Geometry Description:
Plans/Progress within AIDA for a common detector description

Linear Collider Software Meeting
Pere Mato/CERN, Markus Frank/CERN

2/2/2012

Outline

- Geometry Task Goals
- Requirements
- Design Elements
- Prototype
- Discussion

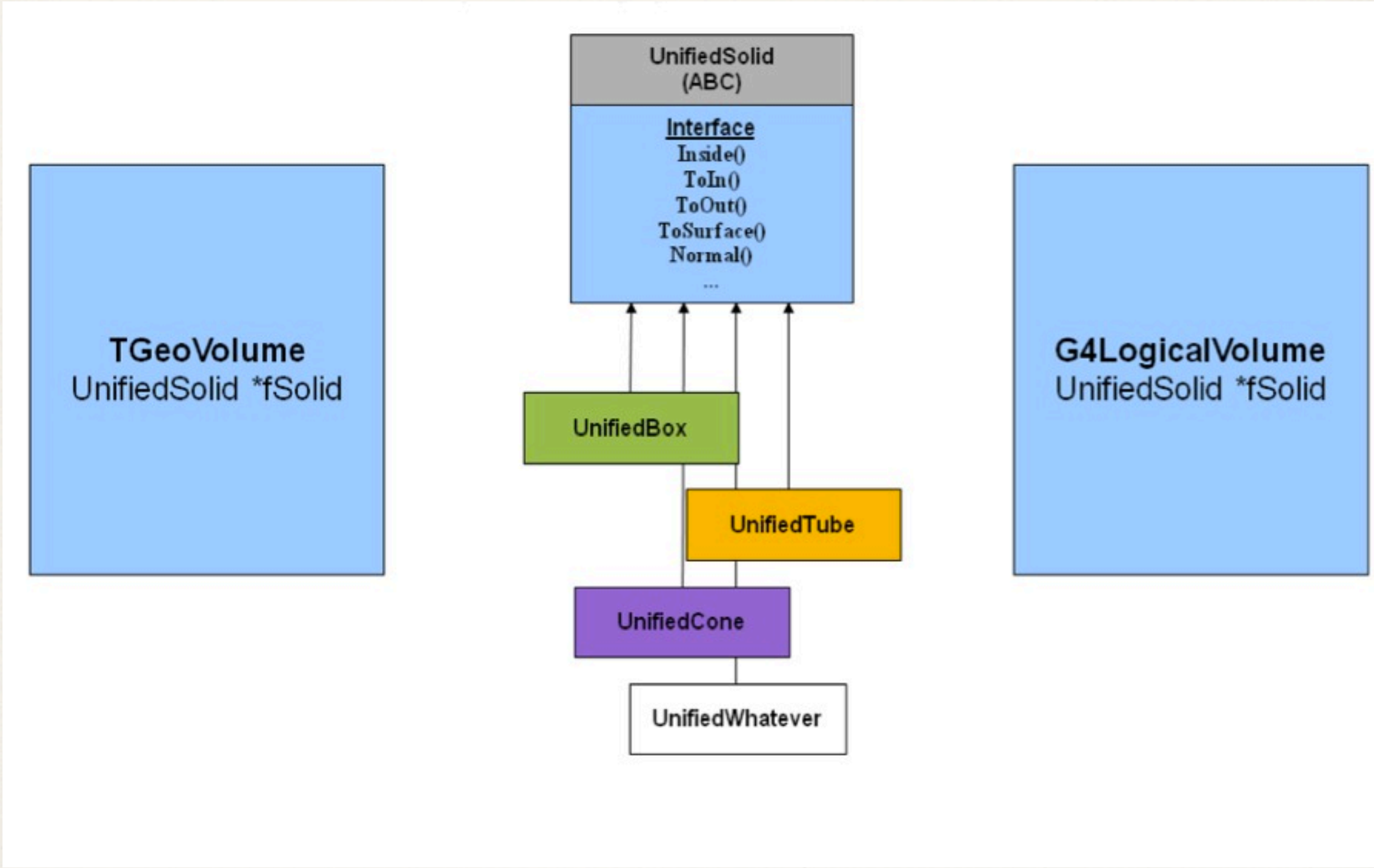
AIDA-WP2 Geometry Tasks

- USolids Library
  - Development of a common library of geometrical representations of the detector elements for solid 3D modeling
  - Common interface defined, started to implement primitives
- Geometry Toolkit
  - Set of software tools which can describe the geometry of the detector, the material it is made from and different ways of detecting particles
  - High/Low level descriptions, primitives library, interchange formats, API for reconstruction, simulation, alignment support, etc.
  - Started the development of a prototype (DD4Hep)

