

Projet de Programmation

Benoit Donnet
Année Académique 2019 - 2020



1

Agenda

Partie 3: Eléments de Programmation Evénementielle

- Chapitre 1: Introduction aux Interfaces Graphiques
- Chapitre 2: Applications Interactives
- Chapitre 3: Pattern MVC

Agenda

- Chapitre 3: Pattern MVC
 - Principe
 - Le Pattern
 - Application
 - Discussion

Agenda

- Chapitre 3: Pattern MVC
 - Principe
 - ✓ Idée
 - ✓ Structure du Modèle
 - Le Pattern
 - Application
 - Discussion

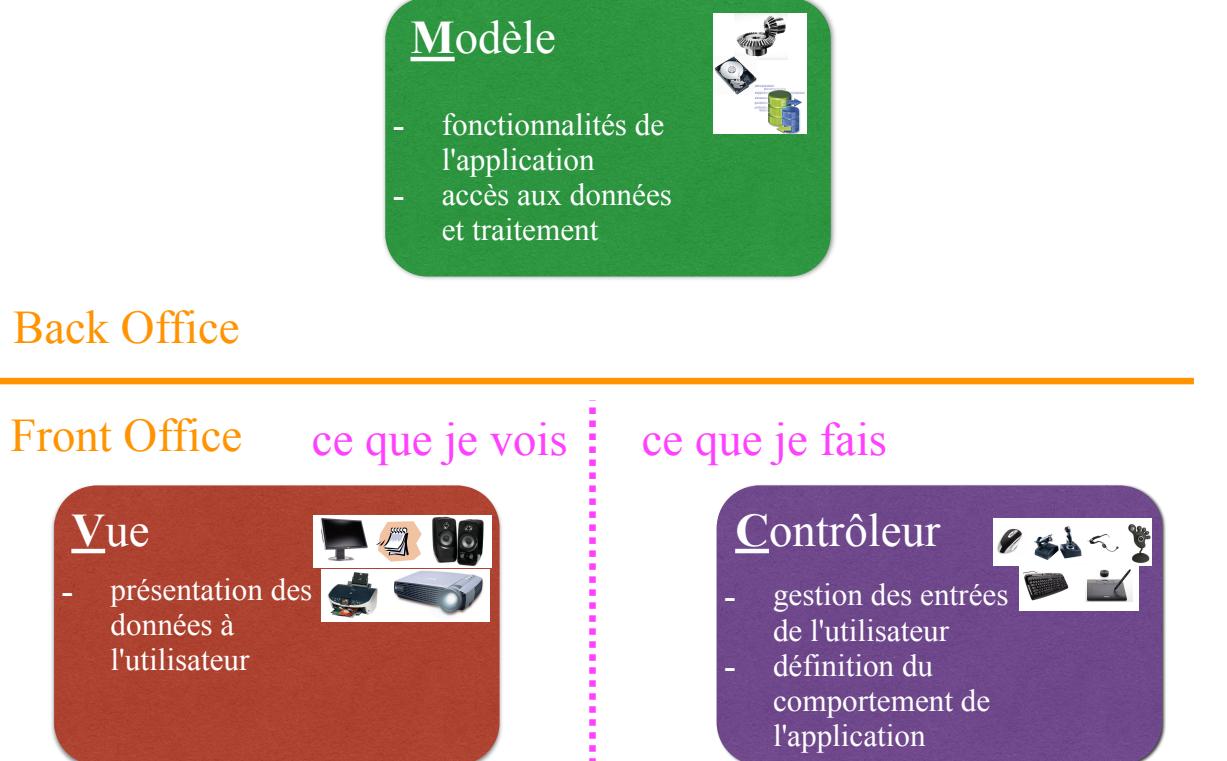
Idée

- **Modèle-Vue-Contrôleur**
 - *MVC*
- MVC est
 - un *patron de conception*
 - ✓ solution standardisée à un problème
 - ✓ indépendante des langages de programmation
 - une *architecture logicielle*
 - ✓ manière de structurer une application ou un ensemble de logiciel
- Introduit en 1979 par Trygve Reenskaug
- Très fortement lié aux principes de la programmation orientée objet
 - smalltalk

