EDM4hep and PODIO

Introduction and Overview

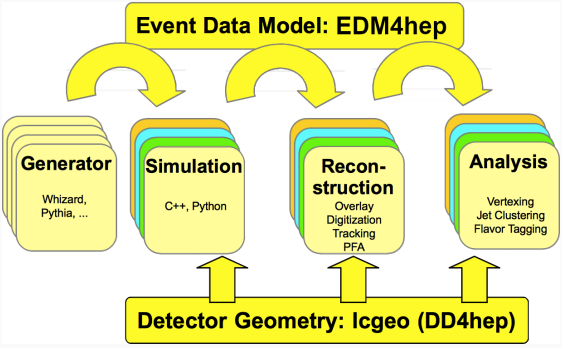
Thomas Madlener

LUXE Computing & DAQ meeting

Nov 09, 2022

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

The EDM at the core of HEP software

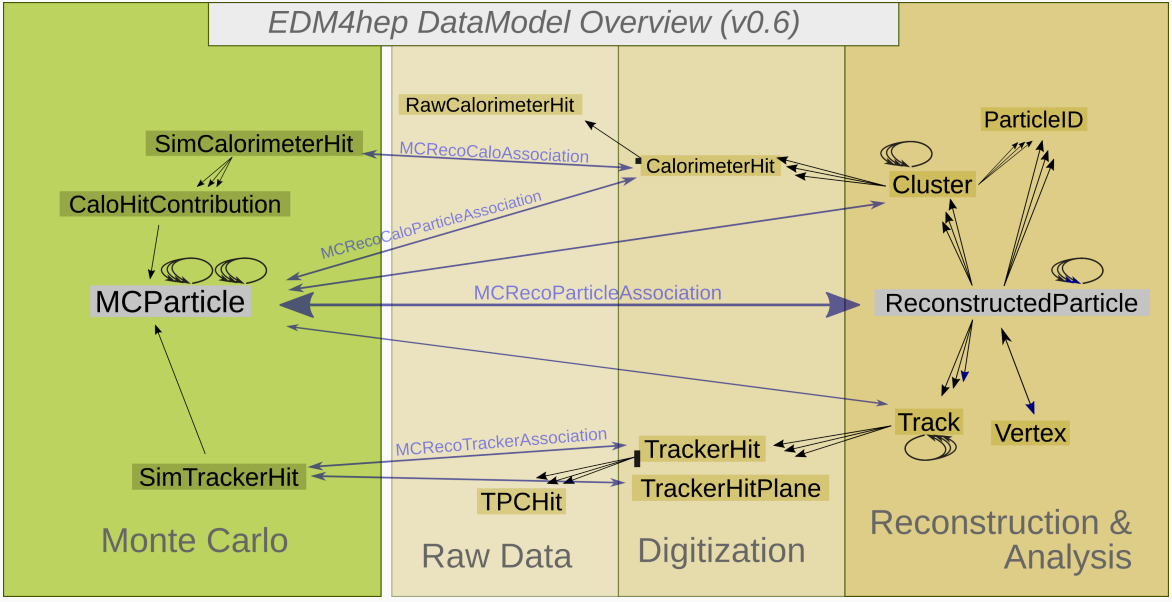


- Different components of HEP experiment software have to talk to each other
- The event data model defines the language for this communication
- Users express their ideas in the same language

EDM4hep - Goals & Motivation

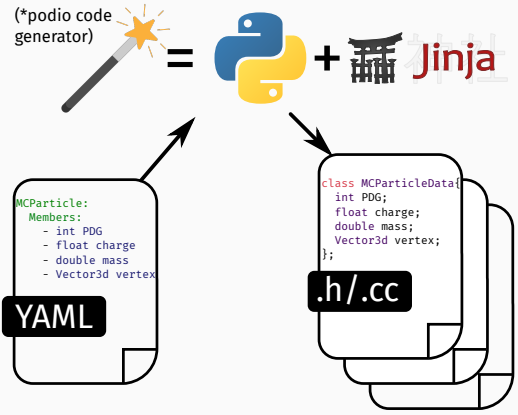
- The Key4hep project aims to develop a common software stack for all future collider projects
  - ILC, CLIC, FCC-ee & FCC-hh, CEPC, EIC, ...
- EDM4hep is the **shared, common EDM** that can be used by **all communities** in the Key4hep project (and others)
- EDM4hep has to uspport different use cases from these communities
- Efficiently implemented, support for multi-threading and with usage on heterogeneous resources in mind
- Built on experience from the “past” - mainly **LCIO**, which has been successfully shared by the LC communities

