Patrones creacionales

Roni Guillermo Apaza Aceituno

Universidad Nacional de San Agustin

rapazaac@unsa.edu.pe

October 26, 2018

Introducción

- Abstrae el proceso de instanciación.
- Independiza el sistema de como sus objetos son creados, compuestos y representados.
- Un clase de este tipo usa herencia para variar la clase que instancia.

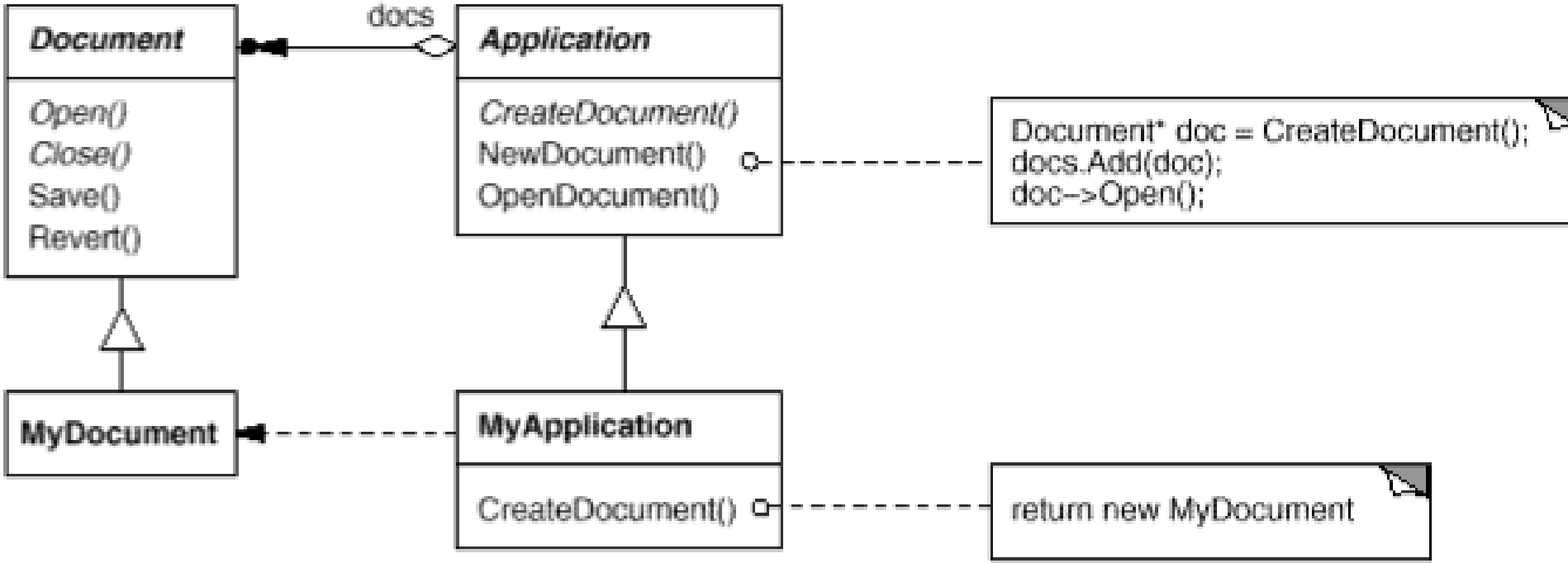
Introducción

- Un objeto de este tipo delega la instanciación a otro objeto.
- Estos patrones son importantes en sistemas que evolucionan dependiendo mas de la composición que de la herencia.
- Si se desea un comportamiento en general, se puede concentrar uno en comportamientos mas fundamentales y componerlos.

Factory Method

- Define la interface para crear un objeto.
- Deja a las subclases que clase instanciar.
- Permite a una clase retrazar la instanciación para subclases.

Factory Method



Factory Method

- Un framework define y mantiene la relación entre objetos usando clases abstractas.
- El frame work es responsable de la creación de esos objetos también.
- Supongamos que el framework crea aplicaciones para multiples documentos.

Factory Method

- Son necesario dos abstracciones para la aplicación y para el documento.
- Los clientes tienen que realizar implementaciones especificas para las subclases.
- La aplicación es responsable de su creación y de su manejo.

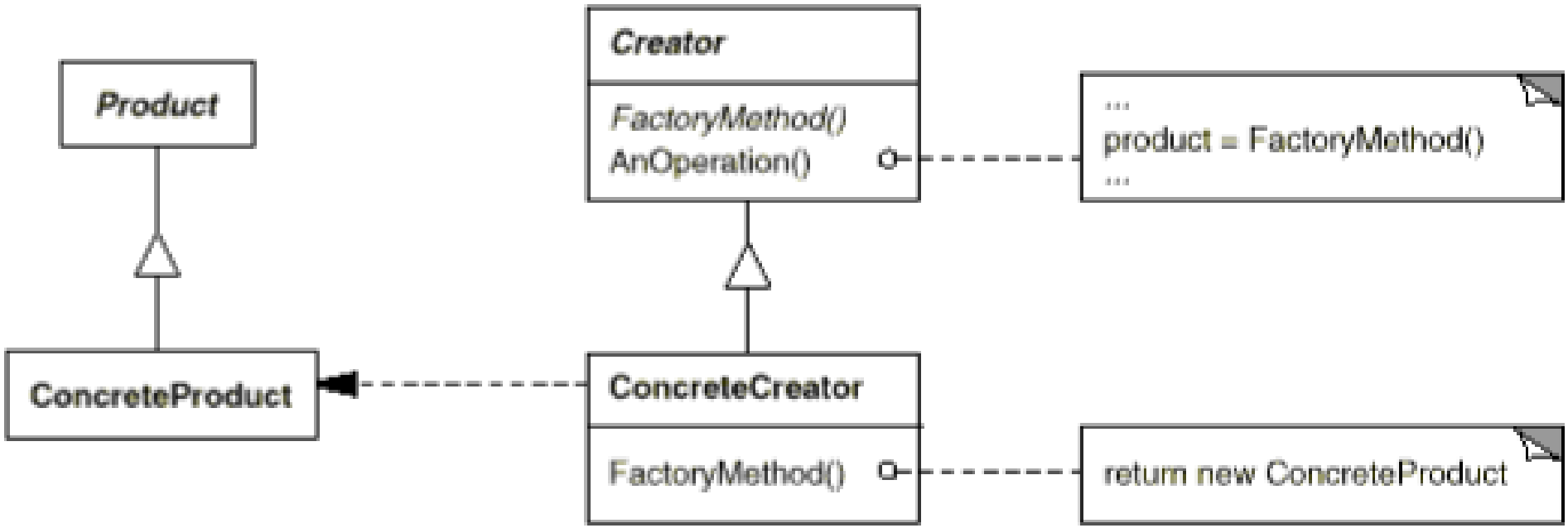
Factory Method

- La aplicación no sabe que tipo de documento se va a crear.
- La aplicación solo sabe cuando un nuevo tipo de documento es creado.
- El framework sabe cuando se crea un nuevo documento no de que tipo es.

Factory Method

- Aplicable cuando no se puede anticipar si una clase de objeto será creado.
- Una clase quiere sus subclases para especificar sus objetos creados.
- Unca clase delega responsabilidades para una de muchas subclases ayudantes.

Factory Method



Factory Method

- Producto (documento) define la interface para los objetos que factory method crea.
- ConcreteProduct (Mi documento) implementa la interface del producto.
- Creador (aplicación) declara el factory method.

Factory Method

- Creador (aplicación) define una implementación por defecto del factory method
- Creador retorna un objeto del tipo Producto concreto.
- Creador concreto (Mi aplicación) sobreescribe el factory method para retornar una instanciación de un producto concreto.

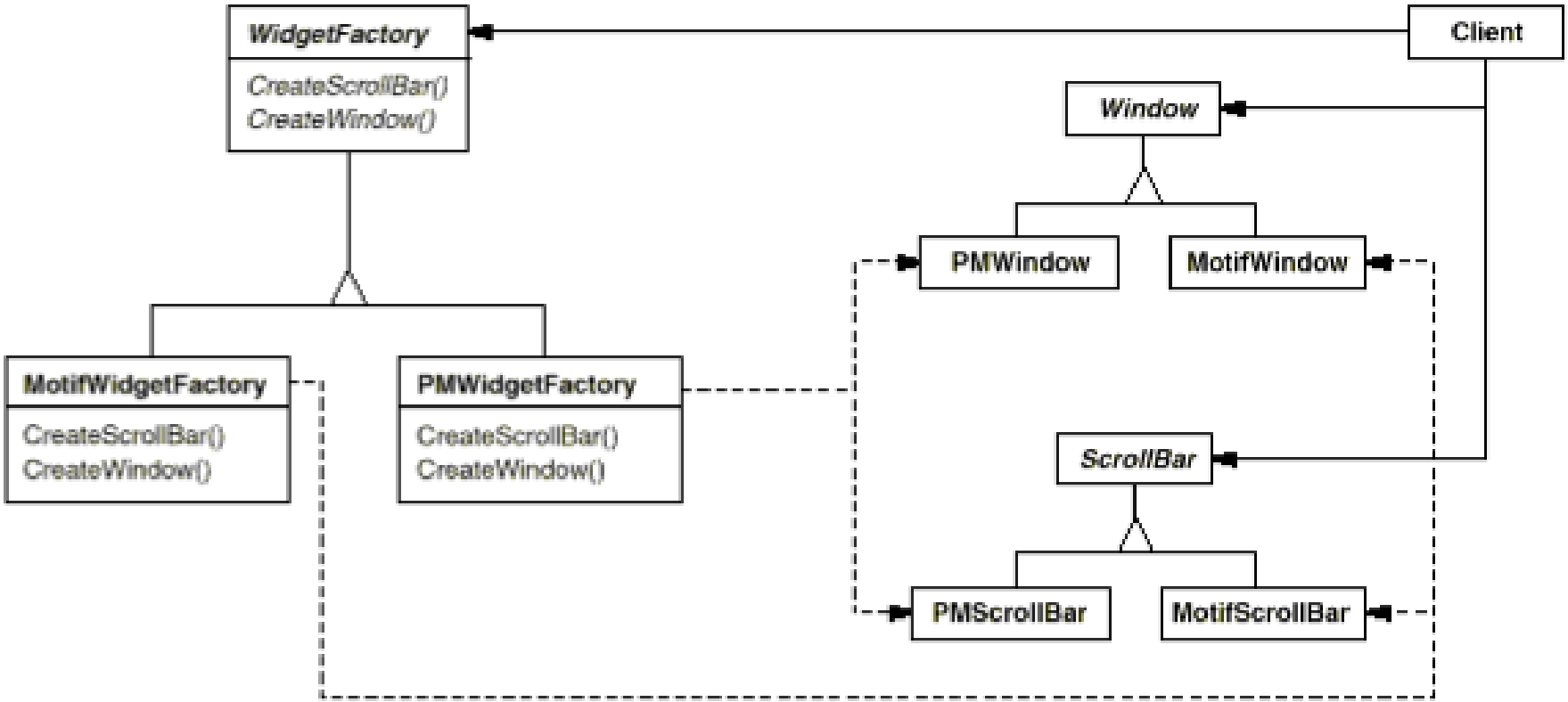
Factory Method

- Una desventaja es obligar a los clientes a crear subclases de la clase Creador para crear un objeto concreto.
- Esto esta bien cuando el cliente quiere hacer subclases de la clase creador.
- Eso obliga al cliente a lidiar con otro punto de evolución.

Abstract Factory

- Provee interfaces para la creación de familias de objetos.
- Estos objetos pueden estar relacionados o ser dependientes.
- Realiza esto sin implementar su clase concreta

Abstract Factory



Abstract Factory

- Desea un conjunto de herramienta con una interfaz de usuario que admita múltiples estándares.
- Esto genera diferentes comportamientos a la vez que diferentes apariencias.
- Después se puede complicar el cambio de apariencia.

Abstract Factory

- Lo ideal es no crear una aplicación con una apariencia en particular.
- La creación de elementos especificos en apariencia y sensación dificulta su cambio posterior.
- Es necesario declarar una interface para crear cada tipo básico de elemento.

Abstract Factory

- Hay una clase abstracta para cada tipo de elemento.
- Existen subclasses concretas implementando elementos para cada especifica estandar de sensación.
- En nuestro caso la interface WidgetFactory tiene una operación que devuelve un nuevo objeto widget por cada clase abstracta widget.

Abstract Factory

- Los clientes llaman para obtener estas operaciones para obtener instancias de widgets.
- No obstante los clientes no conocen las clases concretas que están usando.
- Los clientes se mantienen independientes del aspecto y apariencia prevaleciente.

Abstract Factory

- Los clientes llaman para obtener estas operaciones para obtener instancias de widgets.
- No obstante los clientes no conocen las clases concretas que están usando.
- Los clientes se mantienen independientes del aspecto y apariencia prevaleciente.

