Patrones de Diseño: COMMAND

Autores: Fernando Muñoz - Mariano Ontivero - Matías Rondeau

- También conocido como **Action** y **Transaction**.

- Es clasificado como un **Patrón de Comportamiento** dado que caracteriza la forma en la cual interactúan y se distribuyen responsabilidades clases y objetos.

INTENCIÓN

- Encapsular una petición como un objeto, lo que permite parametrizar clientes con diferentes peticiones, encolar o registrar peticiones y brindar soporte para realizar operaciones cuyos efectos se pueden deshacer.

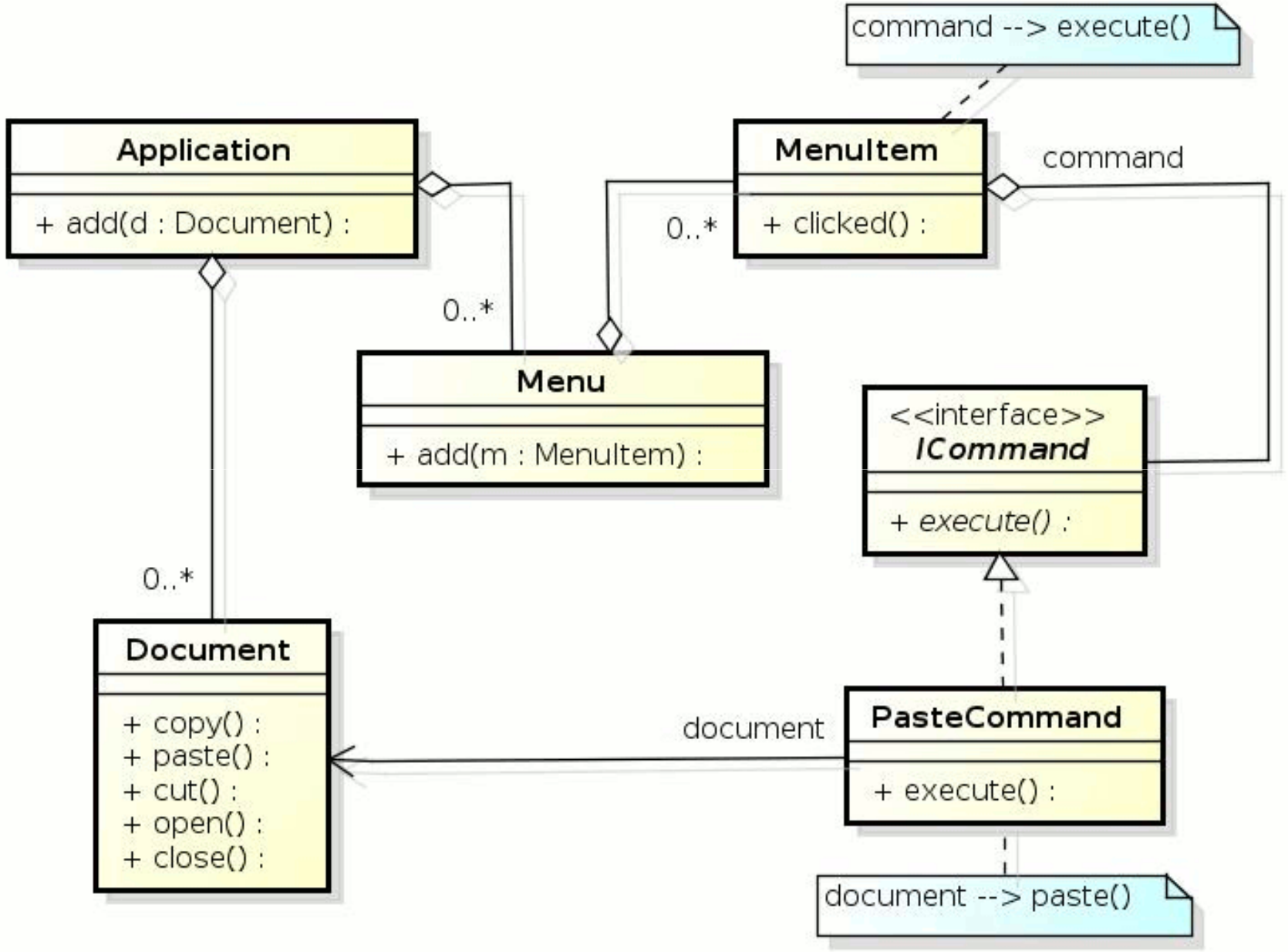
INTENCIÓN

- Encapsular una petición como un objeto, lo que permite parametrizar clientes con diferentes peticiones, encolar o registrar peticiones y brindar soporte para realizar operaciones cuyos efectos se pueden deshacer.

MOTIVACIÓN

- A veces es necesario emitir peticiones a objetos sin conocer la operación que está siendo solicitada ni el receptor de la petición.
- Esto ocurre, por ejemplo, en el diseño de toolkits, frameworks gráficos, etc, los cuales incluyen objetos como botones, menús, etc, que ejecutan operaciones en respuesta a las entradas de los usuarios.
- La implementación de dichas operaciones no puede estar presente en el botón o menú dado que sólo la aplicación que utiliza al toolkit o framework sabe que debe hacerse en dicho objeto.

- El patrón propone crear una clase abstracta `Command` que brinda al cliente un método abstracto `execute()` para ejecutar peticiones.
- Las subclases concretas de `Command` almacenarán el objeto receptor de la petición (`Receiver`) como una variable de instancia e implementarán `execute()`, quién será el encargado de invocar el método correspondiente de `Receiver` (`action()`).
- `action()` es quién sabe como llevar a cabo la petición, es decir, quién implementa la operación asociada a la misma.
- Ejemplo de aplicación: un editor de texto.


command --> execute()
Application
+ add(d : Document) :
MenuItem
+ clicked() :
command
0..*
0..*
Menu
+ add(m : MenuItem) :
<<interface>>
ICommand
+ execute() :
0..*
Document
+ copy() :
+ paste() :
+ cut() :
+ open() :
+ close() :
document
PasteCommand
+ execute() :
document --> paste()