Motivations for a common library

- Optimize and guarantee better long-term maintenance
  - About 70-80% of code for the geometry modeler concerns solids
  - Most of the maintenance effort and debugging is on solids
- Target to create a unique library of high quality implementation (shared between ROOT and G4)
  - Starting from what exists today in Geant4 and ROOT
- Converge on a unique description and interface
  - Allowing to reach also a common persistency GDML schema for solids
- Optimize, extend and rationalize the testing suite

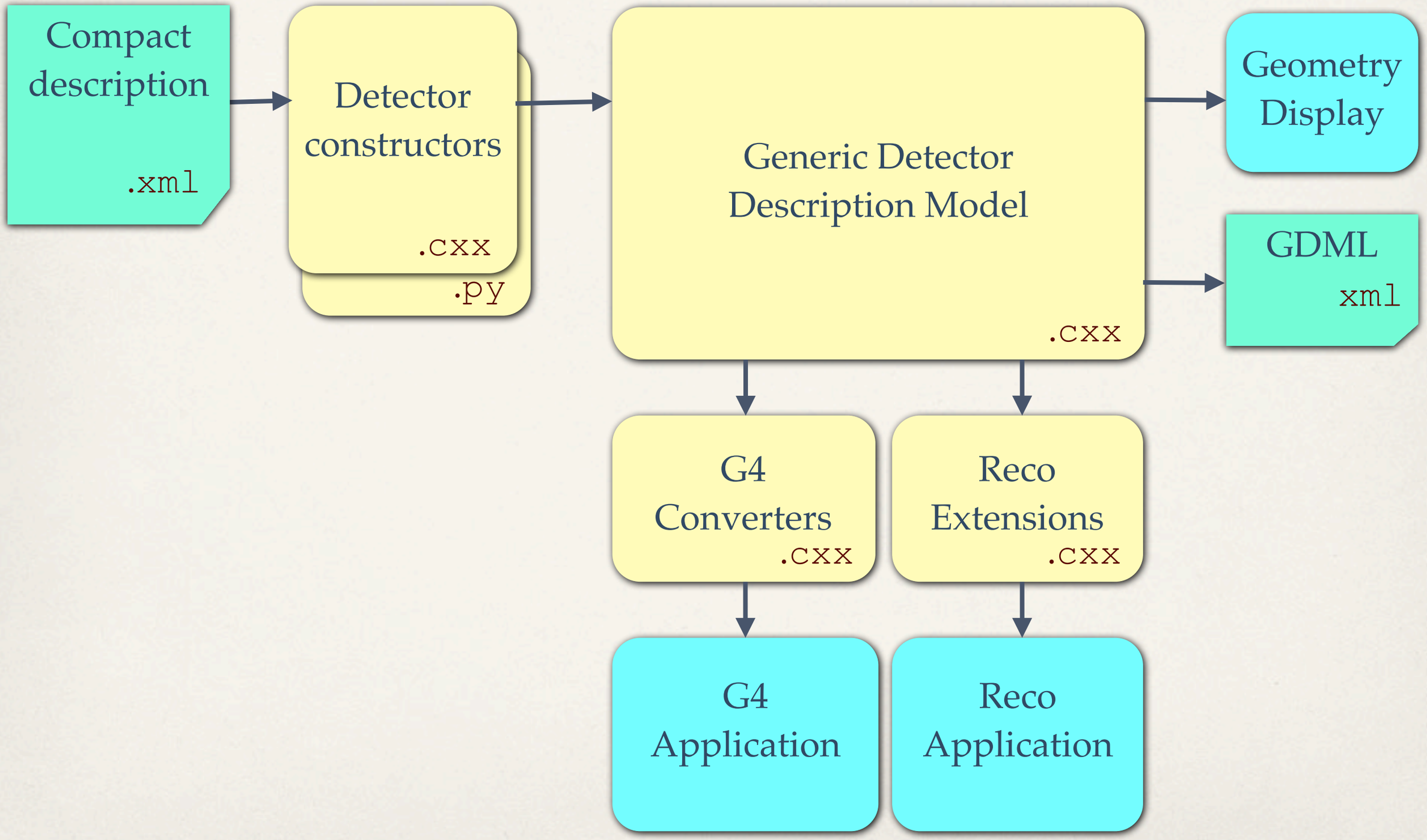
USolids after consolidation



Toolkit Main Requirements

- Full Detector Description
  - It include geometry, materials, visualization, readout, alignment, calibration, etc.
- Full Experiment life cycle
  - Detector concept development, detector optimization, construction, operation. Easy transition from one phase to the next
- Consistent Description
  - Single source of detector information for simulation, reconstruction, analysis, etc.
- Ease of Use
  - Few places to enter information. Minimal dependencies

Current Ideas: The Big Picture



Compact Description

- Reusing the idea of “compact detector description” from SiD software
- Human readable and compact geometry description in XML format
- Used as the main input to the detector description system
- Extendable with new generic detector types together with very specific ones

```
<detector name="VXD" type="ILDExVXD"
          vis="VXDVis" id="1">

  <layer id="1" vis="VXDLayerVis">
    <support thickness="0.01*mm"
             material="Carbon"
             vis="VXDSupportVis"/>
    <ladder  zhalf="65*mm"
             radius="16*mm" offset="-2*mm"
             thickness="0.01*mm"
             material="Silicon" number="10"/>
  </layer>

  <layer id="2"  vis="VXDLayerVis">
    <support thickness="0.01*mm"
             material="Carbon"
             vis="VXDSupportVis"/>
    <ladder  zhalf="65*mm" radius="18*mm"
             offset="-2*mm"
             thickness="0.01*mm"
             material="Silicon" number="10"/>
  </layer>

  ...

</detector>
```

Detector Constructors

- A set of code fragments that are able to convert the XML elements into detector description (DD) objects
  - Generic ones: Material, Element, VisAttributes, Limits, etc.
  - Specific Detectors
- Prototyped two possible implementations
  - C++ functions (XercesC)
  - Python functions (PyROOT)

