INF011 – Padrões de Projeto

20 – *Mediator*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas

Mediator

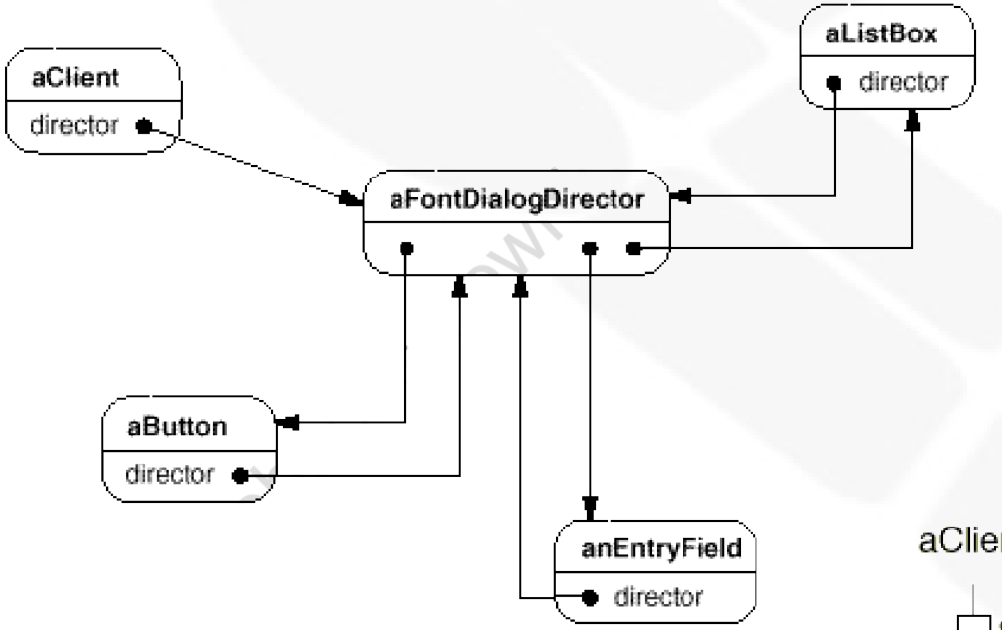
- Propósito:
  - Definir um objeto que encapsula como um conjunto de objetos interagem
- Motivação:
  - Em sistemas orientados a objetos distribui-se o comportamento em um grupo de objetos, com frequentes inter-conexões
  - Particionar o sistema em vários objetos aumenta o reuso porém o excesso de inter-conexões pode impedir que objetos funcionem sem a presença de outros

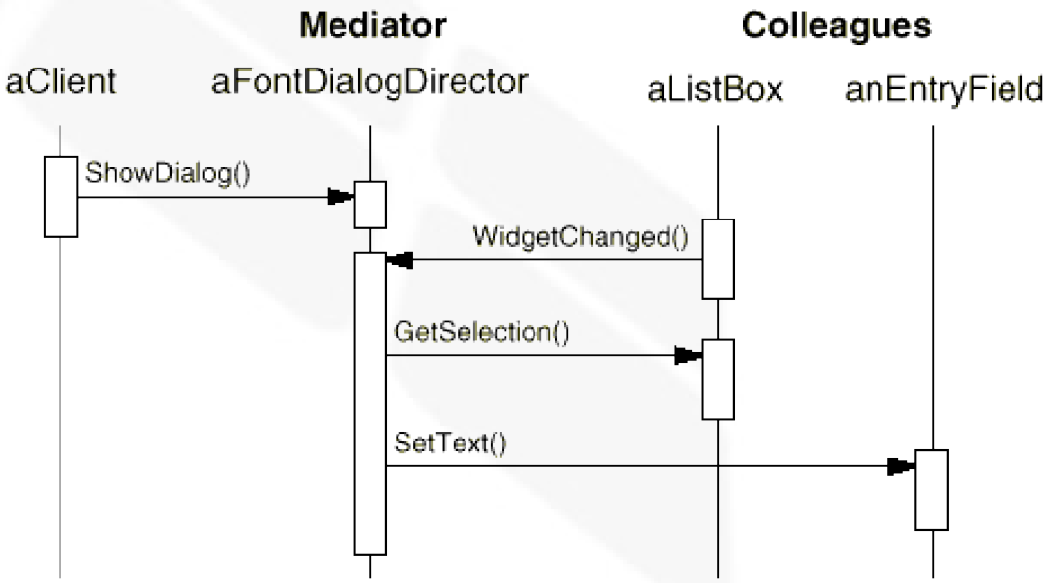
Mediator

- Motivação:
  - Dependências entre os *widgets*
  - Diferentes *dialogbox* terão diferentes dependências
  - Utiliza-se um objeto separado – o *Mediator* – para coordenar e controlar as interações de um grupo de objetos
  - Impede que objetos conheçam uns aos outros, diminuindo dependências

