Entwicklung objektorientierter Software mit formalen Methoden

Design Patterns

Bernhard Beckert

UNIVERSITÄT KOBLENZ-LANDAU

Design Patterns: Libraries for design

Collection of solutions for common software design problems

Design Patterns: Libraries for design

Collection of solutions for common software design problems

Make experiences of designers available to others

$\Rightarrow$ better education

Design Patterns: Libraries for design

Collection of solutions for common software design problems

Make experiences of designers available to others

$\Rightarrow$ better education

Prevent software engineers from searching for already found solutions

$\Rightarrow$ increase of effectiveness

Design Patterns: Libraries for design

Collection of solutions for common software design problems

Make experiences of designers available to others

$\Rightarrow$ better education

Prevent software engineers from searching for already found solutions

$\Rightarrow$ increase of effectiveness

Support communication between developers

$\Rightarrow$ easier orientation

Design Patterns: Libraries for design

Collection of solutions for common software design problems

Make experiences of designers available to others

$\Rightarrow$ better education

Prevent software engineers from searching for already found solutions

$\Rightarrow$ increase of effectiveness

Support communication between developers

$\Rightarrow$ easier orientation

Document design decisions of a software system

$\Rightarrow$ increase of maintenance

Description scheme of design patterns

Motivation

Where does the need for this pattern come from?

Description scheme of design patterns

Motivation

Where does the need for this pattern come from?

Structure

What is the concrete (class) structure of the pattern?

Description scheme of design patterns

Motivation

Where does the need for this pattern come from?

Structure

What is the concrete (class) structure of the pattern?

Applicability

Under which circumstances is it applicable?

Which are the forces it obeys?

Description scheme of design patterns

Motivation

Where does the need for this pattern come from?

Structure

What is the concrete (class) structure of the pattern?

Applicability

Under which circumstances is it applicable?

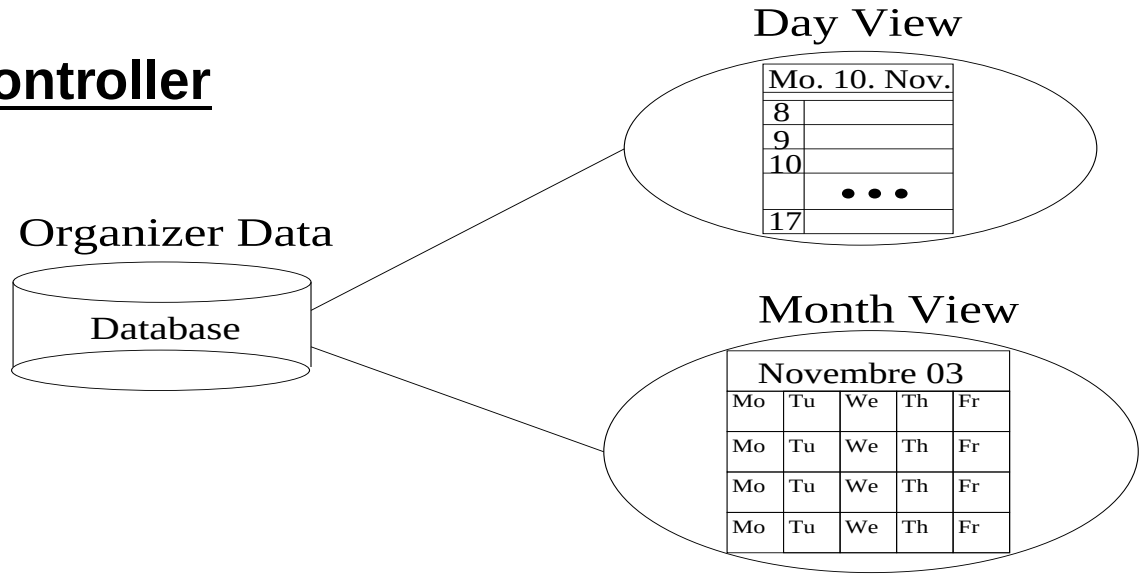
Which are the forces it obeys?

Consequences

What are the effects when using this pattern?

Improving Consistence and Reusability

Model View Controller



Keeping Model, View and Controller separate allows

- **different consistence views**
- **reuse of views in other contexts**
- **dynamic chosen response**

The »Observer«: Separates model and view

Purpose: **Defines a 1-to-n dependance relationship between objects, so that the state change of one object causes a notification of the dependant objects.**

The »Observer«: Separates model and view

Purpose: Defines a 1-to-n dependance relationship between objects, so that the state change of one object causes a notification of the dependant objects.

Applicability:

- If the state change of one object, requires to update several other objects.
- If an object has to inform several other unknown objects.

The »Observer«: Separates model and view

Purpose: Defines a 1-to-n dependance relationship between objects, so that the state change of one object causes a notification of the dependant objects.

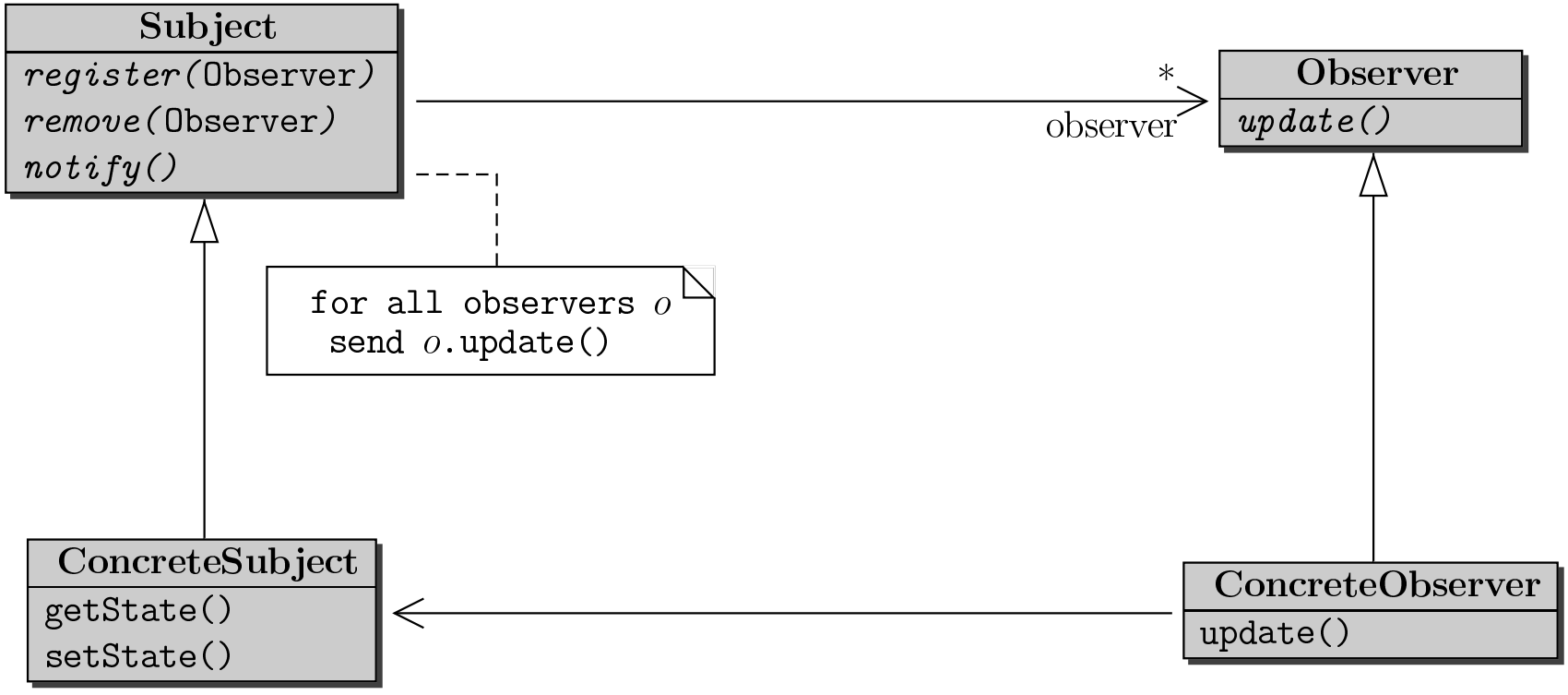
Applicability:

- If the state change of one object, requires to update several other objects.
- If an object has to inform several other unknown objects.

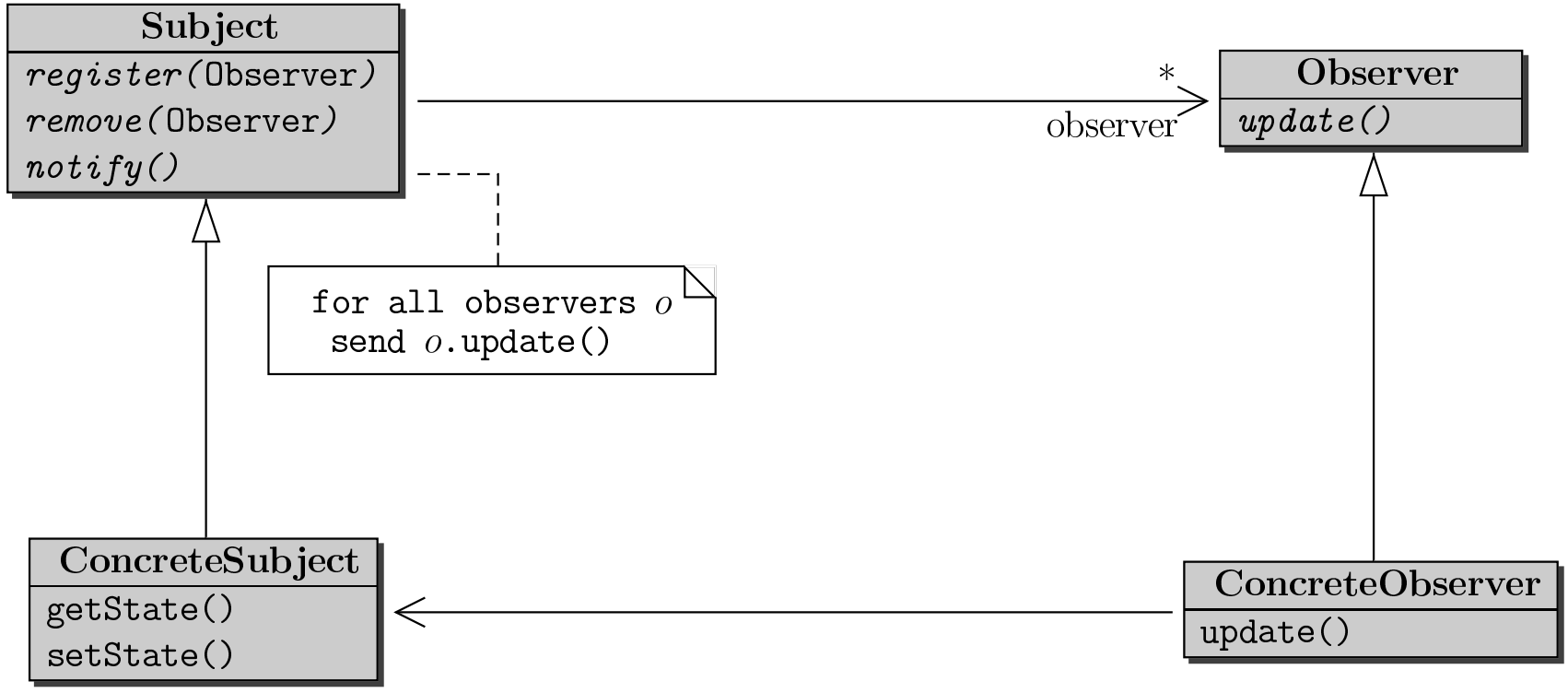
Consequence:

- Independant reuse of subject and observer.
- Observers can be added without changing the observer or subject.

The class structure of the observer pattern



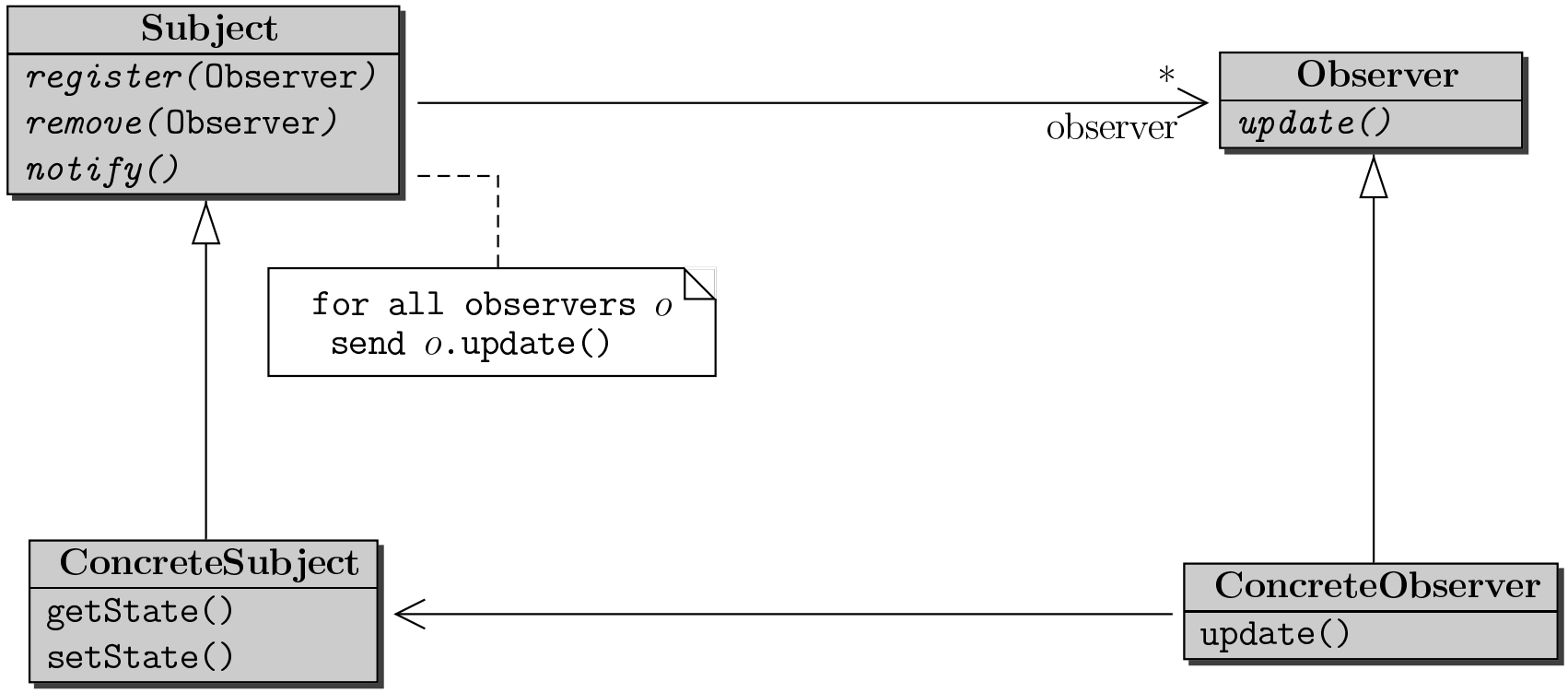
The class structure of the observer pattern