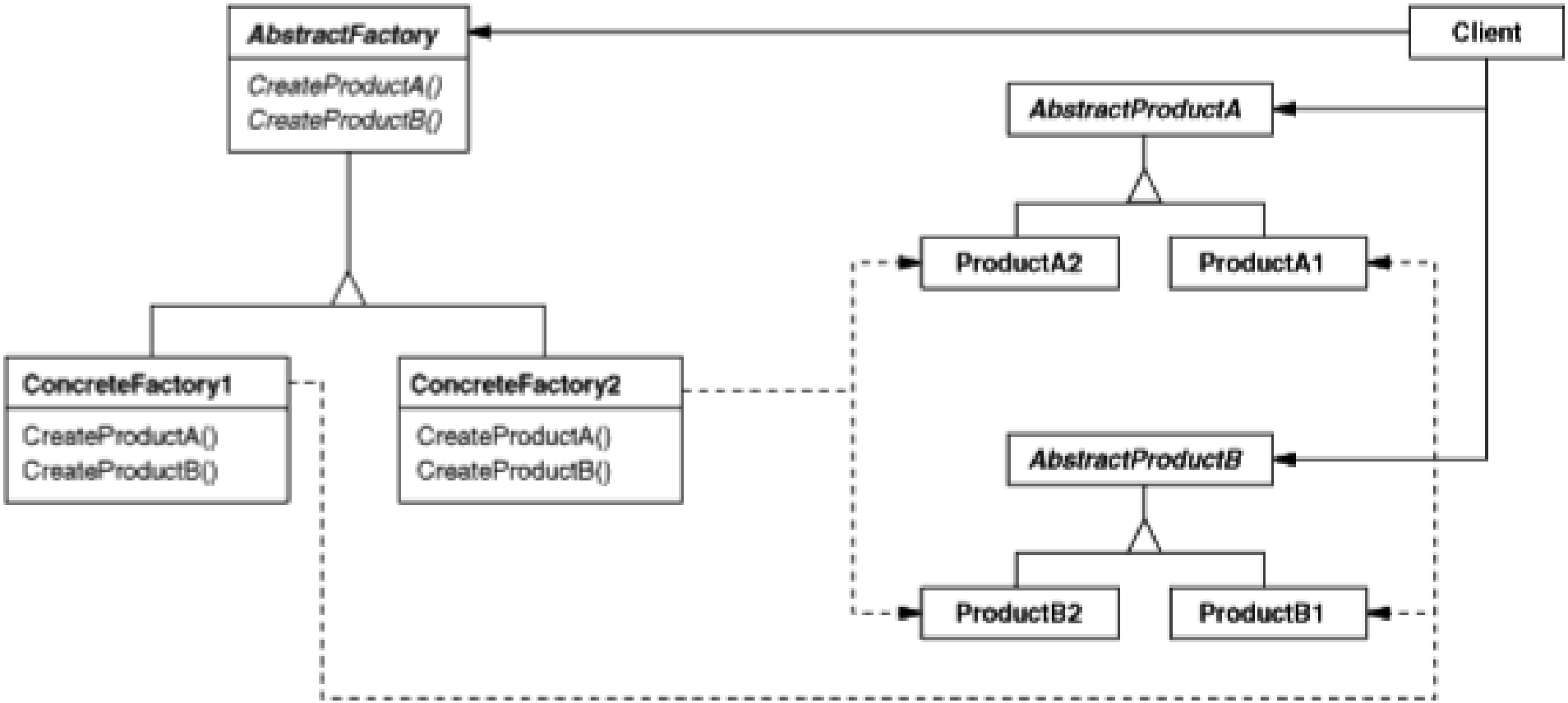
Abstract Factory

- Hay una subclase concreta del WidgetFactory para cada estándar de sensación.
- Cada subclase implementa operaciones para crear el widget apropiado para dicha apariencia.
- El widgetFactory impone dependencia entre las clases widgets concretos.

Abstract Factory

- Un sistema debe ser independiente de como sus productos son creados, compuestos o presentados.
- Un sistema debe ser configurardos con una de muchas familias de productos.
- Una familia de productos de objetos relacionados es diseñado para ser usado juntos y son necesarios para cumplir esta restricción.
- Si tu quieres proveer una libreria de productos y quieres solo revelar sus interfaces, no sus implementaciones.

Abstract Factory



Abstract Factory

- AbstractFactory(WidgetFactory) declara una interface para operaciones que crea objetos productos abstractos.
- ConcreteFactory(MotifWidgetFactory, PMWidgetFactory) implementa las operaciones para crear objetos productos concretos.
- ConcreteProduct(MotifWindow,MotifScrollBar) define un objeto producto para ser creado por la fabrica concreta correspondiente.

Abstract Factory

- ConcreteProduct(MotifWindow,MotifScrollBar) implementa una interface AbstractProduct.
- Client usa solamente interfaces declaradas por las clases AbstractFactory y AbstractProduct.

Prototype

- Especifica el tipo de objeto para crear usando una instancia prototipica.
- Crea nuevos objetos copiando este prototipo.

Prototype

- Por ejemplo se desea crear un editor de partituras musicales personalizado.
- Personaliza el framework para el editor gráfico.
- Debe permitir agregar nuevas notas, silencios y pentagramas.

Prototype

- El framework del editor debe tener una paleta de herramientas para agregar objetos de musica a la partitura.
- Incluira herramientas de sleección, movimiento y manipulación de objetos de música.
- Los usuarios harán clic en la herramienta de notas negras y la utilizarán para agregar notas al puntaje.

Prototype

- Pueden tener la posibilidad de mover las notas hacia arriba y abajo dentro del pentagrama.
- Esto cambiará su tono.
- El framework provee una clase abstracta grafica para componentes gráficos.

Prototype

- La clase abstracta grafica no sabe cómo crear instancias de nuestras clases de músicas para agregar a la partitura.
- Al crear subclases instancias de la clase grafica producira muchas subclases que se diferencia solamente en el tipo de objeto de música.
- Como usar el framework para parametrizar instancias de la herramienta gráfica para la clase gráfica que se supone crear?

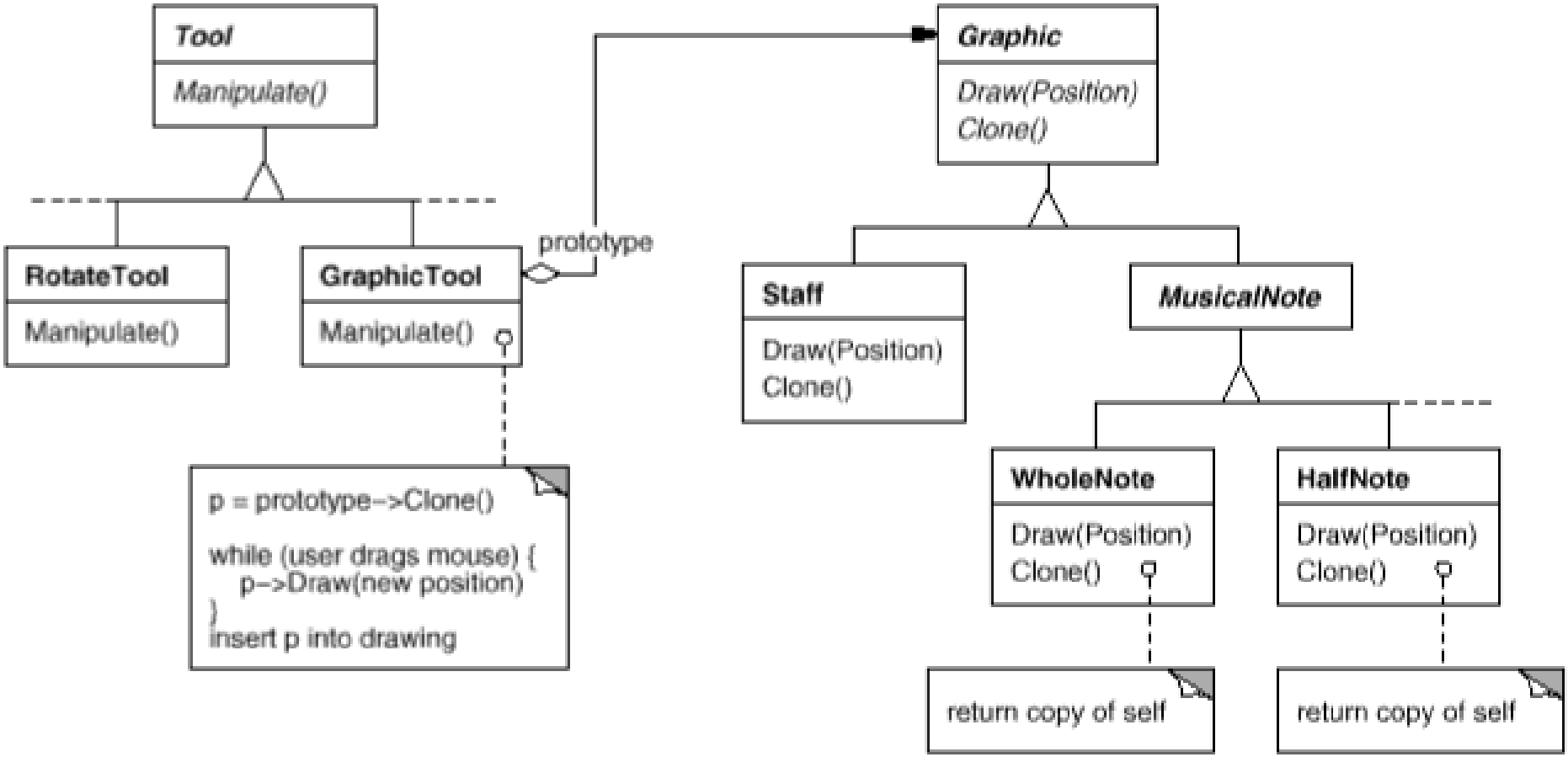
Prototype

- La solución es crear un nuevo objeto gráfico copiando o clonando una instancia de una subclase gráfica.
- Se puede llamar a esta instancia un prototipo (prototype).
- La herramienta gráfica es parametrizada para el prototipo clonandolo y añadiendo al documento.

Prototype

- Si las clases gráficas soportan la operación de clonación entonces la herramienta gráfica puede clonar cualquier tipo de gráfico.
- Cada herramienta crea un objeto musica atravez de una instancia de la herramienta grafica esta inicialización es con un diferente prototipo.
- Cada instanciación de la herramienta gráfica produce un objeto musical por clonación su prototipo y añadido a la partitura.

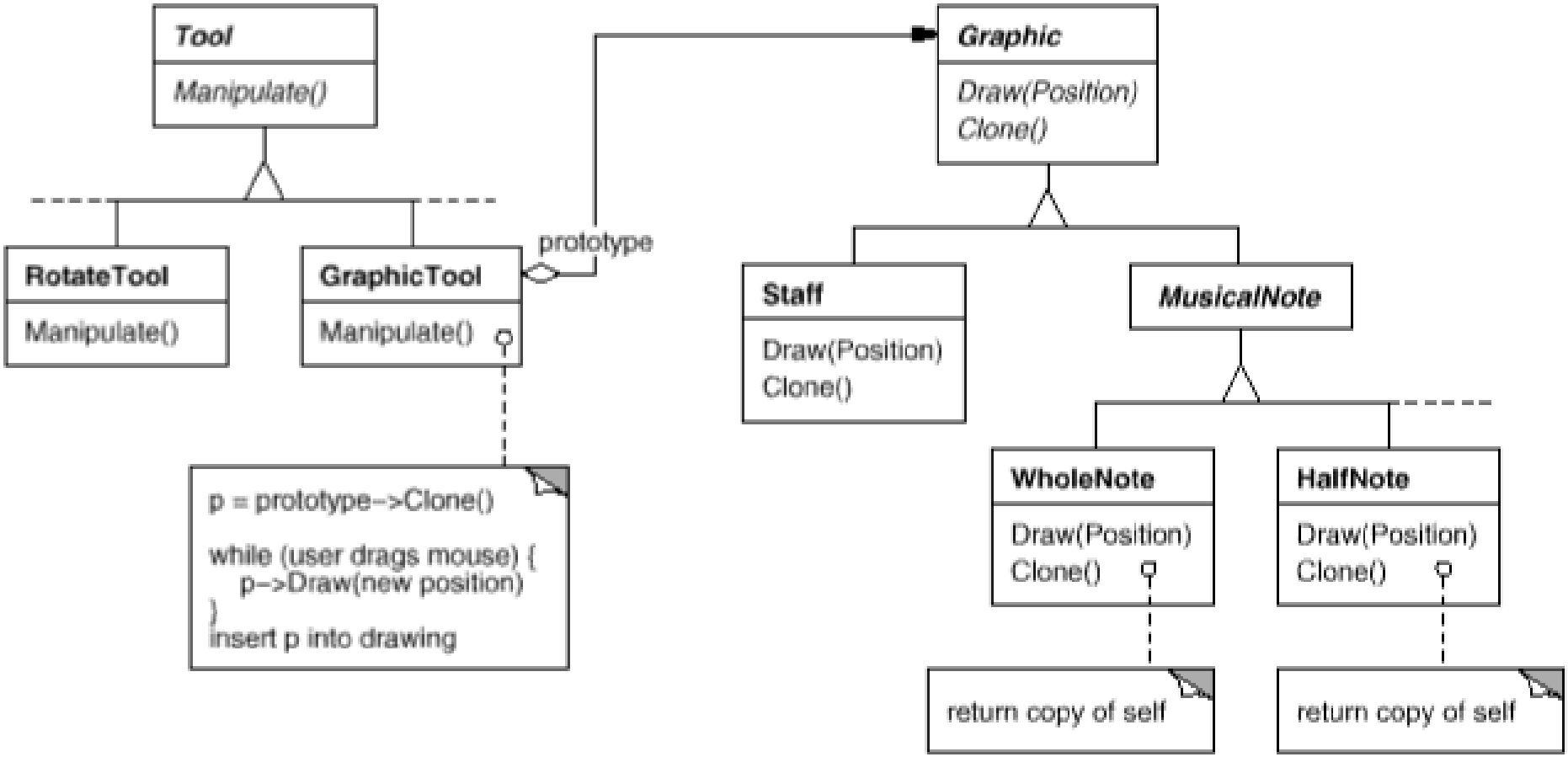
Prototype


Tool
Manipulate()
RotateTool
Manipulate()
GraphicTool
Manipulate()
prototype
Graphic
Draw(Position)
Clone()
Staff
Draw(Position)
Clone()
MusicalNote
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
return copy of self
return copy of self

Prototype

- Se puede usar el patrón prototype para reducir el número de clases.
- En nuestro ejemplo tenemos clases separadas para las notas enteras y medias notas.
- Para solucionar esto se puede crear instancias de la misma clase.

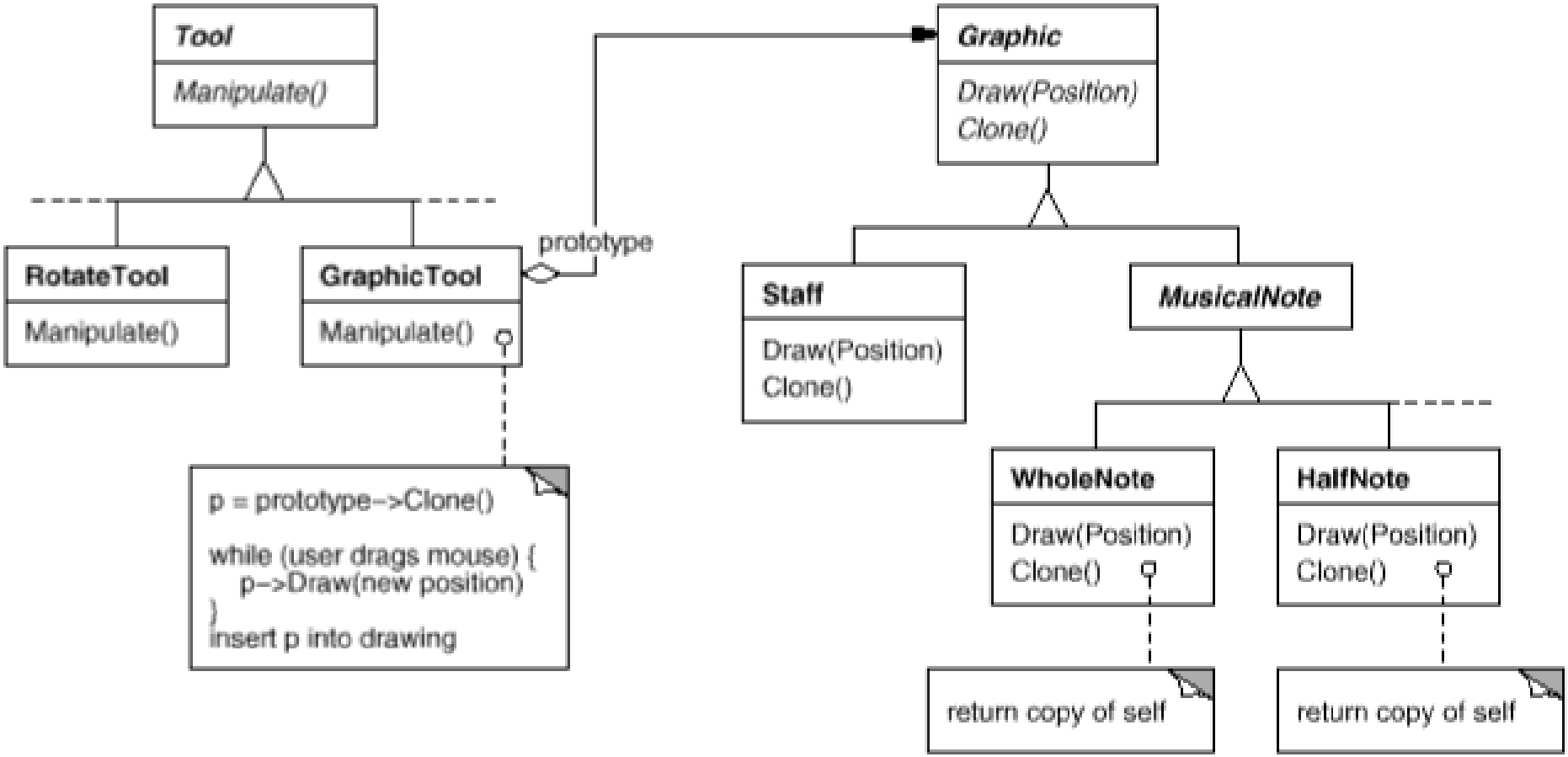
Prototype


Tool
Manipulate()
RotateTool
Manipulate()
GraphicTool
Manipulate()
prototype
Graphic
Draw(Position)
Clone()
Staff
Draw(Position)
Clone()
MusicalNote
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
return copy of self
return copy of self

Prototype

- Esta instanciación puede ser con diferentes mapas de bits y duraciones.
- La herramienta encargada de crear nuevas notas se convierte en un GraphicTool.
- El prototipo de este es una MusicalNote inicializado para ser una nota completa.

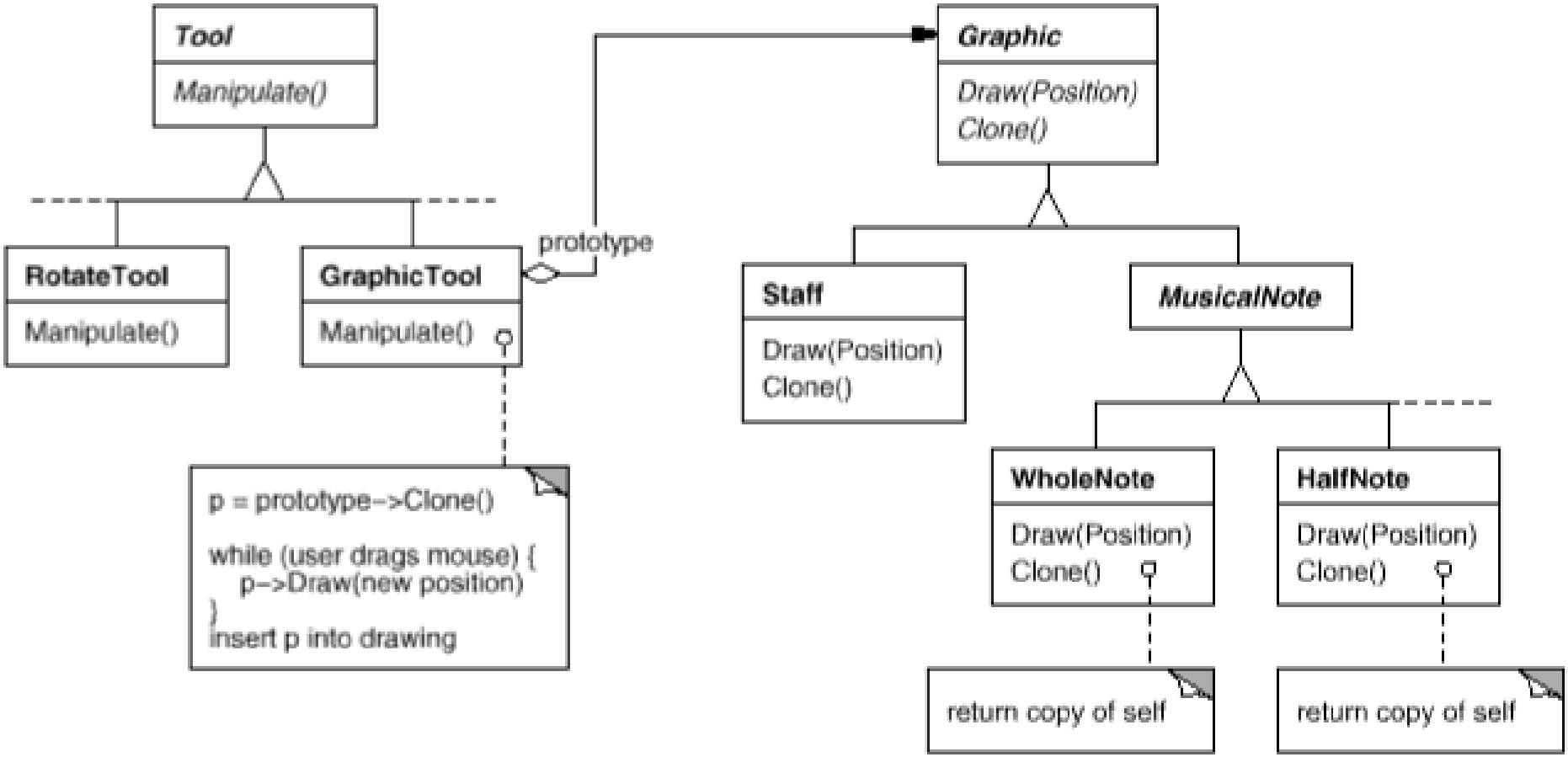
Prototype


Tool
Manipulate()
RotateTool
Manipulate()
GraphicTool
Manipulate()
prototype
Graphic
Draw(Position)
Clone()
Staff
Draw(Position)
Clone()
MusicalNote
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
return copy of self
return copy of self

Prototype

- Esto puede reducir la cantidad de clases en el sistema.
- También posibilita la creación de nuevos tipos de notas.

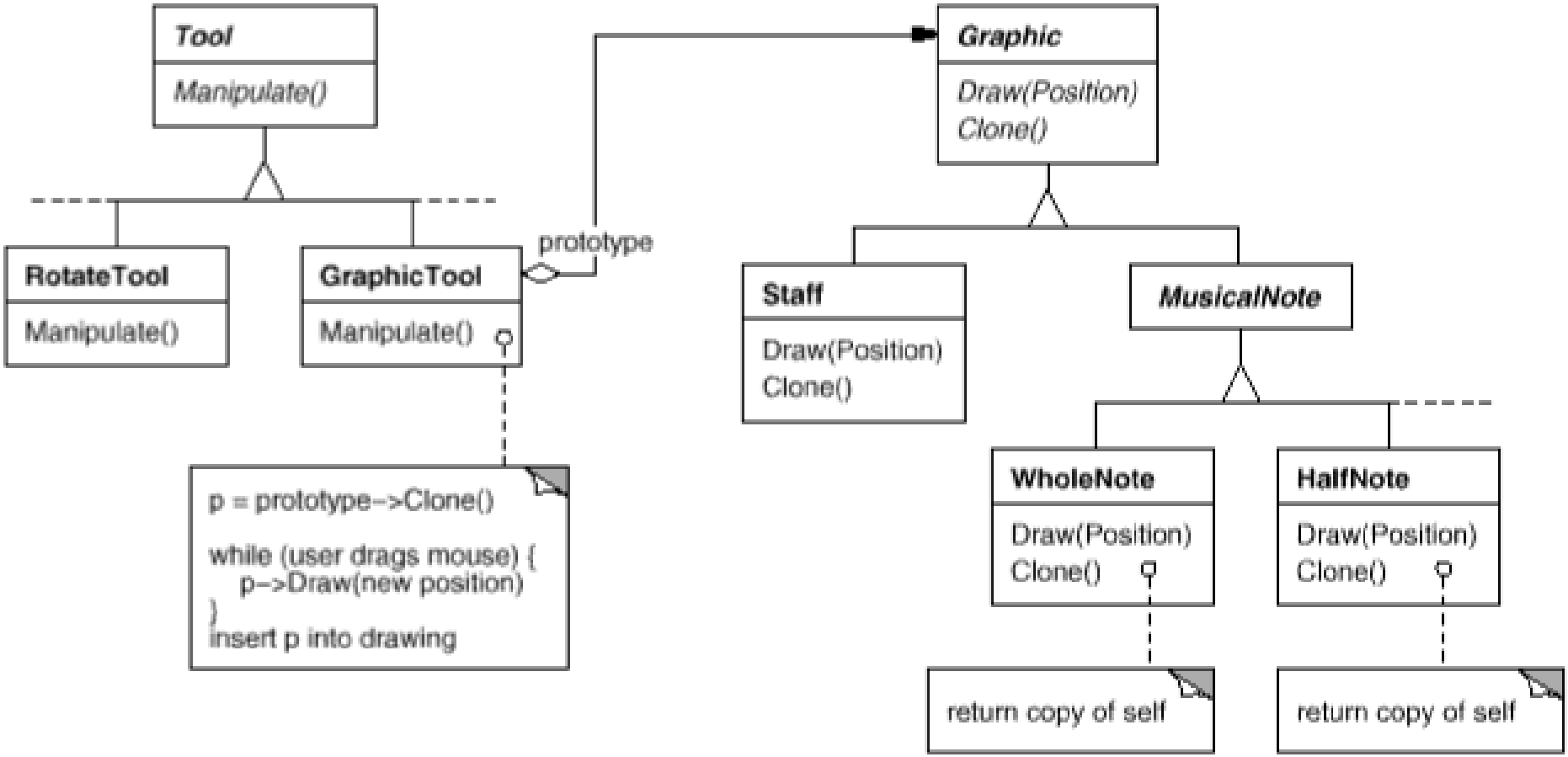
Prototype


Tool
Manipulate()
RotateTool
Manipulate()
GraphicTool
Manipulate()
prototype
Graphic
Draw(Position)
Clone()
Staff
Draw(Position)
Clone()
MusicalNote
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
return copy of self
return copy of self

Prototype

- Se usará el patrón prototype cuando el sistema debe ser independiente de como se crean los productos.
- También puede proveer independencia en caso de como somponen.
- Del mismo modo de como se representan los productos.

