Generic Software Tools for HEP

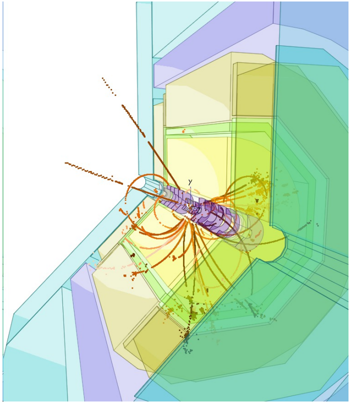
and beyond . . .

F.Gaede, DESY

MT Meeting, Jena, Mar 2019

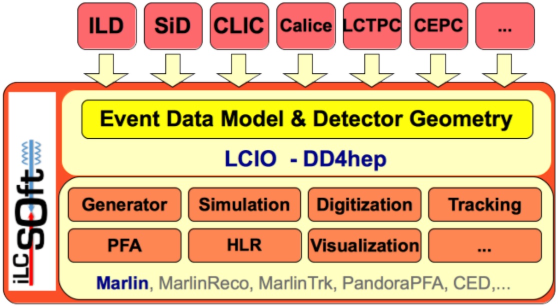
Outline

- Introduction
- DD4hep
- PODIO
- Summary and Outlook

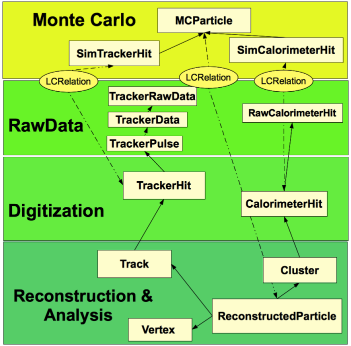


Introduction

- the linear collider community has a long tradition in developing and using common software tools
- from the start put focus on
  - lightweight solutions
  - well defined, clear APIs
  - ease of use
- main core tools in *iLCSoft*:
  - **LCIO**, **DD4hep** and **Marlin**
- see: https://github.com/iLCSoft

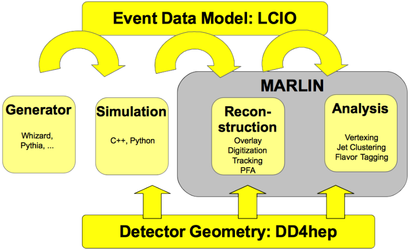


- common event data model (EDM) and persistency for linear collider community
  - joined DESY and SLAC (and LLR) project
  - first presented @ **CHEP 2003** (!)
- features:
  - non-ROOT Object I/O
  - schema evolution, compressed records, pointers . . .
  - EDM decoupled from I/O by interfaces
  - C++, Java (and Fortran)



common EDM proven to be crucial for collaborative SW development across detector concepts

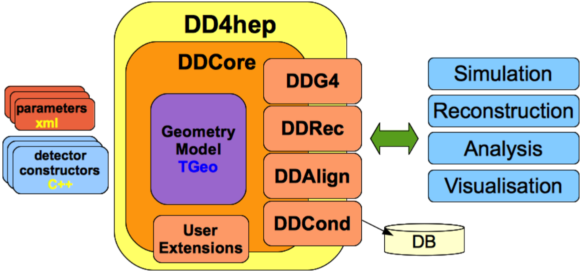
- next to the common Event Data Model a common **description of your detector geometry** is the other main requirement for developing common data analysis and reconstruction code
  - potentially slightly more difficult
  - took many years until **DD4hep** could be realized ...



DD4hep

DD4hep - Introduction and Overview

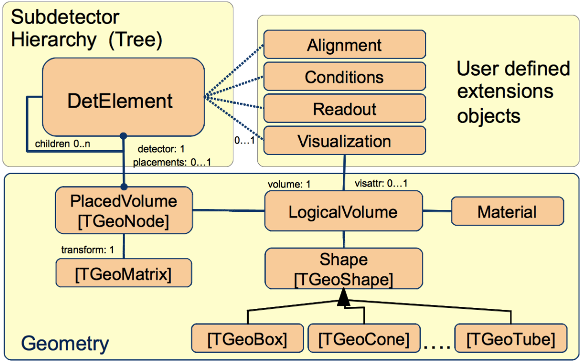
- generic (*experiment independent*) detector description toolkit for HEP
  - one **unique source** of geometry
  - supporting full experiment *life-cycle*
  - joined DESY and CERN project in **AIDA(2020)**, since 2011
- originally needed by Linear Collider but targeted at all of HEP
  - *Hep Software Foundation* incubator project



