Observer

Objective

Define one or more dependencies between objects so that if an object changes its state, these dependencies are automatically reported and updated.

Function

Define a one-to-many dependency between objects, so that when one changes state, all the objects that depend on it are automatically notified and updated.

Structure

As shown in figure 1

- IObservable: Interface that must implement all the objects that want to be observed, it defines the minimum methods that must be implemented.

- ObservableConcrete: Class that wants to be observed, it implements IObservable and must implement its methods.

- IObserver: Interfaces that must implement all the objects that want to observe the changes of IObservable.

- ObservableConcrete: Concrete class that is attentive to the changes of IObserver, this class inherits from IObserver and must implement its methods.

The structure that meets this pattern is shown in Figure 1

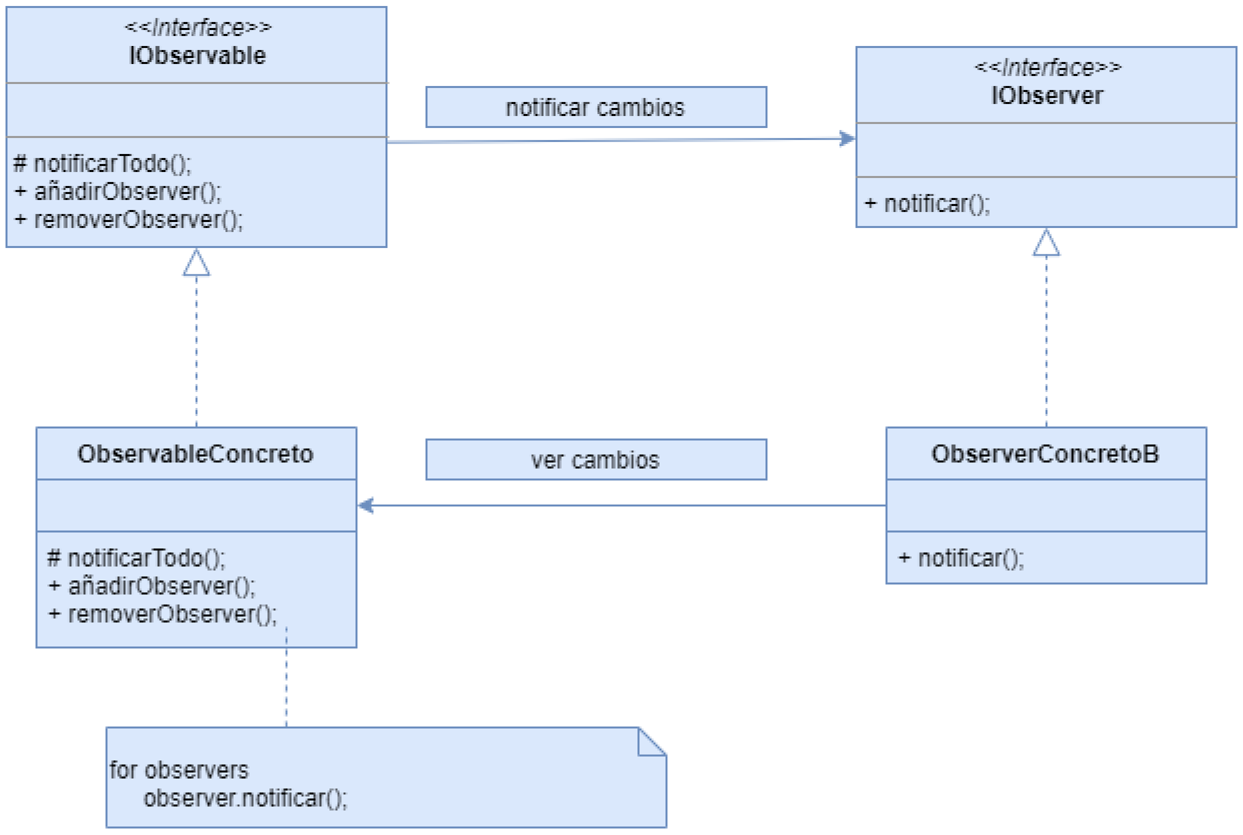

Figure 1: UML Diagram Observer Pattern

Applications

The use of the Observer pattern is recommended when:

- An abstraction requires two aspects, one dependent on the other; which need to be encapsulated in the separate objects so that they can vary and be used independently.
- Changing one object requires changing others, and it is not known exactly how many.
- An object must notify others without needing to know who they are specifically.