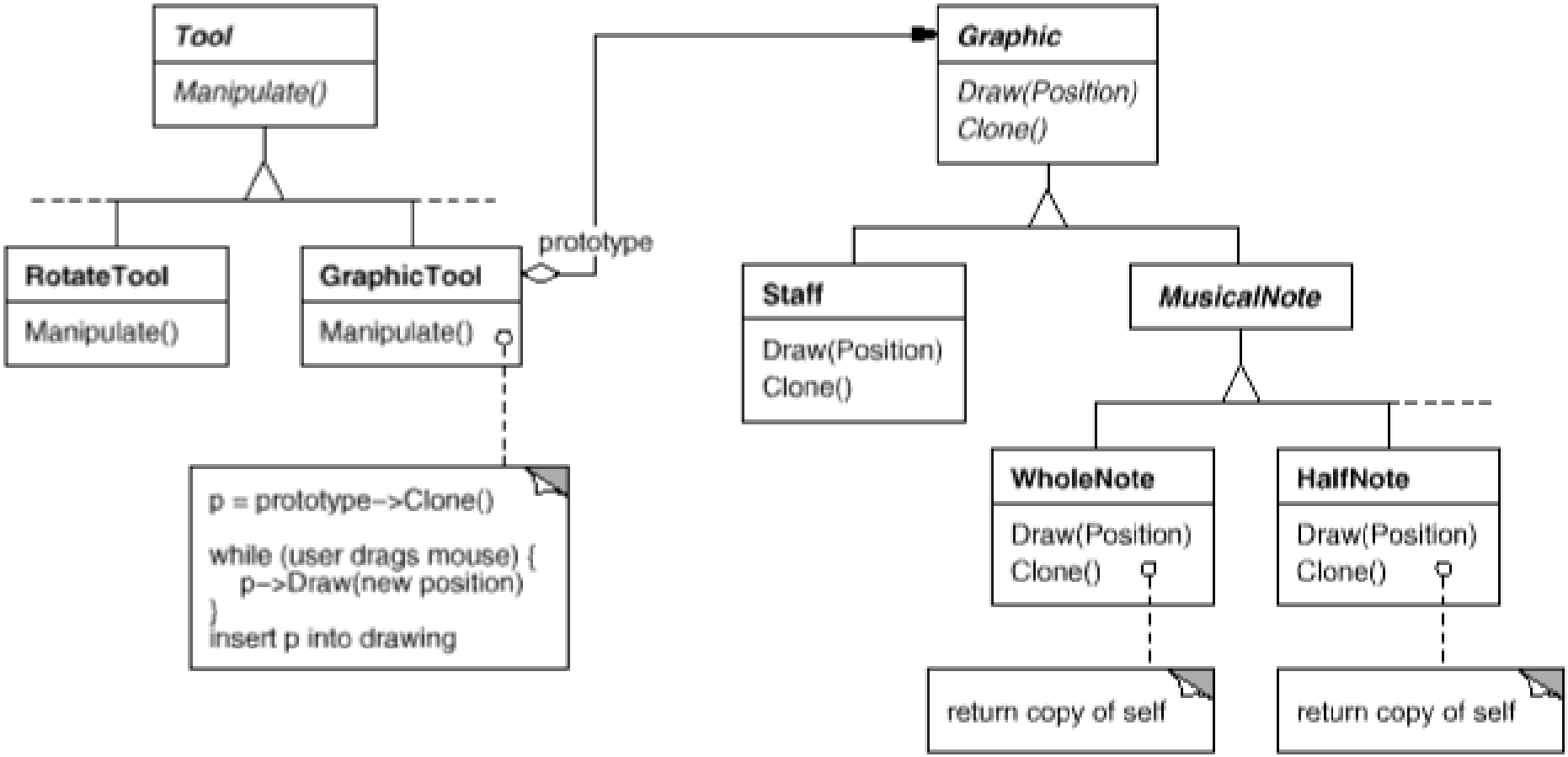
Prototype


Tool
Manipulate()
Graphic
Draw(Position)
Clone()
prototype
RotateTool
Manipulate()
GraphicTool
Manipulate()
Staff
Draw(Position)
Clone()
MusicalNote
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
return copy of self
return copy of self

Prototype

- Cuando las clases a instanciar se especifican en tiempo de ejecución.
- Puede ser en el caso que se haga carga dinámica.
- Dejar un programa en memoria principal y moverlo a otra posición diferente.

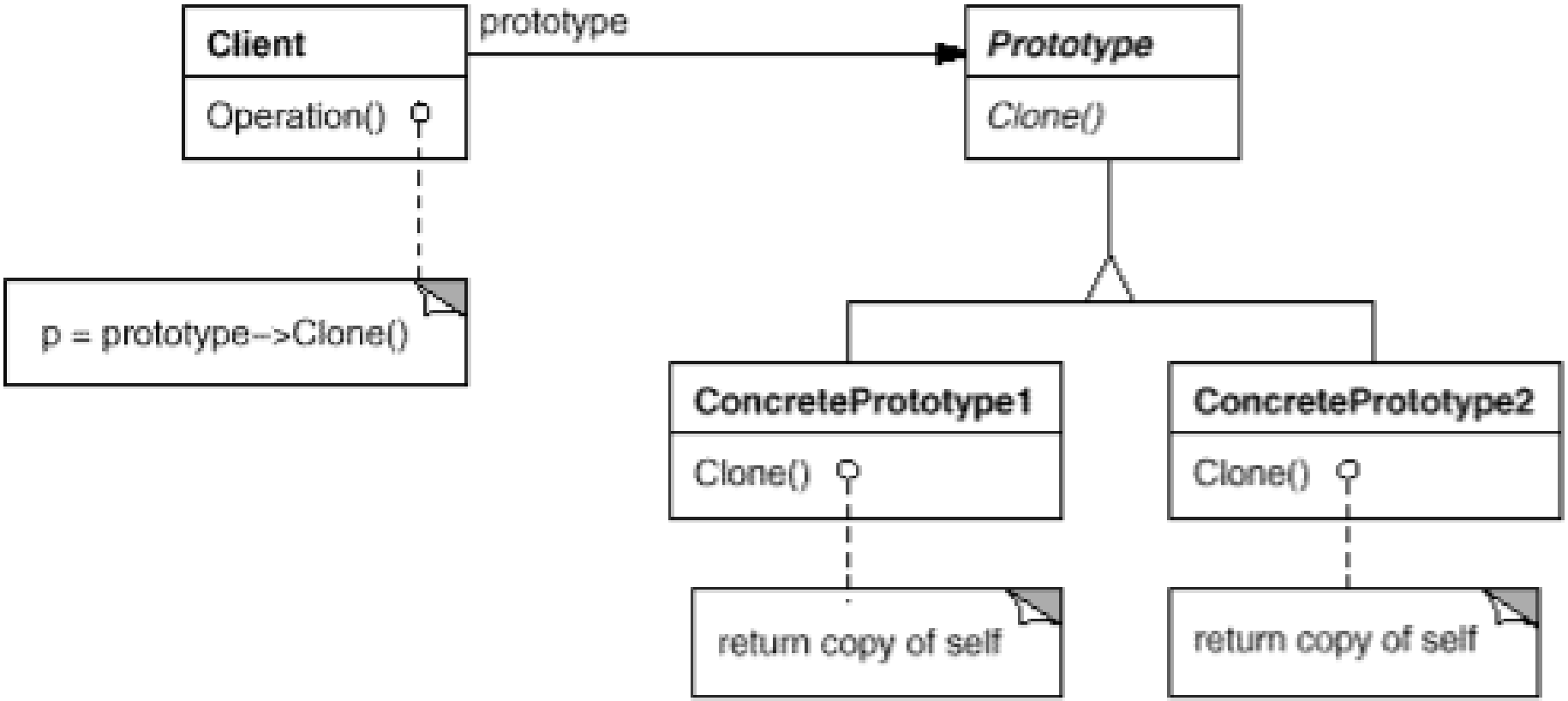
Prototype


Tool
Manipulate()
RotateTool
Manipulate()
GraphicTool
Manipulate()
prototype
Graphic
Draw(Position)
Clone()
Staff
Draw(Position)
Clone()
MusicalNote
WholeNote
Draw(Position)
Clone()
HalfNote
Draw(Position)
Clone()
p = prototype->Clone()
while (user drags mouse) {
p->Draw(new position)
}
insert p into drawing
return copy of self
return copy of self

Prototype

- Evitar la presencia de jerarquia de clases fábricas en paralelo con la jerarquia de clases productos.
- Cuando una de las instancias de la clase puede tener solo una de pocas combinaciones de diferentes estados.
- Es mejor crear un número correspondiente de prototipos y clonarlos en lugar de crear instancias de la clase manualmente. En cada ocasión con el apropiado estado.

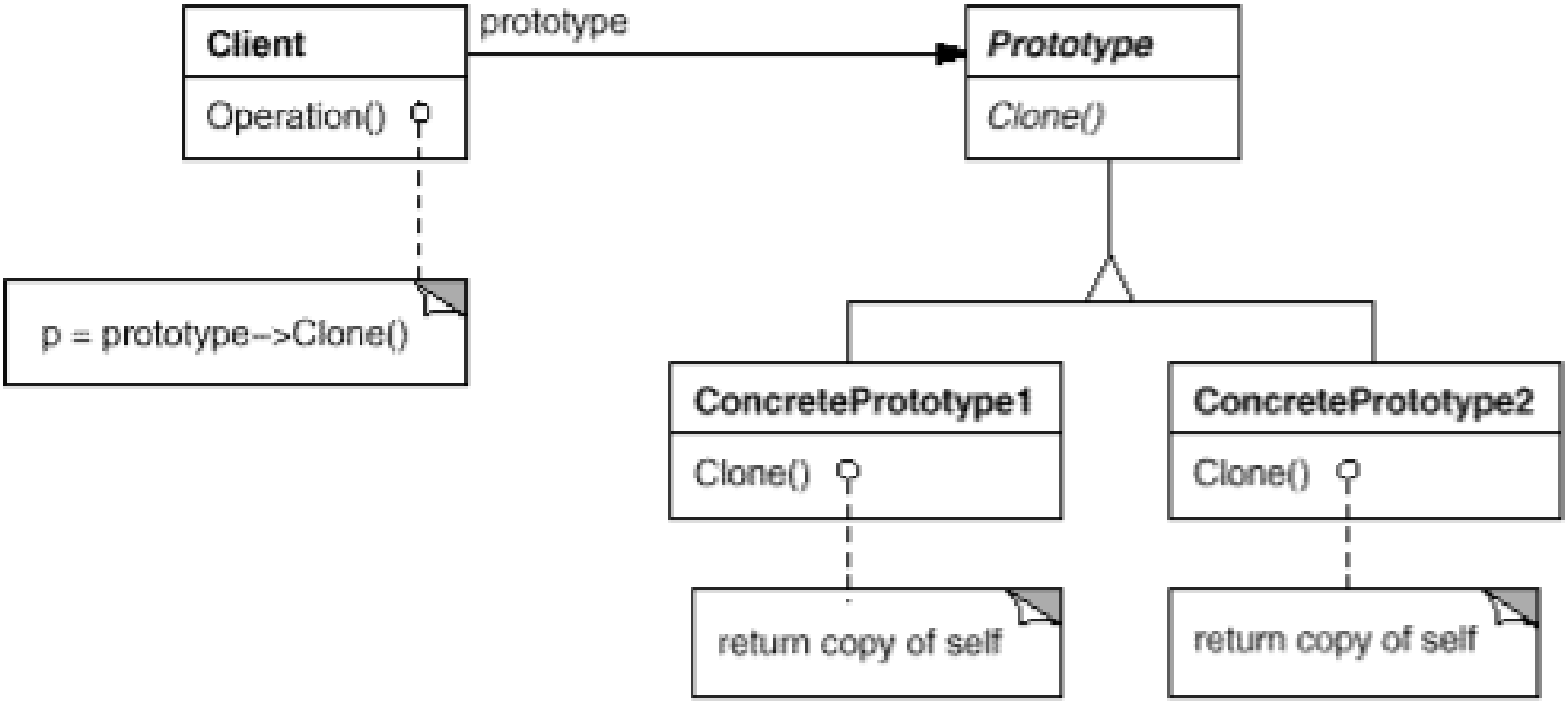
Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Prototype

- La clase prototype puede ser ejemplificada con la clase gráfico.
- Esta declara una interface para clonarse a si misma.

