vinchunqui

Programacion con objetos II - 2023

Cortizas Margarita - margarita.cortizas@gmail.com;

Licciardello Martin - martinlicciardello7@gmail.com;

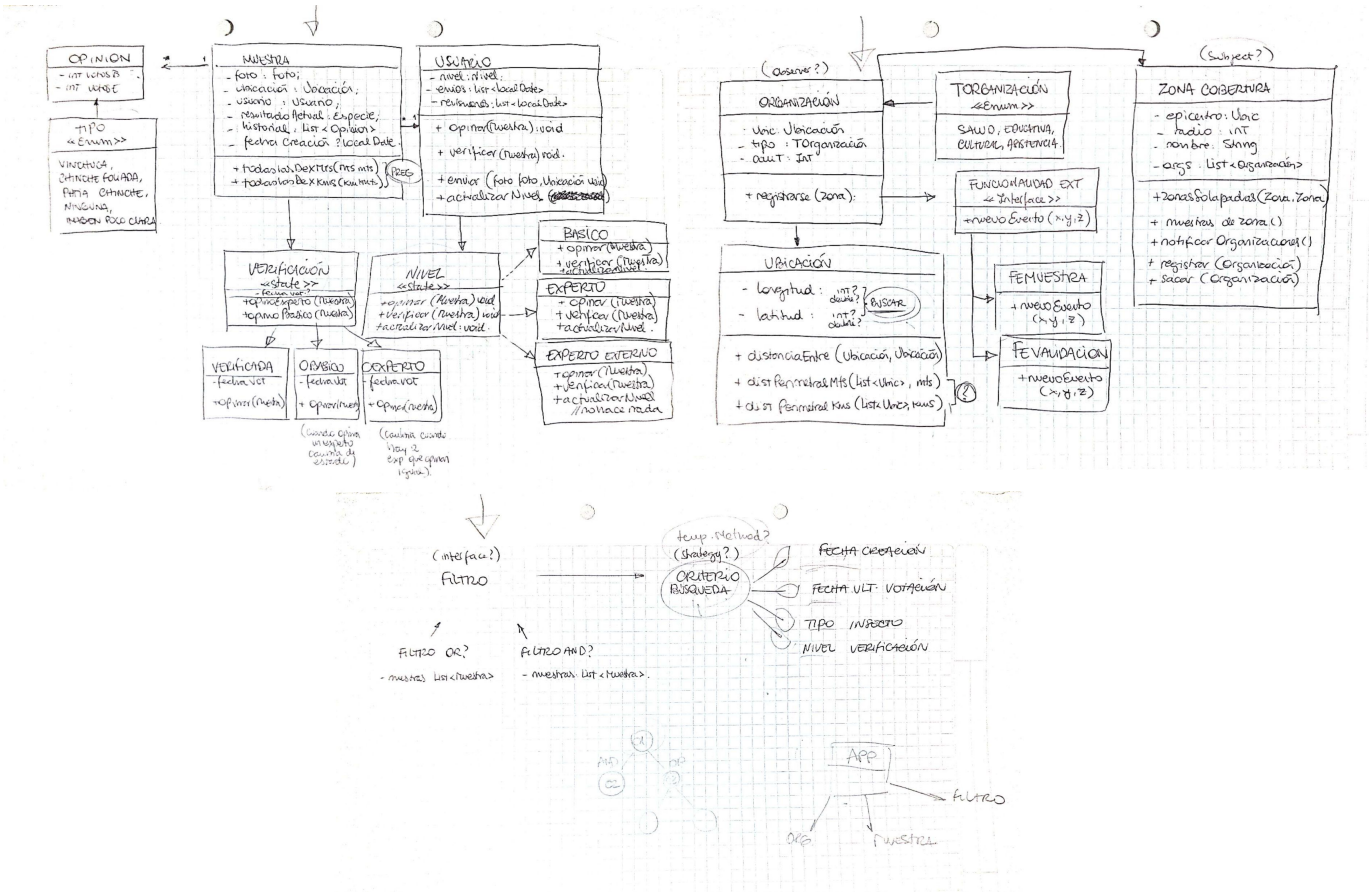
Martínez Lucas Javier - lucasluquitas477@gmail.com;

INTRODUCCIÓN

Este informe redacta una serie de implementaciones y decisiones de diseño para la solución del trabajo final de Objetos 2 del primer semestre del año 2023.

