Detector Geometry Description and Implementation

Teng LI

Shandong University

2024.5.19

Zhengzhou

Introduction

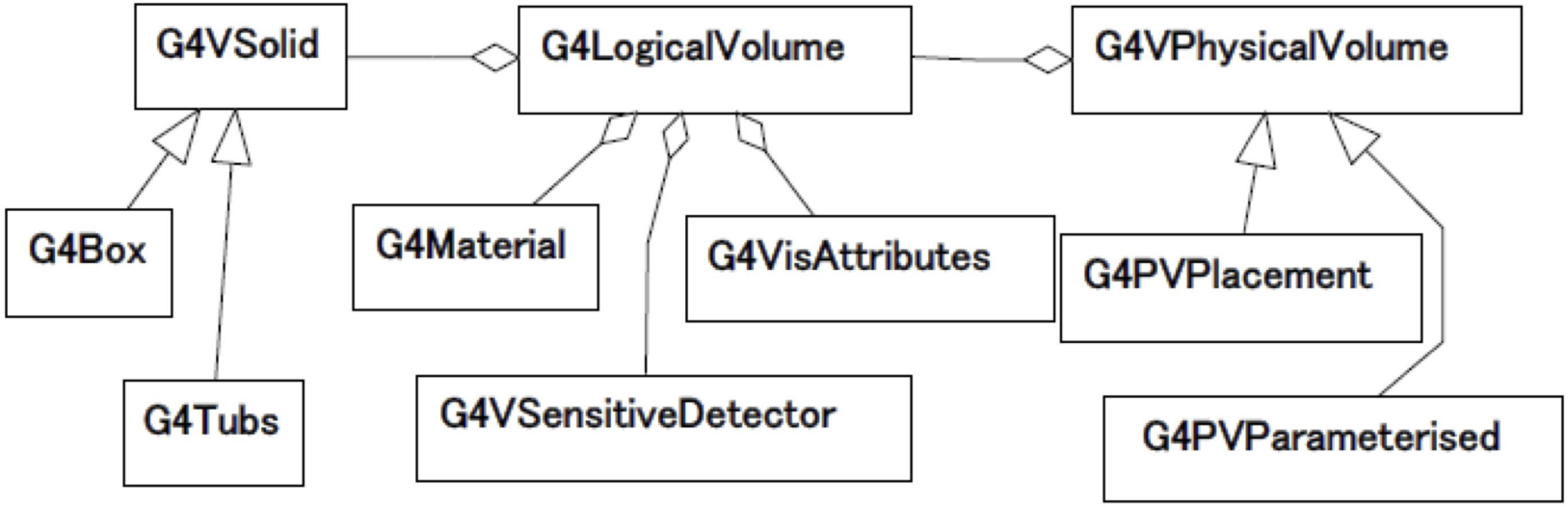
- Geometry management is one of the core components of HEP offline software
- It is safe to assume that in the foreseen-able future, Geant4 based simulation will continue to be the de-facto standard for detector simulation applications
- Geometry modeling concepts in Geant4 (e.g. hierarchical geometry, logical and placed volumes, sensitive detectors and hits) will be general guidelines for our future works
- Two main use-cases, simulation and reconstruction (and others) drives the development of geometry, which should be consistent
- This talk will review the geometry description implementation based on the above mentioned considerations
  - Geometry via code: Geant4 and ROOT
  - Geometry via data source: GDML and DD4hep

Geant4 Implementation

- The essential factors of building Geant4 detector descriptions
  - Construct necessary elements and materials
  - Define shapes/solids required to describe the geometry
  - Construct and place volumes of the detector geometry
  - Define sensitive detectors and associate with detector volumes (optional)
  - Associate magnetic field to detector regions (optional)
  - Define visualization attributes (optional)
  - Define regions (optional)
- In Geant4 all those is implemented via deriving the *G4VUserDetectorConstruction* class and implement the *Construct()* and *ConstructSDandField()* interface

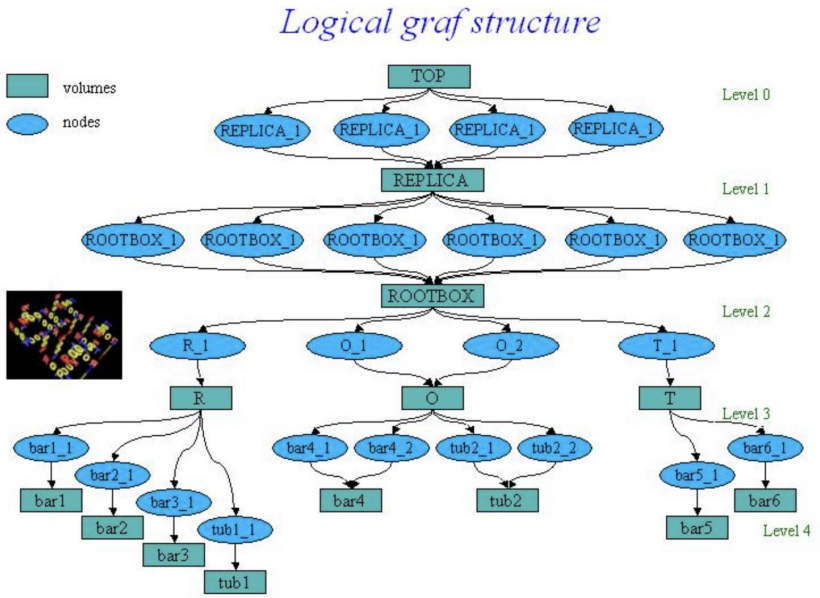
Geant4 Implementation

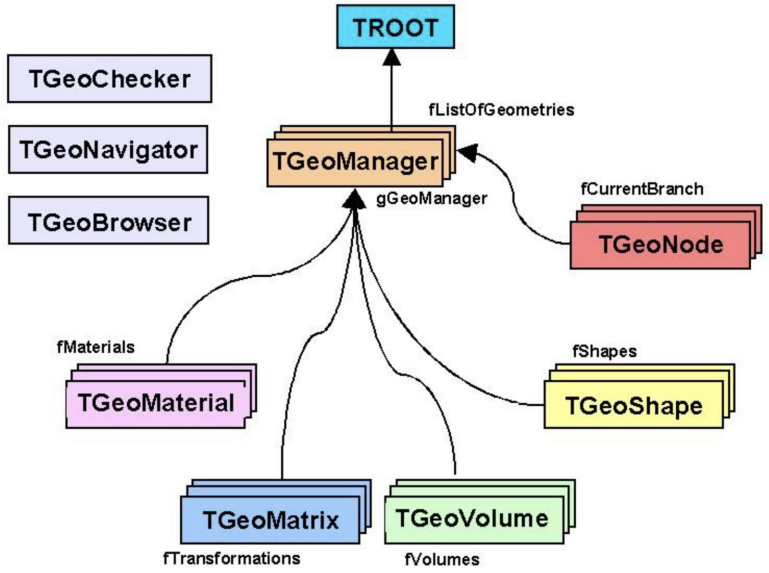
- Three conceptual layers of Geant4 detector description
  - G4VSolid -- shape, size (from libs, BREPS, Boolean, STEP)
  - G4LogicalVolume -- material, sensitivity, magnetic field, etc.
  - G4VPhysicalVolume -- position, rotation (placed, repeated, hierarchical)



