

ITI 1521. Introduction à l'informatique II

Interface utilisateur graphique : **Modèle-Vue-Contrôleur**

Marcel Turcotte



uOttawa

École de science informatique et de génie électrique (SIGE)
Université d'Ottawa

Version du 7 février 2017

Aperçu

Interface utilisateur graphique : Modèle-Vue-Contrôleur

En développement logiciel, un **patron de conception** est un arrangement spécifique de classes. C'est une façon **standard** de résoudre des problèmes bien connus. Les programmeurs de l'industrie sont **familiers** avec ces patrons. Cette semaine, nous découvrons le patron de conception **Modèle-Vue-Contrôleur (MVC)** qui sert dans la réalisation d'interfaces utilisateur graphiques.

Objectif général :

- Cette semaine, vous serez en mesure de concevoir l'interface utilisateur graphique d'une application simple en appliquant le patron de conception Modèle-Vue-Contrôleur.

Objectifs d'apprentissage

- ❖ **Décrire** en vos propres mots le patron de conception Modèle-Vue-Contrôleur.
- ❖ **Utiliser** le patron de conception Modèle-Vue-Contrôleur afin de produire le rendu visuel d'une interface utilisateur graphique.

Lectures :

- ❖ <https://fr.wikipedia.org/wiki/Modèle-vue-contrôleur>
- ❖ http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf

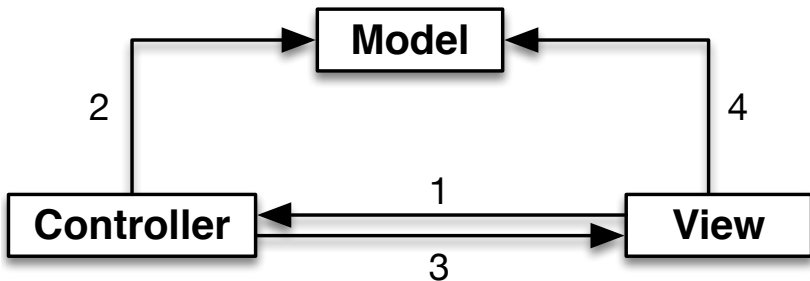
Plan

1. **Préambule**
2. **Théorie**
3. **Exemple**
4. **Prologue**

Patron de conception («*design pattern*»)

- ❖ En développement logiciel, un **patron de conception** est un arrangement spécifique de classes.
- ❖ C'est une façon **standard** de résoudre des problèmes bien connus.
- ❖ Les programmeurs de l'industrie sont **familiers** avec ces patrons.

Modèle-Vue-Contrôleur («Model-View-Controller»)



- MVC permet de séparer les **données**, la **vue** et la **logique** de l'application.

Avantages

- ❖ Permet de modifier ou adapter chacune des parties de l'application de façon **indépendante** ;
- ❖ Favorise l'implémentation de **plusieurs vues** ;
- ❖ L'**association entre le modèle et la vue se fait dynamiquement** au moment de l'exécution (et non au moment de la compilation), ce qui permet de changer la vue au moment de l'exécution.

Définitions

- ❖ **Modèle** – l'implémentation, état : attributs et comportements ;
- ❖ Vue (**View**) – l'interface en sortie, une représentation du modèle pour le monde extérieur ;
- ❖ Contrôleur (**Controller**) – l'interface en entrée, achemine les requêtes de l'utilisateur afin de mettre-à-jour le modèle.

Modèle

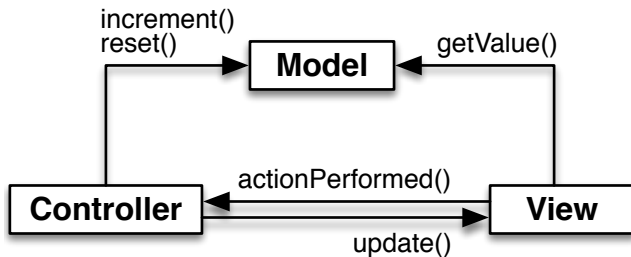
- ❖ Contient les **données de l'application** et les méthodes pour transformer les données ;
- ❖ Possède **une connaissance minimale de l'interface graphique** (parfois aucune connaissance) ;
- ❖ La **vue** et le **modèle** sont très **différents**.

- Une **représentation** (graphique, textuelle, vocale, etc.) du modèle pour l'**utilisateur**.

Contrôleur

- ❖ Un contrôleur est un objet permettant à l'utilisateur de **transformer** les **données** ou la **représentation** ;
- ❖ **Connaît très bien le modèle.**

Counter



- Une interface graphique pour la classe **Counter**.

