8 Composite

Diseño del Software

Grado en Ingeniería Informática del Software

2024-2025

Escuela de Ingeniería Informática
Universidad de Oviedo

César Acebal

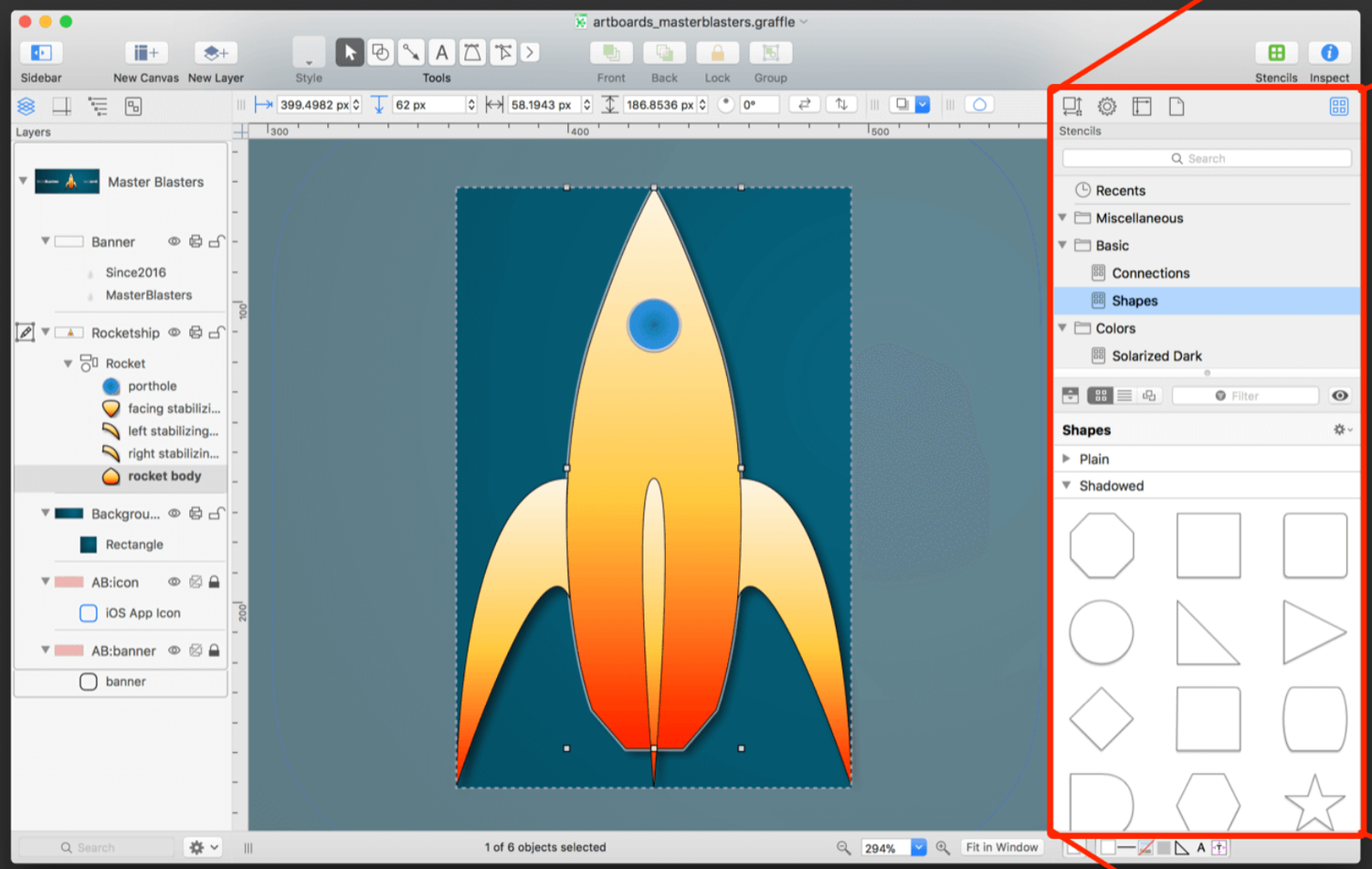
Patrón estructural de objetos

