OBJECT-ORIENTED

Trabajo Final - Programacion Orientada a Objetos 2

Integrantes:

Tomas Centurion - ferc721@gmail.com

Diego Kippes - kippes.diego@gmail.com

Camila Scaglioni - camila.pia.scaglioni@gmail.com

Profesores: Diego Torres, Diego Cano, Matias Butti

Fecha de entrega: 12/06/2022

Decisiones del modelo

Decidimos incluir una aplicación web porque esta recibe los datos de las muestras y las procesa, al igual que un usuario puede registrarse.

Los usuarios están implementados de forma tal en que estos puedan evolucionar, pasar de básico a experto, o ser especialista. Lo cual nos llevó a pensar en un **patrón state**. También, los usuarios pueden ser clonables, en caso de que se quiera hacer un snapshot de su referencia en ese momento. Por ejemplo, cuando un usuario vota una muestra la opinión clona al usuario que votó para no perder su estado (**Knowledge**) en la votación.

Además, las muestras también pueden evolucionar de tipo de votación por usuarios básicos a expertos, hasta que pueda ser verificada. Al igual que con la evolución de los usuarios pensamos en un patrón State, ya que este puede actualizar su estado de votación de la cantidad de opiniones enviadas, así como si un experto voto u fue verificada por 2 expertos.

Por otro lado, para lograr polimorfismo, creamos una interfaz llamada "**Generic Opinión Type**", que es implementada por los Enums "**OpinionType**" y por "**UndefinedOpinion**".

El objetivo de dicha decisión es que el usuario pueda emitir solo opiniones de "**OpinionType**" sin poder utilizar "**UndefinedOpinion"**. Por otra parte, para el filtrado de insectos utilizamos una interfaz genérica, la cual nos permite buscar por "**OpinionType**" y las Opiniones "**Undefined**", esto también se usa para sacar el resultado actual de una muestra.

En la zona de cobertura usamos el **patrón observer** ya que las organizaciones pueden suscribirse a una o varias zonas de cobertura. Para hacer la intersección de zonas se pensó en medir el área de ambas zonas y ver si se intersectan.

Se puso la **funcionalidad externa** para las organizaciones así ellas pueden notificar los eventos de nueva muestra y muestras verificadas.

La búsqueda de muestras se realizan a través de varios filtros individuales:

● Fecha de creación de la muestra. (**FilterCreationDate**)

● Fecha de la última votación. (**FilterLastVoteDate**)

● Tipo de insecto detectado en la muestra. (**FilterInsect**)

● Nivel de verificación (votada o verificada) (**FilterVerificationLevel**)

Estos filtros pueden ser combinables mediante los filtros binarios And y Or, los cuales extienden de la clase abstracta BinaryFilter.

En este caso se pensó en un **patrón Composite** para poder combinar los filtros de cualquier manera deseada e ilimitada. Otro punto es que se utilizó un **patrón Strategy** para los comparadores de mayor, menor e igual utilizados para los filtros por fechas.

En el diseño de UML se incluyen **excepciones** propias para tener un mejor manejo de los posibles errores.

