DD4hep

GENERIC DETECTOR DESCRIPTION
IN PRACTICE

FCC Software Workshop

Marko Petrič

CERN, 2 October 2019
Overview

- **Complete Detector Description**
  - Providing geometry, materials, visualization, readout, alignment, calibration
- **Supports full experiment life cycle**
  - Detector concept development, detector optimization, construction, operation
  - Facile transition from one stage to the next
- **Single source of information → consistent description**
  - Use in simulation, reconstruction, analysis
- **Ease of Use**
  - Few places for entering information
  - Minimal dependencies
- **AIDA-2020 and HSF member project**
Structure and packages

- **DD4hep**: Generic detector description in practice
- **DDCore**: STABLE
- **DDCond**: basics geometry handling STABLE
- **DDG4**: STABLE
- **DDAlign**: Conditions and Alignment support STABLE
- **DDRec**: STABLE
- **DDEve**: Simulation interface to Geant4 STABLE
- **DDG4**: Geometry visualization Event Display BASIC

Marko Petrič (CERN) marko.petric@cern.ch
What is Detector description?

- Description of a tree-like hierarchy of ‘detector elements’
  - Sub-detectors or parts of subdetectors

- Detector Element describes:
  - Geometry
    - points to placed logical volumes
  - Environmental conditions
  - Properties required to process event data
  - Extensions (optionally): experiment, sub-detector or activity specific data, measurement surfaces
DetElement Tree vs. Geometry Hierarchy

- Logical Volumes used to build geometrical hierarchy
  - geometry part delegated to the ROOT classes
- Relationship between DetElement and placements through full path from top of geometry
- DetElement tree is fully expanded
  - DetElement is placed only once in the DetElement tree
The Big Picture

Generic Detector Description Model based on ROOT TGeo

Compact description

Detector constructor

CAD Drawing

CAD Converter

Detector constructor

Folded

CAD

Converter

In

Converter

TGeo → G4

Converter

Reco.

Extensions

Analysis

Extensions

Provided

extensions

Event Display

GDML Converter

TGeo → G4 Converter

Reco.

Extensions

Analysis

Extensions

Geant4 Program

Reco. Program

Analysis Program

Provided extensions

DDDB Converter

Conditions DB

Alignment Calibration

Marko Petrič (CERN) marko.petric@cern.ch

DD4hep: Generic detector description in practice
The Generic Detector Palette

- Generic driver available → scalable and flexible
- Parameters are provided in compact XML files, e.g.

```xml
<detector id="15" name="HCal" type="GenericCalBarrel_o1_v01" readout="HCalCollection">
  <envelope vis="HCALVis">
    <shape type="PolyhedraRegular" numsides="HCal_sym" rmin="HCal_rmin" rmax="HCal_rmax" dz="HCal_dz" material="Air"/>
    <rotation x="0*deg" y="0*deg" z="90*deg-180*deg/HCal_symmetry"/>
  </envelope>
  <dimensions numsides="HCal_sym" rmin="HCal_rmin" z="HCal_dz*2"/>
  <layer repeat="(int) HCal_layers" vis="HCallayerVis">
    <slice material="Steel235" thickness="0.5*mm" vis="HCalAbsorberVis" radiator="yes"/>
    <slice material="Steel235" thickness="19*mm" vis="HCalAbsorberVis" radiator="yes"/>
    <slice material="Polystyrene" thickness="3*mm" sensitive="yes" limits="cal_limits"/>
    <slice material="Copper" thickness="0.1*mm" vis="HCalCopperVis"/>
    <slice material="PCB" thickness="0.7*mm" vis="HCalPCBVis"/>
    <slice material="Steel235" thickness="0.5*mm" vis="HCalAbsorberVis" radiator="yes"/>
    <slice material="Air" thickness="2.7*mm" vis="InvisibleNoDaughters"/>
  </layer>
</detector>
```

- You can scale, change layers, radii and compositions…
- Propagate visualization attributes to Display
- Inspect `.${DD4hep_ROOT}/DDDetectors/src` or compact
static Ref_t create_detector(Detector& theDetector,
   xml_h e, SensitiveDetector sens) {

xml_det_t x_det = e;
Layering layering(x_det);
xml_comp_t staves = x_det.staves();
xml_dim_t dim = x_det.dimensions();
DetElement sdet(det_name, x_det.id());
Volume motherVol = theDetector.pickMotherVolume(sdet);
PolyhedraRegular polyhedra(numSides, rmin, rmax, detZ);
Volume envelopeVol(det_name, polyhedra, air);

for (xml_coll_t c(x_det, _U(layer)); c; ++c) {
xml_comp_t x_layer = c;
int n_repeat = x_layer.repeat();
const Layer* lay = layering.layer(layer_num - 1);
for (int j = 0; j < n_repeat; j++) {
   string layer_name = _toString(layer_num, "layer%d");
   double layer_thickness = lay->thickness();
   DetElement layer(stave, layer_name, layer_num);
   ...}
}
DECLARE_DETELEMENT(GenericCalBarrel_o1_v01, create_detector)

- Users can easily write their own detector drivers, if needed
- Detector geometry extendable with additional info.
- C++ model of separation of ‘data’ and ‘behavior’
  - Classes consist of a single ‘reference’ to the data object
Advice for detector builders

- Make your driver as generic as possible
- Nest your volumes as much as possible
- Don’t place all the volumes in the worldVolume
  - slow navigation in simulation
- Help yourself with teveDisplay compact.xml
- Check for overlaps:
  - ddsim --runType=vis --compact=compact.xml
  - Run overlap check from G4 shell
  - G4 overlap check better than ROOT in our experience
In-memory translation of geometry from TGeo to Geant4
- Materials, Solids, Limit sets, Regions
- Logical volumes, placed volumes and physical volumes

External configuration:
- Plugin mechanism
- Property mechanism to configure plugin instances
- Supports configuration via XML, Python or ROOT-AClick

Use plugin mechanism to configure: Generation, Event Action, Tracking Action, Stepping Action, SensitiveDetector, PhysicsList...

Provides out of the box MC truth handling with record reduction
DDG4 – Configuration example

- DDG4 is highly modular
- Very easily configurable through python
  ```python
  #...
  gen = DDG4.GeneratorAction( kernel , "LCIOInputAction/LCIO1" )
  gen.Input = "LCIOFileReader" + inputFile
  #...
  ```
- Or configure actions, filters, sequences, cuts
  ```python
  #...
  part = DDG4.GeneratorAction(kernel, "Geant4ParticleHandler/ParticleHandler")
  kernel.generatorAction().adopt(part)
  part.SaveProcesses = ['Decay']
  part.MinimalKineticEnergy = 1*MeV
  part.KeepAllParticles = False
  #...
  user = DDG4.GeneratorAction(kernel,"Geant4TCUserParticleHandler/UserParticleHandler")
  user.TrackingVolume_Zmax = DDG4.tracker_region_zmax
  user.TrackingVolume_Rmax = DDG4.tracker_region_rmax
  #...
  ```
DDG4 – your friendly debugging tool

- No need to have the whole FCCSW on your computer/laptop
- DD4hep dependencies: ROOT, Geant4, Boost
- Easy access to G4 shell
- Interactive simulation control via `ddsim`
  - Is the conversion really working as planned
  - Do I see hits in the right detectors on the right spot
- Access to material scans
- Possibility to test a lot of machinery standalone before committing your code to FCCSW
Plugin Palettes

- Providing input handlers, sensitive detectors for most cases...
- Hard to provide Geant4 Sensitive Detectors for all cases
  - Couples detector ‘construction’ to reconstruction, MC truth and Hit production
  - Too dependent on technology and user needs
  - Several possibilities for tracker

- Providing palette of most ‘common’ sensitive components for trackers and calorimeters
- Physics lists, Physics/particle constructors etc.
  - Wrapped factory plugins directly taken from Geant4
  - Users extend physics list (e.g. QGSP)
- Several IO handlers (LCIO, ROOT, StdHep, HepEvt, HepMC)
The user base for DD4hep is growing
A bit more about ddsim

- Get steering file ddsim --dumpSteeringFile > mySteer.py
  - Steering file includes documentation for parameters and examples
  - The python file contains a DD4hepSimulation object at global scope
  - Configure simulation directly from commandline

```
from DD4hepSimulation import DD4hepSimulation
from SystemOfUnits import mm, GeV, MeV
SIM = DD4hepSimulation()
SIM.compactFile = "CLIC_o3_v06.xml"
SIM.runType = "batch"
SIM.numberOfEvents = 2
SIM.inputFile = "electrons.HEPEvt"
SIM.part.minimalKineticEnergy = 1*MeV
SIM.filter.filters ['edep3kev'] = dict(name="EnergyDepositMinimumCut/3keV", parameter=["Cut": 3.0*keV])
```

Tab Completion
For automatic completion of command line parameters: bash
bash-completion, python: argcomplete
eval"$(register-python-argcomplete ddsim)"

Much more convenient interactive use
```
$ ddsim --action.calo --filter.tracker --part.keepAllParticles
--action.mapActions --filter.filters --part.minimalKineticEnergy
--action.tracker --filter.mapDetFilter --part.printEndTracking
--gun.direction --filter.macroFile --part.printStartTracking
--part.saveProcesses --filter.filters --part.routine
--physics.decays --filter.filters --runType
--physicsList --filter.filters --skipNEvents
--physics.rangecut --filter.filters --vertexOffset
--action.calo --filter.filters --vertexSigma
```

ILDSWWS, Feb 2016 A. Sailer: Simulation Steering with DDSim

Marko Petrič (CERN) marko.petric@cern.ch
Simulation Example

- The CLIC Detector Model
- Simulate and view = ddsim + teveDisplay
**DDRec: High Level Information**

**High level view** onto the detectors through DDRec DataStructures extensions for DetElements

- Constructors fill DDRec DataStructures
- DataStructures allow to decouple detector implementation from reconstruction algorithms

DataStructures contain sufficient information to provide geometry information to particle flow clustering via PandoraPFA

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Detector Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConicalSupportData</td>
<td>Cones and Tubes</td>
</tr>
<tr>
<td>FixedPadSizeTPCData</td>
<td>Cylindrical TPC</td>
</tr>
<tr>
<td>LayeredCalorimeterData</td>
<td>Sandwich Calorimeters</td>
</tr>
<tr>
<td>ZPlanarData</td>
<td>Planar Silicon Trackers</td>
</tr>
<tr>
<td>ZDiskPetalsData</td>
<td>Forward Silicon Trackers</td>
</tr>
</tbody>
</table>
Geometry for Track Reconstruction

Information needed for track reco.
- measurement of directions of hits
- local-to-global transforms
- material properties

DD4hep surfaces provide this
- Surfaces can be auto. added

```xml
<plugin name="DD4hep_GenericSurfaceInstallerPlugin">
  <argument value="TrackDet"/>
  <argument value="dimension=2"/>
  <argument value="u_x=-1."/>
  <argument value="v_y=-1."/>
  <argument value="n_z=1."/>
</plugin>
```

- Plugin loops over all DetElements
- Configure surface type and direction in volume
- Automatically average materials
Provides DD4hepPlugin

- **convertDD4hepDetector**
  - Conversion from DD4hep geometry into ACTS geometry
  - Walks through detector geometry in order to access both geometrical and detector relevant information

- **DD4hepLayerBuilder**
  - Uses `DD4hep::DetElements` as input for sensitive surfaces

### Usage:

- Declare layers of sensitive detector and sensitive detector modules as `DD4hep::DetElements` in the constructor
- Use the ActsExtensions

```cpp
std::unique_ptr<Acts::TrackingGeometry> convertDD4hepDetector(DD4hep::Geometry::DetElement worldDetElement,
      Logging::Level     loggingLevel = Logging::Level::INFO,
      BinningType       bTypePhi   = equidistant,
      BinningType       bTypeR     = equidistant,
      BinningType       bTypeZ     = equidistant,
      double            layerEnvelope1 = 0.,
      double            layerEnvelope2 = 0.);
```
Very nice interplay
Sub-packages not in use by FCCSW

- **DDCond**:
  - Provide access to a consistent set of values according to a given time
  - Provide access to consistent set of accompanying data for processing event data
  - Supports multi-threading
  - Configurable and extensible
  - Defined mechanism to manage conditions

- **DDAlign**:
  - does not provide algorithms to determine alignment constants
  - supports hosting the results of such algorithms and applies the resulting imperfections
  - Implemented Global and Local (mis-)alignment

- Integrate solution that can handle running experiments and conceptual design phase projects
Implemented Features not released

- Several new features released since last v1.10
- Expected to be shipped sometime before CHEP
- CMake rewrite – you can comfortably build and develop against a LCG view
- Python 3 compatibility
- Ensure that dictionaries are relocatable
- Handling of optical properties (ROOT 6.20)
- Support for reflections (mirror right handed to left handed volumes)
Summary and Conclusion

- DD4hep provides a consistent single source of detector geometry for simulation, reconstruction, analysis
- Enables full simulation w/ Geant4 of particle collisions in detectors with minimal effort: simple, easy, flexible
- Generic reconstruction packages with *no* framework dependency: tracking, particle flow reconstruction
- If the detector is described via DD4hep, reconstruction comes almost for free
- The DD4hep toolkit is getting accepted by wider HEP community
  - Used by CLIC, LHCb, FCC, ILC, SCT detector communities
- Development continues in parallel with validation
- DD4hep can host user plugins: extensible
- Continued plugin suite development to cover all simulation req.
- Find us at [http://dd4hep.cern.ch](http://dd4hep.cern.ch)
Try on SLC6, CC7 or macOS

- git clone https://github.com/AIDASoft/DD4hep.git
- mkdir build
- cd build
- export LCG_RELEASE="LCG_96b"
- source ../.dd4hep-ci.d/init_x86_64.sh
- cmake -DDD4HEP_USE_GEANT4=ON
  -DBOost_NO_BOOST_CMAKE=ON -DBUILD_TESTING=ON
  -DCMAKE_CXX_STANDARD=17 ..
- make; make install
- . ../bin/thisdd4hep.sh