PROGRAMACIÓN ORIENTADA A OBJETOS II

TRABAJO GRUPAL FINAL

A LA CAZA DE LAS VINCHUCAS

Integrantes:

Misiukowiec Pablo Moises (Comisión 2)

- pablo.misiukowiec@gmail.com

Cáceres Daniel (Comisión 1)

- dragonaladodera2002@hotmail.com

Troche Leonardo Martin (Comisión 1)

- leo.m.troche@gmail.com

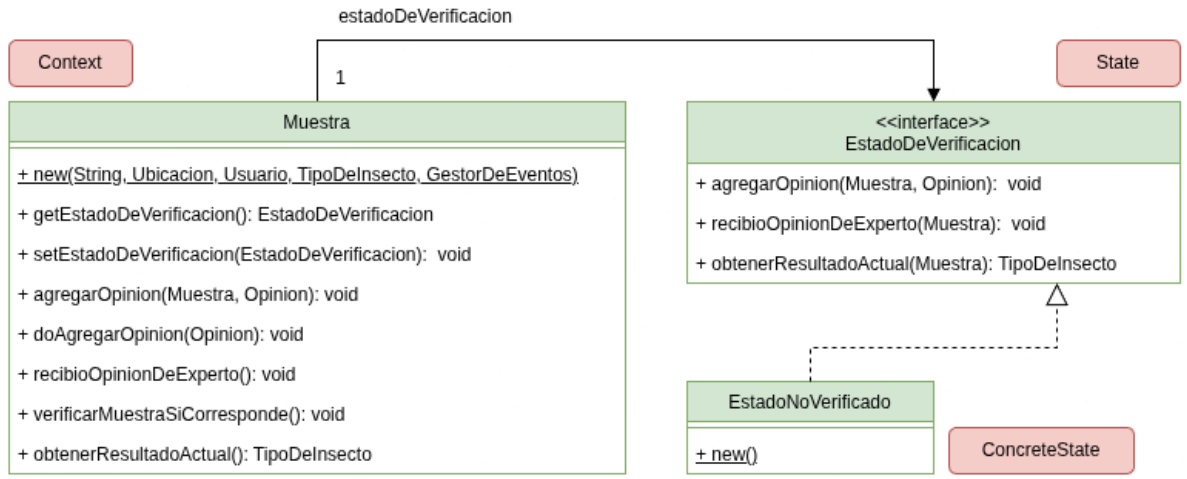
MUESTRAS

Patrones de Diseño

Para modelar el ciclo de vida de una muestra y su comportamiento cambiante frente a las opiniones recibidas, se utilizó el patrón **State**.

Patrón State

Este patrón permite que el objeto Muestra delegue su comportamiento a diferentes objetos que representan su estado actual, lo que evita condicionales complicados y mejora tanto la claridad como la extensibilidad del diseño.



Se definieron tres estados concretos:

- **EstadoNoVerificado**: Estado inicial de la muestra, donde cualquier usuario puede opinar (básico o experto).
- **EstadoEnProceso**: Estado en el que ya ha opinado al menos un experto, y solo se permiten nuevas opiniones de expertos.
- **EstadoVerificado**: Estado final, cuando dos expertos coinciden en su opinión. No se permiten más opiniones.

Cada uno de estos estados implementa una interfaz común que define las operaciones relevantes, como agregarOpinion u obtenerResultadoActual.

Decisiones de Diseño

Se optó por separar completamente la lógica de cada estado en clases distintas para evitar que la clase Muestra tuviera que manejar múltiples responsabilidades.

Esta decisión también facilita el mantenimiento y la extensión del sistema: si en el futuro se agregan nuevos estados (por ejemplo, una muestra contaminada), basta con definir una nueva clase de estado sin modificar las existentes.

Detalles de Implementación

El cálculo del resultado actual de una Muestra depende de su estado y se delega a cada implementación concreta del patrón State, siguiendo estos criterios:

- **EstadoNoVerificado**:

El resultado se obtiene agrupando las opiniones por TipoDeInsecto. La agrupación con mayor cantidad de votos se considera como resultado actual. Esta lógica evita condicionales externos y encapsula el criterio directamente en el estado.

- **EstadoEnProceso**:

La primera opinión emitida por un experto se guarda como resultado actual inicial. A partir de allí, mientras no existan dos coincidencias entre expertos, el resultado es considerado “no definido” por empate. Esto se gestiona internamente con una estructura que lleva el historial de opiniones expertas y controla las transiciones.

- **EstadoVerificado**:

Al momento de verificarse la muestra (es decir, cuando dos expertos coinciden), se bloquean nuevas opiniones. El resultado final queda determinado por la última opinión registrada en el historial, coincidente con el criterio de verificación.