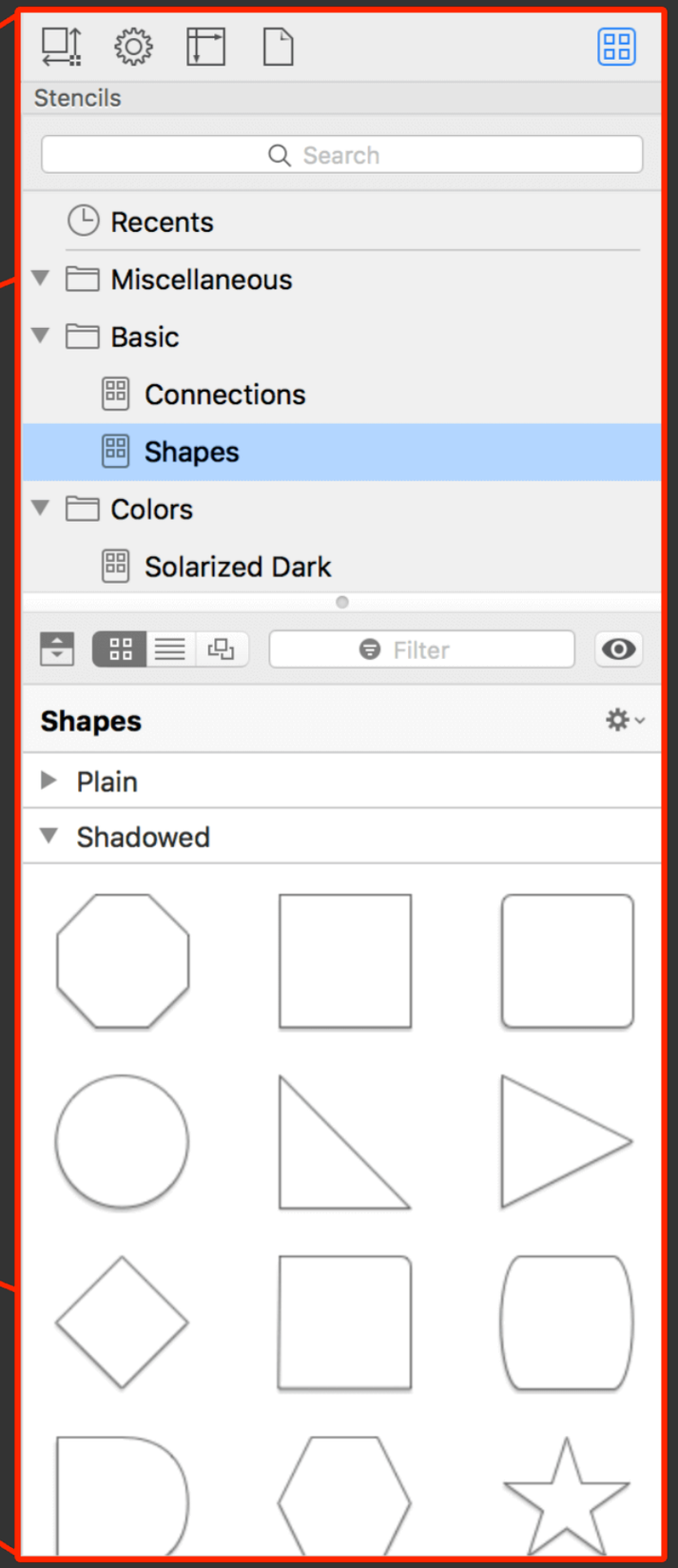
Permite componer objetos en estructuras arbóreas para representar jerarquías de todo-parte, de modo que los clientes puedan tratar a los objetos individuales y a los compuestos de manera uniforme.