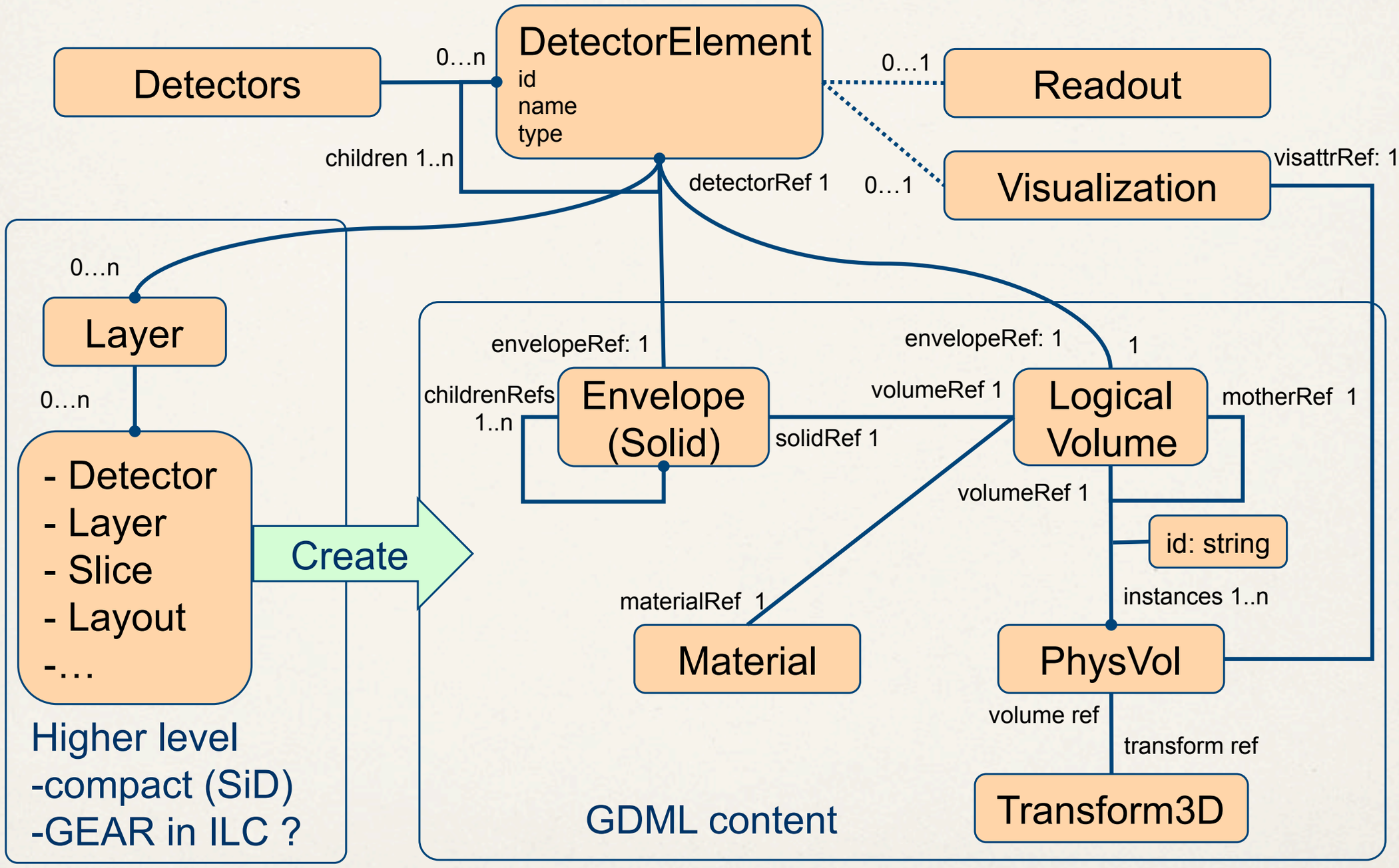
```
<element Z="29" formula="Cu" name="Cu" >
   <atom type="A" unit="g/mol" value="63.5456" />
</element>
```

```
def process_element(lcdd, elem):
  doc = lcdd.document()
  tab = doc.GetElementTable()
  element = tab.FindElement(elem.get('name'))
  if not element:
    atom = elem.find('atom')
    tab.AddElement(atom.get('name'),atom.get('formula'),
                   atom.getI('Z'), atom.getI('value'))
```

```
template <> Ref_t toRefObject<Atom,xml_h>(lcdd_t& lcdd, const
xml_h& e)  {
  xml_ref_t    elem(e);
  TGeoManager* mgr      = lcdd.document();
  XML::Tag_t   eltname  = elem.name();
  TGeoElementTable* tab = mgr->GetElementTable();
  TGeoElement*  element = tab->FindElement(eltname.c_str());
  if ( !element )  {
    xml_ref_t atom(elem.child(_X(atom)));
    tab->AddElement(elem.attr<string>(_A(name)).c_str(),
                    elem.attr<string>(_A(formula)).c_str(),
                    elem.attr<int>(_A(Z)),
                    atom.attr<int>(_A(value))
                    );
    element = tab->FindElement(eltname.c_str());
  }
  return Handle_t(element);
}
```

```
def detector_ILDExVXD(lcdd, det):
  doc    = lcdd.document()
  vdx    = Subdetector(doc, det.get('name'), det.get('type'), det.getI('id'))
  mother = lcdd.pickMotherVolume(vdx)

  for layer in det.findall('layer'):
    support = layer.find('support')
    ladder = layer.find('ladder')
    layername = name + '_layer%d' % layer.getI('id')
    nLadders = ladder.getI('number')
    dphi   = 2.*pi/nLadders
    zhalf  = ladder.getF('zhalf')
    offset = ladder.getF('offset')
    sens_radius = ladder.getF('radius')
    sens_thick  = ladder.getF('thickness')
