Arve Roadshow

Introduction

- **What is Arve and what is it not!**
- **Domain overview**

Domain	Example
• **Subsystems/Modules**	*"Slug"*
• **Control**	
• **Geometry**	
• **Graphics**	*"test_display"*
• **Detector Description**	*"Ecal"*

Introduction

- Developed by Toby Burnett
- Taken as the baseline of the ATLAS OO software.

  Note: I am talking about this first version, not about the additions/changes to come.

- Source code can be found at:

  `/afs/cern.ch/atlas/software/dist/00-00-00/arve`

- More information at:

  `http://wwwinfo.cern.ch/b/burnett/www/arve/`

What is Arve (not)?

X

An object-oriented framework intended to facilitate development of reconstruction and physics analysis code.

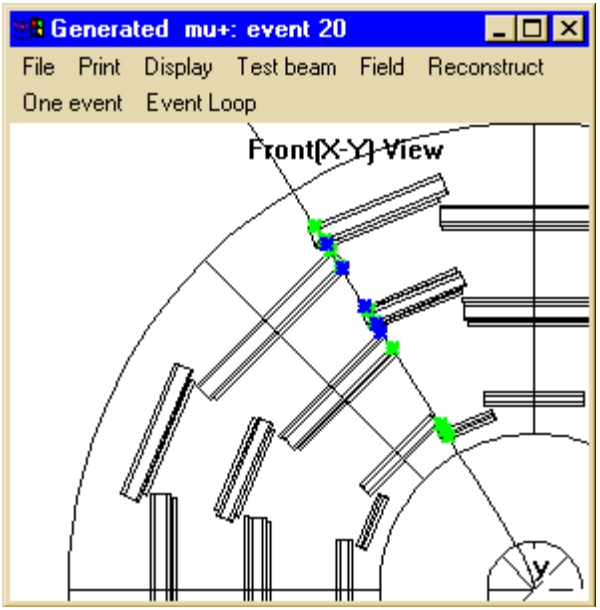
Not a full reconstruction program.

A product primarily intended for developers.

Not for “kumac writers”

What an application looks like

- Title bar
- Menu
- 3-d display of:
  - detector geometry
  - event (tracks)
  - response
  - reconstruction



The Domains

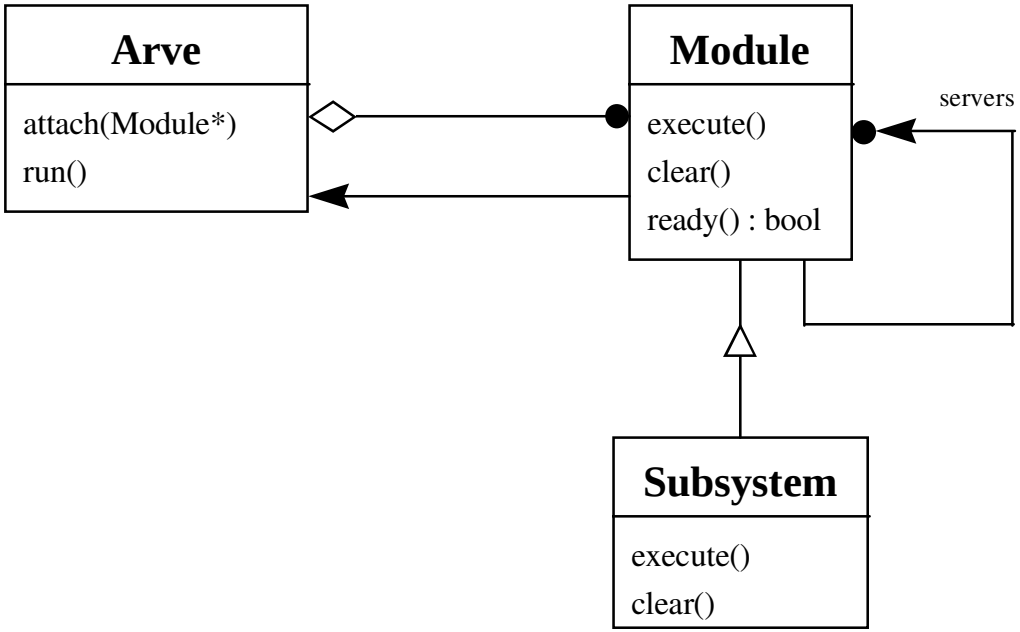
Domain	Description
Subsystems/Modules	For the different detectors, event loaders, etc.
Control	The user interface, command and event loop.
Graphics	3-D display of detector elements, event, etc.
Geometry & Detector Description	Representation of geometric entities and construction of the detector.
Simulation (Gismo)	Material and particle descriptions, generation of events.

