Instituto de computação

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrão MVC

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

Introdução

Como projetar sua aplicação visual de forma que as classes do modelo fiquem desacopladas da visualização?

Como permitir que diferentes formas de interação possam ser feitas de modo intercambiável entre a visualização e o modelo?

Como permitir que a atualização da interface (apresentação) seja feita de modo uniforme independentemente dos elementos que a compõem?

Padrões de Projeto

Metáfora do tocador de música

Cliente/Usuário

As ações na interface vão para o controlador (apertar o botão e tocar)

Controlador

O Controlador analisa a ação e, conforme uma estratégia definida, manipula o modelo (por exemplo, pede ao Player para tocar)

```
public class player{
  void play();
  void rip(){};
  void burn(){};
}
```

Modelo

Contém os dados, estados e lógica da aplicação

O Modelo notifica a visualização sobre uma mudança de seu estado

Visualização

A Visualização é atualizada para o cliente

MVC – Model-View-Controller

Surgiu com a linguagem SmallTalk

Padrões de Projeto

O Padrão MVC

Controlador

1 – O usuário faz algo na interface

Traduz as ações na interface para ações que façam sentido para o modelo

2 – Mude seu estado

Visualização

3 – Atualize sua apresentação

Modelo

```
public class player{
  void play();
  void rip(){};
  void burn(){};
}
```

4 – Meu estado foi modificado

5 – Preciso de dados sobre seu estado atual

Fornece uma representação para o modelo. Busca os dados que precisa diretamente do modelo

Contém os dados, estado e a lógica da aplicação. Desconhece a Visualização e o Controlador porém, fornece uma interface para recuperação de seus dados e notifica observadores sobre mudanças internas

Padrões de Projeto

O Padrão MVC

1. **O usuário interage com a Visualização (interface gráfica) -** A Visualização é a janela para o Modelo. A Visualização informa o Controlador sobre todo tipo de interação feita sobre ela.

2. **O Controlador pede ao Modelo para modificar seu estado -** O Controlador obtém as interações ocorridas na Visualização e as interpreta traduzindo em ações que irão manipular o modelo.

3. **O Controlador pede para a Visualização se modificar -** O Controlador em alguns casos pede para a Visualização se atualizar (por exemplo, desabilitar um botão ou menu), após receber um vento causado por uma interação.

4. **O Modelo notifica a Visualização quando seu estado é modificado -** As mudanças de estado podem ocorrer devidos a eventos externos ou internos.

5. **A Visualização requisita um estado ao Modelo -** A Visualização busca diretamente do modelo o estado requisitado. Ela também pode requisitar um estado do modelo como resultado da ação do Controlador sobre si mesma.

O Padrão MVC é um padrão composto

Um padrão composto combina dois ou mais padrões que resolvem um problema recorrente ou geral

Padrões de Projeto

MVC – um padrão composto

Strategy
Controlador

1 – O usuário faz algo na interface

2 – Mude seu estado

3 – Atualize sua apresentação

Composite
Visualização

A Visualização e o Controlador implementam o Padrão Strategy: a Visualização é configurada com a estratégia fornecida pelo Controlador. A Visualização delega ao Controlador as ações sobre o modelo em função de enventos que ela dispara. Isto desacopla a Visualização das ações sobre o modelo.

Observer
Modelo

```
public class player{
  void play();
  void rip(){};
  void burn(){};
}
```