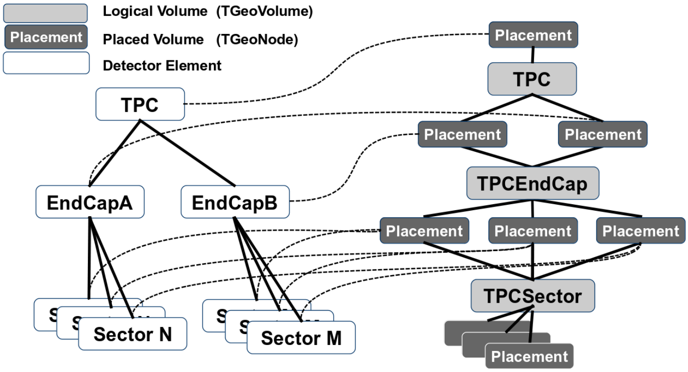
- component based architecture
  - interfaces for:
  - simulation, reconstruction, alignment, conditions data,...

DD4hep - Implementation

- 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*



DD4hep - Geometry hierarchy vs DetElement tree



- fully expanded tree for *DetElements*
- 'degenerated' (collapsed) tree for geometry hierarchy
- need to define *DetElements* for every *touchable object* that needs **user data**

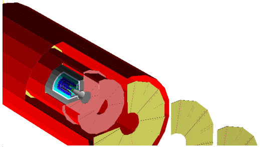
DD4hep - DDG4, gateway to Geant4

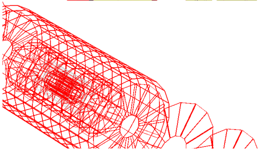
- in-memory translation of geometry from **TGeo** to **Geant4**
  - materials, solids, limit sets, regions, logical volumes, placed volumes and physical volumes
- external configuration via *plugin mechanism*
  - supports configuration via XML, Python or ROOT-AClick
  - property mechanism to configure plugin instances
- use plugin mechanism to configure: **Generation, Event Action, Tracking Action, SensDetector, PhysicsList. . .**
- provides out of the box **Monte Carlo Truth** handling w/o record reduction
- provide large palette of standard HEP sensitive detectors
  - user can add their own dedicated implementations

once you have defined your detector in DD4hep, you can very easily run a full Geant4 simulation

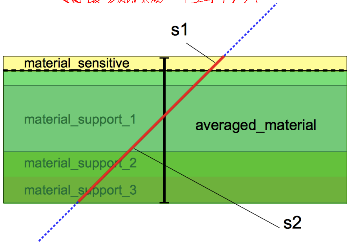
DD4hep - DDRec, interface for reconstruction

- dedicated `DetectorData` classes, describe high level view of generic detectors w/ extends, #layers, thicknesses,...
- `CellIDPositionConverter`: convert cellID to position and vice versa
- `MaterialManager`: access material properties at any point or along any straight line

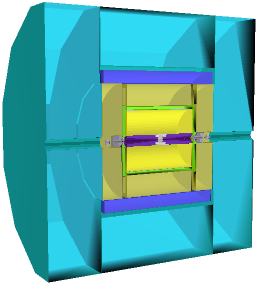


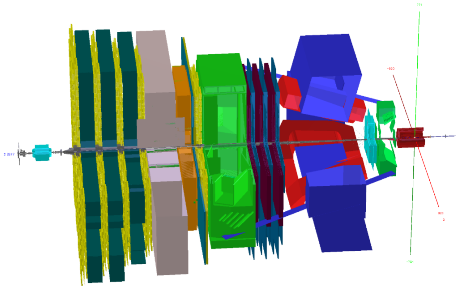


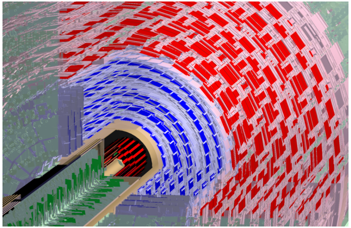
- dedicated `Surface` classes for Tracking
  - provide material properties for *measurement* and *dead material* layers
  - averaged along surface thickness



DD4hep - users







- used by ILC, CLICdp, FCC LHCb, under evaluation by CMS

new users are welcome

- maybe also non-HEP ??