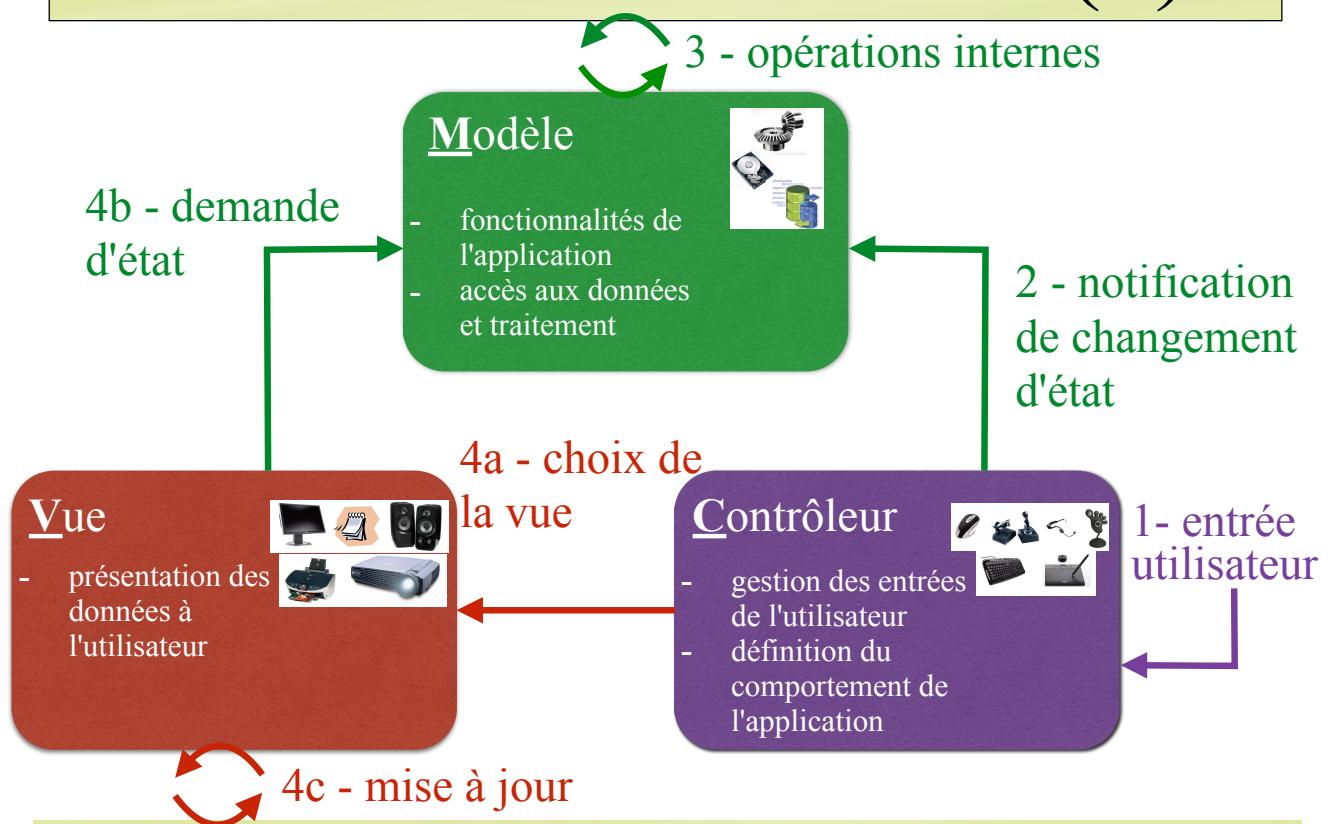
Structure du Modèle

- On organise/structure l'application interactive en séparant
 - les données et leurs traitements
 - ✓ le **modèle**
 - la représentation des données
 - ✓ la **vue**
 - le comportement de l'application
 - ✓ le **contrôleur**

Structure du Modèle (2)

