Capitulo 4. Patrones de Diseño

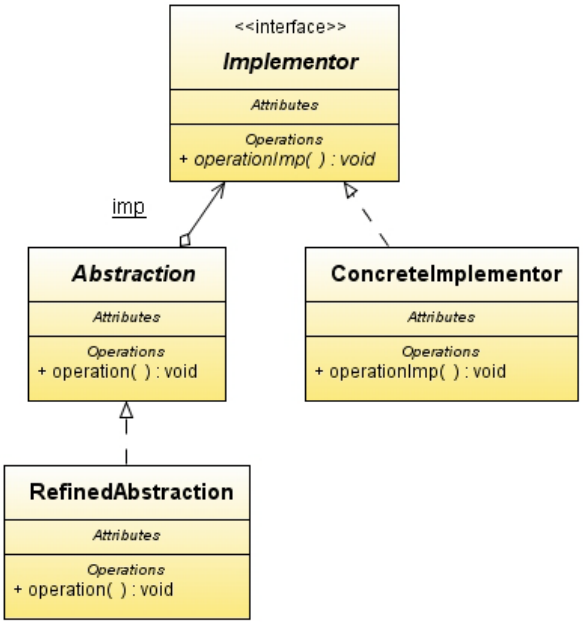
En el presente capítulo se definirán los patrones de diseño usados durante el desarrollo del sistema de ésta tesis. Cada patrón será definido y se presentará su diagrama UML de clases. El orden en el cual se presentan cada uno de los patrones va de acuerdo a su uso en el siguiente capítulo.

4.1 Patrón “Bridge”

Definición:

El patrón “Bridge” desacopla la abstracción de la implementación por lo tanto ambas pueden variar de forma independiente [Gamma, 1995].

Diagrama UML de clases del patrón “Bridge”:

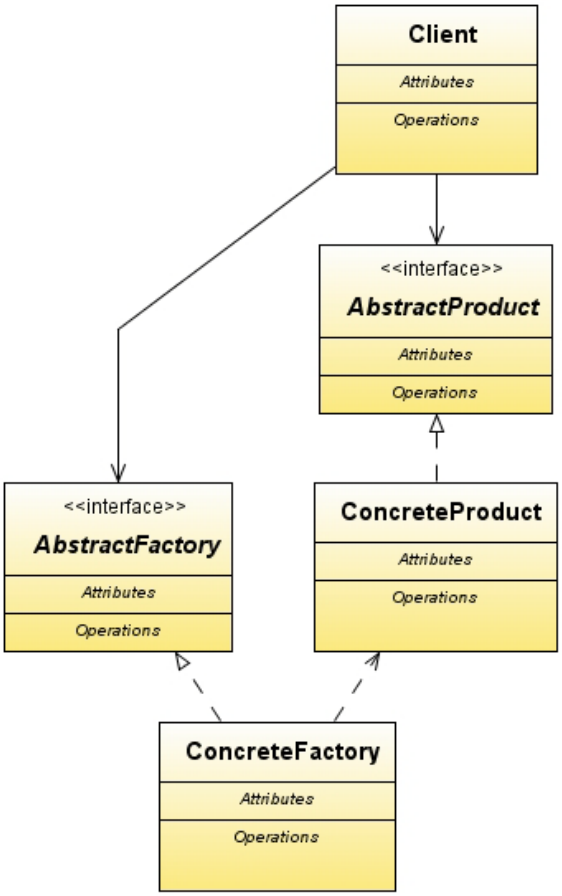


4.2 Patrón “Abstract Factory”

Definición:

El patrón “Abstract Factory” provee una interface para crear una familia de objetos similares o dependientes sin especificar sus clases concretas [Gamma, 1995].

Diagrama UML de clases del patrón “Abstract Factory”:

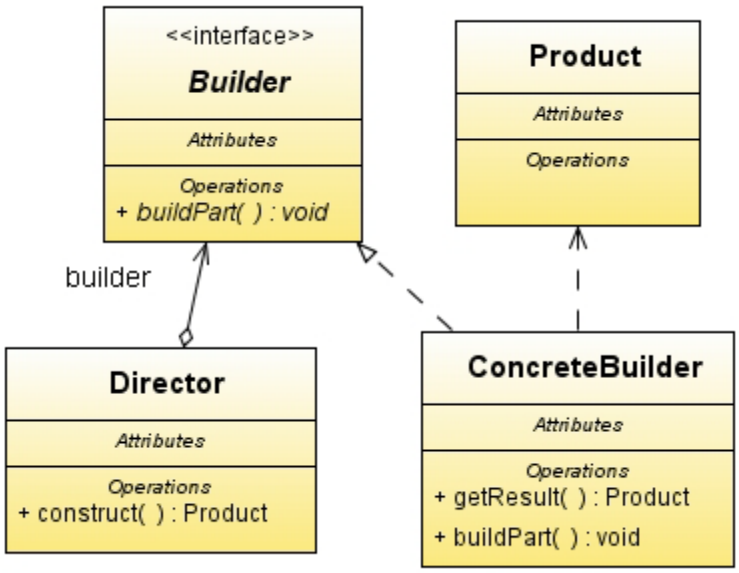


4.3 Patrón “Builder”

Definición:

El patrón “Builder” separa la construcción de la representación de un objeto complejo, por lo tanto la misma construcción puede crear diferentes representaciones [Gamma, 1995].

Diagrama UML de clases del patrón “Builder”:

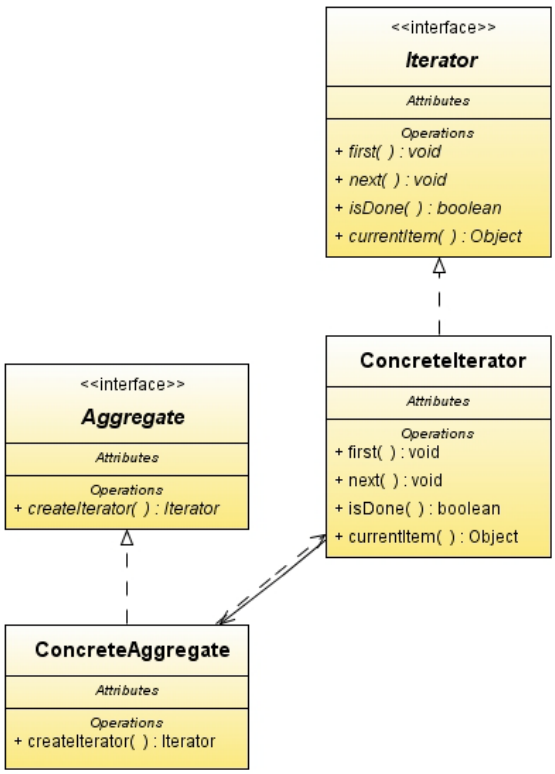


4.4 Patrón “Iterator”

Definición:

El patrón “Iterator” provee una manera de acceder elementos de un objeto agregado sin exponer su representación interna [Gamma, 1995].

Diagrama UML de clases del patrón Iterador:

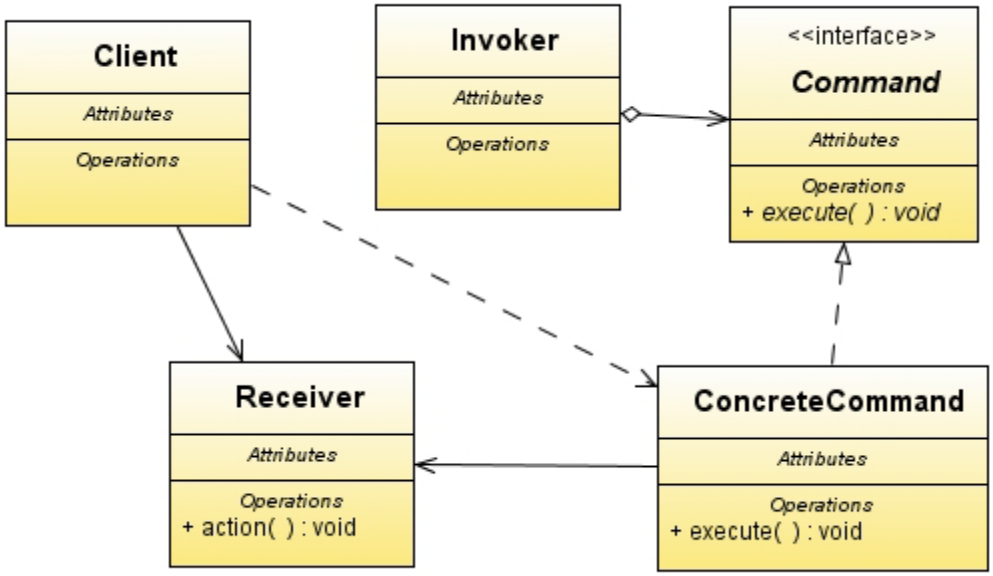


4.5 Patrón “Command”

Definición:

El patrón “Command” encapsula una petición como objeto, por lo tanto permite dar como parámetros otros objetos con diferentes peticiones, colas o peticiones de registro, además soporta el deshacer operaciones [Gamma, 1995].

Diagrama UML de clases del patrón “Command”:

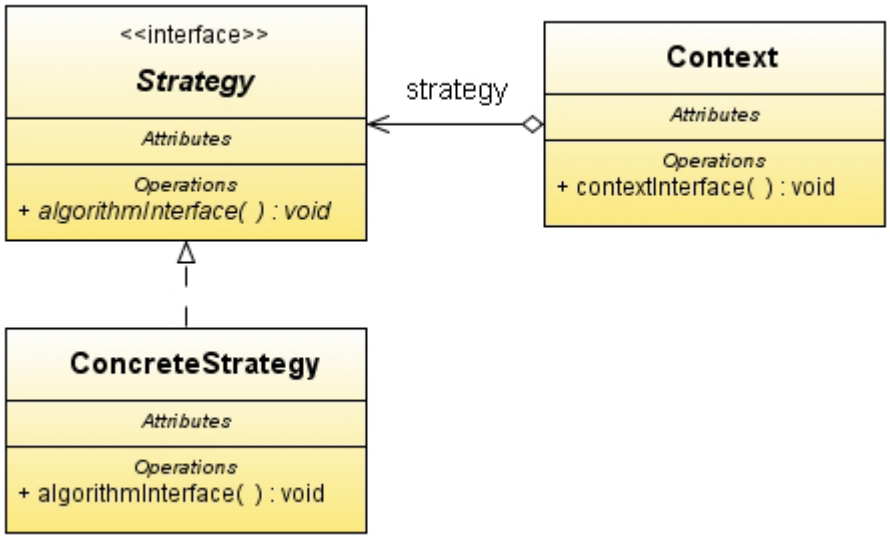


4.6 Patrón “Strategy”

Definición:

Define una familia de algoritmos, los encapsula, y los hace intercambiables. Este patrón permite que los algoritmos varíen de forma independiente de los clientes que lo usan [Gamma, 1995].

Diagrama UML de clases del patrón “Strategy”:

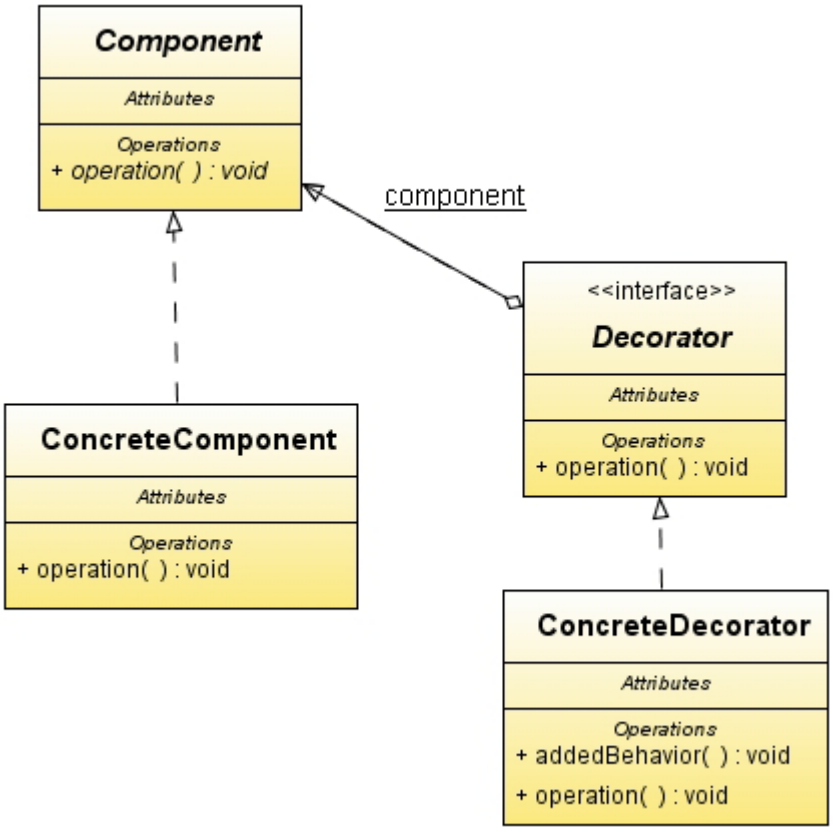


4.7 Patrón “Decorator”

Definición:

El patrón Decorador atribuye responsabilidades adicionales a un objeto de forma dinámica. Decorador provee una alternativa flexible de subclases para extender funcionalidad [Gamma, 1995].

Diagrama UML de clases del patrón Decorador:

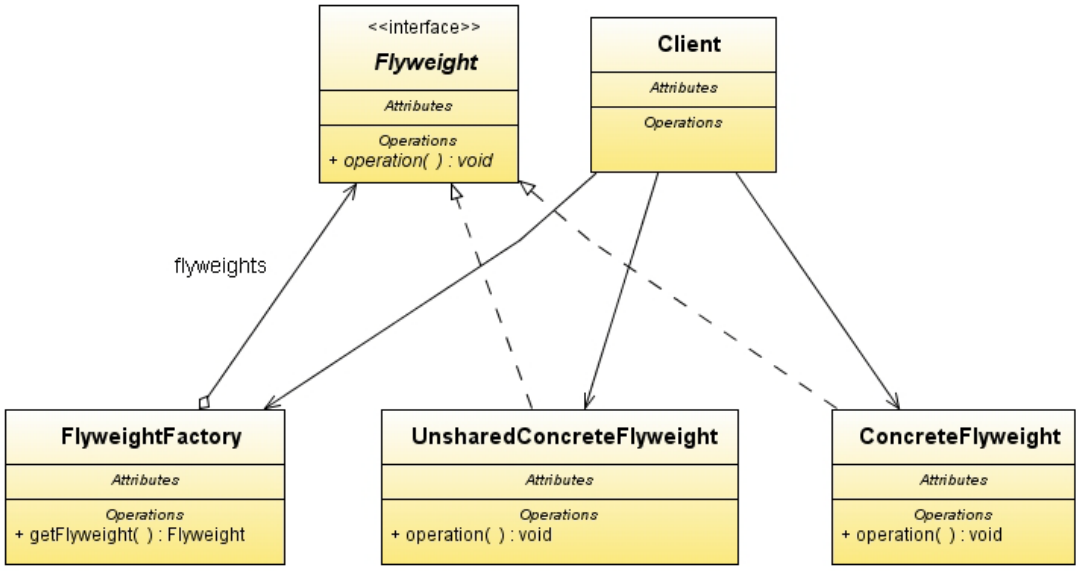


4.8 Patrón “Flyweight”

Definición:

El patrón “Flyweight” comparte estados para soportar un gran número de objetos pequeños aumentando la eficiencia en espacio [Gamma, 1995].

Diagrama de clases UML del patrón “Flyweight”:

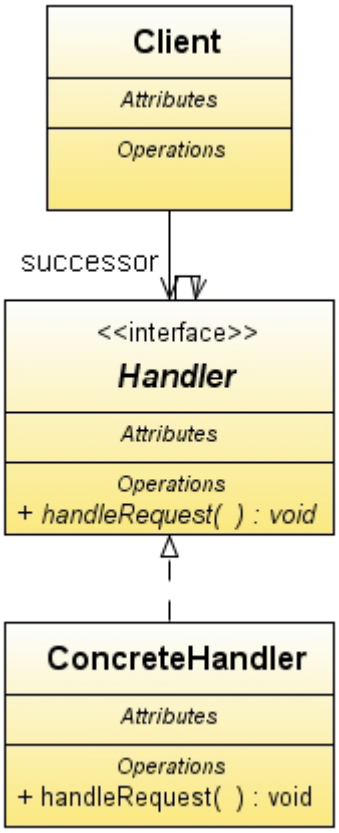


4.9 Patrón “Chain of Responsability”

Definición:

El patrón “Chain of Responsability” evita acoplar el emisor de una petición de su receptor, dando la posibilidad a más de un objeto de responder a una petición. Encadena los objetos receptores y pasa dicha petición a través de la cadena hasta que es procesada por algún objeto [Gamma, 1995].

Diagrama UML de clases del patrón “Chain of Responsability”

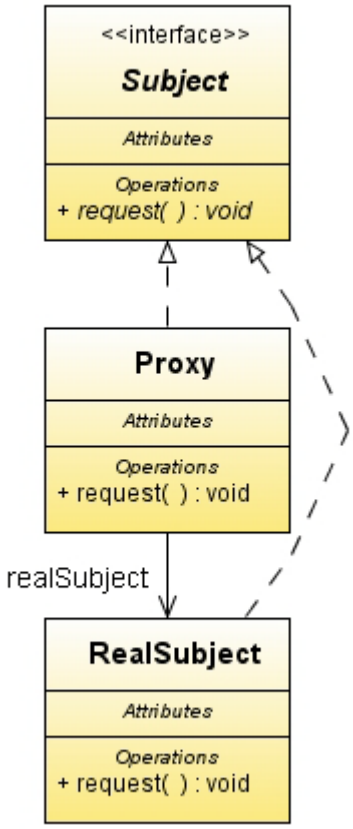


4.10 Patrón “Proxy”

Definición:

El patrón “Proxy” provee un sustituto para controlar el acceso a otro objeto [Gamma, 1995].

Diagrama UML de clases del patrón “Proxy”.

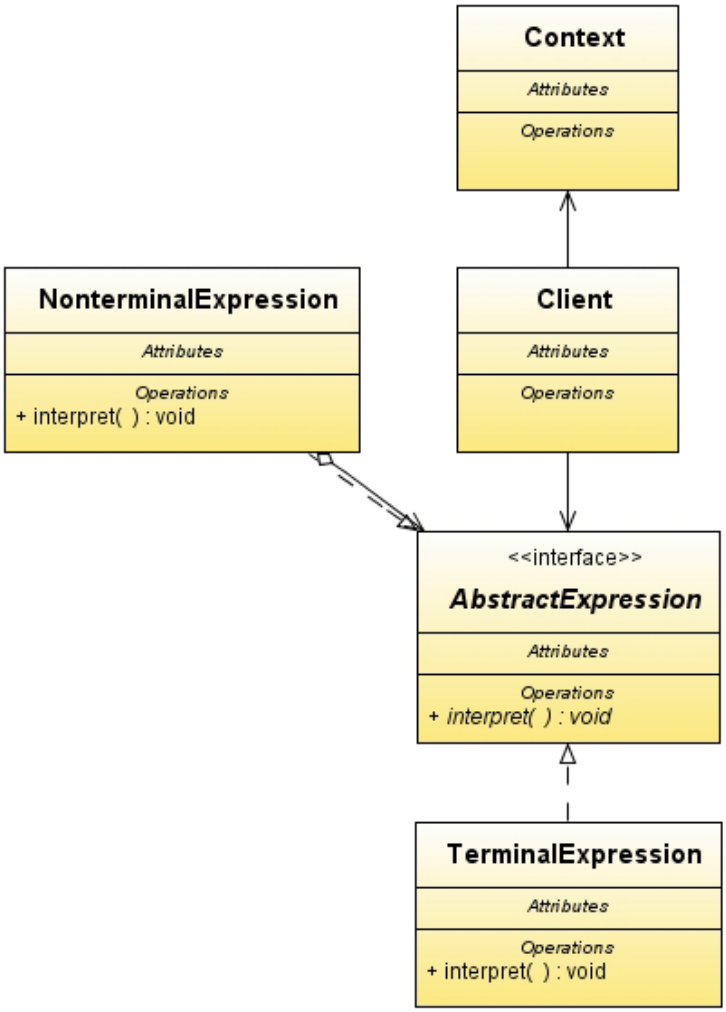


4.11 Patrón “Interpreter”

Definición:

El patrón “Interpreter” dado cierto lenguaje, define una representación para su gramática, y usa la representación para interpretar las sentencias del lenguaje [Gamma, 1995].

Diagrama UML de clases del patrón “Interpreter”


Context
Attributes
Operations
NonterminalExpression
Attributes
Operations
+ interpret( ) : void
Client
Attributes
Operations
<<interface>>
AbstractExpression
Attributes
Operations
+ interpret( ) : void
TerminalExpression
Attributes
Operations
+ interpret( ) : void