4 – Meu estado foi modificado

5 – Preciso de dados sobre seu estado atual

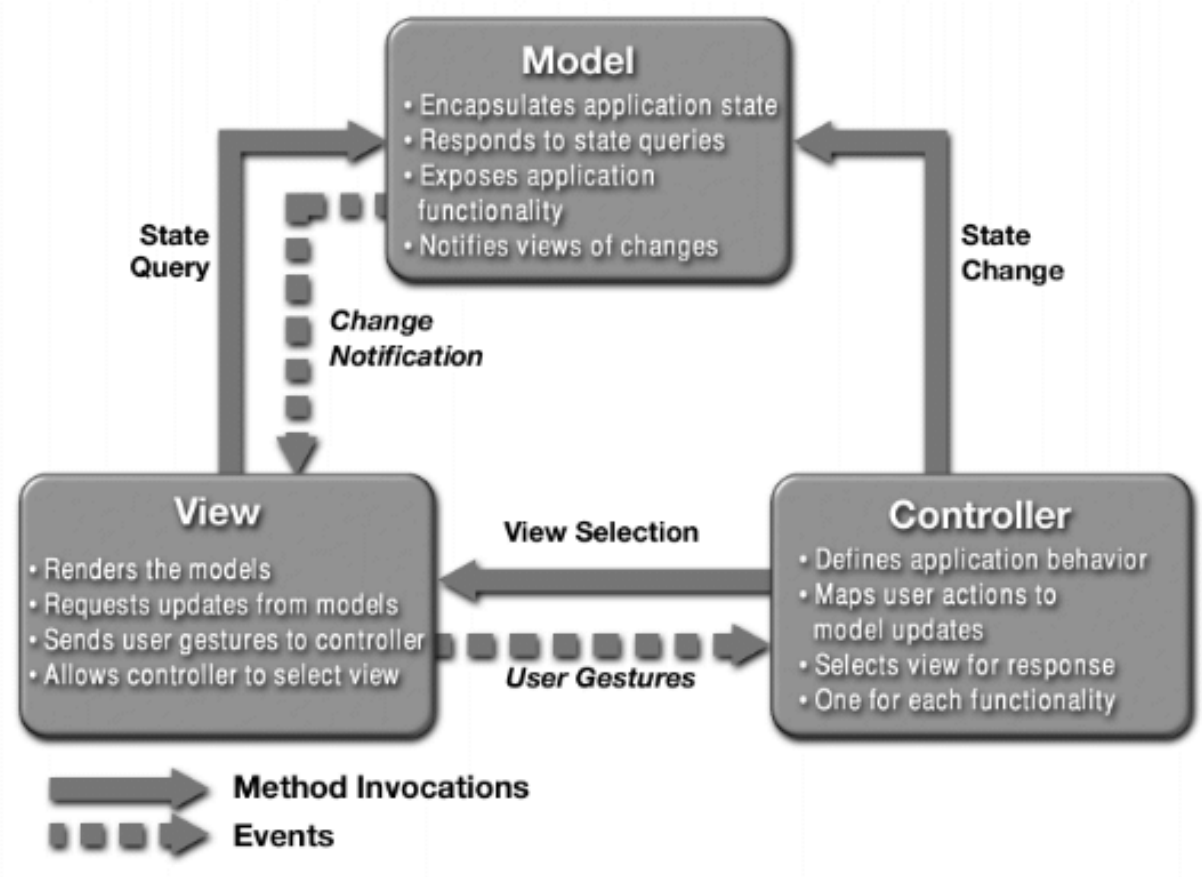
A Visualização é formada por uma composição de elementos de interface, alguns compostos (Panels) e outros individuais (buttons). O Controlador interage apenas com o o elemento na raiz da hierarquia e o Padrão Composite faz o restante da tarefa.

O Modelo implementa o padrão Observer para manter atualizados os objetos interessados em seu estado. Desacopla o modelo da Visualização e do Controlador e permite diferentes e até múltiplas representações do modelo.

Padrões de Projeto

Arquiteturas baseadas em MVC

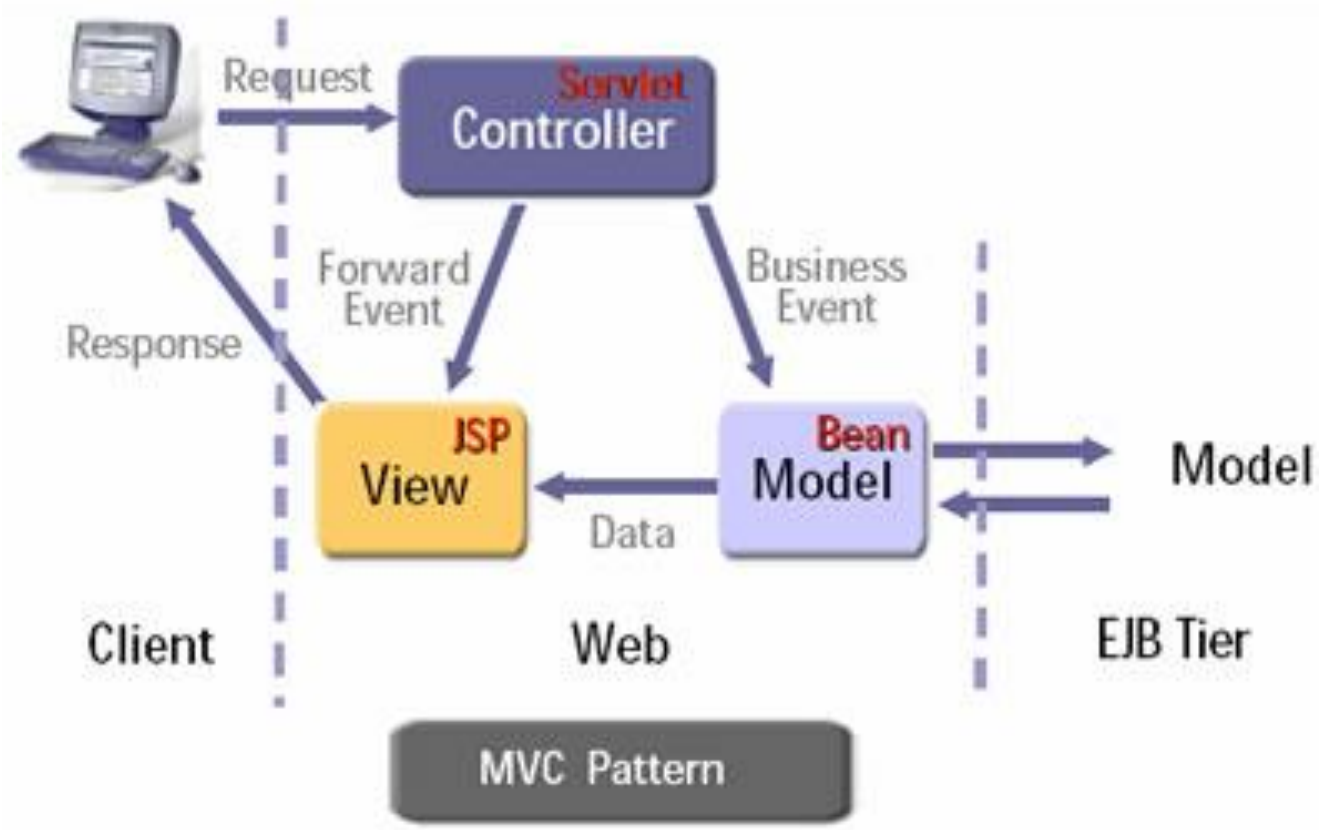
Estratégia desktop

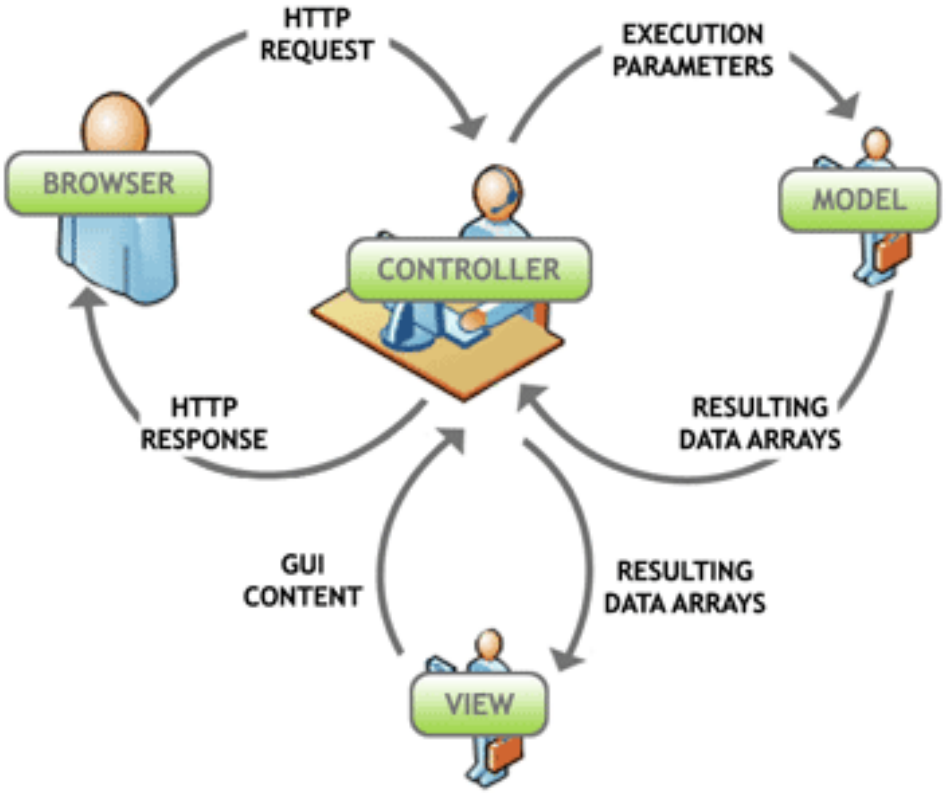


Padrões de Projeto

Arquiteturas baseadas em MVC

Estratégia Web 1




HTTP
REQUEST
EXECUTION
PARAMETERS
BROWSER
CONTROLLER
MODEL
HTTP
RESPONSE
RESULTING
DATA ARRAYS
GUI
CONTENT
RESULTING
DATA ARRAYS
VIEW

Exemplo de Problema

