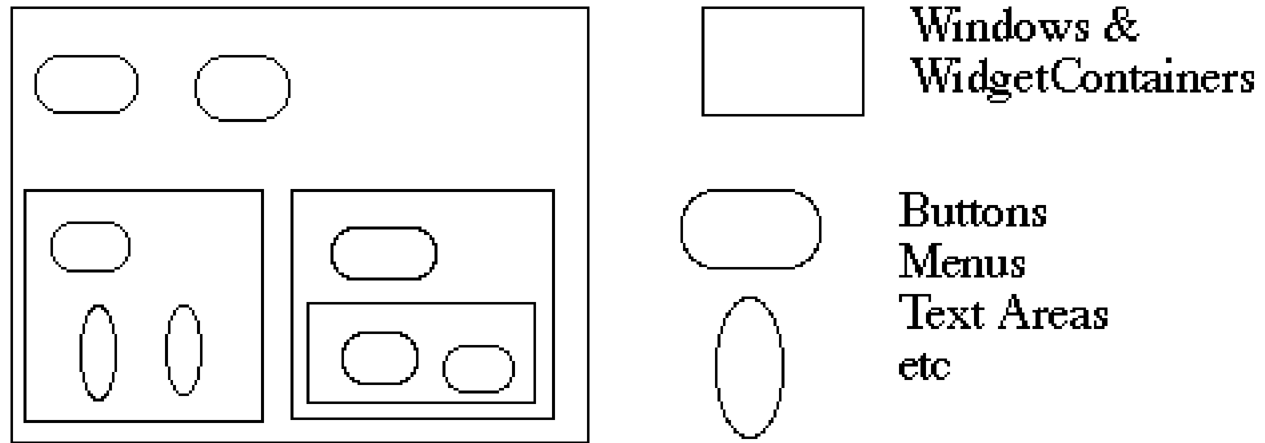
CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2011
Doc 17 Composite, Builder & Mediator
April 20, 2011

Copyright ©, All rights reserved. 2011 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.

Composite

Composite Motivation



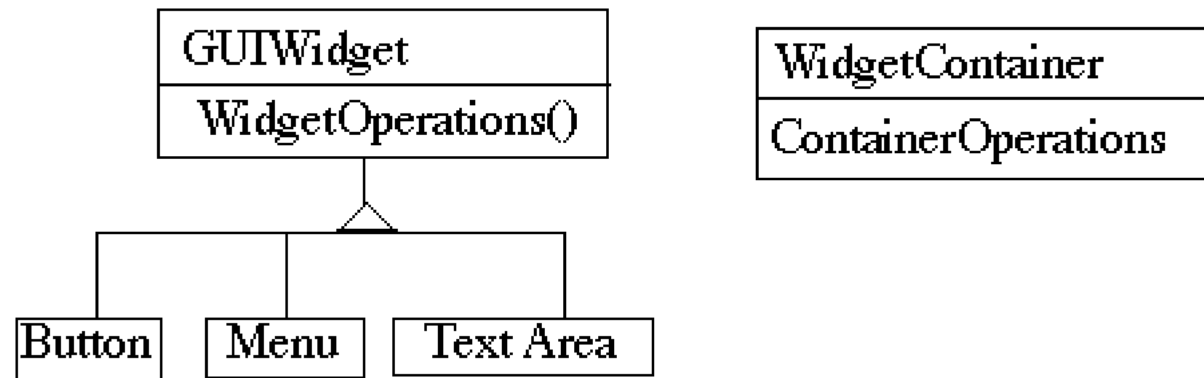
How does the window hold and deal with the different items it has to manage?