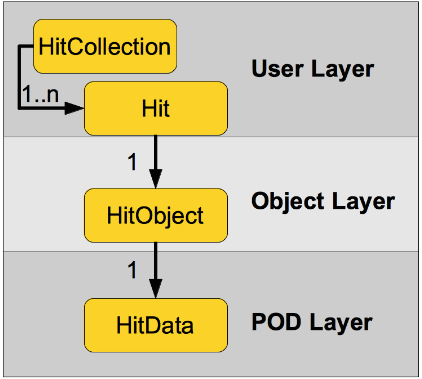
PODIO

Introduction and Motivation for PODIO

- **PODIO** is a generic EDM-toolkit for HEP
  - joined DESY and CERN project in **AIDA2020**, since 2015
  - also one of the first projects adopted by the *HEP Software Foundation*
- aim to overcome shortcomings of previous EDMs at LHC and in LC community:
  - *overly complex, deep object hierarchies with strong use of inheritance, non-optimal I/O performance*
- use **PODs** for the event data objects (**P**lain-**O**ld-**D**ata object)
- developed in context of the **FCC** study, allowing re-use by other HEP groups
- planned application to **LCIO** (>95% done)
- see: https://github.com/AIDASoft/podio

Implementation: the three PODIO layers

- user layer (API):
  - handles to EDM objects (e.g. **Hit**)
  - collection of EDM object handles (e.g. **HitCollection**).
- object layer
  - transient objects (e.g. **HitObject**) handling vector members and *references* to other objects
- POD layer
  - the actual POD data structures holding the persistent information (e.g. **HitData**)



Object ownership in PODIO

clear design of ownership (hard to make mistakes) in two stages:

objects added to event store are *owned by event store*

```
auto& hits = store.create<HitCollection>("hits") ;
auto h1 = hits.create( 1.,2.,3.,42.) ; // init w/ values
auto h2 = hits.create() ; // default construct
h2.energy( 42.) ;
```

objects created stand-alone are *reference counted* and automatically garbage collected:

```
auto h3 = Hit() ;
auto h4 = Hit() ;
hits.push_back( h3 ) ;
//  h1,h2,h3 are automatically deleted with collection
//  h4 is garbage collected
```

Relations between Objects

allow to have 1-1, 1-N or N-M relationships, e.g.

```
auto& hits = store.create<HitCollection>("hits");
auto& clusters = store.create<ClusterCollection>("clusters");
auto hit1 = hits.create(); auto hit2 = hits.create();
auto cluster = clusters.create();
cluster.addHit(hit1);
cluster.addHit(hit2);
```

referenced objects can be accessed via iterator or directly

```
for( auto h = cluster.Hits_begin(),end = cluster.Hits_end(); h!=end ++h){
    std::cout << h->energy() << std::endl;
}
auto hit  = cluster.Hits(42);
```

also standalone relations between arbitrary EDM objects

Code generation

- code (C++/Python) for the EDM classes is generated from yaml files
- EDM objects (data structures) are built from
  - basic type data members
  - components (structs of basic types)
  - references to other objects
- additional user code (member functions) can be defined in the yaml files

```yaml
# LCIO MCParticle
MCParticle:
  Description: "LCIO MC Particle"
  Author : "F.Gaede, B. Hegner"
  Members:
    - int pDG                 // PDG code of the particle
    - int generatorStatus     // status as defined by the generator
    - int simulatorStatus     // status from the simulation
     #...
  OneToManyRelations:
    - MCParticle parents // The parents of this particle.
    - MCParticle daughters // The daughters this particle.
  ExtraCode:
    const_declaration:
    "bool isCreatedInSimulation() const {
        return simulatorStatus() != 0 ;
     } \n"
```

Python Interface

- Python is treated as first class citizen in the provided library
- can use *pythonic* code for iterators etc.
- implemented with PyROOT and some special usability code in Python

Python code example:

```
store = EventStore(filenames)
for i, event in enumerate(store):
  hits = store.get("hits")
  for h in hits:
    print h.energy()
```

I/O implementation

- PODIO's I/O is still rather trivial at the moment
- PODs are directly stored using **ROOT I/O**
  - auto generated streamer code via dictionary
  - not properly optimized for PODs yet
  - object references are translated into *ObjectIDs* before being stored
- **To-Do-item**:
  - implement a direct binary I/O (storing array of structs) for performance comparison with ROOT and to demonstrate the potential performance advantage of storing PODs

GSoC project 2019

- implement an **HDF5** persistency layer for PODIO
- https://hepsoftwarefoundation.org/gsoc/2019/proposal_PODIOHDF5.html
- might make PODIO also interesting for non-HEP users !?

Summary and Outlook

- Linear Collider community has a long tradition of developing *'generic'* software tools
  - needed to support several LC projects with limited man power
- presented two recent examples:
- DD4hep: detector description for HEP
- PODIO: EDM toolkit for efficient I/O of event data using PODs

Outlook

- will continue to develop and improve these tools within DMA
  - investigate **non-HEP** use cases
- follow similar strategy for other/new software tools developed in DMA, i.e:
- develop a solution for a given problem but keep more general application in mind from the start