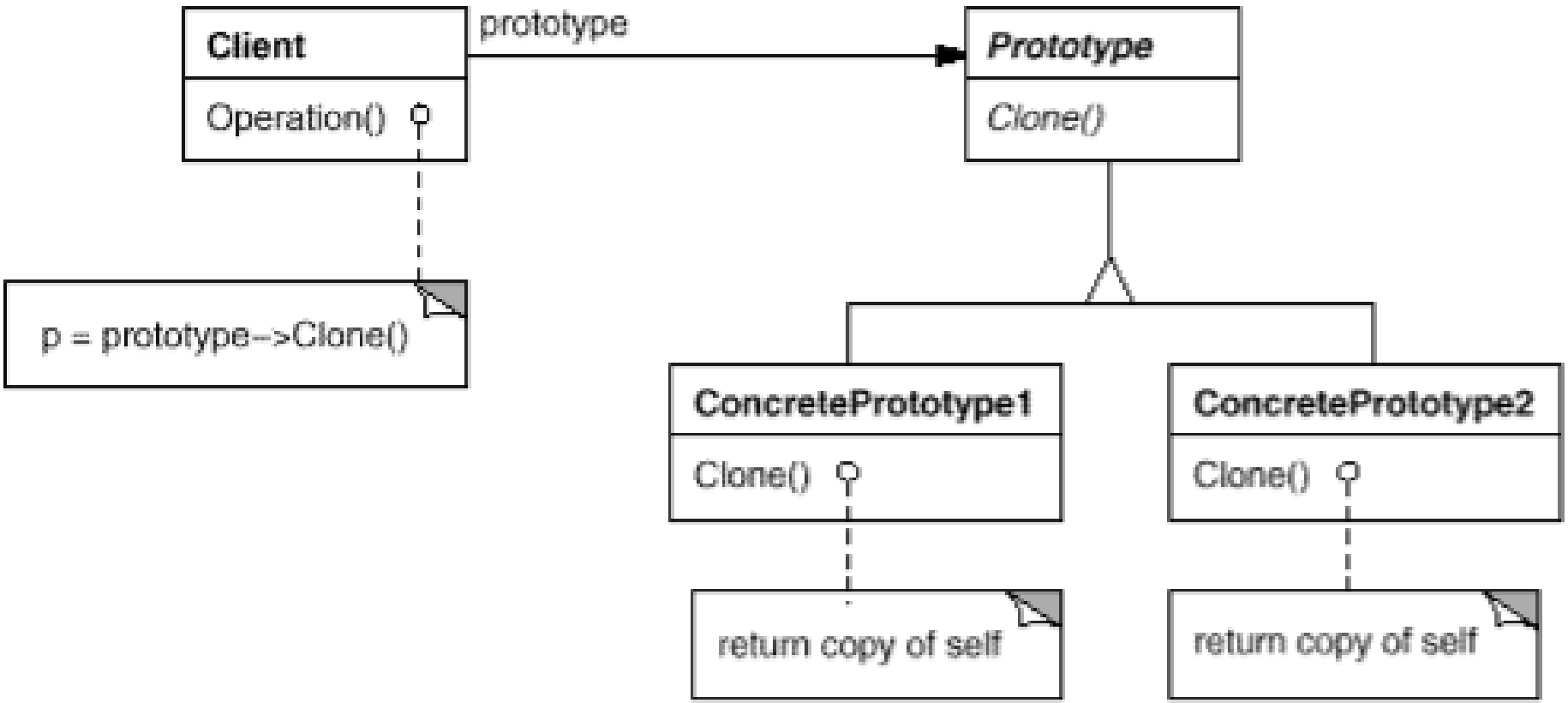
Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Prototype

- La clase concretePrototype puede ser ejemplificada por la clase Nota Entera, Media nota y el pentagrama.
- Implementa una operación para clonarse a si misma.

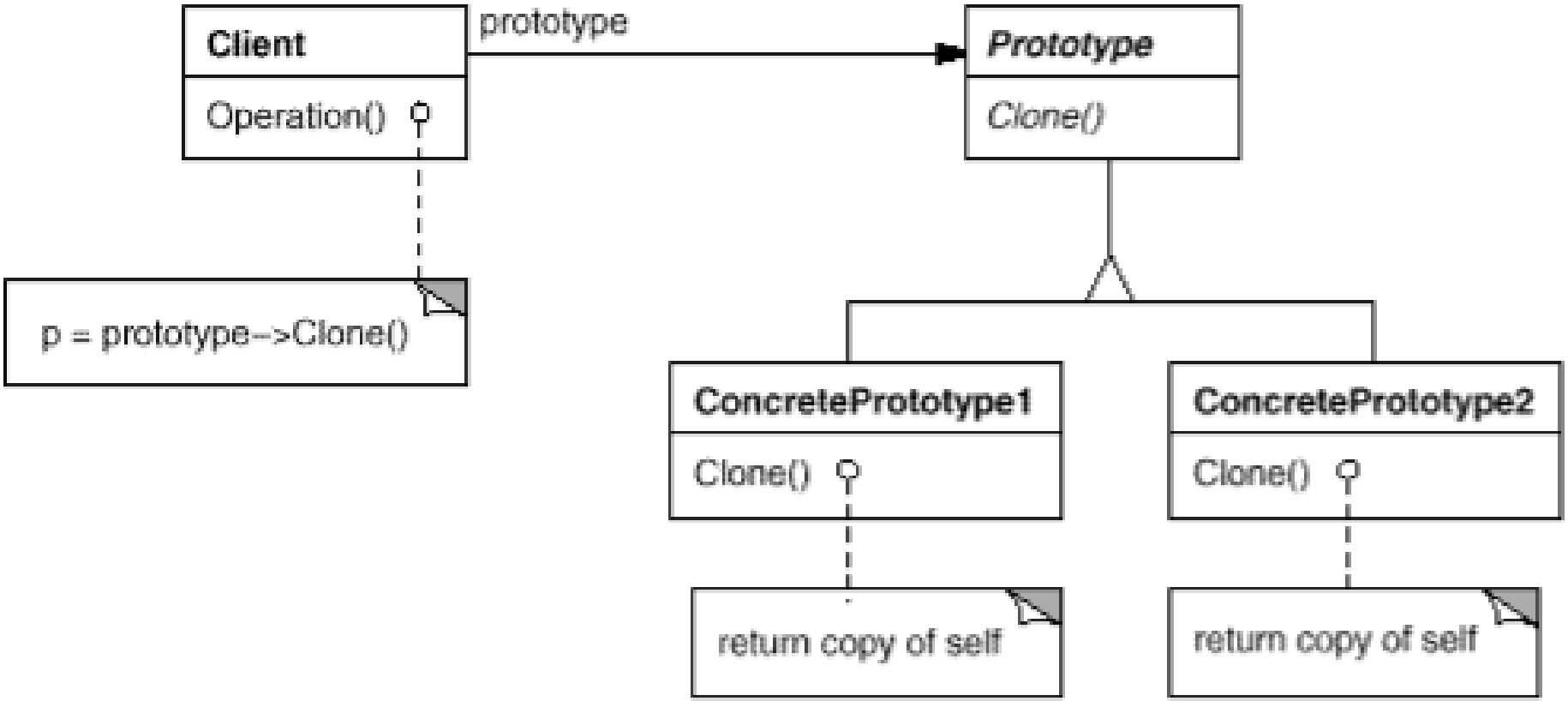
Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Prototype

- La clase Client puede ser ejemplificada por la clase GraphicTool
- Crea un nuevo objeto pidiendo a un prototipo para clonarse a si mismo.

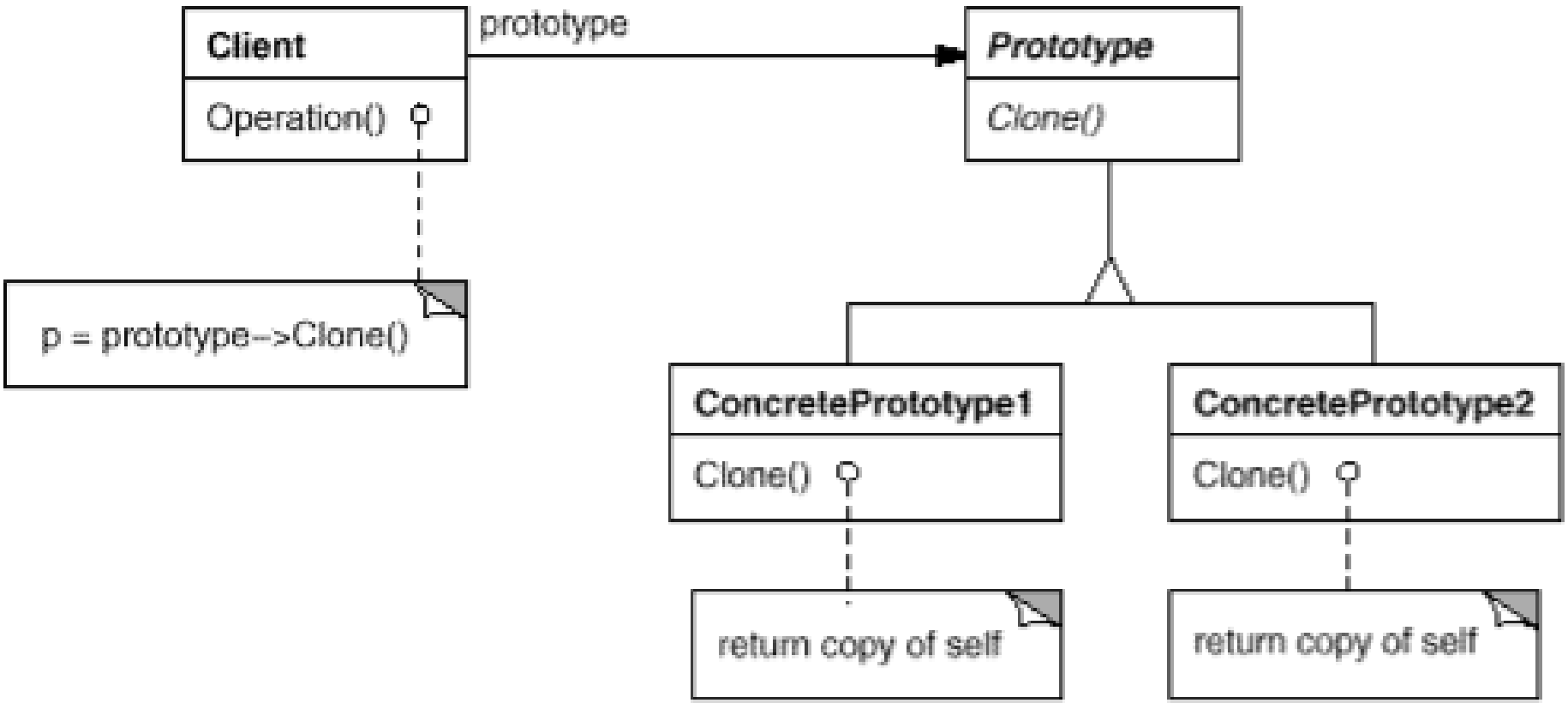
Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Prototype

- Tiene varias de las consecuencias de Abstract Factory y Builder.
- Ocultar las clases de los producto concretos del cliente.
- Lo que reduce la cantidad de nombres que los clientes conocen.
- Estos patrones permiten al cliente trabajar con clases especificas de la aplicación sin modificaciones.

Prototype


Client
Operation()
prototype
Prototype
Clone()
p = prototype->Clone()
ConcretePrototype1
Clone()
ConcretePrototype2
Clone()
return copy of self
return copy of self

Singleton

- Asegurarse de que una clase tenga solo una instancia.
- Proporcionar un punto de acceso global a esta clase.

Singleton

- Es importante que algunas clases tengan exactamente una instancia.
- Asi por ejemplo, si existen varias impresoras, solo existe una cola de impresión.
- Solo debe haber un sistemas de archivos.
- Solo debe haber un administrador de ventanas.

Singleton

- Solo debe haber un administrador de ventanas.
- Solo hay un filtro digital en un convertidor A/D.
- Solo un sistema contable para una empresa.

Singleton

- ¿Cómo creamos una clase que tenga solo una instancia (con garantia) y que sea facilmente accesible la instancia?
- Una variable global hace de un objeto accesible.
- Esa misma variable no impide la creación de múltiples objetos.

Singleton

- Podemos hacer que la clase haga un registro de su propia instancia.
- La clase debe garantizar que no se cree ninguna otra instancia.
- Interceptando la solicitud de creación de nuevos objetos.

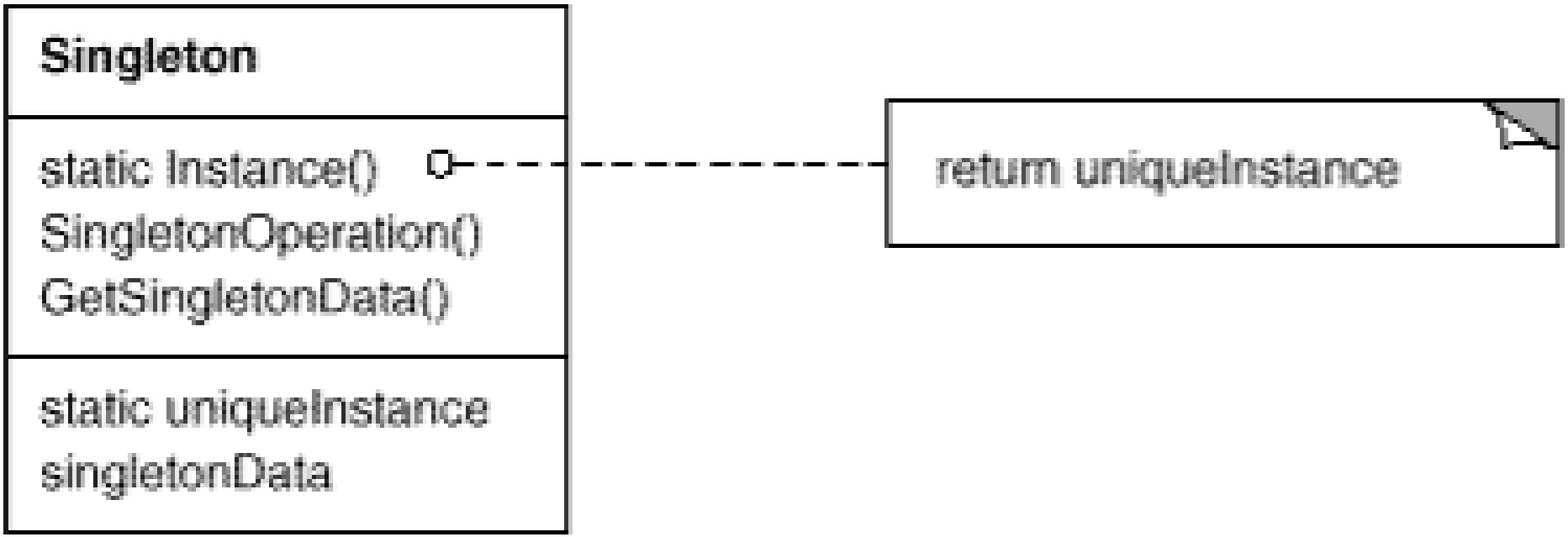
Singleton

- Mediante esta creación garantizada de una sola instancia.
- Junto con la forma de acceder a dicha instancia.
- Esto es el patron Singleton.