Widgets are different that WidgetContainers

Bad News

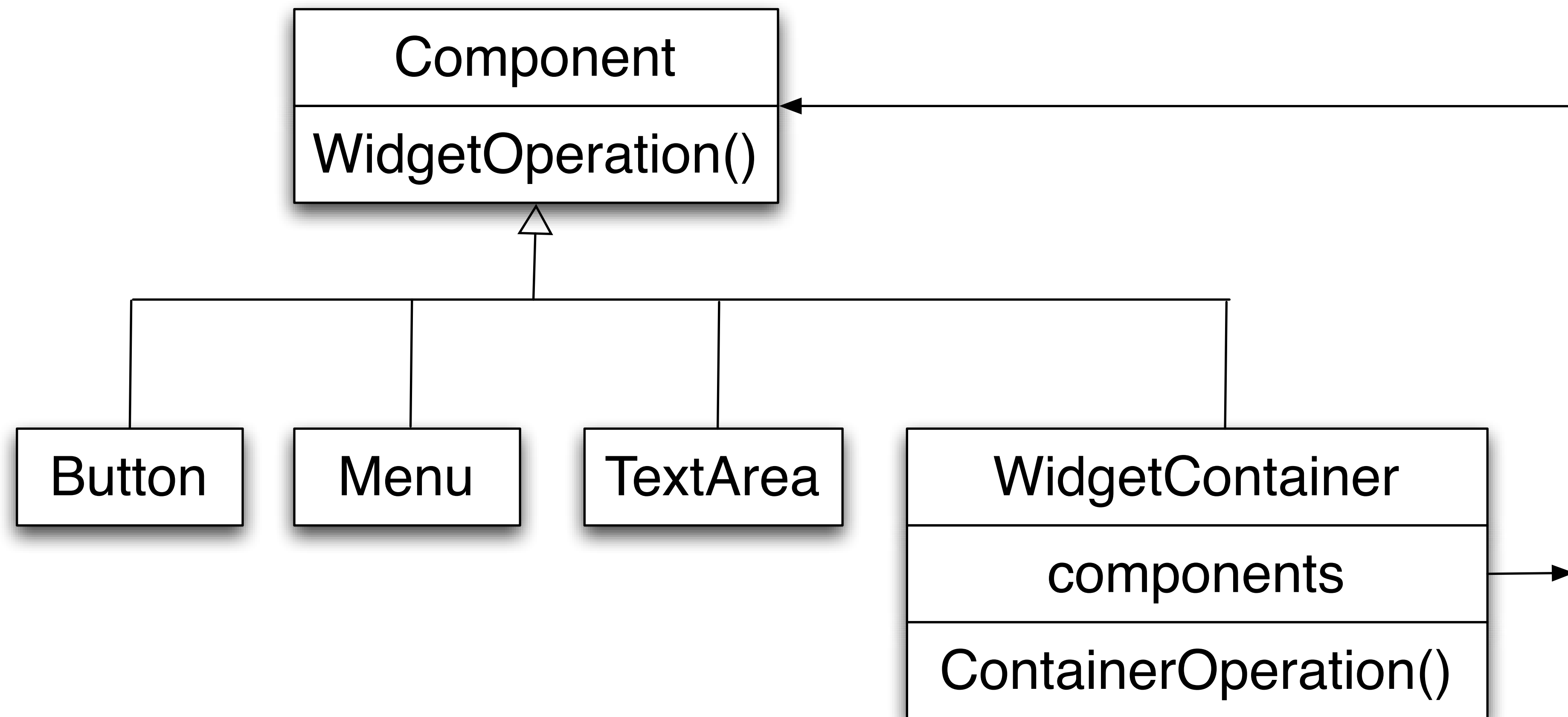
```
class Window {
    Buttons[] myButtons;
    Menus[] myMenus;
    TextAreas[] myTextAreas;
    WidgetContainer[] myContainers;

    public void update() {
        if ( myButtons != null )
            for ( int k = 0; k < myButtons.length(); k++ )
                myButtons[k].refresh();
        if ( myMenus != null )
            for ( int k = 0; k < myMenus.length(); k++ )
                myMenus[k].display();
        if ( myTextAreas != null )
            for ( int k = 0; k < myButtons.length(); k++ )
                myTextAreas[k].refresh();
        if ( myContainers != null )
            for ( int k = 0; k < myContainers.length(); k++ )
                myContainers[k].updateElements();
        etc.
    }
    public void fooOperation(){
        if (myButtons != null)
        etc.
```

An Improvement




```
class Window {
    GUIWidgets[] myWidgets;
    WidgetContainer[] myContainers;

    public void update(){
        if ( myWidgets != null )
            for ( int k = 0; k < myWidgets.length(); k++ )
                myWidgets[k].update();
        if ( myContainers != null )
            for ( int k = 0; k < myContainers.length(); k++ )
                myContainers[k].updateElements();
        etc.
    }
}
```

Composite Pattern



Composite Pattern

Component implements default behavior for widgets when possible

Button, Menu, etc overrides Component methods when needed

WidgetContainer will have to overrides all widgetOperations

```
class WidgetContainer {
	Component[] myComponents;

	public void update() {
		if ( myComponents != null )
			for ( int k = 0; k < myComponents.length(); k++ )
				myComponents[k].update();
	}
}
```

Issue - WidgetContainer Operations

Should the WidgetContainer operations be declared in Component?

Pro - Transparency

Declaring them in the Component gives all subclasses the same interface

All subclasses can be treated alike. (?)

Con - Safety

Declaring them in WidgetContainer is safer

Adding or removing widgets to non-WidgetContainers is an error

One out is to check the type of the object before using a WidgetContainer operation

Issue - Parent References

```
class WidgetContainer
        {
        Component[] myComponents;

        public void update() {
                if ( myComponents != null )
                        for ( int k = 0; k < myComponents.length(); k++ )
                                myComponents[k].update();
        }

        public add( Component aComponent ) {
                myComponents.append( aComponent );
                aComponent.setParent( this );
        }
}

class Button extends Component {
        private Component parent;
        public void setParent( Component myParent) {
                parent = myParent;
        }

        etc.
```

More Issues

Should Component implement a list of Components?

- The button etc. will have a useless data member

Child ordering is important in some cases

Who should delete components?

Applicability

Use Composite pattern when you want

To represent part-whole hierarchies of objects

Clients to be able to ignore the difference between compositions of objects and individual objects

