EDM Toolkit - **PODIO**

F.Gaede, DESY B. Hegner CERN

AIDA2020 Annual Meeting, Apr 6,2017

Outline

- Introduction and Motivation
- Design and Implementation
- Recent Developments
- Next Steps
- Summary and Outlook

Introduction and Motivation for PODIO

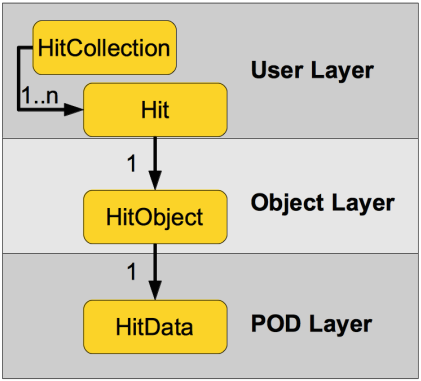
- Motivation: experience from previous event data models (EDMs) revealed some shortcomings:
  - overly complex, deep object hierarchies with strong use of inheritance (LHC)
  - virtual function calls and non-optimal I/O performance (LC)
- new projects, like the FCC are an opportunity to do better this time
  - use what worked well - throw away what didn't
- PODIO is developed in context of the **FCC** study
  - addressing the problem in a generic way
  - allowing potential re-use by other HEP groups
- planned application to **LC** (LCIO) will provide cross check for generality
- one of the first projects adopted by the **HEP Software Foundation**

Design Choices for PODIO

- use **PODs** for the event data objects (**P**lain-**O**ld-**D**ata object)
  - simple memory model
  - support for vectorization
  - allowing for(very) fast I/O
- simple user API and class hierarchies
  - use concrete types: favor composition over inheritance
- apply code generation
- support for C++ and Python
- build in thread-safety
- allow for different I/O implementations

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 standalone 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

```
# 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 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
  - planned for this summer (CERN summerstudent programme)

Recent Developments (since last annual meeting)

- moved to use C++14
- implemented support for I/O of *std::array*
  - needed iteration with ROOT developers
  - requires recent ROOT version
- simplified the example event store
- fixed a number of bugs and issues
- iterated on definition of FCC EDM
  - making use of new std::array features
- implemented streamer methods for EDM classes
- implemented prototype for usage with LCIO

Implementation of streamer methods

- automatically create two methods for each EDM object:
  - proven to be extremely useful in LCIO for debugging and analysis

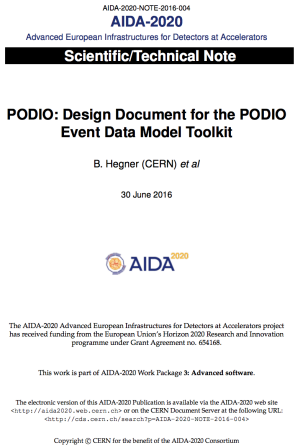
```
// ostream operator for detailed dump of EDM object
// prints every member of the POD struct in one line
std::ostream& operator<<(std::ostream& o,const ConstMCParticle& v);
```

```
// ostream operator for tabular dump of EDM collection
// prints one line per EDM object
std::ostream& operator<<(std::ostream& o,const MCParticleCollection& v);
```

Milestones and Deliverables

Name	What	When
MS19	Design document for EDM Toolkit	M14
MS90	Application of EDM Toolkit to LC	M44
D3.4	Event Data Model Toolkit	M40

- reached MS19 on time
- some time to go for MS90 and D3.4



AIDA-2020-NOTE-2016-004

AIDA-2020

Advanced European Infrastructures for Detectors at Accelerators

Scientific/Technical Note

PODIO: Design Document for the PODIO Event Data Model Toolkit

B. Hegner (CERN) *et al*

30 June 2016

AIDA 2020

The AIDA-2020 Advanced European Infrastructures for Detectors at Accelerators project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement no. 654168.

This work is part of AIDA-2020 Work Package **3: Advanced software.**

The electronic version of this AIDA-2020 Publication is available via the AIDA-2020 web site <http://aida2020.web.cern.ch> or on the CERN Document Server at the following URL: <http://cds.cern.ch/search?p=AIDA-2020-NOTE-2016-004>

Next Steps

implement binary I/O making use of array-of-POD

- need to benchmark the reading performance against current ROOT I/O
  - uses member wise streaming and XDR (big endian)

finalize implementation of LCIO with PODIO

- prototype implementation with ~90% of LCIO exists, missing:
  - vector/array member types -> use new *std::array* feature
  - meta data for run, event and collections
  - suitable *Event Store*, compatible w/ current LCIO/Marlin

modify the treatment of constness

- current implementation has extra types for *const* objects, e.g. *ConstMCParticle*
- prototype implementation exists that ensures constness transparently
  - need to be merged with *master* branch

Summary and Outlook

- EDM toolkit PODIO developed in context of FCC/LC
  - with general HEP in mind
  - storing EDM objects in arrays of PODs
  - currently using ROOT I/O
  - code automatically generated for C++ and Python
  - first implementation in full use by FCC
  - under evaluation for LC

Outlook

- address open points
  - alternative binary I/O
  - full implementation of LCIO
  - modified treatment of *constness*

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`
- plcio (EDM for LCIO w/ podio) git repository:
  - `https://stash.desy.de/projects/IL/repos/plcio`
- PODIO Library Design Document:
  - `http://cds.cern.ch/record/2212785`