EDM4hep schema



podio as generator for EDM4hep

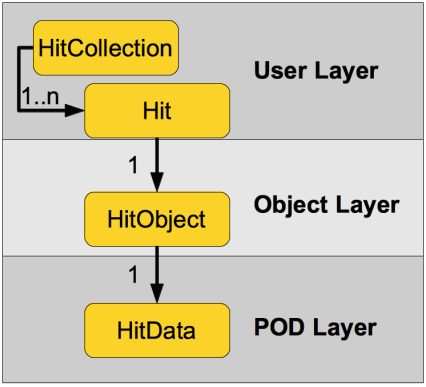
- Traditionally HEP c++ EDMs are heavily Object Oriented
- Use `podio` to generate thread safe code starting from a high level description
- Provide an easy to use interface to the users



AIDASoft/podio

The three layers of podio

- podio favors **composition over inheritance** and uses **plain-old-data (POD)** types wherever possible
- Layered design allows for efficient memory layout and performant I/O implementation



podio - datamodel definition

```yaml
components:
  edm4hep::Vector3f:
    Members: [float x, float y, float z]

datatypes:
  edm4hep::ReconstructedParticle:
    Description: "Reconstructed Particle"
    Author : "F.Gaede, DESY"
    Members:
      - edm4hep::Vector3f     momentum   // [GeV] particle momentum
      - std::array<float, 10> covMatrix  // energy-momentum covariance
    OneToOneRelations:
      - edm4hep::Vertex startVertex // start vertex associated to this particle
    OneToManyRelations:
      - edm4hep::Cluster clusters  // clusters that have been used for this particle
      - edm4hep::ReconstructedParticle particles // associated particles
    ExtraCode:
      declaration: "bool isCompund() const { return particles_size() > 0; }\n"

  edm4hep::ParticleID:
    VectorMembers:
      - float parameters // hypothesis params
```

*extracted from edm4hep.yaml

- Reusable components
- Fixed sized arrays as members
- *VectorMembers* for variable sized array members
- 1 – 1 and 1 – N relations
- Additional user-provided code

podio - features of generated code

```cpp
auto recos = ReconstructedParticleCollection();
// ... fill ...
for (auto reco : recos) {
  auto vtx = reco.getStartVertex();
  for (auto rp : reco.getParticles()) {
    auto mom = rp.getMomentum();
  }
}
```

← c++17 code with “value semantics”

↓ Python bindings via PyROOT

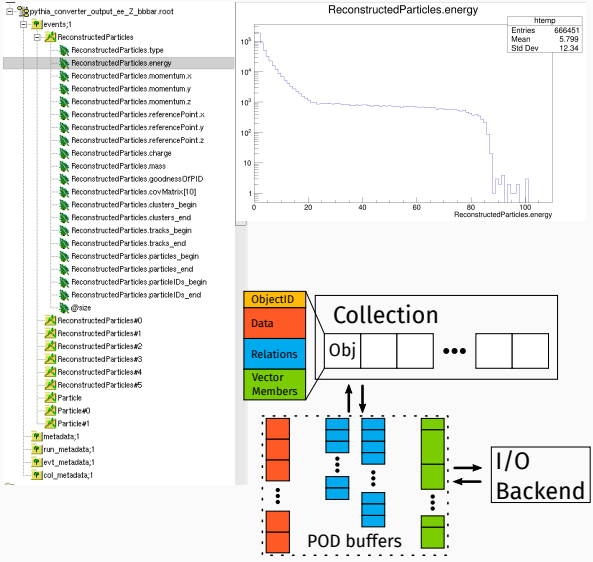
```python
recos = ReconstructedParticleCollection()
#... fill ...
for reco in recos:
  vtx = reco.getStartVertex()
  for rp in reco.getParticles():
    mom = rp.getMomentum()
```

```python
d = ROOT.RDataFrame('events', 'events.root')
h = (d.Define('abs_pdg', 'abs(Particle.PDG)')
      .Define('mu_sel', 'abs_pdg == 13')
      .Define('mu_px',
              'Particle.momentum.x[mu_sel]')
      .Histo1D('mu_px'))
h.DrawCopy()
```

