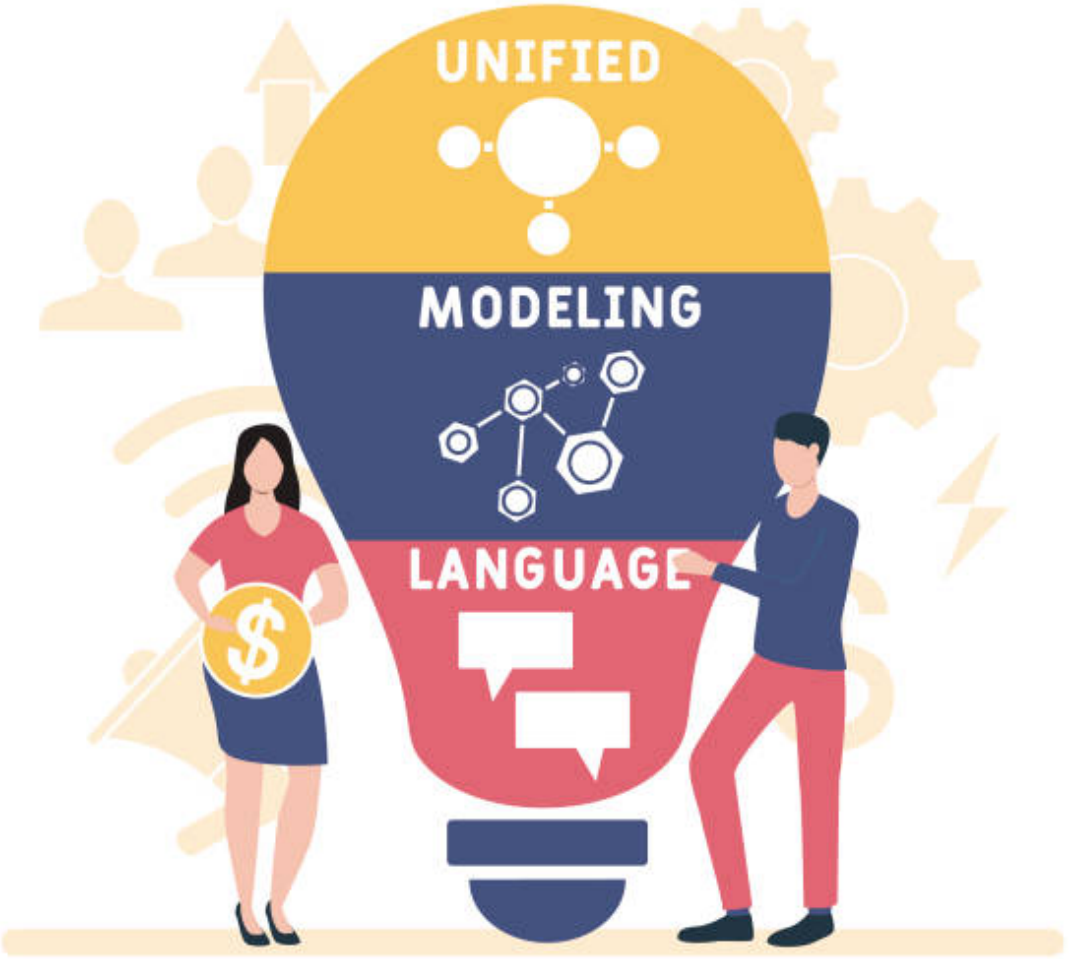
Diseño UML



Los Amatistas

14-3-2022

1. Arquitectura

Con respecto a la arquitectura de la aplicación, hemos realizado un modelo de dominio que permite un fácil entendimiento de la funcionalidad básica que permite la aplicación, así como una serie de diagramas de clases que describen más detalladamente la lógica de la misma.

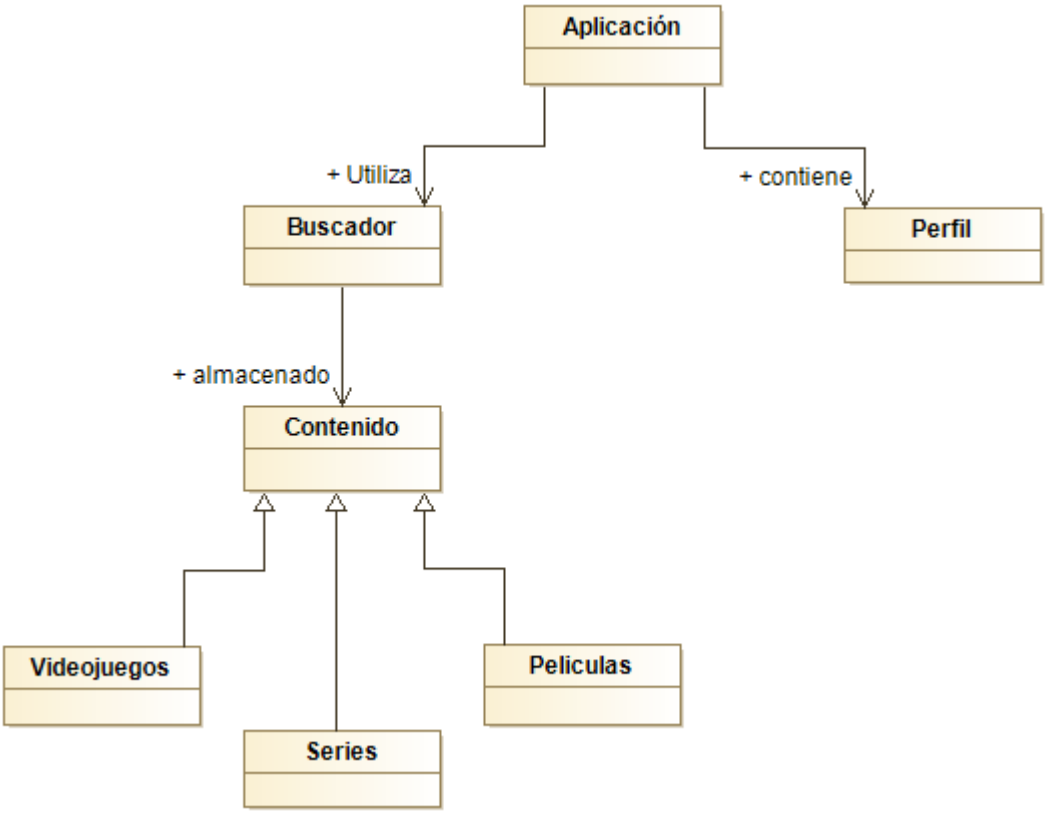


Diagrama de clases que representa el modelo

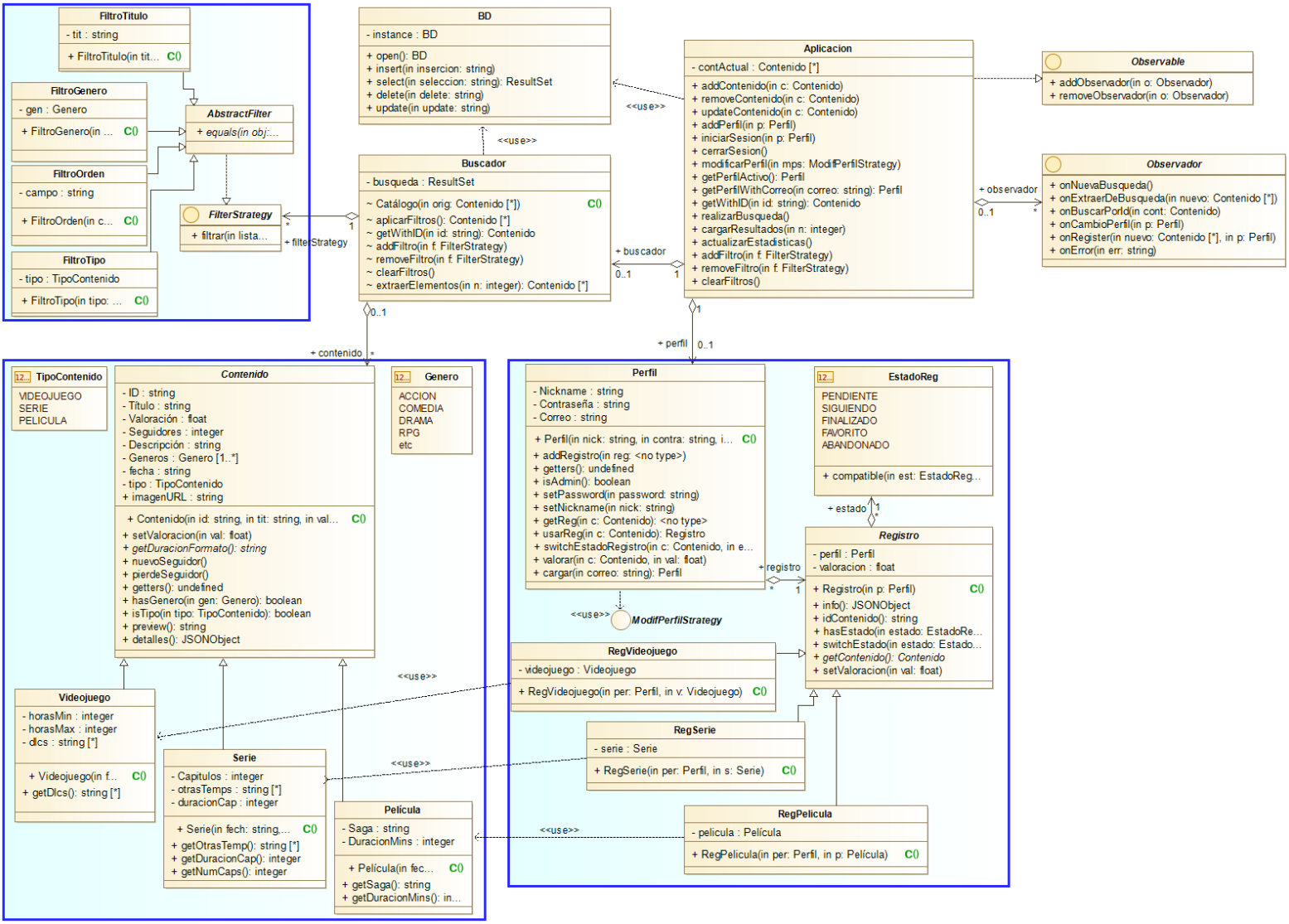


Diagrama de clases del controlador

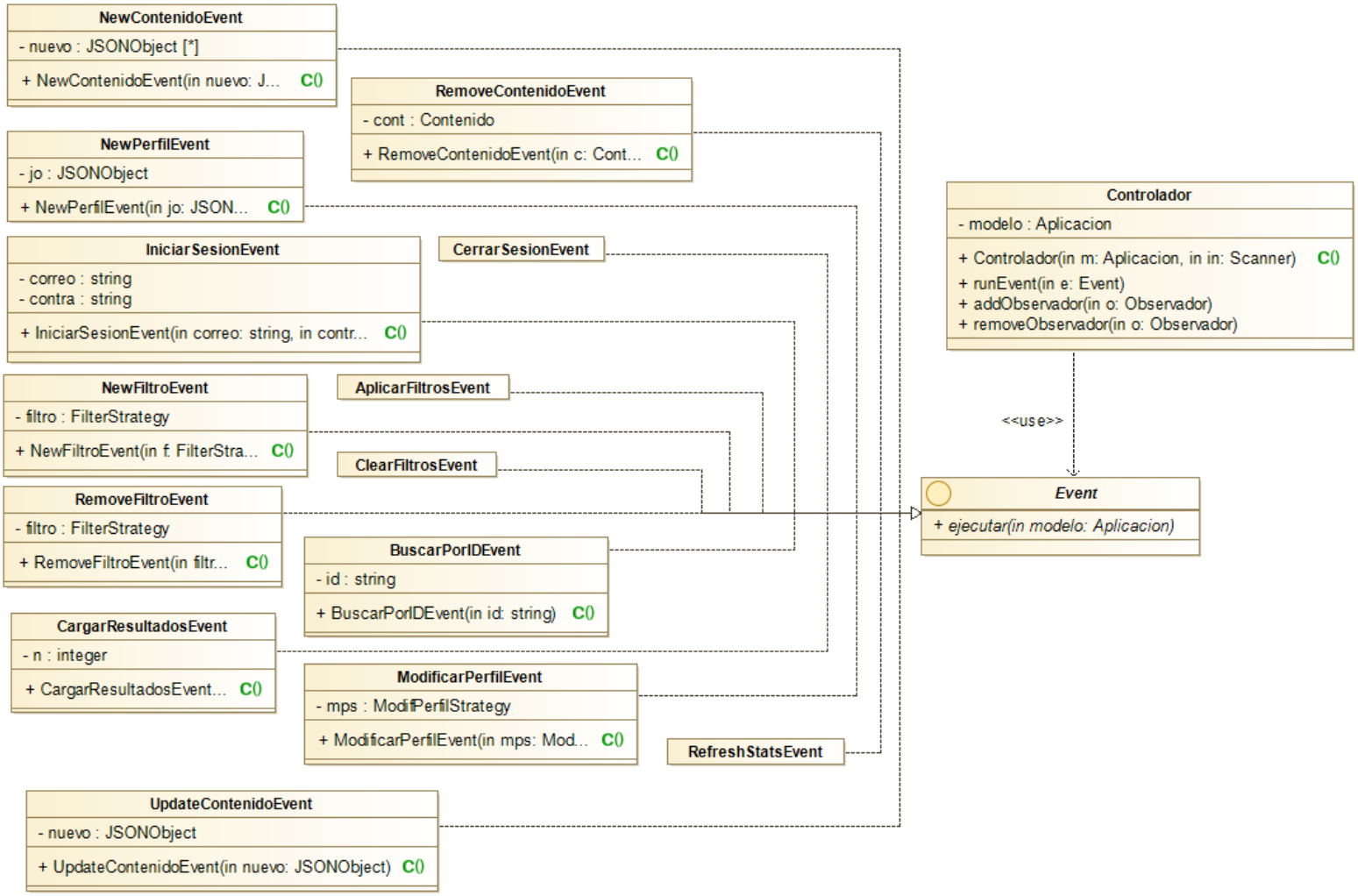


Diagrama de clases de las factorías

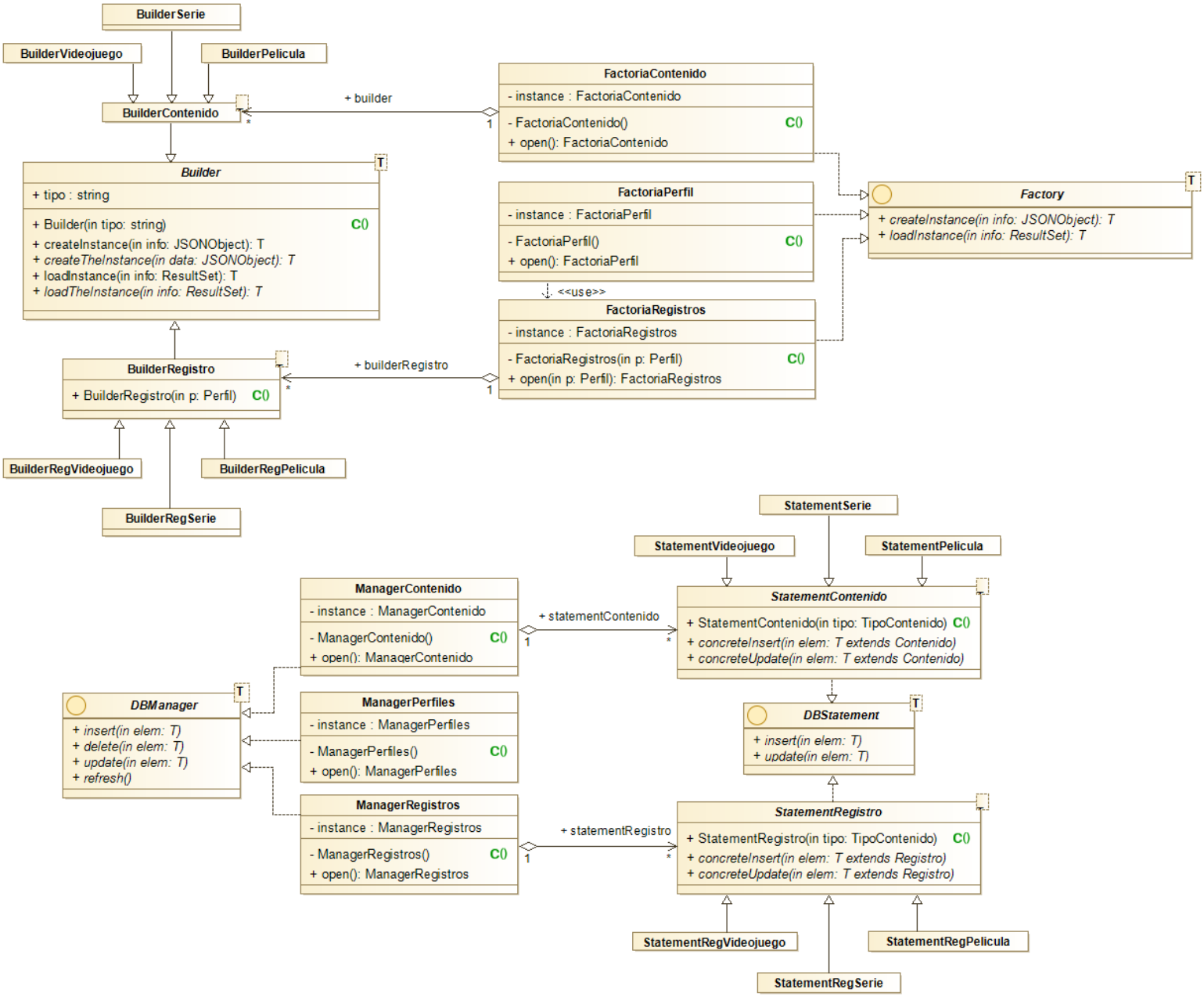
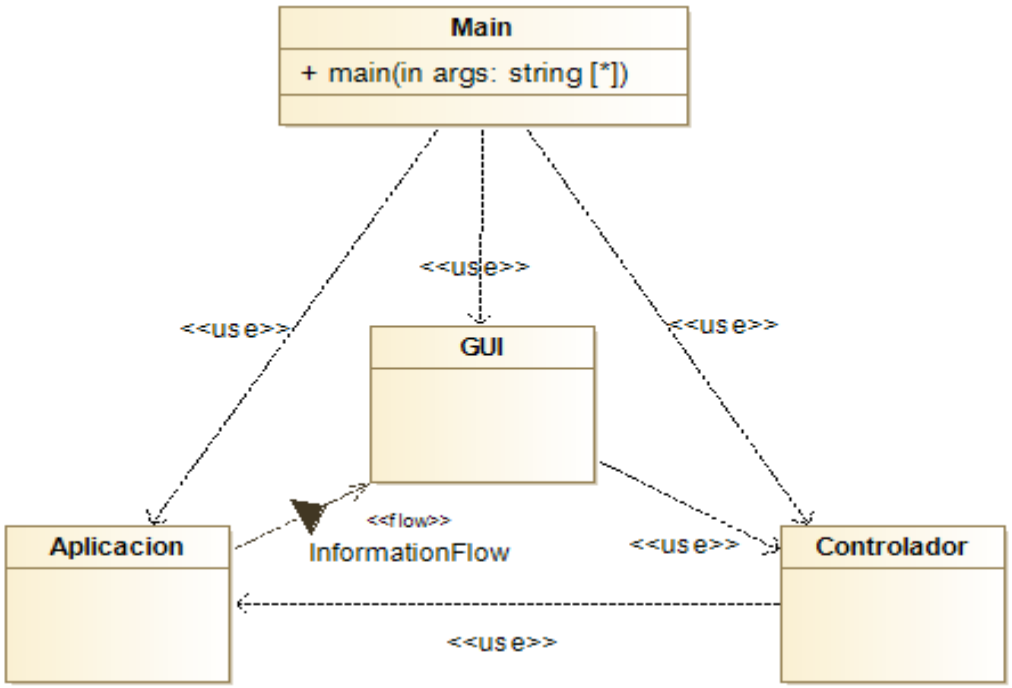


Diagrama de clases que muestra la estructura seguida en el proyecto (Modelo vista-controlador)

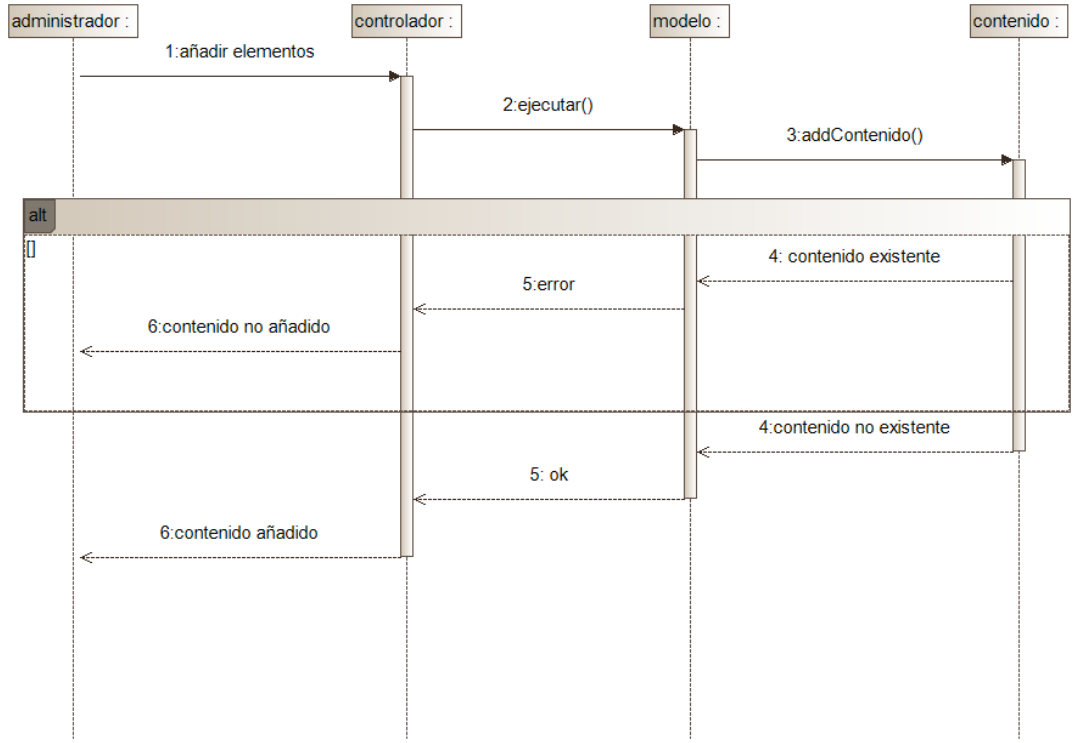


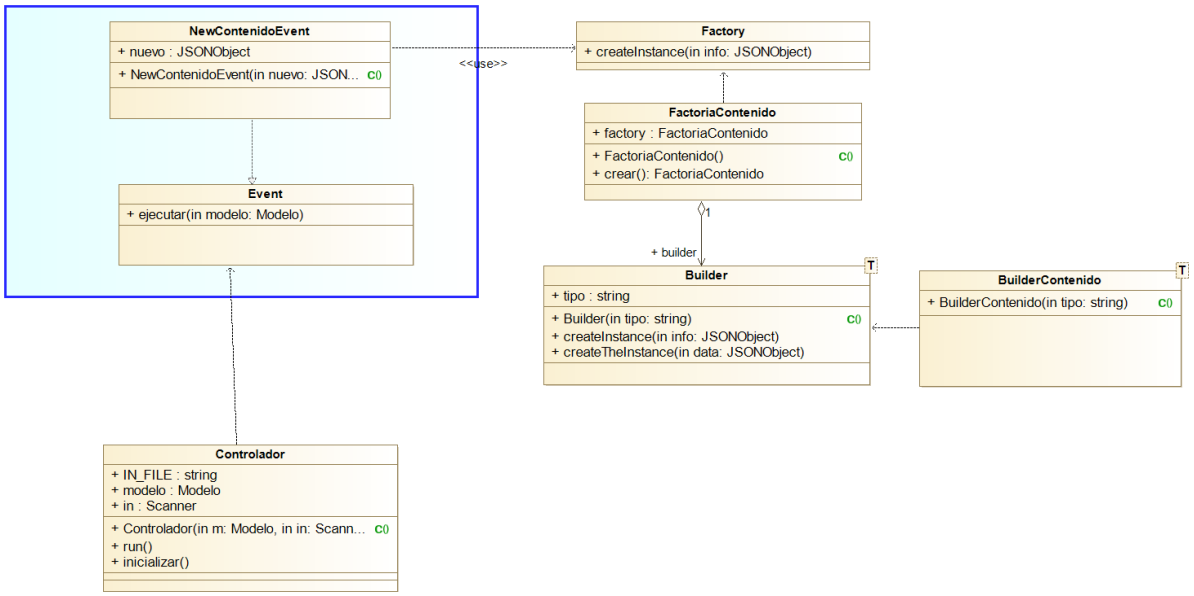
2. Diseño

2.1 Sprint 1

El uso de las siguientes funcionalidades se realiza mediante eventos que siguen el patrón Command.

