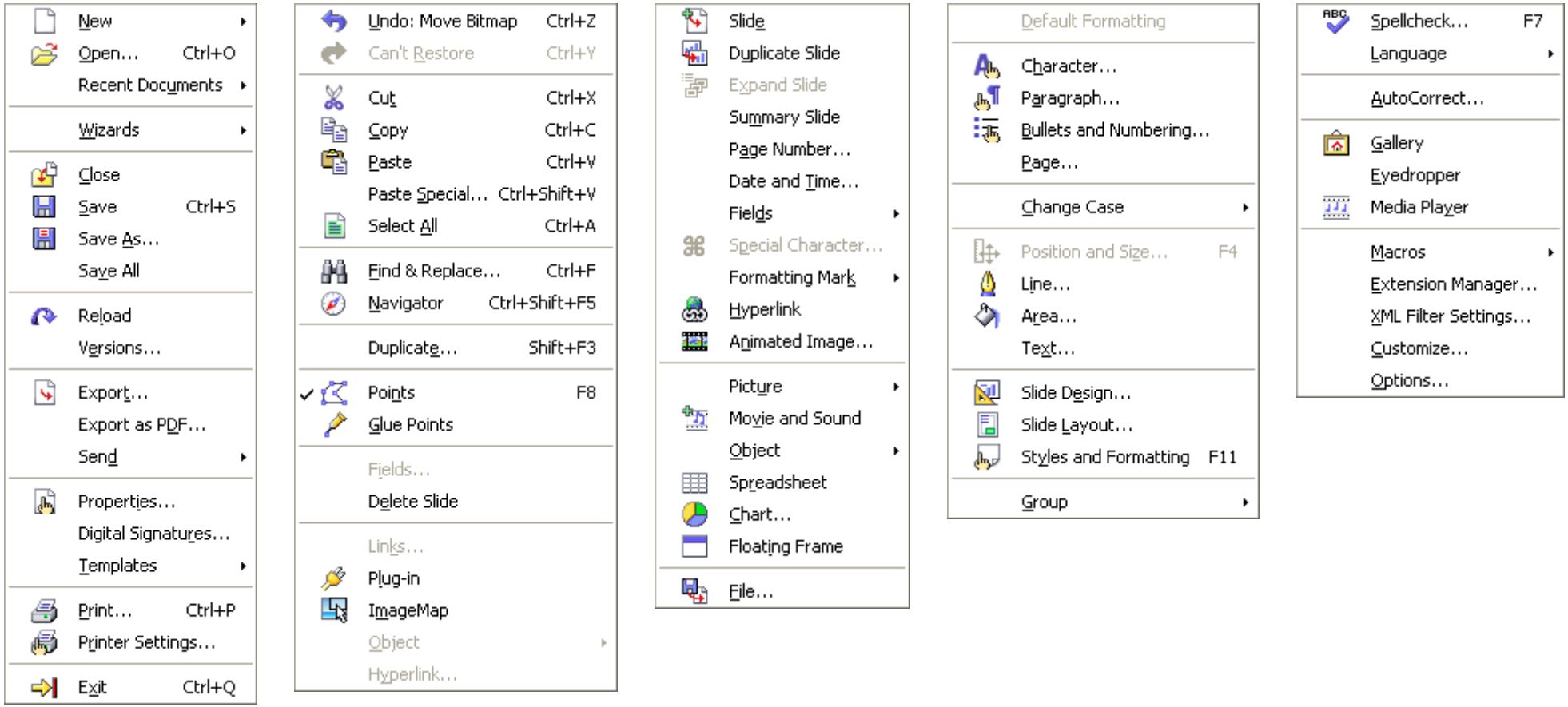
Command Pattern

Hans Vangheluwe and Alexandre Denault

Example


New
Open... Ctrl+O
Recent Documents
Wizards
Close
Save Ctrl+S
Save As...
Save All
Reload
Versions...
Export...
Export as PDF...
Send
Properties...
Digital Signatures...
Templates
Print... Ctrl+P
Printer Settings...
Exit Ctrl+Q
Undo: Move Bitmap Ctrl+Z
Can't Restore Ctrl+Y
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Paste Special... Ctrl+Shift+V
Select All Ctrl+A
Find & Replace... Ctrl+F
Navigator Ctrl+Shift+F5
Duplicate... Shift+F3
Points F8
Glue Points
Fields...
Delete Slide
Links...
Plug-in
ImageMap
Object
Hyperlink...
Slide
Duplicate Slide
Expand Slide
Summary Slide
Page Number...
Date and Time...
Fields
Special Character...
Formatting Mark
Hyperlink
Animated Image...
Picture
Movie and Sound
Object
Spreadsheet
Chart...
Floating Frame
File...
Default Formatting
Character...
Paragraph...
Bullets and Numbering...
Page...
Change Case
Position and Size... F4
Line...
Area...
Text...
Slide Design...
Slide Layout...
Styles and Formatting F11
Group
Spellcheck... F7
Language
AutoCorrect...
Gallery
Eyedropper
Media Player
Macros
Extension Manager...
XML Filter Settings...
Customize...
Options...

Problem

User interface toolkit/widget library includes buttons and menus that carry out a request corresponding to user input.

The buttons and menus (from the library) can't explicitly implement the action, because only an application knows what should be done on which object.

GUIs only provide a button construct. It has no behavior.

It's up to the programmer to give the button a behavior.

How do we encapsulate behavior?

Command Pattern

Encapsulate requests/methods as OBJECTS!

“objectifying” a design is very common in design patterns

Motivation

Separates an operation from the object that executes it.

Before: method is integral part of class.

With the Command Pattern, it is possible to parametrize an object with an operation.