APLICABILIDAD

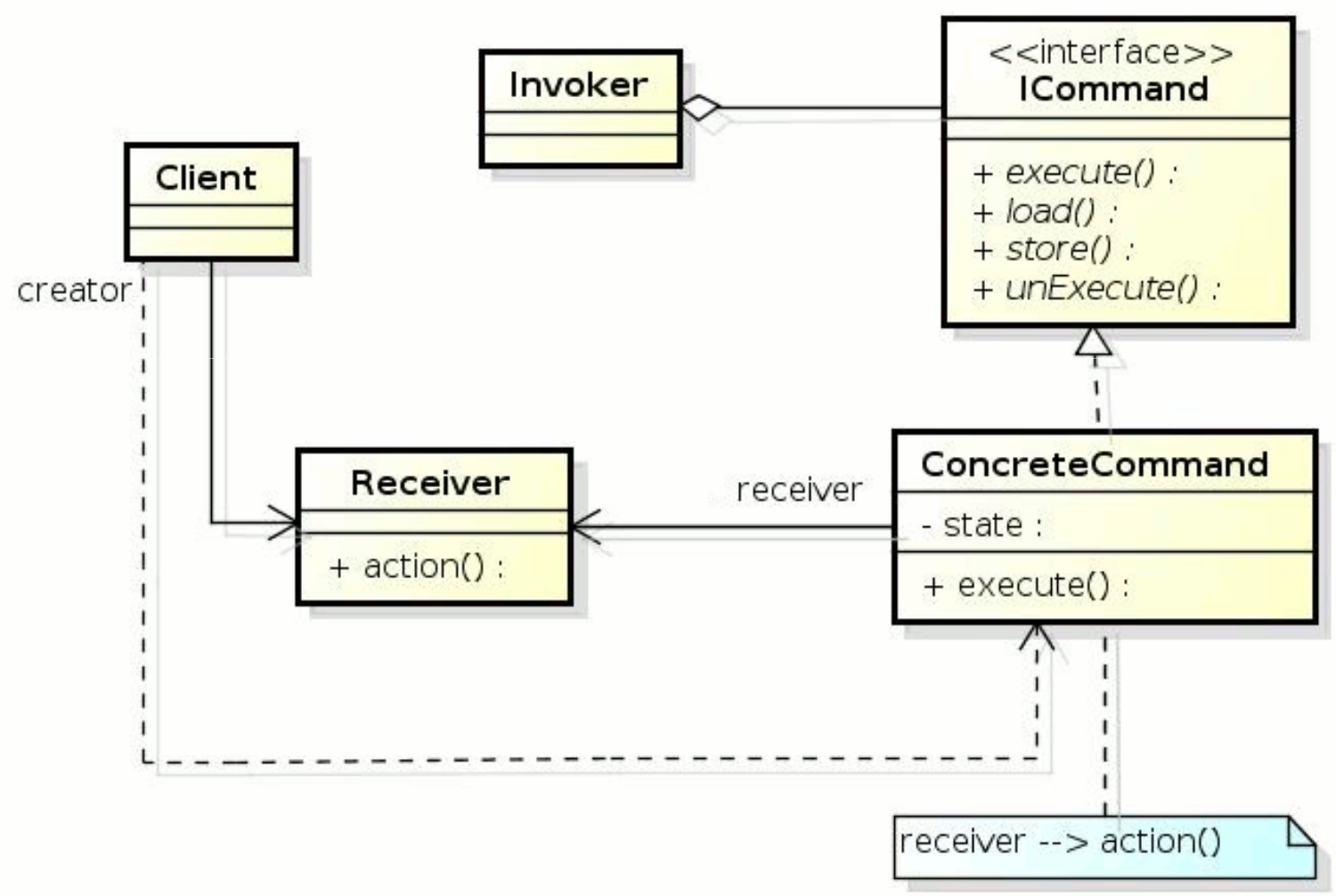
El patrón Command debe usarse cuando se desee:

- Parametrizar objetos según la acción a ejecutar. Los objetos Command son la versión orientada a objetos de las funciones de primera clase en los lenguajes procedurales.
- Especificar, encolar, y ejecutar peticiones en distintos momentos. El tiempo de vida de un objeto Command es independiente de la petición original.
- Permitir deshacer los efectos producidos por la ejecución de los comandos. Los comandos ...

... ejecutados se guardan en una lista que funciona como historial.

- Llevar un registro persistente de los cambios de manera que puedan ser reaplicados en caso de una caída del sistema.
- Estructurar un sistema alrededor de operaciones con un alto nivel de abstracción construidas sobre operaciones primitivas.
- Tener una interfaz común que permita invocar comandos de manera uniforme y agregar nuevos comandos de manera sencilla.

ESTRUCTURA

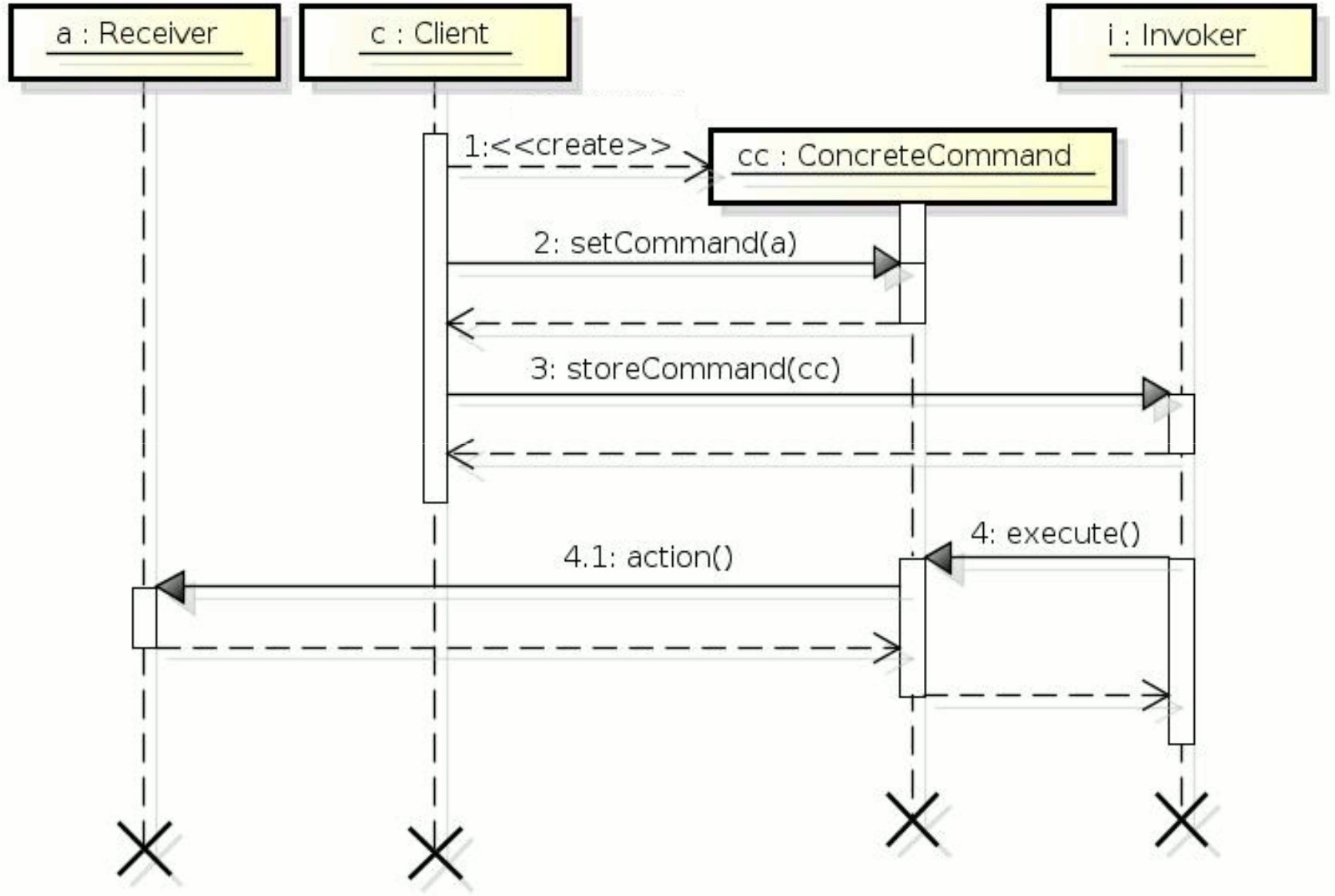

Invoker
<<interface>>
ICommand
+ execute() :
+ load() :
+ store() :
+ unExecute() :
Client
creator
Receiver
+ action() :
receiver
ConcreteCommand
- state :
+ execute() :
receiver --> action()