    supp_thick  = support.getF('thickness')
    supp_radius = sens_radius + sens_thick/2. + supp_thick/2.
    width       = 2.*tan(dphi/2.)*(sens_radius-sens_thick/2.)
    sens_mat    = lcdd.material(ladder.get('material'))
    supp_mat    = lcdd.material(support.get('material'))
    ladderbox = Box(doc, layername+'_ladder_solid',
                    (sens_thick+supp_thick)/2.,width/2.,zhalf)
    laddervol = Volume(doc, layername+'_ladder_volume', ladderbox, sens_mat)
    sensbox   = Box(doc, layername+'_sens_solid', sens_thick/2.,width/2.,zhalf)
    sensvol   = Volume(doc, layername+'_sens_volume', sensbox, sens_mat)
    senspos   = Position(doc,layername+'_sens_position',
                         -(sens_thick+supp_thick)/2.+sens_thick/2.,0,0)
    suppbox   = Box(doc, layername+'_supp_solid', supp_thick/2.,width/2.,zhalf)
    suppvol   = Volume(doc,layername+'_supp_volume', suppbox,supp_mat)
    supppos   = Position(doc,layername+'_supp_position',
                         -(sens_thick+supp_thick)/2.+sens_thick/2.+supp_thick/2.,0,0)
    laddervol.addPhysVol(PhysVol(sensvol),senspos)
    laddervol.addPhysVol(PhysVol(suppvol),supppos)

    for j in range(nLadders):
      laddername = layername + '_ladder%d' % j
      radius = sens_radius + ((sens_thick+supp_thick)/2. - sens_thick/2.)
      rot = Rotation(doc,laddername+'_rotation',0,0,j*dphi)
      pos = Position(doc,laddername+'_position',
                     radius*cos(j*dphi) - offset*sin(j*dphi),
                     radius*sin(j*dphi) - offset*cos(j*dphi),0.)
      ladder_physvol = PhysVol(laddervol)
      mother.addPhysVol(ladder_physvol,pos,rot)
    vdx.setVisAttributes(lcdd, det.get('vis'), laddervol);

  return vdx
```