Subsystems/Modules

- Any detector naturally divides into different physical subsystems, each geometrically distinct, and mostly autonomous.
- But also “actions” as event generation, combined reconstruction and physics analysis qualify as modules.

- They are executed once per event, and are of course ordered.

Module class diagram



Example “*Slug*” (T. Burnett)

Slug.h

```
#include "arve/SubSystem.h"

class Slug : public SubSystem {
public:
   Slug(Arve* app);
   ~Slug() {};

   void execute();                          // called once per event
   void clear();
private:
    int          m_event_number;
    Point        m_data;                     // pseudo-data!
};
```

Example “*Slug*”

Slug.cxx

```
Slug::Slug(Arve* app) : SubSystem(app), m_event_number(0) {}

void Slug::execute()
{
    strstream title;
    title << "Slug event number " << (++m_event_number) << '\0';
    display().setTitle(title.str() );

    SubSystem::execute();
}

void Slug::clear()
{
    SubSystem::clear();
}
```

Example “*Slug*”

main.cxx

```
#include "arve/Arve.h"
#include "Slug.h"

static float world_size=1000;

int main(int argc, char *argv[])
{
   Arve app(world_size);          // create the application Server

   Slug slug(&app);               // instantiate the subsystems

   app.run();
   return 0;
}
```

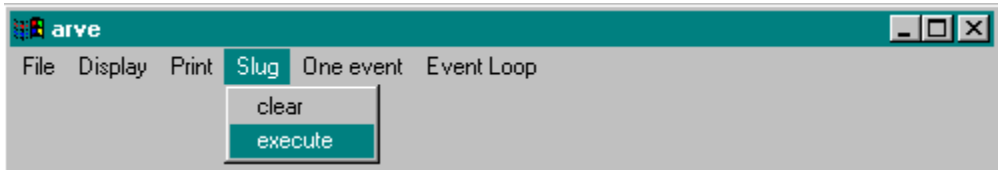
Example “*Slug*” - result

Control

- Controls the console and graphics window.
- Defines menu's and commands (e.g. "EventLoop").

slug.cxx

```
Slug::Slug(Arve* app) : SubSystem(app), m_event_number(0)
{
    SubMenu& sub = menu().sub_menu("Slug");
    sub.add_button("clear", new SimpleCommand<Slug>(this,
                   Slug::clear));
    sub.add_button("execute", new SimpleCommand<Slug>(this,
                   Slug::execute));
}
```



Geometry

- Geometrical base classes:

  Point, *Vector*, *Helix*, *Surface*, *Volume*

- Geometrical entities:

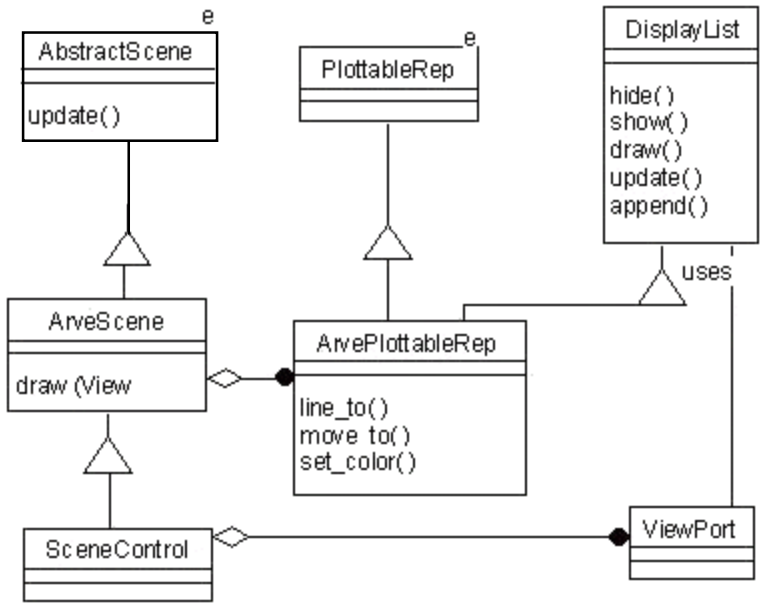
  Box, *Cone*, *Conic*, *Cons*, *Cylinder*, *Sphere*, *Trap*, *Trd1*, *Trd2*, *Tube*, *Wedge*

- Geometrical operations:

  CoordTransform, Intersection

Graphics

- Following the official ATLAS Graphics standard:



- Representations are supplied for all geometrical entities.

Example “*test-display*” (T. Burnett)

main.cxx

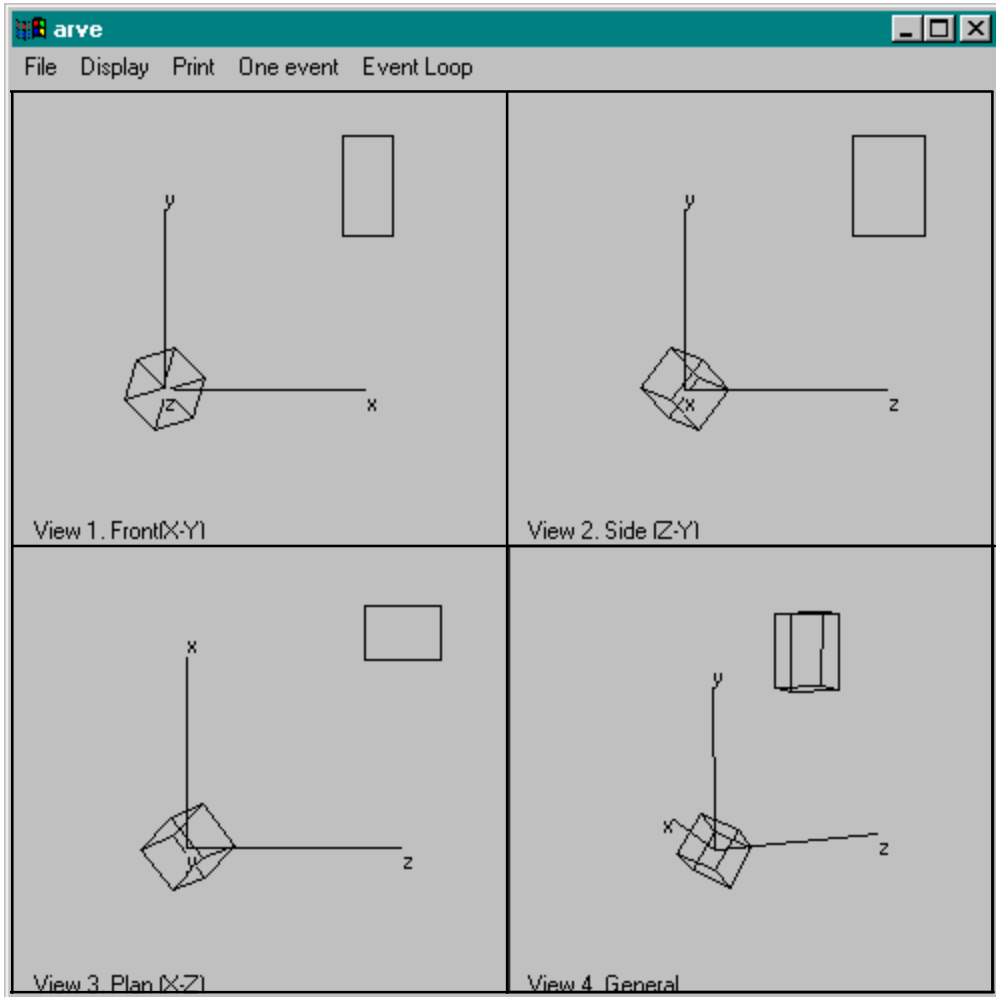
```
static float world_size=500;

int main(int argc, char *argv[])
{
   Arve app(world_size);

   Box box1(25,50,35);
   box1.move(Vector(100, 100, 100));
   BoxRep box_rep1(box1);

   Box box2 (25, 25, 25);
   box2.rotate(1.0, Vector(1, 1, 0));
   BoxRep box_rep2(box2);


...
```

Example “*test-display*”

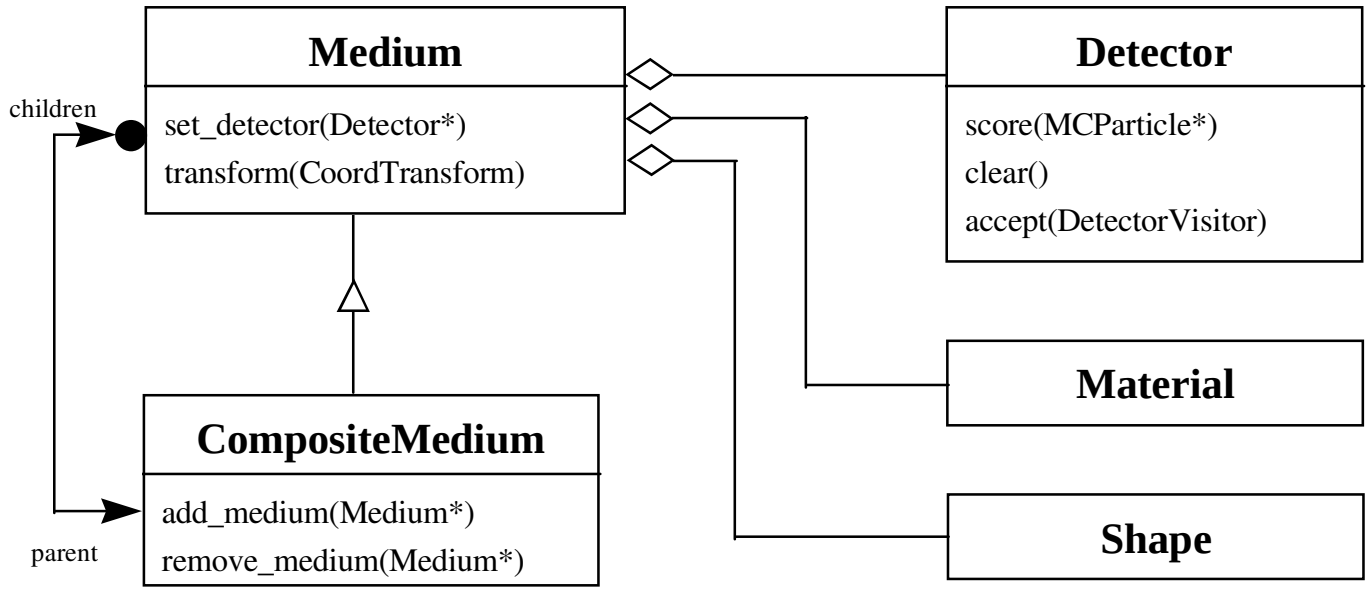
main.cxx - *continued*

```
...

   app.display().add(&box_rep1, "box 1");
   app.display().add(&box_rep2, "box 2");

   app.run();
   return 0;
}
```

