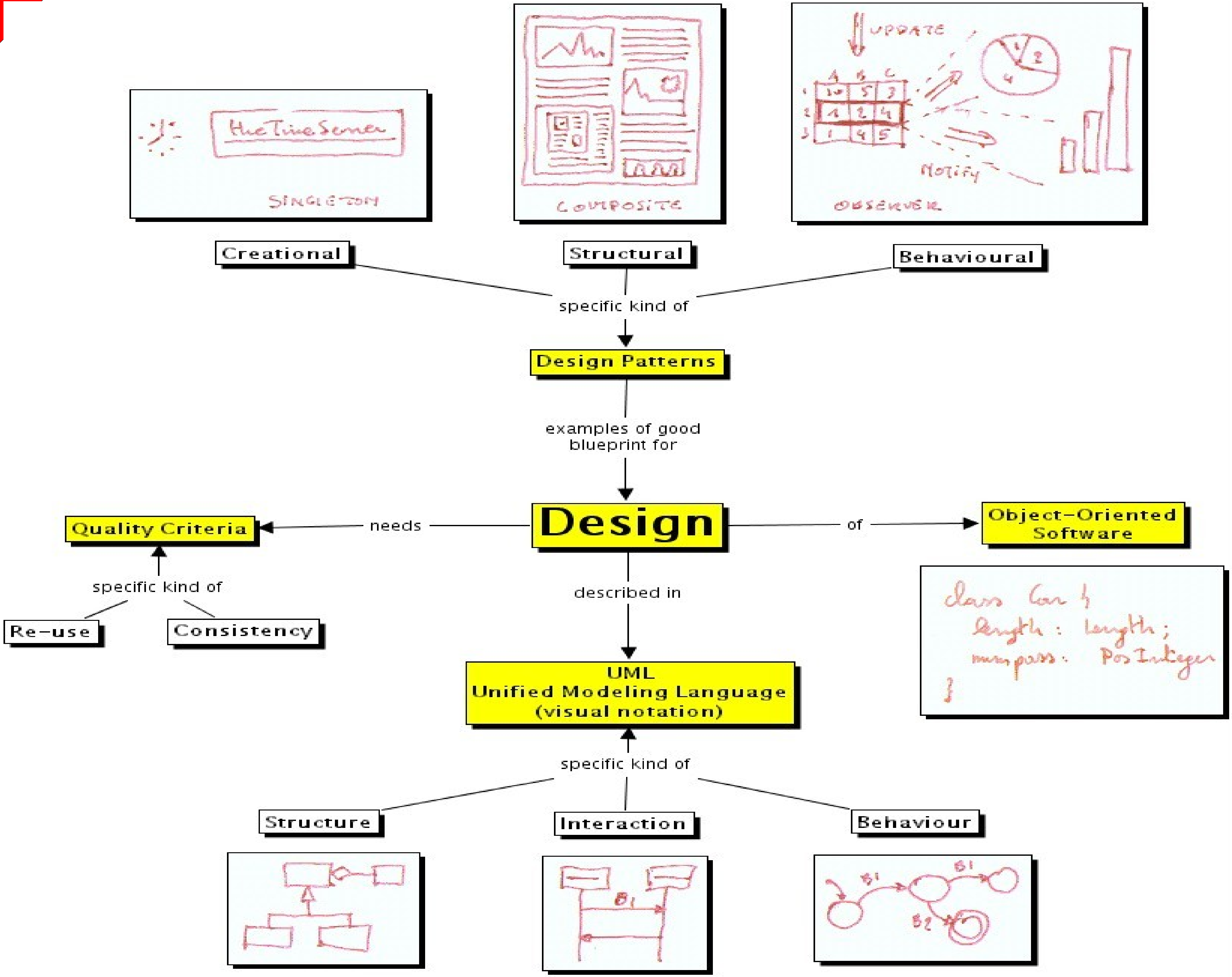

theTimeServer
SINGLETON
COMPOSITE
UPDATE
NOTIFY
OBSERVER
Creational
Structural
Behavioural
specific kind of
Design Patterns
examples of good blueprint for
Design
needs
Quality Criteria
specific kind of
Re-use
Consistency
of
Object-Oriented Software
class Car {
length : length;
num pass : PosInteger
}
described in
UML
Unified Modeling Language
(visual notation)
specific kind of
Structure
Interaction
Behaviour
B1
B1
B1
B2

Only connect ... (Ch2 of GoF)

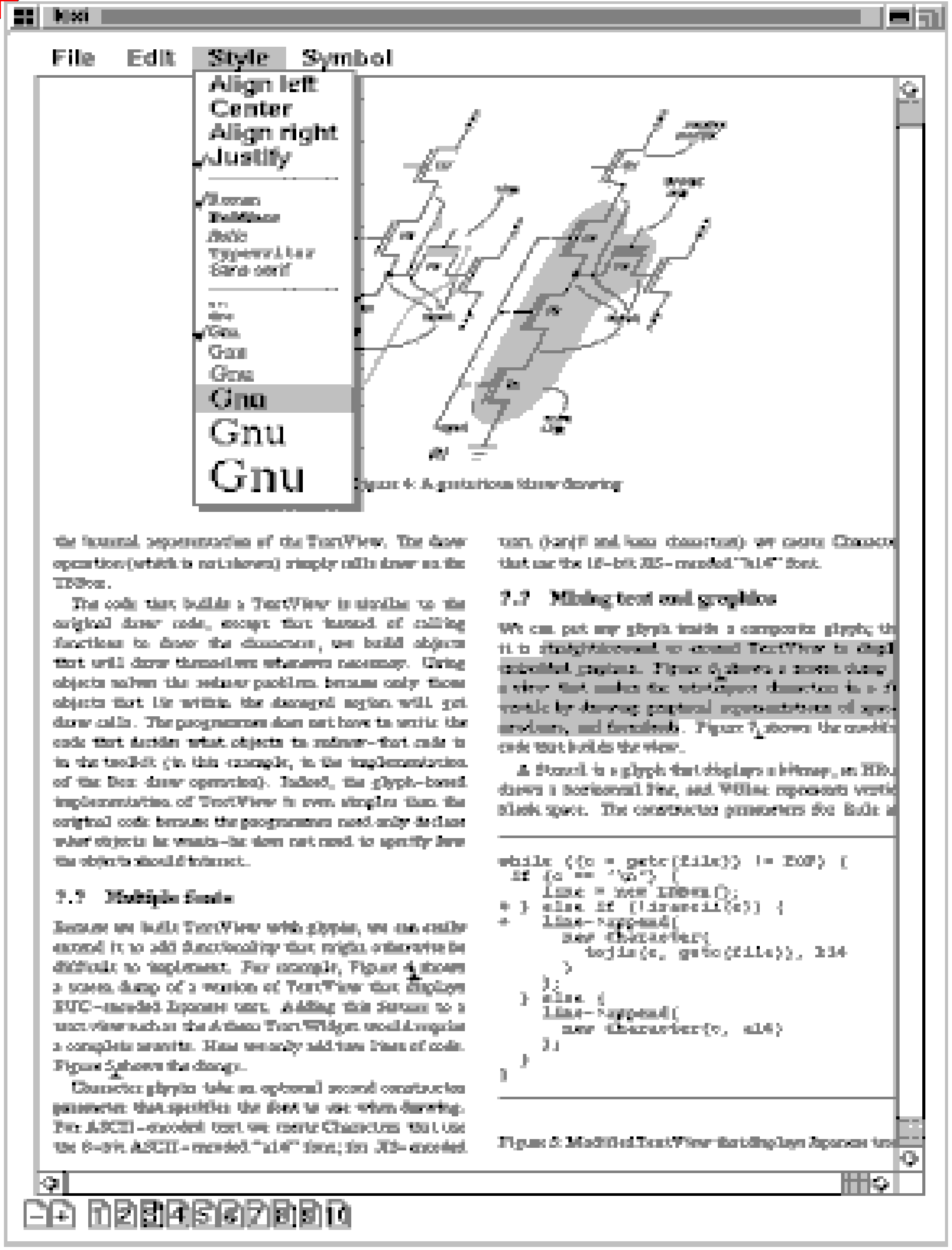


Figure 4: A gratuitous Idraw drawing

[illegible]

7.7 Mixing text and graphics

[illegible]

7.5 Multiple fonts

[illegible]

```
while ((c = getc(file)) != EOF) {
  if (c == '\n') {
    line = new LRBox();
  } else if (!isascii(c)) {
    line->append(
      new Character(
        tojis(c, getc(file)), k14
      )
    );
  } else {
    line->append(
      new Character(c, a14)
    );
  }
}
```