Detector Description Model



Detector Description Prototype

- Developed by Markus Frank
- C++ model separation of ‘data’ and ‘behavior’
  - Classes consist of a only single ‘reference’ to the data object
  - Practical advantages concerning compile/link dependencies
  - Same ‘data’ can be associated to different ‘behaviors’
- Implementation based on TGeom (ROOT)
  - TGeom classes directly accessible (no hiding)
  - Support for aligment

Reconstruction Extensions

- The idea is to 'extend' the DetectorElement class with specific reconstruction code
  - Be able to answer detector questions asked by the reconstruction algorithms. E.g.:
    - transform ECAL 'cell id' to local [global] coordinates
    - amount of material to next layer
- These extensions can be added as 'plug-ins'

```
struct GearTPC : public Geometry::Subdetector {
  typedef TPCData Object;
  GearTPC(const Geometry::RefHandle<TNamed>& e);

  GlobalPadIndex  getNearestPad (double c0, double c1) const;
  double getDriftVelocity () const;
  double getReadoutFrequency () const;
  double getInnerRadius() const;
  double getOuterRadius() const;
};
```

```
double GearTPC::innerRadius() const {
  Subdetector gas   = data<Object>()->gas;
  Tube        tube  = gas.volume().solid();
  return tube->GetRmin();
}
double GearTPC::outerRadius() const {
  Subdetector gas   = data<Object>()->gas;
  Tube        tube  = gas.volume().solid();
  return tube->GetRmax();
}
```

Generic Geant4 Converters

- The Geant4 detector geometry can be created from the DD model
  - Conversion of TGeom to G4Geometry
  - Similarly the way it is done with SLIC (without having to generate an intermediate GDML file since we convert C++ objects to C++ objects)
- This will be facilitated by the USolids library to obtain the exact same behavior

