt]

Introduzca el IDENTIFICADOR del fichero, p.e. X.- nombre.Fichero, el identificador es X > 1

Fichero: fichero1.txt borrado

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[2.- fichero2.txt, 1.- fichero1.txt - COMPARTIDO por user1]

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[2.- fichero2.txt]

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[2.- fichero2.txt]

Imagen 31.- el cliente user1 borra el fichero1.txt que esta compartido pero como es suyo todo ok

```

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
[2] Seleccionar C:\WINDOWS\system32\cmd.exe
[3] Seleccionar C:\WINDOWS\system32\cmd.exe

```

Client [[ONLINE] id=1348429948, nombre=user2] Cliente [[ONLINE] id=1348429947, nombre=user1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 2

Repositorio [[ONLINE] id=1348429945, nombre=repo1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 3

Pareja Cliente - Repositorio [cliente=1348429948, repositorio=1348429945, nombreCliente=user2, nombreRepositorio=repo1] Pareja Cliente - Repositorio [cliente=1348429947, repositorio=1348429945, nombreCliente=user1, nombreRepositorio=repo1]

Menu Servidor
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion > 3

La lista de ficheros de la sesion: 1348429949 ha sido enviada
La lista de ficheros de la sesion: 1348429950 ha sido enviada
La lista de ficheros de la sesion: 1348429948 ha sido enviada

[1] Seleccionar C:\WINDOWS\system32\cmd.exe
Fichero1.txt

Menu Operaciones de Repositorio
1.- Listar clientes
2.- Listar ficheros de clientes
3.- Salir
Seleccione una opcion >

Fichero 1348429947\fichero1.txt recibido y guardado
Fichero 1348429948\fichero2.txt recibido y guardado

[1] Seleccionar C:\WINDOWS\system32\cmd.exe

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[6 Cliente [[OFFLINE] id=1348429948, nombre=user2] Cliente [[ONLINE] id=1348429947, nombre=user1]

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[2.- fichero2.txt]

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >

[2.- fichero2.txt]

Imagen 32.- el cliente user2 se desconecta del sistema, el servidor lista los clientes como vemos el user2 esta offline mientras user1 sigue online.

