2

(IV)

Patrón Composite

(Patrones de diseño)

Diseño del Software

Grado en Ingeniería Informática del Software

Curso 2022-2023

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

Composite (Compuesto)

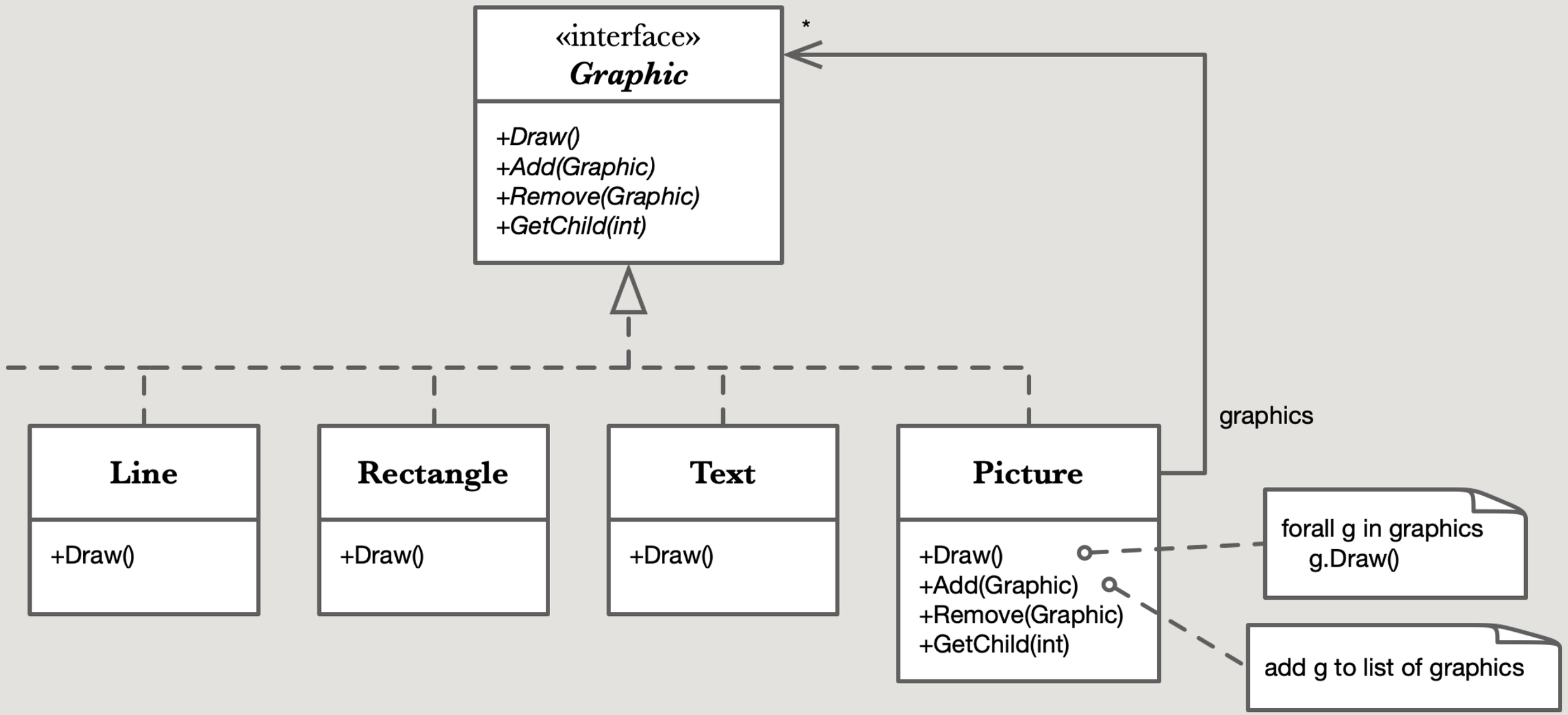
- **Patrón estructural de objetos**
- **Propósito:**

> *Permite componer objetos en estructuras arbóreas para representar jerarquías de todo-parte, de modo que los clientes puedan tratar a los objetos individuales y a los compuestos de manera uniforme.*