Motivación

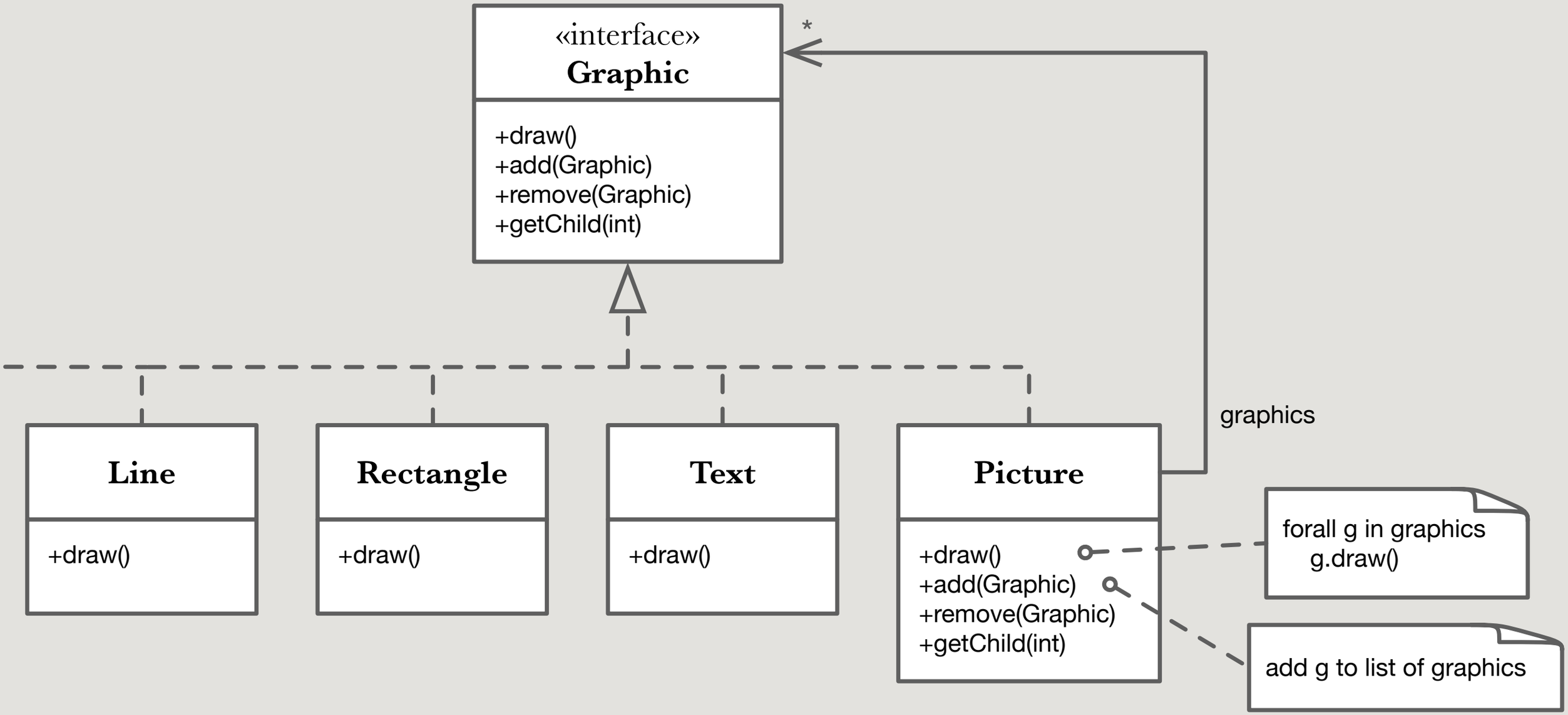
Un editor de dibujo permite realizar dibujos complejos a partir de elementos simples como líneas, rectángulos, etcétera, u otros dibujos.

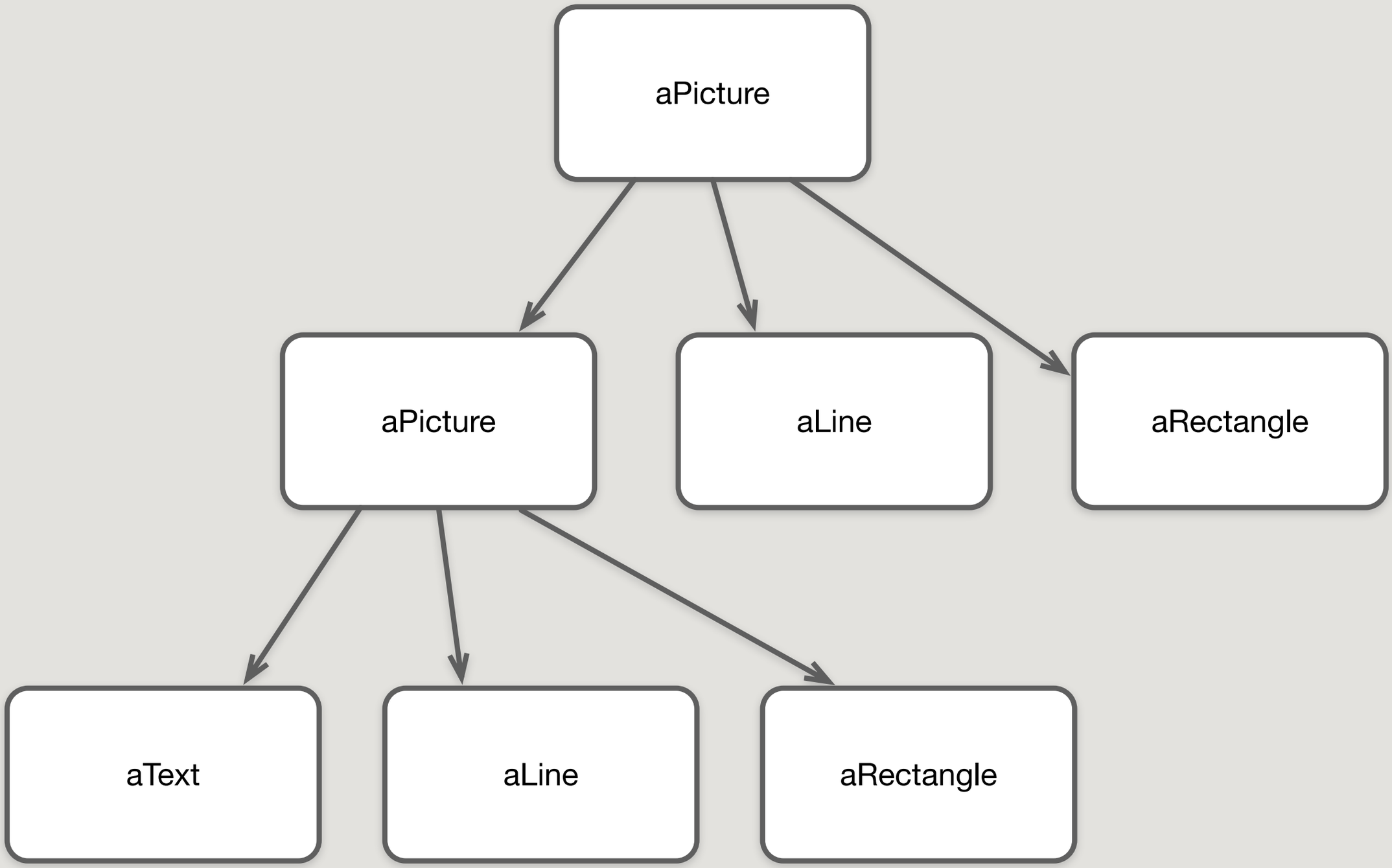
El usuario puede agrupar componentes para formar componentes mayores, que a su vez pueden agruparse para formar componentes aún mayores.


