Universidad Complutense

Facultad de Informática

eGorilla

Diseño del Sistema

Asignatura: ***Ingeniería del Software***

Curso Académico: ***2008/2009***

Grupo: ***4º B***

Índice

1. Introducción

En este documento se procede a la descripción del diseño del sistema, una vez revisada casi por completo la etapa de análisis (siempre habrá algo por revisar correspondiente a ésta etapa) siguiendo el orden establecido por el estándar del Proceso Unificado de Desarrollo ó RUP.

El contenido del presente documento se puede clasificar en los siguientes conceptos:

1.1. Contratos de operaciones

Se analizan y describen las **operaciones más críticas del sistema**.

1.2. Casos de uso reales

Se analizan y describen las **acciones reales producidas por el sistema mediante diagramas de casos de uso** en los que se refleja precisamente dicho comportamiento real

1.3. Diagramas del diseño

Se describen los diagramas correspondientes al diseño, principalmente diagramas de clases, de **actividad ó secuencia** y de **estados** principalmente.

1.4. Diagrama de Clases de Comunicación entre Módulos

En éste diagrama se ilustran la clase ó clases dentro de cada módulo que son las encargadas de la **comunicación** con el resto de los módulos de la aplicación.

2. Contratos de operaciones

2.1. Introducción

Los contratos de operaciones contribuyen a definir el comportamiento de un sistema en términos de los cambios de estado que éste experimenta cuando se invoca una operación. Éste no profundiza en detalles de funcionalidad asociadas a las operaciones invocadas sino que interpreta al sistema como una caja negra.

Éste documento estará redactado declarativamente expresándose a partir de los cambios de estado de las precondiciones y las poscondiciones.

Se describirán las operaciones de mayor relevancia para la funcionalidad principal de la aplicación, éstas estarán condicionadas por el diseño del modelo del dominio y los diagramas de secuencia del sistema.

2.2. Cliente

2.2.1. Contrato para caso de uso conectarServidor

Operación	conectarServidor ()
Responsabilidades	Establece la conexión con el servidor, se identifica en él e informa de los archivos compartidos.
Tipo	Sistema
Referencias Cruzadas	Casos de uso: BuscarArchivos, IniciarDescarga, DesconectarServidor.
Excepciones	Si la versión del cliente no es válido, indicar que hay un error de versión.
Salida	Información al usuario mediante mensaje de información.
Precondiciones	No existe una conexión activa con ningún servidor.
Postcondiciones	• Fue creada una instancia de ServidorCliente. • Fue creada una cola de peticiones de InicioDeDescarga. • Se inicializó estadísticas de sesión. • La conexión con el servidor quedó establecida.

2.2.2. Contrato para caso de uso buscarArchivos

Operación	buscarArchivos (nombre, tipo)
Responsabilidades	Realiza una búsqueda de los archivos que están en la red que cumplan los requisitos de los parámetros.
Tipo	Sistema
Referencias Cruzadas	
Excepciones	Si no tiene establecida una conexión con el servidor se notificará al usuario con un error.
Salida	Información al usuario mediante una tabla con los archivos resultantes de la búsqueda.
Precondiciones	Se ha establecido conexión con el servidor.
Postcondiciones	Se creó una lista de archivos coincidentes a los parámetros buscados.

2.2.3. Contrato para caso de uso iniciarDescarga

Operación	inciarDescarga(identificador)
Responsabilidades	Realiza una serie de peticiones para descargar el archivo solicitado.
Tipo	Sistema
Referencias Cruzadas	
Excepciones	Archivo erróneo, Muestra un mensaje de error.
Salida	Información al usuario mediante una tabla con los archivos resultantes de la búsqueda.
Precondiciones	Se ha establecido conexión con el servidor y además posee el identificador único del archivo a descargar.
Postcondiciones	• Se obtuvo una lista de clientes poseedores del archivo deseado. • Se creó una lista de peticiones de las partes del archivo deseado.

2.2.4. Contrato para caso de uso consultarEstadísticas

Operación	consultarEstadísticas(parametros)
Responsabilidades	Realiza una petición para obtener las estadísticas conforme a la información solicitada por el usuario.
Tipo	Sistema
Referencias Cruzadas	
Excepciones	No existen datos estadísticos, Muestra un mensaje de error.
Salida	Información al usuario mediante una gráfica (de diferente tipo: barras verticales, horizontales, pie) con los datos obtenidos.
Precondiciones	Existencia de datos para mostrar.
Postcondiciones	Se obtuvo una gráfica de con la información solicitada por el usuario.

2.3. Servidor

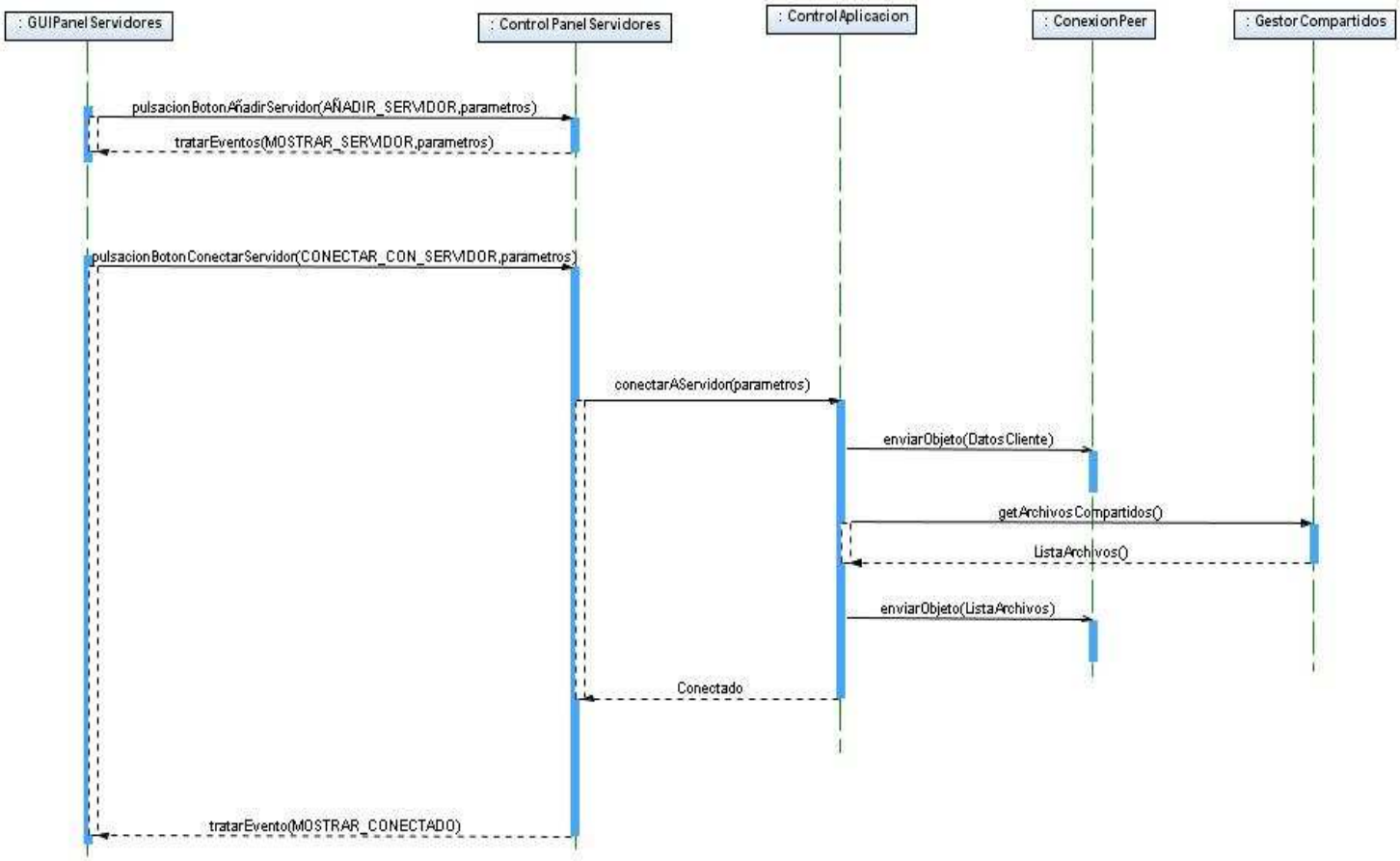
2.3.1. Contrato para caso de uso IniciarServidor

Operación	inciarServidor()
Responsabilidades	Iniciar el sistema Servidor de la aplicación.
Tipo	Sistema
Referencias Cruzadas	
Excepciones	Puerto de escucha ocupado, Muestra un mensaje de error.
Salida	Información al usuario mediante una tabla con los archivos resultantes de la búsqueda.
Precondiciones	No puede haber más de un servidor eGorilla en una misma máquina ejecutándose a la vez.
Postcondiciones	• Se crea una instancia de Servidor. • Se crea una instancia de ArchivosDisponibles. • Se crea una instancia de UsuariosConectados.

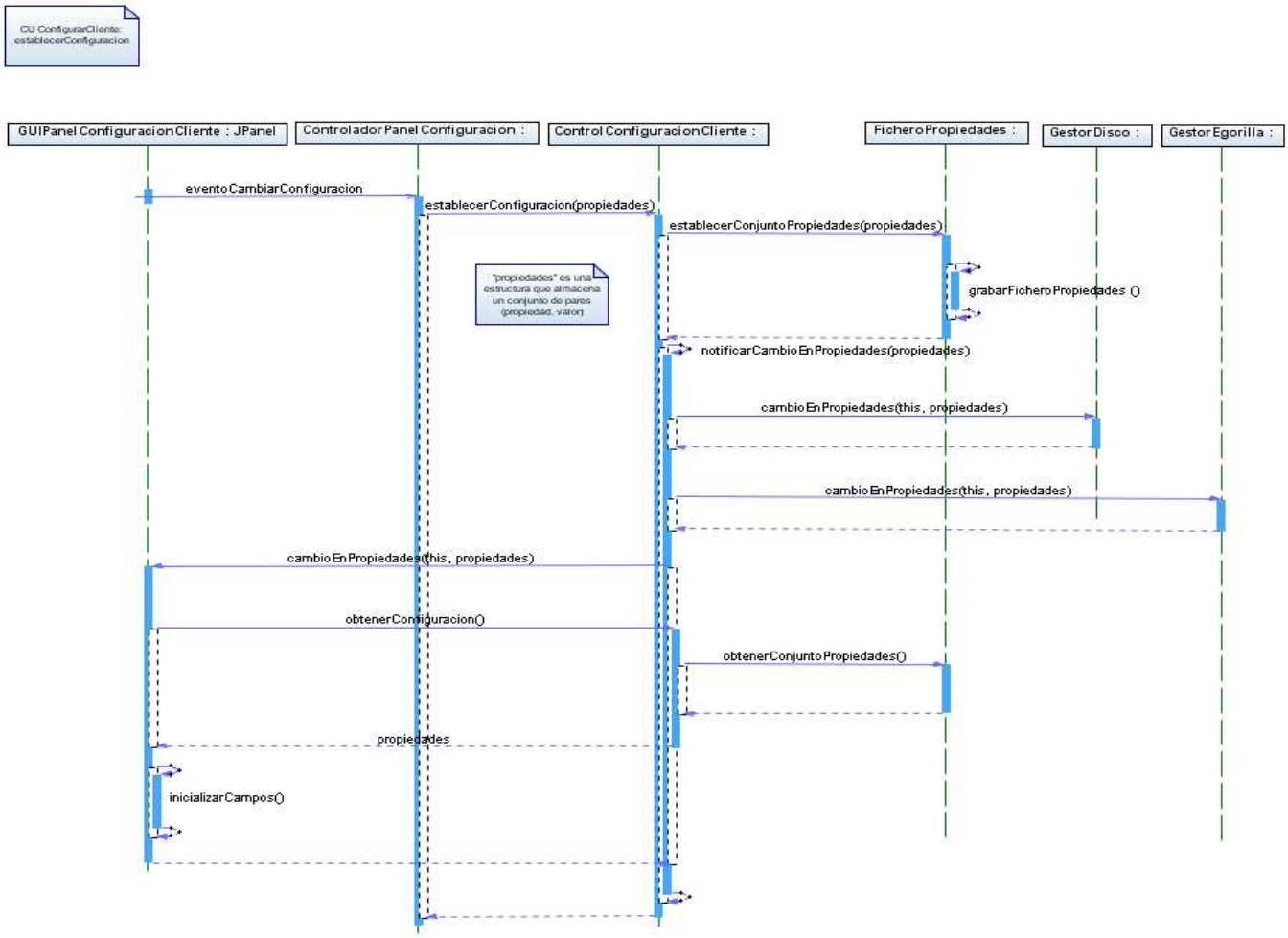
3. Casos de uso reales

3.1. Cliente

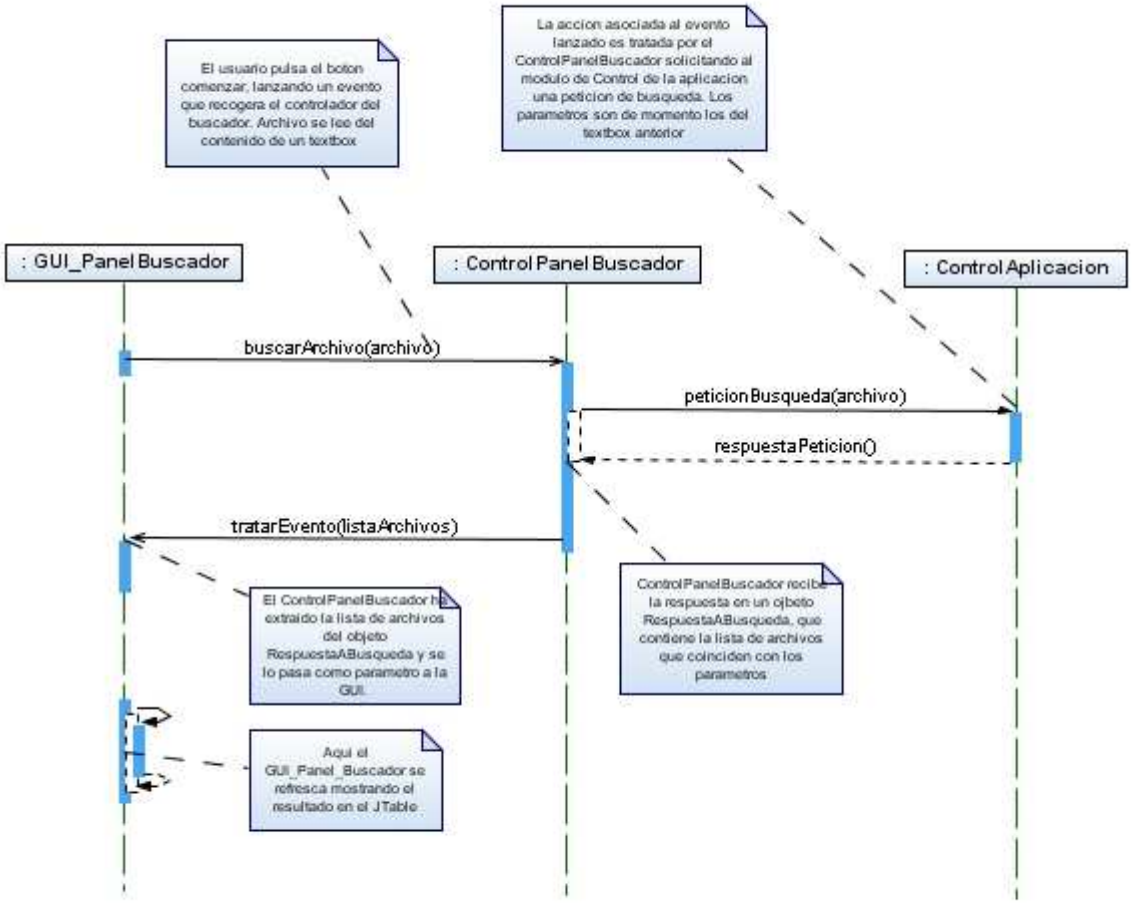
3.1.1. Servidores



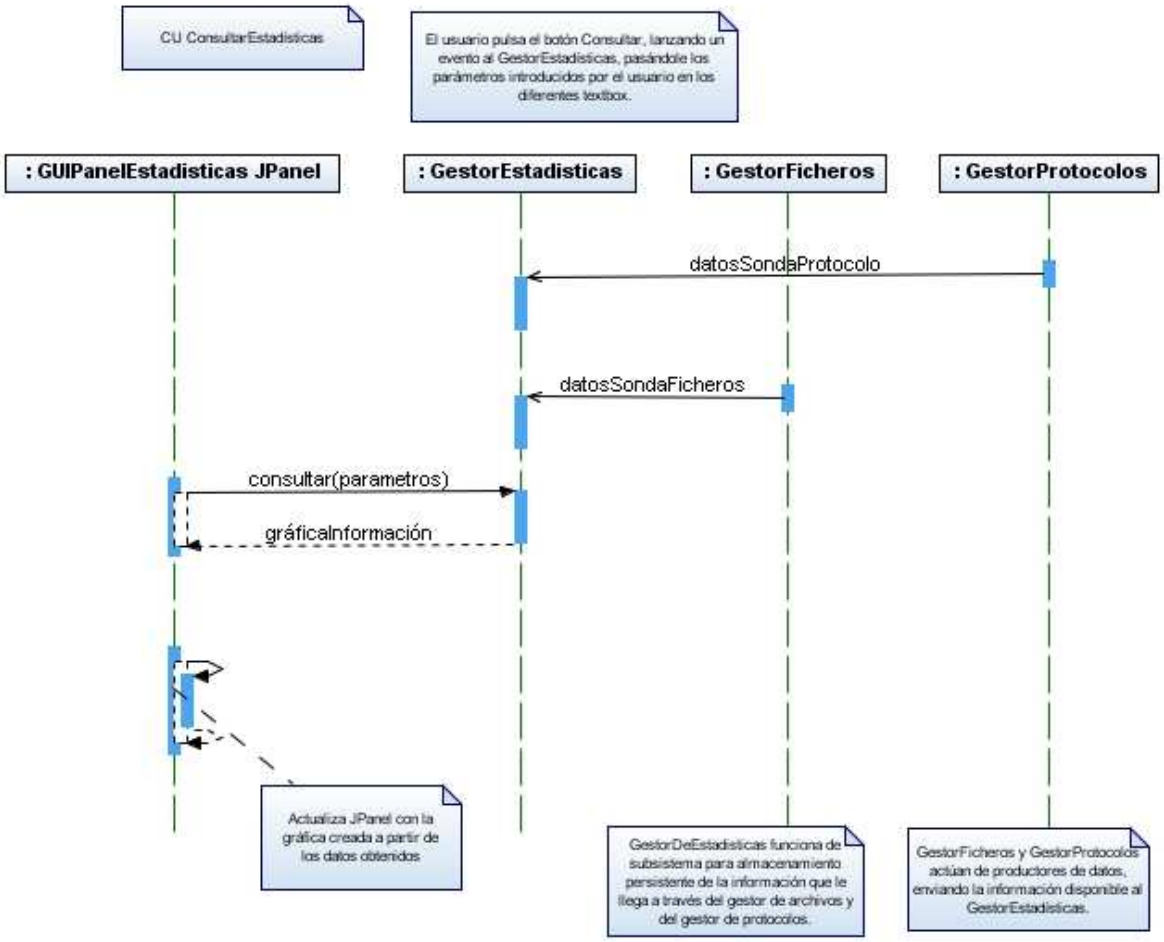
3.1.2. ConfigurarCliente



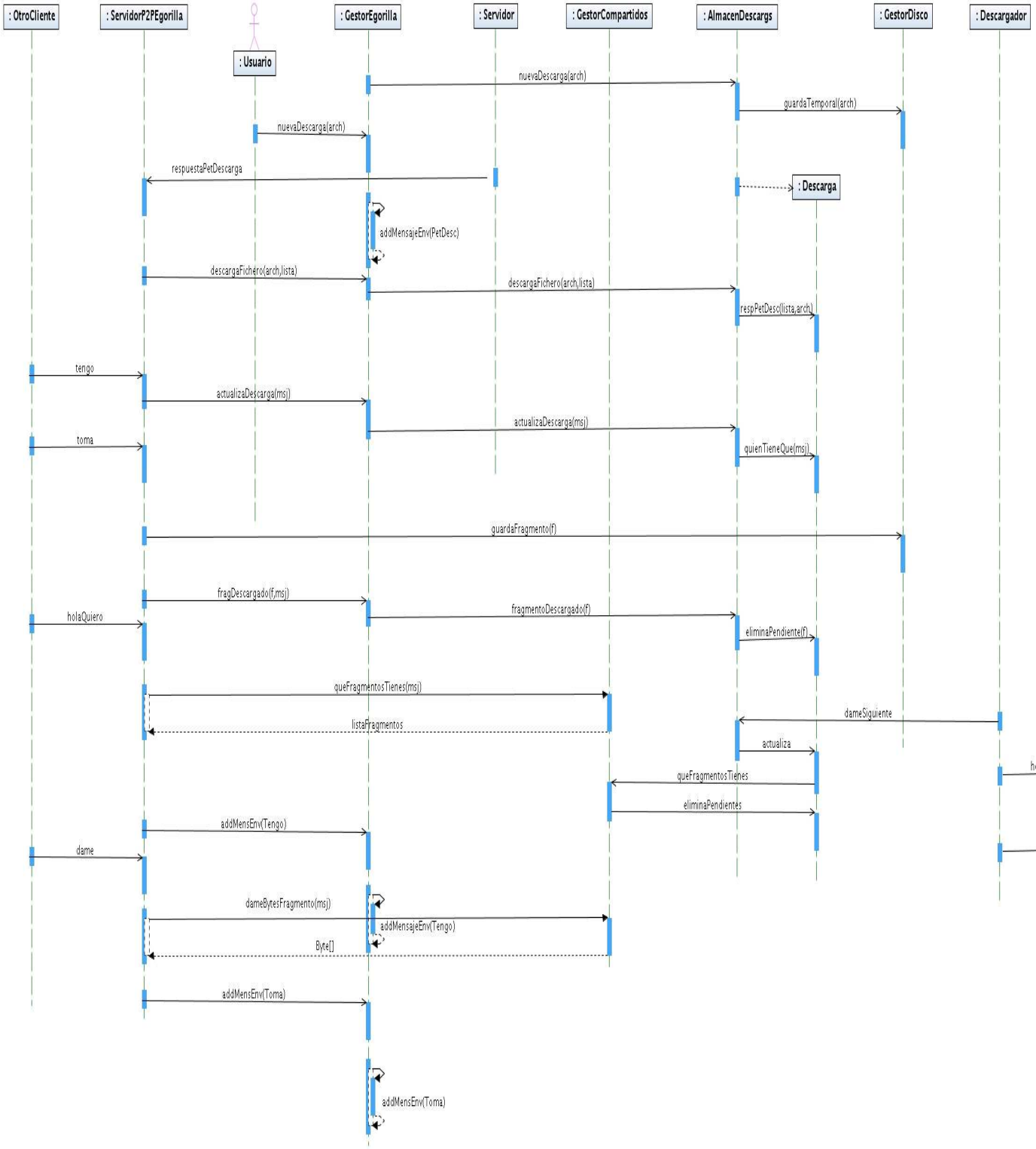
3.1.3. BuscarArchivos



3.1.4. ConsultarEstadísticas

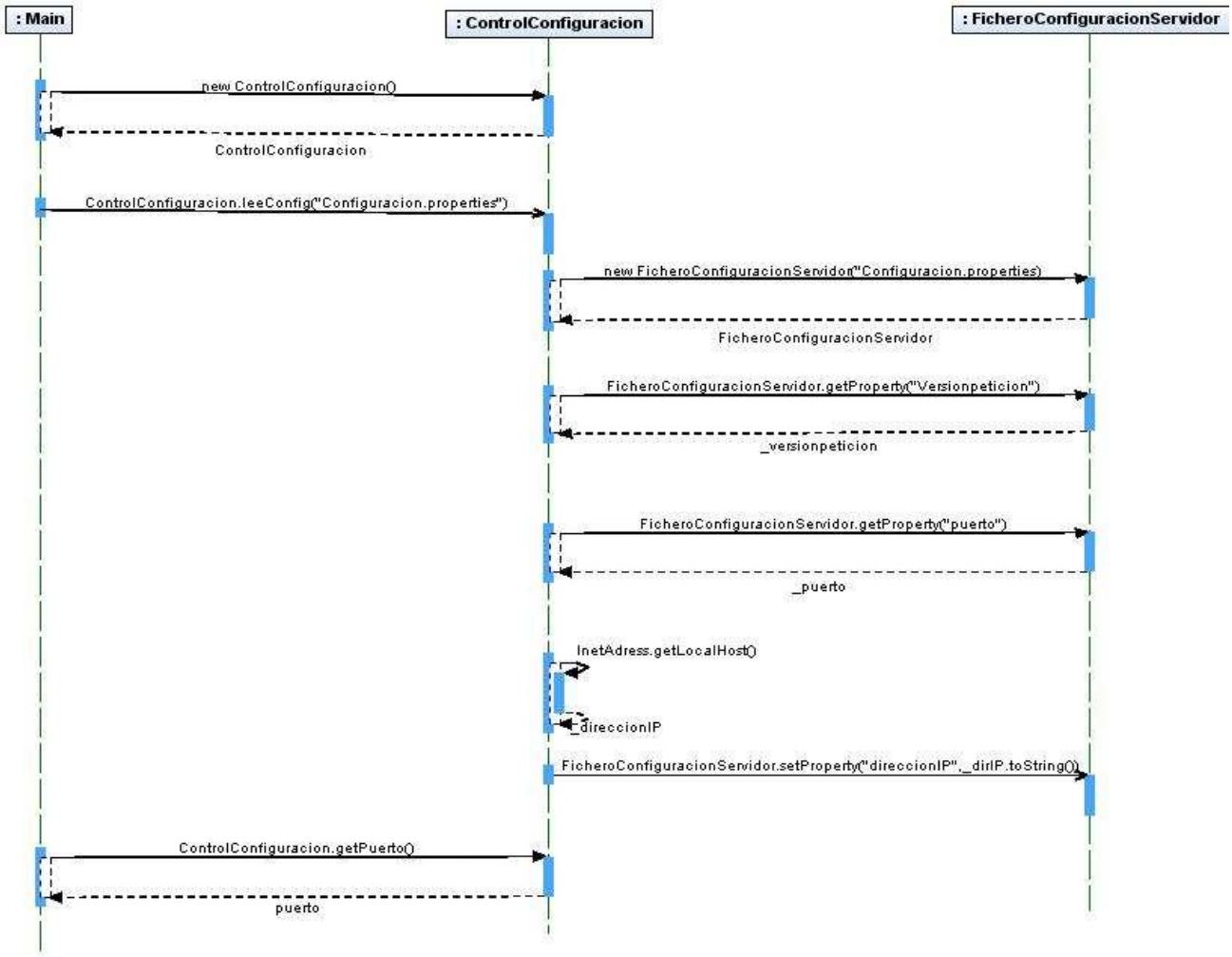


3.1.5. P2P eGorilla



3.2. Servidor

3.2.1. GestorDeConfiguración

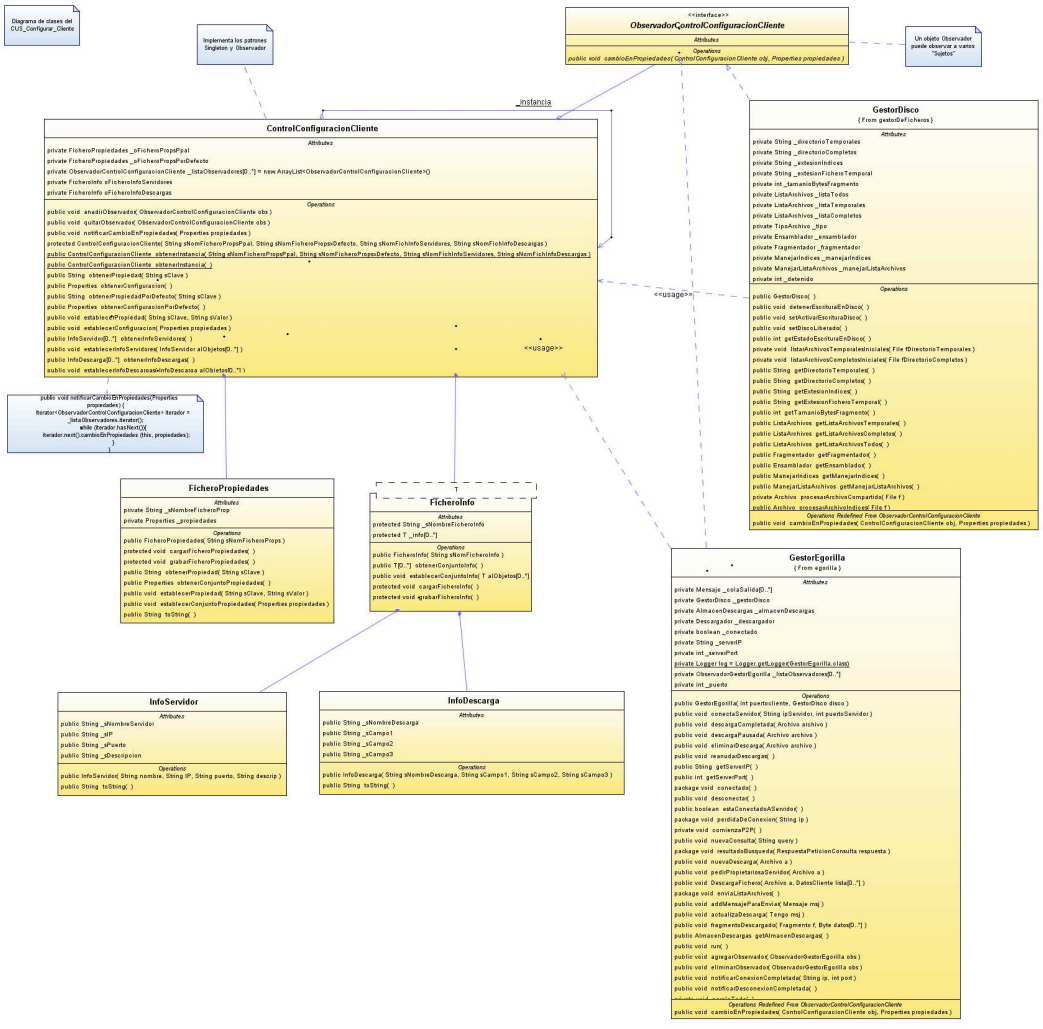


4. Diagramas de diseño

4.1. Cliente

4.1.1. gestorDeConfiguracion

Para el diseño del gestor de configuración del cliente se han empleado 2 patrones bastante habituales: singleton y observador. El primero encaja en el diseño porque se plantea la necesidad de que el gestor de configuración sea un objeto global, accesible desde cualquier parte, pero del que solamente deba existir una única instancia. El segundo resulta apropiado porque los cambios en el gestor de configuración deben propagarse a otros objetos dependientes de él, para que se actualicen convenientemente.



4.1.2. gestorDeErrores

4.1.3. gestorDeFicheros

El gestor de disco puede verse como una "fachada" que concentra ciertas partes comunes a las clases internas y a las externas, pero sin necesidad de notificar dichos cambios debido al uso de las mismas referencias.

En principio se pensó en utilizar el patrón *Fábrica Abstracta* (*Factory Method*) para poder instanciar otro tipo de clases sin depender de estas mismas, permitiendo después reutilizarla. Pero finalmente como las operaciones comunes para los distintos tipos de clases eran mínimas y sólo había dos tipo de clases a instanciar, aparte un ensamblar o fragmentar archivos no había muchas mas posibilidades, se ha optado por no incluirlo en el diseño.

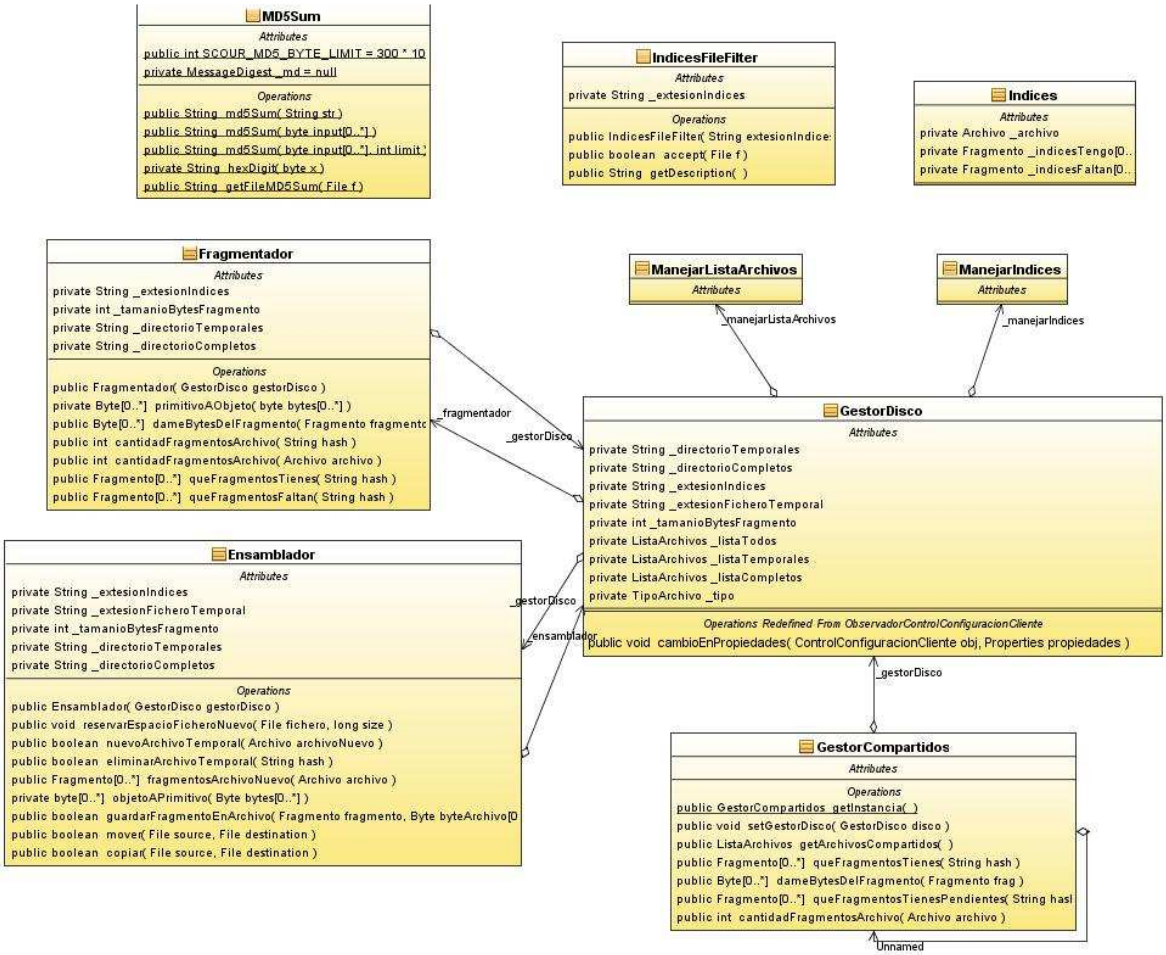


Ilustración – Diagrama de clases para el *GestorDeFicheros*.

También se pensó en el uso del patrón *Decorador* debido a la necesidad de añadir funcionalidad al ensamblador y fragmentador de forma dinámica, para aumentar la flexibilidad que la herencia proporciona en Java. La motivación de esto fue debido a que se podrían necesitar un ensamblador o un fragmentador que pudiera manejar diferentes tipos de índices o listas de archivos, lo que hubiera supuesto crear un *EnsambladorIndicesA, EnsambladorIndicesAListasArchivosA*, *EnsambladorIndicesBListasArchivosA*, *etcétera* y lo mismo con el *Fragmentador*, teniendo un motón de clases para satisfacer todas estas combinaciones. Pero como finalmente se decidió en utilizar un sólo tipo de índices y la mismas listas de archivos no se ha incluido tampoco en el diseño.

A continuación se muestran los diagramas de secuencia que ilustran algunas de las posibilidades de uso de esta parte.

