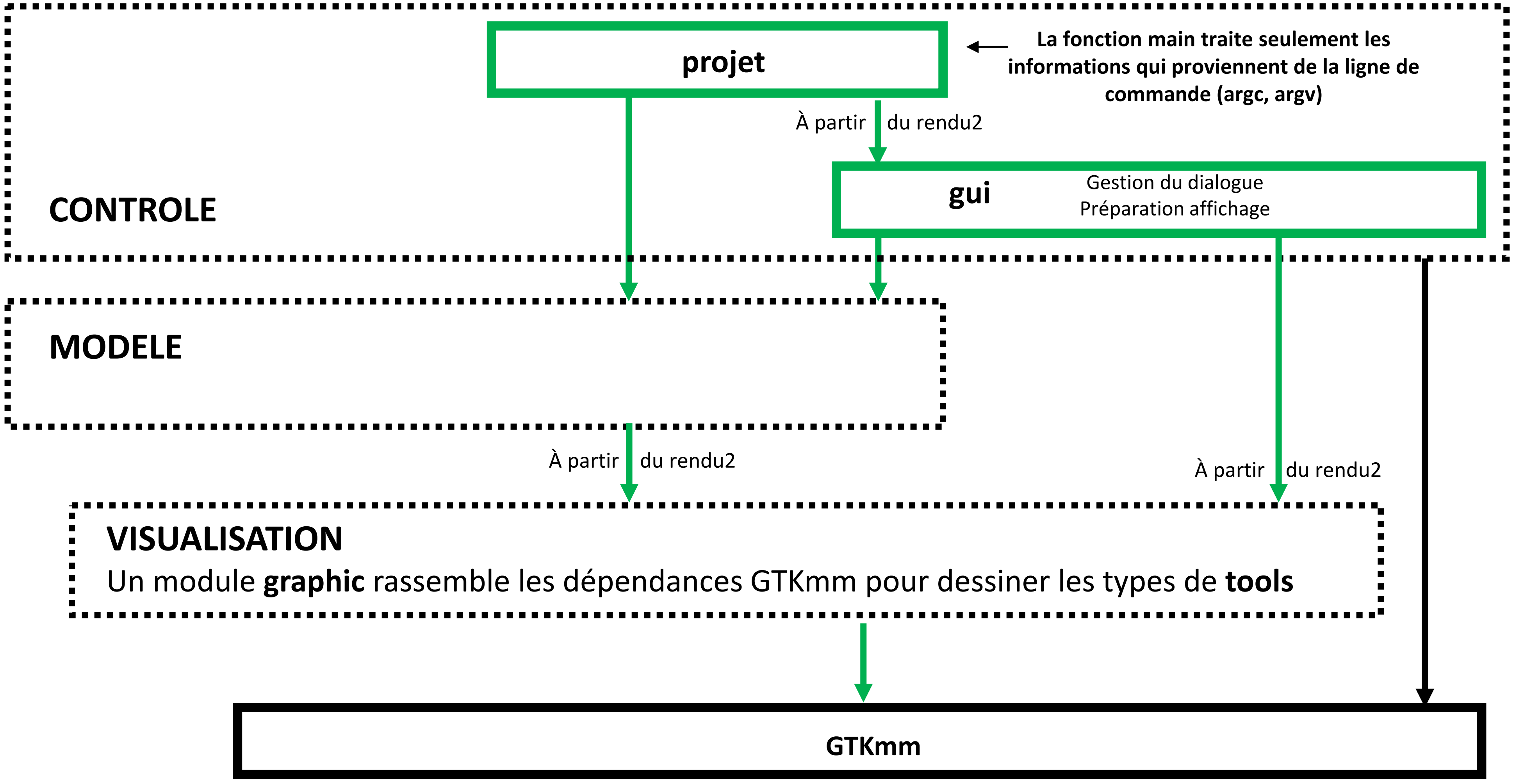
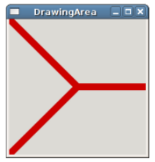
MVC et GTKmm

