Objectgeoriënteerd ontwerp

Command pattern

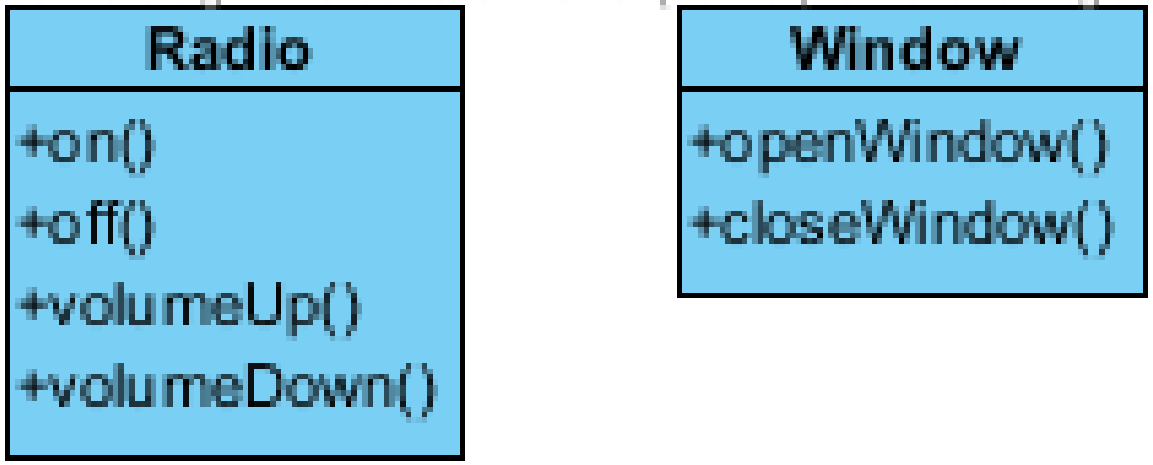
Objectgeoriënteerd ontwerp

Situatie: spraakbediening

- herkent twee commando's: "up" en "down"
- "up" en "down" moeten verschillende apparaten kunnen bedienen

Objectgeoriënteerd ontwerp

Klassen voor de objecten



Probleem: totaal verschillende implementaties van de te bedienen klassen

Objectgeoriënteerd ontwerp

Belangrijke doelstellingen

- Bediening moet objecten kunnen aansturen.
- Bediening mag niets afweten van concrete objecten zoals radio's, vensters, ...
- Bediening moet objecten kunnen aansturen die we nu nog niet kennen.

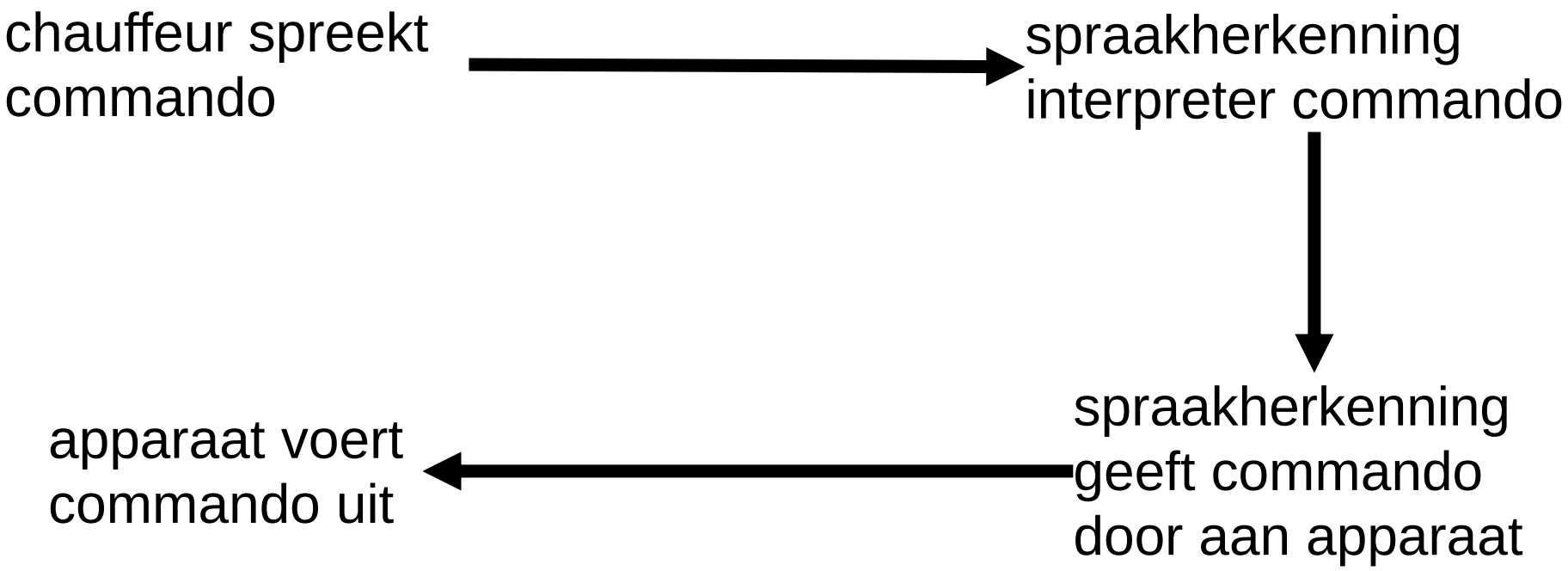
Oplossing

- Command pattern
- Command-objecten creëren die een gemeenschappelijke interface implementeren.
- Operaties om command-objecten aan commando's te verbinden.

Objectgeoriënteerd ontwerp

Het Command Pattern schermt een aanroep af door middel van een object, waarbij je verschillende aanroepen in verschillende objecten kunt opbergen, in een queue kunt zetten of op schijf kunt bewaren, en undo-operaties kunt ondersteunen.

Spraakgestuurde wagen

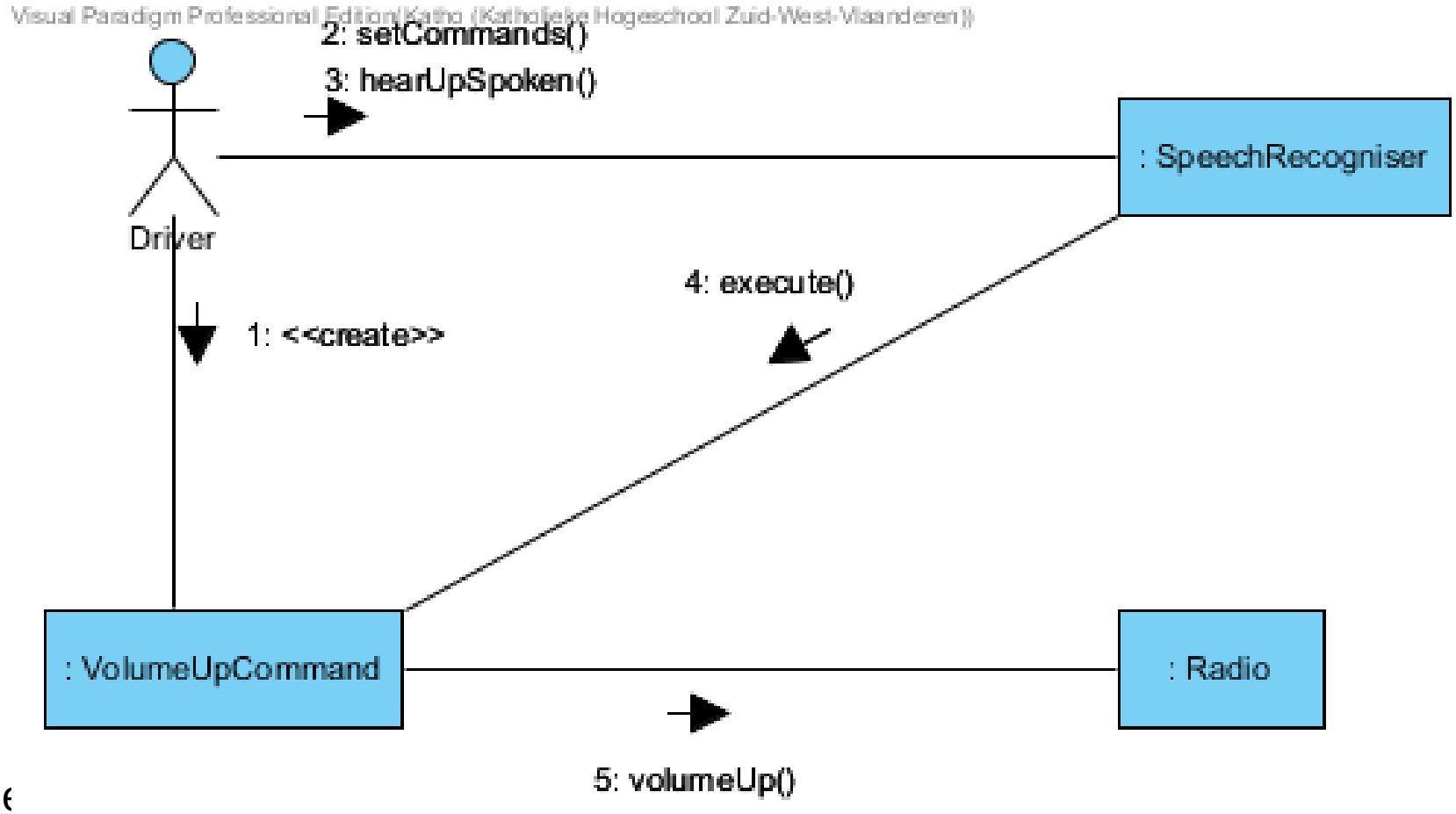


Objectgeoriënteerd ontwerp

Configuratie

Objectgeoriënteerd ontwerp

Gebruik



Objectgeoriënteerd ontwerp

Elk zijn verantwoordelijkheid

- Command: een verzoek om een actie uit te voeren. Weet wie de actie moet uitvoeren.
- SpeechRecogniser: geeft commando's door zonder uitvoerder te kennen.
- Radio, venster: voert commando's uit.

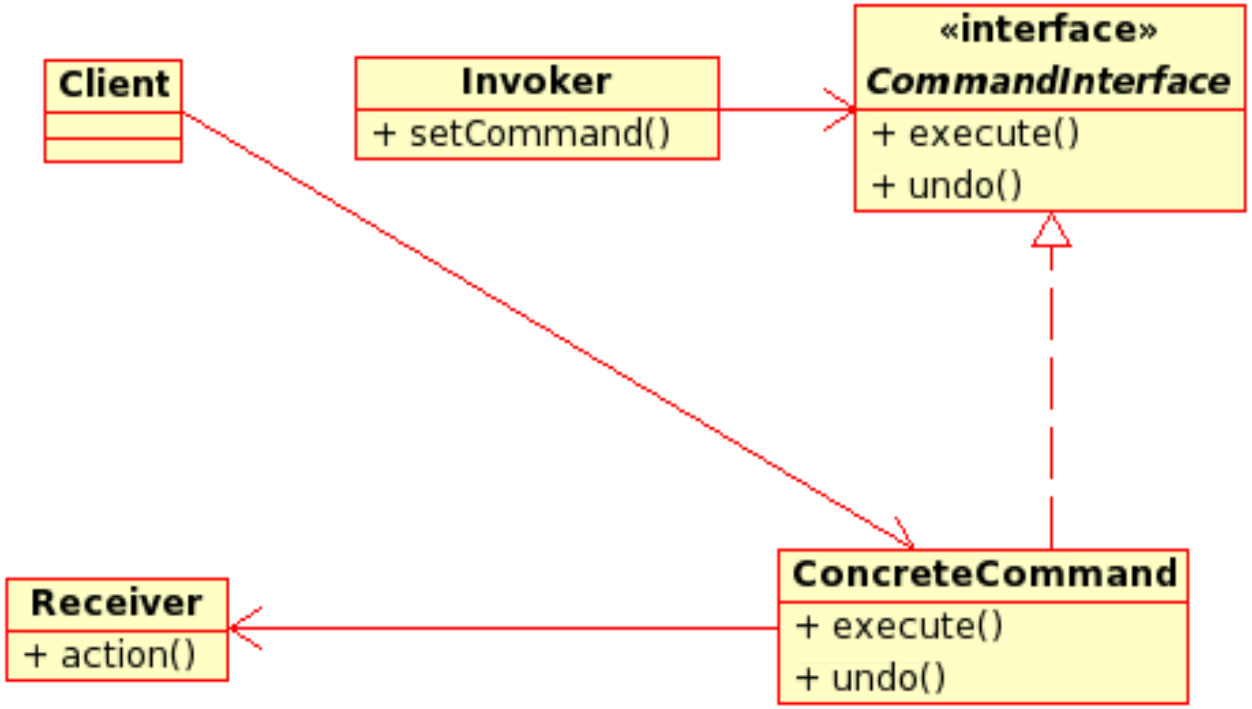
Ontkoppeling

- De speechrecogniser communiceert nooit rechtstreeks met de radio, maar geeft enkel commando's door.
- De radio is volledig ontkoppeld van de speechrecogniser. Hij krijgt zijn orders via een commando.

Rollen in het Command-patroon

- Client: de chauffeur.
- Invoker: de speechrecogniser
- Receiver: radio, venster

Command Pattern



Objectgeoriënteerd ontwerp

uitbreiding: null-command

- toepassing van het Null Object pattern
- een commando dat niets doet.
- als de speechrecogniser geen apparaat mag bedienen
- Zo hoeft invoker niet te testen of er wel een commando is.

Oefenen

- Voeg een klasse Fan toe aan het voorbeeld uit het boek, met bijhorende commando's. De Fan heeft vier standen: uit, 1, 2 en 3.
- Voeg een null-commando toe aan het voorbeeld uit het boek.

Objectgeoriënteerd ontwerp

uitbreiding: undo

- Implementeer een undo-methode bij elk commando.
- Voeg undo-operatie toe aan speechrecogniser

Undo-stijlen

- “het omgekeerde doen”: voer een commando uit dat het tegenovergestelde is van wat je ongedaan wil maken.
- “terug naar vorige toestand”: onthoud bij elk commando de begintoestand. Herstel die bij undo.

Meervoudige undo?

- Een stack van undo-commando's bijhouden.
- Bij elke execute() commando op de stack plaatsen.
- Bij elke undo(): pop bovenste item van de stack en voer undo() uit.

Objectgeoriënteerd ontwerp

uitbreiding: macrocommando's

- Een Command-object dat meer dan één actie uitvoert.
- macrocommand bevat een reeks van Command-objecten.
- Bij execute() ==> voer alle Command-objecten in de rij één voor één uit.

Objectgeoriënteerd ontwerp

```
public class MacroCommand implements Command {
  Command[] commands;

  public MacroCommand(Command[] command) {
    this.commands = commands;
  }

  public void execute() {
    for (int i = 0; i < commands.length; i++) {
      commands[i].execute; }
  }
}
```

Objectgeoriënteerd ontwerp

variatie: “smart” command objects

- Command Objects zonder “receiver”.
- Voeren zelf alle logica uit.

Objectgeoriënteerd ontwerp

uitbreiding: wachtrijen

- Command-objecten worden gecreëerd en in een wachtrij (queue) gezet.
- Andere threads halen de objecten uit de wachtrij en roepen “execute()” aan.

uitbreiding: logging

- Voeg store() en load() aan Command-objecten toe.
- Bij uitvoering: execute() + store()
- Bij herstel na crash: load() + execute()

Oefenen

- De speechrecogniser breidt uit: “up max” en “down min”. Maak commando’s om de radio zo luid mogelijk en zo stil mogelijk te zetten (met undo!)
- Implementeer meervoudige undo