

Curso 2016-2017

PRÁCTICA DE SISTEMAS DISTRIBUIDOS – SISTEMA BÁSICO DE ALMACENAMIENTO EN LA NUBE USANDO JAVA RMI

Manuel Rodríguez Sánchez | mrodrigue212@alumno.uned.es

Contenido

Enunciado	2
Operativa.....	3
Interfaz	3
1. Servidor	6
Inicialización del servidor	6
Listar clientes	7
Listar repositorios	8
Listar parejas Repositorio-Cliente	9
2. Repositorio	10
Inicializar repositorio	10
Registrar repositorio	11
Autenticación del repositorio.....	12
Menú de repositorio.....	15
Listar clientes	15
Listar ficheros de clientes	17
3. Clientes.....	20
Inicializar clientes	20
Registrar nuevo cliente	21
Autenticar cliente	22
Menú de cliente	25
Subir fichero.....	25
Bajar fichero.....	26
Borrar fichero	29
Listar ficheros.....	30
Listar clientes del sistema	32
4. Diagrama de clases	33
5. Conclusiones.....	35

Enunciado

El propósito de la práctica es el desarrollo de un software que implemente un sistema de almacenamiento de ficheros en la nube usando Java RMI, siguiendo el esquema que se expone a continuación:

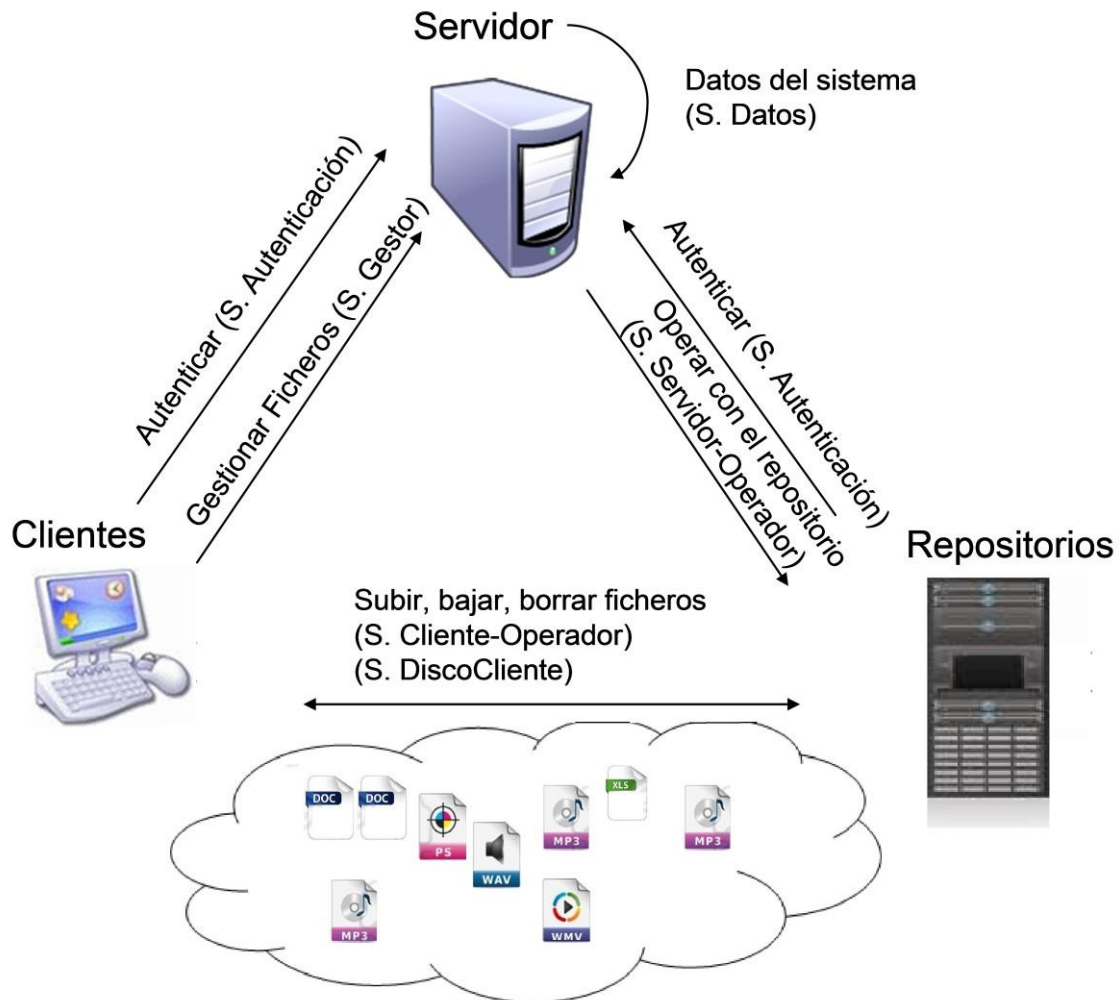


Ilustración 1 - Esquema del sistema distribuido.

En este sistema actuarán tres tipos de actores:

1.- Servidor: La entidad Servidor se encarga de controlar el proceso de almacenamiento de ficheros y de gestionar los recursos de almacenaje, para ello hace uso de tres servicios:

- **Servicio Autenticación:** Se encarga de registrar y de autenticar, cuando sea necesario, las otras entidades participantes en el sistema: clientes y repositorios.
- **Servicio Gestor:** Este servicio se encarga de gestionar las operaciones de los clientes en relación a sus ficheros en la nube (físicamente alojados en los repositorios).
- **Datos:** Este servicio hará las funciones de una base de datos que relacione Clientes-Ficheros-Metadatos-Repositorios. Es decir, mantendrá lista de clientes y repositorios conectados al sistema, junto con los ficheros; y los relacionarán permitiendo operaciones de consulta, borrado y añadido. Los dos servicios anteriores (Servicio Autenticación y Servicio Gestor) harán uso de este servicio para realizar las operaciones sobre el estado de las entidades del sistema y sus atributos.

2.- Repositorios: Estas entidades son las responsables de guardar en sus dispositivos de almacenamiento los ficheros que los clientes suben a la nube. Para hacer su función los repositorios hacen públicas las interfaces de sus dos servicios:

- Servicio Cliente-Operador: Este servicio se encarga de las operaciones de subida de ficheros al repositorio y borrado de los mismos.
- Servicio Servidor-Operador: Este servicio tiene un doble objetivo. Por un lado, suministra los métodos necesarios para que el servidor gestione los lugares de almacenamiento para cada cliente, y por otro lado se encarga de la operación de bajada de ficheros desde el repositorio al cliente.

3.- Clientes: Son los propietarios de los ficheros. Se registran en el sistema a través del servidor para poder subir, gestionar y almacenar sus ficheros en un repositorio en la nube. El cliente publica la interfaz de un servicio cuyo nombre es DiscoCliente que será utilizado por el servicio Servidor-Operador del repositorio para descargar al disco duro local del cliente el fichero que este considere oportuno.

El servidor **nunca se encarga de subir/bajar los ficheros**, sólo de gestionar estas operaciones y llevar un registro mediante su Servicio Datos. Estas operaciones costosas se harán desde los servicios Servidor-Operador y DiscoCliente para evitar cargar al servidor y permitir la escalabilidad del sistema fácilmente.

El sistema **sólo admitirá la gestión de ficheros** y no la creación de un árbol de carpetas por cada cuenta de cliente en el repositorio.

Cada repositorio creará una carpeta por cada cliente que alojará **todos los ficheros** del mismo. (Las carpetas se crearán en el *path* donde se encuentre el ejecutable del repositorio)

Operativa

Inicialmente la entidad servidor levanta sus tres servicios: Autenticación, Gestor y Datos.

El/Los repositorio/s se autentican en el sistema mediante el servicio Autenticación del servidor, devolviéndole éste un identificador único. El repositorio a partir de este momento está listo para almacenar los ficheros de los clientes.

El/Los cliente/s se autentican en el sistema mediante el servicio Autenticación del servidor, devolviéndole éste un identificador único. En este momento, el servidor le asigna al cliente un repositorio y guarda esta relación a través del servicio Datos. Además, el servidor manda crear una carpeta en el dispositivo de almacenamiento del repositorio mediante el servicio Servidor-Operador. Esta carpeta tendrá por nombre el identificador único del cliente y dentro se alojarán todos sus ficheros en propiedad.

Una vez que el cliente está registrado en el sistema, podrá realizar las operaciones de subida/bajada/borrado de ficheros en la nube. Operaciones gestionadas por el servidor y ejecutadas en la carpeta que cada cliente tiene en el repositorio que le corresponde.

Interfaz

- El Servidor debe permitir mediante su interfaz de texto o gráfica las siguientes operaciones:
 - 1.- Listar Clientes.
 - 2.- Listar Repositorios.
 - 3.- Listar Parejas Repositorio-Cliente.

4.- Salir.

- Los Repositorios deben permitir mediante su interfaz de texto o gráfica las siguientes operaciones:
 - 1.- Listar Clientes.
 - 2.- Listar ficheros del Cliente.
 - 3.- Salir.
- Los Clientes deben permitir mediante su interfaz de texto o gráfica las siguientes operaciones:
 1. Subir fichero.
 2. Bajar fichero.
 3. Borrar fichero.
 4. Compartir fichero (Opcional).
 5. Listar ficheros.
 6. Listar Clientes del sistema.
 7. Salir.

Cuando arrancan las aplicaciones de los Clientes y los Repositorios debe aparecer inicialmente un menú con las siguientes opciones antes de los menús anteriores. Éste debe permitir el registro de un nuevo usuario (cliente o repositorio según corresponda) en el sistema y/o autenticarse:

1. Registrar un nuevo usuario.
2. Autenticarse en el sistema (hacer login).
3. Salir

Podemos establecer el siguiente diagrama de casos de uso, el cual nos ayudará a desarrollar la aplicación y la memoria de este trabajo práctico.

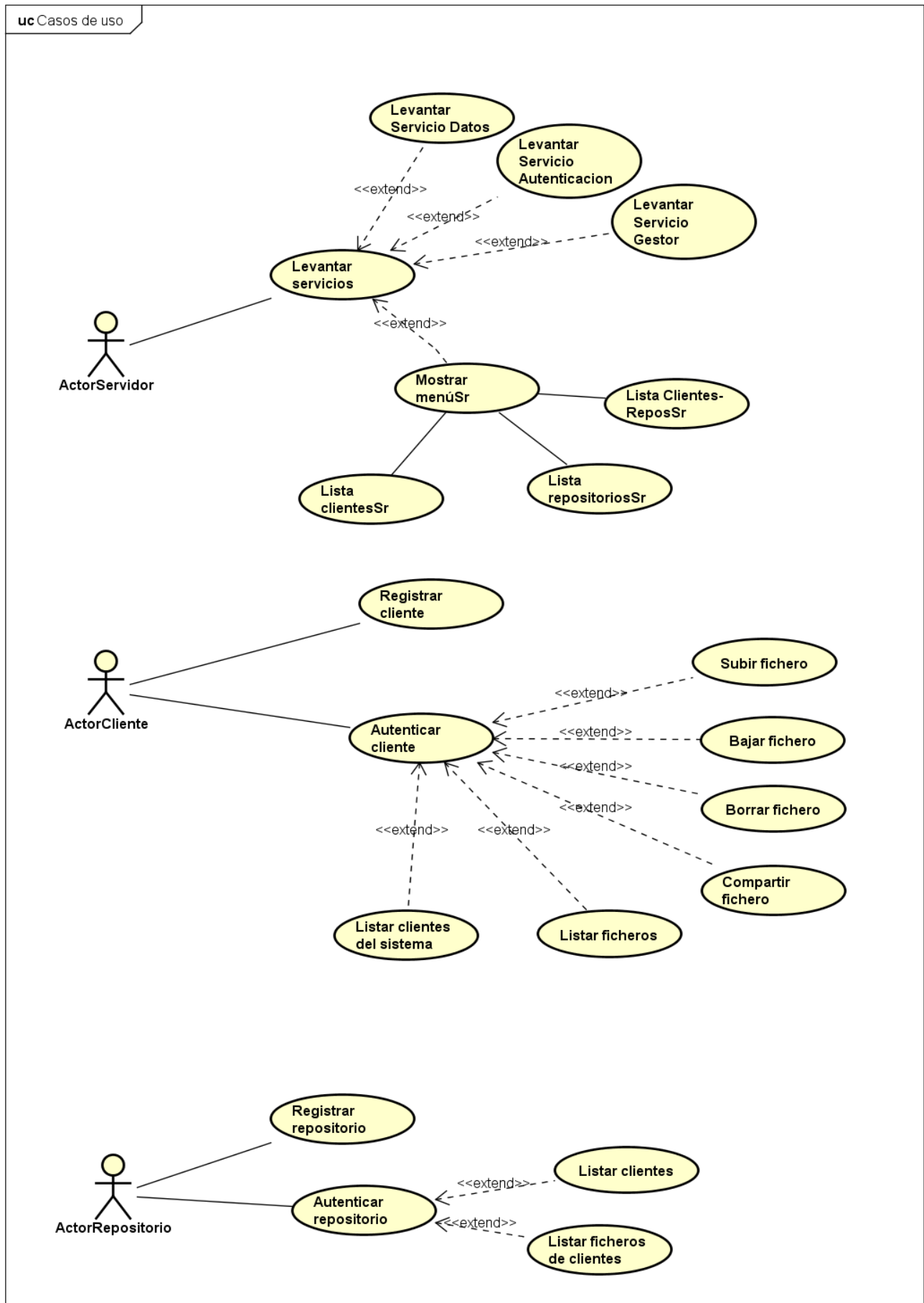


Ilustración 2 - Diagrama de casos de uso

1.Servidor

Se ha creado un fichero llamado **EjecutapRACTICA.bat**, que abre el servidor, repositorio y un cliente a la vez. La ejecución habría que hacerla desde el directorio donde se encuentran los ejecutables c:\..\bin. Aun así se especifica en cada apartado la forma de ejecución.