Design Patterns Collaborators

- The Mediator pattern can act as a mediator between subjects and observers, encapsulating the semantics of complex updating.
- The Observer pattern can make use of the Singleton pattern to make it uniquely and globally accessible to the subject.

Scope of action

Applied at the object level.

Problem

The application requires the creation of dependencies between objects that allow to update the relations between them in the moment that one varies, in the conventional development this process implies redundant code and wide consumption of resources.

Solution

The Observer pattern describes how to establish these relationships; using two essential objects: "subject" and "observer" that perform the process known as publish-subscribe; that is, subject can handle any number of dependent observers, who are notified by means of a "publication" when a subject suffers some kind of change in his status, at that moment observer will consult him to synchronize his status with that of the subject.

Diagram or Implementation

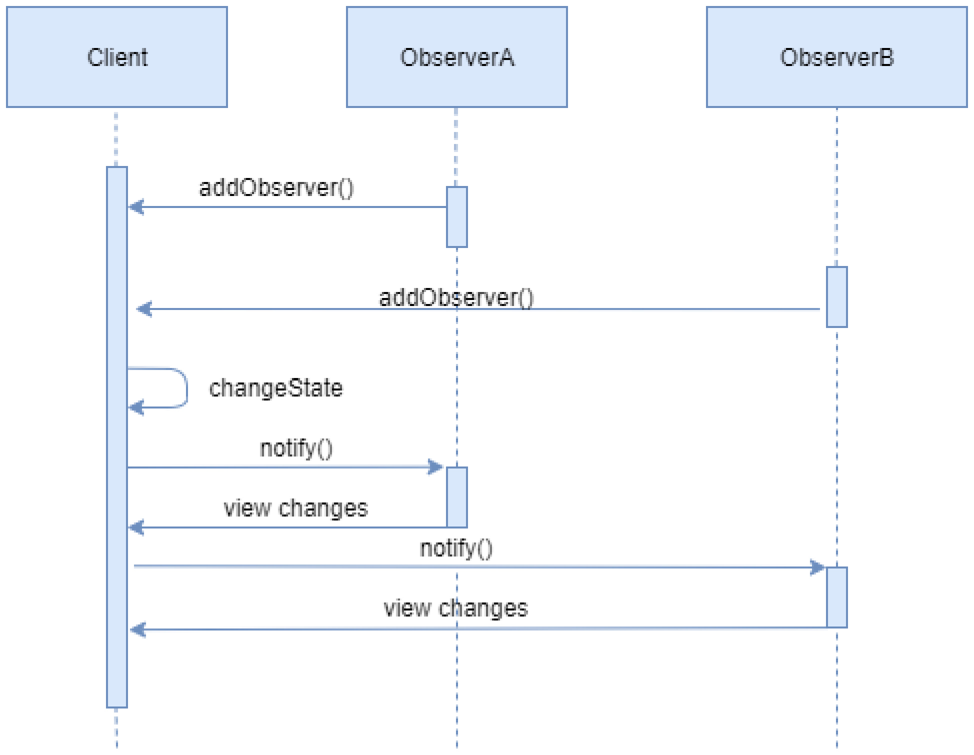

Figure 2: UML Diagram Observer Pattern

Figure 2 explains the behaviour of the pattern by means of a sequence diagram.

- The ObserverA component is registered with the client object to be notified of any changes.

- The ObserverB component is registered with the client object to be notified of any changes.

- There is some change in the client's status.

- All Observers are notified of the change.