Builder

Builder

Separate construction of a complex object from its representation

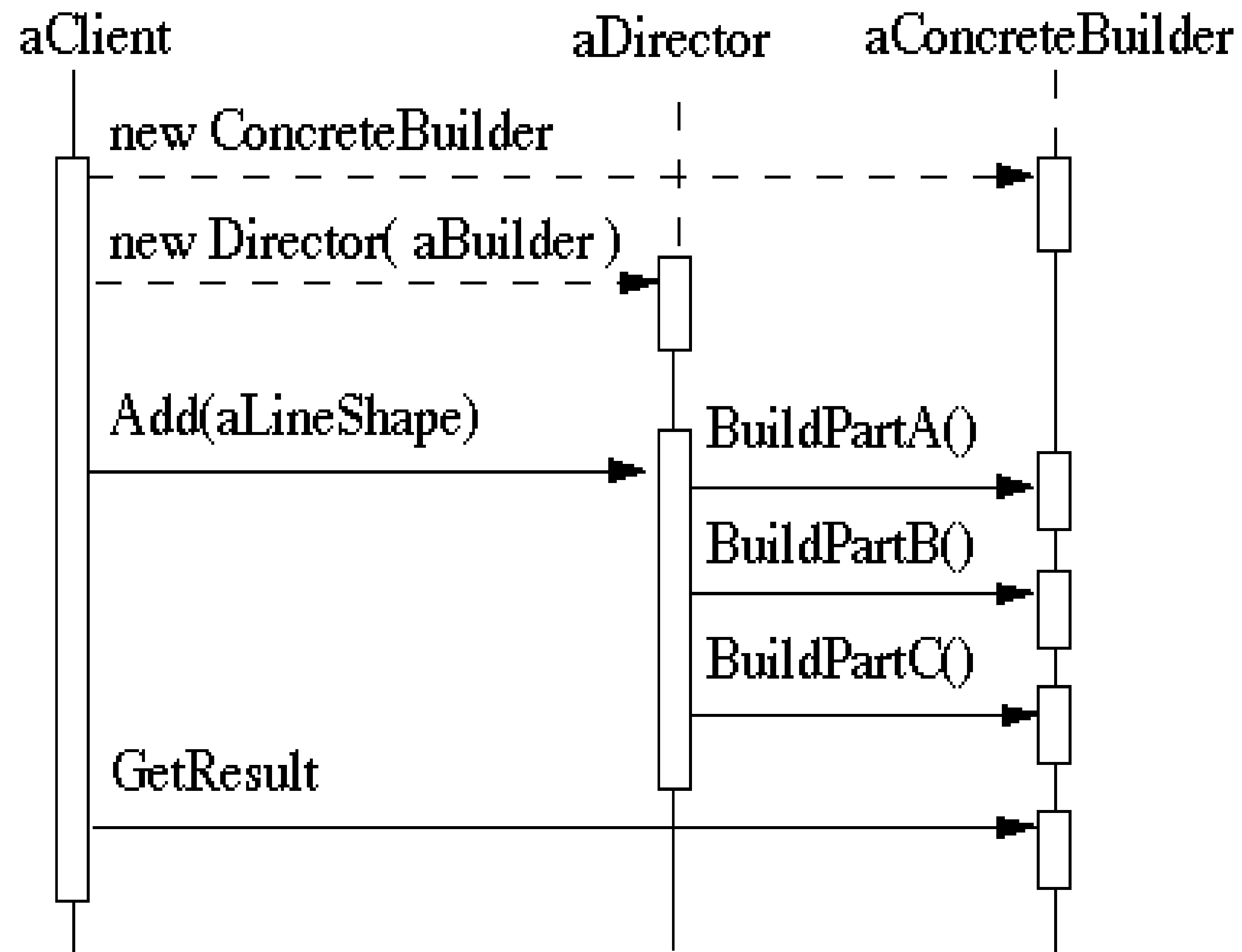
So same construction process can create different representations

Builder

Client

Director

Builder



RTF Converter

A word processing document has complex structure

How to convert Rich Text Format (RTF) to

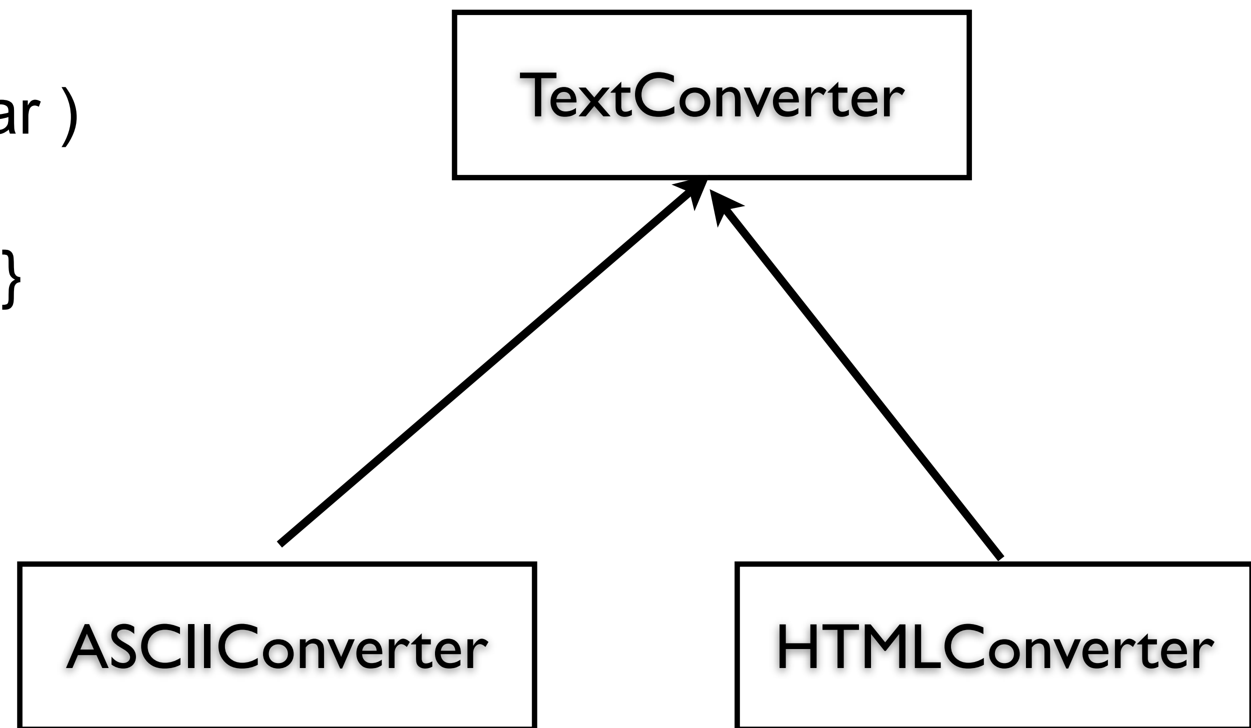
TeX
html
PDF
etc.

