EDM Toolkit

Benedikt Hegner, CERN
Frank Gaede, DESY
AIDA2020 First Annual Meeting
DESY, June 14-17, 2016

AIDA 2020

Outline

- Introduction and Motivation
- Design of PODIO
- Implementation
- Open issues
- Summary and Outlook

Why a new data model library?

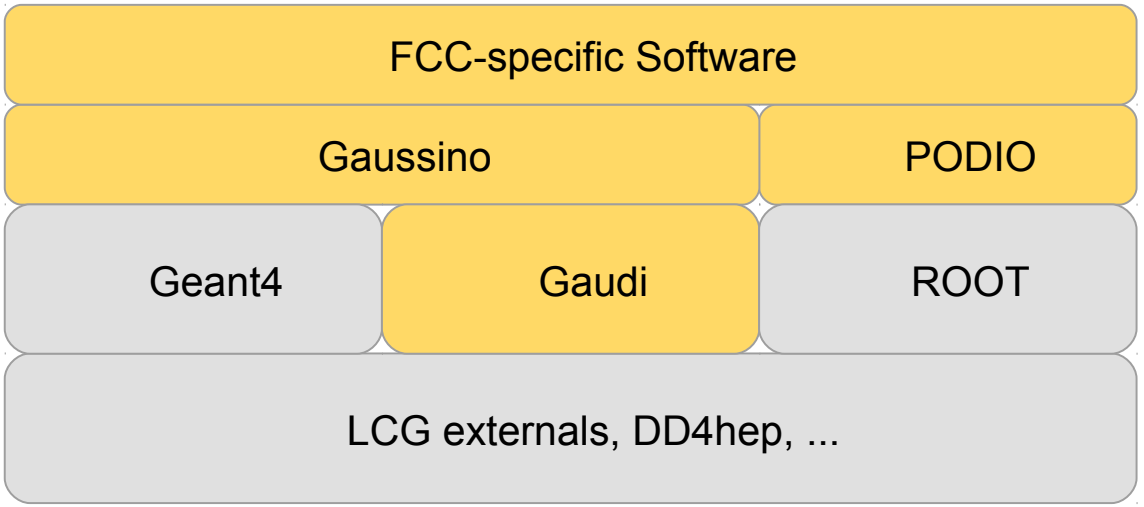
- LHC experiments have overly complex EDMs
  - strong use of inheritance and polymorphism
    - state of the art when the code was written
  - rather expensive virtual calls and memory operations
  - deep object hierarchies

- LCIO used in linear collider community somewhat better in complexity of EDM
  - actual I/O suffers from same issues as LHC solutions

- new activities like the FCC are an opportunity to do better this time
  - solve problem in a generic way

General Context

- PODIO (POD I/O) is one of the many components in the FCC software stack aiming at more general usability in HEP
- Part of the AIDA2020 activities (together with DD4hep, etc)
- One of the first projects in the HEP Software Foundation
  - The guinea pig for the project best-practices document



Driving Design Considerations

- Simple Memory Model
  - Concrete data are contained within plain-old-data structures (PODs)
  - Provide vectorization friendly (or at least not unfriendly) interfaces
- Simple Class Hierarchies
  - Wherever possible user concrete types
  - Favor composition over inheritance
- Simple interfaces on user side
  - In particular avoid ownership problems!
- Employ code generation
  - Quick turn-around for improvements on back-end
  - Easy creation of new types
- Support for both C++ and Python
- Thread-safety
- Use ROOT as first choice for I/O
  - Keep transient to persistent layer as thin as possible

What is a POD ?

- Plain-Old-Data object
- In C++11/14 a POD combines two concepts
  - support for static initialization (trivial class)
  - standard layout
    - no virtual functions and no virtual base classes
    - same access control (i.e. public,private,protected) for all non-static data members
    - …
- In short - a POD is closer to a classical C struct than a C++ object
- PODs are good for memory layout, memory and I/O operations

  ⇒ PODIO !

Separation of Concerns

- using PODs is a good idea - but they are a little bit too simplistic to address all needs

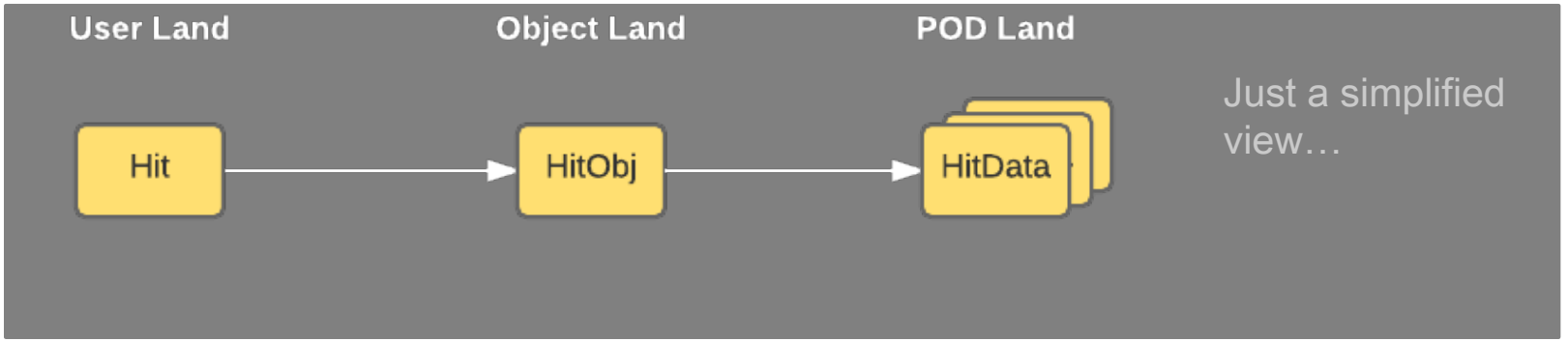
  => need smart layers on top of the PODs to

  - deal with object ownership
  - allow referencing between objects
  - seal with non-trivial I/O operations

- whenever performance is a concern - leave possibility to access the bare PODs

The PODIO layers

- user visible classes (e.g. Hit).
  - act as transparent references to the underlying data,
- a transient object
  - knowing about all data for a certain physics object, including inter-object references (e.g. HitObject),
- plain-old-data (POD)
  - holding the persistent object information (e.g. HitData), and
- a collection containing the user's objects (e.g. HitCollection).



Support for Vectorization

- key for vectorization is struct-of-arrays (SoA) vs. of arrays-of-structs (AoS)
  - which representation is better heavily depends on the use case
- separation of PODIO in layers allows to choose one representation at the implementation layer
  - choice hidden from the non-expert user.
- on demand transformation between complete SoA vs. AoS representations is highly inefficient

  $\Rightarrow$ the decision for the representation of a given data type has to be made upfront
- provide convenience methods for on demand transformation

```
auto x_array = hits.x<10>();
```

- need proper performance measurements on real use cases

Supported Syntax

- objects and collections can be created via factories, ensuring proper ownership:

```
auto& hits = store.create<HitCollection>("hits")
auto hit1 = hits.create(1.4,2.4,3.7,4.2);  // init with
values
auto hit2 = hits.create(); // default-construct object
hit2.energy(42.23);
```

- objects can be created standalone - if not attached to a collection, they are automatically garbage collected:

```
auto hit1 = Hit();
auto hit2 = Hit();
...
hits.push_back(hit1);
...
<automatic deletion of hit2>
```

Object Ownership

- unclear object ownership and memory leaks are a common problem

  $\Rightarrow$ make it as hard as possible to do mistakes

- in PODIO there are two stages in object ownership
  - before registering data into an event store

    $\Rightarrow$ reference counted
  - after adding data into event store

    $\Rightarrow$ ownership with event store

- additional costs on object creation time and no costs later

Relation between Objects

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

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

auto& clusters = store.create<ClusterCollection>("clusters");
auto cluster = clusters.create();

cluster.addHit(hit1);
cluster.addHit(hit2);
```

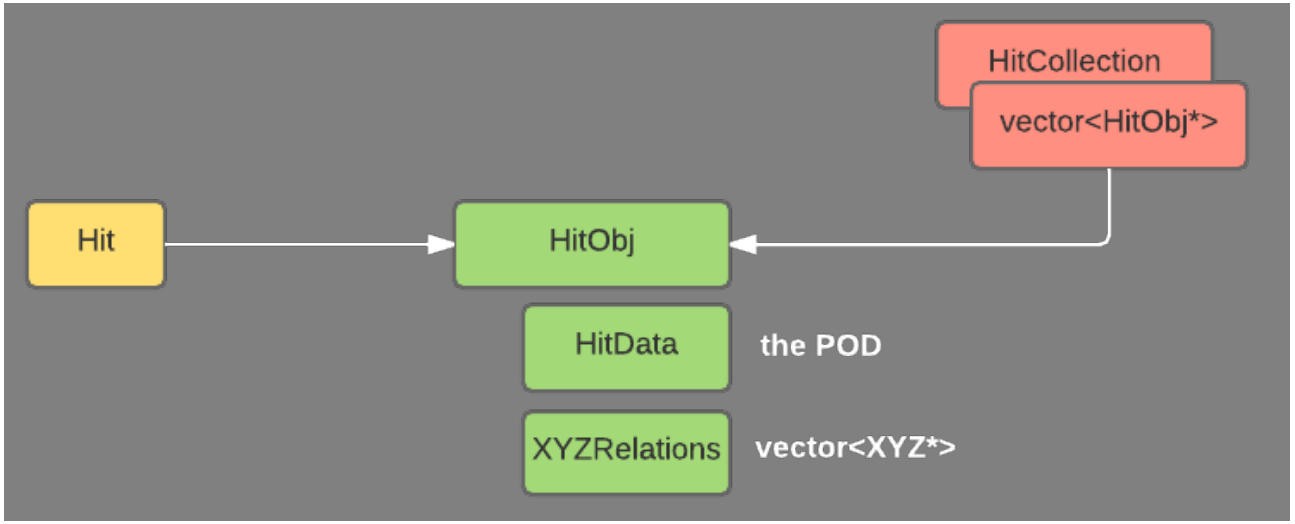
- referenced objects can be accessed via iterator or directly

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

auto hit  = cluster.Hits(<aNumber>);
```

Relations Details

- relations are handled outside the PODs
- the “Object Land” manages the lookup in memory
- every object in PODIO is uniquely identified by collectionID + index
- during I/O every reference is being replaced by its Object ID



Code generation

- code (C++/Python) for the EDM classes is auto 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

```
# 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.
```

Python Interface

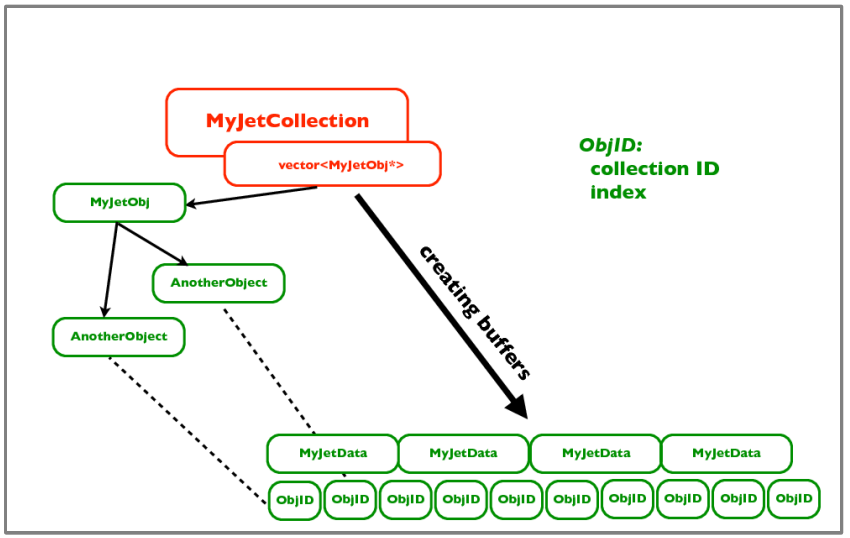
- Python is treated as first class citizen in the provided library:

```
with EventStore(filenames) as store:
   for event in store:
      hits = store.get("hits")
      for hit in hits:
         print hit.energy()
```

- implemented with PyROOT and some special usability code in Python

I/O implementation

- PODIO's I/O is rather trivial at the moment
- PODs are directly stored using ROOT
  - not properly optimized for PODs yet
- object references are being translated into ObjID and then stored



- need to implement a direct binary I/O (storing array of structs) for performance comparison with ROOT

PODIO in use

- PODIO is actively used by the FCC study efforts
  - in combination with Gaudi
  - “Standalone” for other C++ and Python applications
  - current data model definitions are in *fcc-edm*

- currently investigating the use of PODIO as evolution of LCIO
  - improve the I/O performance - keep the EDM (plcio)

- LHCb is interested in PODIO for their data model upgrade

  $\Rightarrow$ lhcbio demonstrator created during a coding sprint

Future Work

- implement missing features
  - vector members...
- polish the rough edges
  - many iterations on different design ideas left some remnants
  - in particular in the SoA support!
- finish the design document - Milestone M 3.2
- performance measurements comparisons
  - e.g. with HEPMC3, LCIO::MCParticles, …
- eventually move into maintenance mode and support LHCb and LC community in evaluating / adapting PODIO for their needs

Summary and Outlook

- EDM toolkit PODIO developed in context of FCC (and LC) with general HEP in mind
  - storing EDM objects as PODs
  - using ROOT I/O - others to follow
  - code for C++ and Python

- first implementation in use by FCC
- under evaluation for LC

- design document (Milestone M 3.2) in preparation

Links and Pointers

- GitHub repository + docs:
  https://github.com/hegner/podio
- doxygen page
  https://fccsw.web.cern.ch/fccsw/podio/index.html
- issue tracker
  https://sft.its.cern.ch/jira/projects/PODIO
- general FCC software documentation page
  https://fccsw.web.cern.ch/fccsw/
- plcio (EDM for LCIO w/ podio) git repository:
  https://stash.desy.de/projects/IL/repos/plcio