USUARIOS

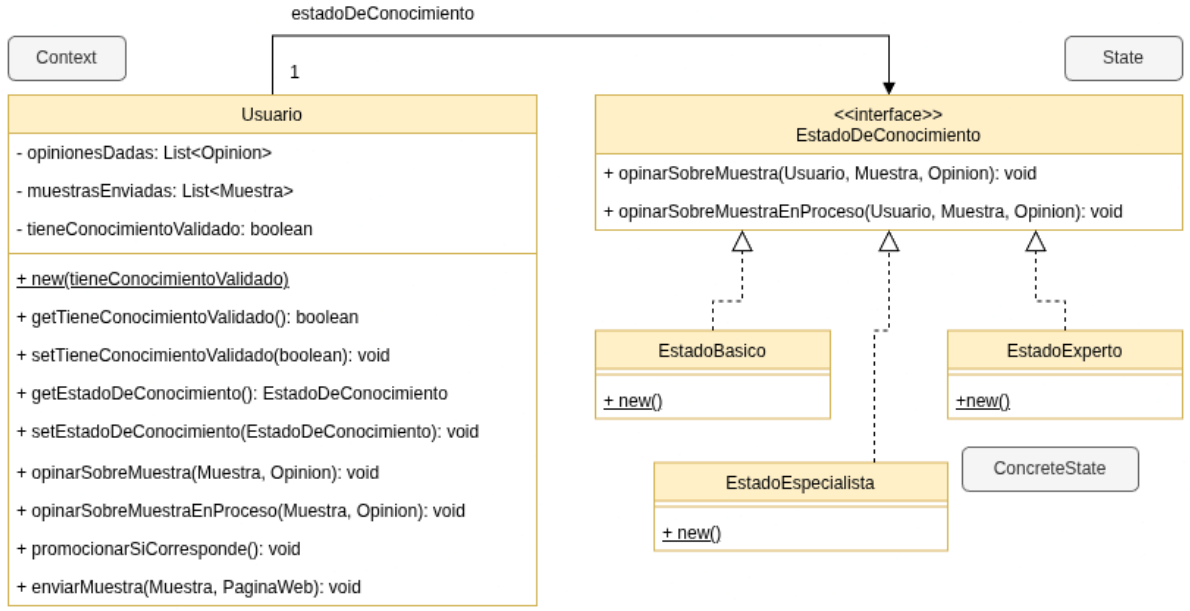
Patrones de Diseño

Para modelar el comportamiento dinámico de los usuarios según su experiencia y validación en el sistema, se aplicó el patrón de diseño State. Este patrón permite que el objeto Usuario modifique su comportamiento interno dependiendo de su estado actual, sin recurrir a estructuras condicionales extensas.

Patrón State

Se definieron 3 estados concretos:

- **EstadoBasico**: Los usuarios básicos pueden opinar sobre muestras, pero sus votos solo cuentan hasta que un experto interviene.
- **EstadoExperto**: Los expertos son personas con conocimientos en el tema, cuyas opiniones son exclusivas una vez emitidas.
- **EstadoEspecialista**: Los especialistas son un tipo más avanzado de experto, con validación científica o institucional, con permisos extra si se desea extender el sistema.



Decisiones de Diseño

Además del uso del patrón State para modelar el comportamiento dinámico de los usuarios, se tomó como decisión de diseño principal que el estado del usuario debe asignarse en el momento de su instanciación, en función de si posee o no conocimientos validados.

- Si el usuario no tiene validación de conocimientos, se lo inicializa en el EstadoBasico.
- Si el usuario presenta conocimientos validados institucionalmente, se lo instancia directamente en el estado EstadoEspecialista.

Esta decisión permite mantener el sistema coherente desde el inicio del ciclo de vida del usuario, evitando ambigüedades sobre su rol.

Por otro lado, se definió que:

- Un usuario puede cambiar entre EstadoBasico y EstadoExperto dinámicamente, de acuerdo con un algoritmo de promoción o degradación, basado en su participación, calidad de opiniones u otras métricas del sistema.
- Los usuarios en EstadoEspecialista no están sujetos a este algoritmo, ya que su validación es externa y permanente; por lo tanto, mantienen su estado fijo durante todo el ciclo de vida del sistema.

Esta política refleja correctamente los distintos niveles de autoridad y confianza que se espera de cada tipo de usuario.

Detalles de Implementación

No se presentan detalles adicionales relevantes para esta sección, ya que la implementación es directa y se ajusta a lo descrito en los apartados anteriores.

AVISO A ORGANIZACIONES

Las organizaciones interesadas pueden suscribirse a distintos eventos geográficos a través de las zonas de cobertura. Actualmente se contemplan dos eventos principales: la **carga de una nueva muestra** y la **validación de una muestra**, aunque el sistema está preparado para soportar más eventos en el futuro.

Patrones de Diseño

Para modelar la comunicación entre las entidades notificadoras y los observadores de eventos se utilizó el patrón **Observer**.