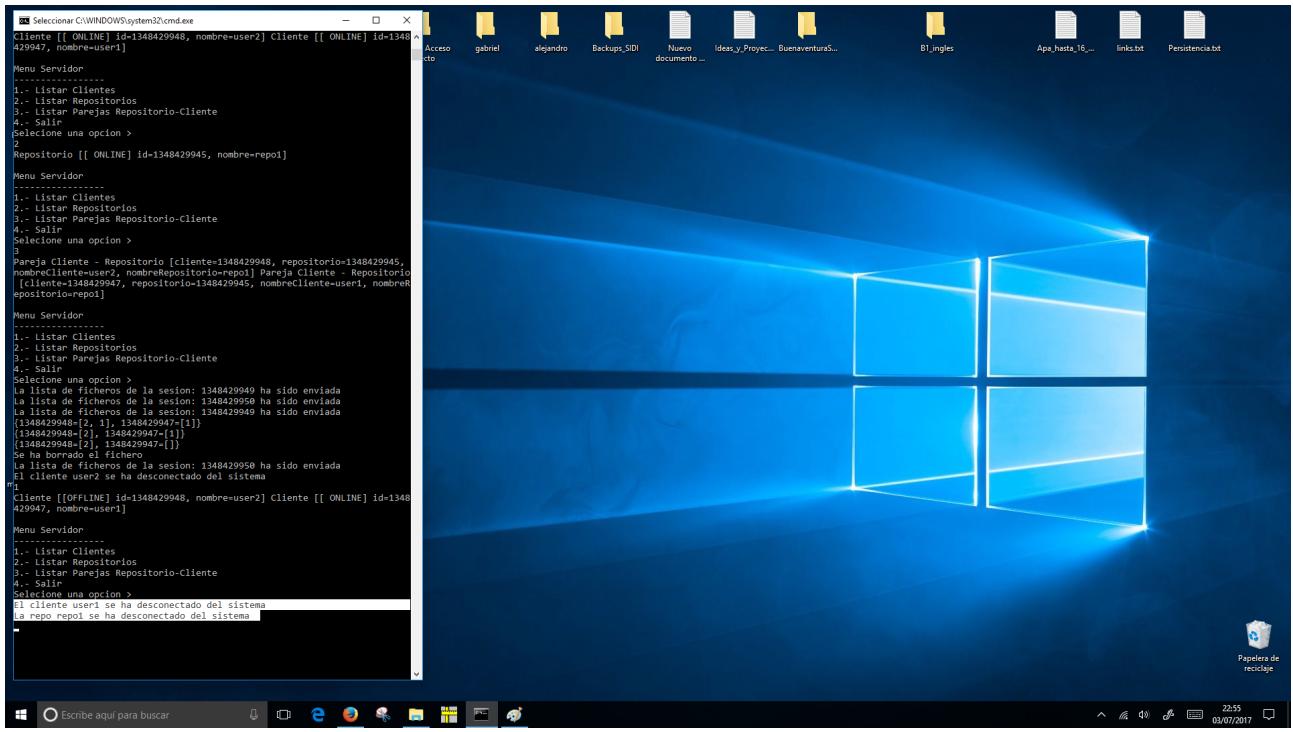


Imagen 33.- el cliente useer1 y la repo 1 se desconectan

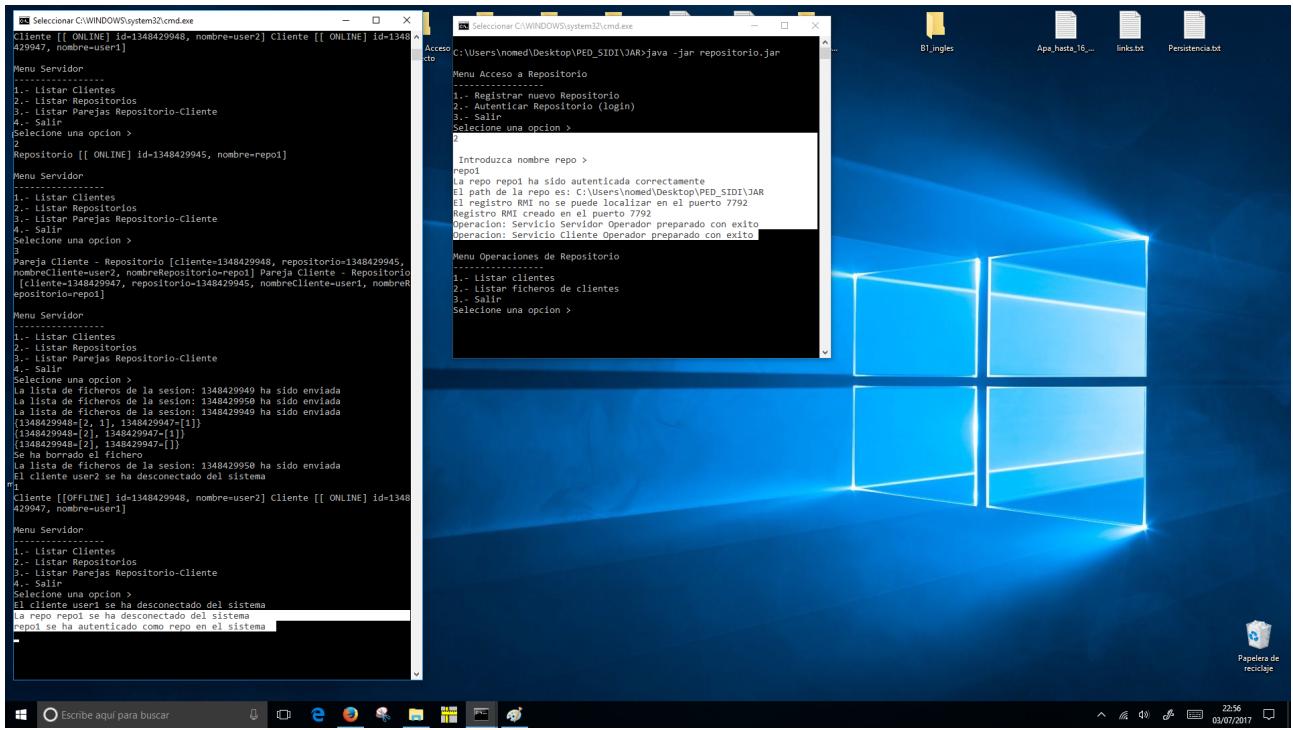


Imagen 34.- la repo1 vuelve a autenticarse

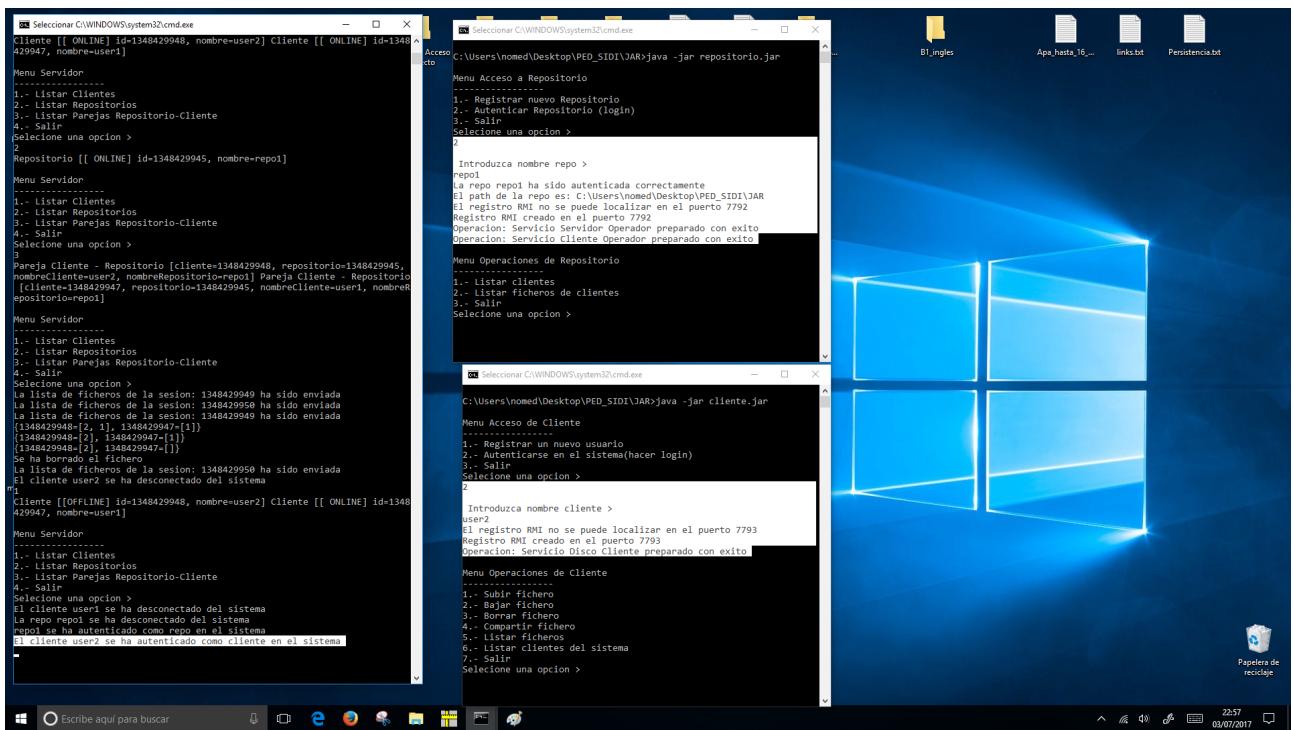


Imagen 35.- el cliente user2 vuelve a autenticarse

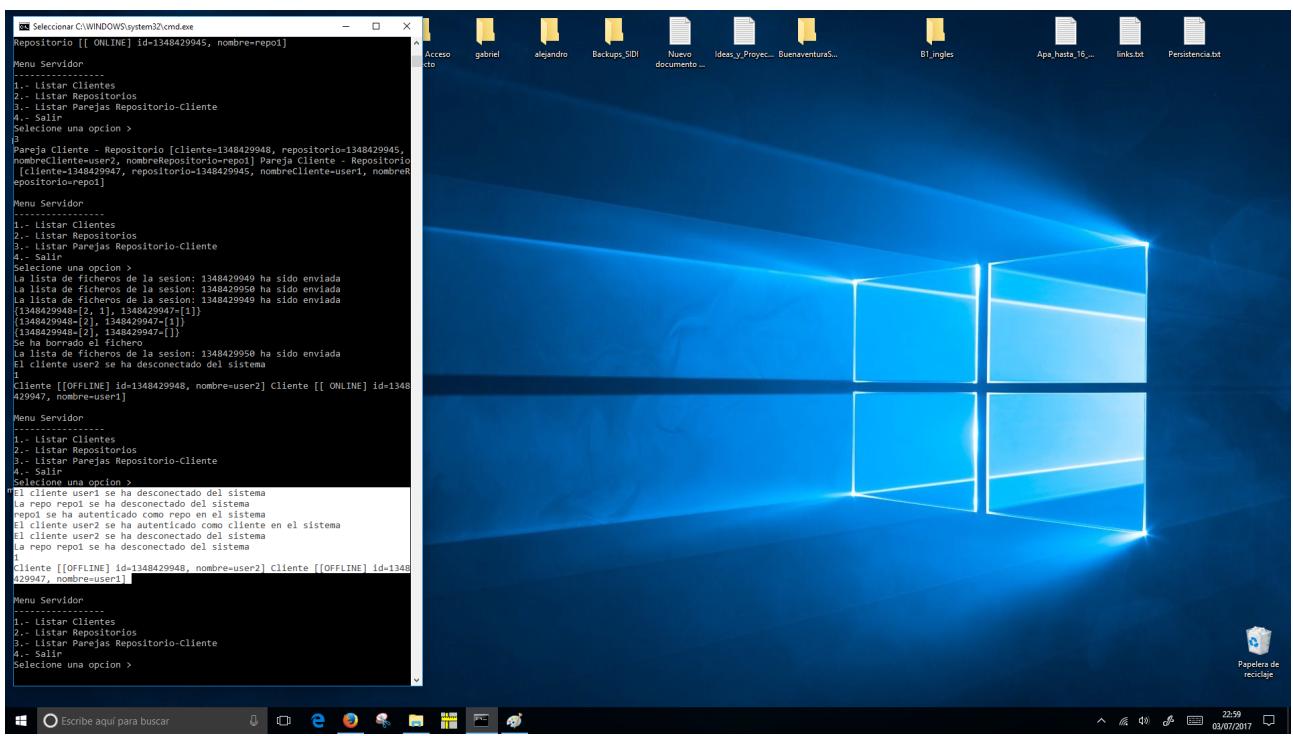


Imagen 36.- los clientes y la repo se desconectan y salen del programa.