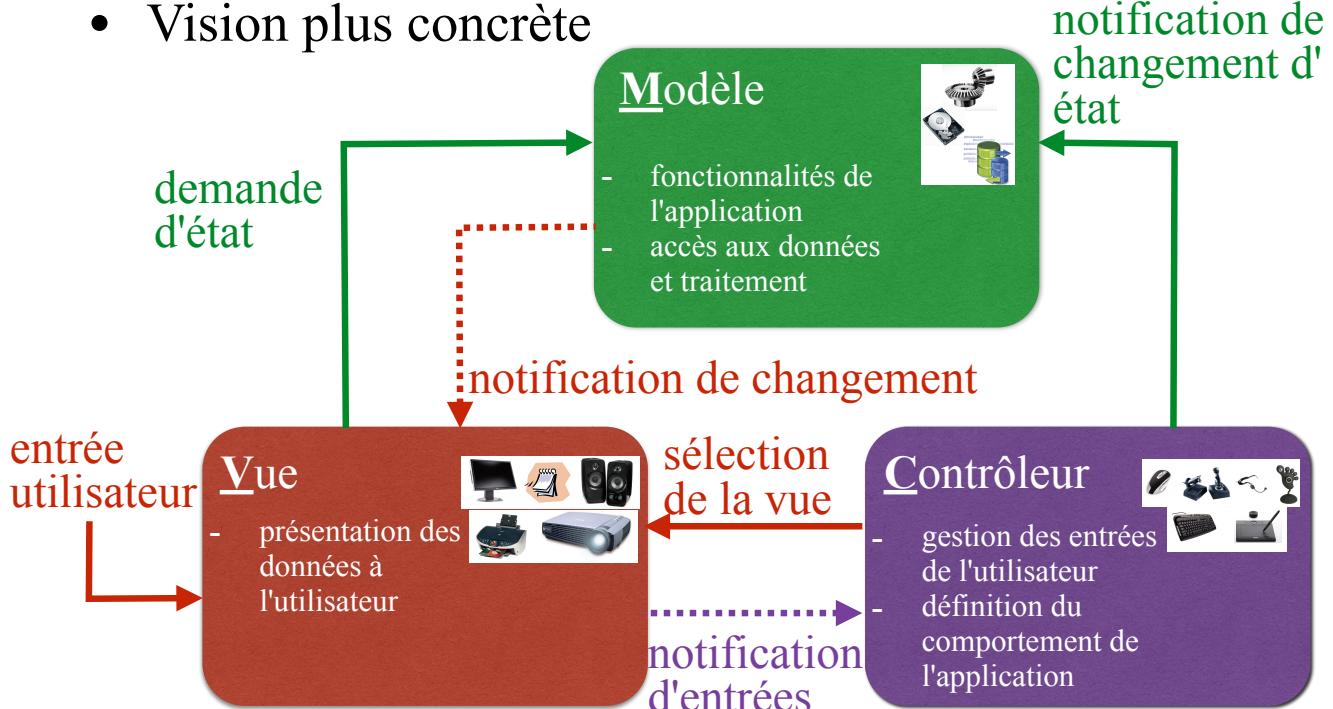


Structure du Modèle (3)



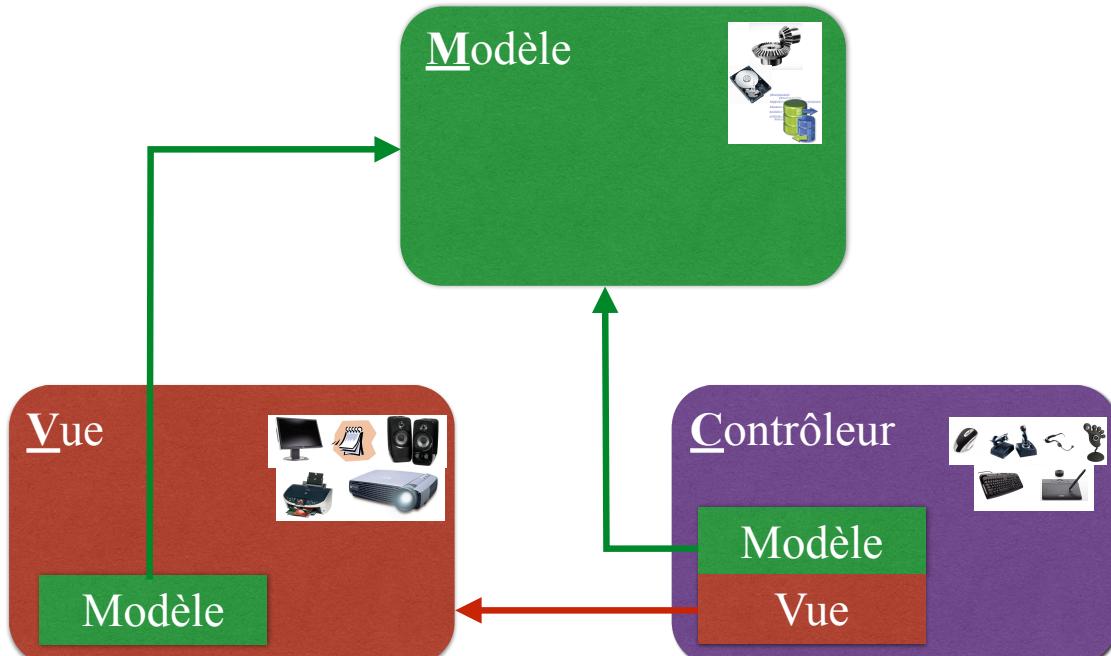
Structure du Modèle (4)

- Vision plus concrète



Structure du Modèle (5)

- Références entre composants



Agenda

- Chapitre 3: Pattern MVC
 - Principe
 - Le Pattern
 - ✓ Modèle
 - ✓ Vue
 - ✓ Contrôleur
 - Application
 - Discussion

Modèle

- Le *modèle*
 - représente les données
 - fournit les accès aux données
 - fournit les traitements applicables aux données
 - expose les fonctionnalités de l'application
- Noyau fonctionnel de l'application

Vue

- La *vue*
 - représente la (ou une) représentation des données du modèle
 - assure la consistance entre
 - ✓ la représentation qu'elle donne
 - ✓ et l'état du modèle/le contexte de l'application
- Sorties de l'application

Contrôleur

- Le *contrôleur*
 - représente le comportement de l'application face aux actions de l'utilisateur
 - fournit la traduction des actions de l'utilisateur en actions sur le modèle
 - fournit la vue appropriée par rapport aux actions de l'utilisateur et des réactions du modèle
- Comportement et gestion des entrées de l'application