Inicialización del servidor

Para inicializar la aplicación Servidor, desde la consola del sistema escribimos:

java servidor.Servidor

Habría que hacerlo desde el path c:\..\bin

Servidor, se va a encargar de arrancar un registro RMI mediante el método `arrancarRegistro()`, donde a este método se le pasará el puerto donde queremos que se cargue el registro.

```
//método para arrancar el registro, si no existe, lo crea
private static void arrancarRegistro(int numPuertoRMI) throws RemoteException
{
    try{
        Registry registro=LocateRegistry.getRegistry(numPuertoRMI);
        registro.list();
    }
    catch(RemoteException e)
    {
        System.out.println("El registro no se puede localizar en el puerto:
"+numPuertoRMI);
        Registry registro=LocateRegistry.createRegistry(numPuertoRMI);
        System.out.println("Registro RMI creado en el puerto: "+numPuertoRMI);
    }//Fin catch
} //fin arrancarRegistro
```

Ilustración 3 – Método para arranque del registro RMI

El método comprobará si está arrancado, y si no lo está, lo arrancará mostrando al finalizar un mensaje en consola. Posteriormente procederá a levantar los tres servicios: datos, autenticación y gestor, creando con ello los objetos remotos "datos", "autenticación" y "gestor", exportando al registro RMI estos objetos remotos (y sus correspondientes métodos remotos) mediante la clase *Naming* y su método "rebind".

```
//se levanta el servicio de datos
ServicioDatosImpl datos=new ServicioDatosImpl();//Crea el objeto remoto.
Naming.rebind("rmi://localhost:"+nRegistro+"/datos", datos)

//se levanta el servicio de autenticación
ServicioAutenticacionImpl autenticacion = new ServicioAutenticacionImpl();
Naming.rebind("rmi://localhost:"+nRegistro+"/autenticacion", autenticacion);

//se levanta el servicio de gestión
ServicioGestorImpl gestor = new ServicioGestorImpl();
Naming.rebind("rmi://localhost:" + nRegistro + "/gestor", gestor);
```

Ilustración 4 - Creación y exportación de objetos remotos.

Una vez levantados los tres servicios, pintamos el menú que se indica en el enunciado.

```
Símbolo del sistema - java servidor.Servidor

C:\PEDSD\PracticaSD2016\bin>java servidor.Servidor
El registro no se puede localizar en el puerto: 2000

Registro RMI creado en el puerto: 2000

Servicios levantados.

-----
MENÚ SERVIDOR
-----
1-Listar clientes
2-Listar repositorios
3-Listar parejas Repositorio-Cliente
4-Salir
Selecciona opción: 1
```

Ilustración 5 – Resultado de la ejecución completa de la parte Servidor

Listar clientes

La opción *Listar Clientes*, nos va a mostrar todos los clientes que se han registrado en el sistema y que evidentemente, están almacenados en una la estructura tipo *hashmap*.

Al seleccionar la opción *1 – Listar clientes*, se hace una llamada al servicio datos, que es quien gestiona nuestra “base de datos” y quien nos devolverá una lista con los clientes registrados, mediante su método remoto *listaClientes()*. Este nos devolverá una estructura *Collection*, que se almacenará en variable *clientes*.

```
Collection<String> clientes = datos.listaClientes();
```

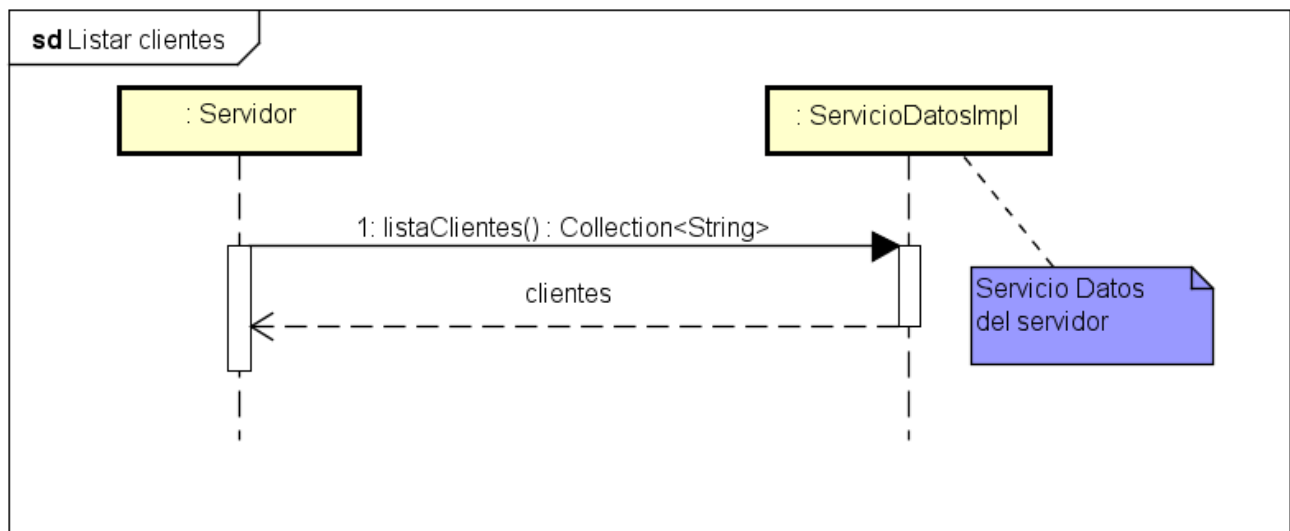


Ilustración 6 - Diagrama de secuencia Listar ClientesSr


```

-----
MENÚ SERVIDOR
-----
1-Listar clientes
2-Listar repositorios
3-Listar parejas Repositorio-Cliente
4-Salir
Selecciona opción: 1

Mostrando clientes registrados:
[ MANUEL ALBERTO SONIA ANGELA ]
  
```

Ilustración 7 - Resultado de la ejecución de la opción 1 - Listar Clientes

Si no encuentra clientes registrados, aparecerá mensaje indicándolo. Esto nos obligaría a dar de alta o registrar clientes desde la parte *Clientes*, en el caso de uso *Registrar clientes* que será abordado más adelante.

Listar repositorios

La opción *Listar repositorios*, nos va a mostrar todos los repositorios que se han registrado en el sistema y que evidentemente, están almacenados en una la estructura tipo *hashmap*.

Al seleccionar la opción 2 – *Listar repositorios*, se hace una llamada al servicio datos, que es quien gestiona nuestra “base de datos” y quien nos devolverá una lista con los repositorios registrados, mediante su método remoto ***listaRepositorios()***. Este nos devolverá una estructura ***Collection***, que se almacenará en variable *repositorios*.

```
Collection<String> repositorios = datos.listaRepositorios()
```

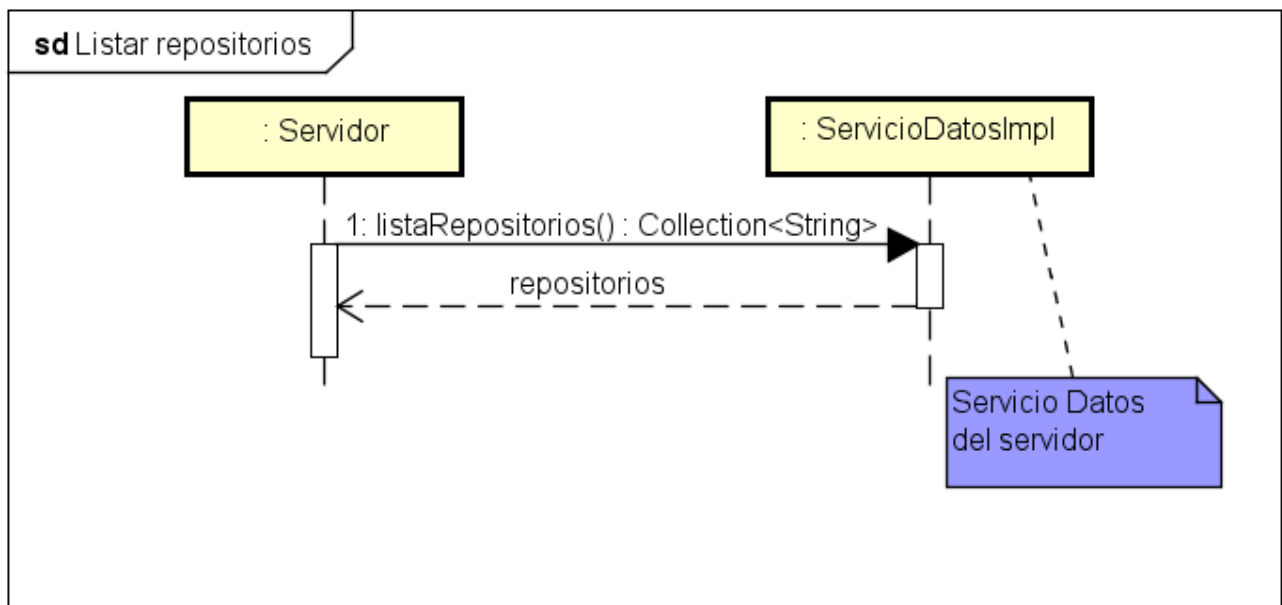
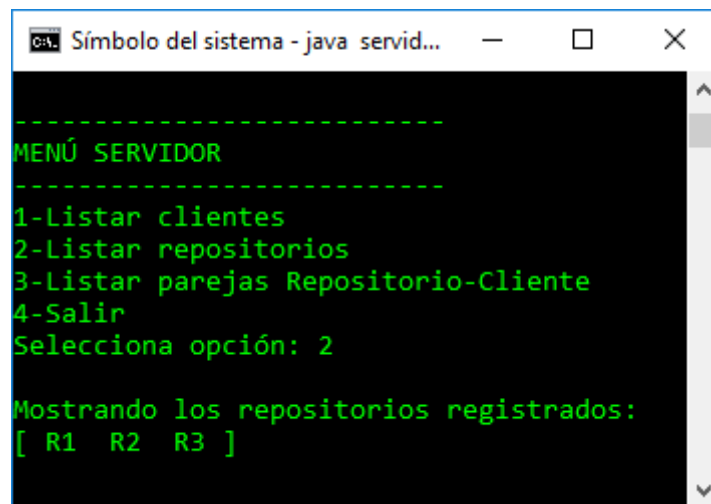


Ilustración 8 - Diagrama de secuencia Listar repositorios.



```
-----  
MENÚ SERVIDOR  
-----  
1-Listar clientes  
2-Listar repositorios  
3-Listar parejas Repositorio-Cliente  
4-Salir  
Selecciona opción: 2  
  
Mostrando los repositorios registrados:  
[ R1 R2 R3 ]
```

Ilustración 9 - Resultado de la ejecución de la opción 2 - Listar repositorios

Si no encuentra repositorios registrados, aparece mensaje indicándolo. Esto provoca que tengamos que dar de alta o registrar repositorios desde la parte *Repositorios* en el caso de uso *Registrar repositorio*, que será abordado más adelante.

Listar parejas Repositorio-Cliente

Cuando se autentica un cliente (no que se registra), automáticamente se le asigna un repositorio, que éste ha de estar también autenticado, a eso se refiere este caso de uso, donde el sistema *Servidor* nos va a mostrar el listado de los clientes autenticados y en el repositorio en el que están. Suponemos que existen varios clientes y repositorios autenticados previamente en el sistema; al seleccionar la opción 3 – *Listar parejas Repositorios-Cientes*, volvemos a llamar al servicio *Datos* y concretamente a su método remoto **listarRepositoriosClientes()**:

```
case 3://muestro las parejas Repositorios-clientes  
    datos.listarRepositoriosClientes();  
    break;
```

Ilustración 10 - Llamada a método remoto para listar las parejas repositorios-clientes

Este método remoto extrae de la estructura de datos Map *clienteRepositorio* los identificadores asociados, buscando en las estructuras Map *sesionCliente* y *sesiónRepositorio*, el nombre de cliente y nombre de repositorio, mostrando por la consola del servidor los resultados.

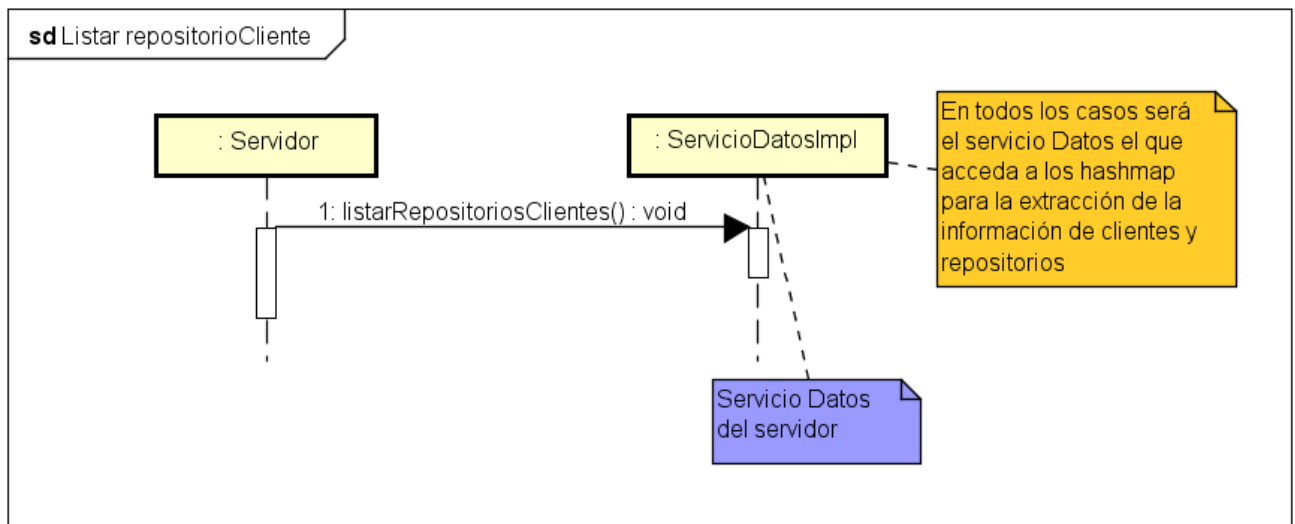


Ilustración 11 - Diagrama de secuencia Listar parejas repositorio-cliente

```

C:\A. Símbolo del sistema - java servid...
-----
MENÚ SERVIDOR
-----
1-Listar clientes
2-Listar repositorios
3-Listar parejas Repositorio-Cliente
4-Salir
Selecciona opción: 3

Repositorio: R1
Clientes asociados: MANUEL ANGELA

Repositorio: R2
Clientes asociados: SONIA ALBERTO
  
```

Ilustración 12 - Captura de consola con el resultado de la ejecución.

Si no hubiera ninguna asociación cliente-repositorio, aparecerá un mensaje indicándolo.

2.Repositorio

Inicializar repositorio

Para inicializar repositorio lanzamos desde la línea de comandos la orden:

```
java repositorio.Repositorio
```

Habría que hacerlo desde el path c:\..\bin

En la ejecución, necesitamos obtener la referencia al objeto remoto, del servicio autenticación, esto se hace con *lookup* de la clase *Naming*, ya que *repositorio* necesitará acceder al servidor a través de este servicio y sus métodos remotos, por que estos son los que pueden acceder al servicio datos del servidor, tal y como se indica en el esquema del enunciado, es decir, al servicio *Datos* que es donde estarán los repositorios y clientes registrados, autenticados y asociados, solo puede acceder el servidor.

```
autenticacion=(ServicioAutenticacionInterface)Naming.Lookup("rmi://localhost:2000/autenticacion");
```

Ilustración 13 - Línea de código para obtener la referencia al objeto Autenticación.

Posteriormente se pinta el primer menú de repositorio.

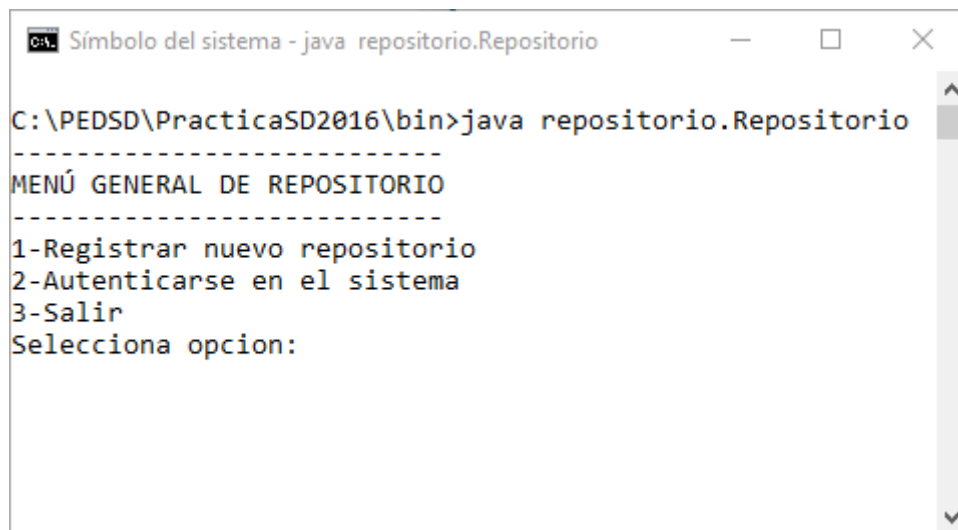


Ilustración 14 - Primer menú de repositorio.

Registrar repositorio

En este caso de uso, vamos a dar de alta o crear un nuevo repositorio. Este es un paso previo para posteriormente autenticarse en el sistema servidor y poner a disposición de la entidad que lo solicite, sus servicios.

Al seleccionar la opción *1- Registrar nuevo repositorio*, se pide el nombre que se le quiere dar al nuevo repositorio, recogido por la variable *nombre*, y se hace la llamada remota al método remoto *registraRepositorio(nombre)* perteneciente al servicio *Autenticación* del servidor, este método a su vez llamará al método remoto *altaRepositorio(nombre)* del servicio *Datos* del servidor, donde primero se comprobará si existe ese nuevo repositorio, y si no, se agrega al hashmap *registroRepositorio* devolviendo true a las anteriores llamadas, y mostrando mensaje por consola de *Repositorio* con el resultado de la operación.

```
//Llamada remota para el alta del repositorio
if(autenticacion.registraRepositorio(nombre))

//Llamada remota al servicio datos desde servicio autenticación
if (datos.altaRepositorio(nombreRepositorio))

//Agregar al hashmap el código(ya calculado) y el nombre del repositorio
//con el método altaRepositorio del servicio datos
registroRepositorio.put(codRepositorio, nombre.toUpperCase());
```

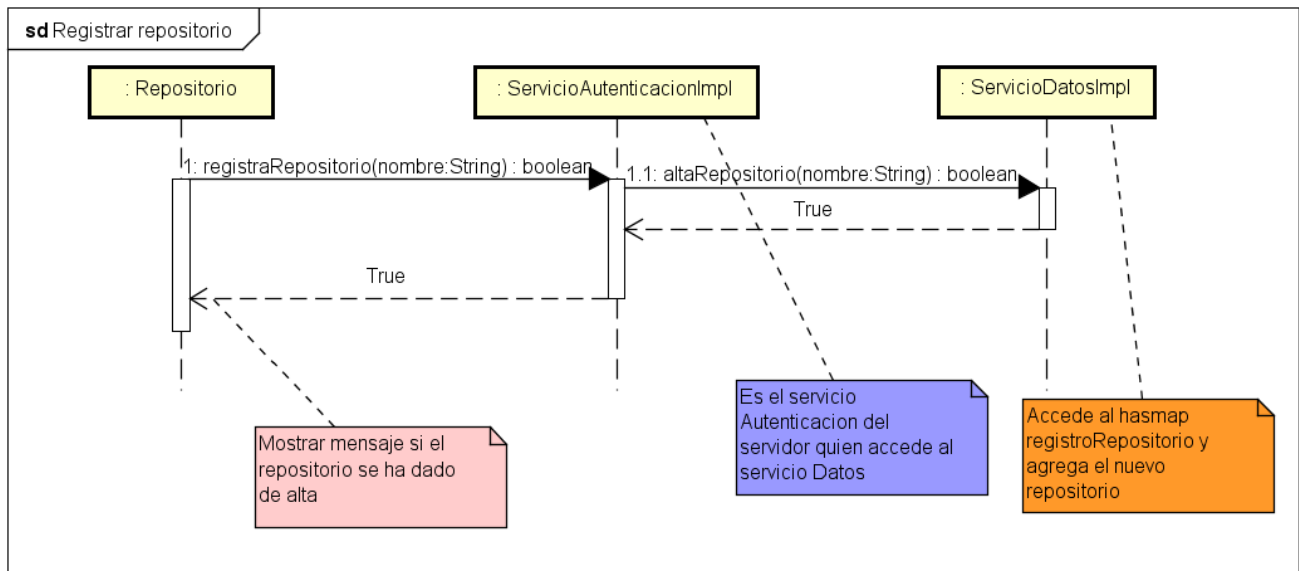


Ilustración 15 - Diagrama de secuencia Registrar repositorio

```

C:\. Símbolo del sistema - java repositorio.Repositorio
MENÚ GENERAL DE REPOSITORIO
-----
1-Registrar nuevo repositorio
2-Autenticarse en el sistema
3-Salir
Selecciona opcion:1

Introduce nombre del repositorio:R5

Se ha registrado el repositorio: R5

-----
MENÚ GENERAL DE REPOSITORIO
-----
1-Registrar nuevo repositorio
2-Autenticarse en el sistema
3-Salir
Selecciona opcion:
  
```

Ilustración 16 - Resultado de la ejecución correcta de nuevo repositorio registrado

Autenticación del repositorio

La autenticación del repositorio va a permitir levantar los servicios Servidor - Operador y Cliente - Operador, y con ello poner a disposición de quien lo solicite los servicios y métodos correspondientes. Es imprescindible que el repositorio esté autenticado para que los clientes puedan hacer uso de él. Si no hubiera ningún repositorio autenticado, los clientes no pueden autenticarse; lo veremos más adelante en la parte cliente.

Por otro lado, el hecho de que no haya un repositorio autenticado, no significa que no haya repositorios registrados o creados.

Al seleccionar la opción 2- *Autenticarse en el sistema*, nos pedirá el nombre del repositorio, recogido por la variable *nombre*, a continuación llamamos al servicio *Autenticación* del servidor mediante el método remoto **autenticarRepositorio()**, pasándole el nombre del repositorio.

```
int idSesionRepositorio = autenticacion.autenticarRepositorio(nombre);
```

El servicio autenticación se encargará de calcular un código de sesión mediante el método local getSession(), este número junto con el nombre será enviado al servicio datos del servidor mediante el método remoto:

```
if (datos.guardarAutenticacionRepositorio(idSesionRepositorio,nombre))
    return idSesionRepositorio;
```

para almacenar en el hashmap sesionRepositorio, el repositorio autenticado:

```
sesionRepositorio.put(idSesionRepositorio, nombre);
return true;
```

Una vez volvemos del servidor, con las operaciones correctamente realizadas, y con el número de sesión de repositorio, procedemos a levantar los dos servicios que corresponden al Repositorio: Cliente-Operador y Servidor-Operador.

```
// LEVANTAMOS EL SERVICIO CLIENTE-OPERADOR(Cliente comunica bidireccionalmente con
//Repositorio)
ServicioClOperadorImpl clienteOperador = new ServicioClOperadorImpl();
String URL_nombreClienteOperador = "rmi://localhost:2000/ServicioClOperador/" +
idSesionRepositorio;
Naming.rebind(URL_nombreClienteOperador, clienteOperador);

// LEVANTAMOS EL SERVICIO SERVIDOR-OPERADOR (Servidor comunica unidireccionalmente con
//Repositorio)
ServicioSrOperadorImpl servidorOperador = new ServicioSrOperadorImpl();
String URL_nombreServidorOperador = "rmi://localhost:2000/ServicioSrOperador/" +
idSesionRepositorio;
Naming.rebind(URL_nombreServidorOperador, servidorOperador);
```

A continuación pintamos el menú secundario de repositorio, tal y como se indica en el enunciado.

```
Símbolo del sistema - java repositorio.Repositorio
C:\PEDSD\PracticaSD2016\bin>java repositorio.Repositorio

-----
MENÚ GENERAL DE REPOSITORIO
-----
1-Registrar nuevo repositorio
2-Autenticarse en el sistema
3-Salir
Selecciona opcion:2

Introduce el nombre del REPOSITORIO para proceder a la autenticación de éste:
r1

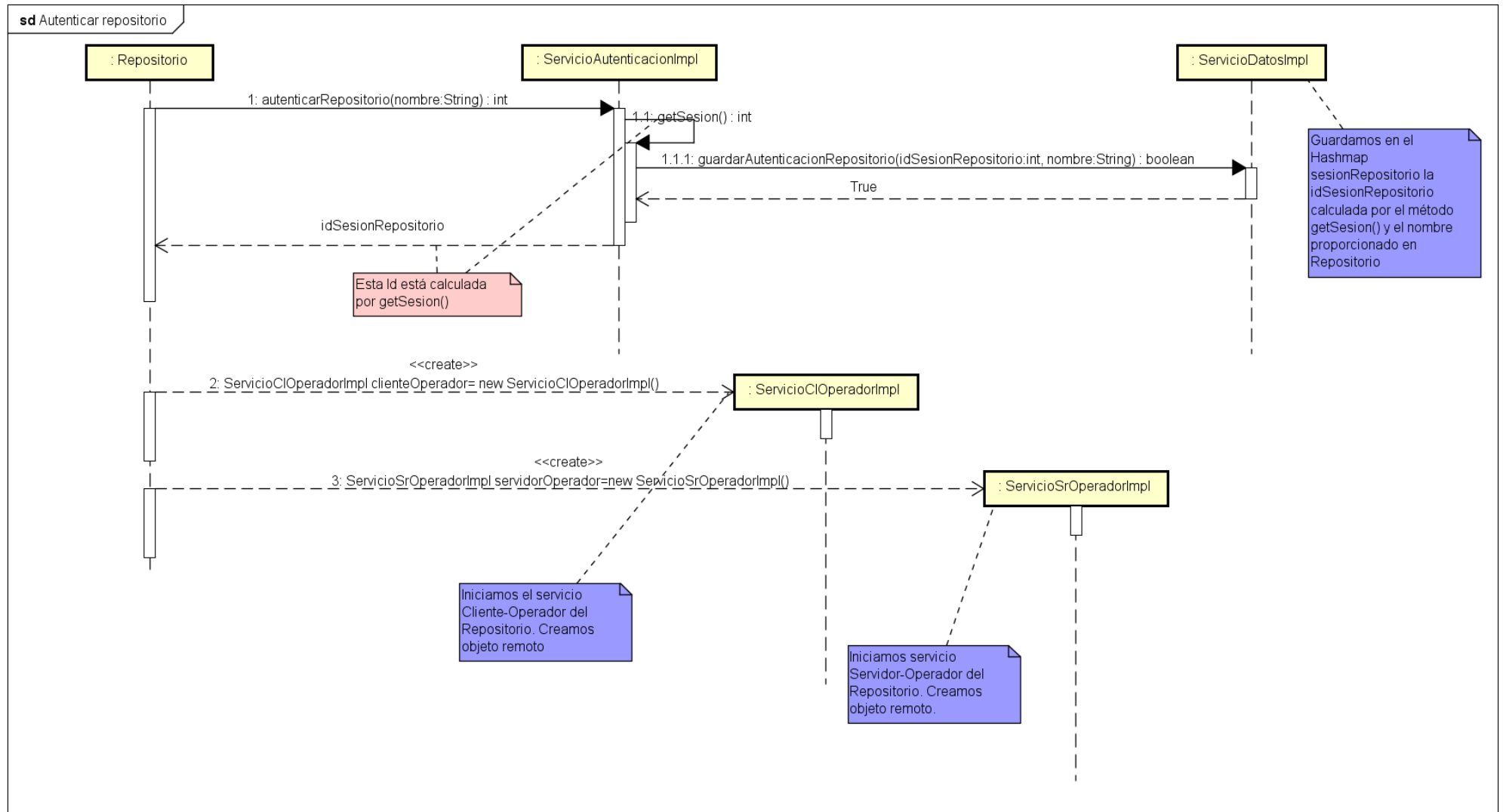
El número de sesion asignado es: 86040749

Servicio Cliente Operador levantado.

Servicio Servidor Operador levantado.

-----
MENÚ DE REPOSITORIO
-----
1.- Listar Clientes.
2.- Listar Ficheros del Cliente.
3.- Salir.
Selecciona opción:
```

Ilustración 17 – Resultado una vez autenticado el repositorio.

Ilustración 18 - Diagrama de secuencia **Autenticar Repositorio**

Menú de repositorio

Listar clientes

La opción 1- *Listar clientes*, listará los clientes autenticados y asociados al repositorio. *Repositorio* hará una llamada al servicio *autenticación* del servidor mediante el método remoto ***listaClientesDeRepositorio(idSesionRepositorio)***, donde le pasamos el número de sesión del repositorio. Este método le devolverá una lista en forma de array con los clientes pertenecientes a ese repositorio.

```
Lista=new ArrayList<String>();
```

```
Lista=autenticacion.listaClientesDeRepositorio(idSesionRepositorio);
```

Dado que el servicio autenticar no tiene acceso a la "base de datos", este realiza otra llamada al método remoto del servicio datos ***recogeListaClientesRepo(idSesionRepositorio)***.

```
public ArrayList<String> listaClientesDeRepositorio(int idSesionRepositorio) throws
RemoteException{
    ArrayList<String> listaClientesRepo=new ArrayList<String>();
    listaClientesRepo=datos.recogeListaClientesRepo(idSesionRepositorio);
    return listaClientesRepo;
}
```

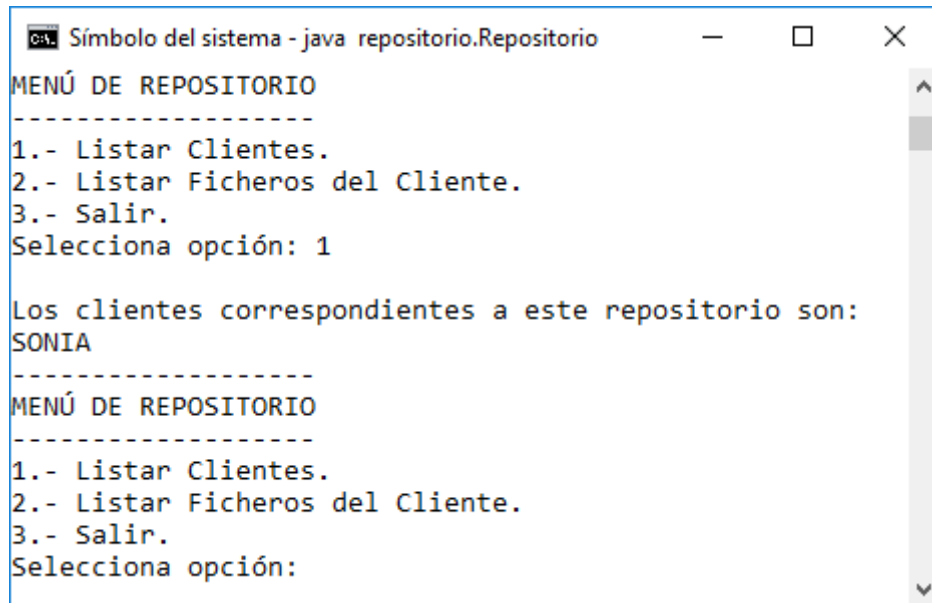
Este método también devolverá una lista en forma de array llamada *listaClientesRepo*:

```
public ArrayList<String> recogeListaClientesRepo(int idSesionRepositorio) throws
RemoteException{
    ArrayList<Integer> clientes = new ArrayList<Integer>(sesionCliente.keySet());
    ArrayList<String> listaClientesRepo=new ArrayList<String>();
    for (int c : clientes) {
        if (clienteRepositorio.get(c) == idSesionRepositorio) {
            listaClientesRepo.add(sesionCliente.get(c));
        }
    }
    return listaClientesRepo;
}
```

Una vez devuelta la lista, *Repositorio* será quien muestre en consola el listado de clientes asociado al repositorio.

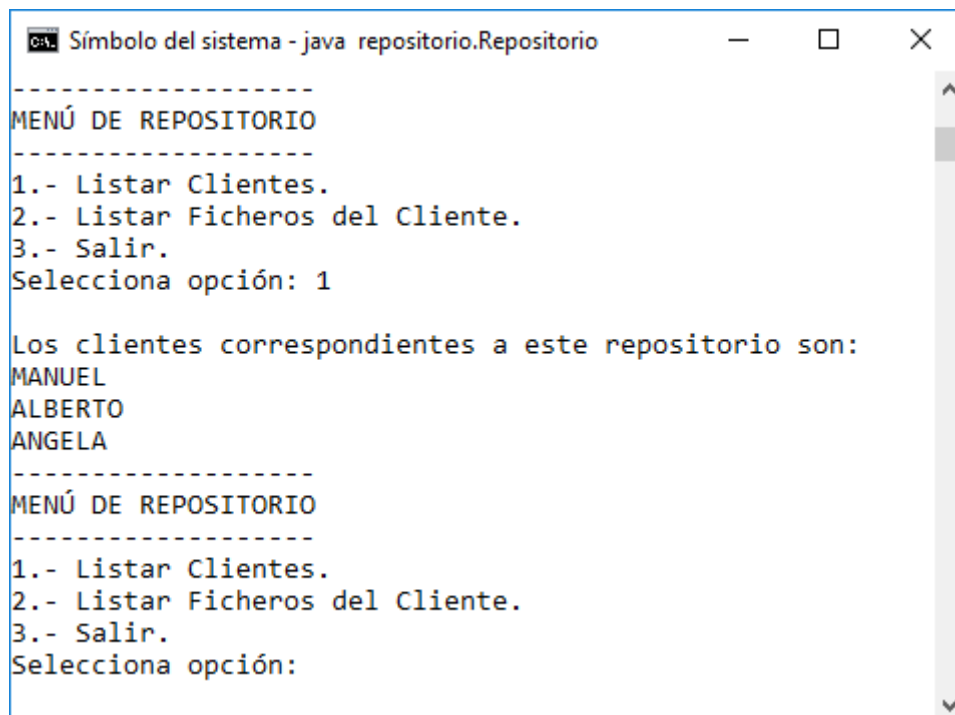
```
System.out.println("Los clientes correspondientes a este repositorio son:");
for(int x=0;x<Lista.size();x++) {
    System.out.println(Lista.get(x));
}
```

A continuación podemos observar el resultado de listar los clientes en dos repositorios levantados y autenticados, junto con el diagrama de secuencia correspondiente.



```
Símbolo del sistema - java repositorio.Repositorio
MENÚ DE REPOSITORIO
-----
1.- Listar Clientes.
2.- Listar Ficheros del Cliente.
3.- Salir.
Selecciona opción: 1

Los clientes correspondientes a este repositorio son:
SONIA
-----
MENÚ DE REPOSITORIO
-----
1.- Listar Clientes.
2.- Listar Ficheros del Cliente.
3.- Salir.
Selecciona opción:
```

Ilustración 19 - Listado de clientes en repositorio 1

```
Símbolo del sistema - java repositorio.Repositorio
-----
MENÚ DE REPOSITORIO
-----
1.- Listar Clientes.
2.- Listar Ficheros del Cliente.
3.- Salir.
Selecciona opción: 1

Los clientes correspondientes a este repositorio son:
MANUEL
ALBERTO
ANGELA
-----
MENÚ DE REPOSITORIO
-----
1.- Listar Clientes.
2.- Listar Ficheros del Cliente.
3.- Salir.
Selecciona opción:
```

Ilustración 20 - Clientes listados del repositorio 2.

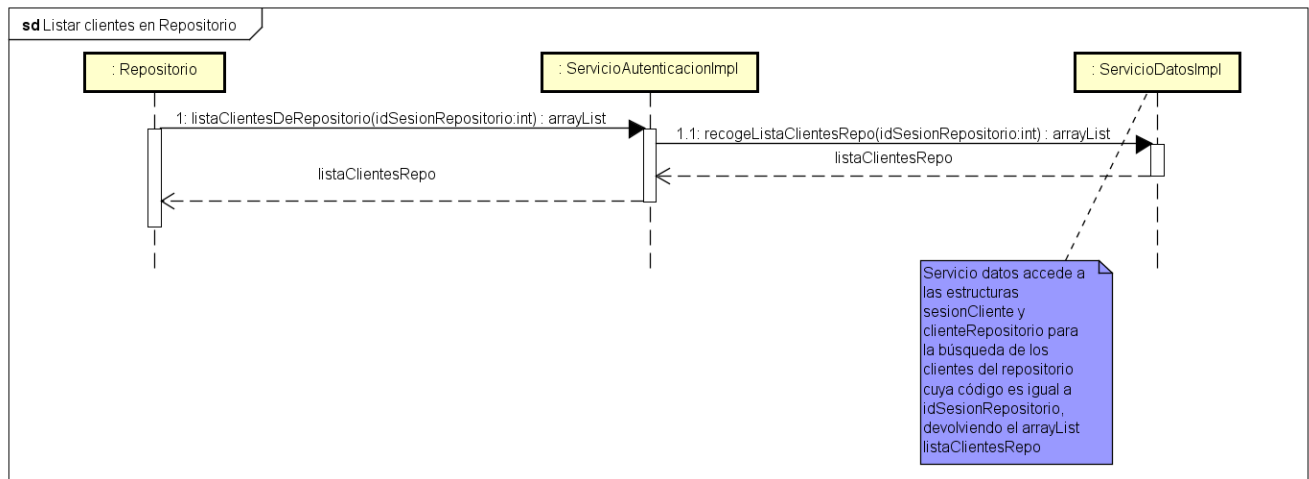


Ilustración 21 – Diagrama de secuencia Listar clientes asociados a repositorio.

Listar ficheros de clientes

Esta opción del segundo menú de *Repositorio*, nos va a permitir listar los archivos de un cliente autenticado, y asociado al repositorio desde el que se está pidiendo el listado. Este listado se lo vamos a pedir al servicio *Gestor* del servidor, pero como *Repositorio* no tiene acceso a este servicio, utilizaremos como intermediario al servicio *Disco-Cliente*, para acceder desde el cliente al servicio *Gestor*, y que este último se comuniqué con el servicio *Datos* del servidor, para extraer la lista de archivos del cliente que le hayamos pasado. Cuando se obtengan los resultados, el servidor los mandará en forma de *arrayList* al repositorio, a través de su servicio *Servidor-Operador*.

Descripción detallada del proceso:

Al seleccionar la opción 2 – *Listar ficheros de cliente*, el sistema pedirá el nombre del cliente para el que queremos mostrar los ficheros que tiene en su carpeta. Recogemos el nombre a través de la variable **nombre**, obtenemos la referencia al objeto remoto *ServicioDiscoCliente* y llamamos a su método remoto **listarFicherosCliente()** pasándole el código de sesión de repositorio y el nombre del cliente recogido anteriormente.

```

System.out.println("Introduce el nombre del cliente: ");
nombre=LecturaConsola.readLine();
SrDiscoCliente=(ServicioDiscoClienteInterface)
Naming.lookup("rmi://localhost:2000/DiscoCliente/"+idSesionRepositorio);

if(!SrDiscoCliente.listarFicherosCliente(idSesionRepositorio, nombre.toUpperCase()))
    System.out.println("No existe el cliente en este repositorio");
  
```

Ilustración 22 - Acceso al cliente a través de su servicio Disco-Cliente

Una vez en el cliente, desde el método obtenemos la referencia al objeto remoto *ServicioGestor* desde el que accederemos a su método remoto *buscarFicherosClientes()* enviándole el número de sesión de repositorio y el nombre del cliente.

```

public boolean listarFicherosCliente(int idRepos,String nombreCliente) throws
RemoteException, MalformedURLException, NotBoundException
{
    ServicioGestor=(ServicioGestorInterface) Naming.lookup("rmi://localhost:2000/gestor");
    if(!ServicioGestor.buscarFicherosClientes(idRepos, nombreCliente))
        return false;
    else return true;}
  
```

Ilustración 23 - Acceso al servicio Gestor

Una vez en el servicio *Gestor*, ya podemos acceder al servicio *Datos* y lo primero que vamos a hacer es comprobar que el cliente está en el repositorio, para ello llamamos al método remoto de *Datos* **comprobarRepositorioCliente()**, y le pasamos el código de sesión de repositorio y el nombre del cliente.

```
if (datos.comprobarRepositorioCliente(idSesionRepositorio, nombreCliente))
```

Este método llamará al método local `recogeListaClientesRepo()`, pasándole el código de sesión de repositorio.

```
public boolean comprobarRepositorioCliente(int idSesionRepositorio, String
nombreCliente) throws RemoteException{
    ArrayList<String> clientesDeRepositorio=new ArrayList<String>();
    clientesDeRepositorio=recogeListaClientesRepo(idSesionRepositorio);
    if (clientesDeRepositorio.contains(nombreCliente))return true;
    else return false;
}
```

```
public ArrayList<String> recogeListaClientesRepo(int idSesionRepositorio) throws
RemoteException{
    ArrayList<Integer> clientes = new ArrayList<Integer>(sesionCliente.keySet());
    ArrayList<String> listaClientesRepo=new ArrayList<String>();
    for (int c : clientes) {
        if (clienteRepositorio.get(c) == idSesionRepositorio) {
            listaClientesRepo.add(sesionCliente.get(c));
        }
    }
    return listaClientesRepo;
}
```

Ilustración 24 - Métodos del servicio Datos para comprobación de existencia de cliente.

Con esto accederá al hashmap `sesionCliente` y `clienteRepositorio`, para buscar los clientes del repositorio; devolverá un `ArrayList` llamado **clientesDeRepositorio**, comprobando la existencia del cliente mediante la expresión **clientesDeRepositorio.contains(nombreCliente)**.

Si la comprobación ha sido exitosa extraemos los ficheros del cliente que tenga en su carpeta mediante el método remoto de *Datos* **extraeListaFicheros()**, y le pasamos el nombre del cliente. Este método nos devolverá un *ArrayList* que lo almacenaremos en objeto del mismo tipo **ListaFicheros**.

```
ListaFicheros=datos.extraeListaFicheros(nombreCliente);
```

Posteriormente obtenemos la referencia al objeto remoto *ServidorOperador*, para acceder a sus métodos remotos, con esto conseguimos mandar los resultados a *Repositorio* a través de su servicio y llamando al método remoto **mostrarListaFicheros()**, pasándole el objeto *ArrayList* **ListaFicheros**.

```
servicioSrOp=(ServicioSrOperadorInterface)
Naming.lookup("rmi://localhost:2000/ServicioSrOperador/"+idSesionRepositorio);
servicioSrOp.mostrarListaFicheros(ListaFicheros);
```

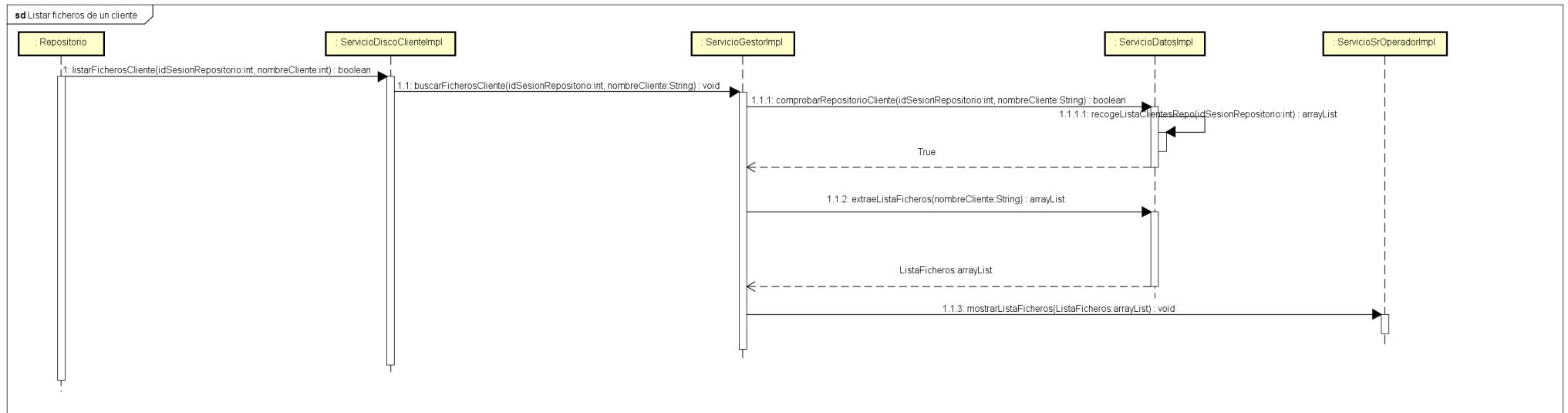


Ilustración 25 - Diagrama de secuencia Listar ficheros de un cliente desde repositorio

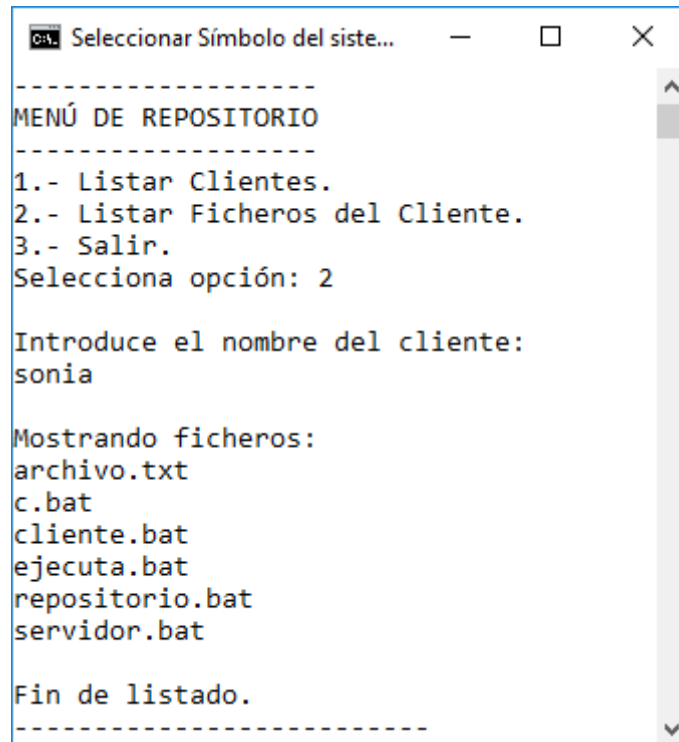


Ilustración 26 - Captura de consola con el resultado de la ejecución **Listar ficheros del cliente**

3.Clientes

Inicializar clientes

Para inicializar el cliente lanzamos desde la línea de comandos la orden:

java cliente.Cliente

Habría que hacerlo desde el path `c:\..\bin`

En la ejecución, necesitamos obtener la referencia al objeto remoto, del servicio autenticación, esto se hace con *lookup* de la clase *Naming*, ya que *cliente* necesitará acceder al servidor a través de este servicio y sus métodos remotos, ya que estos son los que pueden acceder al servicio *Datos* del servidor, tal y como se indica en el esquema del enunciado, es decir, al servicio *Datos*, que es donde estarán los clientes registrados, autenticados y asociados a repositorios, solo puede acceder el servidor.

```
ServicioAutenticacion =  
(ServicioAutenticacionInterface) Naming.Lookup("rmi://localhost:2000/autenticacion");  
servicioGestor=(ServicioGestorInterface) Naming.Lookup("rmi://localhost:2000/gestor");
```

Posteriormente se pinta el menú principal de clientes:

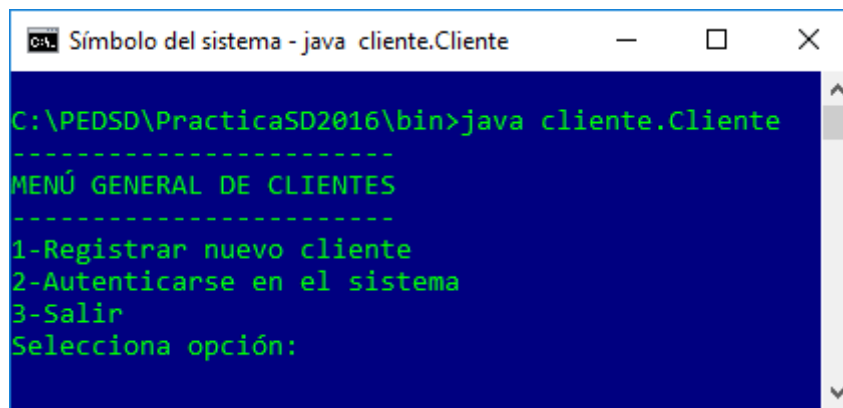


Ilustración 27 - Primer menú de clientes una vez lanzada la ejecución.

Registrar nuevo cliente

En este caso de uso, vamos a dar de alta o crear un nuevo cliente. Este es un paso previo para posteriormente autenticarse en el sistema servidor y poner a disposición de la entidad que lo solicite, su servicio Disco-Cliente y poder realizar todas las operaciones de subida, borrado y bajada de ficheros, entre otros.

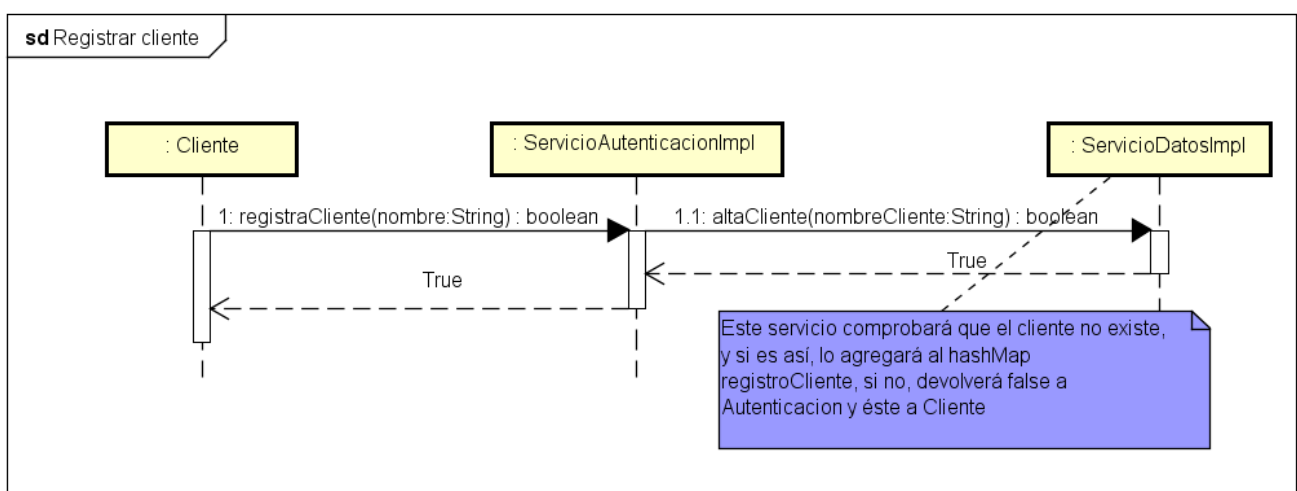
Al seleccionar la opción *1- Registrar nuevo cliente*, se pide el nombre que se le quiere dar al nuevo cliente, recogido por la variable *nombre*, y se hace la llamada remota al método remoto *registraCliente()* perteneciente al servicio *Autenticación* del servidor, pasándole el nombre del cliente.

```
if(ServicioAutenticacion.registraCliente(nombre))
```

Este método a su vez llamará al método remoto *altaCliente(nombre)* del servicio *Datos* del servidor, donde primero se comprobará si existe ese cliente, y si no, se agregará al hashmap *registroCliente*, devolviendo true a las anteriores llamadas, y mostrando mensaje por consola de *Cliente* con el resultado de la operación.

```
datos = (ServicioDatosInterface) Naming.Lookup("rmi://localhost:2000/datos");
if(datos.altaCliente(nombre))
    return true;
else
    return false;
```

Puede observarse que previamente se referencia el objeto remoto mediante *Naming.Lookup*.



```

-----
MENÚ GENERAL DE CLIENTES
-----
1-Registrar nuevo cliente
2-Autenticarse en el sistema
3-Salir
Selecciona opción: 1
Introduce nombre del cliente:
Isabel
Se ha registrado el cliente: Isabel
-----
MENÚ GENERAL DE CLIENTES
-----

```

Ilustración 28 - Resultado en consola del registro de un nuevo cliente.

Autenticar cliente

La autenticación del cliente va a permitir, en primer lugar levantar el servicio Disco-Cliente, y en segundo lugar acceder al menú secundario de cliente. Desde este menú el cliente podrá realizar todas las operaciones de subida, bajada, borrado y listados de ficheros. Tal y como se comentó anteriormente es imprescindible que, como mínimo, haya un repositorio autenticado para que los clientes puedan hacer uso de él. Si no hubiera ningún repositorio autenticado, los clientes no pueden autenticarse.

Al seleccionar la opción 2- *Autenticarse en el sistema*, nos pedirá el nombre del cliente, que será recogido por la variable **nombre**, a continuación llamamos al servicio *Autenticación* del servidor mediante el método remoto **autenticarCliente()**, pasándole el nombre del cliente. Aquí buscamos que el servidor realice dos tareas:

- Controlar que el cliente exista y que no esté autenticado.
- Conseguir un código de sesión de cliente (idSesionCliente).

```
int idSesionCliente = ServicioAutenticacion.autenticarCliente(nombre);
```

Las dos tareas se va a encargar de hacerlas el servicio Datos, el servicio autenticación tan solo calculará un código de sesión mediante el método getSession(), este código será enviado junto con el nombre, en la llamada al método remoto del servicio Datos: guardarAutenticacionCliente().

```
int idSesionCliente = getSession();
```

Previamente, obtenemos referencia al objeto remoto del servicio *Datos*.

```

if (datos.guardarAutenticacionCliente(idSesionCliente, nombre)) {
    servicioGestor=(ServicioGestorInterface)
    Naming.Lookup("rmi://localhost:2000/gestor");
    String
    direccionServicioSrOp=servicioGestor.obtenerServicioServidorOperador(idSesionCliente);
    accesoMetodosSrOp=(ServicioSrOperadorInterface)
    Naming.Lookup(direccionServicioSrOp);
    accesoMetodosSrOp.crearCarpeta(nombre);
}
return idSesionCliente;

```

Dentro del método remoto ***guardarAutenticacionCliente()***, insertaremos los datos del cliente autenticado en el map ***sesionCliente***, y en el map clienteRepositorio, que es donde se guardan las asociaciones de los clientes con los repositorios. Devuelve un código de sesión de cliente.