← Using `RDataFrame` to read ROOT files (`uproot` also possible)

podio supports different I/O backends

- Default **ROOT** backend
  - POD buffers are stored as branches in a `TTree`
  - Files can be interpreted **without EDM library**(!)
  - Can be used in `RDataFrame` or with `uproot`
- Alternative **SIO** backend
  - Persistency library used in `LCIO`
  - Complete events are stored as binary records
- Adding more I/O backends is possible



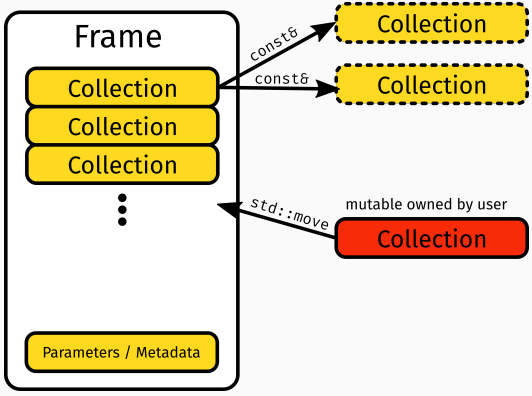
CMake interface for projects using podio

```cmake
find_package(PODIO)

# generate the c++ code from the yaml definition
PODIO_GENERATE_DATAMODEL(edm4hep edm4hep.yaml headers sources IO_BACKEND_HANDLERS "ROOT;SIO")
# compile the core data model shared library (no I/O)
PODIO_ADD_DATAMODEL_CORE_LIB(edm4hep "${headers}" "${sources}")
# generate and compile the ROOT I/O dictionary
PODIO_ADD_ROOT_IO_DICT(edm4hepDict edm4hep "${headers}" src/selection.xml)
# compile the SIOBlocks shared library for the SIO backend
PODIO_ADD_SIO_IO_BLOCKS(edm4hep "${headers}" "${sources}")

# Install the created targets
install(TARGETS edm4hep edm4hepDict edm4hepSioBlocks)
```

- Easy to use functions for integrating a podio generated EDM into a project
- Split into core EDM library and I/O handling for different backends
  - Pick what you need
  - I/O handling parts dynamically loaded by podio on startup

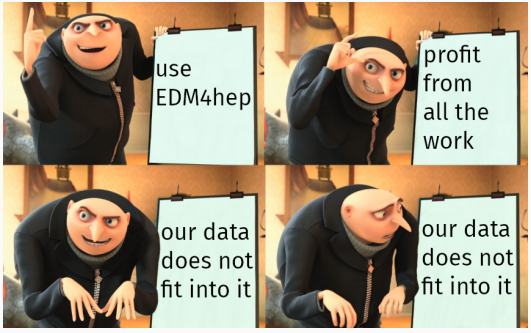
The `Frame` - A generalized (event) data container

- Container aggregating all relevant data
- Defines an *interval of validity* / category for contained data
  - Event, Run, readout frame, ...
- Easy to use and thread safe interface for data access
  - Immuatable read access only
  - Ownership model reflected in API
- Decouples I/O from operating on the data



Prototyping of new datatypes

- podio comes with a mechanism to extend existing (“upstream”) datamodels
- EDM4hep uses this for **prototyping new datatypes**
  - Have to avoid to fracture EDM4hep
  - Goal is always inclusion into EDM4hep
- Used in Key4hep for some detector prototyping
- Room for more detector concepts in EDM4hep!



Ongoing work & Future plans

- Release v1.0 with backwards compatibility from then on
  - Need to finish schema evolution work first
- Propagate Frame based I/O to all currently existing “customers”
  - Framework integration, ddsim (DD4hep) output, ...
- Implement currently missing features
  - E.g. User defined associations between arbitrary types
  - *Interface types* that allow for easier high level workflows (e.g. tracker hits for different technologies)
- Start exploring work on heterogeneous resources

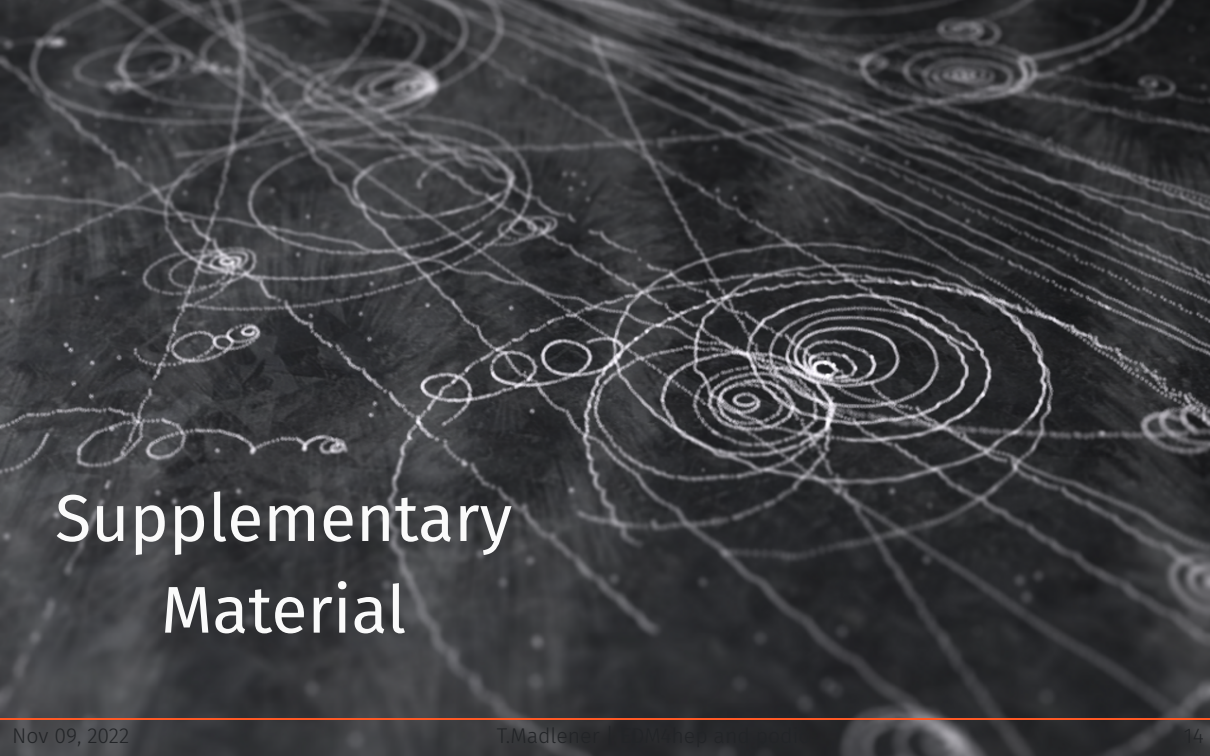
Summary

- EDM4hep is the shared, common EDM for the Key4hep project
  - Community effort is a success
- It is generated via the podio EDM toolkit
- Efficient implementation of data types and flexible I/O capabilities
- podio extension mechanism can be used to add new data types
- EDM4hep is open for new data types for not yet covered detector types

Pointers to software (re)sources

- EDM4hep
  - key4hep/EDM4hep
  - cern.ch/edm4hep
- podio
  - AIDASoft/podio
- Biweekly meetings for podio/EDM4hep discussion
  - indico.cern.ch/category/11461/