Comment garantir l'indépendance du Modèle vis-à-vis de GTKmm


projet
La fonction main traite seulement les informations qui proviennent de la ligne de commande (argc, argv)
À partir du rendu2
gui
Gestion du dialogue
Préparation affichage
CONTROLE
MODELE
À partir du rendu2
À partir du rendu2
VISUALISATION
Un module graphic rassemble les dépendances GTKmm pour dessiner les types de tools
GTKmm

On doit transmettre un pointeur `Cairo::Context` au **Modèle**
⇒ Crée une dépendance envers **GTKmm**



myarea.cc

myarea.h

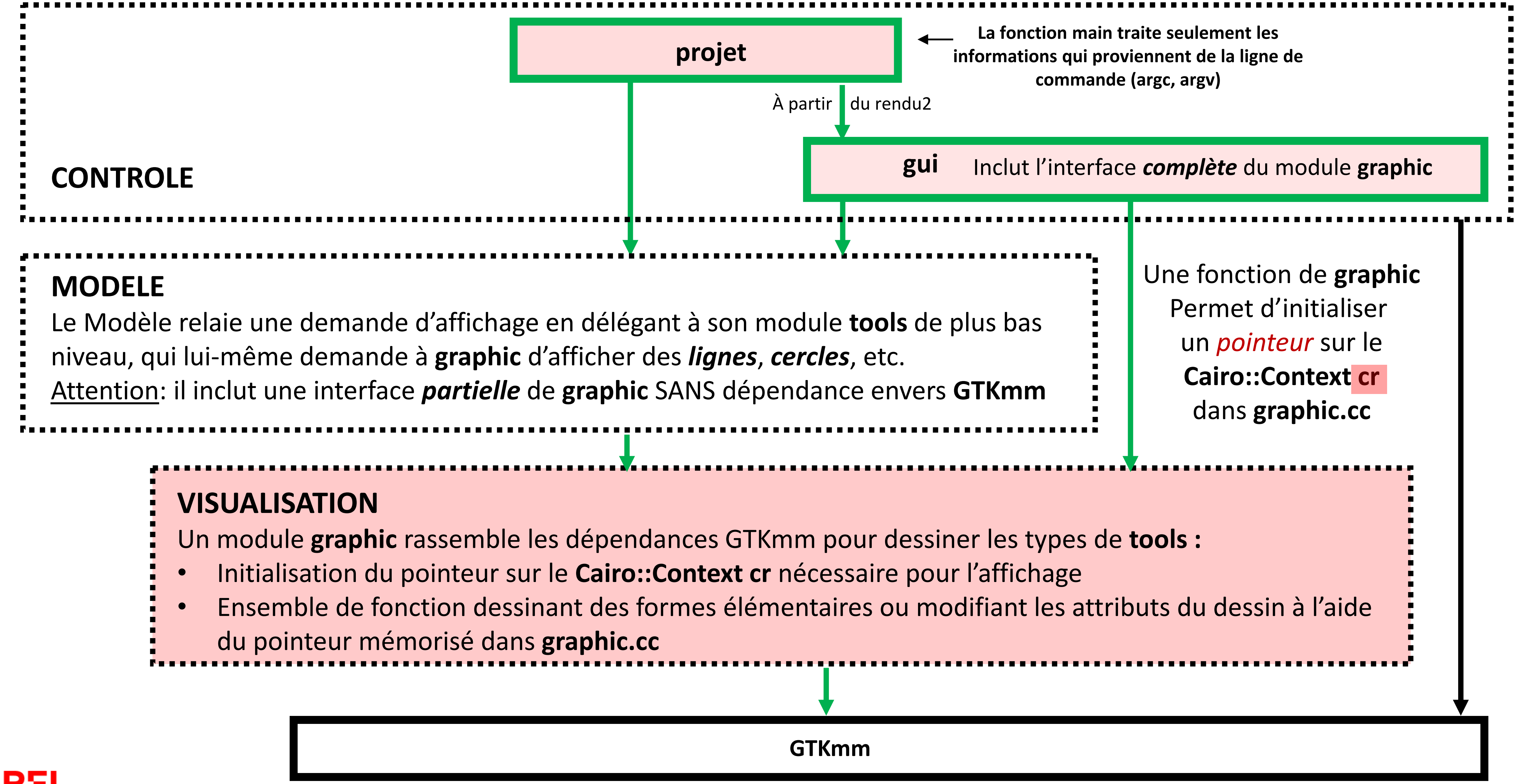
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#include <gtkmm/drawingarea.h>

class MyArea : public Gtk::DrawingArea
{
public:
  MyArea();
  virtual ~MyArea();