2º prueba

En esta segunda, vamos a poner de manifiesto varias situaciones un tanto más complicadas. Lo que se quiere mostrar es el correcto funcionamiento de:

- Sin el servidor corriendo los clientes y las repos no trabajan.
- La persistencia de datos.
- El correcto funcionamiento de 2 repos online, cada una con un cliente.
- El correcto movimiento de ficheros entre clientes.

Primero vamos a preparar el entorno para este segundo experimento y lo que se va a hacer:

1. Vamos a copiar el servidor en una carpeta distinta a los clientes y los repositorios.
2. Creamos dos carpetas repo1 y repo2, cada una deberá recibir los ficheros de sus clientes, confirmando el correcto funcionamiento de los 2 servicios que cuelga cada repo.
3. Creamos tres carpetas user1, user2 y user3, cada cliente tendrá 3 ficheros de textos, cada una de las carpetas deberá recibir los ficheros que descargan, sin mezclarse eso se confirmará con cada bajada del fichero añadiendo el id sesión de cada cliente.
4. Vamos a lanzar el servidor, dos repositorios y dos clientes.
5. En ese momento veremos que se preparan las carpetas de persistencia y los ficheros de logs.
6. Registraremos la repo1 y la autenticamos.
7. Registraremos el cliente user1 y será asignado a la repo1.
8. Desconectaremos la repo1, y registramos y autenticamos la repo2 con el fin de que user2 sea asignado a la repo2.
9. Registraremos el cliente user2 y user3 y será asignado a la repo2, autenticamos solo a user2, el user3 queda de offline, lo cual se vera en la lista de clientes del servidor.
10. Volveremos a autenticar la repo1 y el user1, de esta forma, tendremos dos clientes online uno en cada repo y user3 offline, las dos repos online.
11. Subiremos ficheros, compartiremos ficheros entre clientes. Mostraremos una captura de todo el registro de actividad del servidor y las carpetas que tiene cada repo.
12. Desconectaremos los clientes, las repos, y el servidor, y confirmaremos la persistencia arrancando el servidor y los programas de cliente y repo pero sin todavía autenticar.
13. Acto seguido volveremos a autenticar a los repositorio y clientes sin registrarlos, ya que la persistencia debe funcionar. Listaremos estos datos para confirmarlo.
14. Se realizaran bajadas de los ficheros para confirmar que cada fichero va a la carpeta de cliente que corresponde y daremos por finalizada la prueba.

La secuencia de imágenes etiquetadas es la siguiente.

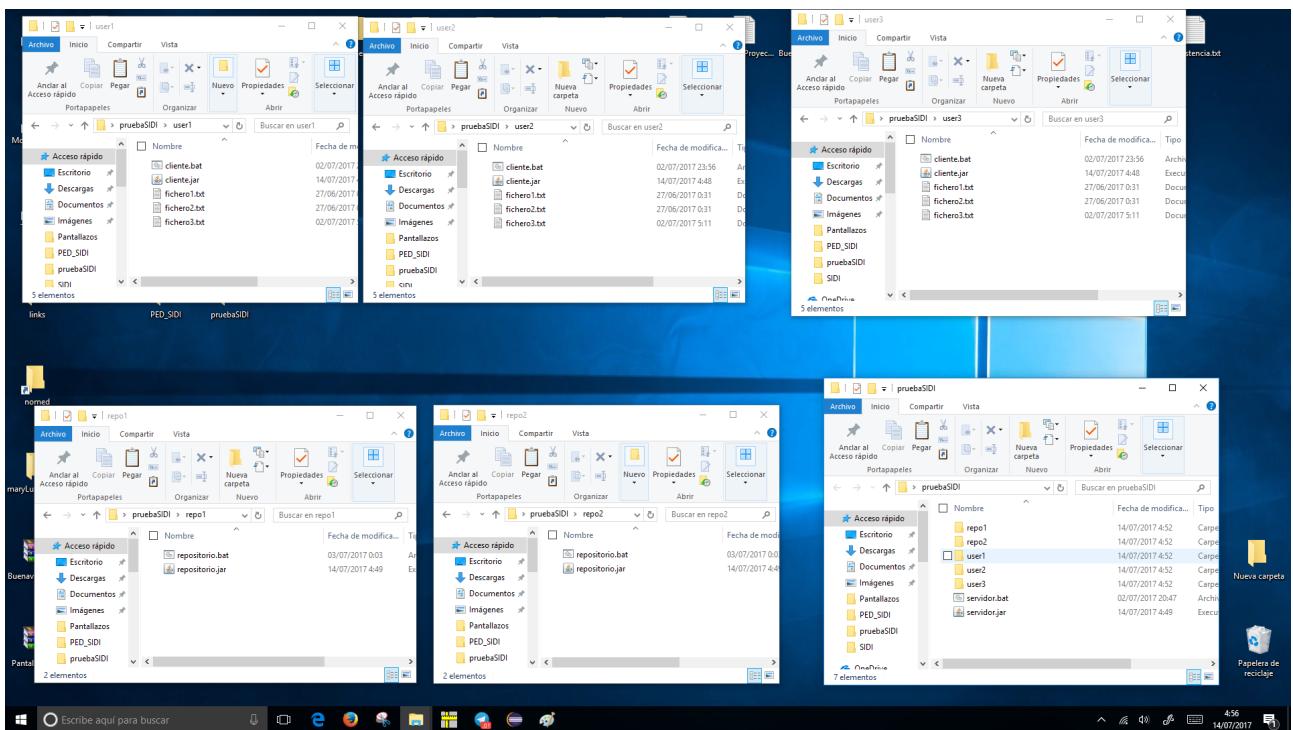


Imagen 37.- Creación de distintas carpetas donde se ejecutará cada programa.