ROOT Implementation

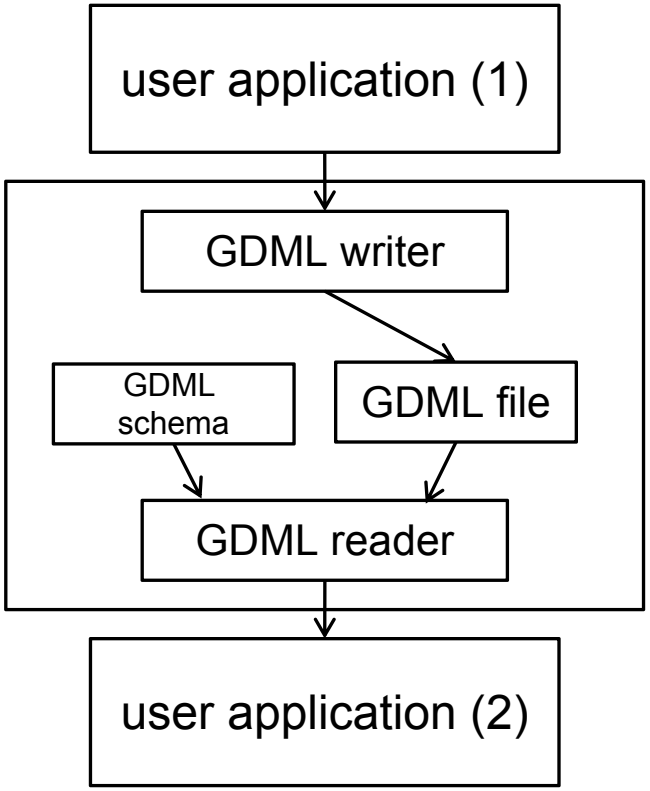
- The ROOT geometry package provides building, browsing, tracking and visualizing detector geometry
- The ROOT geometry does not contain physics constraints, and addresses tracking, reconstruction and visualization
- The geometrical modeler, as the core, implements navigation and tracking functionalities
- ROOT geometry answers questions like ‘where am I?’, ‘How far from the next boundary ?’, ‘Which is the next crossing along a helix’





Geometry Description Markup Language

- The Geometry Description Markup Language (GDML) is an application-independent geometry description format based on XML
- GDML can be used as the primary geometry implementation language, or as an exchange format



- GDML is defined via XML schema (XSD)
  - XSD (XML Document Type Definition)
  - Defines document structure and the list of legal elements
  - Latest: http://cern.ch/gdml (GDML_3_1_7)
  - Need XercesC to parse GDML
- Two toolkit bindings for GDML
  - Geant4
  - ROOT
  - Both support exporting and importing

Geometry Description Markup Language

XML Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
```

GDML Namespace XML Schema Instance Namespace

GDML Schema location

```
<gdml
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="http://service-spi.web.cern.ch/service-spi/app/releases/GDML/Schema/gdml.xsd">
```

Numerical definitions:

Density quantity definition

Isotopes and their use to create an element

Simple elements type definitions

NOTE: elements can't be associated with a volume, only materials can!

Simple material a la element way

Simple material created from two elements by number of atoms

```
<materials>
 <define>
  <quantity type="density" name="ro" unit="g/cm3" value="1234.00"/>
 </define>
 <isotope name="U235" Z="92"   N="235"> <atom type="A" value="235.01"/>
 </isotope>
 <isotope name="U238" Z="92.0" N="238"> <atom type="A" value="238.03"/>
 </isotope>
 <element name="enriched_Uranium">
   <fraction ref="U235" n="0.9"/> <fraction ref="U238" n="0.1"/>
 </element>
 <element name="Hydrogen" formula="H" Z="1."> <atom value="1.01"/>
 </element>
 <element name="Oxygen" formula="O" Z="8."> <atom value="16.0"/>
 </element>
 <element name="Nitrogen" formula="N" Z="7."> <atom value="14.01"/>
 </element>
 <element name="Lead" formula="Pb" Z="82."> <atom value="207.20"/>
 </element>
 <material name="Al" Z="13.0">
  <D value="2.70"/> <atom value="26.98"/>
 </material>
 <material name="Water" formula="H20">
   <D value="1.0"/>
   <composite n="2" ref="Hydrogen"/>
   <composite n="1" ref="Oxygen"/>
 </material>
 <material name="Air">
   <D value="1.290" unit="mg/cm3"/>
   <fraction n="0.7" ref="Nitrogen"/>
   <fraction n="0.3" ref="Oxygen"/>
 </material>
</materials>
```

Examples of some of the available CSG solids

One can place numeric definitions here as well as into materials and structure section.

NOTE: All numeric definitions are global!

Boxes

Cone

Parallelepiped

Sphere

General trapezoid

Trapezoid

Tubes

Make a union where tube "t100" is moved 250.0 mm along x-axis from center of the box "b500"

Subtract the non-rotated tube "t100" from the center of box "b100"

