# Bridge

## Objective

Separate an abstraction from its implementation, so that both may vary independently.

## Function

Decouple an abstraction from its implementation.

## Structure

The client class does not want to deal with platform-dependent details. The Bridge pattern encapsulates this complexity behind a 'wrap'. of abstraction.Bridge emphasizes the identification and decoupling of "interface" abstraction from "implementation" abstraction.

The structure that meets this pattern is shown in Figure 1

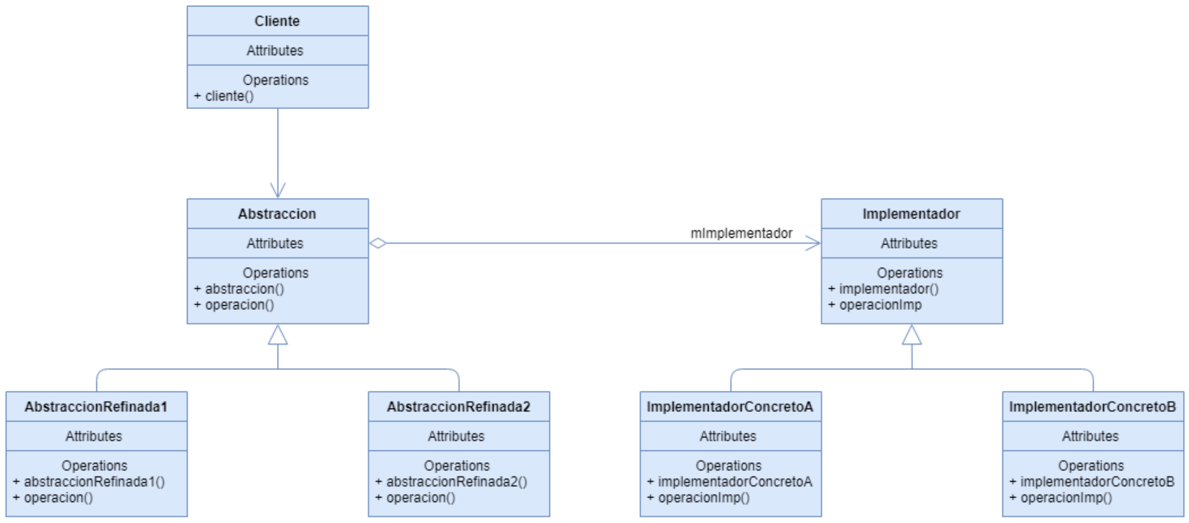

Figure 1: UML Diagram Bridge Pattern

## Applications

- You want to share an implementation among multiple objects, and that this fact is transparent to the client.

- A permanent link between an abstraction and its implementation needs to be removed, for example, when an implementation needs to be linked or changed at run time.

- The need to implement various representations of an abstraction can generate a proliferation of classes.

- Abstraction and implementation must be extensible through aggregation of subclasses. In this case the bridge design pattern allows the combination of different abstractions and implementations and extend them independently.

- Changes in the implementation of an abstraction should not impact the client classes, their code should not be recompiled.

## Design Patterns Collaborators

- The Abstract Factory pattern can create and set a certain bridge pattern.

## Scope of action

Applied at the object level.

## Problem

To make use of different representations of an abstract class, it is required to implement an inheritance since an abstract class defines the interface for such abstraction and the concrete, or inherited, classes implement the different ways, however this implementation is limiting, considering that permanently linked to abstraction, which makes it difficult to modify, extend, and reusing abstractions and implementations independently.

## Solution

The bridge design pattern allows a bridge to connect a abstraction of its different implementations, which creates a hierarchy for the abstractions and another for the implementations, allowing to manage both class hierarchies independently.