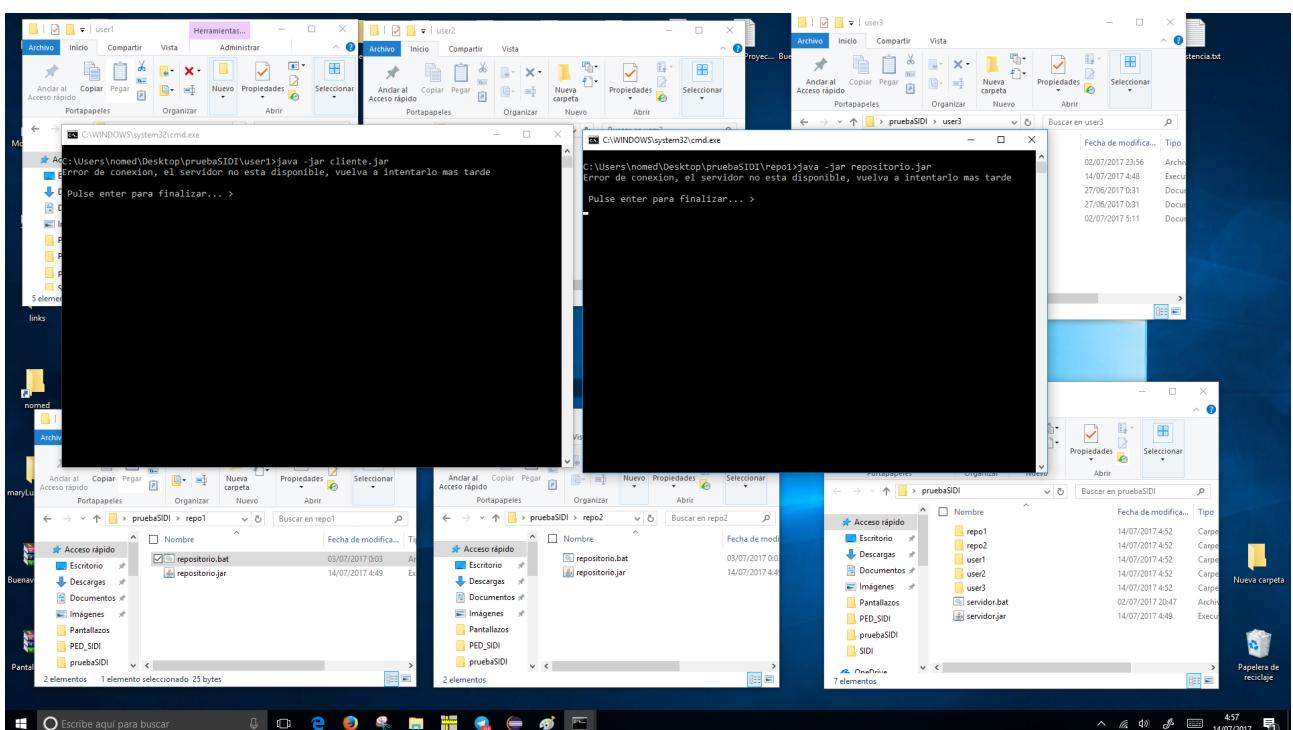


Imagen 38.- Ejecucion de cliente y repo sin que el servidor este ejecutandose, como no lo encuentran lanzan una excepción controlada ConnectionException con mensaje personalizado.

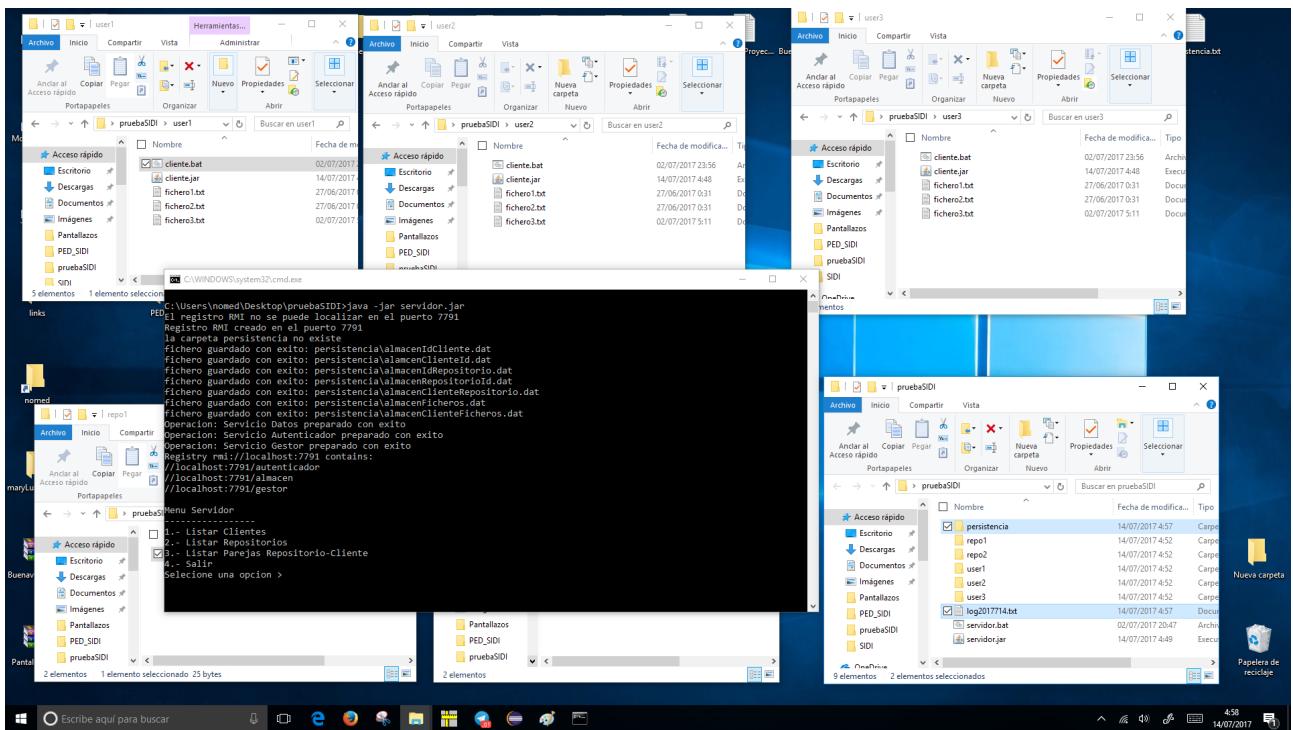


Imagen 39.- no encuentra carpeta persistencia al ser la primera vez que arranca, crea la carpeta con los ficheros persistentes con las tablas vacias y las guarda, genera el fichero de log diario.

```

Buenaventura.Salcedo.Santos.Ólmo-SIDI 2016-2017.edt - LibreOffice Writer
And [x] Seleccionar C:\WINDOWS\system32\cmd.exe

C:\Users\nomed\Desktop\pruebaSIDI>java -jar servidor.jar
El registro RMI no se puede localizar en el puerto 7791
El registro RMI creado en el puerto 7791
la carpeta persistencia no existe
Fichero guardado con exito: persistencia\almacen\cliente.dat
Fichero guardado con exito: persistencia\almacen\clienteId.dat
Fichero guardado con exito: persistencia\almacen\repository.dat
Fichero guardado con exito: persistencia\almacen\repositoryId.dat
Fichero guardado con exito: persistencia\almacen\ficheros.dat
Fichero guardado con exito: persistencia\almacen\ficherosId.dat
Fichero guardado con exito: persistencia\almacen\cliente\ficheros.dat
Operacion: Servicio Datos preparado con exito
Operacion: Servicio Autenticador preparado con exito
Operacion: Servicio Gestor preparado con exito
Registry rmi://localhost:7791 contains:
//localhost:7791/autenticador
//localhost:7791/almacen
//localhost:7791/gestor

Menu Servidor
-----
1.- Listar Clientes
2.- Listar Repositorios
3.- Listar Parejas Repositorio-Cliente
4.- Salir
Seleccione una opcion >
La repo rep0 se ha registrado en el sistema
repo se ha autenticado como reposo en el sistema
El cliente user1 se ha registrado en el sistema
La repo rep0 se ha desconectado del sistema
La repo rep0 se ha registrado en el sistema
repo se ha autenticado como reposo en el sistema
El cliente user2 se ha registrado en el sistema
El cliente user3 se ha registrado en el sistema
El cliente user2 se ha autenticado como cliente en el sistema

C:\Windows\system32\cmd.exe
Registro RMI creado en el puerto 7792
Operacion: Servicio Operador preparado con exito
Operacion: Servicio Cliente Operador preparado con exito
Registry rmi://localhost:7792/operador/2096813398
//localhost:7792/operador/2096813398
//localhost:7792/snapdragon/2096813398

Menu Operaciones de Repositorio
1.- Listar Clientes
2.- Listar Ficheros de clientes
3.- Salir
Se ha elegido una opcion >
Introduzca nombre cliente >
user2
user2 se ha registrado en el sistema, ahora puede autenticarse con opcion
op\pruebaSIDI\repo1

Fichero guardado con exito: persistencia\polilla\carpetas.dat
Operacion: Servicio Operador cerrandose...
Operacion: Servicio Operador cerrado con exito
Operacion: Servicio Cliente Operador cerrando...
Operacion: Servicio Cliente Operador cerrado con exito
Operacion: Registry cerrado con exito

Menu Acceso a Repositorio
1.- Registrar nuevo Repositorio
2.- Autenticar Repositorio (login)
3.- Salir
Seleccione una opcion >
1

Introduzca nombre cliente >
user1
user1 se ha registrado en el sistema, ahora puede autenticarse con opcion
op\pruebaSIDI\repo1

Fichero guardado con exito: persistencia\polilla\carpetas.dat
Operacion: Servicio Operador cerrandose...
Operacion: Servicio Operador cerrado con exito
Operacion: Servicio Cliente Operador cerrando...
Operacion: Servicio Cliente Operador cerrado con exito
Operacion: Registry cerrado con exito

Menu Acceso a Cliente
1.- Registrar un nuevo usuario
2.- Autenticarse en el sistema(hacer login)
3.- Salir
Seleccione una opcion >
1

Introduzca nombre cliente >
user1
user1 se ha registrado en el sistema, ahora puede autenticarse con opcion
op\pruebaSIDI\repo1

Fichero guardado con exito: persistencia\polilla\carpetas.dat
Operacion: Servicio Operador cerrandose...
Operacion: Servicio Operador cerrado con exito
Operacion: Servicio Cliente Operador cerrando...
Operacion: Servicio Cliente Operador cerrado con exito
Operacion: Registry cerrado con exito

Menu Operaciones de Cliente
1.- Subir fichero
2.- Bajar fichero
3.- Borrar fichero
4.- Compartir fichero
5.- Listar ficheros
6.- Listar clientes del sistema
7.- Salir
Seleccione una opcion >
1

Se ha creado la carpeta: 2096813392 en el path: C:\Users\nomed\Desktop\pruebaSIDI\repo1
Se ha creado la carpeta: 2096813393 en el path: C:\Users\nomed\Desktop\pruebaSIDI\repo1

Página 35 de 36 8.920 palabras, 55.210 caracteres
Escríbe aquí para buscar