2.1.1 Introducir elementos

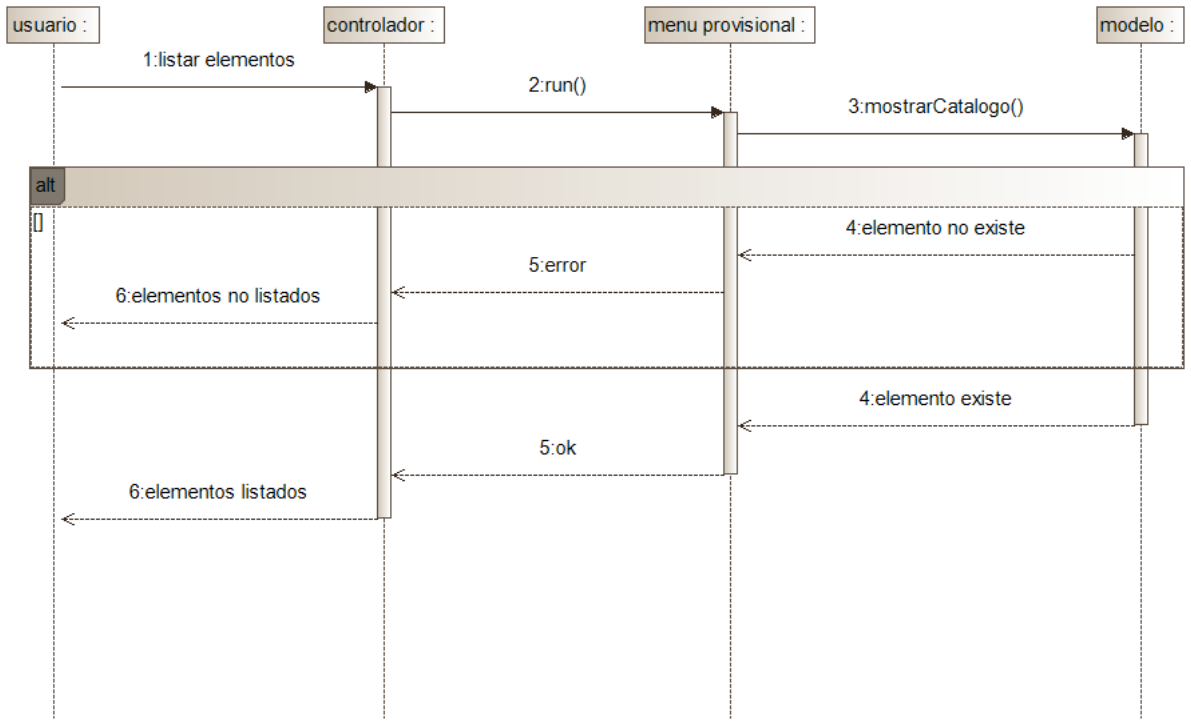


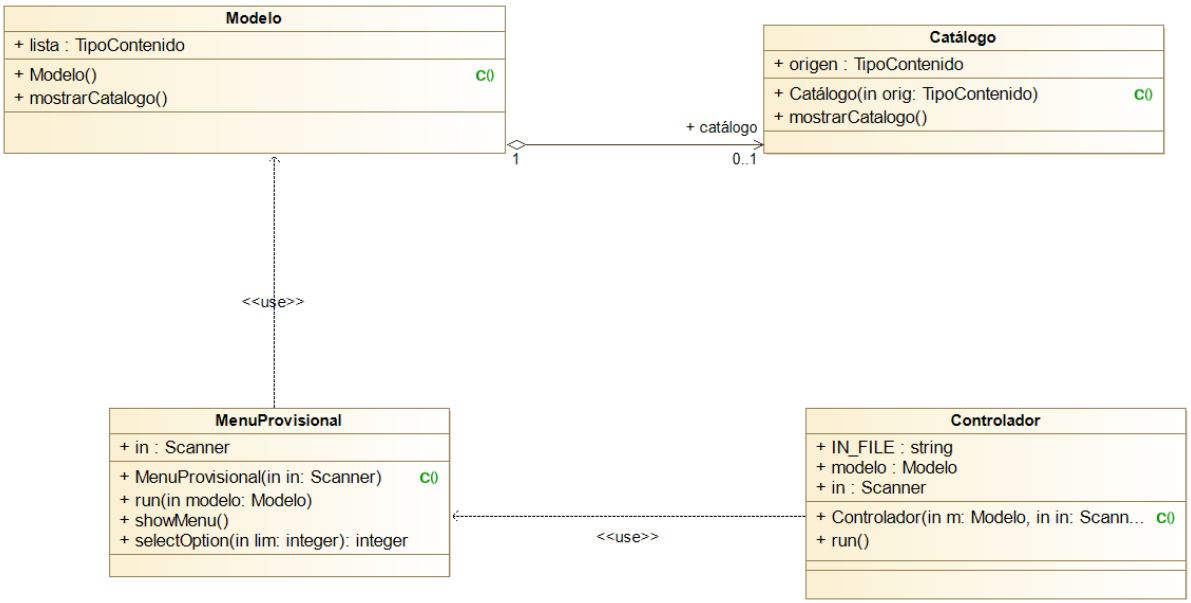


Descripción: Al inicializar la aplicación todos los contenidos del catálogo son añadidos a partir de un archivo de inicio.

Patrones: Para crear los objetos de Contenido se utiliza un patrón de builders, almacenados siguiendo el patrón Factory Method en una factoría de contenido. Esta a su vez está diseñada siguiendo el patrón singleton, puesto que una sola instancia de la factoría puede cumplir la funcionalidad desde cualquier clase.

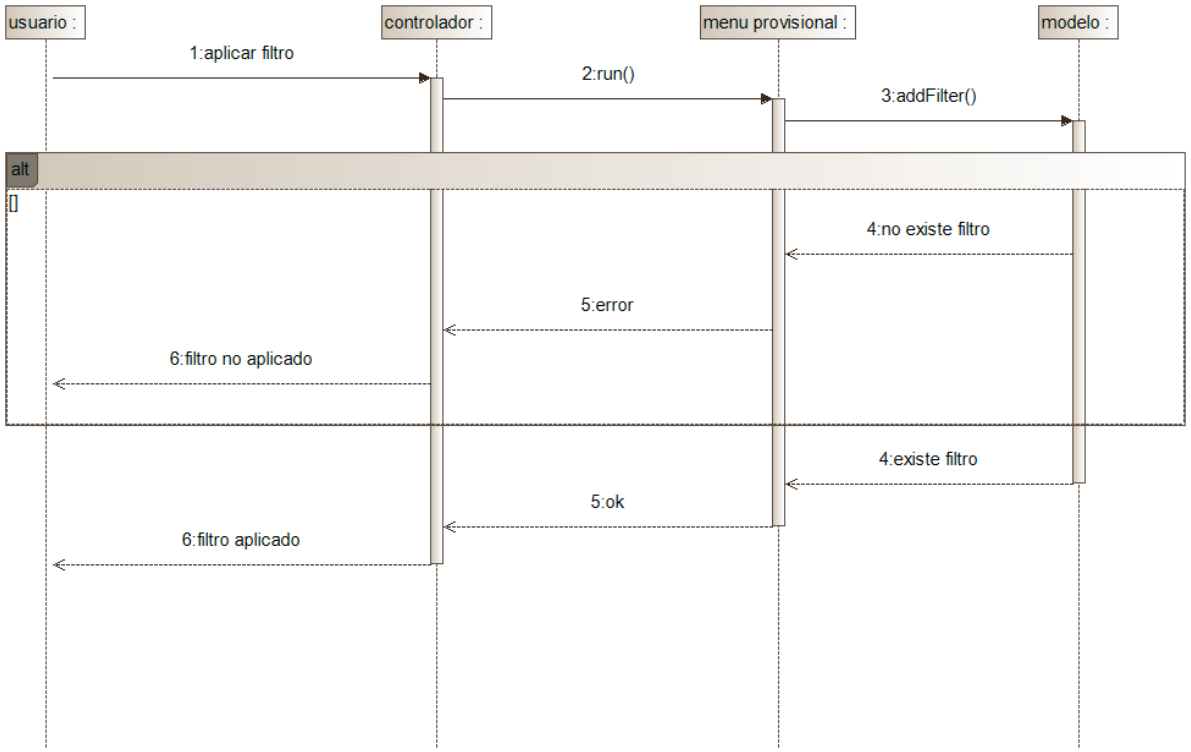
2.1.2 Listar elementos

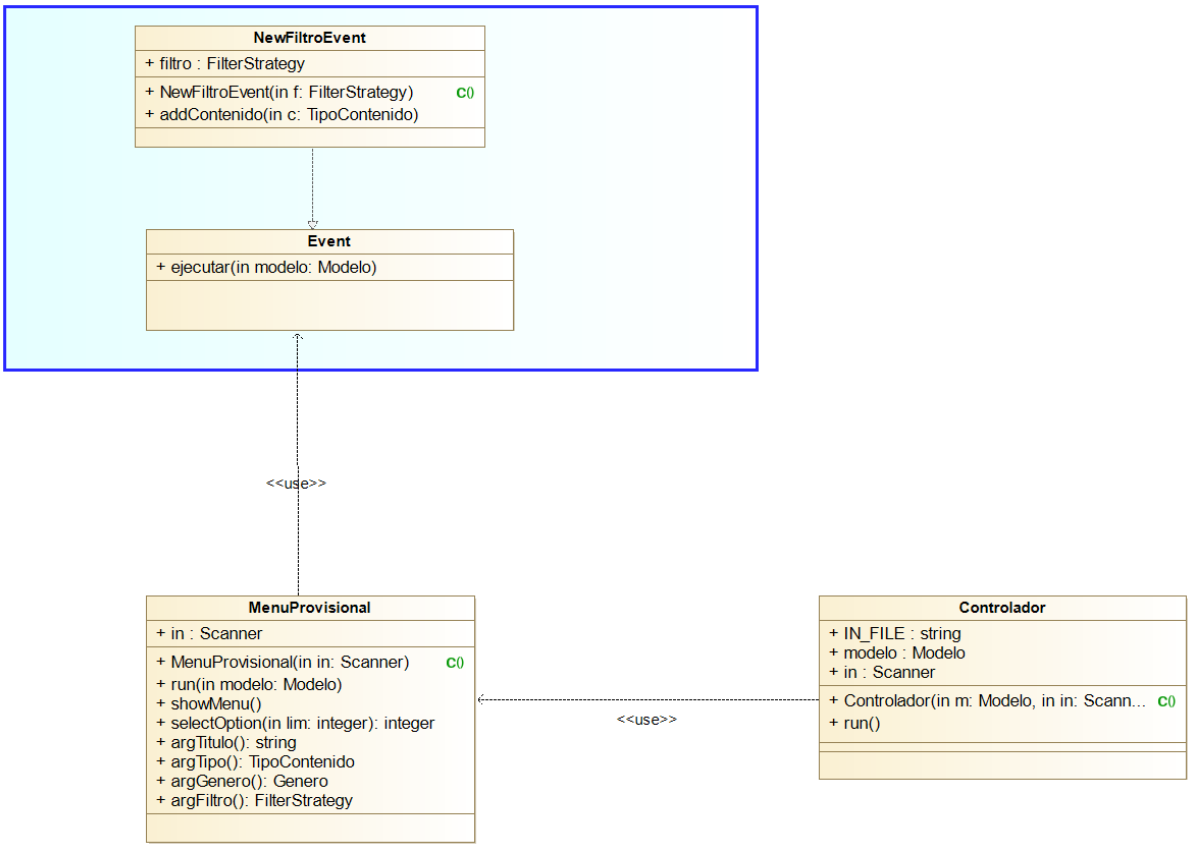




Descripción: Los elementos del catálogo pasan por los filtros que haya aplicados y se muestran por pantalla.

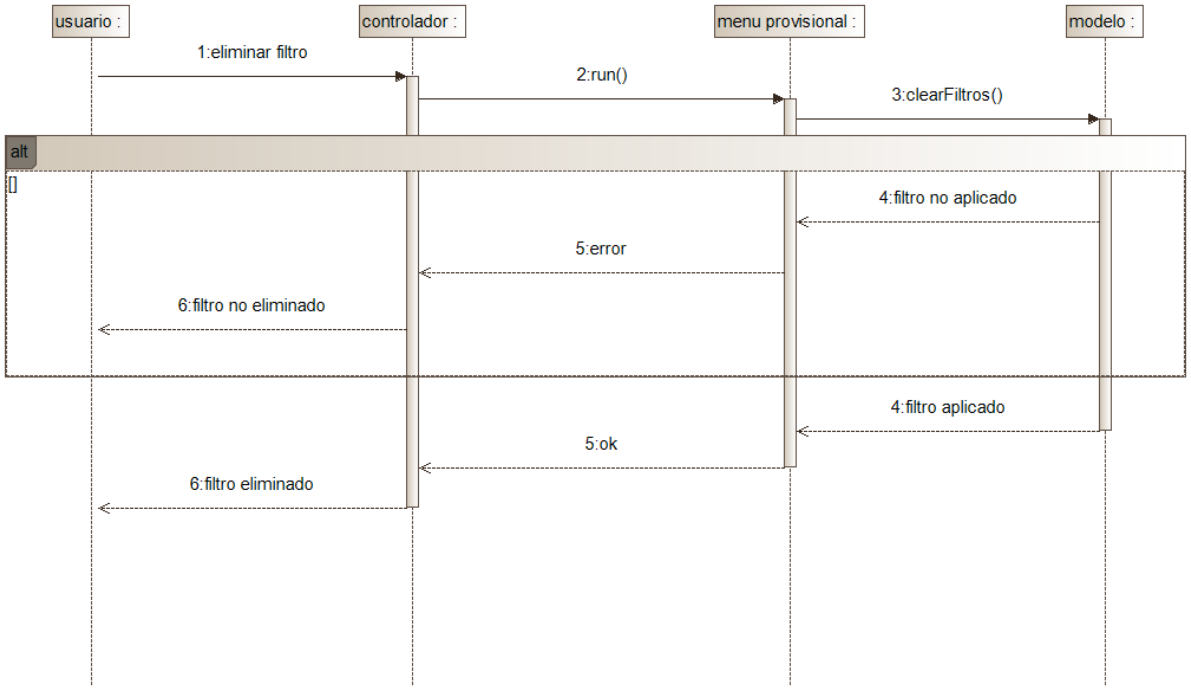
2.1.3 Aplicar un filtro

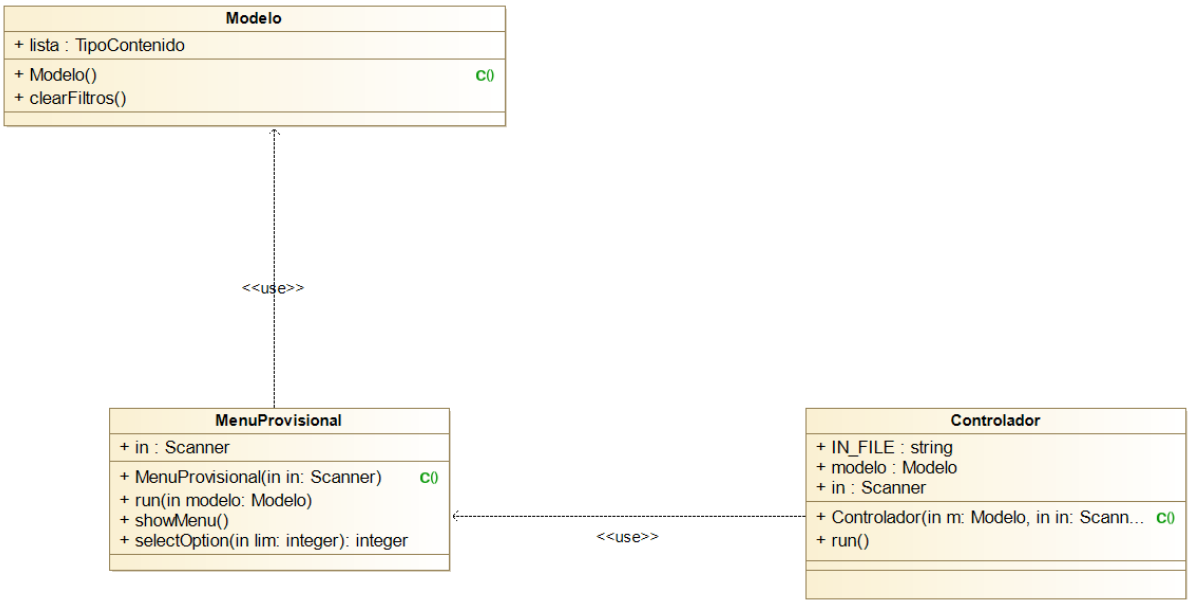




Descripción: Se aplica un filtro de búsqueda para mostrar posteriormente los elementos deseados por el usuario.
Patrones: Los filtros siguen el patrón Strategy, implementando todos ellos el interfaz FilterStrategy, de forma que cada uno filtra las listas de contenido de una forma distinta.

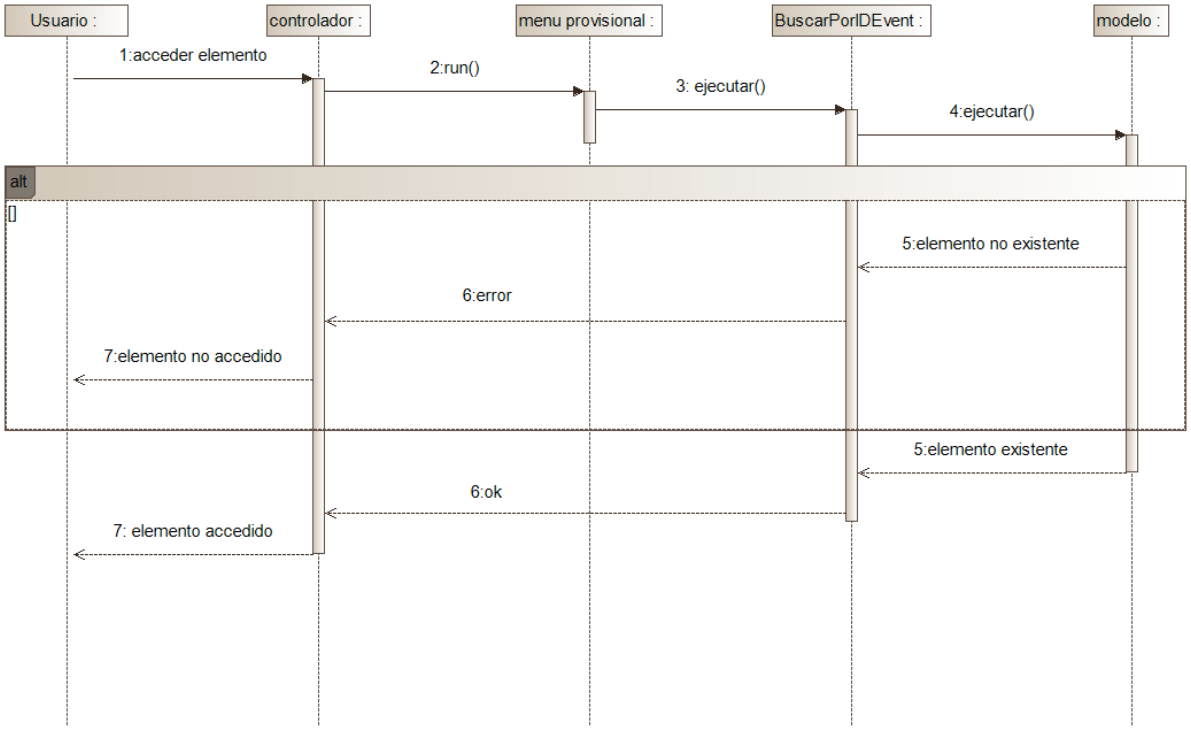
2.1.4 Eliminar filtros

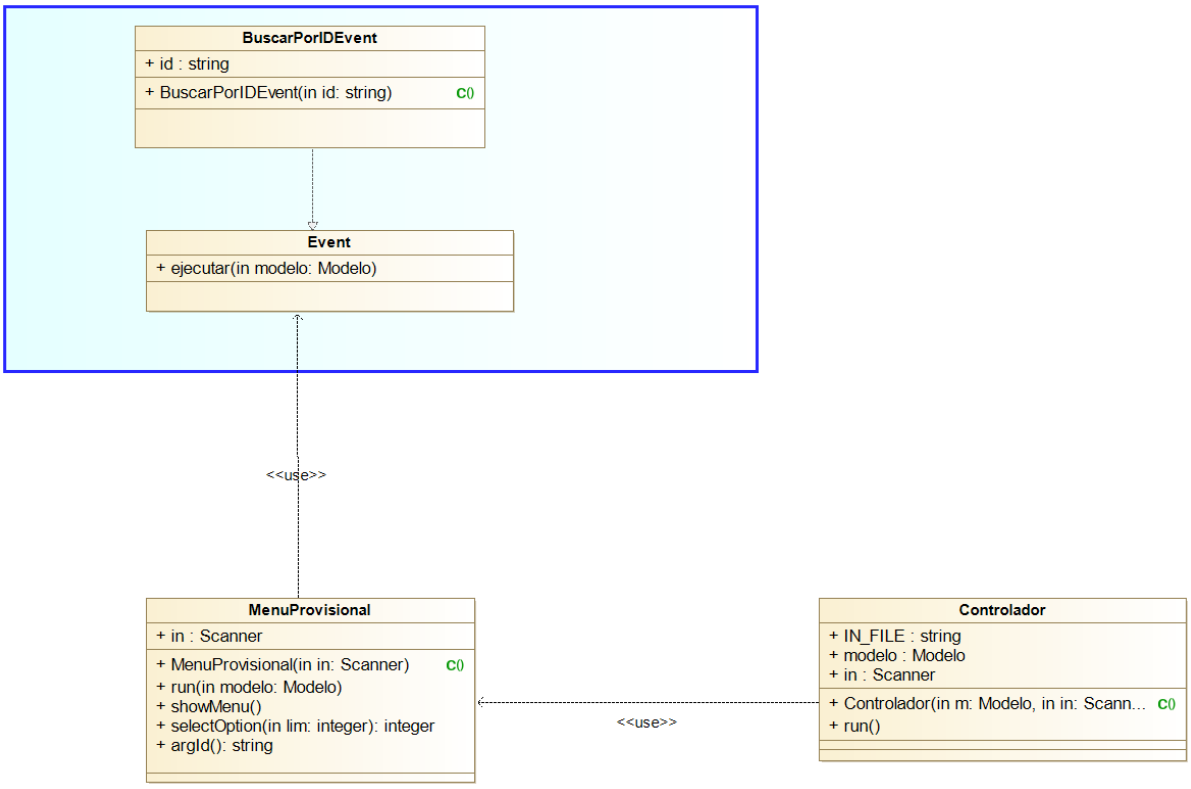




Descripción: Se eliminan todos los filtros que se han usado anteriormente para mostrar los elementos deseados por el usuario.

2.1.5 Acceder a un elemento



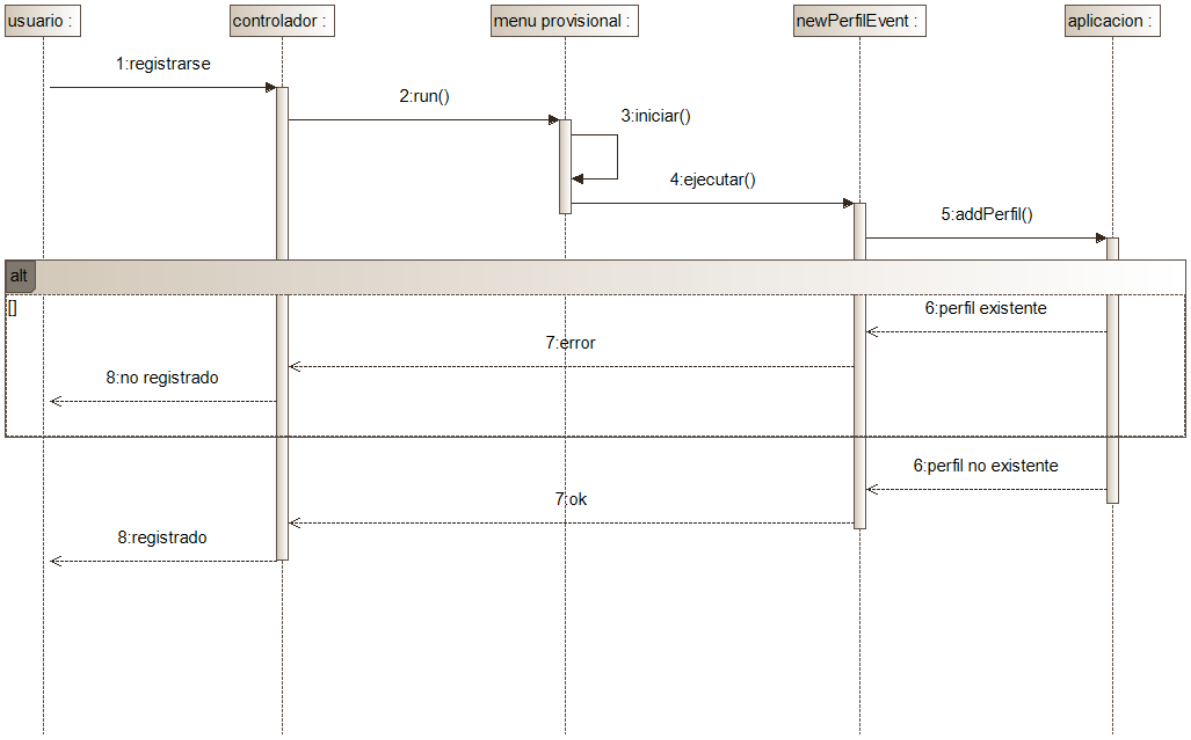


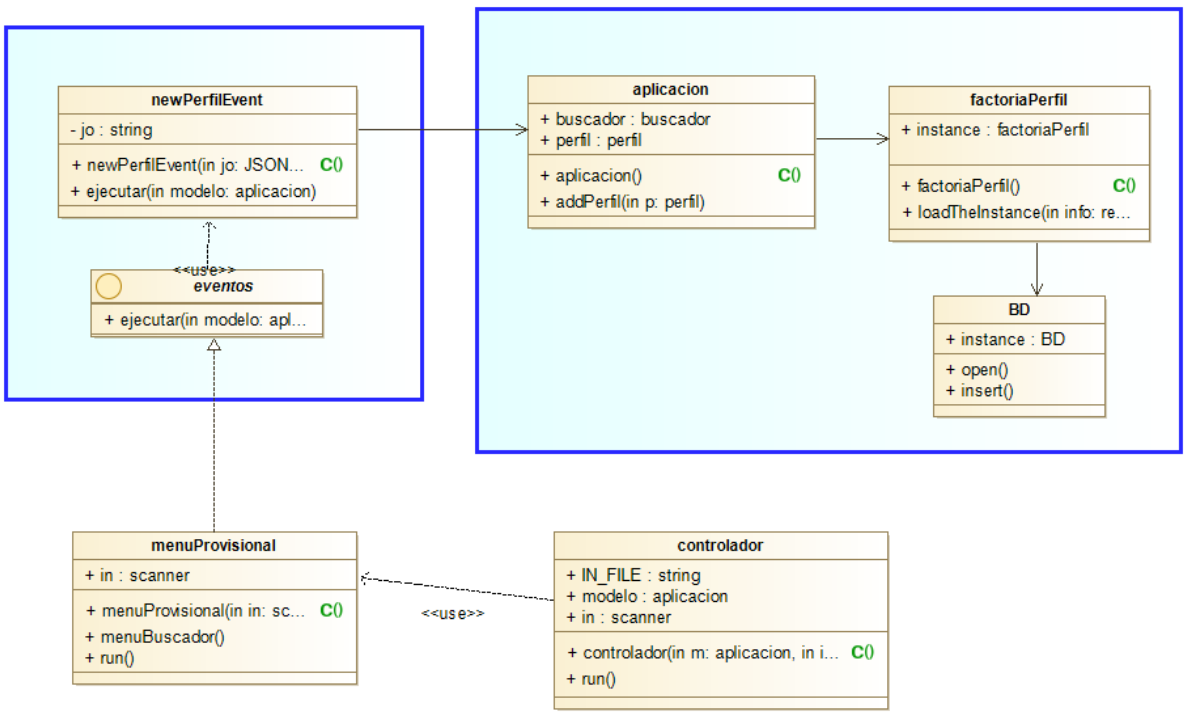
Descripción: Se muestran los detalles del elemento buscado por el id que desee el usuario.

2.2 Sprint 2

El uso de las siguientes funcionalidades se realiza mediante eventos que siguen el patrón Command.
La nueva base de datos integrada utiliza el patrón singleton, de forma que se asegura mantener la conexión establecida durante la ejecución desde el primer uso.

2.2.1 Registrarse

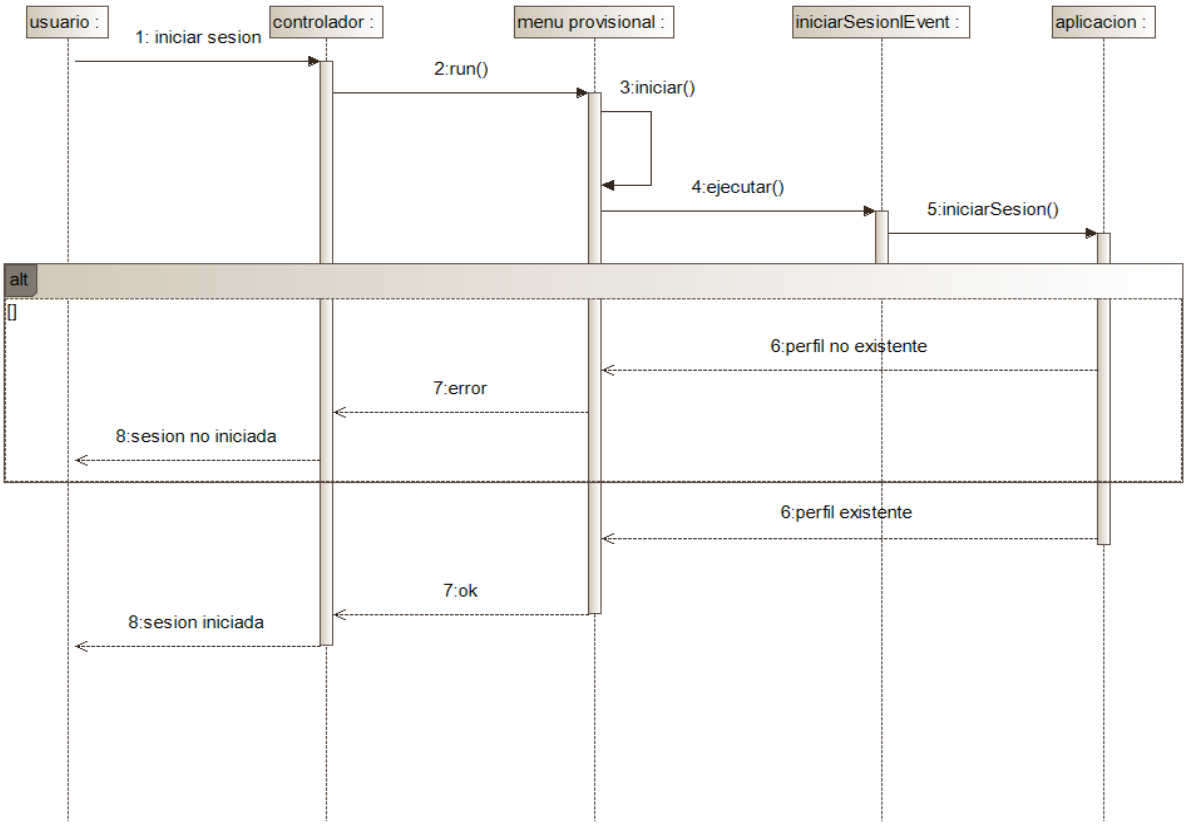


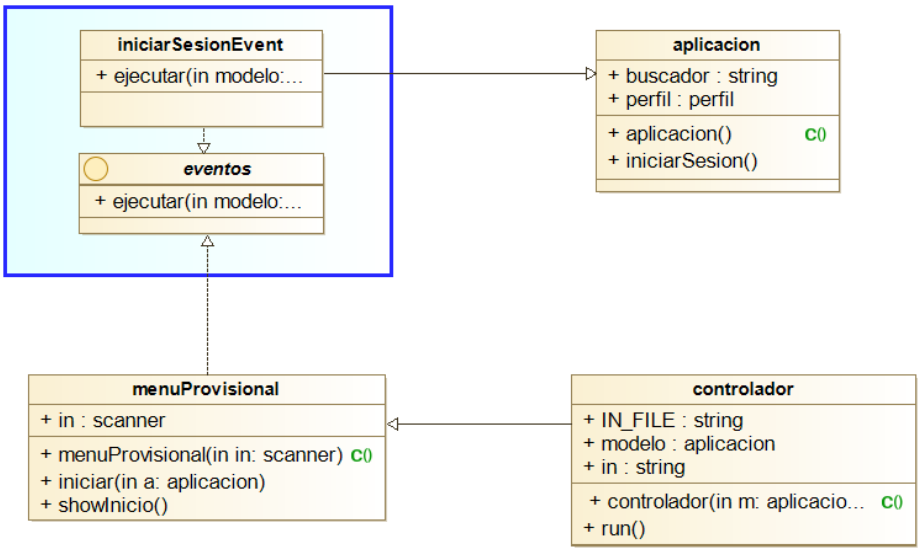


Para registrar un nuevo perfil se introducen y comprueban los datos del usuario, de forma que no se repitan correos entre perfiles (es la clave primaria). El resto de datos no tienen restricción más allá del tamaño.

La factoría de perfiles sigue el patrón singleton y factory method.

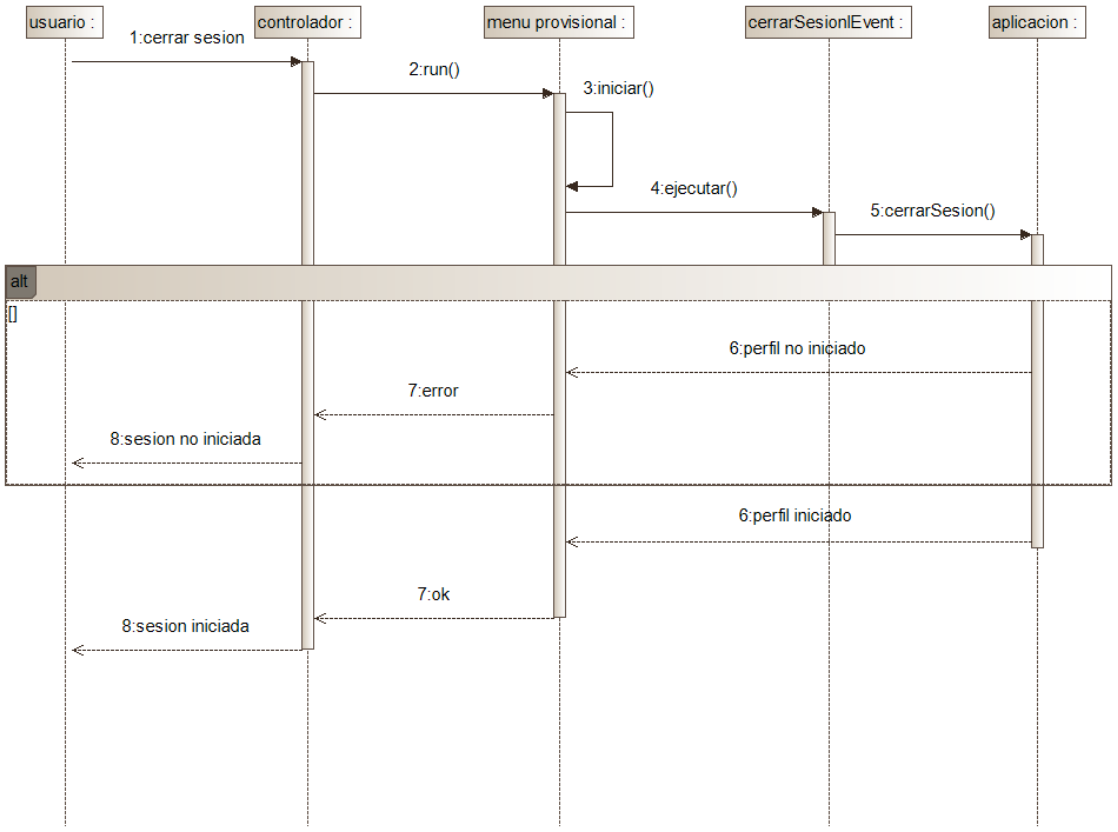
2.2.2 Iniciar Sesión

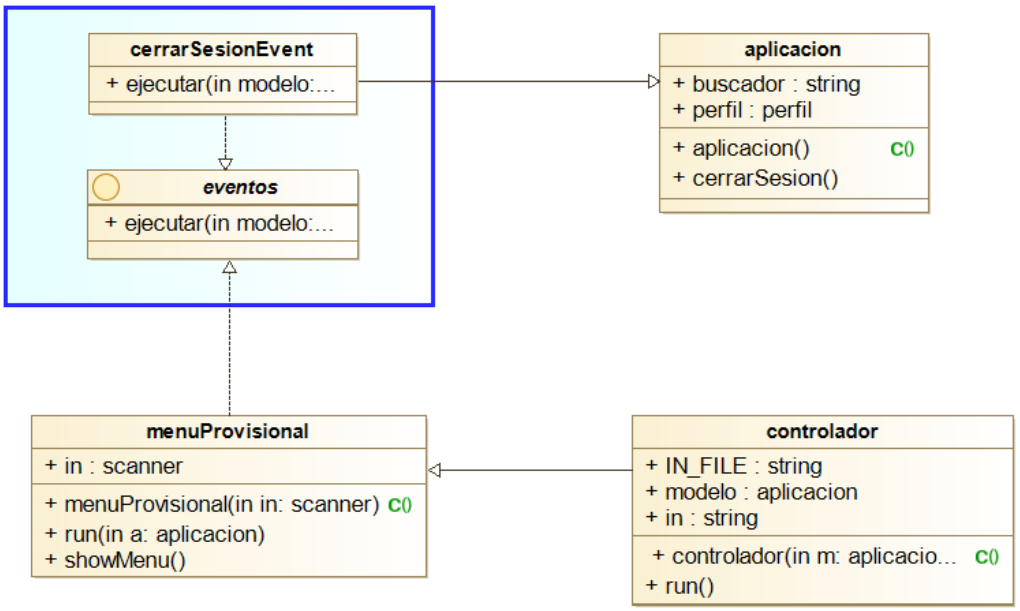




Para iniciar sesión se introduce el correo y contraseña del usuario, y en caso de existir su perfil y coincidir la contraseña, la aplicación lo almacena como perfil activo.

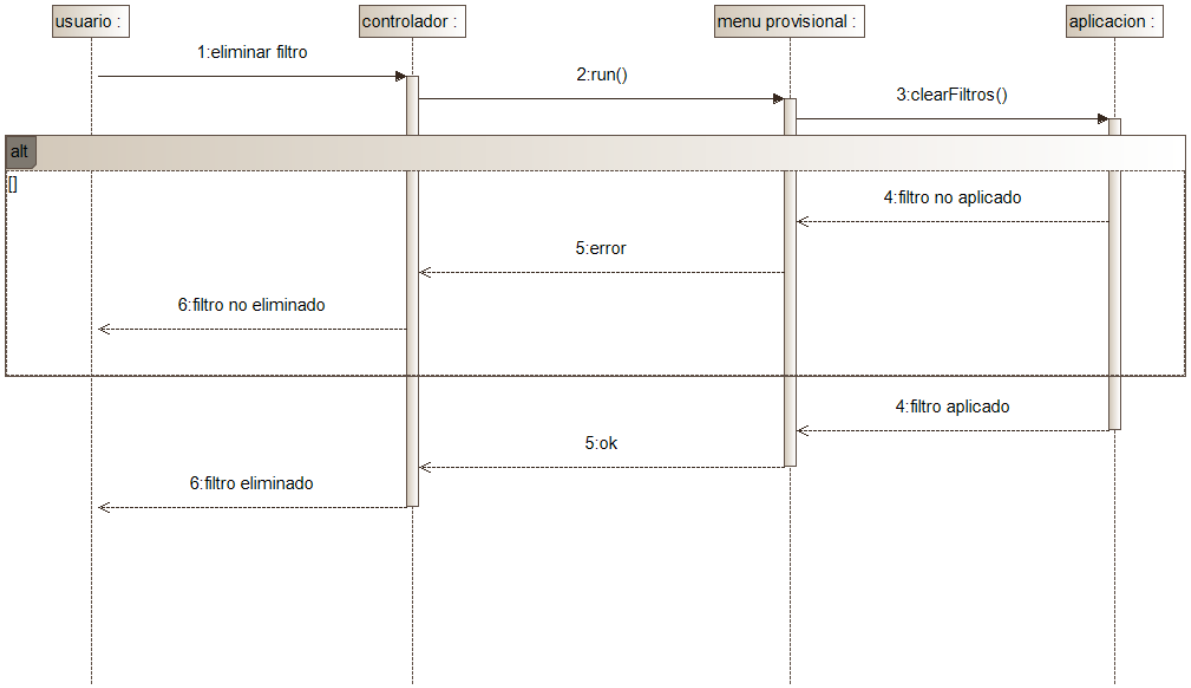
2.2.3 Cerrar Sesión

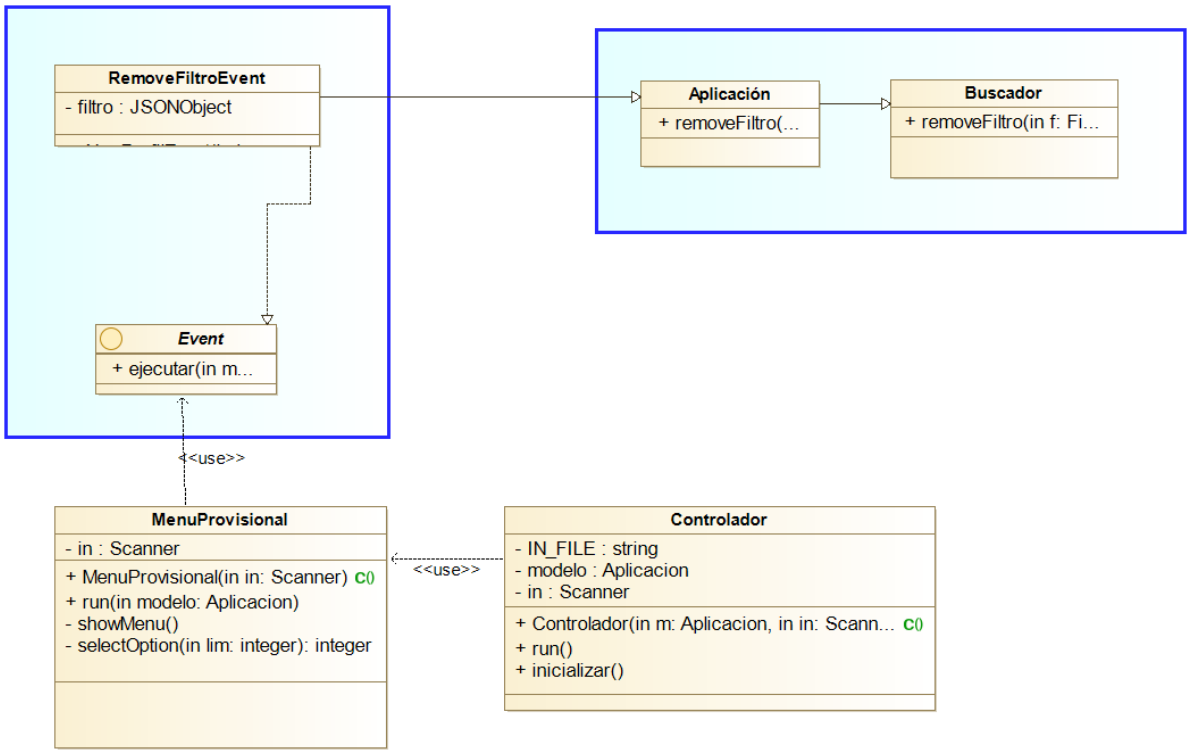




Habiendo un perfil activo en la aplicación, al cerrar sesión este se anula y se deja de tener acceso a él.

2.2.4 Eliminar Filtros

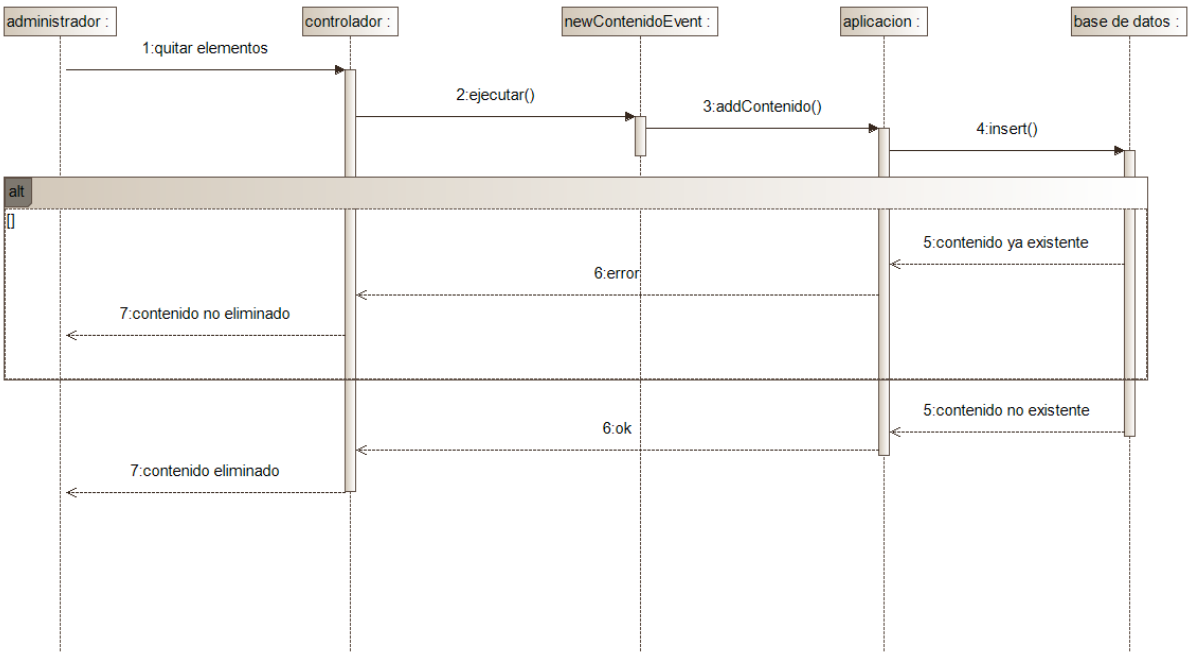


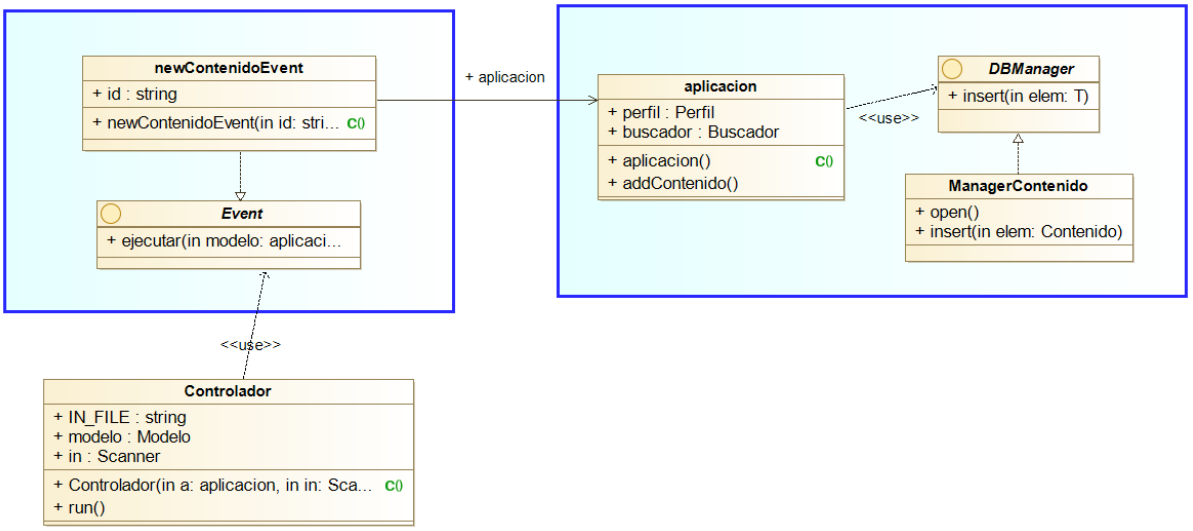


Al seleccionar un filtro a eliminar, se eliminan todos aquellos filtros equivalentes al seleccionado que estuvieran aplicados en el buscador.

