

Software Design
CSE 335, Spring 2006

Object-oriented programming: The Mediator Pattern

Stephen Wagner

Michigan State University

A Simple Example

- Three objects : a scrollbar, a left button, and a right button
- Clicking the buttons moves the scroll bar
- Each button should only be active if the scrollbar can be moved in the appropriate direction



- Assume that buttons and scrollbars can have multiple listeners.
- How to implement?

Another Example

Filemasks: *.c, *.h, [Mm]akefile

Button.h
Button.cc
ButtonListener.h
Catalog.cc
Catalog.h
FontDirectory
Garbage/
IvalSlider.cc
IvalSlider.h
IvalText.cc
IvalText.h

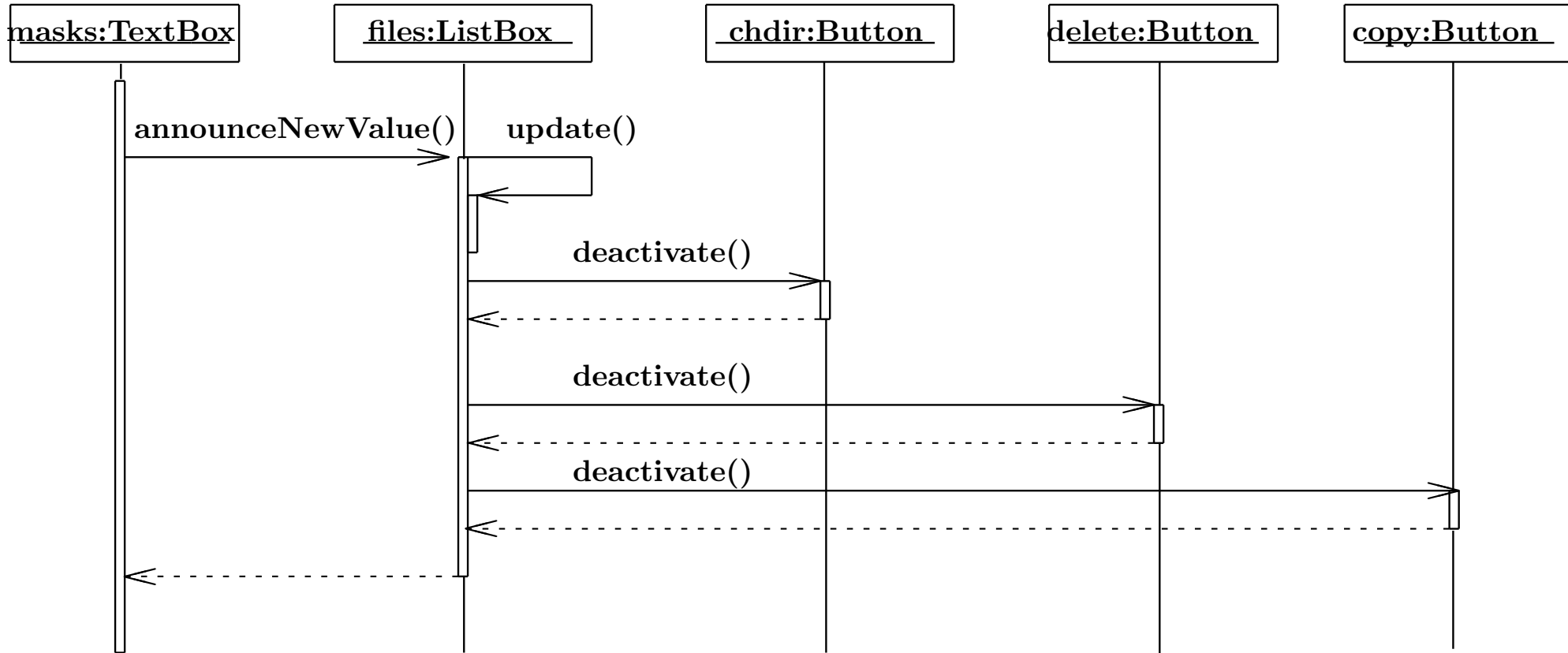
Delete Copy Chdir

Synchronization Collaboration

- What happens when we enter a new mask?

Synchronization Collaboration

- What happens when we enter a new mask?