A continuación, si todo ha sido correcto, se iniciará el servicio *Disco-Cliente*, realizando las siguientes operaciones:

1. Buscamos el número de repositorio correspondiente al cliente mediante el servicio Autenticación
2. Declaramos y creamos el objeto remoto
3. Iniciamos el servicio en el repositorio
4. Obtenemos dirección del servicio gestor, que es el que nos permitirá acceder al servicio Cliente-Operador y sus métodos remotos de subida y borrado de ficheros.
5. Hacemos un lookup de la clase remota que contiene los métodos para gestión de archivos

```
int idRepositorioCliente=ServicioAutenticacion.dameRepositorio(idSesionCliente);
System.out.println("Repositorio del cliente: "+ idRepositorioCliente);
ServicioDiscoClienteImpl discoCliente= new ServicioDiscoClienteImpl();
URL_DiscoCliente = "rmi://localhost:2000/DiscoCliente/"+idRepositorioCliente;
Naming.rebind(URL_DiscoCliente , discoCliente);
System.out.println("Levantado el servicio Disco-Cliente.");
String
direccionServicioClop=servicioGestor.obtenerServicioClienteOperador(idSesionCliente);
accesoMetodosClop=(ServicioClopOperadorInterface) Naming.lookup(direccionServicioClop);
```

Ilustración 29 - Creación y puesta en funcionamiento de servicio Disco-Cliente

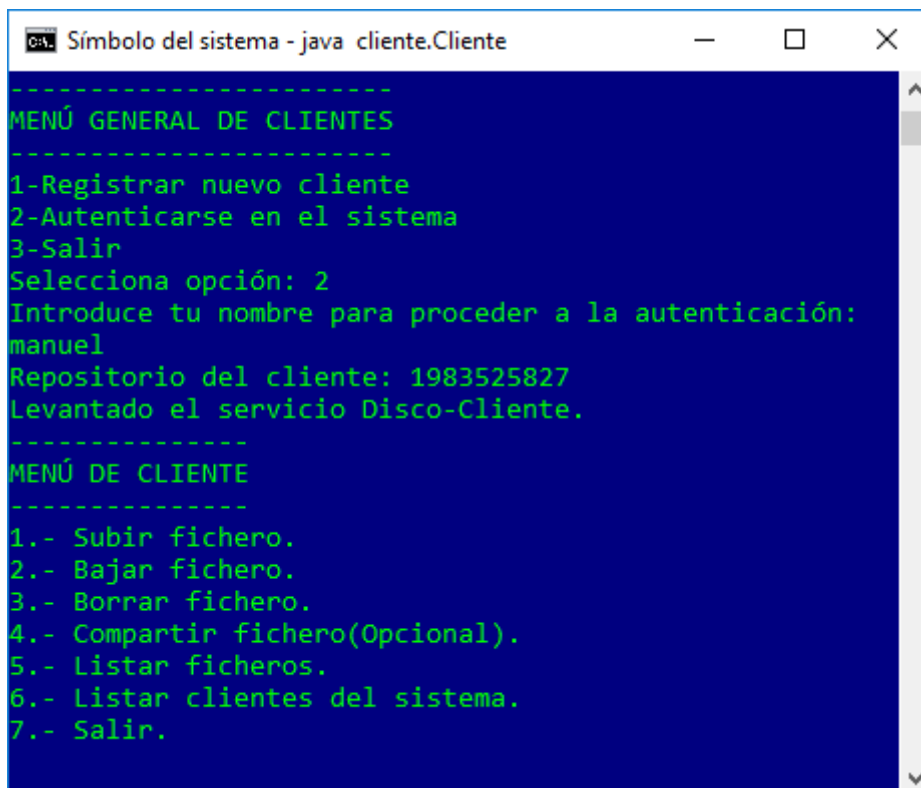


Ilustración 30 - Ejemplo de autenticación de cliente

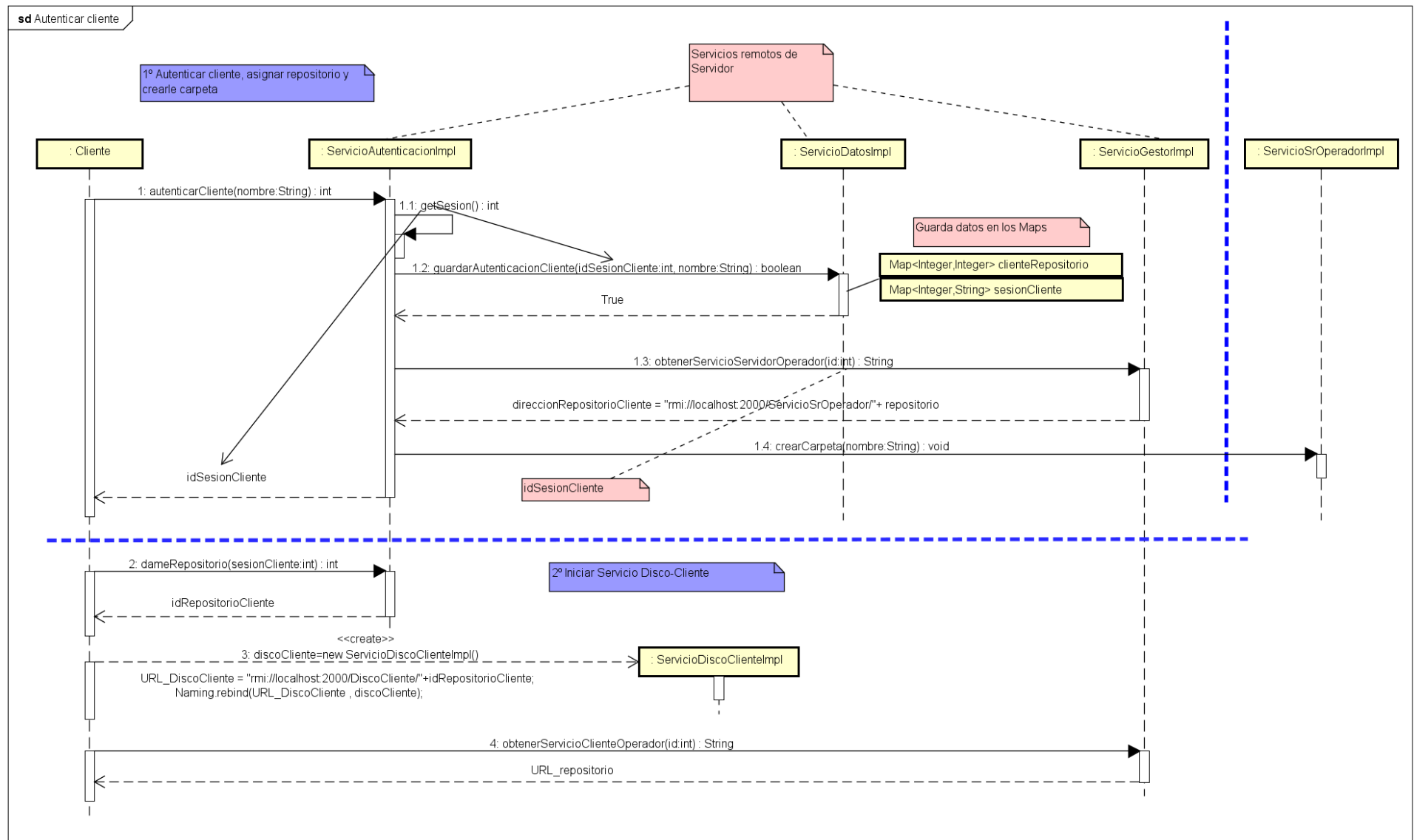


Ilustración 31 - Diagrama de secuencia Autenticar cliente.

Menú de cliente

Subir fichero

En este caso de uso, el cliente podrá subir un archivo a su carpeta de su repositorio. Cuando el cliente selecciona la opción *1- Subir fichero*, el sistema le pregunta la dirección de su disco local donde se encuentra el archivo, es decir, el directorio y este será recogido por la variable ***rutaDirectorio***. A continuación el sistema solicita el nombre completo del archivo, que será recogido por la variable ***archivo***.

```
System.out.println("Introduce la ruta donde se encuentra el archivo:")
rutaDirectorio=LecturaConsola.readLine();
System.out.println("Introduce el fichero a subir: ");
archivo=LecturaConsola.readLine();
```

Antes de hacer la petición a través del método remoto ***subirFichero()***, hemos de construir el objeto *fichero* mediante la clase serializable ***Fichero***, y dependiendo si el usuario ha proporcionado el directorio o no llamaríamos a un constructor de la clase u otro.

```
if (rutaDirectorio.isEmpty()){ //Si está vacía rutaDirectorio mandamos a un constructor
    Fichero fichero = new Fichero(archivo,nombre);
    if (!accesoMetodosCLOp.subirFichero(fichero))
        System.out.println("No existe el fichero.");
}else{// Si no está vacía rutaDirectorio mandamos al otro constructor
    Fichero fichero= new Fichero(rutaDirectorio,archivo, nombre);
```

El servicio encargado de subir el fichero es Servicio Cliente-Operador que es quien tiene el método remoto ***subirFichero()***. A través del objeto remoto accesoMetodosCLOp, cuya referenciación se ha realizado antes del menú secundario de cliente. Se envía una copia del fichero para que guarde ésta en la carpeta del cliente de su repositorio:

```
accesoMetodosCLOp.subirFichero(fichero)
```

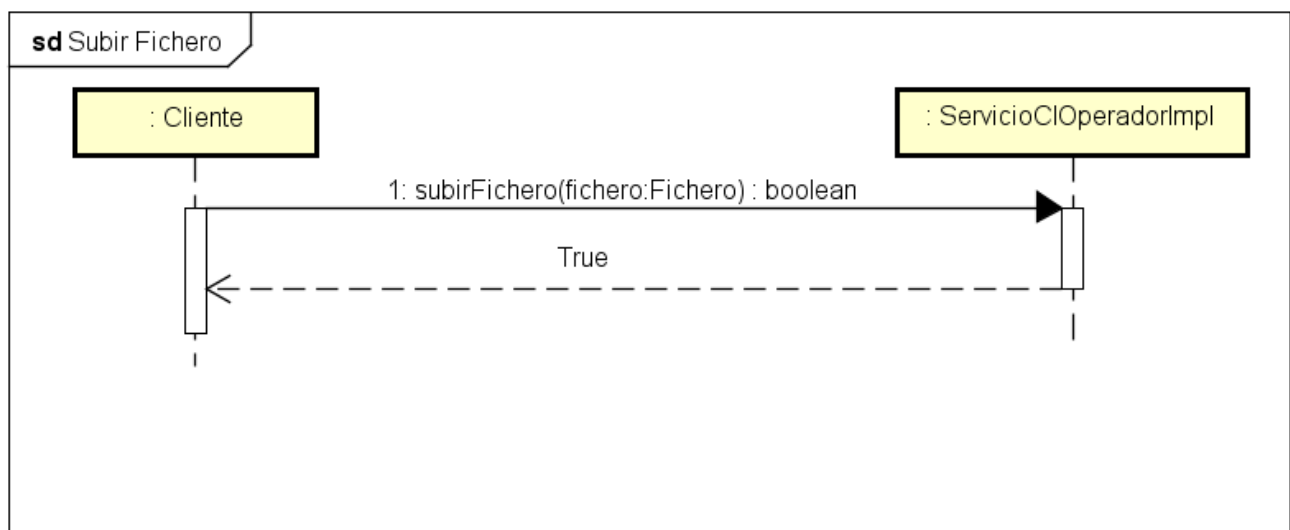


Ilustración 32 - Diagrama de secuencia Subir fichero

```

-----
MENÚ DE CLIENTE
-----
1.- Subir fichero.
2.- Bajar fichero.
3.- Borrar fichero.
4.- Compartir fichero(Opcional).
5.- Listar ficheros.
6.- Listar clientes del sistema.
7.- Salir.
1
Introduce la ruta donde se encuentra el archivo:
c:\users\manuel\desktop
Introduce el fichero a subir:
archivoejemplo.txt
Fichero subido correctamente.
-----
MENÚ DE CLIENTE
-----
1.- Subir fichero.
2.- Bajar fichero.
3.- Borrar fichero.
4.- Compartir fichero(Opcional).
5.- Listar ficheros.
6.- Listar clientes del sistema.
7.- Salir.

```

Ilustración 33 - Ejemplo en consola de subida de un fichero.

Bajar fichero

Aquí nos encontramos con que esta operación la realiza el servicio Servidor-Operador, y nosotros nos encontramos en el cliente, por lo tanto haremos uso del servicio Gestor para acceder al servidor, éste buscará el repositorio haciendo uso del servicio Datos, y posteriormente Gestor mandará el archivo a repositorio a través del servicio Servidor-Operador, y desde repositorio, se mandará al cliente usando el servicio Disco-cliente. Vamos a ver por pasos como se realiza esta operación:

Paso 1 - Cuando el cliente selecciona la opción 2- *Bajar fichero*, se solicita el nombre del fichero que se desea descargar al ordenador local del cliente; el nombre es recogido por la variable **archivo**.

```
System.out.println("Introduce el nombre del fichero a descargar: ");
archivo=LecturaConsola.readLine();
```

Paso 2 - Se manda mensaje al Servicio Gestor del servidor mediante método remoto `bajaraFicheroPaso1()`, mandándole el número de sesión del cliente, nombre del fichero y nombre del cliente.

```
servicioGestor.bajaraFicheroPaso1(idSesionCliente,archivo,nombre);
```

Paso 3 - Dentro de `bajaraFicheroPaso1()` llamamos a su método `obtenerServicioServidorOperador()` y le mandamos el número de sesión del cliente. Este método nos devolverá la dirección URL del repositorio y los métodos remotos implementados por Servidor-Operador.

```
String direccionRepositorio=obtenerServicioServidorOperador(SesionCliente);
```

Obtenemos la referencia al objeto *Datos* y buscamos el número de repositorio del cliente con su método remoto **buscarRepositorio()**.

```
datos = (ServicioDatosInterface) Naming.Lookup("rmi://localhost:2000/datos");
int repositorio = datos.buscaRepositorio(SesionCliente);
```

Paso 4 – Obtenemos la referencia al objeto Servidor-Operador de repositorio y llamamos a su método remoto **bajarFicheroPaso2()** pasándole el repositorio, archivo y nombre de cliente.

```
servicioSrOp=(ServicioSrOperadorInterface) Naming.Lookup(direccionRepositorio);
servicioSrOp.bajarFicheroPaso2(repositorio, archivo, nombreCliente);
```

Paso 5 – Construimos el directorio donde se encuentra el archivo (se incluye éste también) en el repositorio para mandárselo o descargárselo al cliente.

```
URL rutaca =
ServicioSrOperadorImpl.class.getProtectionDomain().getCodeSource().getLocation();
//Se desmembra la dirección y se mete en un array
String []rutaTemporal = rutaca.toString().split("/");
//Se vuelve a construir la dirección pero mas exacta y limpia.
StringBuilder rutaFinal = new StringBuilder();
//Aquí controlamos que se ejecute con .jar o sin .jar
if (Herramienta.comprobarSufijoCadena(rutaTemporal))
    tamanoRutaTemporal=rutaTemporal.length-1;
else
    tamanoRutaTemporal=rutaTemporal.length;

for(int i = 1; i < tamanoRutaTemporal; i++){
    rutaFinal.append(rutaTemporal[i]);
    rutaFinal.append("/");
}

//Se le añade el cliente para mandarla al objeto fichero.
String ruta = rutaFinal.toString() + nombreCliente;
Fichero fichero = new Fichero(ruta,archivo,nombreCliente);
```

Paso 6 – Obtenemos la referencia al objeto remoto *DiscoCliente* y llamamos a su método **bajarFicheroPaso3()** mandándole el objeto **fichero** construido.

```
discoCliente = (ServicioDiscoClienteInterface)
Naming.Lookup("rmi://localhost:2000/DiscoCliente/"+repositorioCliente);

discoCliente.bajarFicheroPaso3(fichero);
```

Este método finaliza la descarga del fichero en el ordenador local del cliente.

