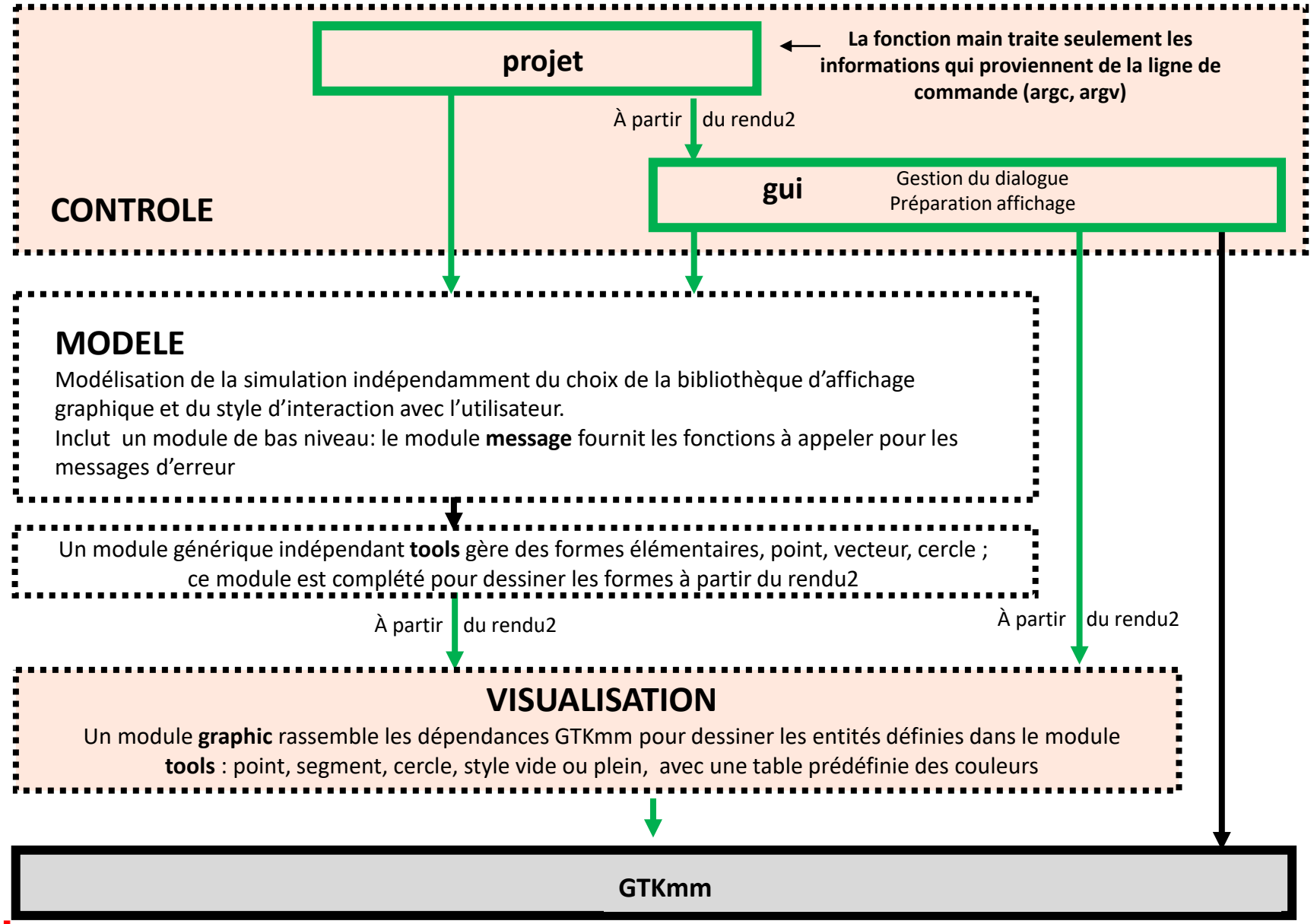
# Architecture MVC et GTKmm4:



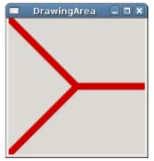

*Comment garantir l'indépendance du Modèle vis-à-vis de GTKmm4 ?*

Le module **graphic** regroupe les dépendances de dessin vis-à-vis d'une bibliothèque externe (GTKmm4).

Le **MODELE** délègue à **shape** la tâche du dessin; grâce à **shape.h** qui contient aussi **graphic.h** on autorise le MODELE à utiliser les symboles prédéfinis pour les couleurs dans **graphic.h**.

On doit transmettre un pointeur `Cairo::Context` au **Modèle**

⇒ Crée une dépendance envers **GTKmm**




myarea.cc

myarea.h

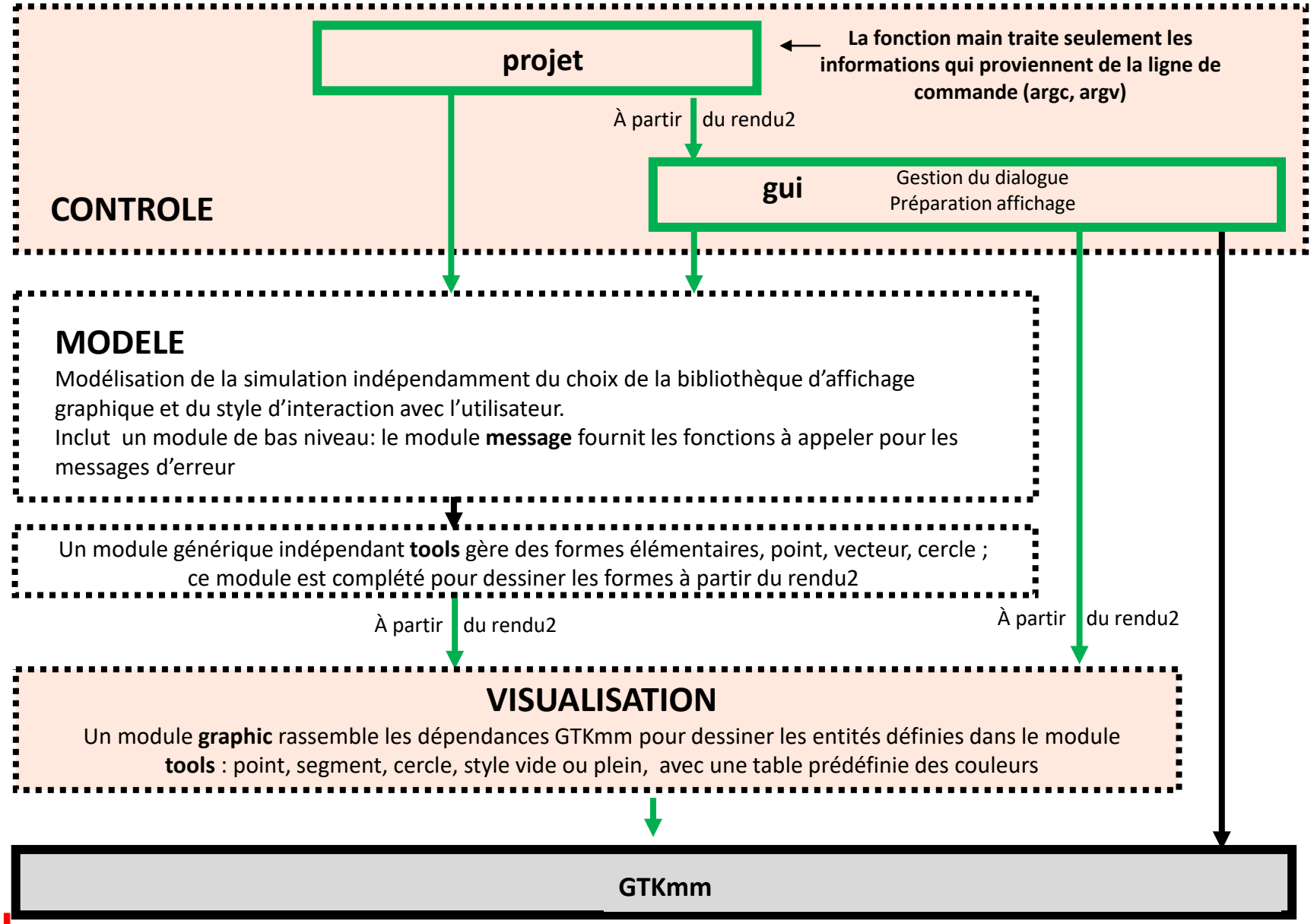
```
#ifndef GTKMM_EXAMPLE_MYAREA_H
#define GTKMM_EXAMPLE_MYAREA_H

#include <gtkmm/drawingarea.h>

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

protected:
  //Override default signal handler:
  void on_draw(const
          Cairo::RefPtr<Cairo::Context>& cr,
          int width, int height);
};
#endif // GTKMM_EXAMPLE_MYAREA_H
```

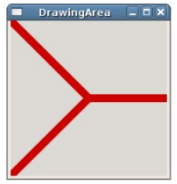
```
#include "myarea.h"
#include <cairomm/context.h>

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

void MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
                     int width, int 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();

}
```

# Problème!

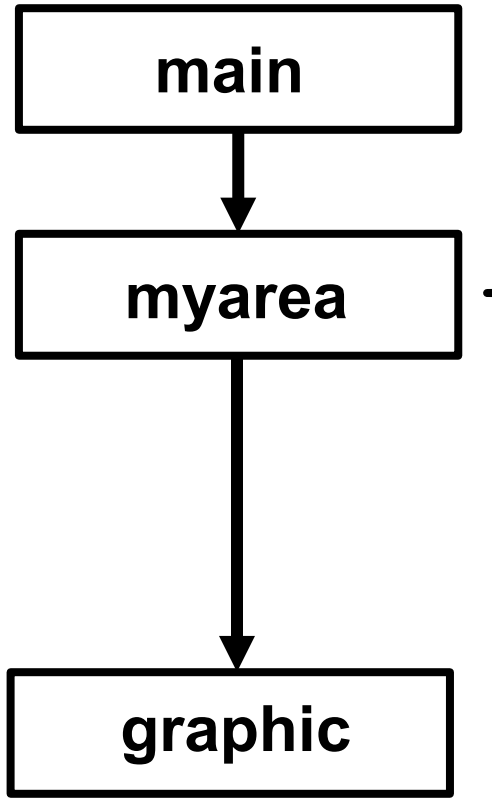
*Tous les appels définissant les attributs du dessin et effectuant le tracé ont besoin du pointeur Cairo::Contex*

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






# 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

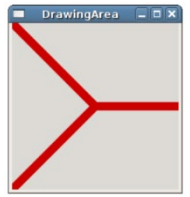
Module **graphic** :

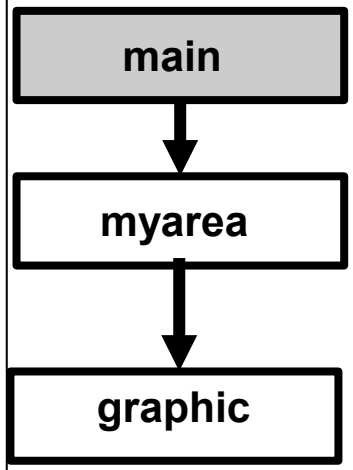
- offre une interface ***complète*** **graphic_gui.h**
- offre une seconde interface ***partielle*** **graphic.h** (projet: pour **tools**)
- Met à jour le pointeur **ptcr** sur **Cairo::Context cr**
- Utilise le pointeur **ptcr** pour toutes les commandes de dessin

# GTKdrawingArea_avec_deux_modules (2)

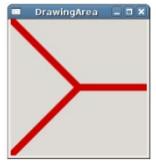
main.cc

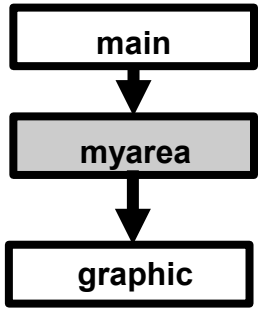
```
#include "myarea.h"
#include <gtkmm/application.h>
#include <gtkmm/window.h>

class ExampleWindow : public Gtk::Window
{
public:
  ExampleWindow();

protected:
  MyArea m_area;
};

ExampleWindow::ExampleWindow()
{
  set_title("DrawingArea");
  set_child(m_area);
}
int main(int argc, char** argv)
{
  auto app = Gtk::Application::create();

  return app->make_window_and_run<ExampleWindow>(argc, argv);
}
```



# GTKdrawingArea_avec_deux_modules (3)



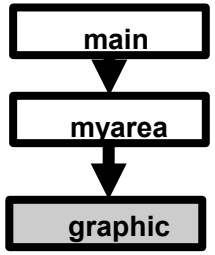



myarea.h

```
#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:
 void on_draw(const
         Cairo::RefPtr<Cairo::Context>& cr,
         int width, int height);
};
#endif // GTKMM_EXAMPLE_MYAREA_H
```

myarea.cc

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

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

void MyArea::on_draw(const Cairo::RefPtr<Cairo::Context>& cr,
                     int width, int height)
{
  graphic_set_context(cr);

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

  graphic_draw_shape(width, height,xc,yc);

}
```




# 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)
{
    ptcr = &cr;
}

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();
}
```

*Met à jour le pointeur **ptcr** sur **Cairo::Context cr***

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

ce mécanisme d'initialisation de **ptcr** sera mis à disposition