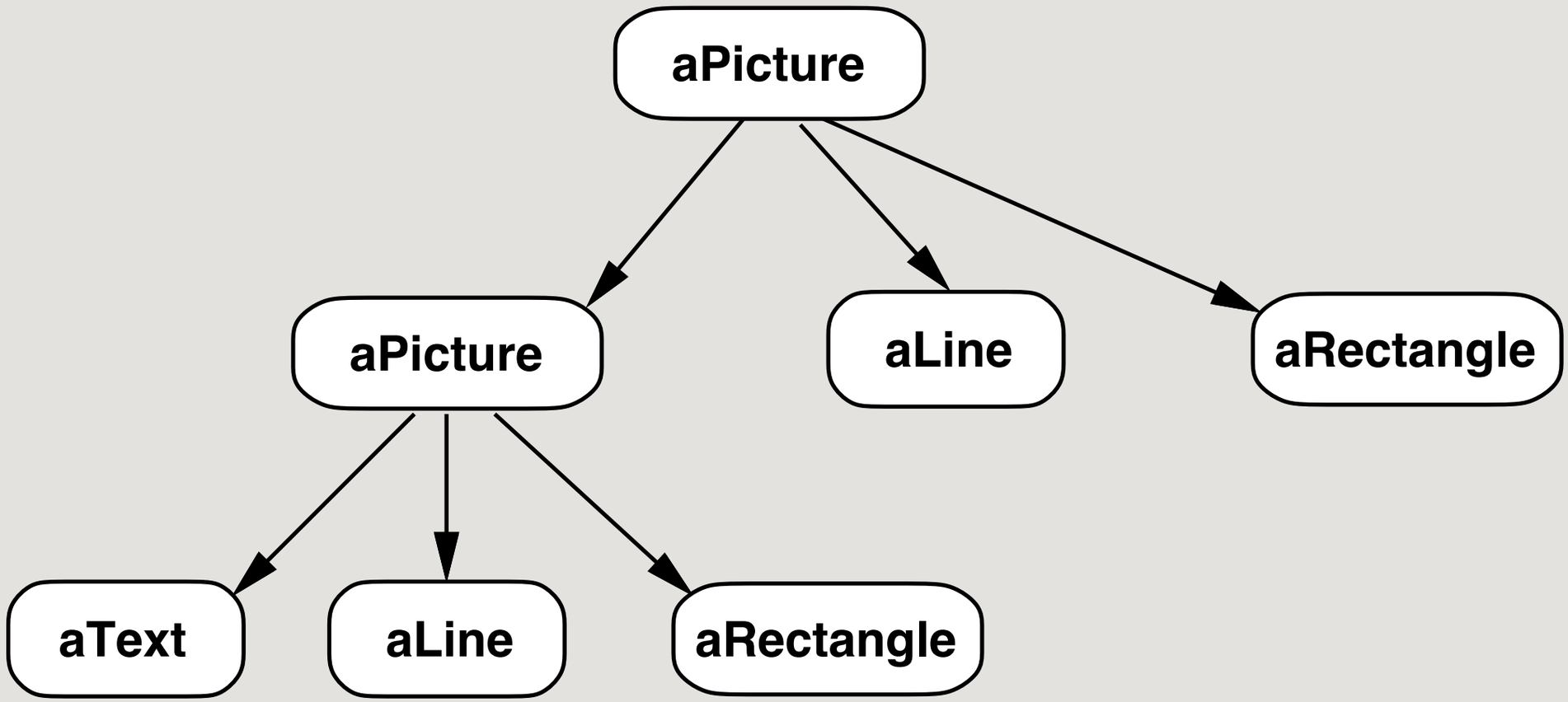
Motivación

- **Un editor de dibujo permite realizar dibujos compuestos de elementos simples (como líneas, rectángulos, etc.) u otros dibujos**
  - ¿Cómo evitamos que los clientes tengan que distinguir entre unos y otros?

Motivación


«interface»
Graphic
+Draw()
+Add(Graphic)
+Remove(Graphic)
+GetChild(int)
*
graphics
Line
+Draw()
Rectangle
+Draw()
Text
+Draw()
Picture
+Draw()
+Add(Graphic)
+Remove(Graphic)
+GetChild(int)
forall g in graphics
g.Draw()
add g to list of graphics