artboards_masterblasters.graffle
Sidebar
New Canvas
New Layer
Style
Tools
Front
Back
Lock
Group
Stencils
Inspect
Layers
Master Blasters
Banner
Since2016
MasterBlasters
Rocketship
Rocket
porthole
facing stabilizi...
left stabilizing...
right stabilizin...
rocket body
Backgrou...
Rectangle
AB:icon
iOS App Icon
AB:banner
banner
Search
1 of 6 objects selected
294%
Fit in Window
Stencils
Search
Recents
Miscellaneous
Basic
Connections
Shapes
Colors
Solarized Dark
Filter
Shapes
Plain
Shadowed


Stencils
Search
Recents
Miscellaneous
Basic
Connections
Shapes
Colors
Solarized Dark
Filter
Shapes
Plain
Shadowed

¿Cómo evitar que los clientes tengan que distinguir entre elementos simples y dibujos compuestos?


«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


aPicture
aPicture
aLine
aRectangle
aText
aLine
aRectangle

Aplicabilidad

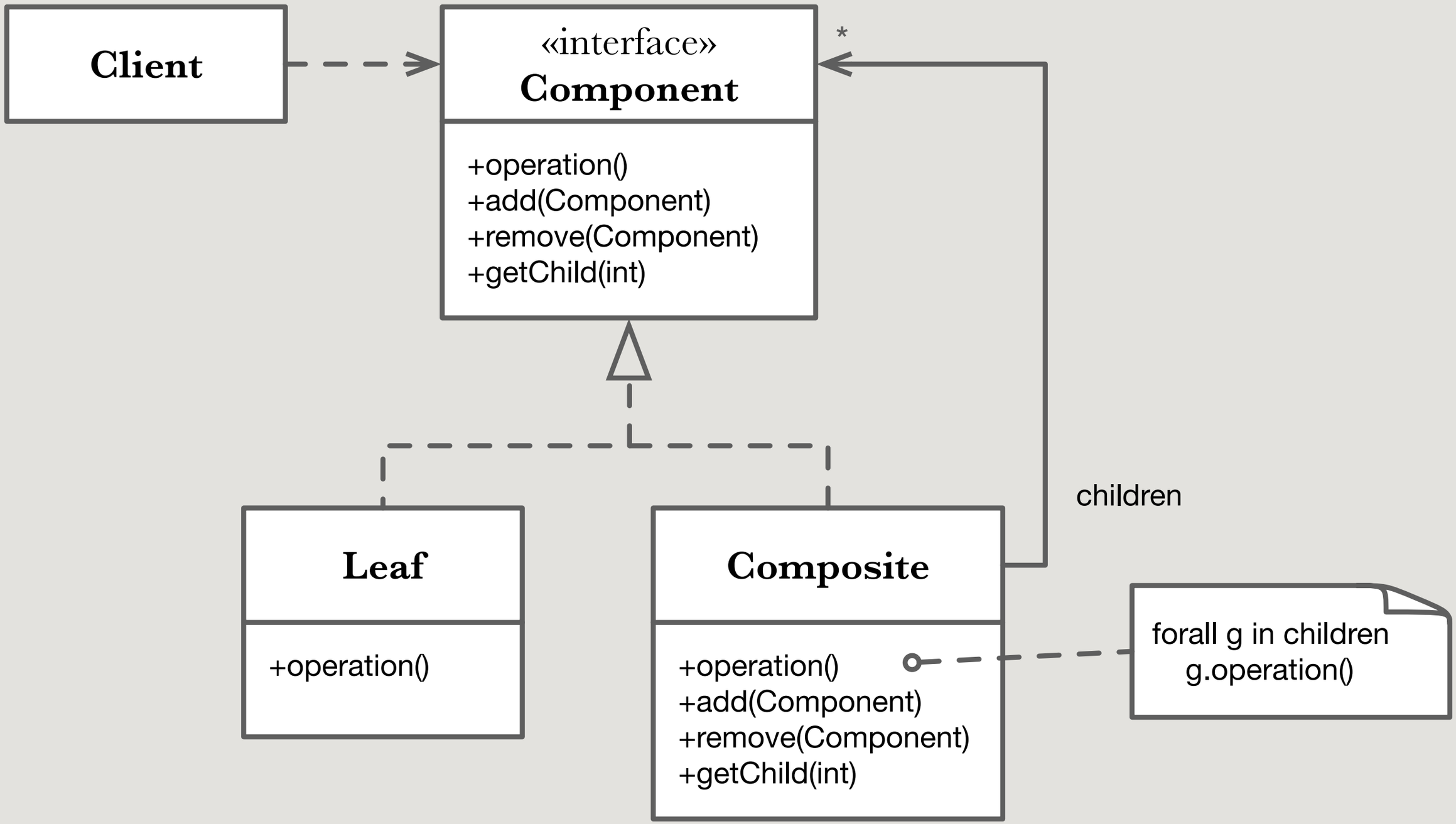
Úsese el patrón Composite cuando

Queremos representar jerarquías de parte-todo.

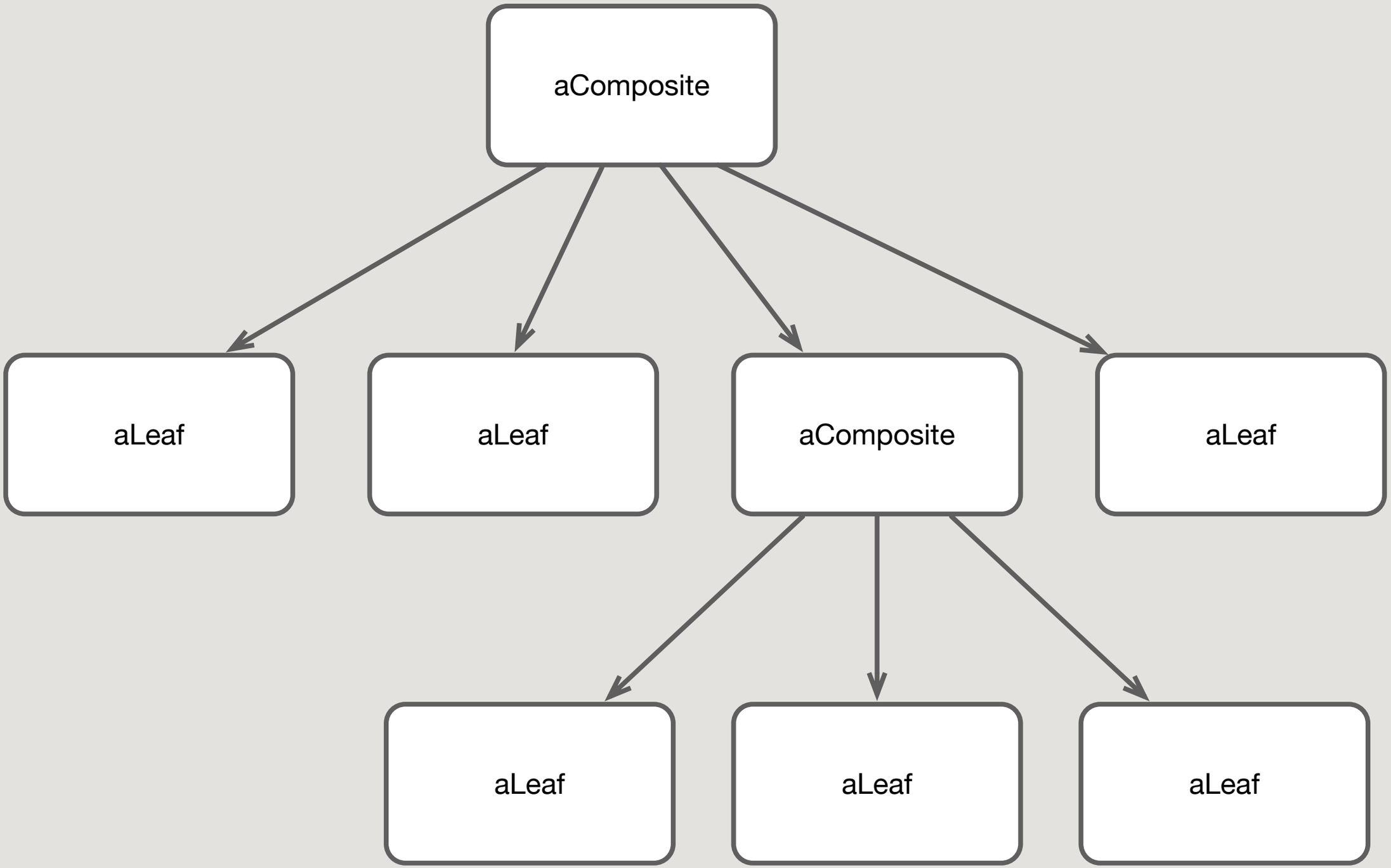
Queremos que los clientes traten **por igual los objetos individuales y los compuestos.**