## Diagram or Implementation

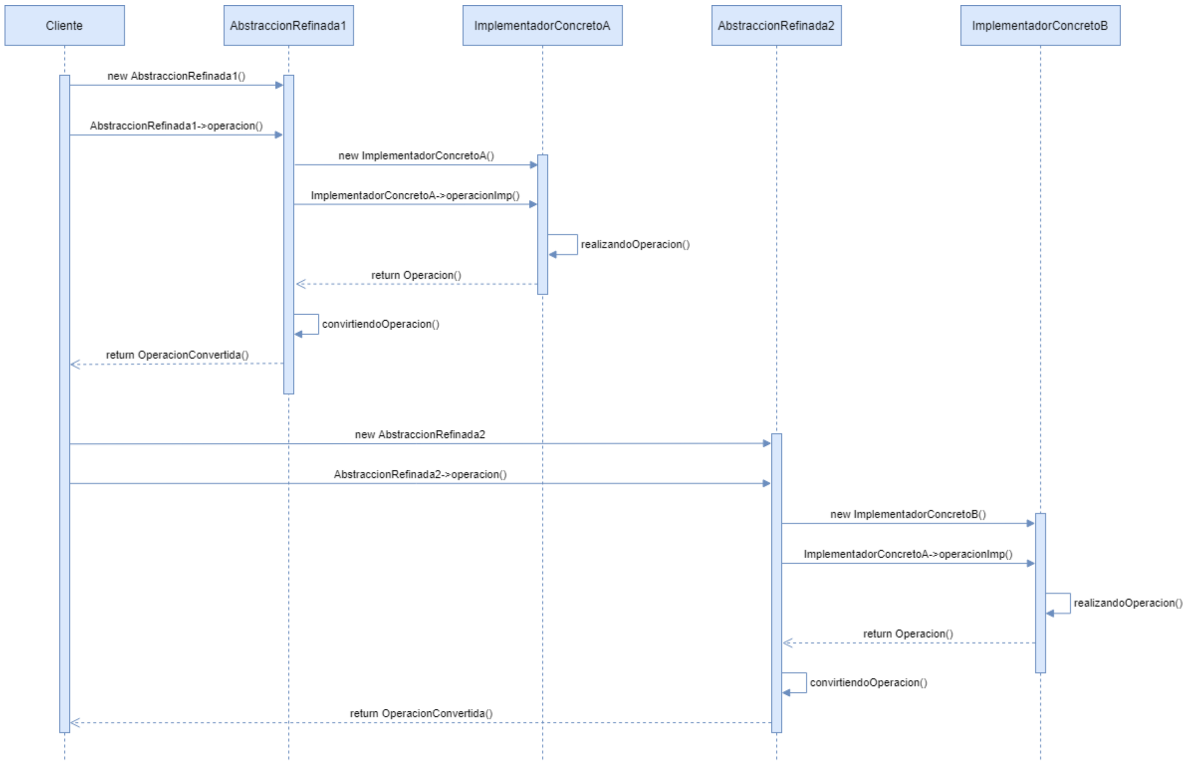
Cliente
AbstraccionRefinada1
ImplementadorConcretoA
AbstraccionRefinada2
ImplementadorConcretoB
new AbstraccionRefinada1()
AbstraccionRefinada1->operacion()
new ImplementadorConcretoA()
ImplementadorConcretoA->operacionImp()
realizandoOperacion()
return Operacion()
convirtiendoOperacion()
return OperacionConvertida()
new AbstraccionRefinada2
AbstraccionRefinada2->operacion()
new ImplementadorConcretoB()
ImplementadorConcretoA->operacionImp()
realizandoOperacion()
return Operacion()
convirtiendoOperacion()
return OperacionConvertida()

Figure 2: UML Diagram Bridge Pattern

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

- Client class executes an AbstractionImplement operation.
- The class AbstractionImplement replicates the request to ConcreteImplementor, in this step the class AbstractionImplement could perform a conversion of the parameters to execute the ConcreteImplementor.
- ConcreteImplementor returns the results to the AbstractionImplement class.
- Finally the AbstractionImplement class converts the results of the ConcreteImplementor to be returned to the client.

# Study Cases

## Drawing Editor System

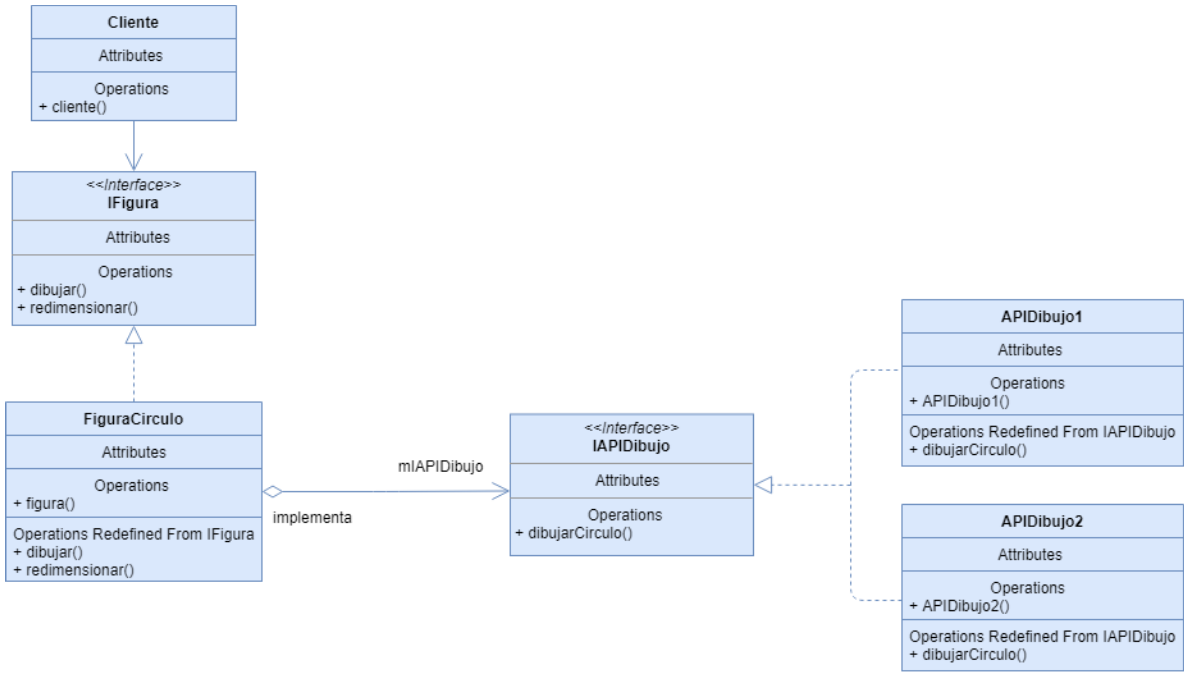

Figure 3: UML Diagram Drawing Editor System

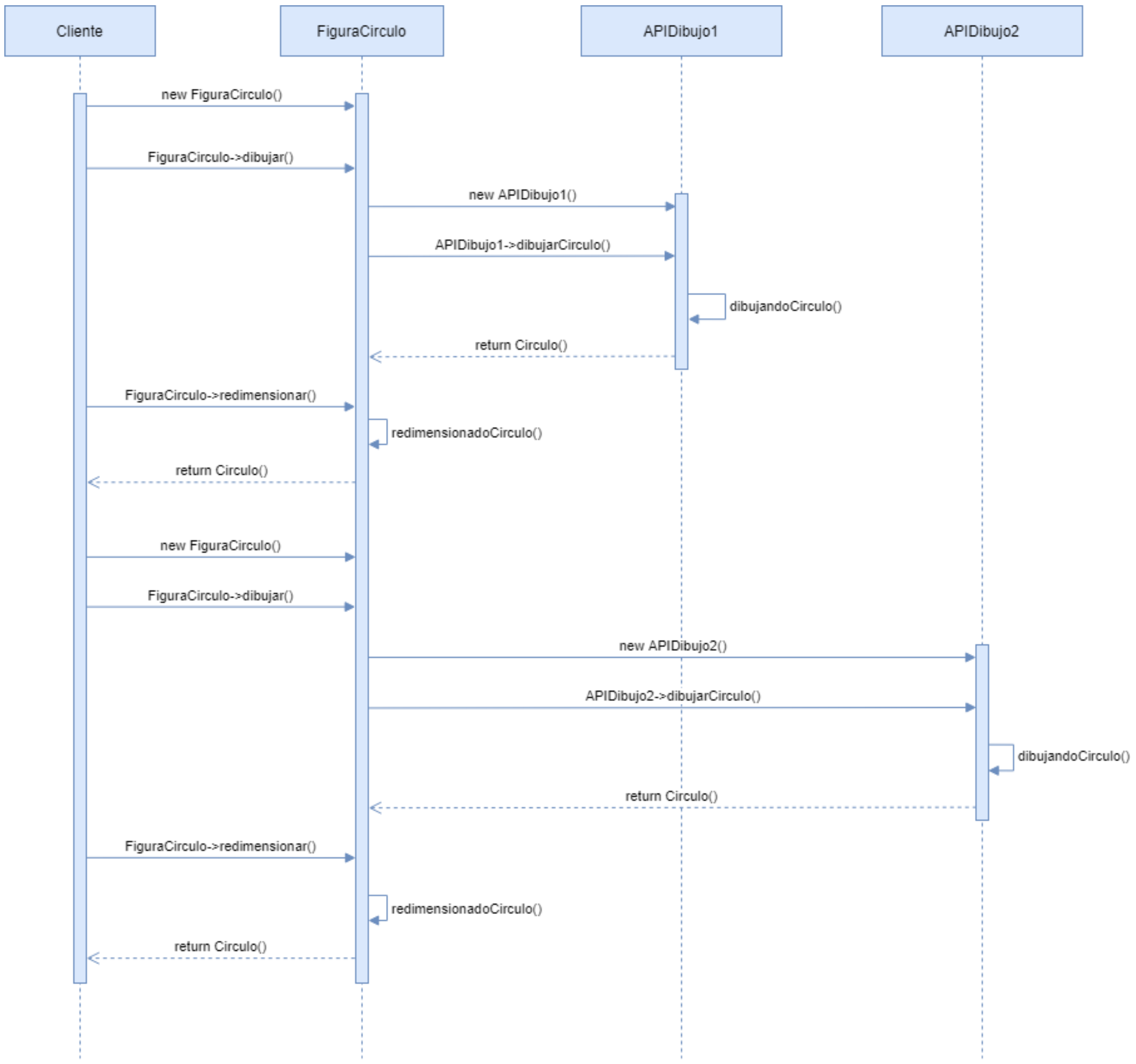

Figure 4: UML Diagram Drawing Editor System

## Tourist Reservation System

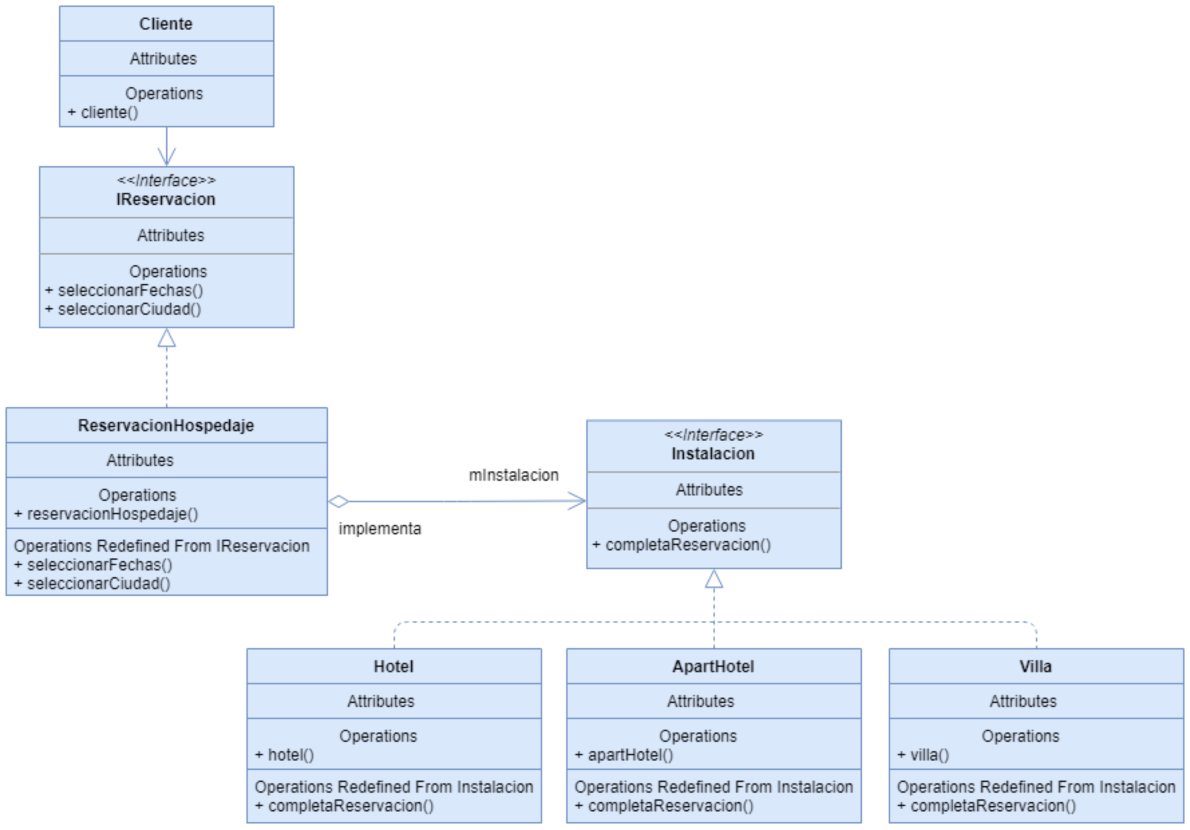

Figure 5: UML Diagram Tourist Reservation System

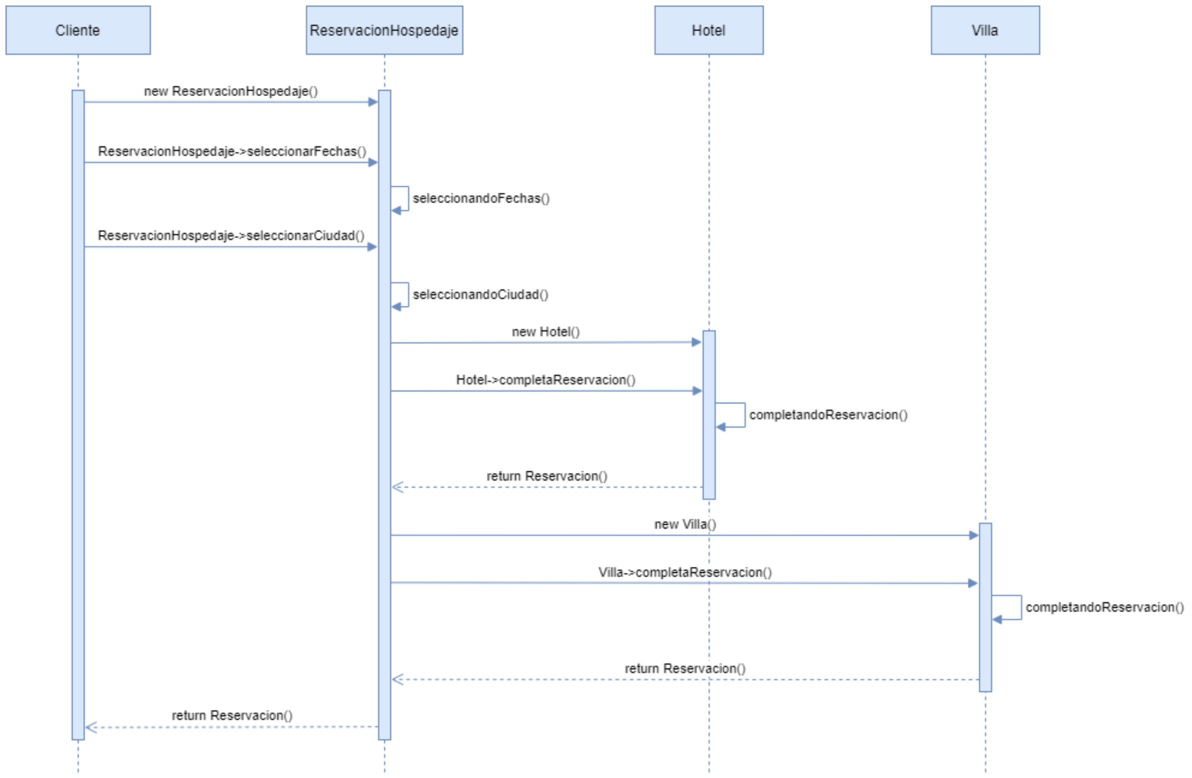

Figure 6: UML Diagram Tourist Reservation System