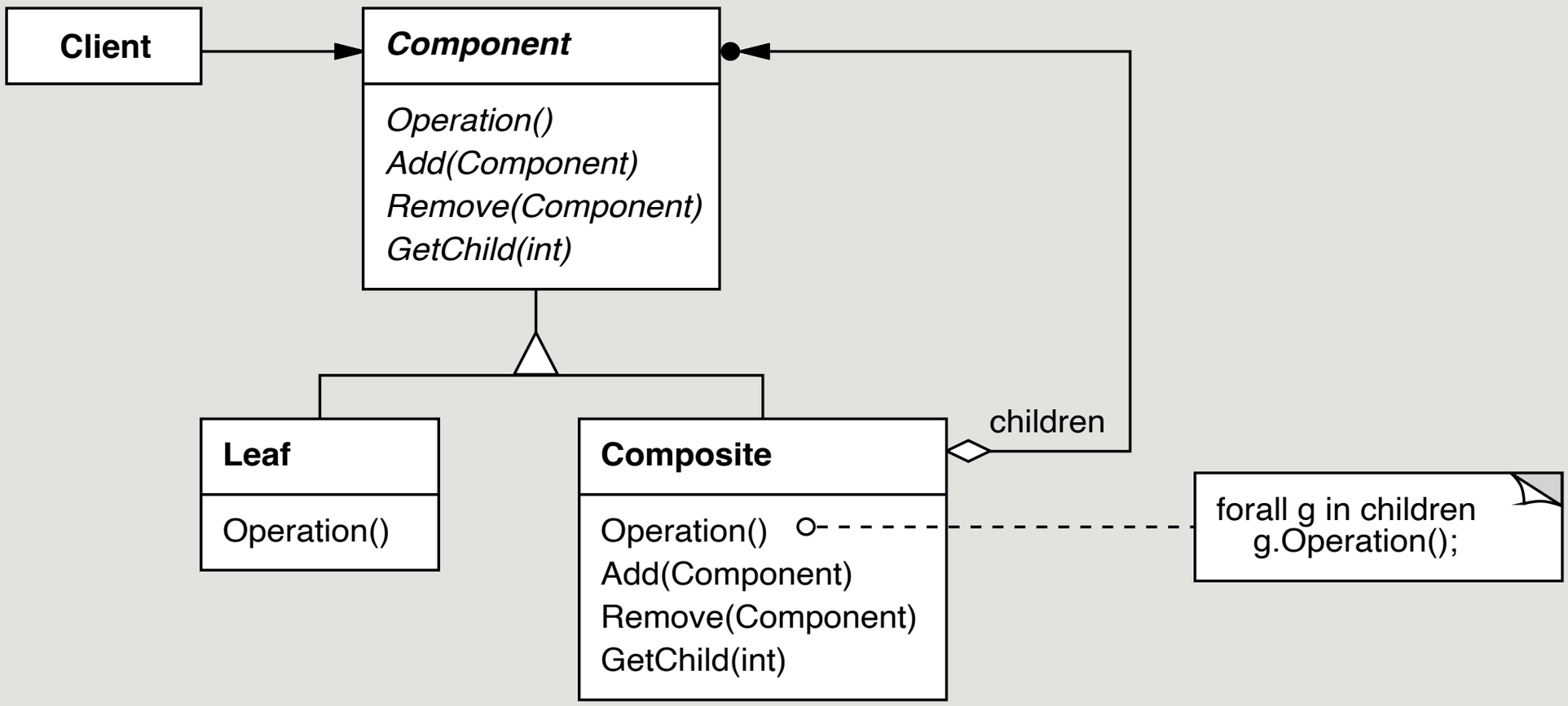
Motivación


aPicture
aPicture
aLine
aRectangle
aText
aLine
aRectangle

Aplicabilidad

