podio (almost) v1.0 - A first stable release of the EDM toolkit

Thomas Madlener (DESY)

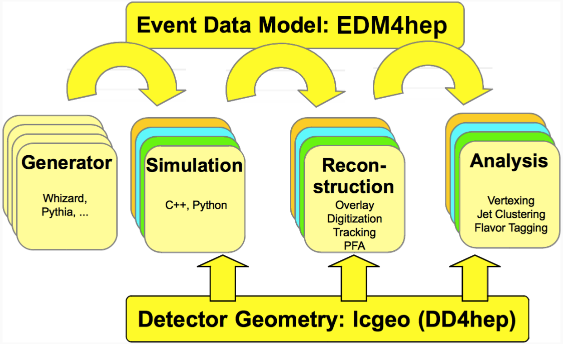
for the podio developers

CHEP 2023

May 11, 2023

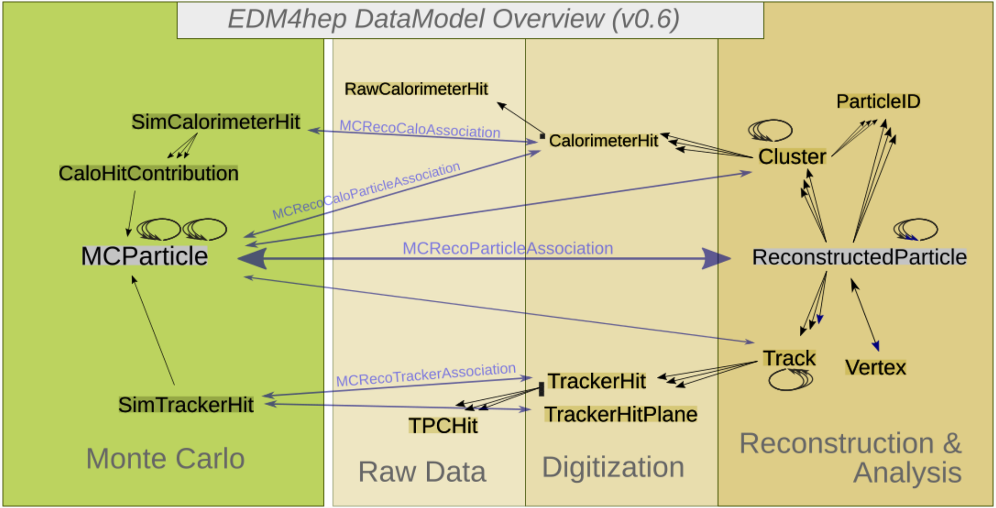
This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101004761.

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

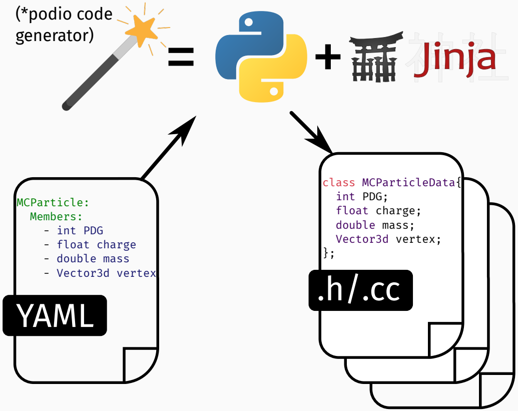
Example: EDM4hep



Common EDM for Key4hep project (Key4hep Progress Report on Integrations)

The podio EDM toolkit

- Implementing a performant event data model (EDM) is non-trivial
- Use `podio` to generate code starting from a high level description
- Provide an easy to use interface to the users
- Main customers
  - key4hep/EDM4hep
  - eic/EDM4eic



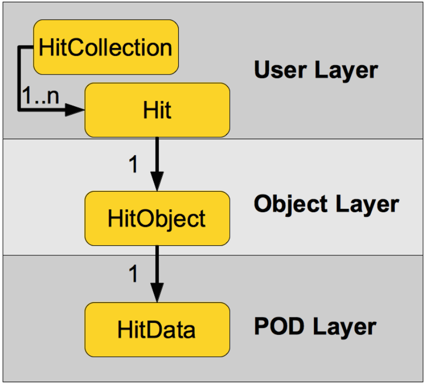
```yaml
MCParticle:
  Members:
    - int PDG
    - float charge
    - double mass
    - Vector3d vertex
```

```cpp
class MCParticleData{
  int PDG;
  float charge;
  double mass;
  Vector3d vertex;
};
```