Considere a implementação de um aplicativo que simula uma batida de uma discoteca (DJViewer).

A interface do aplicativo possui duas partes:

1 - Uma que exibe o número de batidas por minuto (BPM) através de uma barra

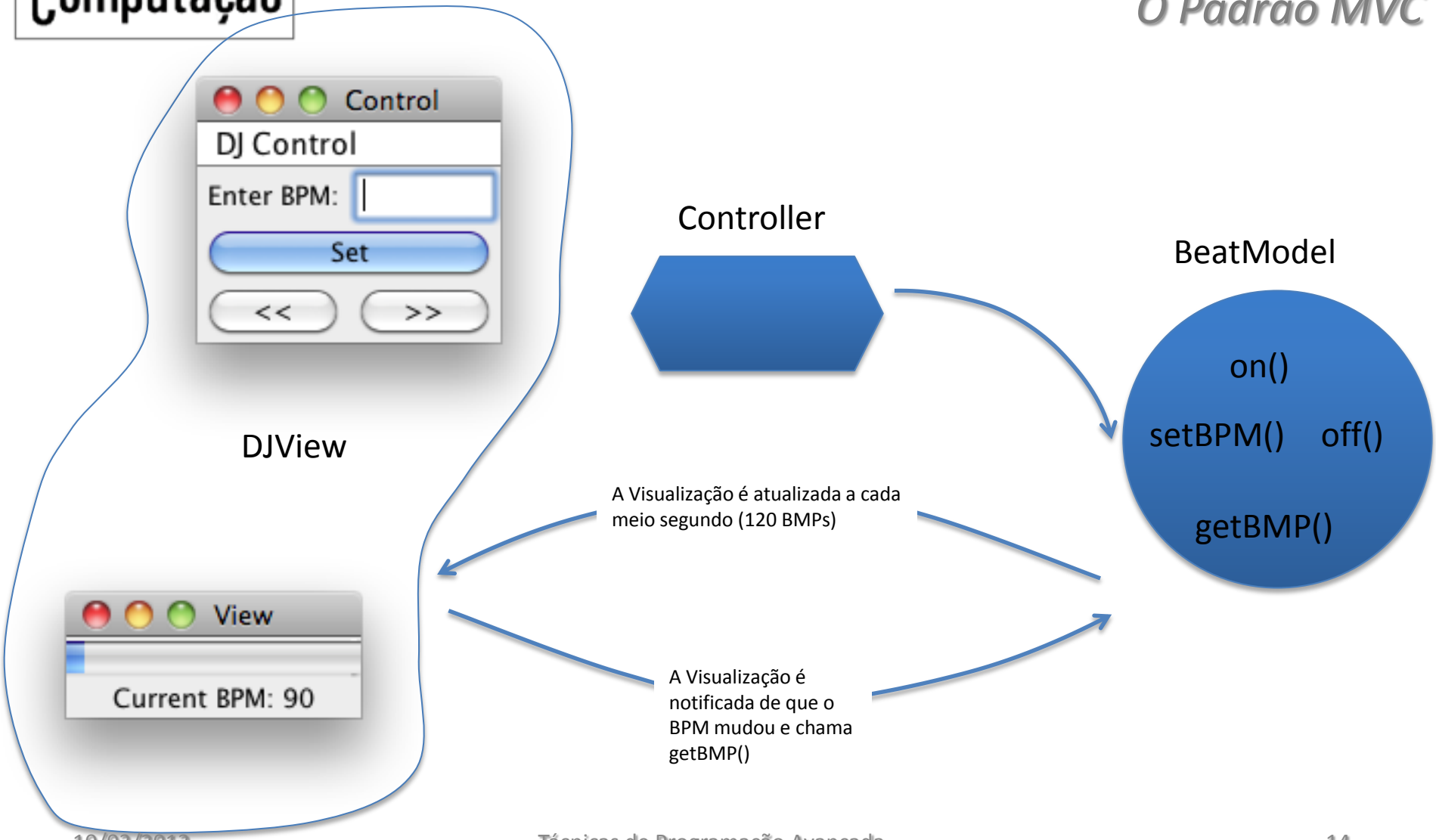
2 - Uma interface que controla o sequenciador MIDI permitindo iniciar a batida, parar a batida e configurar o número de batidas por minuto