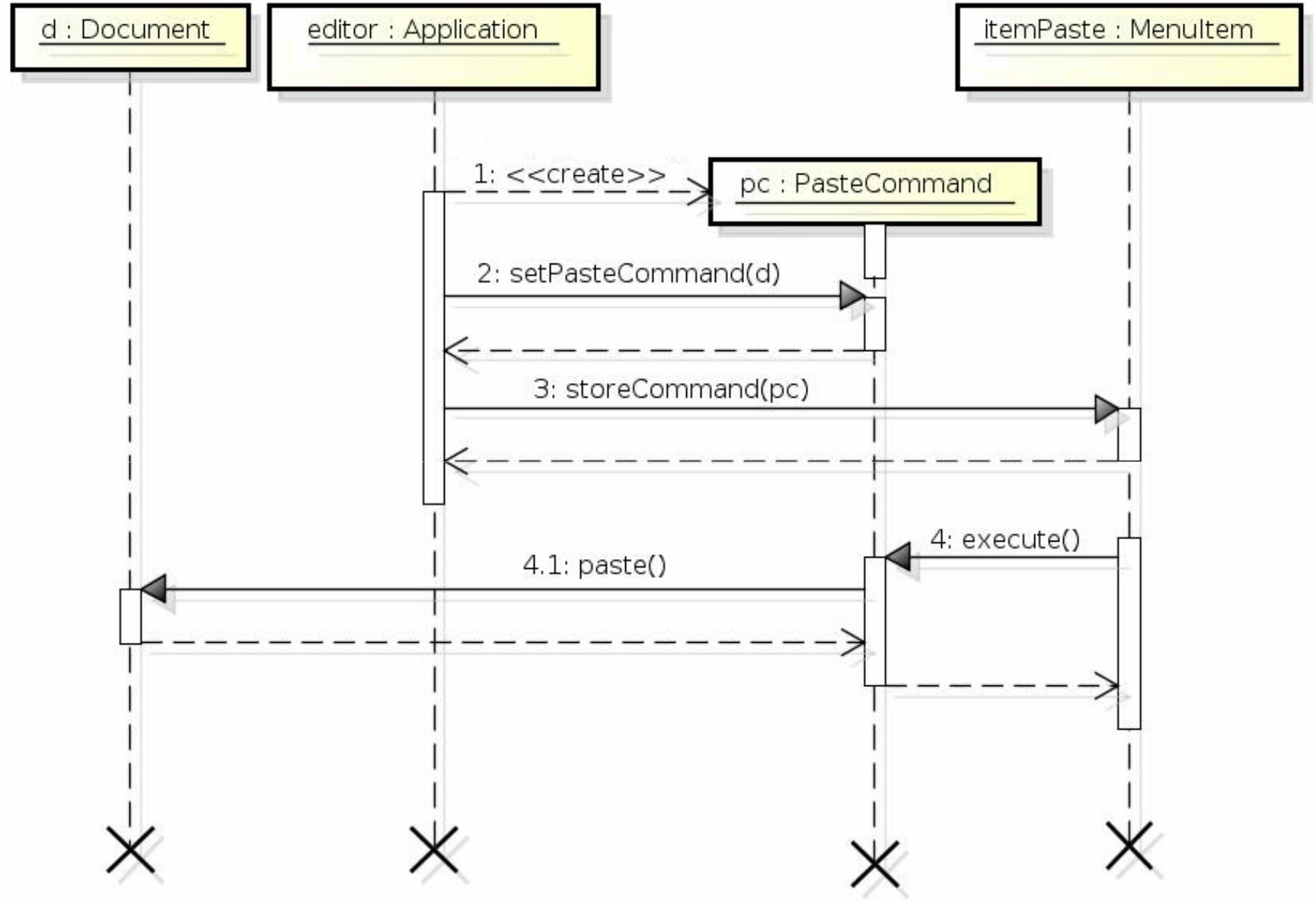
PARTICIPANTES

- **Command:** declara una interfaz para ejecutar una operación.
- **ConcreteCommand:**
  - define un enlace con un objeto `Receiver` y su acción.
  - implementa `execute()` invocando las correspondientes operaciones en `Receiver`.
- **Client:** crea un objeto `ConcreteCommand` y establece su receptor.
- **Invoker:** solicita el comando para llevar a cabo la petición.
- **Receiver:** conoce como ejecutar las operaciones asociadas para llevar a cabo la petición.