AIDASoft/podio

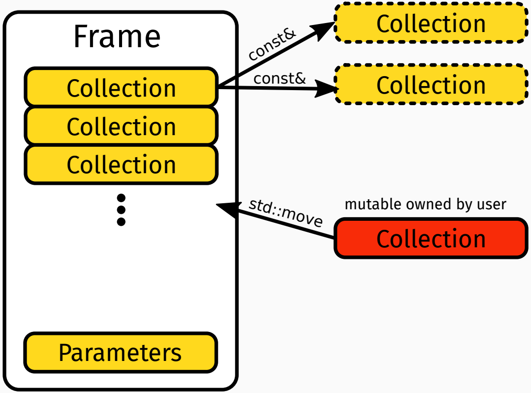
The three layers of podio

- 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



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

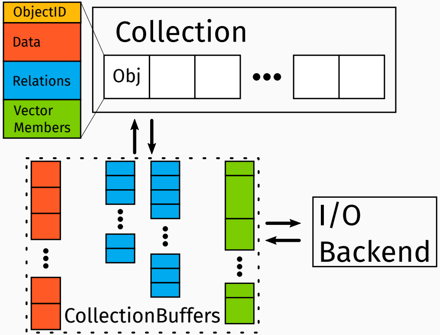
- *Type erased* container aggregating all relevant data
- Defines an *interval of validity* / category for contained data
    - Event, Run, readout frame, ...
- Easy to use and thread safe interface for data access
    - Immuatable read access only
    - Ownership model reflected in API
- Decouples I/O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

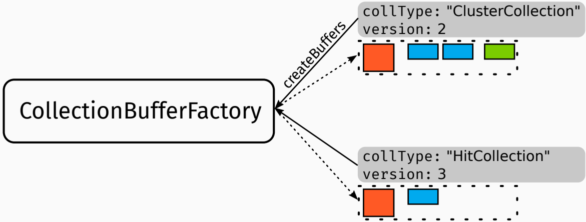


```
template<typename CollT>
const CollT& get(const std::string& name) const;

template<typename CollT, /*enable_if*/>
const CollT& put(CollT&& collection,
                 const std::string& name);
```

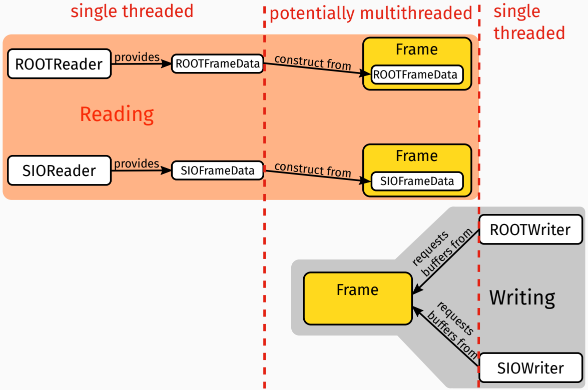
I/O low level basics

- I/O is based on collections
- `CollectionBuffer` holds all necessary data to (de)serialize a collection
  - Simple POD buffers (AoS)
  - I/O backend only needs to handle these
- `CollectionBufferFactory` creates empty buffers
  - (type, version) → `std::function`
  - Populated during datamodel library loading





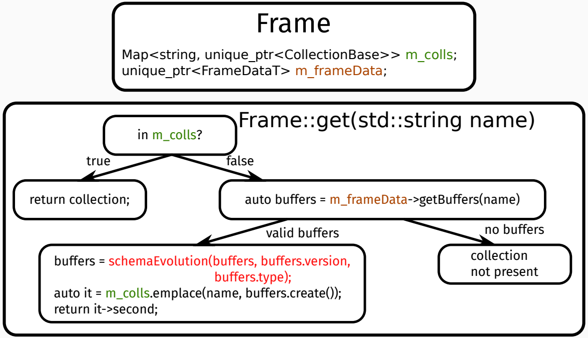
I/O on the Frame level



- Readers & Writers assumed to be single threaded
  - Low level building blocks
- Defer work as long as possible
  - Minimize time in Reader
- Frame can be constructed from “arbitrary” `FrameDataT`
  - Provides `CollectionBuffers`
  - Contain complete data for a Frame