Counter : Modèle

```
public class Counter {
    private int value;
    public Counter() {
        value = 0;
    }
    public void increment() {
        value++;
    }
    public int getValue() {
        return value;
    }
    public void reset() {
        value = 0;
    }
    public String toString() {
        return "Counter: {value="+value+"}";
    }
}
```

View

```
public interface View {  
    void update();  
}
```

- ❖ Pour faciliter le développement de plusieurs vues, nous créons l'interface **View**.
- ❖ Notre exemple aura deux vues : **GraphicalView** et **TextView**.

```
public class TextView implements View {  
  
    private Counter model;  
  
    public TextView(Counter model) {  
        this.model = model;  
    }  
  
    public void update() {  
        System.out.println(model.toString());  
    }  
  
}
```

```
public class GraphicalView extends JFrame implements View {
    private JLabel input;
    private Counter model;
    public GraphicalView(Counter model, Controller controller) {
        setLayout(new GridLayout(1,3));
        this.model = model;
        JButton button;
        button = new JButton("Increment");
        button.addActionListener(controller);
        add(button);
        JButton reset;
        reset = new JButton("Reset");
        reset.addActionListener(controller);
        add(reset);
        input = new JLabel();
        add(input);
    }
    public void update() {
        input.setText(Integer.toString(model.getValue()));
    }
}
```

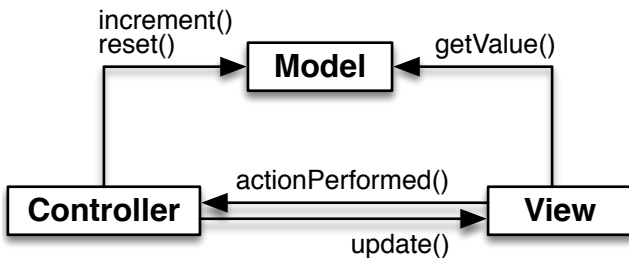


```
public class Controller implements ActionListener {  
  
    private Counter model;  
  
    private View [] views;  
    private int numberOfViews;  
  
    public Controller() {  
  
        views = new View [2];  
        numberOfViews = 0;  
  
        model = new Counter ();  
  
        register (new GraphicalView (model, this));  
        register (new TextView (model));  
  
        update ();  
  
    }  
}
```

```
private void register(View view) {  
    views[numberOfViews] = view;  
    numberOfViews++;  
}  
  
private void update() {  
    for (int i=0; i<numberOfViews; i++) {  
        views[i].update();  
    }  
}
```

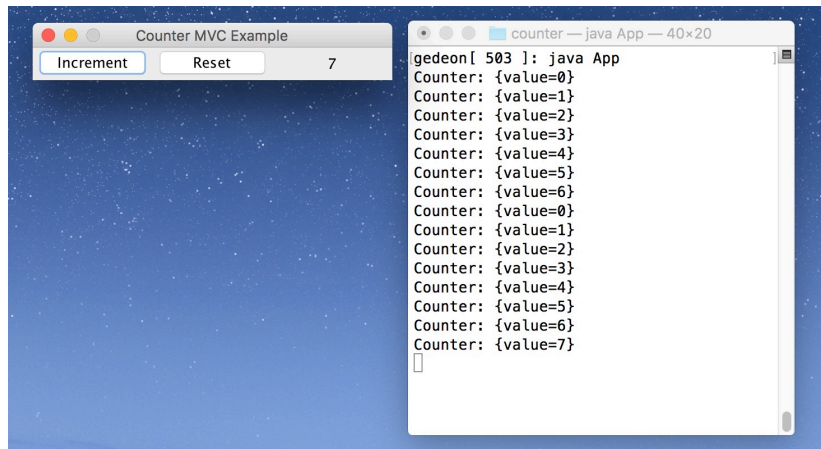
```
public void actionPerformed(ActionEvent e) {  
    if (e.getActionCommand().equals("Increment")) {  
        model.increment();  
    } else {  
        model.reset();  
    }  
    update();  
}  
}
```

Application Counter



```
public class App {  
    public static void main(String [] args) {  
        Controller controller;  
        controller = new Controller();  
    }  
}
```

Counter



- ❖ L'application **Counter** est ses deux vues : textuelle et graphique.

Résumé

- **MVC** permet de séparer clairement les **données**, les **vues**, et la **logique** d'une application.

Exercices

Implémentez chacune des applications suivantes à l'aide du patron de conception Modèle-Vue-Contrôleur (MVC).

- ❖ Un jeu de **tic-tac-toe**
- ❖ Un jeu de **bataille navale**
- ❖ Un jeu de **mémoire**
- ❖ Jeu 2048

Prochain module

- Types **paramétrés** (« *generics* »)

References I



E. B. Koffman and Wolfgang P. A. T.
Data Structures : Abstraction and Design Using Java.
John Wiley & Sons, 2e edition, 2010.



Pensez-y !

L'impression de ces notes n'est probablement pas nécessaire !