```
<solids>
 <define>
  <quantity type="length" name="sizeoft500" unit="mm" value="500.0"/>
  <position name="shiftbysizeoft500" x="500.0"/>
 </define>
 <box name="WorldBox" x="10000.0" y="10000.0" z="10000.0"/>
 <box name="b100" x="100.0" y="100.0" z="100.0"/>
 <box name="b500" x="500.0" y="500.0" z="500.0"/>
 <cone name="c1" z="111.0" rmax1="22.0" rmax2="33.0" deltaphi="TWOPI"/>
 <para name="p1" x="10.0"     y="10.0"     z="10.0"
                 alpha="30.0" theta="30.0" phi="30.0"/>
 <sphere name="s1" rmax="200.0" deltaphi="TWOPI" deltatheta="PI"/>
 <trap name="trap1" z="100.0"     theta="60.0"  phi="60.0"
                    y1="10.0"     x1="10.0"     x2="10.0"
                    alpha1="30.0" y2="10.0"     x3="10.0"
                    x4="10.0"     alpha2="30.0"/>
 <trd name="trd1" x1="10.0" x2="10.0" y1="10.0" y2="20.0" z="30.0"/>
 <tube name="t1000" z="1000.0" rmax="100.0" deltaphi="TWOPI"/>
 <tube name="t900" z="900.0" rmax="100.0" deltaphi="TWOPI"/>
 <tube name="t100" z="102.0" rmax="30.0" deltaphi="TWOPI"/>
 <union name="u2">
  <first ref="b500"/> <second ref="t100"/>
  <positionref ref="px250"/>
 </union>
 <subtraction name="sub2">
  <first ref="b100"/> <second ref="t100"/>
  <positionref ref="center"/> <rotationref ref="identity"/>
 </subtraction>
 <intersection name="intersec1">
  <first ref="b100"/> <second ref="b100"/>
  <positionref ref="px10"/> <rotationref ref="rotatebyx"/>
 </intersection>
</solids>
```

Geometry Description Markup Language

Simple geometry setup

Volume definitions

World volume definition

Place child volume "v2" at the center

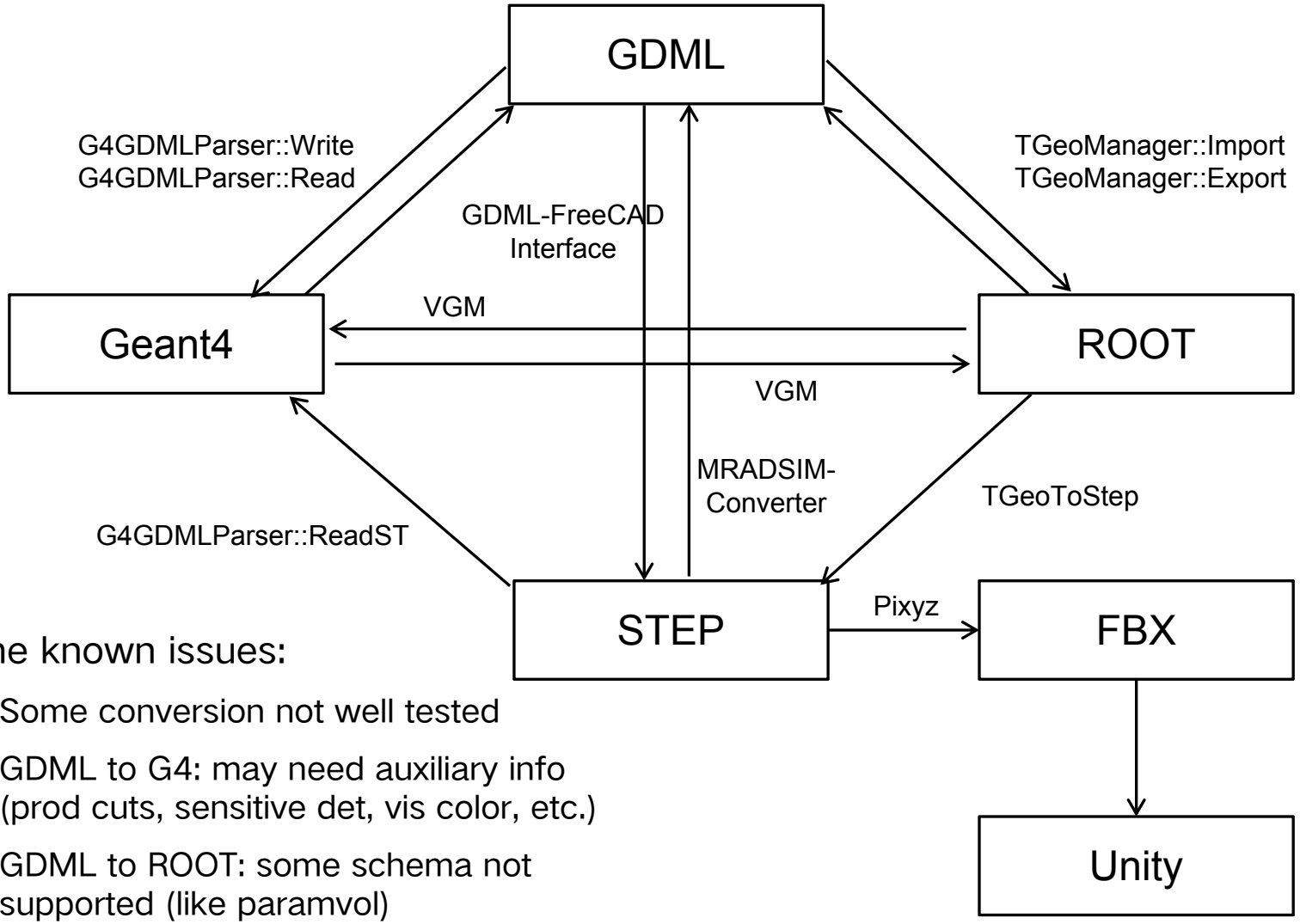
Place the child volume "v1" moved by 200.0 mm along x-axis from the center

Only one setup is defined here, however it is possible define multiple geometry setups choosing different volumes as world volumes from all the already defined volumes

```
<structure>
 <volume name="v1">
  <materialref ref="Al"/> <solidref ref="t1000"/>
 </volume>
 <volume name="v2">
  <materialref ref="Al"/> <solidref ref="sub2"/>
 </volume>
 <volume name="World">
  <materialref ref="Air"/> <solidref ref="WorldBox"/>
  <physvol>
   <volumeref ref="v2"/> <positionref ref="center"/>
                         <rotationref ref="identity"/>
  </physvol>
  <physvol>
   <volumeref ref="v1"/> <positionref ref="px200"/>
                         <rotationref ref="identity"/>
  </physvol>
 </volume>
</structure>
```