Schema evolution in podio

- Many considerations
  - Leverage backend if possible
  - Work for all backends
  - Allow user overrides
- Evolution always to latest version
  - e.g. 1 → 3 and 2 → 3
  - Users only see latest version
- Similar approach as `CollectionBufferFactory`
- Detect potential problems at code generation
  - Expand capabilities as necessary



Frame

Map<string, unique_ptr<CollectionBase>> m_colls;
unique_ptr<FrameDataT> m_frameData;

Frame::get(std::string name)

in m_colls?

true

false

return collection;

auto buffers = m_frameData->getBuffers(name)

valid buffers

no buffers

buffers = schemaEvolution(buffers, buffers.version, buffers.type);
auto it = m_colls.emplace(name, buffers.create());
return it->second;

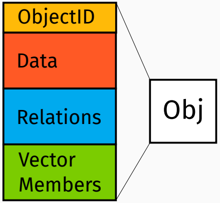
collection not present

```
Comparing datamodel versions v2 and v1

Found 3 schema changes:
 - 'ex2::NamespaceStruct' has an addded member 'y'
 - 'ex2::NamespaceStruct' has a dropped member 'y_old'
 - 'ExampleStruct.x' changed type from 'int' to 'double'

Warnings:
 - Definition 'ex2::NamespaceStruct' has a potential member [...]

ERRORS:
 - Forbidden schema change in 'ExampleStruct' for 'x' [...]
```

Other recent developments

- Stable collection IDs
  - Initially: Insertion order into Frame
  - Now: Hash of collection name
- RNTuple based backend
- Storing datamodel definition in *metadata Frame*
  - String literal embedded into binary
  - Dumping via `podio-dump`
  - Always possible to regenerate datamodel from datafile

```
struct ObjectID {
  int index;
  uint64_t collectionID;
};
```



```
readelf -p .rodata libedm4hep.so | grep options
  [   368]  {"options": {<...>},
"schema_version": 1, "components": {<...>},
"datatypes": {<...>}}
```

Frame based I/O in k4FWCore

- key4hep/k4FWCore offers core Key4hep services for Gaudi
  - **Data service for podio generated EDMs**
  - Historically grown separate implementation
- Replaced custom Reader / Writer with podio provided ones
  - (Almost) completely transparent
- `podio::Frame` not visible to user
- Some usability improvements in the works

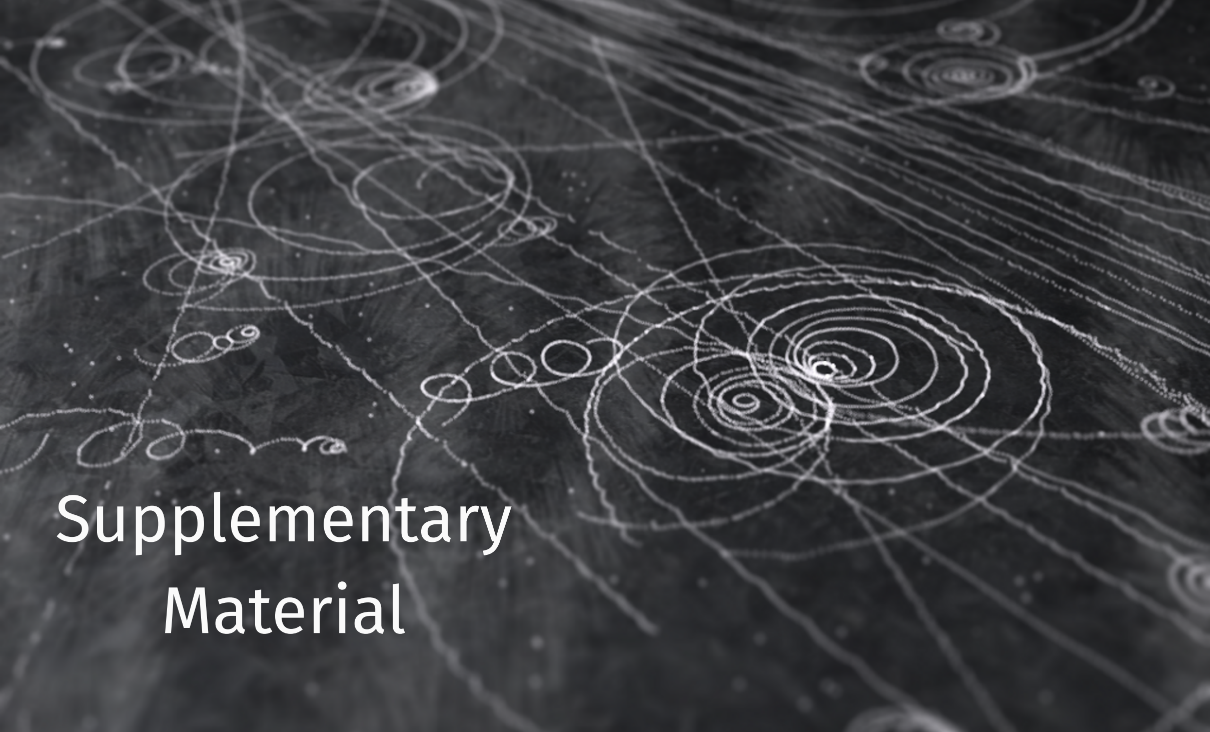
```
using namespace edm4hep;

// declare handle
DataHandle<MCParticleCollection> m_pHandle{
    "Particles",
    Gaudi::DataHandle::Reader,
    this};

// declare handle as property
declareProperty("ParticleColl",
                m_pHandle,
                "mc collection");

// use as
const auto particle = m_pHandle.get();
```

