Le schéma de conception Modèle-Vue-Contrôleur

18 février 2021

Pourquoi un schéma de conception ?

Voici une chambre avec schéma de conception, et une chambre sans.

Pourquoi un schéma de conception ?

Organiser le code **correctement**, en modules qui ont un sens, pour :

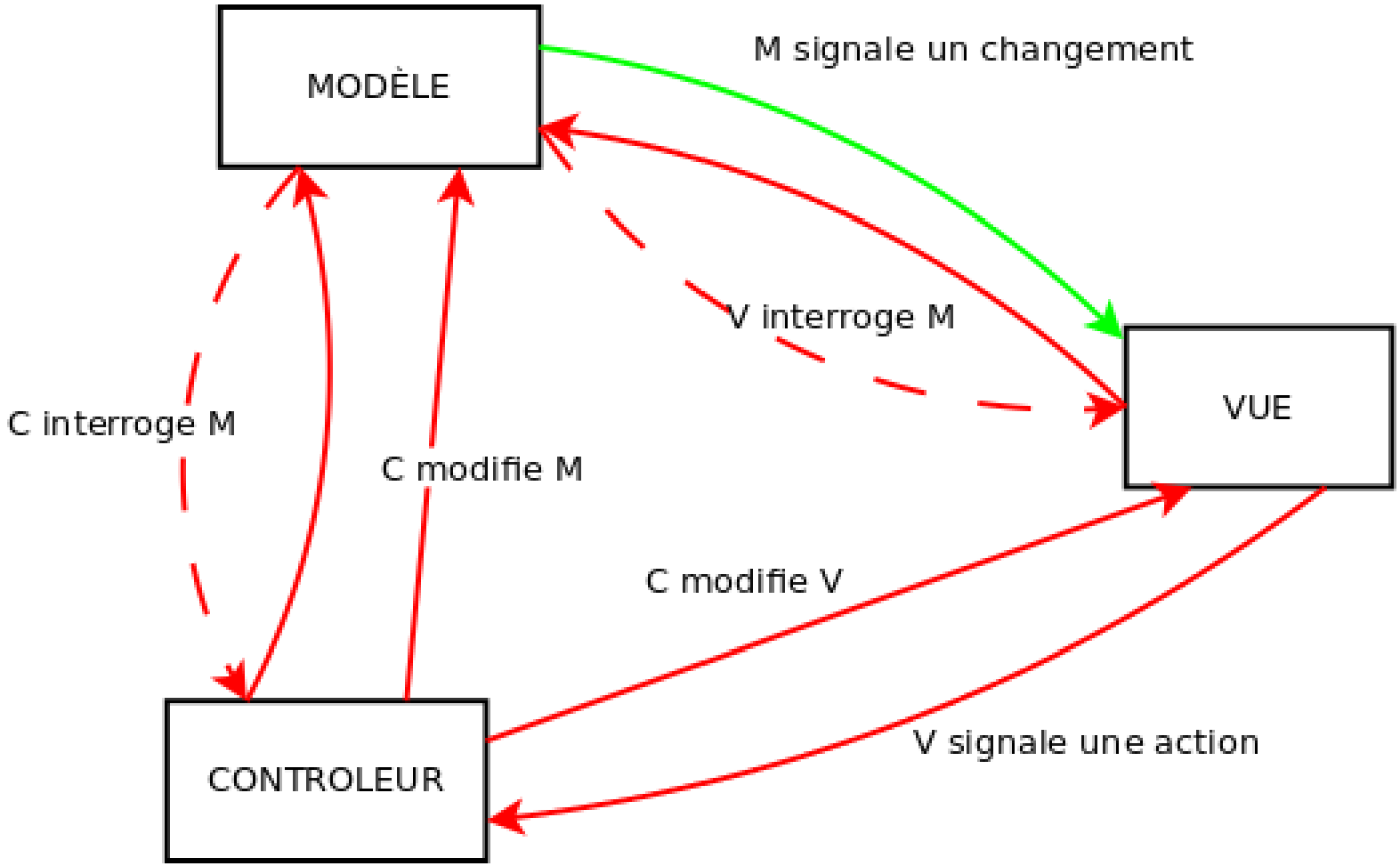
- mieux concevoir : plus rapidement, avec moins de bugs
- mieux comprendre : plus rapidement, avec moins de malentendus
- mieux réutiliser des composants : sans avoir à réécrire tout de 'A' à 'Z'.
- mieux faire évoluer : plus rapidement, sans avoir à revoir toute la structure.

– *Si la maison de votre maman est mieux rangée/organisée que votre code, c'est qu'elle est meilleure informaticienne que vous sur ce point* ;)

– Il y a plusieurs manières d'organiser son code : ranger les membres et méthodes par ordre alphabétique n'est pas une manière correcte.

– En IHM, le modèle classique est le Modèle/Vue/Controleur, mais il y en a d'autres.

M V C : le schéma global



M V C : le modèle

Le modèle ou le **noyau fonctionnel.**

Contient et gère toute l'intelligence technique (les données métiers).

- On y trouve :
  - la définition des types "métiers",
  - les méthodes manipulant ces types.
- Le modèle ne doit pas changer d'une virgule que l'IHM soit :
  - en Swing,
  - en javaFX,
  - avec une simple entrée/sortie clavier sur un terminal texte,
  - sur un terminal spécialisé pour le braille.

M V C : le modèle

Le modèle :

- répond aux demandes du contrôleur,
- répond aux demandes de la vue.
- **peut** signaler un changement de lui-même.

On n'y trouve :

- pas d'affichage.
- pas d'interaction avec l'utilisateur
  (à moins que cela fasse partie du "métier" de l'application).

M V C : le modèle

Exemple d'un jeu d'échec.