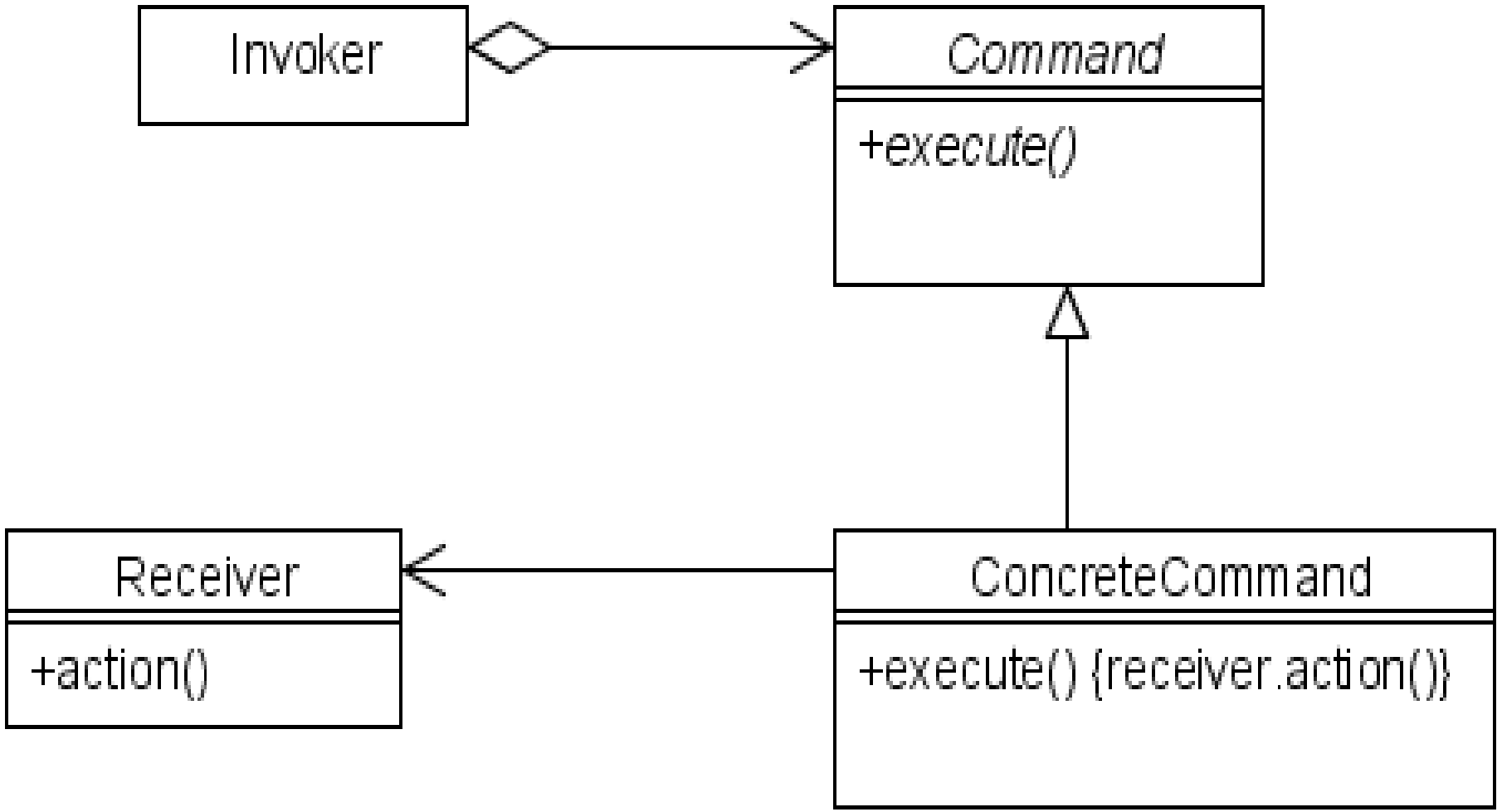
Support undo/redo

Possible to execute the request at a different time and/or at a different location.

How?

By passing the command object to another process.