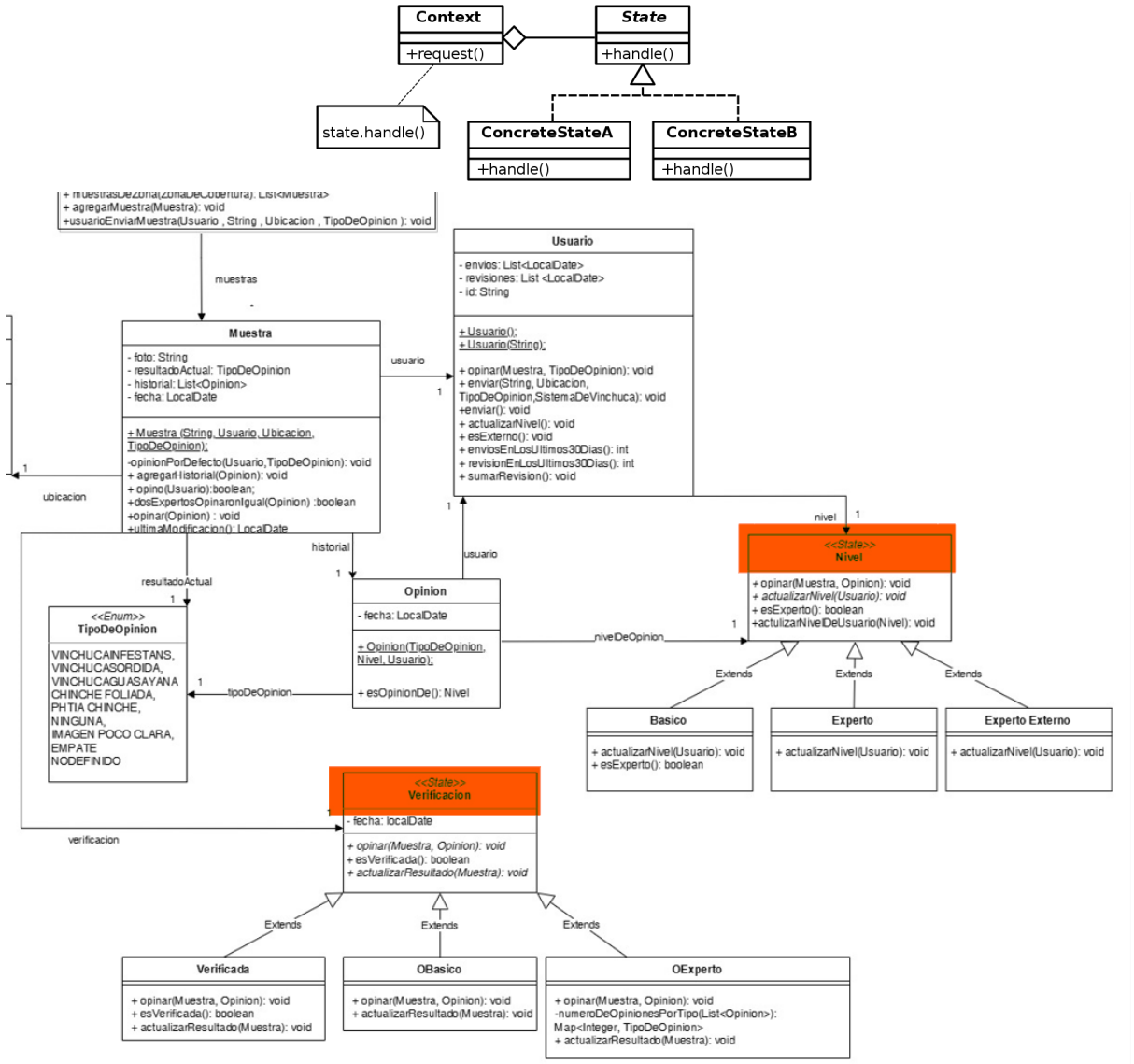
El trabajo consta de realizar un sistema de mapeo de la presencia de Vinchucas en el territorio Argentino. Este mapeo se realiza con la participación de las personas que viven o se encuentran en territorio Argentino, para que puedan enviar fotos y una serie de respuestas a un cuestionario para informar de la aparición del insecto. A partir de los envíos se generan una serie de opiniones, que según el nivel del usuario que opina, deberá tener uno u otro resultado. Además de las opiniones las muestras cuentan con una ubicación, la cual es notificada a diferentes zonas y organizaciones según corresponda.
Todo lo expuesto se tuvo en cuenta para el diseño del trabajo y cada elemento se diseñó a partir de las problemáticas identificadas para la resolución de las mismas.