```
<setup name="Test1" version="1.0">
 <world ref="World"/>
</setup>
```

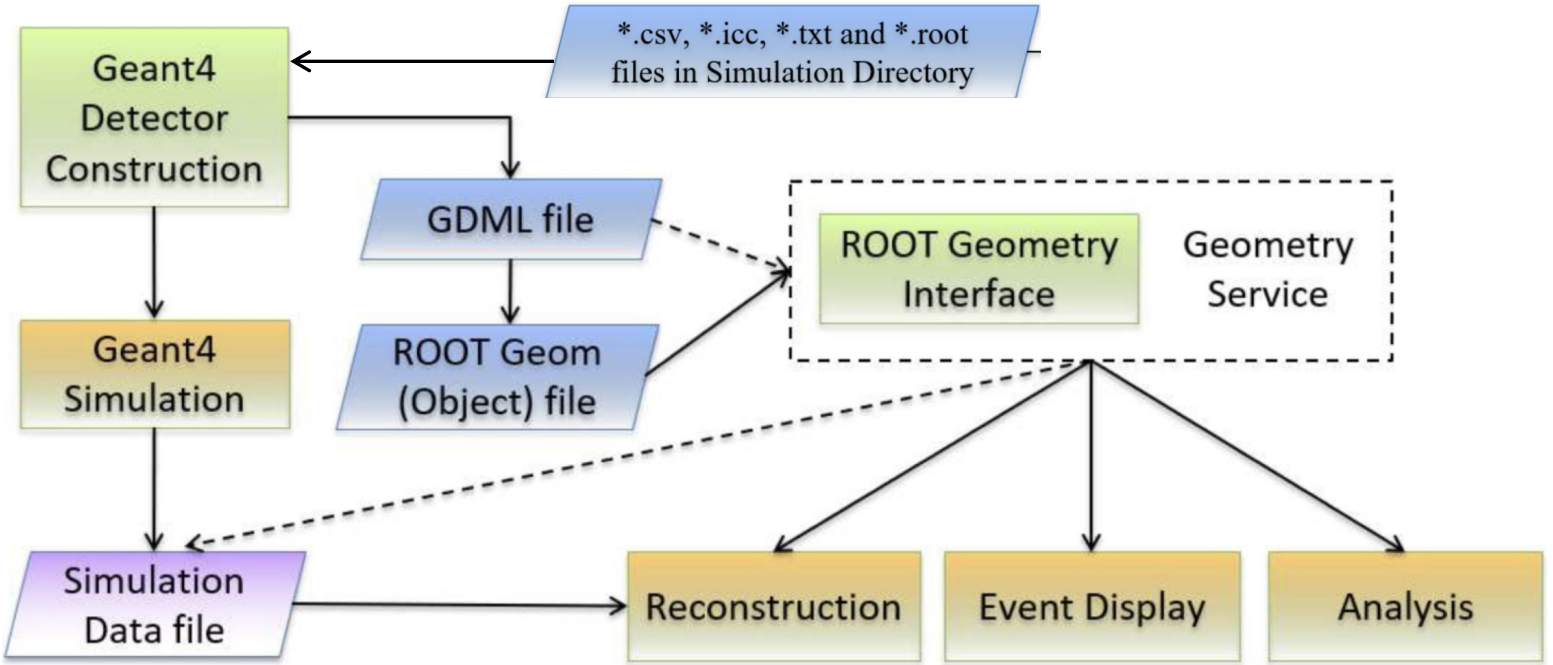
Geometry Format Conversion



- Some known issues:
  - Some conversion not well tested
  - GDML to G4: may need auxiliary info (prod cuts, sensitive det, vis color, etc.)
  - GDML to ROOT: some schema not supported (like paramvol)
  - Version issues

Geometry Management Based on GDML

- Geometry Management in JUNOSW based on GDML
  - Origin geometry defined in Geant4 (and parameters from data/db)
  - G4 geometry converted to GDML, then imported to ROOT
  - Reconstruction, event display and analysis applications are developed based on ROOT geometry

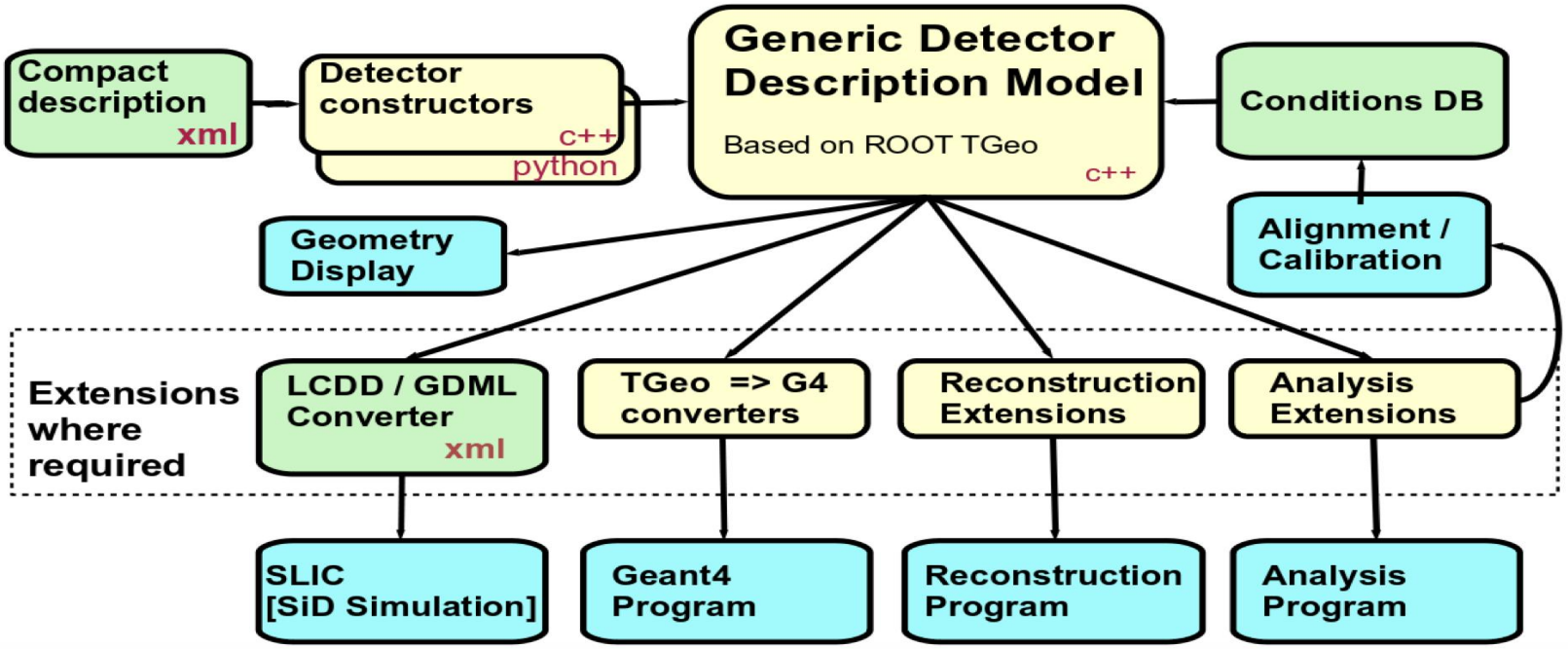


The Next Generation Geometry Management System

- The following features are considered as the requirements of the next generation of GMS
  - Geometry information should be consistent for all applications
  - Geometry must be able to include misalignment and more general condition data
  - Geometry description format should be clear, independent of specific apps
  - Geometry description should be modular
  - Geometry description should allow logical information (sensitive region, b-field), and logical information should also be modular
  - It should be possible to make geometry description persistent
  - Hits file should be self-describing
  - Flexible versioning support should be provided
  - CAD format should be supported

DD4hep: Common Detector Description Toolkit

- DD4hep: the common detector description toolkit for future HEP experiments
  - Full detector description: element, material, geometry, readout, visulization
  - Full experiment life cycle: conceptual, optimization, construction and operation
  - Single source of detector description: consistent for sim., rec., trigger, ana.
  - Ease of use



Detector Description with DD4hep

```
<lccdd>                          The root tag of the XML tree is lccdd (fixed)
    <info> ... </info>           Auxiliary detector model information
    <includes> ... </includes>   Section defining GDML files to be included
    <define> ... </define>       Dictionary of constant expressions and variables
    <materials> ... </materials> Additional material definitions
    <display> ... </display>     Definition of visualization attributes
    <detectors> ... </detectors> Section with sub-detector definitions
    <readouts> ... </readouts>   Section with readout structure definitions
    <limits> ... </limits>       Definition of limit sets for Geant4
    <fields> ... </fields>       Field definitions
</lccdd>
```

```
<includes>
  <file ref="elements.xml"/>
  <file ref="materials.xml"/>
```

```
<materials>
  ...
  <!-- (1) The description of an atomic element o.
  <element Z="30" formula="Zn" name="Zn" >
    <atom type="A" unit="g/mol" value="65.3955" /
  </element>
  <!-- (2) A composite material
  <material name="Kapton">
    <D value="1.43" unit="g/cm3" />
    <composite n="22" ref="C"/>
    <composite n="10" ref="H" />
    <composite n="2" ref="N" />
    <composite n="5" ref="O" />
  </material>
  <!-- (3) A material mixture
  <material name="PyrexGlass">
    <D type="density" value="2.23" unit="g/cm3"/>
    <fraction n="0.806" ref="SiliconOxide"/>
    <fraction n="0.130" ref="BoronOxide"/>
    <fraction n="0.040" ref="SodiumOxide"/>
    <fraction n="0.023" ref="AluminumOxide"/>
  </material>
  ...
</materials>
```

```
<lccdd
    <info name="Simple_CLIC"
      title="CLIC Detector like example detector model"
      author="F. Gaede"
      url="http://ilcsoft.desy.de"
      status="development"
      version="$Id: $">
       <comment>
          A very simple CLIC like detector
          so far only VXD and SiBarrelTracker (named SIT)
    </comment>
    </info>

    <includes>
        <gdmlFile  ref="elements.xml"/>
        <gdmlFile  ref="materials.xml"/>
    </includes>

    <define>
        <constant name="world_side"             value="10*m"/>
        <constant name="world_x"                value="world_side/2"/>
        <constant name="world_y"                value="world_side/2"/>
        <constant name="world_z"                value="world_side/2"/>

        <constant name="tracker_region_rmax" value="2.0*m"/>
        <constant name="tracker_region_zmax"   value="2.0*m"/>

    <constant name="SolenoidField_zMax" value="4.0*m"/>
    <constant name="SolenoidField_outer_radius" value="4.0*m"/>

    <constant name="BField_nominal" value="4.0*tesla"/>

    </define>

    <materials>
        <material name="Graphite">
                <D value="1.7" unit="g/cm3"/>
                <composite n="1" ref="C"/>
        </material>
    </materials>

    <limits>
      <limitset name="Tracker_limits">
        <limit name="step_length_max" particles="*" value="5.0" unit="mm" />
      </limitset>
    </limits>
```

Detector Description with DD4hep

```
    <display>
      <vis name="VXDLayerVis"    alpha="1.0" r="0.5" g=".5"  b=".5"   showDaughters="true"  visible="true"/>
      <vis name="VXDSupportVis" alpha="1.0" r="0.0" g="1.0" b="0.0"  showDaughters="true"  visible="true"/>
      <vis name="SITSupportVis" alpha="1.0" r="0.0" g="0.3" b="0.7"  showDaughters="true"  visible="true"/>
      <vis name="SITLayerVis"    alpha="1.0" r="0.0" g="0.7" b="0.3"  showDaughters="true"  visible="true"/>
    </display>
    <detectors>
      <comment>Trackers</comment>

      <detector name="AirTube" type="AirTube" vis="VXDVis" id="42" insideTrackingVolume="true">
        <dimensions rmin="10.*mm" rmax="11.*mm" zhalf="6.250000000e+01*mm"/>
      </detector>

      <detector name="VXD" type="ZPlanarTracker" vis="VXDVis" id="1" limits="Tracker_limits" readout="VXDCollect