```

Imagen 40.- se registra y autentica repo1, se registra user1, se desconecta repo1, se registra y autentica repo2, y se registran user2 y user3, se autentica user2.

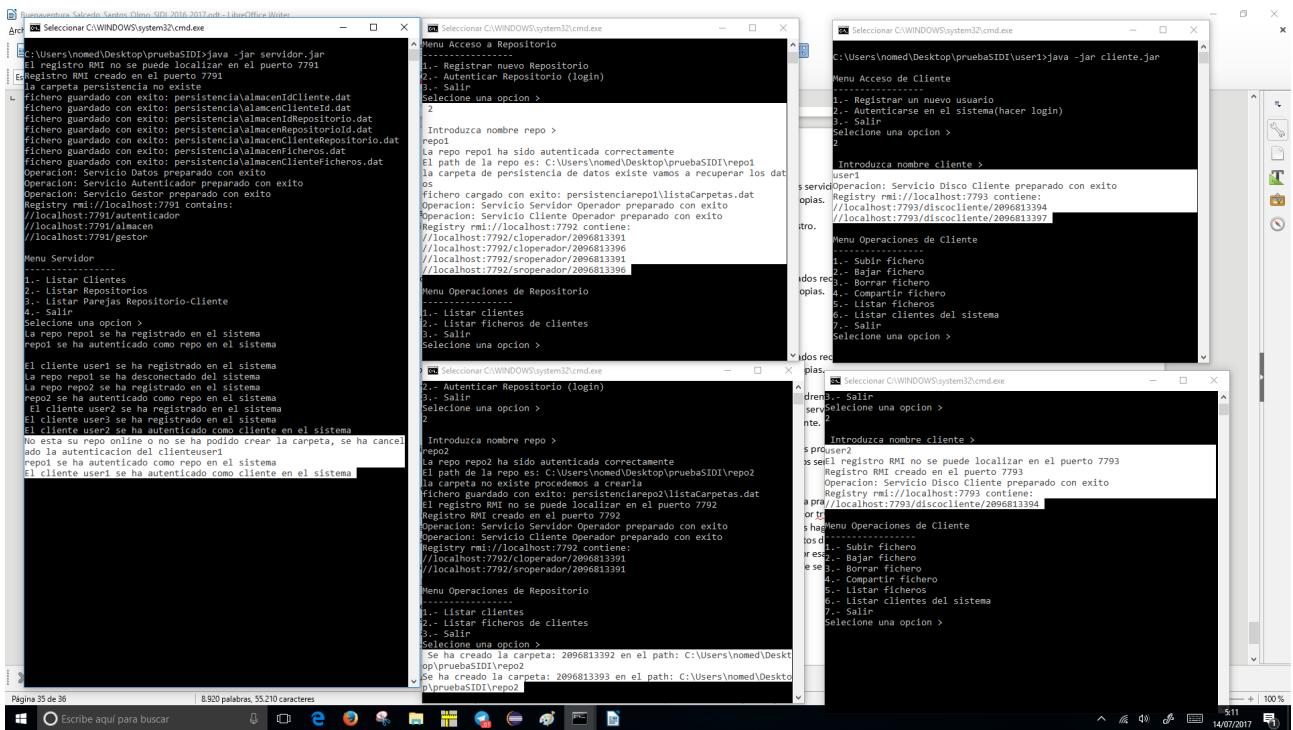


Imagen 41.- todo online, repo1, repo2, user1 y user2. Podemos ver los servicios que hay colgados