Two modes:

- **pull: the observer has to ask for the new subject state (`getState`)**

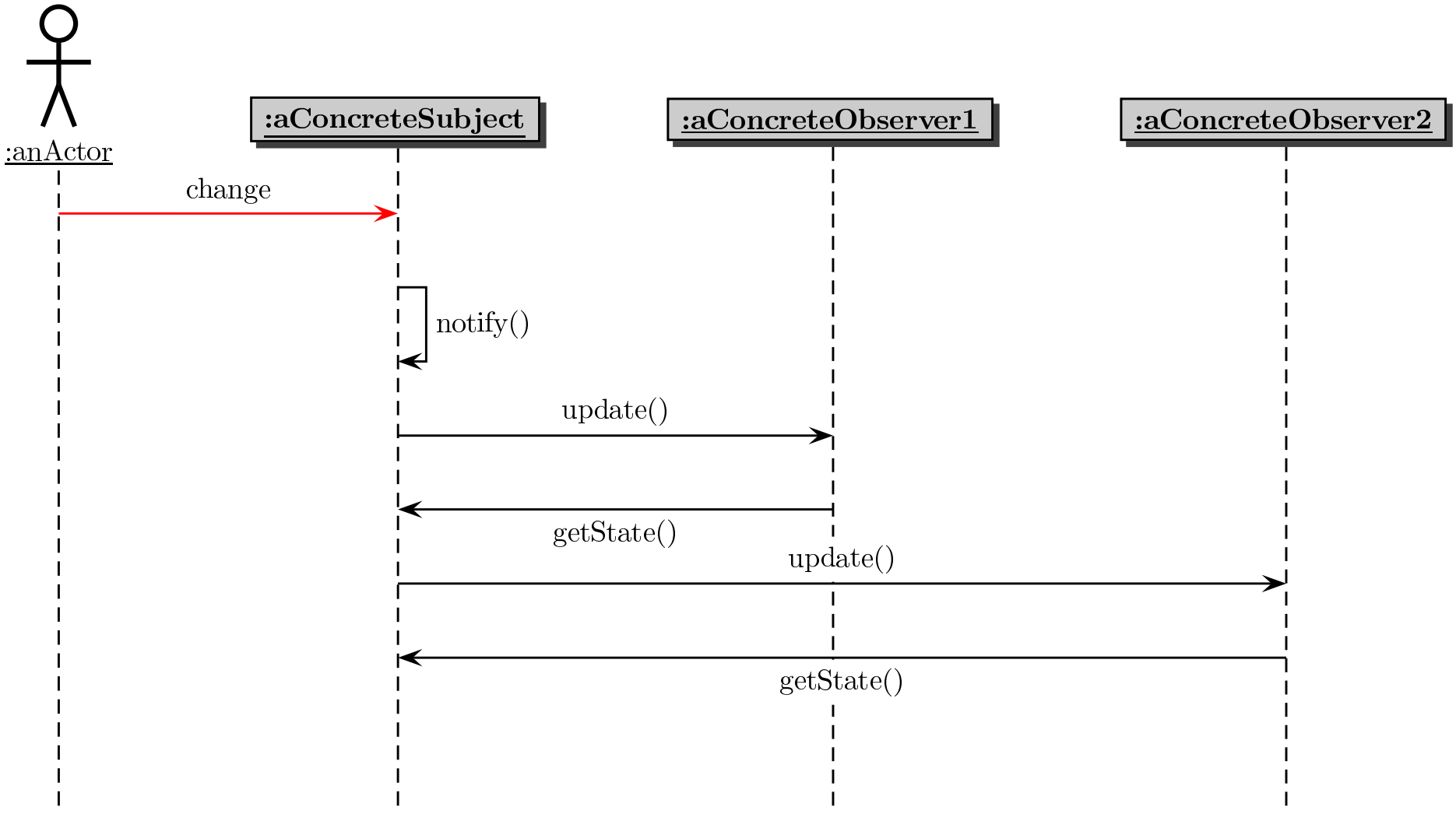
The class structure of the observer pattern



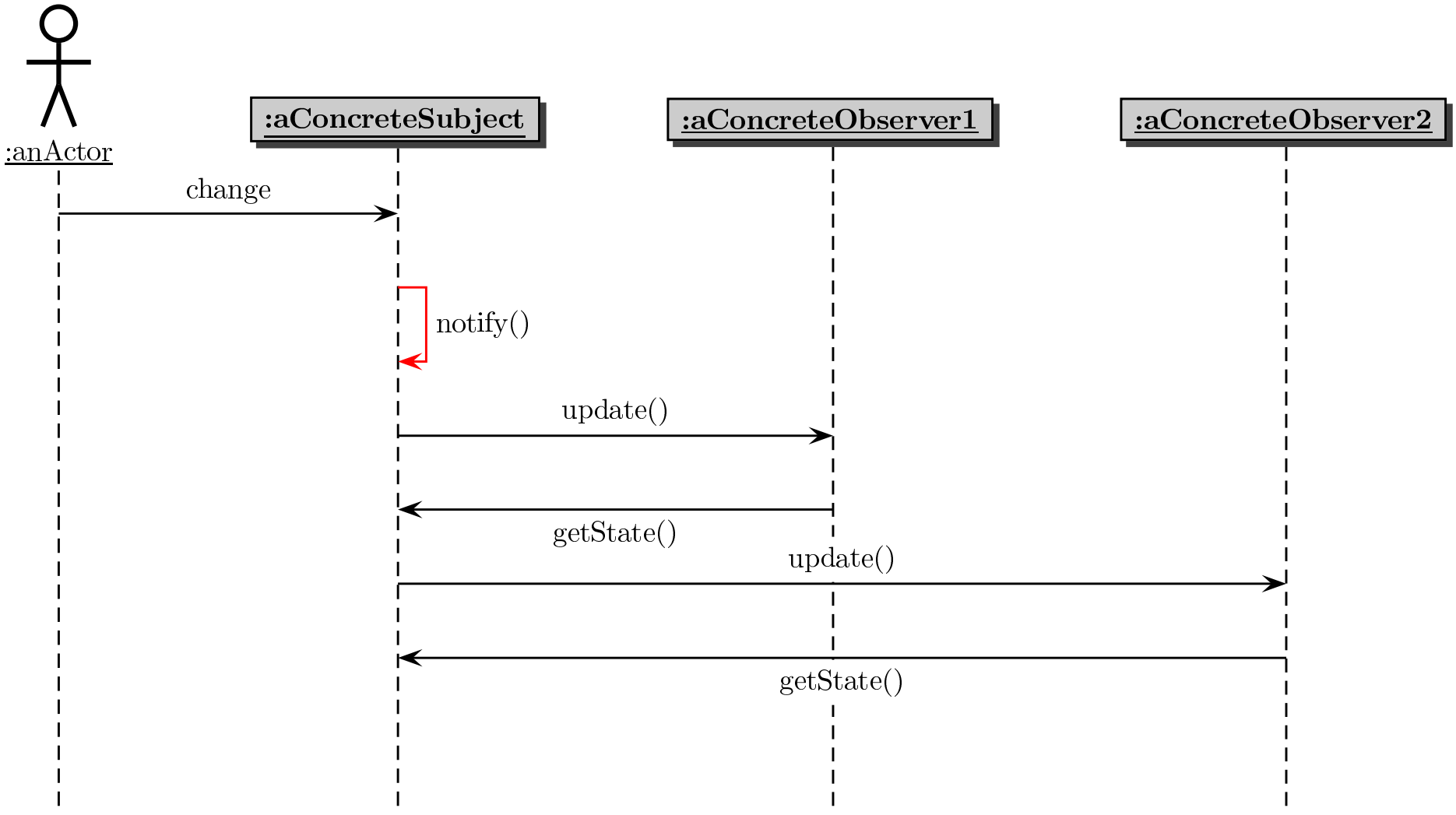
Two modes:

- **pull: the observer has to ask for the new subject state (`getState`)**
- **push: change information is handed over to the observer when notifying**

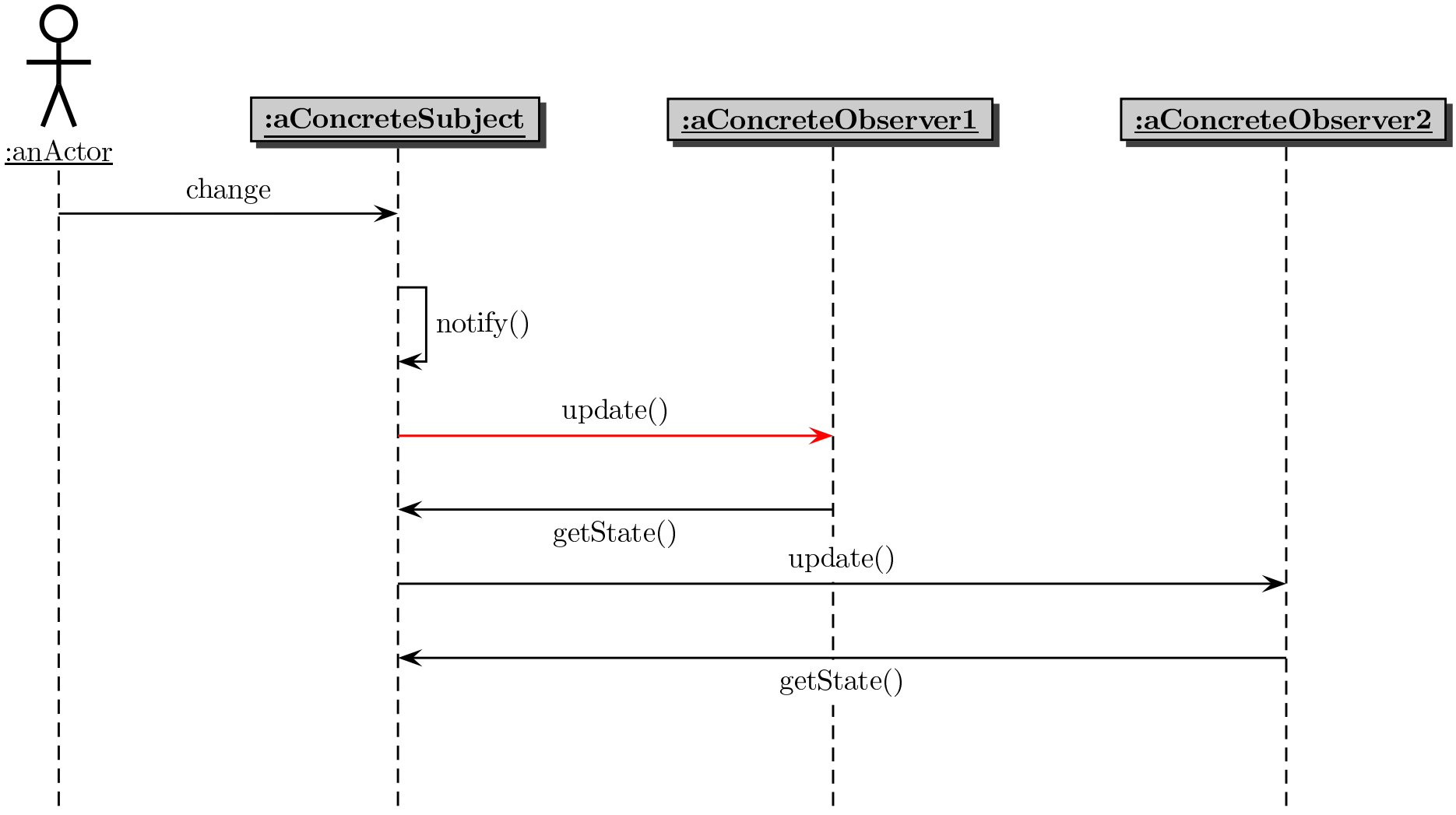
Update of views implementing an observer



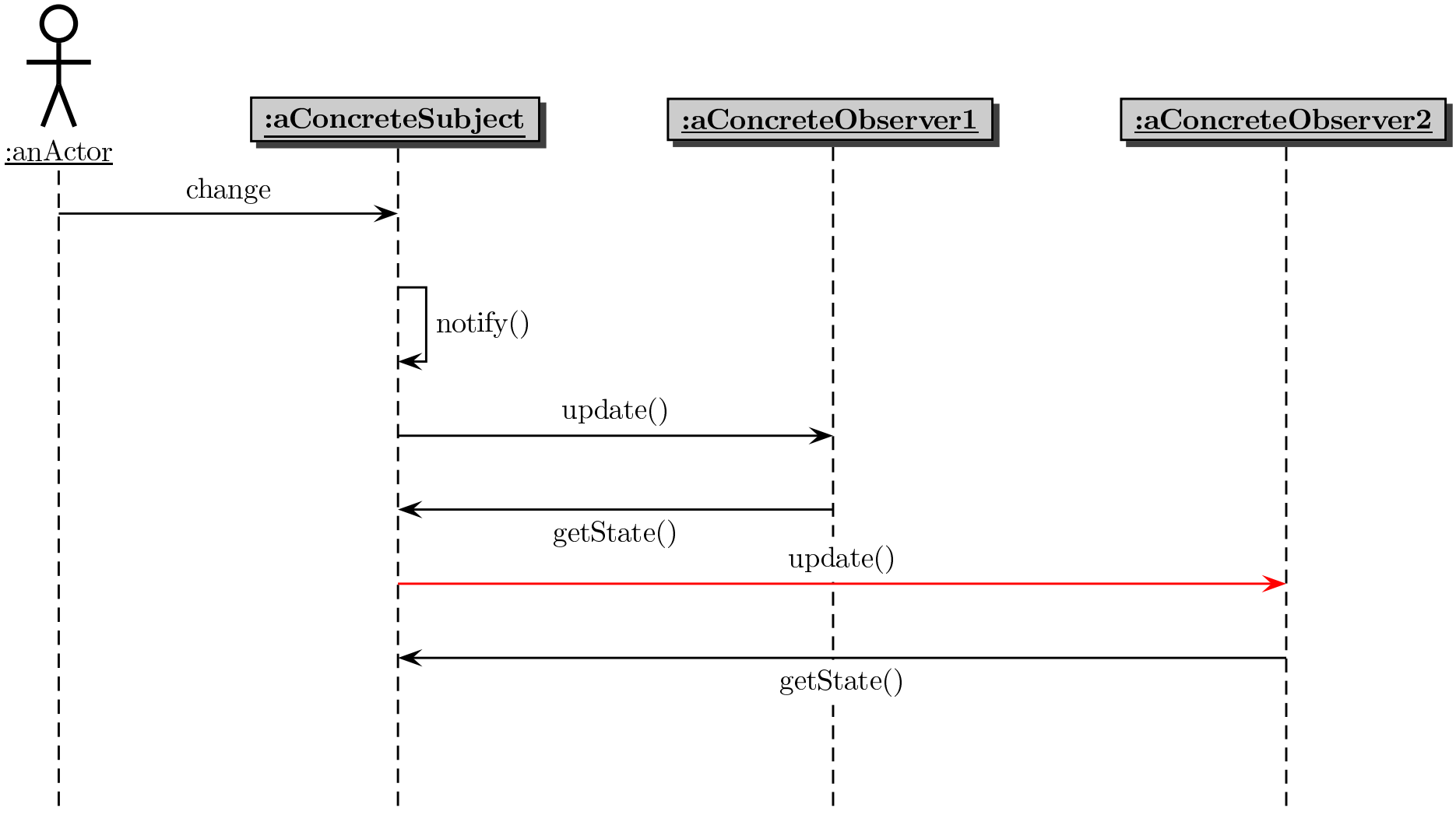
Update of views implementing an observer