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

Problème!

```
#include "myarea.h"
#include <cairomm/context.h>

MyArea::MyArea(){}
MyArea::~MyArea(){}

bool MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
  Gtk::Allocation allocation = get_allocation();
  const int width = allocation.get_width();
  const int height = allocation.get_height();

  // coordinates for the center of the GTKmm window
  int xc, yc;
  xc = width / 2;
  yc = height / 2;

  cr->set_line_width(10.0); // mémorisé à long terme dans cr

  // draw red lines out from the center of the window
  cr->set_source_rgb(0.8, 0.0, 0.0); // idem mémorisation cr
  cr->move_to(0, 0);
  cr->line_to(xc, yc);
  cr->line_to(0, height);
  cr->move_to(xc, yc);
  cr->line_to(width, yc);
  cr->stroke();

  return true;
}
```

Tous les appels définissant les attribut du dessin et effectuant le tracé ont besoin du pointeur Cairo::Contex

Solution : mémoriser un *pointeur* sur **Cairo::Context** **cr** dans **graphic.cc**



projet

La fonction main traite seulement les informations qui proviennent de la ligne de commande (argc, argv)

À partir du rendu2

gui Inclut l'interface ***complète*** du module **graphic**

CONTROLE

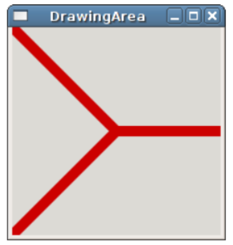
MODELE
Le Modèle relaie une demande d'affichage en délégant à son module **tools** de plus bas niveau, qui lui-même demande à **graphic** d'afficher des ***lignes***, ***cercles***, etc.
Attention: il inclut une interface ***partielle*** de **graphic** SANS dépendance envers **GTKmm**

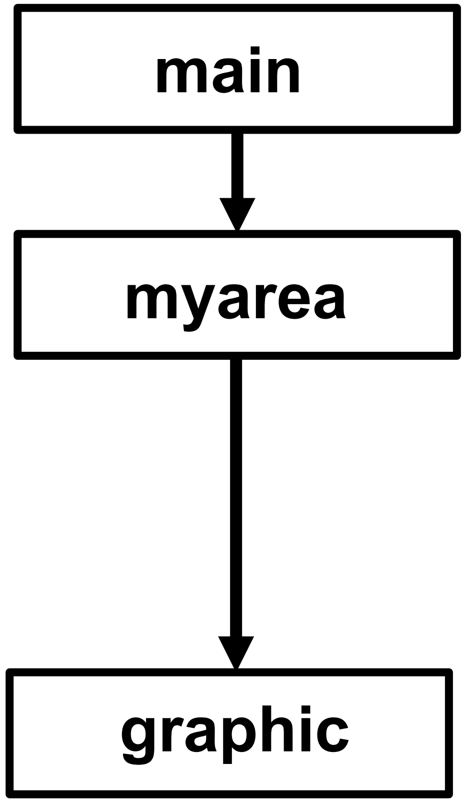
Une fonction de **graphic** Permet d'initialiser un *pointeur* sur le **Cairo::Context** **cr** dans **graphic.cc**

VISUALISATION
Un module **graphic** rassemble les dépendances GTKmm pour dessiner les types de **tools :**
- Initialisation du pointeur sur le **Cairo::Context cr** nécessaire pour l'affichage
- Ensemble de fonction dessinant des formes élémentaires ou modifiant les attributs du dessin à l'aide du pointeur mémorisé dans **graphic.cc**

GTKmm

Exemple : GTKdrawingArea_avec_deux_modules (1)