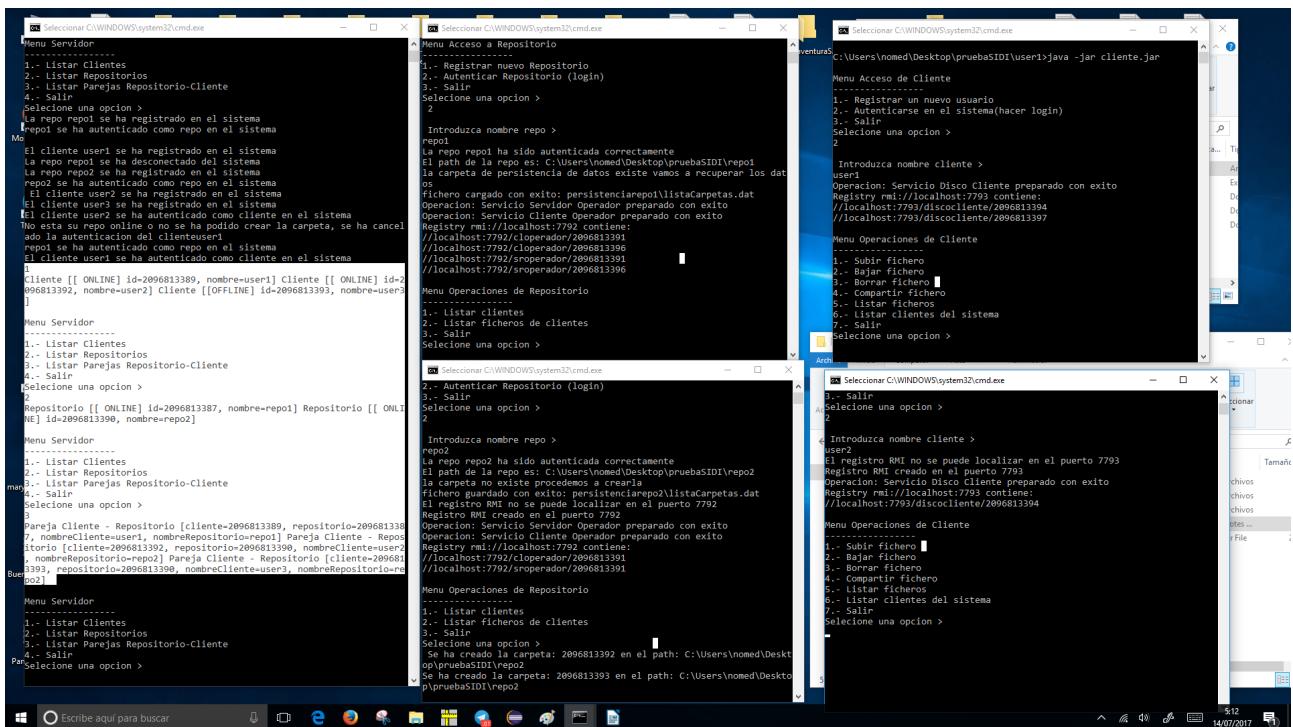


Imagen 42.- lista de clientes registrados y cuales estan online, lista de repositorios registrados y cuales están online. Parejas clientes repositorios con los id unicos.

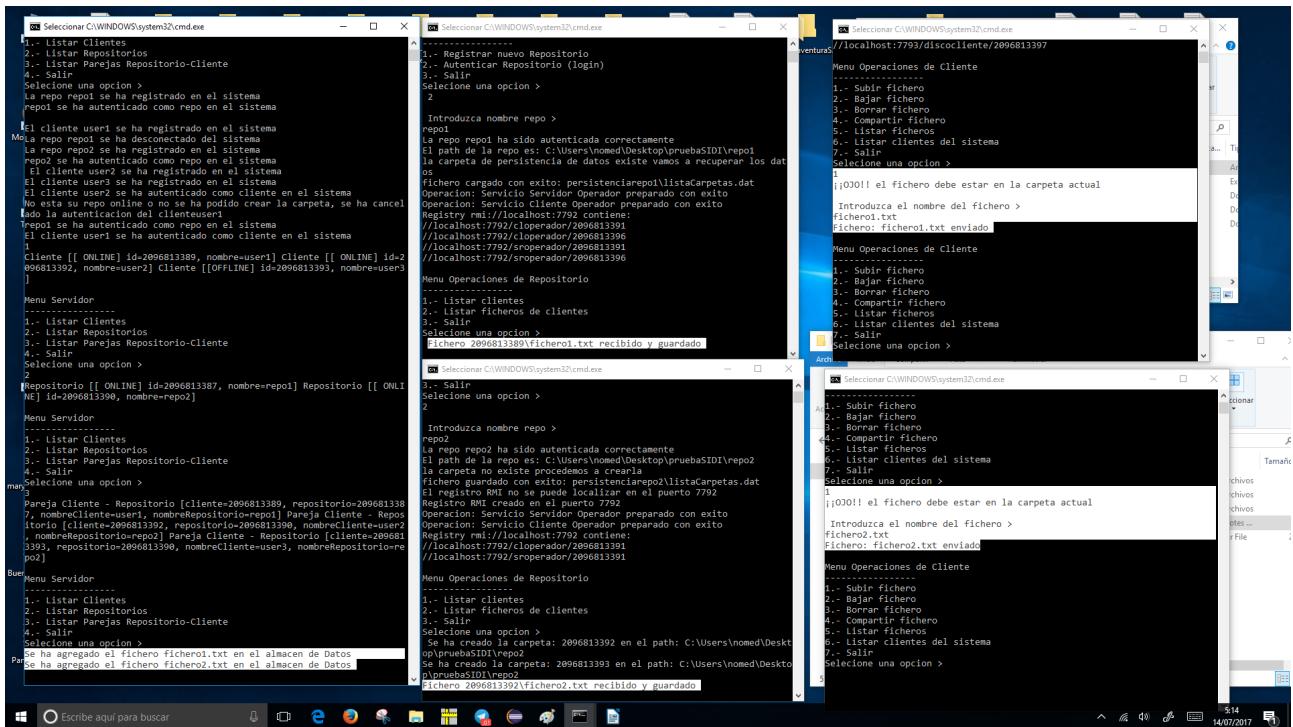


Imagen 43.- user 1 y user2 suben 2 ficheros, cada uno va a un repositorio

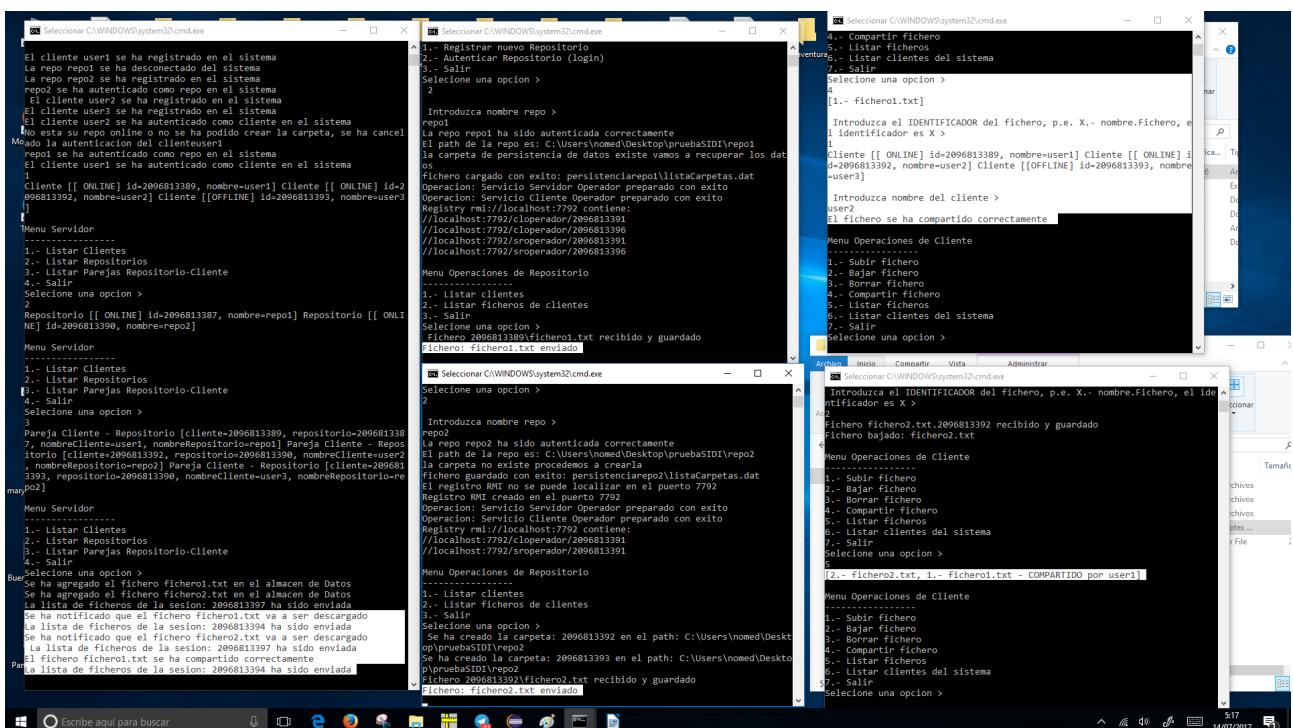


Imagen 44.- user1 y user2 bajan cada uno su fichero, los cuales veremos que tienen que quedarse en cada carpeta de cliente desde donde fue lanzado el jar, viendo que cada servicio discocliente funciona correctamente, guardando en la carpeta de ejecucion.