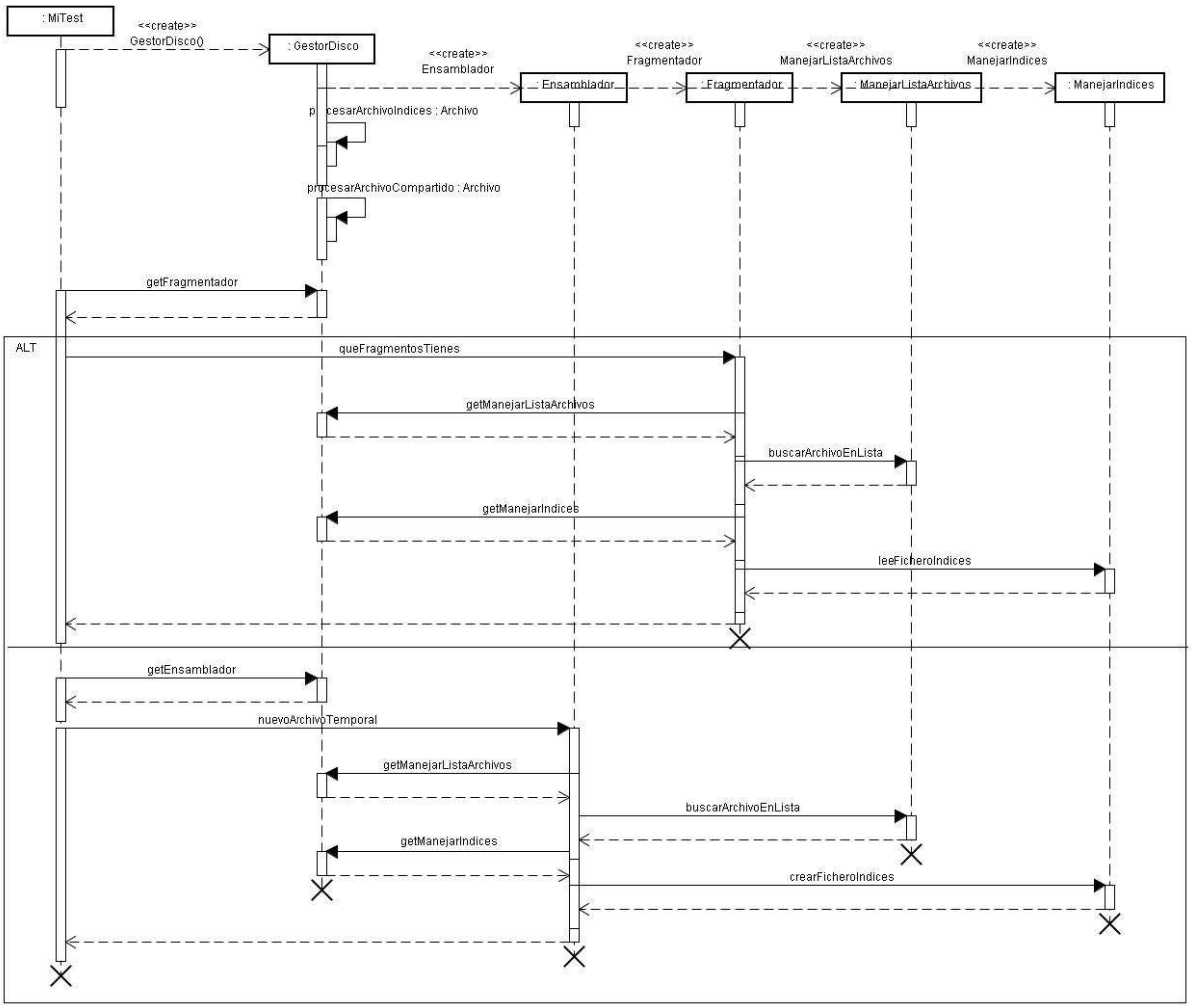


Ilustración – Diagrama de secuencia para el GestorDeFicheros.

4.1.4. gestorDeEstadisticas

4.1.5. main

4.1.6. gui

4.1.6.1. consola

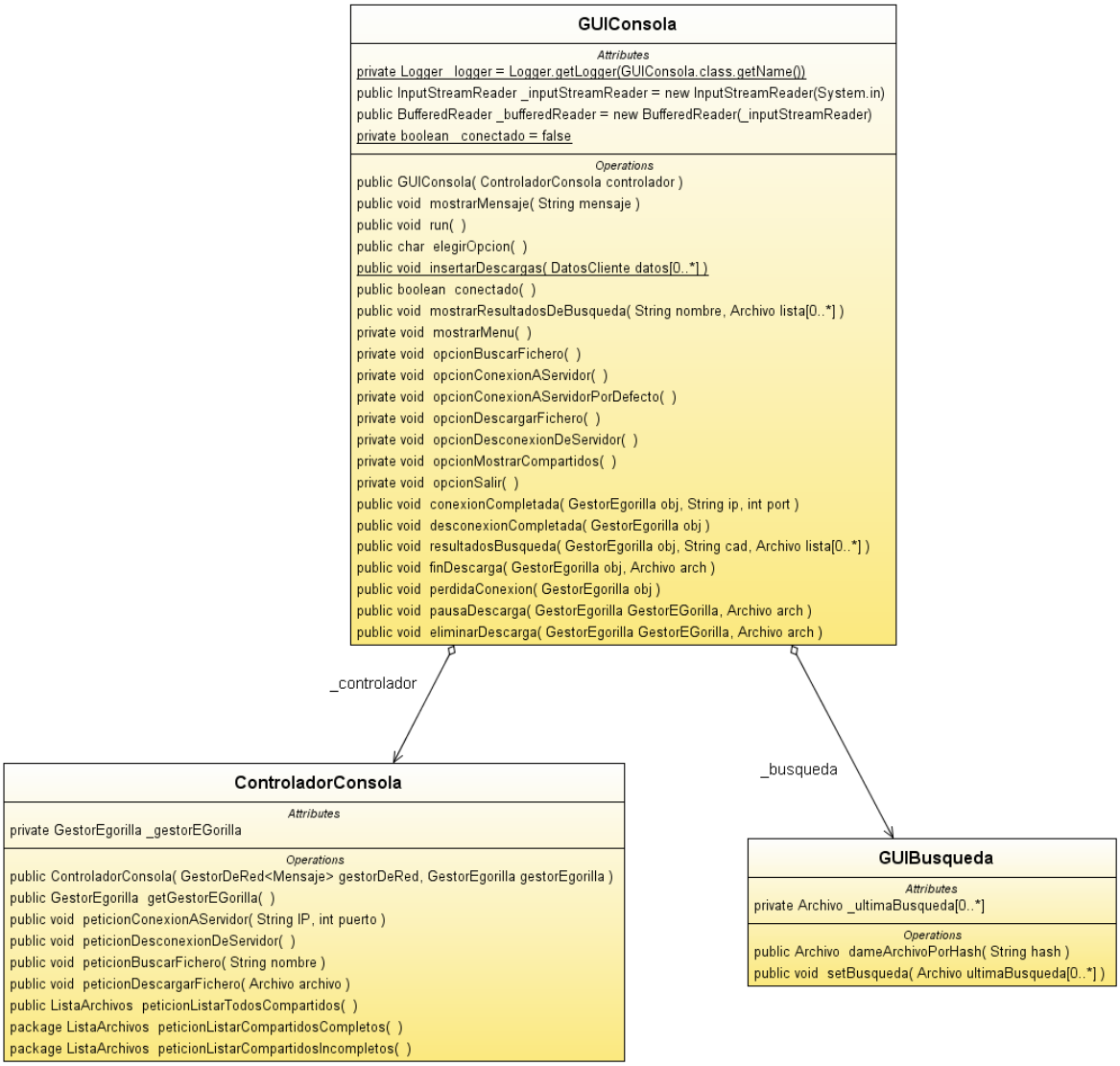


Ilustración – Diagrama de clases para la *GUI de consola*.

La interfaz de la aplicación del cliente eGorilla en modo consola esta compuesta por tres clases: **GUIConsola**, **ControladorConsola** y **GUIBusqueda**. Sigue el patrón modelo-vista-controlador.

GUIConsola es la vista del patrón dado que es la clase encargada de la interacción con el usuario final. Muestra los diferentes menús por los que va navegando el usuario. **GUIConsola** es un observador de GestorEgorilla ya que GestorEgorilla es el modelo de la aplicación. Para ello implementa la interfaz ObservadorGestorEgorilla y se registra en GestorEgorilla para mostrar los cambios que se producen del modelo.

ControladorConsola es el controlador del patrón dado que es el responsable de recibir los eventos de entrada desde la vista (**GUIConsola**). Hace uso de la funcionalidad que le ofrece gestorEGorilla.

GUIBusqueda es la clase encargada de la comunicación con el Servidor de eGorilla. Esta clase es utilizada por **GUIConsola** para realizar busquedas y solicitud de archivos al servidor.

4.1.6.2. grafica

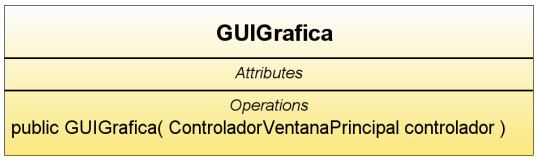


Ilustración – Diagrama de clases de GUIGrafica.

GUIBusqueda es la clase que se encarga de gestionar la gui de la aplicación en modo gráfico. El constructor de la clase recibe como parámetro el controlador de la ventana principal.

Dentro de la clase **GUIGrafica** se crea una instancia de la clase **GUIVentanaPrincipal** pasándole el controlador de la ventana principal a su constructor.

4.1.6.2.1. principal

Ilustración – Diagrama de clases de GUIVentanaPrincipal.

GUIVentanaPrincipal es la clase que gestiona la ventana principal de la aplicación. Esta clase contiene a todos los paneles de la interfaz del cliente.

La clase que controla la **GUIVentanaPrincipal** es **ControladorVentanaPrincipal**. Todos los eventos producidos en la **VentanaPrincipal** son recogidos por su controlador.

Dado que se está utilizando el patrón **MVC** la vista del patrón (**GUIVentanaPrincipal**) implementa la interfaz **ObservadorGestorEgorilla** para registrarse en el modelo de la aplicación que es **GestorEgorilla**.

4.1.6.2.2. buscador

El panel buscador se caracteriza por tener un **JTextField** en el que el usuario introduce el nombre del archivo a buscar pudiendo vaciar su contenido con un **JButton limpiar** si decide que se ha equivocado al introducir el nombre.

Una vez que el usuario quiere comenzar la búsqueda pulsará el **JButton comenzar** o **pulsará la tecla ENTER** mostrándose un **JTabbedPane** que muestra una pestaña definida por los desarrolladores con una funcionalidad extra como por ejemplo cerrar dicha pestaña seleccionando el icono correspondiente sobre ella.

Su contenido es un **JPanel** también desarrollado por los desarrolladores llamado **PanelBusqueda**. Dicho panel muestra cada uno de los archivos que ha devuelto la consulta al servidor y los muestra en forma de tabla. Esta clase se caracteriza por implementar la interfaz ***ObservadorAlmacenDescargas***, lo cual le permite mostrar los archivos que se han seleccionado para su descarga en el mismo momento en el que se ha procedido y aceptado dicha solicitud.

Para realizar la descarga de cualquiera de estos archivos en cualquiera de las pestañas abiertas ante diferentes búsquedas el usuario dispone de 3 opciones:

- **Menú contextual** sobre el archivo en cuestión gracias a la implementación de **JPopupMenuListener**.
- **Dobre click de ratón** sobre el archivo en cuestión.
- **Seleccionar el archivo** en la pestaña correspondiente y pulsar el **JButton Descargar Archivo.** La clase **GUIPanelBuscador** implementa la interfaz ***ObservadorPanelBusqueda*** que informa a la clase anterior cada vez que el usuario selecciona un archivo en la clase **PanelBusqueda** correspondiente por lo que guarda ese archivo en una variable interna llamada *_archivoSeleccionado* y cuando el usuario pulsa el **JButton Descargar Archivo** se enviará la solicitud de descarga de ése archivo en cuestión.

Por último el usuario puede cerrar todas sus búsquedas pulsando el **JButton Limpiar Todo.**

Siguiendo con la política de implementación del *patrón MVC*, la pestaña se comunica con la lógica de la aplicación a través de la clase **ControladorPanelBuscador**. Dicha clase recibe los eventos producidos en la vista (la propia pestaña) y llama a los métodos correspondientes en el modelo (**GestorEGorilla**).

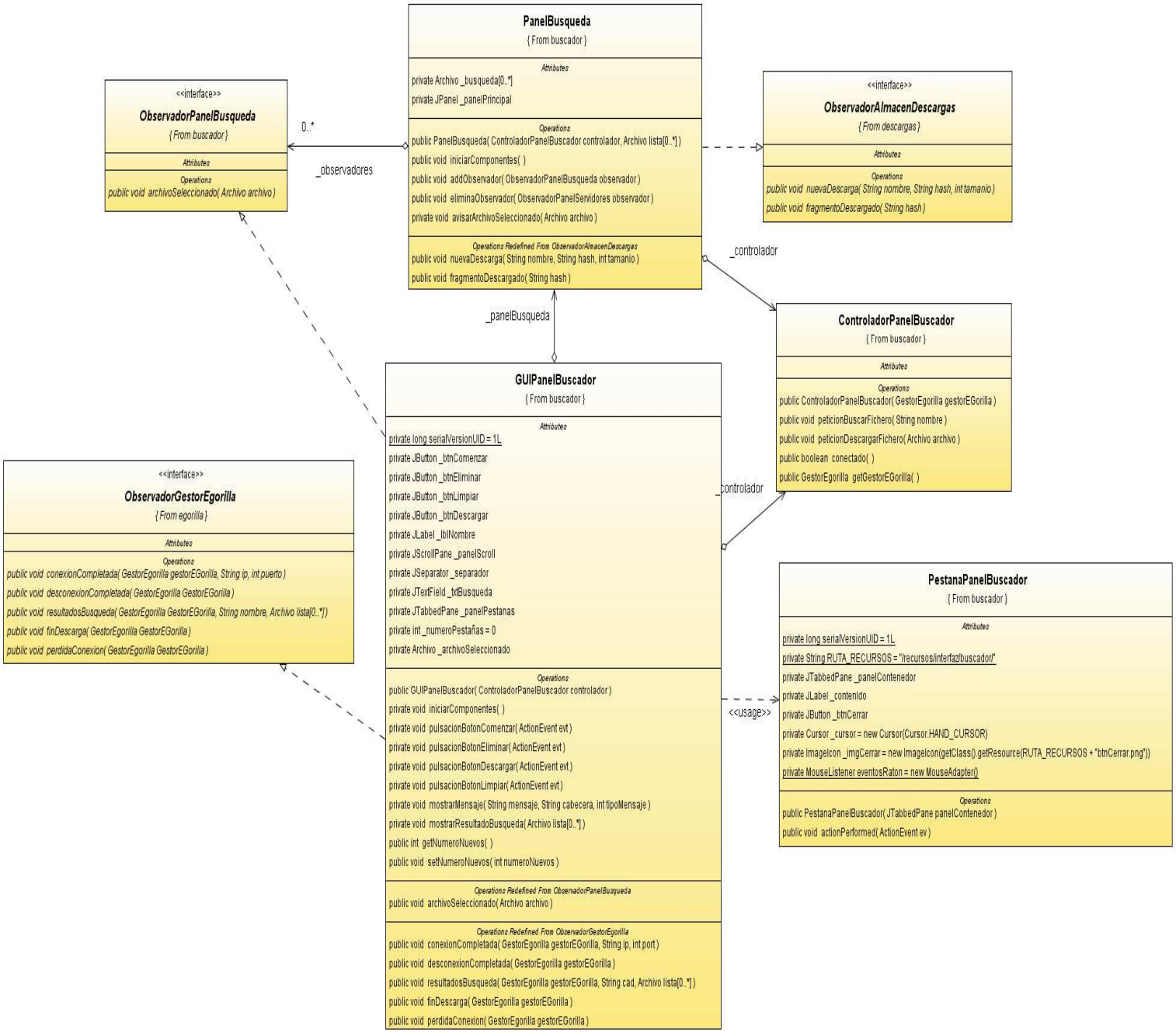


Ilustración - Diagrama de clases del panel buscador

4.1.6.2.3. trafico

El panel de tráfico muestra las descargas del usuario en una tabla, diferenciándose el estado mediante un código de colores.

Dicha lista de descargas también contendrá información de dichas descargas separadas en diferentes campos de información: **nombre**, **hash**, **progreso** y **estado**.

El usuario podrá sobre cada elemento de la tabla de descargas pausar, cancelar y reanudar una descarga.

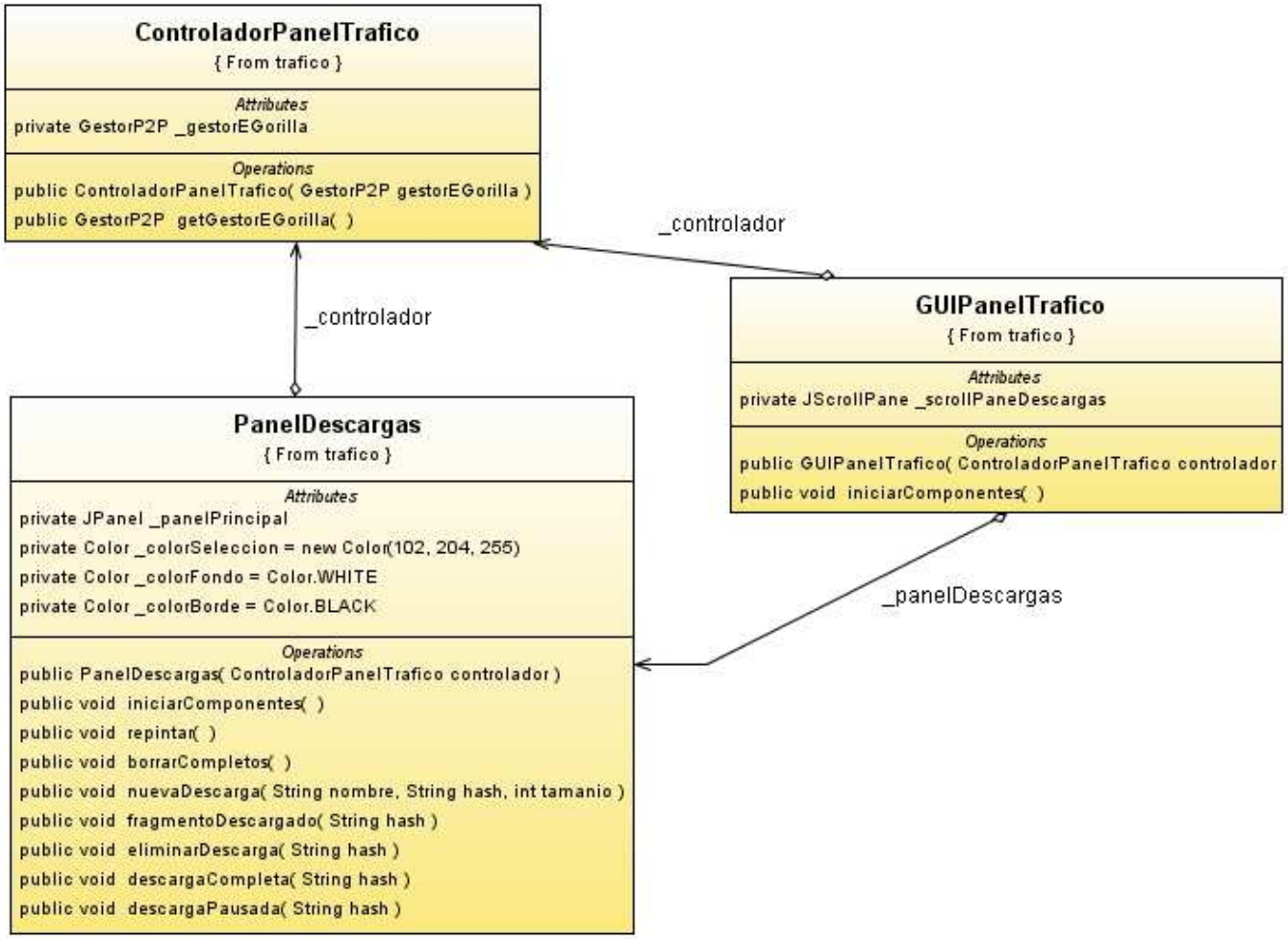


Ilustración - Diagrama de clases del panel tráfico

4.1.6.2.4. estadisticas

4.1.6.2.5. compartidos

El panel de compartidos contiene la información acerca de los archivos compartidos que tiene el cliente eGorilla. Muestra el **nombre**, **tamaño**, **tipo** e **identificador.**

A la izquierda del panel se despliega un árbol de directorios que nos permite filtrar los archivos compartidos según sean **completos** o **incompletos**.

La vista se comunica con la lógica de la aplicación a través de la clase ControladorPanelCompartidos de forma análoga al resto de pestañas y siguiendo con el patrón MVC usado para la interfaz de la aplicación.

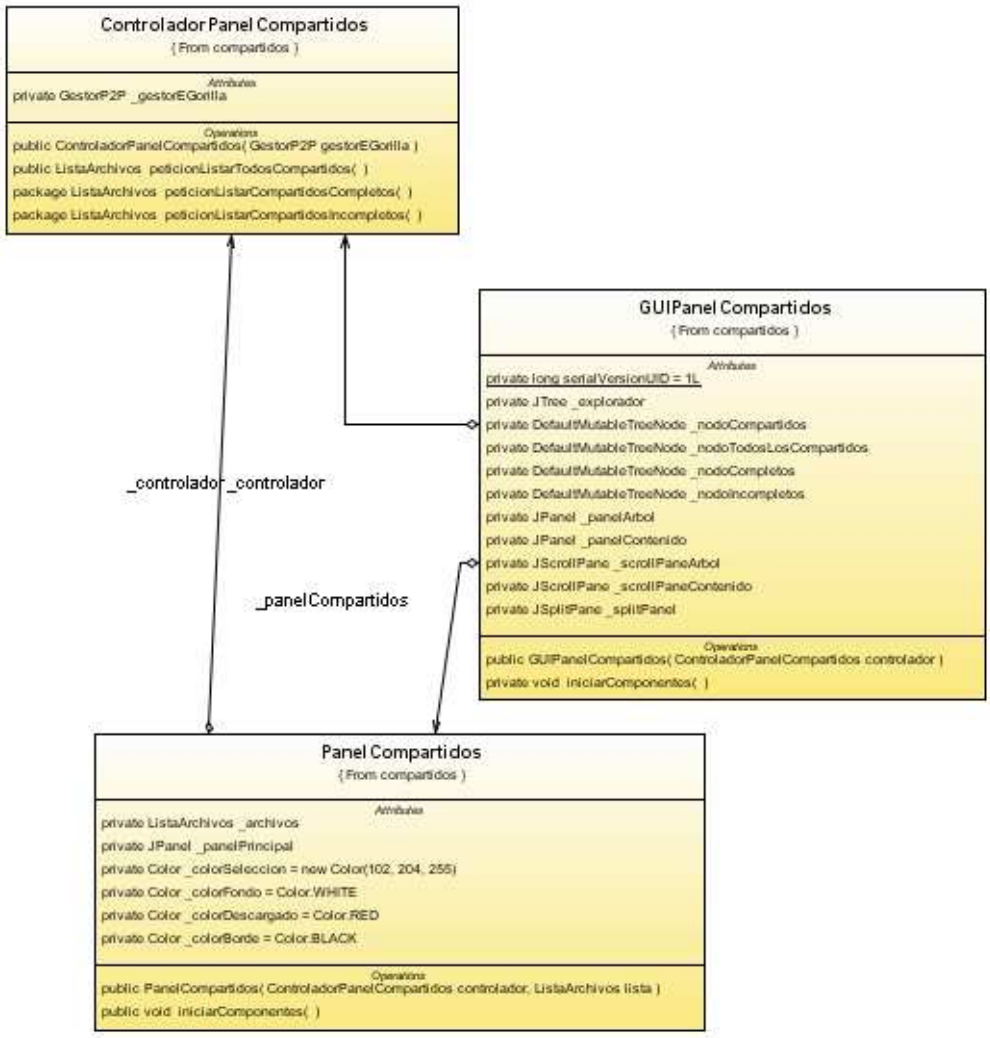


Ilustración - Diagrama de clases del panel de compartidos

A continuación se muestra el Diagrama de Secuencia asociado al panel de compartidos, que refleja las principales operaciones de dicho panel y que se corresponden con la selección del filtro comentando anteriormente (todos, completos, incompletos).

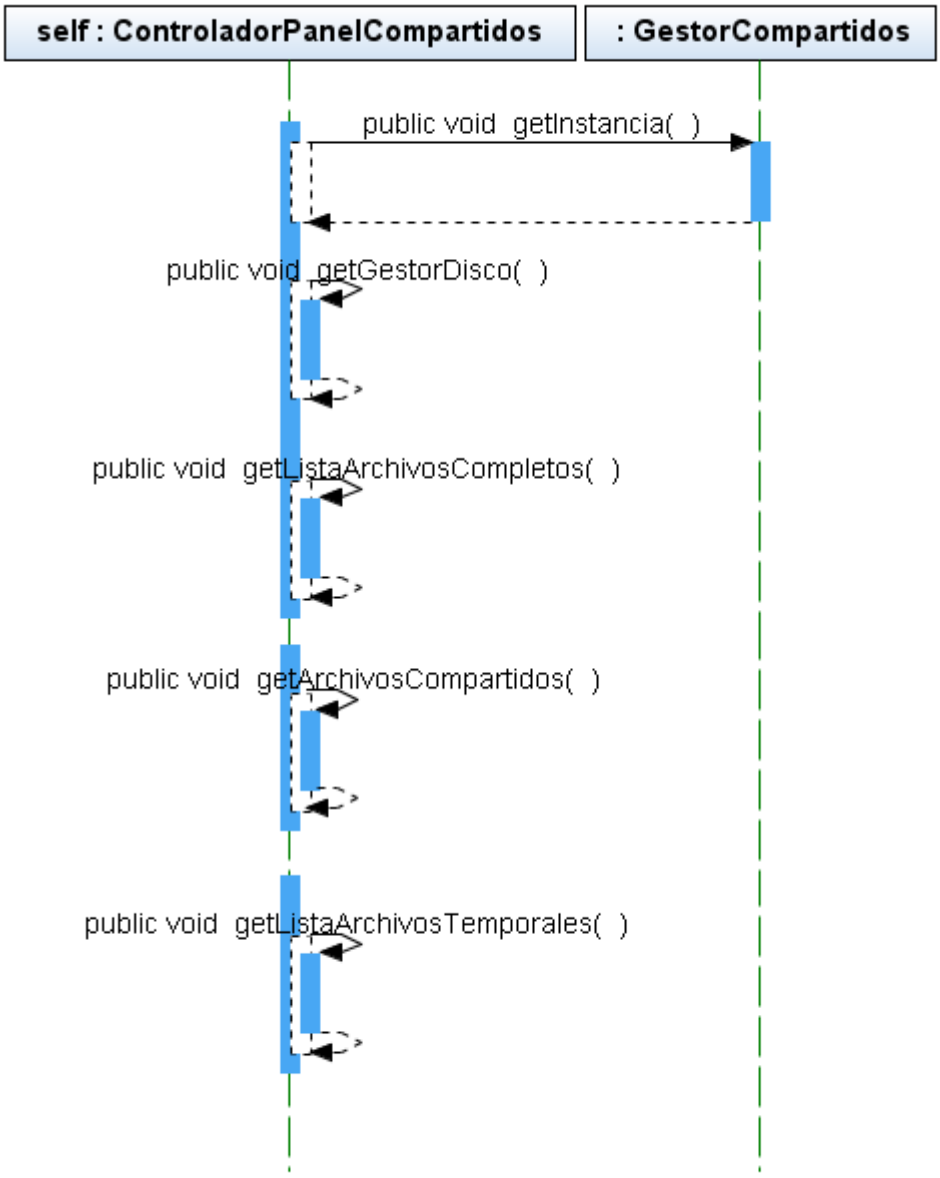


Ilustración - Diagrama de Secuencia del panel de compartidos

4.1.6.2.6. configuración

Para el diseño de esta parte se ha seguido el patrón Modelo-Vista-Controlador. La Vista (un panel) constituye la interfaz con el usuario y notifica eventos al Controlador. Además utiliza al Modelo (los datos de configuración) para actualizarse convenientemente. El Controlador da un tratamiento a los eventos del usuario haciendo uso de la funcionalidad del Modelo. Además puede interactuar directamente con la Vista para actualizarla bajo determinadas circunstancias. Por último el Modelo se ocupa de la configuración y no conoce nada ni de la Vista ni del Controlador. Además el patrón MVC se ha combinado con el patrón Observador (la Vista es observadora del Modelo) como suele ser habitual.

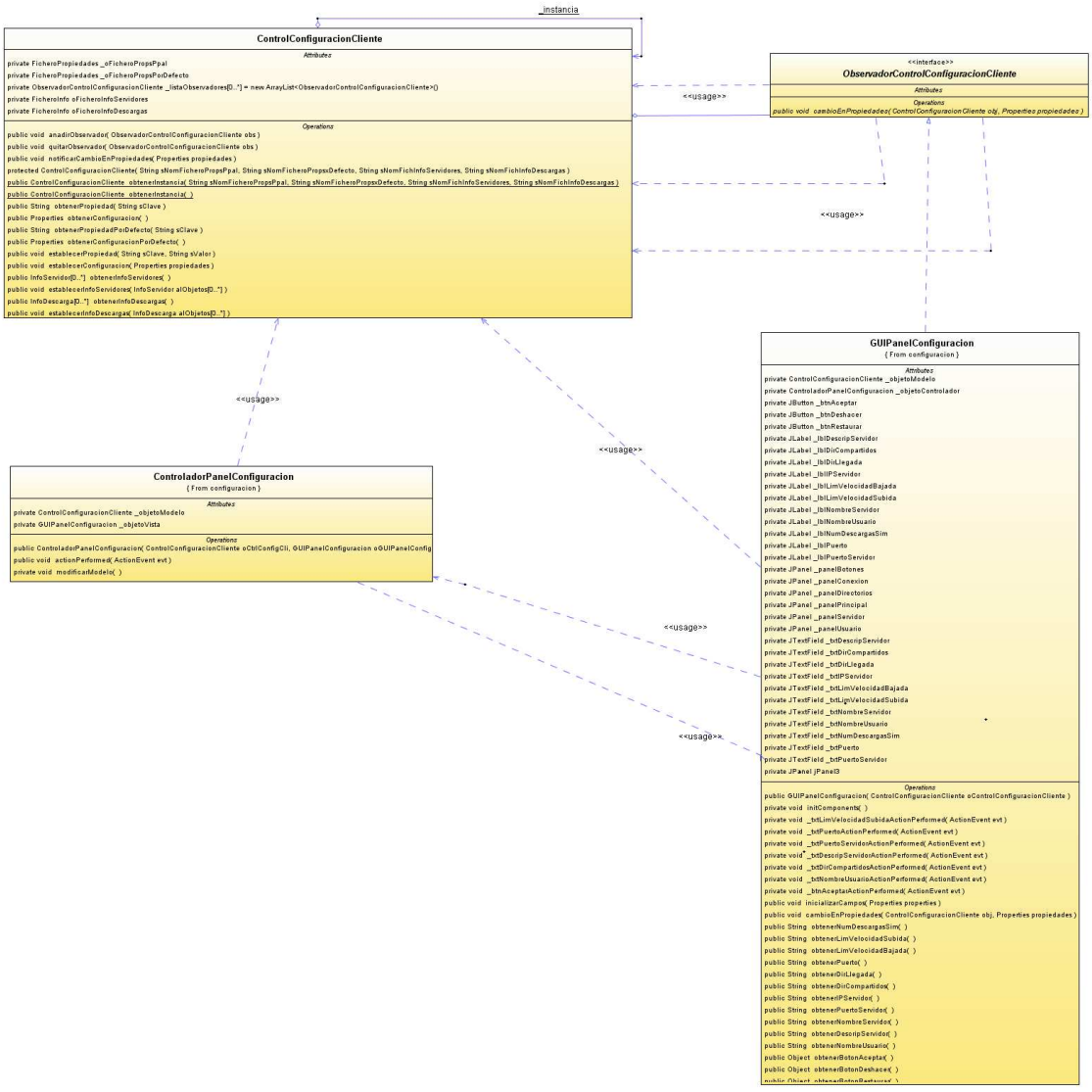


Ilustración – Diagrama de clases del panel de configuración.

4.1.6.2.7. servidores

El panel de servidores contiene los JTextField necesarios para que el usuario introduzca los datos necesarios de servidores a añadir a la lista de servidores disponibles en la aplicación, a saber el **nombre, direccionIP, puerto** y una **breve descripción.**

Una vez que el usuario pulse el **JButton Añadir a la Lista** se cargará en el panel PanelServidores que muestra la lista de servidores disponibles en la aplicación.

Para conectarse al servidor que el usuario decida el usuario tiene las siguientes opciones:

- Seleccionar el servidor en la lista y pulsar el **JButton Conectar**. La clase GUIPanelServidores implementa la interfaz ***ObservadorPanelServidores*** que le informa de la selección de un servidor en el panel PanelServidores.

- Mediante el **menú contextual** de cada servidor.

- Haciendo **doble click** sobre el servidor deseado.

La vista se comunica con la lógica de la aplicación a través de la clase ControladorPanelServidores siguiendo con el patrón MVC usado para la interfaz de la aplicación.

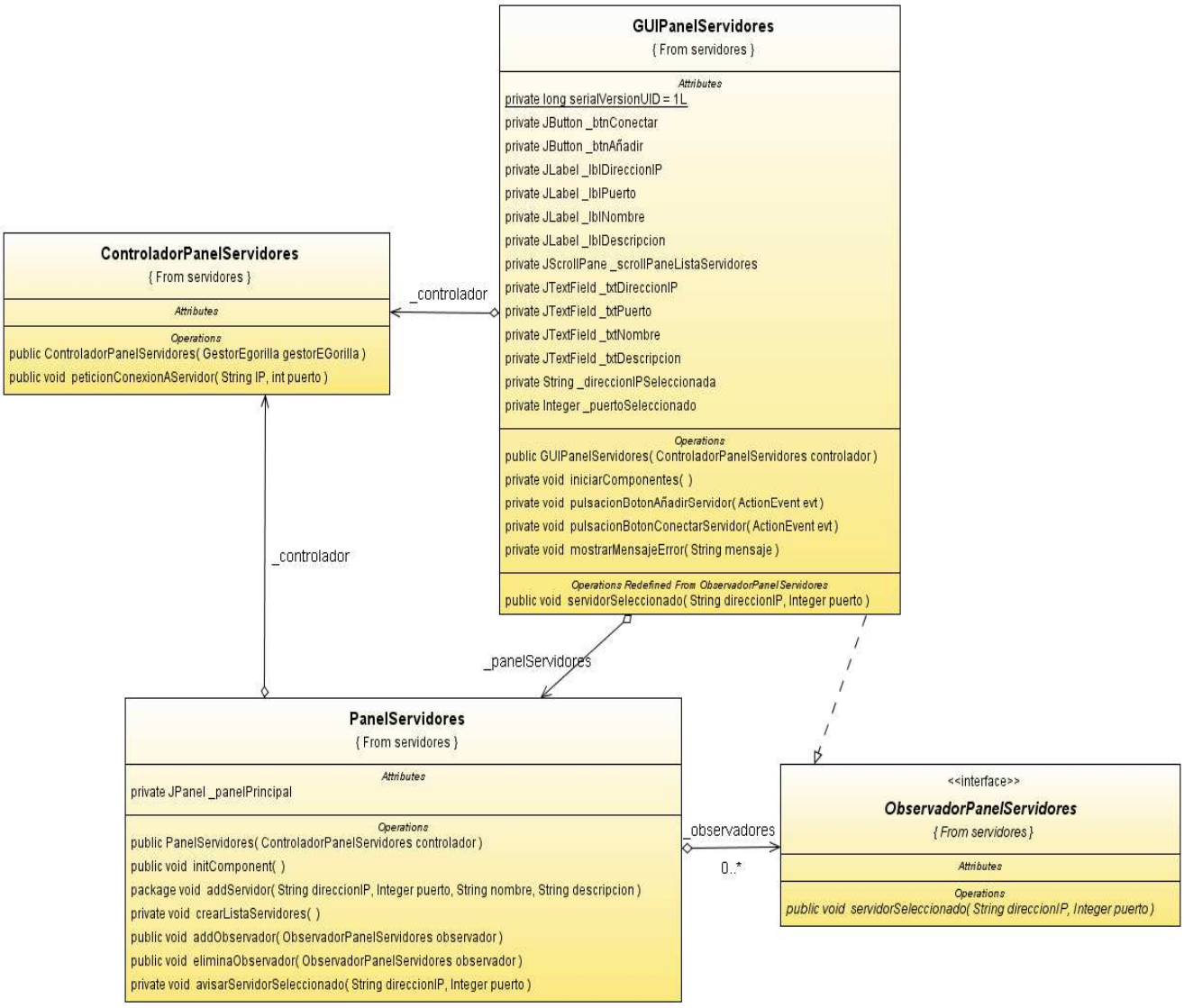


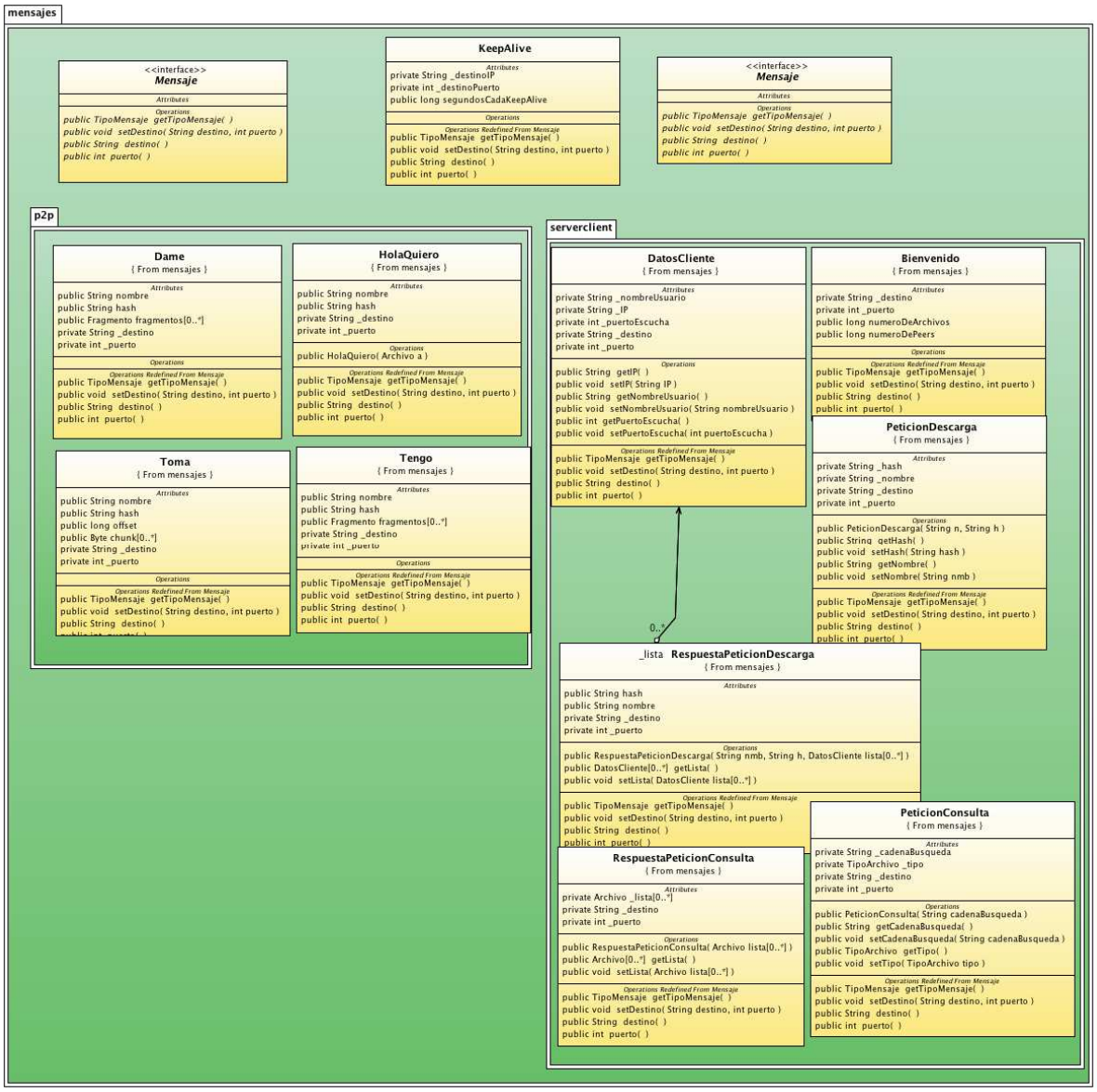
Ilustración – Diagrama de clases del panel de servidores.

4.1.6.2.8. ayuda

4.2. Comunicaciones

4.2.1. Protocolo eGorilla

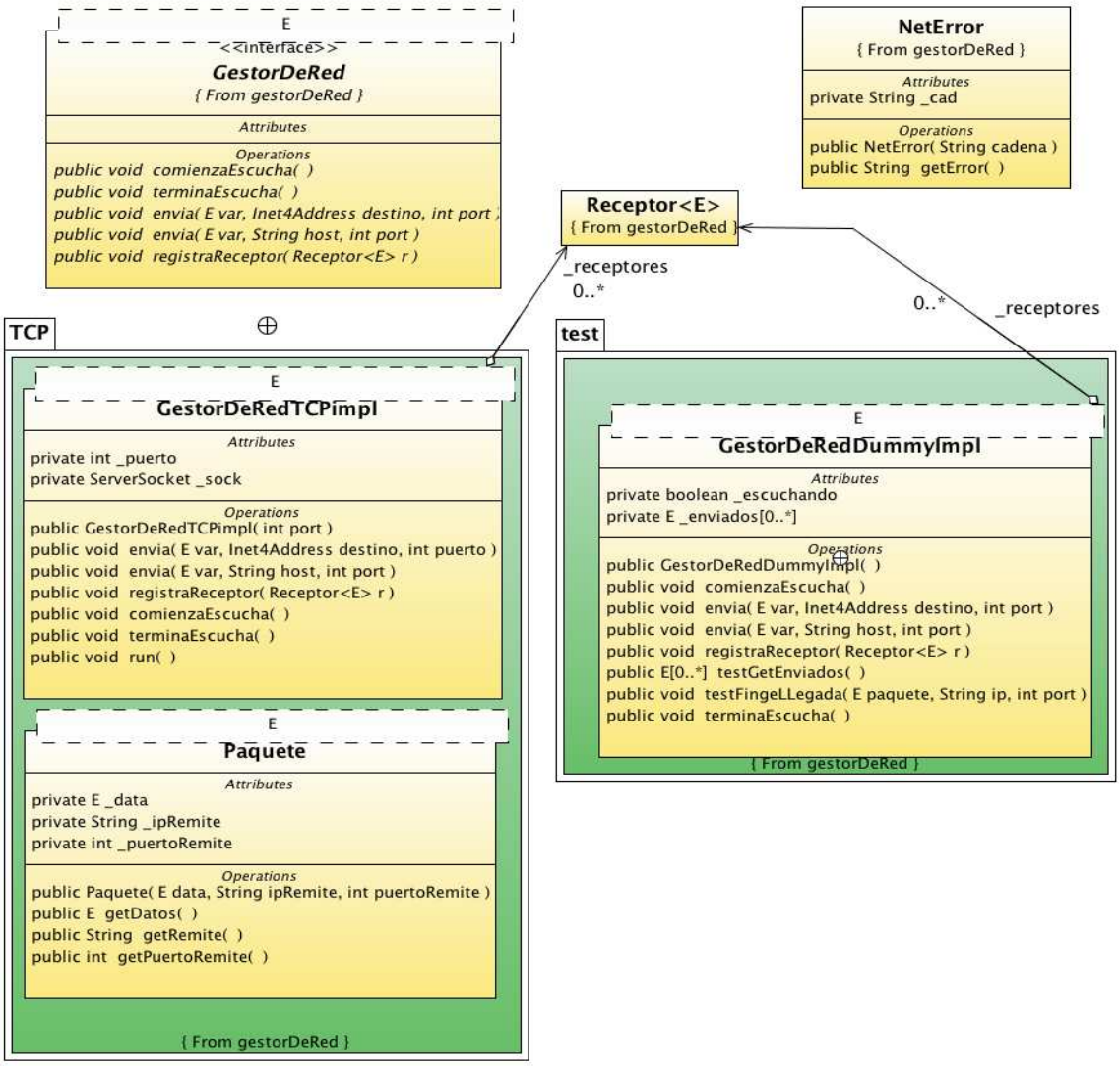
El protocolo eGorilla se comunica compartiendo mensajes ya definidos en un orden establecido. La comunicación puede ser entre dos clientes o entre un cliente y un servidor, por lo que el diálogo es diferente.



4.2.2. gestorDeRed

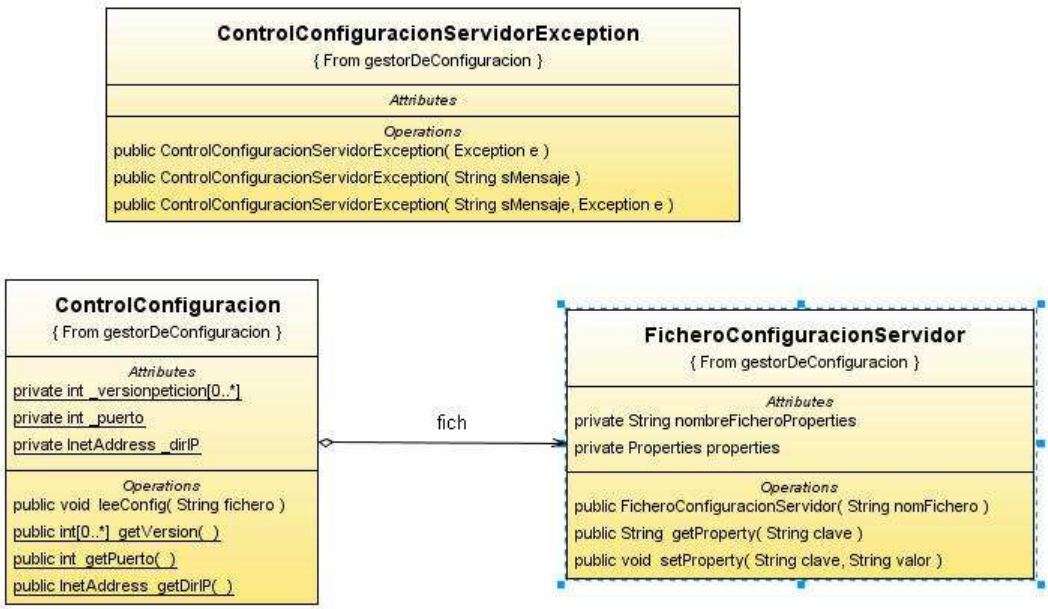
Todas las comunicaciones de red se realizan mediante una interfaz que realiza la abstracción del uso de **sockets** en la aplicación. La **parametrización** de la clase hace que solo sea posible enviar un tipo de objetos. El uso práctico se realizará instanciando la plantilla con la clase Mensaje permitiendo enviar cualquier tipo de mensajes, en nuestro caso se enviarán **mensajes eGorilla**.

Para la realización de pruebas se ha creado una implementación del gestor que no realiza ninguna comunicación de forma que se pueden realizar pruebas sin la necesidad de montar la infraestructura de red.