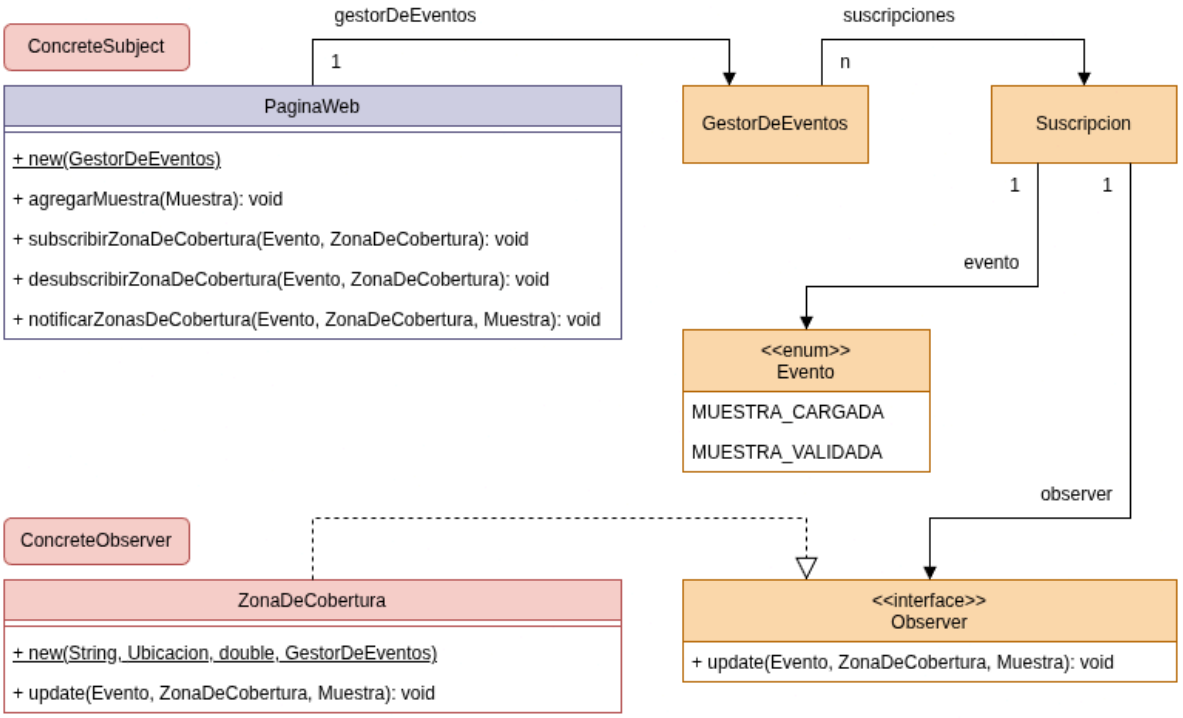
Patrón Observer

Este patrón está presente en **varias partes del sistema**, como:

Desde PaginaWeb a ZonaDeCobertura

Cuando se carga una nueva Muestra desde la interfaz web, la aplicación notifica a todas las instancias de ZonaDeCobertura suscritas al evento de carga.

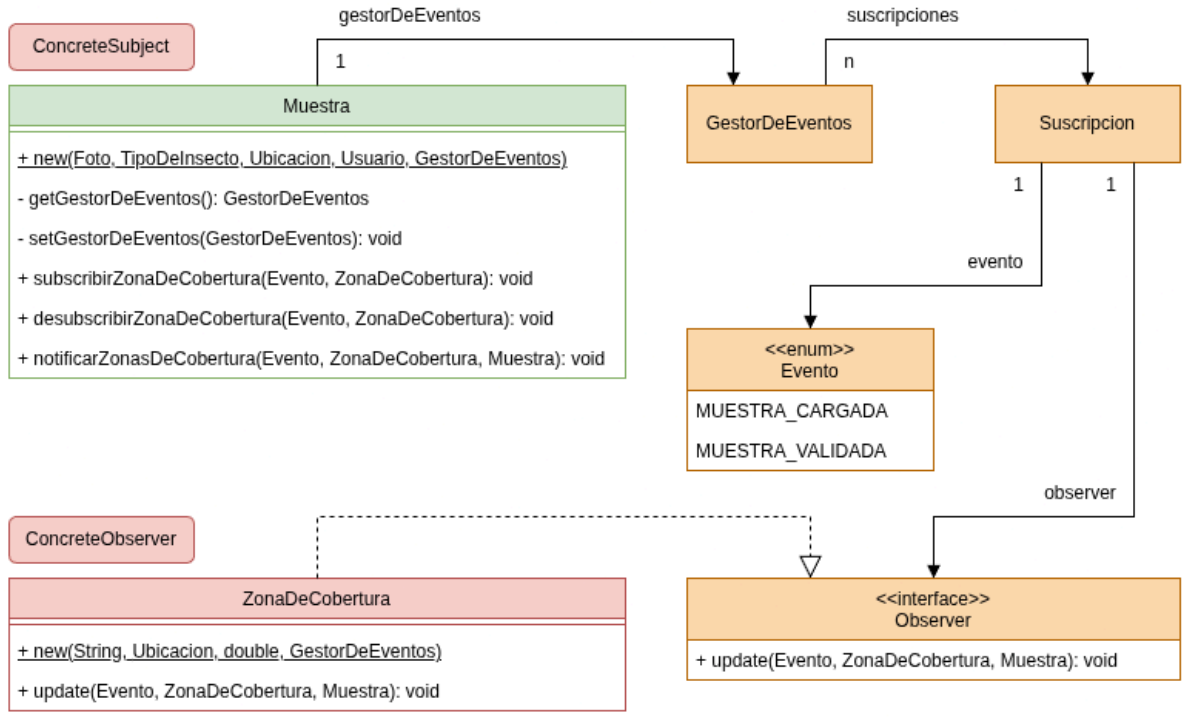
Cada zona verifica si la ubicación de la muestra pertenece a su área geográfica y, en caso afirmativo, agrega la muestra a su colección interna y luego notifica a las Organizaciones suscritas a dicho evento, habilitándolas a ejecutar sus funcionalidades externas asociadas.