Como primer paso implementamos el gráfico UML para poder tener un panorama general. Creando las clases e interfaces que necesitábamos, también identificando los patrones de diseño aprendidos en la cursada.



A partir del planteo en papel nos percatamos que necesitábamos una clase sistema que conozca a “Muestra” y a “ZonaDeCobertura”. La misma es la encargada de hacer los filtros de muestras en cada ubicación, las zonas solapadas, entre otros.

Patrón STATE



Cuando creamos la clase “**Usuario**” pudimos comprobar que ésta posee diferente comportamiento dependiendo del Nivel en el cual se encuentre en el Sistema. A partir de ello determinamos una serie de “Estados” por los cuales el usuario transita dinámicamente. A partir de esta identificación decidimos realizar el **Patrón State**, logrando el pasaje de estados del usuario, los que contempla el **Nivel Básico, Nivel Experto y Nivel Experto Externo**.

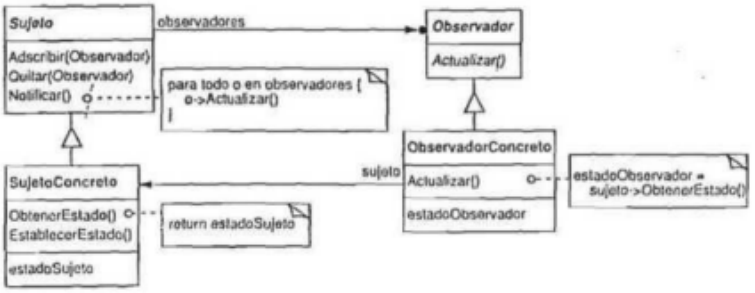
Rol Contexto → Usuario
Rol State → Nivel
Estados Concretos → Basico / Experto / Experto Externo

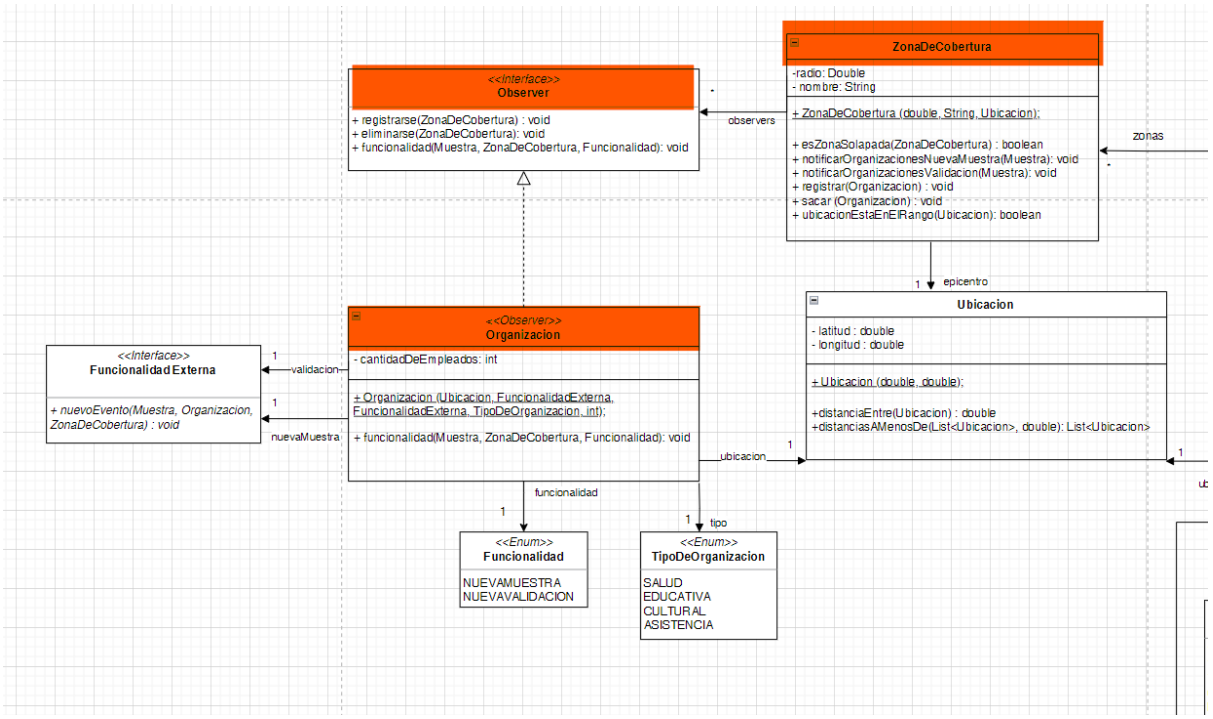
De la misma manera identificamos el patrón State dentro de la clase “**Muestra**”. Notamos los cambios de estadío y aplicamos el patrón dentro del mismo. La muestra tiene un atributo de verificación que va modificándose dentro de los rangos **Verificada, OBasico** (Opinó Básico) y **OExperto** (Opinó Experto). Esto afecta al resultado actual de la Muestra.