Participants



Why?

Each item in the menu is conceptually the same object.

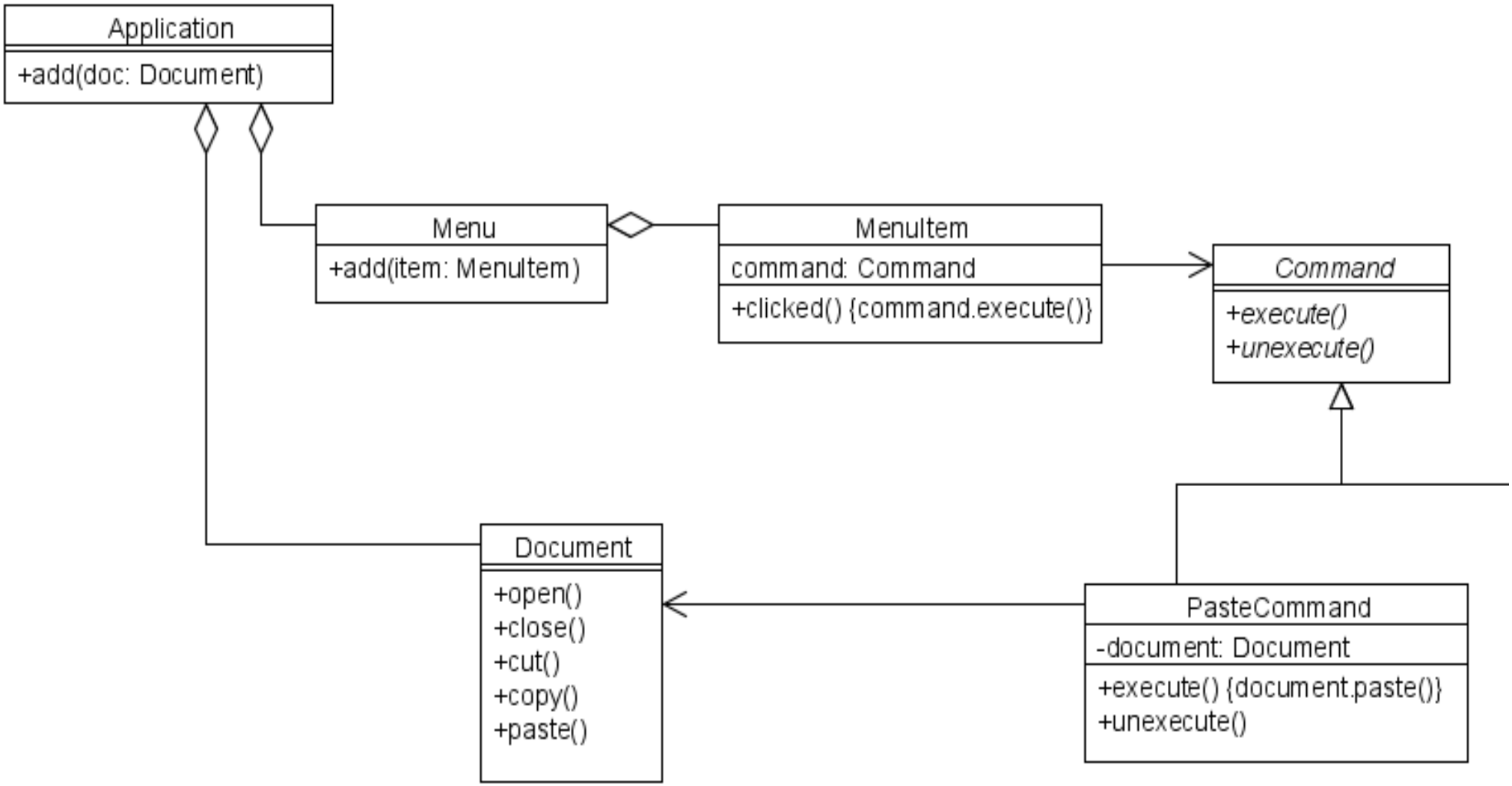
The only difference is with the action that is taken when pressed.

