Strategy

Objective

Define a group of algorithms, encapsulating each one and making it interchangeable; this allows the algorithm to vary independently of the client classes that use it.

Function

Define a family of algorithms, encapsulated and interchangeable. The strategy allows the algorithm to vary independently of the customers using it.

Structure

As shown in figure 1

- Context: Component that encapsulates the strategy to be used, has as a feature that you can establish the strategy to be used at runtime.

- IStrategy: Common interface that all strategies should implement. In this interface the operations that the strategies will have to implement are defined.

- StrategyConcrete: Represents the concrete strategies, which are inherited from IStrategy.

The structure that meets this pattern is shown in Figure 1

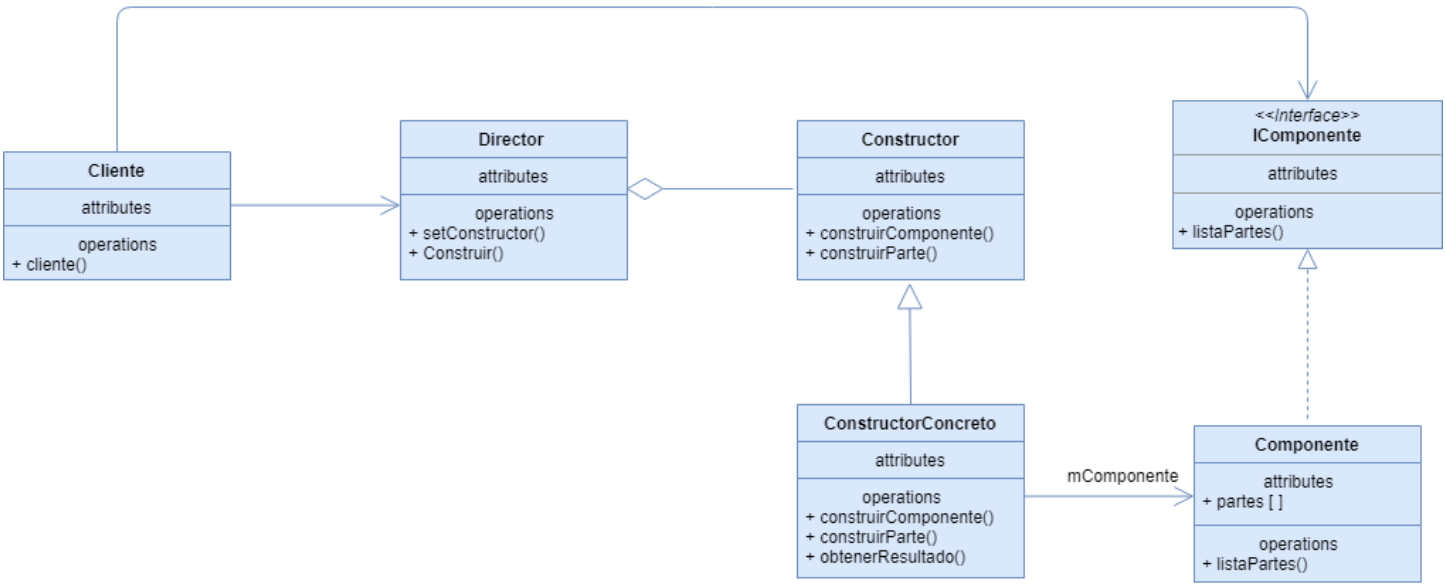

Figure 1: UML Diagram Strategy Pattern

Applications

The use of the Strategy pattern is recommended when:

- Several related classes only differ in their behavior; the strategies provide a way to set up a class with one of many behaviors.

- The application needs different variants of the algorithm; that is, an algorithm could be defined that reflects the exchanges of a variable.

- An algorithm is required to use data that customers should not know; the pattern avoids exposing the complex structures of the data.

- A class must define many behaviors; which appear as multiple conditioning statements in its operations and the pattern is in charge of relating them in branches in its own class.

Design Patterns Collaborators

- The Strategy pattern can eventually develop suitable Flyweight objects.

Scope of action

Applied at the object level.

Problem

The application requires that one of the main design concepts "open-close" is employed; to achieve this goal the details of the interfaces are encapsulated; but this does not cover the aspects of class changes and the impact of the implementation of derived classes.

Solution

The Strategy pattern creates a set of encapsulated and related algorithms in a conductive class called "Context"; the client program can select one of the algorithms or in some cases Context takes care of selecting the most suitable one according to the situation. The key point is that it allows easy switching between algorithms.

Diagram or Implementation

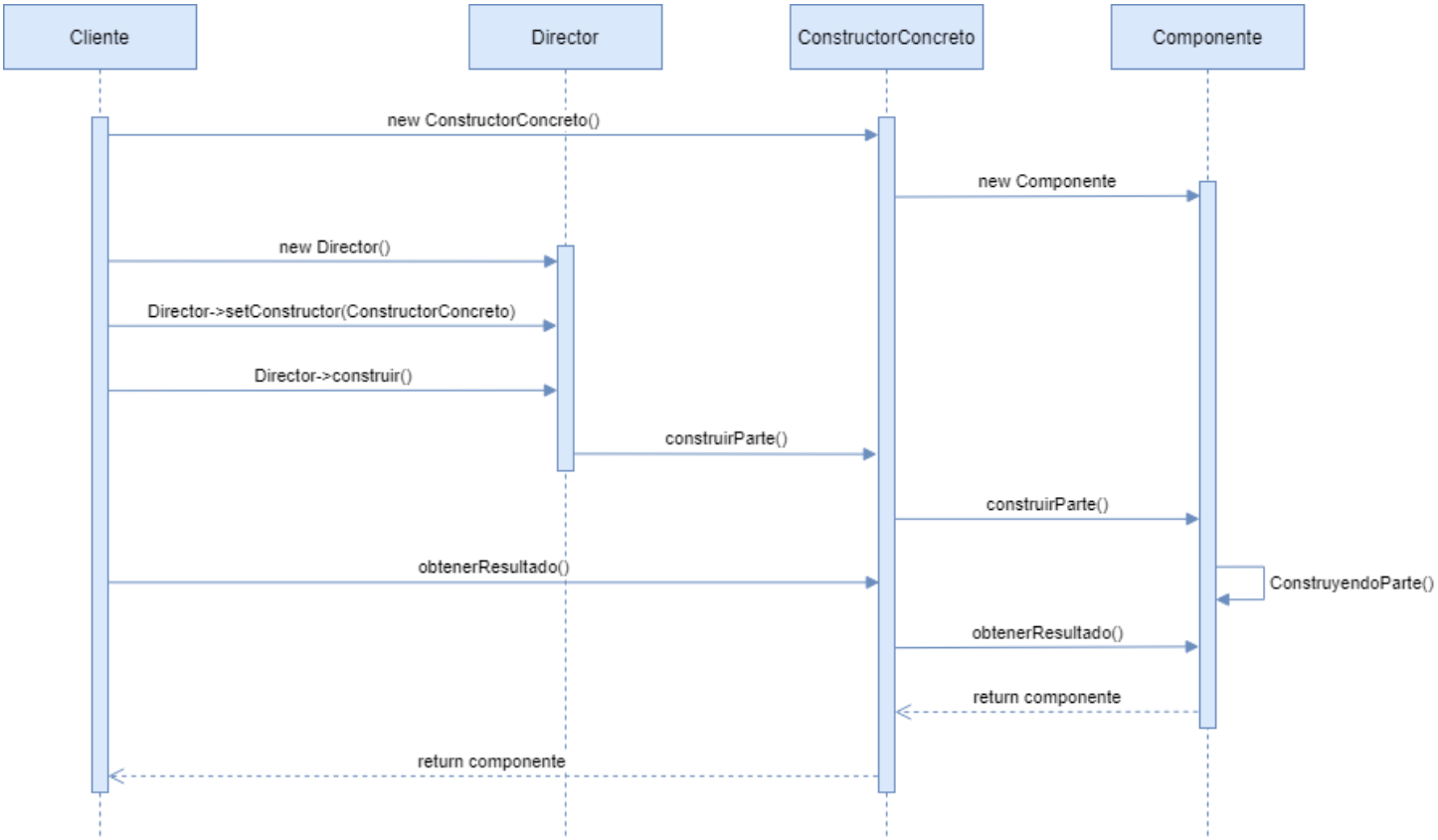

Figure 2: UML Diagram Strategy Pattern

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

- The client class creates a new context component and establishes strategy A.
- The client class executes the doSomething operation.
- The Context component in turn delegates this responsibility to ConcreteStrategyA.
- ConcreteStrategyA performs the operation and returns the result.
- The Context component takes the result and returns it to the client class.
- The client class changes the strategy to the Context component at runtime.
- The client class executes the doSomething operation again.
- The Context component in turn delegates this responsibility to ConcreteStrategyB.

- ConcreteStategyB performs the operation and returns the result.

- The Context component takes the result and returns it to the client class.