Architecture d'un programme interactif graphique

Partie 2: Programmation par événement

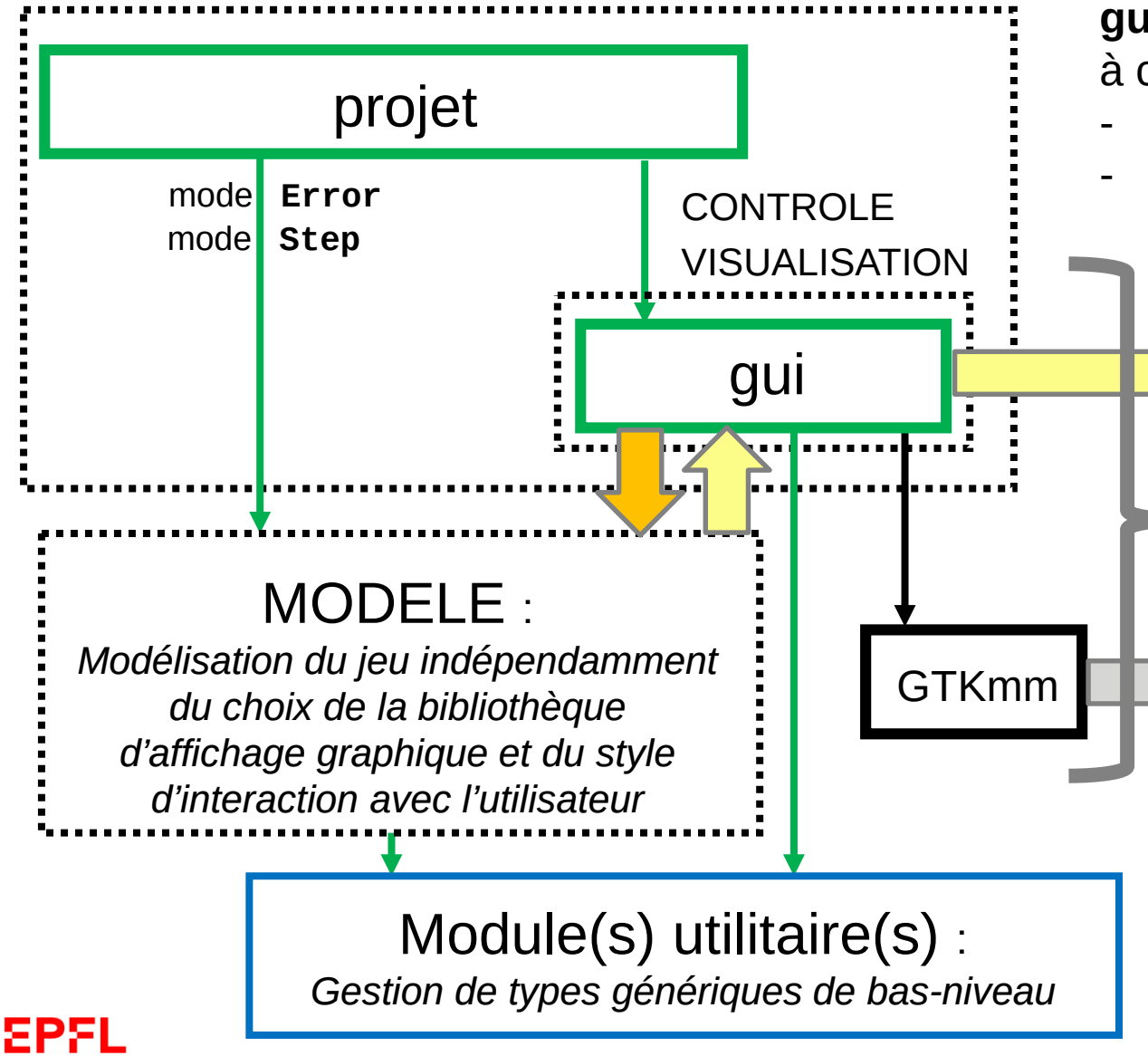
Objectifs:

- maîtriser le concept de programmation par événement

Plan:

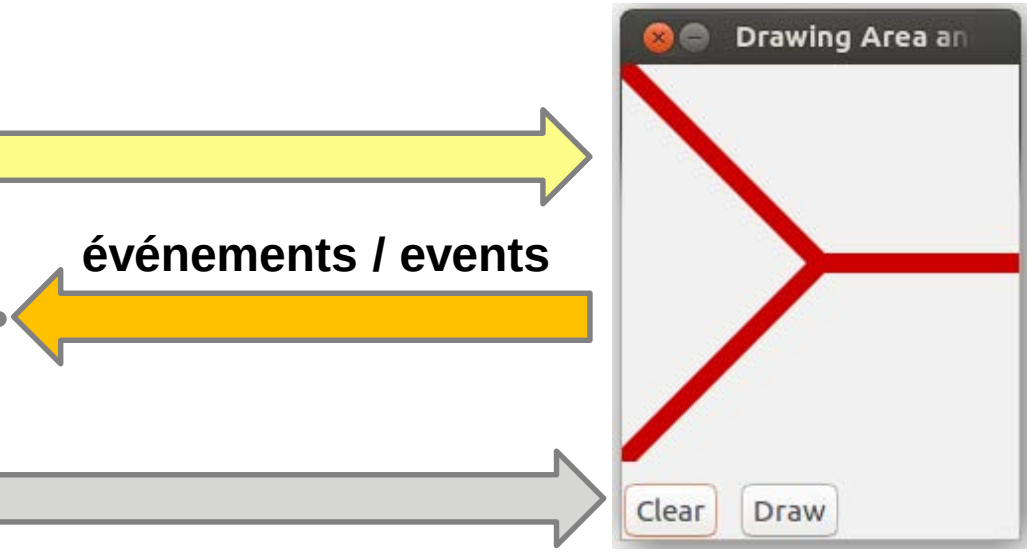
- Programmation par événements
- Exemple détaillé

La programmation par événements



gui exécute les **signal handlers** (*réaction*) associés à certains événements (clic sur boutons, etc…):

- Transfère des commandes au Modèle, etc…
- Met à jour l'affichage du Modèle



GTKmm est responsable de mettre à jour l'affichage des widgets (si nécessaire).

Séparation des fonctionalités au niveau de l'interface graphique

Pour réaliser ou mettre à jour le **dessin**: **dériver** un widget de **DrawingArea**