Singleton

- Se usa cuando se necesita exactamente una instancia de una clase.
- También cuando solo debe ser accesible dicha clase desde un punto bien conocido a los clientes.
- La unica instancia puede ser accesible por subclases.
- Los clientes pueden usar una clase extendida sin modificar el código.

Singleton


Singleton
static Instance()
SingletonOperation()
GetSingletonData()
static uniqueInstance
singletonData
return uniqueInstance

Singleton

- Define una operación de instanciación para garantizar el acceso a los clientes de una instancia única.
- La instancia tiene una operación de clase (una funció miembro estática en C++)

Builder

- Separar la construcción de un complejo objeto de su representación
- También construir con el mismo proceso diferentes representaciones
- Esa es la intención de Builder

Builder

- Nuestro ejemplo utiliza el lector para el formato RTF
- RTF (Rich Text Format) utiliza un mecanismo de intercambio de para convertir a RTF muchos formatos de texto.
- Este lector puede convertir RTF a texto ASCII sin formato.
- Llamar a una pequeña aplicación (widget) de texto para ser editada de forma interactiva.

Builder

- Como abrimos y cerramos las posibles conversiones?
- Debe ser fácil agregar una nueva conversión sin modificar el lector.

Builder

- Una solución es redefinir la clase RTFReader con un objeto TextConverter para convertir RTF a otra representacón textual.
- Al mismo tiempo que RTFReader analiza el documento, TextConverter realiza la conversión.
- Por lo tanto esto se hace por pedazos.

Builder

- Estos pedazos son token RTF, una vez reconocido este token RTF por el RTFReader.
- Puede ser esta una palabra o un elemento de control RTF
- Se envia una solicitud para convertir el token en TextConverter

Builder

- Los objetos TextConverter son responsables de realizar la conversión de estos datos.
- También son responsables de realizar la representación del token en un formato en particular.

Builder

- Las clases derivadas de TextConverter tienen por lo tanto una especialización.
- La especialización de las clases derivadas de TextConverter son conversiones.
- La especialización de las clases derivadas de TextConverter son formatos.

Builder

- Un conversor ASCII puede convertir solo texto plano (o texto sin formato)
- Todas las demás solicitudes son rechazadas
- Existen otros conversores que no hacen eso

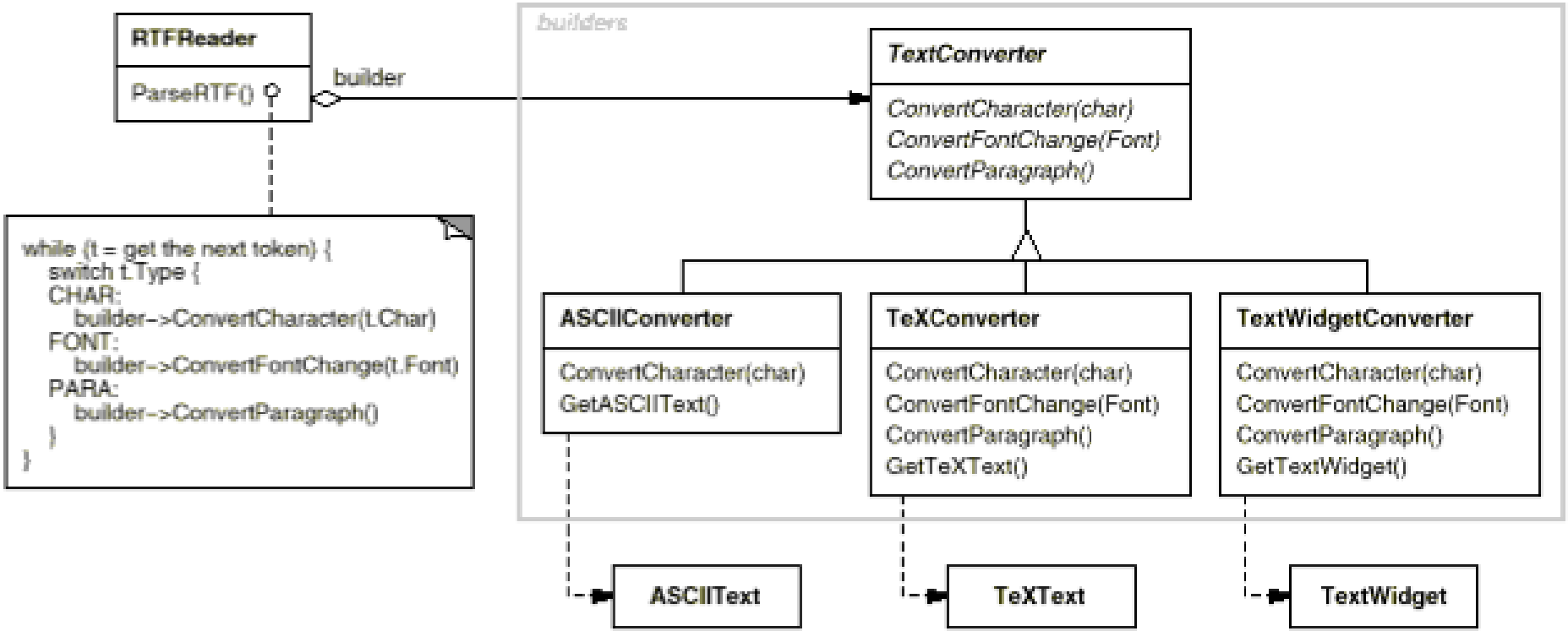
Builder

- Un TexConverter por otro lado implementará todas las operaciones necesarias para todoas las solicitudes para producir una representación TeX
- Esto permitiria capturar toda la información estilística en el texto
- Rescata mas información

Builder

- Un TextWidgetConverter debe producir un objeto interfaz de usuario complejo.
- Este elemento debe permitir al usuario ver el texto
- También le permite editar el texto

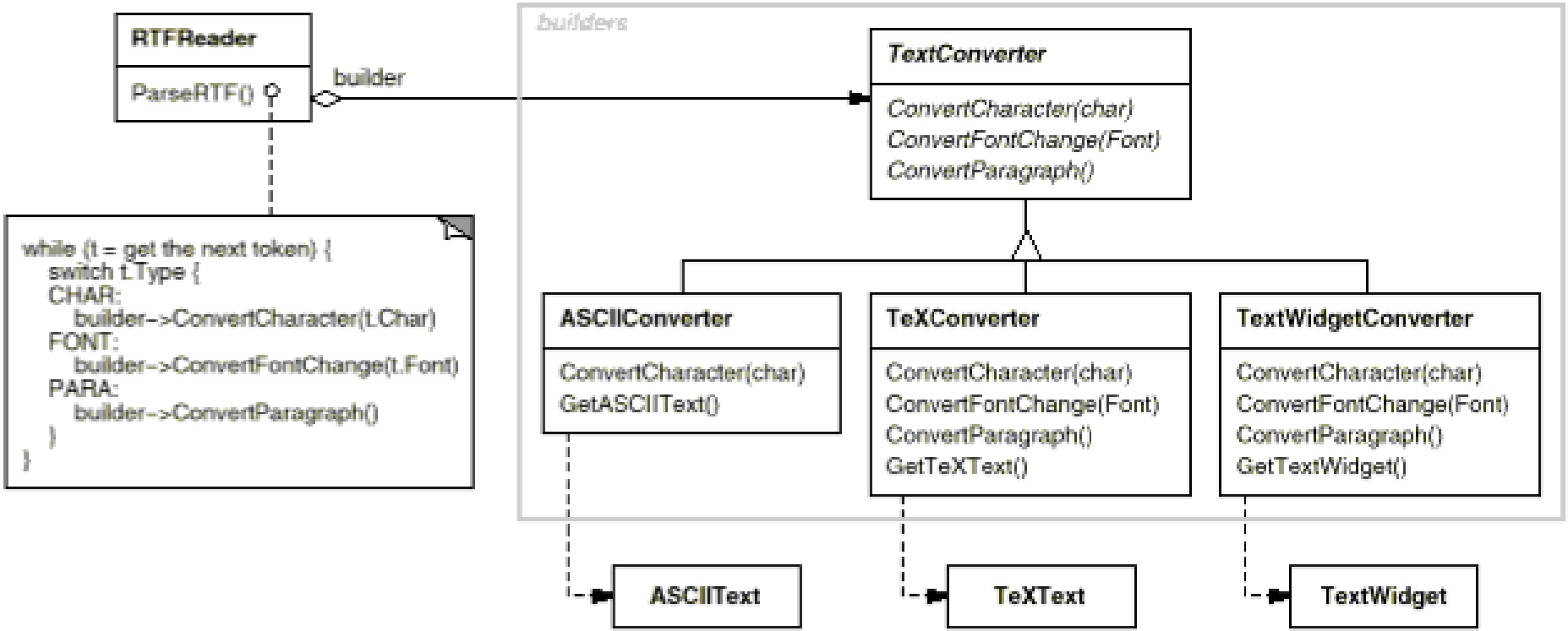
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- Cada tipo de clase convertidor usa el mecanismo de creación y ensamblaje de un objeto complejo.
- Este objeto complejo lo coloca anteponiendose a la interfaz abstracta.
- Este convertidor esta separado del lector.
- El lector analiza el documento RTF.

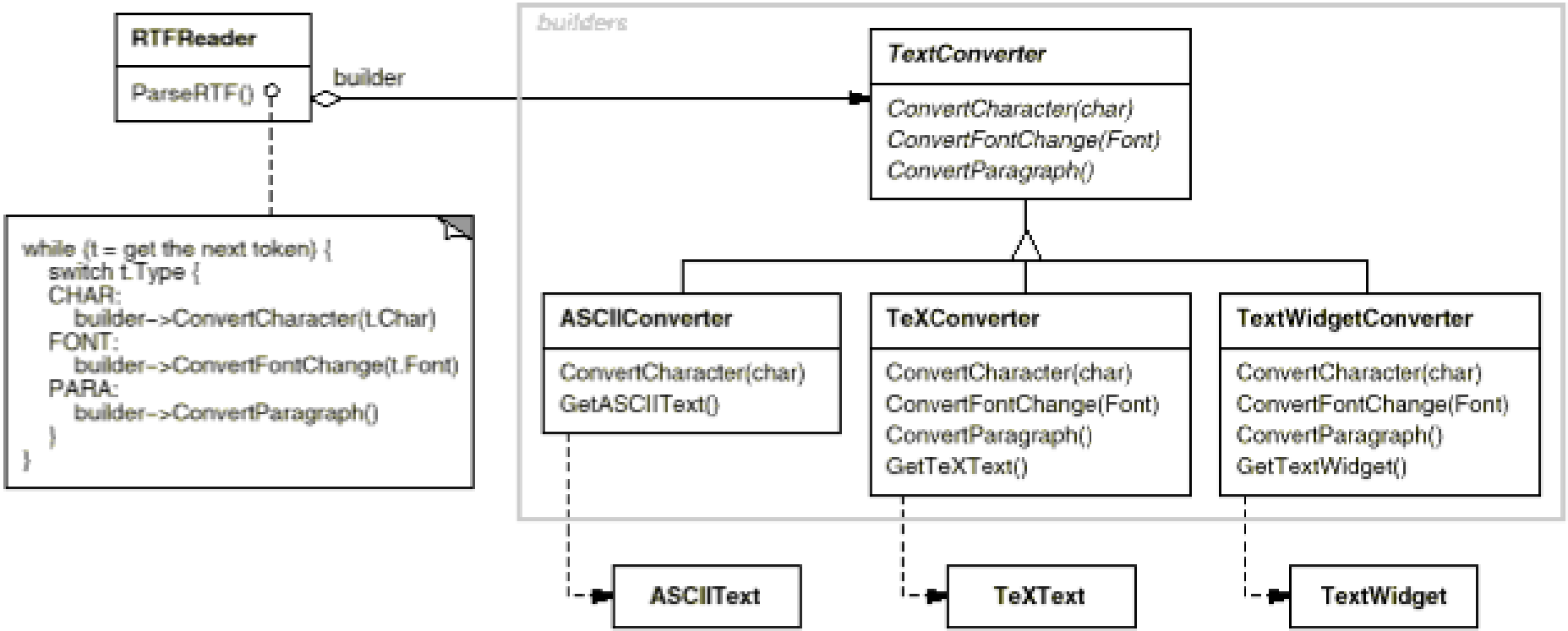
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- El patrón Builder maneja estas relaciones.
- En el patrón la clase convertidor se llama constructor.
- En el patrón la clase lector se llama director.