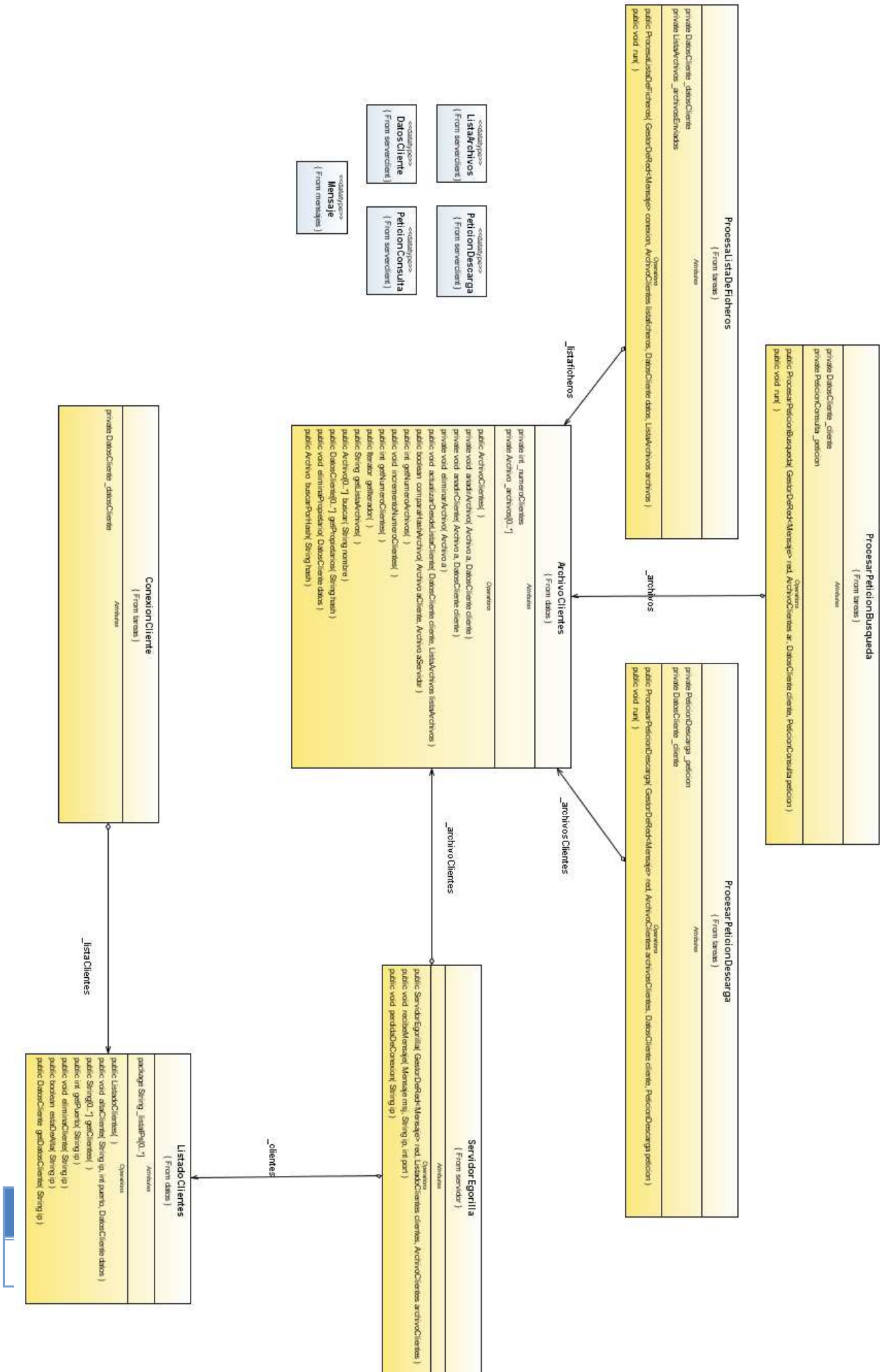
4.3. Servidor

4.3.1. gestorDeConfiguracion

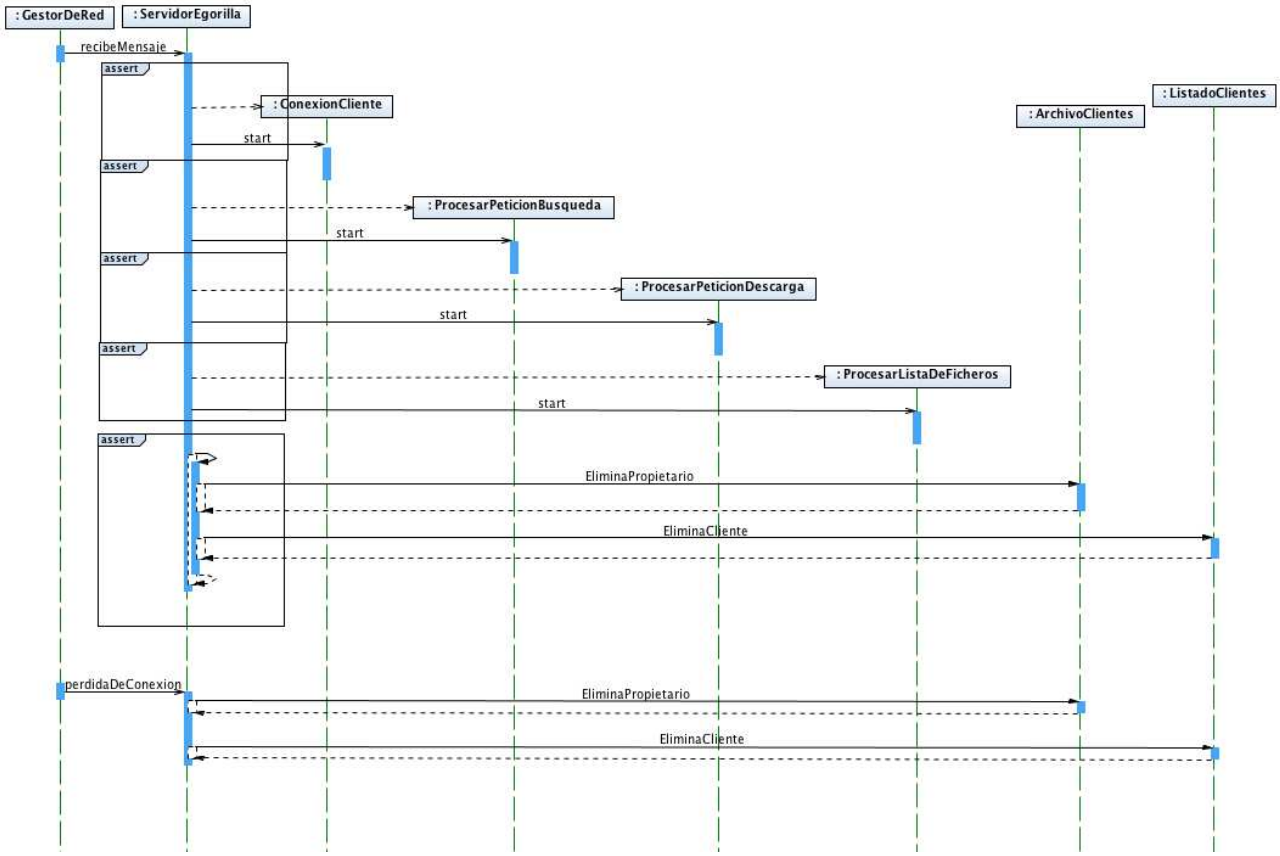


4.3.2. servidor

El servidor eGorilla funciona de forma que se satisfacen las peticiones de los clientes a medida que estas llegan. Por lo tanto, dependiendo del mensaje recibido se realiza una acción u otra. Así mismo, se tratan las peticiones de forma concurrente, siguiendo una metodología en la cual se expande un hilo para tratar cada petición de forma independiente y simultanea.


ProcesarPeticionBusqueda
ProcesaListaDeFicheros
ProcesarPeticionDescarga
ArchivoClientes
ConexionCliente
ListadoClientes
ServidorEgorilla
ListaArchivos
DatosCliente
PeticionDescarga
PeticionConsulta
Mensaje
_listaficheros
_archivos
_archivosClientes
_archivoClientes
_listaClientes
_clientes

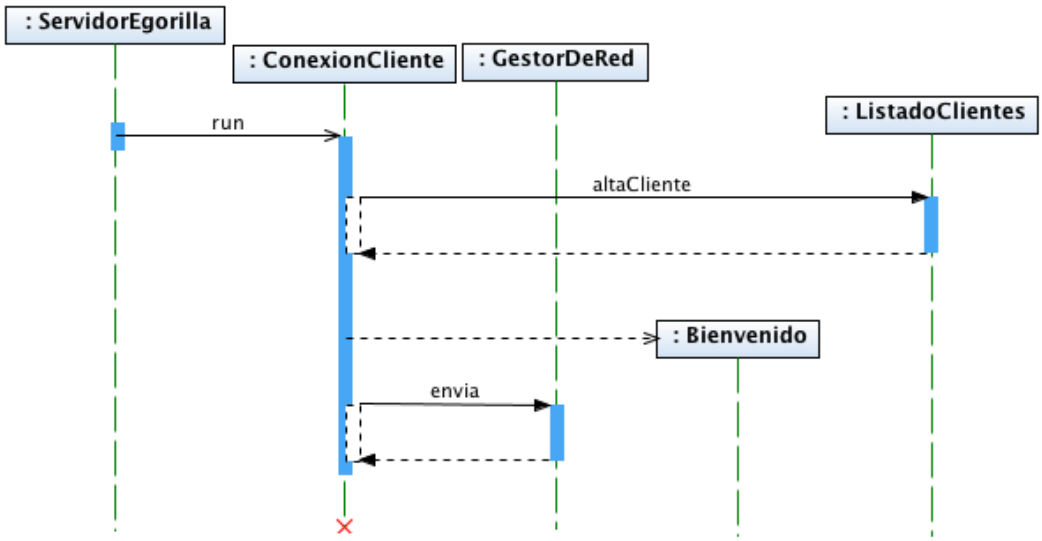
Las peticiones que se tratan en el siguiente diagrama por orden de aparición (marcadas con asserts) son las siguientes: DatosCliente, PeticionConsulta, PeticionDescarga, ListaFicheros y Altoo.
Si no se satisface ninguna de las condiciones, se ignora el mensaje y se continua



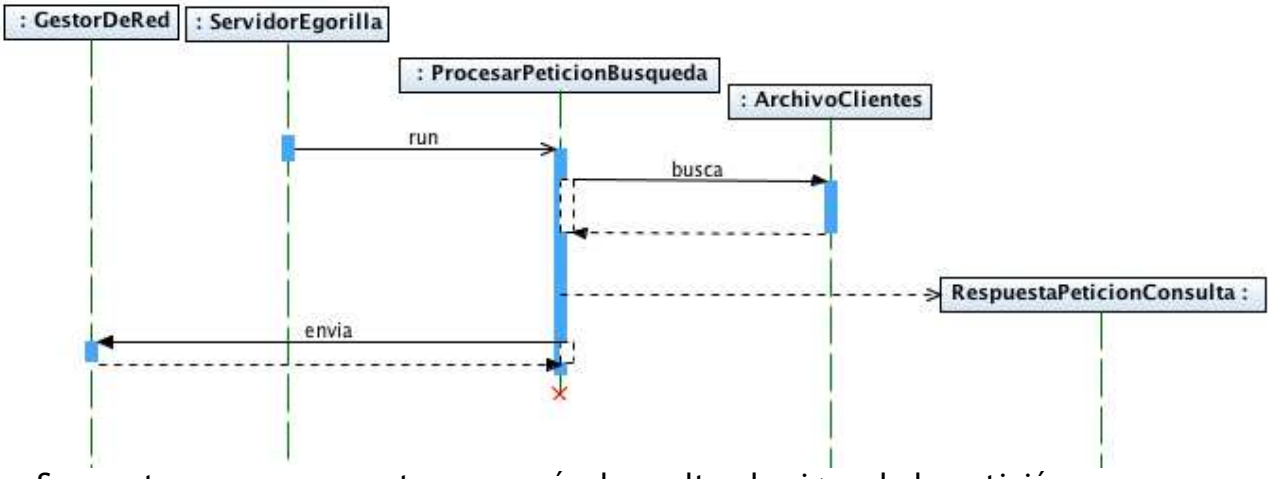
escuchando la red.
A continuación se detalla el tratamiento de cada uno de los hilos que se ejecutan para satisfacer las peticiones.

4.3.2.1. tareas

ConexionCliente:
Da de alta al cliente en el sistema. Almacenando la información de este para que el servidor sea capaz de comunicar esta a otros clientes.

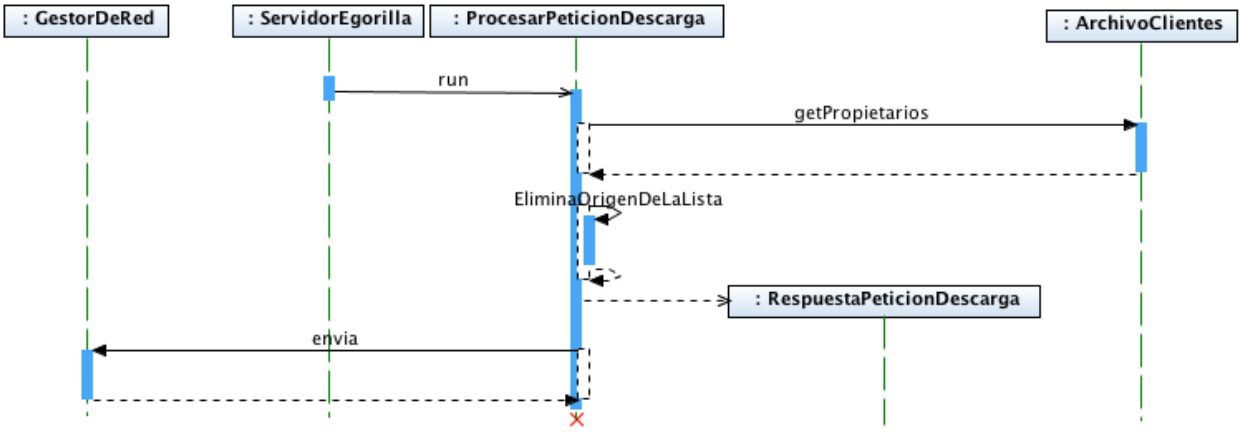


PeticionBusqueda:
Realiza la búsqueda de una cadena de texto entre los nombres de los ficheros dados de alta en el sistema.



Se construye una respuesta y se envía de vuelta al origen de la petición.

PeticionDescarga:
Cuando un Cliente comunica su interés por descargar un archivo, se comprueba que clientes contienen ese archivo y se envía de vuelta una lista con los clientes que tienen ese archivo.



Además se elimina al cliente origen de la consulta de la lista devuelta, ya que este puede estar preguntando por un fichero que tenga a medio descargar, y por supuesto, no va a realizar la descarga desde si mismo.

ListaFicheros:
Un cliente puede comunicar al servidor en cualquier momento la lista de ficheros que comparte, de forma que estos estén disponibles en las búsquedas que realicen otros clientes.

5.

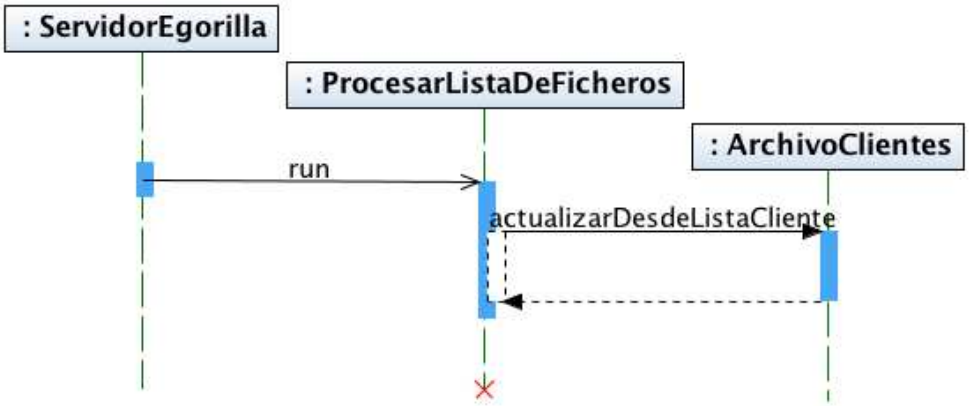


Diagrama de comunicación entre módulos