Summary & Outlook

- The podio EDM toolkit is already used by several communities
- Many crucial features for a stable release were missing
- Introduced the Frame concept
- Complete overhaul of I/O parts to make schema evolution possible
- **Schema evolution**
- Consolidation of k4FWCore and standalone podio
- Tying up last few loose ends for v1.0
  - Backwards compatibility from then on

Future plans

- Explore c++20 features for usage in podio (e.g. concepts, ranges)
- Explore usage on heterogeneous resources



Supplementary Material

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)

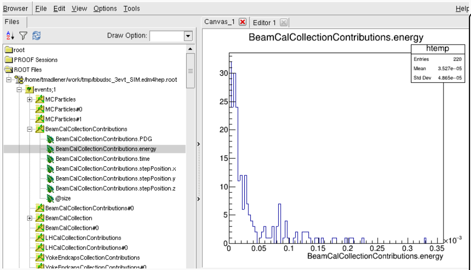
CMake interface for projects using podio

```
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)
```

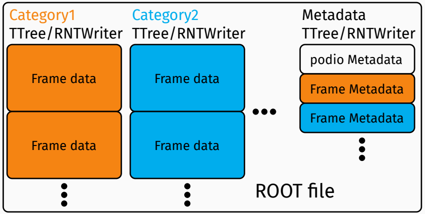
- Easy to use functions for integrating a podio generated EDM into a project
- Split into core EDM library and I/O handling for different backends
  - Pick what you need
  - I/O handling parts dynamically loaded by podio on startup

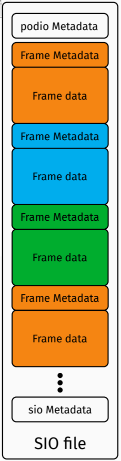
File layouts for Frame based I/O

- Default ROOT backend
  - One branch per collection data
  - More branches for relations and vector members
  - Contents of each *category* fixed by first Frame
  - Columnar on disk from hierarchical in memory
- Alternative SIO based backend
  - I/O library of LCIO (linear collider EDM)
  - Independent Frames



root
PROOF Sessions
ROOT Files
/home/tmadlener/work/tmp/bbudsc_3evt_SIM.edm4hep.root
events;1
MCParticles
MCParticles#0
MCParticles#1
BeamCalCollectionContributions
BeamCalCollectionContributions.PDG
BeamCalCollectionContributions.energy
BeamCalCollectionContributions.time
BeamCalCollectionContributions.stepPosition.x
BeamCalCollectionContributions.stepPosition.y
BeamCalCollectionContributions.stepPosition.z
@size
BeamCalCollectionContributions#0
BeamCalCollection
BeamCalCollection#0
LHCalCollectionContributions
LHCalCollectionContributions#0
YokeEndcapsCollectionContributions
YokeEndcapsCollectionContributions#0


Category1
TTree/RNTWriter
Frame data
Frame data
Category2
TTree/RNTWriter
Frame data
Frame data
Metadata
TTree/RNTWriter
podio Metadata
Frame Metadata
Frame Metadata
ROOT file


podio Metadata
Frame Metadata
Frame data
Frame Metadata
Frame data
Frame Metadata
Frame data
Frame Metadata
Frame data
sio Metadata
SIO file