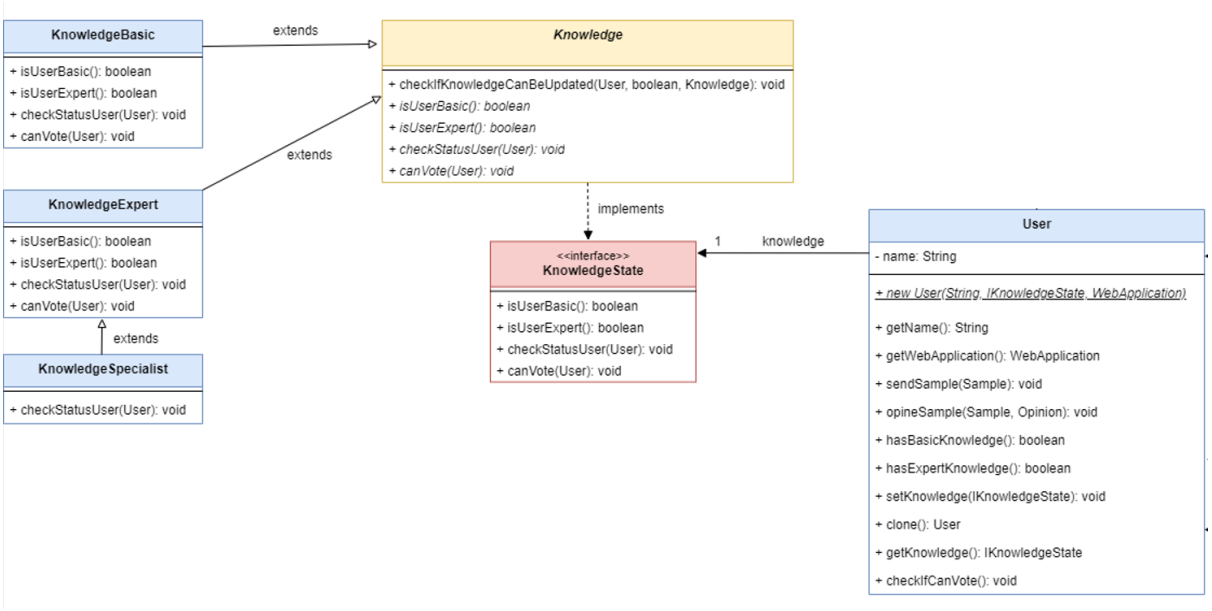
Organización

Como equipo primero desglosamos las tareas, y usando un kanban pusimos las posibles tareas que podríamos tener a medida que pensábamos en el diseño de la aplicación. Por otro lado, una de las mayores dificultades dentro del TP fue cómo íbamos a delegar cada tarea, ya que la parte de los patrones states, o diferentes mensajes de usuarios estaban atados a las muestras y las muestras estaban atadas a las opiniones. Luego de crear una estructura básica esa dificultad mermó en el resto del desarrollo.

El mayor desafío del tp fue identificar los diferentes patrones posibles y cómo implementarlos en base al enunciado. Por otra parte, el mock de los tests nos costó implementarlo, así que como primera solución usamos los objetos reales antes de mockearlos.

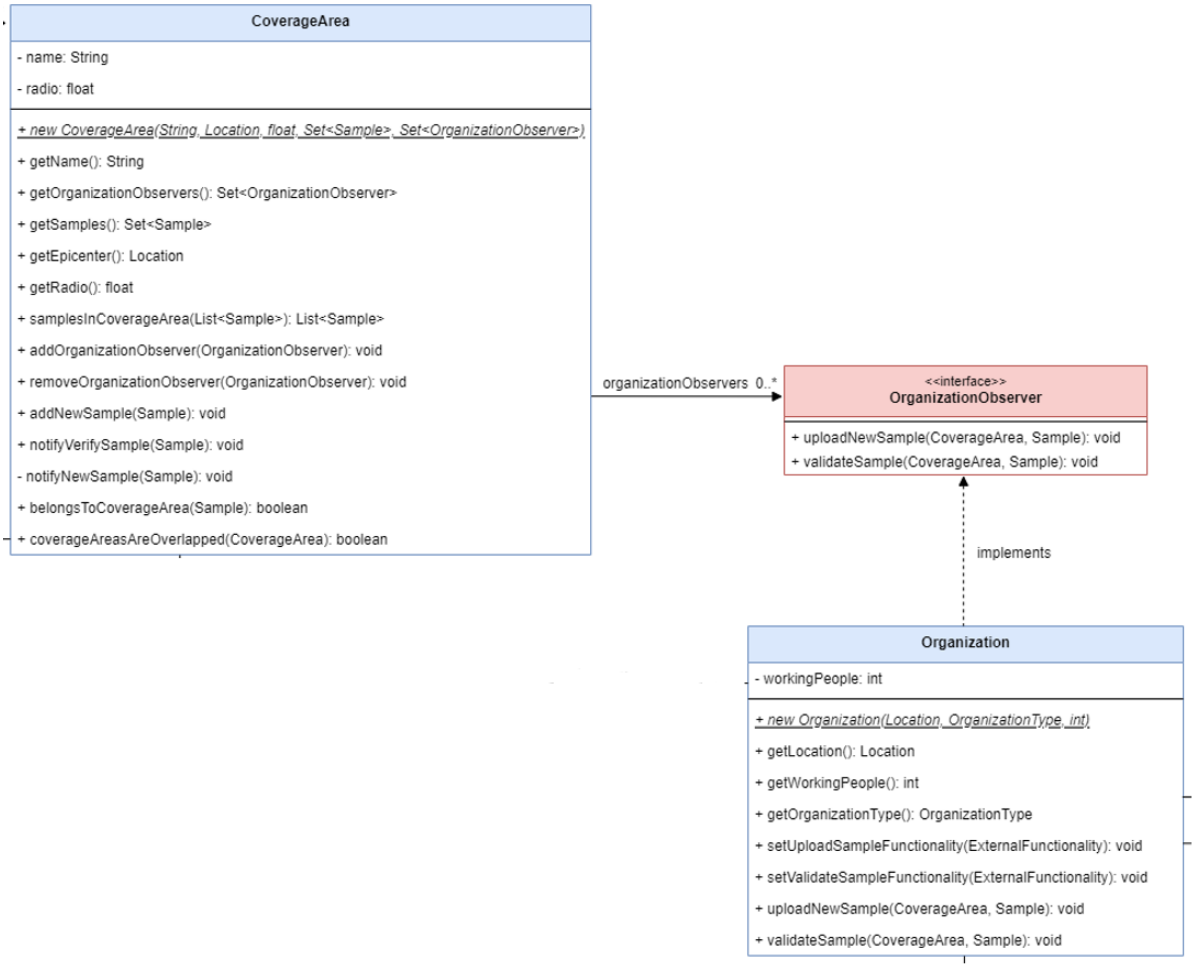
Patrones de diseño y roles

Patrón State



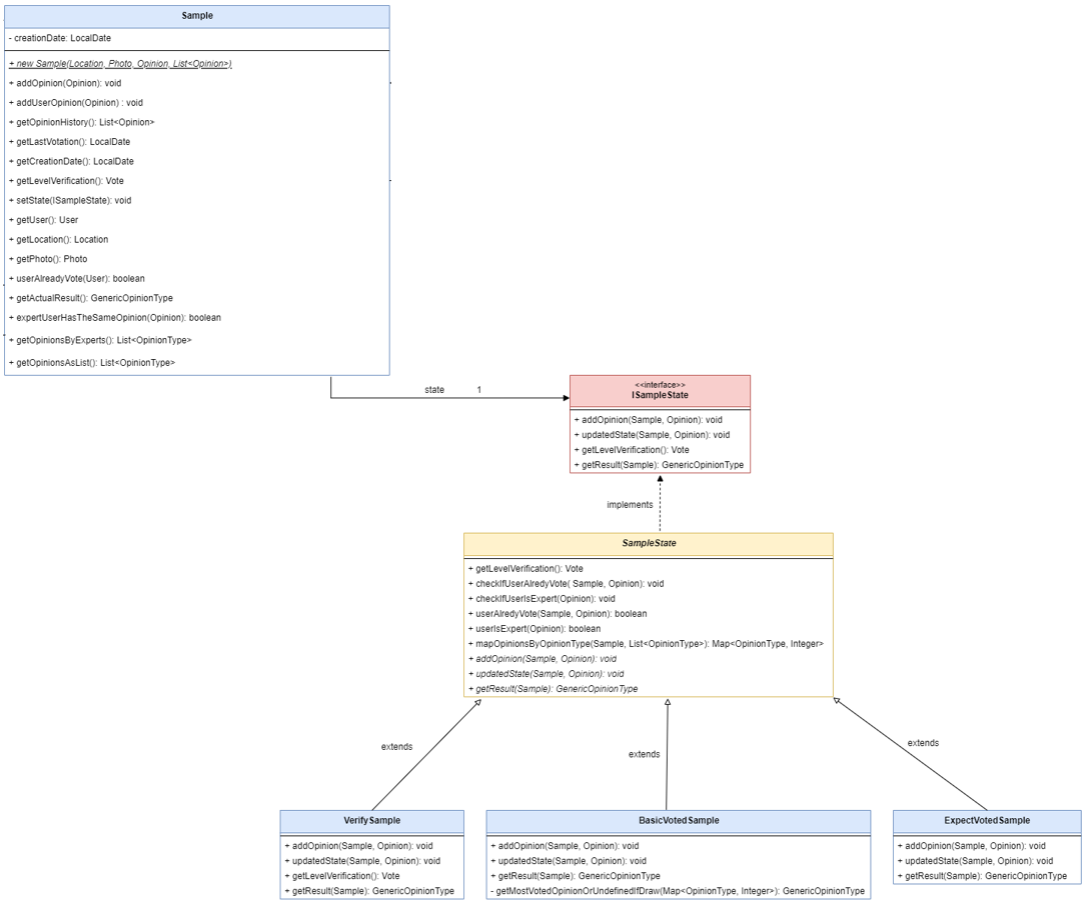
User	Context
KnowledgeState	State
Knowledge	State
KnowledgeBasic	Concrete State Subclasses