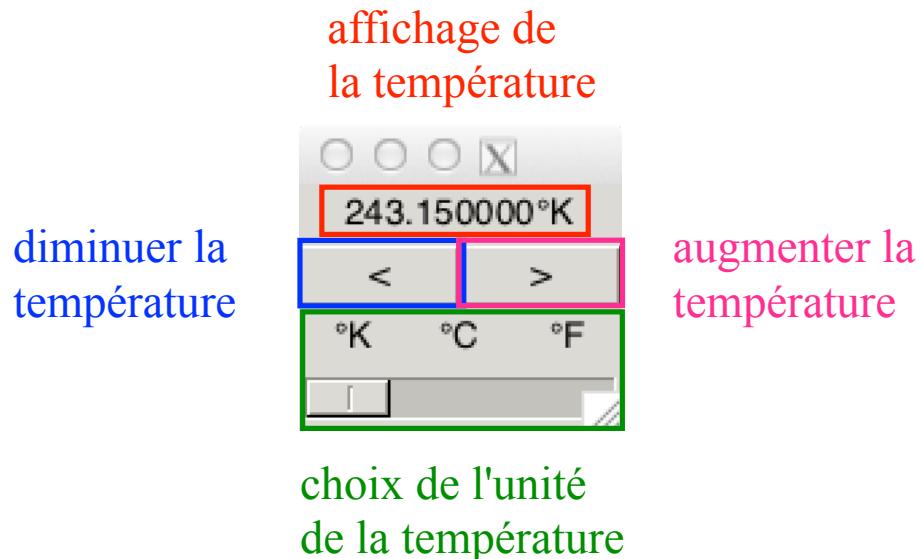
Agenda

- Chapitre 3: Pattern MVC
 - Principe
 - Le Pattern
 - Application
 - ✓ Problème
 - ✓ Analyse
 - ✓ Implémentation
 - Discussion

Problème

- Réaliser une application interactive simulant un thermomètre
 - l'utilisateur peut agir pour contrôler la température
- L'application fournira
 - un affichage textuel de la température courante mesurée par le thermomètre en °C, en °F, ou en °K
 - des contrôles permettant à l'utilisateur de diminuer/ augmenter la température courante du thermomètre
 - un contrôle permettant de choisir l'unité d'affichage de la température

Problème (2)



Analyse

- Le *modèle*
 - données et traitements réalisés
 - ✓ température courante
 - ✓ maintient de l'état de la température courante
 - par défaut, on maintient en °K
 - ✓ conversion de la température en différentes unités
 - fonctionnalités exposées
 - ✓ augmenter la température de 1°
 - si elle ne dépasse pas le maximum autorisé de l'unité
 - ✓ diminuer la température de 1°
 - si elle ne descend pas en dessous du minimum autorisé de l'unité
 - ✓ donner la température en fonction de l'unité choisie

Analyse (2)

```
void rechauffement(ModeleThermometre *mt);  
void refroidissement(ModeleThermometre *mt);
```



modele_thermometre.h



```
double temperature_kelvin(ModeleThermometre *mt);  
double temperature_celsius(ModeleThermometre *mt);  
double temperature_farenheit(ModeleThermometre *mt);
```

Analyse (3)

- La *vue*

- affiche la température courante sous forme de texte
- adapte son affichage à l'unité courante

```
void redessiner(VueThermometre *vt);  
void regler_unite(VueThermometre *vt,  
                  UniteThermometre unite);
```



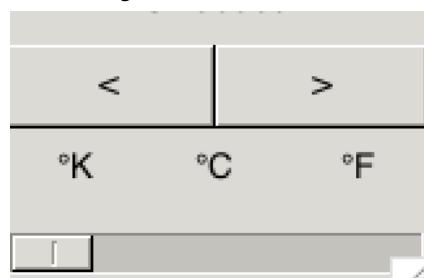
vue_thermometre.h

14.000000°C

Analyse (4)

- Le *contrôleur*

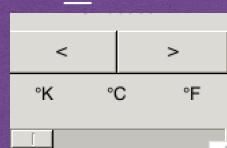
- fournit à l'utilisateur les contrôles sur le modèle
 - ✓ augmenter la température
 - ✓ diminuer la température
- traduit les actions de l'utilisateur en opérations sur le modèle
 - ✓ déclenche les traitements par des appels de fonctions/ procédures du modèle
- sélectionne et met à jour la vue