Mediator

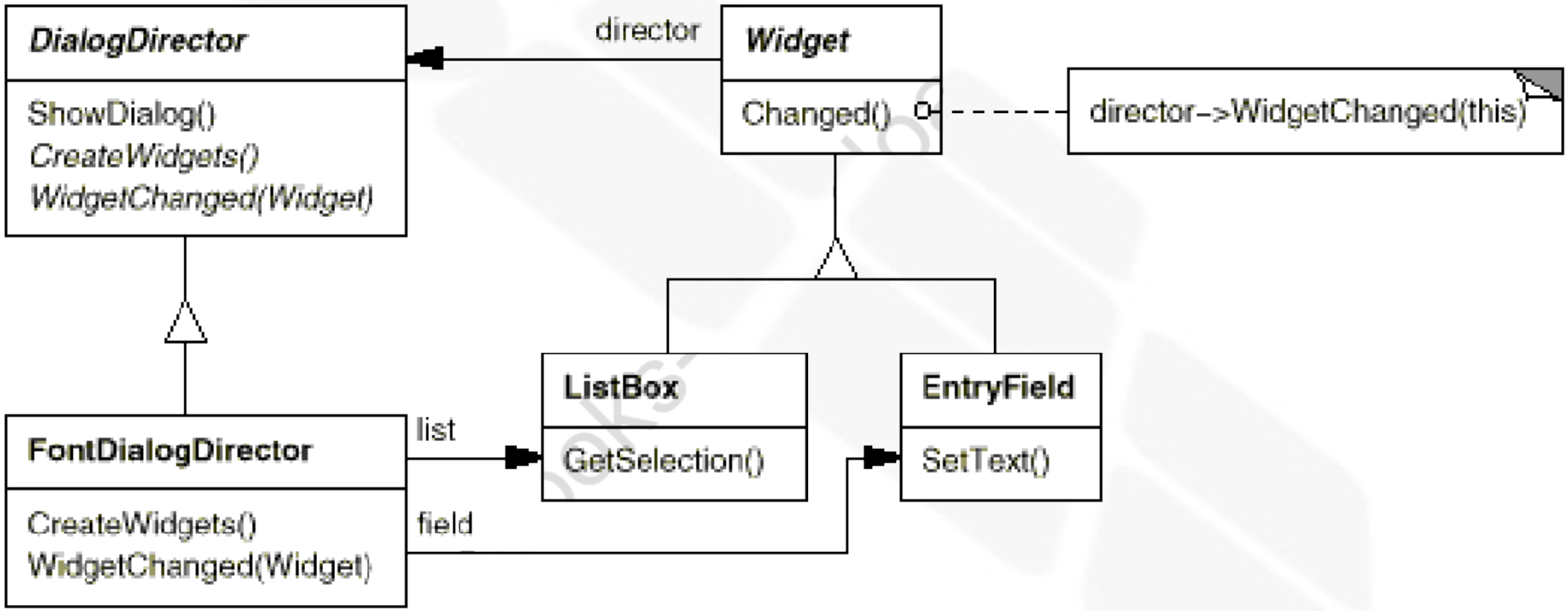
- Motivação:





Mediator

- Motivação:

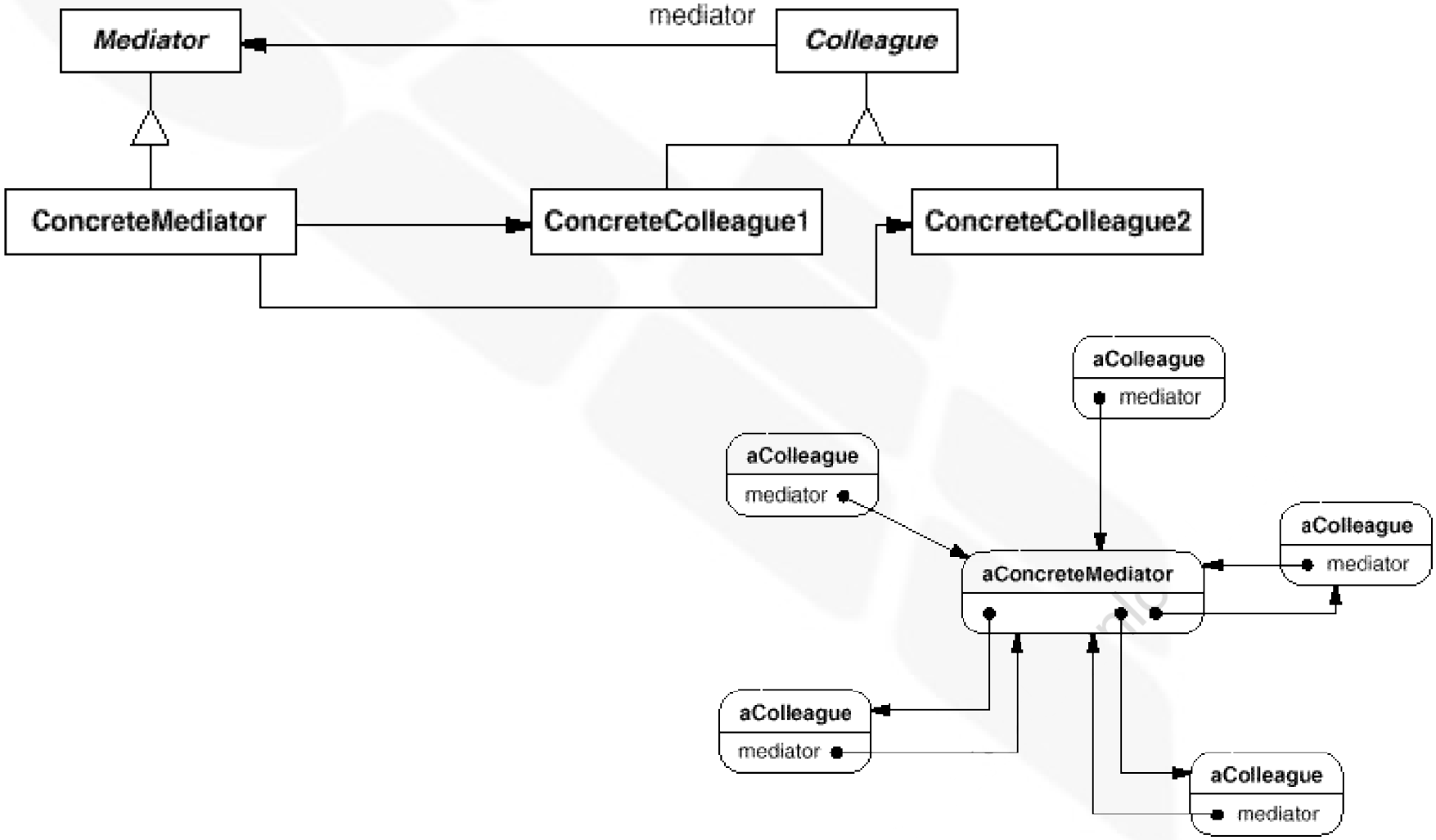


Mediator

- Aplicabilidade:
  - Quando um conjunto de objetos se comunicam de forma bem definida porém complexa. As inter-dependências resultantes são desestruturadas ou difíceis de entender
  - Quando reutilizar um objeto é difícil porque ele se refere e se comunica com um número considerável de objetos
  - Quando um comportamento que está distribuído entre várias classes deve ser configurável e não deseja-se requerer herança de implementação para isso

Mediator

- Estrutura:



Mediator

- Participantes:
  - *Mediator* (DialogDirector):
    - Define a interface de comunicação com objetos *Colleague*
  - *ConcreteMediator* (FontDialogDirector):
    - Implementa um comportamento cooperativo através da coordenação de objetos *Colleague*
    - Conhece e mantém os *Colleagues*
  - Classes *Colleague* (ListBox, EntryField, etc):
    - Cada classe *Colleague* conhece o seu objeto *Mediator*
    - Cada *Colleague* se comunica com seu *Mediator* em todos os casos onde ele se comunicaria com outro *Colleague*

Mediator

- Colaborações:
  - *Colleagues* enviam e recebem requisições para/de um objeto *Mediator*
  - O *Mediator* implementa comportamento cooperativo roteando requisições entre *Colleagues* apropriados