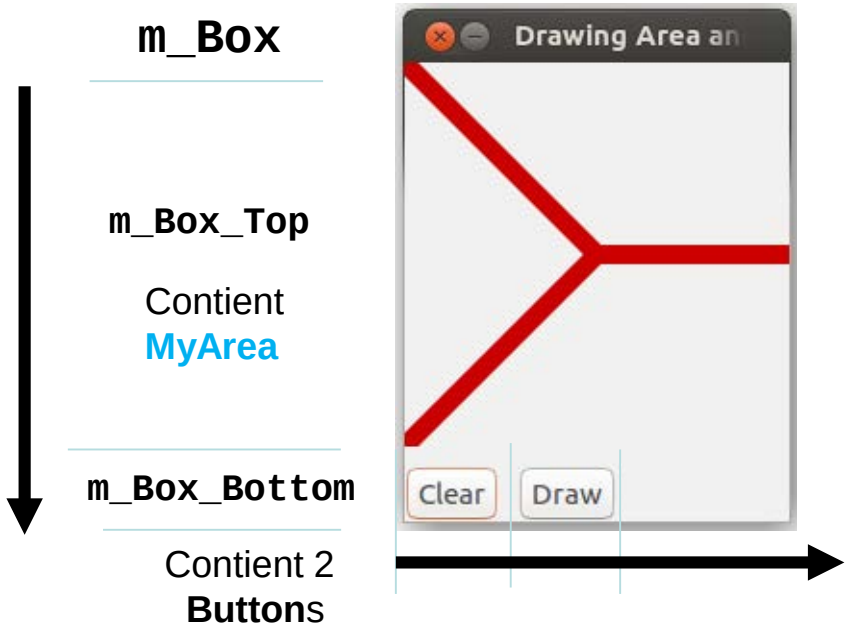
- Appelons ce widget dérivé **MyArea**

Pour définir une interface graphique: **dériver** un widget de **Window**

- Ce widget **possède** des attributs dont, par ex. : Buttons, **MyArea**, Box…

La mise en page est obtenue avec le conteneur **Gtk::Box**

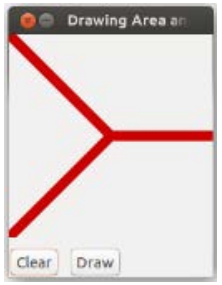
- Un conteneur peut contenir d'autres conteneurs
- A sa création on précise si l'ajout du contenu se fait **horizontalement** ou **verticalement**



Exemple MyEvent(1)

En vert les nouveautés de MyArea
par rapport à l'exemple de la semaine dernière

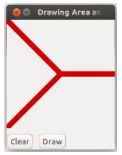
myevent.h




```
…
class MyArea : public Gtk::DrawingArea
{
public:
  MyArea();
  virtual ~MyArea();
  void clear();
  void draw();

protected:
  //Override default signal handler:
  bool on_draw(const
        Cairo::RefPtr<Cairo::Context>& cr)
        override;

private:
  bool empty;
  void refresh();
};
…
```

```
…
class MyEvent : public Gtk::Window
{
public:
  MyEvent();
  virtual ~MyEvent();

protected:
  //Button Signal handlers:
  void on_button_clicked_clear();
  void on_button_clicked_draw();

  Gtk::Box m_Box, m_Box_Top, m_Box_Bottom;
  MyArea       m_Area;
  Gtk::Button  m_Button_Clear;
  Gtk::Button  m_Button_Draw;

private:
  void draw();
};
…
```

Exemple MyEvent(2)



myevent.cc

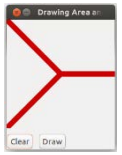
```
…
void MyArea::clear()
{
  empty = true;
  refresh();
}

void MyArea::draw()
{
  empty = false;
  refresh();
}

void MyArea::refresh()
{
  auto win = get_window();
  if(win)
  {
     Gdk::Rectangle r(0,0,
        get_allocation().get_width(),
        get_allocation().get_height());
     win->invalidate_rect(r,false);
  }
}
…
```

```
…
bool MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>&
cr)
{
  if(not empty)
  {
        // ici dessin comme avant
  }
  else
  {
          cout << "Empty !" << endl;
  }

  return true;
}

…
```

Cette méthode produit un *signal*
qui lui-même produit l'appel de **on_draw**

EPFL

Exemple MyEvent(3)

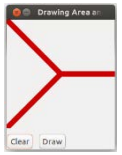


myevent.cc

Partie layout / mise en place des éléments de l'interface

```
…
 // Add outer box to the window
  add(m_Box);

  //Fill the outer box:
  m_Box.pack_start(m_Box_Top);
  m_Box.pack_start(m_Box_Bottom);

  m_Area.set_size_request(200,200); // permet de contrôler la taille de l'espace du dessin
  m_Box_Top.pack_start(m_Area);

  m_Box_Bottom.pack_start(m_Button_Clear,false,false);// keep fixed width
  m_Box_Bottom.pack_start(m_Button_Draw,false,false); // and aligned to left;
 …
```

Exemple MyEvent(4)



myevent.cc

Partie «réactive» / mise en place liens entre les boutons et l'action sur l'affichage

```
…
void MyEvent::on_button_clicked_clear()
{
  cout << "Clear" << endl;
  m_Area.clear();
}

void MyEvent::on_button_clicked_draw()
{
  cout << "Draw" << endl;
  m_Area.draw();
}
…
```