Los filtros siguen un patrón de estrategias.

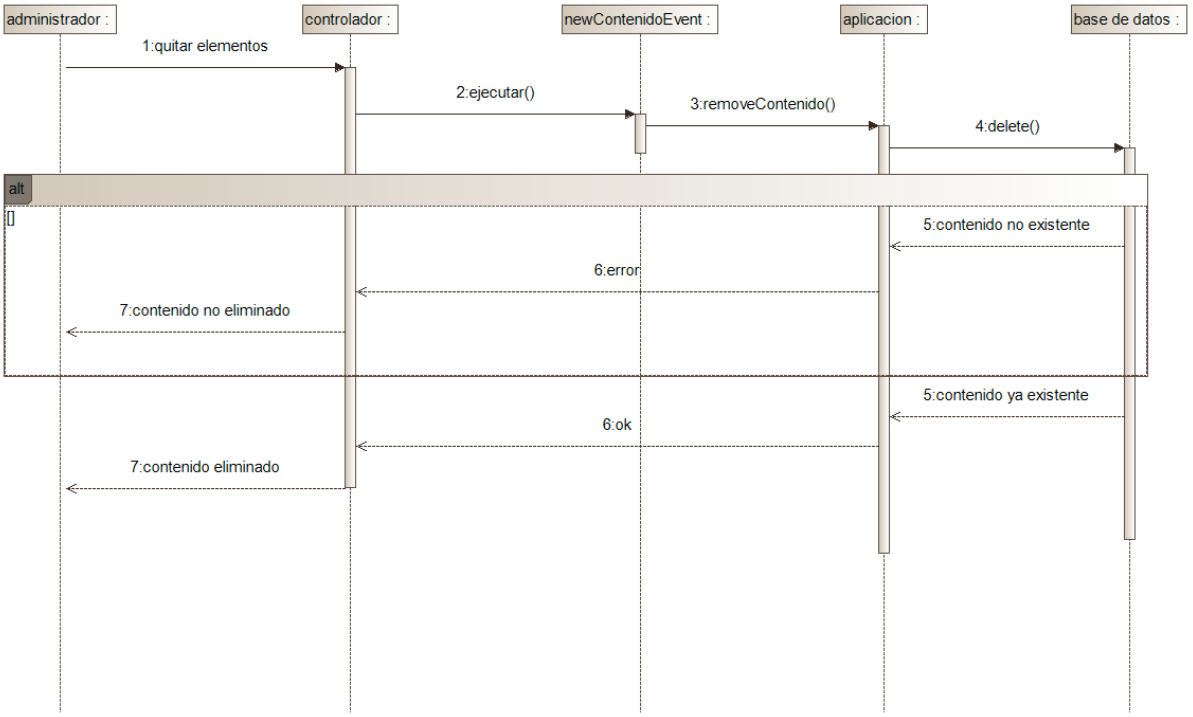
2.2.5 Introducir Elemento

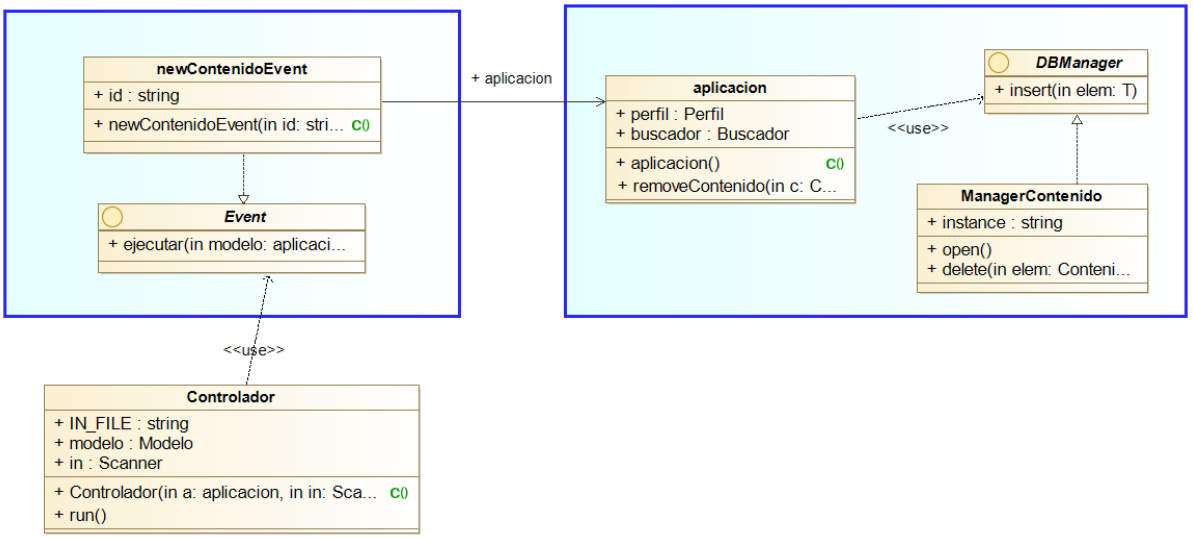




Los elementos ya no se añaden al iniciar la aplicación, al disponer de una base de datos. A su vez, se pueden introducir contenidos desde el menú accediendo como administrador.

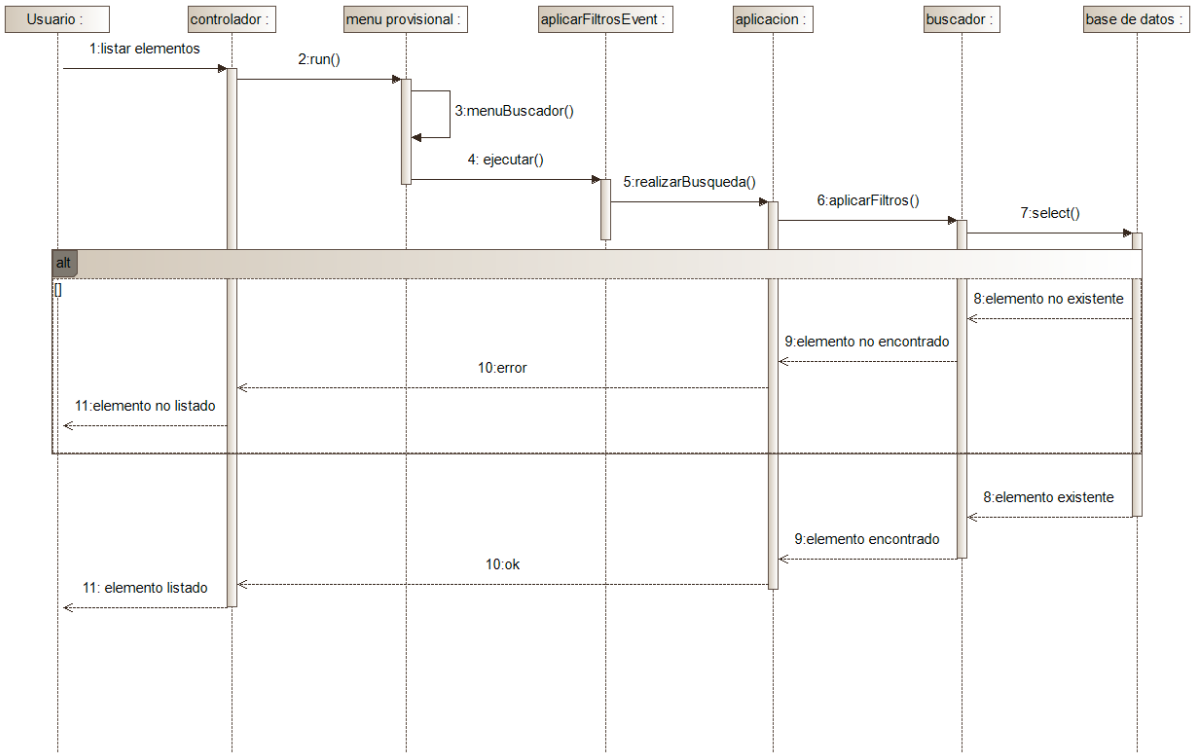
2.2.6 Eliminar Elemento

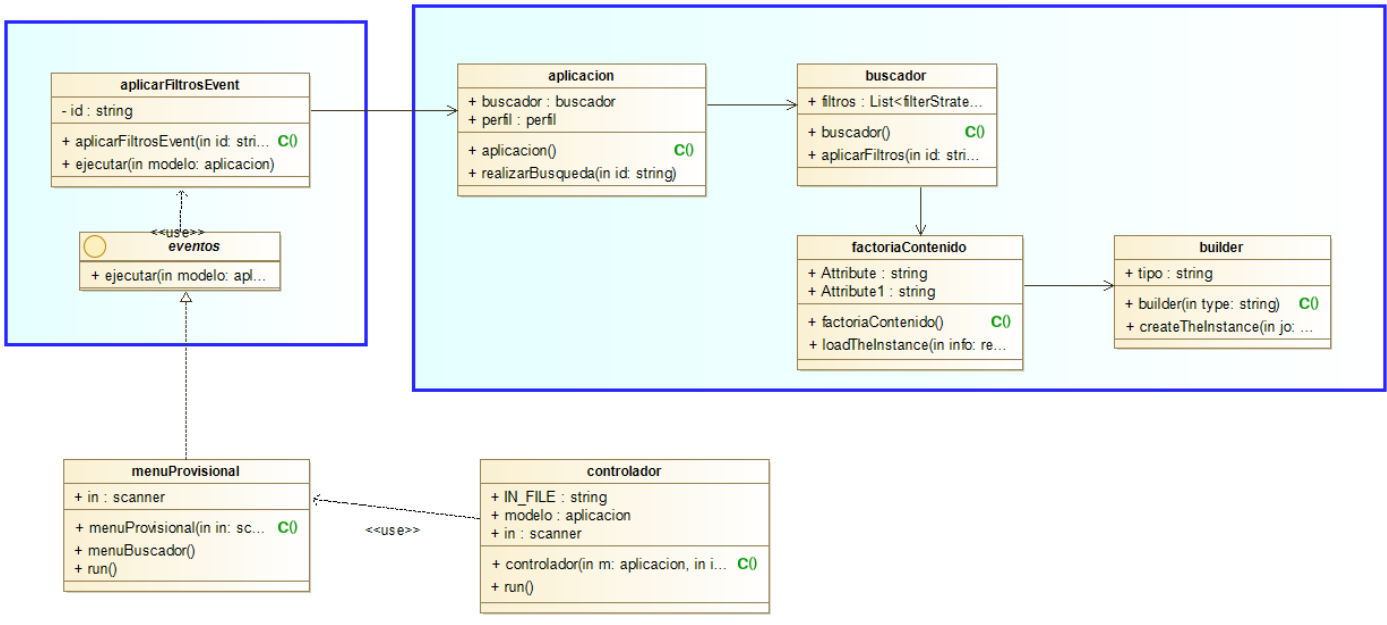




Después de acceder como administrador, se puede eliminar un elemento de la base de datos introduciendo su id (si existe).

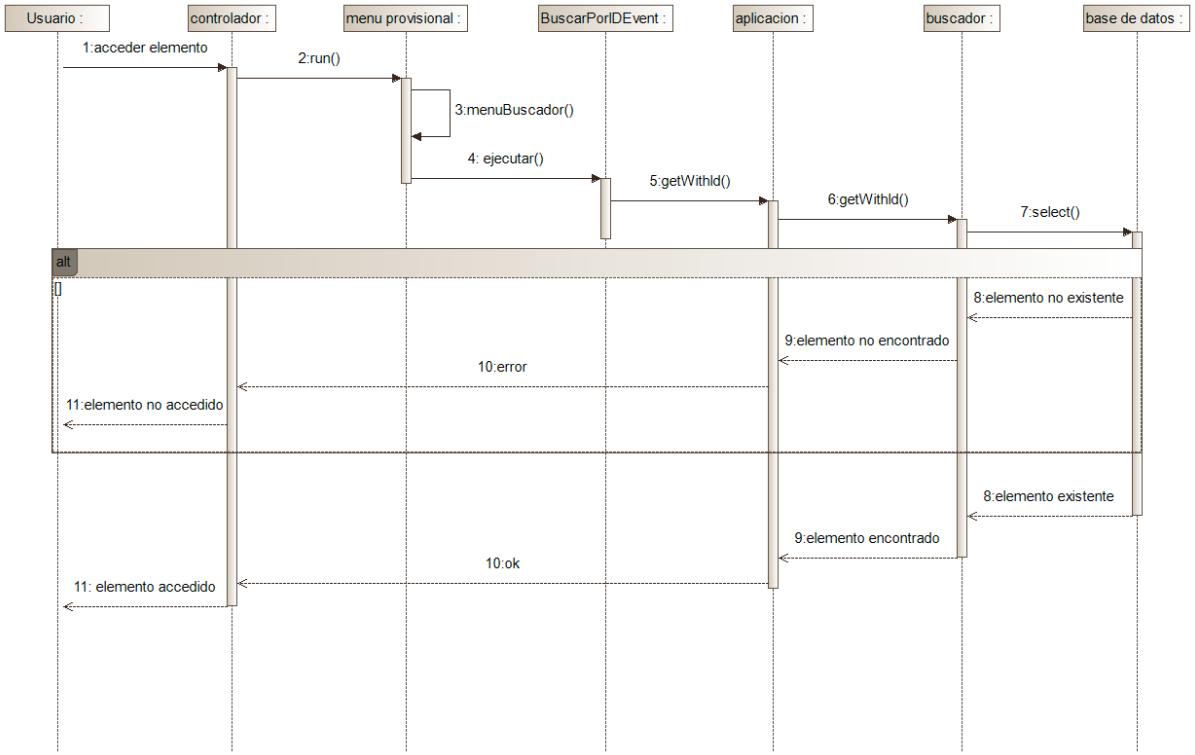
2.2.7 Listar elementos

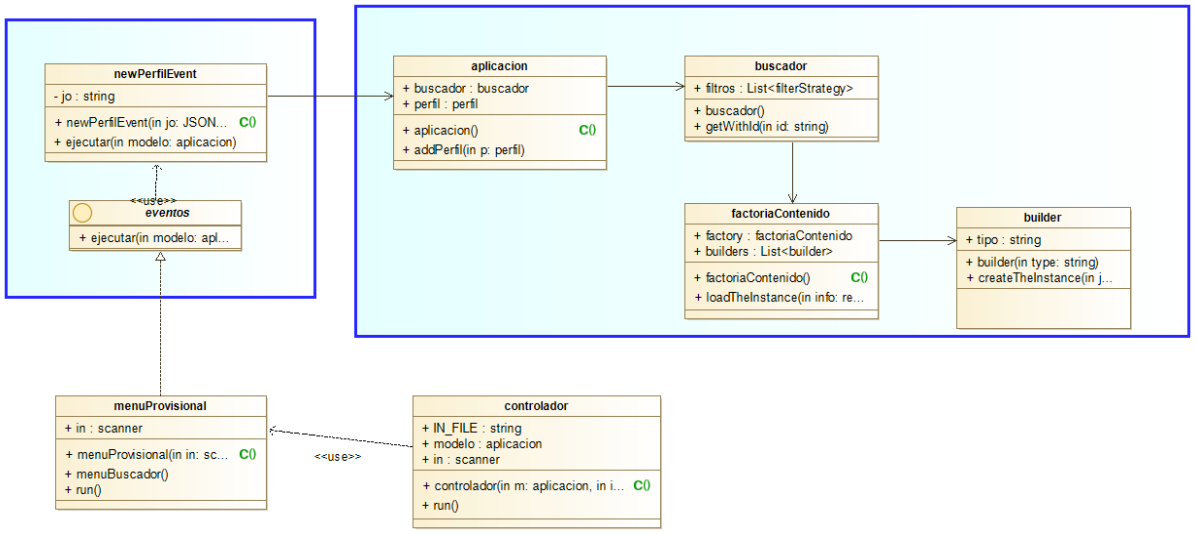




Misma funcionalidad que en el anterior sprint, pero integrando la base de datos.
Además, para construir las consultas que utiliza el buscador se usa una clase externa que funciona como un Builder de srings (patrón builder).

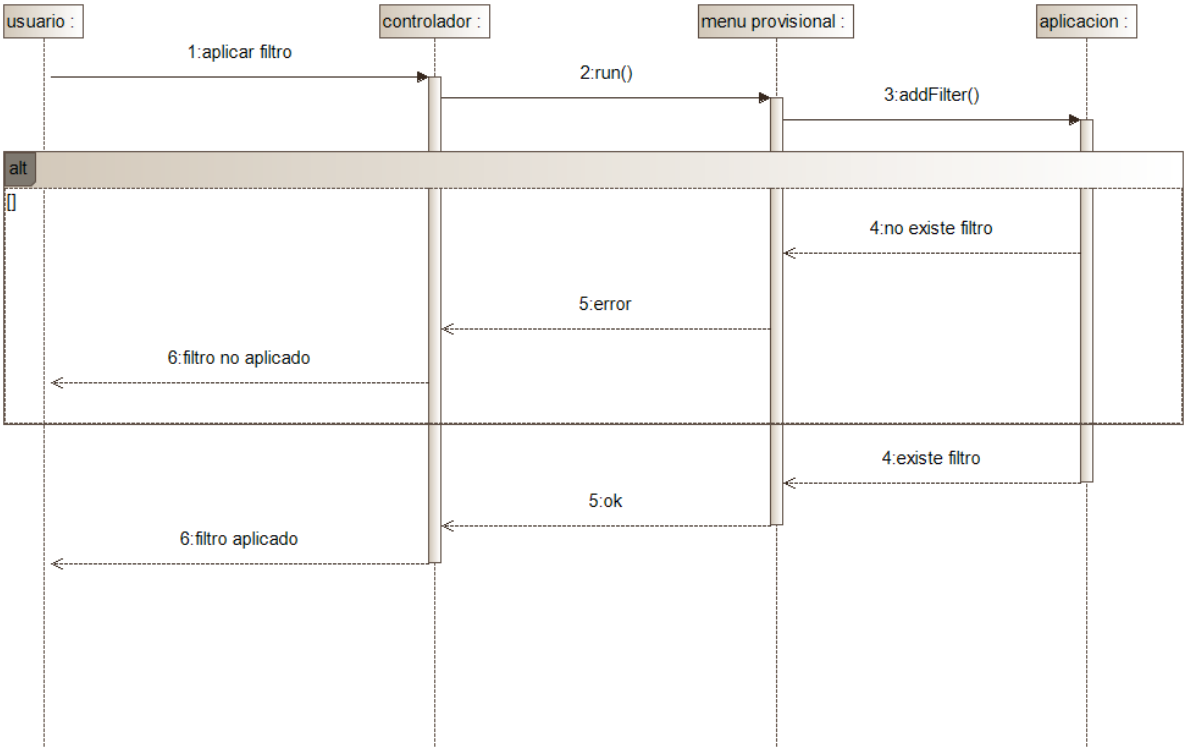
2.2.8 Acceder a un elemento

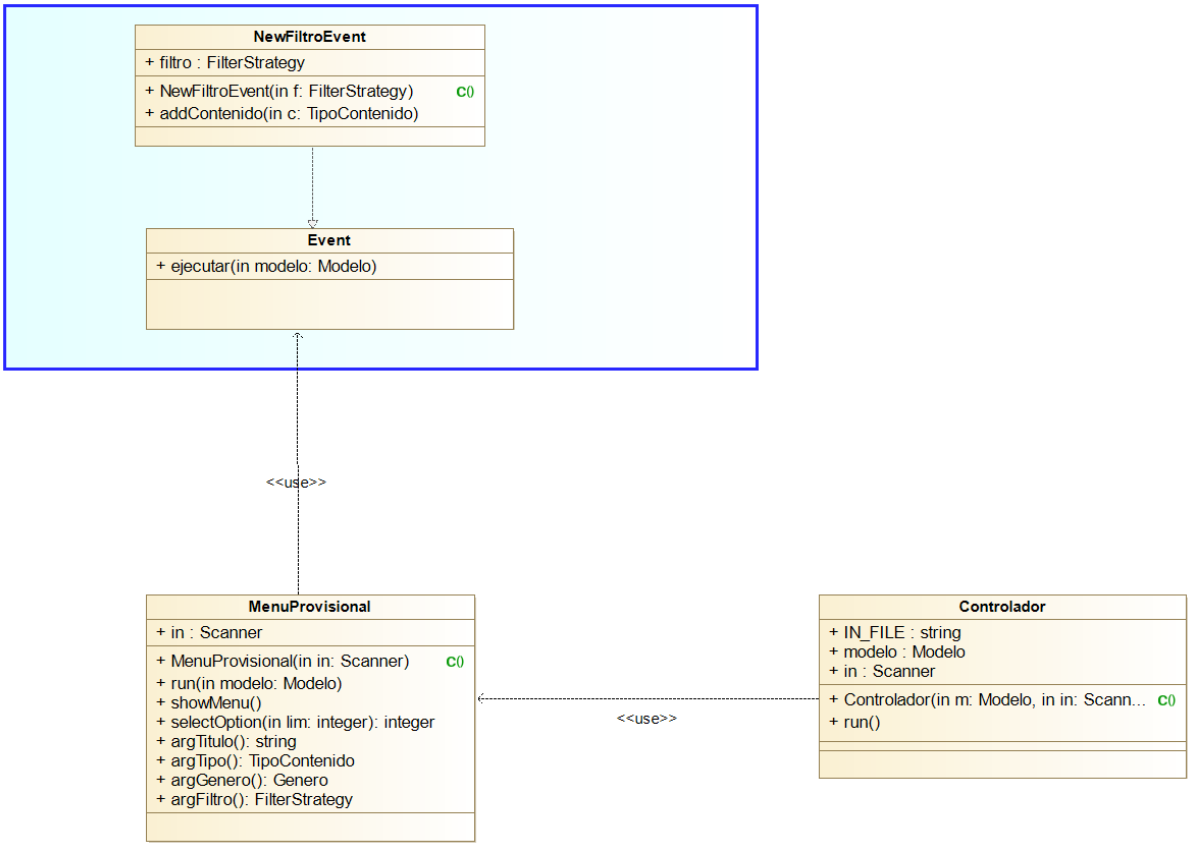




Misma funcionalidad que en el anterior sprint, pero integrando la base de datos.