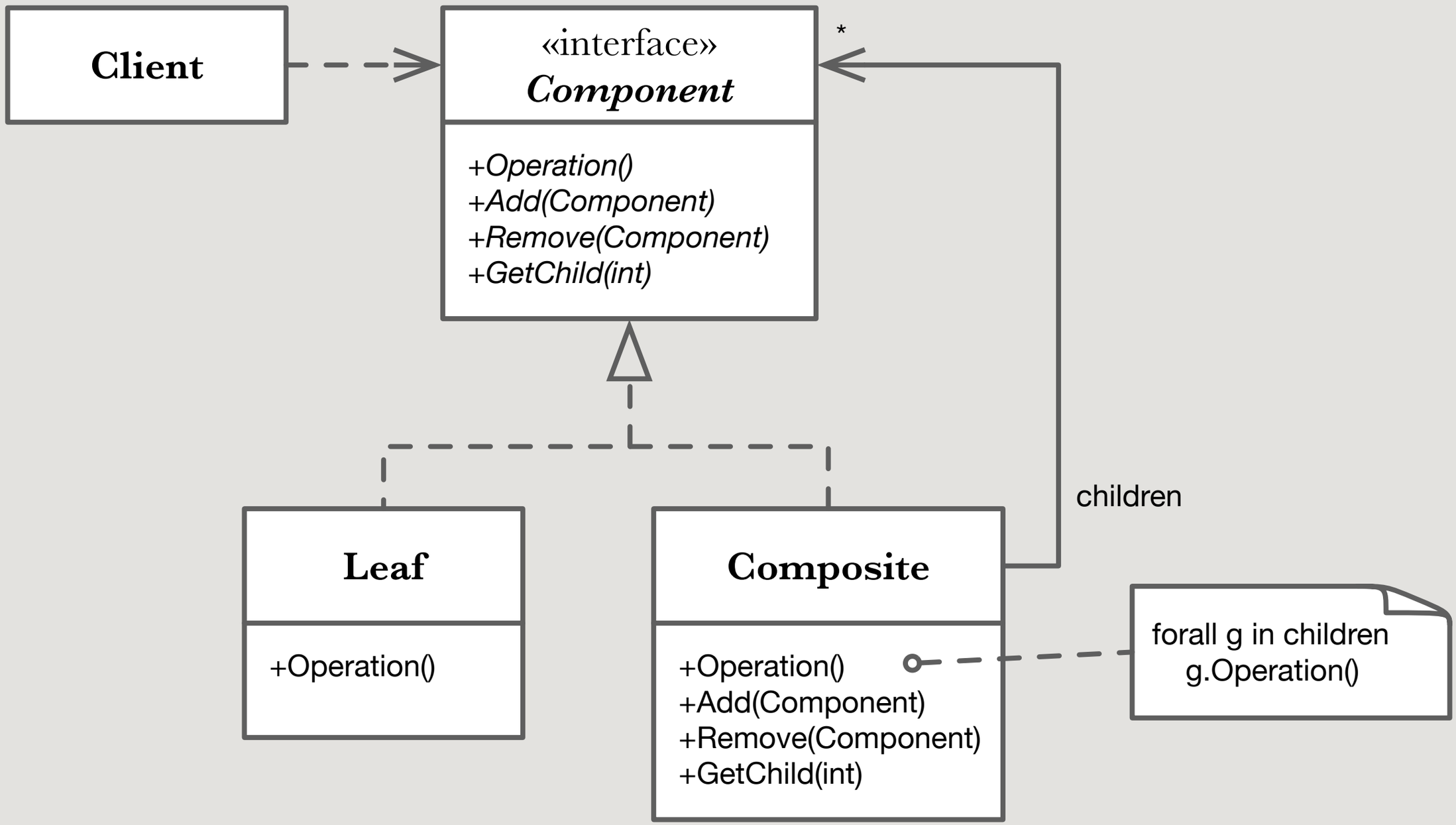
- Representar jerarquías de parte-todo
- Que los clientes traten por igual los objetos individuales y los compuestos