Example “*test-display*” - result



Detector Description



Example “*Ecal*” (T. Burnett)

EcalBarrel.h

```
#include "gismo/CompositeMedium.h"

class EcalBarrel : public CompositeMedium
{
public:
   explicit EcalBarrel(Medium* parent);

   // All the rest is handled correctly by CompositeMedium.
   // Most calls are forwarded to the detector.
};
```

Example “*Ecal*”

EcalBarrel.cxx

```
EcalBarrel::EcalBarrel(Medium* parent) :
   CompositeMedium(parent, new Tube(630, 145, 200), "vacuum")
{
   setTitle("Ecal Barrel");

   // create the four layers each containing an active detector
   for( int i = 0; i< 4; i++) {
        Medium* layer = new Medium(this,
             new Tube(630, layer_info[i].r_min,layer_info[i].r_max),
             "ArLiq", new EcalDetector());

        layer->setTitle(layer_info[i].name);
   }
}
```

Example “*Ecal*”

EcalDetector.h

```
#include "gismo/Detector.h"

class EcalDetector : public Detector
{
public:
   EcalDetector();

   virtual void score(MCParticle*) ;
   virtual void clear();

   virtual void draw(ArvePlottableRep&) const
private:
   EcalData      m_data;
};
```

Example “*Ecal*”

EcalDetector.cxx

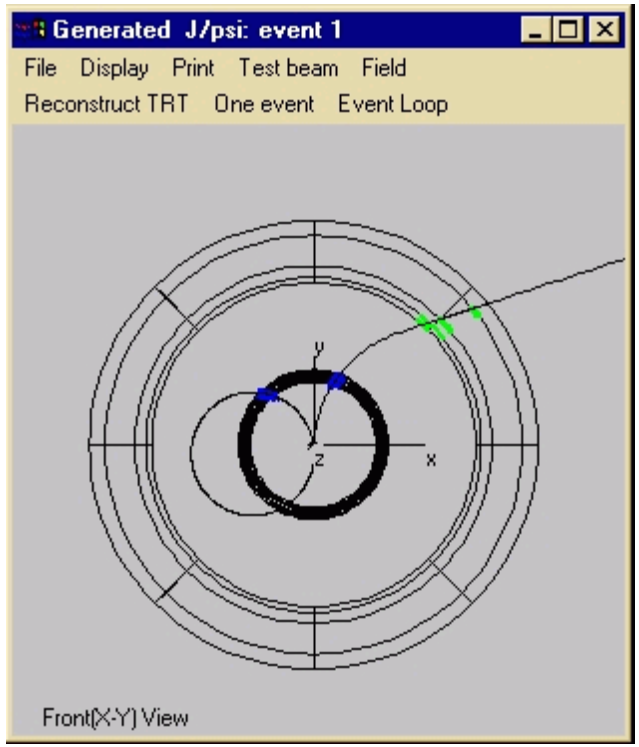
```
EcalDetector::EcalDetector() :
   m_data()
{
}

void EcalDetector::score(MCParticle* track)
{
   float deltaE = track->energyLoss();
   if (deltaE > 0) {
        m_data.add_hit(track->position(),track->direction()), deltaE);
   }
}
```

Example “*Ecal*”

EcalDetector.cxx - *continued*

```
void EcalDetector::clear()
{
   m_data.clear();
}

void EcalDetector::draw(ArvePlottableRep& v) const
{
   m_data.draw();
}
```

Result “ecal” + “trt” (J. Loken)



Conclusion

- Building a first application with Arve is easy.
- Of course, the reconstruction part you have to supply yourself.
  How you do it is up to you.
- For that Arve will help you with:
  - Simulation to test your program.
  - The displaying of your reconstructed objects.