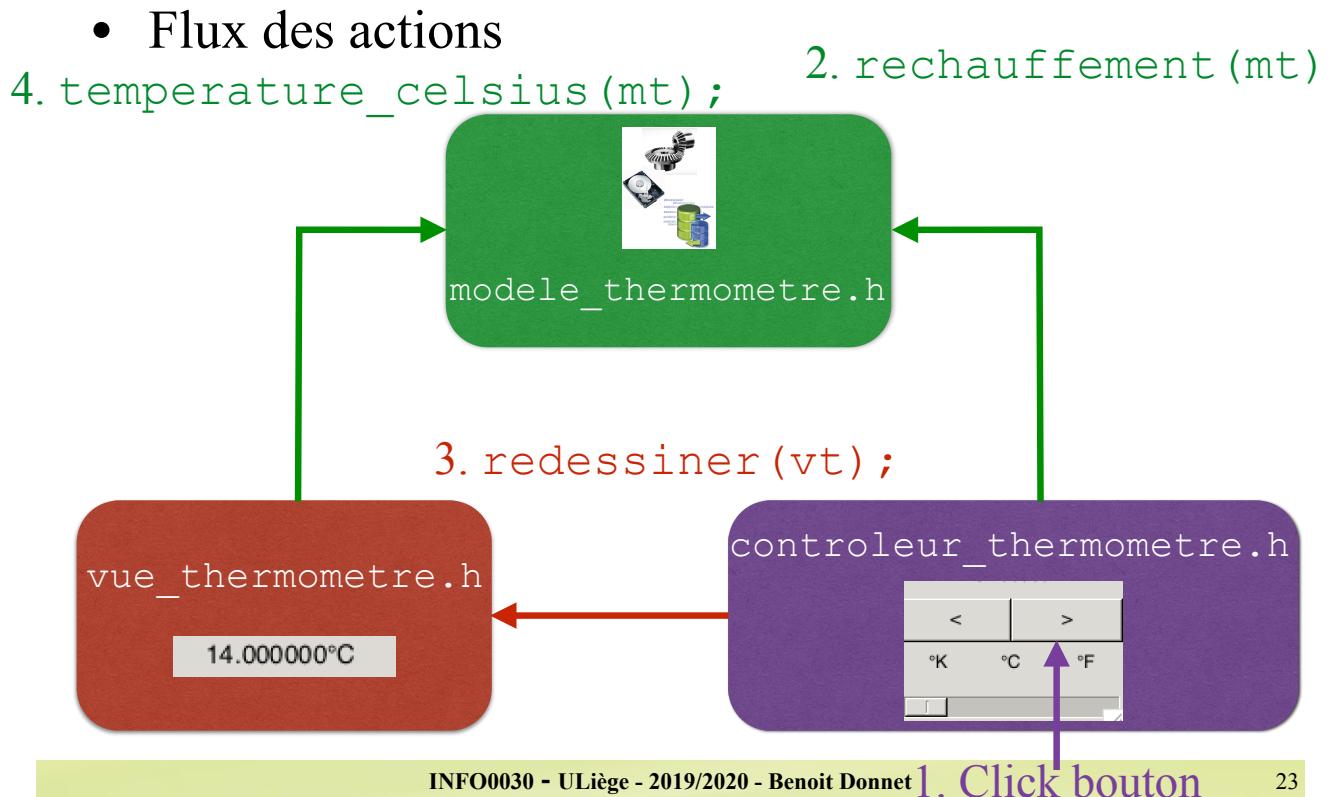
Analyse (5)