Los clientes tratarán a todos los objetos de la estructura de manera **uniforme**.

Estructura



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



Una estructura de objetos típica en tiempo de ejecución

Participantes

Component

(Graphic)

Declara la interfaz común para todos los objetos de la estructura.

Declara las operaciones para acceder a los hijos.*

Opcionalmente, puede definir una interfaz para acceder al padre de cada elemento de la estructura recursiva.

---

* Según el libro. Aunque se discutirá en la sección de implementación, como regla general es más común declarar esas operaciones de gestión de los hijos en el Composite.

johndoe - Thunar

File Edit View Go Bookmarks Help

/home/johndoe/

Places

- Computer
- johndoe
- Desktop
- Recent
- Trash
- 4.18 release
- backgrounds

Devices

- File System

Network

- Browse Network

Desktop Documents Downloads Music

Pictures Public Templates Videos

8 folders | Free space: 44.0 GiB

Leaf

(Rectangle, Line, Text…)

Representa los objetos hoja de la estructura.

Una hoja no tiene hijos.

Define el comportamiento de los elementos primitivos.

Composite

(Picture)

Define el comportamiento de los componentes que tienen hijos.

Guarda sus componentes hijos.

Implementa las operaciones relacionadas con los hijos [definidas en la interfaz de Component].

Client

Manipula los objetos de la composición a través de la interfaz de Component.

Colaboraciones

Los clientes utilizan la interfaz de la clase Componente para interactuar con los objetos de la estructura compuesta.

Si el receptor es una hoja, él mismo responde a la petición directamente.

Si es un compuesto, normalmente reenvía las peticiones a sus componentes hijos.

(Posiblemente realizando operaciones adicionales antes o después del reenvío.)

Consecuencias

Define jerarquías de clases formadas por objetos primitivos y objetos compuestos.

Los objetos primitivos pueden componerse en objetos más complejos, que a su vez pueden componerse, y así **recursivamente**.

Allá donde el cliente espere un objeto primitivo, **podrá recibir un compuesto** y no se dará cuenta.

Simplifica al cliente.

Los clientes pueden tratar de manera **uniforme** las estructuras compuestas y los objetos individuales.

Normalmente no saben (ni les importa) si se trata de una hoja o de un componente compuesto.

Facilita añadir nuevos tipos de componentes.

Las nuevas subclases compuestas o de hoja funcionan automáticamente con las estructuras y los clientes existentes.

No es necesario cambiar los clientes.

Una posible desventaja (según el libro)

Podría hacer el diseño demasiado general.

Si queremos restringir los componentes que pueden formar parte de un compuesto determinado.

La desventaja de facilitar la adición de nuevos componentes es que dificulta la restricción de los componentes de un compuesto.

Con Composite, no podemos confiar en que el sistema de tipos nos imponga esas restricciones. En su lugar, tendremos que utilizar comprobaciones en tiempo de ejecución.*

*

¿Seguro?

Según el GoF, con 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

johndoe - Thunar

File Edit View Go Bookmarks Help

/home/johndoe/

Places

- Computer
- johndoe
- Desktop
- Recent
- Trash
- 4.18 release
- backgrounds

Devices

- File System

Network

- Browse Network

Desktop Documents Downloads Music

Pictures Public Templates Videos

8 folders | Free space: 44.0 GiB

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.

Como las **operaciones de gestión de los hijos**.

Compromiso entre seguridad (comprobación estática de tipos) y transparencia (flexibilidad)

One of the goals of the Composite pattern is to make clients unaware of the specific leaf or composite classes they're using.

To attain this goal, the component class should define as many common operations for composite and leaf classes as possible.

The component class usually provides default implementations for these operations, and leaf and composite subclasses will override them.

However, this goal will sometimes conflict with the principle of class hierarchy design that says a class should only define operations that are meaningful to its subclasses. There are many operations that Component supports that don't seem to make sense for leaf classes. How can Component provide a default implementation for them?

