CS 535 Object-Oriented Programming & Design
Spring Semester, 2003
Doc 15 Observer

Contents

Observer Pattern ................................................................. 2
Example .................................................................................. 4
ValueModels ........................................................................... 11

References

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, pp18-20, 293-304


Copyright ©, All rights reserved. 2003 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Observer Pattern

Define a one-to-many dependency between objects

When one object changes state all its dependents are notified and updated automatically

Publish-subscribe
Event Handler
Dependence mechanism

```
Subject
  addDependent:
  removeDependent:
  changed:

ConcreteSubject
  changeState:
  subjectState

Observer
  update:

ConcreteObserver
  update:
  observerState
```
Basic Steps

• Registration

subject addDependent: observer

• Notification

self changed.
self changed: #someValue

• Update

define update: aSymbol
Example

Classes

Boiler – maintains a pressure
Gauge – displays the pressure
SafetyValve – Releases pressure if too high
Boiler

Smalltalk defineClass: #Boiler
   superclass: #{UI.Model}
   indexedType: #none
   private: false
   instanceVariableNames: 'pressure maximumPressure '
   classInstanceVariableNames: "
   imports: "
   category: 'CS535'

Boiler class methodsFor: 'instance creation'

new
   ^super new initialize

Boiler Instance Methods

addHeat
   self changePressureBy: 15

changePressureBy: aNumber
   pressure := pressure + aNumber.
   self changed: #pressure.
   pressure > maximumPressure ifTrue:[self changed: #overPressure]

initialize
   pressure := 10.
   maximumPressure := 25.

pressure
   ^pressure
Gauge

Smalltalk defineClass: #Gauge
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'subject '
  classInstanceVariableNames: ''
  imports: ''
  category: 'CS535'

Gauge class methodsFor: 'instance creation'

subject: aBoiler
  ^super new setSubject: aBoiler

Gauge Instance Methods

setSubject: aBoiler
  subject := aBoiler.
  subject addDependent: self

update: aSymbol
  "Display pressure"
  aSymbol = #pressure
  ifTrue:
    [Transcript
      print: subject pressure;
      cr;
      flush]
**SafetyValue**

Smalltalk defineClass: #SafetyValue
superclass: #{Core.Object}
indexedType: #none
private: false
instanceVariableNames: 'boiler'
classInstanceVariableNames: 
imports: 
category: 'CS535'

SafetyValue class methodsFor: 'instance creation'

subject: aBoiler
^super new setSubject: aBoiler

**SafetyValue Instance Methods**

setSubject: aBoiler
boiler := aBoiler.
boiler addDependent: self

update: aSymbol
aSymbol = #overPressure ifTrue:[boiler changePressureBy: -10]
Sample Use

boiler := Boiler new.
gauge := Gauge subject: boiler.
valve := SafetyValue subject: boiler.
boiler
  addHeat;
  addHeat;
  addHeat;
  addHeat

Output In Transcript

25
40
30
20
35
25
40
30
20
Observer Pattern

Advantages

• Easy to add new observers

• Coupling between observer and subject is abstract

Disadvantages

• Hard to see how objects in system interact

• Sometimes inefficient
Object Verses Model

Object supports subject behavior

Do not need to make subject class (Boiler) subclass of Model

Using model is cleaner

Object’s dependants list can cause memory leaks

Object uses global dictionary to store dependents

Subject must release its observers/dependents
ValueModels

Encapsulates model(subject) behavior

A model with two messages to change value

value
value:

ValueModel>>onChangeSend: aSymbol to: anObject

“Arrange to send aSymbol to anObject when receiver changes”

a asValue

Creates a ValueHolder, subclass of ValueModel, on a
Example

Smalltalk defineClass: #Boiler
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'pressure '
  classInstanceVariableNames: "
  imports: "
  category: 'CS535'

Boiler class methodsFor: 'instance creation'

new
  ^super new initialize

  Boiler Instance Methods

addHeat
  self changePressureBy: 15

changePressureBy: aNumber
  pressure value: (pressure value + aNumber)

initialize
  pressure := 10 asValue.

pressure
  ^pressure value

pressureHolder
  ^pressure
Smalltalk defineClass: #Gauge
  superclass: #{Core.Object}
  indexedType: #none
  private: false
  instanceVariableNames: 'value '
  classInstanceVariableNames: ''
  imports: ''
  category: 'CS535'

Gauge class methodsFor: 'instance creation'

subject: aValue
  ^super new setSubject: aValue

  **Gauge Instance Methods**

display
  Transcript
    print: value value;
    cr;
    flush

setSubject: aValue
  value := aValue.
  value onChangeSend: #display to: self
Smalltalk defineClass: #SafetyValue
  superclass: #{Core.Object}
  indexedType: none
  private: false
  instanceVariableNames: 'boiler maximumPressure '
  classInstanceVariableNames: "
  imports: "
  category: 'CS535'

SafetyValue class methodsFor: 'instance creation'

subject: aBoiler
  ^super new setSubject: aBoiler

  **SafetyValue Instance Methods**

checkPressure
  boiler pressure > maximumPressure ifTrue: [self reducePressure]

reducePressure
  boiler changePressureBy: -10

setSubject: aBoiler
  boiler := aBoiler.
  boiler pressureHolder onChangeSend: #checkPressure to: self.
  maximumPressure := 25
Inheritance & Composition

Inheritance

• Subclass uses operations/data of parent

• White-box reuse

• Subclass has access to parents inner operations/data

• Boiler in first example uses inheritance

Object Composition

• Class uses operations of instance variable

• Black box reuse

• Boiler in second examples uses object composition
Inheritance Advantages

Inheritance is defined at compile time

Simple to use

Subclasses can modify behavior of parent class

Object Composition

Can change at runtime

Enforces encapsulation