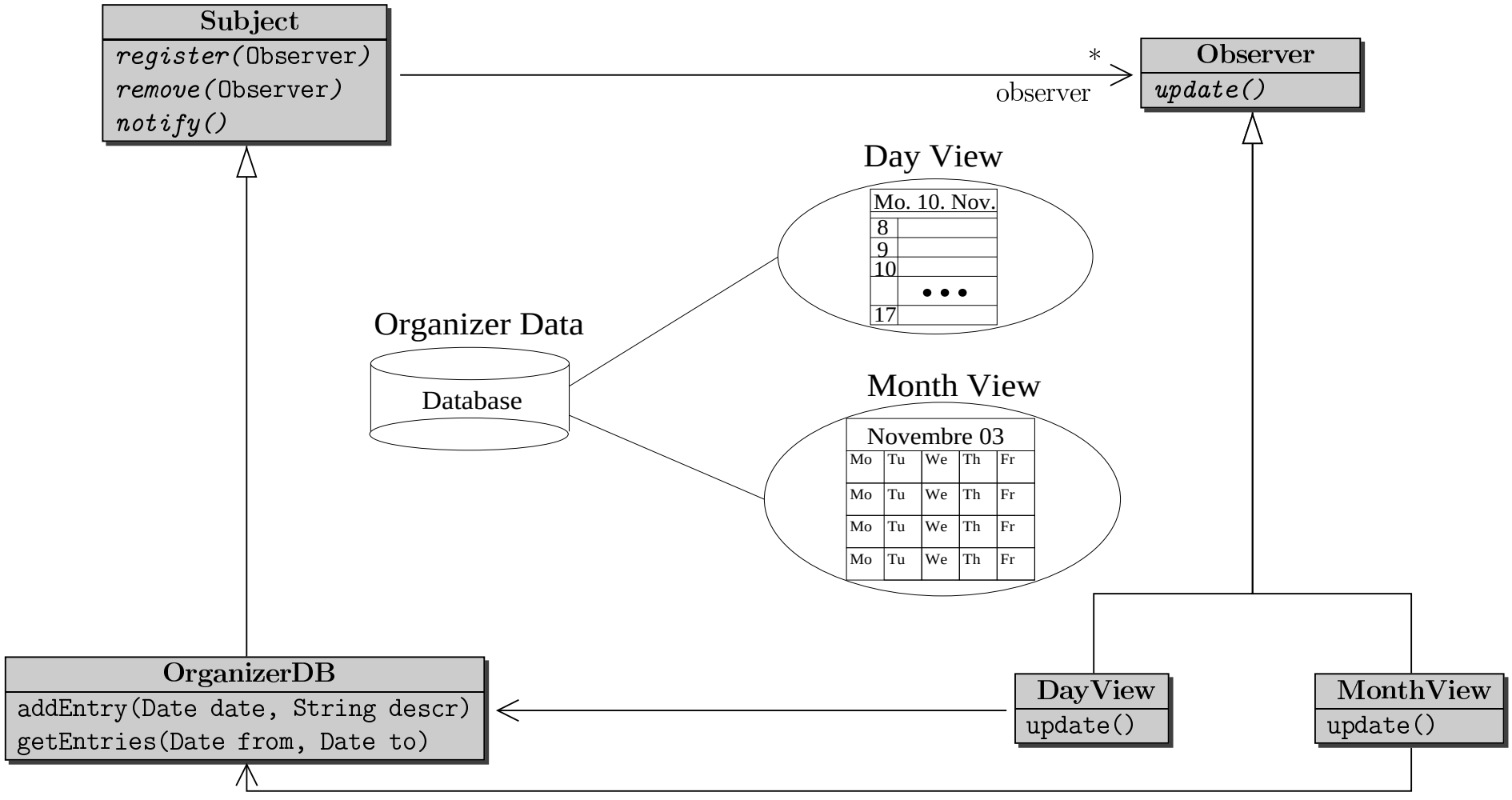
Update of views implementing an observer



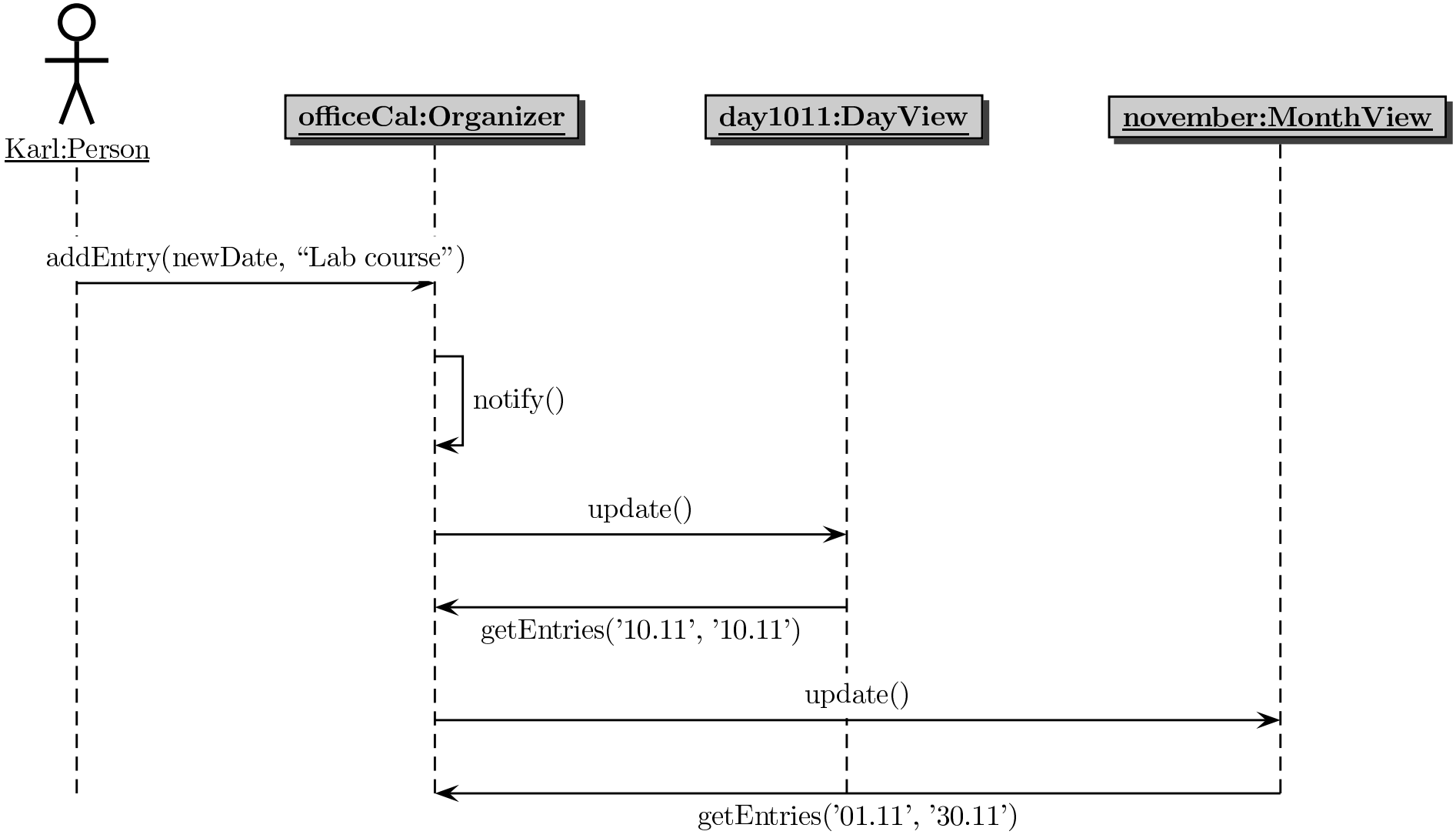
Update of views implementing an observer