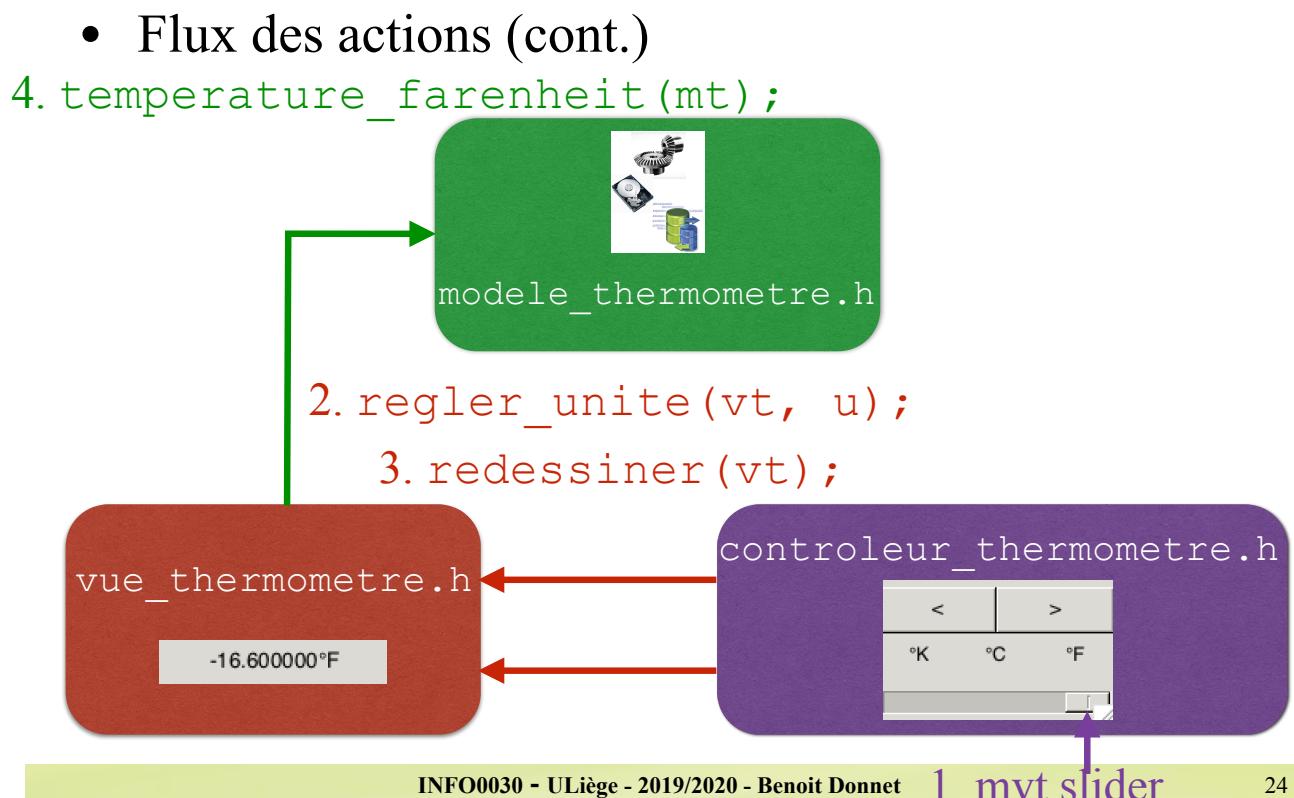
contrroleur_thermometre.h



Analyse (6)



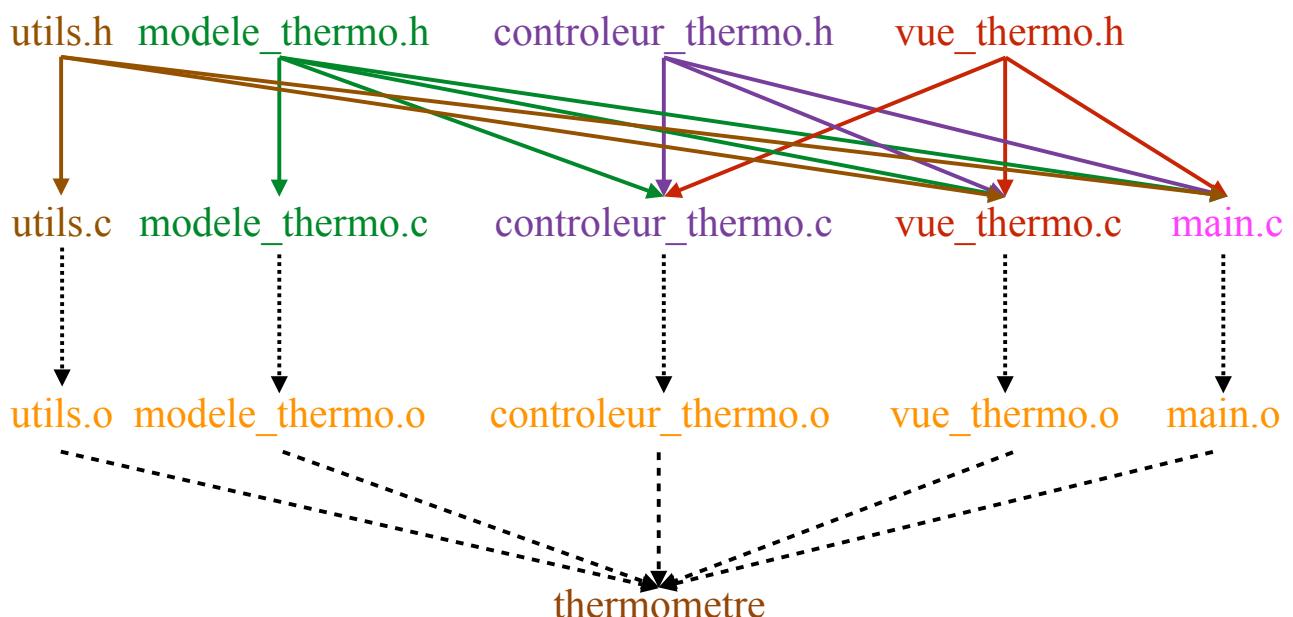
Analyse (7)



Implémentation

- Structure du projet
 - 1 binaire
 - ✓ thermometre
 - 1 programme
 - ✓ main_thermometre.c
 - 4 modules
 - ✓ contrroleur_thermometre.{h/c}
 - ✓ vue_thermometre.{h/c}
 - ✓ modele_thermometre.{h/c}
 - ✓ utils.{h/c}
 - gestion de la documentation via Doxygen
- Les règles de dépendances sont les suivantes
 - main_thermometre dépend de contrroleur_thermometre, vue_thermometre, modele_thermometre
 - un fichier doc_conf a été généré