Estructura



La estructura del patrón *Composite*, tal como aparece en el GoF

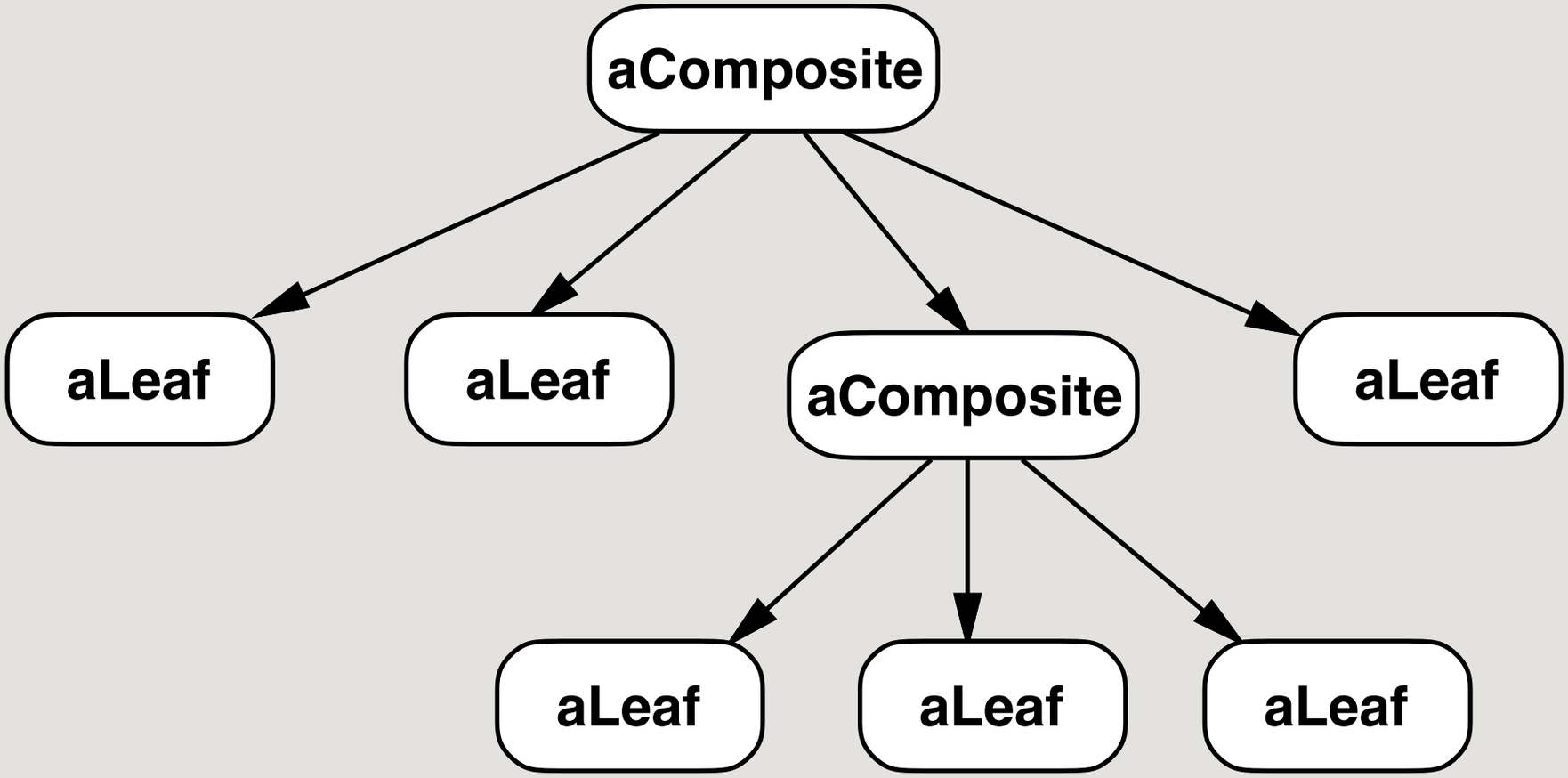
Estructura



La estructura del patrón *Composite*, en UML

Estructura

- Una estructura de objetos típica del patrón *Composite* se parece a ésta:



Participantes

- **Component (Grahic)**
  - Declara la interfaz común
  - Implementa el comportamiento predeterminado que es común a todas las clases
  - Declara operaciones para acceder a los hijos
  - (opcional) Define una interfaz para acceder al padre
- **Leaf (Rectangle, Line, Text...)**
- **Composite (Picture)**
  - Almacena sus componentes hijos
  - Implementa las operaciones relacionadas con los hijos
- **Client**
  - Manipula los objetos de la composición a través de la interfaz de Component

No necesariamente (de hecho, en nuestro caso será más frecuente que las declaremos en el compuesto)

Consecuencias

- **Permite jerarquías de objetos tan complejas como se quiera**
  - Allá donde el cliente espere un objeto primitivo, podrá recibir un compuesto y no se dará cuenta
- **Simplifica el cliente**
  - Al eliminar el código para distinguir entre unos y otros
- **Se pueden añadir nuevos componentes fácilmente**