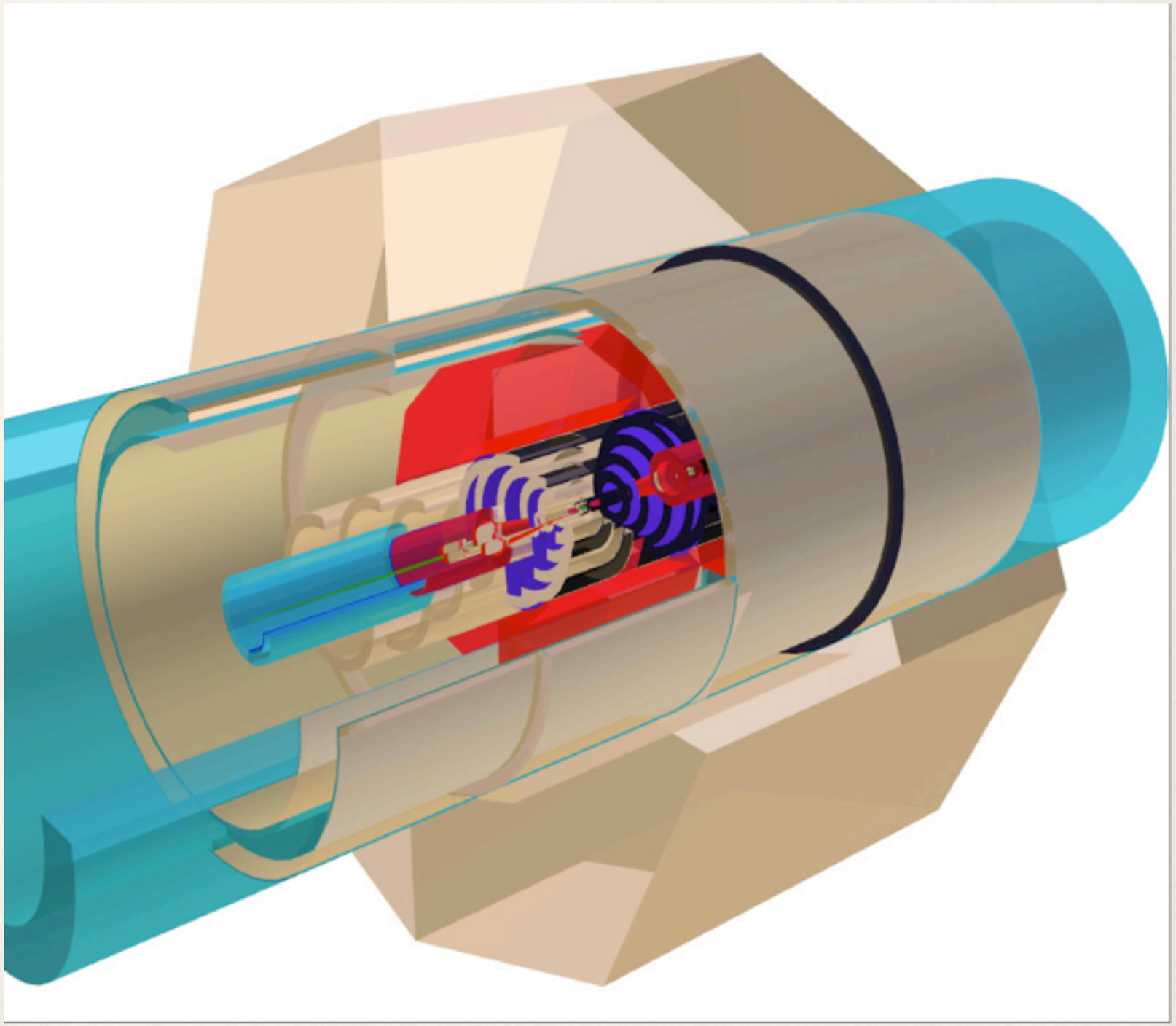
Application Code

- The entry point to the Generic DD model is an ‘singleton’ (e.g. LCDD)
  - Access the detector by its name (e.g. TPC)
  - Associate it a given ‘behavior’ (e.g GearTPC)
  - Start using it
  - Draw it!

```
#include "LCDD.h"
#include "GearTPC.h"

int main(int argc,char** argv)  {

  LCDD& lcdd = LCDD::getInstance();

  lcdd.fromCompact(argv[1]);

  GearTPC tpc = lcdd.detector("TPC");

  cout << "Gear: Inner:" << tpc.getInnerRadius() << endl;
  cout << "      Outer:" << tpc.outerRadius() << endl;

  return 0;
}
```