        <layer nLadders="18" phi0="-1.570796327e+00" id="0">
          <ladder distance="3.100000000e+01*mm" thickness="6.700000000e-02*mm" width="1.150000000e+01*mm" length
          <sensitive distance="3.095000000e+01*mm" thickness="5.000000000e-02*mm" width="1.100000000e+01*mm" len
        </layer>
      </detector>
      <detector name="SIT" type="ZPlanarTracker" vis="SITVis" id="2" limits="Tracker_limits" readout="SITCollect
        <layer nLadders="14" phi0="0.0" id="0">
          <ladder distance="228.975*mm" thickness="1.0*mm" width="104.524099097*mm" length="430.0*mm" offset="0.
          <sensitive distance="229.975*mm" thickness="0.05*mm" width="104.980586046*mm" length="430.0*mm" offset
        </layer>
      </detector>
    </detectors>
    <readouts>
        <readout name="VXDCollection">
            <id>system:5,side:2,layer:9,module:8,sensor:8</id>
        </readout>
        <readout name="SITCollection">
            <id>system:5,side:2,layer:9,module:8,sensor:8</id>
        </readout>
    </readouts>
    <plugins>
        <plugin name="InstallSurfaceManager"/>
    </plugins>
  <fields>
    <field type="solenoid" name="GlobalSolenoid" inner_field="BField_nominal"
      outer_field="-1.5*tesla" zmax="SolenoidField_zMax"
      inner_radius="SolenoidField_outer_radius"
      outer_radius="world_side" />
  </fields>
</lccdd>
```

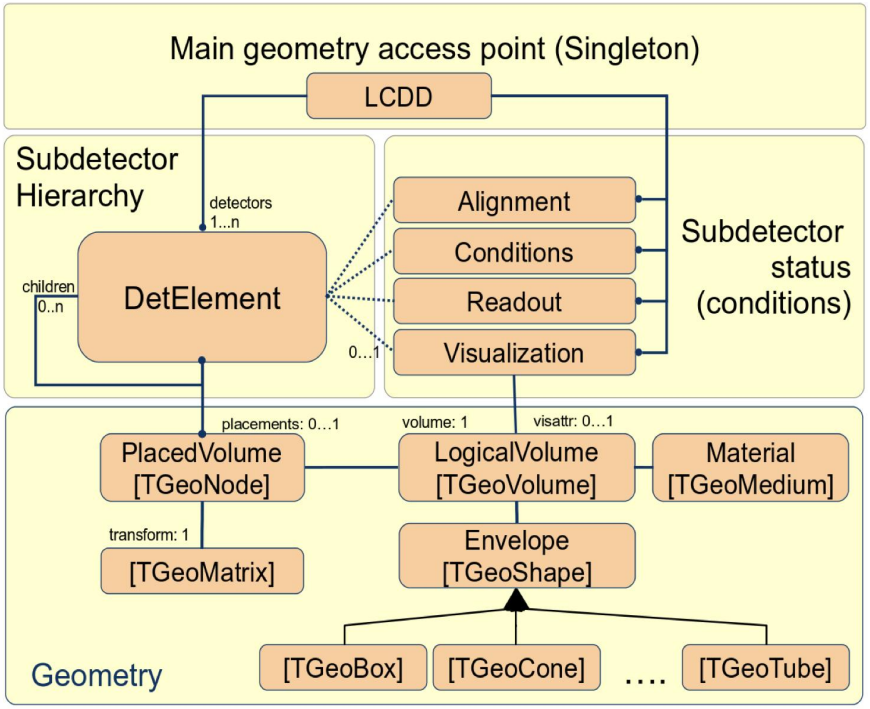
Detector Description with DD4hep

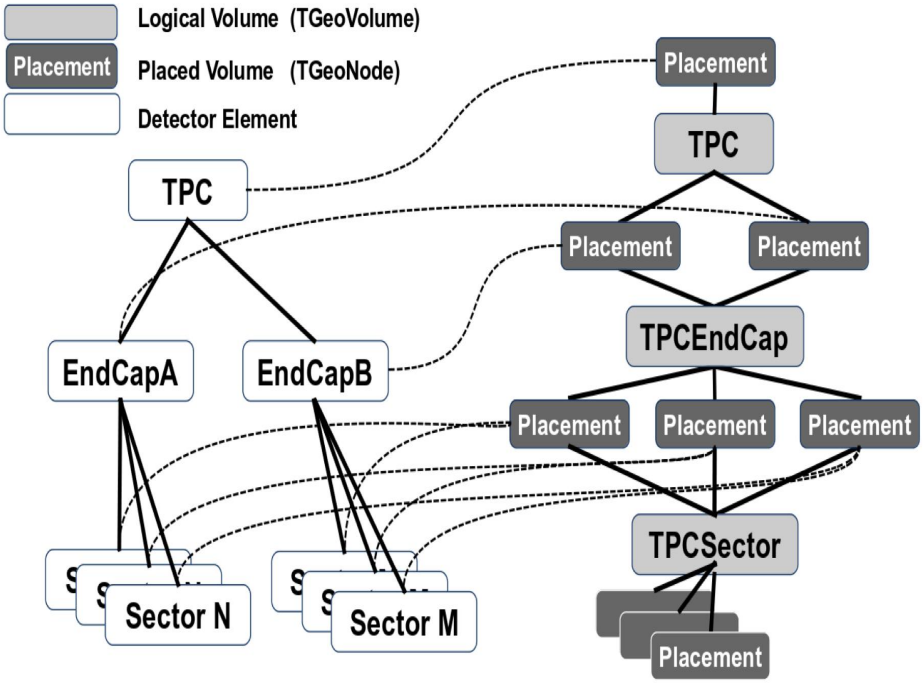
```
static Ref_t create_detector(Detector& description, xml_h e, SensitiveDetector sens)  {
  static double tolerance = 0e0;
  Layering      layering (e);
  xml_det_t     x_det     = e;
  Material      air       = description.air();
   string        det_name  = x_det.nameStr();
  xml_comp_t    x_staves  = x_det.staves();
  xml_comp_t    x_dim     = x_det.dimensions();
  DetElement    sdet      (det_name,det_id);
  Volume        motherVol = description.pickMotherVolume(sdet);
  PolyhedraRegular hedra  (nsides,inner_r,inner_r+totThick+tolerance*2e0,x_dim.z());
  Volume        envelope  (det_name,hedra,air);
  PlacedVolume  env_phv   = motherVol.placeVolume(envelope,RotationZYX(0,0,M_PI/nsides));

  sens.setType("calorimeter");
  { // =====  buildBarrelStave(description, sens, module_volume) =====
    // Parameters for computing the layer X dimension:
    double stave_z  = trd_y1;
    double tan_hphi = std::tan(hphi);
    double l_dim_x  = trd_x1; // Starting X dimension for the layer.
    double l_pos_z  = -(layering.totalThickness() / 2);

    // Loop over the sets of layer elements in the detector.
    int l_num = 1;
    for(xml_coll_t li(x_det,_U(layer)); li; ++li)  {
      xml_comp_t x_layer = li;
      int repeat = x_layer.repeat();
      // Loop over number of repeats for this layer.
     }
   ---
  envelope.setAttributes(description,x_det.regionStr(),x_det.limitsStr(),x_det.visStr());
  return sdet;
}

DECLARE_DETELEMENT(DD4hep_EcalBarrel,create_detector)
DECLARE_DEPRECATED_DETELEMENT(EcalBarrel,create_detector)
```

- Interprets the data from compact xml description
- Developers can custum detector constructors
- C++ and Python supported
- Compact description and Constructor together are sufficient to build detector model

DD4hep: Common Detector Description Toolkit

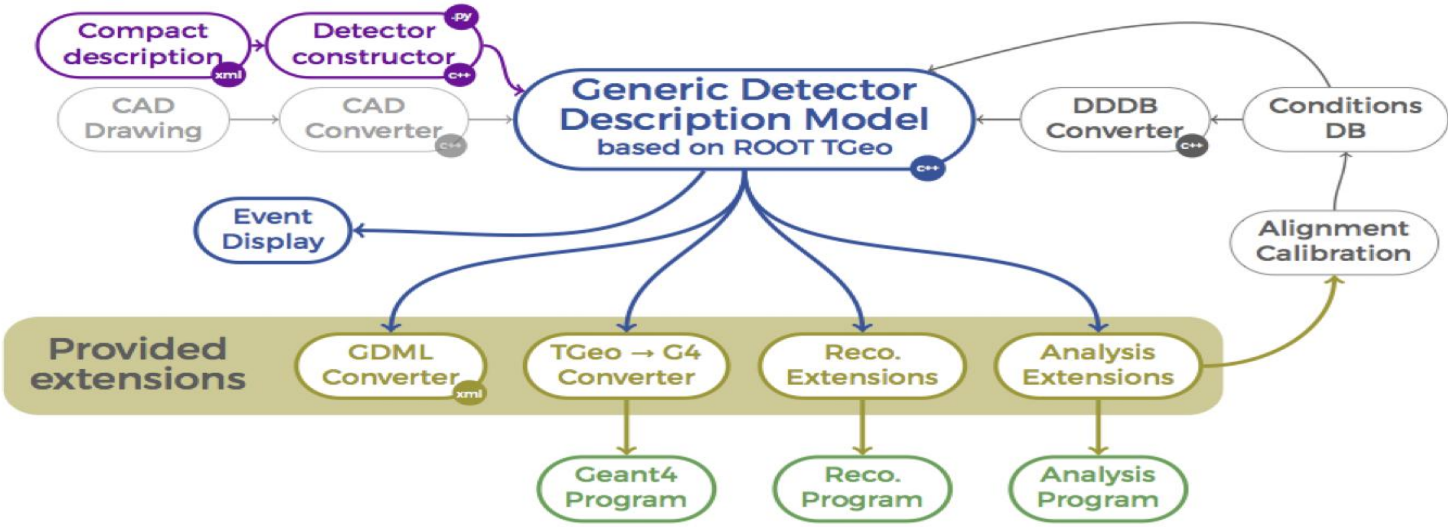
- DD4hep uses TGeo as the concrete implementation of geometry
- Generic Detector Description Model, as the heart of DD4hep, is implemented via DetElement tree
  - Give access to data associated to the detector (geometrical dimensions, alignment, calibration data etc.)





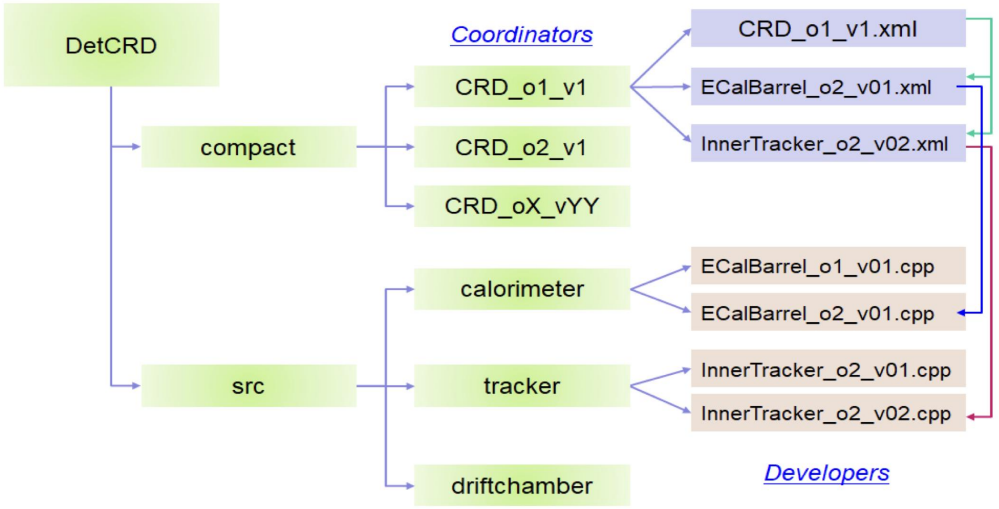
DD4hep plugins

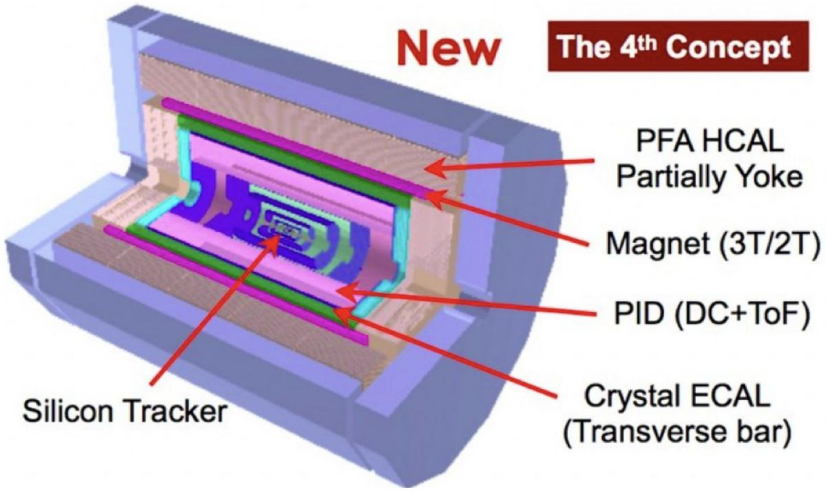
- A set of DD4hep plugins are (or being) developed for various use cases
  - DDG4: Full simulation with Geant4
  - DDRec: high-level interface for reconstruction
  - DDAlign: interface for (mis-)alignment of detector components
  - DDCond: interface to conditions data base
  - DDEve: implements event display
  - DDCAD: import CAD models using assimp package



Detector Description Based on DD4hep

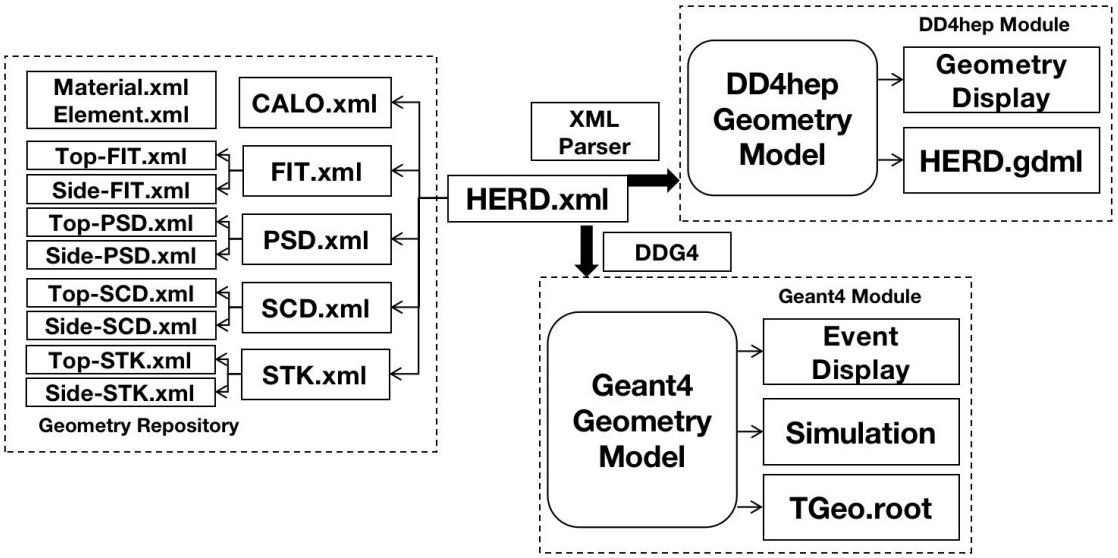
- Detector Description based on DD4hep
  - Elements, materials defined in shared files then composed together
  - Sub detector defined separately by the XML compact files and the C++ constructors, with independent versioning scheme
  - Full detector only configured by the compact files
  - The version scheme allows switching detector description during run time
  - All detector options are managed by database an git repository

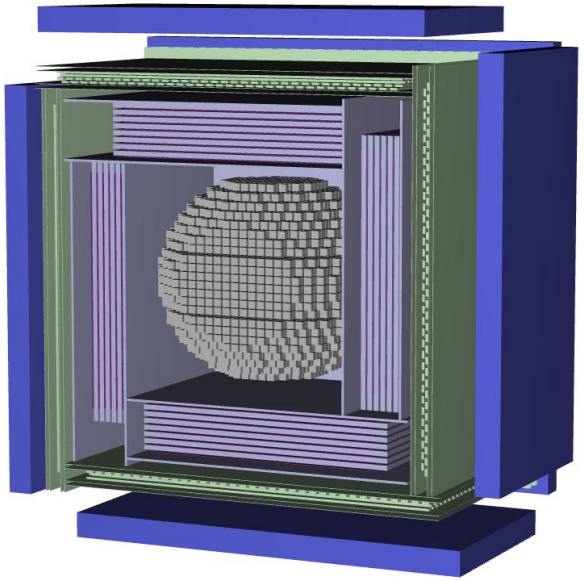




Detector Description Based on DD4hep

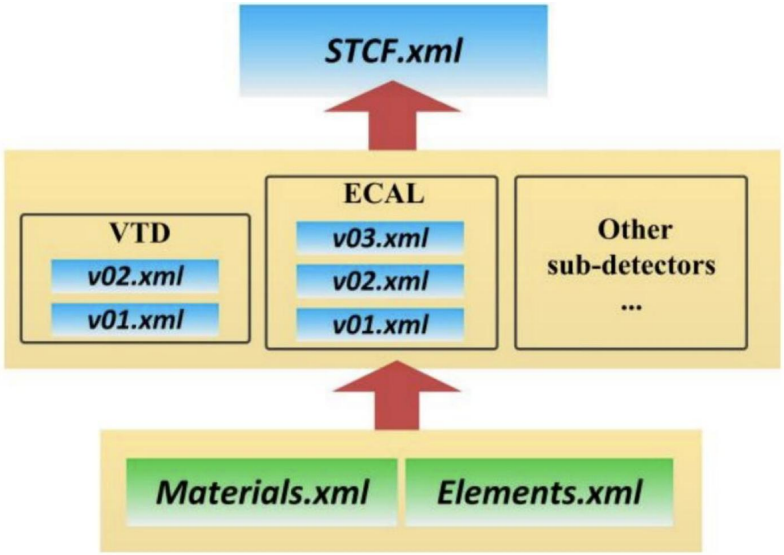
- Detector Description based on DD4hep
  - Elements, materials defined in shared files then composed together
  - Sub detector defined separately by the XML compact files and the C++ constructors, with independent versioning scheme
  - Full detector only configured by the compact files
  - The version scheme allows switching detector description during run time
  - All detector options are managed by database an git repository

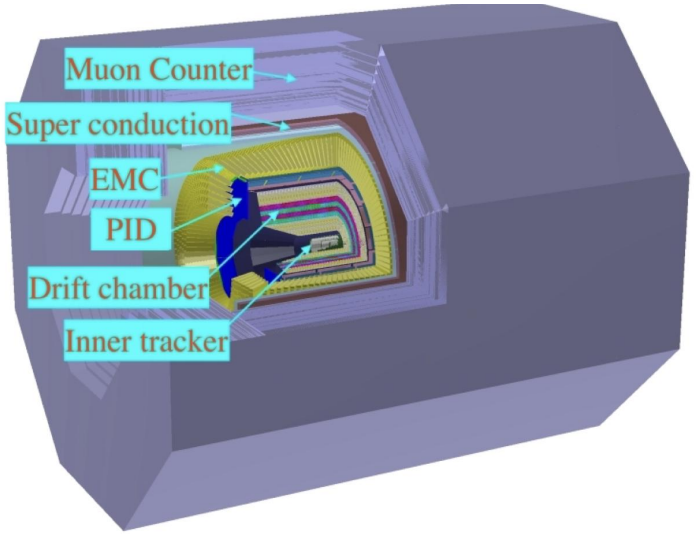




Detector Description Based on DD4hep

- Detector Description based on DD4hep
  - Elements, materials defined in shared files then composed together
  - Sub detector defined separately by the XML compact files and the C++ constructors, with independent versioning scheme
  - Full detector only configured by the compact files
  - The version scheme allows switching detector description during run time
  - All detector options are managed by database an git repository





Geometry Information Service

- To provide an easy-to-use interface for applications, a General Geometry Service is implemented to aggregate and provide various detector description information:
  - Conversion between geometry description formats (XML, CAD, Geant4, ROOT, GDML, ...)
  - Global-Local coordinates conversion
  - Coding scheme conversion
  - Get position, dimension of all detector volumes
  - Calculate track length in physics volumes
  - Provide interface to get information physical volume, placed volume, logical volume

Geometry Information Service

- To provide an easy-to-use interface for applications, a General **Geometry Service** is implemented to aggregate and provide various detector description information:

```
// Get geant4 geometry information
dd4hep::sim::Geant4GeometryInfo* getGeoInfo();
// Get geant4 physical Volume
G4VPhysicalVolume* getPhyVol();
// Get geant4 magnetic field
G4MagneticField* getMagField();
// Get dd4hep detector instance
dd4hep::Detector* getDetDesc();

// Get the global position of cell by its volumeid
dd4hep::Position getPosition(dd4hep::VolumeID &volId);
// Get the global position of cell by its cellcode and systemid
dd4hep::Position getPosition(SubDetector systemId, int cellcode);

// Get the dimensions of cell by its volumeid
std::vector<double> dimension(dd4hep::VolumeID &volId);
// Get the dimensions of cell by its systemid and cellcode
std::vector<double> dimension(SubDetector systemId, int cellcode);

// Get the physical node of cell by its volumeid
TGeoPhysicalNode *getPhyNode(dd4hep::VolumeID &volId);

// Transform from world coordinates to local ones at giving level
dd4hep::Position globalToLocal(const dd4hep::Position &global, int level=-1);
// Transform a point from local coordinates of a given level to global coordinates
dd4hep::Position localToGlobal(const dd4hep::Position &local, dd4hep::VolumeID &volId, int level=-1);
```

```
// Get volume direction in the global system
// Get the x-axis direction of local coordinate system
TVector3 getMainDir(dd4hep::VolumeID &volId);
TVector3 getMainDir(SubDetector systemId, int cellcode);
// Get the direction of fiber
TVector3 getAuxDir(dd4hep::VolumeID &volId);
TVector3 getAuxDir(SubDetector systemId, int cellcode);
// Get the normal direction of the plane
TVector3 getNormDir(dd4hep::VolumeID &volId);
TVector3 getNormDir(SubDetector systemId, int cellcode);

// The local Y axis is aligned vertically or not
bool isVertAligned(dd4hep::VolumeID &volId);
bool isVertAligned(SubDetector systemId, int cellcode);
// Local Y aligned with positive direction or not, of the nearest global axis
bool isPosAligned(dd4hep::VolumeID &volId);
bool isPosAligned(SubDetector systemId, int cellcode);

// Get the direction information
DirectionInfo getDirectionInfo(dd4hep::VolumeID &volId);
DirectionInfo getDirectionInfo(SubDetector systemId, int cellcode);
```

Summary

- We reviewed the detector geometry description and Implementation, based on code and data source
- DD4hep, provides a common solution for the next generation of HEP experiments
- CEPC, STCF and HERD (and more) have been explored the usage of DD4hep
- Development of geometry description system takes more than just using DD4hep

Thanks for listening