Declarar las operaciones de gestión de los hijos

¿Qué clase debe declarar las operaciones de añadir, eliminar y otras operaciones de gestión de hijos?

¿Deberíamos declarar estas operaciones en el componente y darles una implementación que tuviese sentido en las clases hojas, o deberíamos declararlas y definirlas sólo en el compuesto?

Definir la interfaz de gestión de hijos en la raíz de la jerarquía de clases nos da transparencia, porque podemos tratar a todos los componentes de manera uniforme.

Sin embargo, nos cuesta seguridad, porque los clientes pueden intentar hacer cosas sin sentido, como añadir y quitar objetos de las hojas.

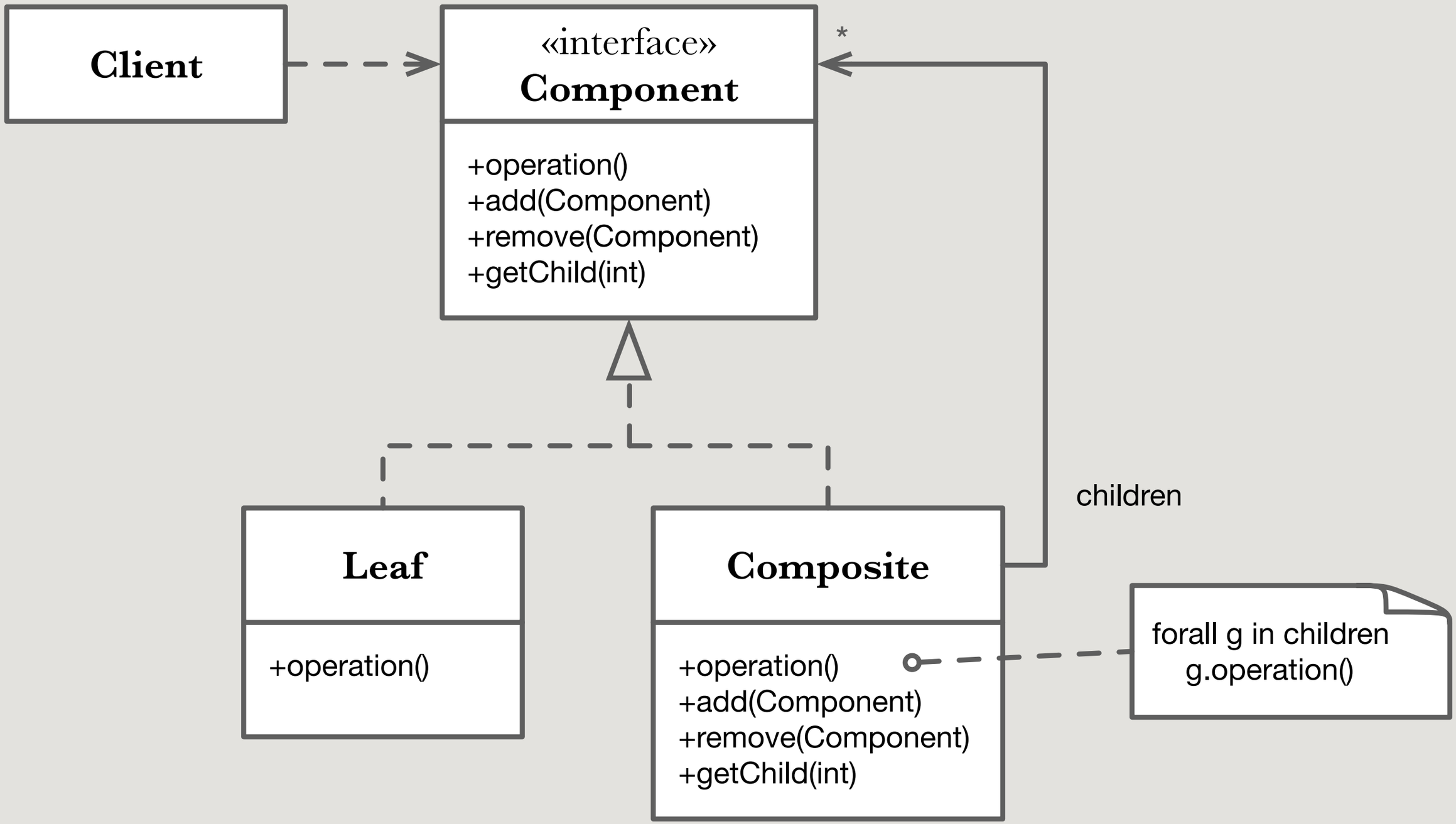
Compromiso entre seguridad y transparencia

Definir la gestión de hijos en la clase Composite nos da seguridad, porque cualquier intento de añadir o eliminar objetos de las hojas será capturado en tiempo de compilación en un lenguaje con comprobación estática de tipos.