Implémentation (2)



Implémentation (3)

- Le modèle
 - `modele_thermometre.h`

```
#ifndef __MODELE_THERMOMETRE__
#define __MODELE_THERMOMETRE__

#define MINIMUM_CELSIUS -30
#define MAXIMUM_CELSIUS 70
#define MINIMUM_FARENHEIT -30
#define MAXIMUM_FARENHEIT 220
#define MINIMUM_KELVIN 0
#define MAXIMUM_KELVIN 500

typedef struct modele_t{
    double temperature;
}ModeleThermometre;
```

Implémentation (3)

- Le modèle (suite)
 - `modele_thermometre.h`

```
ModeleThermometre *creer_modele_thermometre(double
temperature);

void rechauffement(ModeleThermometre *mt);
void refroidissement(ModeleThermometre *mt);

double temperature_kelvin(ModeleThermometre *mt);
double temperature_celsius(ModeleThermometre *mt);
double temperature_farenheit(ModeleThermometre *mt);
```

Implémentation (4)

- Le modèle (suite)
 - `modele_thermometre.c`

```
void rechauffement(ModeleThermometre *mt){  
    assert(mt!=NULL);  
  
    if(mt->temperature < MAXIMUM_KELVIN)  
        mt->temperature++;  
}//fin rechauffement()  
  
void refroidissement(ModeleThermometre *mt){  
    assert(mt!=NULL);  
  
    if(mt->temperature > MINIMUM_KELVIN)  
        mt->temperature--;  
}//fin refroidissement()
```

Implémentation (5)

- Le modèle (suite)
 - `modele_thermometre.c`

```
double temperature_kelvin(ModeleThermometre *mt){  
    assert(mt!=NULL);  
  
    if(mt->temperature < MINIMUM_KELVIN)  
        return MINIMUM_KELVIN;  
  
    if(mt->temperature > MAXIMUM_KELVIN)  
        return MAXIMUM_KELVIN;  
  
    return mt->temperature;  
}//fin temperature_kelvin()
```

Implémentation (6)

- Le modèle (suite)
 - `modele_thermometre.c`

```
double temperature_celsius(ModeleThermometre *mt){  
    assert(mt!=NULL);  
  
    double temperature_tmp = mt->temperature - 273.15;  
  
    if(temperature_tmp < MINIMUM_CELSIUS)  
        temperature_tmp = MINIMUM_CELSIUS;  
  
    if(temperature_tmp > MAXIMUM_CELSIUS)  
        temperature_tmp = MAXIMUM_CELSIUS;  
  
    return temperature_tmp;  
}//fin temperature_celsius()
```

Implémentation (7)

- La vue
 - `vue_thermometre.h`

```
#ifndef __VUE_THERMOMETRE__  
#define __VUE_THERMOMETRE__  
  
#include <gtk/gtk.h>  
#include "modele_thermometre.h"  
#include "controleur_thermometre.h"  
  
typedef enum{  
    kelvin, celsius, farenheit  
}UniteThermometre;  
  
typedef struct vue_t{  
    UniteThermometre u;  
    ModeleThermometre *mt;  
    GtkWidget *pLabel;  
}VueThermometre;
```

Implémentation (8)

- La vue (suite)
 - vue_thermometre.h

```
VueThermometre *creer_vue_thermometre(ModeleThermometre  
*mt);  
void redessiner(VueThermometre *vt);  
void regler_unite(VueThermometre *vt, UniteThermometre  
unite);
```

Implémentation (9)

- La vue (suite)
 - vue_thermometre.c

```
void redessiner(VueThermometre *vt){  
    double temperature_courante = 0;  
    char *sLabel;  
    char dLabel[7] = "\u00B0";  
  
    switch(vt->u){  
        case kelvin:  
            temperature_courante = temperature_kelvin(vt->mt);  
            strcat(dLabel, "K");  
            break;  
        //à suivre  
    } //fin switch  
  
    //à suivre  
} //fin redessiner()
```

Implémentation (10)

```
void redessiner(VueThermometre *vt){
    //cfr. slide précédent

    switch(vt->u){
        //cfr. slide précédent
        case celsius:
            temperature_courante = temperature_celsius(vt->mt);
            strcat(dLabel, "C");
            break;
        case farenheit:
            temperature_courante = temperature_farenheit(
                vt->mt);
            strcat(dLabel, "F");
            break;
    } //fin switch

    sLabel = double_vers_char(temperature_courante);
    strcat(sLabel, dLabel);
    gtk_label_set_text(GTK_LABEL(vt->pLabel), sLabel);
} //fin redessiner()
```

Implémentation (11)