Figure 5: Modified TextView that displays Japanese text

# Composite:
represent document structure,

# Strategy:
different formatting algorithms

# Decorator:
embellishing the user interface

# Abstract Factory:
multiple look-and-feel standards

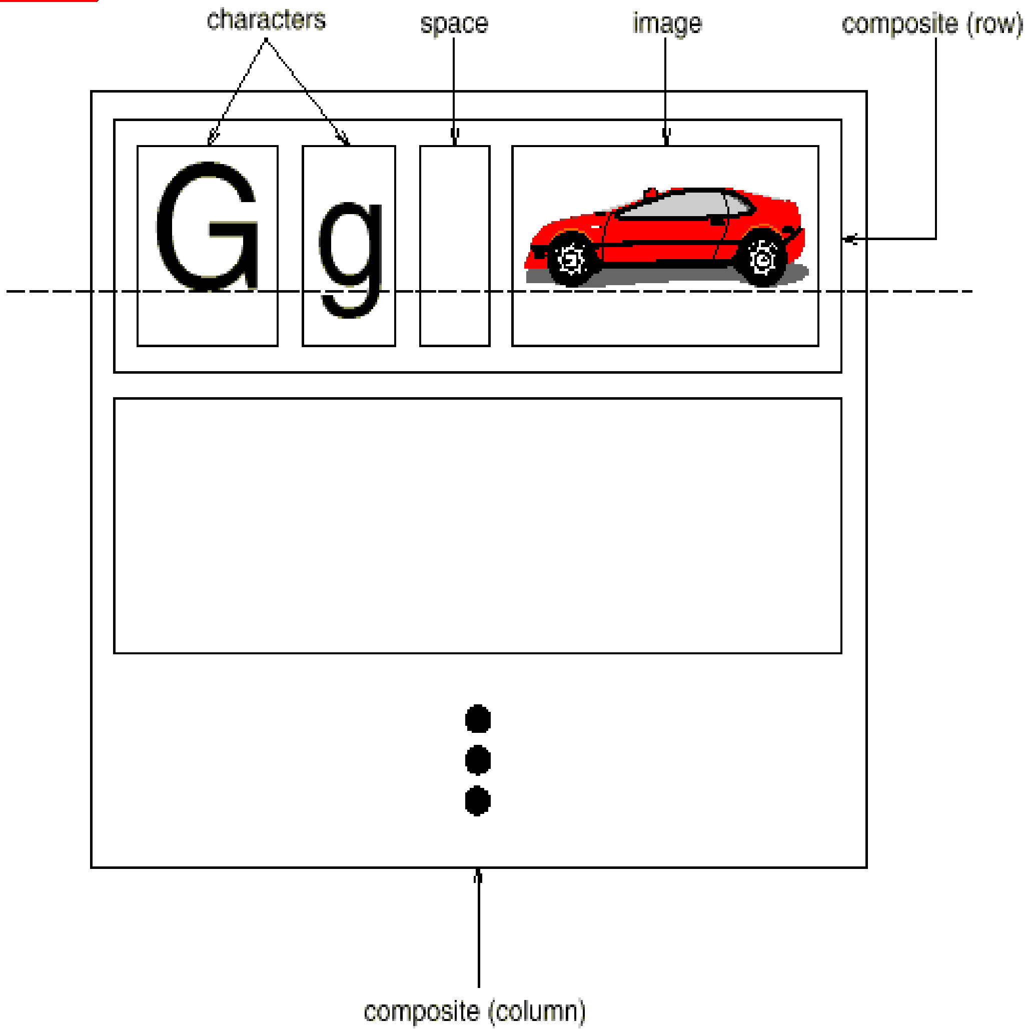
# Bridge:
multiple windowing platforms

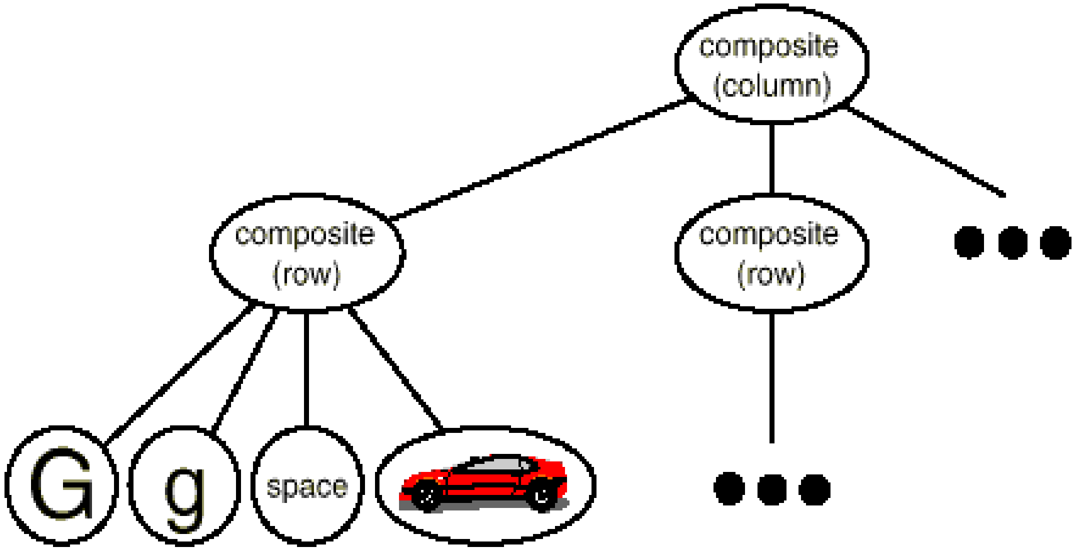
# Command:
undo-able user operations

# Iterator:
traversing object structures

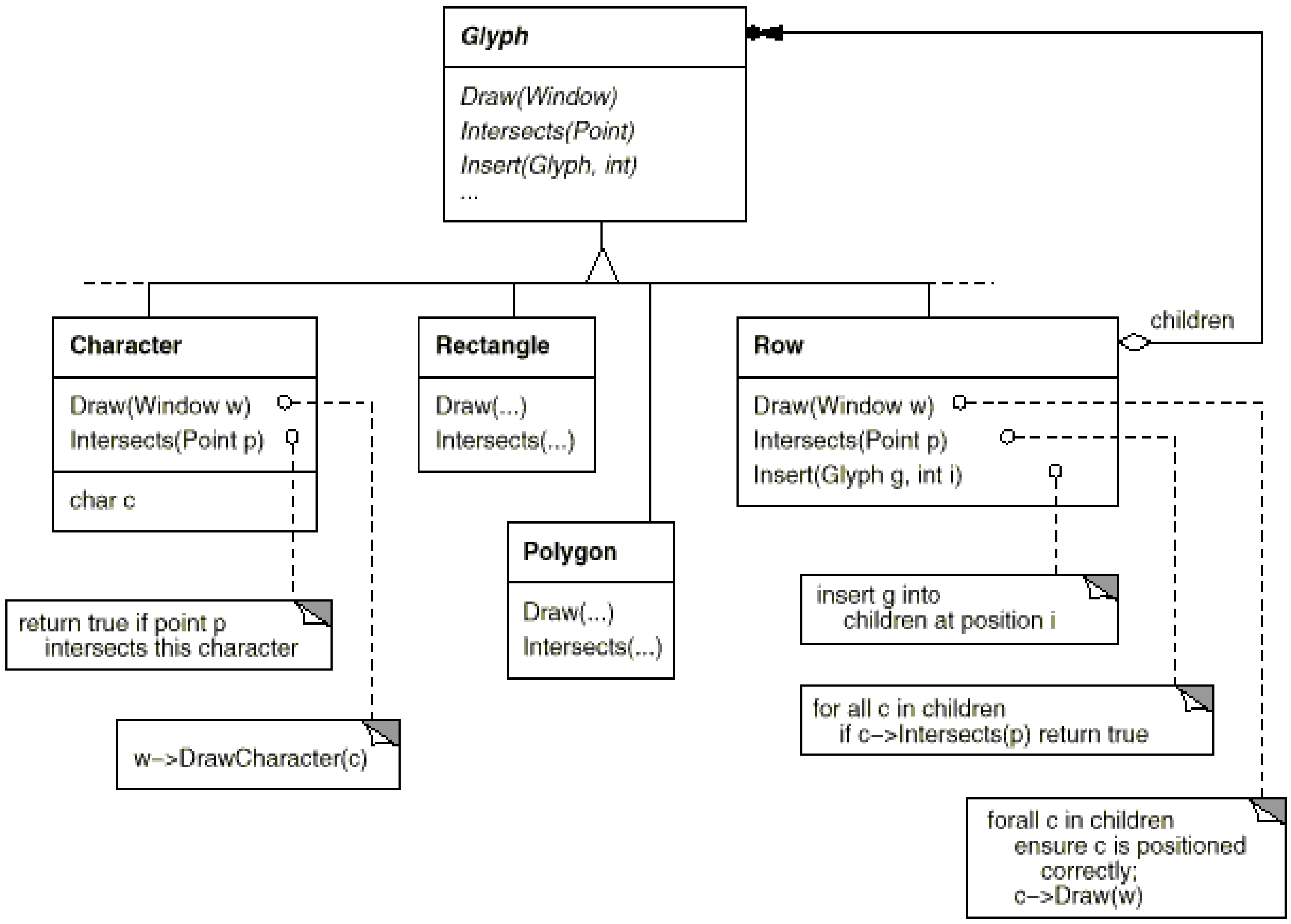
# Visitor:
add functionality independent of document's structure

Composite Pattern





Composite Pattern: Glyphs

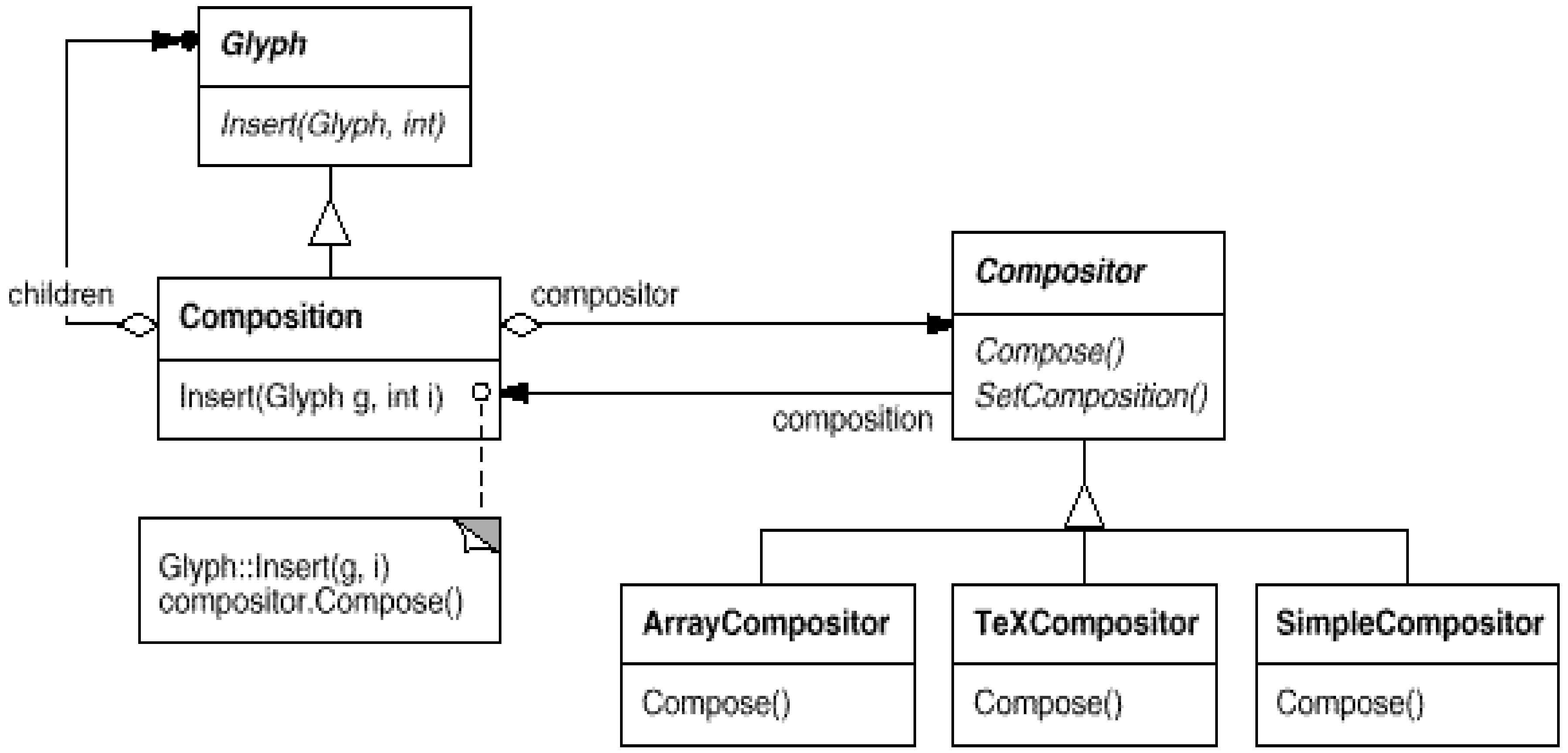


Strategy Pattern (aka Policy)

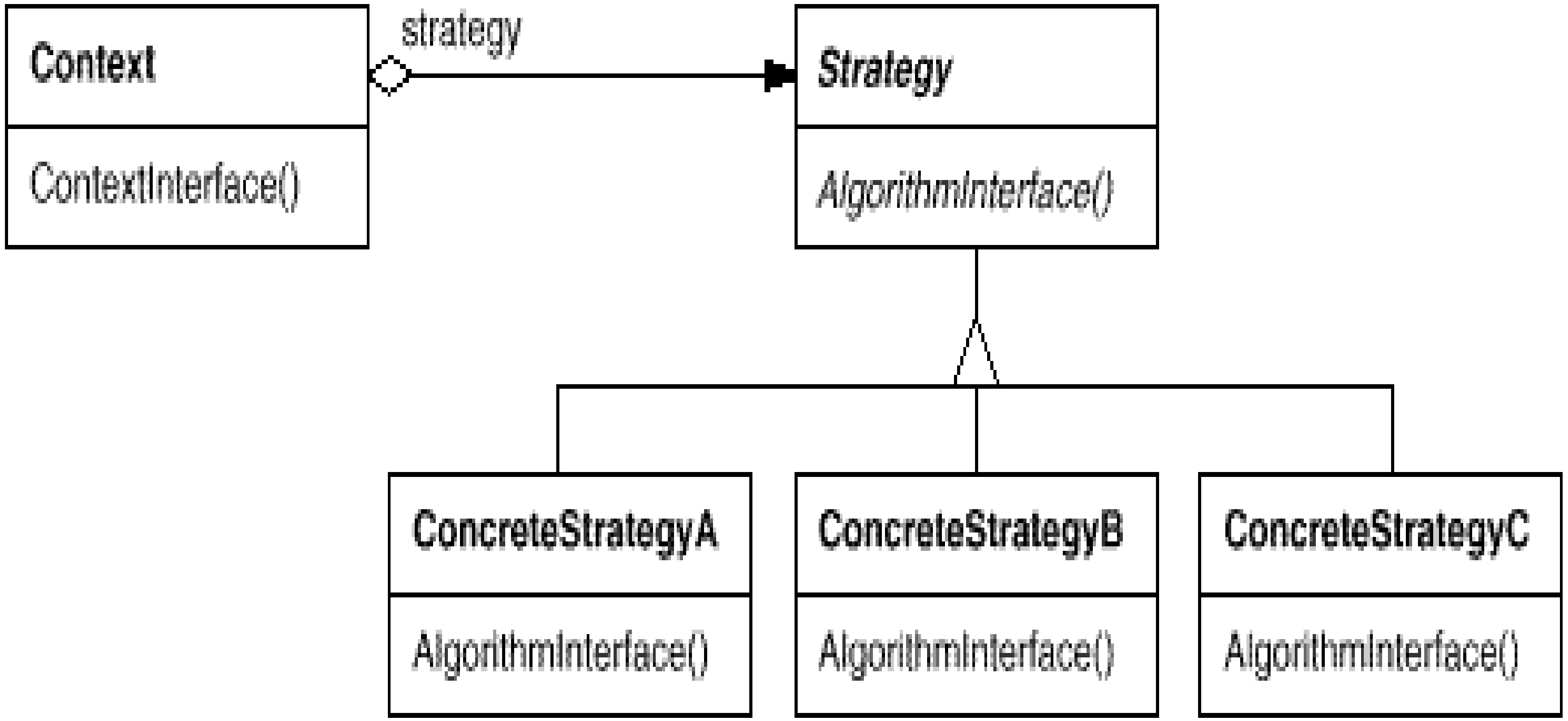
Define a family of algorithms, encapsulate each one,
and make them interchangeable.

Strategy lets the algorithm vary independently from clients that use it.

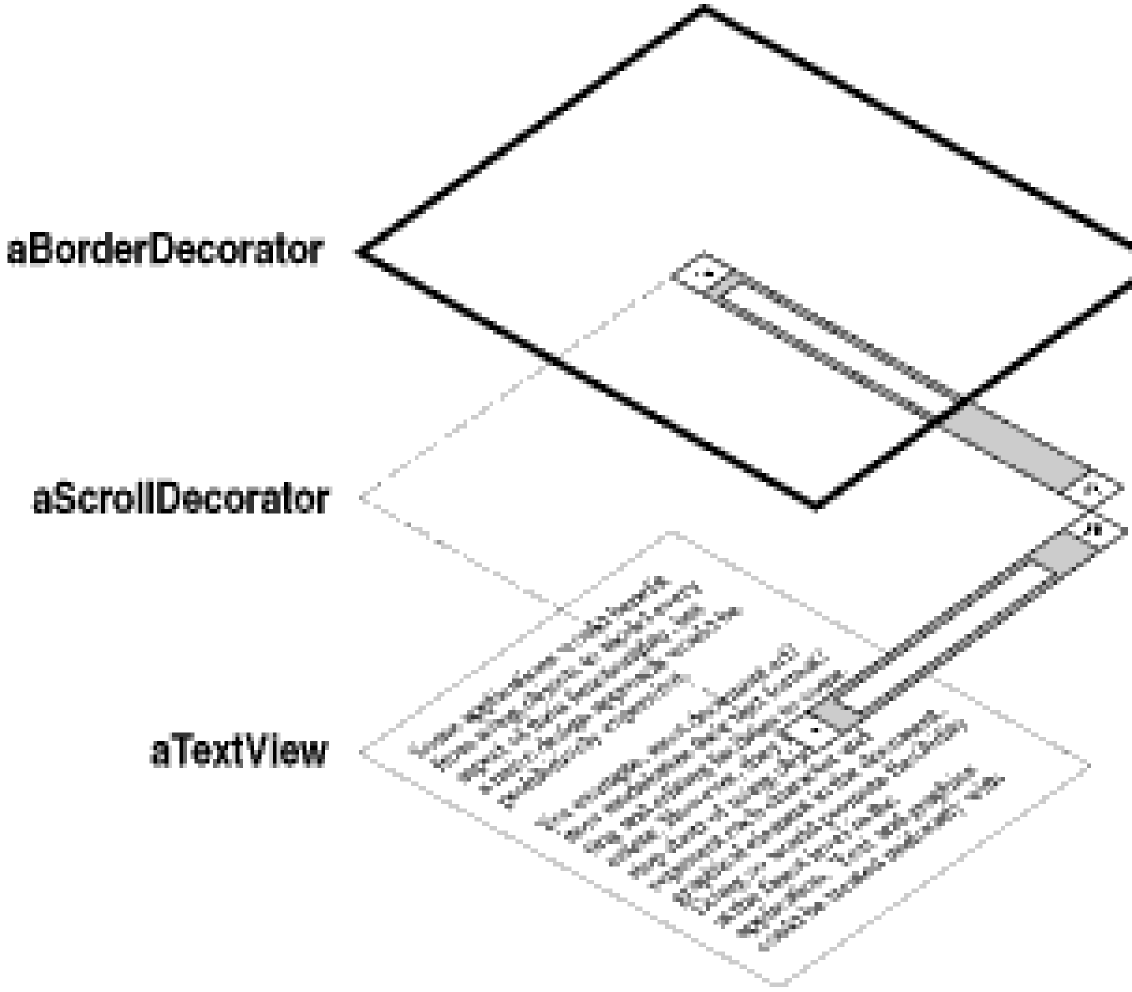
Example: line breaking

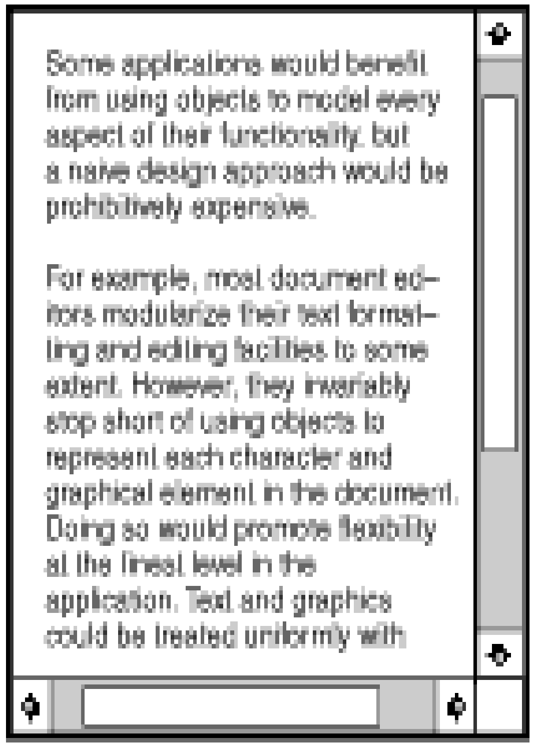


Strategy Pattern



Embellishing: Decorator Pattern





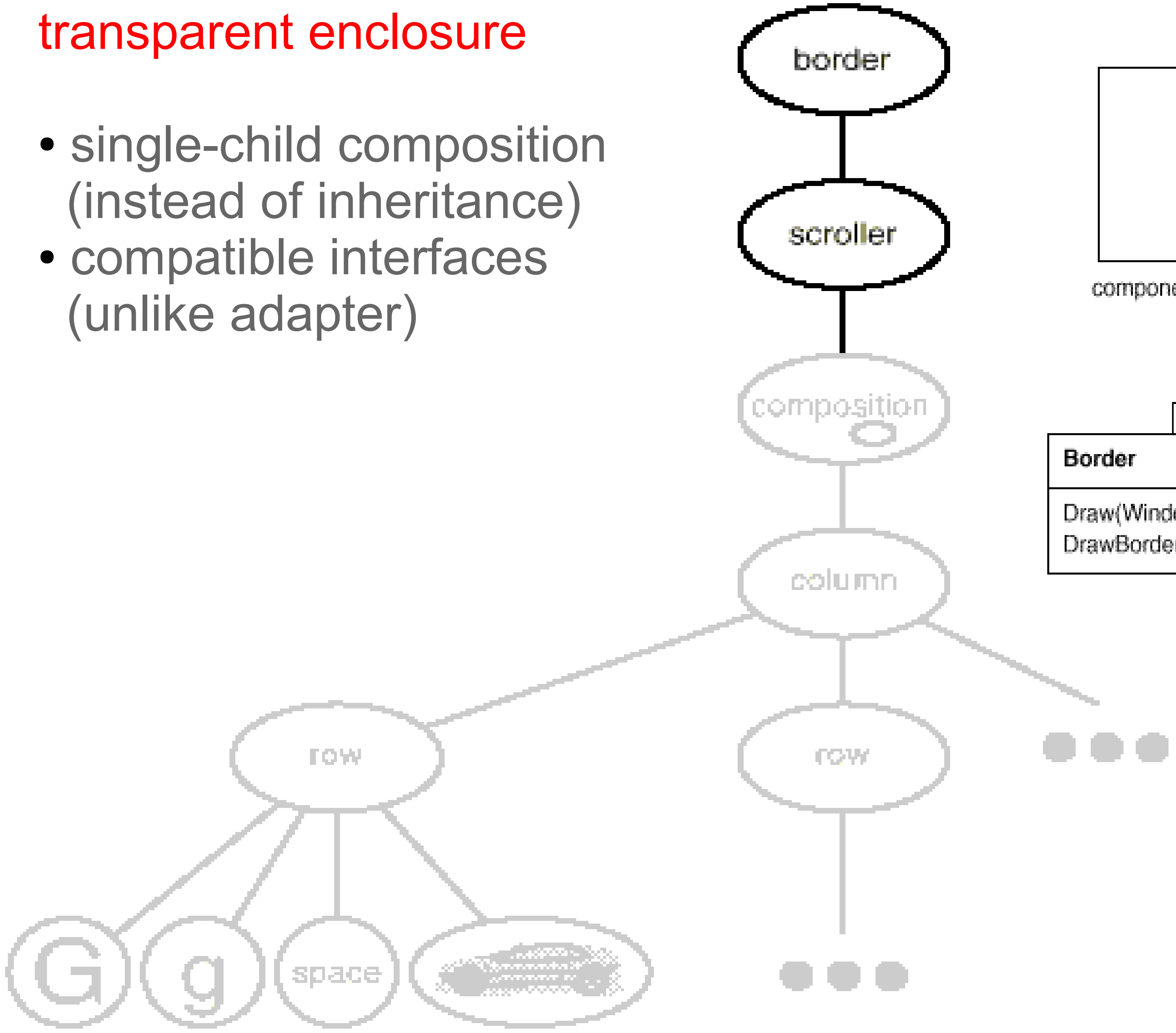
Some applications would benefit
from using objects to model every
aspect of their functionality, but
a naive design approach would be
prohibitively expensive.

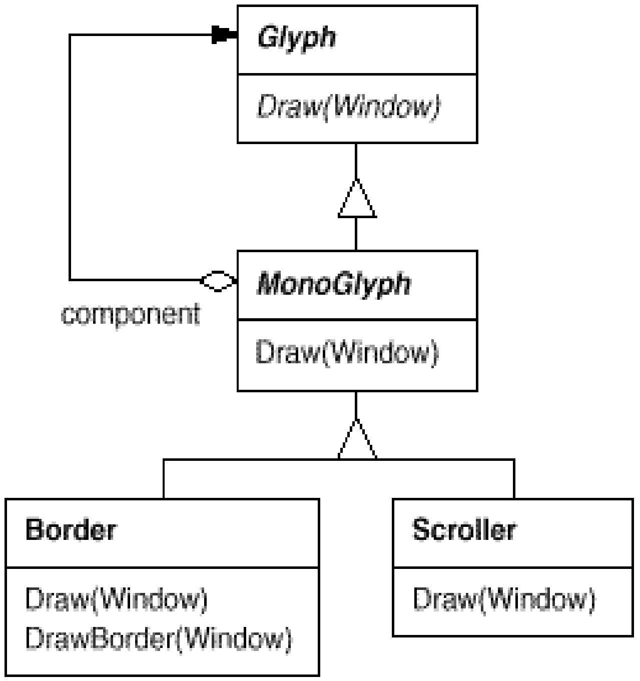
For example, most document ed–
itors modularize their text format–
ting and editing facilities to some
extent. However, they invariably
stop short of using objects to
represent each character and
graphical element in the document.
Doing so would promote flexibility
at the finest level in the
application. Text and graphics
could be treated uniformly with

Decorator Pattern

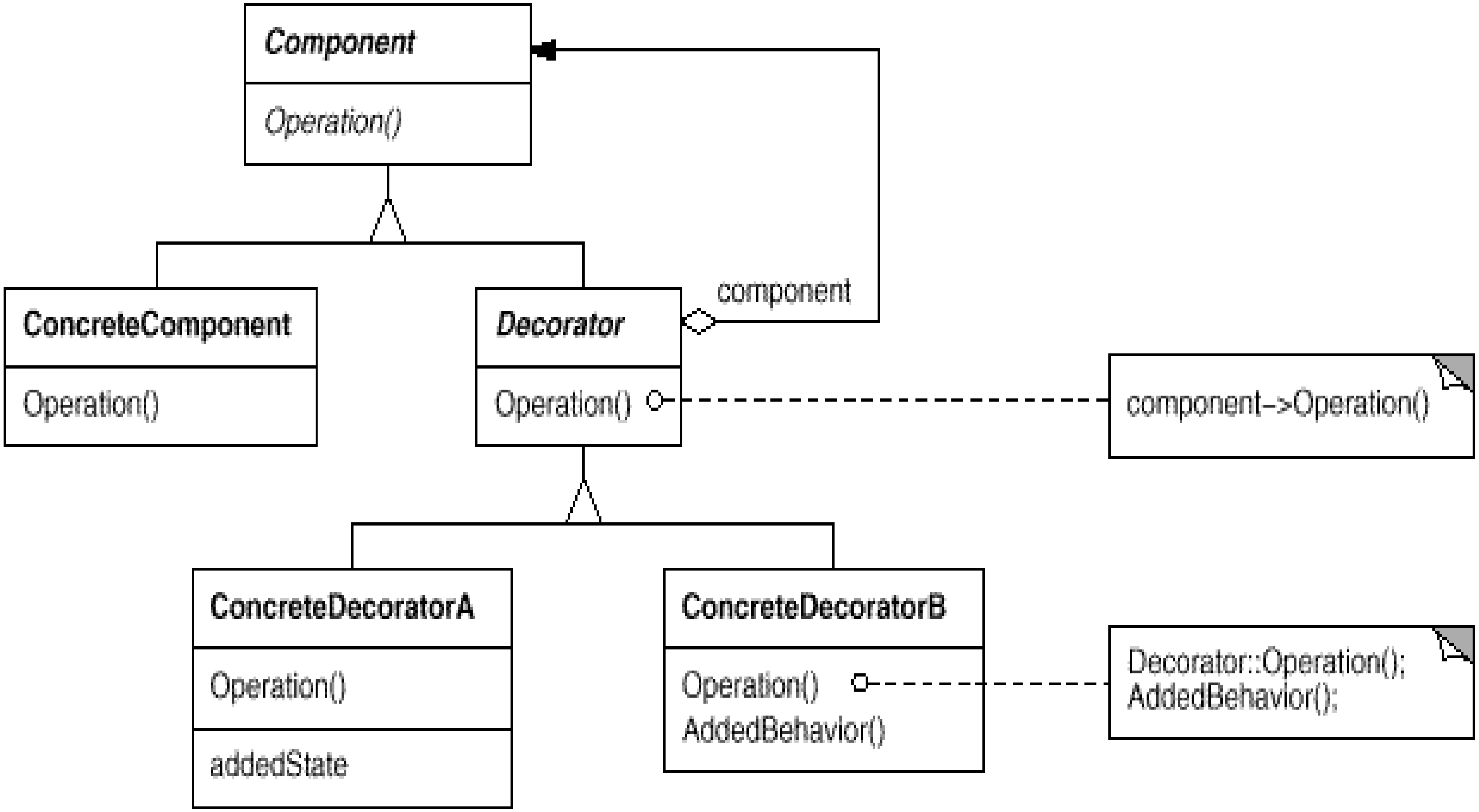
transparent enclosure

- single-child composition (instead of inheritance)
- compatible interfaces (unlike adapter)

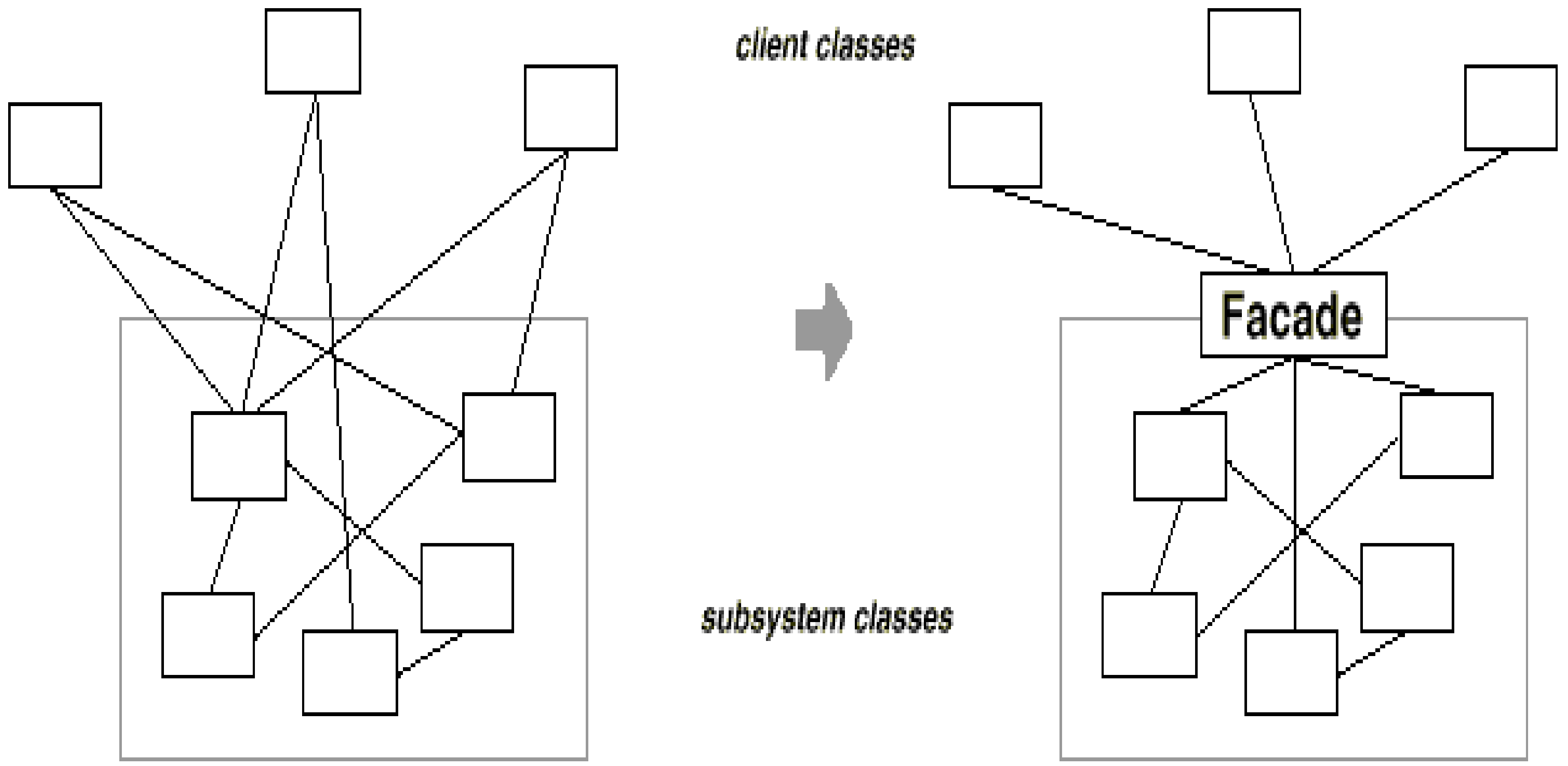




Decorator Pattern

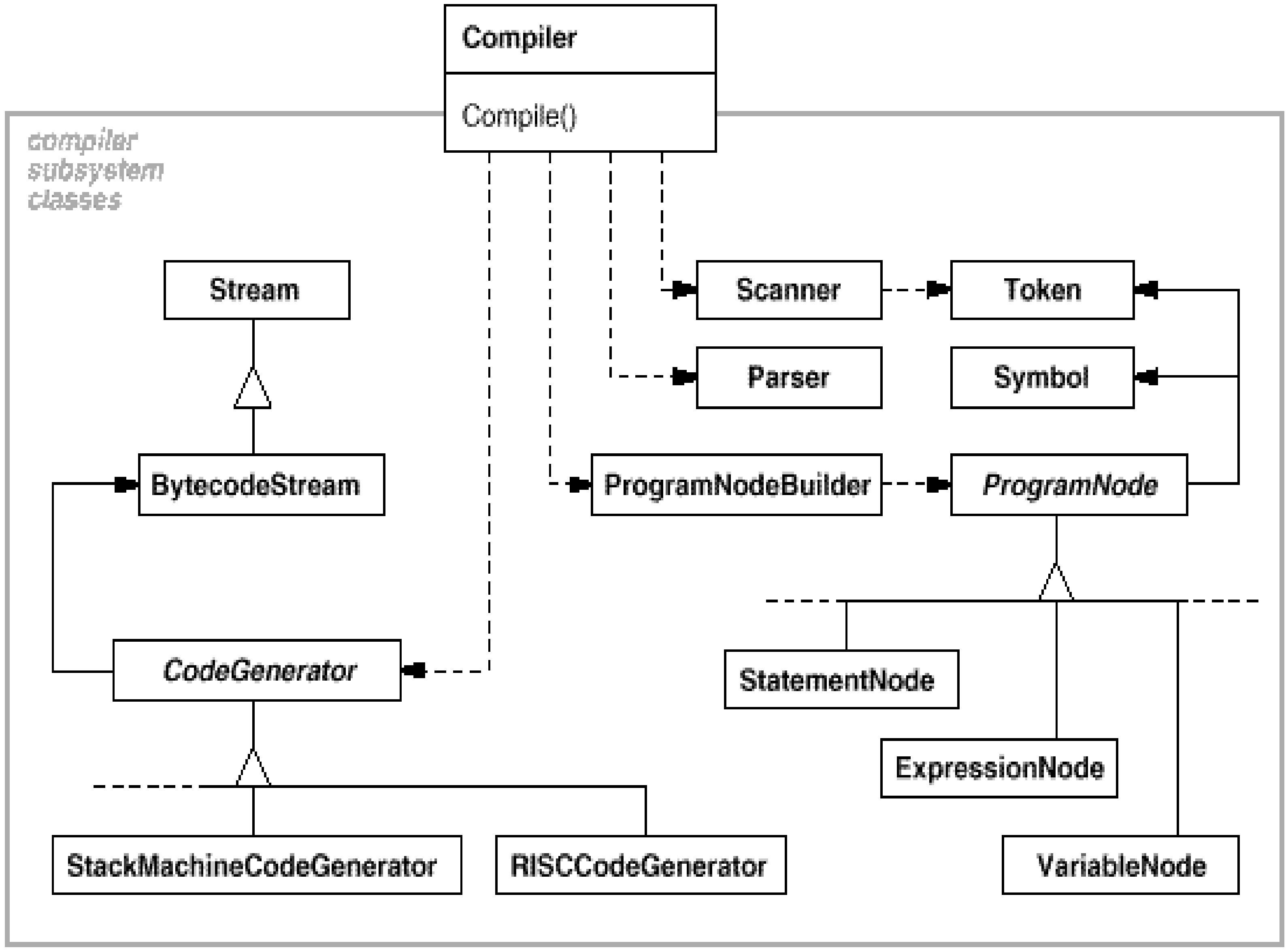


Related to … Facade Pattern

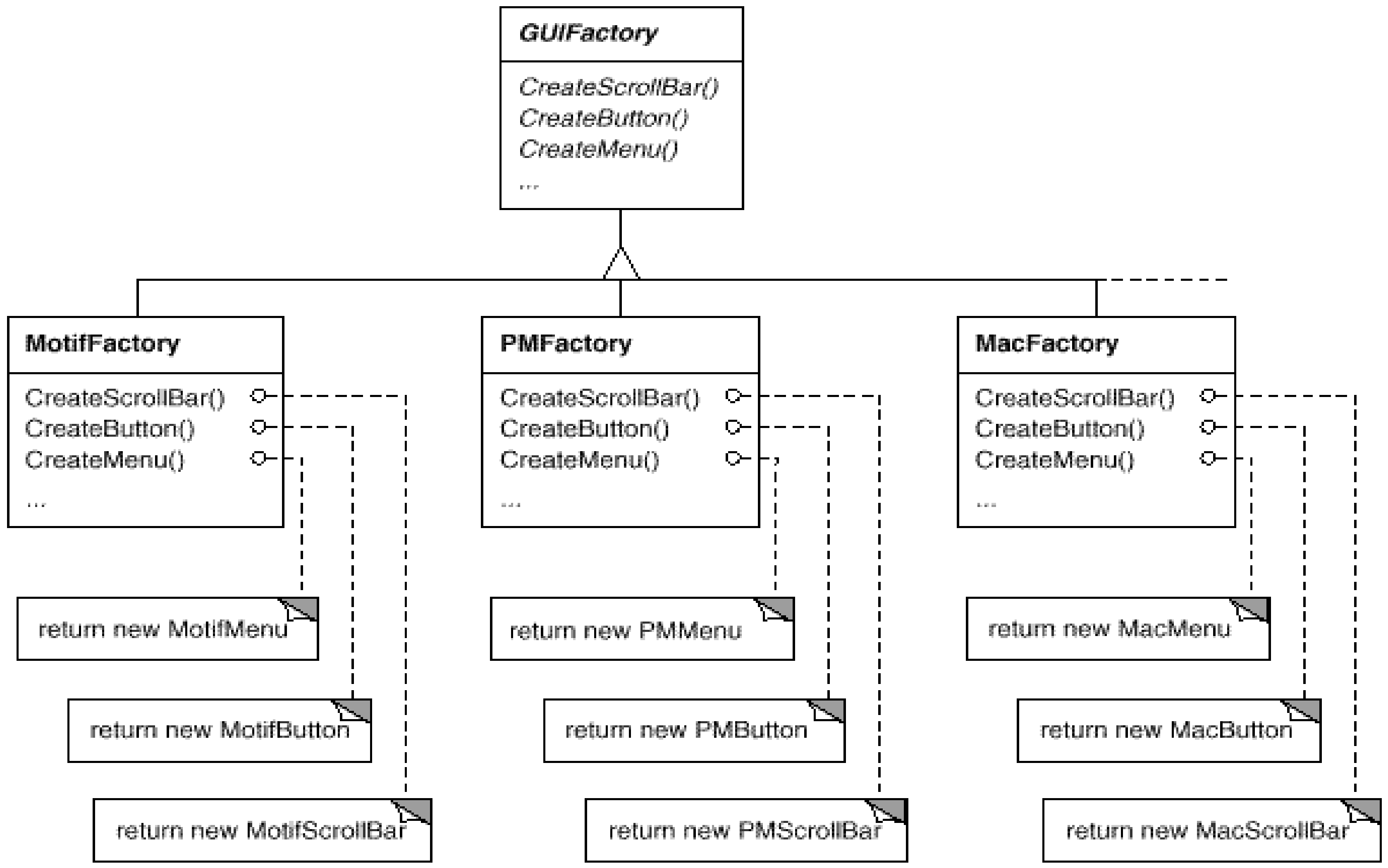


Abstraction to minimize communication/dependency

Facade Pattern example

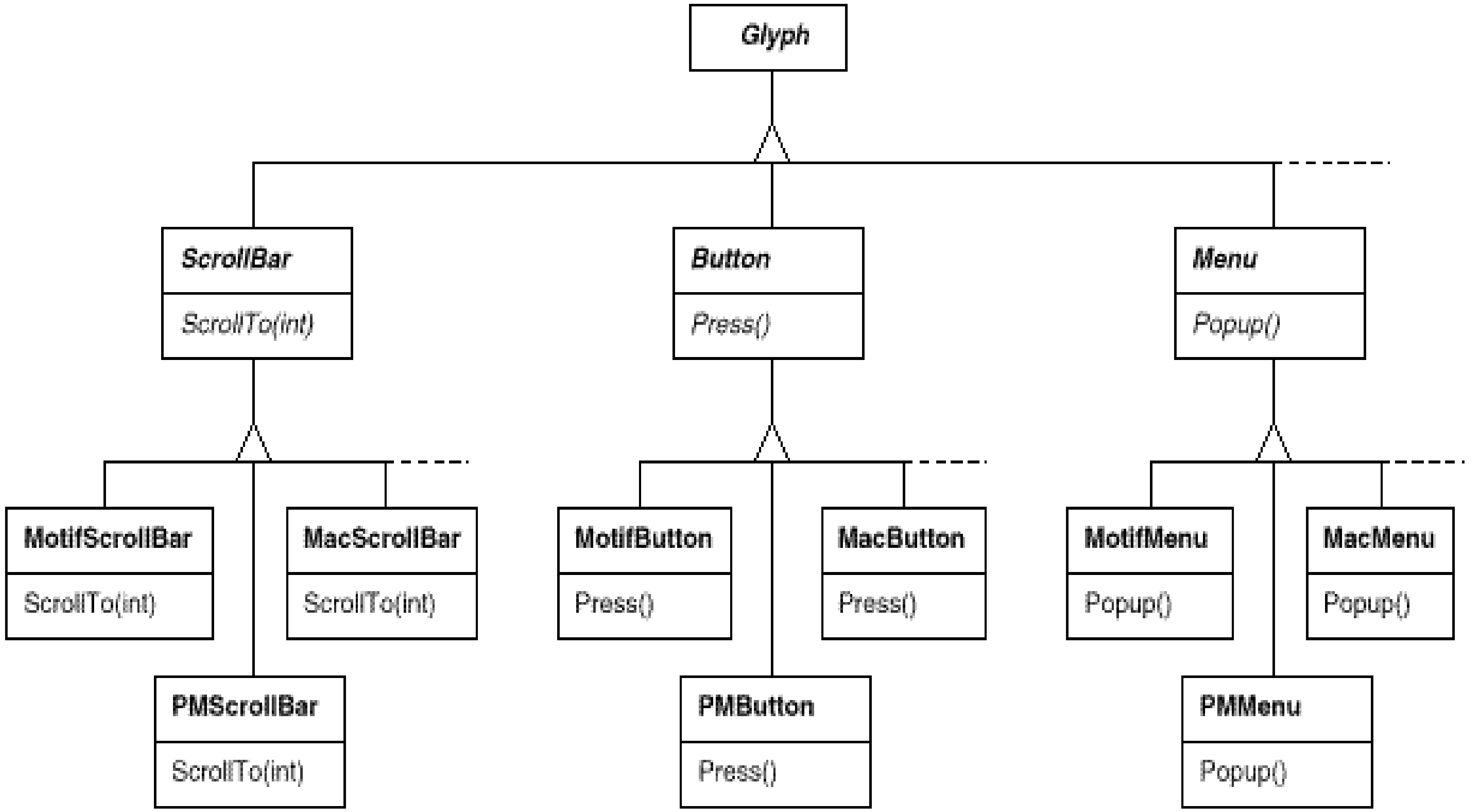


Abstract Factory Pattern

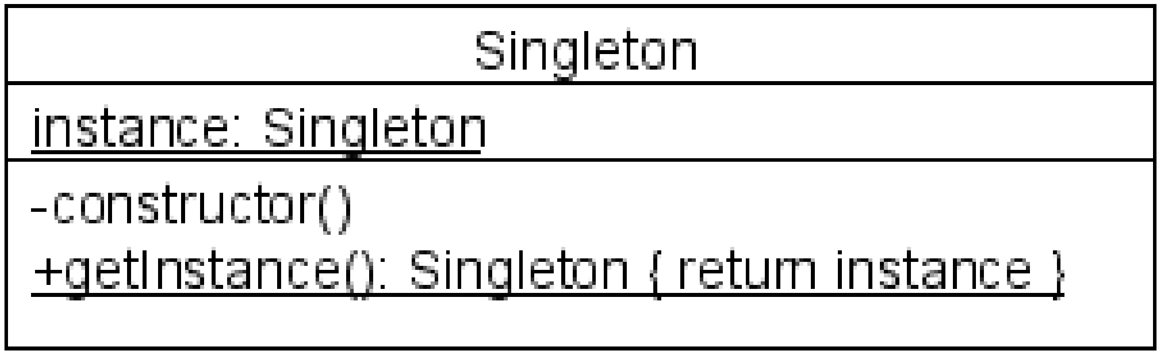


Create families of related products

Abstract Factory: products

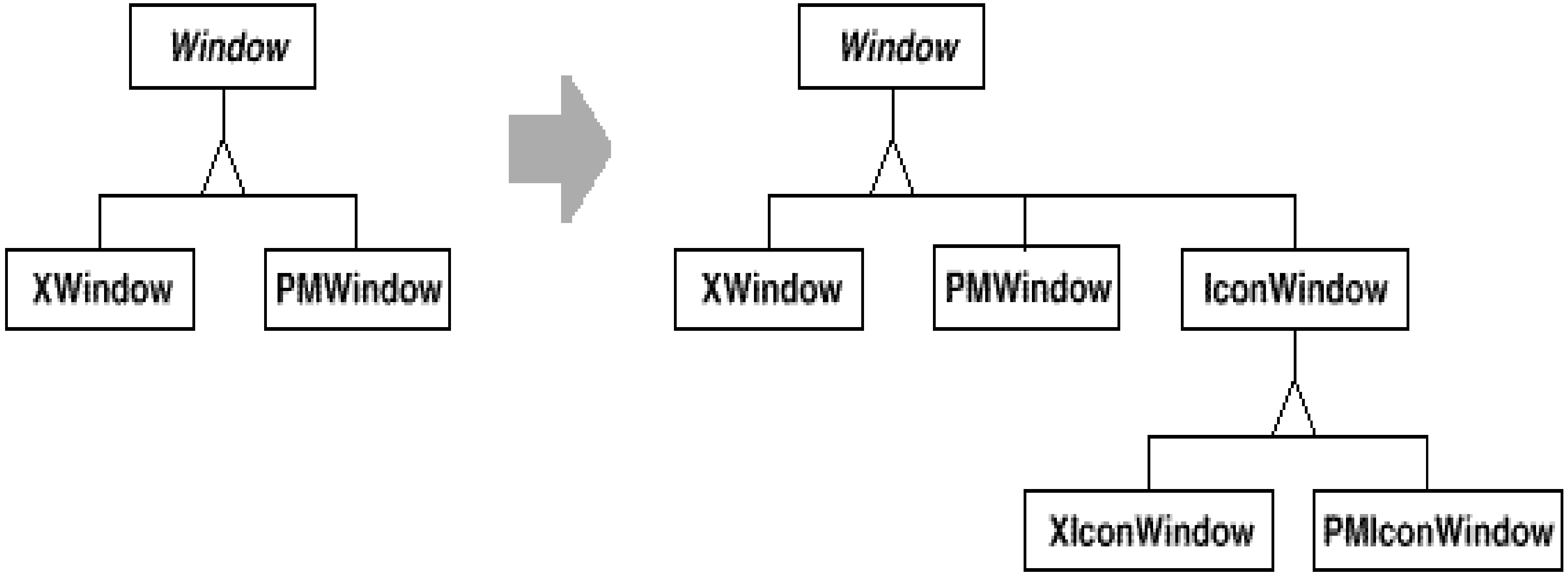


Abstract Factory
is usually a Singleton



Bridge Pattern

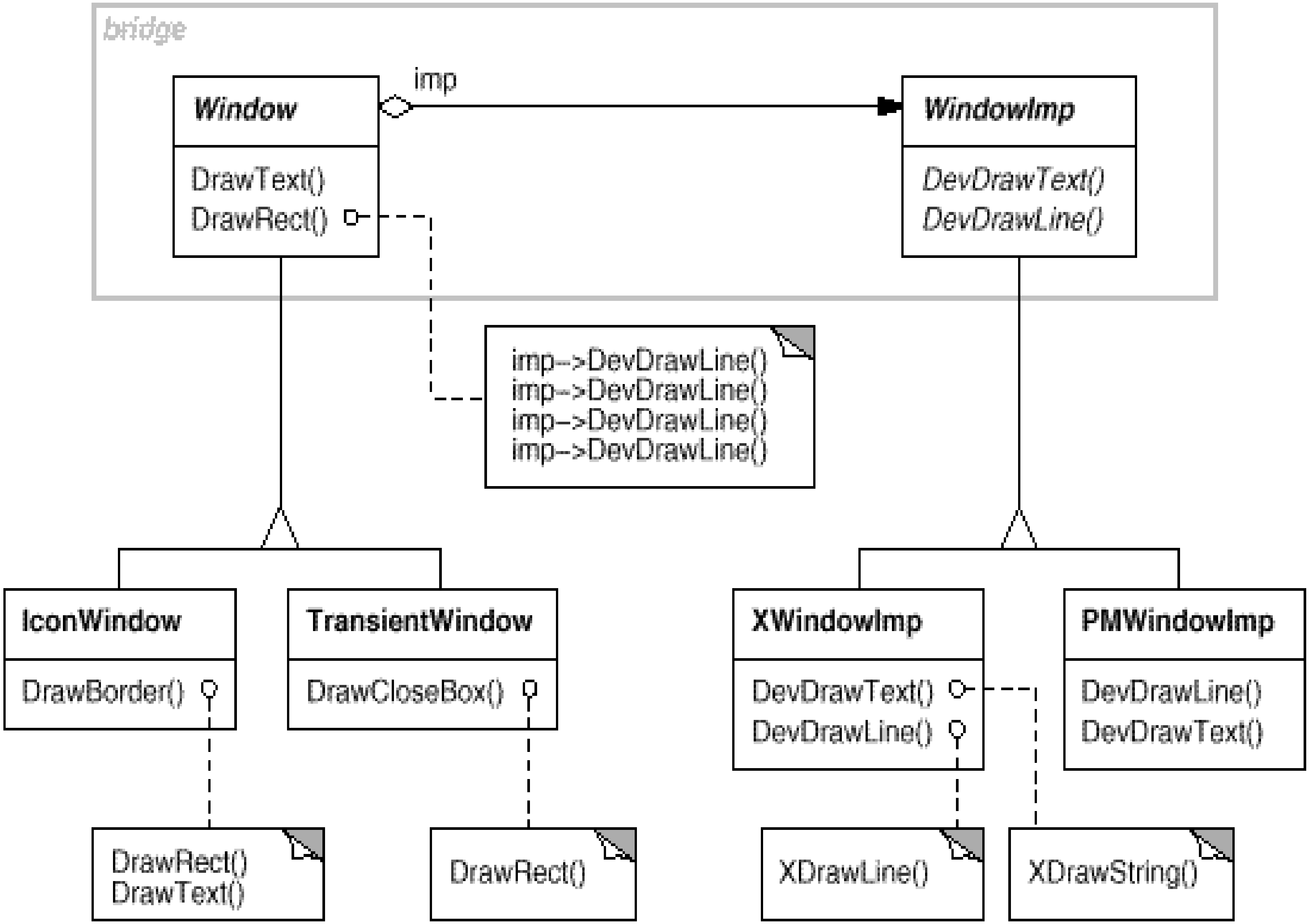
Supporting multiple UI platforms



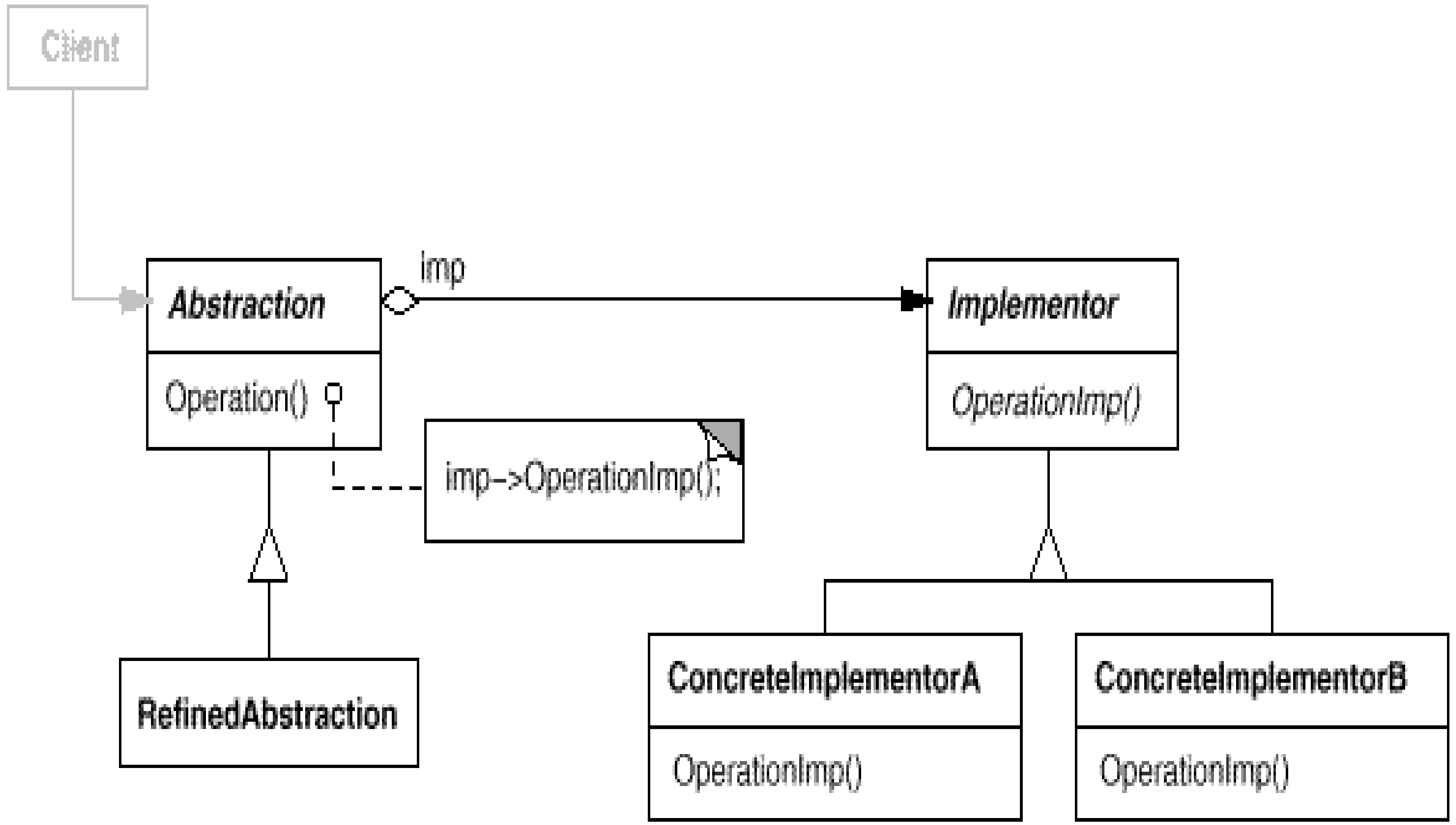
Intent: Decouple an abstraction from its implementation so that the two can vary independently (and dynamically).

Aka: Handle/Body

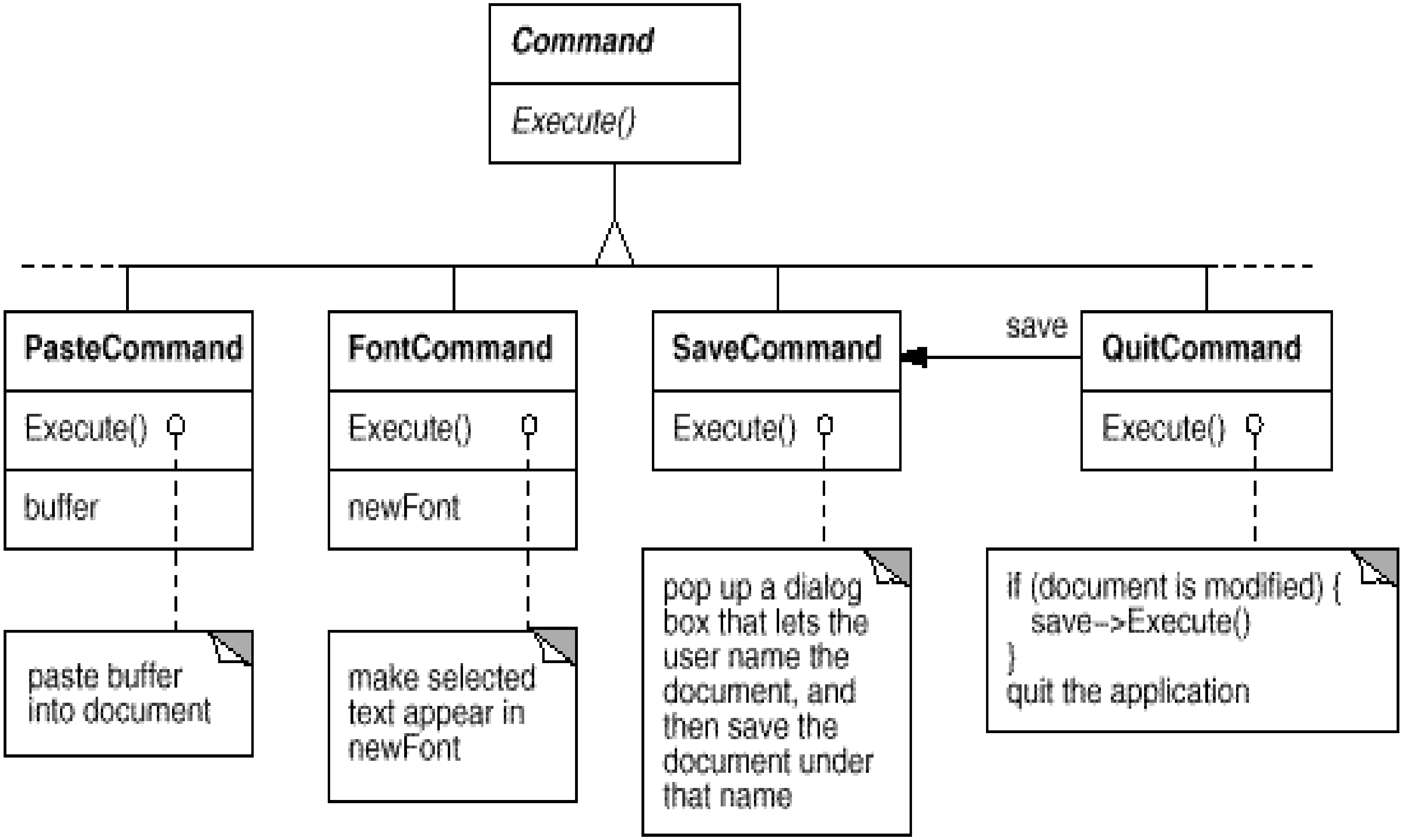
Bridge Pattern



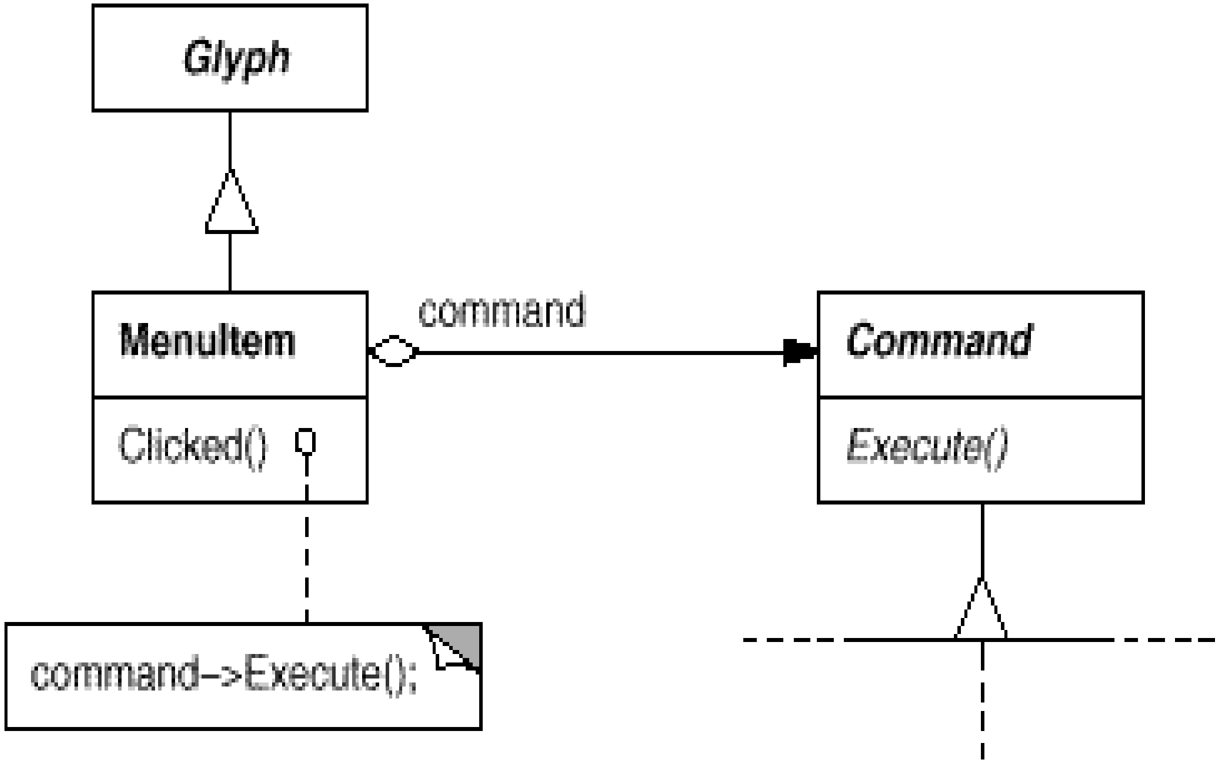
Bridge Pattern



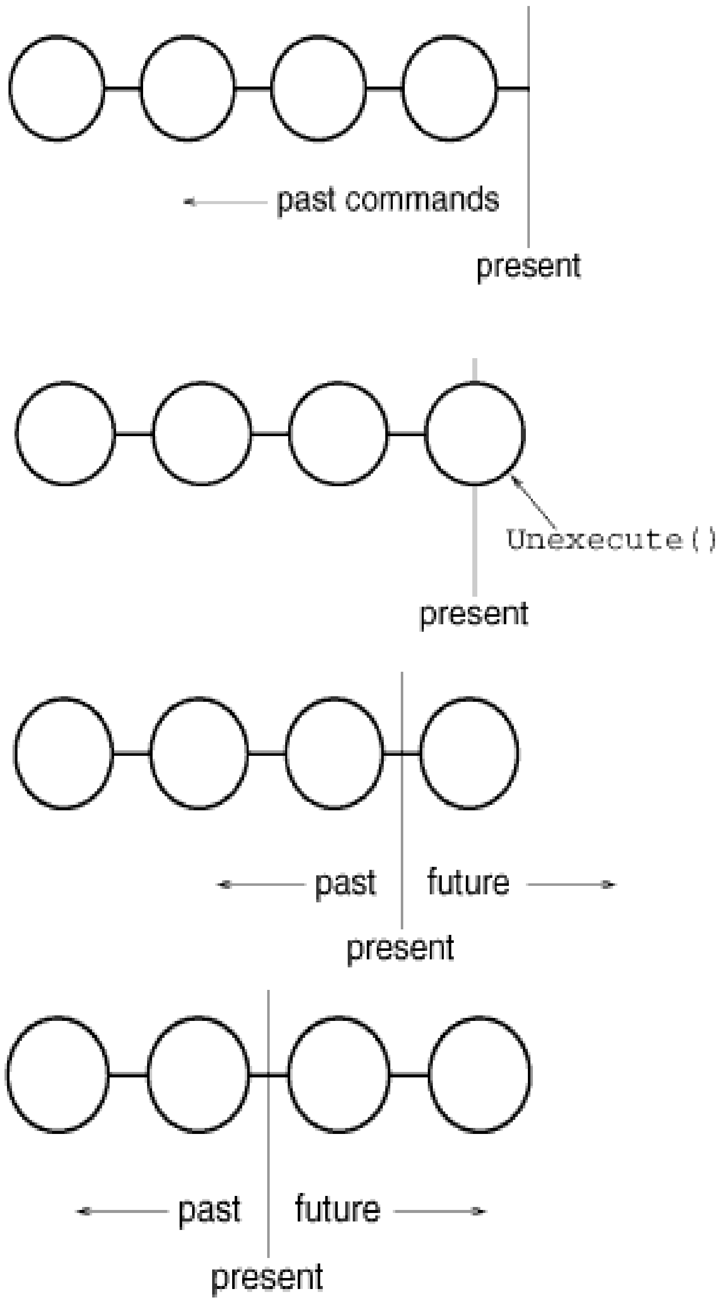
Operations: Command Pattern


Command
Execute()
PasteCommand
Execute()
buffer
FontCommand
Execute()
newFont
SaveCommand
Execute()
save
QuitCommand
Execute()
paste buffer into document
make selected text appear in newFont
pop up a dialog box that lets the user name the document, and then save the document under that name
if (document is modified) {
save->Execute()
}
quit the application

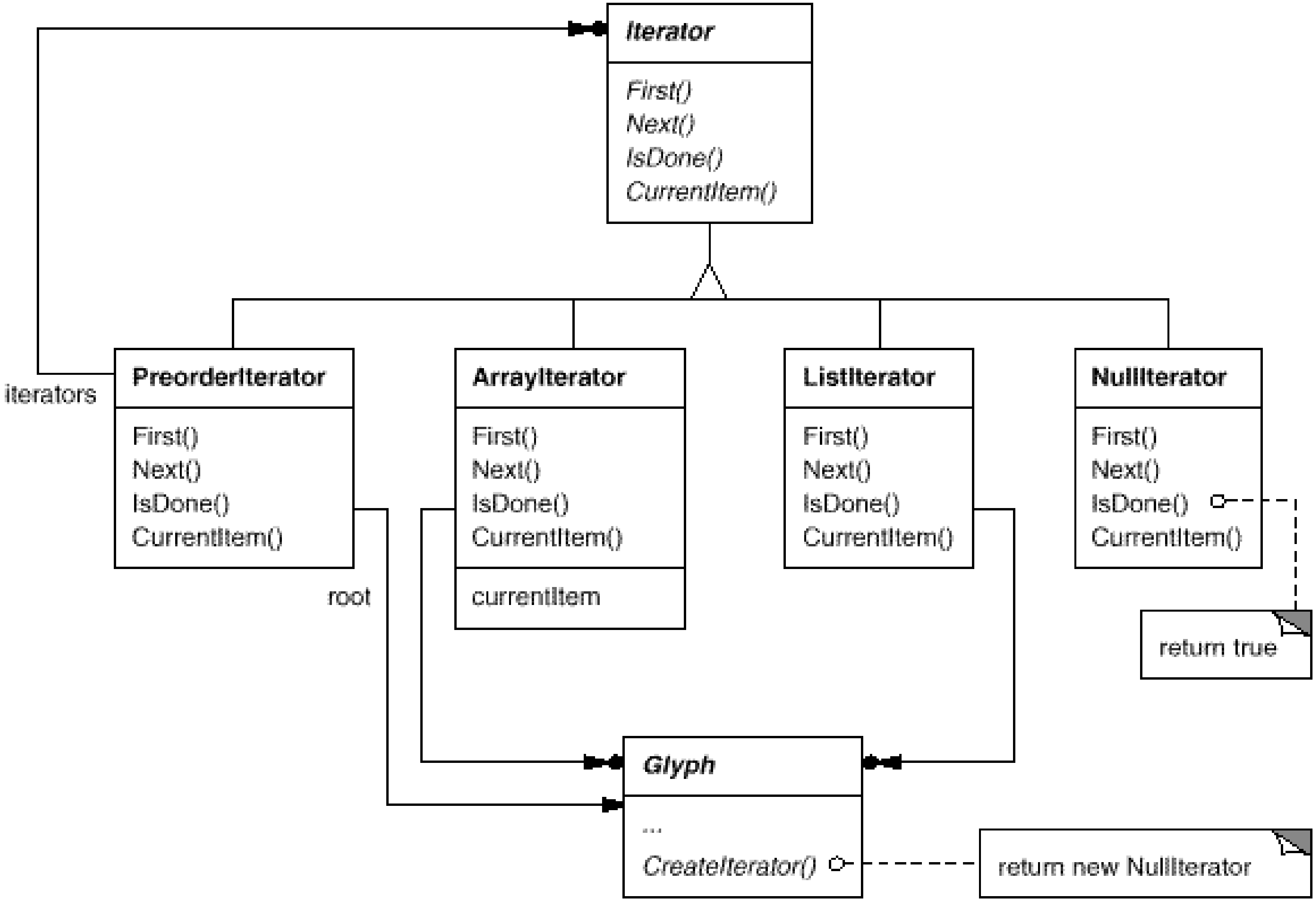
Command Pattern (invoke)



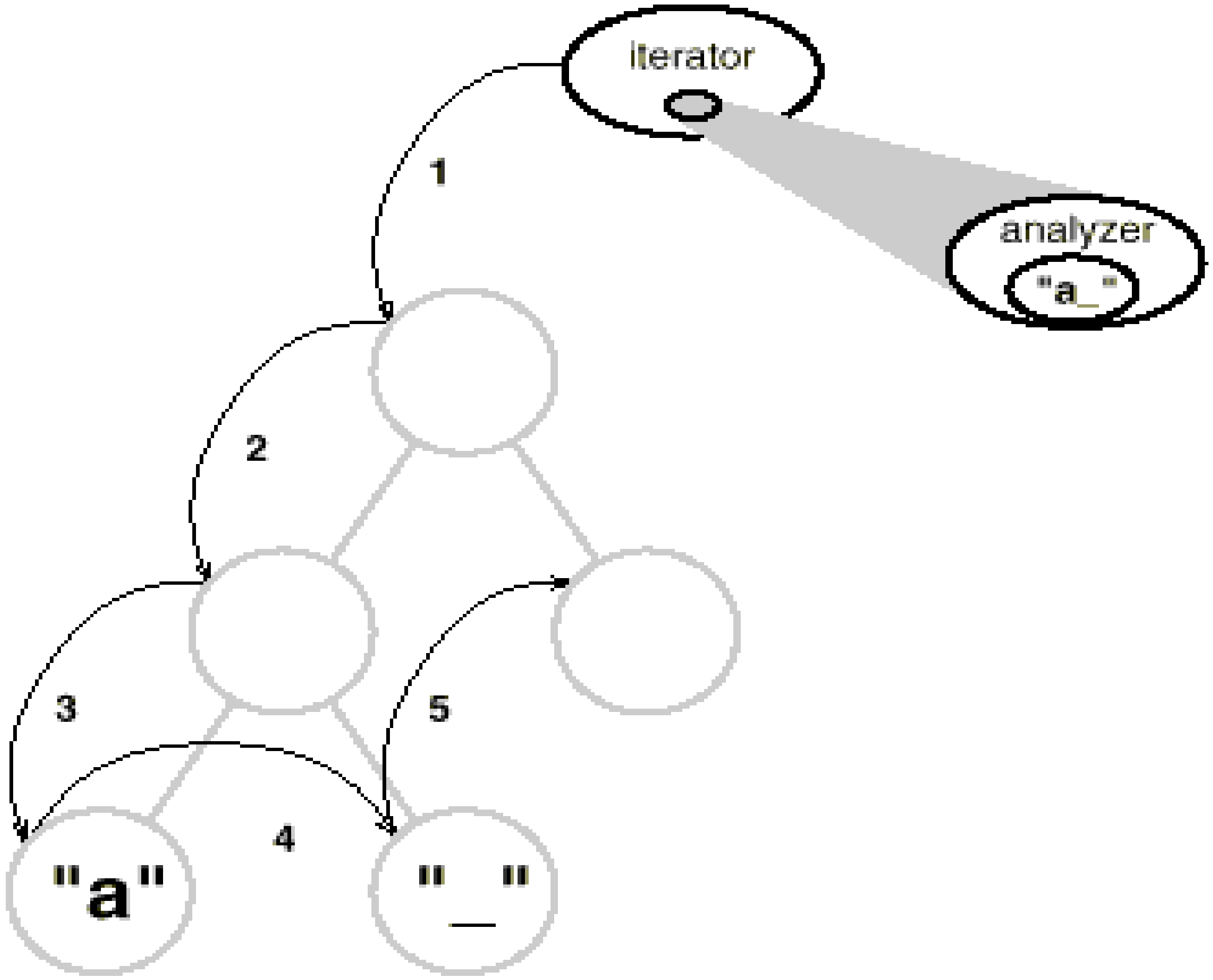
Command Pattern: undo/redo



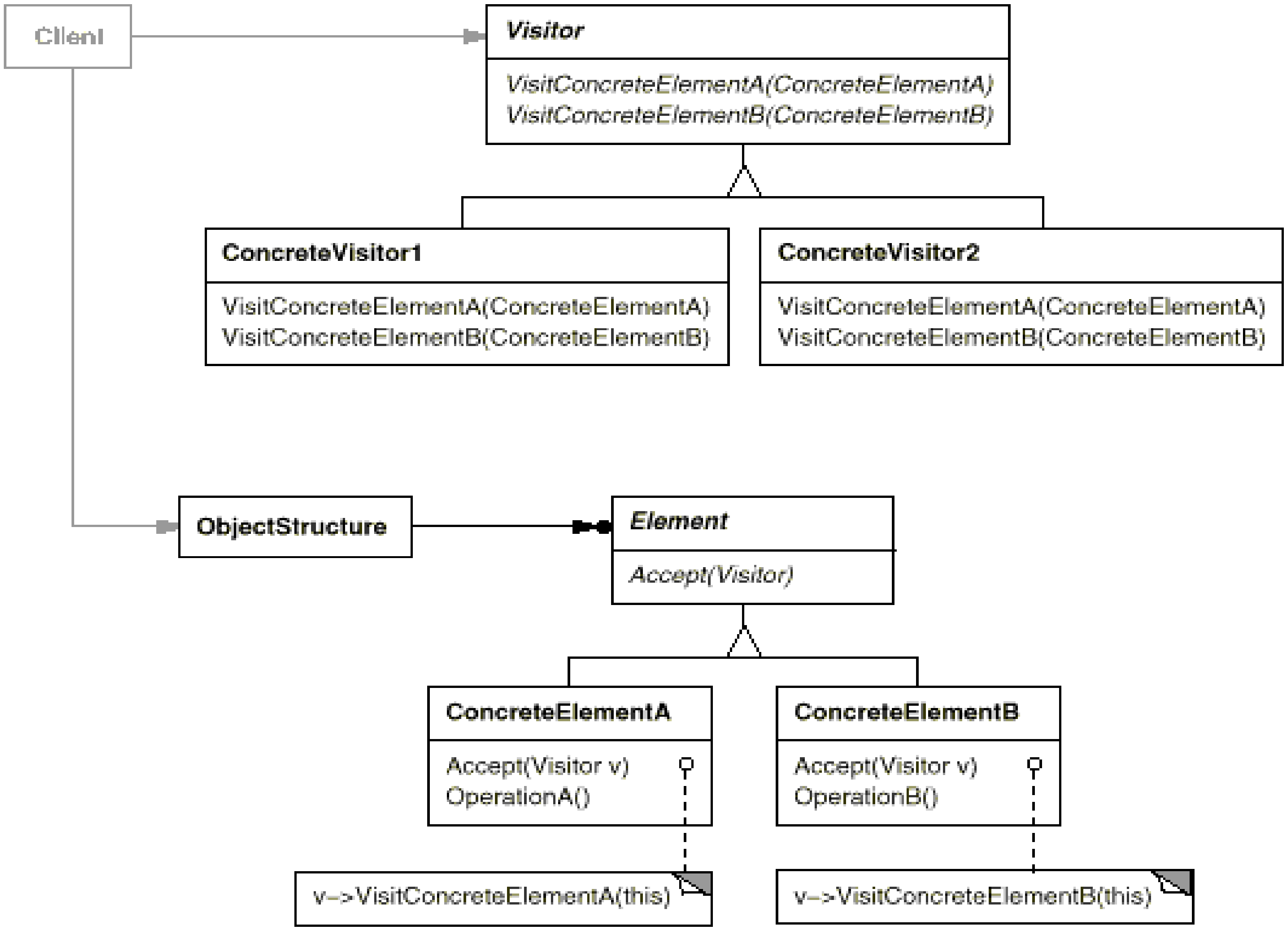
Iterator Pattern



Traversal vs. Traversal Actions



Visitor Pattern



Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor)
ConcreteElementA
Accept(Visitor v)
OperationA()
ConcreteElementB
Accept(Visitor v)
OperationB()
v->VisitConcreteElementA(this)
v->VisitConcreteElementB(this)