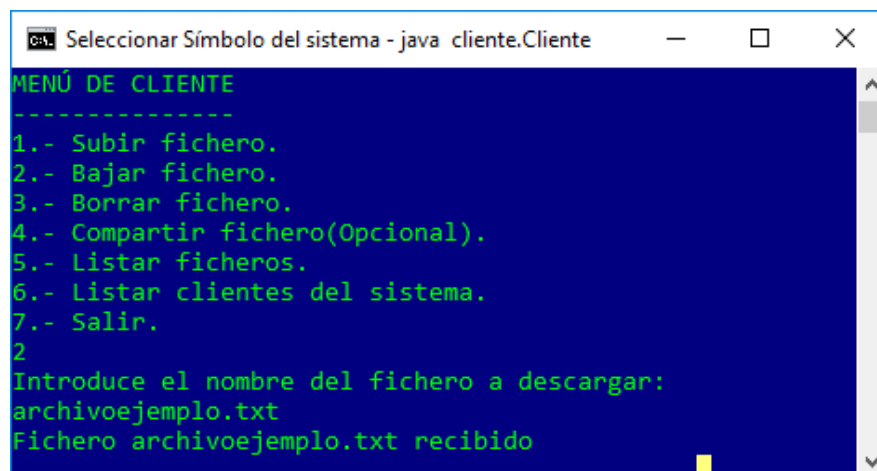
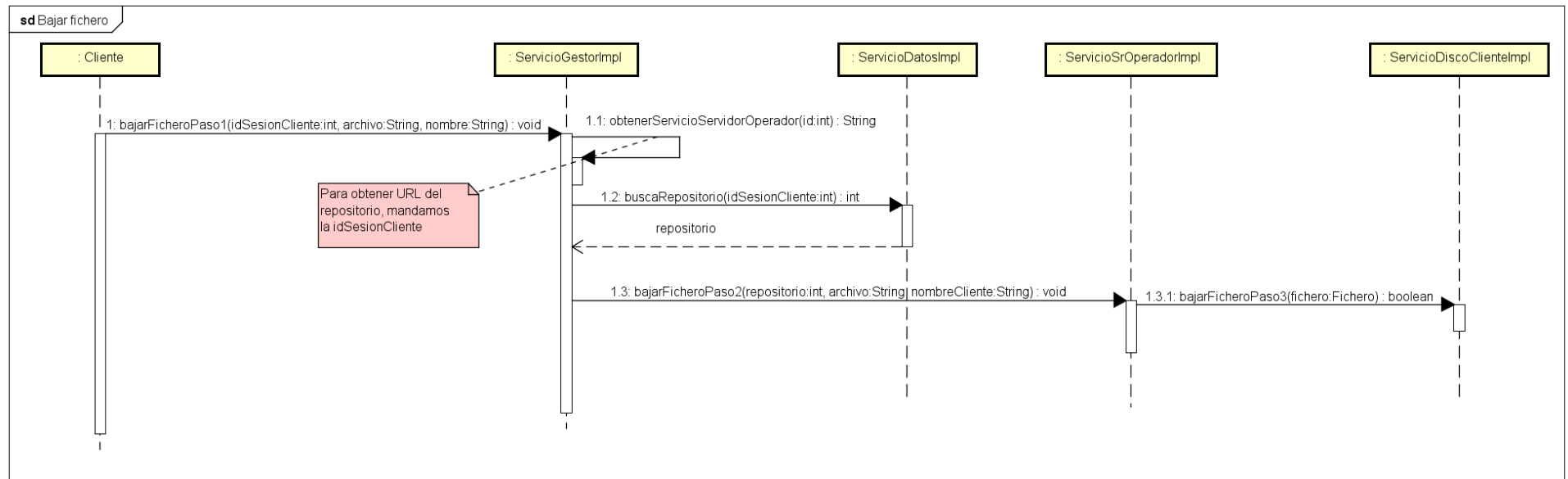


Ilustración 34 - Resultado de descargar fichero a ordenador local

*Ilustración 35 - Diagrama de secuencia Bajar Fichero*

Borrar fichero

Esta operación se encarga de realizarla el servicio Cliente-Operador de repositorio. Cuando el cliente selecciona la opción 3 – Borrar fichero, se solicita el nombre de fichero a borrar, y éste es recogido por la variable **archivo**. Posteriormente se hace la llamada remota al método borrarFichero() perteneciente al servicio Cliente-Operador enviándole el nombre de archivo y el nombre del cliente.

`accesoMetodosCLOp.borrarFichero(archivo,nombre)`

Construimos la ruta del directorio en el repositorio:

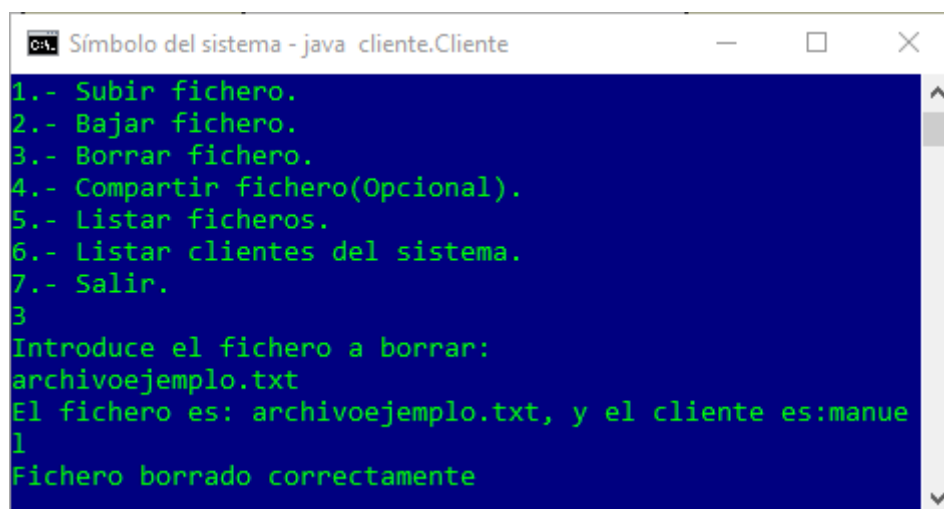
```
URL rutaca =
ServicioCLOperadorImpl.class.getProtectionDomain().getCodeSource().getLocation();

String []rutaTemporal = rutaca.toString().split("/");
StringBuilder rutaFinal = new StringBuilder();
if (Herramienta.comprobarSufijoCadena(rutaTemporal))
    tamanoRutaTemporal=rutaTemporal.length-1;
else
    tamanoRutaTemporal=rutaTemporal.length;

for(int i = 1; i < tamanoRutaTemporal; i++){
    rutaFinal.append(rutaTemporal[i]);
    rutaFinal.append("/");
}
```

Creamos el objeto fichero con la clase serializable, y lanzamos la orden de borrado de archivo con **fichero.delete()**.

```
File ruta = new File(rutaFinal.toString() + carpeta);
if (ruta.exists()) {
    String nombreFichero = ruta + "/" + archivo;
    File fichero = new File(nombreFichero);
    if(fichero.delete())
        return true;
}
```



The screenshot shows a Java IDE window titled 'Símbolo del sistema - java cliente.Cliente'. The console output is as follows:

```
1.- Subir fichero.
2.- Bajar fichero.
3.- Borrar fichero.
4.- Compartir fichero(Opcional).
5.- Listar ficheros.
6.- Listar clientes del sistema.
7.- Salir.
3
Introduce el fichero a borrar:
archivoejemplo.txt
El fichero es: archivoejemplo.txt, y el cliente es:manue
l
Fichero borrado correctamente
```

Ilustración 36 - Ejecución en consola de borrado de un fichero

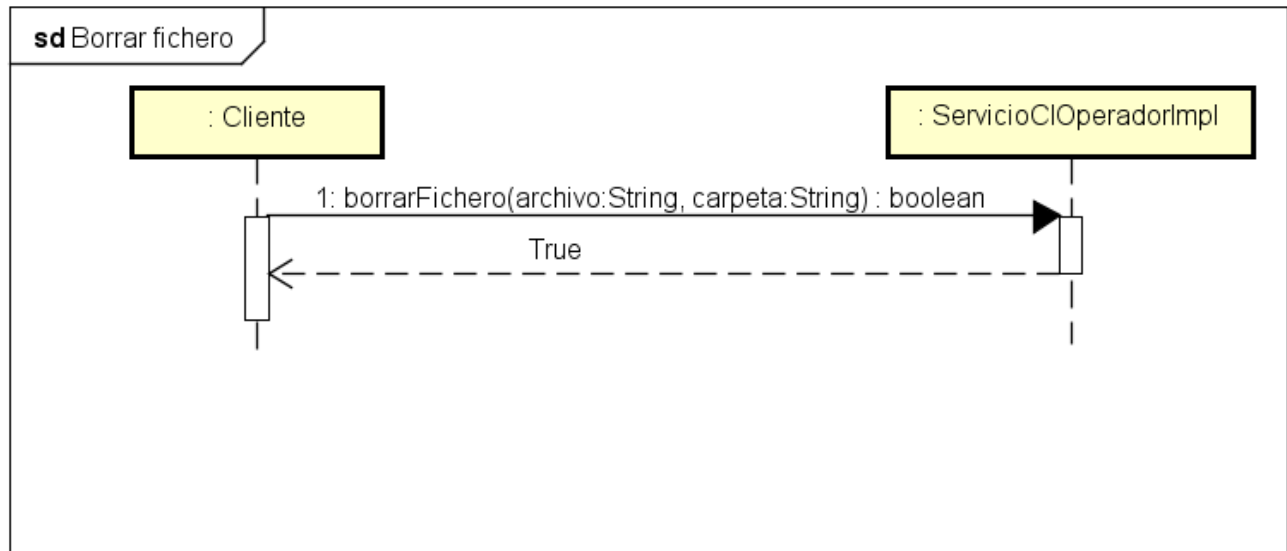


Ilustración 37 - Caso de uso "Borrar Fichero" representado por su diagrama de secuencia.

Listar ficheros

Este listado se va a encargar de proporcionarlo el servicio Cliente-Operador del repositorio. Al seleccionar el usuario la opción 5 – *Listar ficheros de cliente*, el objeto cliente crea un **arrayList** llamado **archivosExtraidos**, donde se almacenarán los ficheros del cliente encontrados.

```
ArrayList<String> archivosExtraidos=new ArrayList<String>();
```

Se hace la llamada al método remoto listarFicherosCliente() de la clase ServicioCLOperadorImpl perteneciente a Repositorio. Se le manda el nombre del cliente.

```
archivosExtraidos=accesoMetodosCLOp.listarFicherosCliente(nombre);
```

Dentro del método remoto, construimos la ruta del directorio del cliente en el repositorio donde se encuentran los archivos, se buscan en la carpeta del cliente y se agregan a un **arrayList** llamado **listaArchivos**, que es el que le vamos a devolver al cliente.

```

ArrayList<String> listaArchivos =new ArrayList<String>();
URL rutaca =
ServicioCLOperadorImpl.class.getProtectionDomain().getCodeSource().getLocation();
String []rutaTemporal = rutaca.toString().split("/");
StringBuilder rutaFinal = new StringBuilder();
for(int i = 1; i < tamanoRutaTemporal; i++){
    rutaFinal.append(rutaTemporal[i]);
    rutaFinal.append("/");
}
File ruta = new File(rutaFinal.toString() + nombreCarpeta);
//Agregamos al arrayList los ficheros.
File[] ficheros = ruta.listFiles();
for (int x=0;x<ficheros.length;x++){
    listaArchivos.add(ficheros[x].getName());
}
// Devolvemos al cliente un arraylist con los archivos
return listaArchivos;
}
  
```

Ilustración 38 - Obtención de la lista de ficheros de cliente.

```

-----
MENÚ DE CLIENTE
-----
1.- Subir fichero.
2.- Bajar fichero.
3.- Borrar fichero.
4.- Compartir fichero(Opcional).
5.- Listar ficheros.
6.- Listar clientes del sistema.
7.- Salir.
Selecciona opción: 5
Listando ficheros de la carpeta del cliente manuel:
Algoritmo Salto del caballo.docx
animation.gif
archivoejemplo.txt
AX7 Introduction for new developers.pptx
lambdaCalculo.pdf
Perspectiva histórica IA.pdf
The New Ax (Dynamics AX7).pptx
-----
MENÚ DE CLIENTE
-----