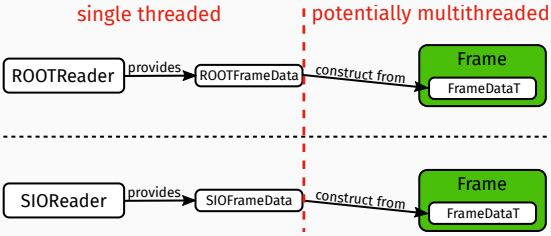
xkcd.com/138



Supplementary Material

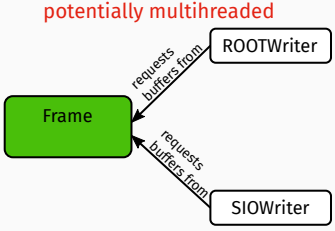
Frame I/O and multithreading model

- Readers provide data for a **complete Frame** in (almost) arbitrary format
- Assume that there is only one thread per file (i.e. Reader/Writer)



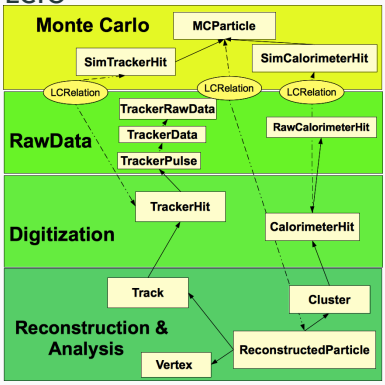
- Reading raw data and constructing a frame from it is a two step process
- Makes it possible to do unpacking on a different thread than the one that reads

- Writing can happen with multiple threads, e.g. each writer on its own thread
- Writers can write different contents, e.g. SIM & RECO into separate files
  - Need one writer “per content”

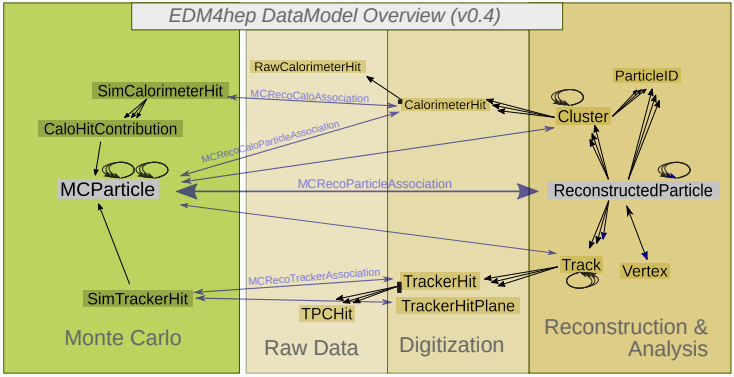


LCIO vs EDM4hep

LCIO

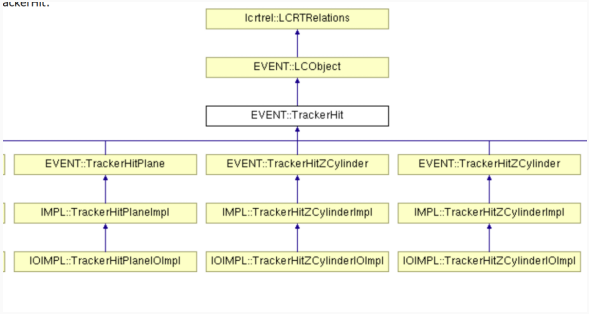


EDM4hep



- Since EDM4hep is based on LCIO the high-level structure is very similar
- Largest differences between the two are due to their implementations
- LCIO has over 15 years of usage. A lot of time to develop tools for it.
  - Not nearly as far with EDM4hep

Interface types



- LCIO uses “classic” polymorphism
  - Common `LCObject` base type for all data types
  - `Impl` classes offer the mutable interface
- This is used in some places to add some structure to data types
  - E.g. `TrackerHit` has various implementations different detector technologies
- Not solved for podio (and EDM4hep)