Solution:

parametrize the menu item object

with a command object.

Class Diagram of Example



Collaborations

Implementation

How “intelligent” should a command be?

- Just call receiver's action (cfr. Adapter pattern)
- Implement all functionality directly in execute()

Supporting Undo/Redo

Since a command is an object,

it can hold state (memory).

A command object could store the information required to undo itself.

The receiver

The arguments to the operation performed on the receiver

The original (changed) values in the receiver or

ability to apply inverse operation

More than one level of undo/redo: use a history list.

Supporting Undo/Redo

Each command should know how to undo and redo itself (one level) by providing an unexecute() method.

A command manager holds the history list of commands:

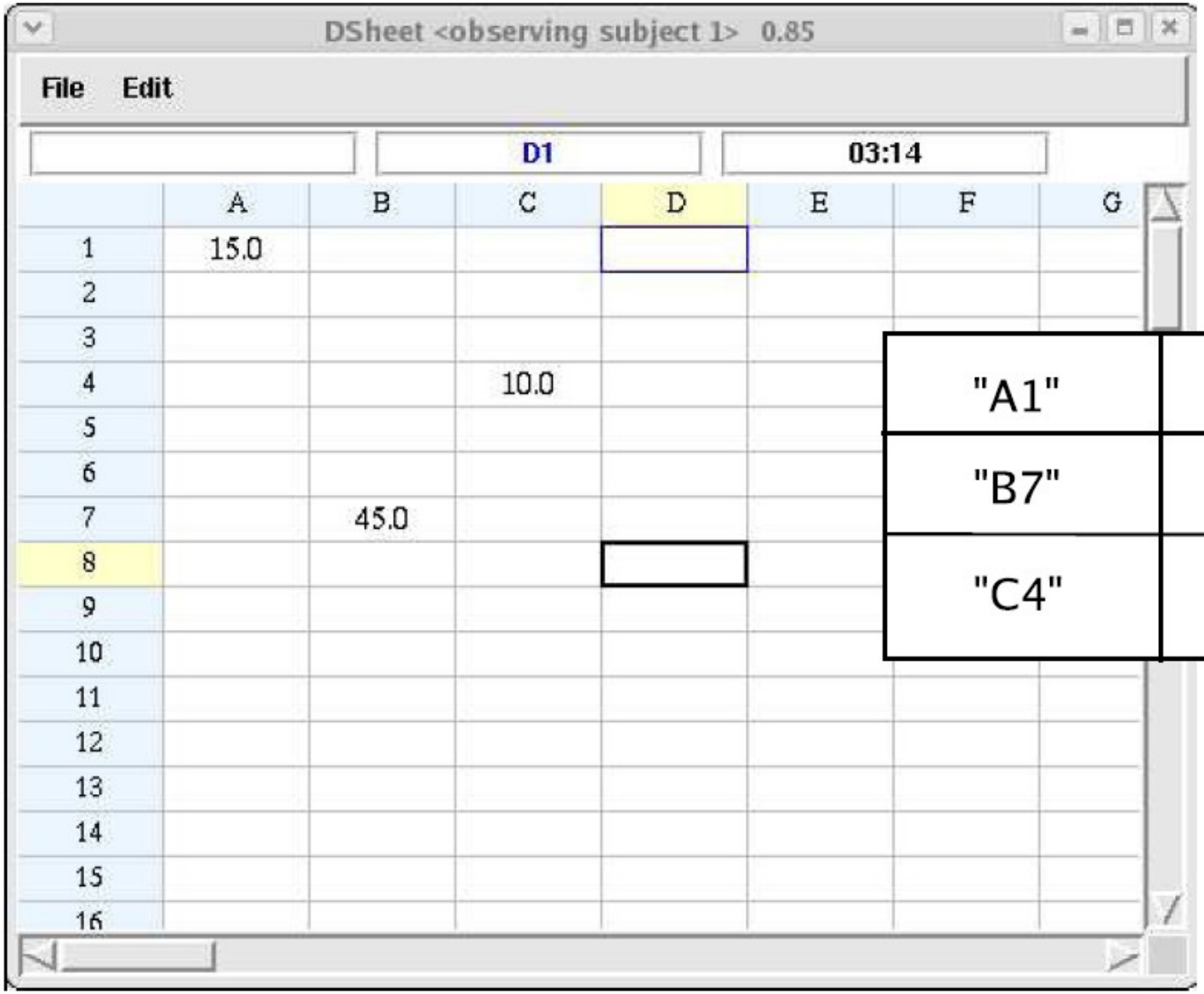
[commandA; commandB; commandC; :::]

Moving backward: undoing commands

Moving forward: redoing commands

Let's go over an example...

DSheet



HashTable:

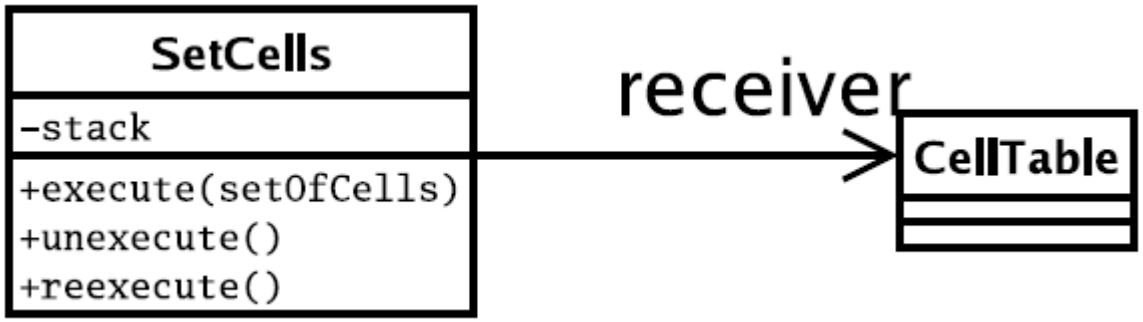
"A1"	Cell("A1", "=5+C4")
"B7"	Cell("B7", "=45")
"C4"	Cell("C4", "=10")