```

Ilustración 39 - Resultado de Listar ficheros del cliente

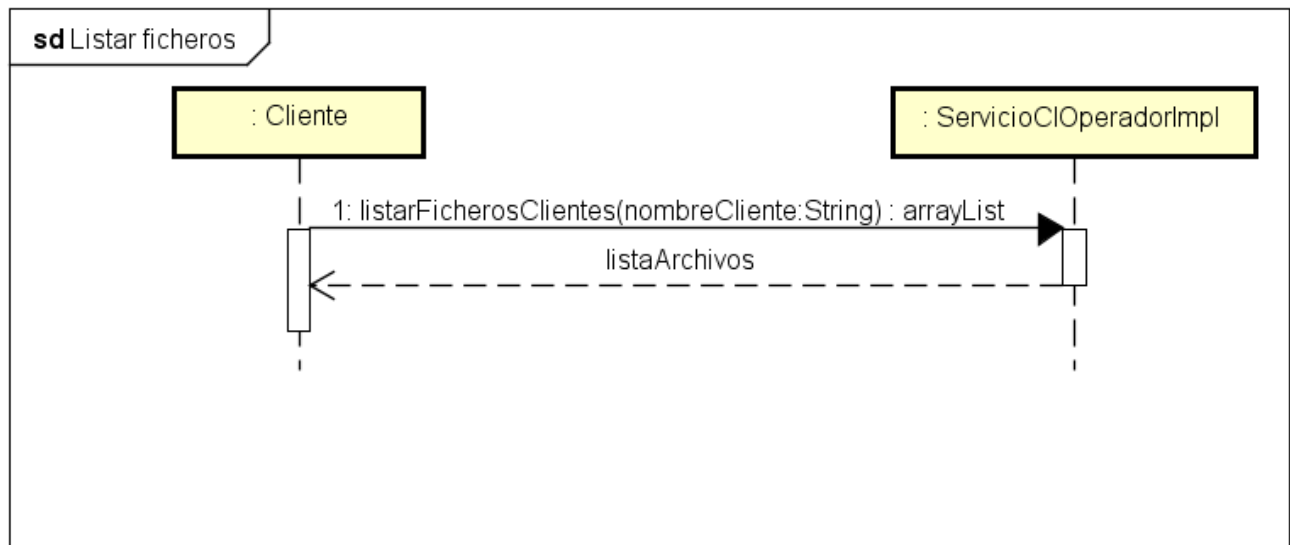


Ilustración 40 - Caso de uso listar ficheros representado por su diagrama de secuencia.

Listar clientes del sistema

En este caso de uso se pretende que un cliente pueda ver que otros clientes están en su mismo repositorio. Aquí necesitaremos acceder al servicio Datos del servidor, que es el que nos puede proporcionar el listado de clientes; para ello haremos uso del servicio Gestor, y su método remoto listarClientesSistema().

```
ArrayList<String> listadoClientes=new ArrayList<String>();

listadoClientes=servicioGestor.listarClientesSistema();
```

La lista de los clientes será recibida por listadoClientes, que es un arrayList donde se almacenará dicha lista.

Una vez en el servicio Gestor, el método listarClientesSistema() llamará al método remoto del servicio Datos: listaClientesAutenticados() (recordemos que solo los servicios del servidor pueden acceder al servicio Datos). Recibiremos un arrayList que se almacenará en listaClientes, y este será el que mandemos al objeto Cliente.

```
ArrayList<String> listaClientes=new ArrayList<String>();
listaClientes=datos.listaClientesAutenticados();
return listaClientes;
```

```
public ArrayList<String> listaClientesAutenticados() throws RemoteException {
    ArrayList<String> clientesA = new ArrayList<String>(sesionCliente.values());
    return clientesA;
}
```

Ilustración 41 - Método del servicio Datos para proporcionar los clientes del sistema.

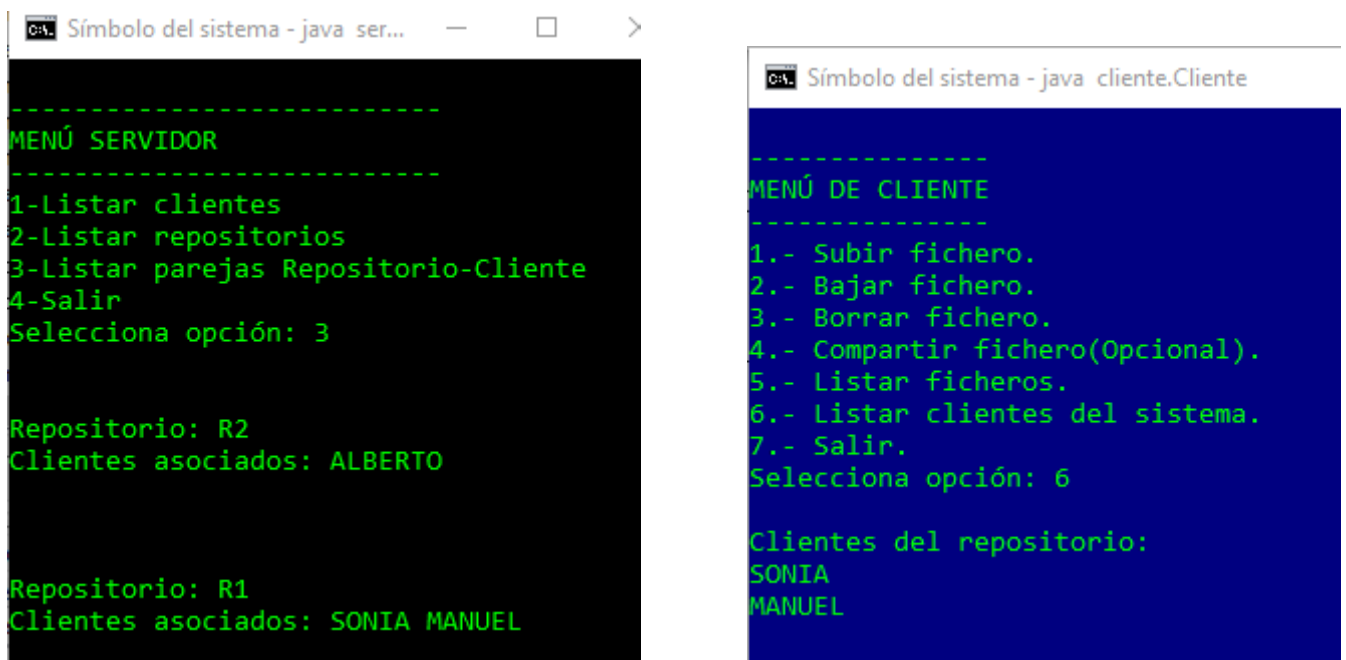


Ilustración 42 - Resultados de listar clientes de repositorio

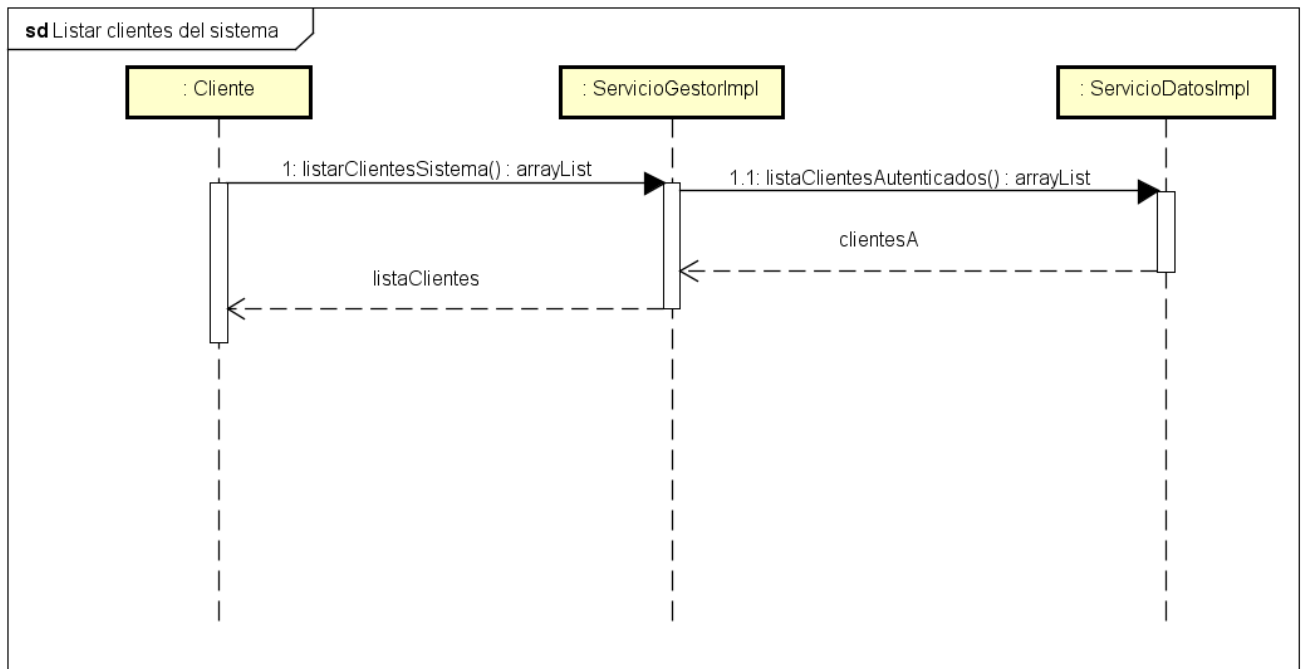
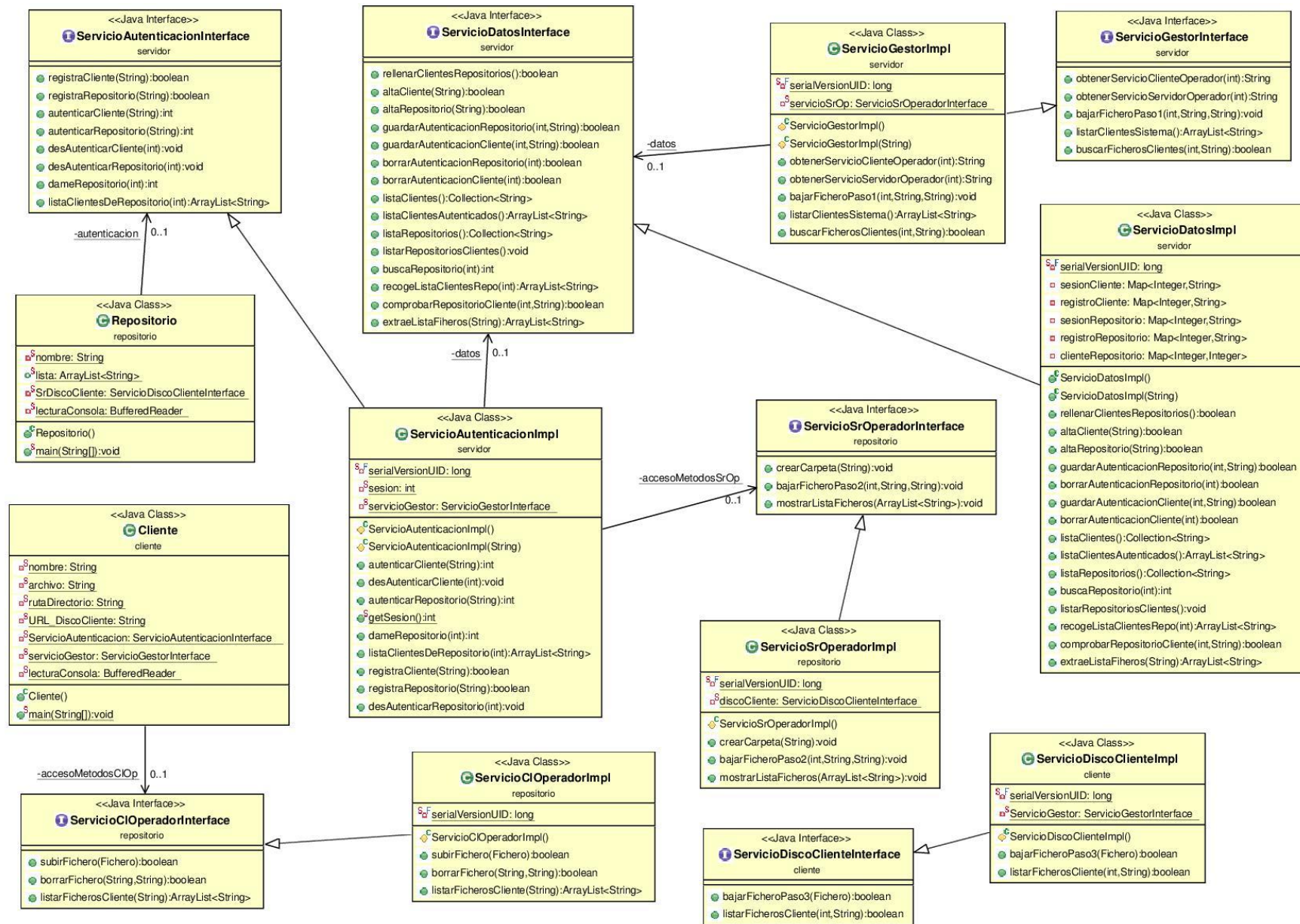


Ilustración 43 - Diagrama de secuencia para listar clientes de un repositorio

4. Diagrama de clases

En la página siguiente se puede observar el diagrama de clases obtenido una vez finalizada la arquitectura del sistema.



5.Conclusiones

La arquitectura del sistema ha costado entenderla, pero si el objetivo era aprender cómo funciona un sistema RMI, podemos darnos por satisfechos, ya que el objetivo está cumplido. Este trabajo práctico ha resultado muy didáctico, y aunque ha causado un gran esfuerzo en su desarrollo, ha merecido la pena por todo lo aprendido.

En definitiva, y como opinión personal, un gran trabajo que me ha permitido aprender más, y aplicar también mucho del conocimiento adquirido en otras asignaturas ya cursadas.

Gracias.