KnowledgeExpert	Concrete State Subclasses
KnowledgeSpecialist	Concrete State Subclasses

Patrón Observer



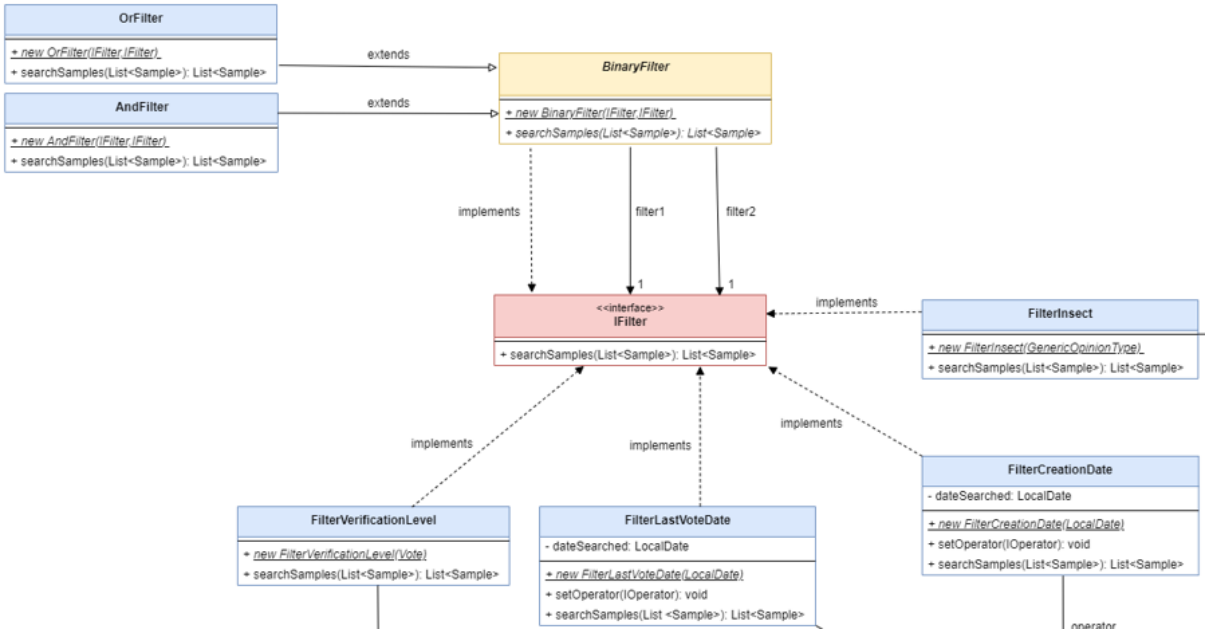
CoverageArea	Concrete Subject (Observable)
OrganizationObserver	Observer
Organization	Concrete Observer