Example: Views of an organizer



Updating the organizer's views



The »Command«: Actions as objects

Purpose: **Encapsulate a command as an object. Allows to keep track of actions (transactions).**

The »Command«: Actions as objects

Purpose: **Encapsulate a command as an object. Allows to keep track of actions (transactions).**

Applicability:

- **parameterise clients,**

The »Command«: Actions as objects

Purpose: **Encapsulate a command as an object. Allows to keep track of actions (transactions).**

Applicability:

- **parameterise clients,**
- **log commands,**

The »Command«: Actions as objects

Purpose: **Encapsulate a command as an object. Allows to keep track of actions (transactions).**

Applicability:

- **parameterise clients,**
- **log commands,**
- **realise undo functionality or**

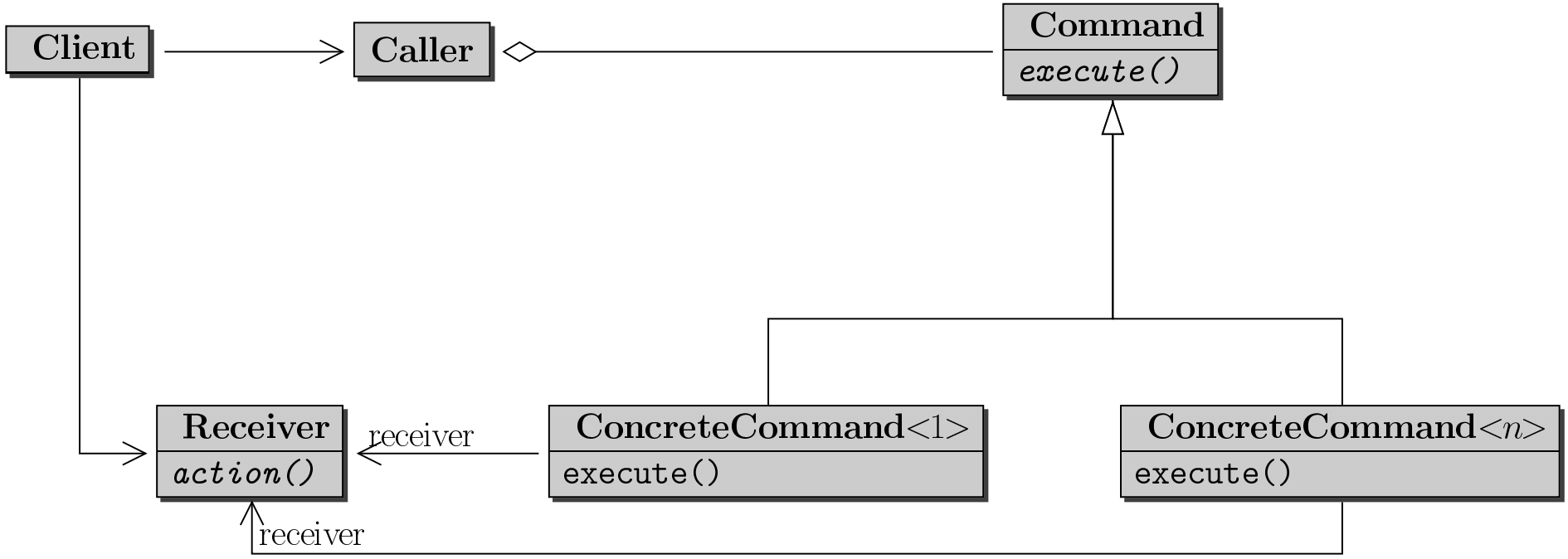
The »Command«: Actions as objects

Purpose: Encapsulate a command as an object. Allows to keep track of actions (transactions).

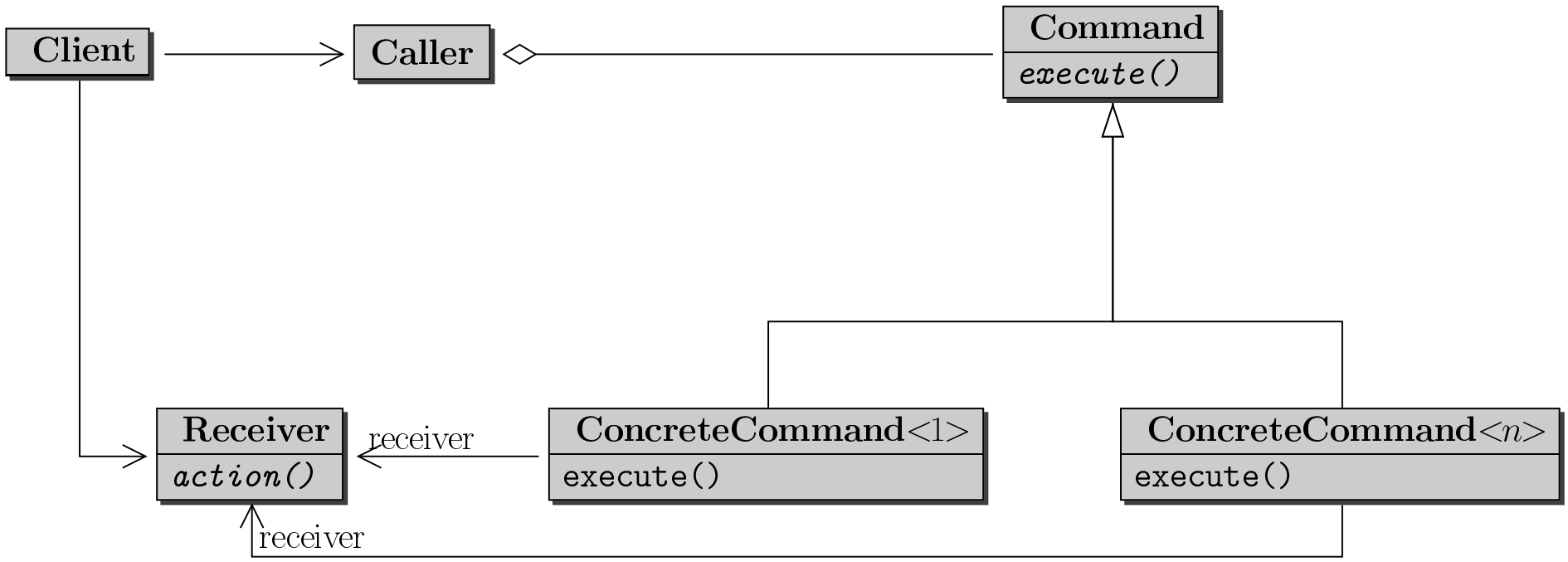
Applicability:

- parameterise clients,
- log commands,
- realise undo functionality or
- cache commands in a queue in order to execute them at a certain time in the future