Rol Contexto → Muestra

Rol State → Verificacion
Estados Concretos → Verificada / OBasico / OExperto

Patrón OBSERVER

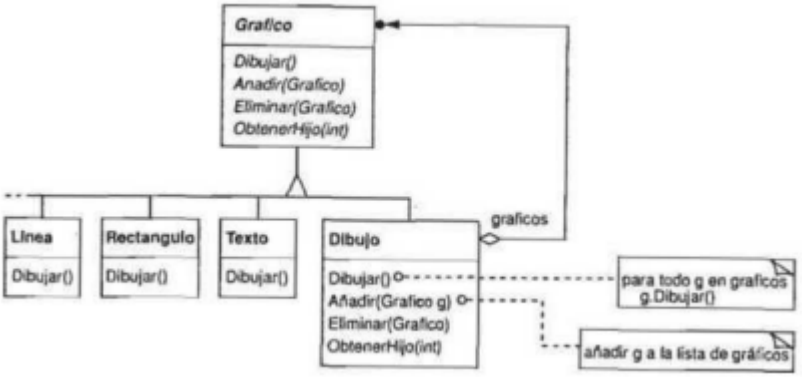


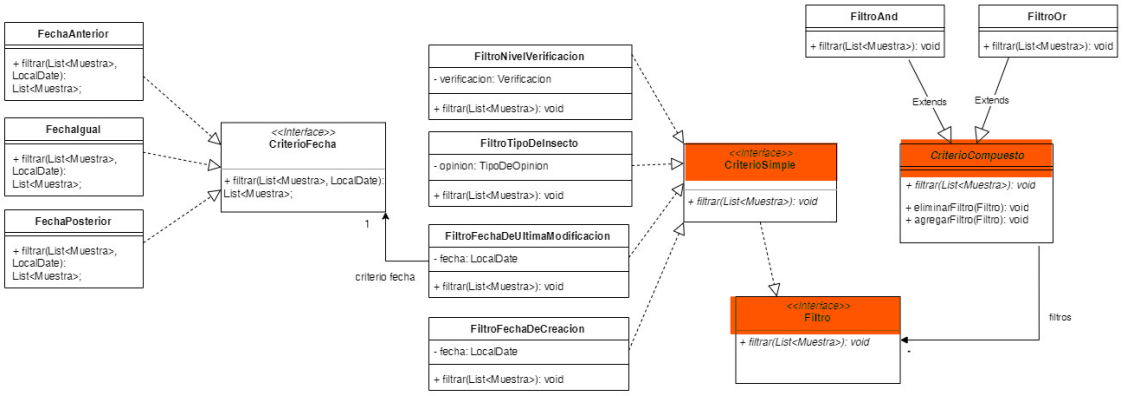


Se decidió implementar el **Patrón Observer** ya que identificamos que las **Organizaciones** debían ser notificadas por cada Evento que suceda en una **Zona** en particular, a la cual debía de estar suscripta.
A partir de la lectura del enunciado, detectamos el patrón Observer ya que cada Organización debía ser notificada solamente por las Zonas a las cuales se tenía interés. Realizamos una serie de métodos que permiten a los observadores ser registrados en las zonas de interés, como también ser eliminados de las mismas, para luego recibir las notificaciones pertinentes.
Hicimos una interfaz **Observador** para que en el caso de querer agregar otro tipo de Observador, que no sean Organizaciones, simplemente pueda implementar la interfaz Observer.