Pseudo Solution

```
class RTF_Reader {
  TextConverter builder;
  String RTF_Text;

  public RTF_Reader( TextConverter aBuilder, String RTFtoConvert ){
    builder = aBuilder;
    RTF_Text = RTFtoConvert;
    }

  public void parseRTF(){
    RTFTokenizer rtf = new RTFTokenizer( RTF_Text );

    while ( rtf.hasMoreTokens() ){
      RTFToken next = rtf.nextToken();

      switch ( next.type() ){
        case CHAR:    builder.character( next.char() ); break;
        case FONT:    builder.font( next.font() ); break;
        case PARA:    builder.newParagraph( ); break;
        etc.
      }
    }
  }
}
```

Builder Classes

```
abstract class TextConverter {
   public void character( char nextChar )
{  }
   public void font( Font newFont )   { }
   public void newParagraph() {}
}
```



Sample Program

```
main(){
  ASCII_Converter simplerText = new ASCII_Converter();
  String rtfText;

  // read a file of rtf into rtfText

  RTF_Reader myReader =
    new RTF_Reader( simplerText, rtfText );

  myReader.parseRTF();

  String myProduct = simplerText.getText();
}
```

The Hard Part

The builder interface

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement param="value">
    <FirstElement>
        Some Text
    </FirstElement>
    <SecondElement param2="something">
        Pre-Text <Inline>Inlined text</Inline> Post-text.
    </SecondElement>
</RootElement>
```

SAX - Builder Pattern

Director

XMLReader

Builder

ContentHandler

ContentHandler Interface

void characters(char[] ch, int start, int length)
    Receive notification of character data.
void endDocument()
    Receive notification of the end of a document.
void endElement(java.lang.String uri, java.lang.String localName, java.lang.String qName)
    Receive notification of the end of an element.
void endPrefixMapping(java.lang.String prefix)
    End the scope of a prefix-URI mapping.
void ignorableWhitespace(char[] ch, int start, int length)
    Receive notification of ignorable whitespace in element content.
void processingInstruction(java.lang.String target, java.lang.String data)
    Receive notification of a processing instruction.
void setDocumentLocator(Locator locator)
    Receive an object for locating the origin of SAX document events.
void skippedEntity(java.lang.String name)
    Receive notification of a skipped entity.
void startDocument()
    Receive notification of the beginning of a document.
void startElement(java.lang.String uri, java.lang.String localName, java.lang.String qName, Attributes atts)
    Receive notification of the beginning of an element.
void startPrefixMapping(java.lang.String prefix, java.lang.String uri)
    Begin the scope of a prefix-URI Namespace mapping.

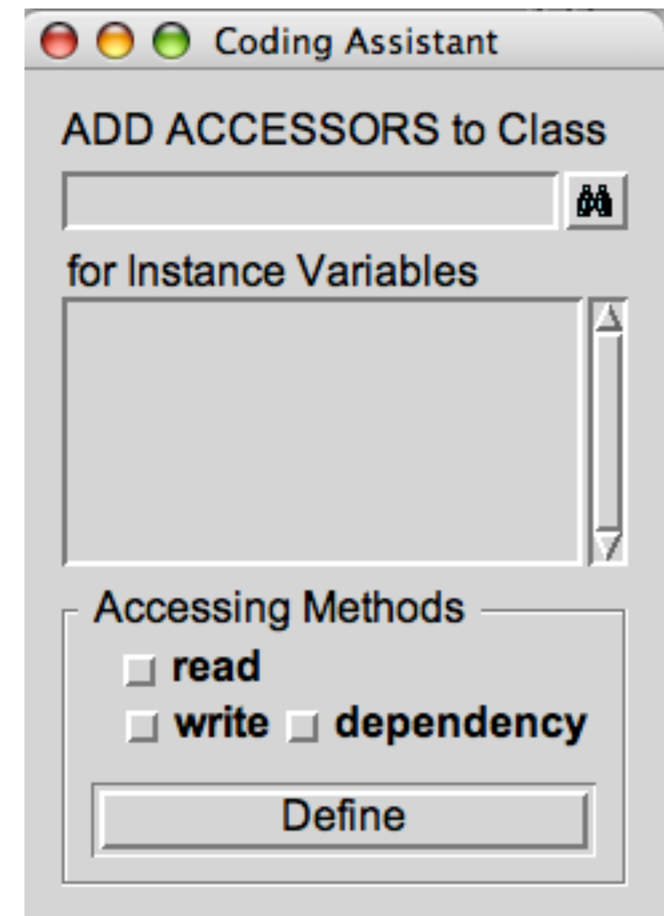
Simple API XML (SAX)