2.2.9 Aplicar un filtro



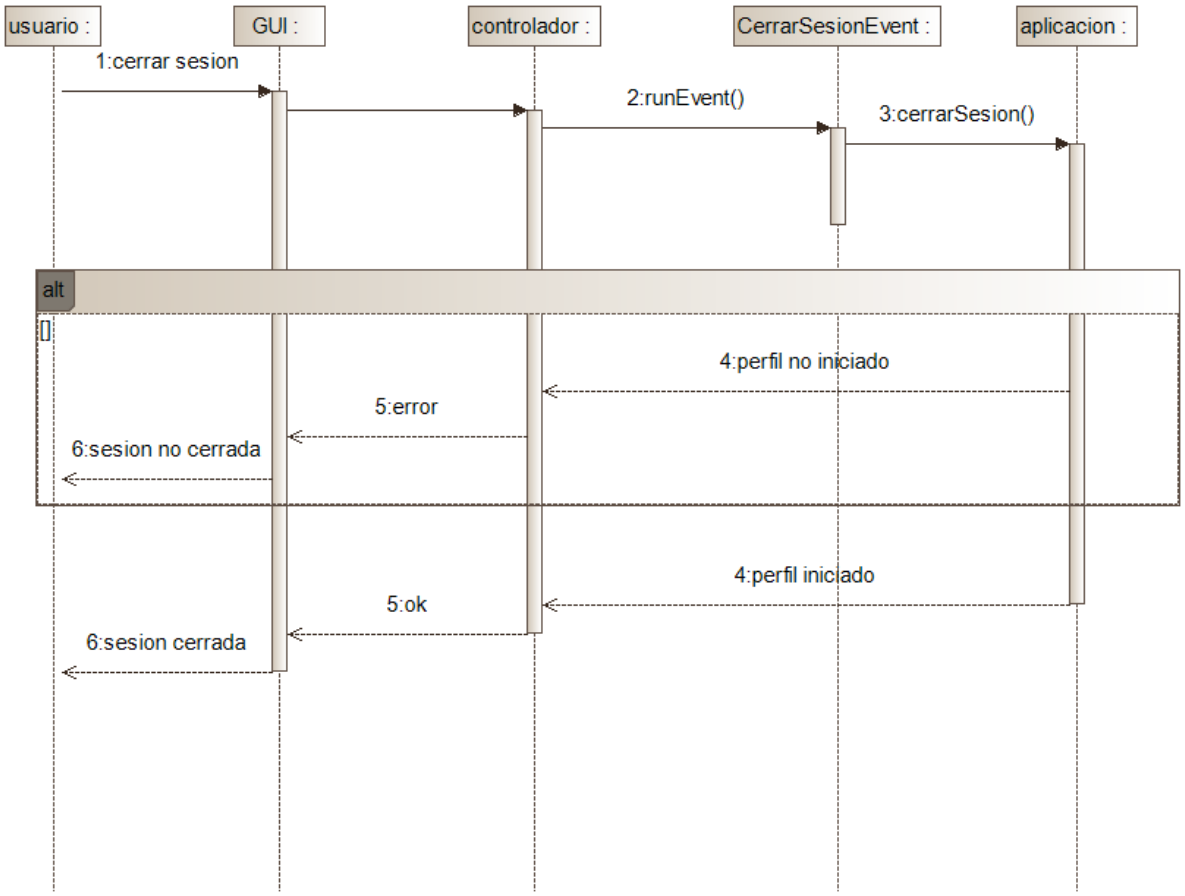


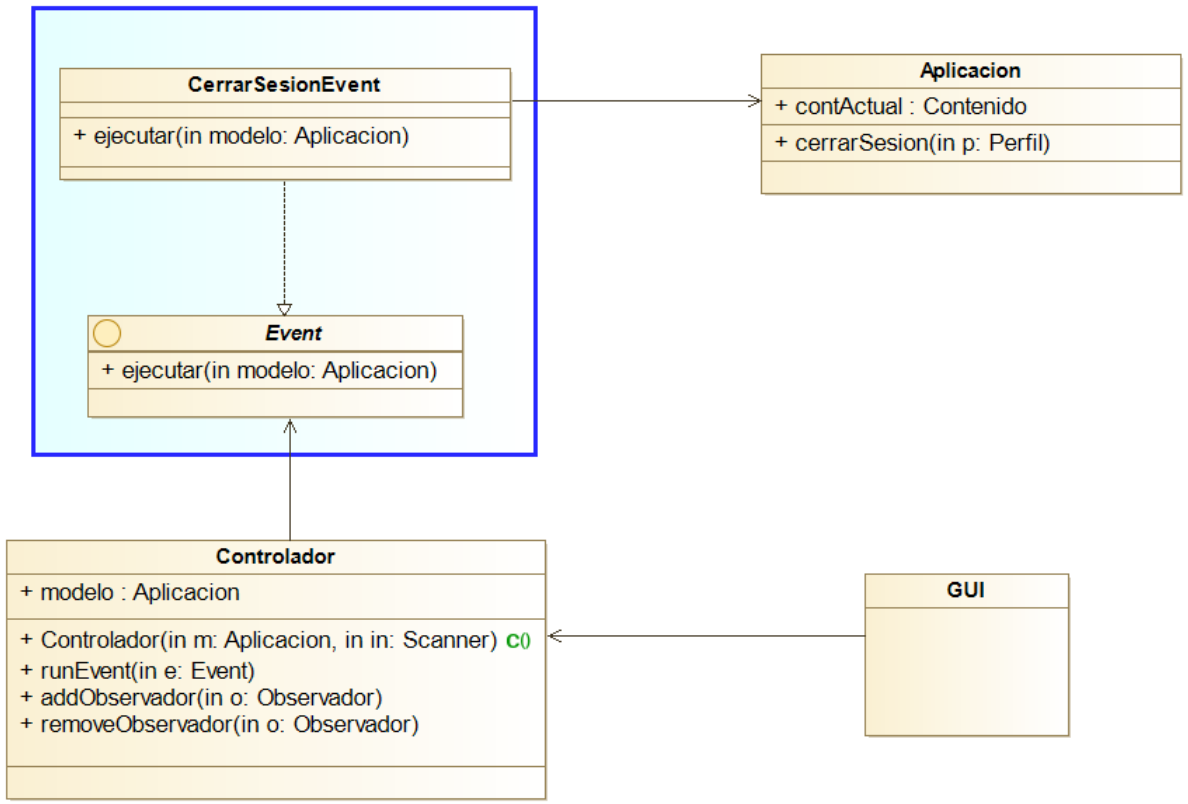
Misma funcionalidad que en el anterior sprint, pero integrando la base de datos.

2.3 Sprint 3

El uso de las siguientes funcionalidades se realiza mediante eventos que siguen el patrón Command.
La nueva base de datos integrada utiliza el patrón singleton, de forma que se asegura mantener la conexión establecida durante la ejecución desde el primer uso.
La nueva interfaz gráfica recibe información de la aplicación mediante el patrón Observer, de modo que sólo se actualice visualmente tras ciertos comandos o sucesos.