De Muestra a ZonaDeCobertura

Cuando una Muestra cambia su estado a EstadoVerificado, notifica a todas las instancias de ZonaDeCobertura que estén suscritas al evento de validación.

Cada ZonaDeCobertura verifica si la muestra pertenece a su área geográfica, y si la condición se cumple, notifica a todas las Organizaciones suscritas a dicho evento, permitiéndoles ejecutar sus funcionalidades externas asociadas.



De ZonaDeCobertura a Organizacion

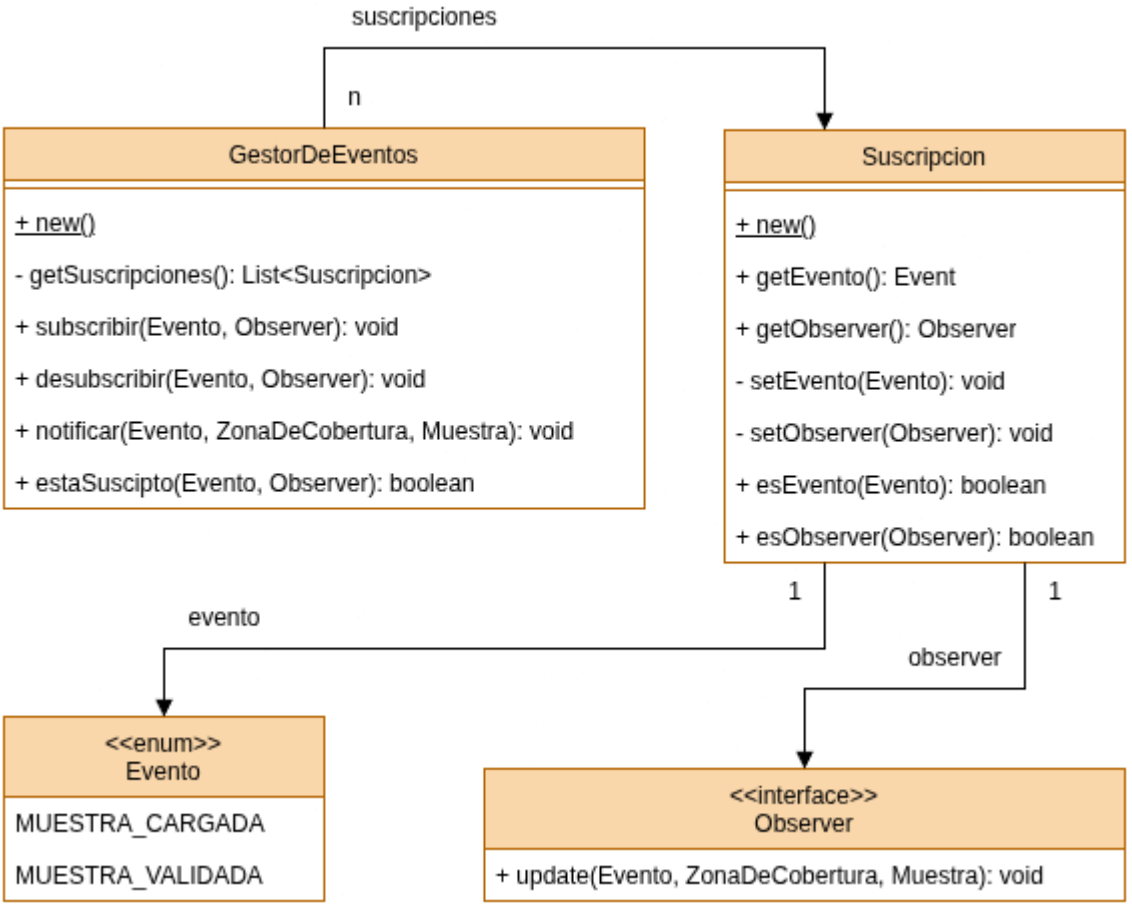
Cuando una ZonaDeCobertura detecta que se ha producido un evento relevante dentro de su área geográfica, notifica exclusivamente a las organizaciones que están suscritas a ese tipo de evento, de modo que estas últimas puedan ejecutar sus funcionalidades externas asociadas en respuesta al evento.

Decisiones de Diseño

Dado que la lógica de suscripción, desuscripción y notificación de observadores se repetiría en múltiples entidades, se decidió extraer esa funcionalidad en un componente reutilizable: el GestorDeEventos (o Event Manager).

Este gestor es inyectado mediante el constructor en todas las clases que actúan como Subject, es decir, aquellas que desean notificar eventos a observadores registrados.

De esta forma, cualquier clase puede delegar en el GestorDeEventos el manejo completo del patrón Observer, manteniéndose enfocada en su responsabilidad principal.



Además, el gestor permite registrar múltiples observadores por tipo de evento, facilitando la suscripción selectiva y evitando acoplamientos innecesarios entre productores y consumidores de eventos.

Detalles de Implementación

Si bien la implementación general del patrón Observer no presenta complejidades particulares, se destaca el uso de un GestorDeEventos centralizado que permite manejar múltiples tipos de eventos de forma desacoplada y escalable.

Cada observador debe implementar la interfaz Observer, la cual define el método update(Evento, ZonaDeCobertura, Muestra).

Esto asegura una forma uniforme de recibir y procesar eventos, independientemente del tipo de entidad que actúe como observador (por ejemplo, una organización).

FUNCIONALIDADES EXTERNAS

Cuando una Organización es notificada de un evento —como la carga o verificación de una Muestra— se ejecuta una funcionalidad externa asociada a dicho evento, permitiendo a la organización realizar acciones específicas (por ejemplo, enviar notificaciones por correo, registrar información en un log, entre otras).

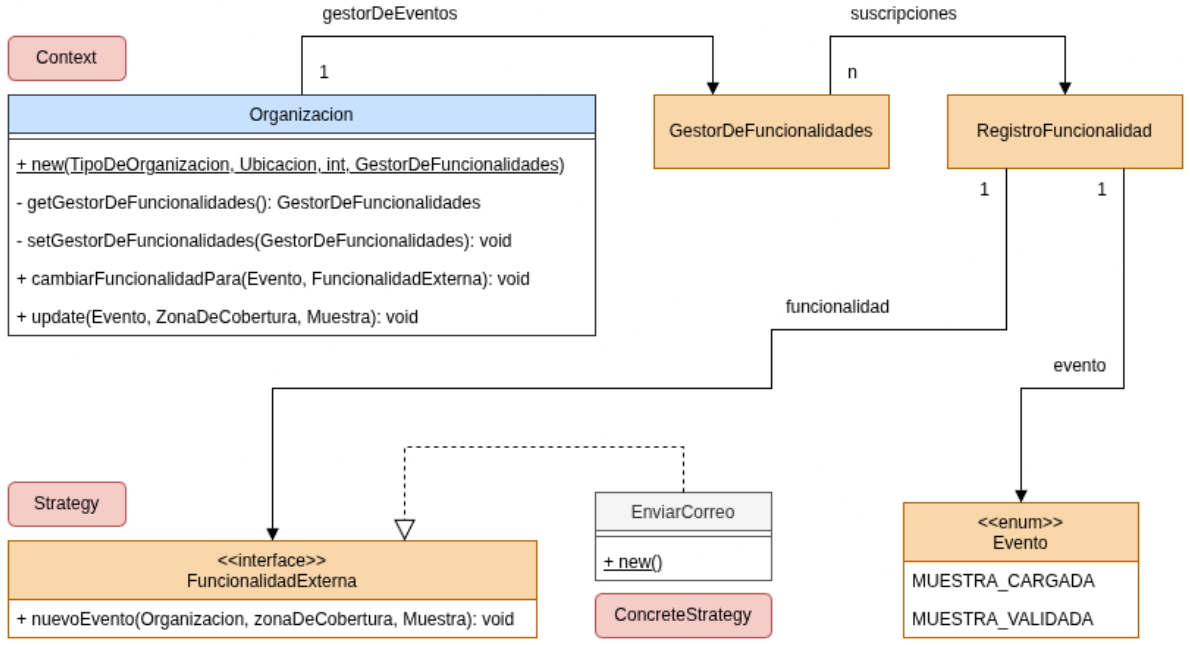
Cada tipo de evento puede tener una única funcionalidad activa, y dicha funcionalidad puede ser intercambiada dinámicamente en tiempo de ejecución.

Patrones de Diseño

Para modelar las funcionalidades externas como comportamientos intercambiables, se utilizó el patrón **Strategy**.

Patrón Strategy

Cada funcionalidad concreta (por ejemplo, EnviarCorreo, GuardarLog, etc.) implementa una interfaz común FuncionalidadExterna, que define un método nuevoEvento(Organizacion, ZonaDeCobertura, Muestra).



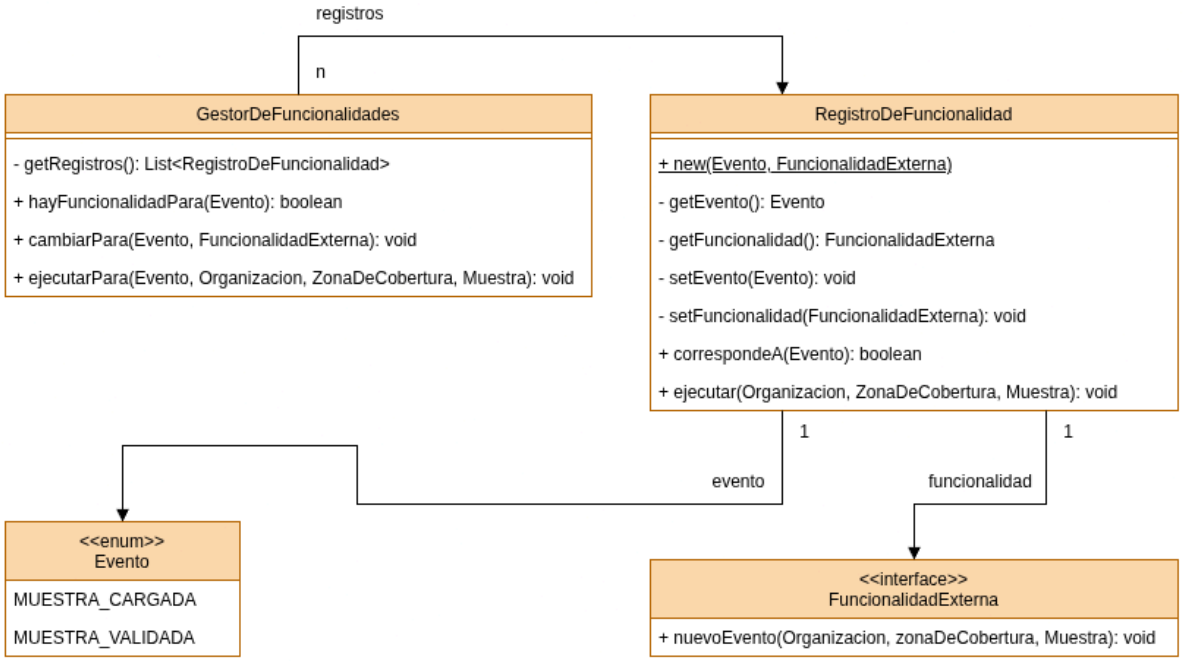
La Organización no conoce los detalles de la funcionalidad concreta; simplemente delega su ejecución al componente configurado en ese momento.

Decisiones de Diseño

Al igual que en el caso del patrón Observer, se decidió encapsular la lógica de gestión de estrategias en un componente separado: el GestorDeFuncionalidades.

Este gestor mantiene un mapeo entre clases de eventos y funcionalidades concretas, permitiendo:

1. Asociar dinámicamente una funcionalidad a un tipo de evento usando el método cambiarFuncionalidadPara(Evento, FuncionalidadExterna).
2. Delegar la ejecución de la funcionalidad correcta cuando se recibe una notificación usando el método ejecutarPara(Evento, Organizacion, ZonaDeCobertura, Muestra).



De este modo, el sistema sigue abierto a extensión (nuevos eventos o nuevas funcionalidades) sin necesidad de modificar las Organizaciones ni la infraestructura general.

Detalles de Implementación

No se requieren elementos adicionales a lo ya descrito: cada Organización configura una única estrategia para cada tipo de evento, la cual se ejecuta directamente mediante el GestorDeFuncionalidades.

Este diseño permite mantener el código modular y limpio, sin necesidad de lógica adicional dentro de las organizaciones.

BÚSQUEDA DE MUESTRAS

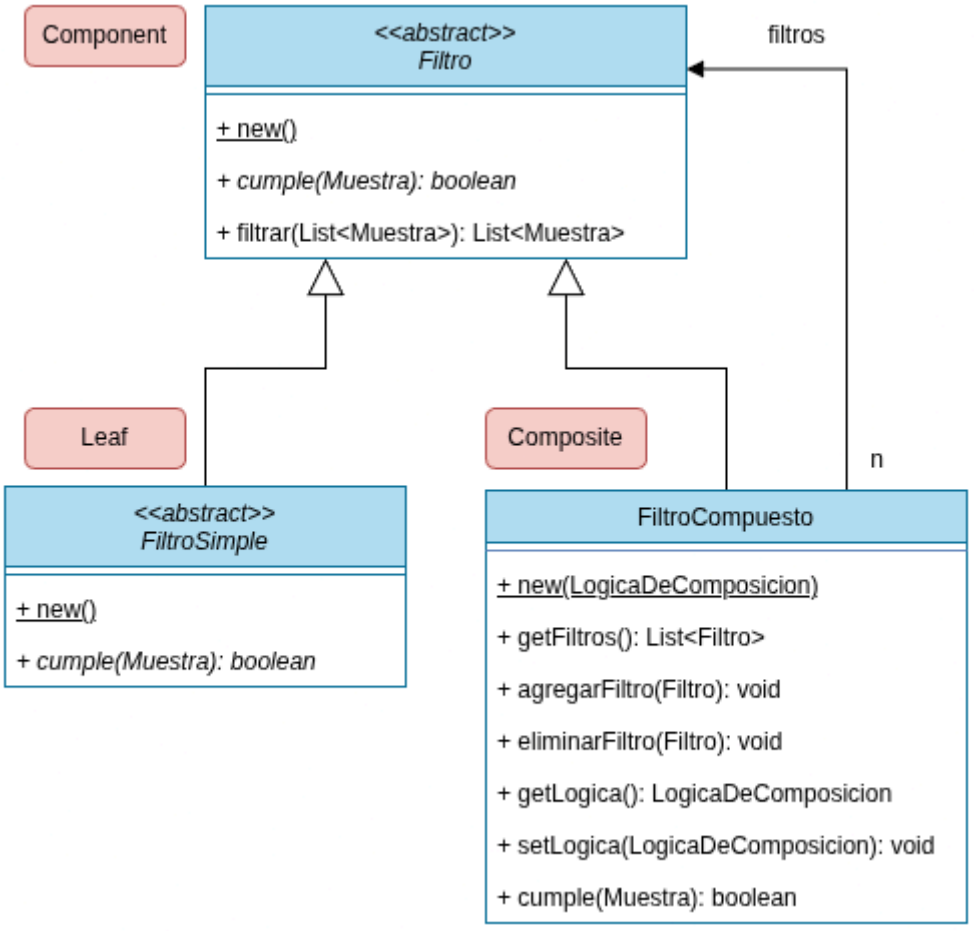
Patrones de Diseño

Para implementar un sistema flexible y extensible de búsqueda de muestras, se utilizaron dos patrones de diseño: **Composite** y **Strategy**.

La elección de estos patrones permitió construir filtros reutilizables, combinables y fácilmente adaptables a diferentes necesidades.

Patrón Composite

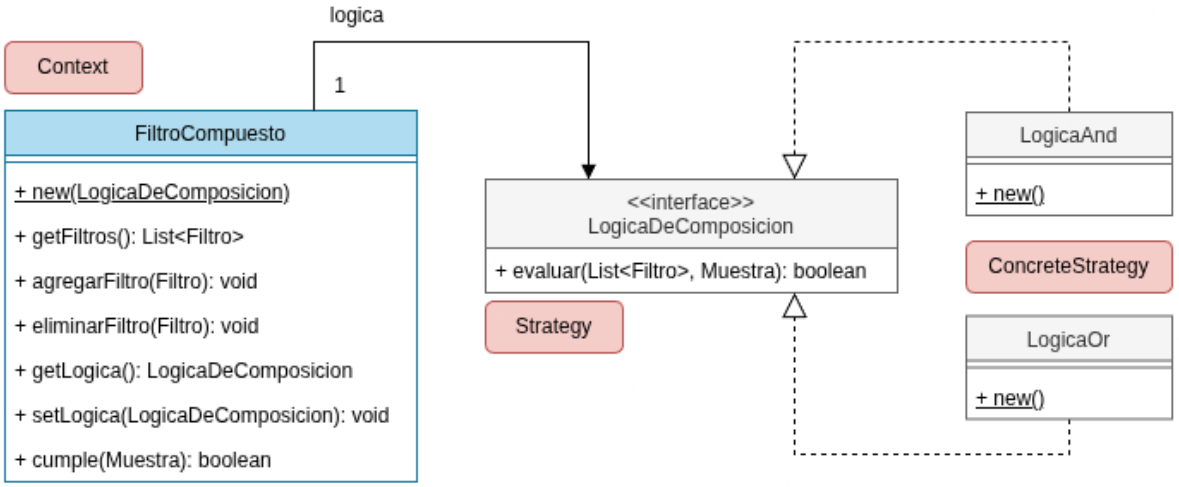
Este patrón se usó para representar filtros de búsqueda que pueden ser simples o compuestos. Es decir, filtros que funcionan por sí solos (como buscar por fecha o por categoría) y filtros que combinan varios criterios a la vez.



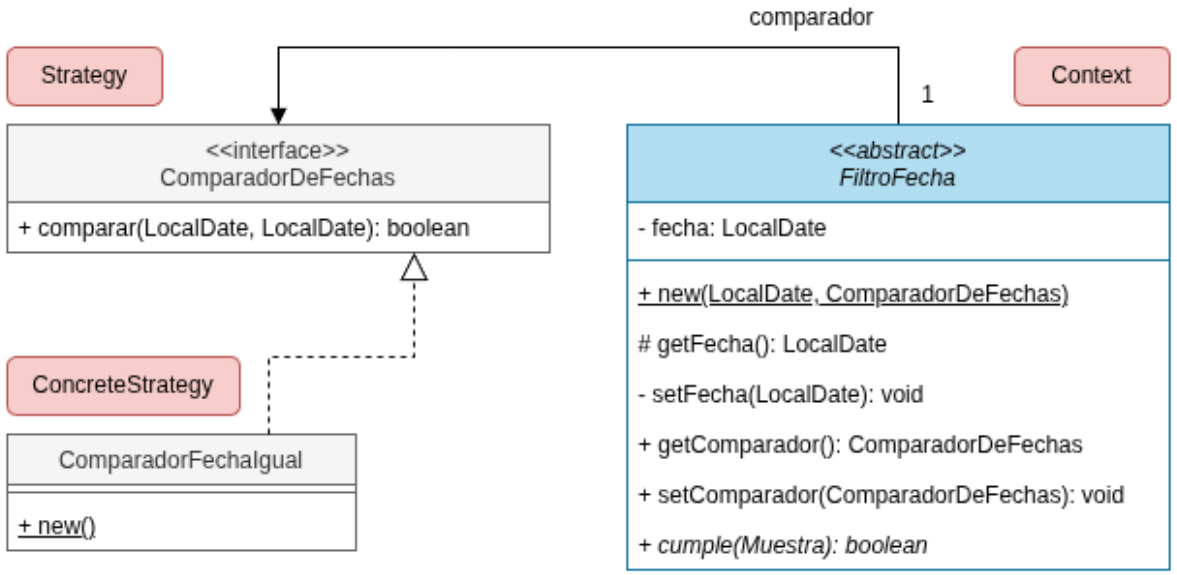
El patrón permite trata a todos los filtros de la misma forma, sin importar si son simples o compuestos. Esto facilita su uso y combinación en distintos escenarios de búsqueda.

Patrón Strategy

Este patrón se aplicó para poder cambiar de forma dinámica cómo se combinan los filtros, según la necesidad del usuario. Por ejemplo, si se quieren muestras que cumplan todas las condiciones (AND) o alguna de ellas (OR).



También se usó este patrón para los filtros de fecha, un tipo de filtro simple, permitiendo cambiar fácilmente el tipo de comparación a realizar: si la fecha es anterior, posterior o igual a una determinada.



Como se puede apreciar, el método, heredado de la clase Filtro, sigue siendo abstracto, de modo que las clases que extienden FiltroFecha —como FiltroFechaDeCreacion y FiltroFechaDeUltimaVotacion— implementen la lógica específica según la fecha que les corresponde.

Decisiones de Diseño

En la clase abstracta Filtro, se implementó un método filtrar marcado como final, encargado de determinar qué muestras cumplen con el criterio del filtro. De este modo, cada subclase —ya sea un filtro simple o compuesto— debe implementar su propia lógica en el método cumple para evaluar si una muestra satisface el filtro correspondiente.

Detalles de Implementación

En esta sección no hay aspectos destacables, ya que el código de cada clase es bastante sencillo y directo, sin complicaciones ni detalles que necesiten una explicación adicional.

CONCLUSIONES

La realización de este trabajo práctico presentó **varios desafíos**, tanto en la **organización del equipo** como en las **decisiones de diseño enfocadas en la responsabilidad de los objetos**.

Uno de los principales retos fue coordinar la interacción entre diferentes patrones de diseño, especialmente en lo que respecta a **la relación entre el estado de una Muestra y el conocimiento del Usuario**.

Otro desafío significativo fue crear un sistema de **filtros de búsqueda** que fuera **escalable, combinable y que permitiera intercambios dinámicos**. Después de evaluar varias opciones, llegamos a la conclusión de que la combinación de los patrones **Composite** y **Strategy** era la solución más flexible y fácil de mantener.

En cuanto a los **gestores de eventos y funcionalidades externas**, al principio pensamos en usar una estructura Map para vincular eventos a acciones. Sin embargo, decidimos descartar esta opción debido a la complejidad adicional que traía consigo, eligiendo en su lugar **soluciones más simples y directas**.

Tanto el **algoritmo** para verificar muestras como el que determina si se promueve o degrada el estado de conocimiento de los usuarios **requirieron una atención especial**. Aunque representaron obstáculos técnicos, logramos superarlos de manera efectiva.

Finalmente, para modelar adecuadamente las ubicaciones geográficas sin añadir complejidad innecesaria, investigamos buenas prácticas sobre el uso de **latitud y longitud**, lo que nos permitió implementar una solución clara y funcional.

En resumen, este trabajo práctico implicó una serie de decisiones y problemas de diseño que abordamos de manera colaborativa. Todos los miembros del equipo participaron activamente en diferentes partes del sistema, lo que resultó en una **solución** que consideramos **escalable**, **bien estructurada** y con un **uso sólido de los patrones de diseño** que aprendimos en clase.