SetCells Command

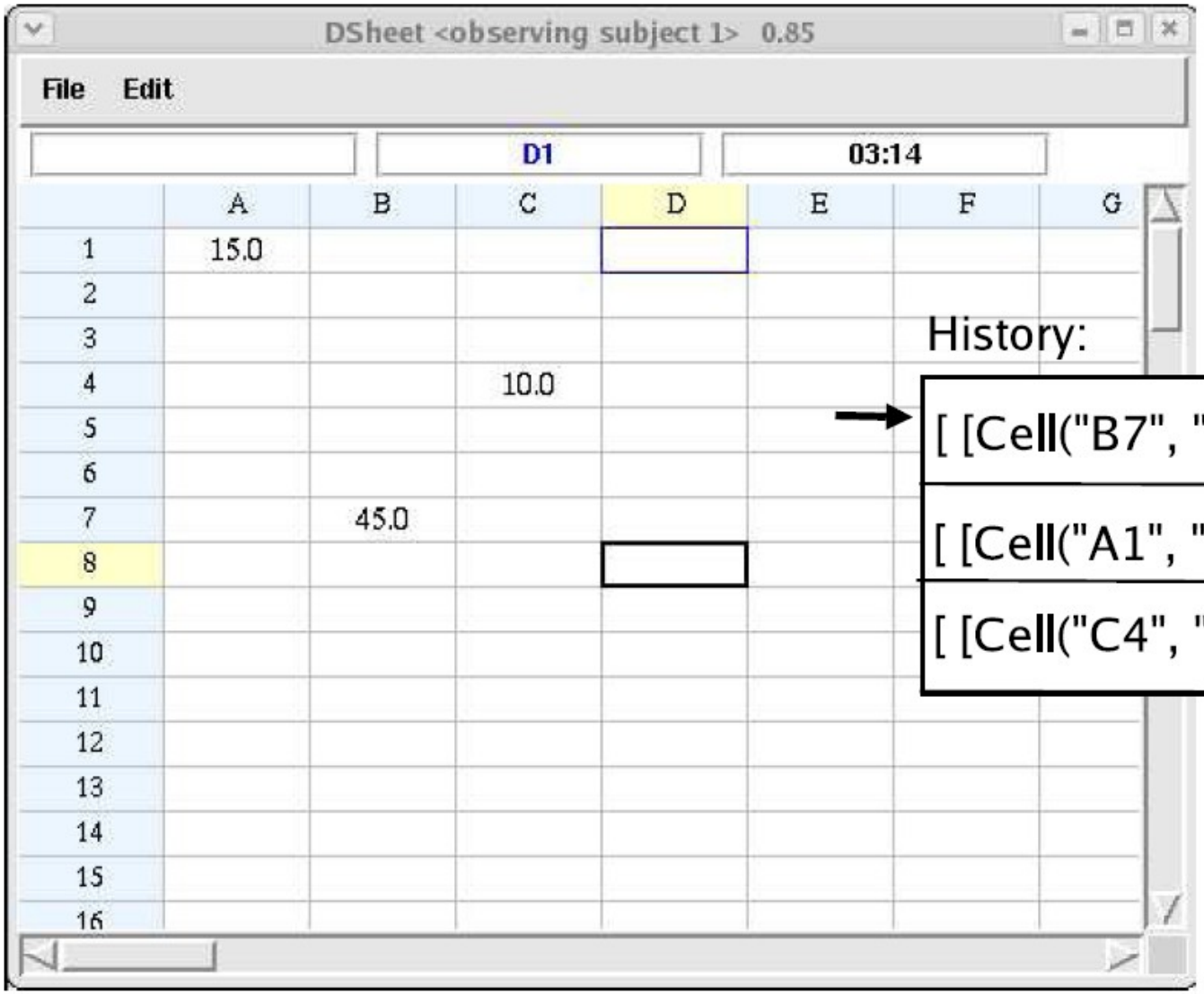
The SetCells command, acting on the CellTable (a HashTable) is used to support undo/redo

The history list is stored directly in the SetCells command. (SetCells Command is a Singleton)