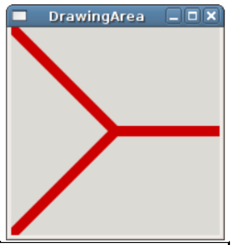


Module équivalent au module **gui** du projet:
- inclut l'interface complète du module graphic **graphic_gui.h**
- utilise la fonction qui initialise le pointeur sur **Cairo::Context cr**
- appelle ensuite une fonction de dessin de **graphic** sans avoir besoin de passer **Cairo::Context cr** en paramètre

Module **graphic** :
- offre une interface ***complète*** **graphic_gui.h**
- offre une seconde interface ***partielle*** **graphic.h** (pour un éventuel Modèle)
- initialise le pointeur **ptcr** sur **Cairo::Context cr**
- Utilise le pointeur **ptcr** pour toutes les commandes de dessin

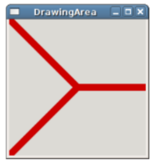
GTKdrawingArea_avec_deux_modules (2)

main.cc



```
#include "myarea.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

int main(int argc, char** argv)
{
   auto app = Gtk::Application::create(argc, argv, "org.gtkmm.example");

   Gtk::Window win;
   win.set_title("DrawingArea");

   MyArea area;
   win.add(area);
   area.show();

   return app->run(win);
}
```

GTKdrawingArea_avec_deux_modules (3)



myarea.h

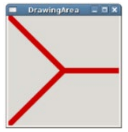
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#include <gtkmm/drawingarea.h>

class MyArea : public Gtk::DrawingArea
{
public:
  MyArea();
  virtual ~MyArea();

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

myarea.cc

```
#include "myarea.h"
#include «graphic_gui.h"
#include <cairomm/context.h>

MyArea::MyArea(){}
MyArea::~MyArea(){}

bool MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr)
{
  Gtk::Allocation allocation = get_allocation();
  const int width = allocation.get_width();
  const int height = allocation.get_height();

  // coordinates for the center of the GTKmm window
  int xc, yc;
  xc = width / 2;
  yc = height / 2;

  graphic_set_context(cr);

  graphic_draw_shape(width, height,xc,yc);

  return true;
}
```

GTKdrawingArea_avec_deux_modules (4)

graphic.h // interface *partielle*



```
#ifndef GTKMM_EXAMPLE_GRAPHIC_H
#define GTKMM_EXAMPLE_GRAPHIC_H

void graphic_draw_shape(const int width,
        const int height, int xc, int yc);

#endif // GTKMM_EXAMPLE_GRAPHIC_H
```

graphic_gui.h // interface ***complète***

```
#ifndef GTKMM_EXAMPLE_GRAPHIC_GUI_H
#define GTKMM_EXAMPLE_GRAPHIC_GUI_H

#include <gtkmm/drawingarea.h>
#include "graphic.h"

void graphic_set_context(const
        Cairo::RefPtr<Cairo::Context>& cr);

#endif // GTKMM_EXAMPLE_GRAPHIC_GUI_H
```

graphic.cc

```
#include "graphic_gui.h"

static const Cairo::RefPtr<Cairo::Context>* ptcr(nullptr);

void graphic_set_context(const Cairo::RefPtr<Cairo::Context>& cr)
{
    static bool init(false);
    if(!init)
    {
        ptcr = &cr;
        init = true;
    }
}

void graphic_draw_shape(const int width, const int height,
                        int xc, int yc)
{
  (*ptcr)->set_line_width(10.0);

  // draw red lines out from the center of the window
  (*ptcr)->set_source_rgb(0.8, 0.0, 0.0);
  (*ptcr)->move_to(0, 0);
  (*ptcr)->line_to(xc, yc);
  (*ptcr)->line_to(0, height);
  (*ptcr)->move_to(xc, yc);
  (*ptcr)->line_to(width, yc);
  (*ptcr)->stroke();
}
```

*initialise le pointeur **ptcr** sur **Cairo::Context cr***

*Utilise le pointeur **ptcr** pour toutes les commandes de dessin*