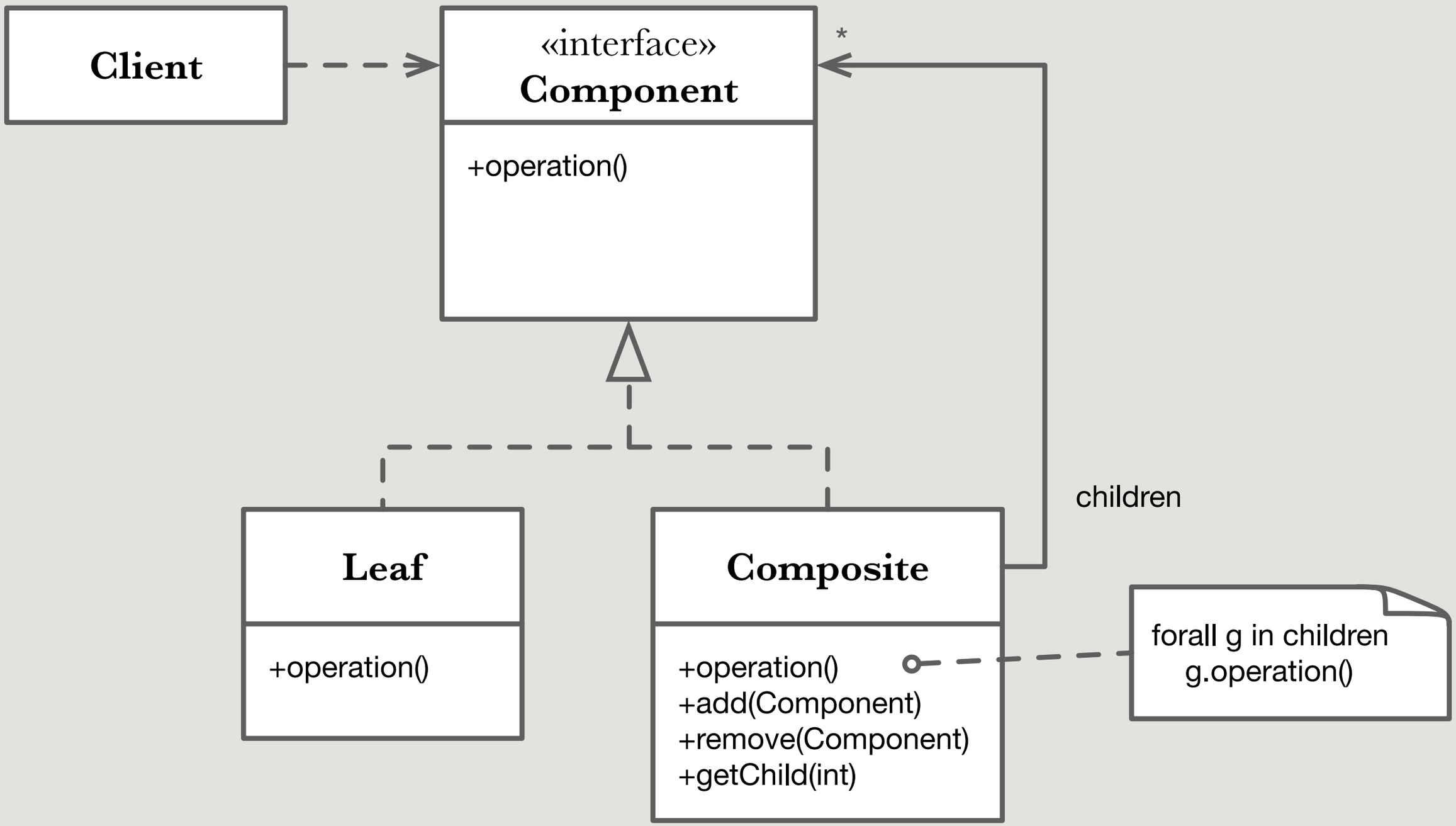
Pero perdemos transparencia, porque las hojas y los compuestos tienen interfaces diferentes.

En el libro se hace hincapié en la transparencia por encima de la seguridad en este modelo.

Nosotros normalmente preferiremos el otro enfoque: definir estas operaciones **solo en el compuesto**.



La estructura del patrón de diseño Composite, tal y como aparece en el GoF



Cómo quedaría la estructura si optamos por la seguridad de tipos frente a la flexibilidad (relativa, en todo caso) de la estructura genérica del patrón

Patrones relacionados

Decorator

Aunque ambos tienen una estructura similar, sus propósitos difieren: un decorador **añade responsabilidades** a un objeto individual, mientras que un compuesto ensambla objetos para representar jerarquías, tratando los grupos y los objetos individuales de manera uniforme; aplica **la misma operación** a cada objeto de la jerarquía.