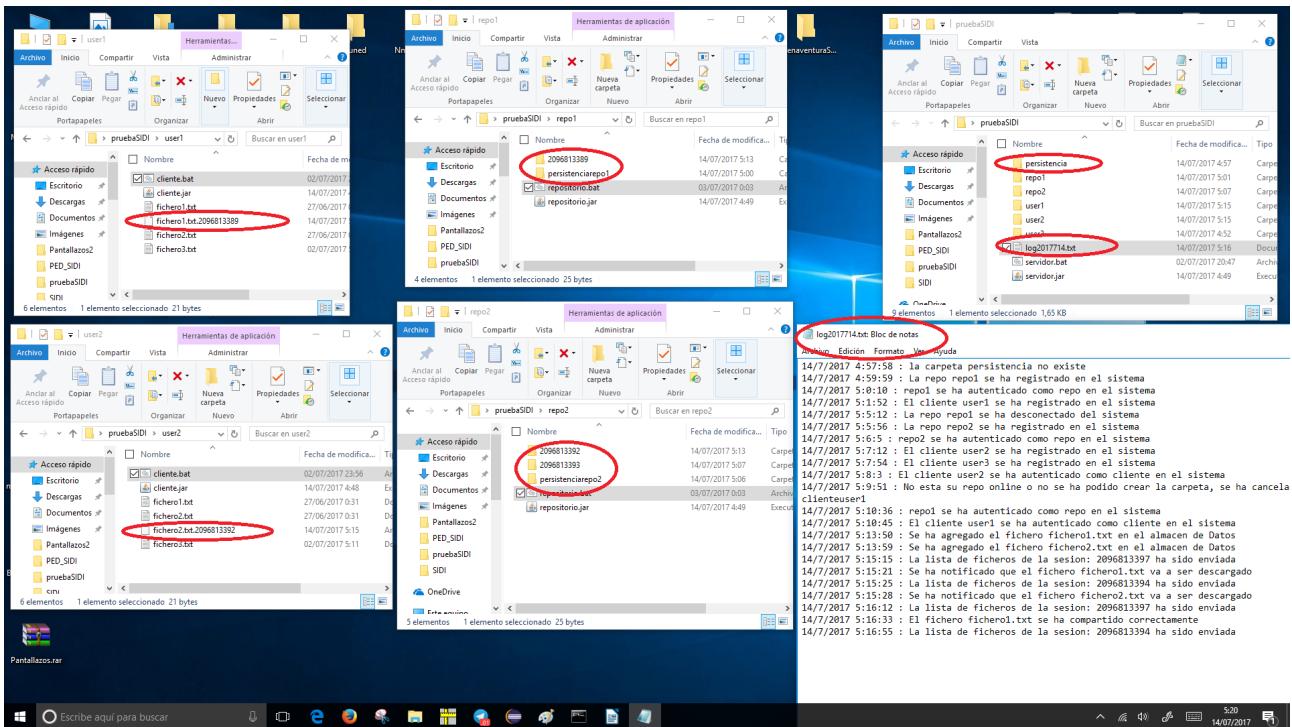


Imagen 45.- imagen de confirmación de que cada fichero subido va a cada carpeta de idúnico del cada repositorio, y cada fichero bajado va a cada carpeta.

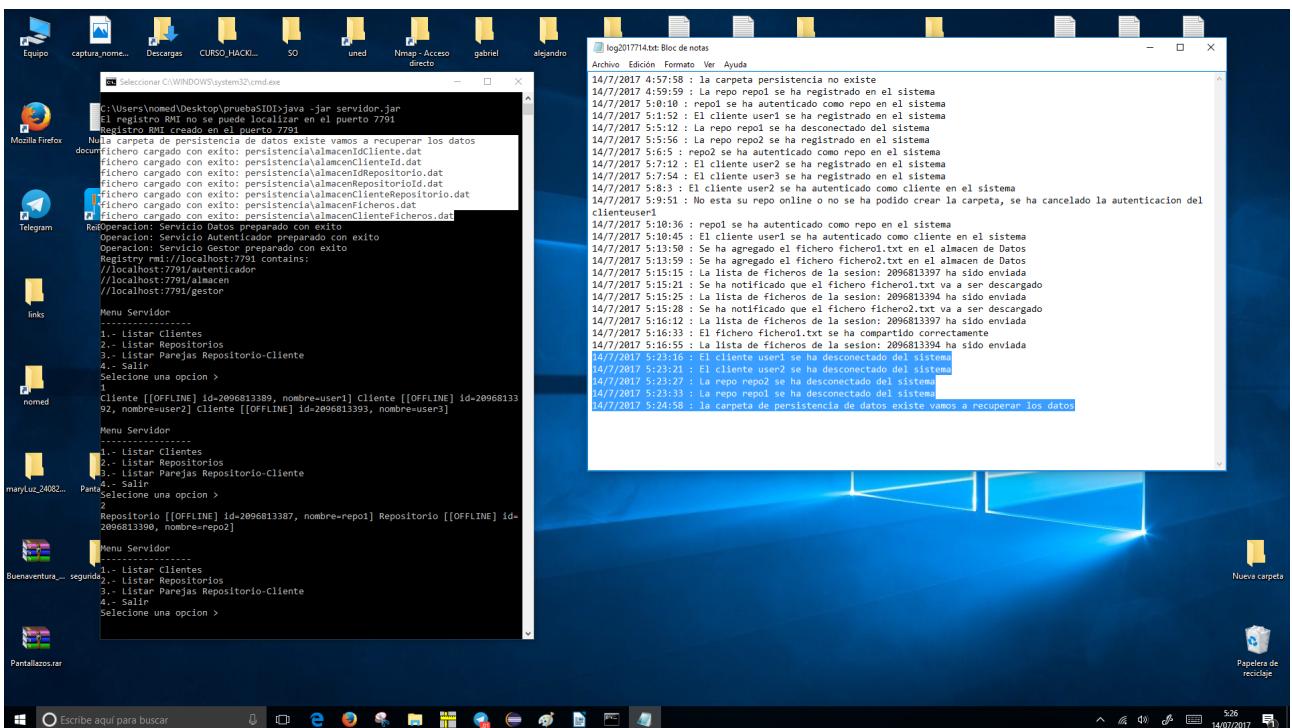


Imagen 46.- Hemos salido de los clientes y las repos y vemos como la persistencia detecta la carpeta al volver a arrancar el servidor cargando los id únicos y las estructuras logicas de los ficheros a traves de los metadatados cargados en el servidor, vemos el funcionamiento de la persistencia y vemos el log diario con las conexiones y desconexiones.