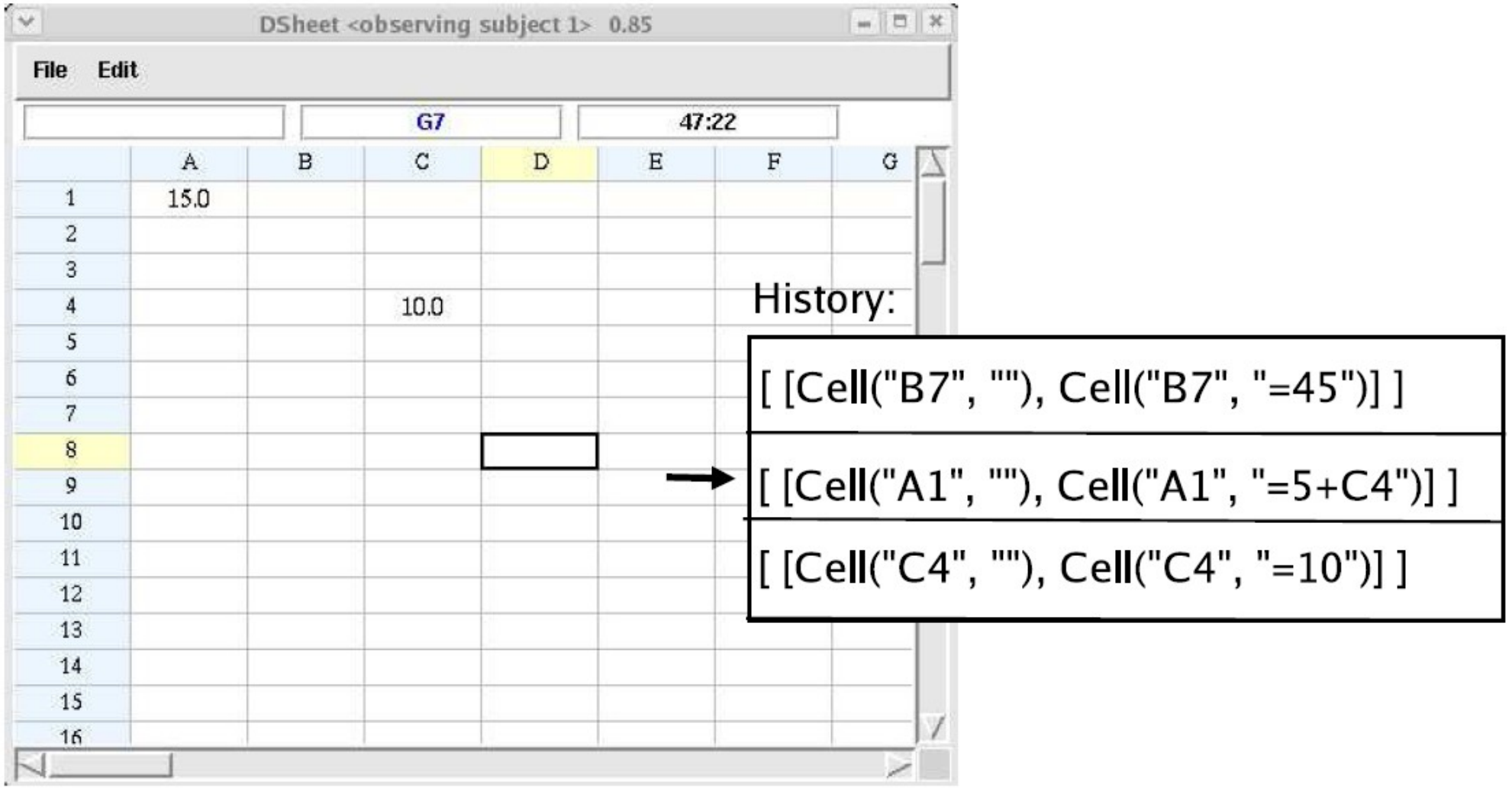
Each time a set of cells is modified, the difference between the previous state and the next state is added on the history stack.



Example (cont.)



Undo


DSheet <observing subject 1> 0.85
File
Edit
G7
47:22
A
B
C
D
E
F
G
15.0
10.0
History:
[[Cell("B7", ""), Cell("B7", "=45")]]
[[Cell("A1", ""), Cell("A1", "=5+C4")]]
[[Cell("C4", ""), Cell("C4", "=10")]]

Undo/Redo

Beware of hystersis (errors accumulating)!

Consequences

Decoupling of the command and the invoker.