The class structure of the command pattern

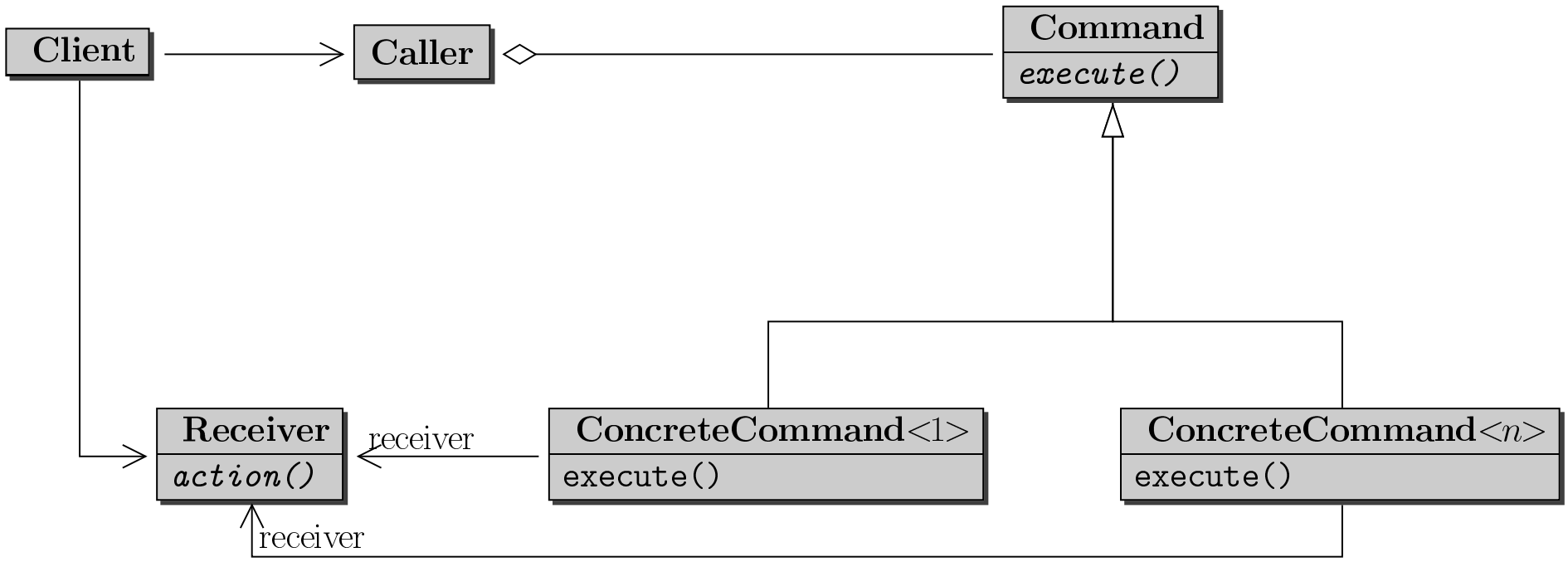


The class structure of the command pattern



Call of `execute()` on `Command` invokes the receiver's `action()` operation.

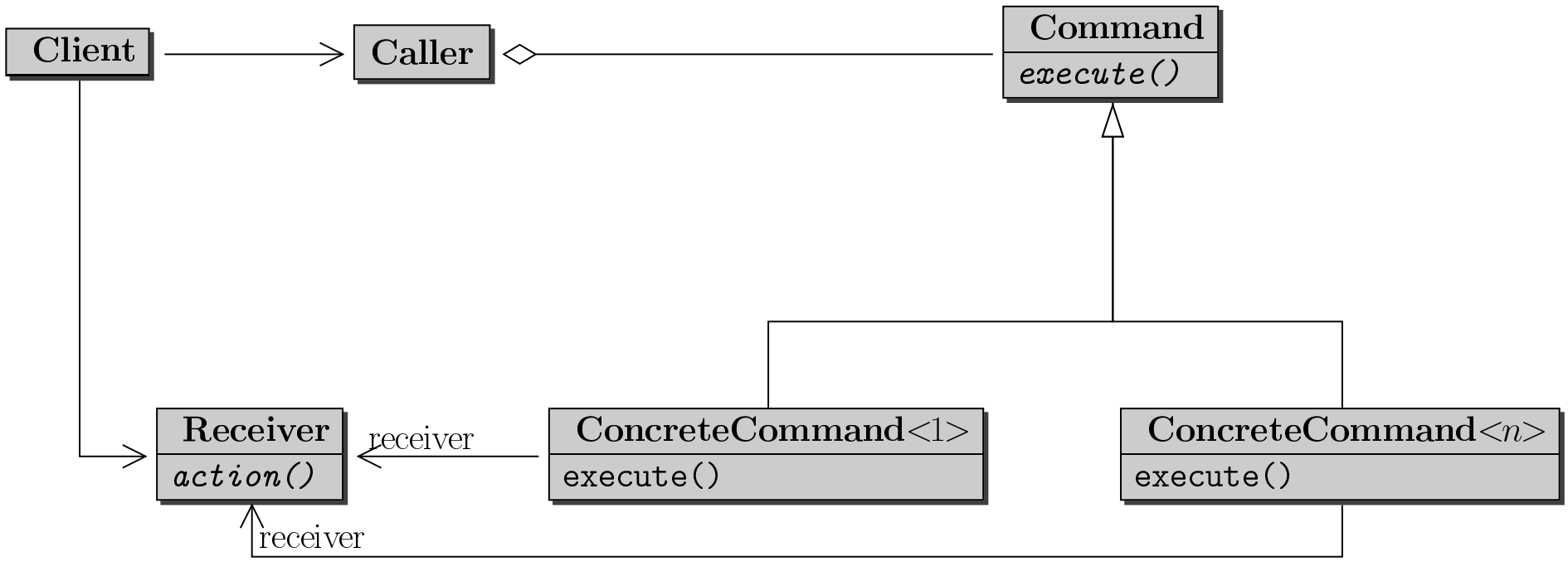
The class structure of the command pattern



Call of `execute()` on `Command` invokes the receiver's `action()` operation.

Different `Command` subclasses may invokes different `action()` operations.

The class structure of the command pattern



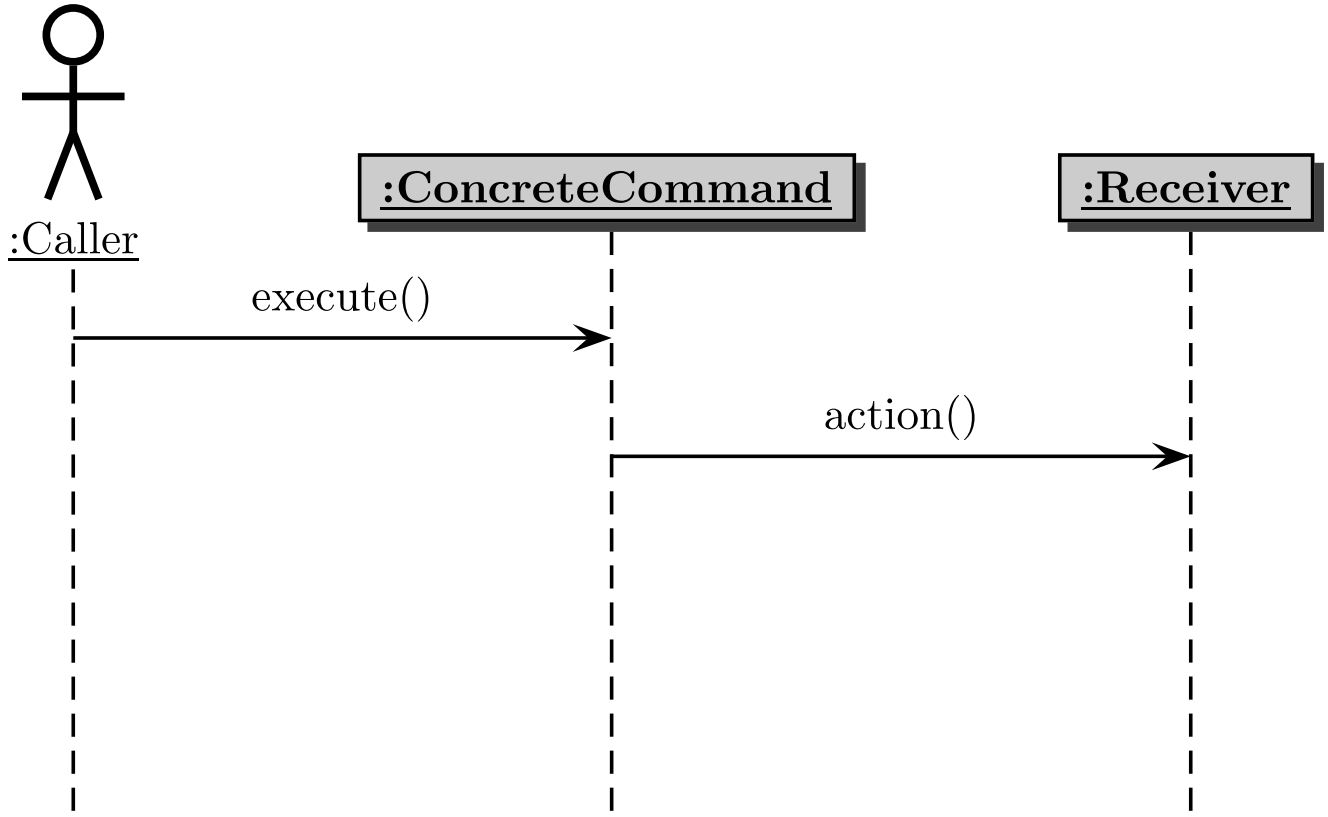
Call of `execute()` on `Command` invokes the receiver's `action()` operation.

Not necessary each `Command` subclass invokes same `action()` operation.

How could the pattern be altered to avoid encoding the receiver association as attribute?

Execution of a command

General



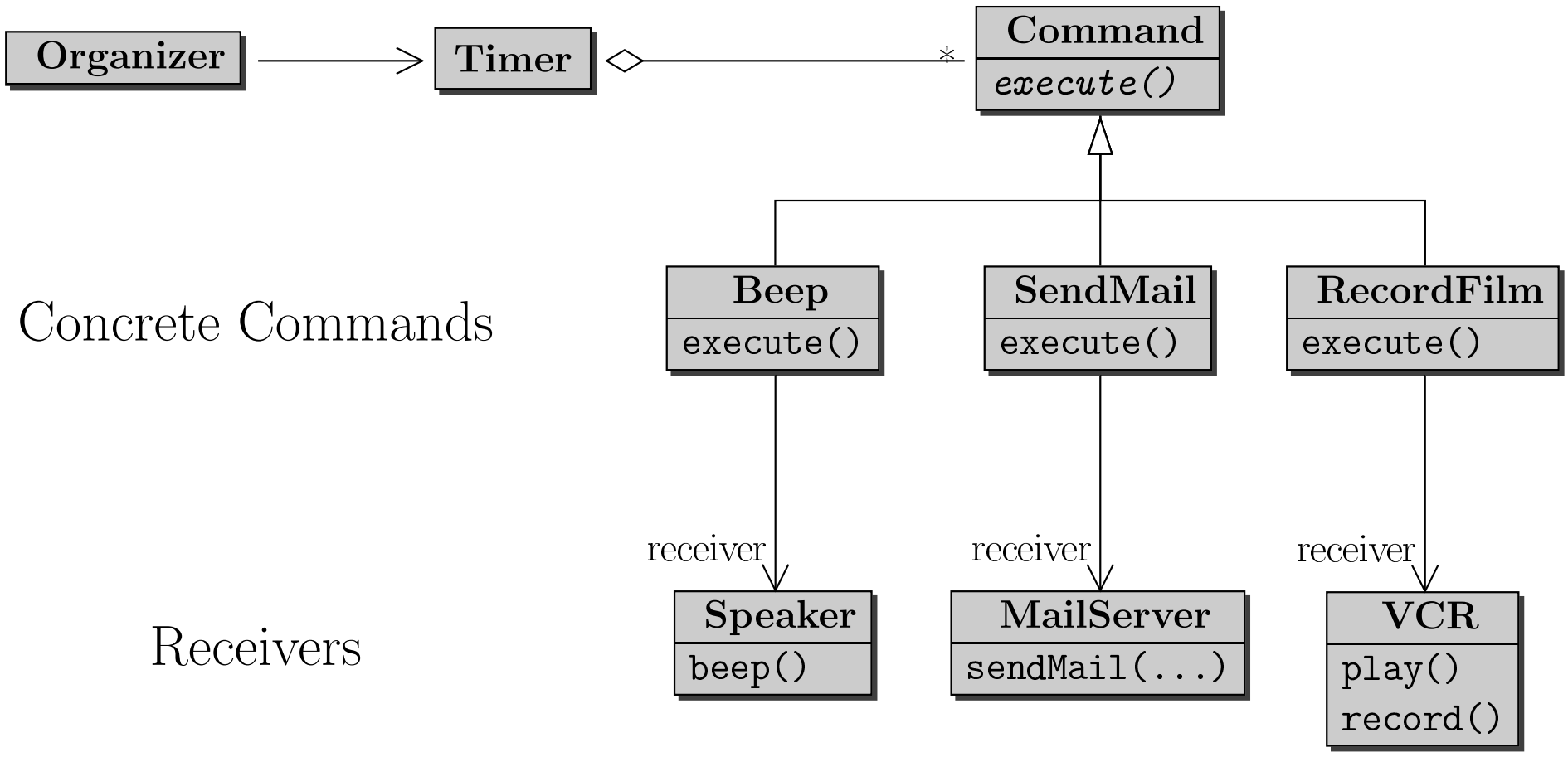
Example: Timer triggered actions

One feature of the organizer in the former examples is to execute actions, when a certain date (time) has been reached.

For example:

- **beep 5 minutes before a meeting**
- **send to all participants of a meeting an invitation e-mail a week before it is scheduled**
- **record a TV film**

Example (cont'd): Timer triggered actions



Literature

- **»Design Patterns: Elements of Reusable Object-Oriented Software«, by E. Gamma, R. Helm, R. Johnson and J. Vlissides, Addison-Wesley (1995)**

- **Lecture Notes »Softwaretechnik«, W.F. Tichy and G. Goos (WS 1998/99)**

- **Lecture Notes »Design Patterns Overview«, Rob Kremer (2002),**
  `http://sern.ucalgary.ca/courses/SENG/443/W02/patterns/index.html`

Design Patterns ...

Libraries For Designing System

Design Patterns ...

Libraries For Designing System

- **document design solutions on an abstract level**

Design Patterns ...

Libraries For Designing System

- **document design solutions on an abstract level**
- **prevent reinvention of the wheel**

Design Patterns ...

Libraries For Designing System

- **document design solutions on an abstract level**
- **prevent reinvention of the wheel**
- **support communication between developers**

Design Patterns ...

Libraries For Designing System

- **document design solutions on an abstract level**
- **prevent reinvention of the wheel**
- **support communication between developers**
- **make expert knowledge usable for beginners**

Design Patterns ...

Libraries For Designing System

- document design solutions on an abstract level
- prevent reinvention of the wheel
- support communication between developers
- make expert knowledge usable for beginners

but:

They don't make your design decisions.