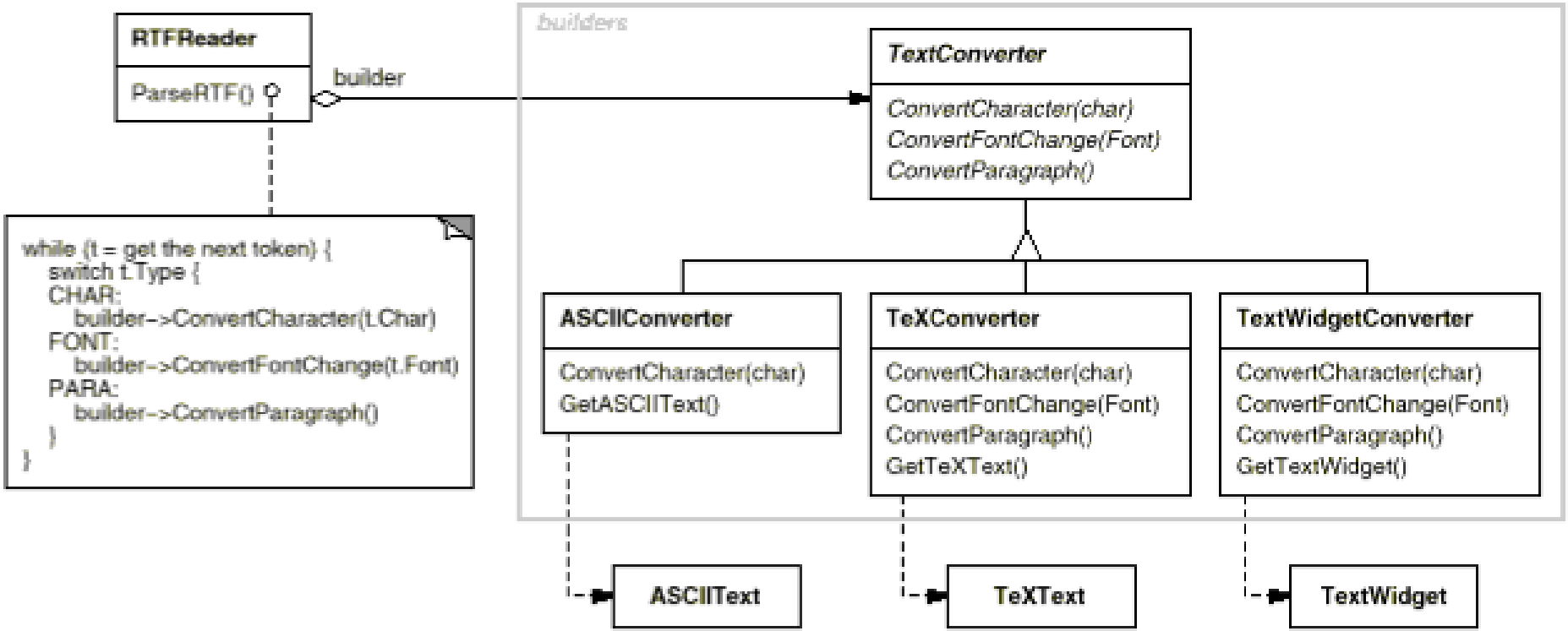
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- En nuestro ejemplo el patron Builder separa el algoritmo para interpretar un fomato textual.
- Separa el analizador del documento (el parser)
- Lo separa de la creación del formato convertido.
- Lo separa de la representación de un formato convertido.

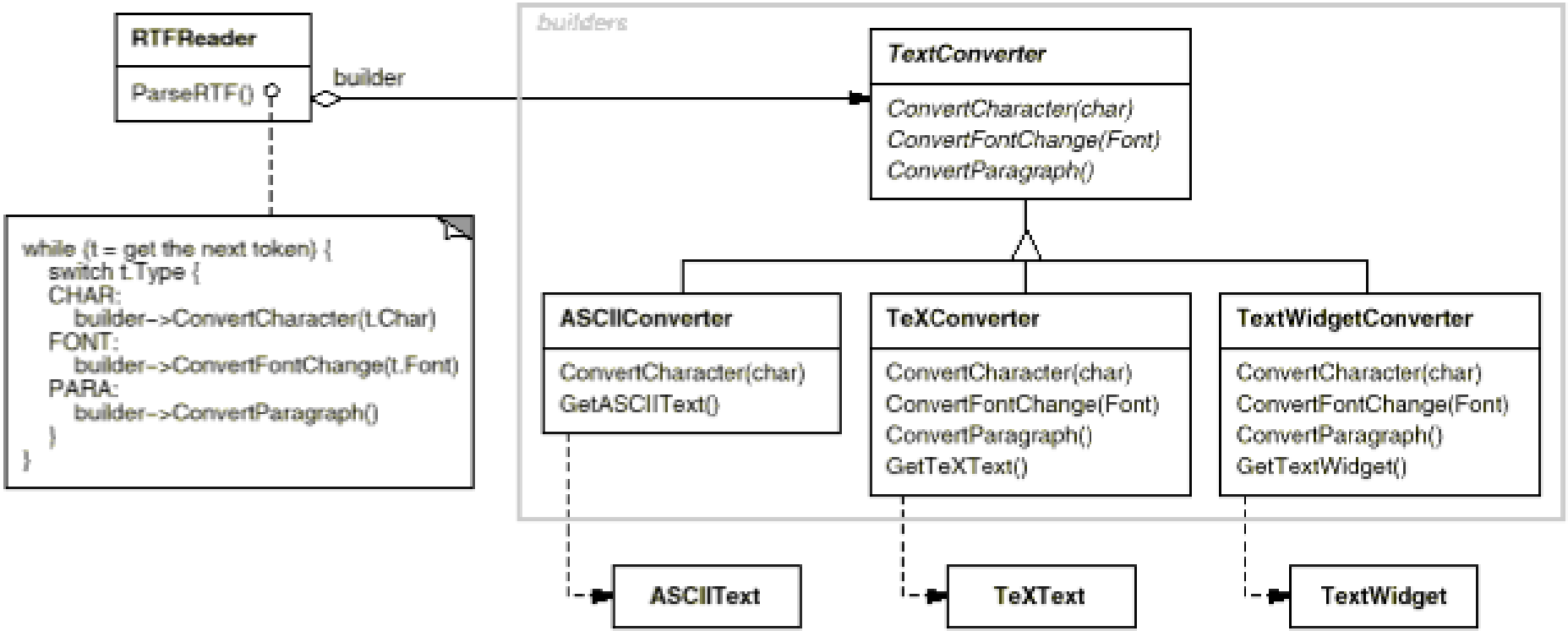
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- Esto permite reutilizar el algoritmo de análisis RTFReader.
- Se crean diferentes representaciones del texto a partir del documento RTF.
- Esto se logra configurando diferentes subclases de TextConverter.

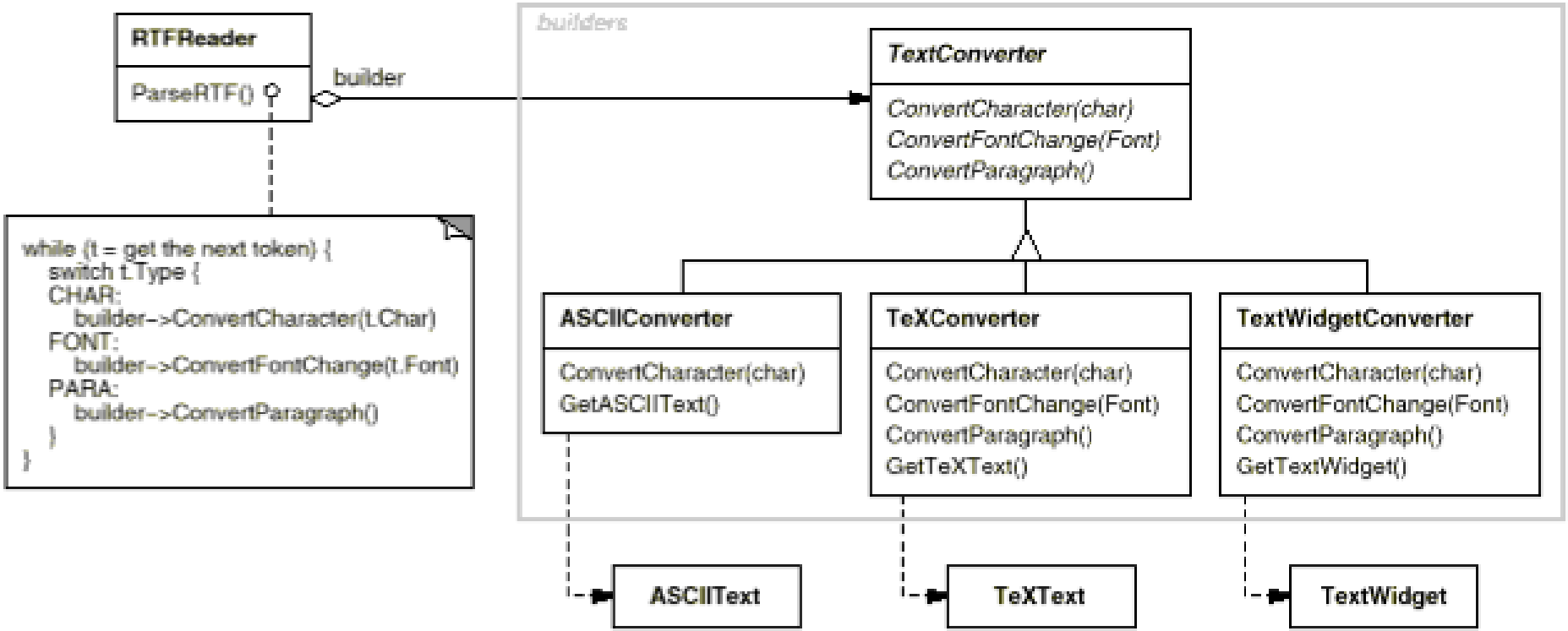
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- Este patrón es usado cuando el algoritmo para crear objetos complejos puede ser independiente.
- Independiente de las partes que hacen al objeto.
- Independiente de como ellas son ensambladas.

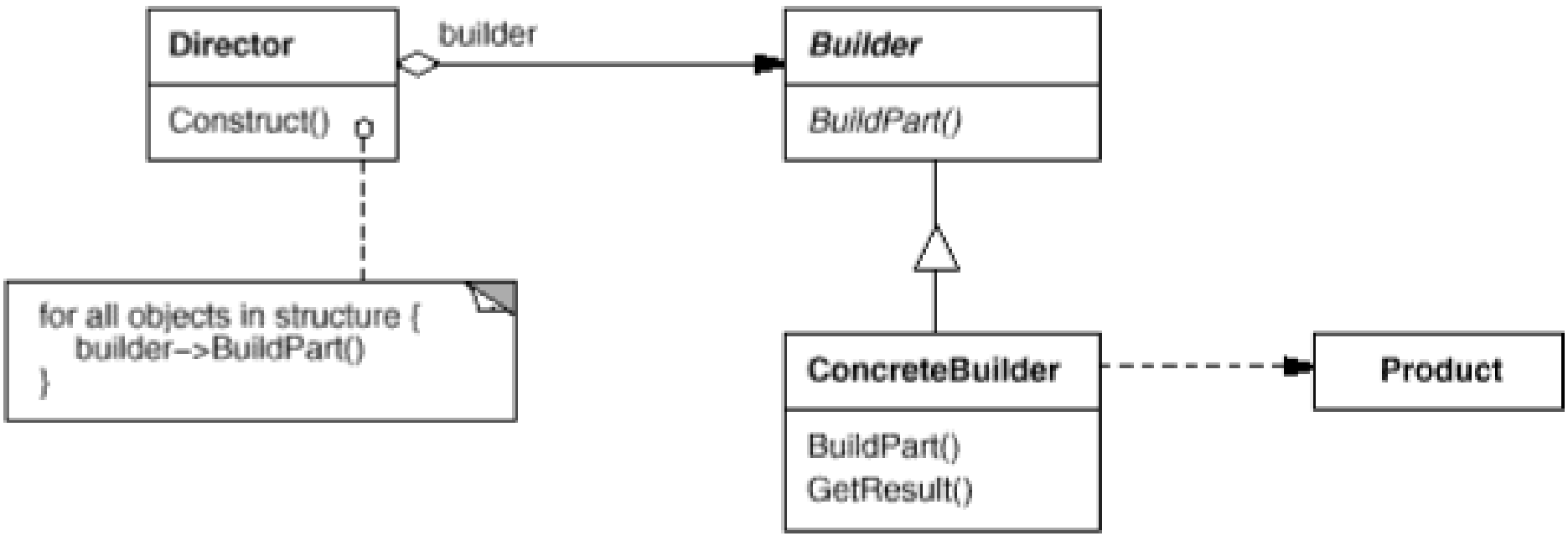
Builder


RTFReader
ParseRTF()
builder
while (t = get the next token) {
switch t.Type {
CHAR:
builder->ConvertCharacter(t.Char)
FONT:
builder->ConvertFontChange(t.Font)
PARA:
builder->ConvertParagraph()
}
}
builders
TextConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
ASCIIConverter
ConvertCharacter(char)
GetASCIIText()
TeXConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTeXText()
TextWidgetConverter
ConvertCharacter(char)
ConvertFontChange(Font)
ConvertParagraph()
GetTextWidget()
ASCIIText
TeXText
TextWidget

Builder

- También es usado para la construcción de objetos permitiendo diferentes representaciones
- Esta representaciones son de objetos que son construidos en el proceso.

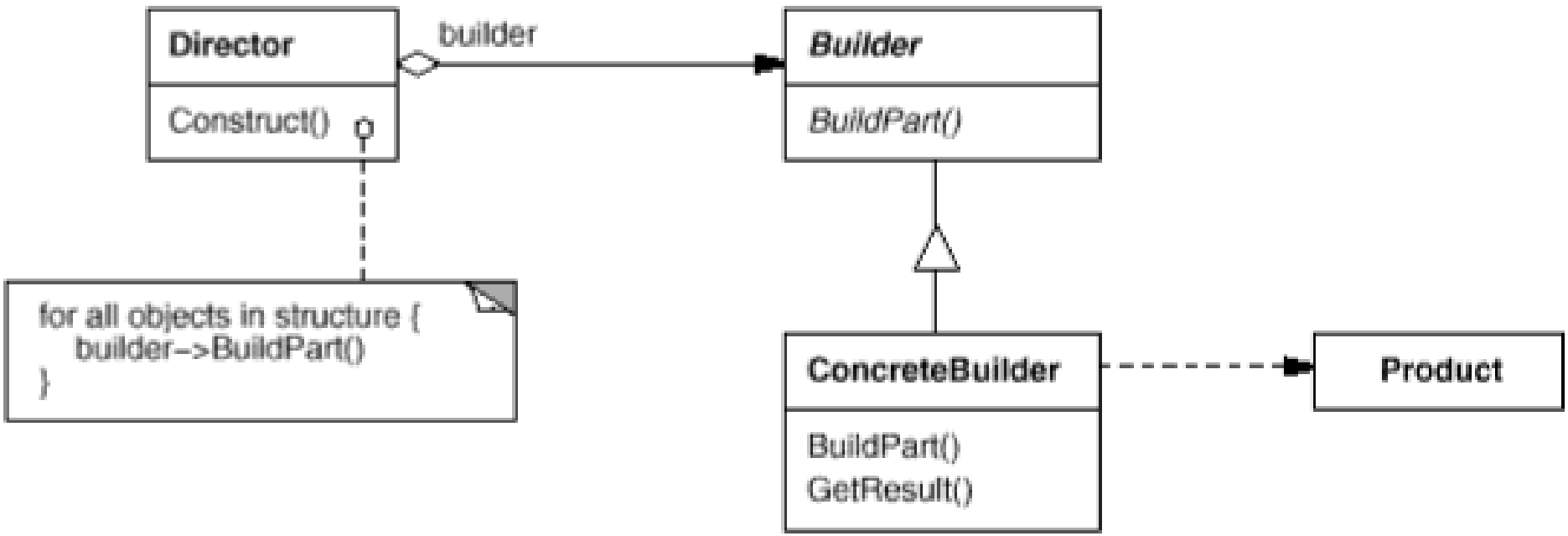
Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Builder

- La clase Builder que en nuestro ejemplo s TextConverter.
- Especifica una interface abstracta para crear partes de un objeto Product.

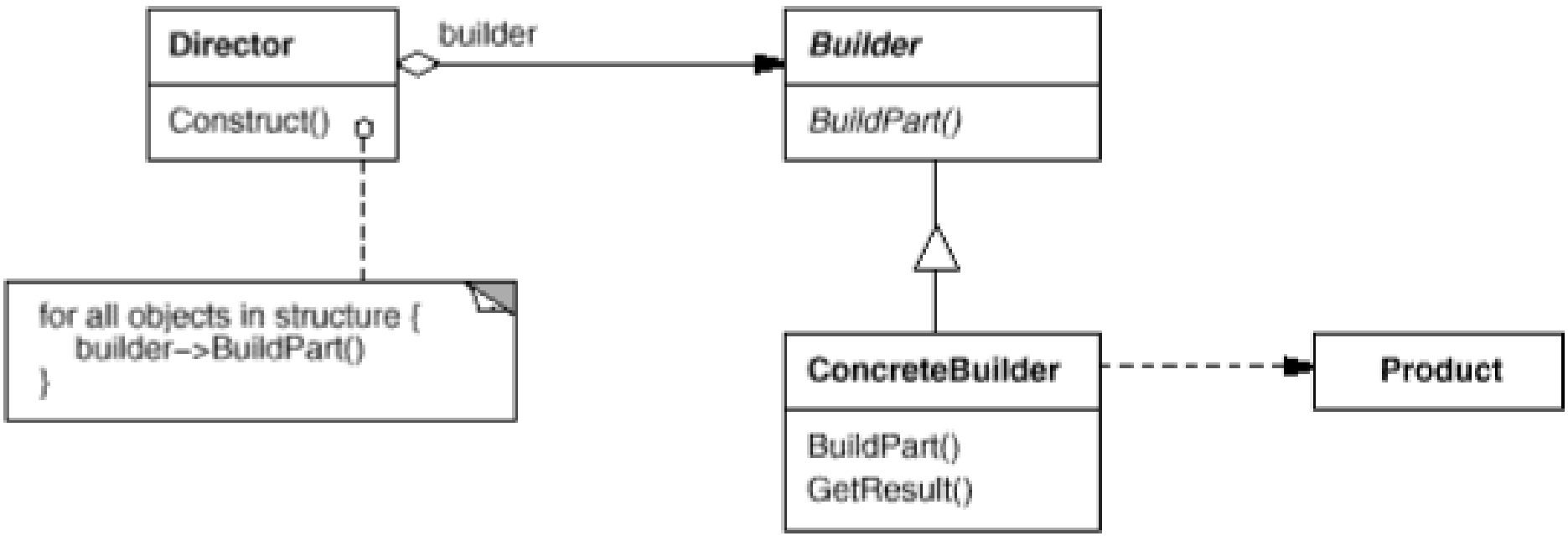
Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Builder

- La clase ConcreteBuilder que en nuestro ejemplo es el ASCIIConverter, el TextConverter y el TextWidgetConverter.
- Construye las partes del Product para implementar la interface Builder y después la ensambla.
- Define la representación creada y sigue dicha representación.
- Genera una interfaz para recuperar Prodcut, en nuestro caso GetASCIIText y GetTextWidget.

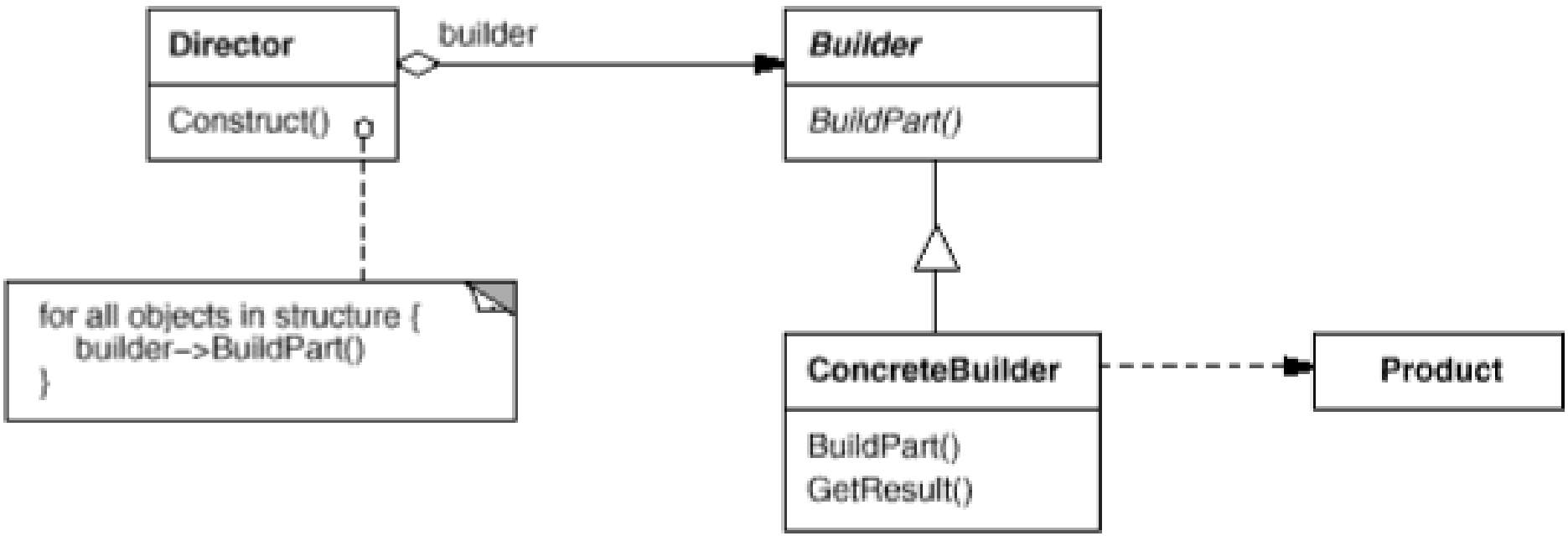
Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Builder

- La clase Director que en nuestro ejemplo es la clase RTFReader
- Construye un objeto utilizando la interface Builder

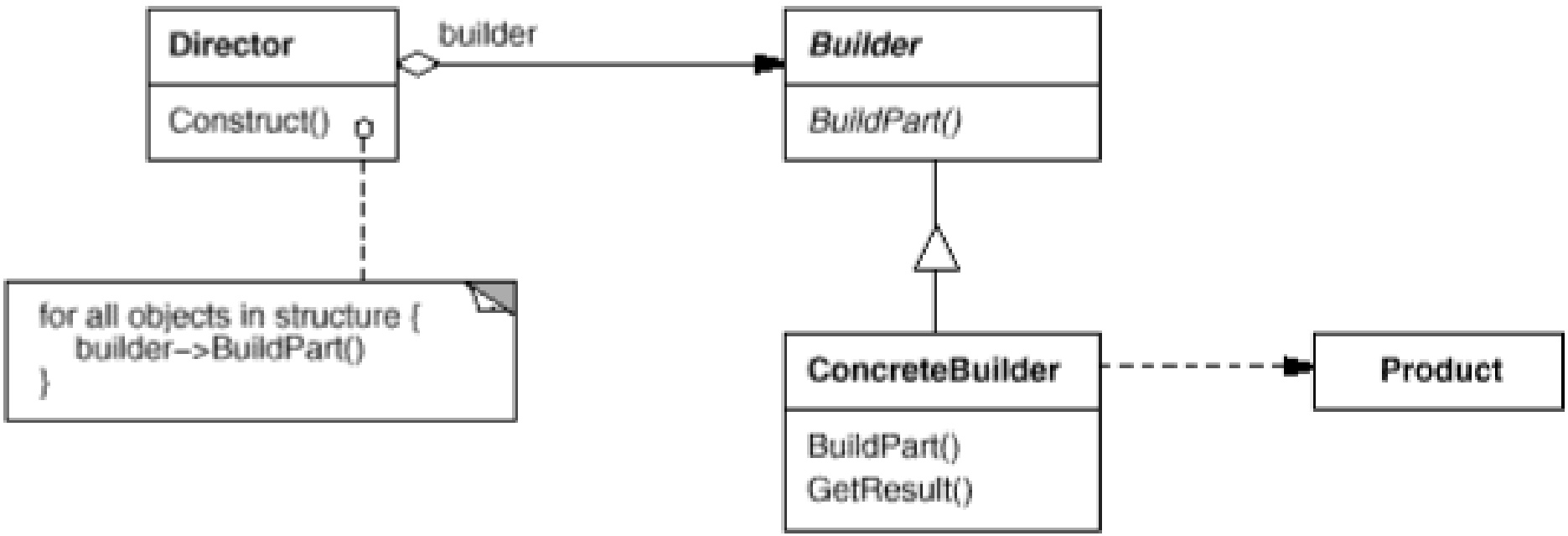
Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Builder

- La clase Product que en nuestro ejemplo es ASCIIText, TeXText, TextWidget
- Representa un objeto complejo bajo construcción.
- ConcreteBuilder construye la representación interna de Product.

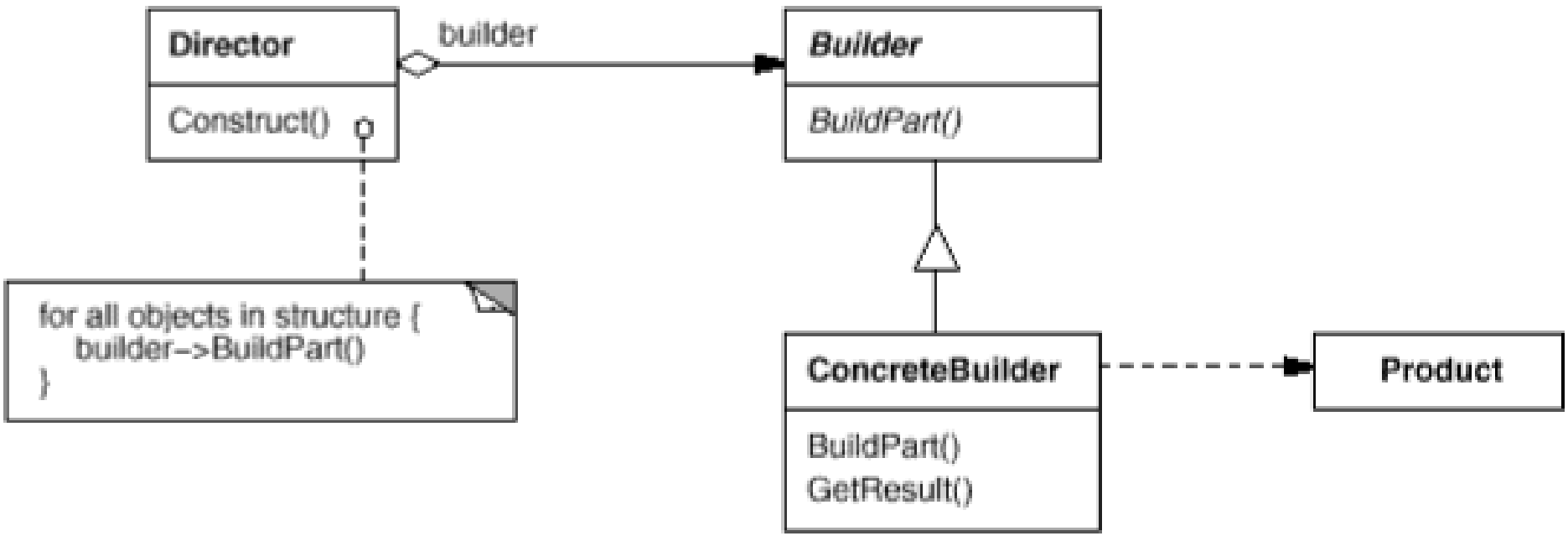
Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Builder

- ConcreteBuilder tambien define el proceso por el cual es ensamblado Product
- También incluye clases que definen las partes constituyentes.
- Esto engloba a las interfaces para ensamblar las partes del resultado final.

Builder


Director
builder
Builder
Construct()
BuildPart()
for all objects in structure {
builder->BuildPart()
}
ConcreteBuilder
Product
BuildPart()
GetResult()

Preguntas