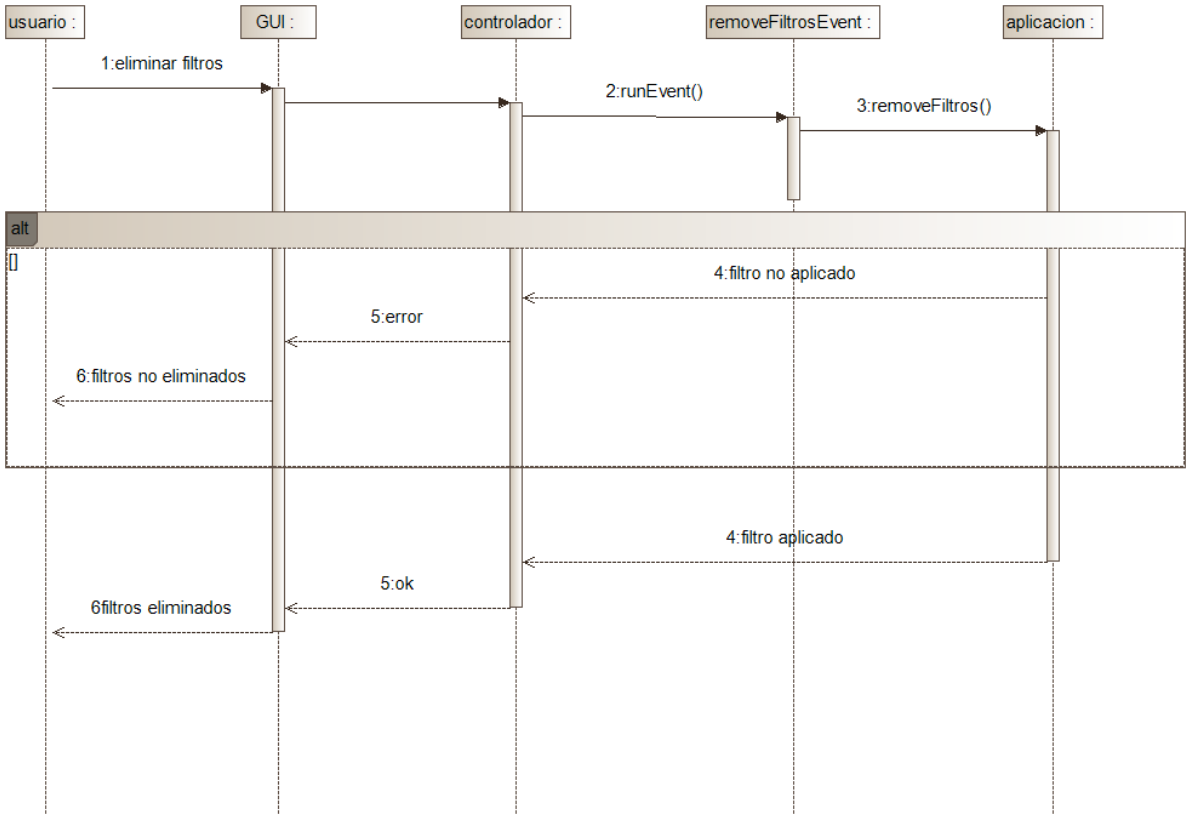
2.3.1 Cerrar sesión

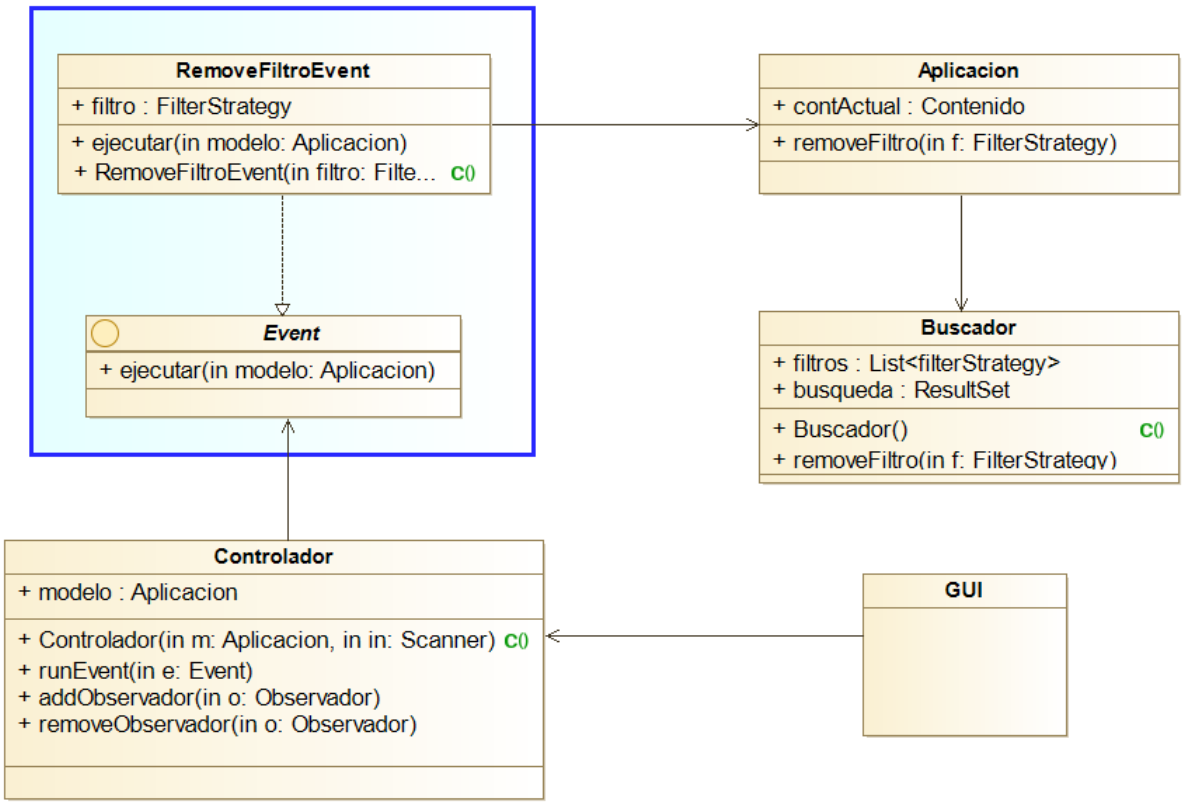




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

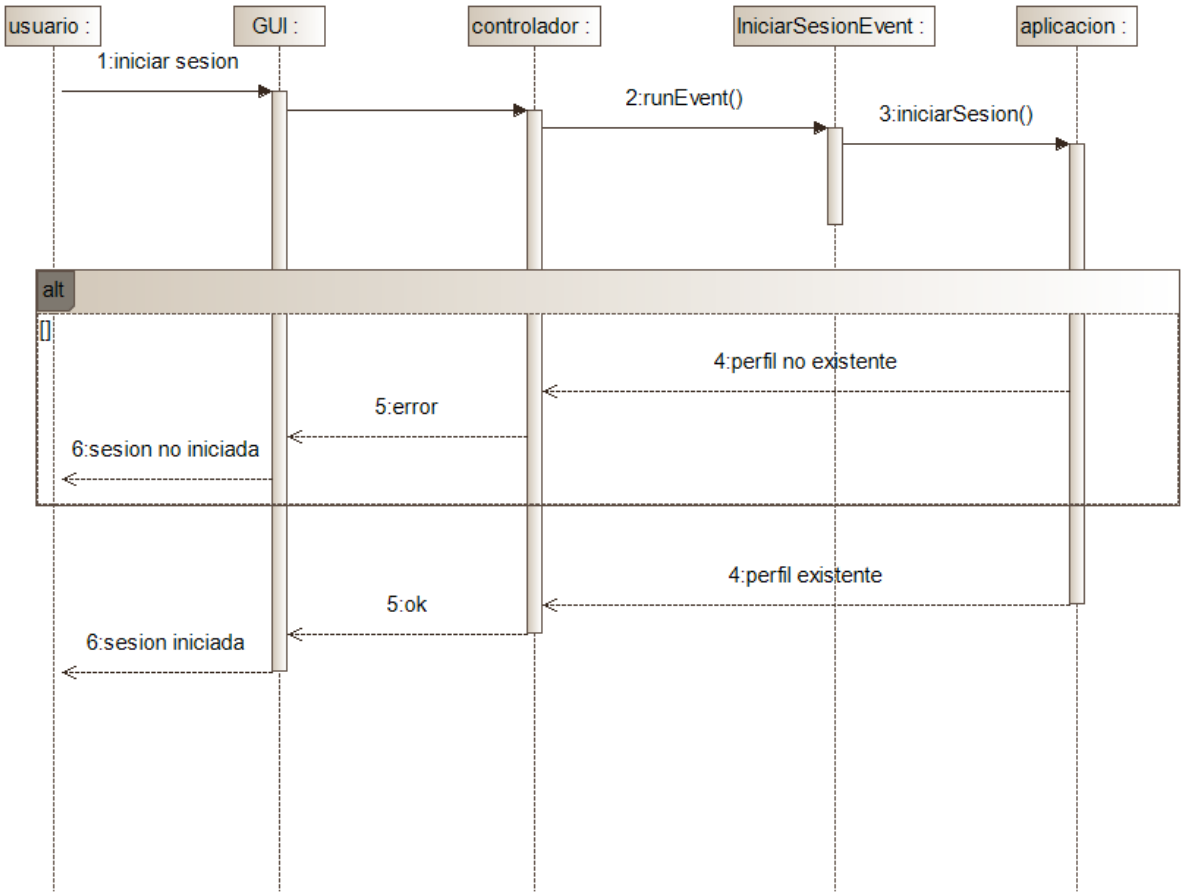
2.3.2 Eliminar filtro

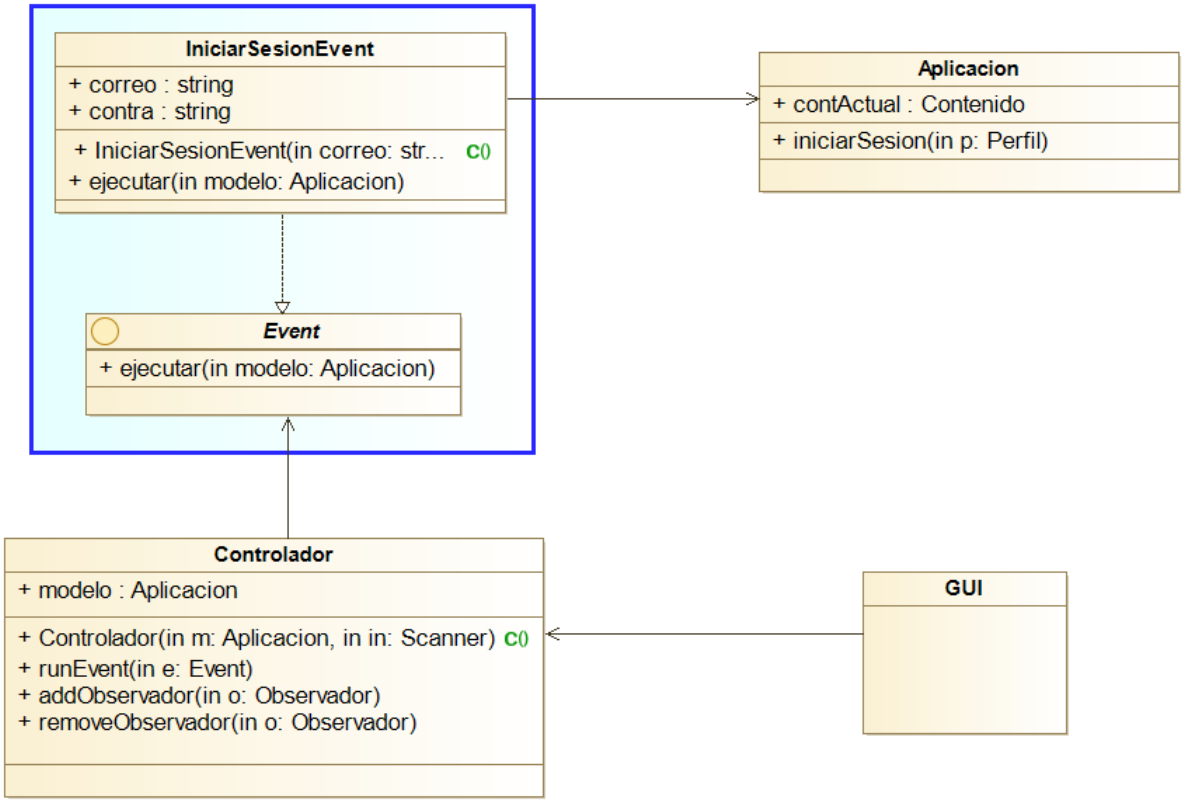




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

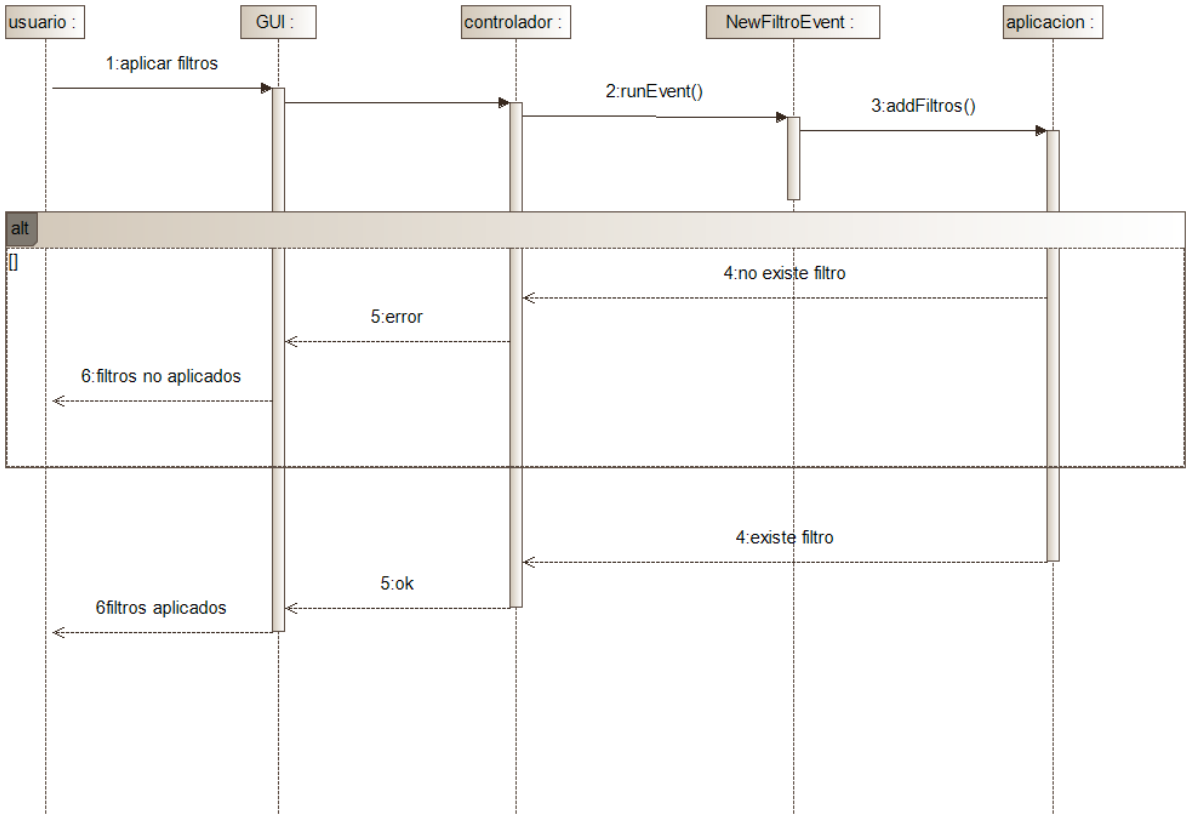
2.3.3 Iniciar sesión

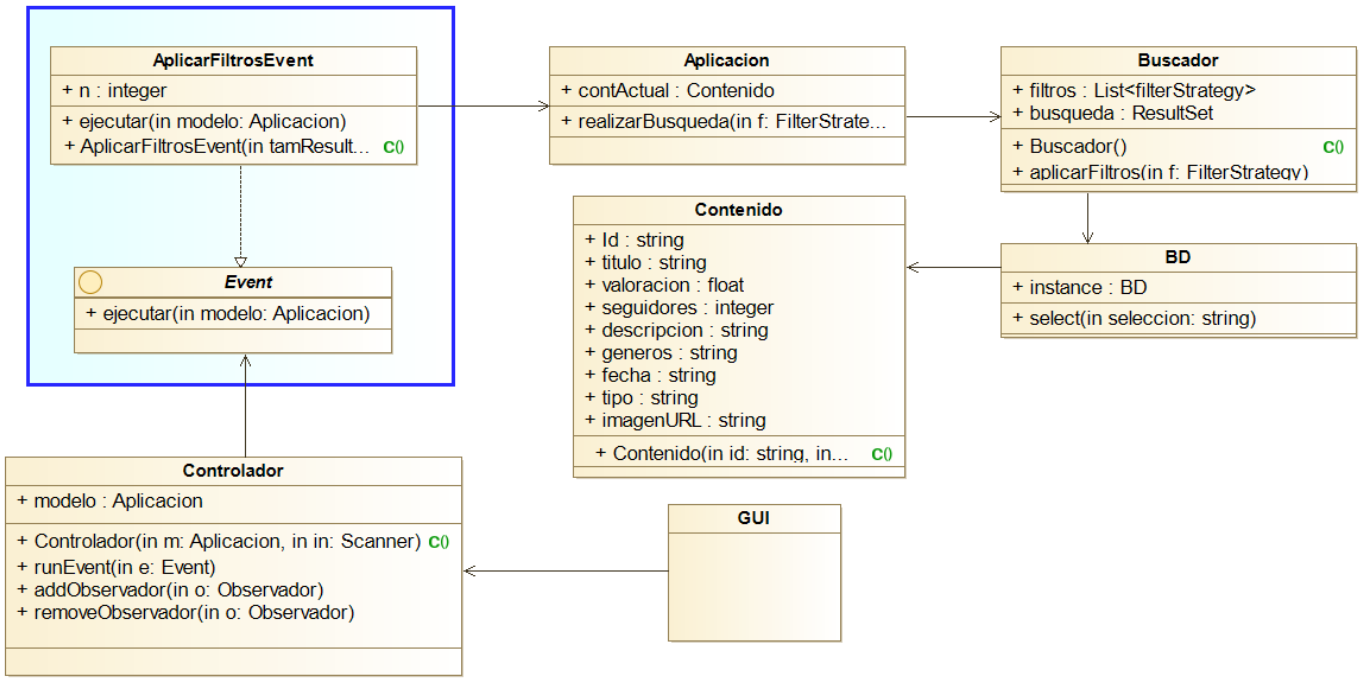




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

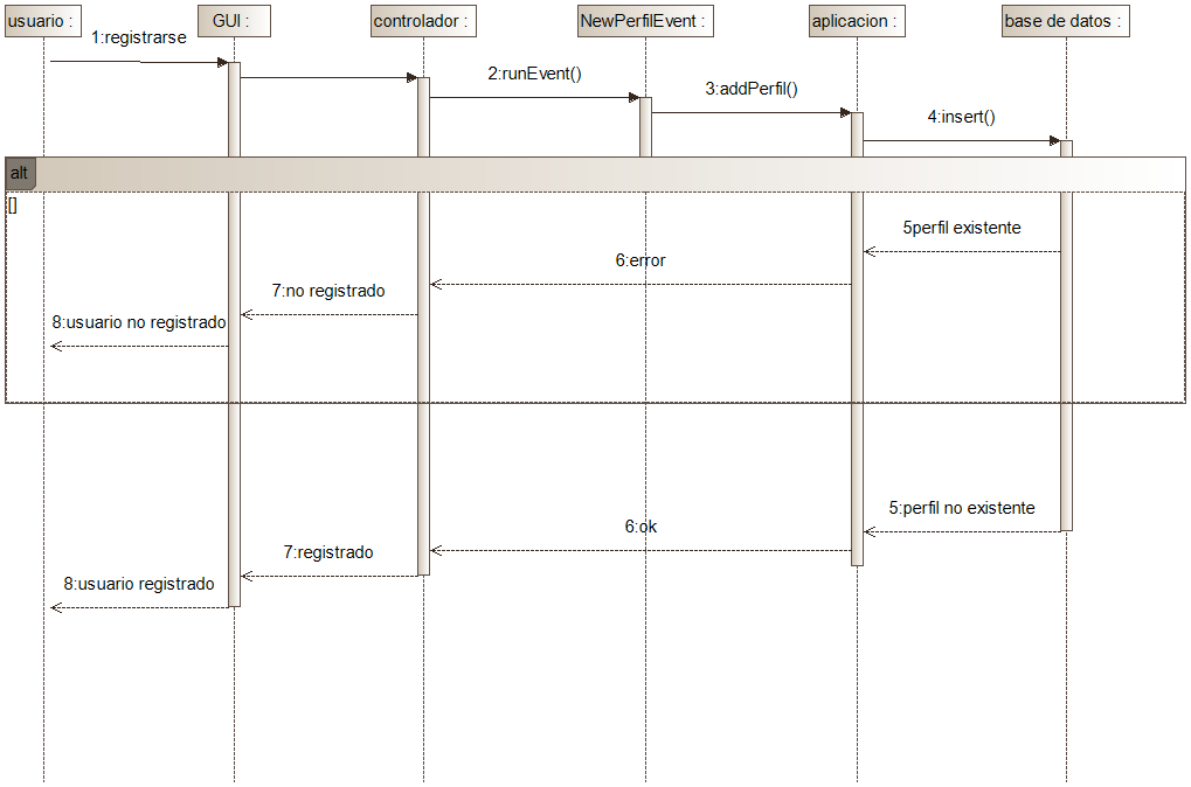
2.3.4 Aplicar filtro

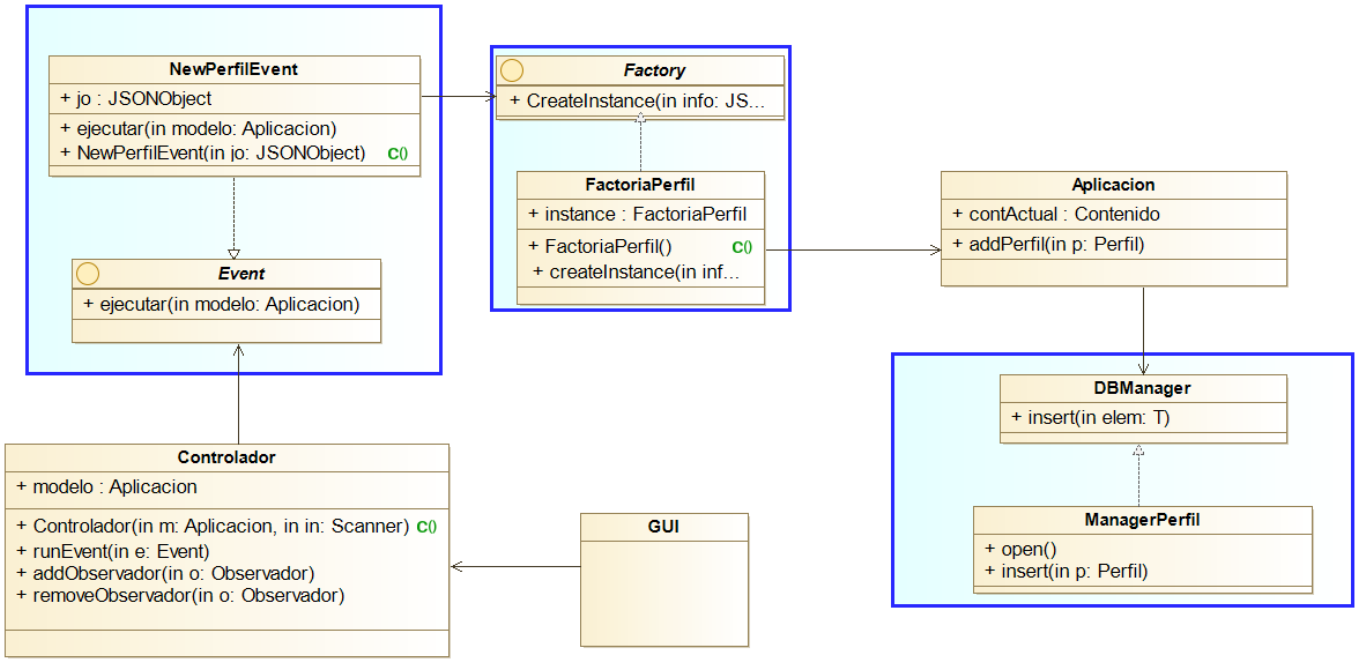




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

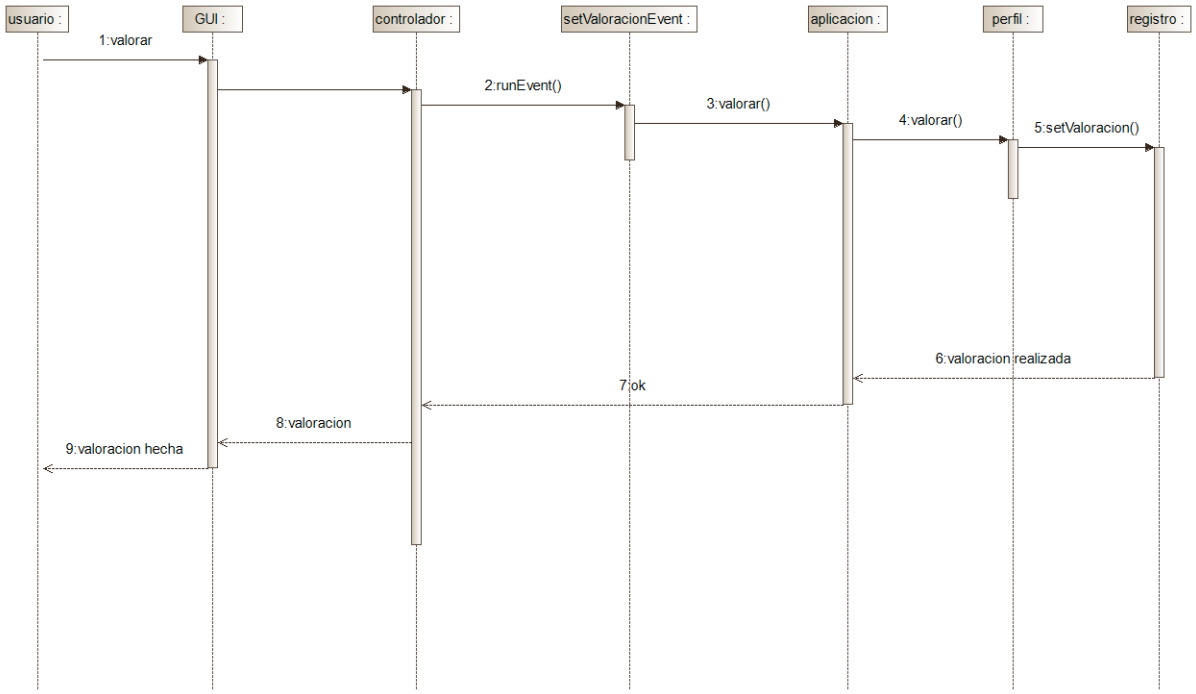
2.3.5 Registrarse

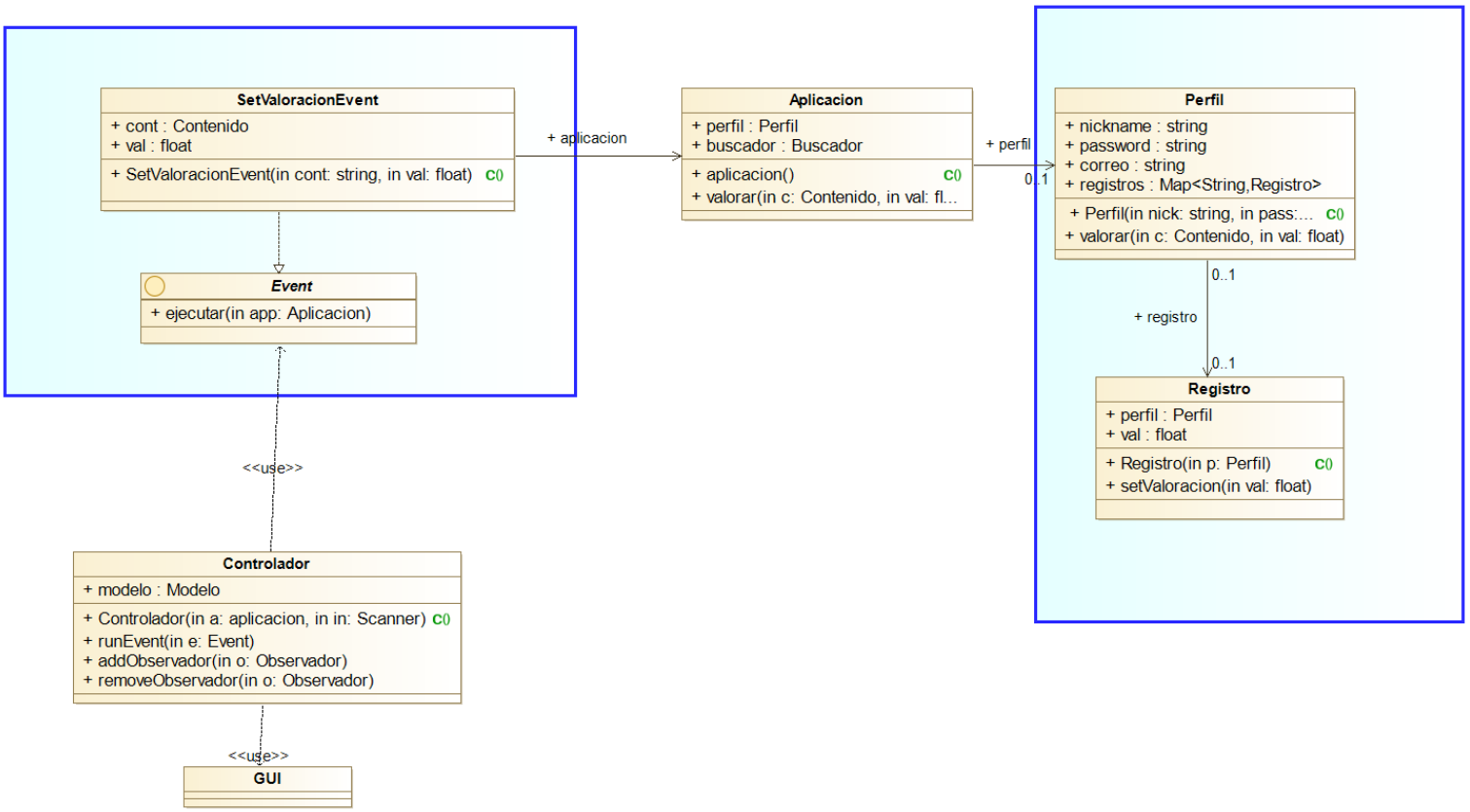




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

2.3.6 Valorar un elemento

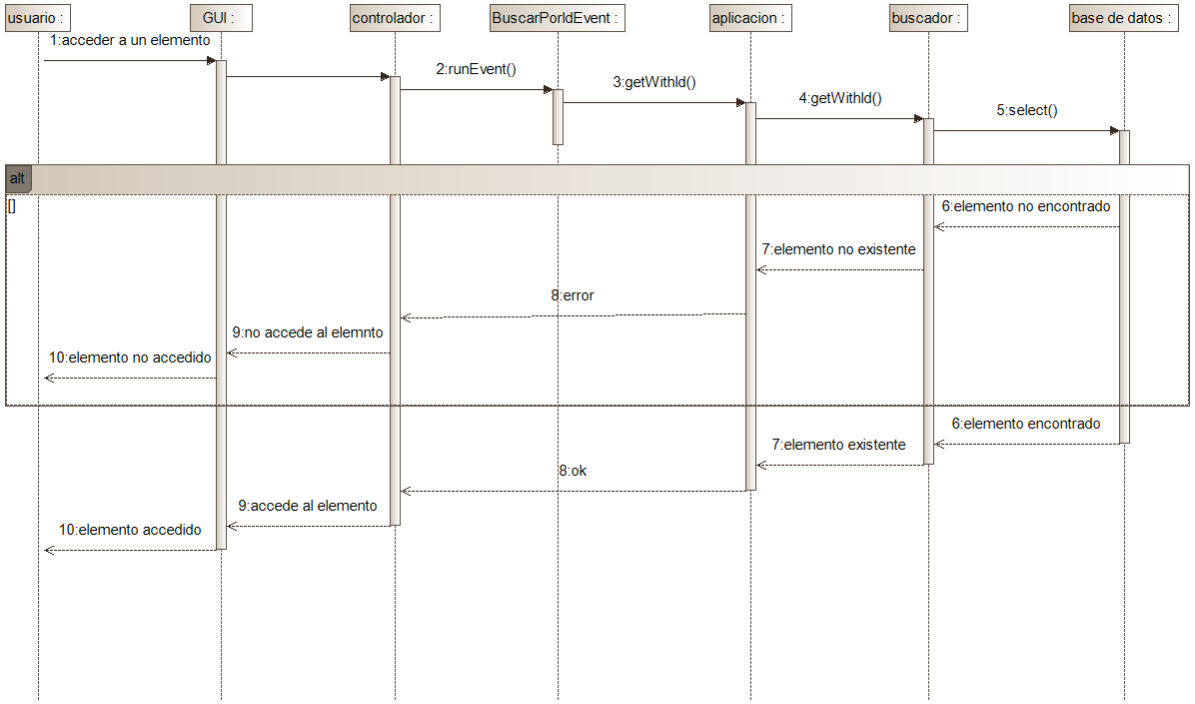


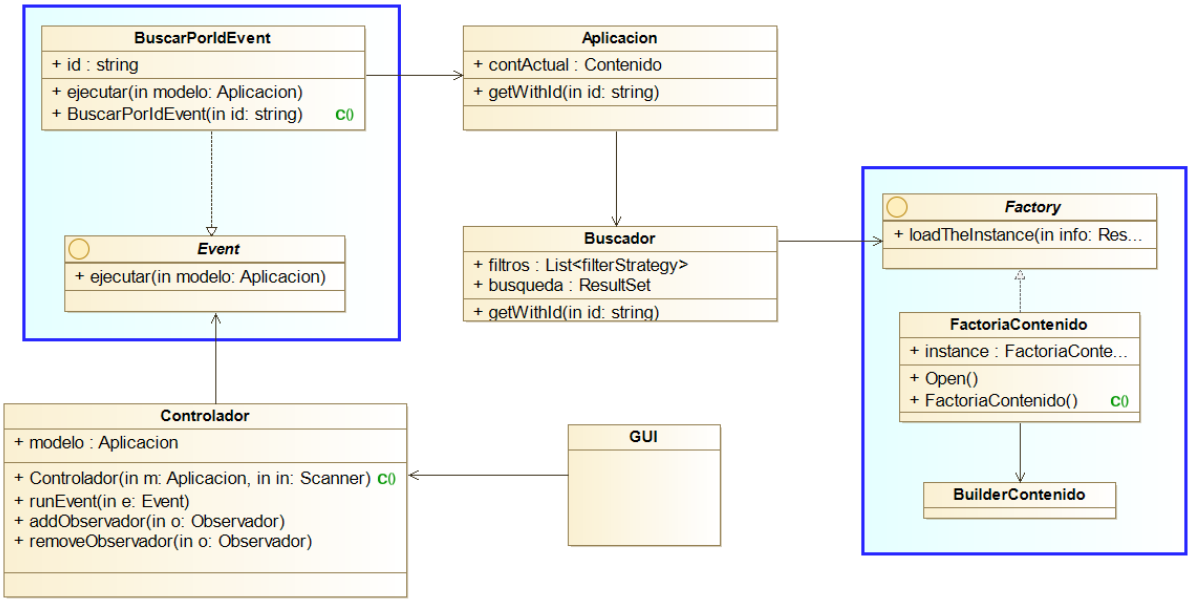


Para la valoración de un título, el usuario selecciona el título y la opción de valorar dándole así la nota que considere. En caso de no existir en el perfil un registro asociado al

contenido a tratar, se crea en una factoría y se inserta en la BD en el manager, ambas clases siguiendo el patrón Factory Method y Singleton.

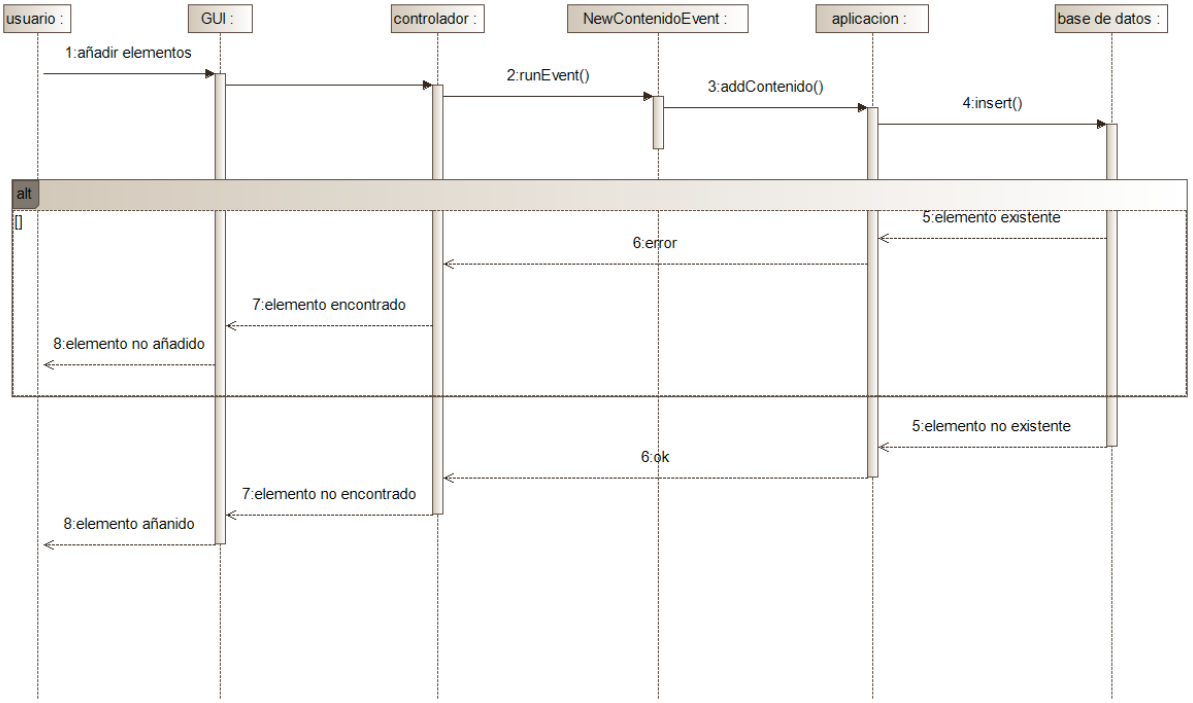
2.3.7 Acceder elemento

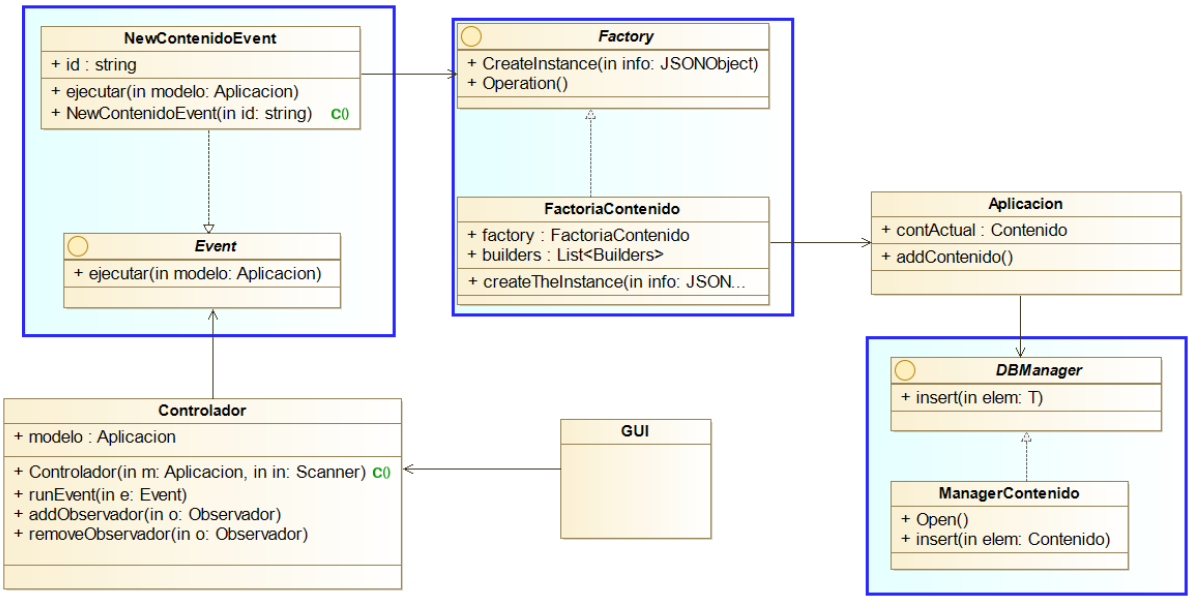




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

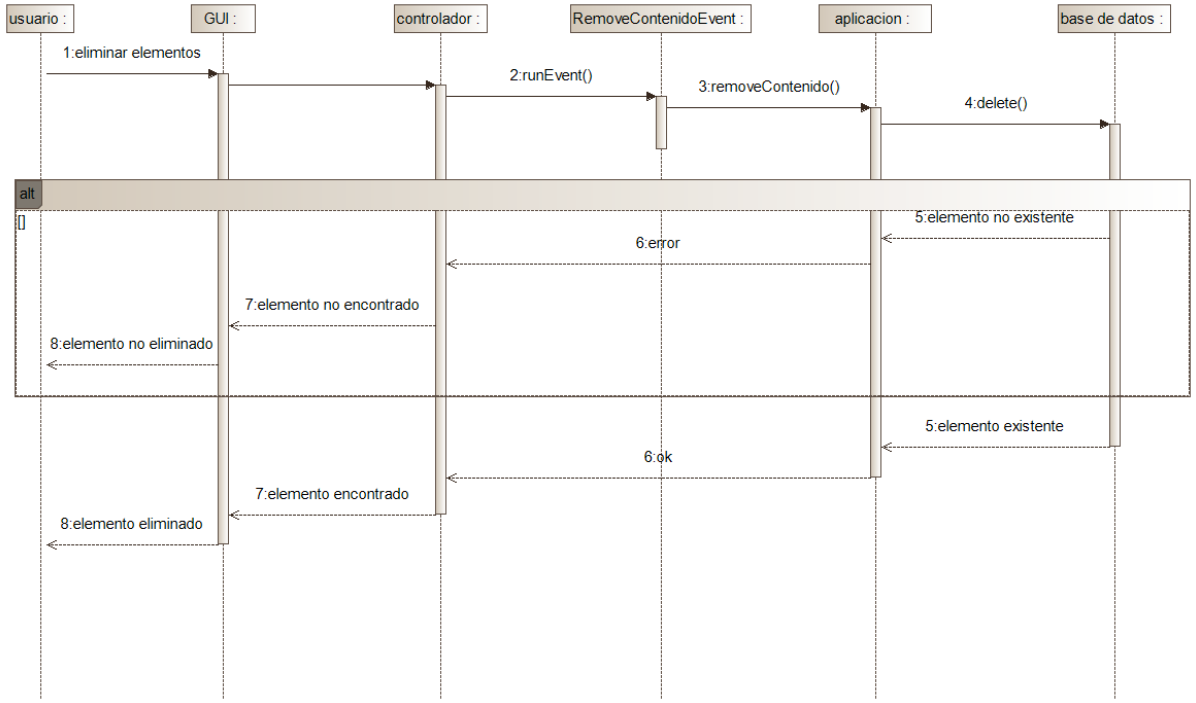
2.3.8 Introducir elementos

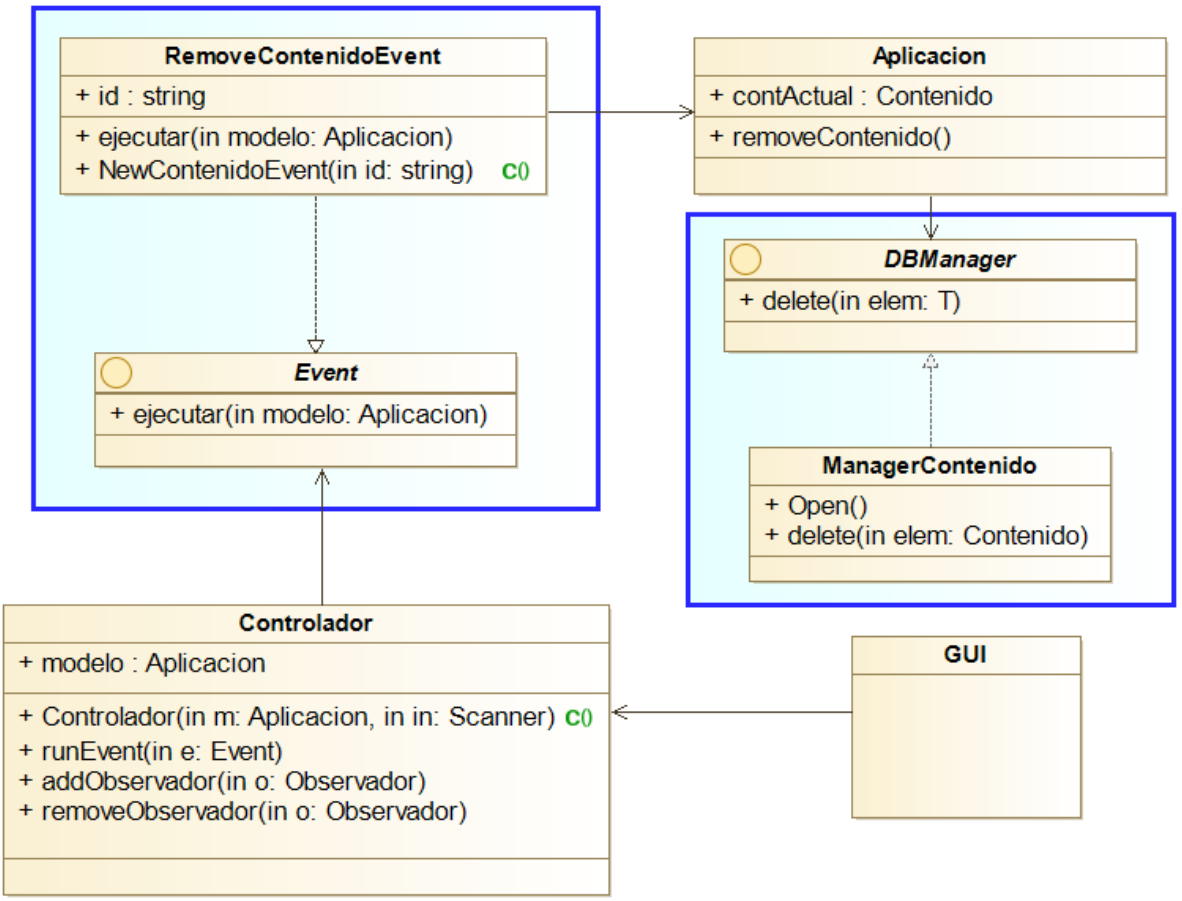




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

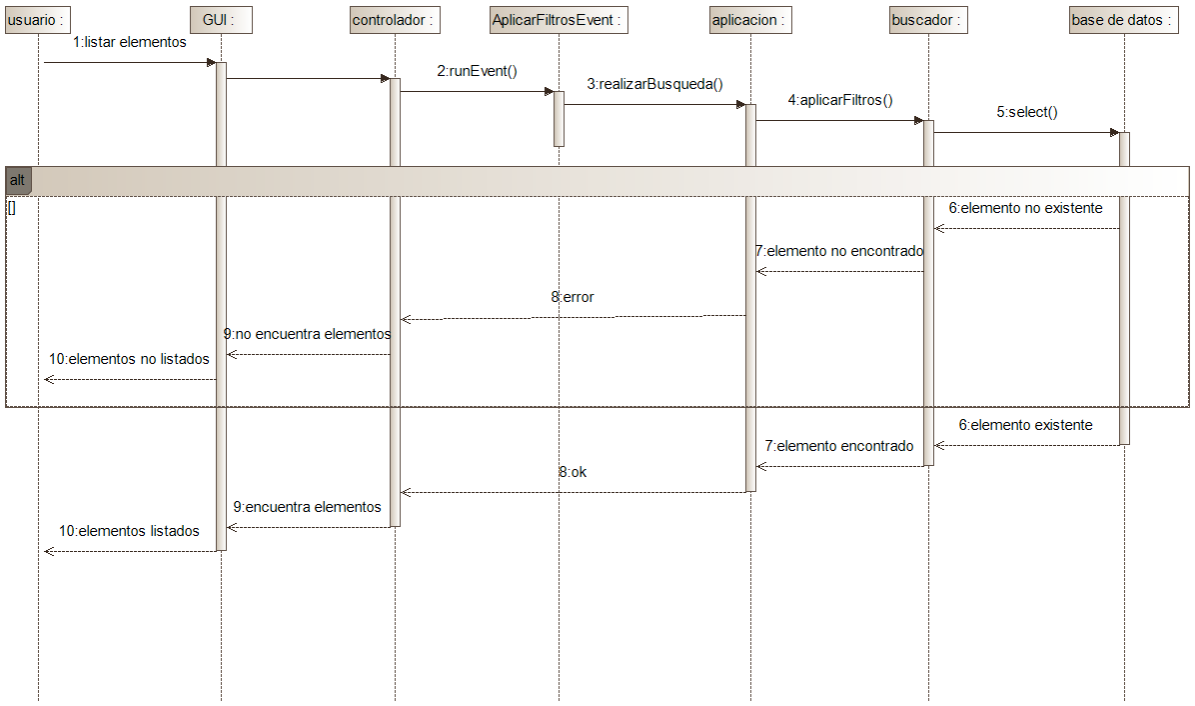
2.3.9 Eliminar elementos

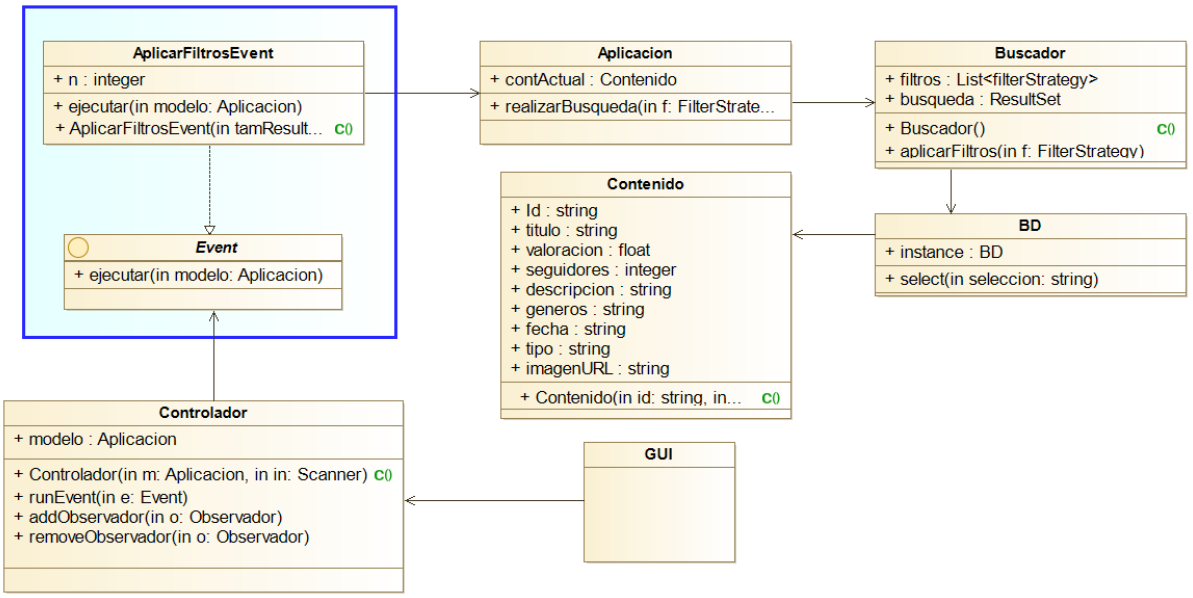




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

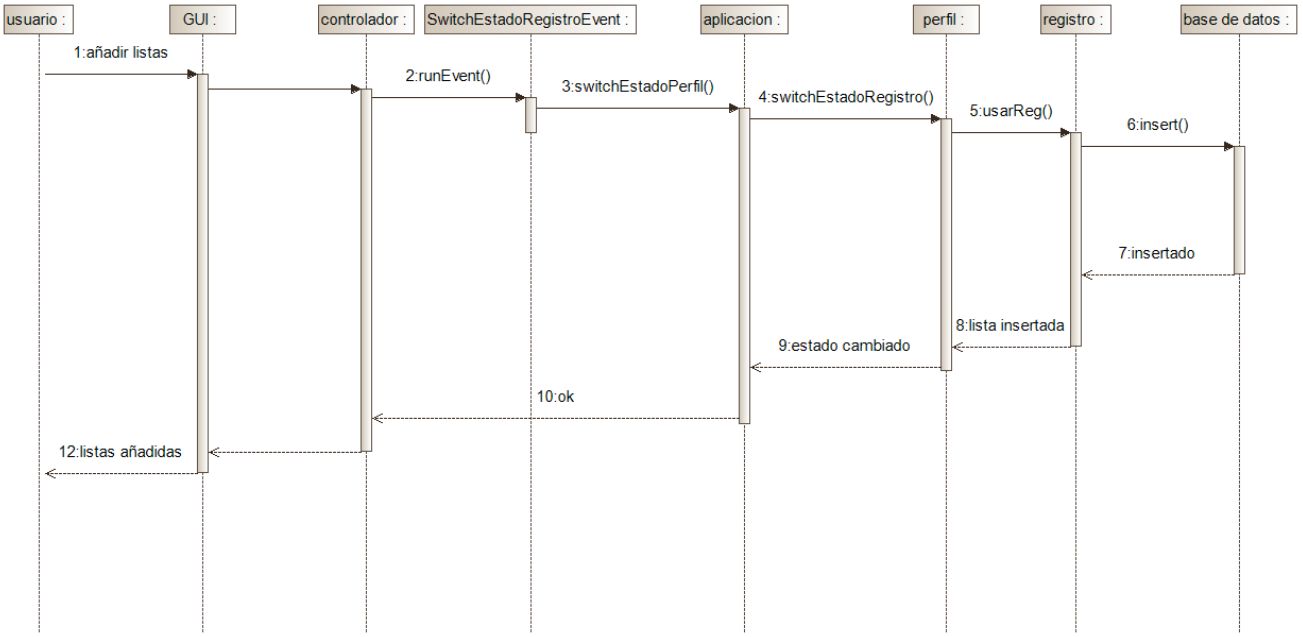
2.3.10 Listar elementos

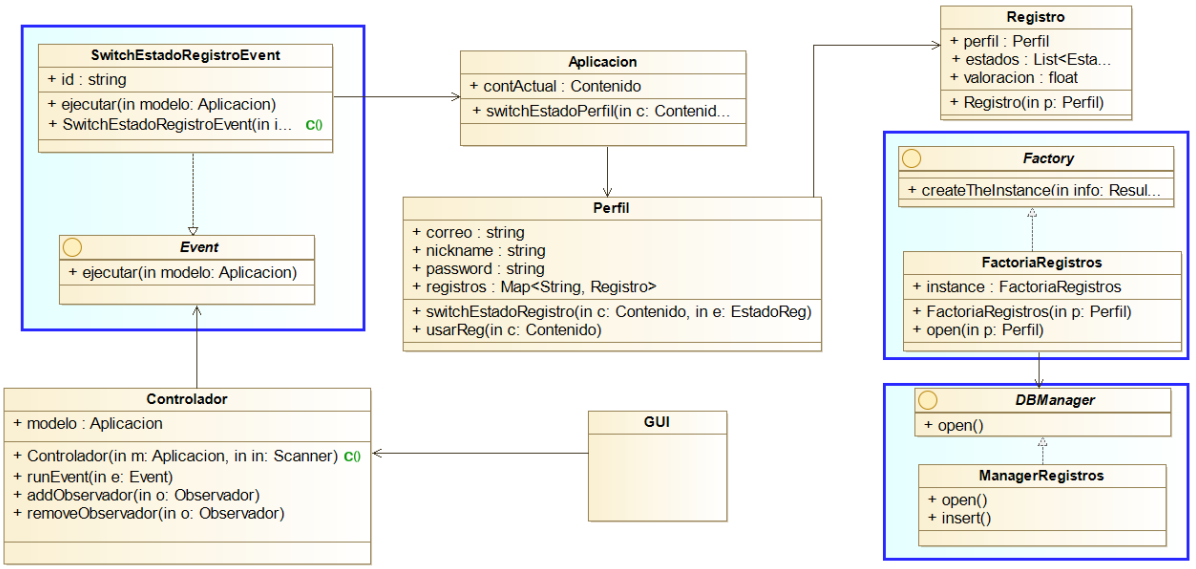




Misma funcionalidad que en el anterior sprint, pero integrando las interfaces gráficas.

2.3.11 Añadir listas / Eliminar listas





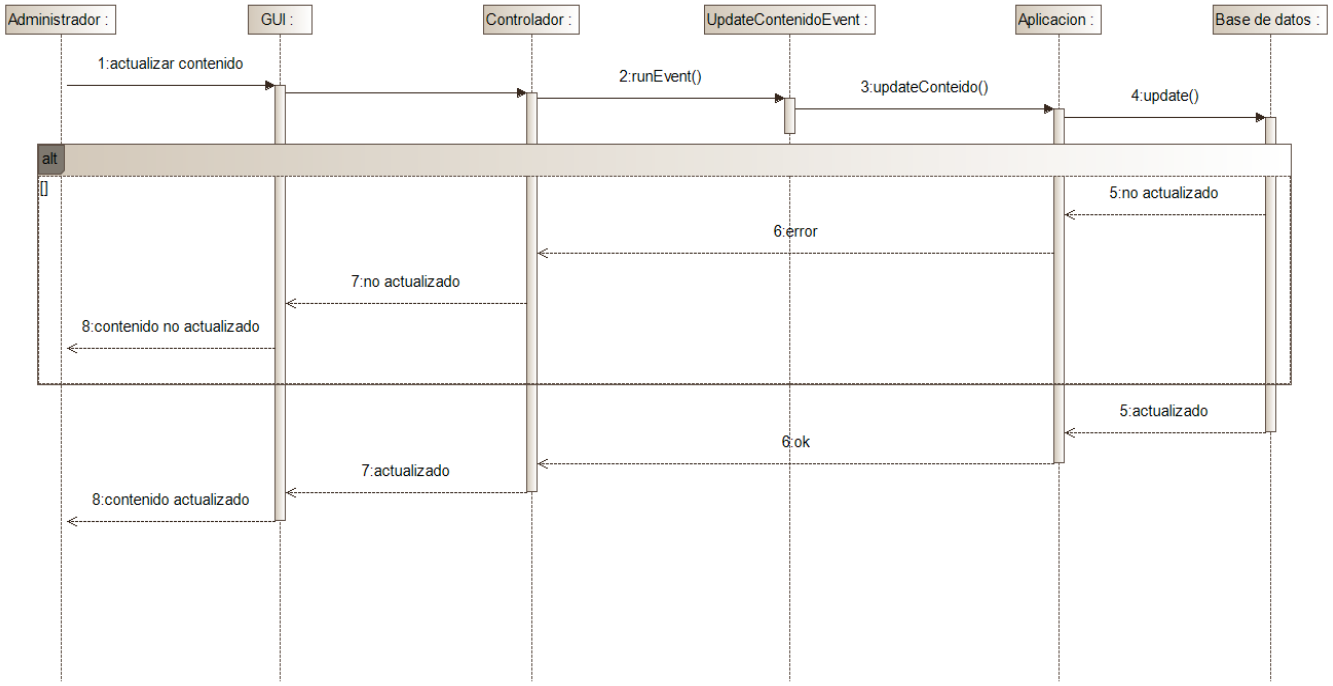
Se ha combinado la funcionalidad añadir y eliminar las listas, puesto que internamente se asignan o eliminan estados sobre un contenido desde el perfil activo, para después consultar según dichos estados. En caso de no existir en el perfil un registro asociado al contenido a tratar, se crea en una factoría y se inserta en la BD en el manager, ambas clases siguiendo el patrón Factory Method y Singleton.

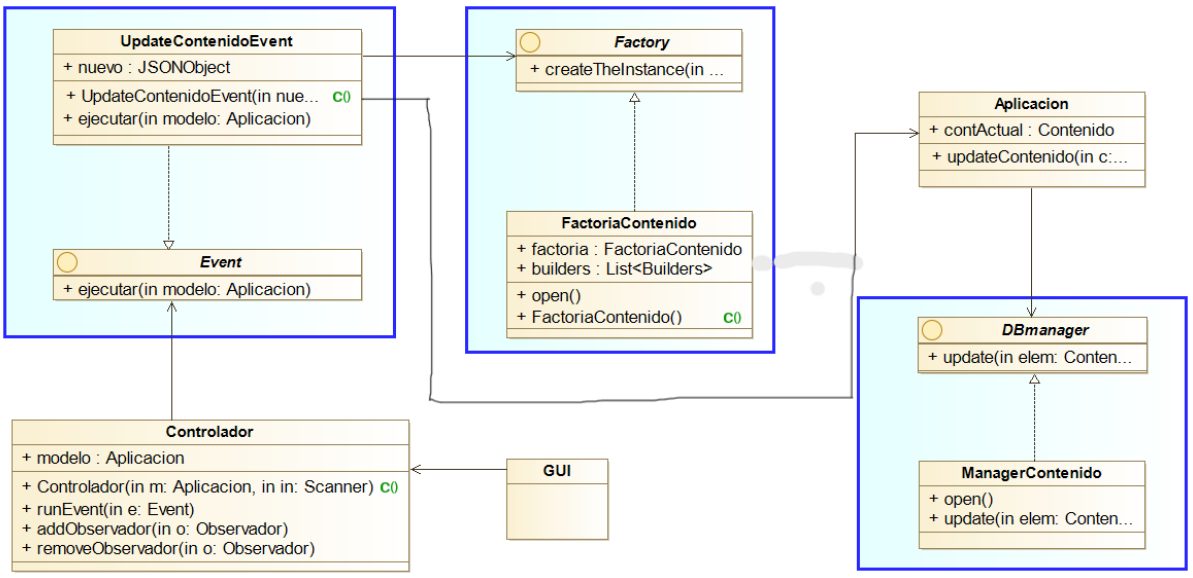
2.4 Sprint 4

2.4.1 Ver información del perfil

El usuario tiene acceso a la información y estadísticas de su perfil a través de distintos paneles que reciben dicho perfil a través de métodos de observador (patrón Observer). No se proporciona un diagrama de clases específico para esta historia por su fuerte vinculación a la interfaz gráfica, y presencia casi nula en la lógica de la aplicación.
A su vez, algunos paneles (SesionPanel) utilizan el patrón builder para su construcción.

