PODIO

The EDM toolkit

F.Gaede DESY

AIDA2020 WP3 Meeting, Apr 23, 2020

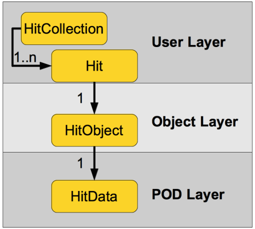
Outline

- Reminder of PODIO features
- Recent developments
    - pLCIO, EDM4hep
    - HDF5, SIO I/O layers
    - I/O benchmarking
- Next Steps and Plans

Reminder: main PODIO features

- PODIO EDM toolkit based on the use of **PODs**
- originally developed in context of FCC
- application to *CLIC/ILC* (**LCIO**) planned from the start

- main features:
  - three implementation layers
  - well defined object ownership
  - relations between objects
  - C++ code generation with Python
  - Python binding/interface



for more details see talk from last annual meeting:

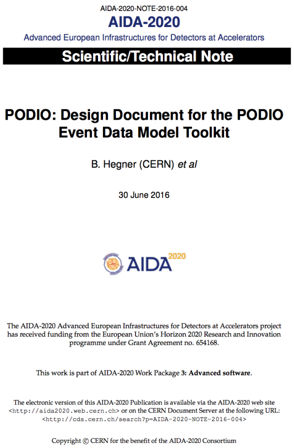
https://indico.cern.ch/event/773447/contributions/337027

Status at Annual Meeting 2019

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

status

- all Milestones and Deliverables **reached on time**
- since then continued to improve PODIO



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

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>

Copyright © CERN for the benefit of the AIDA-2020 Consortium

Developments since last Annual Meeting

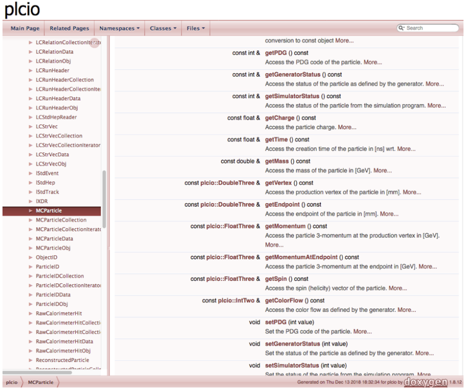
- finalized pLCIO
    - re-implementation of LCIO in PODIO
- started new project: **EDM4hep**

- investigated implementation of **HDF5** I/O layer
- implemented POD-based binary I/O layer with **SIO**
    - benchmarking of SIO performance

- many small improvements
- re-organized cmake builds, CI, etc
- made compatible w/ Python 3

LCIO EDM in PODIO: pLCIO

- **pLCIO**: package that implements **complete LCIO EDM** (almost):
- original idea to be able to create classes that are almost 100% backward compatible did not fully work out
  - true for most of the actual member functions of the EDM classes
  - not true for handling of collections and collection types, creation of objects, user defined parameters, . . .
- planned transition from LCIO to pLCIO would be feasible at ‘*reasonable cost*’



- LCRelationCollectionIterat
- LCRelationData
- LCRelationObj
- LCRunHeader
- LCRunHeaderCollection
- LCRunHeaderCollectionIte
- LCRunHeaderData
- LCRunHeaderObj
- LCStdHepReader
- LCStrVec
- LCStrVecCollection
- LCStrVecCollectionIterator
- LCStrVecData
- LCStrVecObj
- lStdEvent
- lStdHep
- lStdTrack
- lXDR
- MCParticle
- MCParticleCollection
- MCParticleCollectionIterat
- MCParticleData
- MCParticleObj
- ObjectID
- ParticleID
- ParticleIDCollection
- ParticleIDCollectionIterato
- ParticleIDData
- ParticleIDObj
- RawCalorimeterHit
- RawCalorimeterHitCollecti
- RawCalorimeterHitCollecti
- RawCalorimeterHitData
- RawCalorimeterHitObj
- ReconstructedParticle

	conversion to const object More...
const int &	getPDG () const Access the PDG code of the particle. More...
const int &	getGeneratorStatus () const Access the status of the particle as defined by the generator. More...
const int &	getSimulatorStatus () const Access the status of the particle from the simulation program. More...
const float &	getCharge () const Access the particle charge. More...
const float &	getTime () const Access the creation time of the particle in [ns] wrt. More...
const double &	getMass () const Access the mass of the particle in [GeV]. More...
const plcio::DoubleThree &	getVertex () const Access the production vertex of the particle in [mm]. More...
const plcio::DoubleThree &	getEndpoint () const Access the endpoint of the particle in [mm]. More...
const plcio::FloatThree &	getMomentum () const Access the particle 3-momentum at the production vertex in [GeV]. More...
const plcio::FloatThree &	getMomentumAtEndpoint () const Access the particle 3-momentum at the endpoint in [GeV]. More...
const plcio::FloatThree &	getSpin () const Access the spin (helicity) vector of the particle. More...
const plcio::IntTwo &	getColorFlow () const Access the color flow as defined by the generator. More...
void	setPDG (int value) Set the PDG code of the particle. More...
void	setGeneratorStatus (int value) Set the status of the particle as defined by the generator. More...
void	setSimulatorStatus (int value) Set the status of the particle from the simulation program. More

Generated on Thu Dec 13 2018 18:32:34 for plcio by doxygen 1.8.12

- potentially we could use this transition to *evolve the LCIO EDM*
  - or go to **EDM4hep** directly . . .

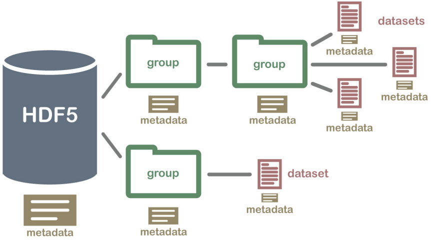
EDM4hep

- idea to have a common event data model toolkit for all future HEP experiments
  - Higgs factories (CEPC, CLIC, FCC-ee, ILC), muon collider, charm-tau factories,. . . .
  - inspired by what LCIO has done for linear collider projects
  - will be at the heart of the *Turnkey Software Stack (Key4hep)*
- base the EDM on experience of **LCIO** and FCC-edm
  - use what worked well over the years, address *idiosyncrasies* and *'historical developments'*
- use **PODIO** for the actual implementation
- project still in somewhat early phase
  - so far have *MCParticle*, *SimTrackerHit*, *SimCalorimeterHit*, *Track* and *TrackState*
  - see: https://github.com/key4hep/EDM4hep

Additional I/O implementations: HDF5

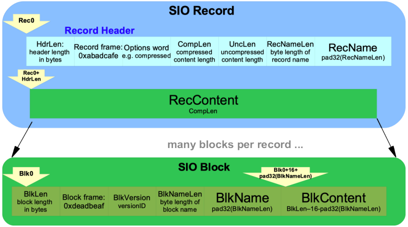
- had *Google Summer of Code* project last year
- **Implementation of an HDF5 I/O layer for PODIO**
  - task turned out to be a bit more involved than anticipated originally
- developed prototype code for writing some example data structures to HDF5 files
  - mapping events to groups and collections to datasets
  - unclear if this is optimal way of doing it in HDF5 !?
  - HEP data is inherently *heterogeneous*. . .

- further work needed on HDF5 implementation . . .
- potentially useful for *Machine Learning* !?



Additional I/O implementations: SIO

- use a simple binary I/O for storing *array-of-structs* directly
- *SIO* - simple I/O, used in **LCIO** (>15 years)
- recently rewritten to be *thread-safe* and allow for parallel *compression* and *un-packing*
  - see talk by R.Ete on MarlinMT
- event data stored in *records* with many *blocks* (collections)
  - compression of complete records/events
- PODIO data collections are stored as they are in memory
  - using essentially *memcpy*
  - no *byte-swapping* done currently


SIO Record
Rec0
Record Header
HdrLen: header length in bytes
Record frame: 0xabadcafe
Options word e.g. compressed
CompLen compressed content length
UncLen uncompressed content length
RecNameLen byte length of record name
RecName pad32(RecNameLen)
Rec0+ HdrLen
RecContent
CompLen
many blocks per record ...
SIO Block
Blk0
Blk0+16+ pad32(BlkNameLen)
BlkLen block length in bytes
Block frame: 0xdeadbeaf
BlkVersion versionID
BlkNameLen byte length of block name
BlkName pad32(BlkNameLen)
BlkContent BlkLen–16-pad32(BlkNameLen)

Compare ROOT to SIO I/O implementation

- compare the performance of these two different I/O implementations:

- ROOT (6.18)
  - *optimized columnar storage*
  - applying XDR (*byte swapping*) for machine independence
  - compression per branch

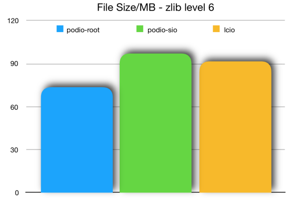
- SIO
  - store plain *array-of-structs*
  - no *byte swapping*
  - compression per record

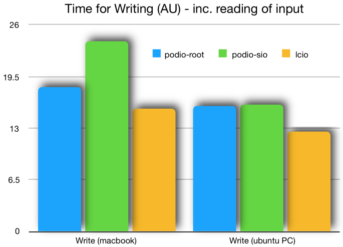
- *"benchmark"* used here:
- convert binary *stdhep* generator ($e^+e^- \rightarrow t\bar{t}$ @ $500 \mathrm{GeV}$) files to the different formats:
  - podio-ROOT, podio-SIO, LCIO

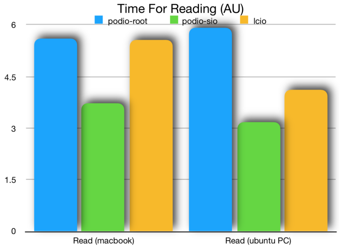
- write and read files
  - very little computation on reading

- two machines:
  - Mac-book-pro with SSD, llvm 10.01
  - x86, quad-core PC w/ Ubuntu 16.04, gcc 5.4

Benchmarking the ROOT vs SIO implementation







- ROOTI/O file size is ~**76%** of SIO file size
- ROOTI/O writing takes **75-99%** of SIO writing time [1]
- ROOTI/O reading takes **150-186%** of SIO reading time

- the improvement in reading speed justifies further investigation
  - maybe even interesting to eventually develop a *POD-mode* in ROOT-I/O ?

[1] includes the time for reading stdhep input file

Next Steps and Plans

- currently working on additional **generic meta data** that can be stored for
    - runs, events and collections

- implement/provide documentation for how to do **schema evolution**
    - absolutely needed for using PODIO in a production environment with *EDM4hep*

- continue the work on the **HDF5** implementation
- generalize the use of **different I/O implementations**
    - HDF5, SIO, potentially others ? . . .
    - need to iterate on and standardize the interface between *EventStore* and *Reader/Writer* implementations

- work planned in AIDAinnova successor project to AIDA2020

Links and Pointers

- GitHub repository + docs:
  - https://github.com/aidasoft/podio
- issue tracker (use Github issues):
  - https://github.com/aidasoft/podio
- EDM4hep:
  - https://github.com/key4hep/EDM4hep
- 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