Mediator

- Consequências:
  - Limita a herança de implementação:
    - O *Mediator* localiza um comportamento que, caso contrário, estaria distribuído em diversos objetos
    - Mudar este comportamente requer a derivação somente do *Mediator*, classes *Colleagues* podem ser reutilizadas como são
  - Desacopla os *Colleagues*:
    - Pode-se variar e reutilizar as classes *Mediator* e *Colleagues* de forma independente
  - Simplifica as interações entre objetos:
    - O *Mediator* substitui interações muitos-para-muitos por interações um-para-muitos, mais fáceis de compreender, manter e estender

Mediator

- Consequências:
  - Abstrai a forma de cooperação dos objetos:
    - Tornar a mediação um conceito independente e encapsulá-lo em um objeto permite focar na forma com que objetos interagem, independente dos seus comportamentos individuais
  - Centraliza o controle:
    - O *Mediator* substitui complexidade de interação por complexidade no *Mediator*
    - O *Mediator* pode se tornar mais complexo que qualquer *Colleague*, se tornando monolítico e difícil de manter

Mediator

- Implementação:
  - Omitindo a classe abstrata do *Mediator*:
    - Pode-se eliminar a necessidade da classe abstrata do *Mediator* se os *Colleagues* trabalham com somente um *Mediator*
  - Comunicação *Colleague-Mediator*:
    - Opção 1: o *Mediator* pode ser implementado como um *Observer* e classes *Colleagues* como *Subjects*
    - Opção 2: interfaces especializada para notificação no *Mediator*, permitindo uma comunicação mais direta por parte dos *Colleagues*

Mediator

- Código exemplo:

```
class DialogDirector {
public:
    virtual ~DialogDirector();

    virtual void ShowDialog();
    virtual void WidgetChanged(Widget*) = 0;

protected:
    DialogDirector();
    virtual void CreateWidgets() = 0;
};
```

Mediator

- Código exemplo:

```
class Widget {
public:
    Widget(DialogDirector*);
    virtual void Changed();

    virtual void HandleMouse(MouseEvent& event);
    // ...
private:
    DialogDirector* _director;
};
```

```
void Widget::Changed () {
    _director->WidgetChanged(this);
}
```

Mediator

- Código exemplo:

```
class ListBox : public Widget {
public:
    ListBox(DialogDirector*);

    virtual const char* GetSelection();
    virtual void SetList(List<char*>* listItems);
    virtual void HandleMouse(MouseEvent& event);
    // ...
};
```

```
class Button : public Widget {
public:
    Button(DialogDirector*);

    virtual void SetText(const char* text);
    virtual void HandleMouse(MouseEvent& event);
    // ...
};

void Button::HandleMouse (MouseEvent& event) {
    // ...
    Changed();
}
```

```
class EntryField : public Widget {
public:
    EntryField(DialogDirector*);

    virtual void SetText(const char* text);
    virtual const char* GetText();
    virtual void HandleMouse(MouseEvent& event);
    // ...
};
```

Mediator

- Código exemplo:

```
class FontDialogDirector : public DialogDirector {
public:
    FontDialogDirector();
    virtual ~FontDialogDirector();
    virtual void WidgetChanged(Widget*);

protected:
    virtual void CreateWidgets();

private:
    Button* _ok;
    Button* _cancel;
    ListBox* _fontList;
    EntryField* _fontName;
};
```

Mediator

- Código exemplo:

```
void FontDialogDirector::CreateWidgets () {
    _ok = new Button(this);
    _cancel = new Button(this);
    _fontList = new ListBox(this);
    _fontName = new EntryField(this);

    // fill the listBox with the available font names

    // assemble the widgets in the dialog
}
```

```
void FontDialogDirector::WidgetChanged (
    Widget* theChangedWidget
) {
    if (theChangedWidget == _fontList) {
        _fontName->SetText(_fontList->GetSelection());

    } else if (theChangedWidget == _ok) {
        // apply font change and dismiss dialog
        // ...

    } else if (theChangedWidget == _cancel) {
        // dismiss dialog
    }
}
```

Mediator

- Usos conhecidos:
  - ET++
  - THINK C
  - Smalltalk

Mediator

- Padrões relacionados:
  - Diferenças entre o *Facade* e o *Mediator*:
    - O *Facade* abstrai um sub-sistema formado por objetos com o objetivo de disponibilizar uma interface mais conveniente. Seu protocolo é uni-direcional (*Facade* → classes do sub-sistema)
    - Em contraste, o *Mediator* permite comportamento cooperativo não disponibilizado pelos objetos *Colleagues*. Seu protocolo é multi-direcional

INF011 – Padrões de Projeto

20 – *Mediator*

Sandro Santos Andrade
sandroandrade@ifba.edu.br

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas