DD4hep

F. Gaede, DESY
Computing Round Table, Jefferson Lab

Nov 7, 2017
Outline

- Motivation
- Design and Implementation (DDCore)
- DDG4
- DDRec
- DDAlign
- DDCond
- Requirements for EIC community

- lots of material in this talk taken from the main developer M.Frank (CERN)
Motivation

- develop a generic detector (geometry) description for HEP
- support the full life cycle of the experiment
  - detector concept development
  - detector optimization
  - construction and operation
  - extendible for future use cases
- consistent description with one single data source
  - for simulation, reconstruction, analysis
- complete description:
  - geometry readout, alignment, calibration (conditions), . . .
- developed in AIDA and AIDA2020
Component based design

- **DD4hep** follows a **modular, component based design**
  - **DDG4**: full simulation with Geant4
  - **DDRec**: *high level* interface for reconstruction
  - **DDAlign**: interface for (mis-)alignment of detector components
  - **DDCond**: interface to conditions database

only use what you need ...
DDCore - core implementation of DD4hep
DD4hep uses **Root TGeo** for the underlying geometry description:
- access to ROOT Open GL viewer for geometry
- debugging (overlap checking) of geometry
- ROOT persistency for geometry

- additional hierarchy of **DetElements** provides access to
  - Alignment, Conditions, Readout (sensitive detectors), Visualization
  - arbitrary *user defined objects*
define *DetElements* for every *touchable object* that needs additional data
Implementation: geometry conversions

- default geometry description in DD4hep:
  - compact xml-files and C++ drivers
  - inspired by org.lcsim/SLIC geometry description
  - other input sources possible

- output formats/interfaces
  - GDML
  - Geant4 geometry
  - easily extendible
Compact Description - XML

- geometry defined in xml files
  - human readable
  - extendible
- interpreter supports units and formulas
- requires interpreting code to create detector geometry in memory: C++ drivers
- large palette of simple ‘standard HEP detectors’ provided in DDDetectors

```xml
<detector id="9" name="Coil"
  type="Tesla_coil00"
  vis="CoilVis">
  <coil
    inner_r="Hcal_R_max+
      Hcal_Coil_additional_gap"
    outer_r="Hcal_R_max+
      Hcal_Coil_additional_gap+
      Coil_thickness"
    zhalf="TPC_Ecal_Hcal_barrel_halfZ+
      Coil_extra_size"
    material="Aluminum">
    </coil>
  </detector>
```
C++ driver code

```cpp
static Ref_t create_element(LCDD& lcdd, const xml_h& e, SensitiveDetector& sens) {
  xml_det_t x_det = e;
  string    name = x_det.nameStr();
  DetElement sdet(name,x_det.id());
  Assembly   assembly(name);
  xml_comp_t x_coil = x_det.child(Unicode("coil"));

  Tube      coilTub(x_coil.inner_r(),x_coil.outer_r(),x_coil.zhalf());
  Volume     coilVol("coil",coilTub,lcdd.material(x_coil.materialStr()));
  coilVol.setVisAttributes(lcdd.visAttributes(x_det.visStr()));
  assembly.placeVolume(coilVol);

  PlacedVolume pv=lcdd.pickMotherVolume(sdet).placeVolume(assembly);
  sdet.setPlacement(pv);
  return sdet;
}

DECLARE_DETELEMENT(Tesla_coil00,create_element);
```

- very simple example: more complex detectors have `tree-structure`
Other input sources for DD4hep

- the detector geometry in DD4hep can be read from any other format
- requires translation/conversion code
- we have for example code that reads the original geometry description from CMS (XML) and LHCb (database)
  - requested by these experiments for evaluation of DD4hep

other formats can be added
- if really needed
- original compact format is quite powerful
Example: CMS Tracker in DD4hep

- converted from original CMS XML-format
Example: LHCb detector in DD4hep

- converted from *original* LHCb database
DDG4 - Gateway to full simulations with Geant4
walk through the geometry tree and convert *on the fly* from TGeo to Geant4
hook into the user entry points provided by Geant4:
- instantiate detector response from (pre-defined) plugins (sensitive detectors)
- instantiate physics list from (pre-defined) plugins (particles, processes)
- start the simulation
DDG4 Geant4 entry points - Sequences

- Flexible definition of the physics list
  - Define particles, processes, physics constructors or use/extend predefined physics lists

- Flexible data input
  - Programmable sequence. Input from particle gun, Icio, stdhep or HepMC (text) – easily extensible
  - Modules to smear and boost primary vertices
  - Modules to construct interaction overlays
  - Further extensions may independently added

- Provide user programmable sequences
  - Either as explicit object type using ABC
  - Or registering a member function as callback

- combine modules into sequences that are called by Geant4
DDG4 Geant4 Sequence Example

- combine simple and re-usable modules to configure the Geant4 simulation, e.g.
  - mix several input streams (signal and background)
  - smear the vertex position
  - combine various input and output formats
- can start sequences (Geant4 application) from:
  - Python program (ddsim from lcgeo)
  - C++ application configured with XML
  - ROOT C++ macros (!)
DDG4 features

- DDG4 provides all modules needed to run full simulations with Geant4 on any detector that is described in DD4hep
- pre-defined palette of standard sensitive detectors (trackers, calorimeters)
  - extendible for special user needs
- pre-defined, extendible palette of I/O handlers
  - Input: stdhep, LCIO, HepEvt(ASCII), HepMC(ASCII)
  - Output: ROOT, LCIO
- MC-Truth handling w/ and w/o record reduction
- physics lists: wrapped factory from Geant4
  - user defined physics lists
DDRec - Interface to the geometry for reconstruction (and analysis)
DetectorData classes

- describe high level view of generic detectors (motivated by ILC/CLIC detectors)
- extends, #layers, thicknesses, ...

<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>

- CellIDPositionConverter
  - convert cellID to position and vice versa

- MaterialManager
  - access material properties at any point or along any straight line
DDRec tracking geometry: surfaces

- tracking needs special interface to geometry
- measurement and dead material *surfaces* (planar, cylindrical, conical)
- surfaces attached to volumes in detailed geometry model

**surfaces:**

- $u,v$, origin and normal
- inner and outer thicknesses and material properties
- local to global and global to local coordinate transforms:
  - $(x, y, z) \leftrightarrow (u, v)$
DDRec surfaces and materials

- material properties are **automatically averaged**
  - from detailed model
  - along normal of the surface along given thicknesses

### Averaging Materials

\[
\langle A \rangle = \frac{\sum_i^N \rho_i t_i}{\sum_i^N \rho_i A_i} \quad t_i \text{ thickness}
\]

\[
\langle Z \rangle = \frac{\sum_i^N \rho_i \frac{t_i Z_i}{A_i}}{\sum_i^N \rho_i \frac{t_i}{A_i}} \quad \rho_i \text{ density}
\]

\[
\langle \rho \rangle = \frac{\sum_i^N \rho_i t_i}{\sum_i^N t_i}
\]

\[
\langle X_0 \rangle = \frac{\sum_i^N t_i}{\sum_i^N \frac{t_i}{X_{0i}}}
\]

\[
\langle \lambda \rangle = \frac{\sum_i^N t_i}{\sum_i^N \frac{t_i}{\lambda}}
\]

roughly equivalent for Bethe-Bloch - identical for multiple scattering.
example: comparison of surfaces vs. detailed model for ILD
DDAlign - Describe misaligned detector geometries
DDAlign Overview

- real detectors have non-ideal placement of sub-detector elements
- DDAlign component allows to describe such *mis-aligned* geometries
- **NB:** DDAlign does not provide the algorithms to find the mis-alignment but only the tools to correctly simulate and reconstruct *mis-aligned* geometries

example: misaligned Aleph TPC
DDAlign Global and Local Alignment

- global alignment corrections
  - physically modify geometry in memory
  - supported by TGeo
  - **not thread safe**
  - possibility to simulate mis-aligned geometries

- local alignment corrections
  - geometry is static (ideal or mis-aligned)
  - **thread-safe**
  - local alignments are conditions
  - provide *delta-transformations* for hit positions

- both approaches are supported
DDAlign $\Delta$-alignment

**DDAlign: Apply $\Delta$ - Parameters**

- Trickle-up the hierarchy and compute the matrices the most effective way
- Re-use intermediate results
DDCond - Interface to conditions data

- DDCond provides an interface to access conditions data
  - ‘slowly’ varying data
  - access through DetElement and Key
  - organized in IOVs (Interval of Validity)
- allow for derived conditions data
- thread-safe access

for more details see:

DD4hep users

- ILD and SiD (ILC)
- CLICdp
- Calice
- FCC-ee, FCC-hh, FCC-eh
- Interest from LHCb, CMS and CEPC

- DD4hep is an *HSF project*
Documentation and Websites

- **Main Web page**
  - http://aidasoft.web.cern.ch/DD4hep

- **Github code repository:**
  - https://github.com/AidaSoft/DD4hep

- **Manuals**
  - available from Github (.doc)
  - or main Web page

- **Doxygen documentation**
  - or build from source
DDCore, DDG4 and DDRec are mature and have reached **production quality**

- large scale Monte Carlo productions for ILD and CLICdp to start soon
- DDCore and DDG4 (partly) used by FCC
- well documented

**DDAlign and DDCond:**

- more recent developments
- plan to integrate into iLCsoft soon
- more detailed documentation pending

**rather stable code basis**

- PRs only merged after successful CI-builds on several platforms
- running ~150 tests in CI
Requirements for EIC community
Requirements 1.

The geometry information should be the same in both simulation and reconstruction.

fulfilled
- this was one of the main motivations to develop DD4hep
Fast simulation systems should, as much as possible, be able to use the common exchange format.

mostly fulfilled

- if using parameterized fast simulation in Geant4 one can use existing conversion converters to any other geometry format can be (easily) added
- FCC uses Geant4 regions for Delphes
Requirements 3.

The geometry system should allow to include misalignment and more general condition data.

- see DDAlign and DDCond
Requirements 4.

Geometry description format should be independent of a specific software technology.

mostly fulfilled

- can read/write various input/output formats for simulation
- existing conversion of geometry to Geant4, TGeo, GDML
- existing import for CMS and LHCb
- others could be added
Geometry description should be modular. It should be possible to specify different geometry components in isolation with ideally zero dependency between different modules (detectors).

**fulfilled**

- done in FCC implementation where any given set of XML files is read at program start
- partly done in LC implementations where include mechanism is used to read subdetector specific XML files that instantiate in mandatory envelope volumes
Requirements 6.

Geometry description should allow to specify logical information (sensitivity, B-Fields) in addition to the solids, material and placements. In particular sensitivity is recognized as a critical issue.

fulfilled

- this is part of the design through existing or additional user extensions added to the ‘DetElement’
Requirements 7.

It should be possible to make the geometry description persistent. Different equivalent output format should be supported (e.g. ROOT files, GDML files) and it should always be possible to translate one format into another in a simple manner.

<table>
<thead>
<tr>
<th>fulfilled</th>
</tr>
</thead>
<tbody>
<tr>
<td>• can write to and read from ROOT w/o data loss</td>
</tr>
<tr>
<td>• GDML is incomplete</td>
</tr>
<tr>
<td>• in general conversions can only convert existing features in a given format</td>
</tr>
</tbody>
</table>
Hits output files produced in a simulation job should be as much as possible self-describing, in particular it should be possible to locate hits in space without the need to run the simulation job. A self-describing format for the hits would be ideal, but in case this is not possible, the additional libraries to manipulate hits should not depend on the simulation stack used to produce the hits.

**fulfilled**

- existing ROOT and LCIO output formats store the hit positions
  - together with cellIDs, if assigned
  - if positions are dropped from output, one can use the CellIDPositionConverter without having to instantiate the Geant4 geometry
Requirements 9.

It should be possible to change sensitivity attributes without changing other static aspects of the geometry.

fulfilled

- can add any (compatible) segmentation/readout to an existing sub-detector
- e.g. change the granularity of a calorimeter layer
Requirements 10.

Geometry exchange format should allow clients to use a subset of the features clearly stating which are the optional ones. We should support existing interested frameworks (e.g. EicRoot, GEMC, Fun4ALL, SLIC,...), without discouraging other R&D activities (e.g. DD4hep). Since it is difficult to support all use-cases, the minimal set of mandatory elements to support should be clearly specified and what to do with non-supported one should be stated (e.g. ignore visualization attributes if not needed).

mostly fulfilled (?)

- see comments on exporting to other formats above
  - new formats could be added
- personal comment: I think it would be good to settle on one source for the geometry early on in the design phase of the experiment
Requirements 11.

Some support for import from CAD should be foreseen.

**not yet fulfilled**

- again this could be added if needed
- with the usual caveats on missing information
Geometry information should have support for versioning.

(mostly) fulfilled

- versioning of the geometry is a client task
- for example the LC community has a dedicated package with C++ drivers and compact XML files
  - versioning done with software releases (code)
  - additionally naming convention for XML files (ILD_l5_v02.xml)
  - see: https://github.com/iLCSoft/lcgeo