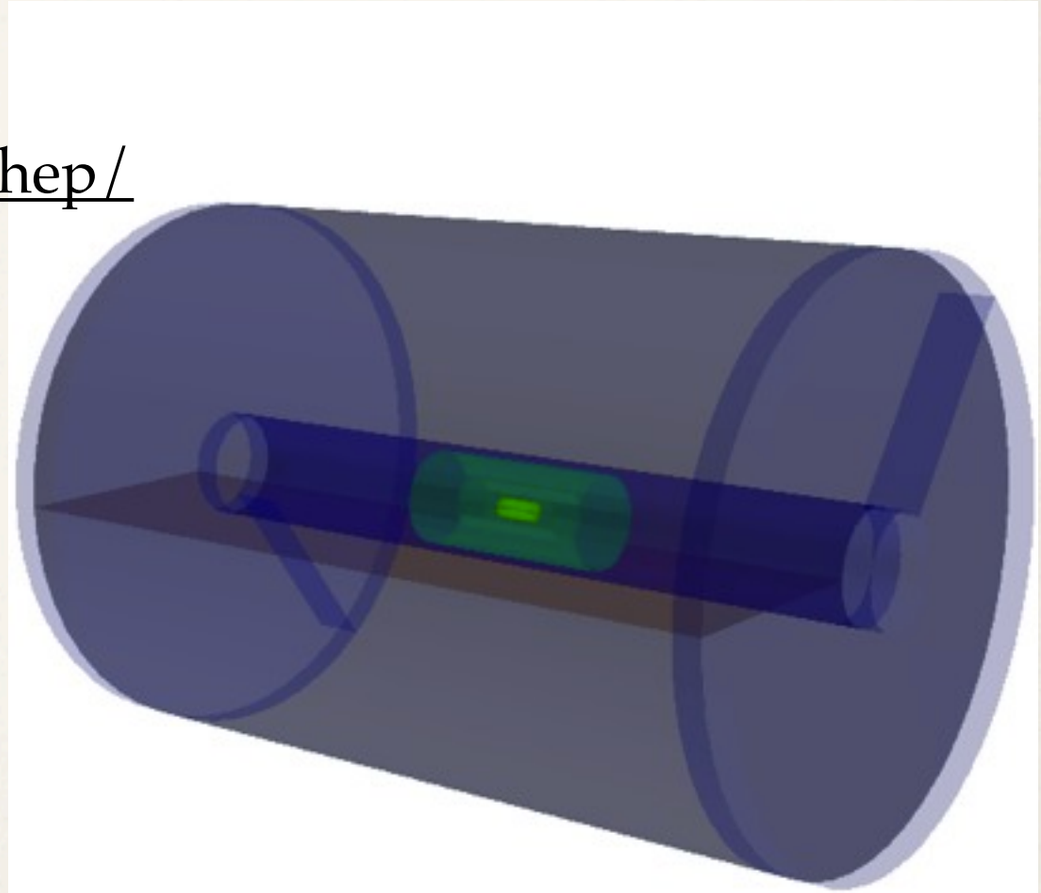
Detector Display

Display of the DD model produced from the SiD compact description (M. Frank)



DD4Hep Prototype

- Started to implement a simple prototype consisting of a simplified structure for the ILD central tracking detectors: VXD, SIT, TPC (Steven Aplin)
  - http://aidasoft.web.cern.ch/DD4Hep
  - http://svnsrv.desy.de/viewvc/aidasoft/DD4hep/



Discussion