CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2001
Doc 22 Mediator

Contents

Mediator ........................................................................................................... 2
Structure ........................................................................................................ 2
Motivating Example ....................................................................................... 4
Issues ............................................................................................................. 6

References

Design Patterns: Elements of Resuable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, Addison Wesley, 1995, pp. 273-282

Copyright ©, All rights reserved. 2001 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/pl.shtml) license defines the copyright on this document.
**Mediator**

A mediator is responsible for controlling and coordinating the interactions of a group of objects (not data structures)

**Structure**

**Classes**

![Diagram of Mediator and Colleague classes]

**Objects**

![Diagram of object interactions]

- **aColleague**: mediator
- **aConcreteMediator**: mediator
- **aConcreteColleague1**: mediator
- **aConcreteColleague2**: mediator
Participants

Mediator

Defines an interface for communicating with Colleague objects

ConcreteMediator

Implements cooperative behavior by coordinating Colleague objects

Knows and maintains its colleagues

Colleague classes

Each Colleague class knows its Mediator object

Each colleague communicates with its mediator whenever it would have otherwise communicated with another colleague
Motivating Example
Dialog Boxes

Objects

- aClient
- aFontDialogDirector
- aListBox
- aButton
- anEntryField

Interaction

Mediator

- aClient
- aFontDialogDirector

Colleagues

- aListBox
- anEntryField

Show Dialog()

Widget

Changed()

GetSelection()

SetText()

How does this differ from a God Class?
When to use the Mediator Pattern

When a set of objects communicate in a well-defined but complex ways.

When reusing an object is difficult because it refers to and communicates with many other objects.

When a behavior that’s distributed between several classes should be customizable without a lot of subclassing.
Issues
How do Colleagues and Mediators Communicate?

1) Explicit methods in Mediator

class DialogDirector
{
    private Button ok;
    private Button cancel;
    private ListBox courses;

    public void ListBoxItemSelected() { blah }

    public void ListBoxScrolled() { blah }
    etc.
}

2) Generic change method

class DialogDirector {
    private Button ok;
    private Button cancel;
    private ListBox courses;

    public void widgetChanged( Object changedWidget) {
        if ( changedWidget == ok ) blah
        else if ( changedWidget == cancel ) more blah
        else if ( changedWidget == courses ) even more blah
    }
}
3) Generic change method overloaded

class DialogDirector
{
    private Button ok;
    private Button cancel;
    private ListBox courses;

    public void widgetChanged( Button changedWidget)
    {
        if ( changedWidget == ok )
            blah
        else if ( changedWidget == cancel )
            more blah
    }

    public void widgetChanged( ListBox changedWidget)
    {
        now find out how it changed and
        respond properly
    }
}
Differences from Facade

Facade does not add any functionality, Mediator does

Subsystem components are not aware of Facade

Mediator's colleagues are aware of Mediator and interact with it