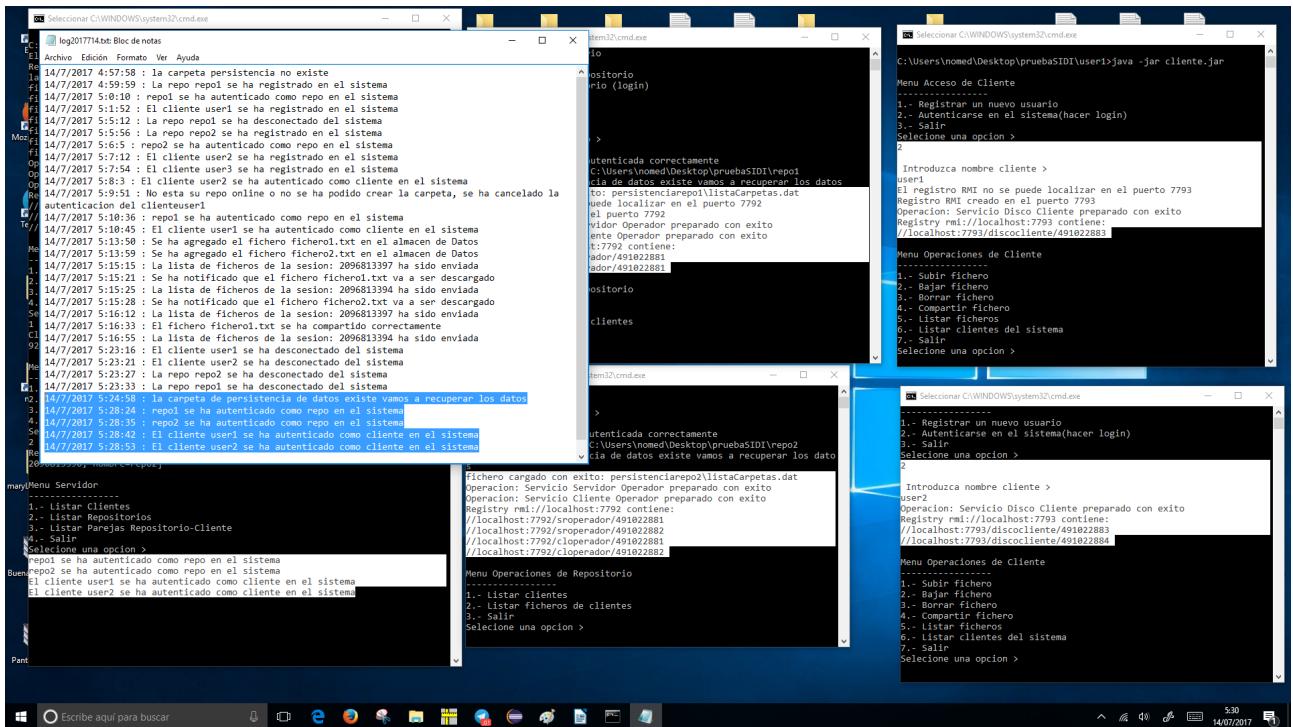


Imagen 47.- volvemos a conectar los repositorios y los clientes pero solo autenticando ya que la persistencia se encargo de guardar todos los registrados.

7.- Conclusiones y mejoras

La práctica ha sido amena y novedosa, en ella hemos visto como trabaja la invocación a objetos remotos. El proceso viene a resumirse en:

Servidor:

Comprobar si se ha colgado el puerto en el Registry y si no esta hacérselo saber.

Publicar las interfaces de los servicios que queremos en el puerto anterior.

Realizar las operaciones propias.

Eliminar los servicios.

Eliminar el puerto del registro.

Cliente*:

Buscar los servicios publicados recibiéndolos como objetos para usar sus métodos.

Realizar las operaciones propias.

Repositorio**:

Buscar los servicios publicados recibiéndolos como objetos para usar sus métodos.

Realizar las operaciones propias.

* Para este esquema tendremos en cuenta que el cliente cuelga su DiscoCliente, en cuyo caso lo

haría de igual forma que el servidor aunque en puertos distintos.

** Lo mismo que con el repositorio.

Entre las operaciones propias, tanto cliente como repositorios harán peticiones de datos y también colgaran sus propios servicios que no tienen porque coincidir con los puertos del Registry del servidor.

Los más tedioso de la practica han sido la excepciones, gracias que Eclipse nos avisa, en unos casos hemos optado por try y en otras por throws. Tratarlas sin que no se rompa la cadena de ejecución y que los métodos hagan lo que deben, resulta difícil, ya que algunos datos se pierden haciendo que los tratamientos de esos errores complicasen mucho el código y por tanto la corrección de la práctica, por esa razón se ha optado mayormente por throws y que por lo menos tengamos una idea de donde se produce el error. Espero que esta decisión no se vea repercutida en la nota.

Vamos destacar de nuevo, que fue interesante tratar con sesiones en vez de id únicos, para mantener la seguridad, y que no se puedan usar las credenciales de sesión una vez desconectado el cliente o repos.

La persistencia con las sesiones no se ha implementado, ya que nos dio un par de problemas, PARA QUE FUNCIONE BIEN TODO, DEBEN SIEMPRE CERRARSE LOS REPOSITORIOS Y EL SERVIDOR CORRECTAMENTE, sino los datos se perderán y puede haber inconsistencias entre los datos que hay en la estructura lógica de carpetas de los repositorios y las que hay en el servidor.

Mejoras que podría haber desarrollado mejor en la práctica:

- Soy consciente del abuso que he hecho de trows en vez de usar try-catch, unas veces por automatización de eclipse. He intentado generar las mínimas excepciones, espero haberlo conseguido y que ello no complique su corrección.
- Alguna comprobación en valores devueltos faltan aunque no parecen que ocasionen problemas podrían provocarlos.
- Quizás los mensajes y su formato de presentación son básicos, pero sencillo para no liar el código al igual que las confirmaciones de los datos introducidos desde teclado, que se intento en la medida de lo posible facilitar la labor pedida en el enunciado de sencillez.
- Evidente es que en entorno gráfico con ventanitas hubiese quedado más bonito y legible, pero también más trabajoso, al no tener un juego de clases por mi desarrollado para agilizar esta labor.
- El tema de la persistencia podría haberse mejorado, pero se ha incluido simplemente para que el corrector no tenga que estar registrando sino que se preocupe una vez registrado solo de la autenticación, cualquier problema derivada de la persistencia espero no sea penalizada, ya que repito se hizo por el simple hecho de que estuviese presente y que el facilitase la labor de corrección.

Mejoras en la arquitectura o enunciado de la práctica:

Se entiende que la arquitectura es completamente válida y apenas cabe objeción, pero sería interesante conocer más detalles, como cuales son realmente las funciones necesarias, para poder realizar las comparaciones con lo desarrollado en la práctica.

Una duda que he tenido durante toda la práctica, ha sido, si hubiese sido descabellado haber levantado cada carpeta de id único en los servicios de la repo en la forma:

rmi://direccion:puerto/servicioXXOperador/propietarioRepo/carpetaidUnicoCliente

En nuestro caso recordemos que hemos usado como propietarioRepo el idsesion de la repo cuando se autentica, me gustaría saber si ello ha sido una decisión correcta o no, si es caso afirmativo, por que no se ha propuesto más explicitamente en el enunciado.

También me hubiese gustado una comunicación más abierta entre los repositorios y el servidor, ya que recordemos la imagen 1 de la arquitectura, los repositorios no pueden hacer uso de otro servicio que no sea el de autenticar. Un poquito de libertad quizás hubiese estado bien.

* * *