Rol Observer → Observer
Rol Observer Concreto → Organizacion
Rol Subject → ZonaDeCobertura

Patrón COMPOSITE





En este caso decidimos utilizar el **Patrón Composite**, ya que debíamos utilizar filtros Compuestos, que tenían dos o más condiciones para ser evaluadas, cada una de ellas podrían ser criterios simples o a su vez compuestos nuevamente. Por lo tanto los criterios **Simples** y **Compuestos** debían tener una interfaz en común (**Filtro**), lo cual derivó a utilizar el patrón composite ya que los filtros debían tratarse independientemente si eran simples o compuestos.
Cada filtro particular hereda de Criterio Simple, contemplando los filtros que se nos pidió en el enunciado (filtro por nivel de verificación, . Y cada operación de conjunto hereda de Criterio Compuesto, contemplando si es una intersección (caso filtro And) o una unión (caso filtro Or) de los filtros simples.

Rol Component→ Filtro
Rol Composite → Criterio Compuesto
Rol Leaf → Criterio Simple

Descripción de la implementación - Sistema Vinchuca

SistemaVinchuca

Esta es la clase que desglosa el sistema de nuestro trabajo.

Es la encargada de realizar las consultas generales, ya que conoce a todas las muestras y a las zonas del Sistema.

Muestra

Esta es una de las clases centrales, que está compuesta de una foto la cual recibe un String, el resultadoActual que implementa del enumerativo TipoDeOpinion (que se basa en la verificación actual que tenga la muestra), el historial que es una lista de las opiniones de los Usuarios, una fecha y una ubicación.

Usuario

Es otra clase central del proyecto, los usuarios son encargados de enviar y opinar acerca de muestras. El usuario cambia de nivel en base al historial de opiniones emitidas y a la cantidad de fotos enviadas.

Un usuario inicia en Básico y llegado a una cierta cantidad de opiniones y envíos pasa al nivel Experto. Éste se mantiene en Experto, siempre y cuando mantenga esa cantidad de opiniones y envíos con respecto a 30 días anteriores; de no tener esa cantidad solicitada pasa a nivel Básico nuevamente. Además de los mencionados anteriormente, existe el usuario que es Experto Externo, que es ajeno al pasaje de estado.

Opinión

Dependiendo del nivel del Usuario, es el nivel de verificación que tiene la muestra. A partir de esta observación implementamos la clase Opinion que es la que conoce al Usuario y a la Muestra. Lo que determina el resultado actual de la muestra es el historial de opiniones (implementado como una lista de opiniones), dependiendo el nivel que poseían esos usuarios y la cantidad de las mismas.

Debido a que no era necesario crear una clase para cada tipo de opinión, decidimos utilizar un enum, ya que el sentido de crear una clase es darle un comportamiento.

Organización

Esta clase tiene una cantidad de empleados que es un entero, una ubicación (la cual tiene una latitud, longitud y se realiza un cálculo de distancias por medio de esta clase). Existen diferentes tipos de organizaciones, estas son : SALUD, EDUCATIVA, CULTURAL y de ASISTENCIA, implementadas con un enum.

La organización es la encargada de delegar las funcionalidades externas. Estas se diferencian con un Enum, proveniente de la ZonaDeCobertura.

ZonaDeCobertura

Esta clase tiene un double que representa el radio y un String que representa el nombre. También posee una lista de observadores, a los cuales se les va a notificar de una nueva muestra o una nueva validación. Se verifica con uno de los métodos creados en esta clase si una zona se encuentra solapada o no.

FiltroOr y FiltroAnd

Estos filtros fueron pensados como operaciones entre conjuntos. Siendo AND una intersección y OR una unión. Para facilitar estas implementaciones se utilizaron métodos propios de java como “addall” y “retainAll”.

CONCLUSIÓN

Durante el desarrollo del trabajo se diseñó una solución al problema planteado, teniendo en cuenta los PRINCIPIOS SOLID, para asegurar un código escalable.

Se utilizaron varios patrones aprendidos en la materia, los cuales no solo ayudaron a mejorar el código, sino también a la comunicación del equipo, ya que al estar todos al tanto del Patrón, cualquier modificación requerida fue fácil de explicar y aplicar.

COMENTARIO AUXILIAR: Conversamos con el profesor Diego Cano acerca del uso del “getClass” en la operación de filtros (permitido, ya que no lo utilizamos en ningún otro lado en la implementación de nuestro trabajo).