```
public static void main (String args[]) throws Exception {
 XMLReader director = XMLReaderFactory.createXMLReader();
 ContentHandler builder = new MySAXApp();
 director.setContentHandler(builder);
 director.setErrorHandler(builder);

 FileReader source = new FileReader("Foo.xml");
 director.parse(new InputSource(source));
 handler.getResult();
 }
```

Examples - VW Smalltalk

ClassBuilder
MenuBuilder
UIBuilder

UIBuilder




```
#(#{UI.FullSpec}
    #window:
    #(#{UI.WindowSpec}
      #label: #(#{Kernel.UserMessage} #key: #CodingAssistant
            #defaultString: 'Coding Assistant' #catalogID: #UIPainter)
      #min: #(#{Core.Point} 242 320 )
      #max: #(#{Core.Point} 242 320 )
      #bounds: #(#{Graphics.Rectangle} 279 140 521 460 ) )
    #component:
    #(#{UI.SpecCollection}
      #collection: #(
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 14 0 12 0 )
          #label: #(#{Kernel.UserMessage} #key: #ADDACCESSORSToClass
            #defaultString: 'ADD ACCESSORS to Class' #catalogID: #UIPainter) )
        #(#{UI.LabelSpec}
          #layout: #(#{Graphics.LayoutOrigin} 16 0 65 0 )
          #label: #(#{Kernel.UserMessage} #key: #forInstanceVariables
              #defaultString: 'for Instance Variables' #catalogID: #UIPainter) )
```

Strategy

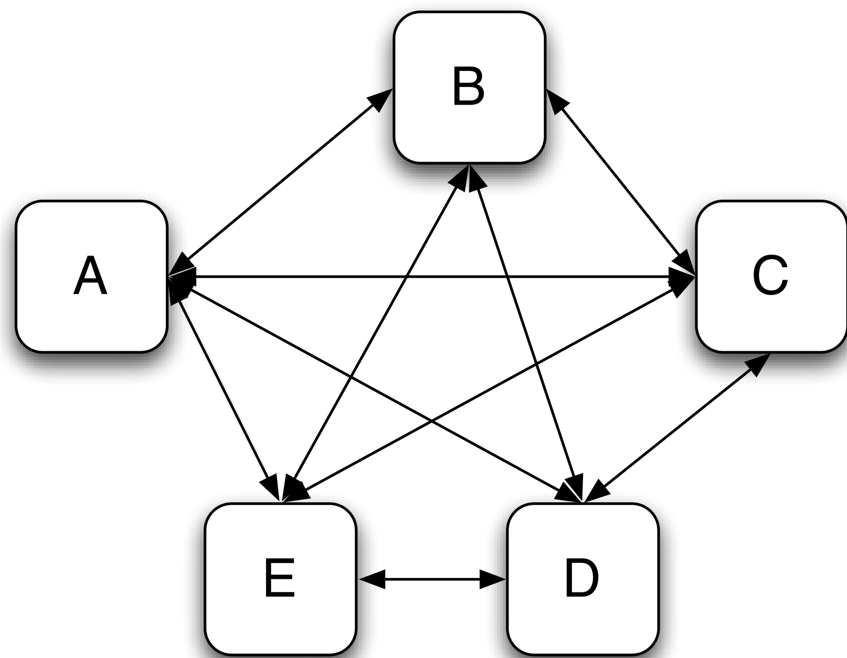
vs

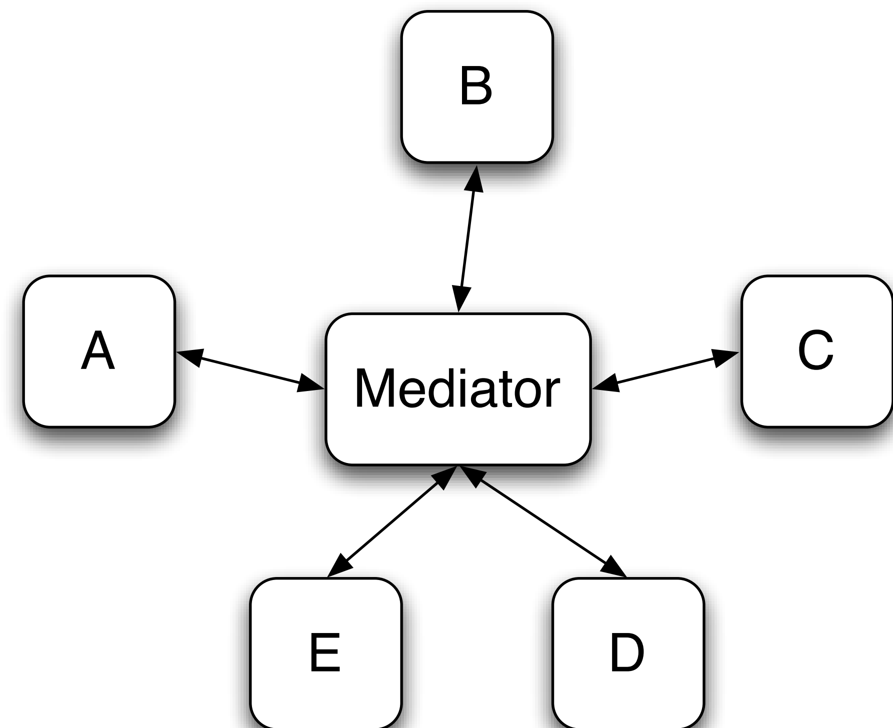
Builder

Mediator

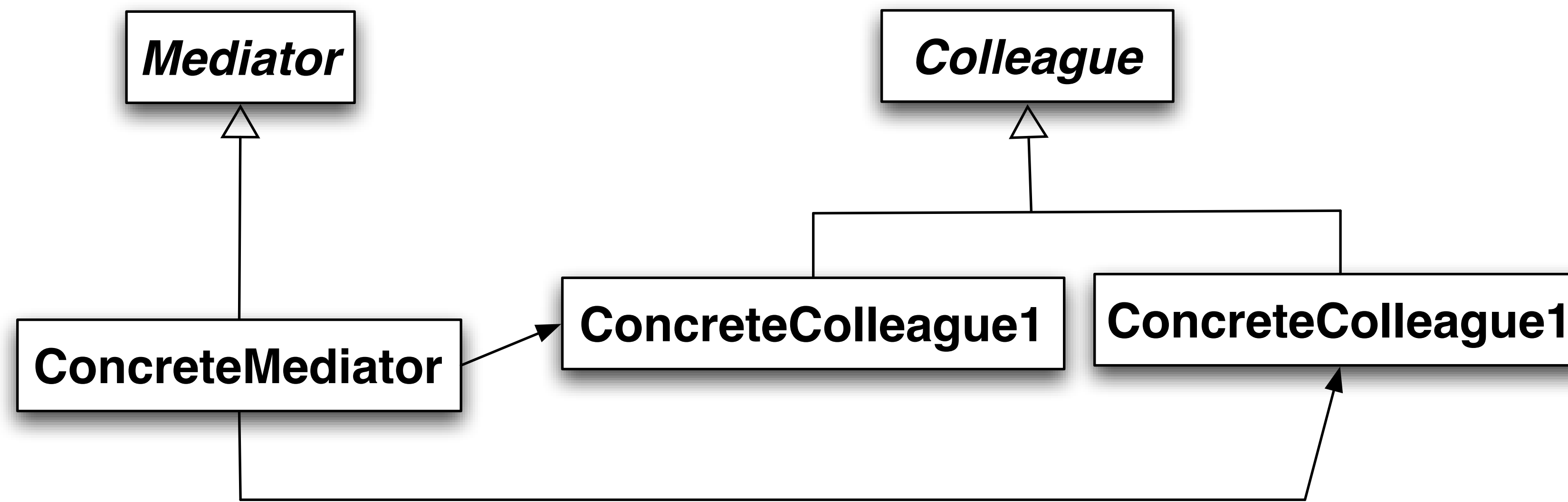
Mediator

A mediator controls and coordinates the interactions of a group of objects





Structure



Participants

Mediator

Defines an interface for communicating with Colleague objects

ConcreteMediator

Implements cooperative behavior by coordinating Colleague objects

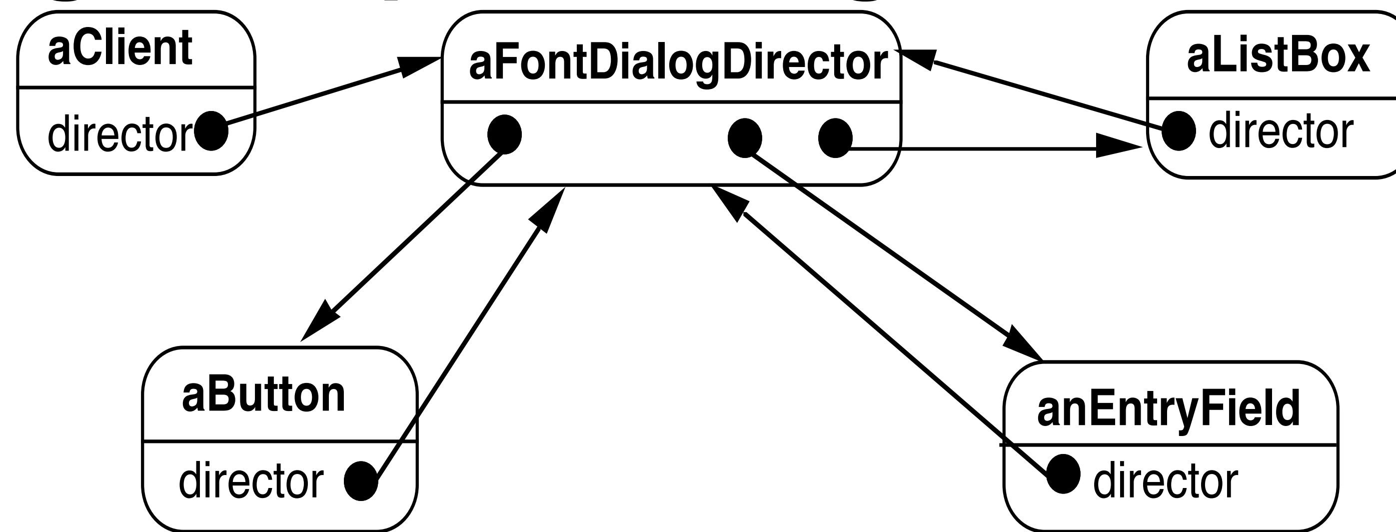
Knows and maintains its colleagues

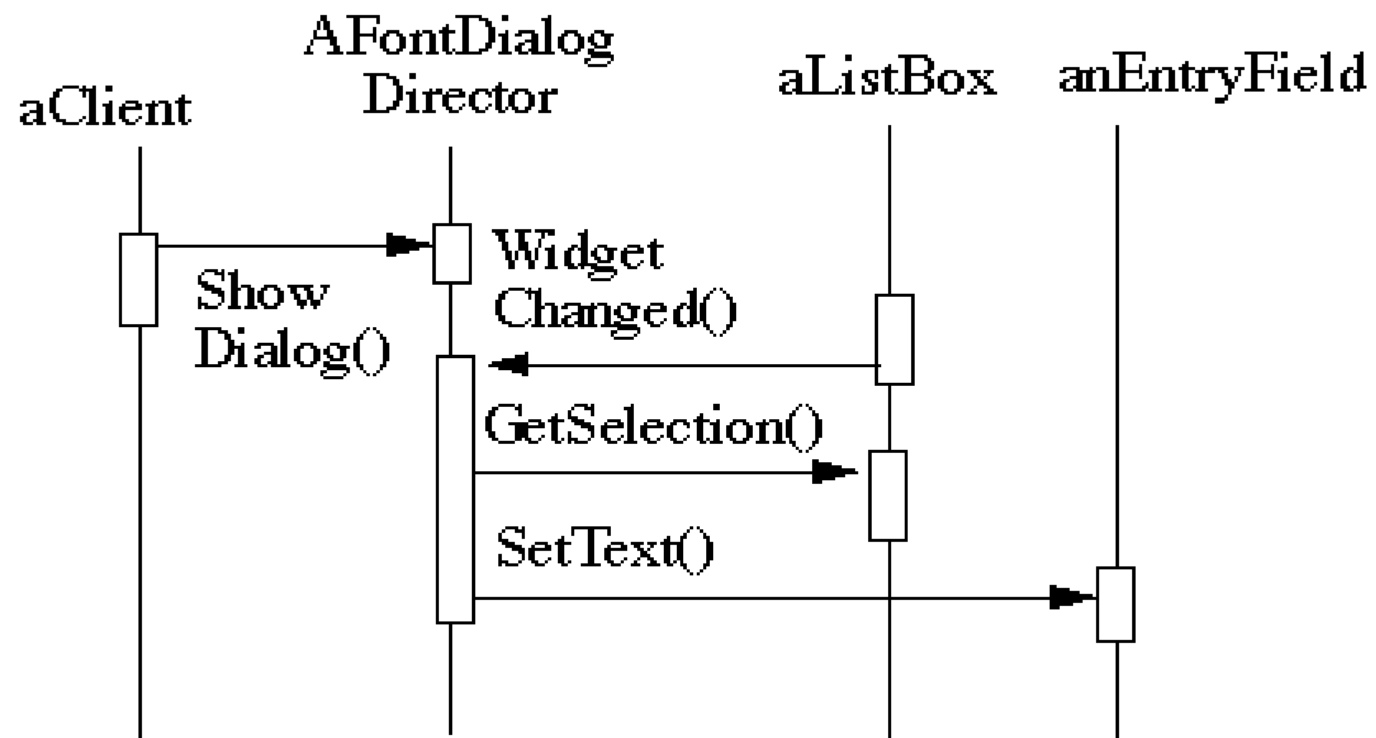
Colleague classes

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





How does this differ from a God Class?

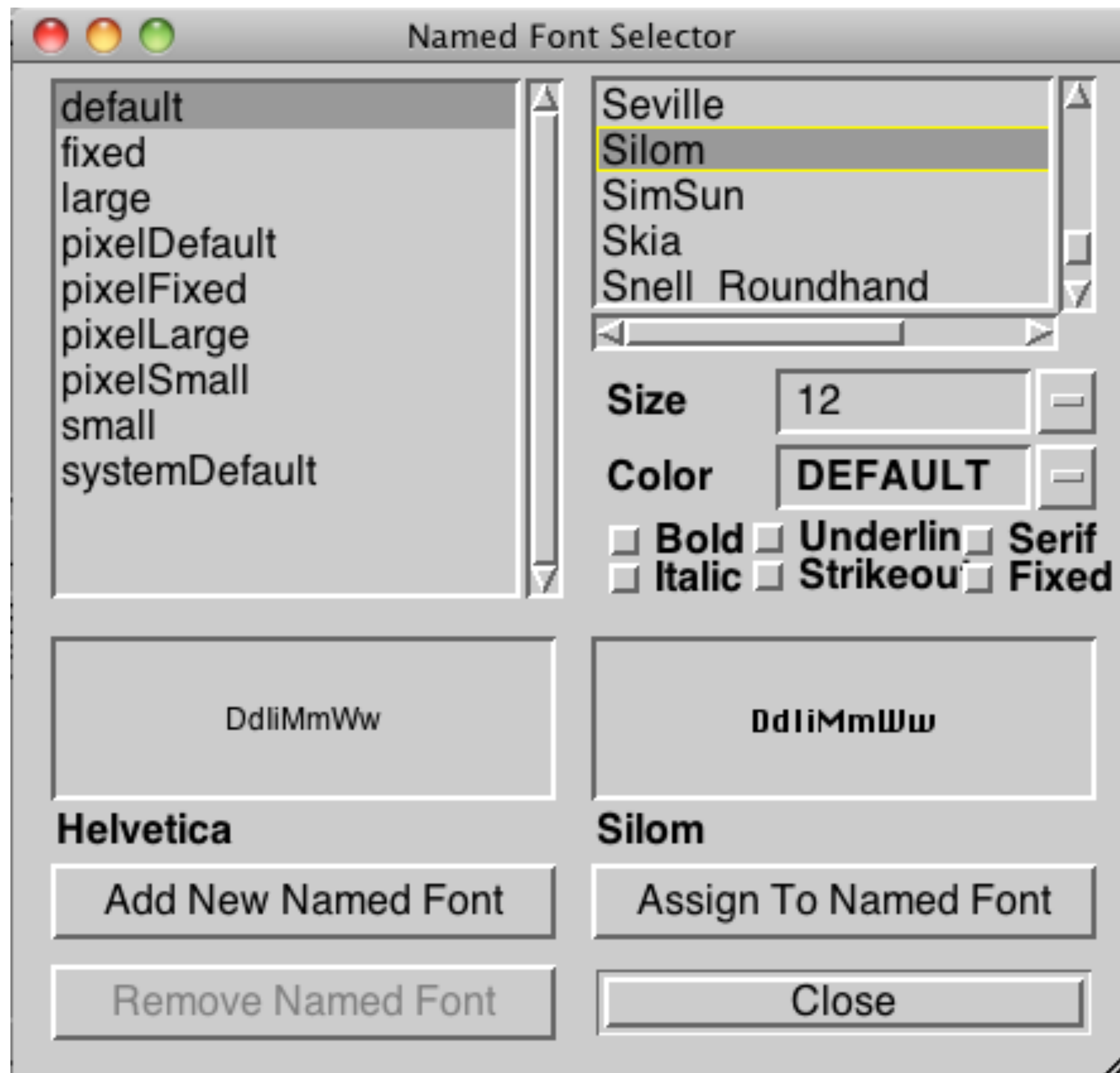
When to use the Mediator Pattern

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

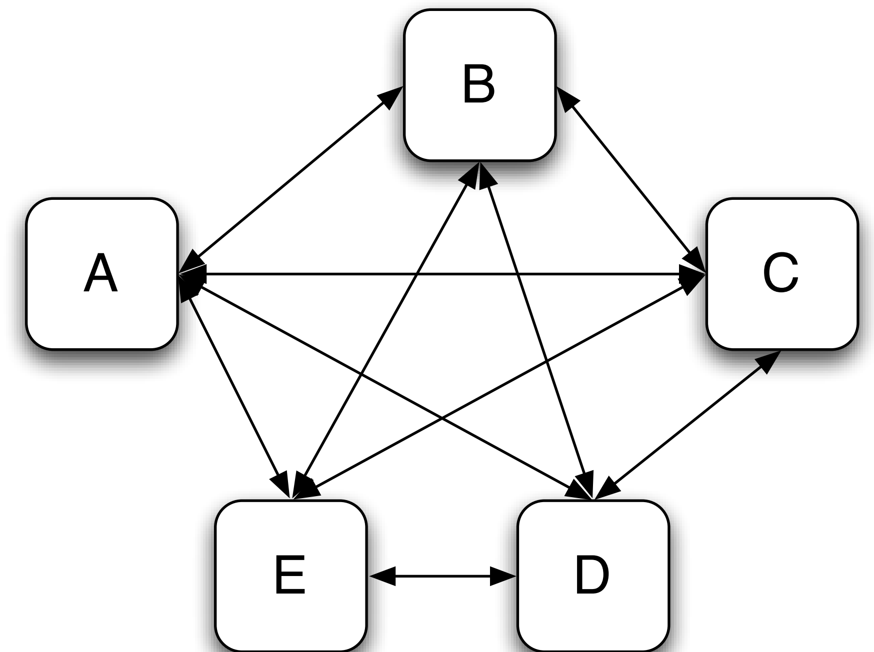
Classic Mediator Example



Simpler Example

Non Mediator Solution

```
class OKButton extends Button {
	TextField password;
	TextField username;
	Database userData;
	Model application;

	protected void processEvent(AWTEvent e) {
		if (!e.isButtonPressed()) return;
		e.consume();
		if (password.getText() = "") {
			notifyUser("Must enter password");
			return;
		}
		if (username.getText() = "") {
			notifyUser("Must enter user name");
			return;
		}
		if (!userData.validUser(password.getText(), username.getTest()))
			notifyUser("Invalid username & password");
			return;
		}
```



Mediator Solution

```
class LoginDialog extends Panel {
    TextField password;
    TextField username;
    Database userData;
    Button ok, cancel;

    protected void actionPerformed(ActionEvent e)  {
        if (!e.isButtonPressed() or e.getSource() != ok) return;
        if (password.getText() = "") {
            notifyUser("Must enter password");
            return;
        }
        if (username.getText() = "") {
            notifyUser("Must enter user name");
            return;
        }
        if (!userData.validUser(password.getText(), username.getTest()))
            notifyUser("Invalid username & password");
            return;
        }
```

What is Different?

Non Mediator Example

Special Button class

OK button coupled to text fields

Mediator Example

No specialButton class

LoginDialog coupled to text fields

Logic moved from button class to LoginDialog

But

Java's event mechanism promotes mediator solution