Requests can be issued without knowing about:

Operation

Receiver

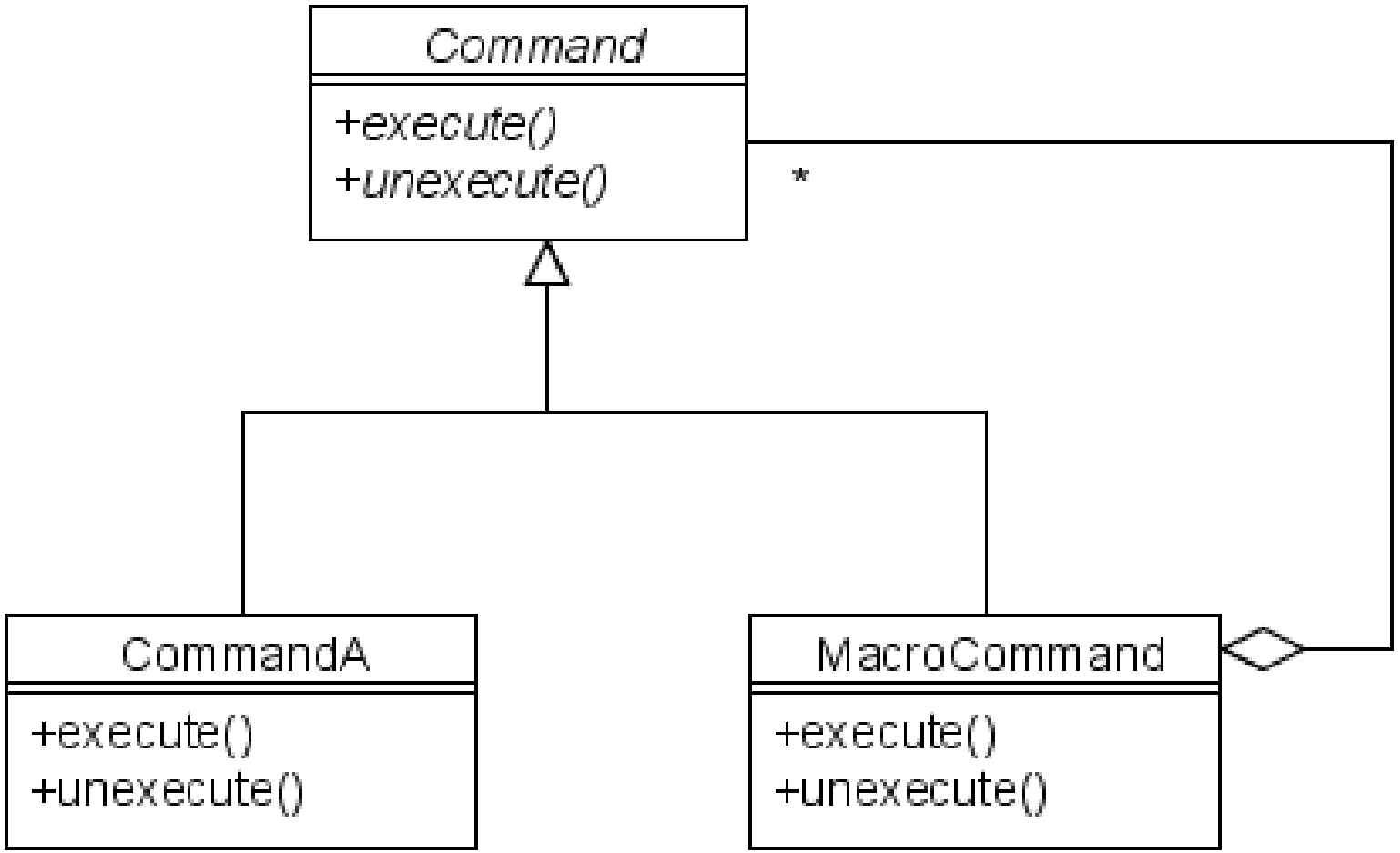
Commands are first-class objects. They can be manipulated (saved, duplicated, passed around, ...)

like any other object.

You can assemble commands into composite commands.

Adding new commands is easy and does not require the modification of existing code.

Hierarchy in Commands (Macro)



How else can it be used?

- Transactional Behavior
- Action Queuing / Progress Monitoring (bar)
- Macro Recording
- Networking / Distributed Actions