Patrón State



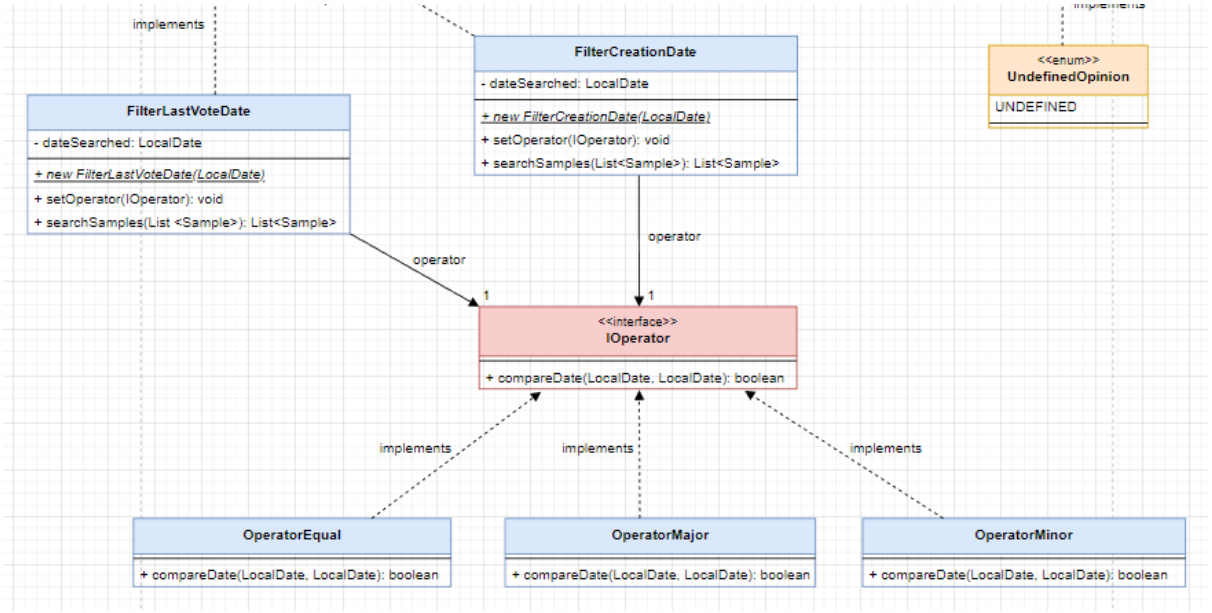
Sample	Context
ISampleState	State
SampleState	State
VerifySample	Concrete State Subclasses
BasicVotedSample	Concrete State Subclasses
ExpectedVotedSample	Concrete State Subclasses

Patrón Composite



IFilter	Component
AndFilter	Composite Subclass
OrFilter	Composite Subclass
BinaryFilter	Composite
FilterVerificationLevel	Leaf
FilterLastVotedDate	Leaf
FilterCreationDate	Leaf
FilterInsect	Leaf

Patrón Strategy



FilterCreationDate	Context
FilterLastVoteDate	Context
IOperator	Strategy
OperatorEqual	Concrete Strategy
OperatorMajor	Concrete Strategy
OperatorMinor	Concrete Strategy