Le Modèle contient :

- l'échiquier et son contenu,
- le tour : blanc ou noir doit jouer
- et fournit les méthodes
  - `boolean estMat()`,
  - `int aQuiLeTour()`
  - `void reinitialise()`,
  - `Piece case (x,y)`
  - `boolean mouvementPossible(x1,y1, x2,y2)`, `void bouger(x1,y1,x2,y2)`
  - `Deplacement suggestionMouvement()`
  - ...

M V C : la vue

La vue contient la partie purement interface avec l'utilisateur :
- la (ou les) fenêtre(s) et tout ce qu'elles contiennent ;
- des méthodes permettant au contrôleur de manipuler la vue.

Cela peut être du javafx, ou des page web, une interface physique.

Elle fournit les méthodes :

- "setter" d'une icône, d'un texte, d'une fonte, d'un graphique ;
- activation/désactivation de boutons/menus/ ou zones de saisies ;
- accesseurs des zones de saisie ou d'affichage ;
- demandes plus générales :
  - affichage d'un message ou d'une fenêtre de confirmation,
  - demande de réactualisation d'une vue ou d'une image.
  - désactivation d'un bloc complet de la vue,
  - etc.

M V C : la vue

La vue :

- signale au contrôleur les actions des utilisateurs.
- interroge l'état du modèle quand elle a besoin d'afficher son état.

Ce n'est que de l'affichage et de la présentation : il n'y a aucune "intelligence" dedans.

M V C : la vue

Dans le cadre d'un programme d'un jeu d'échec, la vue pourrait comprendre :

- la fenêtre avec une vue du plateau de jeu,
- une zone d'affichage de messages,
- le bouton de reinitialisation,
- un moyen de saisir un déplacement *(x1,y1, x2,y2)* : drag et drop sur le plateau par exemple,
- et les méthodes :
  - `void afficheMessage(String)`
  - `void setMouvementPossible(boolean)`
  - `void rechargerPlateau()`
  - `void setCase(Piece,x,y)`
  - ...

M V C : la vue

Elle pourrait :

- signaler au *controleur* un mouvement (à la suite d'un drag et drop)
- signaler au *controleur* une action sur le bouton *reinitialiser*
- demander au *modèle* l'emplacement des pièces (à la suite d'un appel de `rechargerPlateau` par le *controleur*).

La même vue pourrait servir à un autre jeu que le jeu d'echecs avec des rêgles complètement différentes tant que c'est sur un plateau, avec une piece par case, des pieces que l'on bouge, et des messages que l'on affiche.

M V C : le contrôleur

Le contrôleur est le **maître du jeu**. Il :

- appelle les méthodes du modèle pour le modifier ;
- interroge le modèle sur son état ;
- demande à la vue de faire certaines manipulations :
  - changer l'affichage,
  - désactiver certains composants, etc,
- reçoit des signalement de la vue (action utilisateur) :
  - récupère des données sur la vue si besoin,
  - contrôle ces données (test de validité) et appelle les méthodes du modèle
  - change la vue si nécessaire.

M V C : le contrôleur

Pour un jeu d'échec, le *contrôleur* :

- initialiserait le jeu et la fenêtre.
- attendrait la demande de mouvement ou de reinitialisation.
- fournirait les méthodes
  - `void demandeMouvement(x1,y1,x2,y2)` appelée par la vue lors du drag et drop
  - `void reinitialiseJeu()` appelée par la vue lors de l'action sur le bouton

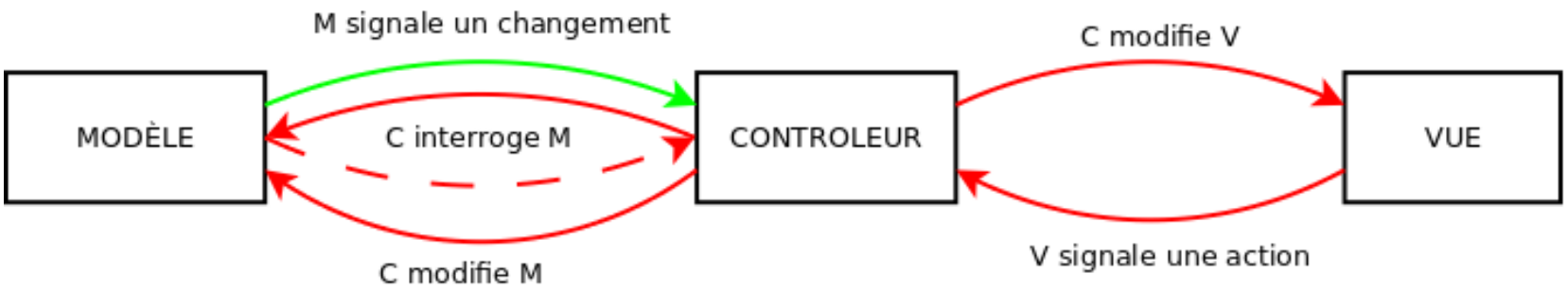
M V C : le contrôleur

méthode appelée par la vue après un drag et drop

```
void demandeMouvement(x1,y1,x2,y2) {
   if (modele.mouvementPossible(x1,y1,x2,y2)) {
      modele.bouger(x1,y1,x2,y2);
      vue.setCase(null,x1,y1);
      vue.SetCase(p,x2,y2);
      if (modele.estMat()){
         vue.setMouvementPossible(false);
         vue.message("la partie est finie");
      }
      else
         vue.message("le tour est à"+modele.aquiLeTour());
      }
   else
      vue.message("le mouvement est impossible");
   }
```

Variantes

Il y a d'autres manières d'organiser le code : Seeheim, Arch...



Elles reviennent souvent à :

– casser le lien entre M et V, et faire tout passer le controleur.

– définir (et découper) différemment le modèle et la vue