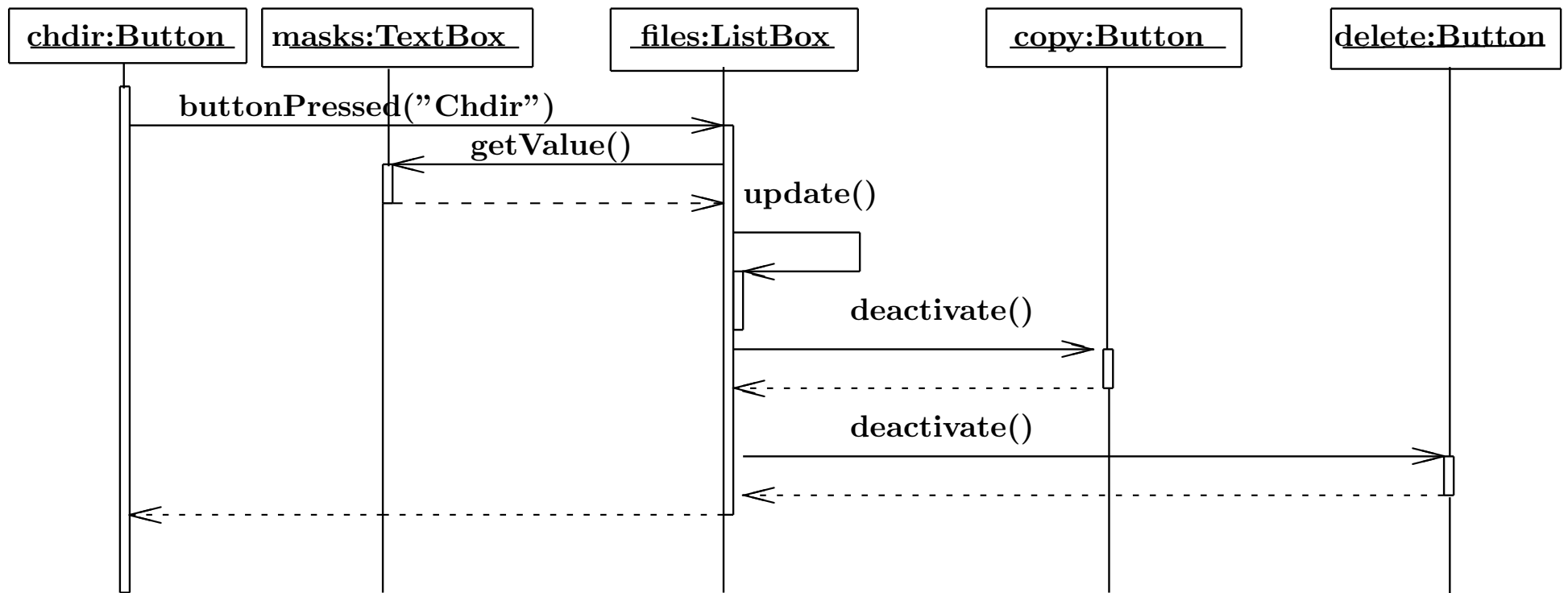


Synchronization Collaboration

- What happens when we change directory?

Synchronization Collaboration

- What happens when we change directory?



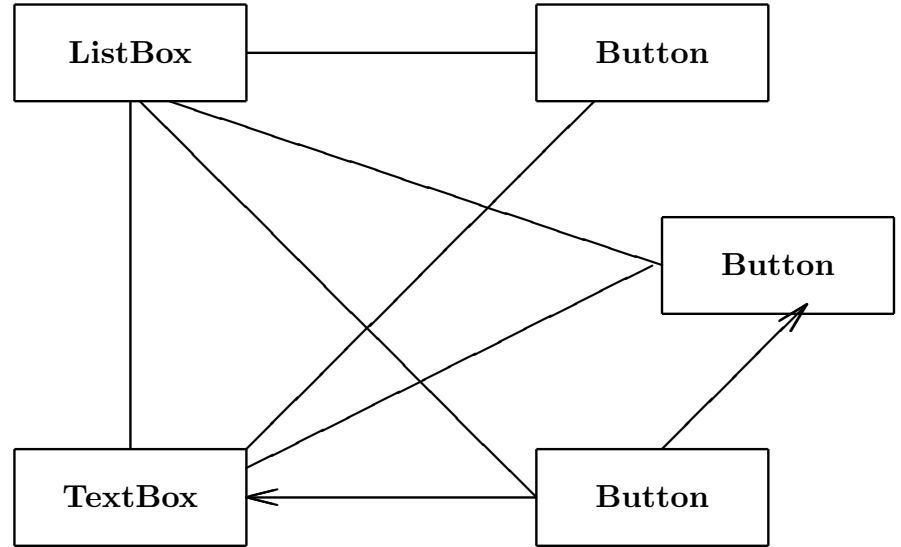
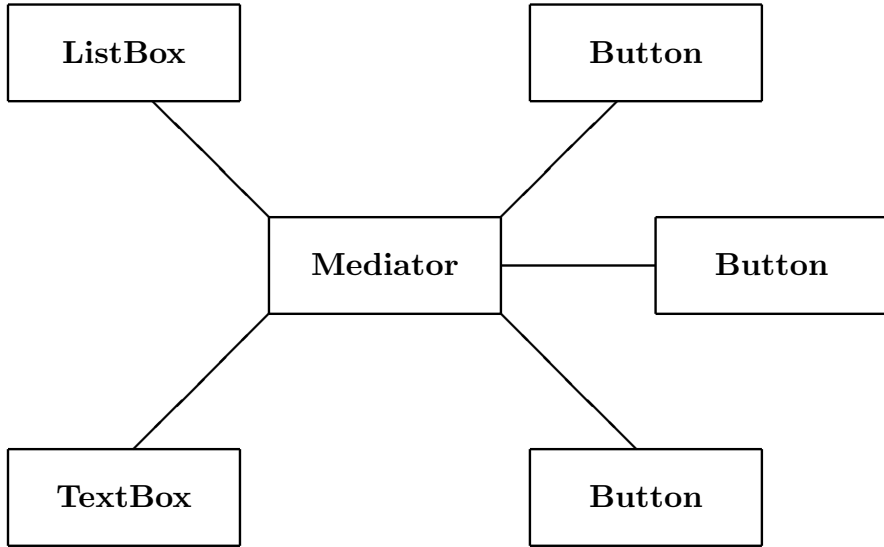
Problem

- Assembly from reusable collaborations can lead to *tight coupling* between application objects
- Disadvantages:
 - The adaptor classes will be overly complicated and not reusable
 - Complicates configuration
 - Distributes “invariant logic” among a large class of entities
 - Complicates collaboration synthesis due to large number of related collaborations (update cycles)

Solution: Mediator

- A single object that encapsulates how a set of objects interact
- Advantages:
 - Promotes *loose coupling* by keeping objects from referring to each other explicitly
 - Central point for cyclic updates
 - Central manifestation of “invariant logic”

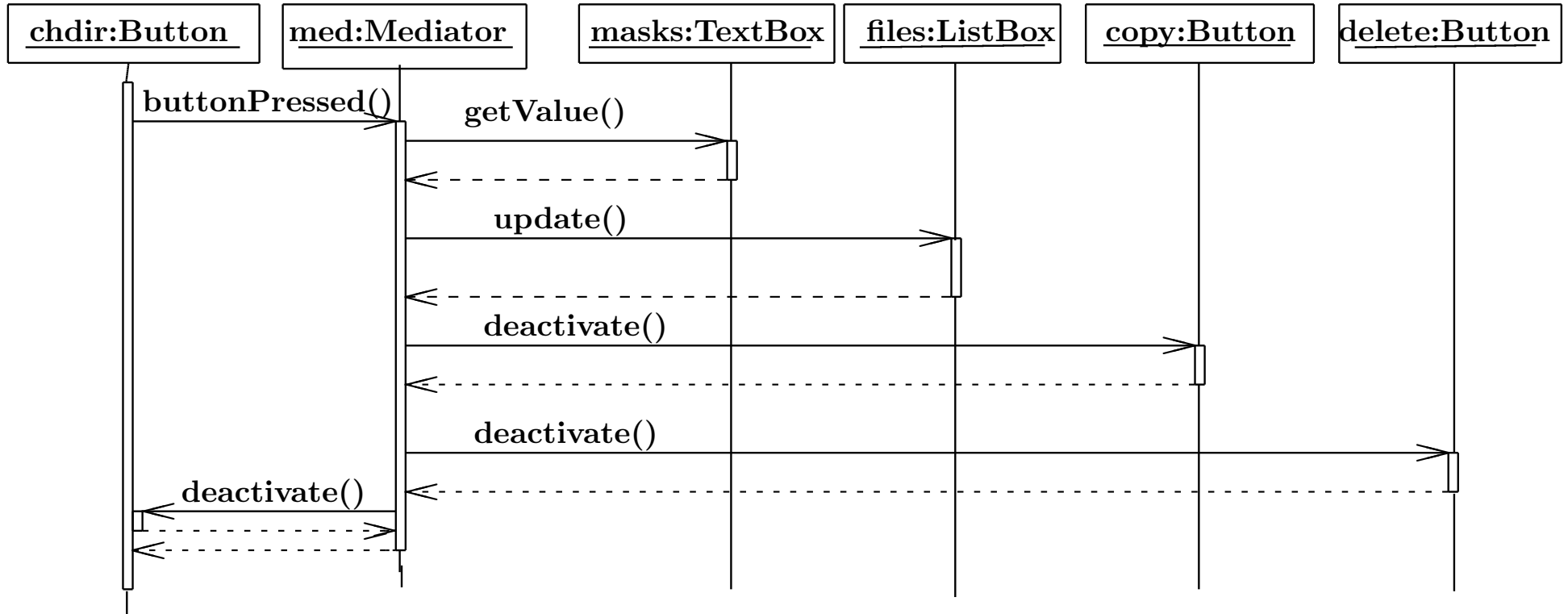
Mediated vs. Distributed Collaboration



Mediated vs. Distributed Collaboration

- The Mediator class is designed to solve a specific problem
 - It will be tightly coupled with all of the other objects in the collaboration
 - It will generally not be reusable
- The other objects will be unadapted and reusable
- In the distributed design
 - There will be a large number of adaptor classes
 - None of these classes will be reusable

Mediated Collaboration



Mediator Synthesizes Listener Roles

```
class Director : public ButtonListener,
                public TextboxListener,
                public ListBoxListener
{
    public:
    Director();

    void buttonPressed(const string &);
    void notify(const string &);
    void entryAction(const string &);

    private:
    Button *copy;
    Button *delete;
    ListBox *fileList;
    TextBox *fileMasks;    }
```

The Mediator Class

```
class Director : public ButtonListener,  
                public TextBoxListener,  
                public ListBoxListener
```

- The Mediator class implements a number of interfaces
- The Mediator class is not an adaptor class
 - it does not adapt an existing class
- The Mediator class will contain private members
 - It needs to have pointers to all the objects in the collaboration
 - It could have additional members
 - It could also have additional member functions

Mediator Localizes Configuration Code

```
Director::Director()  
{  
    ok = new Button();  
    cancel = new Button();  
    fileList = new ListBox();  
    fileMasks = new TextBox();  
  
    //  
}
```

- It usually makes sense to have the mediator create the objects it mediates
- Who should register with the buttons?

Mediator Localizes Configuration Code

```
Director::Director()  
{  
    ok = new Button();  
    cancel = new Button();  
    fontList = new ListBox();  
    fontName = new TextBox();  
  
    ok.registerListener(this);  
    cancel.registerListener(this);  
    fontList.registerListener(this);  
    fontName.registerListener(this);  
}
```

- Why do objects not register with the mediator?

What's Wrong Here?

```
Director::Director()  
{  
    Button ok;  
    Button cancel;  
    ListBox fontList;  
    TextBox fontName;  
  
    ok.registerListener(this);  
    cancel.registerListener(this);  
    fontList.registerListener(this);  
    fontName.registerListener(this);  
}
```

Mediators and Observers

- The Mediator is implemented as an Observer
 - The Mediator observes all of the other objects in the collaboration
 - None of the objects need to know directly about the Mediator or any of the other objects
- The “Design Patterns” book discusses another approach where each other class used in the collaboration has direct knowledge of the Mediator
- In C++ the observer approach is preferred