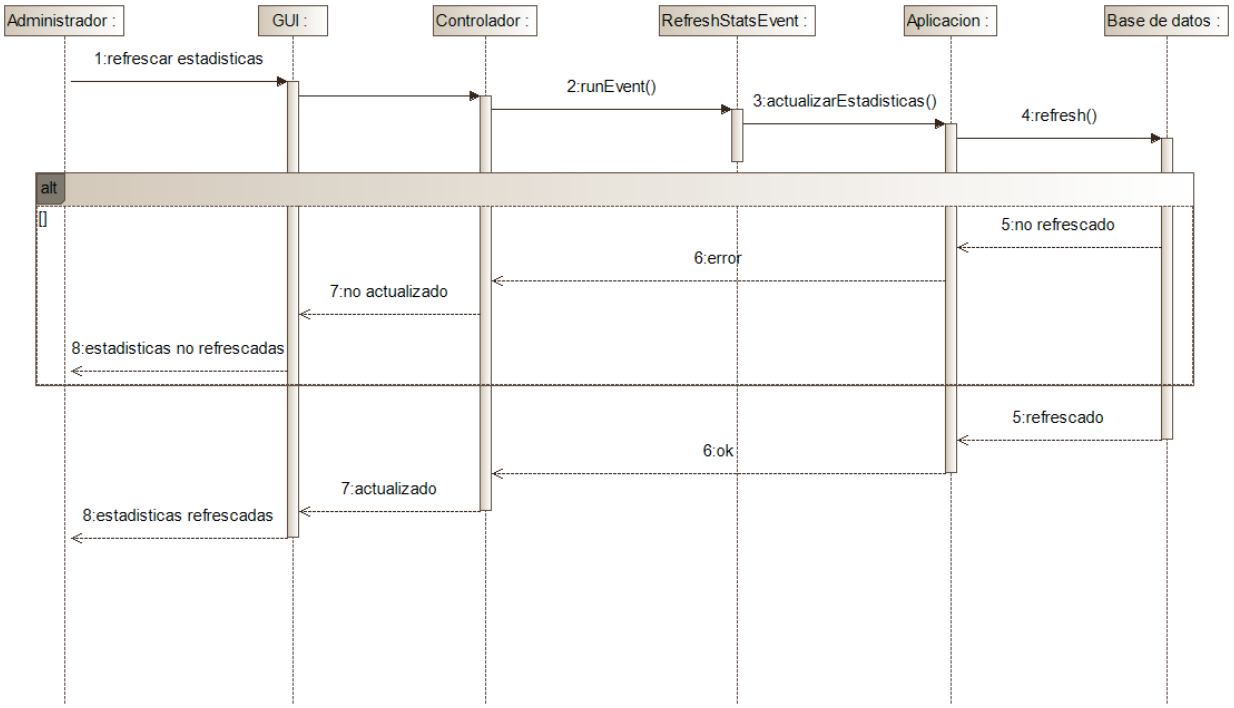
2.4.2 Actualizar elementos

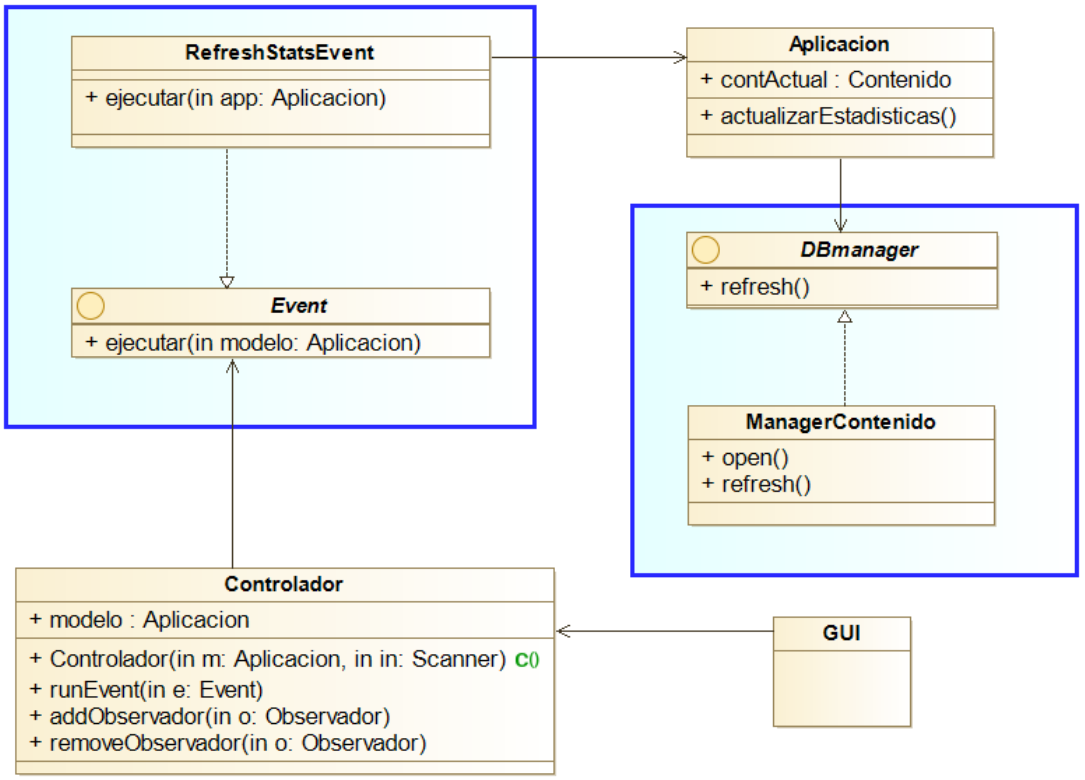




El administrador puede cambiar la información de un cierto contenido tras buscarlo por su ID. Para crear el contenido final e insertarlo en la base de datos se utilizan clases que siguen el patrón Factory Method y Singleton.

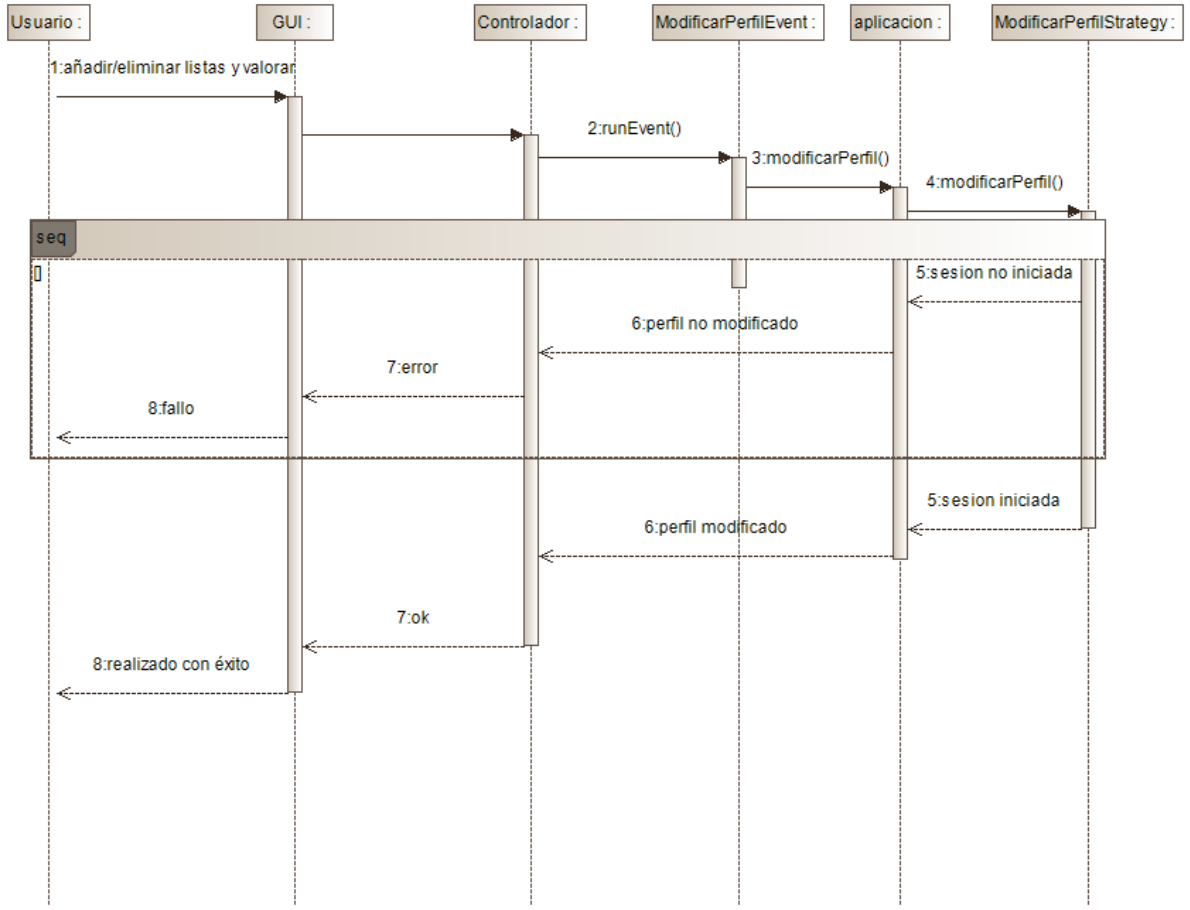
2.4.3 Refrescar estadísticas de elementos

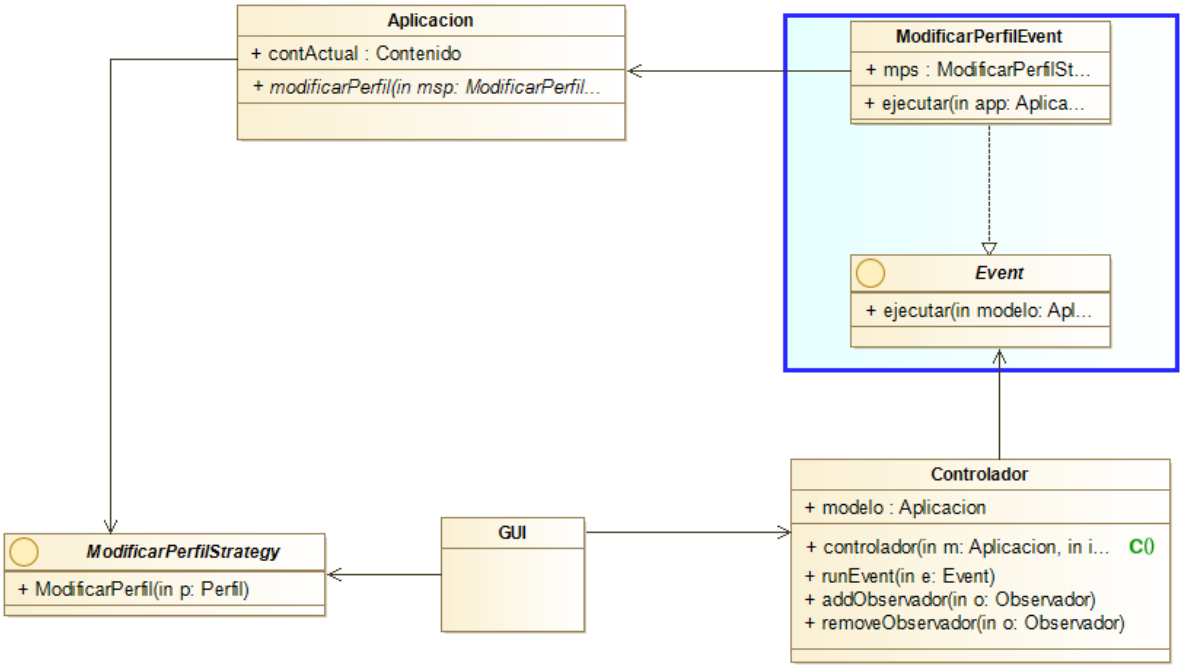




El administrador es quien decide cuándo se actualizan las estadísticas del contenido de la aplicación, de forma que se calculan y actualizan de una vez. Esto previene a la base de datos de recibir demasiadas operaciones en tiempo real, en el caso de haber muchos usuarios activos interactuando con los elementos.

2.4.4 Añadir/eliminar de listas

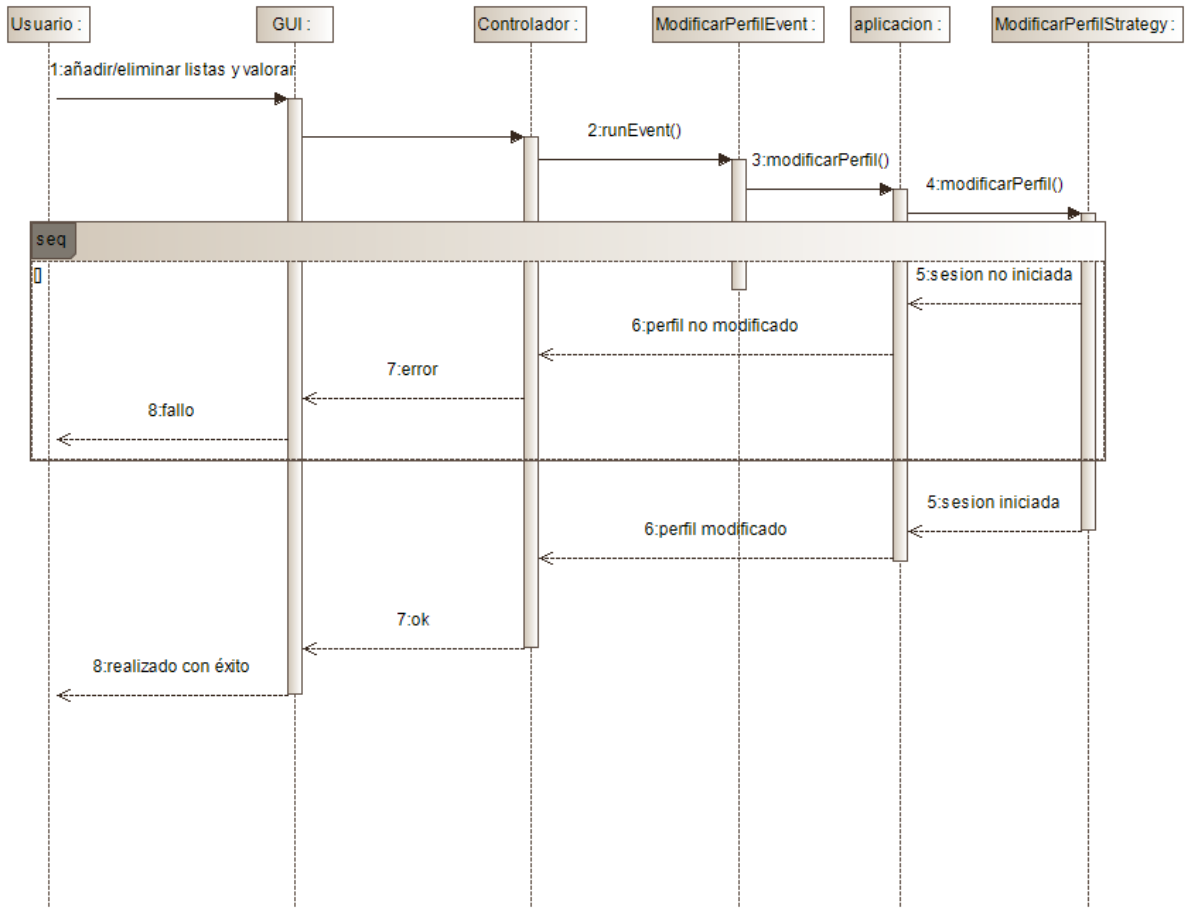


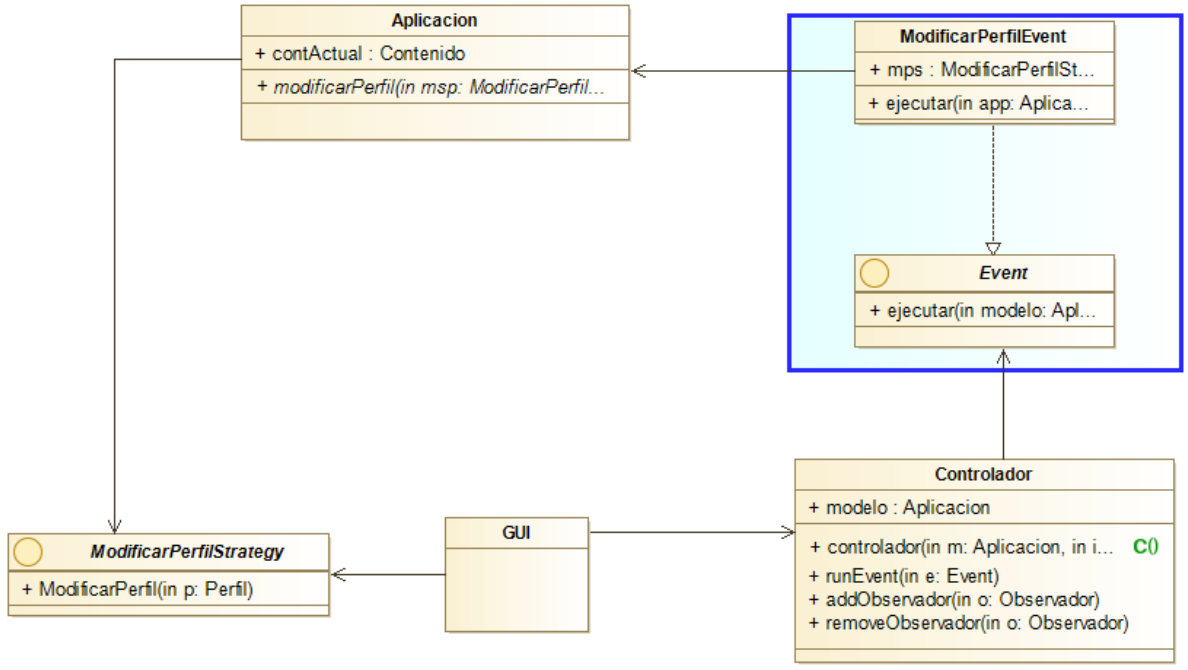


Misma funcionalidad que en el anterior sprint, pero más versátil al utilizar de nuevo el patrón Strategy para determinar cómo se modifica un perfil (en este caso modificando el estado de los elementos con los que ha interactuado).

2.4.5 Valorar un elemento

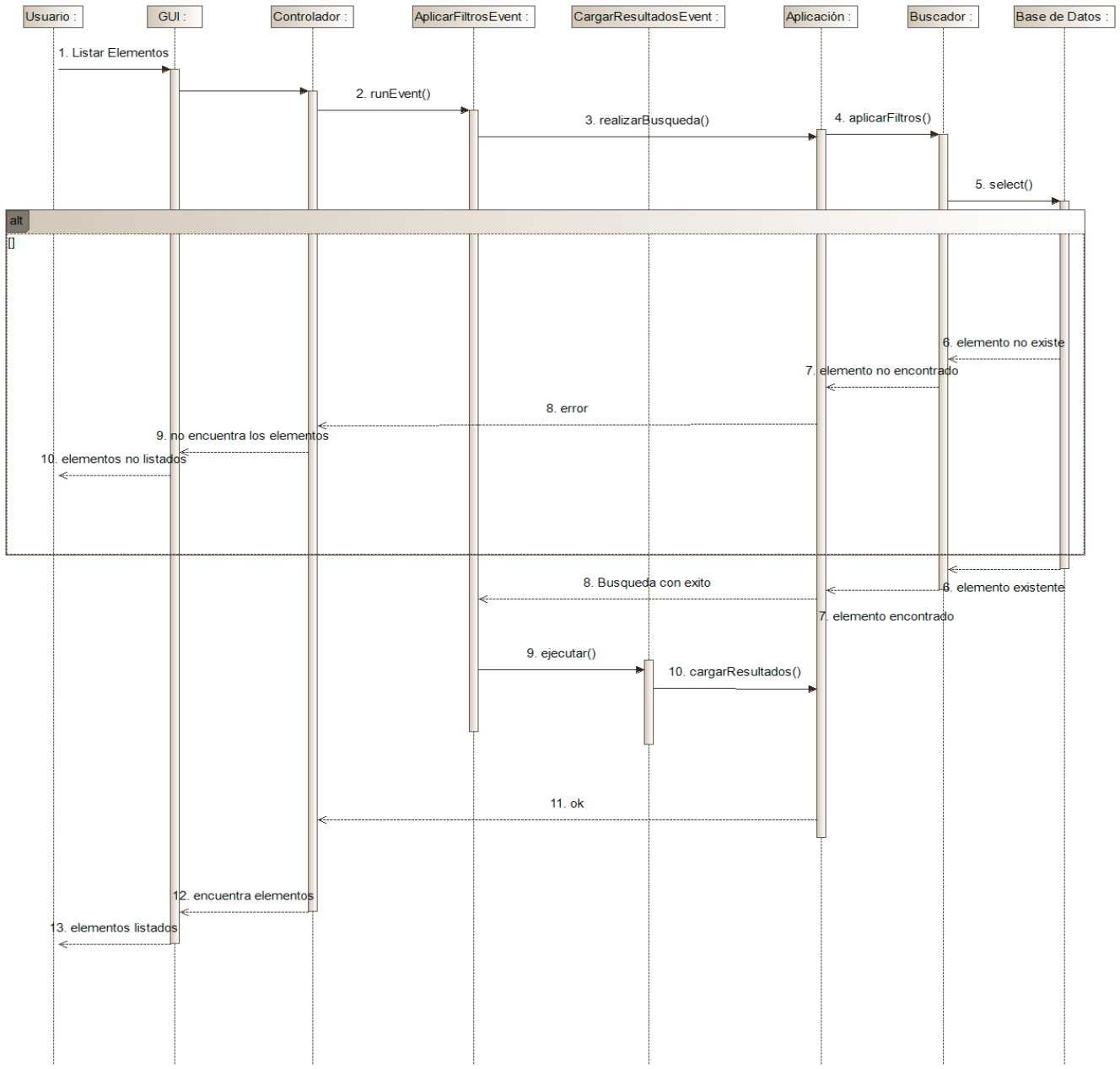
Misma funcionalidad que en el anterior sprint, pero más versátil al utilizar de nuevo el patrón Strategy para determinar cómo se modifica un perfil (en este caso modificando la valoración que asigna a un elemento).

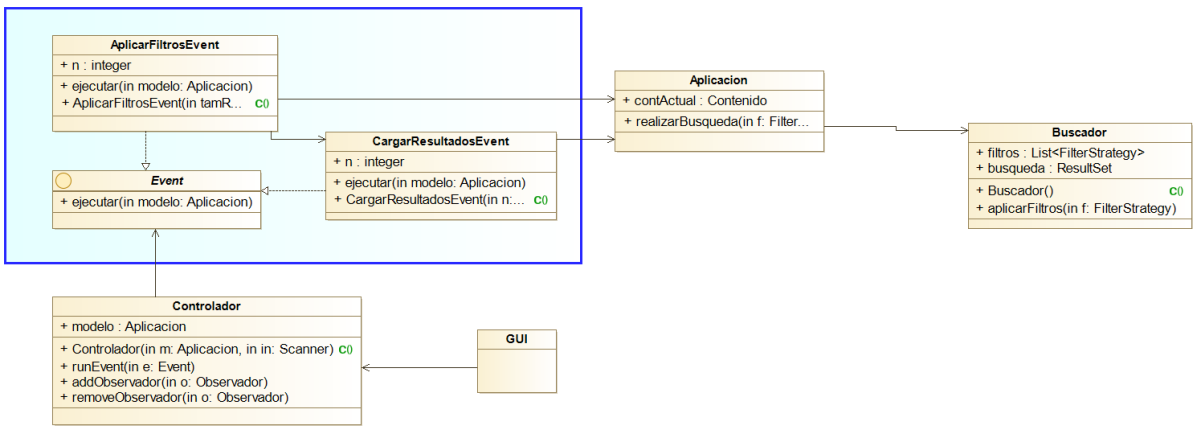




2.4.6 Listar elementos

Misma funcionalidad que en el anterior sprint, pero integrando la base de datos.
Además, para construir las consultas que utiliza el buscador se usa una clase externa que funciona como un Builder de srings (patrón builder).


Usuario :
GUI :
Controlador :
AplicarFiltrosEvent :
CargarResultadosEvent :
Aplicación :
Buscador :
Base de Datos :
1. Listar Elementos
2. runEvent()
3. realizarBusqueda()
4. aplicarFiltros()
5. select()
alt
[]
6. elemento no existe
7. elemento no encontrado
8. error
9. no encuentra los elementos
10. elementos no listados
8. Busqueda con exito
6. elemento existente
7. elemento encontrado
9. ejecutar()
10. cargarResultados()
11. ok
12. encuentra elementos
13. elementos listados


AplicarFiltrosEvent
+ n : integer
+ ejecutar(in modelo: Aplicacion)
+ AplicarFiltrosEvent(in tamR... C0
Event
+ ejecutar(in modelo: Aplicacion)
CargarResultadosEvent
+ n : integer
+ ejecutar(in modelo: Aplicacion)
+ CargarResultadosEvent(in n:... C0
Aplicacion
+ contActual : Contenido
+ realizarBusqueda(in f: Filter...
Buscador
+ filtros : List<FilterStrategy>
+ busqueda : ResultSet
+ Buscador() C0
+ aplicarFiltros(in f: FilterStrategy)
Controlador
+ modelo : Aplicacion
+ Controlador(in m: Aplicacion, in in: Scanner) C0
+ runEvent(in e: Event)
+ addObservador(in o: Observador)
+ removeObservador(in o: Observador)
GUI

2. Recopilación de patrones de diseño

A continuación se enumeran los distintos patrones de diseño utilizados y en qué aspectos del proyecto se han aplicado.

- MVC: Arquitectura de la aplicación.

- Command: Con el nombre de “Eventos”, encapsulan las acciones que un usuario puede hacer, por lo que están fuertemente vinculados con las historias de usuario.

- Observer: Dicta todo el flujo de información entre el modelo y la vista.

- Strategy: Se utilizan, por un lado, a la hora de aplicar filtros, alterando la consulta en uso para permitir esta funcionalidad; y por otro, para realizar cambios sobre la información de un perfil en base a los elementos con los que ha interactuado.

- Builder: Las consultas que utiliza la aplicación se construyen mediante una clase que sigue esté método; así como el panel de la GUI InfoContenidoPanel, construido mediante un builder anidado.

- Singleton: Todas las factorías y la clase principal de la base de datos se instancian mediante este patrón, de forma que se asegura la conexión constante a la base de datos.

- Factory Method: Todas las factorías, ya sea las que crean elementos o las que los manejan en la base de datos, se construyen como factorías con un conjunto de builders internos (que no tienen nada que ver con el patron builder).

- Adapter: Para implementar los métodos de observador, se han utilizado clases abstractas que actúan de puente entre paneles de la GUI que sólo manejan una parte de la información del modelo, y el resto de funcionalidad que un observador debe cubrir (a menudo mediante implementaciones vacías).