Considere a implementação de um aplicativo que simula uma batida de uma discoteca (DJViewer).

A interface do aplicativo possui duas partes:

1 - Uma que exibe o número de batidas por minuto (BPM) através de uma barra

2 - Uma interface que controla o sequenciador MIDI permitindo iniciar a batida, parar a batida e configurar o número de batidas por minuto


O usuário faz algo na interface
Control
DJ Control
Enter BPM:
Set
<<
>>
DJView
Controller
BeatModel
on()
setBPM()
off()
getBMP()
A Visualização é atualizada a cada meio segundo (120 BMPs)
View
Current BPM: 90
A Visualização é notificada de que o BPM mudou e chama getBMP()

Padrões de Projeto

Exemplo de Problema

```
public class BeatController implements ControllerInterface {

  BeatModelInterface model;
  DJView view;

  public BeatController(BeatModelInterface model) {
    this.model = model;
    view = new DJView(this, model);
    view.createView();
    view.createControls();
    view.disableStopMenuItem();
    view.enableStartMenuItem();
    model.initialize();
  }

  public void start() {
    model.on();
    view.disableStartMenuItem();
    view.enableStopMenuItem();
  }

  public void stop() {
    model.off();
    view.disableStopMenuItem();
    view.enableStartMenuItem();
  }
```

```
public interface ControllerInterface {

  void start();
  void stop();
  void increaseBPM();
  void decreaseBPM();
  void setBPM(int bpm);

}
```

```
  public void increaseBPM() {
     int bpm = model.getBPM();
     model.setBPM(bpm + 1);
  }

  public void decreaseBPM() {
     int bpm = model.getBPM();
    model.setBPM(bpm - 1);
  }

  public void setBPM(int bpm) {
     model.setBPM(bpm);
  }

}
```

Padrões de Projeto

Exemplo de Problema

```
public interface BeatModelInterface {
  void initialize();
  void on();
  void off();
  void setBPM(int bpm);
   int getBPM();
   void registerObserver(BeatObserver o);
   void removeObserver(BeatObserver o);
   void registerObserver(BPMObserver o);
   void removeObserver(BPMObserver o);
}
```

Padrões de Projeto

Exemplo de Problema

```
public class BeatModel implements BeatModelInterface, MetaEventListener {
   Sequencer sequencer;
   ArrayList beatObservers = new ArrayList();
  ArrayList bpmObservers = new ArrayList();
  int bpm = 90; // other instance variables here
  public void initialize() {

setUpMidi();
     buildTrackAndStart();
}
public void on() { sequencer.start(); setBPM(90);
}
public void off() { setBPM(0);
sequencer.stop();
}
public void setBPM(int bpm) { this.bpm = bpm;
sequencer.setTempoInBPM(getBPM()); notifyBPMObservers();
}
public int getBPM() { return bpm;
}
void beatEvent() { notifyBeatObservers();
}
// Code to register and notify observers // Lots of MIDI code to handle the beat
```

Padrões de Projeto

O Padrão Iterator

O Padrão Iterator provê um modo de acessar os elementos de um objeto agregado sequencialmente sem expor sua representação interna

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books

- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman