PODIO

Recent developments in the Plain Old Data EDM toolkit

F.Gaede DESY
B. Hegner, G.A.Stewart CERN

CHEP 2019, Adelaide, Nov 7, 2019
Outline

- Introduction
- Reminder of PODIO features
- Recent developments
  - vector members
  - new I/O implementations
- Benchmarking I/O performances
- Next Steps
- Summary and Outlook
Introduction

- PODIO is an *Event-Data-Model* (EDM) toolkit for HEP
  - developed in Horizon2020 project *AIDA2020*
  - based on the use of **PODs** for the event data objects (**Plain-Old-Data object**)

- PODIO originally developed in context of the **FCC** study
  - addressing the problem in a generic way
  - allowing potential re-use by other HEP groups
- application to **CLIC/ILC** (**LCIO**) planned from the start

- one of the first projects adopted by the **HEP Software Foundation (HSF)**

- PODIO is used since its first release by the FCC studies (see **FCC-edm**)
  - adaptation to linear collider and LCIO ongoing (see later this talk)
Implementation: the three PODIO layers

- **user layer (API):**
  - handles to EDM objects (e.g. **Hit**)
  - collections of EDM object handles (e.g. **HitCollection**).

- **object layer**
  - transient objects (e.g. **HitObject**)
  - handling references to other objects and vector members

- **POD layer**
  - the actual POD data structures holding the persistent information (e.g. **HitData**)

Direct access to POD also possible - if needed for performance reason
Core Features of PODIO I

- clear design of **ownership** (hard to make mistakes) in two stages:
  - objects added to event store are **owned by event store**
  - objects created stand-alone are **reference counted** and automatically garbage collected:
- allow to have **1-1, 1-N or N-M relationships**
  - referenced objects can be accessed via iterator or directly
  - also stand-alone relations between arbitrary EDM objects

- **code generation** (C++/Python) for EDM classes from **yaml** files
  - EDM objects (data structures) are built from basic types and components
  - additional user code (member functions) can be defined in the yaml files
Core Features of PODIO II

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

- PODIO I/O is based on **ROOT-I/O** at the moment:
  - auto generated streamer code via dictionary
  - one branch per pod member
  - not really optimized for PODs
  - object references are translated into `ObjectIDs`
    - `ObjectIDs = CollectionID << 32 + CollectionIndex`
Recent Developments in PODIO (since CHEP 2016)

- moved to use C++14 and C++17
- implemented support for I/O of `std::array`
  - needed iteration with ROOT developers
- simplified the example `EventStore`
- implemented streaming with `operator<<(...)` for all EDM classes and collections:

```cpp
std::ostream& operator<<(std::ostream& o, const ConstMCParticle& v);
std::ostream& operator<<(std::ostream& o, const MCParticleCollection& v);
```

- moved GitHub repository to: https://github.com/aidasoft/podio
  - added some standard templates from iLCSoft for release notes, issues, etc.
- implemented **vector member** streaming
Vector Members in PODIO

- strictly speaking vector members break the *PODness* of the data classes
- treat vector members analogous to the reference vectors:
  - hold vector in *Object*-layer, start/end in *POD*-layer and iterator in *User*-layer
  - stream vector members as one large vector per collection
  - after reading keep in one large vector to minimize copying

```
plcio::LCGenericObject:
Description: "LCIO LCGenericObject"
Author : "F.Gaede, DESY"
Members:
- int isFixedSize // all objects have a fixed size
- std::string typeName // type name of the user class
- std::string dataDescription // data description string
VectorMembers:
- int intVals // integer value at given index.
- float floatVals // float value at given index.
- double doubleVals // double value at given index.
```
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’

potentially we could use this transition to evolve the LCIO EDM
- or go to EDM4hep directly...
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*
- see A.Sailer: Towards a Turnkey Software Stack for HEP Experiments

developed under umbrella of HSF

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 is in rather early phase

- so far have *MCParticle, SimTrackerHit* and *SimCalorimeterHit*
- *Track* and *TrackState* under discussion (should also fit w/ ACTS)
had *Google Summer of Code* project this 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 R.Ete: MarlinMT - parallelising the Marlin framework
- 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
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 \((e^+e^- \rightarrow t\bar{t} @ 500\text{GeV})\) 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

- ROOT I/O file size is approximately 76% of SIO file size.
- ROOT I/O writing takes 75-99% of SIO writing time.\(^1\)
- ROOT I/O reading takes 150-186% of SIO reading time.

The improvement in reading speed justifies further investigation.

\(^{1}\) includes the time for reading stdhep input file.
Next Steps and Plans

- continue the work on the **HDF5** implementation

- implement/provide documentation for how to do **schema evolution**
  - absolutely need for using PODIO in a production environment, e.g. with *EDM4hep*

- 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
EDM toolkit PODIO developed in *AIDA2020*, in context of FCC with general HEP in mind
  - already actively used by FCC
  - transition from LCIO to pLCIO: *work in progress*

recently adopted by **EDM4hep** project

started to implement **HDF5** I/O in **GSoC** project

developed alternative binary I/O with **SIO**

  - initial benchmarking show some advantages for reading in as *array-of-structs*

**next steps**

- continue work on **HDF5** and implement *schema evolution*
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/HSF/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