COLABORACIONES

- El objeto `Client` crea un objeto `ConcreteCommand` especificando su receptor.
- Un objeto `Invoker` almacena el objeto `ConcreteCommand`.
- El objeto `Invoker` emite una petición llamando `execute()` sobre el comando. Si el comando puede deshacerse, `ConcreteCommand` almacena el estado antes de invocar a `execute()`.
- El objeto `ConcreteCommand` invoca operaciones de su receptor para llevar a cabo la petición.


a : Receiver
c : Client
i : Invoker
1:<<create>>
cc : ConcreteCommand
2: setCommand(a)
3: storeCommand(cc)
4: execute()
4.1: action()

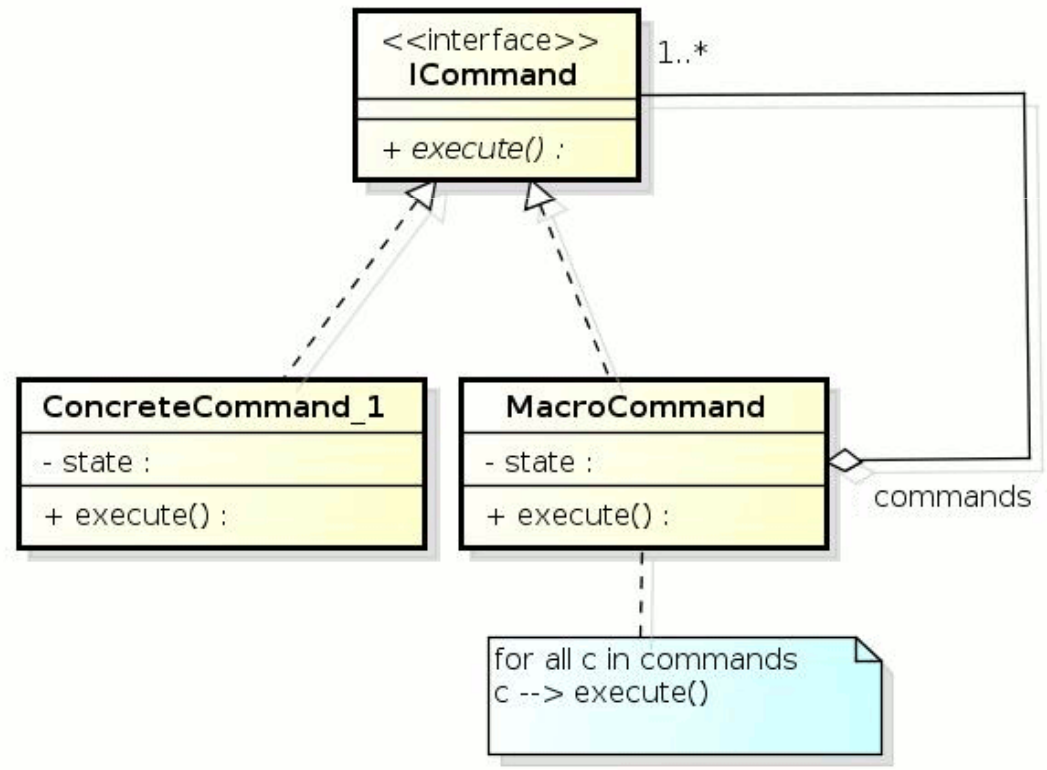

d : Document
editor : Application
itemPaste : MenuItem
1: <<create>>
pc : PasteCommand
2: setPasteCommand(d)
3: storeCommand(pc)
4: execute()
4.1: paste()

CONSECUENCIAS

1. Desacopla el objeto que invoca la operación de aquel que implementa la funcionalidad real.
2. Los comandos son objetos de primera clase (objetos sin restricciones de uso).
3. Permite ensamblar comandos dentro de un comando compuesto (MacroCommand).
4. Facilidad para agregar nuevos comandos, dado que no es necesario modificar las clases existentes.

PATRONES RELACIONADOS

- Para implementar `MacroCommands` se utiliza junto al patrón estructural Composite:



BIBLIOGRAFÍA

- Erich Gamma, Richard Helm, Ralph Johnson, John Vlissedes, “Design Patterns, Elements of Reusable Object Oriented Software”, Addison-Wesley, 1995.