- Le contrôleur
 - contrroleur_thermometre.h

```
#ifndef __CONTROLEUR_THERMOMETRE__
#define __CONTROLEUR_THERMOMETRE__

#include <gtk/gtk.h>

#include "modele_thermometre.h"
#include "vue_thermometre.h"

typedef struct controleur_t{
    struct vue_t *vt;
    ModeleThermometre *mt;
    GtkWidget *pBoutonAugmenter;
    GtkWidget *pBoutonDiminuer;
    GtkWidget *pSlider;
}ControleurThermometre;
```

Implémentation (12)

- Le contrôleur (suite)
 - `controlleur_thermometre.h`

```
ControleurThermometre
*creer_controleur_thermometre(struct vue_t *vt,
                                ModeleThermometre *mt);

void click_diminuer(GtkWidget *pF, gpointer data);
void click_augmenter(GtkWidget *pF, gpointer data);
void click_scrollbar(GtkWidget *pWidget, gpointer data);
```

Implémentation (13)

- Le contrôleur (suite)
 - `controlleur_thermometre.c`

```
void click_diminuer(GtkWidget *pF, gpointer data){
    ControleurThermometre *ct =
        (ControleurThermometre *)data;

    refroidissement(ct->mt);
    redessiner(ct->vt);
} //fin click_diminuer()
```

Implémentation (14)

- Le contrôleur (suite)
 - `controleur_thermometre.c`

```
void click_scrollbar(GtkWidget *pWidget, gpointer data){  
    gint iValeur;  
    ControleurThermometre *ct =  
        (ControleurThermometre *)data;  
  
    // Récupération de la valeur de la scrollbar  
    iValeur = gtk_range_get_value(GTK_RANGE(pWidget));  
  
    //modification de l'unité  
    regler_unite(ct->vt, iValeur);  
  
    //redessiner  
    redessiner(ct->vt);  
} //fin click_scrollbar()
```

Implémentation (15)

- Le programme
 - `main_thermometre.c`

```
#include <stdlib.h>  
#include <gtk/gtk.h>  
#include <string.h>  
  
#include "modele_thermometre.h"  
#include "vue_thermometre.h"  
#include "controleur_thermometre.h"  
#include "utils.h"
```

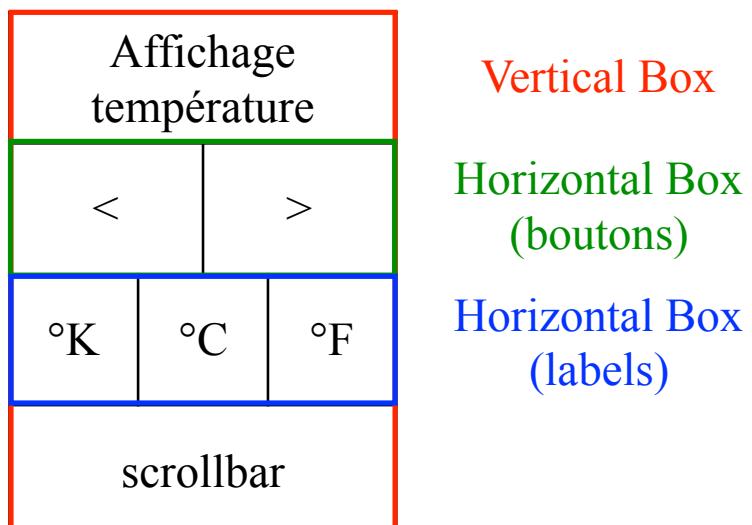
Implémentation (16)

- Le programme (suite)
 - main_thermometre.c

```
int main(int argc, char **argv){  
    ModeleThermometre *mt = creer_modele_thermometre(240);  
    if(mt==NULL)  
        return EXIT_FAILURE;  
  
    VueThermometre *vt = creer_vue_thermometre(mt);  
    if(vt==NULL)  
        return EXIT_FAILURE;  
  
    ControleurThermometre *ct =  
        creer_controleur_thermometre(vt, mt);  
    if(ct==NULL)  
        return EXIT_FAILURE;  
  
    //lancement de l'IHM  
} //fin programme
```

Implémentation (17)

- Détails sur la partie IHM proprement dite



Agenda

- Chapitre 3: Pattern MVC
 - Principe
 - Le Pattern
 - Application
 - Discussion
 - ✓ Avantages du MVC
 - ✓ Inconvénients du MVC

Avantages

- Structuration "propre" de l'application
- Indépendance
 - données
 - représentations
 - comportements
- Parfaite adaptation à l'*Orienté Objet*
- Modulaire et réutilisable
 - vues interchangeables
 - contrôleurs interchangeables
- Facilite les vues et contrôleurs multiples
 - synchronisation quasi implicite

Inconvénients

- Adaptation parfaite à l'*Orienté Objet*
 - il y a une certaine "lourdeur" en C
- Mise en place complexe dans le cas d'applications importantes
- Mises à jour potentiellement nombreuses dans le code
 - "spaghettis" dans le code
 - temps d'exécution
- Contrôleur et vue restent souvent fortement liés au modèle