Consecuencias

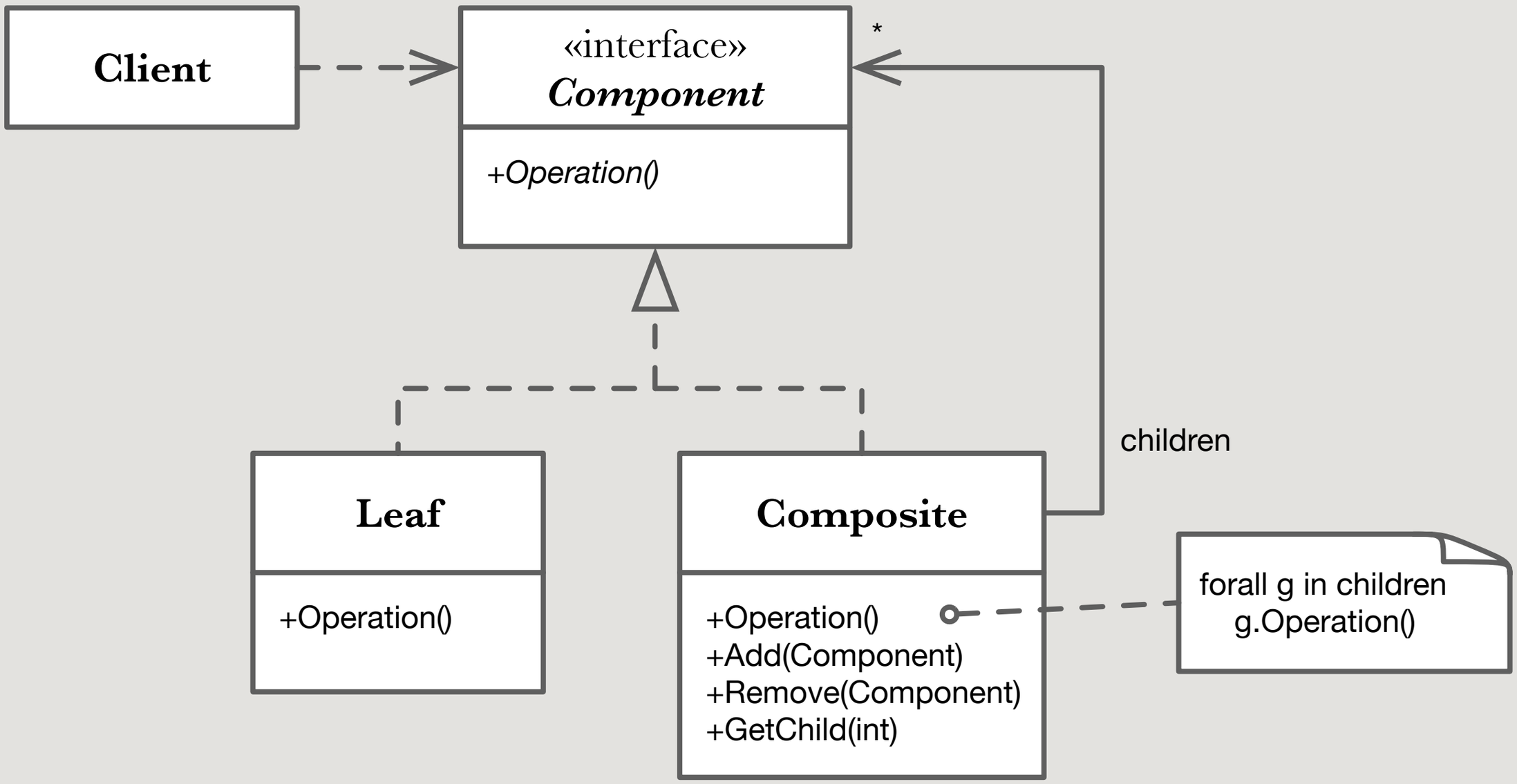
- **Como desventaja, podría hacer el diseño demasiado general**
  - Si queremos restringir los componentes que pueden formar parte de un compuesto determinado

Según el GoF, con el Composite no se puede confiar en el sistema de tipos para garantizar estas restricciones, y hay que hacerlo con comprobaciones en tiempo de ejecución. Tal vez sea así si queremos añadir componentes dinámicamente, pero si lo sabemos a priori o podemos modificar el diseño, suele ser fácil hacerlo (introduciendo nuevos tipos).

Implementación

- **Referencias explícitas al padre**
- **Maximizar la interfaz de *Component***
  - El componente puede proporcionar implementaciones predeterminadas que luego las clases hoja y las compuestas redefinan
  - Problema: puede haber operaciones que tengan sentido en unas pero no en otras
    - Ejemplo: las operaciones de gestión de los hijos
    - Compromiso entre seguridad (comprobación estática de tipos) y transparencia (flexibilidad)

Estructura



Cómo quedaría la estructura optando por la seguridad de tipos frente a la flexibilidad (relativa)

Implementación

- **Orden de los hijos**
  - Si es significativo, hay que diseñar las interfaces de acceso y gestión de los hijos cuidadosamente
    - Por ejemplo, mediante un iterador (*Iterator*)